ISBN 978-5-94074-366-8

ЦРИТО
Центр Развития ИТ-Образования

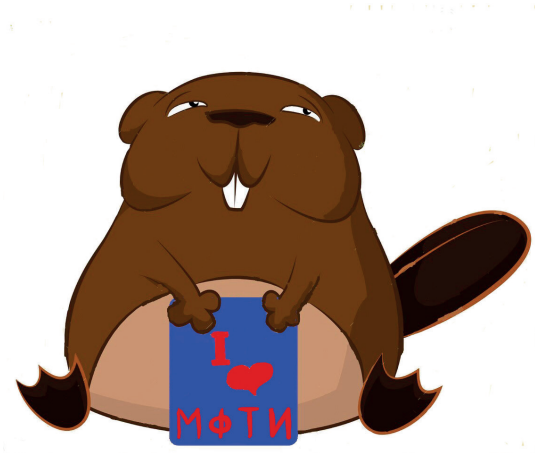
ОЛИМПИАДНЫЕ ШКОЛЫ МФТИ

МФТИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ»
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЦЕНТР РАЗВИТИЯ ИТ-ОБРАЗОВАНИЯ



ТЕОРМИНИМУМ
для поступивших в МФТИ

Бабичева Татьяна Сергеевна, Бабичев Сергей Леонидович,
Бабичев Дмитрий Сергеевич, Бабичева Наталья Николаевна,
Голубенко Дмитрий Александрович, Гущин Дмитрий Денисович,
Жогов Александр Александрович, Молчанов Евгений Геннадьевич,
Плетнев Никита Вячеславович, Шпакова Татьяна Александровна

Московский физико-технический институт
(национальный исследовательский университет)
(МФТИ)

Теорминимум для поступивших в МФТИ

Москва, 2019

УДК 004.01
ББК 30ф

Московский физико-технический институт
(национальный исследовательский университет) (МФТИ)

Теорминимум для поступивших в МФТИ. – М.: ДМК Пресс, 2019. – 130 с. : ил.

ISBN 978-5-94074-366-8

ISBN 978-5-94074-366-8

Оглавление

1 Введение

Если вы держите в руках это методическое пособие, значит, вы хотите учиться или, возможно, уже учитесь в МФТИ. Мы не будем расписывать, чем отличается МФТИ от других университетов, а также чем отличаются одни факультеты от других (это, например, есть в проспектах физтех-школ), а дадим набор базовых знаний для успешной учёбы.

Главным образом здесь раскрываются темы, которые изучаются не во всех школах, но большинством преподавателей воспринимаются как широко известные (основы олимпиадной математики и информатики). Также даются рекомендации по предметам, являющимся спецификой непосредственно физтех-школы прикладной математики и информатики, ФУПМа (дискретный анализ и лабораторные работы по базам данных) или Физтеха в целом.

При написании данной книги авторы ставили цель дать читателю определённые знания, которые помогут не растеряться на первых парах, а прийти на них, уже имея в голове некоторый опорный минимум. Эта книга — аналог «Лекций по математике» Босса: она не может и не призвана заменить учебники, а лишь является дополнением к ним; её следует внимательно прочитать в самом начале первого семестра, а лучше — еще раньше.

Данное методическое пособие включает в себя много тем, вызывающих трудности у первокурсников МФТИ, но мы понимаем, что невозможно охватить всё. Мы задаём вам направление движения. В дальнейшем пути вам помогут лекции, семинары, учебники, друзья, здравый смысл и Google.

Авторы искренне благодарят всех своих друзей, учеников и коллег за вдохновение.

Многие ссылки в данной книге кликабельны, а также многие рисунки в оригинале были нарисованы в цвете, поэтому для читающих бумажную версию данной книги будет актуальной постоянная ссылка на неё www.babichev.org/FUPM/teormin.pdf.

Найденные опечатки и пожелания можно (и нужно!) отправлять следующим людям:

- Татьяна Бабичева eriright@gmail.com
  http://vk.com/tyanko
- Дмитрий Голубенко golubenko@mccme.ru
- Никита Плетнев nikita_pletnev@list.ru
  http://vk.com/npletnev1997

2 Квантор общности и другие иероглифы

Начиная с первых же занятий, вам, во-первых, придётся писать свои конспекты лекций, а, во-вторых, читать чужие конспекты и учебники.

В математике очень часто используются различные сокращения и обозначения. Много интересного про историю обозначений можно прочитать в следующей статье википедии.

Начнём, надеемся, со знакомых для всех символов. И да, их можно и нужно использовать для замены в том числе обычных слов в конспектах, так как от количества письма на Физтехе почерк сильно портится, а вам ещё нужно будет писать письменные экзамены так, чтобы преподаватели их поняли.

- $\Rightarrow$ — Импликация, следование, «следовательно». $A \Rightarrow B$ обозначает, что если верно A, то верно и B.
- $\Leftrightarrow$ — Равносильность, «если и только если», «тогда и только тогда, когда ... »
- $\bigwedge$ — Конъюнкция, «и». Подробнее об данной операции написано в разделе, посвященном математической логике.
- $\bigvee$ — Дизъюнкция, «или».
- $\neg$ — Отрицание, «не».
- $\forall$ — Квантор всеобщности (общности). Может быть расшифровано как «для любых», «для всех», «для всякого», ...
- $\exists$ — Квантор существования. «Существует».
- $\nexists$ — «Не существует».
- $:=$ — «Определено, как», «положим, что». $x := 4$ — «положим x равным 4».
- $\emptyset$ — Пустое множество. Если вас так называет преподаватель, пора паниковать.
- $\mathbb{N}$ — Натуральные числа. Ну тут всё понятно, это числа 1,2,3,... Например, число людей в аудитории обычно является натуральным (так как преподаватель всё-таки обычно присутствует).
- $\mathbb{Z}$ — Целые числа.
- $\mathbb{Q}$ — Рациональные числа. Числа, представимые в виде $\frac{m}{n}$, где m — целое, а n— натуральное.

- $\mathbb{R}$ — Действительные числа.
- $\in$ — Принадлежность/непринадлежность к множеству. Расшифровывается, как «принадлежит», «из». Например, $5 \in \mathbb{Z}$ — «5 принадлежит множеству целых чисел».
- $\notin$ — Не принадлежит.
- $\subset$ — Подмножество, «является подмножеством», «включено в». Например, $\mathbb{N} \subset \mathbb{Z}$ — «Натуральные числа — это подмножество целых».
- $\bigcup$ — Объединение. «Объединение ... и ...», «..., объединённое с ...». Например, $\{2,3,4\} \bigcup \{1,3,6\} = \{1,2,3,4,6\}$.
- $\bigcap$ — Пересечение. «Пересечение ... и ...», «..., пересечённое с ...». Например, $\{2,3,4\} \bigcap \{1,3,6\} = \{3\}$.
- $\setminus$ — Разность множеств. «разность ... и ...», «минус», «... без ...». $\mathbb{Z} \setminus \mathbb{N} = \{0, -1, -2, -3, -4, ...\}$.
- ∞ — Бесконечность.
- $\sum$ — Сумма (набора чисел).
- $\prod$ — Произведение.
- $(\cdots)!$ — Факториал. Произведение всех натуральных чисел от 1 до числа, от которого берём факториал. $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$. Исключение — $0! = 1$.
- $const$ — Константа, постоянная величина, например, $g = 9{,}8.. = const$ — ускорение свободного падения.
- def — Дефиниция (определение). Def: аббревиатура def обозначает «определение».
- exp — Экспоненциальная функция, $exp(x) = e^x$. Используется зачастую для того, чтобы не плодить этажей в вычислениях.
- inf — Точная нижняя грань (*infimum*), самое большое значение, которым функция или последовательность ограничена снизу. Например, $\inf(\cos(x)) = -1$.
- sup — Точная верхняя грань (*supremum*), самое маленькое значение, которым функция или последовательность ограничена сверху. Например, $\sup(\cos(x)) = 1$.
- lim — Предел последовательности, функции или вообще слово «предел» в конспекте.

- max — Максимум функции или множества.
- min — Минимум функции или множества.
- НОД (GCD)— Наибольший общий делитель.
- НОК (LCM) — Наименьшее общее кратное.
- sgn — Функция «сигнум» (также обозначается как $sign$).
  Равна 1, если число положительно или ноль, иначе равна -1.

Достаточно быстро вам попытаются сломать мозг на математическом анализе выражениями, составленными полностью из кванторов. Например, как расшифровать такое выражение:

$$\forall x > 5\, \exists n \in \mathbb{N} : n < x?$$

Знак «:» (двоеточие) расшифровывается в таких выражениях как «такой, что».

Тогда выражение можно прочитать, как

«Для любого x, большего 5, существует такое натуральное число n, что n строго меньше x.»

Для начала обучения на Физтехе понимание указанных обозначений и кванторов вам будет достаточно. Остальные обозначения вы пройдёте на занятиях.

Глава I

Логика и техники доказательства

Существуют разные способы поиска ответов на вопросы; как правило, самый эффективный и безотказно действующий в математике — *доказательство*, то есть логический вывод теоремы из некоторого набора аксиом и других заведомо корректных утверждений. Наглядный и знакомый со школы пример доказательства — решение уравнения: из условия на x мы делаем вывод, какие значения может принимать x. Доказательство подобно пазлу: оно может состоять из любых элементов — можно доказывать геометрические утверждения алгебраически[1] или алгебраические теоремы геометрически[2] — но эти элементы должны стыковаться друг с другом, то есть должны быть доказаны не только сами утверждения, но и следствия одних утверждений из других.

В этой главе мы начнём с короткого введения в теорию множеств и алгебру логики, затем разберём, как формулировать математические утверждения, и приведём простейшие техники доказательства.

1 Логические высказывания

Классическая логика оперирует высказываниями, которые бывают *истинными* или *ложными*. Например, высказывание «существует простое число, большее 10^9» истинно, а высказывание «у равнобедренного прямоугольного треугольника один из углов равен 30°» неверно. Из одних высказываний можно составлять более сложные с помощью операций $\neg$ (отрицание), $\wedge$ (и), $\vee$ (или), $\to$ (влечёт) и кванторов $\exists$ (существует) и $\forall$ (для любого). Добавляя квантор, нужно вставить соответствующую переменную (или набор переменных).

Пример 1. Высказывание «n четно» означает, что существует некоторое $k \in \mathbb{Z}$, что $n = 2k$, что можно записать как

$$n \text{ четно} \iff \exists k \in \mathbb{Z} \quad n = 2k$$

Пример 2. По определению, если натуральное число p простое то не существует натуральных чисел, превышающих 1, дающих в произведении p. Это можно записать в виде высказывания вида $A(\tau) \wedge \neg(\exists \tau B(\tau) \wedge C(\tau) \wedge D(\tau))$:

$$p \text{ простое} \iff (p \neq 1) \wedge \neg\big(\exists x, y \in \mathbb{N} \quad (x > 1) \wedge (y > 1) \wedge (xy = p)\big)$$

[1] Таким методам посвящена аналитическая геометрия, которая будет изложена позже.

[2] Докажите, что если целые положительные числа a, b, c удовлетворяют $a^2 + b^2 = c^2$, то существуют целые m,n,k такие, что $a = k(m^2 - n^2)$, $b = 2kmn$, $c = k(m^2 + n^2)$.

Есть и иной способ выразить высказыванием простоту числа: число p просто, если в любом представлении $p = xy$ хотя бы одно из чисел равно 1. Последнее высказывание имеет вид $A(\tau) \wedge \forall\tau B(\tau) \to C(\tau) \vee D(\tau)$:

$$p \text{ простое} \iff (p \neq 1) \wedge \forall x, y \in \mathbb{N} \quad (xy = p) \to (x = 1) \vee (y = 1)$$

Пример 3. Классическое ϵ-δ определение предела словесно выражается так: X называется пределом последовательности $(x_n)_{n\in\mathbb{N}}$, если для любого $\epsilon > 0$ существует такое $N \in \mathbb{N}$, что для всех $n > N$ расстояние между x_n и X меньше ϵ. Соответствующая формула имеет вид $\forall\tau_1\exists\tau_2\forall\tau_3 A(\tau_2, \tau_3) \to B(\tau_1, \tau_3)$:

$$X = \lim_{n\to\infty} x_n \iff \forall\epsilon > 0\ \exists N \in \mathbb{N}\ \forall n \in N\ (n > N) \to (|x_n - X| < \epsilon)$$

Абстрагируясь от конкретных высказываний, введем *булевы переменные* как переменные, принимающие значения **И** (истина, обозначаемая зачастую единицей 1) или **Л** (ложь, обозначаемая зачастую нулём 0), а *логические функции* от булевых переменных $x_1, \dots x_n$ как соответствия каждому набору значений переменных значения **И** или **Л**. Логическую функцию можно задать *таблицей истинности*: в каждой строке стоят значения $x_1, \dots x_n, f(x_1, \dots x_n)$, каждый набор значений переменных представлен ровно одной строкой. Для функции $\neg x$ таблица истинности имеет вид

x	$\neg x$
И	**Л**
Л	**И**

Для бинарных операций таблица истинности приведена ниже:

x	y	$x \vee y$	$x \wedge y$	$x \to y$
И	**И**	**И**	**И**	**И**
И	**Л**	**И**	**Л**	**Л**
Л	**И**	**И**	**Л**	**И**
Л	**Л**	**Л**	**Л**	**И**

Пример 4. Составим таблицу истинности формулы $f(x_1, x_2) = (\neg x_1 \wedge x_2) \vee x_1$ (для удобства оставим столбцы для промежуточных вычислений):

x_1	x_2	$\neg x_1$	$\neg x_1 \wedge x_2$	$f(x_1, x_2)$
И	**И**	**Л**	**Л**	**И**
И	**Л**	**Л**	**Л**	**И**
Л	**И**	**И**	**И**	**И**
Л	**Л**	**И**	**Л**	**Л**

Мы можем сказать, что высказывание можно получить, подставив в некоторую логическую комнату вместо переменных другие высказывания. Истинность такого высказывания можно установить, зная истинность подставленных высказываний и таблицу истинности соответствующей формулы.

Задача I.1. Корректно ли следующее рассуждение?

Либо sergey не дурак, но лентяй, либо он дурак. sergey дурак.
Следовательно, sergey не лентяй.

Решение. Пусть X — утверждение «sergey дурак», а Y — утверждение «sergey лентяй». Привёденный выше аргумент имеет вид

$$F(X,Y) = ((\neg X \wedge Y) \vee X) \wedge X \to \neg Y.$$

Надо проверить, имеет ли место следствие; для этого надо построить таблицу истинности указанной импликации. Мы уже строили немного раньше таблицу истинности $(\neg X \wedge Y) \vee X$, воспользуемся же полученным результатом.

X	Y	$(\neg X \wedge Y) \vee X$	X	$\neg Y$	$F(X,Y)$
И	**И**	**И**	**И**	**Л**	**Л**
И	**Л**	**И**	**И**	**И**	**И**
Л	**И**	**И**	**Л**	**Л**	**И**
Л	**Л**	**Л**	**Л**	**И**	**И**

Как мы можем видеть, подобный аргумент не корректен: sergey может быть и дураком, и лентяем одновременно, в таком случае предположения («sergey не дурак, но лентяй, либо он дурак. sergey дурак.») верны, а вывод («sergey не лентяй») неверен. □

Для удобства также будем рассматривать операцию $\equiv$ (эквивалентность): зададим $x \equiv y$ как $(x \to y) \wedge (y \to x)$. Нетрудно видеть, что $x \equiv y$ титтк x и y принимают одно и то же значение.

Задача I.2. Остров населен рыцарями, всегда говорящими правду, и лжецами, которые всегда лгут. Трем проходившим мимо туземцам был задан вопрос: «Сколько твоих компаньонов являются рыцарями?» Первый сказал: «Ноль». Второй ответил: «Один». Что ответил третий?

Решение. Обозначим за X высказывание «Первый путник — рыцарь», за Y — высказывание «Второй путник — рыцарь», а за Z — «Третий путник — рыцарь». По ответам двух путников составим две логические формулы, а из условия их истинности выведем возможные значения X, Y и Z.

Первый респондент, назвавший своих спутников лжецами, прав титтк (тогда и только тогда, когда) и второй, и третий путник действительно являются лжецами, поэтому соответствующее высказывание имеет вид

$$S_1(X,Y,Z) = X \equiv (\neg Y \wedge \neg Z).$$

Второй же оказывается прав титтк либо первый является рыцарем, а третий — лжецом или наоборот, по его словам составляем выражение

$$S_2(X,Y,Z) = Y \equiv \big((X \wedge \neg Z) \vee (\neg X \wedge Z)\big).$$

Теперь строим таблицу истинности этих выражений.

X	Y	Z	$\neg Y \wedge \neg Z$	$X \wedge \neg Z$	$\neg X \wedge Z$	S_1	S_2	$S_1 \wedge S_2$
И	**И**	**И**	**Л**	**Л**	**Л**	**Л**	**Л**	**Л**
И	**И**	**Л**	**Л**	**И**	**Л**	**Л**	**И**	**Л**
И	**Л**	**И**	**Л**	**Л**	**Л**	**Л**	**И**	**Л**
И	**Л**	**Л**	**И**	**И**	**Л**	**И**	**Л**	**Л**
Л	**И**	**И**	**Л**	**Л**	**И**	**И**	**И**	**И**
Л	**И**	**Л**	**Л**	**Л**	**Л**	**И**	**Л**	**Л**
Л	**Л**	**И**	**Л**	**Л**	**И**	**И**	**Л**	**Л**
Л	**Л**	**Л**	**И**	**Л**	**Л**	**Л**	**И**	**Л**

Так как по условию рыцари говорят правду, а лжецы всегда лгут, то $S_1 \wedge S_2$ должно выполняться, что происходит только при $X =$ **Л**, $Y =$ **И**, $Z =$ **И**. Следовательно, третий путник — рыцарь, который скажет, что один из его путников рыцарь. □

Будем говорить, что логические формулы $f(x_1, \dots x_n)$ и $g(x_1, \dots x_n)$ *эквивалентны*, если $f \equiv g$ верна на всех наборах значений переменных, то есть значения f и g равны на всех наборах значений переменных. Например, функции $(x \to y) \wedge (x \to \neg y)$ и $\neg x$ эквивалентны:

x	y	$x \vee y$	$x \to y$	$x \to \neg y$	$\neg x$
И	**И**	**И**	**Л**	**Л**	**Л**
И	**Л**	**Л**	**И**	**Л**	**Л**
Л	**И**	**И**	**И**	**И**	**И**
Л	**Л**	**И**	**И**	**И**	**И**

Операции $\vee$ и $\wedge$ обладают следующими свойствами:

- $x \vee y \equiv y \vee x$, $x \wedge y \equiv y \wedge x$ (коммутативность);
- $x \vee (y \vee z) \equiv (x \vee y) \vee z$, $x \wedge (y \wedge z) \equiv (x \wedge y) \wedge z$ (ассоциативность);

- $x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$, $x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$ (дистрибутивность);
- $x \vee x \equiv x$, $x \wedge x \equiv x$ (идемпотетность);
- $x \vee (x \wedge y) \equiv x$, $x \wedge (x \vee y) \equiv x$ (формулы поглощения).

Закон двойного отрицания гласит, что $\neg\neg x \equiv x$, то есть двойное отрицание можно сократить. Взаимодействие отрицания с $\vee$ и $\wedge$ отражается в следующих *формулах де Моргана*:

$$\neg(x \wedge y) \equiv (\neg x) \vee (\neg y), \quad \neg(x \vee y) \equiv (\neg x) \wedge (\neg y)$$

Проверить все эти эквивалентности можно, построив соответствующие таблицы истинности.

Отрицание можно «пронести» и через кванторы: естественным отрицанием «выполнения всюду» является «невыполнение где-то» и наоборот. Иными словами,

$$\neg(\forall \tau A(\tau)) \equiv \exists \tau(\neg A(\tau)), \quad \neg(\exists \tau A(\tau)) \equiv \forall \tau(\neg A(\tau))$$

Увидим, как это работает, составив отрицания к утверждениям в примерах в начале параграфа.

Пример 5. Высказывание «n чётно» означает, что существует некоторое $k \in \mathbb{Z}$, что $n = 2k$, соответственно, отрицание этого утверждения формулируется как «$n \neq 2k$ для любого $k \in \mathbb{Z}$». В кванторах имеем

$$\begin{aligned} n \text{ четно} &\iff \exists k \in \mathbb{Z} \quad n = 2k \\ n \text{ нечетно} &\iff \forall k \in \mathbb{Z} \quad n \neq 2k \end{aligned}$$

Пример 6. Высказывание «p простое» записывается в виде $A(\tau) \wedge \neg(\exists \tau B(\tau) \wedge C(\tau) \wedge D(\tau))$:

$$p \text{ простое} \iff (p \neq 1) \wedge \neg\big(\exists x, y \in \mathbb{N} \quad (x > 1) \wedge (y > 1) \wedge (xy = p)\big).$$

Соответственно, высказывание «p – не простое» согласно закону о двойном отрицании имеет вид $A(\tau) \vee \exists \tau B(\tau) \wedge C(\tau) \wedge D(\tau)$:

$$p \text{ не простое} \iff (p = 1) \vee \big(\exists x, y \in \mathbb{N} \quad (x > 1) \wedge (y > 1) \wedge (xy = p)\big).$$

Пример 7. По определению предела последовательности

$$X = \lim_{n \to \infty} x_n \iff \forall \epsilon > 0 \; \exists N \in \mathbb{N} \; \forall n \in N \; (n > N) \to (|x_n - X| < \epsilon)$$

Строя утверждение «X не является пределом последовательности $(x_n)_{n\in\mathbb{N}}$», мы должны «протащить» отрицание через кванторы в выражении вида $\forall\tau_1\exists\tau_2\forall\tau_3 A(\tau_2,\tau_3)\to B(\tau_1,\tau_3)$. Получится выражение вида

$$\neg\Big(\forall\tau_1\exists\tau_2\forall\tau_3 A(\tau_2,\tau_3)\to B(\tau_1,\tau_3)\Big)\equiv\exists\tau_1\neg\Big(\exists\tau_2\forall\tau_3 A(\tau_2,\tau_3)\to B(\tau_1,\tau_3)\Big)\quad\equiv$$
$$\exists\tau_1\forall\tau_2\neg\Big(\forall\tau_3 A(\tau_2,\tau_3)\to B(\tau_1,\tau_3)\Big)\equiv\exists\tau_1\forall\tau_2\exists\tau_3\neg\Big(A(\tau_2,\tau_3)\to B(\tau_1,\tau_3)\Big)$$

Несложно заметить, что $x\to y\equiv\neg(x\wedge\neg y)$, поэтому отрицание к импликации имеет вид $x\wedge\neg y$. Итого утверждение «X не является пределом последовательности $(x_n)_{n\in\mathbb{N}}$» имеет вид $\exists\tau_1\forall\tau_2\exists\tau_3 A(\tau_2,\tau_3)\wedge\neg B(\tau_1,\tau_3)$:

$$X\neq\lim_{n\to\infty}x_n\iff\exists\epsilon>0\;\forall N\in\mathbb{N}\;\exists n\in N\;(n>N)\wedge(|x_n-X|\geqslant\epsilon)$$

2 Теория множеств

Не вдаваясь подробно в основания математики, будем считать, что *множество* — набор некоторых объектов[3] (чисел, точек плоскости, таблиц...). Например, множествами являются

- множество натуральных чисел $\mathbb{N}=\{1,2,\ldots\}$;
- множество простых чисел $\{2,3,5,7,11,\ldots\}$;
- множество связных графов на 100 вершинах;
- множество отрезков на плоскости.

Объекты, лежащие в множестве A, будем называть *элементами множества* A. Выражение $x\in A$ означает «x является элементом множества A»; если же x не является элементом A, то будем писать $x\notin A$. Множество, состоящее из элементов x_1, x_2, $\ldots x_n$, будем обозначать $\{x_1,x_2,\ldots x_n\}$. Например, 3 является элементом множества простых чисел, а 6 — нет.

Множество, не содержащее ни одного элемента, будем называть *пустым множеством* и обозначать как $\varnothing$.

Скажем, что множество B является *подмножеством* A, если все элементы B являются элементами A; в таком случае пишут $B\subset A$. Пустое подмножество является подмножеством любого множества.

[3]Кантор вводил множество как «...любое объединение в одно целое M определенных вполне различаемых объектов m из нашего восприятия или мысли».

Два множества A и B *совпадают*, если любой элемент A является элементом B и наоборот. Иными словами, множества A и B совпадают, если $A \subset B$ и $B \subset A$.

Теперь введем бинарные операции с множествами. Скажем, что *объединение множеств* A и B есть множество

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

Введем *пересечение множеств* A и B есть множество

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

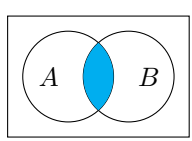
Разность множеств A и B есть множество

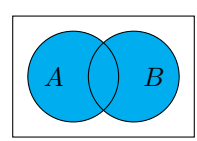
$$A \setminus B = \{x \mid x \in A \wedge x \notin B\}$$

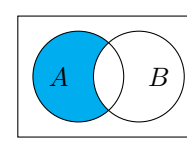
Например, если $A = \{1, 2, 3, 4, 5\}$ и $B = \{2, 4, 6, 8, 10\}$, то

$$A \cap B = \{2, 4\}, \quad A \cup B = \{1, 2, 3, 4, 5, 6, 8, 10\}, \quad A \setminus B = \{1, 3, 5\}.$$

Мы будем схематически изображать результаты операций над множествами в виде *диаграмм Венна* (или Эйлера-Венна), кружками мы будем изображать множества. На этих рисунках изображены диаграммы Венна для пересечения, объединения и разности множеств соответственно.







Нетрудно видеть, что операции пересечения и объединения коммутативны:

$$A \cap B = B \cap A, \quad A \cup B = B \cup A,$$

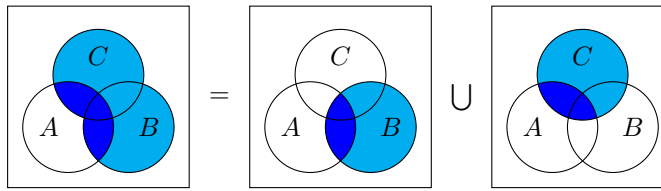
и ассоциативны:

$$A \cup (B \cup C) = (A \cup B) \cup C, \quad A \cap (B \cap C) = (A \cap B) \cap C.$$

Теперь установим дистрибутивность операций объединения и пересечения.

Задача I.3. Докажите, что для любых множеств A, B, C верно

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$



Решение. Фактически дистрибутивность операций над множествами следует из дистрибутивности операций с высказываниями. Формально,

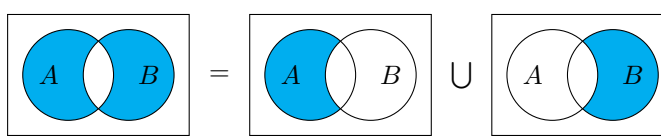
$$x \in A \cap (B \cup C) \iff x \in A \wedge (x \in B \vee x \in C) \iff$$
$$\iff (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C) \iff x \in (A \cap B) \cup (A \cap C).$$

□

Задача I.4. Докажите, что для любых множеств A и B верно

$$(A \cup B) \setminus (A \cap B) = (A \setminus B) \cup (B \setminus A)$$

Решение. Если $x \in (A \cup B) \setminus (A \cap B)$, то x лежит ровно в одном из множеств A или B — хотя бы в одном, но не в двух сразу. Если этот элемент лежит в A, он не лежит в B и принадлежит таким образом $A \setminus B$; тем же способом получаем, что если $x \in B$, то $x \in B \setminus A$. Тогда $x \in (A \setminus B) \cup (B \setminus A)$ и $(A \cup B) \setminus (A \cap B) \subset (A \setminus B) \cup (B \setminus A)$. Обратное вложение доказывается аналогично. □



Декартовым произведением[4] множеств A и B называется упорядоченное множество пар элементов A и B:

$$A \times B \stackrel{\text{def}}{=} \Big\{(a,b) \mid a \in A,\ b \in B\Big\}$$

Мы обозначаем декартово произведение множеств A и B как $A \times B$. Например, декартовым произведением отрезка $[0;1]$ на себя будет квадрат $[0;1] \times [0;1]$, а декартово произведение $[0;1] \cup [2;3]$ и $[1;2] \cup [3;4]$ — объединение четырех квадратов.

[4]Рене Декарт именовал себя по-латински Картезием, поэтому в англоязычной литературе оно называется Cartesian product.

Заметим, что $A \times \varnothing = \varnothing$: если хотя бы одна пара (a,x) лежала в $A \times \varnothing$, то x лежал бы в пустом подмножестве, а такого x не существует. Аналогично $\varnothing \times A = \varnothing$.

Обратите внимание, что $A \times B$ и $B \times A$ не совпадают как множества. Они, впрочем, *изоморфны* — между ними существует взаимооднозначное соответствие, сопоставляющее паре $(a,b) \in A \times B$ пару $(b,a) \in B \times A$.

Задача I.5. Пусть множества A, B, C, D таковы, что $A \times B \subset C \times D$. Верно ли, что $A \subset C$ и $B \subset D$?

Решение. Рассмотрим $A = D = \varnothing$, $B = C = \mathbb{N}$. Тогда $A \times B = C \times D = \varnothing$ и $A \times B \subset C \times D$. Впрочем, $B \not\subset D$: множество натуральных чисел не может быть подмножеством пустого множества. □

3 Техники доказательства

После того, как мы сформулировали математическое утверждение, можно доказывать или опровергать его. Разумеется, одними лишь логическими манипуляциями нельзя проделать все доказательство целиком, однако они позволяют упрощать условие задачи, разбивать ее в более мелкие задачи.

Опровергать теоремы можно, найдя контрпример. Утверждение вида $\forall \tau \;\; A(\tau)$ неверно, если при некотором значении τ выполняется $\neg A(\tau)$; соответствующее значение τ называется *контрпримером* к утверждению $\forall \tau \;\; A(\tau)$. Например, высказывание «при любых n число $2^n + 1$ простое» неверно: контрпримером является $n = 3$, при котором $2^3 + 1 = 3^2$.

Теоремы вида «X влечет Y» можно доказывать следующим образом:

Положим, что X истинно. Докажем истинность Y. Следовательно, $X \to Y$.

Вместо доказательства истинности импликации $X \to Y$ мы доказываем Y, используя гипотезу X; очень часто сделать последнее оказывается проще.

Пример 8. Докажем, что если вещественные числа a и b удовлетворяют $0 < a < b$, то $a^2 < b^2$. Положим $0 < a < b$ в качестве гипотезы, теперь надо установить $a^2 < b^2$. Дальше идет непосредственно само доказательство: $a^2 < ab < b^2$.

Пример 9. Докажем, что если целое число a четное, то a^2 делится на 4. Предположим, что a четно, то есть $a = 2k$ для некоторого $k \in \mathbb{Z}$, теперь надо установить $4|a^2$. Дальше идет непосредственно само доказательство: $a^2 = (2k)^2 = 4 \cdot k^2$, следовательно, a^2 делится на 4.

Иногда оказывается тяжело строить доказательства импликаций $X \to Y$ напрямую, доказывая Y в предположении истинности X. Иногда можно воспользоваться обратным способом:

Положим, что Y ложно, то есть $\neg Y$ истинно. Докажем истинность $\neg X$. Следовательно, $\neg Y \to \neg X$. Таким образом, $X \to Y$.

Несложно проверить, что $(X \to Y) \equiv (\neg Y \to \neg X)$, а доказывать импликацию $\neg Y \to \neg X$ порой бывает проще.

Пример 10. Пусть a,b,c — вещественные числа и $a > b$. Покажем, что если $ac \leqslant bc$, то $c \leqslant 0$. Доказывая $(ac \leqslant bc) \to (c \leqslant 0)$, допустим $\neg(c \leqslant 0)$, то есть $c > 0$, и докажем $\neg(ac \leqslant bc)$, то есть $ac > bc$. Действительно, если $a > b$ и $c > 0$, то эти неравества можно умножить и получить $ac > bc$.

3.1 Доказательства от противного

Иногда можно доказать утверждение, продемонстрировав невозможность обратного утверждения. Это называется *доказательством от противного*:

Положим, что X ложно, то есть $\neg X$ истинно. Выведем противоречие. Тогда X истинно.

Получить противоречие можно, например, доказав истинность импликаций $\neg X \to Y$ и $\neg X \to \neg Y$ для некоторого утверждения Y. Как правило, именно так мы и будем получать противоречия.

Пример 11. Докажем, что существует бесконечно много простых натуральных чисел. Положим обратное: допустим, что множество простых чисел конечно и имеет вид $\{p_1, \dots p_N\}$. Рассмотрим целое число $x = p_1 \cdot \ldots \cdot p_N + 1$. Нетрудно видеть, что x больше любого простого числа, значит, оно не является простым и должно делиться на хотя бы одно число из множества $\{p_1, \dots, p_N\}$. Это же, однако, невозможно: $x = 1 \mod p_i$ для любого простого p_i. Итак, мы получили противоречие:

- должно существовать хотя бы одно p_i, делящее x;
- не существует ни одного p_i, делящего x.

Разумеется, чтобы правильно строить доказательства от противного, надо правильно строить отрицания к утверждениям.

Задача I.6. Найдите ошибку в доказательстве приведенной ниже теоремы и укажите к ней контрпример.

Теорема 1. Допустим, что x, y — вещественные числа такие, что $x + y = 10$. Тогда $x \neq 3$ и $y \neq 8$.

Доказательство. Докажем это утверждение от противного. Тогда $x = 3$ и $y = 8$, но в таком случае $x + y = 11$, что противоречит условию $x + y = 10$. Таким образом $x \neq 3$ и $y \neq 8$. □

Решение. Согласно формуле де Моргана отрицание к утверждению $(x \neq 3) \wedge (y \neq 8)$ имеет вид $(x = 3) \vee (y = 8)$, то есть либо $x = 3$, либо $y = 8$. В приведённом доказательстве неверно составлено отрицание. Контпример же привести легко, достаточно взять $x = 3$, $y = 7$. □

3.2 Доказательства с кванторами

Если мы хотим доказывать утверждения вида $\forall \tau\ A(\tau)$ (для любого τ выполняется $A(\tau)$), то можно поступить так:

Для произвольного τ покажем выполнимость $A(\tau)$. В силу произвольности τ выведем $\forall \tau\ A(\tau)$.

Обобщать доказательство истинности $A(\tau)$ на случай любого τ можно, если доказательство не зависит от значения τ.

Пример 12. Докажем, что для любых множеств A, B, C таких, что $A \setminus B \subset C$, верно $A \setminus C \subset B$. Сначала положим, что $A \setminus B \subset C$, в этом предположении будем доказывать $A \setminus C \subset B$. Рассмотрим элемент $x \in A \setminus C$, докажем, что $x \in B$. Сделаем это от противного: если $x \notin B$, то раз $x \in A$, то $x \in A \setminus B$ и согласно условию $x \in C$, что противоречит $x \in A \setminus C$.

Если мы хотим опровергнуть утверждение вида $\forall \tau\ A(\tau)$, то нам нужно доказать утверждение вида $\exists \tau\ \neg A(\tau)$, то есть то, что существует контрпример к $A(\tau)$ — значение τ, при котором $A(\tau)$ ложно. Как правило, для этого достаточно указать это самое значение τ.

Пример 13. Опровергнем фразу «все числа вида $6^n + 1$ — простые». Действительно, при $n = 3$ имеем $6^3 + 1 = 217 = 7 \cdot 31$.

Поскольку $\exists \tau\ A(\tau)$ эквивалентно $\neg(\forall \tau\ \neg A(\tau))$, то доказывать $\exists \tau\ A(\tau)$ можно, найдя контрпример к $\neg A(\tau)$, то есть найдя хотя бы одно значение, при котором $A(\tau)$ истинно. Соответственно, чтобы опровергнуть $\exists \tau\ A(\tau)$, нужно доказать, что всегда верно утверждение $\neg A(\tau)$.

3.3 Принцип Дирихле

Еще один часто встречающийся метод доказательства — принцип Дирихле[5]. В простой формулировке он гласит, что если в n клетках сидит $n+1$ голубь, то хотя бы в одной клетке сидит хотя бы 2 голубя.

Докажем принцип Дирихле от противного: допустим, что в каждой клетке сидит не более одного голубя; тогда в n клетках сидит не более n голубей, что противоречит условию — всего в клетках $n+1$ голубей.

Разумеется, это утверждение можно обобщить:

Теорема 2 (принцип Дирихле)**.** Если в n попарно непересекающихся множествах $A_1, \dots A_n$ всего не менее $kn+1$ элемент, то в некотором A_i лежит хотя бы $k+1$ элемент.

Доказательство совершенно аналогично вышеприведенному.

Задача I.7. Покажите, что среди любых 7 целых чисел найдутся хотя бы три, сумма которых делится на 3.

Решение. Разобьем наши числа на три подмножества, каждое из которых соответствует остатку по модулю 3. Иными словами, число лежит в множестве A_i, если оно дает остаток i по модулю 3. Поскольку всего чисел $7 = 2 \cdot 3 + 1$, то согласно принципу Дирихле хотя бы три числа окажутся в одном A_i, а значит, их сумма дает остаток $3i = 0$ по модулю 3. □

3.4 Метод математической индукции

Иногда в решении задач можно использовать *математическую индукцию*. Обычно она полезна при доказательстве утверждений вида X_n, зависящих от натурального параметра n, вроде этих:

- сумма натуральных чисел от 1 до n равна $\frac{n(n+1)}{2}$;
- в любом связном дереве на n вершинах $n-1$ ребро [6];
- $n^2 \leqslant 2^n$ для любого натурального n...

Сначала устанавливается истинность X_1, затем доказывается, что любое X_n влечёт X_{n+1}, то есть если X_n верно, то верно и X_{n+1}. Тогда в силу истинности X_1 верно также и X_2, истинность X_2 влечет истинность X_3, аналогично верны

[5] В англоязычной литературе он называется pigeonhole principle.

[6] Про деревья и другие виды графов написано в главе «Дискретный анализ».

X_4, X_5 и так далее. Поскольку любое натуральное число n рано или поздно встретится, то истинность любого X_n будет доказана.

Теорема 3 (Метод математической индукции)**.** Дана последовательность высказываний X_n такая, что X_0 и $\forall n \in \mathbb{N} \quad X_n \to X_{n+1}$ истинны. Тогда $\forall n \in \mathbb{N} \quad X_n$.

Утверждение X_1 называется *базой индукции*, а $\forall n \in \mathbb{N} \quad X_n \to X_{n+1}$ — *шагом индукции*. Сначала проверяется истинность базы: если же X_1 ложно, то даже если переход верен, то X_2 может быть как верным, так и неверным. Затем доказывается шаг индукции. Здесь мы пользуемся техникой доказательства импликации $X \to Y$ и в предположении X доказываем Y. В данном случае это означает, что в *предположении индукции* X_n мы доказываем X_{n+1}. Важно отметить, что все утверждения $X_n \to X_{n+1}$ доказываются одновременно: чтобы индукция сработала, все импликации $X_n \to X_{n+1}$ должны быть верны.

Задача I.8. Докажите, что для любого натурального n выполняется равенство:

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

Решение. Докажем утверждение индукцией по n. В данном случае утверждение X_n имеет вид $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$. База индукции выполняется при $n = 1$:

$$1 = \frac{1 \cdot 2}{2}.$$

Проверим переход. Согласно предположению индукции

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

Докажем, что

$$1 + 2 + \cdots + n + (n+1) = \frac{(n+1)(n+2)}{2}.$$

Имеем

$$\underbrace{\big(1 + 2 + \ldots + n\big)}_{= \frac{n(n+1)}{2} \text{ по } X_n} + (n+1) = (n+1)\big(\frac{n}{2} + 1\big) = \frac{(n+1)(n+2)}{2}.$$

□

Задача I.9. Докажите, что при всех натуральных n выполняется тождество

$$1 \cdot 1! + 2 \cdot 2! + \cdots + n \cdot n! = (n+1)! - 1.$$

Решение. Проведём индукцию по n. При $n = 1$ имеем $1 \cdot 1! = (1+1)! - 1$, тем самым база установлена. Теперь докажем переход индукции, а именно, что для любого $n \in \mathbb{N}$ имеет место следствие

$$1 \cdot 1! + \cdots + n \cdot n! = (n+1)! - 1 \Rightarrow 1 \cdot 1! + \cdots + (n+1) \cdot (n+1)! = (n+2)! - 1.$$

Сравним приращения левой и правой частей и покажем, что они равны. Приращение левой части равно $(n+1)\cdot(n+1)!$, в то время, как приращение правой части равно

$$(n+2)!-1-((n+1)!-1) = (n+2)!-(n+1)! = (n+1)!\cdot(n+2-1) = (n+1)\cdot(n+1)!$$

Переход доказан, значит, задача решена. □

Иногда задачу можно решить, введя натуральный параметр и проведя индукцию по нему.

Задача I.10. Назовем простой (не имеющий петель и кратных ребер) граф планарным, если его можно изобразить на плоскости без пересечения ребер[7]. Покажите, что для любого связного планарного графа имеет место равенство $V - E + F = 2$, где V — число вершин, E — число ребер, а F — число граней этого графа.

Решение. Переформулируем условие задачи в виде утверждения, которое можно доказать по индукции. В качестве натурального параметра рассмотрим число вершин графа. Рассмотрим последовательность утверждений вида

X_n: для любого связного планарного графа на n вершинах выполняется
$$V - E + F = 2.$$

Нетрудно видеть, что утверждение $\forall n \in \mathbb{N} \quad X_n$ эквивалентно требуемому утверждению: любой связный планарный граф G имеет некоторое конечное число вершин n, а раз соответствующее X_n верно, то для G выполняется равенство $V - E + F = 2$.

Осталось доказать $\forall n \in \mathbb{N} \quad X_n$, что мы сделаем индукцией по n. Сначала проверим базу: любой граф с одной вершиной состоит из этой вершины и не имеет ребер, поэтому $1 - 0 + 1 = 2$. Теперь установим шаг индукции $\forall n \in \mathbb{N} \quad X_n \to X_{n+1}$; предположим истинность X_n и докажем, что X_{n+1} верно. В силу истинности X_n для любого связного планарного графа на n

[7]Верно даже более сильное утверждение: планарный граф можно изобразить на плоскости так, что его ребра — непересекающиеся *отрезки*; соответствующее утверждение называется теоремой Фари, доказательство которой — еще одно применение математической индукции.

вершинах выполняется $n - E + F = 2$. Любой планарный граф на $n+1$ может быть получен из некоторого n-вершинного планарного графа добавлением одной вершины, скажем, v_{n+1}, и некоторых ребер, соединяющих ее со старыми вершинами. Хотя бы одно такое ребро есть — в противном случае связность графа нарушается. Первое добавленное ребро, смежное с v_{n+1}, не делит никакую грань на две части: v_{n+1} становится висячей вершиной внутри какой-то грани. Каждое последующее ребро делит некоторую грань на две: в силу планарности это ребро лежит внутри некоторой грани и делит ее таким образом на две части. Тогда k ребер, смежных с v_{n+1}, добавляют $k-1$ новых граней; для этого графа получаем

$$(n+1) - (E+k) + (F+k-1) = n - E + F = 2.$$

□

Напоследок приведём пример ошибочного индуктивного рассуждения.

Задача I.11. Найдите ошибку в доказательстве следующего утверждения.

Теорема 4. Любые n лошадей одного цвета.

Доказательство. Проведем индукцию по n — числу лошадей. База очевидна — любая лошадь такого же цвета, как и она сама. Проверим теперь переход. В силу предположения индукции любые n лошадей одного цвета; докажем, что любые $n+1$ лошадей одного цвета. Рассмотрим любые $n+1$ лошадей: $Л_1, Л_2, \ldots, Л_n, Л_{n+1}$, по предположению индукции, лошади из множества $\{Л_1, \ldots, Л_n\}$ одного цвета, причём лошади из множества $\{Л_2, \ldots Л_{n+1}\}$ также одного цвета. Но лошадь $Л_2$ находится в обоих множествах, значит и все $n+1$ лошади одного цвета. □

Решение. На самом деле доказательство перехода здесь отсутствует и не работает для следствия $X_1 \to X_2$: в этом случае $Л_2$ не находится в обоих множествах, и рассматриваемые множества вообще не пересекаются. Хотя остальные переходы верны — из второго утверждения следует третье, и третьего — четвёртое, и так далее, — индукция работать не будет. □

Порой, впрочем, доказать переход вида $\forall n \in \mathbb{N} \quad X_n \to X_{n+1}$ бывает трудно: предположения истинности X_n может быть недостаточно. Поэтому предлагается рассматривать *метод полной математической индукции*, в котором истинность X_{n+1} доказывается в предположении истинности $X_0, \dots X_n$.

Теорема 5 (Метод полной математической индукции)**.** Дана последовательность высказываний X_n такая, что утверждение $\forall n \in \mathbb{N} \quad (\forall k \leqslant n \; X_n) \to X_{n+1}$ истинно. Тогда $\forall n \in \mathbb{N} \quad X_n$.

База индукции уже зашита в «усиленное» предположение индукции: истинность X_0 следует из истинности $\forall k < 0 \;\; X_k$, которое автоматически выполняется — не существует натуральных чисел, меньших 0. Далее истинность X_0 влечет истинность X_1. Так как $X_0 \wedge X_1$ истинно, то X_2 верно итд.

Задача I.12. Известно, что число $x + \dfrac{1}{x}$ является целым. Докажите, что при любом натуральном n число $x^n + \dfrac{1}{x^n}$ также является целым.

Решение. Воспользуемся методом полной математической индукции. Предполагая, что при всех $k \leqslant n$ число $x^k + \frac{1}{x^k}$ целое, покажем, что $x^{n+1} + \dfrac{1}{x^{n+1}}$ также целое. Действительно,

$$x^{n+1} + \frac{1}{x^{n+1}} = \left(x^n + \frac{1}{x^n}\right)\left(x + \frac{1}{x}\right) - \left(x^{n-1} + \frac{1}{x^{n-1}}\right)$$

В силу предположения индукции $x^n + \dfrac{1}{x^n}$, $x + \dfrac{1}{x}$ и $x^{n-1} + \dfrac{1}{x^{n-1}}$ целые, таким образом, $x^{n+1} + \dfrac{1}{x^{n+1}}$ также целое. □

Задача I.13. При каких $n > 3$ набор гирь с массами $1, 2, 3, \dots, n$ граммов можно разложить на три равные по массе кучки?

Решение. Чтобы гири можно было разделить требуемым образом, суммарная масса гирь должна делиться на 3. Сумма масс от 1 до n равна $n(n+1)/2$, и поэтому n может давать только остатки 0 или 2 при делении на 3.

Если можно разбить на три равные по массе кучки набор из n гирек, то и набор из $n+6$ гирек тоже можно разбить требуемым образом, поскольку гири массами $n+1, n+2, \dots, n+6$ легко разложить на три равные по массе кучки $((n+1)+(n+6) = (n+2)+(n+5) = (n+3)+(n+4))$.

Легко проверить, что если число гирек равно числу 5, 6, 8 или 9, то существуют требуемые разбиения:

- $1 + 4 = 2 + 3 = 5$;

- $1 + 6 = 2 + 5 = 3 + 4$;
- $1 + 2 + 3 + 6 = 4 + 8 = 5 + 7$;
- $1 + 2 + 3 + 4 + 5 = 6 + 9 = 7 + 8$.

Значит, разбить можно все наборы из количества гирь, дающего остатки 0 или 2 при делении на 3. □

4 Что читать дальше

Например, на фупме математическая логика разбита на два предмета. Алгебра логики будет в первом семестре, на ней будут разбираться таблицы истинности, нормальные формы и полиномы Жегалкина[8]. Стандартный учебник кафедры моу [18] вам в помощь. Непосредственно сама матлогика будет в третьем семестре в виде нечто под названием «Теория формальных систем». Функции классической матлогики можно задать не с помощью таблиц истинности, как было в этой главе, а с помощью набора аксиом, которым они удовлетворяют. «Теория формальных систем» развивает эту тему. Там будут формальные выводы, неклассическая логика и машины Тьюринга. Конспекты лекций оформлены в виде учебника.

С математической логикой можно ознакомиться и по другим книгам. «Введение в метматематику» [21] — неустаревающаяя классика Стивена Клини, учебник прямиком из старых добрых времен. Книга не очень проста в прочтении[9], вместо упражнений — оставленные автором для самостоятельного доказательства теоремы (обычно несложные), некоторые обозначения эндемичные — например, импликация обозначается $\supset$[10], но мы настоятельно рекомендуем ознакомиться с ней — все определения и конструкции объяснены подробно.

Если же подобное чтиво покажется вам сложным, то можно воспользоваться трехтомником Верещагина и Шеня «Лекции по математической логике». Первый том [11] посвящен теории множеств, второй [12] — собственно матлогике, а третий [13] — теории вычислимости. Шень известен своей нелюбовью к формализмам, так что здесь объяснения буквально на пальцах (а иногда даже и проиллюстрированы). Упражнений здесь много, почти все они совершенно

[8] Разумеется, никто даже не заикнется о том, зачем все это нужно, и к моменту, когда все это реально понадобится, а именно на курсе алгоритмов и моделей вычисления в четвертом семестре, все благополучно забывается.

[9] в те годы математики не стремились делать свои курсы детсадовскими по сложности, и переводчик Есенин-Вольпин отмечает «ее можно порекомендовать начинающему – при условии, что он не боится трудностей»

[10] Действительно, почему же старик Клини выбрал такое обозначение?

тривиальные. Некоторое отсутствие строгости, конечно, является помехой, но при первом ознакомлении с матлогикой этим, наверное, можно и пренебречь. Также можно посоветовать рассказ Шеня о математической строгости, затрагивающий в основном геометрические доказательства [37].

Чтобы подробнее ознакомиться с техниками математических доказательств, можно прочитать книжку Дэна Веллемана «Как это доказывать» [2]. Это не олимпиадная методичка — автор концентрируется на логической структуре доказательств, показывает примеры некорректных рассуждений и попутно объясняя необходимые понятия такие, как множества, функции итп. К каждой главе прилагаются упражнения, в целом тривиальные, но любой, кто хочет называться адекватом, должен уметь решать их все. По нашему скромному мнению «Как это доказать» Веллемана — то, что доктор прописал, и начинать надо именно с нее или чего-то очень на нее похожего. Книга пока есть только на английском, возможно, скоро появится русский перевод.

Если вы хотите ознакомиться подробнее с методами доказательств (принцип крайнего, раскраски итп), то любой хороший олимпиадный сборник или книжка в духе «Как решают нестандартные задачи» [20] Канель-Белова и Ковальджи будет полезна. Обычно олимпиадные сборники содержат только задачи и их решения, но некоторые вроде «ММО 1993-2005» (с греческими бегунами на обложке) содержат еще и методы решения этих задач. Некоторые соавторы данной книжки взяли на себя тяжелый труд агрегировать задачи с самых разных олимпиад[11] и пишут и издают «Пособие по олимпиадной математике» [4, 6] в нескольких частях (первая часть уже была издана).

Пожалуй, для начала хватит.

[11] Имеются в виду ММО, Всероссийская олимпиада по математике, Турнир городов итп — нормальные олимпиады, а не письменный экзамен по вступительной математике, законодательно загримированный под олимпиаду.

Глава II
Дискретный анализ

1 Теория чисел

1.1 Алгоритм Евклида

Одна из первых задач, которую вам, скорее всего, дадут на семинарах по информатике — реализовать алгоритм Евклида. Подразумевается, что вы все прошли его в шестом классе, и вам, наверняка, будет очень стыдно спрашивать, что же это такое... Этот алгоритм позволяет, например, найти НОД двух чисел. В первую очередь, приведём рассуждения, как можно его найти, пользуясь разложением на простые множители.

Основная теорема арифметики гласит: любое натуральное число можно единственным образом (с точностью до порядка) разложить на простые множители. Канонический вид такого разложения следующий:

$$N = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n},$$

где $p_1 < p_2 < \dots < p_n$ — простые числа, а $\alpha_1,\alpha_2,\dots,\alpha_n$ — натуральные.

Пусть $N = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$ и $M = p_1^{\beta_1} p_2^{\beta_2} \dots p_n^{\beta_n}$, тогда *наибольшим общим делителем* (сокращенно *НОД*) этих чисел будет число

$$(M,N) = p_1^{\min(\alpha_1,\beta_1)} p_2^{\min(\alpha_2,\beta_2)} \dots p_n^{\min(\alpha_n,\beta_n)}.$$

Числа M и N называются *взаимно простыми*, если их наибольший общий делитель равен 1. Достаточно очевидное следствие из основной теоремы арифметики: если какое-либо простое число участвует в разложениях на простые множители двух чисел, то эти числа не являются взаимно простыми.

Наименьшим общим кратных (сокращенно *НОК*) чисел M и N будет число

$$[M,N] = p_1^{\max(\alpha_1,\beta_1)} p_2^{\max(\alpha_2,\beta_2)} \dots p_n^{\max(\alpha_n,\beta_n)}.$$

Отсюда, в частности, следует формула

$$[M,N] \cdot (M,N) = M \cdot N.$$

Вернёмся теперь к **алгоритму Евклида**. В чем его суть? Если у нас есть пара чисел (a, b) $(a > b)$, то их НОД будет совпадать с НОД пары $(a-b, b)$. Обозначим $S = a+b$. Заметим, что при таком переходе сумма чисел уменьшается, но они все ещё неотрицательны, следовательно, их сумма не может стать отрицательной. В силу того, что S тоже делится на искомый НОД, то в какой-то момент она станет равна ему. Алгоритм Евклида, для нахождения НОД двух чисел, переведённый на язык современной математики, звучит примерно так: $(a,b) = (a-b,b)$, если $a > b$. Каждым его шагом будет вычитание из большего числа меньшего. Естественным его улучшением[12] является **обобщённый алгоритм Евклида**, в котором числа не вычитаются, а делятся друг на друга с остатком. Например, $(315,100) = (100,15) = (15,10) = (5,10) = (0,5) = 5$

Задача II.1. Найдите НОД чисел $\underbrace{11\ldots11}_{m}$ и $\underbrace{11\ldots11}_{n}$, $m > n$.

Решение. Воспользуемся алгоритмом Евклида с модификацией, то есть будем удалять из одного из чисел множители, точно не являющиеся делителями второго. Сначала вычтем из первого числа второе

$$(\underbrace{11\ldots1}_{n},\underbrace{11\ldots1}_{m}) = (\underbrace{11\ldots1}_{m-n}\underbrace{00\ldots0}_{n},\underbrace{11\ldots1}_{n}).$$

Заметим теперь, что НОД двух данных чисел не может делиться ни на 2, ни на 5, следовательно,

$$(\underbrace{11\ldots1}_{m-n}\underbrace{00\ldots0}_{n},\underbrace{11\ldots1}_{n}) = (\underbrace{11\ldots1}_{m-n},\underbrace{11\ldots1}_{n}).$$

Шаг модифицированного алгоритма — не что иное, как шаг обычного алгоритма Евклида, применённый к количеству единиц в числах. Так как в результате применения алгоритма Евклида получается НОД двух чисел (m,n), то в результате применения модифицированного алгоритма, получится число, состоящие из (m,n) единиц. □

Определение. *Диофантово уравнение* — уравнение, решением которого является набор целых чисел. Оно имеет следующий вид

$$F(x_1, x_2, \ldots, x_n) = 0, x_i \in \mathbb{Z} \ \ \forall i \in \overline{1,n}.$$

Обычно в диофантовых уравнениях (или системах) переменных больше, чем уравнений. Тем не менее, уравнения обычно имеют либо конечное число решений, либо серию решений, зависящую от параметра. Примеры диофантовых уравнений: $x^3 + y^3 = z^3$; $7x + 11y = 15$; $2^x - 1 = t^2$.

[12] Сравните время работы обычного и обощенного алгоритма Евклида, например, на парах чисел вида 2^n и 2.

Пожалуй, одной из самых известных теорем математики является: *Великая теорема Ферма*, которая утверждает, что не существует решений в натуральных числах у уравнения

$$x^n + y^n = z^n, n \geqslant 3$$

таких, что $xyz \neq 0$.

Не правда ли, очень короткая и красивая формулировка? Несмотря на это, на доказательство этой теоремы ушло более 350 лет, и окончательно она была доказана чуть более 20-ти лет назад. Полное доказательство занимает внушительный объём: около 500 страниц. В попытках доказать теорему Ферма математики существенно развили теорию колец[13], алгебраическую геометрию и теорию модулярных форм.

Расширенный алгоритм Евклида отлично подходит для решения простейших диофантовых уравнений. На примере следующей задачи покажем его метод работы.

Задача II.2. Решить уравнение в целых числах:

$$7x + 10y = 16.$$

Решение. 1) Поделим 10 на 7 с остатком: получим 1 в неполном частном и 3 в остатке: $10 = 7 \cdot 1 + 3$. Тогда $7x + 7y + 3y = 16$ или $7(x + y) + 3y = 16$. Введём новую переменную: $z = x + y$. Тогда $7z + 3y = 16$.

2)Поделим 7 на 3 с остатком: получим 2 в неполном частном и 1 в остатке: $7 = 3 \cdot 2 + 1$. Тогда $3 \cdot 2 \cdot z + z + 3y = 16$ или $3(2z + y) + z = 16$. Введём новую переменную: $t = 2z + y$. Тогда $3t + z = 16$.

3) Один из коэффициентов полученного уравнения равен единице, поэтому $z = 16 - 3t$ для любого t — целого числа. Возвращаемся обратно к исходным переменным: $t = 2z + y \Rightarrow y = t - 2z = t - 2(16 - 3t) = 7t - 32$; $z = x + y \Rightarrow x = z - y = 16 - 3t - (7t - 32) = 48 - 10t$.

Окончательно ответ записывается в виде: $(x,y) = (48-10t,7t-32)$ для любого целого t. □

[13]Теорему Ферма для $n = 3$ можно доказать, заявив, что $x^3 = z^3 - y^3$ и рассмотрев кольцо чисел Эйзенштейна, в котором $z^3 - y^3$ раскладывается на линейные по y и z множители, а затем прийти к противоречию, пользуясь единственностью разложения чисел Эйзенштейна на множители. Ламе в 1846 году попытался обобщить это доказательство на все n, но оказалось, что соответствующие кольца могут быть не факториальны, то есть разложение на простые не обязательно единственно.

1.2 Сравнения по модулю

Определение. Говорят, что числа a и b *сравнимы по модулю* x, если их разница делится на x. Записывают это как $a \equiv b \pmod{x}$, или как $a = b \pmod{x}$.

Нетрудно убедиться в том, что следующее утверждение равносильно определению:

Числа a и b сравнимы по модулю x, тогда и только тогда, когда они имеют одинаковый остаток при делении на x.

Сравнивать по модулю можно также и отрицательные числа:

Например, $13 \equiv 7 \pmod{3}$ или $11 \equiv -3 \pmod{7}$.

Если $a \equiv 0 \pmod{x}$, то a делится на x.

Со сравнениями по модулю можно работать так же, как и с обычными равенствами: их можно складывать, умножать и возводить в степень: если $a \equiv b \pmod{x}$ и $c \equiv d \pmod{x}$, то

1. $a + c \equiv b + d \pmod{x}$;
2. $a \cdot c \equiv b \cdot d \pmod{x}$;
3. $a^n \equiv b^n \pmod{x}$ для любого натурального n.

Так как речь идёт о целых числах, то о делении говорить не будем, хотя его в ряде случаев можно определить как операцию, обратную умножению: например, так как $3 \cdot 4 \equiv 2 \pmod{5}$, то $2/3 \equiv 4 \pmod{5}$.

Порой использование сравнений по модулю позволяет существенно упростить вычисления. Рассмотрим это на примере:

Задача II.3. Найти остаток от деления $2013 \cdot 2014 \cdot 2015 + 2016^3$ при делении на 7.

Решение. Найдём остаток от деления 2013 на 7. Разумеется, это можно сделать, использовав деление в столбик, но мы, как абитуриенты или уже студенты ФУПМа, вспомним важный факт (который использовался в доказательстве многих признаков делимости в углублённой школьной программе): $1001 = 7 \cdot 11 \cdot 13$, откуда число 2002 делится на 7, то есть $2002 \equiv 0 \pmod{7}$, но $11 \equiv 4 \pmod{7}$, откуда $2002 + 11 \equiv 0 + 4 = 4 \pmod{7}$. Значит $2014 \equiv 5 \pmod{7}$ и $2015 \equiv 6 \pmod{7} \Rightarrow 2013 \cdot 2014 \cdot 2015 \equiv 4 \cdot 5 \cdot 6 = 120 \equiv 1 \pmod{7}$ $2016 \equiv 0 \pmod{7}$, а значит $2016^3 \equiv 0^3 \pmod{7}$. Окончательно: $2013 \cdot 2014 \cdot 2015 + 2016^3 \equiv 1 + 0 = 1 \pmod{7}$. □

В некоторых случаях удобнее перейти к отрицательным числам: например $8^{100} \equiv (-1)^{100} = 1 \pmod 9$. Прошлая задача, кстати, не исключение: если 2016 делится на 7, то $2013 \cdot 2014 \cdot 2015 \equiv (-3) \cdot (-2) \cdot (-1) \equiv -6 \equiv 1 \pmod 7$.

Когда мы используем сравнения по модулю, многие из известных признаков делимости открываются нам с новой стороны. Например, признаки делимости на 9 и на 3 выглядят следующим образом:

$$\overline{a_n a_{n-1} \dots a_2 a_1 a_0} \equiv a_n + a_{n-1} + \dots a_2 + a_1 + a_0 \pmod 9$$

и

$$\overline{a_n a_{n-1} \dots a_2 a_1 a_0} \equiv a_n + a_{n-1} + \dots a_2 + a_1 + a_0 \pmod 3.$$

Одним из методов решения задач на делимость является перебор остатков. В простых задачах найти модуль, по которому перебираются остатки, не составляет труда, в более сложных — до него ещё нужно догадаться.

Задача II.4. Доказать, что ни при каких целых n число n^2+3n+4 не делится на 9.

Решение. Целое число может давать остатки 0,1,2,...,8 при делении на 9. Переберём все эти случаи. Для удобства запишем остатки в виде таблицы:

$n \pmod 9$	0	1	2	3	4	5	6	7	8
$n^2 \pmod 9$	0	1	4	0	7	7	0	4	1
$3n \pmod 9$	0	3	6	0	3	6	0	3	6
$n^2+3n+4 \pmod 9$	4	8	5	4	5	8	4	2	2

Рассмотрим, например, случай, когда n даёт остаток 5 при делении на 9. Тогда $n \equiv 5 \pmod 9 \Rightarrow n^2 + 3n = 4 \equiv 5^2 + 3 \cdot 5 + 4 = 25 + 15 + 4 \equiv 7 + 6 + 4 = 17 \equiv 8 \pmod 9$; остальные случаи разбираются аналогично. Видим, что ни в одном случае не получился остаток 0, а это значит, что данное выражение не делится на 9 ни при каких целых n.

□

Задача II.5. Может ли число, составленное из 13 двоек, 13 троек, 13 четвёрок и 13 пятёрок быть полным квадратом?

Решение. Докажем, что это невозможно. Так как речь в задаче идёт о числе, цифры которого известны, но не известен их порядок, воспользуемся признаком делимости на 3. Обозначим число за N.

Тогда сумма цифр числа N равна $13(2 + 3 + 4 + 5)$. По признаку делимости, получаем: $N \equiv 13 \cdot 14 \equiv 1 \cdot 2 = 2 \pmod 3$.

Посмотрим, может ли квадрат натурального числа давать остаток 2 по модулю 3, как и в предыдущей задаче разберём случаи:

$x \pmod 3$	0	1	2
$x^2 \pmod 3$	0	1	1

Делаем вывод: квадрат натурального числа не может давать остаток 2 по модулю 3, значит, составить квадрат из всех данных цифр невозможно.

□

2 Комбинаторика

2.1 Перебор случаев

Что такое комбинаторика? Грубо говоря, это математическая наука об подсчёте числа способов сделать что-либо. Соответственно, главный вопрос, по которому можно опознать комбинаторную задачу в рамках школьной или олимпиадной математики — это вопрос «Сколько?».

Многие условия комбинаторных задач кажутся достаточно очевидными, но, тем не менее, ошибок при их решении допускается очень много, в основном данные ошибки связаны с пониманием условия задачи.

Самый простой метод решения комбинаторных задач — это, как бы не глупо и просто звучало, метод перебора случаев.

Например, пусть хулиган Вася пытается кинуть один из предметов, лежащих у него в карманах (телефон, пакет семечек, носовой платок или бутылку кока-колы) в окно своего друга Пети. Очевидно, что тут у него всего четыре способа. Если же он считает, что одного ненужного предмета Пете будет мало, и надо кинуть два предмета, то способов будет уже 6:

1. телефон, пакет семечек;
2. телефон, бутылка кока-колы;
3. телефон, носовой платок;
4. пакет семечек, носовой платок;
5. пакет семечек, бутылка кока-колы;
6. носовой платок, бутылка кока-колы.

Когда случаев становится больше, то очень легко или упустить какой-то из случаев, или написать какой-то дважды, но, в то же время, после сведения задач к более простым подзадачам гораздо проще просто представить оставшиеся варианты, чем вспоминать или придумывать нужную формулу.

Например, к одной из таких задач можно отнести следующую задачу:

Задача II.6. Сколько существует трёхзначных чисел, сумма цифр которых равна 3?

Решение. Выписав числа в порядке возрастания, получим:

102, 111, 120, 201, 210, 300.

То есть, всего чисел 6. □

Также можно вспомнить классическую задачу о беспорядках:

Задача II.7. Три гостя при входе в ресторан отдали швейцару свои шляпы, а при выходе получили их обратно. Невнимательный швейцар раздал шляпы случайным образом. Сколько существует вариантов, при которых каждый гость получил чужую шляпу?

Решение. Занумеруем трёх гостей номерами 1, 2, 3. Пусть у каждого гостя есть шляпа с его же номером. Тогда первый может надеть только шляпу 2 или 3. Если первый надел шляпу 2, то третьему достанется только шляпа 1, а второму шляпа 3. Аналогично, если первый надел шляпу 3, то второму достанется только шляпа 1, а третьему шляпа 2. □

На самом деле, данная задача для большего числа гостей решается уже настоящими комбинаторными методами, и их мы тоже затронем.

В некоторых задачах выписывать все варианты не очень хочется, да и вообще проверяющие не совсем поймут данный шаг, но, тем не менее, почти очевидно, сколько их на самом деле.

Рассмотрим одну из задач с заочной олимпиады «Физтех» 2013-го года.

Задача II.8. Сколько пар натуральных чисел удовлетворяет равенству $2x + 5y = 90000$?

Решение. Так как $5y$ и 90000 делятся на 5, то $2x$ тоже должно делиться на 5. Так как 2 простое, то, значит, x можно представить как $5a$. Рассмотрев аналогично делимость на 2, получим, что $y = 2b$. После сокращения на 10, уравнение свелось к $a + b = 9000$. Так как числа натуральные, то для любого a от 1 до 8999, b определяется однозначно. Ввиду линейности замены (так как

каждый корень определяется из замены однозначно через систему линейных уравнений), новых корней не получится, а, значит, всего пар 8999. □

2.2 «И» или «ИЛИ»

Перейдём к более интеллектуальным методам решения задач.

1. Правило сложения: если объект A можно выбрать n способами, а объект B можно выбрать m способами, то выбрать объект A **или** B можно $m+n$ способами.

2. Правило умножения: если объект A можно выбрать n способами, а объект B можно выбрать m способами, то выбрать пару объектов A **и** B можно $m \cdot n$ способами.

Если вы новичок в этой теме, то при решении задачи следует каждый раз проговаривать про себя — какой союз нужно поставить, «и» или «или», и умножать или складывать соответственно.

Задача II.9. У хулигана Васи есть 5 разных камней, один из которых он выбирает и бросает в одно из 4 окон квартиры своего друга Пети. Сколькими способами он может это сделать?

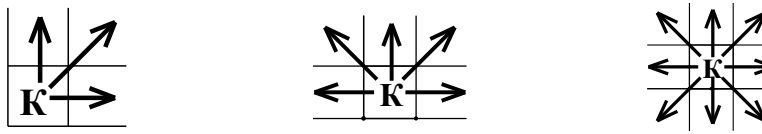
Решение. Пользуемся правилом умножения: нужно выбрать камень, который он бросает, **и** окно, в которое он его бросает. Итого получается $5 \cdot 4 = 20$ способов. □

Задача II.10. Сколькими способами на шахматной доске можно расставить двух королей — чёрного и белого, чтобы они не били друг друга.

Решение. Попробуем воспользоваться правилом умножения: первого короля можно расставить 64 способами - на любое поле. Но вот уже количество способов расстановки второго короля будет зависеть от того, куда мы поставили первого короля: в угол, на границу или на остальные поля.

Поэтому сначала придётся воспользоваться правилом сложения: первый король стоит в углу **или** на границе **или** на остальных полях.

В первом случае король бьёт три поля, и ещё на одном стоит, поэтому второго короля можно поставить $64 - 4 = 60$ способами. Так как угловых поля всего 4, то расставить двух королей в этом случае мы можем $4 \cdot 60$ способами — расставляем первого короля 4 способами **и** второго короля — 60 способами.



а) б) в)

Рис. 2.1: Различные способы расстановки короля.

Граничных полей всего 24, значит, первого короля мы можем поставить 24 способами **и** второго короля — $64 - 6 = 58$ способами (король, стоящий на границе, бьёт 5 полей и ещё одно поле занимает). То есть в этом случае есть $24 \cdot 58$ способов.

В третьем случае способов $36 \cdot 55$, по аналогичным соображениям. Итого, получается $4 \cdot 60 + 24 \cdot 58 + 36 \cdot 55 = 3612$ способов.

□

Сформулируем ещё одно важное правило:

3. Правило деления: если каждый способ при подсчёте был учтён n раз, то результат нужно поделить на n.

В разобранных задачах таких моментов ещё не возникало — каждый способ там был подсчитан ровно по 1 разу. Что нельзя сказать про следующую задачу:

Задача II.11. Кошка поймала 20 мышек. Сколькими способами она может выбрать себе ужин, если на ужин она съедает 2 мышки?

Решение. Воспользуемся правилом умножения: первую мышку она может выбрать 20 способами и вторую — 19 способами, так как одна мышка уже выбрана. Значит ответ — $19 \cdot 20$? А вот и нет. Дело в том, что мы каждый способ выбора мышек подсчитали по 2 раза. Действительно, рассмотрим двух мышек — Васю и Петю. Есть два способа: сначала выбран Вася, потом Петя, или наоборот — сначала Петя, а потом Вася. Поэтому искомое количество способов равно $19 \cdot 20/2 = 190$.

□

2.3 Размещения и сочетания

После того, как мы упомянули правило деления, можно переходить уже и к чему-то выглядящему посолиднее и вводить обозначения.

Пусть имеется множество, содержащее n элементов. Произвольный упорядоченный набор, составленный из k различных элементов данного множества, называется *размещением* из n элементов по k элементов (или просто размещением из n по k). Число размещений из n элементов по k элементов обозначается A_n^k (читается A из эн по ка). Это число упорядоченных наборов из k элементов (или число цепочек длины k), выбранных из n-элементного множества. Например, это количество способов составить бутерброд из ровно 3 предметов из списка «Хлеб, Колбаса, Варенье, Сало, Сыр и Сгущёнка».

Введём понятие *числа сочетаний* из n объектов по k объектов. По определению C_n^k (читается как «Цэ из эн по ка»[14]) — это количество способов выбрать k объектов из n. Например, это может быть количество способов взять 3 книги с полки, на которой стоит 10 книг, или выбрать 4 сорта колбасы из 12 для бутерброда.

Найдём число сочетаний C_n^k, пользуясь правилами умножения и деления. Первый объект можно выбрать n способами. Второй объект уже можно выбрать $n-1$ способами, так как один объект уже выбран, следующий — $n-2$, и так далее. Последний, k-ый объект может быть выбран $n-k+1$ способами. Все эти числа нужно перемножить, потому что мы выбираем первый объект **и** второй **и** третий, и так далее. Но полученный результат нужно поделить на $k!$, так как каждый способ посчитан именно столько раз (это количество способов переставить выбранные объекты). Итого, получаем $C_n^k = \dfrac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{k!} = \dfrac{n!}{k!\cdot(n-k)!}$.

Теперь мы можем ответить на вопрос о выборе книг: существует $C_{10}^3 = \dfrac{10\cdot 9\cdot 8}{6} = 120$ способов выбрать 3 книги из 10.

В частности, $C_n^0 = 1; C_n^1 = n; C_n^2 = \dfrac{n(n-1)}{2}$.

Формулы сокращённого умножения $(a+b)^2 = a^2+2ab+b^2$ и $(a+b)^3 = a^3+3a^2b+3ab^2+b^3$ проходят в школе. А что произойдёт, если возводить в четвёртую, в пятую, и так далее степени?

Оказывается, верна формула:

$$(a+b)^n = C_n^n a^n b^0 + C_n^{n-1} a^{n-1} b^1 + \cdots + C_n^1 a^1 b^{n-1} + C_n^0 a^0 b^n.$$

Приведённая формула называется *биномом Ньютона*, именно поэтому числа сочетаний называют также биномиальными коэффициентами.

[14] В англоязычной литературе почти всегда используется обозначение $\binom{n}{k}$.

Треугольник Паскаля — это способ записи биномиальных коэффициентов, приведенный на рисунке 2.2).

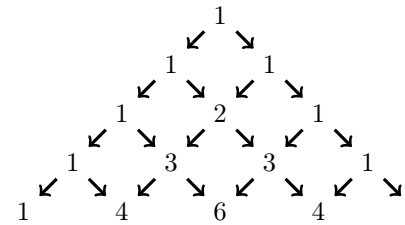


Рис. 2.2: Треугольник Паскаля.

Нетрудно заметить, что он обладает следующим свойством: каждый элемент в нём равен сумме двух элементов, которые стоят сверху слева и сверху справа от него. Докажем это свойство: $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$. По определению, C_n^k это количество способов выбрать k объектов из n. Все эти способы можно разбить на два случая:

1. Объект номер 1 был выбран, тогда из оставшихся $n-1$ объектов нужно выбрать $k-1$, это по определению C_{n-1}^{k-1} способов.
2. Объект номер 1 выбран не был, тогда из оставшихся $n-1$ объектов нужно выбрать k, это по определению C_{n-1}^k способов.

Так как возможен первый **или** второй случаи, то количество способов нужно сложить, что завершает доказательство.

Помимо приведённого свойства, треугольник Паскаля обладает ещё рядом красивых свойств.

Задача II.12. Игральная колода состоит из 52 карт. Какое есть количество способов выбрать 6 карт, чтобы среди них встречались карты всех мастей?

Решение. Заметим, что есть два случая выбора мастей карт:

1. Какой-то масти было выбрано 3, а остальных по 1.
2. Каких-то двух мастей было выбрано по 2, а остальных по 1.

Найдём количество способов выбрать карты в первом случае. Сначала нужно выбрать «лидирующую» масть — ту, которой должно быть 3 карты. Это можно сделать $C_4^1 = 4$ способами. Теперь нужно выбрать 3 карты этой лидирующей масти. Это можно сделать C_{13}^3 способами, так как каждой масти по 13 карт. И наконец, каждую из оставшихся мастей можно выбрать C_{13}^1 способами. Перемножая все полученные числа, получаем $C_4^1 \cdot C_{13}^3 \cdot (C_{13}^1)^3$.

Количество способов во втором случае считается аналогично: способов выбрать две «лидирующие» масти C_4^2, по две карты каждой из этих мастей — C_{13}^2 и оставшиеся по C_{13}^1, итого $C_4^2 \cdot (C_{13}^2)^2 \cdot (C_{13}^1)^2$.

Значит, общее число способов равно $C_4^1 \cdot C_{13}^3 \cdot (C_{13}^1)^3 + C_4^2 \cdot (C_{13}^2)^2 \cdot (C_{13}^1)^2$.

Как правило, в задачах на комбинаторику не обязательно доводить задачу до численного ответа, формул без многоточий вполне достаточно.

□

Задача II.13. Сколькими способами можно выбрать из полной колоды (52 карты) 10 карт так, чтобы

а) среди них был ровно один туз?

б) среди них был хотя бы один туз?

в) среди них была хотя бы одна карта бубновой масти?

Решение. а) Вначале выберем какого-либо туза. Его можно выбрать 4 способами. Далее осталось выбрать 9 карт из $52 - 4 = 48$ карт, не являющимися тузами. Это можно осуществить C_{48}^9 способами. Значит, всего способов будет $4 \cdot C_{48}^9$.
б) Число способов, в которых был выбран хотя бы один туз, равно числу всех способов, за исключением тех способов, когда ни один туз выбран не был. Всего есть C_{52}^{10} способов выбрать 10 карт из колоды, из которых C_{48}^{10} способов не содержат туза. Поэтому искомое число способов равно $C_{52}^{10} - C_{48}^{10}$.
в) Решение этого пункта аналогично решению предыдущего. Всего есть C_{52}^{10} способов выбрать 10 карт, из которых в C_{52-13}^{10} отсутствует бубновая масть. Итого, получается $C_{52}^{10} - C_{39}^{10}$ способов.

□

Задача II.14. У Васи дома есть чай, кофе, какао, апельсиновый и яблочный соки, вода и кока-кола. Он решил составить себе меню напитков на завтрак, обед и ужин на 4 дня так, чтобы наборы напитков не совпадали (Васе не важно, на какой приём пищи он пил, например, чай, важно только, какие напитки он пил в течение дня). Сколькими способами он может это сделать?

Решение. Узнаем, сколько у Васи есть способов выбрать себе 3 напитка в день. Всего есть 7 вариантов напитков, поэтому способов будет C_7^3. Во второй день он не может выбрать набор, который он выбрал в первый день, поэтому у него остаётся $C_7^3 - 1$ способ. В третий день у него есть $C_7^3 - 2$ способов, и в четвёртый — $C_7^3 - 3$. Итого, у Васи есть $C_7^3 \cdot (C_7^3 - 1) \cdot (C_7^3 - 2) \cdot (C_7^3 - 3)$ способа составить себе меню. □

2.4 Задача о шарах и перегородках

Рассмотрим далее следующие классические задачи комбинаторики:

Задача II.15. Имеется 7 ящиков, занумерованных номерами от 1 до 7. Сколькими способами в эти ящики можно разложить 25 одинаковых шаров так, чтобы никакой ящик не оказался пустым?

Решение. Положим все 25 шаров в ряд. Для того, чтобы определить, какие шары в каком ящике лежат, нужно поставить 6 перегородок между шарами. Все шары разобьются на 7 групп: шары первой группы положим в первый ящик, второй группы — во второй, и так далее. Всего есть 24 места, на которые можно поставить перегородки — именно столько промежутков между шарами. Никакие две перегородки нельзя ставить на одно место, иначе тогда в каком-то ящике не будет ни одного шара. Значит, искомое количество способов — C_{24}^6.

□

Задача II.16. Имеется 7 ящиков, занумерованных номерами от 1 до 7. Сколькими способами в эти ящики можно разложить 25 одинаковых шаров (на этот раз ящики могут оставаться пустыми)?

Приведём два решения этой задачи:

Решение 1. Зарезервируем $25 + 6 = 31$ место для шаров и перегородок. На 6 любых мест из этих 31 поставим перегородки. Докажем, что любая такая расстановка будет однозначно определять расположения шаров по ящикам. Действительно, все шары, которые лежат до первой перегородки, положим в первый ящик, находящиеся между первой и второй перегородками — во второй ящик, и так далее. При этом если перегородки стоят рядом, значит, ящик остался пустым, что допускается. Поэтому искомое количество способов — это C_{31}^6 — выбрать 6 мест из 31 для перегородок, а шары ставятся однозначно.

Решение 2. Сопоставим каждому расположению 25 шаров по ящикам такое расположение 32 шаров по тем же ящикам, чтобы при этом не оставалось пустых ящиков: добавим в каждый ящик по одному шару. Такое соответствие будет взаимно-однозначным, поэтому и количество способов у них совпадает. Но количество способов разложить шары, чтобы не было пустых ящиков, мы считали в предыдущей задаче: их C_{31}^6.

Отметим, что построение взаимно-однозначного соответствия используется во многих разделах математики.

Число способов, которыми можно разложить m одинаковых шаров по n различным ящикам, называется *числом сочетаний с повторениями* из n по m и обозначается $\overline{C_n^m}$.

Таким образом, $\overline{C_n^m} = C_{n+m-1}^{n-1}$.

Задача II.17. 20 человек голосуют по 5 предложениям. Сколькими способами могут распределиться голоса, если каждый голосует только за одно предложение, и учитывается лишь количество голосов, поданных за каждое предложение?

Решение. Приведём взаимно-однозначное соответствие между этой задачей и задачей о шарах и перегородках. Пусть 5 предложений соответствуют 5 ящикам, голоса — это шары. Так как важно лишь количество голосов, поданных за каждое предложение, то важно лишь количество шаров, которые попали в тот или иной ящик. То есть нужно найти количество способов разместить 20 шаров по 5 ящикам, при этом ящики могут оставаться пустыми — например, если по какому-то предложению не было поданных голосов. Искомое число способов равно $C_{20+5-1}^{5-1} = C_{24}^{4}$.

□

2.5 Формула включений-исключений

Перейдём к так называемой формуле включений-исключений.

Пусть рассматривается некоторый набор объектов. Этому набору соответствует некоторый набор свойств $\alpha_1,\alpha_2,\ldots,\alpha_n$. Каждый объект может как обладать любым из указанных свойств, так и не обладать. Например, если α_1 — свойство делимости числа на 3, а α_2 — свойство числа быть полным квадратом, то число 16 не обладает первым свойством, но обладает вторым. Пусть N — общее количество объектов (конечное число), $N(\alpha_k)$ — количество объектов, обладающих k-ым свойством (при этом эти объекты могут как обладать остальными свойствами, так и не обладать). Аналогично, пусть $N(\alpha_k,\alpha_l)$ — количество объектов, обладающих k-ым и l-ым свойствами. Подобным образом определяем количество объектов, обладающих сразу тремя, четырьмя, и так далее, вплоть до всех n, свойствами. Тогда количество объектов, которые не обладают ни одним из свойств (обозначается как $N(\overline{\alpha_1},\overline{\alpha_2},\ldots,\overline{\alpha_n})$) можно найти по формуле:

$$N(\overline{\alpha_1},\ldots,\overline{\alpha_n}) = N - N(\alpha_1) - N(\alpha_2) - \cdots - N(\alpha_n) +$$
$$+N(\alpha_1,\alpha_2) + N(\alpha_1,\alpha_3) + \cdots + N(\alpha_{n-1},\alpha_n) -$$
$$-N(\alpha_1,\alpha_2,\alpha_3) - \ldots + (-1)^n \cdot N(\alpha_1,\alpha_2,\ldots\alpha_n),$$

которая называется *формулой включений-исключений*. Это название говорит само за себя: первое слагаемое идёт с плюсом: мы его включаем, следующим с минусом — мы их исключаем, потом снова включаем, и так далее.

При решении задачи нужно понять, какие именно свойства нужно рассматривать — нужно, чтобы эти свойства были бинарными: задав вопрос, обладает

ли объект свойством, должен получаться один из ответов «да» или «нет». Например, делимость на 3 таковым свойством является, а вот остаток при делении на 3 уже нет. Рассмотрим классическую задачу на формулу включений-исключений: посчитаем количество счастливых билетов.

Задача II.18. Билет — это последовательность из 6 цифр. Билет называется счастливым, если сумма его первых трёх цифр равна сумма последних трёх. Какое существует количество счастливых билетов?

Решение. Напомним задачу о шарах и перегородках: есть n одинаковых шаров и m различных ящиков. Тогда существует C_{n+m-1}^{m-1} способов разложить шары по ящикам. Переформулируем условие: имеется уравнение $x_1+x_2+..+x_m = n$ и требуется определить количество решений этого уравнения в целых неотрицательных числах. Ответ в этой задаче, а называется она задачей Муавра, такой же: C_{n+m-1}^{m-1}. В этом нетрудно убедиться. Построим взаимно-однозначное соответствие между разложением шаров по ящикам и решением уравнения: первое слагаемое шаров кладём в первый ящик, второе — во второй и так далее.

Вернёмся к задаче о счастливых билетах: нужно найти количество таких наборов цифр $\{a_1,a_2,a_3,a_4,a_5,a_6\}$, для которых $a_1+a_2+a_3 = a_4+a_5+a_6$. Вместо рассматриваемого набора будем рассматривать набор $\{a_1,a_2,a_3,9-a_4,9-a_5,9-a_6\}$. Количество таких наборов совпадает, так как между этими множествами можно установить взаимно-однозначное соответствие. Для рассматриваемого набора, для удобства назовём его $\{b_1,b_2,b_3,b_4,b_5,b_6\}$ должно выполняться свойство: $b_1+b_2+b_3+b_4+b_5+b_6 = 27$.

При этом все числа $b_1,b_2,\ldots,b_6$ должны быть не больше 9. Поэтому логичным будет ввести свойство α_i, которое означает, что $b_i > 9$. Тогда:

$$N(\overline{\alpha_1},\ldots,\overline{\alpha_6}) = N - C_6^1 \cdot N(\alpha_1) + C_6^2 \cdot N(\alpha_1,\alpha_2).$$

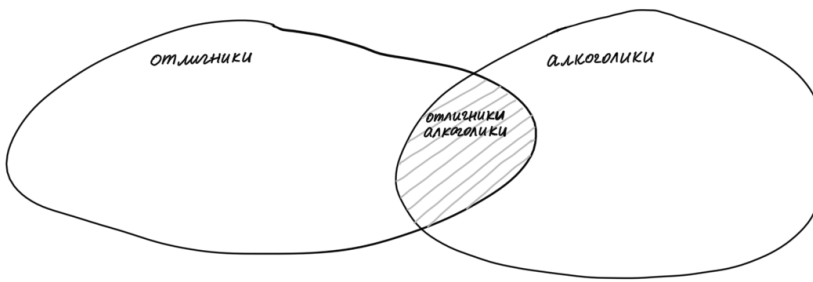
Тут мы воспользовались тем, что больше трёх чисел не могут быть больше 9, а также симметрией. $N = C_{32}^5$, это задача Муавра. Найдём теперь $N(\alpha_1)$. Оно равно количеству решений системы:

$$\begin{cases} b_1+b_2+b_3+b_4+b_5+b_6 = 27 \\ b_1 > 9 \end{cases}$$

в целых неотрицательных числах. Установим взаимно-однозначное соответствие, сделав замену $x = b_1 - 10$, тогда нужно найти количество решений уравнения $x+b_2+b_3+b_4+b_5+b_6 = 17$, это снова задача Муавра, и количество решений будет C_{22}^5. Окончательно,

$$N = C_{32}^5 - 6 \cdot C_{22}^5 + 15 \cdot C_{12}^5 = 55252$$

В принципе, формула включений-исключений достаточно заметна при изображении диаграммы Эйлера, особенно для малого набора свойств. Диаграммой Эйлера (кругами Эйлера-Венна, диаграммой Венна) называется схематичное изображение всех возможных пересечений нескольких множеств. □



Задача II.19. Пусть $N = p_1^{\beta_1} p_2^{\beta_2} p_3^{\beta_3}$. Найти количество чисел, меньших N и взаимно простых с ним.

Решение. Обозначим свойством α_1 делимость числа на β_1, аналогично введём свойства α_2 и α_3.

Число будет взаимно простым с числом N, если оно не имеет с ним ни одного общего делителя, то есть не обладает ни одним из свойств α_1,α_2 и α_3, поэтому требуется найти $N(\overline{\alpha_1},\overline{\alpha_3},\overline{\alpha_3})$.

По формуле включений-исключений для трёх свойств:

$$N(\overline{\alpha_1},\overline{\alpha_3},\overline{\alpha_3}) = N - N(\alpha_1) - N(\alpha_2) - N(\alpha_3) +$$
$$+N(\alpha_1,\alpha_2) + N(\alpha_2,\alpha_3) + N(\alpha_1,\alpha_3) - N(\alpha_1,\alpha_2,\alpha_3).$$

Найдём $N(\alpha_1)$: количество чисел, меньших N, и делящихся на p_1: $N(\alpha_1) = \dfrac{N}{p_1}$, каждое p_1-ое число делится на p_1. Аналогично, $N(\alpha_2) = \dfrac{N}{p_2}$; $N(\alpha_3) = \dfrac{N}{p_3}$;

$N(\alpha_1,\alpha_2) = \dfrac{N}{p_1p_2}$; $N(\alpha_1,\alpha_3) = \dfrac{N}{p_1p_3}$; $N(\alpha_2,\alpha_3) = \dfrac{N}{p_2p_3}$; $N(\alpha_1,\alpha_2,\alpha_3) = \dfrac{N}{p_1p_2p_3}$.
Отсюда:

$$N(\overline{\alpha_1},\overline{\alpha_3},\overline{\alpha_3}) = N - \frac{N}{p_1} - \frac{N}{p_2} - \frac{N}{p_3} + \frac{N}{p_1p_2} + \frac{N}{p_1p_3} + \frac{N}{p_2p_3} - \frac{N}{p_1p_2p_3} =$$

$$= N\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\left(1 - \frac{1}{p_3}\right),$$

что и требовалось найти.

□

Задача II.20. (Задача о беспорядках). Войдя в ресторан, 5 гостей оставили швейцару свои шляпы, а на выходе получили их обратно. Швейцар раздал шляпы случайным образом. Сколько существует вариантов, при которых каждый гость получит чужую шляпу?

Решение. Пусть свойство α_k будет означать, что гость номер k получил свою шляпу. Тогда требуется найти $N(\overline{\alpha_1},\overline{\alpha_2},\overline{\alpha_3},\overline{\alpha_4},\overline{\alpha_5})$ — количество способов, в которых ни один из гостей не получил свою шляпу.

Всего существует 5! способов раздать шляпы людям, не обращая внимания, кто из них чью получил: $N = 5!$.

Найдём $N(\alpha_1)$: количество способов, когда первый получил свою шляпу. Тогда требуется раздать остальные 4 шляпы 4 людям, что можно сделать 4! способами. Аналогично $N(\alpha_2) = N(\alpha_3) = N(\alpha_4) = N(\alpha_5) = 4!$.

$N(\alpha_1,\alpha_2)$ это количество способов раздать шляпы так, чтобы первый и второй получили свои шляпы. Тогда нужно раздать 3 оставшиеся шляпы 3 людям, что можно сделать 3! способами. Аналогично, $N(\alpha_k,\alpha_j) = 3!$ для любой пары человек. Всего существует C_5^2 пар человек.

$N(\alpha_1,\alpha_2,\alpha_3) = 2$, и существует C_5^3 троек человек;
$N(\alpha_1,\alpha_2,\alpha_3,\alpha_4)$=1, и всего существует C_5^4 четвёрок человек;
$N(\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5) = 1$.

Теперь можно записать формулу включений-исключений:

$$N(\overline{\alpha_1},\overline{\alpha_2},\overline{\alpha_3},\overline{\alpha_4},\overline{\alpha_5}) = N - C_5^1N(\alpha_1) + C_5^2N(\alpha_1,\alpha_2) - C_5^3N(\alpha_1,\alpha_2,\alpha_3)+$$

$$+C_5^4N(\alpha_1,\alpha_2,\alpha_3,\alpha_4) - C_5^5N(\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5) =$$

$$= 120 - 5 \cdot 24 + 10 \cdot 6 - 10 \cdot 2 + 5 \cdot 1 - 1 = 44,$$

что и требовалось найти. □

2.6 Рекурренты в комбинаторике

Задача II.21. Спускаясь каждое утро по лестнице, Вася заметил, что она содержит 10 ступенек (он находится на нулевой, и ему нужно попасть на десятую). Сколькими способами Вася может по ней спуститься, если

(а) он может перепрыгивать через любое количество ступенек?

(б) он может наступать только на следующую ступеньку или через ступеньку?

Решение. **(а)** На последней ступеньке Вася обязательно должен побывать. На остальных же ступеньках, а их девять штук, он может как побывать, так и пропустить их — для каждой ступеньки есть 2 способа. Значит всего есть $2^9 = 512$ способов спуститься по лестнице.

(б) Попробуем решить более общую задачу: пусть на лестнице n ступенек. Тогда если $n = 1$, то способ спуститься только один. Если $n = 2$, то существует 3 способа спуститься: наступив на промежуточную ступеньку, или не наступив. Пусть X_n — количество способов спуститься с лестницы, содержащей n ступенек. Мы показали, что $X_1 = 1$; $X_2 = 2$. Найдём, чему равно X_n. С первый шагом у Васи есть два выбора:

- Он может наступить на следующую ступеньку, тогда ему ещё нужно пройти $n - 1$ ступеньку, он может сделать это X_{n-1} способом.
- Он может пропустить следующую ступеньку, тогда ему нужно пройти ещё $n - 2$ ступеньки, это можно сделать X_{n-2} способами.

Получаем, что $X_n = X_{n-1} + X_{n-2}$. Отсюда $X_3 = X_2 + X_1 = 2 + 1 = 3$; $X_4 = X_3 + X_2 = 3 + 2 = 5$, получаем последовательность:

$$1,2,3,5,8,13,21,34,55,89,\ldots$$

То есть у Васи есть 89 способов спуститься. □

Числа, которые получились в этой задаче, называются *числами Фибоначчи*, с одним лишь отличием, числа Фибоначчи начинаются с чисел 1 и 1, и последовательность выглядит так:

$$F_n = F_{n-1} + F_{n-2} : F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8, \ldots$$

Нетрудно убедиться, что в общем случае существует F_{n+1} способов спуститься с лестницы, содержащей n ступенек, делая шаги по 1 и по 2 ступеньки.

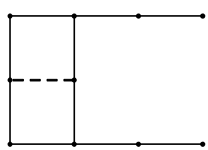
Задача II.22. Сколько существует способов разрезать прямоугольник 2 на 15 на прямоугольники размером 1 на 2?

Решение. Пусть X_n - количество способов разрезать прямоугольник $2 \times n$ на прямоугольники размером 1 на 2. Нетрудно понять, что $X_1 = 1$ и $X_2 = 2$. Попробуем вывести рекуррентное соотношение для последовательности X_n.

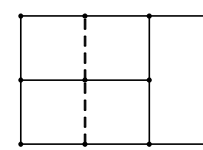
Рассмотрим левую нижнюю клетку прямоугольника. Она обязательно занята каким-то прямоугольником 1×2. Возможны 2 случая:

1. Прямоугольник 1×2 расположен вертикально (рис. 2.3а), тогда требуется разрезать прямоугольник размером $2 \times (n-1)$, это можно сделать X_{n-1} способами

2. Прямоугольник 1×2 расположен горизонтально, тогда над ним также обязан быть расположен горизонтальный прямоугольник (рис. 2.3б) и остаётся замостить доску размером $2 \times (n-2)$, что можно сделать X_{n-2} способами.



а) вертикально



б) горизонтально

Рис. 2.3: Разбиение прямоугольника $2 \times n$.

Поэтому для чисел X_n выполняется рекуррентное соотношение: $X_n = X_{n-1} + X_{n-2}$. Отсюда нетрудно понять, что $X_n = F_{n+1}$, где F_n — числа Фибоначчи. Поэтому $X_{15} = F_{16} = 987$. □

Рекуррентным соотношением, как можно заметить, называется выражение членов последовательности через несколько предыдущих, в том числе рекуррентные соотношения часто используются в информатике.

3 Теория графов

3.1 Степень вершины графа

Графы не затрагиваются в школьной программе по математике (а не все поступающие на ФУПМ сдавали информатику), но олимпиадные задачи на эту тему встречаются достаточно часто.

Так что же такое граф? Формально, *граф* G — это набор множества *вершин* V и *ребер* E, неупорядоченных пар вершин. Ребро, соединяющее вершины x и y, будем обозначать $(x,y) \in E$, в таком случае будем говорить, что x и y *смежны*. Не все вершины обязаны быть связаны ребрами.

Наглядно изобразить это можно так: пусть вершины — это некоторые города, а рёбра графа — это дороги, их соединяющие (рисунок 3.4). При этом в графе не важно расположение вершин или факт пересечения рёбер — один и тот же граф может быть изображён различными способами.

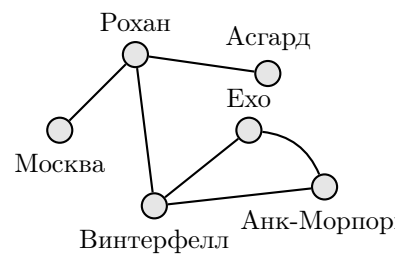


Рис. 3.4: Граф в виде городов и дорог между ними.

Одним из важных понятий графа является:

Определение. *Степень вершины графа* — это количество рёбер, которые выходят из этой вершины.

Например, на рисунке 3.4 степень вершины «Рохан» — 3, так как из неё исходят три дороги — в Москву, в Винтерфелл и в Асгард.

Определение. *Подграф* графа G — граф, множества вершин и рёбер которого есть подмножества множеств вершин и рёбер графа G соотвественно.

Сформулируем и докажем следующее важное утверждение:

Лемма. Сумма степеней всех вершин равна удвоенному количеству рёбер.

Доказательство. Вспомним, что степень вершины — это количество рёбер, которые из неё выходят. Значит, сумма степеней — это количество всех рёбер, умноженное на два, так как каждое ребро было подсчитано два раза — с каждого из двух концов. □

Задача II.23. Кандидат в мэры города Бобров в своей предвыборной компании обещал соединить все имеющиеся в городе 25 домов телефонными линиями так, что каждый дом будет соединён ровно с 5 другими. Докажите, что

он не сможет сдержать своё обещание.

Решение. Пусть дома — это вершины, а телефонные линии — это рёбра графа. Тогда степень каждой вершины равна 5. А сумма всех степеней равна $5 \cdot 25 = 125$, что должно быть равно удвоенному количеству рёбер. Но это невозможно, так как 125 — нечётное число. □

Много ошибок при решении задач на графы относится к разбору частных случаев, или так называемого «лучшего» случая.

Определение. Граф называется *полным*, если каждая вершина соединена с каждой.

Определение. Граф называется *связным*, если из любой его вершины можно добраться до любой другой, перемещаясь по рёбрам.

Любой граф, не являющийся связным, разбивается на связные части, называемые *компонентами связности* графа.

Задача II.24. В некоторой стране каждый город соединён ровно с 10 другими, при этом из любого города можно добраться в любой. Одну дорогу закрыли на ремонт. Доказать, что из любого города всё ещё можно добраться до любого другого.

Решение. Снова представим себе, что города — это вершины, а рёбра — это дороги. Тогда в задаче рассматривается связный граф, степень каждой вершины которого равна 10. Предположим, что после закрытия дороги (ребра) граф перестал быть связным. Это значит, что граф разбился на две несвязанные между собой части (см. рис. 3.5). Посчитаем сумму степеней в левом подграфе. Пусть в ней осталось k городов. Из каждого города, кроме одного, по прежнему выходит 10 дорог, значит сумма степеней вершин в этом графе равна $10k-1$. Но это нечётное число, а значит, мы получили противоречие. Таким образом, после удаления ребра граф остался связным, что и требовалось доказать. □

Заметим, что невозможно точное изображение графа — если мы действительно будем проводить по 10 рёбер из каждой вершины, рисунок будет очень громоздким и некрасивым. Поэтому при изображении графов часто рисуют не все рёбра и вершины, оставляя достаточную для понимания сути структуру.

Задача II.25. В общежитии живёт 214 студентов. Каждый час ровно 4 из них отправляются на кухню перекусить. Может ли так получиться, что в

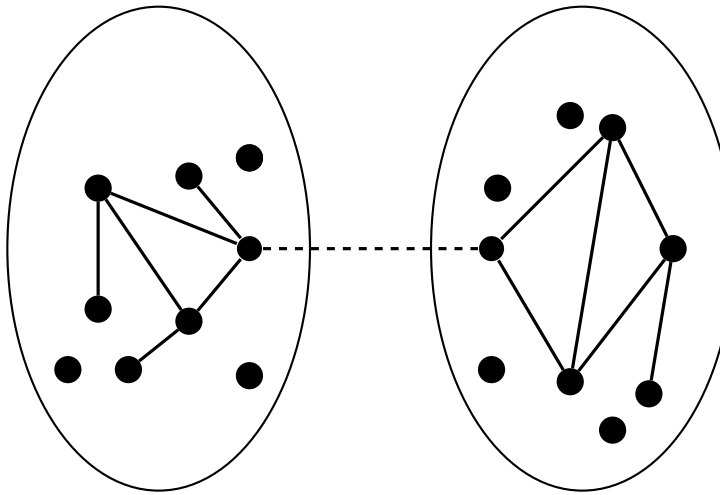


Рис. 3.5: При удалении ребра граф распадается на 2 компоненты

некоторый момент времени каждый из студентов столкнулся с каждым на кухне ровно по одному разу?

Решение. Заметим, что всего существует $\frac{214 \cdot 213}{2} = 107 \cdot 213$ пар студентов. Каждый час на кухне встречается 4 из них, это $\frac{4 \cdot 3}{2} = 6$ пар. Но так как $107 \cdot 213$ число нечётное, то оно не может делиться на 6. Поэтому не могло получиться так, что в какой-то момент времени каждый из студентов столкнулся с каждым ровно по разу.

Заметим, что задачу можно сформулировать на языке графов: есть полный граф с 214 вершинами, требуется узнать, можно ли его разбить на полные подграфы на 4 вершинах.

□

Задача II.26. В стране Ёжландия кандидат в президенты после выборов обещает сделать самолётное сообщение. При этом из столицы будет выходить 7 рейсов, из всех остальных городов — по 6, а из города Бобров — всего один рейс. Жители города Бобров считают, что теперь они даже с пересадками не смогут долететь до столицы. Докажите, что они ошибаются.

Решение. Предположим, что из города Бобров нельзя будет долететь до столицы. Это означает, что граф, вершинами которого являются города, а рёбрами — авиалинии, является несвязным. При этом столица и город Бобров

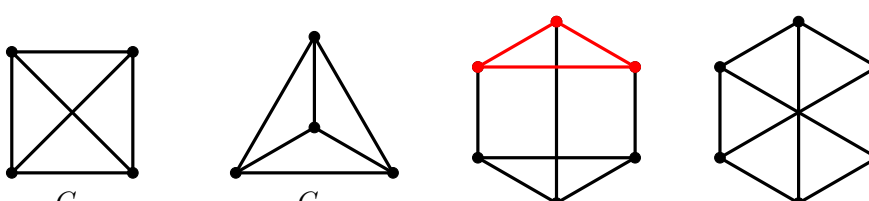
находятся в разных компонентах связанности. Рассмотрим ту компоненту, в которой находится город Бобров: степени всех вершин, кроме одной, равны 6, и степень одной вершины равна 1. Но тогда сумма всех степеней является нечётным числом, что невозможно. Поэтому жители города Бобров ошибаются, и из него точно можно долететь до столицы. □

3.2 Планарные графы

В графе не важно расположение вершин или факт пересечения рёбер — один и тот же граф может быть изображён различными способами. Дадим строгое определение:

Определение. Графы называются *равными* (*изоморфными*), если они имеют одинаковое количество вершин, и эти вершины можно занумеровать таким образом, что вершины соединены в первом графе тогда и только тогда, когда вершины с такими же номерами соединены во втором графе.

На картинке ниже графы G_1 и G_2 изоморфны: они оба являются полными графами на 4 вершинах. Покажем, изображенные на той же картинке графы G_3 и G_4 не изоморфны. В графе G_3 есть цикл длины 3, отмеченный красным цветом. Если бы G_3 и G_4 были изоморфны, то в G_4 можно было бы найти три попарно смежные вершины, то есть существовал бы цикл длины 3. В графе G_4 не может быть цикла длины 3; убедиться в этом можно самыми разными способами, хотя бы перебором[15].



Определение. *Циклом* в графе называется путь, который начинается и заканчивается в одной вершине, и при этом не проходит ни через одну из других вершин дважды.

Например на рисунке 3.6 $B - C - D - E - B$ и $F - G - H - F$ — циклы, они выделены зелёным цветом.

[15]Более высоколобый способ — проверить двудольность графа G_4, а в двудольных графах циклы могут иметь лишь четную длину.

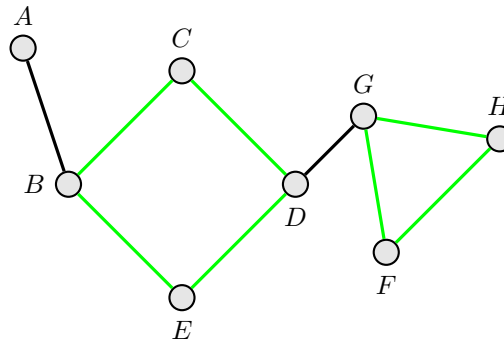


Рис. 3.6: Пример графа.

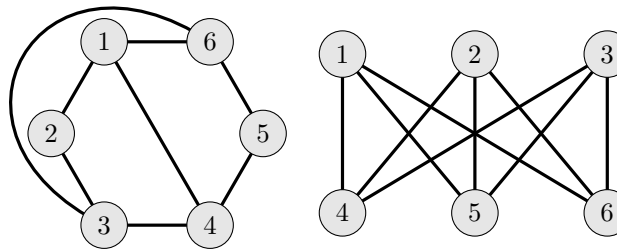
Определение. *Дерево* — это связный граф без простых циклов. У дерева количество рёбер на одно меньше количества вершин.

Определение. Граф называется *планарным*, если его можно так изобразить на плоскости, чтобы у него не пересекались рёбра (рёбра графа не обязательно должны быть прямыми отрезками!).

Гранями планарного графа называются циклы, которые не могут быть разбиты на более мелкие циклы. Для планарных графов выполняется формула Эйлера: Г + В − Р = 2, где Г— число граней, В — число вершин и Р — число рёбер.

Задача II.27. Есть три дома и три магазина — продуктовый, аптека и хозяйственный. Можно ли от каждого дома проложить тропинку до каждого из магазинов так, чтобы тропинки не пересекались?

Решение. Без труда строится пример, в котором проведены все тропинки, кроме одной, а вот последнюю провести не получается. (рис. 3.2а)



«почти» пример все тропинки проведены

Пусть дома и магазины — это вершины графа, а тропинки — это рёбра. Пусть вершины под номерами 1, 2 и 3 — это дома, а 4, 5 и 6 — магазины (рис. 3.2б). Количество вершин — 6, количество рёбер — 9. Для того, чтобы граф был планарным, необходимо, чтобы у него было количество граней, равное $\Г = \Р + 1 - \В = 9 + 1 - 6 = 4$. Но можно насчитать по крайней мере 5 циклов: $1-4-2-5-1$; $1-4-3-6-1$; $2-5-3-6-2$; $1-6-2-4-1$; $2-6-3-4-2$. Значит, граф не является планарным, следовательно, его нельзя изобразить на плоскости так, чтобы его рёбра не пересекались.

□

Задача II.28. Дана волейбольная сетка размером 12 на 25 квадратиков. Какое наибольшее число верёвочек можно разрезать так, чтобы сетка не распалась?

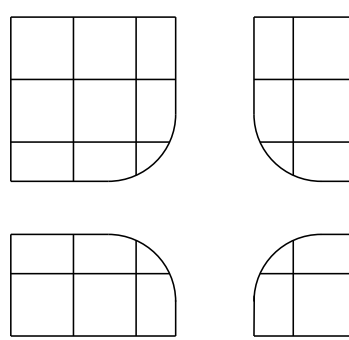


Рис. 3.7: Волейбольная сетка.

Решение. Представим сетку в виде графа, где узелки — это вершины, а верёвочки — рёбра. Нам требуется удалить как можно больше рёбер так, чтобы

граф при этом оставался связным. Будем поступать следующим образом: пока в графе есть цикл, можно убрать одно ребро из этого цикла. При этом граф останется связным. Будем «резать» циклы до тех пор, пока не останется ни одного. Связный граф без циклов — это дерево. В оставшемся графе будет $13 \cdot 26 = 338$ вершин, значит, в нём будет $338 - 1 = 337$ рёбер. А вначале в графе было $13 \cdot 25 + 12 \cdot 26 = 637$ рёбер. Поэтому можно разрезать максимум $637 - 337 = 300$ верёвочек. Больше ни одной верёвочки разрезать не получится — дерево при удалении любого из рёбер перестаёт быть связным.

Заметим, что в условии задачи присутствует слово «наибольшее» — это значит, что она на оценку и пример. В данном случае и оценка и пример идут бок о бок, из построения примера следует доказательство оценки.

□

Задача II.29. Можно ли между каждыми двумя из 5 городов провести дорогу так, чтобы эти дороги не пересекались?

Решение. Рассмотрим граф на 5 вершинах, вершинами которого являются города, а рёбрами — дороги между ними. Тогда в условии задачи спрашивается, является ли такой граф планарным. Докажем, что не является. Воспользуемся ддя доказательства формулой Эйлера. Подсчитаем количество вершин, рёбер и граней. Вершин — 5, рёбер $C_5^2 = 10$, так как каждое ребро соединяет две вершины. Грани — это простые циклы. Так как все вершины соединены между собой, то любые три вершины образуют простой цикл, поэтому есть хотя бы 10 граней. Но по формуле Эйлера $\Г = \Р + 1 - \В = 10 + 1 - 5 = 6 < 10$. Полученное противоречие доказывает, что граф планарным не является, значит, невозможно искомым образом провести дороги.

□

Задача II.30. Докажите, что в любом планарном графе есть вершина, степень которой не превосходит 5.

Решение. Воспользуемся методом доказательства «от противного»: пусть степень каждой вершины превосходит 5. Так как степень каждой вершины — целое число, то степени всех вершин $\geqslant 6$. Поэтому всего из всех вершин выходит хотя бы $6B$ ребёр, каждое из которых было посчитано ровно по 2 раза, отсюда

$$\Р \geqslant \frac{6\В}{2} = 3\В \Leftrightarrow \В \leqslant \frac{\Р}{3}.$$

Найдём зависимость между количеством граней и количеством рёбер. Каждая грань состоит как минимум из трёх рёбер. Значит, все грани состоят из хотя

бы 3Г рёбер, каждое из которых было посчитано не более двух раз. Поэтому

$$\text{Р} \geqslant \frac{3\text{Г}}{2} \Leftrightarrow \text{Г} \leqslant \frac{2\text{Р}}{3}.$$

Сложим получившееся неравенства для граней и вершин: $\text{В} \leqslant \frac{\text{Р}}{3}$ и $\text{Г} \leqslant \frac{2\text{Р}}{3}$, откуда: $\text{Г} + \text{В} \leqslant \text{Р}$. Полученное неравенство противоречит формуле Эйлера. Полученное противоречие завершает доказательство данной задачи. □

3.3 Ориентированные графы

Граф называется *ориентированным*, если его рёбра имеют направление. Изображается это обычно стрелкой на конце в какую-либо сторону (рис. 3.8). Если это не оговорено, в ориентированном графе обычно не могут существовать два ребра между двумя вершинами, одно из которых направлено в одну сторону, а другое — в другую. Неориентированный граф разбивается на компоненты связности. Этот термин обычно не применяют к ориентированным графам, там имеется другая терминология.

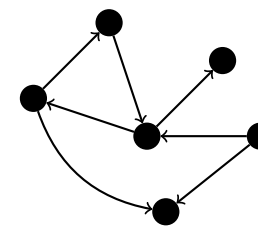


Рис. 3.8: Ориентированный граф.

В задачах на ориентированные графы обычно упоминаются всякие реки, потоки или дороги с односторонним движением. Иногда — односторонние отношения между людьми ;-)

Определение. Вершины u и v графа называются *сильно связанными*, если существует путь по рёбрам от u до v и от v до u (передвигаться можно только по направлению стрелок!).

Определение. Граф называется *сильно связным*, если любые две его вершины сильно связаны.

Любой граф разбивается на компоненты сильной связанности, между которыми проведены ориентированные рёбра (рис. 3.9).

Задача II.31. В некоторой стране города соединены односторонними авиалиниями, причём между некоторыми городами нет рейса ни в одну, ни в другую сторону. Известно, что выполняется условие: вылетев из любого города, нельзя в него вернуться, пользуясь этими авиалиниями. Докажите, что можно дополнить систему авиалиний так, чтобы любые два города были соединены авиалинией и при этом условие невозможности возвращения в город сохранялось бы.

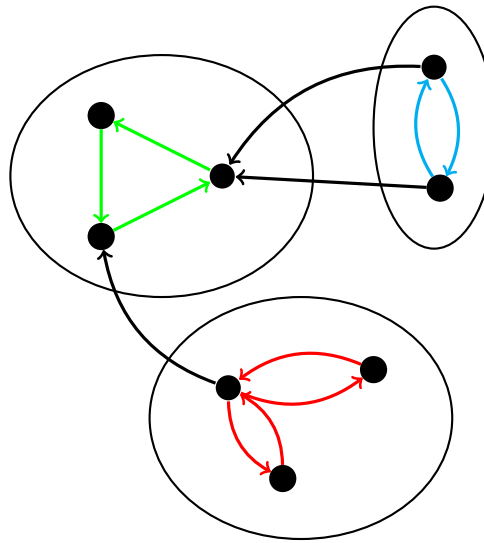


Рис. 3.9: Компоненты сильной связности.

Решение. Переформулируем задачу на язык графов: дан ориентированный граф, в котором нет циклов, так как выполняется условие: вылетев из города, нельзя вернуться обратно. Рассмотрим два произвольных города A и B, между которыми нет ребра, и докажем, что их можно соединить либо направленным ребром AB, либо направленным ребром BA. Предположим, что это не так. Если нельзя провести направленное ребро AB, значит при его проведении появляется цикл: существует путь S_1 от B до A. Аналогично, если нельзя провести направленное ребро BA, то существует путь S_2 от A до B. Но тогда, вылетев из вершины A и двигаясь сначала по пути S_2, а потом по пути S_1, мы снова вернёмся в вершину A — противоречие. Поэтому можно провести какое-то направленное ребро между вершинами A и B. Продолжим подобную процедуру, пока между любыми двумя вершинами не будет проведено какое-то направленное ребро, что и требовалось достигнуть. □

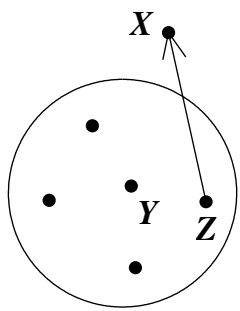
Задача II.32. В некоторой стране каждый город соединён с каждым дорогой с односторонним движением. Докажите, что найдётся город, из которого можно добраться в любой другой.

Решение. Воспользуемся методом математической индукции. База очевидна. Пусть утверждение выполняется для n городов, докажем его для $n+1$ города. Выберем n городов из этих $n+1$ города. Любые два из них соединены односторонней дорогой, поэтому к ним можно применить предположение ин-

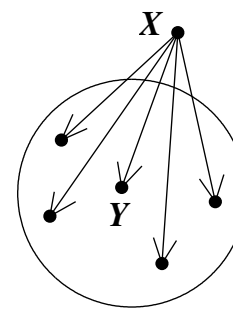
дукции: существует город Y, из которого можно добраться до любого из этих n городов. Обозначим невыбранный город за X. Возможны 2 случая:

1. Какая-то дорога ведёт в X. Тогда искомый город — Y, из него можно добраться до любого города, и в том числе до X (рис. 3.10а).

2. В X не ведёт ни одна дорога, тогда наоборот, все дороги выходят из X, он и будет искомым городом — из него можно добраться до любого (рис. 3.10б).



а)1 случай.



б)2 случай.

Рис. 3.10: К задаче 2.

□

Граф, описанный в данной задаче, имеет своё название.

Определение. Направленный полный граф называется *турниром.*

Задача II.33. По кругу записаны 7 натуральных чисел. Известно, что в каждой паре соседних чисел одно делится на другое. Докажите, что найдётся хотя бы одна пара чисел, не стоящих рядом, обладающая этим же свойством.

Решение. Представим числа как вершины графа, а делимости как ориентированные рёбра: если первое число делится на второе, проведём от первого ко второму стрелку. Тогда между любыми двумя соседними числами обязательно проведена стрелка, в какую-либо из сторон.

Докажем, что существует две стрелки, направленные в одну сторону (по часовой стрелке или против неё). Предположим противное: пусть некоторая стрелка направлена по часовой стрелке, тогда следующая - против, и так далее. Но тогда седьмая стрелка будет направлена по часовой стрелке, и нашлись две

соседние стрелки, направленные в одну сторону. Поэтому некоторое число a делится на соседнее b, а b делится на соседнее c. Значит, $a \vdots c$ — существует два несоседних числа, одно из которых делится на другое, что и требовалось доказать.

□

Задача II.34. Дано 2014 точек. Докажите, что можно соединить их стрелками так, чтобы из каждой точки в каждую можно было попасть, пройдя либо по одной стрелке, либо по двум.

Решение. Докажем по индукции, что $4n + 2$,$n \geqslant 1$ точек можно соединить искомым образом.

База $n = 1$ изображена на рисунке 3.11.

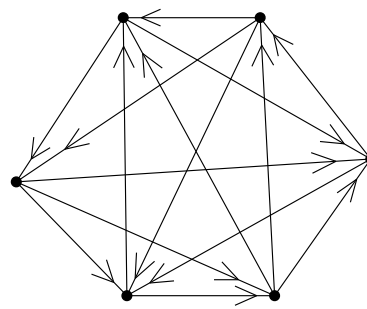


Рис. 3.11: База: $n = 1$.

Переход: пусть $4n+2$ точек соединены искомым образом. Добавим ещё 4 точки, и докажем, что можно так добавить стрелки, чтобы условие по-прежнему сохранялось.

Разобьём $4n + 2$ точек на пары: A_1,B_1; A_2,B_2; $\dots$ A_{2n+1},B_{2n+1}, пусть стрелки идут от A_k к B_k для $k = 1, \dots, 2n + 1$. Добавим 4 новые точки: X,Y,Z,T. Направим стрелки от X,Y,Z,T до A_k и от B_k, $k = 1, \dots, 2n$ до X,Y,Z,T. Между точками A_{2n+1},B_{2n+1},X,Y,
Z,T построим базу так, чтобы ребро шло от A_{2n+1} к B_{2n+1}. Докажем, что условие задачи не перестало выполняться. По предположению индукции между точками
A_1,$B_1, \dots, A_{2n+1}$,B_{2n+1} можно добраться, используя одну или две стрелки. От X,Y,Z,T можно добраться до любой A_k, $k = 1, \dots, 2n$ по одной стрелке; от X,Y,Z,T можно добраться до любой B_k, $k = 1, \dots, 2n$ по двум стрелкам: $X \to A_k \to B_k$. Аналогично, можно добраться и в обратную сторону. Между точками A_{2n+1},B_{2n+1},X,Y,Z,T можно добраться, используя одну или две

стрелки, в силу построения.

Осталось заметить, что $2014 = 503 \cdot 4 + 2$, поэтому между 2014 точками можно провести стрелки искомым образом, что и требовалось доказать. □

Ориентированные графы часто также называют «орграфы».

Мы ввели понятие сильной связности, а, значит, бывает ещё и слабая? Да, бывает.

Определение. Граф называется *слабо связным*, если каждая вершина может быть достижима из каждой, возможно, с нарушением (против стрелок).

Существует ещё один вид связности:

Определение. Граф называется *односторонним*, если для любой пары вершин одна из них достижима из другой.

Остальные орграфы называются несвязными.

Определение. *Степенью исхода* $od(v)$ вершины v называется количество выходящих из неё рёбер. *Степенью захода* $id(v)$ вершины v называется количество входящих в неё рёбер.

Лемма 1 (Ориентированная лемма о рукопожатиях)**.** В любом орграфе сумма степеней захода всех вершин равна сумме степеней исхода всех вершин.

Доказательство. При подсчёте степеней каждая дуга (u,v) будет учтена дважды: для степеней захода — как дуга, входящая в v, а для степеней исхода — как дуга, исходящая из u. □

Для самостоятельного решения оставляем вам следующую задачу:

Задача II.35. (Самостоятельно) В некоторый момент однокругового турнира из 100 человек оказалось, что все игроки, кроме Лошикова, выиграли по 26 игр, а проиграли по 25. Докажите, что Лошиков совсем не умеет играть.

Аналогично обыкновенным графам, могут быть введены и такие определения, как:

Определение. *Гамильтонова орцепь* орграфа — простая цепь, проходящая через каждую вершину орграфа ровно один раз.

Определение. *Гамильтоновым орциклом* называется орцикл орграфа, который проходит через каждую его вершину.

Определение. Орграф называется *полугамильтоновым*, если он имеет гамильтонову орцепь, и гамильтоновым, если обладает гамильтоновым орциклом.

Очевидно, что всякий гамильтонов орграф сильно связен. Легко понять, что это необходимое условие гамильтоновости орграфа не является достаточным. Мы укажем (без доказательства) два достаточных условия гамильтоновости орграфов.

Определение. *Петля* — ребро графа, исходящее из вершины и входящее в неё же. Обычно мы рассматриваем графы, не содержащие петли.

В информатике и алгоритмике существует ещё задача о гамильтоновом цикле. Её суть — выяснить, имеет ли заданный граф G гамильтонов цикл. Данная задача является NP-полной[16].

Определение. Маршруты, проходящие по всем вершинам, называются *остовными*.

Определение. *Эйлеровым* (*полуэйлеровым*) называется орграф, в котором существует замкнутый (незамкнутый) маршрут, проходящий по каждому ребру ровно один раз.

Задача II.36 (самостоятельно)**.** Докажите, что что связный неодновершинный орграф эйлеров тогда и только тогда, когда у каждой вершины полустепень исхода равна полустепени захода.

Задача II.37. В королевстве каждый город соединен с каждым дорогой. Может ли сумасшедший король ввести на дорогах одностороннее движение так, чтобы выехав из любого города, в него нельзя было вернуться?

Решение. Тут мы имеем дело с другим видом задач: введением ориентации (с заданными свойствами) на обычном графе. Занумеруем все вершины и будем ставить все стрелки от меньшего номера к большему. Можно заметить, что в построенном графе каждая вершина - отдельная компонента сильной связности. □

Обратная задача: дан ориентированный граф, в котором, выехав из любой вершины, в нее нельзя вернуться (то есть, нет ориентированных циклов); надо занумеровать его вершины так, чтобы все ребра шли от меньшего номера к большему — называется топологической сортировкой графа. Существует довольно красивый и не очень сложный алгоритм топологической сортировки.

[16] для объяснения этого термина понадобилась бы не одна страница, поэтому оставим его как он есть. Пока запомним, что NP-полные задачи неразрешимы за ограниченное время для достаточно больших входных данных

Задача II.38. В королевстве из каждого города выходит четное число дорог и из каждого города можно доехать до любого другого. Докажите, что мудрый король сможет ввести одностороннее движение так, чтобы сохранилось это свойство, а, сверх того, из каждого города выходило столько же дорог, сколько входило.

Решение. Если в предыдущей задаче надо было ввести ориентацию, делая как можно больше компонент сильной связности, то теперь надо сделать их как можно меньше. Какой же такой обход графа придумать? Давайте воспользуемся четностью степеней всех вершин. По критерию Эйлера, в графе (раз он связный!) существует эйлеров цикл. А давайте просто пойдем по эйлерову циклу и расставим стрелки в направлении нашего движения. Получим большой замкнутый ориентированный путь, содержащий все ребра. А все вершины он и подавно содержит, ч.т.д. □

Задача II.39. Докажите, что в полном ориентированном графе с тремя и более вершинами можно поменять направление одного ребра так, чтобы он стал сильно связным (если он исходно не сильно связный).

Решение. Воспользуемся индукцией по числу вершин. База — 3 вершины. Если граф не сильно связный, то он изоморфен примеру с 3-мя вершинами к предыдущей задаче. А там можно развернуть одно ребро так, чтобы получить ориентированный цикл длины 3.

Переход: опять же, давайте удалим вершину, сделаем оставшийся граф сильно связным (по предположению индукции) и вернем вершину на место. Чтобы после возвращения граф сохранил сильную связность, удаленная вершина должна иметь как входящие, так и выходящие ребра. А если такой вершины нет? Тогда рассмотрим вершину A, из которой все ребра выходят (очевидно, что откуда-то ребра должны выходить), и любые другие вершины B и C. В B и C ведут ребра из A, поэтому в них все ребра входят. Но как же быть с ребром BC? Оно должно выходить из одной из этих вершин?! Значит, такое невозможно, что и требовалось доказать. □

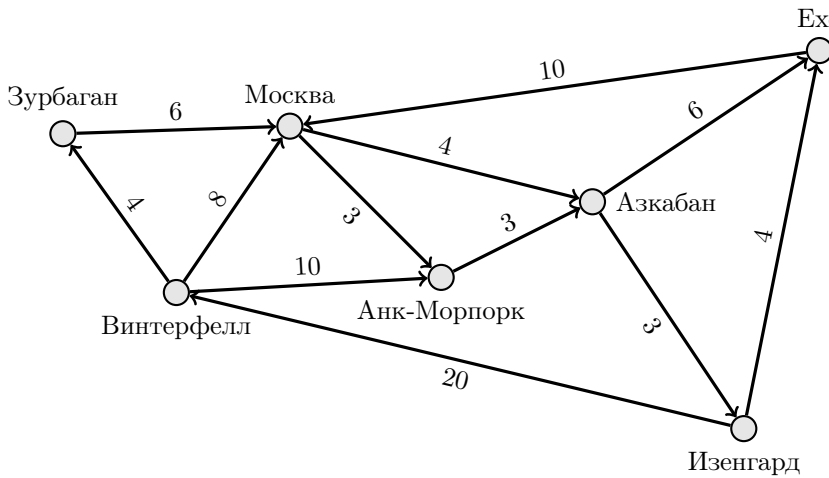
3.4 Взвешенные графы

В дискретном анализе и информатике популярен ещё один вид графов — *взвешенные.*

Например, рассмотрим простой пример — граф дорожной сети. Дороги бывают разной длины, значит, можно сказать, что граф является взвешенным —

каждому ребру можно поставить в соответствие число, например, являющееся длиной соответствующей дороги.

Если все дороги находятся в одинаковом состоянии и абсолютно пусты, то данная картина уже может многое сказать о том, сколько времени, например, мы будем добираться от одного города до другого и через какие города нам лучше всего ехать.



В данном графе, например, мы можем убедиться, что из Винтерфелла в Exo быстрее всего добираться через Москву и Азкабан.

Каждый человек, собираясь в путь, смотрит на дорожную ситуацию. Большинство людей, выбирая траекторию движения, полагают, что в течение всего пути транспортная ситуация меняться не будет, то есть, выбирает кратчайшую дорогу на ориентированном графе с постоянными весами рёбер.

Выбрать кратчайший путь на простых графах — очень легко. А что делать, если мы ищем кратчайший путь на большом графе?

Расскажем о самом известном из неперебoрных алгоритмов — *алгоритме Дейкстры*. Это один из фундаментальных алгоритмов и его вы точно будете проходить на Физтехе, причём, скорее всего, не на одном предмете и не один раз.

В алгоритме Дейкстры ищутся кратчайшие пути из заданной вершины во все остальные. Можно считать, что алгоритм Дейкстры строит направленный граф кратчайших путей без циклов. Графы без циклов называются деревьями, так что алгоритм Дейкстры строит Дерево Кратчайших Путей (ДКП), на английском языке — Shortest Path Tree, SPT. Этот алгоритм нужен не только в наших задачах о поездках, он применяется, например, ещё и в ком-

пьютерных сетях. Есть много его вариантов, один из которых можно было прочитать в первом издании данной книги, но в этом мы его опускаем по причине не такой сильной его необходимости в первом семестре. Описание этого и некоторых других алгоритмов на графах приведено, например, в лекциях по теории игр и исследованию операций [5].

4 Что читать дальше

Несмотря на то, что данная глава была в первую очередь посвящена основам дискретной математики — предмету, который выделен, например, на фупме в трёхсеместровый курс, материалы данной главы активно используются и в других предметах, проходимых в МФТИ, в первую очередь — в информатике.

Дискретный анализ — трёхсеместровая сага кафедры математических основ управления (моу) об основах дискретной математики, состоящая из трех предметов, по каждому из которых есть учебник. Непосредственно комбинаторика и теория графов — предмет первого семестра [17], приложения алгебры в теории чисел проходятся на курсе оваитк во втором семестре [16], третий же семестр дискрана — упоминавшийся ранее курс «Теория формальных систем» [18]. Курс «Алгоритмы и модели вычисления» четвертого семестра рассматривается кафедрой моу как естественная кульминация фупмовского дискрана, где почти все полученные до этого знания находят свое применение. Впрочем, амв оказался слишком перегруженным для одного семестра, поэтому он разбит на два курса — собственно амв и курс «Основные алгоритмы» второго семестра. Разумеется, на курсе алктг некоторые вещи не получается рассмотреть подробно (например, графы и реккуренты), поэтому на алгоритмических курсах эти темы напомнят. По алгоритмическим предметам кафедра моу пока не издала учебников, но возможно, закроет эту лакуну в будущем.

Коллектив из пяти авторов написал учебник [30] по дискретной математике для ФКН ВШЭ, найти его можно в электронном виде на домашней странице Рубцова. Поскольку среди соавторов был Шень, то ожидайте простых и понятных объяснений. Учебник хороший, содержит много простых упражнений и не требует никаких пререквизитов[17], так что можете спокойно заниматься сразу по нему.

Еще один вводный учебник по комбинаторике, который может быть полезен студентам фупма — «Конкретная математика» [15] Кнута, Грэма и Паташника. Дискретная математика рассматривается здесь как основание информатики, поэтому все необходимые темы сюда включены. На полях написаны цитаты великих умов, исторические заметки и шутки уровня группы мхк («Вы

[17] то есть все, что нужно для понимания этой книжки, содержится в самой книжке.

называете это правило мнемоническим? Я бы назвал его пневматическим – одно сотрясение воздуха»), некоторые из которых непереводимы на русский[18], а к каждой главе прилагается набор задач (как правило простых, иногда не очень простых). Книга в целом совершенно тривиальна, темы, не относящиеся напрямую к информатике, например, гипергеометрические функции, изложены достаточно скудно — самое подходящее чтиво для физтехов.

Чуть ли не единственная тема в комбинаторике, с которой можно успеть разобраться на курсе алктг — производящие функции. «Лекции о производящих функциях» [28] Ландо развивают эту тему. Его же учебник [29] по дискретной математике — надстройка над этими лекциями, в которую включены также главы про теорию графов и про формальные языки. Будучи также топологом, Ландо включил в свой курс такие темы, как инварианты графов и формулу Римана-Гурвица, так что заинтересованным граждам эта книжка может прийтись по вкусу.

К сожалению, серьезного курса теории графов на первых двух курсах фупма нет. Официальная программа кафедры моу ссылается на классический учебник Харари [36] — хороший, но немного устаревший. Более полным и подробным является учебник Бонди и Мурти [1], изданный в 80-е и переизданный в 2008 году. Любители также могут изучить раздел про теорию графов книги Прасолова «Элементы комбинаторной и дифференциальной топологии» [34] и узнать, в частности, доказательство теорем Фари, Куратовского и Менгера. Лучшим же на данном этапе обучения нам представляется изданный этой зимой учебник Омельченко [31] — там совершенно элементарные объяснения, большой и подробный список задач, к самым простым из которых там написаны решения.

С этого можно начать знакомство с миром дискретной математики.

[18] Kilroy wasn't Hoare — отсылка к американской поговорке Kilroy was here времен второй мировой, перевести эту фразу без потери отсылки не получится.

Глава III

Алгебра, матанализ, аналит и линал

1 Системы линейных уравнений

Умение решать системы линейных уравнений (СЛУ) — один из самых базовых навыков, необходимый при решении задач математического анализа, линейной алгебры и аналитической геометрии, поэтому вам надо будет научиться делать это весьма быстро и чётко.

Методы подстановки, которыми большинство решают системы уравнений, зачастую слишком длинны и громоздки. Не у всех это получается легко (хотя вроде бы это было в школе). Почему?

Причина в том, что методы, которыми СЛУ решались в школе, отличаются от того, что требуется от нас теперь. Давайте перечислим способы решения, которые можно найти в школьном учебнике или у Д. В. Беклемишева:

- метод подстановки;
- почленное сложение/вычитание уравнений системы;
- метод Крамера;
- метод Гаусса;
- с помощью обратной матрицы.

Как решать системы уравнений методом подстановки, знают все — выразить одну переменную через остальные из какого-либо уравнения, подставить во все остальные; количество переменных и уравнений уменьшилось на 1; повторить. Когда уравнений мало, этот метод эффективен. Однако обычно возникают дробные коэффициенты, что приводит к ошибкам. Поэтому даже для систем из трёх уравнений этот метод лучше не использовать (обычно даже для двух переменных он тоже не очень хорош, поэтому лучше оставить его школьникам — им-то как раз и приходится мучиться).

Метод почленного сложения/вычитания в школьном варианте:

$$\begin{cases} 2x + y = 3; \\ x + 3y = 4 \end{cases}$$

Умножим второе уравнение на 2 и вычтем из него первое, исключив x:

$$\begin{cases} 2x + y = 3; \\ 5y = 5 \end{cases}$$

Отсюда находим $y = 1$ и подставляем в первое уравнение, после чего из $2x = 2$ получаем $x = 1$.

Для трёх уравнений метод работает аналогично:

$$\begin{cases} 2x + y - 2z = 1; \\ x + 3y + z = 5; \\ 3x - z = 2 \end{cases}$$

Для разнообразия вычтем из второго уравнения утроенное первое:

$$\begin{cases} 2x + y - 2z = 1; \\ -5x + 7z = 2; \\ 3x - z = 2 \end{cases}$$

А теперь прибавим к умноженному на 5 третьему уравнению утроенное второе:

$$\begin{cases} 2x + y - 2z = 1; \\ -5x + 7z = 2; \\ 16z = 16 \end{cases}$$

Из третьего уравнения находим $z = 1$, подставляем в первое и второе:

$$\begin{cases} 2x + y = 3; \\ -5x = -5 \end{cases}$$

Из второго уравнения получаем $x = 1$, из первого — $y = 1$.

Давайте займёмся более содержательными вещами. Они требуют умения считать определители. А их в школьной программе не было, как не было и матриц... Поэтому некоторые определения придётся сформулировать.

Матрица размера $m \times n$ — совокупность mn чисел, расположенных в виде m строк и n столбцов

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Числа a_{ij}, составляющие матрицу, называются *элементами матрицы*. Если число строк матрицы равно числу её столбцов, то матрица называется *квадратной*, а число её строк — *порядком матрицы*.

Нулевая матрица размера $m \times n$ — это матрица размера $m \times n$, все элементы которой равны нулю.

Единичная матрица порядка n — это матрица размера $n \times n$, у которой все элементы, расположенные на главной диагонали (т. е. a_{ij} при $i = j$) равны 1, а все остальные — 0. Она обозначается E[19].

Что можно делать с матрицами? Например, матрицы одинакового размера можно складывать (покомпонентно, так же, как векторы[20]).

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix}$$

где $c_{ij} = a_{ij} + b_{ij}$.

Матрицы можно *транспонировать* (т. е. симметрично отражать относительно главной диагонали):

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}^T = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{pmatrix}$$

Иногда матрицы можно умножать. Точнее, матрицы размеров $m \times n$ и $p \times q$ можно перемножать (строго в таком порядке!) тогда и только тогда, когда $n = p$.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nq} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1q} \\ c_{21} & c_{22} & \dots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mq} \end{pmatrix}$$

где $c_{ij} = \sum\limits_{k=1}^{n} a_{ik} b_{kj}$. Как это запомнить? Элемент ij результата — это произведение i строки первого множителя на j столбец второго; произведение же строки на столбец ассоциируется со скалярным произведением соответствующих им векторов (помнится, в школе не мучились и записывали векторы обоими способами). Не стоит путать это с произведением столбца на строку:

[19] Иногда также E_n, если подчеркивается, что это матрица порядка n.

[20] Вообще-то n-мерные вектора можно рассматривать как матрицы вида $n \times 1$.

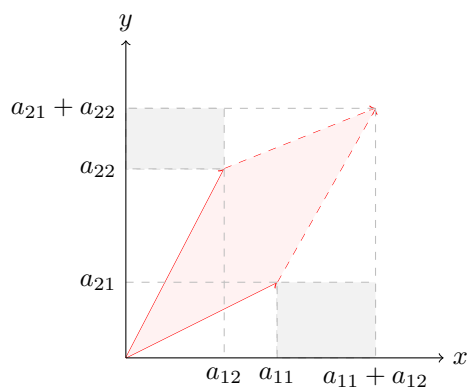
в качестве упражнения попробуйте понять, что будет получено в подобном задании, это вам потребуется при решении задания по аналитической геометрии.

Так же у квадратных матриц можно вычислять *определитель* (он же *детерминант*; обозначается прямыми чертами, как модуль, или $\det A$). Он существует только для квадратных матриц. Введём детерминанты квадратных матриц порядка 2, общего определения[21] пока давать не будем.

Для 2×2-матриц детерминант определяется как

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

Заметим, что в данном случае $\det A$ есть ориентированная площадь параллелограмма, натянутого на вектора (a_{11}, a_{21}) и (a_{12}, a_{22}):



$$S = (a_{11} + a_{21})(a_{12} + a_{22}) - a_{21}a_{12} - a_{21}a_{12} - 2 \cdot (\frac{1}{2}a_{11}a_{12} + \frac{1}{2}a_{21}a_{22}) = a_{11}a_{22} - a_{21}a_{12}$$

Определитель третьего порядка (то есть матрицы 3×3) можно считать поразному. Основной способ — разложение по столбцу или строке (т. к. $\det A^T = \det A$, эти способы эквивалентны):

[21] Самое общее определение очень громоздкое на первый взгляд и требует знания алгебры Грассмана, но оно легко обобщается и инкорпорирует в себя все свойства детерминанта, а теоремы вроде Коши-Бине доказываются в одну строчку.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} (*) \tag{1}$$

Раскладывать можно не только по первой строке, но и по любой другой, главное — следить за знаками. Плюс перед слагаемым будет в том случае, если сумма номеров строки и столбца, на пересечении которых стоит множитель-элемент — чётное число; иначе — минус.

Очевидно, что с помощью разложения по строке или столбцу можно также дать индуктивное определение детерминантов высших порядков.

Расписав далее правую часть 1 и сгруппировав слагаемые, получим

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - \\ -a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}$$

Оказывается, что если квадратная матрица A — *невырожденная*, то есть такая, что $\det A \neq 0$), то существует и единственна *обратная к* A *матрица* A^{-1}, то есть такая матрица B, что $AB = BA = E$.

Элементы обратной матрицы к A можно вычислять по формуле

$$x_{ij} = (-1)^{i+j} \frac{d_{ji}}{\det A},$$

где d_{ji} — дополнительный *минор элемента* a_{ji}, то есть определитель матрицы, полученной из A вычёркиванием j строки и i столбца (на пересечении которых расположен a_{ji}). Для матриц второго порядка это даёт формулу

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Научившись работать с матрицами, применим их к решению систем линейных уравнений. Будем решать систему 3 порядка (для 2 порядка аналогично; заметим также, что методы работают для систем любого порядка)

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1; \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2; \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

Эту систему можно записать как $Ax = b$, где

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}; \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}; \quad b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

Теперь трудно не заметить, что если $\det A \neq 0$, то решение системы легко находится как $x = A^{-1}b$. Это и есть решение СЛУ **с помощью обратной матрицы**.

Данный метод неприменим, если A — вырожденная. Но об универсальном методе Гаусса поговорим позже, а пока продолжим решать «хорошие» системы. Ведь не всегда обратная матрица ищется легко. Может быть, проще посчитать 4 определителя... Так и работает **метод Крамера**.

Введём обозначения:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}; \quad \Delta_1 = \begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix};$$

$$\Delta_2 = \begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix}; \quad \Delta_3 = \begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix}.$$

Оказывается (точнее, будет доказано ближе к концу семестра; любопытные уже сейчас могут открыть бессмертную книгу в зелёной обложке — и нет, не «Руководство по производству полётов S7», а учебник Д. В. Беклемишева), что $x_i = \dfrac{\Delta_i}{\Delta}$ при $i = 1,2,3$ — решение системы.

Вот такая красота. Для второго порядка легко записывается «по образу и подобию». И всё равно не работает для вырожденных систем, поэтому изучим **метод Гаусса**.

Будем записывать систему с помощью так называемой расширенной матрицы — это сделает изложение решение задач более кратким. Система уравнений

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases}$$

имеет расширенную матрицу

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} & b_n \end{array}\right)$$

С расширенной матрицей (и, соответственно, системой линейных уравнений) можно производить следующие операции, называемые *элементарными*:

- перестановка строк;
- умножение строки на ненулевое число;
- прибавление одной строки к другой.

Очевидно, что элементарные операции не меняют множество решений системы.

Метод Гаусса: с помощью элементарных операций привести систему к верхнему треугольному виду (т. е. все элементы, расположенные ниже главной диагонали, должны стать равными нулю). Если в каких-то строках номер самого левого ненулевого элемента совпадает, продолжаем действовать так, чтобы от таких ситуаций избавиться; легко показать, что это возможно всегда. После этого возможны следующие случаи:

- появилась строка вида $(0\ 0\ \ldots\ 0 \mid c)$, $c \neq 0$. В этом случае система несовместна, т. е. не имеет решений;
- таких строк нет, но есть строка $(0\ 0\ \ldots\ 0 \mid 0)$. Это значит, что в исходной системе были линейно зависимые уравнения; если нулевую строку удалить, множество решений системы не изменится;
- в любой строке нет нулей левее разделителя. Тогда, начиная с самой нижней строки, можно последовательно, начиная снизу, находить значения переменных (если не повезло, и в очередной строке более одной ещё не вычисленной переменной, то выбираем какую-то одну и выражаем её через остальные).

Давайте рассмотрим пример применения метода Гаусса. Увидим, что это просто научное название известного способа сложения/вычитания уравнений.

$$\begin{cases} 5x_1 + x_2 - 2x_3 = 1; \\ -x_1 + 2x_2 - x_3 = 0; \\ 2x_1 - x_2 + x_3 = 3 \end{cases}$$

Расширенная матрица системы:

$$\left(\begin{array}{ccc|c} 5 & 1 & -2 & b_1 \\ -1 & 2 & -1 & 0 \\ 2 & -1 & 1 & 3 \end{array}\right)$$

Нам нужно привести к верхнему треугольному виду. Поэтому умножим вторую строку на 2 и прибавим её к третьей:

$$\left(\begin{array}{ccc|c} 5 & 1 & -2 & 1 \\ -2 & 4 & -2 & 0 \\ 0 & 3 & -1 & 3 \end{array}\right)$$

Умножим вторую строку на $\frac{5}{2}$

$$\left(\begin{array}{ccc|c} 5 & 1 & -2 & 1 \\ -5 & 10 & -5 & 0 \\ 0 & 3 & -1 & 3 \end{array}\right)$$

Прибавим ко второй строке первую:

$$\left(\begin{array}{ccc|c} 5 & 1 & -2 & 1 \\ 0 & 11 & -7 & 1 \\ 0 & 3 & -1 & 3 \end{array}\right)$$

Дальше труднее — числа плохие. Однако, умножим вторую строку на 3, а третью на -11.

$$\left(\begin{array}{ccc|c} 5 & 1 & -2 & 1 \\ 0 & 33 & -21 & 3 \\ 0 & -33 & 11 & -33 \end{array}\right)$$

Прибавим вторую строку к третьей:

$$\left(\begin{array}{ccc|c} 5 & 1 & -2 & 1 \\ 0 & 33 & -21 & 3 \\ 0 & 0 & -10 & -30 \end{array}\right)$$

Из третьей строки найдём $x_3 = \dfrac{-30}{-10} = 3$.

Из второй: $x_2 = \dfrac{3 + 21x_3}{33} = \dfrac{1 + 7x_3}{11} = \dfrac{1 + 7 \cdot 3}{11} = 2$.

Первое уравнение после подстановки преобразуется к виду $5x_1 + 2 - 2 \cdot 3 = 1$, или $5x_1 = 5$, откуда $x_1 = 1$.

Ответ: $(x_1, x_2, x_3) = (1, 2, 3)$.

Итак, мы разобрали способы решения систем линейных уравнений. Рассчитываем, что это поможет вам; за подробностями советуем ходить на лекции и читать учебники.

2 Многочлены

Когда вы были в 7-м классе, наверняка вас учили делить многочлены друг на друга. Например, для деления многочлена на выражение вида $(x-a)$ обычно используется так называемая схема Горнера, которая не очень любима преподавателями МФТИ.

Рассмотрим более классический и универсальный способ — деление многочленов столбиком. Данный способ практически идентичен методу деления в столбик обычных чисел.

Выписываем наш многочлен и рядом, «на полочке», многочлен, на который будем делить — все, как с числами (обозначать будем так, как это принято в странах постсоветсткого пространства, а также Франции, Бельгии, Испании и Монголии; в других странах алгоритм тот же, но обозначения немного другие, если поступите на посдок в, например, Америку и начнёте преподавать у студентов, вы будете вначале удивлены....):

$$\begin{array}{rl} x^2+2x-12 & \big| \, x+5 \\ \cline{2-2} x^2+5x & \big| \, x-3 \\ \cline{1-1} \quad -3x-12 & \\ \quad -3x-15 & \\ \cline{1-1} \qquad\quad 3 & \end{array}$$

Задача III.1. Разделить многочлен $3x^5+2x^4+x^2-x+1$ на многочлен x^3+2x^2+x.

Решение. Опишем алгоритм деления многочленов для этого примера по шагам.

1. Запишем оба многочлена в порядке убывания степеней

$$3x^5+2x^4+x^2-x+1 = 3x^5+2x^4+0x^3+x^2-x+1,$$

$$x^3+2x^2+x = x^3+2x^2+x+0$$

2. Делим первый член делимого $3x^5$ на первый член делителя x^3. Получаем первый член частного $3x^2$.

3. Умножаем первый член частного $3x^2$ на делитель x^3+2x^2+x. Получаем многочлен $3x^5+6x^4+3x^3$ и записываем под соответственными членами.

4. Вычитаем из делимого $3x^5+2x^4+0x^3+x^2-x+1$ написанный под ним многочлен. Получаем первый остаток $-4x^4-3x^3+x^2-x+1$.

5. Делим первый член первого остатка $-4x^4$ на первый член делителя x^3. Получаем второй член частного $-4x$.

6. Умножаем второй член частного $-4x$ на делитель x^3+2x^2+x. Получаем многочлен $-4x^4-8x^3-4x^2$ и записываем его под соответственными членами первого остатка.

7. Вычитаем из первого остатка $-4x^4-3x^3+x^2-x+1$ написанный под ним многочлен. Получаем второй остаток $5x^3+5x^2-x+1$.

8. Делим первый член второго остатка $5x^3$ на первый член делителя x^3. Получаем третий член частного 5.

9. Умножаем третий член частного 5 на делитель x^3+2x^2+x. Получаем многочлен $5x^3+10x^2+5x$ и записываем его под вторым остатком.

10. Вычитаем из второго остатка $5x^3+5x^2-x+1$ написанный под ним многочлен. Получаем третий остаток $-5x^2-6x+1$.

11. Степень третьего остатка меньше степени делителя, следовательно, процесс деления завершен. Частное от деления $3x^2-4x+5$, остаток $-5x^2-6x+1$.

Данная схема реализуется следующим образом:

$$
\begin{array}{rrrrrr|l}
3x^5 & +2x^4 & +0x^3 & +x^2 & -x & +1 & x^3+2x^2+x \\
\cline{7-7}
3x^5 & +6x^4 & +3x^3 & & & & 3x^4-4x+5 \\
\cline{1-3}
 & -4x^4 & -3x^3 & +x^2 & & & \\
 & -4x^4 & -8x^3 & -4x^2 & & & \\
\cline{2-4}
 & & 5x^3 & +5x^2 & -x & & \\
 & & 5x^3 & +10x^2 & +5x & & \\
\cline{3-5}
 & & & -5x^2 & -6x & +1 &
\end{array}
$$

Алгоритм можно записать следующим образом:

1. Делим старший элемент делимого на старший элемент делителя, помещаем результат под чертой

2. Умножаем делитель на полученный выше результат деления (на первый элемент частного). Записываем результат под первыми элементами делимого.

3. Вычитаем полученный после умножения многочлен из делимого, записываем результат под чертой

4. Повторяем предыдущие 3 шага, используя в качестве делимого многочлен, записанный под чертой, пока степень старшего члена оставшегося делимого не станет меньше, чем степень старшего члена делителя.

□

При делении на двухчлены с остатком проще всего проверять себя с помощью теоремы Безу.

Теорема Безу: если многочлен $P(x)$ разделить на двучлен $(x-a)$, то в остатке получится $P(a)$ (напомним, что при делении одного многочлена на другой в остатке получается многочлен, степень которого ниже степени многочлена-делителя).

В частности, отсюда следует, что если a — корень многочлена $P(x)$, то $P(x) = (x-a)\cdot Q(x)$, где $Q(x)$ — многочлен $n-1$ степени. Отсюда же можно вывести, что многочлен $P(x)$ степени n имеет не более n различных корней: в противном случае P имеет хотя бы $n+1$ корней вида $a_1, \dots a_{n+1}$, то $P(x) = (x-a_1)\dots(x-a_{n+1})Q(x)$ и имеет таким образом степень хотя бы $n+1$.

Для поиска рациональных корней многочлена с целыми коэффициентами можно воспользоваться следующим утверждением.

Теорема 6 (о рациональных корнях)**.** Пусть $P(x) = a_n x^n + a_{n-1}x^{n-1} + \dots + a_0$ — многочлен с целыми коэффициентами. Тогда если рациональное число $\frac{p}{q}$ является корнем $P(x)$, то $p|a_0$ и $q|a_n$.

Доказательство. По условию имеем $a_n(\frac{p}{q})^n + a_{n-1}(\frac{p}{q})^{n-1} + \dots + a_0 = 0$. Умножим обе части этого равенства на q^n и получим $a_n p^n + a_{n-1}p^{n-1}q + \dots + a_0 q^n = 0$. Отсюда

$$a_n p^n = -q\big(a_{n-1}p^{n-1} + \dots a_1 p q^{n-2} + a_0 q^{n-1}\big)$$

Это равенство двух целых чисел, откуда следует, что $a_n p^n$ делится на q. Поскольку $\frac{p}{q}$ — несократимая дробь, то p и q взаимнопросты, а значит, p^n и q взаимнопросты; таким образом, $q|a_n$. Аналогично получим $p|a_0$. □

Однако следует понимать, что не каждый многочлен с целыми коэффициентами имеет целый корень (приведите пример!).

3 Разложение на множители

Теорема Безу очень хорошо помогает быстро разложить многочлены на множители (нам это будет требоваться очень часто).

Задача III.2. Решить уравнение: $P(x) = x^3 - 3x^2 - 6x - 20 = 0$.

Решение. Пользуясь теоремой о рациональных корнях, перебираем делители свободного члена: это

$$\{1, 2, 4, 5, 10, 20, -1, 2, -4, -5, -10, -20\}.$$

Убеждаемся, что $P(5) = 0$. Пользуемся теоремой Безу: делим $x^3-3x^2-6x-20$ на $x-5$, получаем x^2+2x+4 (в этом можно убедиться, пользуясь делением столбиком). Квадратное уравнение $x^2 + 2x + 4 = 0$ не имеет корней, значит, единственным корнем уравнения будет $x = 5$. □

Вообще говоря, не каждый многочлен имеет n действительных корней. Комплексные числа позволяют решить эту проблему.

Лемма 2 (основная теорема алгебры)**.** Любой многочлен $P(x)$ с комплексными коэффициентами имеет хотя бы один комплексный корень[22].

Как следствие, любой многочлен с вещественными коэффициентами можно разложить на множители:

$$p \cdot (x - a_1)^{n_1} \cdot (x - a_2)^{n_2} \cdot ..... \cdot (x - a_k)^{n_k} \cdot$$
$$(x^2 + b_1x + c_1)^{m_1} \cdot (x^2 + b_2x + c_2)^{m_2} \cdot ..... \cdot (x^2 + b_lx + c_l)^{m_l},$$

где p — вещественное число, $a_1, ..., a_k$ — корни многочлена кратностей $n_1, ..., n_k$, а трёхчлены $x^2 + b_ix + c_i$ всюду больше нуля (то есть не имеют корней)[23].

Для разложения на множители следует воспользоваться методом подбора корней, указанным выше.

Задача III.3. Разложить на множители многочлен

$$x^7 + x^6 - 5x^5 - 11x^4 + 22x^2 + 24x + 8.$$

Решение. Перебираем делители свободного члена: это {1,2, 4,8, $-1, -2, -4, -8$}, и убеждаемся, что $P(-1) = 0$. Пользуемся теоремой Безу: делим $x^7 + x^6 - 5x^5 - 11x^4 + 22x^2 + 24x + 8$ на $x + 1$ (делим столбиком, это мы уже умеем), получаем $x^6 - 5x^4 - 6x^3 + 6x^2 + 16x + 8$.

[22] Эта теорема доказывается красивым геометричным способом, если хотите разобраться, то можете получить доказательство самостоятельно, прорешав в книжке Алекссева «Теорема Абеля в задачах и примерах» [3] задачи 265, 266 и 267.

[23] Вывести наличие подобного разложения вы сможете, решив из все той же книжки Алексеева [3] задачи 269, 270 и 271.

Опять перебираем делители свободного члена и находим ещё один корень, опять же $x = -1$, снова делим на $x+1$ и получаем $x^5 - x^4 - 4x^3 - 2x^2 + 8x + 8$.

Ещё одна итерация: делим на $x + 1$ и получаем $x^4 - 2x^3 - 2x^2 + 8$.

Перебираем делители свободного члена, находим корень $x = 2$. Делим на $x - 2$ и получаем $x^3 - 2x - 4$.

И опять делим на $x - 2$ и получаем $x^2 + 2x + 2$.

Дискриминант данного квадратного трёхчлена отрицателен, поэтому мы заканчиваем итерации и получаем следующее разложение:

$$x^7 + x^6 - 5x^5 - 11x^4 + 22x^2 + 24x + 8 = (x-1)^3(x+2)^2(x^2 + 2x + 2).$$

□

4 Разложение рациональных дробей

Как только у вас начнутся интегралы (формально они будут во втором семестре, но многие преподаватели начинают давать это и в первом, да и в физике интегралы нужны сразу), вы начнёте сходить с ума из-за интегрирования рациональных дробей. Простые дроби решаются методами, близкими к табличным, что же делать с большой дробью?

Если у нас есть дробь вида $\dfrac{P(x)}{Q(x)}$, мы должны представить её в виде суммы так называемых рациональных дробей и воспользоваться тем фактом, что интеграл от суммы равен сумме интегралов.

Если степень числителя не меньше степени знаменателя, то нам необходимо воспользоваться делением с остатком и получить, $P(x) = A(x) \cdot Q(x) + B(x)$, из чего следует, что $\dfrac{P(x)}{Q(x)} = A(x) + \dfrac{B(x)}{Q(x)}$. Интеграл от многочлена ищется очень просто, так как это сумма табличных интегралов.

Напомним табличные значения:

$$\int A \cdot dx = A \cdot x + C,$$

$$\int A \cdot x^n dx = A \cdot \frac{x^{n+1}}{n+1} + C.$$

Не забываем при нахождении неопределённого интеграла прибавлять константу!

Но как найти интеграл от дроби вида $\dfrac{B(x)}{Q(x)}$, где степень числителя меньше, чем степень знаменателя?

Предварительно необходимо разложить знаменатель данной дроби на элементарные множители.

Разложив на множители знаменатель, получим

$$\frac{B(x)}{p\cdot(x-a_1)^{n_1}\cdot\dots\cdot(x-a_k)^{n_k}\cdot(x^2+b_1x+c_1)^{m_1}\cdot\dots\cdot(x^2+b_lx+c_l)^{m_l}}.$$

Такая дробь представима в виде

$$\sum_{i=1}^{n_1}\frac{A_i}{(x-a_1)^i}+\dots.\sum_{i=1}^{m_1}\frac{D_ix+E_i}{(x^2+b_1x+c_1)^i}+\dots.$$

Расшифруем данные иероглифы...

Задача III.4. Представьте дробь $\dfrac{x^4+10x^2+12x+10}{x^5-x^4-4x^3-2x^2+8x+8}$ в виде суммы элементарных.

Решение.

$$\frac{x^4+10x^2+12x+10}{x^5-x^4-4x^3-2x^2+8x+8}=\frac{A}{(x-2)}+\frac{B}{(x-2)^2}+\frac{C}{x+1}+\frac{Dx+E}{x^2+2x+2}.$$

Домножим обе части на знаменатель левой части для избавления от дробей:

$$x^4+10x^2+12x+10=A\cdot(x-2)(x+1)(x^2+2x+2)+$$
$$+B\cdot(x+1)(x^2+2x+2)+C\cdot(x-2)^2(x^2+2x+2)+$$
$$+(Dx+E)\cdot(x-2)^2(x+1).$$

Раскроем скобки слева, получим:

$$x^4+10x^2+12x+10=A(x^4+x^3-2x^2-6x-4)+$$
$$+B(x^3+3x^2+4x+2)+C(x^4-2x^3-2x^2+8)+$$
$$+(Dx+E)(x^3-3x^2+4),$$

$$x^4+10x^2+12x+10=(A+C+D)x^4+(A+B-2C+E-3D)x^3+$$
$$+(-2A+3B-2C-3E)x^2+(-6A+4B+4D)x+$$
$$+(-4A+2B+8C+4E).$$

Так как нам необходимо получить тождество, приравняем коэффициенты при каждой степени переменной x:

$$\begin{cases} 1 &= A + C + D \\ 0 &= A + B - 2C + E - 3D \\ 10 &= -2A + 3B - 2C - 3E \\ 12 &= -6A + 4B + 4D \\ 10 &= -4A + 2B + 8C + 4E \end{cases}$$

Решив систему линейных уравнений методами, описанными в 1, получаем

$$A = 0; B = 3; C = 1; D = 0; E = -1.$$

Таким образом,

$$\frac{x^4 + 10x^2 + 12x + 10}{x^5 - x^4 - 4x^3 - 2x^2 + 8x + 8} = \frac{3}{(x-2)^2} + \frac{1}{x+1} - \frac{1}{x^2 + 2x + 2}.$$

□

5 Основы аналитической геометрии и векторной алгебры

Координаты. Расстояние между точками

Система координат в пространстве — три взаимно перпендикулярные оси x, y и z.
Пусть точка M — середина отрезка AB. Её координаты находятся по формуле:

$$x_M = \frac{x_A + x_B}{2}; \quad y_M = \frac{y_A + y_B}{2}; \quad z_M = \frac{z_A + z_B}{2}$$

Координаты точки $M(x,y,z)$, делящей отрезок M_1M_2 между точками $M_1(x_1,y_1,z_1)$ и $M_2(x_2,y_2,z_2)$ в отношении $M_1M : MM_2 = \lambda$, определяются формулами:

$$x = \frac{x_1 + \lambda x_2}{1 + \lambda}; \quad y = \frac{y_1 + \lambda y_2}{1 + \lambda}; \quad z = \frac{z_1 + \lambda z_2}{1 + \lambda}$$

Расстояние между точками $A(x_1,y_1,z_1)$ и $B(x_2,y_2,z_2)$ определяется по формуле:

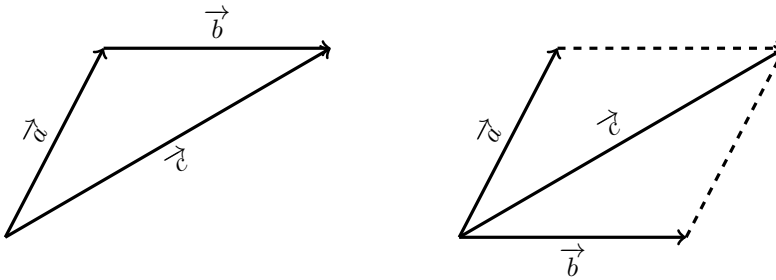
$$\rho(A,B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Векторы

Координаты вектора $\vec{a} = \overrightarrow{AB}(x_B - x_A; y_B - y_A; z_B - z_A)$

Длина вектора $|\vec{a}| = \sqrt{x_a^2 + y_a^2 + z_a^2} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$

Сложение векторов по правилу треугольника и правилу параллелограмма:



Сумма векторов, их разность, произведение вектора на число и скалярное произведение определяются так же, как и на плоскости. Только координат не две, а три. Возьмём векторы $\vec{a}(x_a,y_a,z_a)$ и $\vec{b}(x_b,y_b,z_b)$.

Сумма векторов:

$$\vec{a} + \vec{b} = \vec{c}(x_a + x_b,\ y_a + y_b,\ z_a + z_b)$$

Разность векторов:

$$\vec{a} - \vec{b} = \vec{d}(x_a - x_b,\ y_a - y_b,\ z_a - z_b)$$

Произведение вектора на число:

$$\lambda\vec{a} = \vec{p}(\lambda x_a,\lambda y_a,\lambda z_a)$$

Скалярное произведение векторов

$$\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos\varphi = x_a \cdot x_b + y_a \cdot y_b + z_a \cdot z_b$$

Косинус угла между векторами:

$$\cos\varphi = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|} = \frac{x_a \cdot x_b + y_a \cdot y_b + z_a \cdot z_b}{\sqrt{x_a^2 + y_a^2 + z_a^2} \cdot \sqrt{x_b^2 + y_b^2 + z_b^2}}$$

Будем говорить, что вектора $\vec{a}$ и $\vec{b}$ *ортогональны*, если $\vec{a} \cdot \vec{b} = 0$. Косинус угла, образованный ненулевыми ортогональными векторами, равен 0, то есть сам угол есть $\pm 90°$.

Работа с определителями

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - cb$$

$$\begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = a_1 \cdot \begin{vmatrix} b_2 & b_3 \\ c_2 & c_3 \end{vmatrix} - a_2 \cdot \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix} + a_3 \cdot \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix}$$

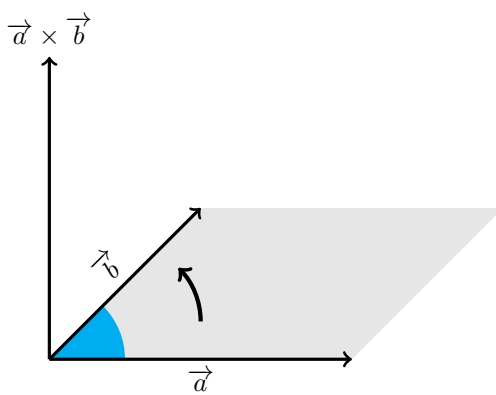
Векторное произведение векторов

$\vec{c} = \vec{a} \times \vec{b} \quad \Leftrightarrow \quad |\vec{c}| = |\vec{a}| \cdot \vec{b} \cdot \sin \angle(\vec{a},\vec{b});\ \vec{c} \perp \vec{a};\ \vec{c} \perp \vec{b}$

$$\vec{a} \times \vec{b} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_a & y_a & z_a \\ x_b & y_b & z_b \end{vmatrix}$$

$\vec{a} \times \vec{b}$ по модулю равен площади параллелограмма, построенного на векторах $\vec{a}$ и $\vec{b}$.

$S_{ABC} = \frac{1}{2}\left|\overrightarrow{AB} \times \overrightarrow{AC}\right|$ — площадь треугольника ABC. Сам же результирующий вектор будет ортогонален векторам $\vec{a}$ и $\vec{b}$ и будет ориентирован по правилу буравчика.



Смешанное произведение векторов

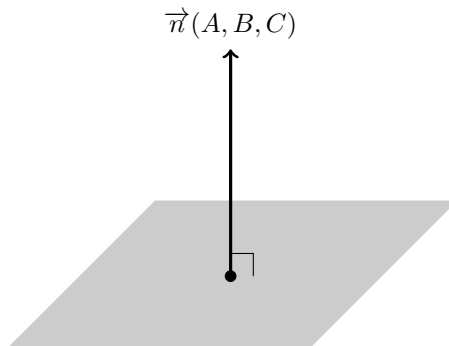
$$(\vec{a},\vec{b},\vec{c}) = (\vec{a},\ \vec{b} \times \vec{c}) = \begin{vmatrix} x_a & y_a & z_a \\ x_b & y_b & z_b \\ x_c & y_c & z_c \end{vmatrix}$$

$(\vec{a},\vec{b},\vec{c})$ равно объёму параллелепипеда, построенного на векторах $\vec{a}$, $\vec{b}$ и $\vec{c}$. $V_{ABCD} = \frac{1}{6}\left|(\overrightarrow{AB},\overrightarrow{AC},\overrightarrow{AD})\right|$ — объём треугольной пирамиды $ABCD$.

Уравнение плоскости и вектор нормали

$$Ax + By + Cz + D = 0,$$

где A,B и C — координаты вектора, перпендикулярного этой плоскости. Его называют *нормалью* к плоскости.



Найти уравнение плоскости можно двумя способами:

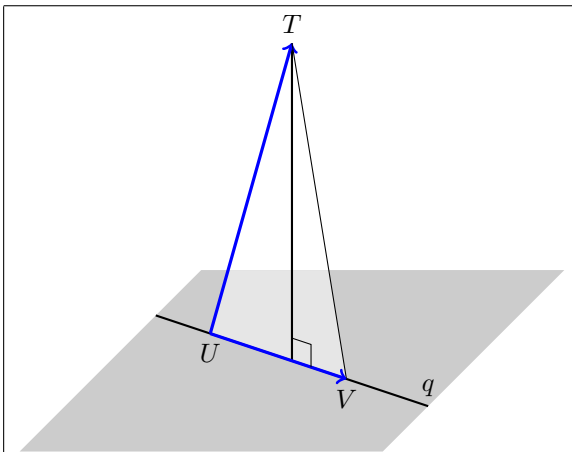
- Подставляем в уравнение плоскости координаты любых трёх точек. Получаем систему трёх линейных уравнений с четырьмя неизвестными (A,B,C и D). Вместо одной из переменных подставляем любое число, отличное от нуля. Решая систему, находим остальные три неизвестные. (Может оказаться, что система не имеет решения, тогда надо поменять одну из точек, либо воспользоваться вторым способом)

- Уравнение плоскости ищем в виде:

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0$$

  Далее, раскрывая определитель, приводим его к виду $Ax+By+Cz+D = 0$.

Расстояние от точки до прямой



$$\rho(T,q) = \frac{|\overrightarrow{UT} \times \overrightarrow{UV}|}{|\overrightarrow{UV}|}$$

Угол между двумя прямыми

Для нахождения угла между векторами, параллельными данным прямым, используем формулу

$$\cos\varphi = \frac{|x_1x_2 + y_1y_2 + z_1z_2|}{\sqrt{x_1^2 + y_1^2 + z_1^2} \cdot \sqrt{x_2^2 + y_2^2 + z_2^2}}$$

Угол между прямой и плоскостью

Угол между прямой и плоскостью равен $\varphi = 90^\circ - \gamma$, где γ — угол между прямой и нормалью к плоскости.

Вектор нормали к плоскости, задаваемой уравнением
$Ax + By + Cz + D = 0$ есть $\vec{n}(A,B,C)$.

Угол между двумя плоскостями

Ищем угол между нормалями (перпендикулярами) к плоскостям. Нормаль к плоскости можно находить несколькими способами (см. выше).

Нахождение расстояния от точки M до плоскости α

Находим уравнение плоскости. Тогда искомое расстояние равно:

$$\rho(M,\alpha) = \frac{|Ax_0 + By_0 + Cz_0 + D|}{\sqrt{A^2 + B^2 + C^2}},$$

где $M(x_0,y_0,z_0)$, а плоскость α задана уравнением $Ax + By + Cz + D = 0$.

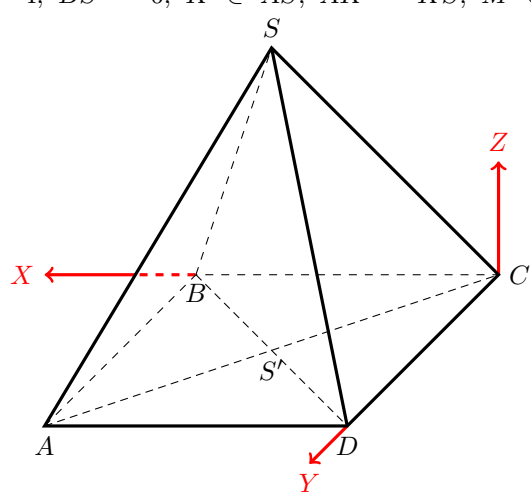
Расстояние между скрещивающимися прямыми

Вычисляем объём пирамиды, вершины которой — 2 точки первой и 2 точки второй прямой. Тогда

$$\rho(AB,CD) = \frac{6V_{ABCD}}{AB \cdot CD \cdot \sin\varphi},$$

где A и B — точки на одной прямой, C и D — точки на другой прямой, φ — угол между данными прямыми.

Задача III.5. Имеется правильная четырёхугольная пирамида $ABCDS$, в которой $AB = 4$, $BS = 6$, $K \in AS$, $AK = KS$, $M \in DS$, $MS = 1{,}5$.



Найти:

- угол между прямыми KM и CS;
- расстояние между прямыми KM и CS;
- расстояние от точки K до прямой MC;

- расстояние от точки K до плоскости SMC;
- угол между прямой KM и плоскостью SMC;
- угол между плоскостями KMC и SMC.

Решение. • Найдём координаты точек K, M, C, S.

$C(0;0;0)$ $S(2;2;2\sqrt{7})$, $K(3;3;\sqrt{7})$ $M(\frac{1}{2};\frac{7}{2};\frac{\sqrt{7}}{2})$

- Вектор $\overrightarrow{KM}(-\frac{5}{2};\frac{1}{2};-\frac{\sqrt{7}}{2})$;

  Вектор $\overrightarrow{CS}(2;2;2\sqrt{7})$;

  Пусть α — угол между прямыми KM и CS.

$$\cos\alpha = \frac{\left|-\frac{5}{2}\cdot 2+\frac{1}{2}\cdot 2-\frac{\sqrt{7}}{2}\cdot 2\sqrt{7}\right|}{\sqrt{\frac{25}{4}+\frac{1}{4}+\frac{7}{4}}\cdot\sqrt{4+4+28}} = \frac{\sqrt{33}}{9},$$

$$\alpha = \arccos\frac{\sqrt{33}}{9}.$$

- Угол между плоскостями KMC и SMC равен углу между нормалями данных плоскостей $\overrightarrow{n_1}$ и $\overrightarrow{n_2}$.

  Уравнение плоскости KMC:

$$\begin{vmatrix} x & y & z \\ 3 & 3 & \sqrt{7} \\ \frac{1}{2} & \frac{7}{2} & \frac{\sqrt{7}}{2} \end{vmatrix} = 0$$

  Раскрываем:

$$x\left(\frac{3\sqrt{7}}{2}-\frac{7\sqrt{7}}{2}\right) - y\left(\frac{3\sqrt{7}}{2}-\frac{\sqrt{7}}{2}\right) + z\left(3\cdot\frac{7}{2}-\frac{3}{2}\right) = 0$$

$$2\sqrt{7}x + \sqrt{7}y - 9z = 0$$

  Тогда $\overrightarrow{n_1}(2\sqrt{7};\sqrt{7};-9)$.

  Уравнение плоскости SMC:

$$\begin{vmatrix} x & y & z \\ 2 & 2 & 2\sqrt{7} \\ \frac{1}{2} & \frac{7}{2} & \frac{\sqrt{7}}{2} \end{vmatrix} = 0$$

$$x\left(\sqrt{7}-7\sqrt{7}\right)-y\left(\sqrt{7}-\sqrt{7}\right)+z\,(7-1)=0$$

$$-\sqrt{7}x+z=0$$

Тогда $\overrightarrow{n_2}(-\sqrt{7};0;1)$.

Пусть β — угол между векторами $\overrightarrow{n_1}$ и $\overrightarrow{n_2}$. Тогда

$$\cos\beta=\frac{\left|-2\sqrt{7}\cdot\sqrt{7}+\sqrt{7}\cdot 0-9\cdot 1\right|}{\sqrt{28+7+81}\cdot\sqrt{7+0+1}}=\frac{23\sqrt{58}}{232},$$

$$\beta=\arccos\frac{23\sqrt{58}}{232}.$$

- Пусть γ — угол между прямой KM и плоскостью SMC.

$$\sin\gamma=\cos(\overrightarrow{n_2},\overrightarrow{KM})=\frac{\left|-\frac{5}{2}\cdot(-\sqrt{7})+\frac{1}{2}\cdot 0-\frac{\sqrt{7}}{2}\cdot 1\right|}{\sqrt{\frac{25}{4}+\frac{1}{4}+\frac{7}{4}}\cdot\sqrt{7+0+1}}=\frac{\sqrt{462}}{33},$$

$$\gamma=\arcsin\frac{\sqrt{462}}{33}.$$

- Расстояние от точки K до плоскости SMC

$$KQ=\rho(K,SMC)=\frac{\left|-\sqrt{7}\cdot 3+0\cdot 3+1\cdot\sqrt{7}\right|}{\sqrt{\sqrt{7}^2+0^2+1^2}}=\frac{\sqrt{14}}{2}.$$

- Расстояние от точки K до прямой MC

$$\rho(K,MC)=\frac{\left|\overrightarrow{CK}\times\overrightarrow{CM}\right|}{\left|\overrightarrow{CM}\right|},$$

$$\overrightarrow{CK}\times\overrightarrow{CM}=\begin{vmatrix}\overrightarrow{i} & \overrightarrow{j} & \overrightarrow{k}\\ 3 & 3 & \sqrt{7}\\ \frac{1}{2} & \frac{7}{2} & \frac{\sqrt{7}}{2}\end{vmatrix}=$$

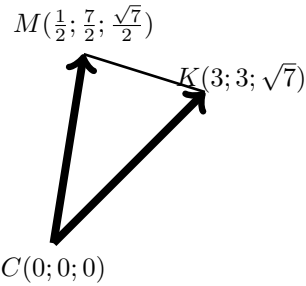
$$=\overrightarrow{i}\cdot\left(3\cdot\tfrac{\sqrt{7}}{2}-\tfrac{7\sqrt{7}}{2}\right)-\overrightarrow{j}\cdot\left(3\tfrac{\sqrt{7}}{2}-\tfrac{\sqrt{7}}{2}\right)+\overrightarrow{k}\cdot\left(\tfrac{3\cdot 7}{2}\cdot\tfrac{3}{2}\right)=$$

$$=-2\sqrt{7}\,\overrightarrow{i}-\sqrt{7}\,\overrightarrow{j}+9\,\overrightarrow{k},$$

$$\left|\overrightarrow{CK} \times \overrightarrow{CM}\right| = \sqrt{28+7+81} = 2\sqrt{29},$$

$$\left|\overrightarrow{CM}\right| = \sqrt{\frac{1}{4}+\frac{49}{4}+\frac{7}{4}} = \frac{\sqrt{57}}{2},$$

$$\rho(K,MC) = \frac{2\sqrt{29}}{\frac{\sqrt{57}}{2}} = \frac{4\sqrt{1653}}{57}.$$



- Расстояние между прямыми KM и CS

$$\rho(KM,CS) = \frac{6V_{KMCS}}{|\overrightarrow{KM}| \cdot |\overrightarrow{CS}| \cdot \sin\alpha}.$$

$$V_{KMCS} = \frac{1}{6}\left|\begin{vmatrix} 2 & 2 & 2\sqrt{7} \\ 3 & 3 & \sqrt{7} \\ \frac{1}{2} & \frac{7}{2} & \frac{\sqrt{7}}{2} \end{vmatrix}\right| = \frac{1}{6}\left|2\cdot(-2\sqrt{7}) - 2\cdot\sqrt{7} + 2\sqrt{7}\cdot 9\right| = 2\sqrt{7}.$$

$$\rho(KM,CS) = \frac{6\cdot 2\sqrt{7}}{\frac{\sqrt{33}}{2}\cdot 6} = \frac{4\sqrt{231}}{33}.$$

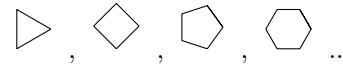
□

6 Последовательности и пределы

Первые объекты, с которыми работает математический анализ — это **последовательности**. Под ними подразумевается бесконечный упорядоченный набор элементов или *членов последовательности*:

$$\{a_1, a_2, a_3, \ldots, a_n, \ldots\}.$$

Элементом последовательности может быть всё что угодно: числа, геометрические объекты, точки на плоскости или в пространстве... Ниже представлена последовательность правильных n-угольников



В математическом анализе вы будете работать вначале с последовательностями из чисел. Например, представьте, что у вас есть последовательность, у которой n-й элемент $a_n = \frac{1}{n}$. Тогда если вы возьмёте калькулятор и попробуете посчитать первые несколько членов, то увидите, как увеличивается число нулей после запятой. Тем более, если вас спросят чему равен миллионный член, то проще сказать, что это почти ноль, чем выписывать громадное количество нулей в этом числе.

Первая задача математического анализа и конкретизирует это слово «почти» в предыдущем предложении. А именно, вводится понятие **предела последовательности**, подразумевая под этим, что чем больше номер члена последовательности, тем «ближе» он к пределу. Строго говоря,

$$A = \text{предел}\,\{a_k\}_{k=1}^{\infty} \Leftrightarrow \forall \epsilon > 0\, \exists N = N(\epsilon) : \forall n \geqslant N \mapsto |a_n - A| < \epsilon.$$

В этом определении подразумевается, что какую бы маленькую ϵ-окрестность числа A, то есть интервал $U_\epsilon(A) = (A - \epsilon; A + \epsilon)$, мы не выбрали, то, начиная с некоторого члена, все члены последовательности будут в этой окрестности. Принято говорить, что последовательность **сходится**, если у нее есть предел.

В некоторых случаях вполне понятно, что должно быть пределом последовательности. Например, для правильных многоугольников выше пределом будет окружность. Однако на матане вас заставят строго доказывать, что число является пределом.

В случае с последовательностью $\{\frac{1}{n}\}_{n=1}^{\infty}$ поможет **принцип Архимеда**, который гласит, что для любого действительного числа существует натуральное число большее его. Воспользовавшись им, можно сказать, что существует число $n_\epsilon \in \mathbb{N}$, которое будет больше $\frac{1}{\epsilon}$. Тогда все члены последовательности, имеющие индекс больший n_ϵ, будут лежать в $U_\epsilon(0)$ и по определению получаем, что ноль — предел этой последовательности.

Нельзя забывать, что не у всех последовательностей есть предел, например, его нет у следующей последовательности

$$\{1, -1, 1, -1, 1, -1, 1, -1, 1, -1, \dots\}.$$

В каждом таком случае всегда можно строго доказать, что последовательность не имеет предела или говорят, что она *расходится*. Для этого надо

доказать, что $\forall A \in \mathbb{R}$ A не является пределом заданной последовательности $\{a_k\}_{k=1}^{\infty}$. Последнее же на языке кванторов выглядит следующим образом

$$\exists \epsilon > 0 \colon \forall N \ \ \exists n \geqslant N \colon |a_n - A| \geqslant \epsilon.$$

На примере чередующейся последовательности единиц и минус единиц можно заметить, что все числа $A \in \mathbb{R}, A \neq 1, A \neq -1$ не будут пределами последовательности (в этих случаях достаточно взять $\epsilon = \frac{|1-|A||}{2}$), поэтому имеет смысл рассматривать только 1 и -1. Они в свою очередь не будут пределами, потому что при $\epsilon = 1$ какой бы большой индекс N мы не выбрали, будет больший либо чётный, либо нечётный (в зависимости от того, что мы предполагаем $A = 1$ или $A = -1$) $n \geqslant N$ такой, что $|a_n - A| = 2 > \epsilon$.

Однако, как мог заметить читатель, если разбить последовательность $\{a_k\}_{k=1}^{\infty} = \{1, -1, 1, -1, \dots\}$ на две подпоследовательности: состоящую из 1 и состоящую из -1, то они обе имеют пределы. В этом случае говорят, что последовательность $\{a_k\}_{k=1}^{\infty}$ имеет два *частичных предела* — 1 и -1. В общем случае **частичный предел** — предел какой-либо сходящейся подпоследовательности.

Запомните, что чтобы вывести из какого-либо выражения в кванторах утверждение, говорящее что-то о характере сходимости последовательности, то полезно «вручную» строить подходящие последовательности, подставляя $\epsilon_n = \frac{1}{n}$. Объясним, что подразумевается в предыдущем предложении; для этого докажем следующее утверждение.

Утверждение 1. Следующие три утверждения эквивалентны:

1° A — частичный предел последовательности $\{a_k\}_{k=1}^{\infty}$;

2° В любой $U_\epsilon(A)$ имеется бесконечное число членов последовательности;

3° Для любого натурального N и любого $\epsilon > 0$ найдётся $n \geqslant N \colon |a_n - A| < \epsilon$.

Доказательство. (1° $\Rightarrow$ 2°) Так как элементы подпоследовательности являются элементами последовательности, то по определению в любой ϵ-окрестности числа A, начиная с некоторого индекса, будут находится все члены подпоследовательности, то есть бесконечное количество элементов подпоследовательности. А значит, так как все они являются членами $\{a_k\}_{k=1}^{\infty}$, следовательно, выполнено 2° условие.

(2° $\Rightarrow$ 3°) И тут почти все очевидно: для любого ϵ в $U_\epsilon(A)$ бесконечное число элементов последовательности. Следовательно, если мы выкинем из нее конечное число членов (вплоть до N), то все ещё найдётся такой индекс $n \geqslant N$, что $a_n \in U_\epsilon(A) \Rightarrow |a_n - A| < \epsilon$.

(3° $\Rightarrow$ 1°) Наконец воспользуемся тем методом, который мы описали в абзаце

до этого утверждения. А именно возьмём $N = 1$ и $\epsilon = 1$, тогда есть элемент $a_{\alpha_1} \in U_1(A)$.

Далее возьмем $N = \alpha_1 + 1$ и $\epsilon = \frac{1}{2}$, тогда есть элемент $a_{\alpha_2} \in U_{\frac{1}{2}}(A)$.

Повторим итерацию для $N = \alpha_2 + 1$ и $\epsilon = \frac{1}{3}$ и получим элемент a_{α_3}...

Таким образом, мы получим подпоследовательность $\{a_{\alpha_k}\}_{k=1}^{\infty}$, у которой будет предел равен A. Это означает, что A — частичный предел последовательности $\{a_k\}_{k=1}^{\infty}$. □

7 Что читать дальше

Каждый лектор по матану на физтехе считает своим священным долгом оформить конспект своих лекций в виде пособия. Кудрявцев, Яковлев, Бесов, Тер-Крикоров, Шабунин, Петрович [35, 10, 26, 32, 38]— все они имеют пособие имени себя (ну или нескольких из них) по матану. Можно пойти альтернативным путём и открыть мехматскую классику. Мы советуем в таком случае осилить учебник Зорича или Фихтенгольца [19]. Хороший, глубокий учебник по матану был в своё время написан Курантом [27]. Вообще учебников по матану несметное множество, главное — не спугнуть экзаменатора незнакомой ему формулировкой. (Это же, впрочем, касается и остальных математических дисциплин.) Кстати, имейте в виду, не все лекторы любят ответы не по его лекциям, так же, как и молодой экзаменатор, не поняв вашего ответа, может обратиться к лектору и опять же получить плохую рекомендацию. Поэтому искренне советуем в своём ответе покрыть всё, рассказанное лектором вашего потока, а дополнительную литературу использовать сверх, а не вместо лекционных материалов.

Аналитическая геометрия — предмет первого семестра, и зелёный учебник Беклемишева [7] целиком покрывает предмет. Все изложено как надо, ни убавить ни прибавить. Непосредственно геометрии per se в нем немного, поскольку по умолчанию физтехи в ней не нуждаются, поэтому любителям стоит обратить взор на другую литературу — особенно на «Геометрию» [8, 9] Марселя Берже.

На фупме алгебра разведена по двум предметам второго семестра. «Континуальная» алгебра, посвященная линейным операторам, собственным подпространствам и LU-разложениям, проходится на линейной алгебре, а «дискретная» алгебра, посвященная группам, кольцам и полям составляет предмет «Основы высшей алгебры и теории кодирования»[24]. Линейная алгебра по-

[24]В последнее время мастодонты кафедры моу вполне справедливо решили, что непосредственно самой криптографии на фупме маловато, и добавили костыль в виде дополнительного спецкурса (обязательного для всех, разумеется) в четвертом семестре. Что получилось

крывается учебником Беклемишева[25], а оваитк — стандартным учебником кафедры моу [16].

Обе эти книжки, впрочем, покрываются стандратным мехматским учебником Кострикина и задачником Манина и Кострикина [22, 23, 24, 25]. Авторы считают, что алгебру лучше познавать целиком, а не по частям, поэтому ознакомление с алгеброй по-математически будет полезным. Если кто-то позарился на уровень НМУ, то может осилить курс Городенцева [14] — у него отличный учебник со строгими, но не громоздкими формулировками и прекрасные подборки задач в листках. По теории групп зеленому физтеху можно посоветовать также «Теорему Абеля в задачах и решениях» Алексеева [3]. Читателю предлагается по ходу книжки разобраться с теорией групп, началами комплексного анализа (включая основную теорему алгебры и ее следствия) и даже с римановыми поверхностями, а в конце воспользоваться полученными знаниями и доказать жемчужину классической математики. Решения всех задач есть в конце, но мы настоятельно советуем прорешать задачи самостоятельно — автор разбил сложное доказательство на много простых частей.

Последнее, что можно добавить — все эти бесконечные учебники-пособия-конспекты совершенно бесполезны, если вы не умеете решать задачи. Как бы ни ценилось знание теории, умение решать задачу первостепенно и является главной причиной, почему вы вообще учитесь здесь, а не на психфаке МГУ. Поэтому во всех учебниках старайтесь прорешать как можно больше задач.

на практике — спросите своих кураторов.

[25]Раздел с тензорами, впрочем, написан на физическом уровне строгости, и тензоры лучше познавать по другой книжке.

Глава IV
Основы информатики

1 Введение

Невозможно представить себе современный мир без информационных технологий. Если вы попали на ФУПМ, то, вероятно, вы уже понимаете, что ваша деятельность в институте и, скорее всего, после него, непосредственным образом будет связана с информацией и её обработкой. Для очень большого количество выпускников физтеха знание алгоритмов и умение программировать на каком-либо языке обеспечивает хлеб с маслом (и, возможно, с икрой), какую бы вы кафедру на выбрали. Если же вы свяжете свою дальнейшую судьбу с математическим моделированием — без знания эффективных алгоритмов вы далеко не уедете или не улетите. Экономика без умения обрабатывать поступающую информацию будет не сильно отличаться от якобы предмета «Политэкономия», изучавшегося в Советском Союзе и представлявшего собой в большинстве мест изучения чисто гуманитарные беседы о неизбежном счастливом будущем. Ну а о чисто информатических кафедрах и говорить нечего.

Все физтеховские преподаватели в курсе, как информатику преподают в школе. Школьная информатика скорее похожа на уроки труда в 20-м веке, когда вам дают в руки какие-то инструменты и учат делать что-то типа табуреток — с одной стороны, достаточно практично, с другой — безумно скучно. На информатике в школе вы учились, в основном, работе с инструментом — компьютером. Обычно больше половины времени уходило на подготовку с помощью компьютера разнообразных документов — презентаций, роликов, картинок. При всём своём уважении к школьному образованию, невозможно сказать, что вы занимались именно информатикой. Умение набирать красивый текст, конечно, полезно, но к информатике его отнести никак нельзя — это просто полезный житейский навык, который вам пригодится в будущем.

А ведь информатика — наука об информации — является частью математики, прикладной математики. Настоящая информатика (за рубежом она называется *Computer Science*) изучает информацию и методы её обработки. Краеугольным понятием информатики является алгоритм, то есть, строго определённый порядок действий по обработке информации. Для записи алгоритмов используют формализованные языки — языки программирования. На первом курсе вы будете изучать один из наиболее простых языков, на котором написаны триллионы строк кода — язык `Си`. В отдельных группах вас уже в первом семестре будут давать основы `C++`, в других группах это может быть

отложено до 4-го семестра (на тех факультетах, на которых на информатику отведено 4 семестра). Во втором семестре вас познакомят с архитектурой современных вычислительных машин и кто-то из вас научится понимать программы, написанные на языках ассемблера. Почему именно языках? Потому, что каждая компьютерная архитектура имеет свой набор примитивных команд и свои способы записи этих команд — язык ассемблера. Третий семестр вы будете изучать теорию операционных систем — то, что позволяет компьютеру одновременно исполнять несколько взаимодействующих друг с другом программ, в то время, как другие программы тоже исполняются на том же компьютере в изолированной среде. В четвёртом семестре (некоторым группам и раньше) вам предстоит проект — более-менее значительная программа, делающая что-либо полезное. Обязательные условие для получения отличной оценки в четвёртом семестре — проект должен исполняться группой из 2-4 человек. Вы должны будете освоить средства совместной разработки и самостоятельно изучить какие-либо инструменты для создания пользовательского интерфейса, возможно, графического. Дальнейшее изучение информатики зависит от базовых кафедр и на некоторых оно продолжается все 12 семестров, до диплома.

2 Программирование

Мы можем разделить программирование на две большие составные части — что делать и как делать.

Представьте, что вам дали задание: написать программу, моделирующую движение на перекрёстке. Что понадобится для её реализации? Какие этапы возникнут перед вами при этом?

1. **Разработка модели**. Требуется понимание физической и психологической моделей: как ведут себя автомобили на дороге; какими уравнениями описывается движение автомобиля; что есть взаимодействие автомобилей между собой и с элементами дорожной инфраструктуры... Этот список можно продолжать долго и это — необходимая часть для реализации задания.

2. **Организационная часть**. Возможно, подобную задачу ваши работодатели уже решали. Имеются ли у них готовые фрагменты программ, которые вам помогут? Возможно, они имеются, но на языке `FORTRAN`, а вам требуется реализовать на `C++`. Сколько людей в наличии для исполнения проекта? Каковы их роли? Что они умеют?

3. **Алгоритмическая часть**. Какие средства применять для реализации деталей программы? Стоит ли, например, для реализации расположения

машин вблизи перекрёстка применять двусвязный список или можно использовать массив? Нужно ли при расчёте оптимального маршрута движения по городу использовать алгоритм Дейкстры или можно обойтись каким-либо более простым алгоритмом? Знание различных алгоритмов и их свойств помогает выбрать наиболее подходящий способ решения задачи.

4. **Реализационная часть**. А как на выбранном языке программирования реализовать абстракцию «отображение»? Какие классы требуется создать для удобства дальнейшей разработки в команде? Какой вид записи цикла наиболее эффективен? Когда приступать к этапу оптимизации программы? Эти вопросы задаются уже при написании самой программы, или, как говорят, при её кодировании.

Мы не называем эти части этапами, так как при реализации программы можно вернуться к модели и уточнить её, изменив заодно что-то в алгоритмической части.

Первая часть — то, чем традиционно сильны программисты, заканчивающие Физтех. Невозможно создать качественную программу, не вникнув в физические принципы взаимодействия объектов и не построив хорошую математическую модель — это вам скажут на любой базовой кафедре.

Без достаточного внимания к организационной части даже фирма с сильными программистами недолго останется на плаву. Хорошая организация работ позволит собрать группу разноплановых людей и они вместе смогут создать что-то интересное и полезное. На некоторых кафедрах вы будете изучать и эту сторону программной индустрии.

Алгоритмическая часть должна бы изучаться параллельно с реализационной, но у нас этим занимаются разные кафедры. К сожалению, при изучении предмета «Алгоритмы и методы вычислений» программировать и реализовывать только что пройденные алгоритмы вам не потребуется. А при изучении языков программирования обычно дают самые базовые алгоритмы. К сожалению, на физтехе пока нет предмета, который объединил бы эти две части воедино.

Если говорить кратко, то *алгоритм* — это последовательность команд для некоего *исполнителя*, которая обладает рядом свойств:

- **полезность**, то есть умение решать поставленную задачу;
- **детерминированность**, то есть каждый шаг алгоритма должен быть строго определён во всех возможных ситуациях.
- **конечность**, то есть способность алгоритма завершиться для любого множества входных данных

- **массовость**, то есть применимость алгоритма к разнообразным входным данным.

Любой алгоритм заключается в обработке *входных данных* с целью получения *выходных данных*. В процессе обработки алгоритмы могут использовать *промежуточные данные* и часто бывает удобным, чтобы эти данные были каким-либо образом *упорядочены*, образуя *структуры данных*. Вам предстоит достаточно сложный курс алгоритмов и методов вычислений, на котором вы более формально сможете изучить свойства алгоритмов и доказать их корректность.

Каждый алгоритм для своего исполнения (ещё говорят *вычисления*) требует от исполнителя некоторых *ресурсов*. *Программа* есть запись алгоритма на формальном языке.

Одну и ту же задачу зачастую можно решить несколькими способами, несколькими алгоритмами, которые могут отличаться использованием ресурсов, таких, как *элементарные действия* и *элементарные объекты*. Например, исполнитель алгоритма *компьютер* использует устройство *центральный процессор* для исполнения таких элементарных действий, как сложение, умножение, сравнение, переход и других, и устройство *память* как хранителя элементарных объектов целых и вещественных чисел. Способность алгоритма использовать ограниченное количество ресурсов называется *эффективностью*.

3 Языки программирования

Вряд-ли кто-нибудь из вас не знает о существовании языка `Pascal`, а кто-то проходил его в школе. Можно ли считать этот язык способным описывать алгоритмы? Конечно. Вместо того, чтобы оперировать элементарными операциями такими, как «сравнить две ячейки памяти; если первая больше второй, то выбрать следующую команду для исполнения в ячейке 135; присвоить третьей ячейки значение, извлечённое из второй ячейки; перейти к ячейке 128;» можно написать

```
if a > b then c := b;
```

Другой популярный язык — `Python`. Имеется мнение, что ему учиться легче, чем многим другим языкам. Для небольших — строк 100-200 — он вполне может подойти. Те же самые действия, которые мы проводили только что, иллюстрируя `Pascal`, на `Python` смотрятся очень похоже:

```
if a > b:
    c = b;
```

Кстати, и на Си мы видим нечто подобное:

```
if (a > b) c = b;
```

Мы видим, что для формулировки своих мыслей (если так, то делаем вот так) все приведённые выше примеры очень похожи и отличаются мелочами — здесь нужно слово then, здесь — знак двоеточия :, здесь — ничего не нужно или же нужно окружить выражение скобками, но суть у всех этих способов выражения своей мысли одна и та же. Известно, что все современные языки программирования достаточно мощны для того, чтобы решать большое количество задач и выбор того или иного определяется либо привязанностью конкретного человека или конкретной группы людей либо наличием в конкретном языке неких средств, наиболее подходящих для решения конкретной задачи.

Для небольших задач, до 10-20 тысяч строк кода, всё вышесказанное справедливо. Для более крупных или специализированных проектов (или специализированных проектов) выбираются языки, предназначенные для этой цели. Например, даже небольшие задачи, которые требуют большого количества вычислений, писать на Python не стоит, так как тот же самый алгоритм, реализованный на Си, может исполняться в 50-100 раз быстрее. Одно дело, если программа на Си исполняется одну секунду, а программа на Python — 50 секунд. Другое дело, если программа на Си исполняется час или сутки. В этом случае реализовывать алгоритм на Python просто бесполезная трата сил и времени. Какие-то языки идеальны для разработки программ одиночками, но имеют массу недостатков, если попытаться с их помощью реализовать совместный проект, другие языки приходится изучать в коллективе, но это позволяет писать программы в миллионы строк.

В любом случае:

> Язык программирования — инструмент для записи алгоритмов неким формализованным образом с целью их исполнения некоторым исполнителем.

> Нет понятия *лучший язык программирования*. Есть понятие: *этот язык программирования хорошо подходит для решения данной задачи.*

4 Понятие о компиляции и интерпретации

Перед тем, как писать программы, стоит понять, как происходит процесс разработки программ.

Итак, в качестве итога нашего непосильного труда мы получили текст программы на каком-либо языке. Что происходит после того, как мы написали программу? Можно ли *исполнить* написанный нами код, не преобразуя его в что-либо другое? Очевидно, что нет. Подробно процесс преобразования удобного для человека представления программы в представление, удобное для исполнителя *компьютер* мы рассмотрим немного позже, а пока нам нужно ввести несколько терминов, которые мы будем использовать в дальнейшем.

Трансляция — процесс перевода из одного представления программы в другое. Это может быть перевод с `Pascal` на `Си` (существуют и такие трансляторы). Это может быть перевод с `Си` в машинный код. Транслировать (переводить) можно несколькими способами.

Первый способ — переводить предложение за предложением, как это делают переводчики, например, на встрече представителей разных стран. Переводчик выслушал предложение и тут же перевёл его на другой язык. В компьютере это смотрится так: мы ввели строчку программы — она перевелась на машинный язык и тут же исполнилась, и так строчка за строчкой. Этот способ называется *интерпретация*.

Второй способ — прочитать весь переводимый текст и только после этого попытаться перевести его как единое целое. Так переводят книги. То же самое и с программой — чтобы исполнить алгоритм, описанный программой нужно перевести всю программу в машинный код и только после этого исполнить. Этот способ называется *компиляцией*. Похоже на то, что интерпретация удобнее в использовании для написания мелких программок, а компиляция позволит добиться существенно лучшего качества большой программы.

Языки `Си` и `Pascal` обычно используют способ компиляции. Программа, переводящая текст на языке программирования в машинные коды, называется *компилятором*. Мы постоянно будем упоминать этот термин. `Python`, наоборот, использует способ интерпретации. Впрочем, про `Python` нельзя сказать, что программа выполняется строчка за строчкой, так делал в своё время `BASIC`. Программа на `Python` переводится в некий другой *промежуточный* код, который затем интерпретируется *исполняющей системой*. В том числе поэтому программы на `Python` исполняются существенно медленнее эквивалентных программ на `Си`.

5 Немного о языке `Си`

В каждом языке имеются как данные, которые он способен обрабатывать, так и действия, которые можно производить над этими данными. Хорошие языки позволяют собирать разнородные данные в единые *объекты* и это улучшает

понимание программ людьми. А почему Си?

- Синтаксис этого языка послужил основой для синтаксиса таких языков, как C++,Java, C#. Изучив Си, достаточно легко переключиться на более новые языки.
- Близок к машине, программы написанный на нём исполняются быстро.
- Очень компактен. Для его изучения не требуется много времени.
- Много готового кода уже написано и он имеется только на Си
- После Си любой язык покажется удобнее...

Можно даже сказать следующее: язык Си настолько близок к современным вычислительным машинам, что в подавляющем большинстве случаев для создания *эффективных* алгоритмов программирования на языке ассемблера не потребуется. Мы не говорит, что язык ассемблера не нужен — для *полного* контроля над компьютером он необходим, но почти всегда его использования можно избежать или ограничить его применение отдельными критическими фрагментами.

5.1 Простая программа на Си

Если вы не писали ни на одном из языков программирования, вас можно только пожалеть. Но не опускайте руки. Всё поправимо.

Чтобы с чего-то начать, давайте посмотрим, как выглядит простейшая программа на языке Си, которая хоть что-то делает. Мы иногда будем нумеровать строки для того, чтобы на них было удобнее ссылаться.

```
01  #include <stdio.h>
02
03  int main() {
04      printf("Hi again\n");
05      return 0;
06  }
```

Первая строка — указание компилятору на то, что требуется включить файл из *стандартной библиотеки* языка Си под именем stdio.h. Забегая вперёд скажем, что в этом файле содержится нечто, которое позволяет нам использовать *библиотечные функции*.

Вторая строчка — пустая, мы просто отделяем один законченный фрагмент программы от другого. Такое разделение важно, если вы собираетесь не только писать одноразовые программы, но и намерены читать их в дальнейшем, или если программа пишется несколькими людьми.

Третья строка — *определение* (*definition*) новой функции, под именем `main`, которая возвращает целое значение (`int`) и которая не имеет аргументов `()`. *Тело функции* заключается в фигурные скобки (всё, что заключено в фигурных скобках называется *блоком*). Закрывающая фигурная скобка находится на 6-й строке и тело функции содержит строки 4 и 5.

Четвёртая строка — вызов функции под именем `printf`, имеющий один *аргумент*, *строчный литерал*[26]. Любое имя в `Си` должно быть перед использованием описано. *Описание* функции `printf` содержится в *заголовочном файле* `stdio.h`. Сама функция `printf` по определённым правилам *форматирует* и выводит некий текст на *стандартный вывод*, в данном случае — экран.

Пятая строка — функция `main` заявляет о завершении своего выполнения, возвращая при этом значение `0` в систему. `0` — признак успешного завершения всей программы.

Шестая строка — закрывающая фигурная скобка *блока*, начатого в 3-й строке.

Как видите, даже самая простейшая программа требует нескольких понятий, в число которых входит *включение файлов* и *функции*. Это — некий шаблон, позволяющий нам писать крошечные законченные программы.

Блок — одна из главных конструкций языка. Все функции по сути есть блоки с аргументами. Внутри блока располагаются *операторы*, которые можно разделить на операторы объявлений и описаний данных и операторы исполнения. Исполнение любой программы начинается с функции `main`. Все операторы внутри `main` исполняются по-очереди. Как только *поток управления* достиг закрывающей фигурной скобки, программа завершается. Поток управления состоит из более мелких единиц, *операторов*.

5.2 Ещё одна простая программа

В фитнес-центре сломались весы и они показывают вес только в фунтах. Поможем несчастным.

```
01  #include <stdio.h>
02
03  int main() {
04      double pounds;
```

[26] Литерал — нечто, представляющее себя самого. Например, `123`, `"Hello"`

```
05      scanf("%lf", &pounds);
06      double kilogramms = pounds * 0.454;
07      printf("Your weight is %.1lf kilogramms\n", kilogramms);
08      return 0;
09  }
```

Четвёртая строка — *оператор декларации*. Перед тем, как производить какие-либо действия над данными, требовалось все данные *определить* или *задекларировать*. Такие объявления можно помещать в любое место программы, где разрешён оператор. Второй оператор декларации — в 6-й строке, где мы объявили ещё одну переменную `kilogramms` и сразу присвоили ей начальное значение, инициализировали. Крайне рекомендуется *инициализировать* все объявленные переменные. Объявление переменной выделяет в *памяти* некий поименованный участок. В данном примере `pounds` и `kilogramms` вещественные значения, то есть, которые могут представлять дроби.

Пятая строка — вызов функции `scanf`, которая умеет вводить данные с клавиатуры и присваивать их значения переменным. Для того, чтобы указать, что функции разрешено присваивать переменной `pounds` — используется значок амперсанда. Если вы его не поставите, программа завершится аварийно. Литерал `"%lf"` показывает функции `scanf`, что требуется ввести число типа `double`. Другой полезный тип — `int`, представляющий целые числа, чтобы его ввести (или вывести) используется `"%d"`;

Седьмая строка — вывод результатов. При выводе `"%.1lf"` будет заменено на представление значения переменной `kilogramms` с одним знаком после десятичной точки (да-да, в `Си` для дробных значений используется отнюдь не запятая).

5.3 Оператор присваивания

Язык `Си` и некоторые его последователи различают операции присваивания и оператор присваивания. Оператор — всегда что-то завершённое, а операция — всегда что-то незавершённое. Знаком завершения в `Си` служит точка с запятой.

Простейший оператор присваивания:

```
a = 5;
```

Математики, не умеющие программировать, это — не уравнение!!! Это запись того, что с этого момента переменная `a` становится равной `5`.

Операция присваивания в Си имеет много вариантов. Почти все арифметические и побитовые операции имеют вариант с одновременным присваиванием.

Мы можем написать

```
abra_shvabra_cadabra = abra_shvabra_cadabra * 3;
```

и это будет корректно. Но как мы произносим эту строчку, когда читаем программу? Так: «умножим abra_shvabra_cadabra на три». Мы опускаем заключение фразы «и присвоим это значение той же самой переменной».

Запись

```
abra_shvabra_cadabra *= 3;
```

более точно отражает алгоритмическую сущность происходящего и, произнося фразу «умножим abra_shvabra_cadabra на три» мы читаем именно то, что видим.

```
a += 4;
b = (c *= 2) + 7;
d <<= 1;     // Это комментарий, a <<= - операция побитового сдвига
e &= 0xFF; // &= - операция побитового И
```

5.4 l-значения

В левой части операции присваивания может находиться только то, что способно «принять в себя» какое-то значение. Мы пока знаем только переменные, но слева может находиться и элемент массива и «именующее выражение» (пока мы не знаем указателей и поэтому пропустим уточнение термина).

Нельзя написать:

```
3 += a;  // Слева - константа
a + 2 = 4; // Значение a+2  существует только в данном выражении
```

так как в левой части операций присваивания находятся не l-значения.

5.5 Оператор `if`

`Си` — *императивный*[27] язык. Чтобы изменить порядок исполнения, нужно проверить условие.

```
if (money >= 75) {
   money -= 75;
   eat_icecream();
}
```

Если у меня достаточно денег (`money >= 75`), то я заплачу деньги за мороженное (`money -= 75`) и скушаю его `eat_icecream();`.

Внутри обязательных круглых скобок за `if` — *логическое выражение*. Сформировать его можно с помощью знаков сравнения (`>,<,>=,<=,==,!=`). Обратите внимание на знаки сравнения «равенство» (`==`) и «неравенство» `!=`. Имеются также такие знаки, как логическое И (`&&`), ИЛИ (`||`) и НЕ (`!`).

Мы используем *отступы* для того, чтобы показать тому, кто читает программу, что данный блок или единичный оператор исполняется только при соблюдении определённых условий. Сам язык `Си` не заставляет этого делать, в отличие от языка `Python`, но читать и понимать программу становится намного проще, если будет соблюдена *дисциплина программирования*, в данном случае заключающаяся в соблюдении некоторых правил записи программы.

Вторая форма оператора `if` содержит *альтернативную ветку*, обозначенную ключевым словом `else`

```
if (money >= 75 && money < 1000) {
    money -= 75;
    eat_icecream();
} else if (money >= 1000) {
    money -= 1000;
    see_the_movie();
} else {
    go_sweem();
}
```

[27] Императивные языки программирования основаны на приказах исполнителю выполнить ту или иную команду, возможно, в зависимости от каких-либо условий, то есть описание, *как* решать задачу (`Pascal`, `Fortran`, `C++`, `Java`, `Python`, ...). Декларативные языки программирования описывают `что` надо решать, оставляя на усмотрение компилятора средства решения (`Lisp`, `Haskell`, `erlang`, ...)

Если есть деньги на мороженное и нет денег на кино, то ем мороженное, если есть деньги на кино, то иду в кино, иначе иду купаться. Все три случая взаимно исключающие.

5.6 Оператор `while`

`while` — первый пример того, как можно создавать программы, выполняющие много действий с помощью небольшого количества строк.

Этот цикл напоминает «обыкновенный» оператор `if`, без части `else`. Единственное отличие заключается в том, что всё продолжается, пока условие остаётся истинным. Условие истинно — мы заходим внутрь цикла, исполняем все его операторы и, как только мы исполнили последний оператор внутри блока, бежим проверять условие снова. Как только это условие стало ложным, мы в блок с телом не заходим. Вот такой непрекращающийся `if`.

Задача IV.1. Вход алгоритма — целое число `n` до 10^9. Выход — наибольшая число `m` такое, что $3^m \leqslant n$.

Решение. Идея решения: вычислять очередную степень тройки до тех пор, пока она не станет больше `m`, после чего вернуться на единицу назад.

Вторая идея: степень тройки 3^x вычислять можно по индукции, имея вычисленное 3^{x-1}.

```
int pow3 = 1, x = 0, result = 0;
while (pow3 < n) {
    result = x;
    pow3 *= 3;
    x++;
}
```

□

При реализации любого цикла убедитесь, что в каждой *итерации* имеется некий прогресс, продвижение переменных к удовлетворению условия. Бесконечный цикл — характерная ошибка, не так редко встречающаяся.

Задача: Вычислить число e^x с точностью до 6-го знака после запятой для $0 \leqslant x \leqslant 10$ по формуле разложения функции e^x в ряд (его вы пройдёте на математическом анализе, пока будем просто верить, что данный ряд верен):

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Идея решения: будем рекуррентно вычислять очередной член ряда до тех пор, пока он не станет меньше 10^{-7} (это математически нестрого, но достаточно для решения данной задачи).

```
double sum = 1;
int n = 1;
double el = 1;
while ( (el *= x / n) > 1e-7) {
   sum += el;
   n++;
}
```

Мы воспользовались здесь тем, что `Си` — язык, основанный на выражениях. Значением выражения `el *= x / n` будет `el`, которое в этом же выражении сравнивается с константой 10^{-7}. Этот `el` и есть наш очередной элемент.

5.7 Оператор `for`

Это — самый мощный оператор цикла. Имеющийся в `Pascal` цикл `for` обладает лишь малой частью функционала цикла `for` языка `Си`.

Задача IV.2. Найти сумму кубов всех чисел от 1 до заданного n.

Решение. Математики, не возмущайтесь, не все программисты знают формулу суммирования кубов натуральных чисел. Дадим им вычислить сумму прямолинейным образом.

```
// Вход алгоритма: n
// Выход алгоритма: sum
int sum = 0, i;
for (i = 0; i < n; i++) {
   sum += i*i*i;
}
```

□

Обратим внимание на то, что в скобках, идущих после ключевого слова `for`, имеются *две* точки с запятой. Это — необходимое условие. Эти точки с запятой делят пространство внутри скобок на *три* выражения.

Первое выражение исполняется однократно, как только начинается цикл.

Второе выражение — условие продолжение цикла. Это — логическое выражение, которое проверяется каждый раз, когда *итерация* начинается. В *Си* на этом месте может находиться любое выражение, которое будет считаться истинным, если оно отлично от нуля (вспомним, что в `Си` нет логического типа данных).

Третье выражение исполняется `после того`, как будет исполнено *тело* цикла. В данном случае тело — `sum += i*i*i;` — набор операторов в фигурных скобках, блок. Вместо блока можно разместить ровно один какой-либо оператор, но мы уже знаем, что для того, чтобы избежать плохо отлавливаемых ошибок, лучше оформлять блок, то есть ставить фигурные скобки, даже для единичного оператора.

Применённый нами оператор `for` мало отличается от того, что мы видели в `Pascal`. Сила Сишного оператора `for` в том, что первое, второе и третье выражения независимы друг от друга и мы можем в них производить произвольные действия.

Вспомним задачу по нахождению степени тройки, не превосходящей заданного числа.

```
int pow3, x, result = 0;
for( pow3 = 1, x = 0; pow3 < n; pow3 *= 3, x++) {
   result = x;
}
```

Используя `for` можно уменьшить вероятность допустить ошибку, поместив изменение переменной, за значением которой мы следим `pow3`, в заголовок цикла. Таким образом мы явно показываем её продвижение к цели (она увеличивается и когда-нибудь достигнет `n`). Программу легче и писать, и читать.

И первое, и второе, и третье выражения можно опустить. Если опустить только первое и третье выражения, то то, что получилось, станет полным эквивалентом `while`. Если опустить второе выражение, то оно будет считаться истинным. `for(;;) {}` — бесконечный цикл. Но как же выйти из бесконечного цикла? Об этом — следующий раздел.

Если внутри любого цикла мы напишем оператор `break;`, то управление будет передано на оператор, непосредственно находящийся ниже оператора цикла.

Например, только что разобранный пример может быть записан и таким образом:

```
int pow3, x, result = 0;
for( pow3 = 1, x = 0; ; pow3 *= 3, x++) {
    if (pow3 < n) {
        break;
    }
    result = x;
}
// Как только сработает break, управление перейдёт сюда.
```

В этом примере второго выражения в заголовке оператора `for` нет, и это означает, что если бы мы не поставили где-то `break` (или `return`, если нужно покинуть функцию, а не цикл), то цикл никогда бы не закончился.

5.8 Функции

Функция — фундаментальное понятие всех современных языков программирования. В `Си` функция — основной строительный блок, позволяющий упорядочить структуру программы и уменьшить её сложность. Функции мы применяем в двух основных случаях: если требуется логически выделить фрагмент кода и если какие-то фрагменты кода повторяются.

Перед использованием функции её нужно где-либо *определить*.

Самым простым способом определения функции является помещение её *тела* в *исходном файле* программы где-нибудь *перед* её использованием. Функции расширяют возможности языка, добавлением новых *абстракций*.

Давайте рассмотрим простейший пример. В `Си`, например, нет операции возведения в квадрат, которая есть, например, в `Pascal`. Но её очень легко дописать.

```
double sqr(double x) {
    double result = x * x;
    return result;
}
```

Использовать её можно, например, присвоив какой-либо переменной результат её *вызова*:

```
double t = 16;
double t2 = sqr(t); // теперь t2 = 256
```

Ключевое слово `return` *возвращает результат*. Говорят, что *функция вернула значение*.

Давайте на этом закончим знакомство с `Си`. Мы не затронули массивы, структуры, такое уникальное средство, как *указатели* и ещё довольно много конструкций языка. В стороне осталось знакомство с системной библиотекой (в ней много тысяч разных функций). Впрочем, это вам ещё предстоит...

6 Немного об объектно-ориентированном программировании, ООП

Мы уже упомянули, что сложные программы лучше писать на одних языках, а простые — на других. А как понять, сложная программа или нет? Что является мерой сложности программы?

Оказывается, что сложность программы можно определить как комбинаторную функцию общего количества различных комбинаций всевозможных связей между объектами программы. Например, пусть у нас имеется программа, в которой 10 функций. Если кто-то описал переменную вне функций (в `Си` так делать можно и такая переменная будет называться *глобальной*), то теперь появилось 10 новых связей. Другой программист хочет установить, для чего используется эта переменная. Ему потребуется просмотреть код всех 10 функций. При добавлении кем-то новой функции нам также понадобится просмотреть все остальные функции для того, чтобы определить, вызывает ли кто её. Количество связей при таком подходе растёт квадратичным образом и добавлять новую функцию и переменную становится всё сложнее. Если определять сложность как количество возможных связей, то сложность будет пропорциональна N^2, где N — количество блоков в программе. С целью уменьшения сложности следует уменьшить либо само количество блоков, либо количество связей между ними, путём запрета определённых связей. Давайте посмотрим, как исторически происходила эта борьба.

Программирование в машинных кодах

Программирование в машинных кодах вынуждает программиста использовать адреса переменных в качестве строительных блоков данных и машинные команды в качестве строительных блоков кода. Машинные команды переда-

чи управления могут ссылаться на любой машинный адрес (в том числе и невозможный). Для программирования требуется скурпулёзно вести таблицу соответствия желаемых переменных с их адресами. Метрика сложности программы получается очень высокой — каждый блок данных требуется выбирать из большого количества возможных адресов (высока энтропия системы) и в каждой команде перехода адрес назначения также можно выбирать из такого же множества возможных машинных адресов, сами машинные команды тоже выбираются из пространства чисел.

Программирование на языке ассемблера

Сложность программы немного уменьшается за счёт то, что введение языка ассемблера позволяет немного уменьшить энтропию системы:

1. коды команд теперь выбираются из осмысленного множества сокращений, например, операция сложения, которая в машинных кодах кодировалась числом `107` (например), теперь записывается как `ADD`.
2. блоки данных выбираются из множества идентификаторов, придуманных программистом.
3. переходы на блоки кода теперь возможны не в любую точку программы, а только на метки, назначенные программистом.

Первые языки высокого уровня

Первые языки программирования служили целью упростить, в первую очередь, написание кода программы и они были во многом, по сути, тем же языком ассемблера. Типы данных и операции позволяли их точное отображение на машинный язык. Полезным стало введение библиотек — процедур и функций, которые уже кто-то реализовал до нас и включил в поставку. Программист, однако, мог допустить ошибку в имени переменной, которая осталась бы незамеченной.

Структурное программирование

В структурном программировании весь поток управления программы делится на так называемые управляющие структуры — *если*, *альтернативное если*, несколько видов *циклов*.

Основной целью структурного программирования была попытка уменьшить сложность написания, и, что главное — верификации программного кода.

Все управляющие структуры имели ровно один вход и ровно один выход. Гарантировалось, что если управление попадает на вход управляющей структуры, то оно (при правильном использовании управляющих данных) обязательно дойдёт до выхода из управляющий структуры. В классическом варианте были категорически запрещены операторы перехода изнутри управляющей структуры наружу и снаружи внутрь. Это реально уменьшило эффективную сложность программ, и, как ни странно, вначале усложнило жизнь программистам, приучившихся к анархическому программированию по принципу «всё дозволено».

Впрочем, структурное программирование почти не повлияло на сложность программ, связанную с блоками данных — оно затронуло только блоки кода и переходы между ними.

Процедурное программирование

Процедурное программирование расширяет структурное. Задача разбивается на процедуры, связь между которыми осуществляется либо через параметры процедур (что предпочтительнее), либо через общие переменные (что мы, как уже видели, приводит к более сложным программам). Процедурное программирование позволило немного уменьшить сложность, связанную с блоками данных — благодаря введению понятия *локальные для процедуры переменные*. Это позволило уменьшить количество общих переменных, что, как мы уже видели, благотворно влияет на уменьшение сложности программы.

Процедурное программирование привело к созданию понятия *библиотека функций* — например, графическая библиотека, или библиотека математических функций или библиотека линейной алгебры. Это, наконец-то, позволило набрать критическую массу удобных и полезных алгоритмов. Появилась возможность выбирать, каким образом проектировать большую программу — методом сверху вниз или методом снизу вверх.

Метод сверху вниз разбивает большую проблему на ряд более мелких, которые

- можно реализовывать разным программистам;
- проще в реализации.

Каждая из подпроблем в свою очередь разбивается на подподпроблемы и так далее. Все под(под)проблемы делятся на уже реализованные и ещё не реализованные. Для тестирования нереализованные подпроблемы заменяют *заглушками*, которые просто возвращают управление назад (возможно, что-то передавая для корректной работы вызывающей части). По мере готовности заглушки заменяют на реально решённые подпроблемы.

Метод снизу вверх сначала опирается на создание необходимых библиотек. Эти библиотеки отлаживаются на тестовых программах, затем, после их готовности, на них реализуется более крупная подзадача.

Практика показала, что оба способа годятся, хотя каждый из них имеет и свои недостатки. Например, при проектировании снизу вверх часто оказывается, что большая часть процедур и функций, которые имеются в написанных для задачи библиотеках, никогда не используются — они просто не потребовались (при проектировании сверху вниз о них бы и не вспомнили). Зато следующий проект на существующих библиотеках (отлаженных библиотеках!) уже создавать гораздо легче.

Здесь у нас впервые появляется понимание того, что большую программу пишет не один человек, и что, например, для библиотеки её автором может быть один, а использовать эту библиотеку может совершенно другой человек. Важным становится их взаимодействие, прямое — через личное общение, или косвенное — через документацию к библиотеке (кстати, такую документацию может писать ещё кто-то другой — библиотекарь, технический писатель). Мы в дальнейшем будем разделять тех, кто пишет функцию (библиотеку, модуль, класс, ...) — *авторов* и тех, кто ими пользуется — *пользователей*.

Модульное программирование

Развитие процедурного программирования, в котором вводится понятие *модуль*. Так же, как в процедурном программировании масса переменных покинула глобальную область видимости и осела внутри процедур (и функций), в модульном программировании стало возможным скрыть наличие ряда функций (и, наконец-то, данных) и собрать их использование внутри модуля. Появилось разделение на *интерфейс* (*interface*) модуля и его *реализацию* (*implementation*). В интерфейсе модуля описывалось, какие процедуры, функции и переменные может использовать субъект, которого мы назвали *пользователем*. Ему не разрешено менять детали реализации (возможно, ему даже не предоставят возможность узнать алгоритмы реализации), но тем, что описано в интерфейсе модуля, он пользоваться может.

По сути дела, все современные, не объектно-ориентированные языки давно предоставляют своим пользователям именно модульный подход. Если мы вспомним `Pascal` и `Delphi`, то конструкция

```
uses crt;
```

служит именно для подключения модуля с названием `crt` к нашему коду. После этой строки программе становятся доступны новые имена процедур, например `clrscr`.

В классическом Си включение

```
#include <stdio.h>
```

тоже можно рассматривать (с некоторым, но достаточно хорошим приближением) как включение модуля стандартного ввода-вывода. В самом деле, в *пространство имён* добавляются новые типы данных (такие, как `FILE`), константы (`NULL`, `EOF`), функции (`printf`, `scanf`, `fopen`) и т. д.

Вообще-то надо отметить, что управление *пространством имён* становится очень важным именно для больших программ и именно для программ, которые разрабатываются не одним программистом, а группой. Это позволяет каждому из участников проекта не заботиться о точном знании всех имён, которые могут использоваться во всём проекте остальными участниками. В процедурном программировании это не так. Если один из участников проекта, использующего процедурное программирование, написал функцию с именем `max`, принимающую массив целых чисел, то любой другой участник этого же проекта не может создать функцию с этим же именем, принимающую другие аргументы. В модульном программировании это оказывается возможным, если эти функции принадлежат разным модулям.

Объектно-ориентированное программирование

ООП есть дальнейшее развитие модульного программирования и в него вводится понятие *объект*. Если модуль, как мы уже знаем, есть совокупность процедур и данных, выполняющих части одной задачи, то что есть объект? Может ли объект представиться модулем? Да, может, модульное программирование есть чистое подмножество объектно-ориентированного. А может ли модуль представиться объектом? В общем случае — нет, не может. Объектно-ориентированное программирование — очередная итерация, попытка отобразить предметный мир на язык математических моделей. Это не панацея, существует множество проблем, для решения которых можно и нужно применять другой подход, например, функциональное программирование — не путать с процедурным программированием (это всё равно, что перепутать программирование как прикладную науку и раздел математики, называющийся *математическое программирование*).

7 Что есть ООП и почему мы о нём говорим

Итак, основным понятием ООП является *объект*, а не *модуль* или *процедура*. Основная программа в этом случае решает проблему — как разделить модель

на объекты, что будет представлять из себя каждый объект и какую часть общей задачи он готов на себя принять.

Для примера давайте возьмём игру в шахматы. Какие объекты мы можем увидеть при попытке смоделировать шахматную игру?

Объекты. Методы и свойства

- *Шахматная доска.* Она имеет ряд *свойств*, таких, как количество горизонталей `lines` и количество вертикалей `columns`. В классических шахматах и то, и другое числа равны 8. Однако, в игре в международные шашки, проводящейся на похожей доске, количество клеток уже 100. Не исключено, что разработчик программы для игры в шахматы захочет в будущем попробовать силы в программировании международных шашек, поэтому в разных проектах эти свойства доски могут быть разными.

- Шахматная доска состоит из *полей*, каждое из которых имеет свой *цвет* и на которых могут размещаться

- шахматные фигуры. Их несколько видов и каждый из этих видов характеризуется набором возможных на пустой доске ходов. Более того, набор возможных ходов изменяется в зависимости от цвета фигуры — например, белые пешки могут двигаться только вверх, чёрные — только вниз. Расположение фигур на шахматной доске называется

- *позицией.* Являются ли одинаковыми позиции с одинаковым расположением на ней фигур? Как ни странно, могут не являться. Например, право на рокировку теряет та фигура, которая уже делала ход. Если в одной позиции белый король стоит на поле `e1`, ладья на `h1` и эти фигуры до этого ещё не ходили, то при отсутствии других ограничений (фигур между ними, король находится под шахом или проходит битое поле, совершая рокировку) среди возможных ходов в этой позиции будет присутствовать короткая рокировка. Если же в той же визуально позиции либо король, либо ладья уже совершали ход и вернулись на место, рокировки среди возможных ходов не будет. Можно сказать, что *позиция* имеет следующие *свойства* — «белый король совершал ход», «белая ладья h1 совершала ход» и так далее и что эти свойства, в отличие от количества горизонталей и вертикалей могут изменяться в зависимости от того, как эта позиция получилась из игры.

  Позиция имеет множество свойств — таких, как *очередь хода* (ход белых или ход чёрных), *номер хода в партии.* Каждое из свойств меняется не само по себе — нельзя, например, назначить позиции из играющейся

партии ход белых или принудительно назначить следующий ход сороковым — эти свойства меняются только после совершения каких-либо *действий*. Только *действия* могут изменить состояние шахматной позиции и действие это (в шахматах) называется совершением хода. Однако, совершаемый ход должен удовлетворять правилам шахматной игры, такое действие, как совершение хода, зависит от текущей позиции, в которой ход совершается. После совершения хода состояние *объекта* шахматная доска меняется — меняется ряд *свойств*, таких, как наличие права рокировки, очередь хода, расположение фигур. Такие действия, направленные на изменение состояния объекта мы будем называть *методами*.

Три кита ООП: инкапсуляция

Продолжая рассмотрение нашей шахматной программы, представим, что мы — разработчики *ядра* программы, то есть, той компоненты, которая отвечает за саму игру. Другая группа отвечает за внешний вид программы — как именно выглядит доска и фигуры, как программа взаимодействует с пользователем. Наша, разработчиков ядра, цель — обеспечить *визуальщикам* (тем, кто отвечает за программирование взаимодействия с пользователем, `GUI` — `Graphical User Interface`) все возможности для работы. Например, мы должны быть способны ответить на вопросы:

- корректна ли расставленная пользователем позиция?
- корректен ли сделанный пользователем ход?
- какой ход программа считает наилучшим? И т. п.

Чтобы ответить на эти вопросы наше ядро должно предоставить визуальщикам набор чего-то, только чего? Функций? Данных? Давайте предоставим им набор *объектов*, каждый из которых будет обладать своими *свойствами* и *методами*. Давайте учтём, что визуальщики правил игры не знают (точнее сказать, они имеют право их не знать), да и квалификация у них, как у программистов, скажем, так, не очень. Не забываем законы Мёрфи — если систему можно использовать неверным образом, именно так она и будет использована. Поэтому нашими целями будут

1. предоставить набор, обладающий всеми необходимыми функциями
2. не разрешать пользоваться этими функциями неправильно

.

Для примера возьмём поле доски. Мы, скорее всего, должны предоставить визуальщикам возможность работы с полями (например, если пользователь взялся за коня, то они могут подсветить доступные этому коню ходы). Мы, ядерщики, проектируем работу с объектами-полями. У нас возникает вопрос: каким образом представлять координаты поля? В виде строки (`"D5"`)? Это будет неудобно при вычислении возможных ходов фигуры — придётся каждый раз преобразовывать строку в координаты и наоборот. В виде пар координат? Это будет неудобно при печати хода и, возможно, неудобно при вычислении ходов. Просто числом, номером поля, чтобы было удобно составлять таблицы перемещений? Это удобно для вычисления, но неудобно ни для печати хода, ни для его визуализации.

Так что же делать?

Выход здесь прост: давайте скроем от пользователя детали реализации кода, но предоставим ему различные способы получения. Что там будет внутри пусть пользователя нашего объекта никак не касается. Мы скрываем представление, *инкапсулируем* его. Для получения информации мы будем использовать соответствующие *методы*, например, представить поле в текстовом виде или представить поле в виде пары координат. Если пользователю объектов не предоставили знания о внутреннем содержимом объекта, то единственный способ работать с этим объектом — использовать *методы*. Если разработчик объекта решит изменить внутреннее представление, не затронув методов работы с ним, пользователь объекта даже знать не будет о произошедших изменениях. Хорошо это? Отлично! Разработчик теперь не привязан к первоначальному проекту и может изменять реализацию, как удобно ЕМУ, а не пользователю, пользователя это не затронет совсем.

Резюмируя — инкапсуляция нужна для сокрытия необязательных деталей реализации объекта. Меньше знаешь — крепче спишь.

Три кита ООП: полиморфизм

Итак, у нас имеется несколько типов фигур. Вероятно, понадобится метод (да, мы теперь так будем говорить) узнать, какие ходы возможны конкретной фигурой с конкретного поля. Предположим, что этот метод мы захотели назвать `get_available_moves`. В классическом языке программирования это может быть функцией или процедурой. Однако возникает вопрос? Какие аргументы передавать в эту процедуру/функцию? Цвет фигуры, это понятно. А что насчёт фигуры? Если мы передадим туда код фигуры (например, 1 — король, 2 — ферзь и так далее), то мы должны создать функцию, которая разбирается в ходах всех фигур — и короля, и ферзя... Хорошо бы было, если бы можно было передать объект с самой фигурой, но тогда потребуется 6

функций с одним именем и с разными аргументами, но и Си, и Паскаль и тем более Фортран запрещают нам создавать несколько функций с одним именем! Как выход из положения в этих языках к имени функции добавляют тип аргументов (например, `knight_get_available_moves`), но это уже довольно громоздко и немного неуклюже. Ну что же, *полиморфизм*, второй из китов ООП, позволяет нам создавать различные функции с одним и тем же именем.

Три кита ООП: наследование

Последний кит — *наследование*. В нашей шахматной программе имеется несколько различных видов фигур и было бы неплохо иметь объекты, которые моделируют каждый тип фигуры — например, модель ферзя могла бы определять возможные ходы с нужного поля. Если мы заведём каждой фигуре объект своего типа возникнет вопрос: как же теперь представлять позицию без излишнего усложнения логики программы? На помощь приходит *наследование*. Позиция может представляться, например, массивом *фигур*, причём король, ферзь, слон, конь, ладья и пешка будут именно *фигурами*, они *унаследуют* от *базового объекта* такие свойства, как цвет, местоположение, а часть *методов* для них будет разной по действиям, но одинаковой по имени. Например, если все фигуры будут иметь метод `getAvailableMoves`, то для того, чтобы получить полный список всех возможных в позиции ходов будет достаточно вызвать этот метод для всех фигур с нужным цветом!

8 Краткая шпаргалка по `printf/scanf`

Функция `printf`

Эта функция умеет достаточно многое и достигается это новыми для времени создания языка `Си` средствами. Предположим, что вы пишете программу на `Pascal` и вам нужно вывести 5 переменных, `a,b,c,d,e`. Это можно сделать так:

```
writeln('a=', a, ' b=', b, ' c=', c, ' d=', d, ' e=', e);
```

Не знаю, как это смотрится для вас, но многие такой стиль не любят, так как трудно сразу заметить, какой вид будет иметь выходная строка. В `Си` подход совсем другой:

```
printf("a=%d b=%d c=%d d=%d e=%d\n", a, b, c, d, e);
```

По первому аргументу (строке) мы видим общий вид вывода: вместо `%d` будут подставлены десятичные значения соответствующих переменных.

Первым аргументом в `printf` идёт *форматная строка* в которой имеется выводимы текст (`"a="`)и, возможно, несколько *шаблонов* или *спецификаций формата* (`"%d"`). `printf` следует по строке слева направо, выводя символ за символом то, что в этой строке находится. Как только он замечает *метасимвол* `%`, он пока перестаёт выводить текст и начинает собирать *шаблон*. Шаблон заканчивается одной из предопределённых букв, например, буква `d` означает, что вывод должен производиться в десятичной (`decimal`) системе счисления. Перед буквой, определяющей формат вывода и тип посланного в `printf` значения может тоже что-то находиться. Но это давайте посмотрим на примерах, описать **все** возможность `printf` всё равно не выйдет.

Хотя знак процента является метасимволом, напечатать его тоже можно. Правда, не пройдёт фокус `printf("%");`, хороший компилятор вас предупредит об ошибке, но вот `printf("%%");` уже сделает то, что мы просили — выведет одиночный знак процента.

```
int  i = 123;
char c = 'a';
unsigned u = 256;
unsigned long ul = 4095ul;  // Помните про суффиксы?
long long ll = 65535ll; //
unsigned long long ull = 1024ull;
float f = 123.456;
double d = 12345678.9012345;
// Вывод будем писать в кавычках для того, чтобы видеть и пробелы
printf("i=%d i=%4d i=%04d i=%-4d", i); // "i=123 i= 123 i=0123 i=123 "
printf("c=%c c=%d", c); // "c=a c=66"

printf("u=%u u=%o u=%x u=%X", u, u, u, u);
// "u=256 u=400 u=ff u=FF"

printf("ul=%ul ull=%ull", ul, ull); // "ul=65535 ull=1024"

printf("f=%f f=%g f=%e", f, f, f);
// "f=123.456 f=1.23456e5 f=123.456"
printf("d=%lf d=%.1lf d=%.7lf d=%10.3lf", d, d, d, d);
//"d=12345678.901235 d=12345678.9 d=12345678.9012345 d=12345678.901"
printf("Result=%.2lf%%", result);
// "Result=10.75%"
```

Функция `scanf`

Для ввода можно использовать функцию `scanf`.

Она создавалась в пару к функции `printf` и весьма на неё похожа.

Первым аргументом у неё выступает форматная строка — то, что функция ожидает на вводе. В ней присутствуют такие же знаки процента, извещающие `scanf`, что ей потребуется ввести число в каком-то формате. Сами форматы в основном совпадают с теми, которые используются в `printf`.

> Функция `scanf` требует, чтобы в неё передавались `l-значения`, то есть, адреса, по которым можно присвоить введённое значение.

Несколько примеров:

```
int  i;
char c;
unsigned u;
unsigned long ul;
long long ll;
unsigned long long ull;
float f;
double d;
int code = scanf("%d", &i);
code = scanf("%c %u %lu %llu %f %lf", &c, &u, &lu, &llu, &f, &d);
```

Функция `scanf` возвращает число тех адресов, по которым ей удалось положить значение. А почему она могла не сделать какой-то работы? Например, потому, что мы просили число, а на входе оказалось нечто нечисловое. Может быть, закончился входной файл (в том числе и стандартный ввод).

Пусть на входе имеется строка вида `17/12/2017`. Тогда ввести её можно так:

```
int day, month, year;
code = scanf("%d/%d/%d", &day, &month, &year);
```

Если при этом code окажется равным трём, то всё ввелось успешно.

Функции `printf` и `scanf` настолько сложны и многогранны, что на их полное описание понадобились бы не один десяток страниц. Можно сказать, что они формируют небольшой язык описания форматов. Поэтому мы с ними расстаёмся, но не навсегда, на семинарах и в контрольных работах вы их будете много раз использовать.

Глава V
Различные предметы

1 Английский язык

НЕ ПРОГУЛИВАЙТЕ! И делайте домашние задания. Остальное вам должны были уже рассказать кураторы.

2 Физкультура

Привезите «хорошую» медицинскую справку из дома. Хорошую — это 086-у и где будет указана категория годности.

НЕ ПРОГУЛИВАЙТЕ! Больше нам добавить нечего ;)

3 LaTeX

Такого предмета нет, но особо продвинутые преподаватели (как правило, с кафедры моу) требуют или предлагают большие бонусы за оформление домашней работы в LaTeX. Стоит ли говорить, что никакого подробного введения в LaTeX не предполагается. Изучение LaTeX на физтехе подобно изучению БД — за тем исключением, что никто не проверяет исходных файлов, а времени на изучения LaTeX не предусмотрено. Однако получить базовые навыки вёрстки в LaTeX можно достаточно быстро.

Прежде всего дистрибутив LaTeX необходимо установить. Для пользователей `Windows` мы можем порекомендовать сначала установить `miktex`, а затем, если вам не понравится штатный редактор `Texworks`, который идёт с ним в комплекте (а он обычно никому не нравится), установить другой редактор, например, `TeXstudio`[28]. В `Linux` различных версий LaTeX обычно доставляется менеджерами пакетов, каким образом — зависит от дистрибутива. Впрочем, если вы используете `Linux`, вы сами этого хотели. На компьютерах под управлением `MacOS` установка тоже не сложная. Сначала скачиваете дистрибутив `TexLive` в версии для `Mac` с сайта Tex Users Group, устанавливаете полученный дистрибутив, затем скачиваете `TeXstudio` (соответствующую версию, ко-

[28] Если вы сначала установите `TeXstudio`, а затем `miktex`, то компилятор, который используется `TeXstudio`, придётся настроить вручную.

нечно) и на этом настройка заканчивается. Кстати, `TeXstudio` имеется и под `Linux`.

Теперь стоит изучить **примеры вёрстки**. Для этого достаточно разобрать хорошие исходники (например, тот, что приложен к этой книге) и писать свои домашки на их основе. Если же возникают вопросы, можно отправиться на форум tex.stackexchange.com или же забить свой вопрос в поисковой строке — в топе гугла по запросу обязательно будут несколько страниц с этого сайта, из которых можно выбрать подходящую. Если не удастся найти интересующую страницу, переформулируйте свой запрос и продолжите искать. Если найти не удаётся, а часы пробили второго ночи, — что ж, можно попытаться придумать решение самому. В конце концов латехом люди пользуются очень часто, поэтому проблема, которая у вас возникнет, скорее всего была разрешена гораздо раньше, а соответствующие исходники валяются в сети.

Конечно, для вёрстки понадобятся дополнительные знания. Мы можем посоветовать учебник по LaTeX на wikibooks — он лаконичен, покрывает практически все темы и изобилует примерами, которые можно сразу же скопировать в исходник. Разумеется, разговор идёт об англоязычной версии, — русская версия очень сырая, как и почти все википроекты на русском языке. Также мы советуем сборник примеров Воронцова.

Рисунки в можно не только импортировать из других файлов, но и рисовать в самом LaTeX— в помощь вам библиотека `TikZ`, например. Документация насчитывает почти 900 страниц, поэтому в поисках конкретных примеров можно отправиться на сайт texample.net. К каждому примеру прилагается исходник.

Учебник Львовского, русскоязычную библию по LaTeX, мы можем порекомендовать лишь тем, у кого достаточно времени — он хороший, подробный и изобилует многими полезными знаниями, необходимыми для вёрстки (такими, например, как то, что для вёрстки на русском языке надо использовать французские кавычки, а не английские), но его как настоящий учебник надо спокойно и обстоятельно читать.

4 Физика (Лабораторный практикум)

4.1 Принципы подготовки к лабораторным

Наверное, каждого из вас, ещё когда вы были абитуриентами, пугали таинственными «лабами» (лабораторными работами по физике), которые придётся делать в течение 5 семестров? Кратко расскажем про то, чего от них ждать, ведь не все выполняли эксперименты на уроках физики в школе.

Собственно, все требования к выполнению и оформлению работ прописаны в начале выданного в библиотеке «Лабораторного практикума...». Причины возникновения проблем:

1. Несоблюдение данных рекомендаций студентами.
2. Выдвижение преподавателями требований, не соответствующих общепринятым.

В общем, всё как в отечественной гражданской авиации. К сожалению, на кафедре общей физики CRM (управление ресурсами экипажа) сведено к двум советским правилам, которые должен знать и соблюдать каждый студент:

1. Преподаватель всегда прав.
2. Если преподаватель неправ, смотри правило №1.

Говорят, у многих вторых пилотов есть блокнотик под названием «Индивидуальные особенности командиров моей авиакомпании». В нашем случае роль этого блокнотика выполняет сайт wikimipt.org, на котором всегда можно почитать про «индивидуальные особенности» своего преподавателя (кстати, это относится не только к лабораторным работам!).

Немного общих рекомендаций.

0. Нужно иметь железные нервы.
1. К первой работе готовиться так, как будто Ваш преподаватель — самый строгий на кафедре (конкретнее напишем ниже).
2. На первом же занятии попросить преподавателя сформулировать все требования. Вероятно, они будут отличаться от описанных в Сборнике.
3. Строго выполнять указания преподавателя.
4. Изучать теорию. Ребята, «переписывание лабника» и рисование таблиц — это ещё не вся подготовка!

Как оценивается работа в лаборатории? В старые времена было две оценки: первая за подготовку и выполнение, вторая — за качество отчёта и ответы на вопросы при сдаче. Теперь произошли небольшие изменения: согласно опубликованным правилам, первая оценка ставится за отчёт, а вторая — за знание теории. Подготовка и выполнение формально не учитываются, но это зависит от преподавателя (иные и минус-баллы ставят). В худшем случае можно оказаться не допущенным к работе, что приведёт к отставанию от графика (а это, если нет уважительной причины, почти конец света: из всех знакомых одного из соавторов, столкнувшихся с такой проблемой, получить зачёт смог только один — 25 января, в последний день, когда это было разрешено. Он всё равно не закрыл ту сессию, но это отдельная история...).

Итак, процедура подготовки к работе:

1. Написать (строго по центру!): «Лабораторная работа №...», на полях — дату выполнения. На следующей строке — название работы. Кажется, мелочи. Но если на них забить, можно и плохую оценку получить.

2. Переписать из лабника пункты «цель работы» и «в работе используются». Нарисовать (или распечатать и вклеить) схему установки со всеми обозначениями.

3. Крайне желательно включить краткий (или не очень) конспект необходимой теории: в том или ином виде его требуют все.

4. Далее следует большой раздел «выполнение работы». ПРЕДУПРЕЖДЕНИЕ: встречаются лабы, которые выполняются по инструкции, выдаваемой в кабинете, а не по книге. Поэтому к подготовке советуем приступать не в ночь перед выполнением, а раньше, чтобы использовать свежую версию.

5. В этом разделе нужно написать по пунктам, что мы делаем в работе, подготовить таблицы для записи результатов измерений. Желательно оставить место, чтобы был запас.

6. После того, как тетрадь (строго формата А4) готова, нужно изучить теорию. Необходимо знать (как к экзамену!) всё, что прямо относится к лабе; близкие темы тоже не должны быть тёмным лесом. Рекомендуется использовать не только лабник, но и Сивухина с Кириченко. Причём это относится и к выполнению, и к сдаче работы!

Итак, пусть мы написали всё необходимое, провели измерения, ответили на каверзные вопросы преподавателя и получили «плюсик» в журнал и подпись лабника в тетрадь (без неё считается, что лаба не сделана, со всеми вытекающими последствиями). Надо обработать результаты. Не будем писать, как это

делается — всё есть в лабнике (это слово, не признаваемое словарём, имеет два значения — Лабораторный практикум, в котором собраны рекомендации и описания работ, и преподаватель, ведущий лабы). Отметим лишь, что обязательными частями являются расчёт погрешностей и вывод. Причём вывод должен содержать как результат, так и сравнение его с теоретическими значениями + объяснение несовпадения, если оно случилось.

4.2 Расчёт погрешностей

Поговорим о расчёте погрешностей.

Согласно определению, абсолютная погрешность величины x (обозначается σ_x или Δx) — это отклонение результата измерения от истинного значения: $\sigma_x = |x_{\text{ист}} - x_{\text{изм}}|$.

Относительная же погрешность — отношение абсолютной погрешности к истинному значению. Т. к. вычислять погрешность точно даже звучит бредово (хотя мы видели человека, который попытался. Преподаватель полпары над ним смеялся), а истинное значение мы всё равно никогда не узнаем, оно заменяется на измеренное:
$\varepsilon_x \approx \dfrac{\sigma_x}{x_{\text{изм}}}$.
Это основные определения. В наших лабах каждая величина получается одним из способов:

- как среднее значение серии измерений;
- как коэффициент (или свободный член) линейной зависимости;
- вычисляется из других величин по какой-то формуле.

Никаких строгих выводов не будет, для этого есть специальный предмет — «Математическая статистика». Пока придётся верить на слово.

Пусть величина получена как результат серии измерений (пример: нам нужна длина маятника; одного измерения нам показалось мало. А в серии получился некоторый разброс). «Разумно» (на самом деле, это тоже связано с принципом наименьших квадратов — минимизация среднеквадратического отклонения; предлагаем всем самостоятельно придумать правдоподобное рассуждение, приводящее к такому выводу) выглядит идея взять в качестве приближения среднее значение всех измерений.

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

Среднеквадратичная ошибка среднего арифметического:

$$\sigma_x = \sqrt{\frac{\sum\limits_{i=1}^{n}(x_i - \overline{x})^2}{n(n-1)}}.$$

Рассмотрим случай, когда величина получена как коэффициент или свободный член линейной зависимости $y = ax + b$, где задана таблица для x и y. Тут есть разные случаи: если погрешности измеренных x и y велики, то смысла считать по формулам нет, и надо просто отмечать точки на координатной плоскости и «визуально» проводить наиболее точно приближающую прямую (погрешности на графике обозначают «крестиками», и такая прямая пройдёт через все крестики). Если же эти погрешности малы, то лучше воспользоваться методом наименьших квадратов; на первый план выходит статистическая погрешность.

Смысл метода: хотим подобрать a и b так, чтобы мера отклонения $\sum\limits_{i=1}^{n}(y_i - ax_i - b)^2$ была минимально возможной. На втором курсе вы научитесь исследовать функции многих переменных на экстремум, а пока поверьте, что

$$a = \frac{\langle xy\rangle - \langle x\rangle\langle y\rangle}{\langle x^2\rangle - \langle x\rangle^2}; \quad b = \langle y\rangle - a\langle x\rangle.$$

Погрешности метода наименьших квадратов:

$$\sigma_a = 2\sqrt{\frac{1}{n-2}\left(\frac{\langle y^2\rangle - \langle y\rangle^2}{\langle x^2\rangle - \langle x\rangle^2} - a^2\right)};$$

$$\sigma_b = \sqrt{\langle x^2\rangle}\sigma_a.$$

Здесь $\langle x\rangle$ — это среднее значение x, то же, что и $\overline{x}$.

И остался ещё один случай: когда мы ищем погрешность величины, вычисляемой по формуле. Простые случаи (а мы не помним, чтобы встречались другие):

$$\sigma_{a+b}^2 = \sigma_a^2 + \sigma_b^2$$

$$\varepsilon_{x^m y^n}^2 = m^2\varepsilon_x^2 + n^2\varepsilon_y^2$$

Обе формулы обобщаются на любое количество переменных; показатели степени считаются постоянными и необязательно целыми.

Иногда бывает, что результаты измерений ну никак не укладываются в привычную картину мира. Например, у измерения, которое должно быть точным,

получилась погрешность 20%; или значение ускорения свободного падения оказалось равным 2 м/с^2... Объяснить такое фиаско можно разными причинами:

1. ошибки в обработке данных;
2. неисправность экспериментальной установки;
3. неудовлетворительная точность используемой методики измерения;
4. выполнение работы кривыми руками.

На некоторых лабах (это относится ко второму курсу — Электричество и магнетизм, Оптика) вторая причина встречается в 90% случаев. Однако если вы будете объяснять непопадание результата в пределы допустимой погрешности, то преподаватель не попадёт в пределы желаемой оценки. Ведь во всём, конечно, студент виноват, как поётся в одной старой песне... Давайте разберёмся, что же делать, если результаты вычислений всё-таки не радуют.

RESULTS UNRELIABLE NON-NORMAL PROCEDURE

1. Проверить вычисления, в идеале — повторить их. Очевидно, что в связи с большими объёмами необходимо использование компьютера (об этом позже).
2. Если данные обработаны в соответствии с предложенной методикой, но получен некорректный результат, значит, есть два варианта: либо неудовлетворительна методика, либо неправильны данные.
3. Используя полученные при подготовке к работе теоретические знания и литературу, оценить точность применённой методики. Если эта оценка объясняет расхождение, нам повезло.
4. Если же нам не повезло, и мы поняли, что экспериментальные данные некорректны, то придётся признать, что к выполнению работы мы подошли несерьёзно. Чтобы избежать таких казусов, советую не отступать от инструкций и быть аккуратным на каждом шаге выполнения работы.

Немного об использовании компьютера при обработке экспериментальных данных. Она требует выполнения большого объёма однотипных вычислений. Для применения метода наименьших квадратов существуют различные онлайн-сервисы; по личному опыту, для обработки данных удобны «офисные» программы MS Excel и OpenOffice.

Что касается оформления работ в электронном виде, то отношение преподавателей к нему различно. Одни резко против, другим безразлично, третьи

приветствуют. Если вам не повезло с почерком (а он портится из-за необходимости много и быстро писать), то оформление в электронном виде может быть полезным. Обычно используют LaTex или Origin.

Ну и в заключение скажем: эти рекомендации относятся к тому случаю, когда вы попали к строгому преподавателю. Обычно всё гораздо проще, требования значительно мягче. Но не расслабляйтесь! И успехов в учёбе!

5 Базы данных

Базы данных (БД) — это один из самых знаменитых предметов на ФУПМе. Преподаётся, как и большинство факультетских предметов, кафедрой МОУ. Базы данных, наравне с английским языком и лабораторными работами по физике, — один из тех предметов, на которые ходить необходимо.

Раньше этот предмет проходился в четвёртом семестре, поэтому люди, склонные к большому количеству пропусков, до него не доживали. Сейчас, из-за переноса баз данных на первый семестр, знакомство с данным предметом состоится очень скоро.

Курс посвящён работе с системами управления базами данных. Всего 10 лабораторных работ, в каждой требуется выполнить своё задание, которое вам необходимо будет по выданному номеру найти на сайте http://bdis.umeta.ru в разделе «Навигация по материалам». Каждая работа оценивается от 1 до 8 баллов; суммарное количество баллов определяет оценку за зачёт.

Одна из главных целей курса — развить навыки работы с документацией. Действительно, любую требуемую информацию по функциям языка SQL, используемого при работе с базами данных, можно найти в справке MSDN или просто с помощью Google.

Также может быть очень полезным документ, который вот уже несколько лет пополняется на общественных началах:

Большинство вопросов к сдачам преподаватели берут отсюда [33].

Первое занятие — ознакомительное, на нём рассказывают правила игры и раздаёт варианты; все последующие — сдачи. Прогуливать, особенно первое занятие, не рекомендуется! На сдачах формируется очередь, опаздывать нельзя. Работающие скрипты необходимы, но не достаточны для получения достойной оценки! Надо готовиться и по теории.

Сдавать лабораторные надо в срок, а лучше — с запасом. Ведь если опоздать, можно попасть в дисконт, т. е. в режим сдачи, при котором за каждую работу ставят не более 4 баллов, а вопросы задают более сложные, причём, чем больше отставание, тем труднее сдавать. Дисконтирование начинается, если номер недели минус количество сданных лаб больше четырёх. Кажется, что запас в 4 недели — это много, но, ввиду того, что вы можете заболеть, проспать и т.д., а лабораторные работы (особенно вторая и третья) требуют времени и, возможно, нескольких попыток сдачи, советуем, наоборот, попытаться сдавать всё заранее и вперёд. При должном старании, к октябрю можно получить зачёт по данному предмету и освободить время на другие.

Выбраться из дисконта можно, но это сложно и зависит от везения. Тут как с курением: проще не начинать. Именно с дисконтом и особым отношением

преподавателей кафедры к «провинившимся» связаны страшилки про БД. А то, что этот предмет перенесли на первый семестр, говорит о том, что кафедра МОУ теперь — «санитары ФУПМа», как кафедра высшей математики — санитары Физтеха.

Ещё один момент. По БД есть лекции. Они кажутся бесполезными, так как лектор отстаёт от графика сдач на те самые 4 недели. Но там отмечают. И могут учесть посещаемость... Говорят, когда-то лекции не влияли вообще ни на что. А в более поздние времена те, у кого было слишком много пропусков, не могли получить зачёт.

На лекции, во избежание неприятностей, следует ходить. Лучше прийти на лекцию и делать там задание по математическому анализу, чем не прийти на неё вовсе. К тому же, так вы хоть что-то, о чём говорили на лекции, запомните. Учитывайте только, что лектор любит выключать свет и показывать слайды, но появление на лекции с налобным фонариком вызовет много вопросов, хотя и не запрещено.

Случается, и довольно часто, что после сдачи всех лабораторных не хватает баллов до желаемой оценки, а время до закрытия ведомостей ещё есть. В таких случаях разрешается брать дополнительные задания. Их сложность зависит от количества недостающих баллов, но обычно выше, чем у основных лабораторных. Однако все, кто хотел, их выполнили.

Также по базам данных есть экзамен. Он необязателен для всех у кого выходит оценка удовлетворительно и выше. При желании на него можно пойти, но сдать на хорошую оценку в этом случае очень сложно, лучше уж сдавать лабы в течение семестра и получить свой автомат.

В общем, если относиться к курсу «Базы данных» серьёзно с самого начала (а не когда всё стало плохо), готовиться к сдачам, пользоваться справочниками и головой, то всё будет хорошо, а страшилки останутся где-то за бортом. Удачи!

Список литературы

[1] Bondy J. A., M. U. S. R. Graph theory / Murty U. S. R. Bondy J. A. — Springer, 2008.

[2] J., V. D. How to prove it: a structured approach / Velleman D. J. — Cambridge University Press, 2006.

[3] Алексеев В. Б. Теорема Абеля в задачах и решениях / Алексеев В. Б. — Москва: МЦНМО, 2001.

[4] Бабичева Т.С., Бабичев Д.С. Учебно-методическое пособие к лекционному видеокурсу по математике для 8-11 классов школ Москвы. «Разбор тем, вызывающих наибольшие трудности при решении олимпиадных задач по математике» / Бабичева Т.С., Бабичев Д.С. — Кафедра высшей математики МФТИ, 2014.

[5] Бабичева Т.С., Бабичев С.Л. Введение в теорию игр и исследование операций. Материалы Летней Олимпиадной Школы / Бабичева Т.С., Бабичев С.Л. — 2016.

[6] Бабичева Т.С., Яковлев И.В., Бабичев Д.С., Бабичев С.Л., Бабичева Н.Н., Жогов А.А., Подаев М.В., Федоров Н.М. Пособие по олимпиадной математике. Уровень А1 / Бабичева Т.С., Яковлев И.В., Бабичев Д.С., Бабичев С.Л., Бабичева Н.Н., Жогов А.А., Подаев М.В., Федоров Н.М. — Эдитус, 2018.

[7] Беклемишев Д. В. Курс аналитической геометрии и линейной алгебры: Учеб. для вузов / Беклемишев Д. В. — 11 edition. — Москва: Физико-математическая литература, 2009.

[8] Берже М. Геометрия. Том I / Берже М. — Москва: Мир, 1984.

[9] Берже М. Геометрия. Том II / Берже М. — Москва: Мир, 1984.

[10] Бесов О. В. Курс лекций по математическому анализу / Бесов О. В. — Москва: МФТИ, 2004.

[11] Верещагин Н. К., Шень А. Х. Лекции по математической логике. Часть 1. Начала теории множеств / Верещагин Н. К., Шень А. Х. — 4 edition. — Москва: МЦНМО, 2012.

[12] Верещагин Н. К., Шень А. Х. Лекции по математической логике. Часть 2. Языки и исчисления / Верещагин Н. К., Шень А. Х. — 4 edition. — Москва: МЦНМО, 2012.

[13] Верещагин Н. К., Шень А. Х. Лекции по математической логике. Часть 3. Вычислимые функции / Верещагин Н. К., Шень А. Х. — 4 edition. — Москва: МЦНМО, 2012.

[14] Городенцев А. Л. Алгебра. Учебник для студентов-математиков. Часть I / Городенцев А. Л. — Москва: МЦНМО, 2013.

[15] Грэм Р., Кнут Д., Паташник О. Конкретная математика / Грэм Р., Кнут Д., Паташник О. — Москва: Мир, 1998.

[16] Журавлев Ю.И., Флеров Ю.А., Вялый М. Н. Дискретный анализ. Часть II. Основы высшей алгебры и теории кодирования / Журавлев Ю.И., Флеров Ю.А., Вялый М. Н. — Москва: МФТИ, 1999.

[17] Журавлев Ю.И., Флёров Ю.А. Дискретный анализ. Часть I. Алгебра логики, комбинаторика и теория графов. / Журавлев Ю.И., Флёров Ю.А. — Москва: МФТИ, 1999.

[18] Журавлев Ю.И., Флёров Ю.А., Вялый М. Н. Дискретный анализ. Часть III. Теория формальных систем / Журавлев Ю.И., Флёров Ю.А., Вялый М. Н. — Москва: МФТИ, 1999.

[19] Зорич, В.А. Математический анализ / Зорич, В.А. — 1984.

[20] Канель-Белов А.Я., Ковальджи А.К. Как решают нестандартные задачи / Канель-Белов А.Я., Ковальджи А.К. — Москва: МЦНМО, 2003.

[21] Клини С. К. Введение в метаматематику / Клини С. К. — Москва: Издательство иностранной литературы, 1857.

[22] Кострикин А. И. Введение в алгебру. Часть I. Основы алгебры / Кострикин А. И. — Москва: Физико-математическая литература, 2001.

[23] Кострикин А. И. Введение в алгебру. Часть II. Линейная алгебра / Кострикин А. И. — Москва: Физико-математическая литература, 2001.

[24] Кострикин А. И. Введение в алгебру. Часть III. Основные структуры / Кострикин А. И. — Москва: Физико-математическая литература, 2001.

[25] Кострикин А. И., Манин Ю. И. Линейная алгебра и геометрия / Кострикин А. И., Манин Ю. И. — Москва: Наука, 1986.

[26] Кудрявцев Л. Д. Краткий курс математического анализа / Кудрявцев Л. Д. — 3 edition. — Москва: Физико-математическая литература, 2005.

[27] Курант Р. Курс дифференциального и интегрального исчисления. Том I / Курант Р. — Москва: Наука, 1966.

[28] Ландо С. К. Лекции о производящих функциях / Ландо С. К. — Москва: МЦНМО, 2004.

[29] Ландо С. К. Дискретная математика / Ландо С. К. — Москва: МЦНМО, 2012.

[30] М. Вялый, В. Подольский, А. Рубцов, Д. Шварц, А. Шень. Лекции по дискретной математике. — публикация в сети. — 2017.

[31] Омельченко А. В. Теория графов / Омельченко А. В. — Москва: МЦНМО, 2018.

[32] Петрович А. Ю. Курс математического анализа / Петрович А. Ю. — Москва: МФТИ, 2017.

[33] ВМК МГУ. — Практикум по Базам Данных, методические материалы, 2012.

[34] Прасолов В. В. Элементы коибинаторной и дифференциальной топологии / Прасолов В. В. — Москва: МЦНМО, 2009.

[35] Тер-Крикоров А. М., Шабунин М. И. Курс математического анализа / Тер-Крикоров А. М., Шабунин М. И. — Москва: Физико-математическая литература, 2001.

[36] Харари Ф. Теория графов / Харари Ф. — Москва: Мир, 1973.

[37] Шень А. Х. О «математической строгости» и школьном курсе математики / Шень А. Х. — Москва: МЦНМО, 2006.

[38] Яковлев Г.Н. лекции по математическому анализу / Яковлев Г.Н. — ГУП Облиздат, 2004.