УДК 372.851
ББК 22

Федеральное государственное автономное образовательное
учреждение высшего образования
«Московский Физико-Технический Институт
(Государственный университет)»


# Теорминимум для поступивших на ФУПМ

Бабичева Татьяна Сергеевна, Бабичев Сергей Леонидович, Бабичев Дмитрий Сергеевич, Бабичев Олег Николаевич, Молчанов Евгений Геннадьевич, Плетнев Никита Вячеславович, Шпакова Татьяна Александровна.

# Содержание

# Часть I
# Введение

Молчит Молчанов... мир застыл в молчании...
На мой вопрос пока ответа нет...
О, сколько будет чувств в его звучании,
Когда я (вскоре?) получу ответ!

Ведь на ФУПМе я так хочу учиться:
Случайные процессы постигать,
И неслучайно буду не лениться,
Все пары обещаю посещать!

Ведь ФУПМ любого факультета круче,
И знаю я, в чём эта крутизна:
Дискретной математике научат,
И алгоритмы твёрдо буду знать.

А силу информатики признают,
Когда к нам подкрадётся новый крах,
А я и так давно всё это знаю:
Что мир почти у прогеров в руках.

Я верю, что поможет мат.статистика
Понять, куда покатит этот мир,
Подправить под себя характеристики,
И ФУПМ --- надёжный мой ориентир.

Борис Галицкий & редакторы.

Если вы держите в руках это методическое пособие, значит, вы, как и автор этого стихотворения, хотите учиться на ФУПМе. Мы не будем расписывать, чем отличается ФУПМ от других факультетов (это, например, есть в проспекте факультета), а лучше дадим некоторую базу для успешной учёбы.

Главным образом здесь раскрываются темы, которые изучаются не во всех школах, но большинством преподавателей воспринимаются как очевидные (основы олимпиадной математики и информатики). Также даются рекомендации по предметам, являющимся спецификой 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. Поэтому книга не предназначена для того, чтобы заменить учебники; её следует внимательно прочитать в самом начале первого семестра, а лучше — перед ним.

Авторы благодарят за предложенные замечания следующих студентов и выпускников МФТИ, а также других вузов:

- Алевтина Шакирова
- Валерия Петрушенко
- Илья Улитин
- Александр Пау
- Артём Адамов
- Артемий Городилов
- Юлия Городилова
- Валерия Немычникова
- Богдан Багно
- Яков Фиронов
- Мария Носарева
- Василий Короленков
- Константин Павлов

- Александр Марков
- Александр Кильянов

Методическое пособие подготовлено при поддержке деканата ФУПМ.

# Часть II
# Математика

## 1 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 http://goo.gl/WcU0Ss из 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 7.
- $\bigvee$ —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 $1, 2, 3, \ldots$ 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 ...». $\mathbb{Z} \setminus \mathbb{N} = \{0, -1, -2, -3, -4, ...\}$.

- $\infty$ — 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 $n < x$.»

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

# 2 Метод математической индукции

## 2.1 Метод математической индукции

**Метод математической индукции** (ММИ) используется для доказательства ряда утверждений, при этом каждое следующее утверждение доказывается из предыдущего:

$$Y_1 \to Y_2 \to Y_3 \to \dots Y_n \to Y_{n+1} \to \dots$$

Первое утверждение называется **базой индукции**.

Каждой стрелкой мы обозначили **индуктивный переход**: из предположения, что утверждение $n$ верно, мы доказываем верность $n+1$-го. Индукция работает следующим образом: из первого утверждения следует второе, из второго — третье и так далее вплоть до любого $n$. При этом обычно все переходы доказываются аналогичным образом.

Индукция подобна принципу домино: толкнув первую доминошку (база индукции), каждая доминошка будет толкать следующую (индукционный переход), и цепочка упавших домино добежит до любого домино.

**Задача 1.** Доказать, что для любого натурального $n$ выполняется равенство: $1^2 + 2^2 + \dots + n^2 = \dfrac{n(n+1)(2n+1)}{6}$.

**Решение.** База индукции выполняется при $n = 1$, действительно:

$$1^2 = \frac{1(1+1)(2 \cdot 1 + 1)}{6}.$$

Переход: $n$-ое утверждение формулируется так:

$$1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6};$$

$n+1$-ое утверждение:

$$1^2 + 2^2 + \dots + n^2 + (n+1)^2 = \frac{(n+1)((n+1)+1)(2(n+1)+1)}{6}.$$

Пользуясь $n$-ым утверждением, докажем $n+1$-ое:
Сумма квадратов первых $n$ чисел равна $\dfrac{n(n+1)(2n+1)}{6}$, отсюда: $1^2+2^2+\cdots+n^2+(n+1)^2 = \dfrac{n(n+1)(2n+1)}{6}+(n+1)^2$. А требуется доказать, что такая сумма равна $\dfrac{(n+1)(n+2)(2n+3)}{6}$.
После несложных преобразований получаем, что

$$\frac{n(n+1)(2n+1)}{6}+(n+1)^2 = \frac{(n+1)(n+2)(2n+3)}{6}, \text{ так как}$$

$$n(2n+1)+6(n+1) = (n+2)(2n+3), \text{ так как}$$

$$2n^2+n+6n+6 = 2n^2+7n+6.$$

Отметим, что база индукции, как правило, хоть и очевидна, всё же нуждается в проверке. Если нет базы, нет и индукции — некому толкнуть первое домино, чтобы запустить цепную реакцию.

Напоследок приведём пример ошибочного рассуждения: докажем, что любые $n$ лошадей одного цвета.

База: $n=1$. Действительно, база очевидна — любая лошадь такого же цвета, как и она сама.

Переход: утверждение $n$: любые $n$ лошадей одного цвета. Отсюда докажем, что любые $n+1$ лошадей одного цвета, действительно: 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.

Предлагаем читателю остановиться на минутку и задуматься — где ошибка в приведённом рассуждении?

А ошибка кроется в следующем: на самом деле, не верен переход $Y_1 \to Y_2$, в этом случае $Л_2$ не находится в обоих множествах, и рассматриваемые множества вообще не пересекаются.

Хоть остальные переходы верны: из второго утверждения следует третье, и третьего — четвёртое, и так далее, индукция работать не будет: первое домино упало, но не задело второе, и цепочка прервалась.

**Задача 2.** Определите, на сколько частей делят плоскость $n$ прямых, никакие две из которых не параллельны, и никакие три из которых не пересекаются в одной точке.

**Решение.** Попробуем сначала проследить закономерность, по которой изменяется число прямых. Нетрудно понять, что 1 прямая делит плоскость на 2 части, 2 прямых — на 4 части, 3 прямых — на 7 частей и 4 прямых на 11 частей (рис. 1).

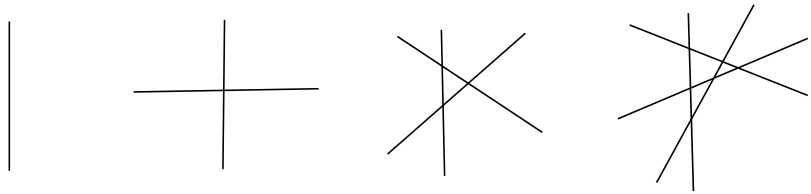


Рис. 1: Количество частей при 1,2,3,4 прямых.

Прослеживается закономерность: каждая следующая прямая добавляет на одну часть больше, чем предыдущая. Докажем, что $n$-ая прямая добавит $n$ частей. Действительно, она пересечёт каждую из остальных $n-1$ прямых, добавив при этом $n-1$ точку пересечения. Эти $n-1$ точка разделит прямую на $n$ кусков, каждый из которых разрежет один из кусков на 2 части: число кусков увеличится на $n$. Например на рисунке 2 $n=4$, и прямая разрежет каждый из кусков $A, B, C, D$ на две части. Итак, $n$-ая прямая добавит $n$ кусков, и, следовательно, все прямые разрежут плоскость на $2+2+3+\cdots+n$ кусков. Докажем по индукции, что

$$2+2+3+\cdots+n=\frac{n^2+n+2}{2}.$$

База индукции выполняется при $n=2$.

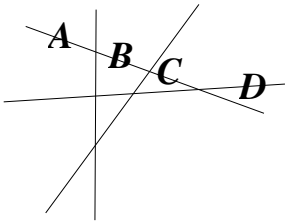



Рис. 2: Увеличение количества частей при добавлении прямой.

Переход: требуется доказать $n+1$-ое утверждение:

$$2+2+3+\cdots+n+n+1=\frac{(n+1)^2+(n+1)+2}{2}.$$

Докажем, что приращение левой части равно приращению правой части:

$$n=\frac{(n+1)^2+(n+1)+2}{2}-\frac{n^2+n+2}{2}\Leftrightarrow$$

$$\Leftrightarrow n=\frac{n^2+2n+1+n+1+2-n^2-n-2}{2}\Leftrightarrow n=n.$$

Переход доказан, поэтому $n$ прямых общего положения делят плоскость на $\dfrac{n^2+n+2}{2}$ кусков, что и требовалось найти.

**Задача 3.** Докажите, что

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

**Решение.** База индукции: $n=1$, тогда $1\cdot 1!=(1+1)!-1$.
Переход. Докажем $n+1$-ое утверждение:

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

Докажем, что приращение левой части равно приращению правой части. 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.

## 2.2 Метод полной математической индукции

Напомним метод математической индукции: он состоит из базы индукции (утверждение 1) и индукционного перехода (из $n$-ого утверждения следует $n+1$-ое утверждение).

$$Y_1 \to Y_2 \to Y_3 \to \cdots \to Y_n \to Y_{n+1} \to \ldots$$

Особенность **метода полной математической индукции** в том, что при доказательстве утверждения номер $n+1$ можно пользоваться истинностью всех утверждений от номера 1 до номера $n$. Формально это можно записать так:

$$\text{из } \{Y_1, Y_2, \ldots, Y_n\} \text{ следует } Y_{n+1}.$$

Обычно при применении метода полной математической индукции используются лишь несколько предыдущих утверждений, а не сразу все. Можно привести следующую аналогию: метод математической индукции можно было представить, как набор домино, и чтобы упала текущая доминошка, её должна была «подтолкнуть» предыдущая. А теперь домино каким-то образом связали некими шарнирами, и для того, чтобы упала $n+1$-ая домино, должны упасть все домино, с которыми она соединена.

**Задача 4.** Известно, что число $x + \dfrac{1}{x}$ является целым. Доказать, что при любом натуральном $n$ число $x^n + \dfrac{1}{x^n}$ также является целым.

**Решение.** Докажем сначала утверждение для $n = 2$: действительно, возведём $x + \dfrac{1}{x}$ в квадрат: отсюда $x^2 + 2 + \dfrac{1}{x^2}$ — целое число, а значит и $x^2 + \dfrac{1}{x^2}$ — также целое. Попробуем доказать утверждение под номером $n+1$: требуется доказать, что $x^{n+1} + \dfrac{1}{x^{n+1}}$ — целое. Утверждение под номером $n$ утверждает, что $x^n + \dfrac{1}{x^n}$ — целое, а утверждение 1: $x + \dfrac{1}{x}$ — целое. Перемножим эти два числа, отсюда получается, что $\left(x^n + \dfrac{1}{x^n}\right)\left(x + \dfrac{1}{x}\right) =$

$x^{n+1}+\dfrac{1}{x^{n+1}}+x^{n-1}+\dfrac{1}{x^{n-1}}$ — также целое число. Но $n-1$-ое утверждение гласит, что $x^{n-1}+\dfrac{1}{x^{n-1}}$ — целое число, а отсюда следует, что и $x^{n+1}+\dfrac{1}{x^{n+1}}$ — также целое число, что и требовалось.

Заметим, что в этой задаче мы доказали $Y_{n+1}$, пользуясь $Y_n$, $Y_{n-1}$ и $Y_1$. То есть $n+1$-ое домино соединено шарнирами с $n$-й, $n-1$-й и 1-й домино.

**Задача 5.** При каких $n > 3$ набор гирь с массами $1, 2, 3, \ldots, n$ 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 $n+1, n+2, \ldots, n+6$ 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.

# 3 Алгебра

## 3.1 Системы линейных уравнений

Умение решать системы линейных уравнений (СЛУ) — один из самых базовых навыков. Решение подобных задач потребуется и при решении задач математического анализа, и в задачах 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, 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 $E$.

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

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

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: 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 $\det A$). Поначалу в курсе аналитической геометрии потребуются определители только 2 и 3 порядков, поэтому не будем давать общего определения.

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

Определитель третьего порядка можно считать по-разному. Основной способ — разложение по столбцу или строке (т. к. $\det A^T = \det A$, эти способы эквивалентны):

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

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 $(*)$ 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 (т. е., по определению, $\det A \neq 0$), то существует и единственна обратная к $A$ матрица $A^{-1}$ такая, что $AA^{-1} = A^{-1}A = E$.

Элементы обратной матрицы можно вычислять по следующим формулам:

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

где $d_{ji}$ — дополнительный минор элемента $a_{ji}$, т. е. определитель матрицы, полученной из матрицы $A$ вычёркиванием $j$ строки и $i$ столбца (на пересечении которых расположен $a_{ji}$). Для матриц второго и третьего порядка этого способа достаточно.

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)

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

Если внимательно посмотреть на эту систему, то можно увидеть,

что она кратко записывается 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

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} & b_m \end{array}\right)$$

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 (по индукции 2.1) 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:

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

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/или читать учебники.

## 3.2 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):

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

**Задача 6.** 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. Также отсюда следует способ поиска целых корней многочлена: нужно перебрать все делители свободного члена. Однако, следует понимать, что не обязательно каждый многочлен с целыми коэффициентами имеет целый корень.

## 3.3 Разложение многочлена на множители

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

**Задача 7.** Решить уравнение: $P(x) = x^3 - 3x^2 - 6x - 20 = 0$.

**Решение.** 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 $n$ корней, но следствие из основной теоремы алгебры гласит, что многочлен $n$-ой степени не может иметь больше $n$ корней.

Если быть более точными, любой многочлен можно разложить на следующие 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 $a_1, ..., a_k$ — корни знаменателя кратностей $n_1, ...., n_k$, а трёхчлены $x^2 + b_ix + c_i$ всюду больше нуля (то есть не имеют корней).

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

**Задача 8.** 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 $x^6 - 5x^4 - 6x^3 + 6x^2 + 16x + 8$.

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:

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

## 3.4 Разложение дроби в сумму элементарных

Как только у вас начнутся интегралы (а это будет примерно на втором семинаре по математическому анализу), 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 $\frac{P(x)}{Q(x)}$, 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 $\frac{P(x)}{Q(x)} = A(x) + \frac{B(x)}{Q(x)}$. 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 $\frac{B(x)}{Q(x)}$, 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 9.** 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 3.1, получаем

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

Таким образом,

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

# 4 Теория чисел

## 4.1 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...

Напомним основную теорему арифметики: 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 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 называются взаимно простыми, если их наибольший общий делитель равен 1.

Наименьшим общим кратных 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

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

.

Вспомним также **алгоритм Евклида** для нахождения НОД двух чисел: $(a, b) = (a - b, b)$, если $a > b$: каждым его шагом будет вычитание из большего числа меньшего. Естественным его

улучшением 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 10.** Каким может быть число $\dfrac{(a+b)(b+c)(c+a)}{abc}$, если известно, что это целое число и $a, b$ и $c$ — попарно взаимно простые натуральные числа?

**Решение.** Посмотрим, есть ли среди чисел $a$,$b$,$c$ равные. Возможны 2 варианта:

1) среди них есть два равных. Пусть это, не умаляя общности, числа $a$ и $b$. Так как они должны быть взаимно простыми, то это возможно, только тогда, когда $a = b = 1$. Поэтому

$$\frac{(a+b)(b+c)(c+a)}{abc} = \frac{2(c+1)^2}{c}.$$

Но числа $c+1$ и $c$ всегда взаимно простые, поэтому и $(c+1)^2$ и $c$ взаимно простые, но $2(c+1)^2 \vdots c$, значит $2 \vdots c$, откуда $c = 1$ или $c = 2$. В первом случае значение $\dfrac{(a+b)(b+c)(c+a)}{abc}$ равно 8, во втором - 9.

2) среди чисел $a, b, c$ все различные. Не умаляя общности можно считать, что $a < b < c$. Так как числа $a$, $b$ и $c$ попарно взаимно простые, то ни $a+c$, ни $b+c$ не может делиться на $c$, значит, $a+b \vdots c \Leftrightarrow a+b = kc$ для некоторого целого $k$. Аналогично, $a + c = lb$ для целого $l$.

Так как $a + b < 2c$, и $a + b = kc$, то $a + b = c \Rightarrow a = c - b$. Отсюда $a+c = (c-b)+c = lb \Leftrightarrow 2c = b(l+1)$. Правая часть этого выражения делится на $b$, значит, и левая тоже должна делиться на $b$, но $c$ и $b$ взаимно просты, следовательно, 2 должно делиться на $b \Rightarrow b = 1$ или $b = 2$. Но $1 \leqslant a < b$, откуда $b$ может быть равно только 2. Так как $a < b$, то $a = 1$, $c = a + b = 3$. В этом случае $\dfrac{(a+b)(b+c)(c+a)}{abc} = \dfrac{3 \cdot 5 \cdot 4}{1 \cdot 2 \cdot 3} = 10.$

**Задача 11.** 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 это ни 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.

## 4.2 Сравнения по модулю

В этом разделе мы будем рассматривать только целые числа.

**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 12.** 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 ФУПМа, вспомним красивый факт: $1001 = 7 \cdot 11 \cdot 13$, 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 $2014 \equiv 5 (\mathrm{mod}\, 7)$ и $2015 \equiv 6 (\mathrm{mod}\, 7) \Rightarrow 2013 \cdot 2014 \cdot 2015 \equiv 4 \cdot 5 \cdot 6 = 120 \equiv 1 (\mathrm{mod}\, 7)$ $2016 \equiv 0 (\mathrm{mod}\, 7)$, а значит $2016^3 \equiv 0^3 (\mathrm{mod}\, 7)$. Окончательно: $2013 \cdot 2014 \cdot 2015 + 2016^3 \equiv 1 + 0 = 1 (\mathrm{mod}\, 7)$.

В некоторых случаях удобнее перейти к отрицательным числам: например $8^{100} \equiv (-1)^{100} = 1 (\mathrm{mod}\, 9)$.

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:

$$\overline{a_n a_{n-1} \ldots a_2 a_1 a_0} \equiv a_n + a_{n-1} + \ldots a_2 + a_1 + a_0 (\mathrm{mod}\, 9)$$

и

$$\overline{a_n a_{n-1} \ldots a_2 a_1 a_0} \equiv a_n + a_{n-1} + \ldots a_2 + a_1 + a_0 (\mathrm{mod}\, 3).$$

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 13.** Доказать, что ни при каких целых $n$ число $n^2 + 3n + 4$ не делится на 9.

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

| $n (\mathrm{mod}\, 9)$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $n^2 (\mathrm{mod}\, 9)$ | 0 | 1 | 4 | 0 | 7 | 7 | 0 | 4 | 1 |
| $3n (\mathrm{mod}\, 9)$ | 0 | 3 | 6 | 0 | 3 | 6 | 0 | 3 | 6 |
| $n^2 + 3n + 4 (\mathrm{mod}\, 9)$ | 4 | 8 | 5 | 4 | 5 | 8 | 4 | 2 | 2 |

Рассмотрим, например, случай, когда $n$ даёт остаток 5 при делении на 9. Тогда $n \equiv 5 (\mathrm{mod}\, 9) \Rightarrow n^2 + 3n = 4 \equiv 5^2 + 3 \cdot 5 + 4 = 25 + 15 + 4 \equiv 7 + 6 + 4 = 17 \equiv 8 (\mathrm{mod}\, 9)$; 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 14.** 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: $N \equiv 13 \cdot 14 \equiv 1 \cdot 2 = 2 (\mathrm{mod}\, 3)$.

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

| $x (\mathrm{mod}\, 3)$ | 0 | 1 | 2 |
|---|---|---|---|
| $x^2 (\mathrm{mod}\, 3)$ | 0 | 1 | 1 |

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

# Часть III
# Дискретный анализ

## 5 Комбинаторика

### 5.1 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 15.** 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 16.** 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 17.** Сколько пар натуральных чисел удовлетворяет равенству $2x + 5y = 90000$ ?

**Решение.** Так как $5y$ и 90000 делятся на 5, то $2x$ тоже долж-

но 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.

## 5.2 «И» или «ИЛИ»

Перейдём к более интеллектуальным методам решения задач.

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

2. Правило умножения: если объект А можно выбрать $n$ способами, а объект В можно выбрать $m$ способами, то выбрать пару объектов А **и** В 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 18.** 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?

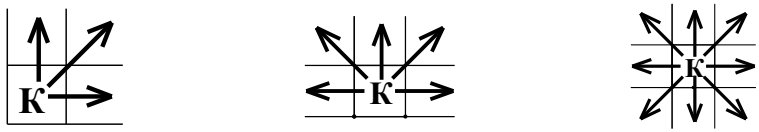
**Решение.** Пользуемся правилом умножения: нужно выбрать камень, который он бросает, **и** окно, в которое он его бросает. Итого получается $5 \cdot 4 = 20$ способов.

**Задача 19.** 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.




а) б) в)

Рис. 3: Различные способы расстановки короля.

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.

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 20.** 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 $19 \cdot 20/2 = 190$.

## 5.3 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») — 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 чис-

ла 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 «**Бином Ньютона**», именно поэтому числа сочетаний называют также биномиальными коэффициентами.

**Треугольник Паскаля** — это способ записи биномиальных коэффициентов (рис. 4)

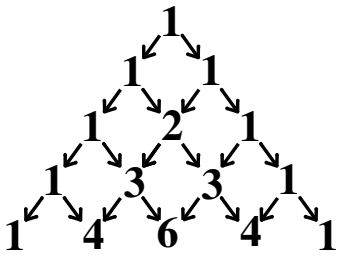


Рис. 4: Треугольник Паскаля.

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 21.** 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 22.** 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 23.** 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.

## 5.4 Задача о шарах и перегородках

Рассмотрим далее следующие классические задачи комбинаторики:

**Задача 24.** 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 25.** 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 каж-

дый 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 26.** 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 $C_{20+5-1}^{5-1} = C_{24}^{4}$.

## 5.5 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 $\alpha_1$ — свойство делимости числа на 3, а $\alpha_2$ - 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 27.** 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 $\alpha_i$, 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:

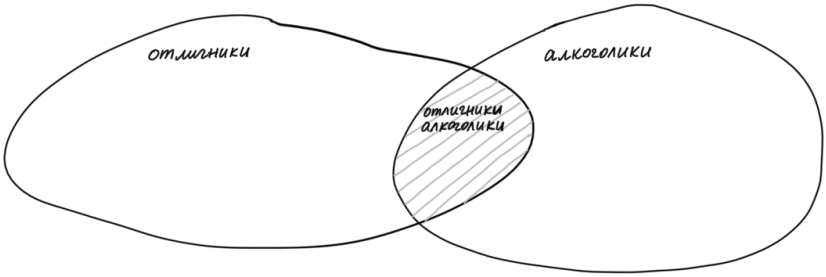
$$\begin{cases} b_1 + b_2 + b_3 + b_4 + b_5 + b_6 = 27 \\ b_1 > 9 \end{cases}$$

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 28.** 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 $\alpha_1$ делимость числа на $\beta_1$, аналогично введём свойства $\alpha_2$ и $\alpha_3$.

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 $\alpha_1, \alpha_2$ и $\alpha_3$, 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, $N(\alpha_2) = \dfrac{N}{p_2}$; $N(\alpha_3) = \dfrac{N}{p_3}$; $N(\alpha_1, \alpha_2) = \dfrac{N}{p_1 p_2}$; $N(\alpha_1, \alpha_3) = \dfrac{N}{p_1 p_3}$; $N(\alpha_2, \alpha_3) = \dfrac{N}{p_2 p_3}$; $N(\alpha_1, \alpha_2, \alpha_3) = \dfrac{N}{p_1 p_2 p_3}$. Отсюда:

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

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

что и требовалось найти.

**Задача 29.** (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 $\alpha_k$ 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:

$$N(\overline{\alpha_1}, \overline{\alpha_2}, \overline{\alpha_3}, \overline{\alpha_4}, \overline{\alpha_5}) = N - C_5^1 N(\alpha_1) + C_5^2 N(\alpha_1, \alpha_2) - C_5^3 N(\alpha_1, \alpha_2, \alpha_3) +$$
$$+ C_5^4 N(\alpha_1, \alpha_2, \alpha_3, \alpha_4) - C_5^5 N(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) =$$
$$= 120 - 5 \cdot 24 + 10 \cdot 6 - 10 \cdot 2 + 5 \cdot 1 - 1 = 44,$$

что и требовалось найти.

## 5.6 Рекурренты в комбинаторике

**Задача 30.** 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:

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

2. 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.

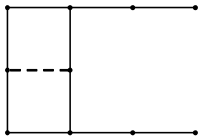
**Задача 31.** Сколько существует способов разрезать прямоугольник 2 на 15 на прямоугольники размером 1 на 2?

**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 $X_n$.

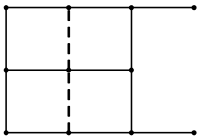
Рассмотрим левую нижнюю клетку прямоугольника. Она обязательно занята каким-то прямоугольником $1 \times 2$. Возможны 2 случая:

1. Прямоугольник $1 \times 2$ расположен вертикально (рис. 5а), тогда требуется разрезать прямоугольник размером $2 \times (n-1)$, это можно сделать $X_{n-1}$ способами

2. Прямоугольник $1 \times 2$ расположен горизонтально, тогда над ним также обязан быть расположен горизонтальный прямоугольник (рис. 5б) и остаётся замостить доску размером $2 \times (n-2)$, что можно сделать $X_{n-2}$ способами.



а) вертикально



б) горизонтально

Рис. 5: Разбиение прямоугольника $2 \times n$.

Поэтому для чисел $X_n$ выполняется рекурентное 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.

# 6 Теория графов

## 6.1 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 — это набор вершин, некоторые из них соединены рёбрами. Наглядно изобразить это можно так: пусть вершины — это некоторые города, а рёбра графа — это дороги, их соединяющие (рисунок 6). При этом в графе не важно расположение вершин или факт пересечения рёбер — один и тот же граф может быть изображён различными способами.


Рис. 6: 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 6 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 32.** 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 связанности** графа.

**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.

**Задача 33.** 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 убирания дороги (ребра) граф перестал быть связным. Это значит, что граф разбился на две несвязанные между собой части (см. рис. 7). Посчитаем сумму степеней в левом подграфе. Пусть в ней осталось $k$ городов. Из каждого го-

рода, 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.

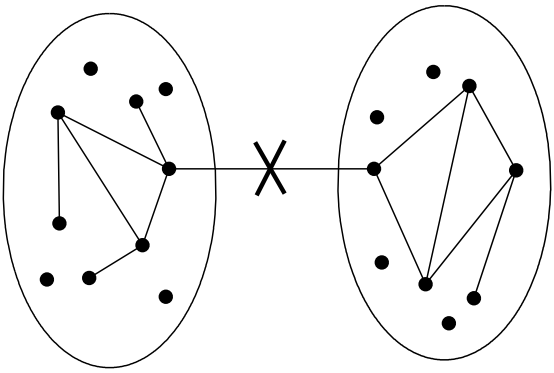


Рис. 7: При удалении ребра граф распадается на 2 компоненты

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 34.** В общежитии живёт 214 студентов. Каждый час ровно 4 из них отправляются на кухню перекусить. Может ли так получиться, что в 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 35.** 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.

## 6.2 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.

Например на рис. 8 $B - C - D - E - B$ и $F - G - H - F$ — это циклы.

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

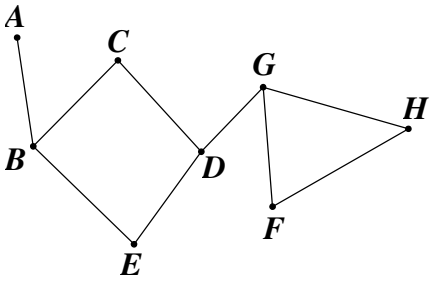



Рис. 8: Пример графа.

вершин.

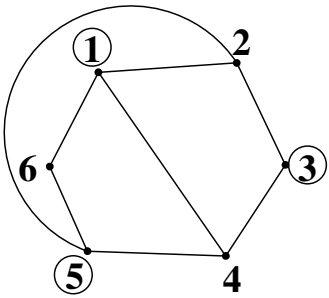
**Определение.** Граф называется **планарным**, если его можно так изобразить на плоскости, чтобы у него не пересекались рёбра (рёбра графа не обязательно должны быть прямыми отрезками!).

**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.

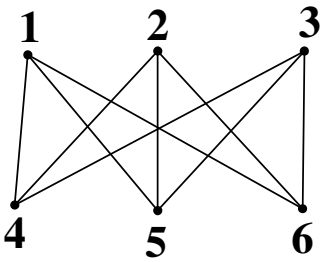
**Задача 36.** Есть три дома и три магазина — продуктовый, аптека и хозяйственный. Можно ли от каждого дома проложить тропинку до каждого из магазинов так, чтобы тропинки не пересекались?

**Решение.** Без труда строится пример, в котором проведены все тропинки, кроме одной, а вот последнюю провести не получается. (рис. 9а)

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. 9б). 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




а) «почти пример»




б) все тропинки проведены

Рис. 9: К задаче о домах и магазинах.

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

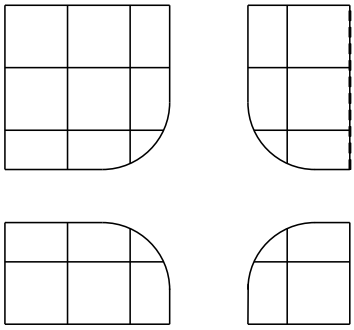


Рис. 10: Волейбольная сетка.

**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 38.** 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 39.** 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

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

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

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

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

## 6.3 Ориентированные графы

Граф называется ориентированным, если его рёбра имеют направление. Изображается это обычно стрелкой на конце в какую-либо сторону (рис. 11). 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.

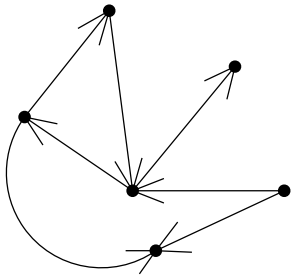


Рис. 11: Ориентированный граф.

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 связанным, если любые две его вершины сильно связаны.

Любой граф разбивается на компоненты сильной связанности, между которыми проведены ориентированные рёбра (рис. 12).

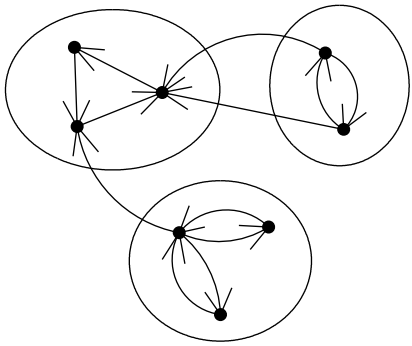


Рис. 12: Компоненты сильной связанности.

**Задача 40.** 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.

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

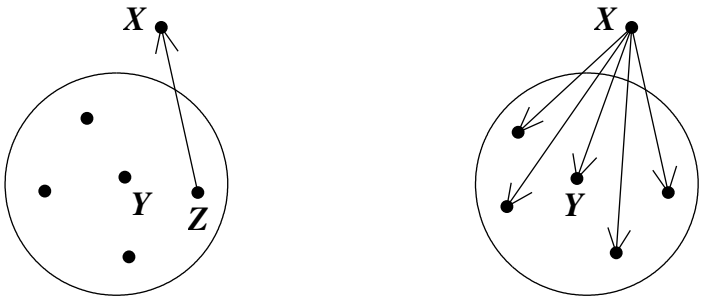
1. Какая-то дорога ведёт в $X$. Тогда искомый город — $Y$, из него можно добраться до любого города, и в том числе до $X$ (рис. 13а).

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. 13б).

Граф, описанный в данной задаче, имеет своё название.

**Определение.** Направленный полный граф называется турниром.

**Задача 42.** 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.




а)1 случай. б)2 случай.

Рис. 13: К задаче 2.

**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 43.** 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 14.

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

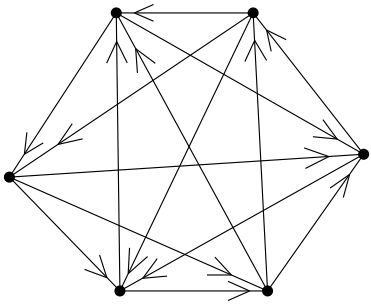


Рис. 14: База: $n = 1$.

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. □

**Задача 44.** 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.

Напомним вначале теоремы для неориентированных графов

В 1952 году было сформулировано **условие Дирака** существования гамильтонова пути: пусть $p$ — число вершин в данном графе и $p > 3$; если степень каждой вершины не меньше, чем $\frac{p}{2}$, то данный граф — гамильтонов.

Условие **Óре**: пусть $p$ — количество вершин в данном графе и $p > 2$. Если для любой пары несмежных вершин $(x, y)$ выполнено неравенство $\deg x + \deg y \geqslant p$, то данный граф – гамильтонов (другими словами: сумма степеней любых двух несмежных вершин не меньше общего числа вершин в графе).

Сформулируем **условие Дирака** для ориентированных графов:

Если сильно связный орграф $G$ без кратных дуг и петель содержит $n$ вершин, где $n > 2$, и для всякой его вершины $v$ выполнено неравенство $od(v) + id(v) > n$, то орграф $G$ является гамильтоновым.

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[1].

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

[1] 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 46.** 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.

**Задача 47.** 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 48.** 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.

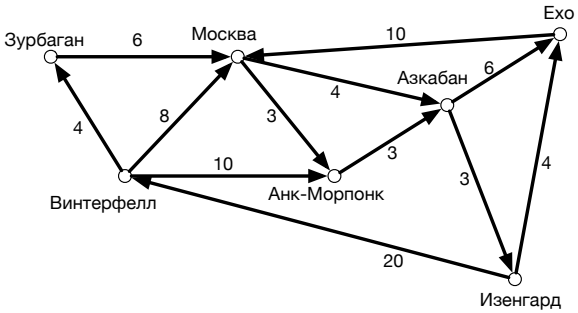
Переход: опять же, давайте удалим вершину, сделаем оставшийся граф сильно связным (по предположению индукции) и вернем вершину на место. Чтобы после возвращения граф сохранил сильную связность, удаленная вершина должна иметь как входящие, так и выходящие ребра. А если такой вершины нет? Тогда рассмотрим 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, ч.т.д.

## 6.4 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 неперебо́рных 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 компьютерных сетях. Есть много его вариантов, давайте рассмотрим тот, который предложил сам Дейкстра.

Этот алгоритм состоит из конечного количества шагов (вы же понимаете, что алгоритм, состоящий из бесконечного количества

шагов бесполезен, так как никогда не завершится?). Каждый шаг заключается в том, что к уже построенному до этого ДКП мы добавляем ровно одну вершину. В самом начале алгоритма ДКП состоит ровно из одной *корневой* вершины, которую мы назовём $s$, от слова *source* — источник. Для алгоритма нам понадобится массив $d$, значениями элементов $d_i$ которого будут кратчайшие расстояния от корневого узла до узла $i$. Вначале все элементы этого массива содержат бесконечность, за исключением элемента с номером $s$. Матрица $w$ содержит элементы $w_{ij}$, которые равны расстоянию от узла $i$ до узла $j$, если они — соседи или бесконечность, если они — не соседи.

Один шаг, или, как говорят алгоритмисты, *итерация* алгоритма заключается в следующем:

1. В ДКП заносится корневой узел (исток).

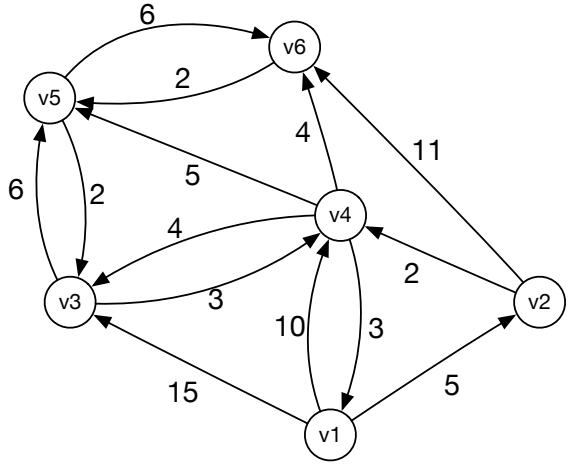
2. На каждом шаге в ДКП добавляется одно ребро, которое формирует кратчайший путь из истока в не-ДКП. Операция изменения старой кратчайшей длины на новую называется *релаксацией*.

3. Вершины заносятся в ДКП в порядке их расстояния по ДКП от начальной вершины.

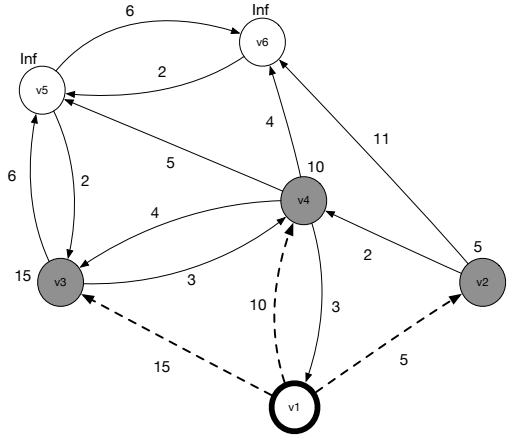
Посмотрим на примере.

Рядом с каждым из узлов будем писать значение уже достигнутого расстояния до него от корневого узла.

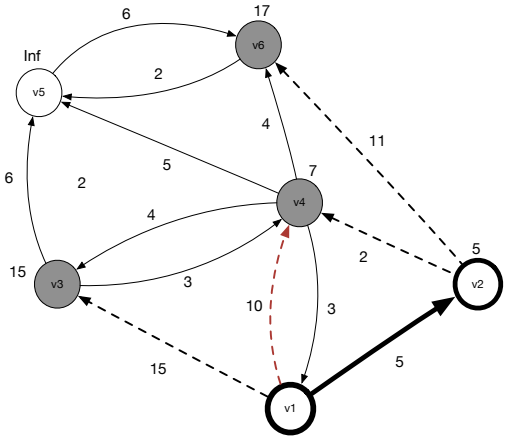
Исходный граф




Помещаем $v_1$ в ДКП, а $v_2$, $v_3$ и $v_4$ помечаем серым цветом, они — наши кандидаты в узлы выбора для следующего шага.

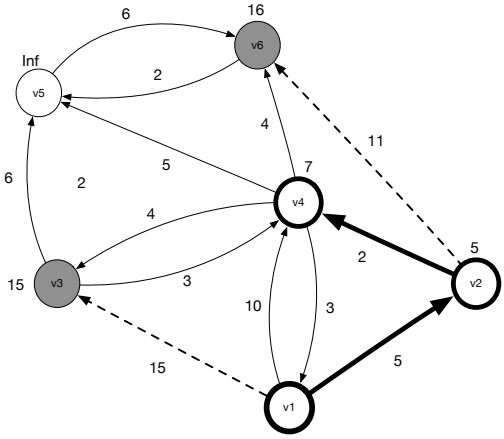



Пробежимся по всем серым узлам. Пусть, например, выбран узел $v_2$. Мы корректируем расстояния от него. Релаксация: бывший кратчайший путь $(1 \to 4)$ заменён на $(1 \to 2 \to 4)$. Пересчитываем расстояние и пишем новое число рядом с 4. Узел $v_2$ — самый близкий к ДКП, выкрашиваем его в чёрный цвет.

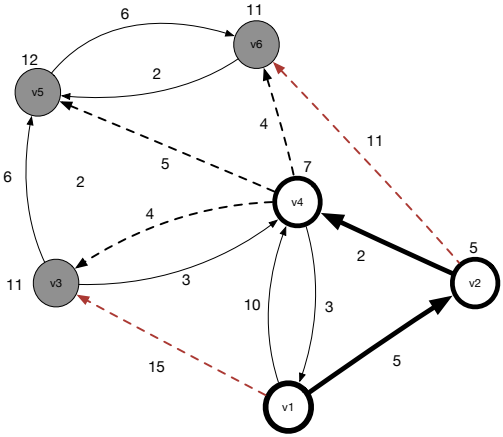


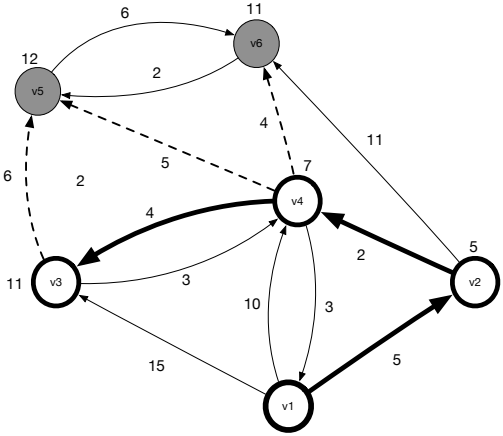

В ДКП у нас $v_1$ и $v_2$, в не-ДКП - остальные.

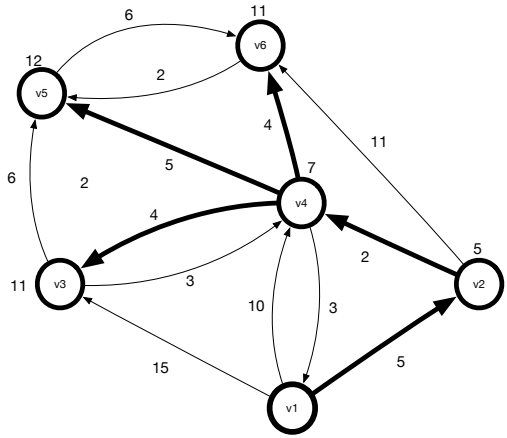
Нетрудно убедиться, что теперь в ДКП попадёт узел $v_4$, как наиболее близкий от не-ДКП к ДКП. Проводим все релаксации.




При просмотре узла $v_5$ как кандидата проводится группа релаксаций: $(1 \to 2 \to 6)$ заменено на $(1 \to 2 \to 4 \to 6)$, $(1 \to 3)$ — на $(1 \to 2 \to 4 \to 3)$.


v6
v5
v4
v3
v2
v1
6
11
12
2
4
11
5
6
2
7
4
2
5
11
3
10
3
5
15


v6
v5
v4
v3
v2
v1
6
11
12
2
4
11
5
6
2
7
4
2
5
11
3
10
3
5
15




У этого алгоритма очень много реализаций, мы рассмотрели одну из самых простых.

На выходе алгоритма получено искомое Дерево Кратчайших Путей.

Таким образом, можно, например, оценить, сколько времени ехать в каждое место на карте. Аналогичные схемы используются при оценках времени в пути в различных мобильных приложениях.

# 7 Основы алгебры логики

```
Лекция по филологии.
Старый профессор рассказывает: --- В некоторых языках мира
двойное отрицание означает согласие. В других, двойное
отрицание так и остается отрицанием, но нет ни одного
языка в мире, в котором двойное согласие означает отрицание.

Голос с задней парты: --- Да-да, конечно.
```

Алгебра логики изучает операции над высказываниями. Высказывание может быть истинным (обозначается 1) или ложным (обозначается 0).

Логическая (булева) переменная — это переменная, которая может принимать значения 0 (ложь) или 1 (истина). Высказывание может содержать логические переменные и константы, соединённые знаками логических операций.

Высказывание задаёт некоторую логическую функцию, заданную на булевом кубе $\{0,1\}^n$ и принимающую значения 0 и 1.

Таблица истинности высказывания — это таблица, задающая ту же функцию, что и данное высказывание.

Какие мы знаем логические операции? Из возникающих «естественно»:

- отрицание («не», обозначается $\overline{x}$ или $\neg x$). Отрицание высказывания истинно тогда и только тогда, когда само высказывание ложно.

| $x$ | $\neg x$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

- дизъюнкция («или», обозначается $x \vee y$). Дизъюнкция выражений истинна тогда и только тогда, когда хотя бы одно из высказываний истинно.

| $x$ | $y$ | $x \vee y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

- конъюнкция («и», обозначается $x \wedge y$). Конъюнкция выражений истинна тогда и только тогда, когда оба высказывания истинны.

| $x$ | $y$ | $x \wedge y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

В простом изложении определение высказывания формулируют, имея в виду именно эти операции. На самом деле, конъюнкцию или дизъюнкцию можно исключить. Что это значит? Что любую логическую функцию можно записать с помощью двух оставшихся элементарных операций.

Два высказывания называются эквивалентными (если хотите, равными), если они зависят от одного и того же набора переменных, и при любых значениях входящих в них переменных они принимают одинаковые значения. Похоже на определение равенства функций (а точнее, является его частным случаем).

Покажем, что высказывания $\neg(\neg x \vee \neg y)$ и $x \wedge y$ эквивалентны. Для второго из них мы уже выписали таблицу истинности, а для первого построим. Это делается последовательным вычислением входящих в него более простых высказываний.

| $x$ | $y$ | $\neg x$ | $\neg y$ | $\neg x \vee \neg y$ | $\neg(\neg x \vee \neg y)$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

Если выбросить из таблицы промежуточные выкладки, очевидно, что таблицы истинности совпадают.

Доказательство того, что конъюнкцию можно выразить через дизъюнкцию и отрицание, оставляем в качестве упражнения.

Ну а более сложным вещам вас научат на парах по АЛКТГ (алгебра логики, комбинаторика, теория графов). Но дойдут до этих пар не все физтехи. Успехов!

# Часть IV
# Физика

## 8 Лабораторные работы по физике

### 8.1 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? Давайте поговорим 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.

## 8.2 Расчёт погрешностей

Поговорим о расчёте погрешностей.

Согласно определению, абсолютная погрешность величины $x$ (обозначается $\sigma_x$ или $\Delta x$) — 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):

$$\sigma_{a+b}^2 = \sigma_a^2 + \sigma_b^2$$

$$\varepsilon^2_{x^m y^n} = m^2\varepsilon^2_x + n^2\varepsilon^2_y$$

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 (например,

http://www.chem-astu.ru/science/lsq/); 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 скажу: 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!

# Часть V
# Основы информатики

## 9 Языки программирования

### 9.1 Введение

В современным мире одним из самых важных фактором успеха становится информация. Человечество создает огромные потоки данных, и умение манипулировать информацией и контролировать её становится основой успеха для каждой хоть сколько-то крупной компании.

Как следствие, люди нуждаются в соответствующих инструментах для обработки, хранения информации и различного рода взаимодействия с ней. Появление компьютеров позволило обрабатывать большие объемы данных за короткие промежутки времени, и меньше, чем за столетие, компьютеры прошли путь от шкафов-калькуляторов до многокластерных распределенных суперкомпьютеров.

Для того, чтобы с компьютером могли работать не только посвященные адепты битовых операций, но и простые смертные, необходимо создавать удобные программы, а это не так просто, как кажется на первый взгляд. Увы, но даже самая «умная» часть компьютера, процессор, по факту умеет не больше, чем делать простые арифметические операции и работать с памятью, хотя и делает это много миллионов раз в секунду.

Само собой, компьютер не понимает человеческую речь, а для того, чтобы он начал делать что-то полезное, необходимы специальные языки для написания программ. Языки программирования шли вместе с компьютерами рука об руку на протяжении всего времени существования компьютеров. На сегодняшний день существует большое количество языков программирования под самые разные требования (про них мы ещё упомянем), а сейчас взглянем на то, с чего все начиналось.

## 9.2 Assembler

Если не уходить совсем вглубь к перфокартам и непосредственному вбиванию единичек и ноликов в память вычислительных
устройств, первым, кого можно называть языком программирования, был, пожалуй, assembler. Многие про него слышали, но мало кто видел, не так ли? Давайте взглянем, на простейшую программу, реализующую сложение двух чисел, написанную на этом языке.

```
mov eax,[x]        ;Загружаем значение x в eax
mov edx,[y]        ;Загружаем значение y в edx
add eax,edx        ;eax = eax+edx
mov [z],eax        ;z = eax = x + y
```

Ни в коем случае не пытайтесь понять, как именно работает каждая строчка. Но если вы присмотритесь, вы увидите, что по существу все эти операции — это обертки над простыми арифметическими операциями или операциями работы с памятью. Можно представить, как выглядел в 50-х годах человек, который понимал, что эти строчки делают... Тем не менее, это лишь первая ступень эволюции языков программирования.

Хотя это был невероятный прорыв в возможностях создания программ, недостатки этого языка всплыли мгновенно. Во-первых объем кода. Нам потребовалось написать 4 строки, чтобы сложить два числа, а если вам нужно, например, решить систему уравнений, сколько вам понадобится строк кода, чтобы сделать это? Помимо этого, у разных процессоров мог быть разный ассемблер, код очень сильно зависел от того, на каком устройстве его собирались запускать.

## 9.3 Языки высокого уровня, Fortran

Все это долго продолжаться не могло, и в 50-х годах появился первый язык, команды которого не были напрямую привязаны к

командам процессора. Такие языки стали называть языками высокого уровня. Этот язык был назван Fortran(Formula translator). Перепишем $x + y$ еще раз.

```
program calculate
implicit none
integer*4 :: x,y,z
read *, x
read *, y
z = x + y
print *,'result is  ‘,z
end program calculate
```

Мы также не призываем вас понять каждую строчку (назначение одной понимает только один из авторов данного пособия), но взгляните на нее «с высоты птичьего полета» и сравните с ассеблером. По крайней мере, в данном языке уже даже неподготовленный человек сможет найти непосредственно сложение. Чтобы писать на этом языке, нужно гораздо меньше времени на подготовку, и такой код гораздо лучше читается.

Нет ничего удивительного в том, что этот язык сразу стал безумно популярен. На нем было написано настолько много кода, что есть места, где этот язык используется до сих пор просто потому, что нет никакого смысла переписывать код на другой язык. Один из таких примеров — NASA, космические корабли которой до сих пор управляются программами, написанными на языке Fortran. Опять же, для студентов Физтеха актуально ещё и то, что на нём программируют на некоторых базовых кафедрах (особенно, связанных с физикой) и в CERNe.

Однако этот код уже не так приятен для чтения компьютером, который понимает только простейшие операции. Поэтому для кода, написанного на языках высокого уровня, требуется вспомогательная программа, *компилятор*, которая переводит код, написанный человеком в код, понятный компьютеру.

На волне успеха появилось множество похожих языков, но, в основном, они быстро канули в лету.

## 9.4 Процедурные (структурные) языки

Сложность программ стремительно росла, как и их количество, появилось желание использовать одновременно несколько программ. Требовались новые подходы к организации программ и управлению ими. Спасибо человеку по фамилии Дейкстра (который уже упоминался в разделе, связанном с графами), который считается одним из идейных основателей процедурного языка. Физические ничего не поменялось, процессоры как умели складывать числа и писать их в память, так больше ничего особо и не умеют, но изменился сам взгляд на программы. Отныне мы говорим, что есть данные, которые во главе всего и есть методы (или процедуры), которые эти данные обрабатывают.

Эта концепция очень ярко отражена в языке *C* (Си) (который, кстати, изучается в МФТИ), в котором появилась возможность объединять разнородные данные в структуры и работать с ними как с единым целым. Выглядит это примерно так:

```
struct Point{
     int x;
     int y;
};

Point move_point(Point point ,int delta_x, int delta_y){
     point.x = point.x + delta_x;
     point.y = point.y + delta_y;
}
```

Опять же мы не просим полностью понимать код, этому будет посвящен ваш первый (а, возможно, и не только первый) курс в МФТИ. Главное, обратите внимание, что в нём есть разделение данных, в данном случае структуры Point (точка) с координатами $x$ и $y$, и кода, который работает с этими данными; в данном случае функция $move\_point$ двигает точку на $delta\_x$, $delta\_y$.

Новый язык позволял строить программы с более сложной организацией, нежели предыдущие языки.

Помимо этого было большое желание использовать одновременно много программ, но при этом им придется делить между собой один компьютер, иначе говоря, потребовались программы, которые могли бы распределять ресурсы компьютера между другими программами, эти программы называются операционными системами (всем нам знакомые *Windows*, *Linux*, *MacOS*).

Все это обернулось в 70х-80х годах тем, что большая часть современных операционных систем написана на языке *C*. Помимо этого, на *C* написано большое количество программ, так как *C* тесно связан с операционными системами, которые на нем написаны.

Казалось бы, наступило счастье, есть удобный язык, на котором можно писать программы, есть операционные системы, где их можно запускать, что еще надо? До этого языки стремились увеличить свои возможности, в языке *C* вы можете буквально все, вы можете хоть напрямую писать в память, требовать любые ресурсы компьютера, но у всего этого есть обратная сторона — огромнейший простор для ошибок.

Представьте себе, что вам на ступню сел комар, а у вас в руках есть только ружье, ваша цель — убить комара и постараться не прострелить себе при этом ногу... Для большинства прикладных задач *C* — стрельба по комарам из ружья (да и вообще для большинства задач вам хватит только пакетов прикладных программ). Но уметь стрелять хоть как-то нам точно придётся научиться.

# 10 Язык *C*

Ваша задача сейчас — получить самые базовые навыки программирования на *C*. Самое главное — это преодолеть пропасть между «Я никогда не писал ни одной программы» и «Ура, оно работает». Возможно вы уже писали что-то на паскале или бейсике, тогда ваша первая цель — понять различия и привыкнуть к новому языку.

Давайте напишем первую программу.

```
#include <stdio.h>

int main() {
 printf("Hello, MIPT");
 return 0;
}
```

Взглянем поподробнее. Этот минимальный набор кода показывает структуру любой программы, написанной на *C*. Первой строкой вы видите странную команду *#include <stdio.h>* — она нужна лишь для того, чтобы программа могла считывать и писать данные, смело копируйте ее в любую программу (главное, в начало).

Затем идет *int main()* и фигурные скобки, так мы говорим системе, откуда начинается наша программа, также смело копируем. После этого идет команда *printf()*. Эта команда нужна для того, чтобы вывести информацию на экран. В простейшем случае, мы можем вывести строку, для этого в скобках пишем эту строку в двойных кавычках. Сейчас в своей голове вы должны заменить этот *printf(...)* на *«здесь будет мой код»*. После этого идёт команда *return 0*, она нужна, чтобы сообщить системе о завершении работы программы с кодом 0, то есть без ошибок. Копируем и ставим данную строку прямо перед закрывающей фигурной скобкой.

## 10.1 Сложение двух чисел

Теперь попробуем написать простую программу, которая будет складывать два числа. Для этого нам нужно будет сделать две важные вещи. Вначале необходимо считать два числа с клавиатуры, а затем — куда-то их сохранить перед тем, как мы сможем с ними что-то сделать. Для того, чтобы иметь возможность записать что-то в память, эту память нужно у компьютера попросить.

Однако компьютер с разными данными работает по-разному, поэтому надо точно сказать, что мы будем использовать. А что мы можем просить? На самом деле, много чего, но сейчас остановимся на целых числах. Для этого в нашем коде мы можем написать

*int number;*

«int» означает, что это будет целое число, а «number» — это имя для этого участка памяти. Имена нужны, чтобы различать разные участки памяти и понимать, кого же мы всё-таки зовём.

Чтобы считать целое число с клавиатуры, нужно написать сложную на первый взгляд команду:

```
scanf(“%d”, &number);
```

Разберёмся поподробнее. Можно разделить данное выражение мысленно на 3 части.

1. *scanf()* — это команда языка *C*, которая предназначена для считывания данных с клавиатуры.

2. дальше необходимо сообщить, что именно мы будем считывать. Если мы хотим ввести целое число, то нужно указать «*%d*».

3. через *&* мы должны сказать, в какой участок памяти сохранить введенное значение, тут и пригодятся имена переменных.

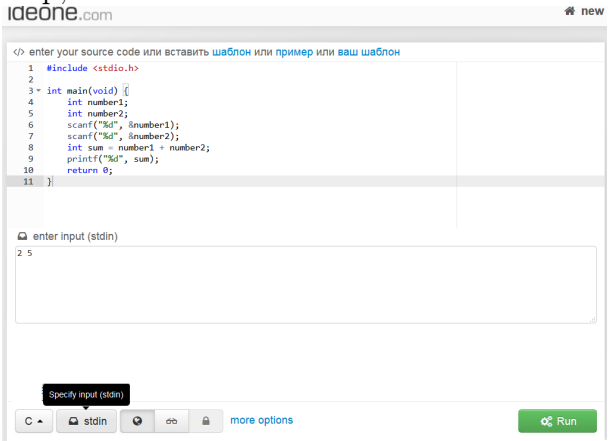
Теперь мы готовы к написанию первой серьезной программы.

```
#include <stdio.h>

int main(void) {
    int number1;
    int number2;
    scanf("%d", &number1);
    scanf("%d", &number2);
    int sum = number1 + number2;
    printf("%d", sum);
    return 0;
}
```

Единственный нюанс, чтобы напечатать число, нужно использовать *printf* по аналогии со *scanf*, но без знака *&*.

Как это запустить? Пока попробуем ничего не устанавливать и воспользоваться так называемым онлайн-компилятором. Например, ideone.com




Вводим наши числа, нажав на кнопочку «stdin», выбираем язык программирования «*C*» и нажимаем на «Run».

Не забудьте попробовать ввести не числа!

Ну а теперь поздравляем: вы только что написали первую программу и даже сумели её запустить.

## 10.2 Условные операторы

Почувствовали себя большими и сильными? А если вы стали большими, то у вас должно появиться логичное желание, не дать маленьким школьникам воспользоваться вашей программой, но разрешить остальным. Это значит, что ваша программа будет работать по-разному в зависимости от каких-то условий.

Давайте, например, запретим пользоваться программой людям, которым меньше 18-ти лет, будем считать, что они недостойны этого. Но для этого нам понадобятся условные операторы. В любом месте программы вы можете написать

```
if (условие) {
     много кода...
} else {
     еще больше кода...
}
```

Есть догадки, как это работает? Сначала идет *if (условие)*, где мы говорим, от чего будет зависеть дальнейший ход программы. Затем идут две пары фигурных скобок перед и после *else*. Первый блок выполнится, если условие истинно, а второе — если ложно.

Поподробнее поговорим про условия, которые можно использовать. Например, вы можете использовать обычные операции сравнения чисел: больше *(>)*, меньше *(<)*, равны *(==)*, не равны *(!=)*, больше или равно *(>=)*, и т. д.. Заметьте, что равенство обозначается двумя знаками равно, так как одиночное равно уже занято для присвоения переменной значения. Выглядеть это будет примерно так

```
if (5 > 3){
     выполнится код здесь
} else {
     а этот код не выполнится
}
```

Также можно спокойно использовать переменные, созданные ранее. В разделе 7 вас уже познакомили с булевой алгеброй. В *C* базовые логические операции выглядят как *&&*(И), *||*(ИЛИ), *!*(НЕ).

Например, выражение $1 > 2$ *||* $2 < 3$ истинно, а выражение $1 == 2$ *&&* $1 != 2$ ложно.

Куда записывать результат логических операций? В *C* истина и ложь рассматриваются только как то, что число 0 компьютер воспринимает как ложь, а все остальные числа как истину. Т.е. можно написать код вида

```
if(10) {
     этот код выполнится всегда
} else {
      этот код не выполнится никогда
}
```

Теперь мы готовы к тому чтобы написать ещё одну программу.

```
#include <stdio.h>

int main(void) {
    int age;
    printf("Сколько вам лет?\n");
    scanf("%d", &age);
    if (age < 18) {
         printf("Go away!\n");
    } else {
         printf("Welcome! \n");
    }
    return 0;
}
```

Собственно все не так страшно, как могло показаться. Единственное, мы добавили $\backslash n$ в сообщения для пользователя. Это специальный символ конца строки. Когда сообщение будет выведено на экран, $\backslash n$ на экране не появится, но сообщение будет

выведено так, как будто на месте $\backslash n$ компьютер нажал *«enter»* и перевел курсор на новую строку.

## 10.3 Циклы

Здесь же собрались математики? Срочно найдите сумму первых 100 натуральных чисел (1, 2, ... , 100). Что вы сделаете? Очевидно,

$$Sum = \frac{a_1 + a_n}{2} \cdot n = \frac{1 + 100}{2} \cdot 100 = 5050$$

Пока все хорошо, а что если мы прикажем найти сумму первых 100 *четных* натуральных чисел? Уже немного сложнее, но совсем чуть-чуть и для студента физтеха это, надеемся, не является проблемой. А если сумму первых ста простых чисел? А если вы захотите найти сумму первых ста четных простых чисел, то у нас для вас плохие новости...

Запомните! Многие программисты не знают формул и боятся любого действия, которое нужно сделать на листке бумаги, похлеще, чем преподаватели физтеха боятся святой воды и солнечного света.

Давайте зададимся вопросом, как, например, вывести на экран первые 5 натуральных чисел? Очевидно же, это мы уже умеем!

```
printf("1''); //printf(``%d'', 1) эквивалентно
printf("2");
printf("3");
printf("4");
printf("5");
```

Хорошо, а 10? Надеемся, что даже если вас еще не затрясло от бессмысленности происходящего, то хотя бы напрягло. Для решения задач по запуску одного и того же кода множество раз подряд существует такая вещь, как циклы. Для начала познакомимся с циклом *while*. Он имеет следующий вид:

```
while (условие){
     код
}
```

На самом деле, всё, что он делает — запускает код внутри фигурных скобок, пока условие внутри круглых скобок истинно. Давайте попробуем решить предыдущую задачу:

```
int i = 1;
while (i <= 5){
     printf(“%d”, i);
     i = i + 1;
}
```

Результат выполнения программы: 12345

Кажется, всё стало немного адекватнее. Не бойтесь сделать бесконечный цикл. Но если вдруг условие будет *всегда* истинным, то ваша программа, скорее всего, станет бесполезным куском кода. Например, если в предыдущей задаче забыть увеличить $i$ на единицу, то все будет печально и программа будет бесконечно выводить 1 на консоль!

```
int i = 1;
while (i <= 5){
     printf(“%d”, i);
}
```

Результат: 111111111111111....

Вот теперь вы можете найти сумму натуральных чисел как настоящий программист!

```
#include <stdio.h>

int main(void) {
    int sum = 0;      //переменная для хранения суммы
    int i = 1;        //переменная, в которой хранится текущее
     //число
    while (i <= 100){ //цикл продолжается, пока текущее число
```

```
   // меньше или равно 100
        sum = sum + i;//прибавляем текущее число к сумме
        i = i + 1;    //увеличиваем число на единицу
    }
    printf("%d", sum);
    return 0;
}
```

Результат: 5050

Великолепно, и нам не понадобились канцелярские принадлежности. А если только четные числа?

```
#include <stdio.h>

int main(void) {
    int sum = 0;
    int i = 1;              //теперь здесь количество чисел,
    //которые мы прибавили к сумме
    int current_number = 2;// а здесь текущее число
    while (i <= 100 {       //цикл будет продолжаться, пока мы
                            // не прибавим к сумме 100 чисел
        sum = sum + current_number;
        i = i + 1;
        current_number = current_number + 2;
             //увеличиваем текущее число на 2, чтобы
             //оно оставалось четным
    }
    printf("%d", sum);
    return 0;
}
```

Результат: 10100

Помимо цикла *while*, есть цикл *for*, но мы отложим его изучение для другой темы, в которой польза от его применения будет более наглядной.

## 10.4 Функции

Пришло время поговорить о проблемах того кода, который мы писали. Да, как бы красив он ни был, он не идеален. Основная его проблема в том, что весь наш код был внутри фигурных скобок после *main()*, это плохо тем, что если количество кода будет расти, то за ним будет сложнее следить. Возьмем простую задачу — найти максимум из двух чисел, считанных с клавиатуры. Наверное вы можете решить её как-то так:

```
scanf("%d", &a);
scanf("%d", &b);
int max = a;
if (b > max){
    max = b;
}
printf("%d", max);
```

Вроде бы, всё кажется пока адекватным. А что, если нужно найти максимум из трех чисел? Хорошо, поехали:

```
int a;
int b;
int c;
scanf("%d", &a);
scanf("%d", &b);
scanf("%d", &c);
int max = a;
if (b > max){
    max = b;
}
if (c > max){
    max = c;
}
printf("%d", max);
```

Видите, что происходит? Мы делаем одно и то же несколько раз. Величайшее правило программирования: если вы видите

одинаковый код, то выносите его в отдельную функцию! Осталось лишь понять, как это сделать, и сделаем мы это на примере функции максимума двух чисел. Взглянем на этот код:

```
#include <stdio.h>

int max(int a, int b){
    if (a > b){
        return a;
    } else {
        return b;
    }
}

int main(void) {
    int n = max(10, 20);
    printf("%d", n);
    return 0;
}
```

Результат: 20

Прежде всего обратите внимание, что появился код не внутри фигурных скобок *main()*, а перед ними. По своей сути функция — это блок кода, который можно многократно переиспользовать. Функция объявляется следующим образом: сначала идет тип данных, которые вернет функция, в нашем случае это *int*, затем название функции — имя, которые вы придумываете сами, которое должно отражать назначение функции (в нашем случае это *max*), затем в круглых скобок аргументы функции — данные, на основе которых будет вычисляться результат функции *(int a, int b)* в нашем случае. Затем в фигурных скобках код функции.

Чтобы функция закончила свою работу и «вернула» результат, используется ключевое слово *return*, после которого идет что-то того типа данных, который указан в начале функции.

Не замечаете нечто странное?

```
int main(){
```

```
    return 0;
}
```

*main* — это тоже функция, её отличие от всех остальных лишь в том, что с нее программа всегда начинается и ей всегда заканчивается.

Если вы не хотите, чтобы функция вам что-либо возвращала, то нужно указать вместо типа возвращаемых данных слово *void*. Например, если мы пишем функцию, которая просто печатает на экран числа.

```
void print_number(int a){
    printf(“%d”, a);
}
```

Слово *return* в этом случае не требуется, однако оно обязательно, если функция не *void*.

А как найти максимум трех чисел? Напишем для этих целей еще одну функцию:

```
#include <stdio.h>

int max(int a, int b){
    if (a > b){
        return a;
    } else {
        return b;
    }
}

int max3(int a, int b, int c){
    int result = max(a, b);
    result = max(result, c);
    return result;
}

int main(void) {
    int n = max3(10, 40, 20);
```

```
    printf("%d", n);
    return 0;
}
```

Результат: 40

Как видим, ничто не мешает нам использовать наши функции внутри других функций, компьютер достаточно умён, чтобы найти используемые функции и вызвать их.

## 10.5 Объектно-ориентированное программирование

Пожалуй одной из самых продуктивных идей стала идея объектно-ориентированного программирования (ООП). Идеи ООП возникли в начале 80-х и моментально стали объектом поклонения огромного числа разработчиков. Мы говорили про структурный подход, где разделили данные и код, это, конечно, хорошо, но обнаружилось, что с ростом программ появляется путаница, какой именно код какие именно данные обрабатывает.

Т.е. в большинстве случаев это приводит к тому, что образовывается две большие кучи — куча данных и куча кода и непонятно, что с чем совмещать. Поэтому умные люди предложили объединять данные и код, который их обрабатывает, в отдельные сущности и назвали их классами, т.е. классы стали такими небольшими островками, на которых есть данные и код, который обрабатывает их и только их.

Большое распространение ООП получило с языком *C++*. Что-то знакомое... *C* и *C++*, не находите? *C++* это надстройка над *C*, которая реализует объектно-ориентированные идеи. И самое главное, он обратно совместим с *C*, т.е. весь код, который был написан до этого на *C* (а это оо-очень много кода) работает в *C++*. Неудивительно, что язык стал популярен.

Взглянем, как *C++* выглядит на деле:

```
class Cat {
private:
```

```
    int angry;              // уровень злости
public:
    void stroke(){          //гладить
        angry = angry - 1;
    }
};
```

В МФТИ *C++* подробно проходится на втором курсе и в нем несколько больше деталей, чем умещается в этом примере, здесь мы хотим только обратить внимание на то, как менялся взгляд людей на программы, которые они пишут. В этом примере вы можете увидеть, как объединяется код и данные. Класс *кот* имеет данные в виде уровня злости и это данные приватные, т.е. никакой код снаружи не может к ним достучаться. Зато есть публичный метод погладить, который делает котика на единичку менее злым.

Таким образом мы можем еще лучше структурировать код, однако возможности выстрелить себе в ногу все еще предостаточно. Так или иначе, эти языки очень тесно сплетены с операционными системами и очень эффективны. Однако вы не замечаете, что языки уходят все дальше и дальше от процессора, который умеет только простейшие вещи? Языки обрастают кучей мишуры и как следствие усложняется способ донести код до процессора, давайте немного поговорим об этом.

## 10.6 Способ выполнения

Как уже говорилось, *C* и *C++* — компилируемые языки. Это значит, что для того, чтобы код заработал, его нужно перевести на язык, понятный машине. Этим занимается *компилятор*, он берет файл с кодом и создает новый файл, который может исполнять процессор. Но вот в чем вопрос: если вы создали исполняемый файл, который работает на вашем компьютере, заработает ли он на другом?

В общем случае нет; если вы хотите запустить код например и на Windows, и на Linux, то вам придется создать разные ис-

полняемые файлы, а это значит, что для каждой операционной системы нужен свой компилятор... а создать компилятор, это немного не то же самое, что написать складыватель двух чисел. А если вы используете в коде какие-либо особенности операционной системы, то тогда ваш код не запустится на другой ОС в принципе и вам придется переписывать код под другую систему, учитывая ее особенности.

Так человечество пришло к естественному желанию не переписывать код под каждую операционную систему. (* от этого желания так и прет естественностью, ну да ладно *) Именно с таким намерением в двухтысячные вошла компания Sun Microsystems (ныне Oracle) и с лозунгом «Да запустится везде однажды написанное» («Write once, run anywhere») выпустила язык java. И вкусили люди этот плод солнца микросистемного и стали программы медленными.

Лирика — лирикой, но сделали они действительно неплохую штуку. Основная идея довольно проста. Давайте придумаем некоторый абстрактный процессор и будем компилировать весь код для него, этот процессор всегда одинаковый. А затем напишем небольшие (в сравнении с размерами компиляторов) программы, которые будут переводить команды этого процессора в команды реального процессора. А теперь выдохните!

Если это было слишком сложно для понимания, ничего страшного. Главное что вы должны понять, что люди придумали возможность запускать однажды написанную программу на *любом* компьютере. Но и цена была большой, прежде всего производительность, теперь у вас есть код, который не может непосредственно выполниться на компьютере, его нужно из команд абстрактного процессора переводить в команды реального. И называется все это JIT (Just in time) компиляцией или компиляцией во время исполнения.

И есть еще третий основной подход к выполнению программ, это полностью интерпретируемые языки, который на самом деле схож со вторым по принципу работы, но может переводиться в код, понятный компьютеру прямо построчно, написали строч-

ку, она выполнилась, написали вторую, она тоже выполнилась; такие языки иногда называют *скриптовыми*, на них удобно писать *скрипты*, которые тут же и выполняются без всяких размышлений о компиляции и т. п. вещах. По такому пути пошел, например, python, который также изучается на некоторых факультетах в курсе информатики.

Возможно, для кого-то эта глава была достаточно тяжёлой. Главное, что вы должны запомнить, что есть два основных подхода к выполнению программ: компиляция, когда ваша программа состоит из команд реального процессора и для запуска которой кроме нее самой ничего не требуется, и интерпретация, когда программа находится в каком-то промежуточном представлении и для её выполнения её должна запустить другая программа.

# Часть VI
# Базы данных

## 11 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 легендарным Кириллом Борисовичем Теймуразовым (Теймур, КБТ, как его часто называют за глаза) состоится очень скоро.

### Главный по базам данных

В отличие от других предметов, в данном, из-за малого количества преподавателей, можно рассказать о них поподробнее.

На Физтехе много легендарных преподавателей, например, наверняка вы все уже слышали про Дмитрия Владимировича Беклемишева. В рамках ФУПМа конкуренцию ему может составить человек, с которым вы точно столкнётесь в рамках предметов кафедры МОУ. Как в старых советских лозунгах Ленин и партия неразделимы, так и сложность БД невозможно оценить, не узнав ответственного за курс, Кирилла Борисовича.

Первое: Теймуразов — очень умный человек. Все предметы, которые ведёт — БД, ТРЯП, АМВ — он знает очень хорошо, «шарит». Расшифровку этих страшных аббревиатур вы узнаете через год. Помимо всего прочего, он является президентом Долгопрудненского клуба интеллектуальных игр. Если вам повезло попасть к нему на сдаче, скучать не придётся: вопросы задаёт

очень интересные (но сложные).

Второе: КБТ крайне принципиален. И строго подходит к соблюдению правил (установленных, естественно, им же). Немало людей оценили это, что называется, на своей шкуре. Причём его отношение к студенту будет тем более суровым, чем меньше трудолюбия проявлял этот студент.

Третье: Кирилл Борисович ленив. Например, это проявляется в нежелании принимать лабы даже через минуту после звонка. Ну и манера общения при проверке у него специфическая. Иногда кажется, что он спит и не слушает, но любую ошибку замечает на раз; на просмотр кода уходит несколько секунд — и опять же, ошибка от него не ускользнёт. Впрочем, эти особенности следуют из первого пункта.

Опыт показывает следующее: как Теймуразов решил, так и будет. Поэтому не следует его злить плохими знаниями или «забиванием» на предмет!

## Основные принципы

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:

http://docs.google.com/document/d/
1ev6i–qUeILCU6s_6VVpIy2Y9lp8NcQ52_cqk-kwXkI

Большинство вопросов к сдачам Кирилл Борисович и его коллеги берут с

http://spprac.cs.msu.ru/files/dbprac.pdf.

Первое занятие — ознакомительное, на нём КБТ рассказывает 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 Кирилла Борисовича 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. А вот как учитывать посещаемость, Теймуразов решает в самый последний момент, причём в разные дни выставления зачёта принцип может быть разным.

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.

За май 2016 года Теймуразов попробовал разные схемы. В один из дней он предложил пришедшим угадать количество посещённых ими лекций (это было сложно, учитывая тот факт, что иногда студенты записываются от имени проспавших друзей). Тем, кому повезло, подарили +1 к зачёту.

В остальное время от количества посещённых лекций отнимали 8. Если сданы все лабораторные и результат положительный, его прибавляли к общему числу баллов. Если же оба условия нарушены, то... тоже прибавляли данное, на этот раз отрицательное, число. Кто-то из-за таких сюрпризов получил более высокую оценку, а кто-то получил кирзовые сапоги...

Эти неожиданности подсказывают, что 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 Теймуразов разрешает брать дополнительные задания. Их сложность зависит от количества недостающих баллов, но обычно выше, чем у основных лабораторных. Однако все, кто хотел, их выполнили.

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!

# Часть VII
# Заключение

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

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


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


## Список литературы


[1] Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. (2009). Алгоритмы. Построение и анализ:[пер. с англ.]. Издательский дом Вильямс.

[2] Бабичева Т.С., Бабичев С.Л. Введение в теорию игр и исследование операций. Материалы Летней Олимпиадной Школы. — Паблит, 2016.

[3] Беклемишев Д. В. Курс аналитической геометрии и линейной алгебры: Учеб. для вузов — 11-е изд., испр. — М:ФИЗМАТЛИТ, 2006.

[4] Тер-Крикоров А. М., Шабунин М. И. Курс математического анализа //М.: Физико-математическая литература, 2001.

[5] 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» Курс разработан на кафедре высшей математики МФТИ, 2014.

[6] Гладуш М.Г., Гуденко А.В., Извекова Ю.Н., Кузьмичев С.Д., Максимычев А.В., Попов П.В., Филатов Ю.Н. Модели и концепции физики: Механика. Лабораторный практикум. Обработка результатов измерений. — М.:МФТИ, 2011.

[7] ВМК МГУ. Практикум по Базам Данных, методические материалы, 2012. Электронный ресурс http://spprac.cs.msu.ru/files/dbprac.pdf

[8] Журавлев Ю.И., Флеров Ю.А. Дискретный анализ. Ч. 1: Учебное пособие. — М.: МФТИ, 1999.

[9] Сканави М. И. Элементарная математика. — 2-е изд., перераб. и доп. — М.: Наука, 1972.

Подписано в печать 21.08.2016.
Формат 60 × 84/16. Усл. печ. л. 13,49. Тираж 200 экз.
Федеральное государственное автономное образовательное учреждение высшего образования «Московский Физико-Технический Институт (Государственный Университет)»
141700, Московская обл., г.Долгопрудный, Институтский пер., 9
Отпечатано в типографии ООО «Паблит» 127282, Москва, ул. Полярная, д.31В, стр.1
Тел.:(495)685-93-18