

Лекции К.И.Костенко, но нормально

SilentObserver

Оглавление

1	Общие вещи	6
1.1	Предикаты	6
1.2	Отображения	7
1.3	Мощность множества	8
1.4	Бинарные отношения на множествах	10
1.5	Отношение эквивалентности	11
1.6	Отношения порядка	14
2	Комбинаторика	16
2.1	Основные комбинаторные правила	16
2.1.1	Правило птичьих гнезд	16
2.1.2	Правило умножения	16
2.1.3	Правило сложения	17
2.2	Сочетания и размещения	17
2.2.1	Размещения	17
2.2.2	Сочетания	18
2.3	Разбиение множества на части	19
2.4	Формула включений-исключений	20
3	Функции алгебры логики	22
3.1	Функции	22
3.2	Формулы	24
3.3	Разложение функций по переменным	25
3.4	Минимальные ДНФ	27
3.5	Геометрическая интерпретация ДНФ	28
3.6	Максимальные конъюнкции	30
3.7	Полные системы ФАЛ	30
3.8	Полиномы Жегалкина	32
3.9	Классы T_0 и T_1	32
3.10	Двойственные функции	34
3.11	Класс S	36
3.12	Класс M	37
3.13	Класс L	39
3.14	Критерий полноты в P_2	41

3.15	Предполные классы ФАЛ	43
3.16	Схемы из функциональных элементов	44
4	Графы	46
4.1	Определение и способы представления	46
4.2	Пути и циклы в графах	48
4.3	Транзитивное замыкание графа	50
4.4	Геометрическая реализация графа	52
4.5	Критерий планарности графа	55
4.6	Деревья	56
4.7	Циклы Эйлера	59
4.8	Циклы Гамильтона	61
4.9	Суммы графов	62
4.10	Фундаментальное семейство циклов	63
4.11	Хроматические графы	65
4.12	Внутренняя и внешняя устойчивость графов	67
5	Зачетные задачи	69
5.1	Задача 1	69
5.2	Задача 2	72
5.3	Задача 3	78
5.4	Задача 4	81
5.5	Задача 5	85
5.6	Задача 6	99
5.7	Задача 7	102
6	Конечные автоматы	109
6.1	Определения	109
6.2	Числовые функции, вычисляемые автоматами	111
6.3	Переработка автоматами периодических сверхслов	114
6.4	Отличимость состояний автомата	115
6.5	Минимизация автомата	118
6.6	Распознавание слов конечным автоматом	123
6.7	Операции над конечными автоматами	125
6.8	Структурный автомат	127
7	Рекурсивные функции	129
7.1	Классы рекурсивных функций	129
7.2	Тезис Чёрча	134
7.3	Деревья определения рекурсивных функций	134
7.4	Неразрешимые свойства рекурсивных функций	138
7.4.1	Проблема остановки	138
7.4.2	Проблема всюду определенности	141
7.4.3	Проблема эквивалентности	142

8	Вычислительная сложность алгоритмов	144
8.1	Сложность алгоритмов	144
8.2	Быстрая сортировка	147
9	Системы Поста	149
9.1	Определения	149
9.2	Задачи, решаемые системами Поста	152
9.3	Свойства слов, выводимых системами Поста	155
9.4	Функции, вычисляемые системами Поста	158
10	Транспортные сети	164
11	Зачетные задачи	169
11.1	Задача 1	169
11.2	Задача 2	171
11.3	Задача 3	182
11.4	Задача 4	192
11.5	Задача 5	200
11.6	Задача 6	215
11.7	Задача 7	237
11.8	Задача 8	242
11.9	Бонусный раздел, построение систем Поста	245
11.10	Задачи 9-10	258

Введение

Эта книга — собрание моих объяснений лекций нашего глубокоуважаемого Константина Ивановича, созданная, чтобы не объяснять одно и то же по несколько раз. Это — не полный конспект лекций, тут не содержатся точные определения и формулировки, здесь находятся только объяснения лекций и доказательств. Хотя лекции записаны на 2019–2020 учебный год, скорее всего крупных несовпадений не будет и значительно позже. Кроме того, для еще большей понятности здесь находятся некоторые вопросы моих одноклассников, которые они мне задавали в процессе объяснения, и мои ответы на них, хотя большинство этих объяснений уже скорректированы с учетом этих вопросов.

Важно: На зачетах и экзаменах не стоит использовать «простые» объяснения отсюда. Эти объяснения правильные и помогут вам лучше понять происходящее на лекциях (что вам пригодится, поверьте), но они недостаточно формальны для того, чтобы понравиться Костенко:

«Мысль должна бегать»

— КОНСТАНТИН ИВАНОВИЧ КОСТЕНКО.

I семестр

Глава 1

Общие вещи

1.1 Предикаты

Высказывание — некое утверждение, которое или правда, или ложь

Предикат — «функция», принимает какие-то аргументы, возвращает логическое значение — или правду, или ложь. Иначе говоря, просто *высказывание*, которое утверждает что-то про конкретные объекты, а не в общем и целом.

Операции над предикатами — стандартные логические операции, конъюнкция (И, $\&$), дизъюнкция (ИЛИ, \vee), импликация (ЕСЛИ-ТО, \rightarrow), отрицание (НЕ, \bar{x}). Позволяют комбинировать предикаты в более сложные предикаты, т.е. если P и Q предикаты, то $P \& Q$ — тоже предикат.

Кванторы — символы, вводящие новые переменные, и каким то образом «перебирающие» все их значения, аналог циклов.

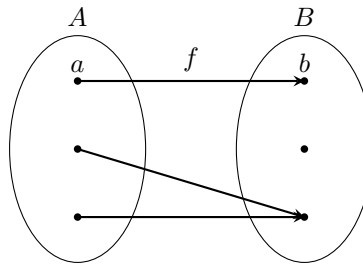
- $\forall x(P(x))$ — квантор всеобщности. Выдает истину только если предикат P верен для всех возможных значений x .
- $\exists x(P(x))$ — квантор существования. Выдает истину если существует хотя бы один такой x , для которого предикат P верен, т.е. чтобы доказать какое то условие с квантором существования, достаточно просто привести единственный пример x , для которого все работает.

Кванторы являются двойственными, т.е. $\overline{\forall x(P(x))} = \exists x(\overline{P(x)})$ и наоборот. Если расшифровать, то это значит, что «не для всех x выполняется P » — то же самое, что «существует такой x , для которого P не выполняется», что достаточно логично.

1.2 Отображения

Отображение — обычная функция из математики. Запись $f : A \rightarrow B$ означает, что отображение f преобразует элементы из множества A в элементы из множества B . Важная характеристика отображения: каждому элементу из A (далее буду называть их «входами») соответствует ровно один элемент B («выход»), два нельзя.

Нарисовать отображение можно так:



Свойства отображений

Инъективность — разные элементы A отображаются в разные элементы B , т.е. несколько стрелочек в одну точку — нельзя.

Сюръективность — в каждый элемент B что-нибудь отображается, т.е. «пустых» элементов справа быть не должно.

Биективность — инъективность + сюръективность

Обратные отображения

Примерно то же самое, что и обратные функции — просто «разворачиваем стрелки» и смотрим, какие элементы A отображаются в каждый конкретный элемент B . В лекциях вводится обозначение A_b , но это просто множество всех «входов», которые отображаются в один конкретный «выход» b . Если это множество пустое, то обратной функции «некуда идти», а если там несколько элементов, то невозможно выбрать один правильный. Поэтому обратная функция существует только тогда, когда для всех b множество A_b одноэлементно.

Теорема. Обратное отображение есть тогда и только тогда, когда отображение — биекция.

Доказательство. Такие доказательства «тогда и только тогда» рассматриваются как 2 теоремы в одной: из «1 следует 2» и из «2 следует 1». \square

1. Если обратное отображение есть, то все A_b одноэлементны. Тогда f — инъекция, потому что иначе какое-то множество было бы по меньшей

мере из 2 элементов, и f — *сюръекция*, потому что иначе какое-то множество было бы пустым. Значит f — *биекция*.

- (а) Если f — *биекция*, то она и *инъекция*, и *сюръекция*. Тогда, т.к. инъекция, то A_b не может содержать больше 1 элемента (иначе 2 стрелки в одну точку), а т.к. сюръекция, то A_b не может содержать 0 элементов, значит A_b — одноэлементно, и обратное отображение существует.

1.3 Мощность множества

Мощность множества — если говорить по простому, то это количество элементов в множестве. Для конечных множеств практически так и есть, но для бесконечных начинаются проблемы.

Эти проблемы состоят в том, что мы не можем найти количество элементов в бесконечном множестве, и «бесконечно» — не ответ. Поэтому формальное определение такое: *мощность множества* — множество всех множеств, равномоощных данному. Да, множество всех множеств. Т.е. если $A = \{1, 2, 3\}$, то мощность $|A|$ не просто 3, а множество всех трехэлементных множеств, например там есть $\{a, b, c\}$. Впрочем, для простоты мы все равно называем это 3. Но формально так.

Равномоощные множества — множества, между которыми можно провести биективное соотношение (соотношение один к одному).

Например $\{1, 2, 3\}$ и $\{a, b, c\}$ — равномоощны, потому что между ними можно провести соотношение 1 к 1:

$$f(x) = \begin{cases} a & , \text{если } 1 \\ b & , \text{если } 2 \\ c & , \text{если } 3 \end{cases}$$

И да, множество натуральных чисел называют *счетно-бесконечным*, как и все равномоощные множества, и вместе со всеми конечными их называют *счетными*.

Булеан A — множество всех подмножеств A , обозначается 2^A . Например: $A = \{1, 2, 3\}$, $2^A = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. Думаю понятно, как работает. И да, 2^A , потому что для конечных множеств количество так и меняется, заметьте, $|A| = 3$, $|2^A| = 2^3 = 8$.

Теорема Кантора. *Множества A и 2^A не равномоощны.*

Доказательство. Доказываем методом от противного, предположим, что равномоощны, тогда есть биекция.

Назовем эту волшебную функцию f (см. рисунок 1.1), и пусть она каждый элемент множества A преобразует в подмножество. Т.е. если $x \in A$,

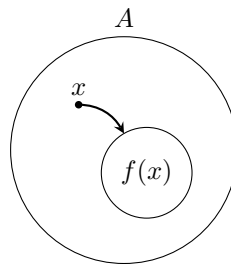


Рис. 1.1: Теорема Кантора

то $f(x) \subseteq A^1$. Тогда некоторые x могут лежать в своих $f(x)$, а некоторые могут не лежать. Возьмем все x , которые не лежат в своих подмножествах, и положим их в множество D . f у нас сюръекция, так что каждому подмножеству («выходу») должен соответствовать какой-то элемент («вход»). Тогда D тоже соответствует какой-то x . У него есть 2 варианта: \square

1. $x \in D$, тогда он должен не лежать в своем $f(x)$, но его $f(x)$ - и есть D , противоречие.

(а) $x \notin D$, тогда он должен лежать в своем $f(x)$, но его $f(x)$ - и есть D , противоречие.

Доказательство. Тогда наше предположение неверно и теорема доказана. \square

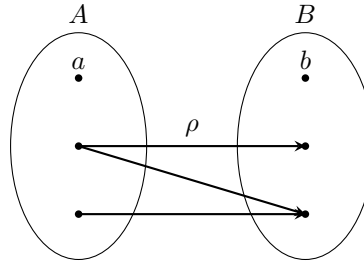
Q: А почему в конспекте указано $D = \{x | x \in A \text{ и } x \notin B_x\}$, а ты ничего про A не сказал?

A: Потому что мы в принципе работаем в множестве A , и все элементы и подмножества рассматриваем оттуда. Но формально да, нужно сказать, что D — множество всех элементов A , которые не лежат в собственном $f(x)$ (ну или B_x). Отношения

Отношение — как отображение, но элементы из A не обязаны быть связаны ровно с одним элементом B . Формально — множество пар вида (a, b) , $a \in A, b \in B$, в котором есть те пары, между элементами которых есть связи.

Рисуются аналогично отображениям:

¹В лекциях Костенко введено обозначение $f(x) = B_x$. Зачем, без понятия.



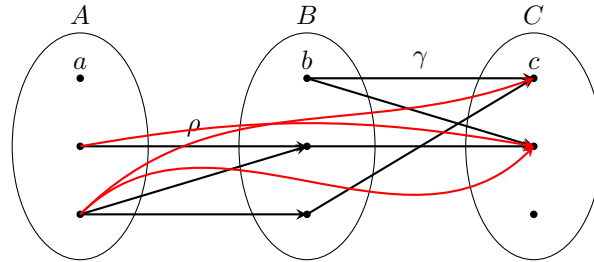
Также их можно задавать таблицей, например отношение выше будет

	b_1	b_2	b_3
a_1	0	0	0
a_2	0	1	0
a_3	0	1	1

Операции над отношениями

Операции над отношениями существует 2: обращение и умножение. Обращение очень простое — мы просто меняем местами элементы пар (и, соответственно, множества тоже), т.е. разворачиваем стрелки в обратную сторону.

Произведение отношений чуть сложнее: для него требуются отношения $\rho \subseteq (A, B), \gamma \subseteq (B, C)$, и тогда $\rho \circ \gamma \subseteq (A, C)$. Т.е. тут используется 3 множества, первое отношение связывает первые два, а второе — последние два. Тогда в произведении отношений есть связь тогда и только тогда, когда из первого множества можно «дойти» до третьего через эти отношения. Проще показать диаграмму:

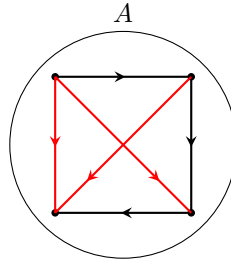


1.4 Бинарные отношения на множествах

Когда множества слева и справа в отношении — одно и то же множество, то это называют бинарным отношением на множестве. У бинарных отношений на множествах могут быть или не быть 4 свойства: рефлексивность,

симметричность, транзитивность и антисимметричность.² Если $\rho \subseteq (A, A)$, то

1. ρ — рефлексивно $\Leftrightarrow \forall x \in A(x\rho x)$. Это значит, что все элементы связаны в отношении ρ сами с собой, т.е. у всех элементов есть «петли».
2. ρ — симметрично $\Leftrightarrow \forall x, y \in A(x\rho y \rightarrow y\rho x)$. Это значит, что все связи в отношении — парные: если есть связь в одну сторону, то есть и в другую.
3. ρ — транзитивно $\Leftrightarrow \forall x, y, z \in A(x\rho y \& y\rho z \rightarrow x\rho z)$. Это значит, что все «пути» в отношении «замкнуты накоротко», т.е. если от одного элемента x можно дойти до другого по связям, то между ними есть связь и напрямую. Пример на диаграмме: если есть черные связи, то обязаны быть и красные.



Если в отношении нет «путей» длины больше 1, то оно всегда транзитивно, т.к. нет контрпримеров, когда путь есть, а между его концами нет связи напрямую.

4. ρ — антисимметрично $\Leftrightarrow \forall x, y \in A(x\rho y \& y\rho x \rightarrow x = y)$. Это значит, что в отношении нет парных связей (не считая петель). Антисимметричность — не то же самое, что несимметричность, отношение может быть одновременно симметрично и антисимметрично, если в нем есть только петли.

1.5 Отношение эквивалентности

Отношение эквивалентности — отношение, которое рефлексивно, симметрично и транзитивно.

Такие отношения называются отношения эквивалентности, потому что с точки зрения этих отношений между элементами есть связь тогда и только тогда, когда они в каком то смысле «эквивалентны». Например, числа,

²Определения этих свойств — пожалуй, самые часто используемые определения в 1 семестре, так что их важно запомнить.

у которых последняя цифра одинаковая, или люди, которые живут в одном городе, или просто значения некоторой абстрактной функции для этих объектов равны³.

Разбиение множества A — набор множеств, которые

1. не пустые
2. не пересекаются
3. вместе образуют все множество A .

Теорема. *Любое отношение эквивалентности образует разбиение множества A на классы эквивалентных элементов.*

Доказательство. Для начала, что нам нужно:

1. Придумать набор множеств, который будет разбиением.
2. Доказать, что он — разбиение.
3. Доказать, что все элементы внутри одного класса эквивалентны, т.е. связаны отношением.
4. Доказать, что все элементы из разных классов не эквивалентны, т.е. не связаны.

1. Придумаем это семейство множеств.

Для каждого элемента x множества A сделаем множество $[x]$. Это множество всех y , которые с ним связаны: $[x] = \{y \mid xry\}$. Раз ρ у нас отношение эквивалентности, то оно рефлексивно и выполняется xrx . Тогда $x \in [x]$, т.е. x лежит в своем собственном множестве.

Итак, набор множеств придумали, давайте посмотреть, что нам нужно для разбиения:

- (a) Все классы непустые (есть, мы доказали, что хотя бы один элемент (x) есть).
- (b) Классы не пересекаются.
- (c) В объединении все классы дают изначальное множество.

Раз у нас каждый x лежит хотя бы в одном множестве (собственно $[x]$), то ни один из них не теряется, и все $[x]$ в объединении дадут изначальное множество A .

Остался только 2 пункт. Проблема в том, что у нас множества могут по много раз повторяться, так что мы просто выкинем повторы и назовем семейство классов F . 1-й пункт от этого не теряется, 3-й тоже, потому что мы выкидываем только повторы, так что это и будет наше разбиение: просто множество всех $[x]$, откуда выкинули все повторы.

³Да, это отсылка к 1 зачетной задаче.

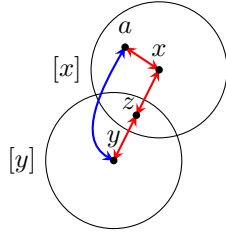


Рис. 1.2: 2-й пункт доказательства

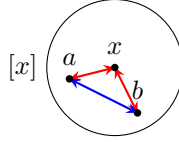


Рис. 1.3: 3-й пункт доказательства

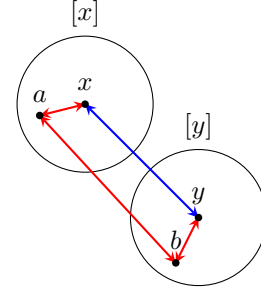


Рис. 1.4: 4-й пункт доказательства

2. Докажем, что это семейство — разбиение.

1-й и 3-й пункты уже доказаны, остался 2-й пункт: множества не должны пересекаться.

Будем доказывать от противного: предположим, что есть какие то 2 класса, которые пересекаются. Пусть это будут $[x]$ и $[y]$, и они разные, $[x] \neq [y]$. Раз они пересекаются, то есть хотя бы один общий элемент, назовем его z , $z \in [x] \cap [y]$ (см. рисунок 1.2).

Теперь смотрим, что это значит. Если z лежит в $[x]$, то значит $x\rho z$ (мы так классы определяли). Еще $y\rho z$, потому что z в $[y]$. На самом деле эти классы будут совпадать, $[x] = [y]$. Это то же самое, что $[x] \subseteq [y]$ и $[y] \subseteq [x]$. Чтобы доказать первое, нужно рассмотреть случайное $a \in [x]$. Тогда $x\rho a$. У нас ρ — отношение эквивалентности, так что оно симметрично и транзитивно, т.е. $a\rho x$ есть, $x\rho z$, $z\rho y$, и тогда по транзитивности $a\rho y$, т.е. $a \in [y]$. Это означает, что $[x] \subseteq [y]$, и аналогично можно доказать $[y] \subseteq [x]$. Это значит, что классы $[x]$ и $[y]$ совпадают, а мы в шаге 1 выкинули все повторы, так что получили противоречие. Значит наше F — и правда разбиение.

3. Элементы внутри классов связаны.

Доказательство аналогично 2-му пункту: Возьмем $a, b \in [x]$. Это значит, что $x\rho a$ и $x\rho b$. Из-за симметричности $a\rho x$, и из-за транзитивности $a\rho b$, что и требовалось доказать (см. рисунок 1.3).

4. Элементы в разных классах не связаны.

Предположим, что связаны. Если так, то имеем $a \in [x], b \in [y], [x] \neq [y], a\rho b$, т.е. $x\rho a, y\rho b, a\rho b$. Из-за симметричности $x\rho a, a\rho b, b\rho y$ и из-за транзитивности $x\rho y$ (см. рисунок 1.4), но тогда $x \in [y]$ и в то же время $x \in [x]$, но т.к. классы не пересекались, то это невозможно, значит элементы в разных классах и правда не связаны.

□

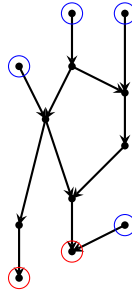


Рис. 1.5: Отношение порядка (красные элементы — минимальные, синие — максимальные)

1.6 Отношения порядка

Отношение порядка — отношение, которое рефлексивно, транзитивно и антисимметрично.

Так же, как отношение эквивалентности похоже на $a = b$, отношение порядка похоже на $a \geq b$, ключевое отличие только в том, что могут существовать пары несравнимых элементов, для которых неверно и $a \geq b$, и $b \geq a$. Среди обычных чисел такого не бывает, но среди каких то других элементов это вполне возможно.

Мы можем рисовать диаграммы для подобных отношений, но полные диаграммы были бы слишком запутанными, поэтому мы просто не рисуем некоторые связи, а именно петли и «транзитивные дуги», т.е. те, которые потом восстанавливаются по транзитивности. Пример диаграммы на рисунке 1.5.

Также существуют 4 вида «особенных» для этого отношения элементов:

1. Наибольший элемент — тот, который больше всех. Т.е. его можно сравнить со всеми и он будет больше. Такой элемент может быть только 1.
2. Наименьший элемент — тот, который меньше всех. Тоже может быть только 1.
3. Максимальный элемент — тот, больше которого не существует. Таких элементов может быть несколько за счет того, что их нельзя сравнивать между собой. Даже просто одиночно стоящий элемент, который нельзя ни с кем сравнивать — максимальный. Если такой элемент только 1, то он и наибольший. *Т.к. сравнение в отношении порядка — нестрогое, то этот элемент может быть «больше или равен» самому себе, поэтому определение именно такое, какое есть: $\forall b(bra \rightarrow a = b)$.*
4. Минимальный элемент — тот, меньше которого не существует. Аналогично максимальному.

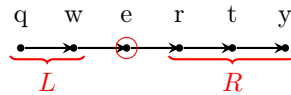


Рис. 1.6: Отношение линейного порядка

Отношения линейного порядка

Отношение линейного порядка — такое отношение порядка, где все элементы можно сравнивать, т.е. в каждой паре один элемент больше другого.

Теорема. *Диаграмма отношения линейного порядка — прямая.*

Доказательство проще разобрать на примере: пусть $A = \{e, q, r, t, w, y\}$ (отсортировано по алфавиту) и отношение линейного порядка таково, что $q < w < e < r < t < y$, т.е. то, которое правее на клавиатуре — больше. Возьмем «первый» элемент из множества A (он не обязан быть первым для нашего отношения, он может быть просто случайный), $a_0 = e$. Сравним этот элемент со всеми остальными и сформируем 2 множества: L и R^4 , где все элементы множества L меньше a_0 , а все элементы R — больше. Тогда $L = \{q, w\}$, $R = \{r, t, y\}$. По индуктивному предположению, мы уже умеем строить диаграммы множеств, которые меньше данного, и их диаграммы — действительно прямые. Тогда мы можем расположить диаграмму множества L слева a_0 (см. рисунок 1.6), т.к. все его элементы меньше a_0 , а диаграмму множества R — справа, что доказывает теорему.

⁴ Да, это QuickSort, неожиданно, правда?

Глава 2

Комбинаторика

2.1 Основные комбинаторные правила

2.1.1 Правило птичьих гнезд¹

Крайне банальное и очевидное правило, многим непонятно, зачем это в принципе нужно записывать в отдельное правило. Если коротко, то оно заключается вот в чем: если у нас есть n мест и $n + 1$ (или больше) объектов, которые мы пытаемся в эти места расположить, то у нас не получится расположить их так, чтобы в каждом месте было не больше 1 объекта. Доказательство от противного: предположим, что мы смогли их так разместить, тогда давайте достанем их обратно. Из каждого места мы достаем ≤ 1 объект, делаем мы это n раз, т.е. всего мы достанем $\leq n$, но положили $n + 1$, несостыковка, значит теорема доказана.

2.1.2 Правило умножения

Правило умножения считает последовательности. Причем именно последовательности, а не то, что вы хотите, чтобы оно считало. Для того, чтобы эта группа последовательностей подходила под правило умножения, она должна удовлетворять некоторым свойствам: первый элемент всегда можно выбрать m_1 способами, второй элемент всегда можно выбрать m_2 способами, *если уже выбран первый* (это важно), третий элемент можно выбрать m_3 способами, *если уже выбраны первые два*, и т.д. Если все это выполняется, то общее количество таких последовательностей будет $m_1 \cdot m_2 \cdot m_3 \cdot \dots \cdot m_n$.

Это «если выбраны все предыдущие» очень важно: рассмотрим набор из 10 последовательностей длины 4: $\{0000, 1111, 2222, 3333, \dots, 9999\}$. Если мы уберем это условие, то мы будем считать так: первый элемент — один из $\{0, \dots, 9\}$, $m_1 = 10$ вариантов, второй элемент — тоже один из $\{0, \dots, 9\}$, $m_2 = 10$ вариантов, аналогично $m_3 = m_4 = 10$. Таким образом, мы тогда

¹Оно же принцип Дирихле

получим, что в нашем наборе должно быть $10 \cdot 10 \cdot 10 \cdot 10 = 10000$ последовательностей, но там только 10, значит что-то мы посчитали не так.

Правильно будет считать так: $m_1 = 10$, тут мы не ошиблись, но $m_2 = 1$, потому что *если мы уже выбрали первую цифру*, то следующую мы можем выбрать только одним способом — выбрать ту же самую. Аналогично $m_3 = m_4 = 1$, и общее количество последовательностей будет $10 \cdot 1 \cdot 1 \cdot 1 = 10$, как и ожидалось.

Доказательство этого тоже достаточно очевидно, используем математическую индукцию: если все предыдущие элементы последовательности по индуктивному предположению можно выбрать $m_1 \cdot m_2 \cdot m_3 \cdot \dots \cdot m_{n-1}$ способами, и каждому из них соответствует по m_n способов выбрать следующий, то всего способов выбрать эту последовательность будет $m_1 \cdot m_2 \cdot m_3 \cdot \dots \cdot m_n$.

2.1.3 Правило сложения

Если имеется несколько *непересекающихся* множеств объектов, и мы можем посчитать количество объектов в каждом, то общее количество будет просто суммой всех отдельных количеств. Крайне очевидно и снова непонятно, зачем это считать отдельным правилом. Доказательство просто говорит, что в «общем количестве» каждый объект будет считаться по 1 разу, и в «сумме отдельных количеств» тоже, потому что этот объект будет находиться только в одном из непересекающихся множеств.

2.2 Сочетания и размещения

2.2.1 Размещения

Размещение — особый вид последовательности объектов, где длина фиксирована, и все объекты набираются из общего множества. В случае, если можно брать один и тот же объект несколько раз, это называется *размещением с повторениями*, а если это запрещено — то *размещением без повторений*.

Важно: размещение с повторениями *не обязано* иметь эти самые повторения, оно просто *может*.

Общее число размещений длины m , где каждый элемент выбирается из n -элементного множества *без повторений* обозначается A_n^m , а с *повторениями* — \overline{A}_n^m . Вывод обеих формул производится через правило умножения:

1. Размещения без повторений

Возьмем, например, множество $\{1, \dots, 6\}$ и будем набирать из него последовательности длины 4. Сколько способов выбрать первый элемент? 6. Второй, после выбора первого? 5, потому что мы не можем выбрать тот же самый, у нас запрещены повторения. Третий? 4. Четвертый? 3. Итого вариантов $6 \cdot 5 \cdot 4 \cdot 3$. В общем виде это будет $n(n-1)(n-2) \dots (n-m+1)$.

Q: Почему именно $n - m + 1$? Зачем здесь $+1$?

A: Просто так вышло, в этом нет какого то особо глубокого смысла. Просто проверьте на нашем примере, у нас $n = 6, m = 4, n - m = 2$, а последнее число в произведении должно быть 3, отсюда и $+1$.

Эта формула не очень удобная, ее лучше упростить. Есть такая хорошая вещь, как факториал, $n! = n(n - 1)(n - 2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$. К сожалению у нас тут слишком много слагаемых, а именно $(n - m)! = (n - m)(n - m - 1) \cdot \dots \cdot 2 \cdot 1$ здесь лишние. Тогда если мы разделим $n!$ на $(n - m)!$, мы получим как раз то, что нужно:

$$A_n^m = \frac{n!}{(n - m)!}.$$

2. Размещения с повторениями

Аналогично размещениям без, будем действовать по правилу умножения. Первый элемент можно выбрать n способами, второй тоже n способами, потому что повторы разрешены и можно брать и тот же самый элемент, третий тоже n , и вообще все n . Так m раз. И все это умножается. Итого:

$$\bar{A}_n^m = n^m.$$

2.2.2 Сочетания

Сочетание — почти то же самое, что и размещение, но *совокупность*, т.е. множество, а не *последовательность*, как размещение. Иначе говоря, в размещениях важен порядок, и размещения $(1, 2, 3)$ и $(2, 3, 1)$ будут считаться разными, в то время как сочетания $\{1, 2, 3\}$ и $\{2, 3, 1\}$ будут считаться одним и тем же.

Аналогично размещениям, существуют сочетания без повторений и с ними, и их количество обозначается C_n^m и \bar{C}_n^m соответственно.

1. Сочетания без повторений

Раз они настолько похожи на размещения, было бы неплохо выразить их через уже известную формулу. Давайте как всегда рассмотрим пример: сочетания из множества $\{1, 2, 3, 4, 5\}$ по 3 элемента. Соответствующих размещений у нас будет $A_5^3 = \frac{5!}{2!} = \frac{120}{2} = 60$. Рассмотрим например размещение $(1, 2, 3)$. Ему соответствует сочетание $\{1, 2, 3\}$. Только вот еще этому сочетанию соответствуют еще 5 размещений: $(1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)$. Всего $6 = 3! = m!$. Тогда размещений у нас в 6 раз больше, чем сочетаний, тогда чтобы найти сочетания, нужно разделить размещения на 6: $C_5^3 = \frac{A_5^3}{3!} = \frac{60}{6} = 10$. В общем виде это будет:

$$C_n^m = \frac{A_n^m}{m!} = \frac{n!}{m!(n - m)!}.$$

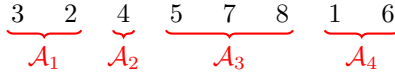


Рис. 2.1: Разбиение на части, $m_1 = 2, m_2 = 1, m_3 = 3, m_4 = 2$

2. Сочетания с повторениями

Вот тут все становится значительно сложнее. Пусть у нас есть множество $\{1, 2, 3, 4, 5, 6, 7\}$, и нам нужно набрать «множество»² из 5 элементов. Это значит, что мы должны каждому из 7 элементов сопоставить число, сколько этих элементов мы будем брать, и в сумме оно должно давать 5. Достаточно сложная задача. К счастью, можно посчитать какие то другие вещи, а потом доказать, что они — равноценны. Пусть нашим тестовым сочетанием будет $[1, 1, 3, 5, 6]$. Обозначим наше сочетание вот таким вот странным образом: $\star\star || \star || \star | \star |$ (на языке Костенко, «0011011010»). Каждая « \star » показывает один объект, а разделители, собственно, разделяют группы одинаковых элементов. Думаю, понятно, как это работает. Так вот, у нас здесь есть 5« \star » и 6« $|$ » (m и $n - 1$, соответственно), причем любой способ их разместить соответствует какому то сочетанию с повторениями. Тогда если мы просто выберем из всех 11 ($n + m - 1$) позиций 5 (m) позиций под « \star », то мы как раз посчитаем разбиения, итого:

$$\bar{C}_n^m = C_{n+m-1}^m.$$

Доказательство, что наши множества — равномощны, т.е. что преобразование — биекция, не составляет труда: оно инъективно, потому что разные сочетания дадут разные количества « \star » хотя бы в той «группе», в которых у сочетаний есть отличия, и оно сюръективно, потому что из каждой комбинации « \star » и « $|$ » можно восстановить обратно исходную комбинацию. Что и требовалось доказать.

2.3 Разбиение множества на части

Пусть у нас есть множество \mathcal{A} , и мы хотим его разбить на множества $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_k$, причем в них мы хотим получить по m_1, m_2, \dots, m_k элементов соответственно (причем $m_1 + m_2 + \dots + m_k = n$, т.е. все элементы должны попадать в какое то множество). Есть два вывода двух разных (но эквивалентных) формул для подсчета количества способов это сделать:

1. Через размещения

Мы можем расставить все элементы множества \mathcal{A} в каком то порядке (сделать это существует $A_n^n = n!$ способов), и поместить первые m_1

²Это не настоящее множество, потому что элементы могут повторяться по несколько раз. Формально это называется «комплекс», но так никто не говорит.

элементов в \mathcal{A}_1 , следующие m_2 в \mathcal{A}_2 и т.д. (см. рисунок 2.1) После этого получаем, что каждому разбиению соответствуют $m_1! \cdot m_2! \cdot \dots \cdot m_k!$ размещений, потому что каждую из групп можно записать $m_i!$ разными способами за счет изменения порядка. Итого общее число разбиений будет равно

$$\frac{n!}{m_1! \cdot m_2! \cdot \dots \cdot m_k!}.$$

2. Через последовательность действий

Будем по очереди выбирать элементы для каждого из множеств $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_k$. Существует $C_n^{m_1}$ способов выбрать элементы для множества \mathcal{A}_1 , $C_{n-m_1}^{m_2}$ способов выбрать элементы для \mathcal{A}_2 , если \mathcal{A}_1 уже выбрано, и $C_s^{m_k}$ способов выбрать элементы для \mathcal{A}_k , где $s = n - \sum_{i=1}^{k-1} m_i = m_k$, если уже выбраны элементы для всех предыдущих множеств. Такие последовательности действий соответствуют разбиениям 1 к 1, так что их будет ровно столько же. Тогда, количество разбиений равно

$$C_n^{m_1} \cdot C_{n-m_1}^{m_2} \cdot \dots \cdot C_s^{m_k}.$$

2.4 Формула включений-исключений

Крайне простая вещь, описанная сложным языком. Пусть нам даны множества A_1, A_2, \dots, A_n (возможно пересекающиеся) и мы хотим посчитать мощность их объединения. Как мы будем это считать для $n = 2$, т.е. искать $|A \cup B|$? Если мы просто сложим $|A| + |B|$, то все элементы в пересечении посчитаются 2 раза, поэтому необходимо вычесть мощность этого пересечения. Тогда

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Для 3 аналогично, это лучше нарисовать на диаграмме Венна: каждая точка означает 1 раз подсчета элементов в этой области (см. рисунок 2.2).

В общем виде это получается, что нужно сложить все мощности множеств, потом вычесть все пересечения по 2, потом прибавить все пересечения по 3, вычесть по 4 и т.д. Тогда $n_1 = \sum_i |A_i|$ — сумма всех мощностей, $n_2 = \sum_{i \neq j} |A_i \cap A_j|$ — сумма мощностей всех пересечений по 2, $n_3 = \sum_{i \neq j \neq k} |A_i \cap A_j \cap A_k|$ — по 3, и общей формулой будет

$$|\bigcup_i A_i| = n_1 - n_2 + n_3 - n_4 + \dots + (-1)^{k-1} n_k.$$

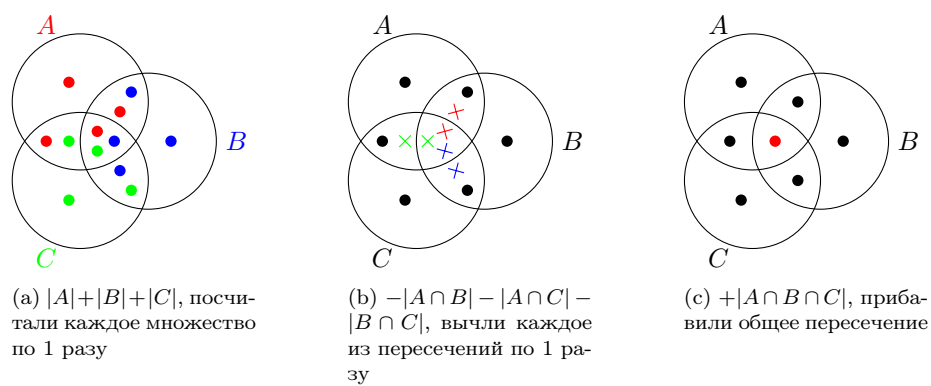


Рис. 2.2: Формула включений-исключений, $n = 3$

Глава 3

Функции алгебры логики

3.1 Функции

Для начала, обозначения:

E_2 : Множество из 0 и 1, элемент E_2 — один бит.

E_2^n : Множество из двоичных наборов длины n , элемент E_2^n — n бит, собранные вместе, например элементы E_2^8 — различные значения 1 байта (например, $01010010 \in E_2^8$).

P_2^n : Множество всех функций, которые принимают двоичный набор длины n и выдают 0 или 1, т.е. $f : E_2^n \rightarrow E_2$. Например, конъюнкция, $f(x_1, x_2) = x_1 \& x_2$, $f \in P_2^2$.

P_2 : Множество вообще всех функций алгебры логики (далее ФАЛ), принимающих двоичные наборы (любой длины) и выдающих 0 или 1.

Хороший способ представлять ФАЛ — некий механизм/микросхема, которая принимает несколько бит на вход и выдает 1 бит на выход для каждого из вариантов входов. Так, полным описанием ФАЛ будет таблица, где написаны все возможные комбинации входов и значение, которое функция выдает на каждом из них, например, что-то вроде такого:

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Эта таблица полностью описывает некоторую функцию $f(x_1, x_2, x_3)$ от 3 переменных. Названия переменных не так важны, важны только номера входов, т.е. $f(x, y, z)$ — то же самое, что $f(x_1, x_2, x_3)$. Мы не знаем смысл этой функции и зачем она считает именно это, мы знаем только то, какая она и какие значения принимает на каких входах.

Вопрос, сколько всего существует функций от n переменных? Из комбинаторики мы знаем, что количество двоичных наборов длины n равно 2^n ($|E_2^n| = 2^n$), потому что, по правилу умножения, каждая переменная может принимать одно из 2 значений, или 0, или 1, независимо от предыдущих. Тогда общее количество вариантов n переменных будет 2^n . В нашем случае, это значит, что если переменных n , то всего строк в таблице 2^n .

Далее, функция у нас полностью определяется только столбцом значений, для всех функций первые n столбиков, соответствующие переменным, будут абсолютно одинаковыми. Всего в столбце значений функции есть 2^n нулей или единиц, причем каждый такой двоичный набор (длины 2^n) полностью определяет 1 функцию. Сколько таких наборов длины 2^n (и, следовательно, функций) всего существует? 2^{2^n} . И да, это $2^{(2^n)}$, не $(2^2)^n$. Итак,

$$|P_2^n| = 2^{2^n}.$$

Если переменная (вход) функции ничего никогда не меняет, то она *несущественная*. Это значит, что при любых вариантах других переменных от изменения этой переменной значение функции не меняется. Если формально, то $\forall(\sigma_1, \sigma_2, \dots, \sigma_{i-1}, \sigma_{i+1}, \dots, \sigma_n) : f(\sigma_1, \sigma_2, \dots, \sigma_{i-1}, 0, \sigma_{i+1}, \dots, \sigma_n) = f(\sigma_1, \sigma_2, \dots, \sigma_{i-1}, 1, \sigma_{i+1}, \dots, \sigma_n)$. Сигмы — конкретные значения соответствующих переменных, которые мы выбираем для всех, кроме x_i . На входы функции мы подаем эти самые значения, а вместо x_i сначала 0, а потом 1. Если при этом ничего не меняется (причем для всех наборов), то переменная x_i — несущественная. Ну а существенные переменные хотя бы на одном наборе значений что-то меняют, тут все просто.

Алгоритм удаления достаточно прост, мы просто удаляем половину строк (оставляем только те, на которых у остальных переменных разные значения) и удаляем столбик x_i . Добавить еще проще, просто добавляем этот столбик обратно и дублируем все строки вместе со значениями функции. Одна копия для $x_i = 0$, другая для $x_i = 1$. Ну и функции равны, если они отличаются только количеством несущественных переменных, это логично.

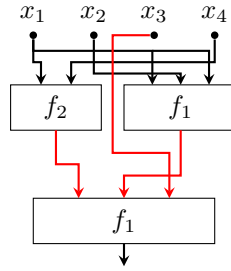
Насчет записи, любую функцию можно записывать 3 способами:

1. префиксная запись: $\&(x_1, x_2)$.
2. инфиксная запись: $x_1 \& x_2$.
3. постфиксная запись: $x_1 x_2 \&$.

Из всех них нам привычнее всего инфиксная, но она не очень хорошо расширяется на 3 переменные (и нет, мы не будем делать, как в C++, $x_1 ? x_2 : x_3$), поэтому для функций обычно используется префиксная запись.

3.2 Формулы

Формула — комбинация функций в одну кучу. Если продолжать аналогию с микросхемами, то мы можем подавать выходы одних микросхем на входы других и собирать таким образом намного более сложные схемы из простых элементов. Причем если мы положим эту получившуюся большую схему в «коробку», мы ее вполне можем называть «микросхемой» — у нее есть несколько бит входов и 1 бит выхода (мы не рассматриваем функции, выдающие несколько бит на выход). Аналогично можно работать и с функциями, например, если у нас есть некие функции $f_1(x_1, x_2, x_3)$ и $f_2(x_1, x_2)$, мы можем подавать выход одной функции как вход другой и получить намного более сложную *формулу* $f_1(f_2(x_1, x_2), f_1(x_1, x_2, x_3), x_3)$. Продолжая аналогию, эта *формула* соответствует схеме



Итак, *формула* — это запись, описывающая, как скомбинировать несколько *функций* в нечто более сложное. Это «нечто более сложное» само можно рассматривать, как функцию, и ее табличную запись мы получим, если переберем все возможные входные наборы и запишем выходные значения нашей «схемы».

Мы можем более формально определить формулу, а именно формулу над B . B — просто множество функций, которые мы можем использовать, набор доступных «микросхем». Так вот, формула — это

1. или просто одна функция из доступных
2. или функция, которой на входы подаются или напрямую переменные (причем возможно в другом порядке или даже с повторами), или выходы других формул над B , т.е. других таких же наборов из все тех же функций из B .

И да, есть много разных способов выразить одну и ту же функцию через другие, составить «схему» по «спецификации» (описанию необходимой функции) из «микросхем». Разные формулы, которые выражают одну и ту же функцию (игнорируя несущественные переменные), называются эквивалентными.

И да, обозначения. U — формула. $U(f_1, f_2, \dots, f_m)$ — формула, использующая (имеющая внутри себя) функции f_1, f_2, \dots, f_m . $f_U(x_1, \dots, x_n)$ —

x_1	x_2	x_3	x_4	x_5	f
0	0	0	0	0	0
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0	0	1	1	1	0
0	1	0	0	0	1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
0	1	1	1	1	0
1	0	0	0	0	1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
1	0	1	1	1	1
1	1	0	0	0	1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
1	1	1	1	1	0

Таблица 3.1: Таблица истинности функции f

функция, которую представляет формула U . Еще это можно записать как

$$f_U(x_1, \dots, x_n) = U(f_1, f_2, \dots, f_m).$$

Существует много различных правил, которые описывают, что можно делать с формулами, чтобы получить эквивалентные, типа менять местами «слагаемые», использовать законы де Моргана и прочее подобное. Не думаю, что все эти соотношения стоит как-то отдельно пояснять.

3.3 Разложение функций по переменным

Как всегда, все понятнее на примерах. Пусть наша функция f — функция 5 переменных ($n = 5$) (см. таблицу 3.1) и мы будем раскладывать ее по 2 переменным ($m = 2$)

Чтобы это сделать, мы выделим те группы наборов, у которых первые $m = 2$ переменных (x_1, x_2) принимают одинаковые значения. В таблице 3.1 эти группы выделены разными цветами.

Нам потребуется способ «выбирать» какие то отдельные наборы при помощи формул, т.е. какая то формула, которая 1 на выбранной нами группе наборов и нигде больше. Тут вступают в дело элементарные конъюнкции.

$$\text{Для начала, определим вспомогательное обозначение: } x^\sigma = \begin{cases} x, & \sigma = 1 \\ \bar{x}, & \sigma = 0 \end{cases}.$$

Тут σ — некая константа, и в зависимости от нее мы или отрицаем x , или нет. Также нетрудно проверить, что $x^\sigma = 1$ только тогда, когда x совпадает с σ . Хорошо. Теперь объединим несколько таких вещей конъюнкцией: $x_1^{\sigma_1} \& x_2^{\sigma_2} \& \dots \& x_m^{\sigma_m}$. Тогда будет получаться, что это нечто равно 1 толь-

ко тогда, когда все x_1, x_2, \dots, x_m равны заранее выбранным $(\sigma_1, \sigma_2, \dots, \sigma_m)$. Именно то, что нужно!

Итак, к нашей функции. Если мы составим конъюнкции $x_1^0 \& x_2^0, x_1^0 \& x_2^1, x_1^1 \& x_2^0, x_1^1 \& x_2^1$ (или, иначе, $\overline{x_1} \& \overline{x_2}, \overline{x_1} \& x_2, x_1 \& \overline{x_2}, x_1 \& x_2$), то каждая из них будет равна 1 только на наборах соответствующего цвета в таблице. Значения функции на этих частях будут, соответственно, $f(0, 0, x_3, x_4, x_5), f(0, 1, x_3, x_4, x_5), f(1, 0, x_3, x_4, x_5), f(1, 1, x_3, x_4, x_5)$. Тогда, если мы соберем все вместе, получим формулу

$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5) &= \overline{x_1} \& \overline{x_2} \& f(0, 0, x_3, x_4, x_5) \\ &\vee \overline{x_1} \& x_2 \& f(0, 1, x_3, x_4, x_5) \\ &\vee x_1 \& \overline{x_2} \& f(1, 0, x_3, x_4, x_5) \\ &\vee x_1 \& x_2 \& f(1, 1, x_3, x_4, x_5) \end{aligned}$$

или, в более компактном виде,

$$f(x_1, x_2, x_3, x_4, x_5) = \bigvee_{(\sigma_1, \sigma_2) \in E_2^2} x_1^{\sigma_1} \& x_2^{\sigma_2} \& f(\sigma_1, \sigma_2, x_3, x_4, x_5).$$

Этот большой символ « \bigvee » — примерно то же самое, что « \sum », но для дизъюнкции.

Можем рассмотреть это на конкретных значениях $x_1 \dots x_5$, а именно 10010 (это и есть те самые $\gamma_1 \dots \gamma_n$ в доказательстве):

$$\begin{aligned} f(1, 0, 0, 1, 0) &= \overline{1} \& \overline{0} \& f(0, 0, 0, 1, 0) \\ &\vee \overline{1} \& 0 \& f(0, 1, 0, 1, 0) \\ &\vee 1 \& \overline{0} \& f(1, 0, 0, 1, 0) \\ &\vee 1 \& 0 \& f(1, 1, 0, 1, 0) \end{aligned}$$

Все конъюнкции, кроме $\overline{1} \& 0$ будут равны 0, поэтому единственное оставшееся значение — это $f(0, 1, 0, 1, 0)$, что верно, и, кроме того, эти первые «0, 1» были подставлены в функцию еще до того, как мы выбрали конкретный набор $x_1 \dots x_5$, т.е. после такого действия функцию можно было упростить. Но что здесь гораздо важнее — это частные случаи, а особенно $n = m$, когда мы раскладываем по всем переменным. Тогда в функции подставляются значения всех переменных и их значения можно полностью посчитать заранее, т.е. компонент « $f(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5)$ » — просто константа, и если она 0, то всю конъюнкцию можно убрать. Тогда вся функция превращается просто в дизъюнкцию всех конъюнкций, которые соответствуют наборам, на которых наша функция равна 1. Например, пусть наша функция будет

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Тогда из 8 конъюнкций останутся только 3, соответствующие наборам 001, 011, 110, на которых наша функция равна 1. Тогда разложение по переменным будет иметь вид

$$f(x_1, x_2, x_3) = x_1^0 \& x_2^0 \& x_3^1 \vee x_1^0 \& x_2^1 \& x_3^1 \vee x_1^1 \& x_2^1 \& x_3^0$$

или

$$f(x_1, x_2, x_3) = \overline{x_1} \& \overline{x_2} \& x_3 \vee \overline{x_1} \& x_2 \& x_3 \vee x_1 \& x_2 \& \overline{x_3}.$$

Такая формула, представляющая функцию f , называется ее *совершенной дизъюнктивной нормальной формой*, или *СДНФ*.

3.4 Минимальные ДНФ

Итак, если мы хотим как-то записать функцию при помощи функций попроще, мы можем построить ее СДНФ. К сожалению, это очень неэффективно, потому что СДНФ очень быстро становятся просто огромными. Для того, чтобы ее упростить, мы разрешим помещать в конъюнкцию не все переменные, а только часть. А дизъюнкцию таких вещей будем называть *ДНФ*, без «С». Еще определим *сложность ДНФ* — это количество операций конъюнкции и дизъюнкции в записи этой ДНФ. Сложность ДНФ D обозначается как $L(D)$. Тогда мы можем ввести понятие *минимальной ДНФ* для функции — это просто ДНФ наименьшей сложности, которая представляет нашу функцию.

Теорема. Для всех функций, кроме константы 0, существует минимальная ДНФ.

Доказательство. Ну, для начала, насчет 0, для него в принципе не существует ДНФ, если не считать ДНФ пустую формулу. Так вот, для всех остальных функций, мы для них всегда можем построить хотя бы одну ДНФ — это СДНФ. Теперь нужно доказать, что всего ДНФ для функции — конечное число, и тогда если у нас есть конечное число ДНФ и оно не 0, то мы всегда сможем найти минимум, так? Так вот, нужно найти число ДНФ для функции, а точнее, в принципе число ДНФ от n переменных.

Рассмотрим элементарные конъюнкции. Для каждой из n переменных у нас есть 3 варианта: она может или быть в конъюнкции, или не быть, или быть там с отрицанием, причем варианты каждой из переменных не зависят от других. Тогда, по правилу умножения получаем, что всего элементарных конъюнкций будет 3^n . Но, среди этого числа учтена «конъюнкция», где нет ни одной переменной, такая не считается, так что на самом деле их $3^n - 1$.

Рассмотрим ДНФ. Каждая из конъюнкций (которых $3^n - 1$), может там или быть, или не быть — 2 варианта. Итого, по правилу умножения, у нас будет 2^{3^n-1} ДНФ. Но, мы опять же учли пустую ДНФ, которая не должна считаться, так что всего ДНФ от n переменных существует $2^{3^n-1} - 1$, т.е. конечное число. Тогда мы всегда сможем найти минимальную ДНФ для функции. \square

3.5 Геометрическая интерпретация ДНФ

К сожалению, искать так минимальную ДНФ, перебирая буквально все ДНФ для функции и проверяя на минимальность — очень долго. К счастью, есть способы и получше. Но для этого, нам нужно представить наше множество входных значений E_2^n как n -мерный куб. Просто выберем начало координат, это будет наш набор $0 \dots 0$. Выберем n осей координат, и вершина, в которую мы попадаем, двигаясь по направлению номер i , соответствует тому же самому набору, но x_i заменено с 0 на 1. Достроив до куба, получаем все наборы E_2^n (см. рисунок 3.1).

Тогда для каждой функции мы можем определить множество вершин куба, на которых функция дает 1. Т.к. функция у нас выражается через ДНФ, то это множество N_f можно рассматривать как объединение всех множеств N_{K_i} , соответствующих элементарным конъюнкциям. Пусть в нашей конъюнкции K_i используется r переменных из n . Это значит, что она 1, когда эти r переменных принимают фиксированные значения, а остальные $n - r$ переменных могут принимать любые значения, от них значение конъюнкции не поменяется. Каждая из них принимает одно из 2 значений, значит всего наборов, на которых наша конъюнкция равна 1 будет 2^{n-r} . Соответственно, это и количество точек в нашей грани N_{K_i} . Такое множество точек называется $n - r$ -мерной гранью n -мерного куба. Ну то есть 0-мерная грань — это 1 точка, 1-мерная грань — ребро, 2-мерная — обычная плоская грань, 3-мерная — весь куб (или какой-то 3-мерный куб внутри кубов более высоких измерений).

Например, конъюнкция $x_1 \& x_4$ соответствует грани на рисунке 3.1, выделенной красным.

Итак, чем меньше длина конъюнкции, тем больше грань, т.е. чтобы получить минимальную ДНФ, нужно «покрыть» множество N_f как можно большими гранями, причем такими, у которых $N_K \subseteq N_f$, т.е. которые не добавляют к нашему множеству лишних точек (потому что их не получится потом убрать).

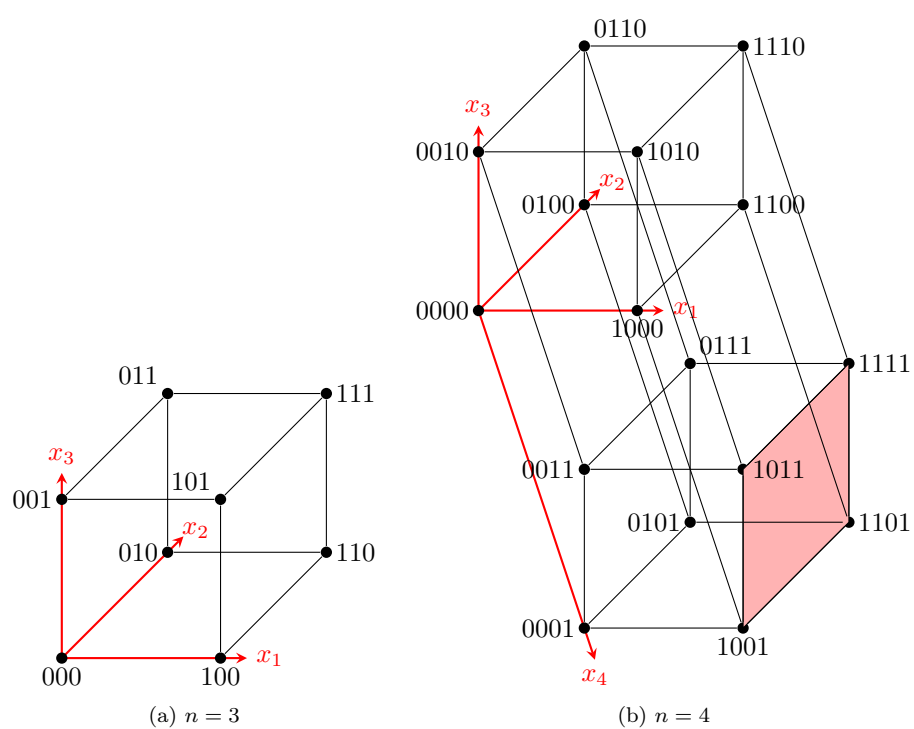


Рис. 3.1: n -мерные кубы

3.6 Максимальные конъюнкции

Конъюнкция называется максимальной для функции f , если она

1. Может находиться в ДНФ функции f , т.е. не равна 1 ни на каких лишних наборах ($N_K \subseteq N_f$)
2. При расширении конъюнкции (убирании из нее каких то переменных) она уже больше не подходит, т.е. любая расширенная конъюнкция K' , для которой $N_K \subset N_{K'}$, выполняется $N_{K'} \not\subseteq N_f$, т.е. конъюнкция не подходит.

Проще говоря, конъюнкция называется максимальной, если ее нельзя расширить (убрав переменные) так, что она все еще не добавляет лишних наборов в N_f .

Теорема. В минимальных ДНФ все конъюнкции — максимальные.

Доказательство. Предположим противное, у нас есть не максимальная конъюнкция K_i , и ее можно расширить до K' , и при этом все еще выполняется $N_{K'} \subseteq N_f$. Как мы обнаружили, если размер грани увеличивается, то длина записи конъюнкции (и ее сложность) уменьшается, т.е. $L(K_i) > L(K')$. Тогда если мы заменим K_i на K' , то получим D' , причем $L(D) > L(D')$, т.е. мы получили ДНФ меньше, и она все еще выражает ту же самую функцию, потому что мы не убрали ни одного набора, где функция 1 (потому что мы расширили грань, все, что было — так и осталось), и не добавили новых (потому что $N_{K'} \subseteq N_f$). Итак, D' по сложности меньше, представляет ту же функцию, значит D — не минимальная ДНФ, противоречие, теорема доказана. \square

3.7 Полные системы ФАЛ

А вот отсюда начинаются уже по настоящему сложные темы (по крайней мере по мнению моих одноклассников). Итак, если B — какое-то множество ФАЛ, то замыканием B называется множество всех функций, которые можно получить из функций из B путем различного комбинирования. Перечитайте эту фразу еще раз, она достаточно важна. Обозначается это дело как $[B]$. Если снова вернуться к нашей аналогии функций и «микросхем», то замыкание B — это все функции, которые мы можем получить, используя только микросхемы из набора B .

Например, если $B = \{x, \bar{x}\}$, то $[B] = \{x, \bar{x}\}$ (функция $f(x) = x$ — функция, которая ничего не делает, identity, она просто «провод»). По аналогии микросхем, это означает, что если у нас есть только микросхема с отрицанием и «провод», то мы, как бы их ни комбинировали, получим только отрицание, и схемы, которые выдают то же самое (например если сделаем двойное отрицание).

Другой пример, $B = \{\&, \vee, \bar{x}\}$. Для него $[B] = P_2$. Это значит, что при помощи этих трех функций/микросхем мы можем получить абсолютно все,

что угодно, любую из существующих ФАЛ (P_2). Такие наборы/системы функций называются *полными*. В это же время, такие наборы, как $\{x, \bar{x}\}$, у которых $[B] = B$, называются *замкнутыми*, потому что из них ничего нового получить не возможно, только то, что уже есть.

Теорема редукции. *Если у нас имеется 2 системы, B_1 и B_2 , причем B_1 — полная, и все функции из нее можно записать при помощи B_2 , то B_2 — тоже полная.*

Доказательство. Чтобы доказать, что B_2 — полная, нужно просто доказать, что через нее можно выразить любую функцию. Назовем эту функцию, которую будем пытаться выразить, h . Т.к B_1 — полная, то по крайней мере через нее то мы можем выразить нашу h , это и сделаем. Тогда $h(x_1, \dots, x_2) = U(f_1, \dots, f_m)$, где все f лежат в B_1 . Это значит, что мы нашли формулу U , в которой используются функции f_1, \dots, f_m из B_1 и которая представляет нашу функцию h . Итак, прекрасно. По условию, все эти функции f_1, \dots, f_m мы можем выразить через B_2 , т.е.

$$\begin{aligned} f_1 &= U_1(\dots) \\ &\vdots \\ f_m &= U_m(\dots) \end{aligned}$$

где все формулы используют только функции из B_2 . По крайне простой и очевидной теореме о замене равных¹, которая просто говорит, что мы можем в формуле менять кусочки на эквивалентные, и ничего не сломается, мы можем просто взять и заменить наши f_1, \dots, f_m в U на U_1, \dots, U_m и у нас будет $h(x_1, \dots, x_2) = U(U_1, \dots, U_m)$. Теперь в нашей формуле используются только функции из B_2 , следовательно мы смогли выразить h через B_2 , а т.к. h — произвольная функция, то B_2 — полная система. \square

Хорошим примером будет система $B = \{|\}$. Да, только одна функция, штрих Шеффера, он же NAND², он же — отрицание конъюнкции. Но при этом мы можем записать

$$\begin{aligned} \bar{x} &= x | x \\ x_1 \&x_2 &= \overline{(x_1 | x_2)} = (x_1 | x_2) | (x_1 | x_2) \\ x_1 \vee x_2 &= \overline{\bar{x}_1 | \bar{x}_2} = (x_1 | x_1) | (x_2 | x_2) \end{aligned}$$

и это дает нам возможность любую функцию сначала записать через $\&, \vee, \bar{x}$, а потом заменить их все на $|$, и поэтому $B = \{|\}$ — полная система по теореме редукции.

¹Что самое интересное, она — придумка Костенко, больше нигде я упоминаний о ней не нашел

²Это достаточно известный факт, вплоть до того, что многие радиолюбители Советского Союза буквально все делали через микросхему К561ЛА7, представлявшую из себя 4 штриха Шеффера в одном корпусе.

3.8 Полиномы Жегалкина

Неожиданно простая тема среди сложных. Полином Жегалкина — представление обычных функций через умножение и сложение по модулю 2 (т.е. все четные числа = 0, а все нечетные = 1). Таким образом, система функций $\{x_1 \cdot x_2, x + 1\}$ получается эквивалентной $\{x_1 \& x_2, \bar{x}\}$, которая является полной. Таким образом, любую функцию можно выразить через умножение и сложение (по модулю 2), а если раскрыть скобки и использовать то, что $x \cdot x = x$ (у нас только 0 и 1, учтите) и $x + x = 0$ (сложение по модулю 2), то можно привести абсолютно любую функцию к сумме нескольких произведений, и еще возможно $+1$. Например, $x_1 \rightarrow x_2 = x_1x_2 + x_1 + 1$.

Более того, каждой функции соответствует единственный полином Жегалкина, что можно доказать, посчитав общее количество полиномов Жегалкина для n переменных. Для начала, всего существует 2^n различных произведений n переменных, потому что каждая из них может в нем или быть, или нет, причем пустое произведение считаем за 1. Аналогично, если существует 2^n произведений, то существует 2^{2^n} различных сумм из этих произведений, потому что каждое из произведений может или быть в сумме, или нет. Причем пустая сумма — это просто тождественный 0. Итак, есть 2^{2^n} функций и 2^{2^n} полиномов Жегалкина, и каждую из функций можно преобразовать в свой полином Жегалкина, причем очевидно, что у каждой функции свой, иначе тогда полином Жегалкина должен на одних и тех же наборах давать значения для 2 разных функций. Тогда получается, что все полиномы уже «заняты», и значит соответствие между функциями и полиномами — биекция, что и требовалось доказать.

3.9 Классы T_0 и T_1

Снова вернемся к сложным темам. Функция называется *сохраняющей ноль*, если на нулевом наборе она выдает значение 0, т.е. $f(0, \dots, 0) = 0$. Множество всех таких функций называется T_0 . Таблицы таких функций выглядят так:

x_1	\dots	x_n	f
0	\dots	0	0
0	\dots	1	?
\vdots	\ddots	\vdots	\vdots
1	\dots	1	?

Т.е. у функций, сохраняющих 0, нам известно значение на одном наборе из 2^n , а на остальных $2^n - 1$ наборах оно может быть любым, и функция будет все еще сохранять 0. Тогда всего таких функций $2^{2^n - 1}$.

Теорема. Класс T_0 — замкнут, т.е. $[T_0] = T_0$.

Доказательство. Что ж, начнем... Чтобы доказать это, нужно доказать, что любая формула, составленная из функций, сохраняющих 0, будет сама представлять функцию, сохраняющую 0. Будем доказывать это индукцией по глубине формулы.

1. *Базис индукции:* $d = 1$. Тогда наша формула — просто сама функция из T_0 , возможно с переставленными переменными. Тогда очевидно, что $f(0, \dots, 0) = 0$, т.е. что от перестановки переменных свойство сохранения нуля не теряется.
2. *Индуктивное предположение:* $d \leq k$. Предположим, что для формул глубины меньше k у нас все хорошо работает, т.е. они все представляют функции из T_0 .
3. *Индуктивный переход:* $d = k + 1$. Итак, у нас есть некая формула над T_0 , а именно $U = f(A_1, \dots, A_m)$, где $f \in T_0$, а A_1, \dots, A_m — или просто переменные, или другие формулы над T_0 , причем глубины $\leq k$.

Не очень удобно, что наши A_i — или переменные, или формулы, поэтому давайте каждой из них сопоставим функцию.

- (а) Если $A_i = x_j$, то $f_{A_i}(x_j) = x_j$, и, очевидно, $f_{A_i} \in T_0$.
- (б) Если A_i — формула над T_0 , причем ее глубина $\leq k$, то по индуктивному предположению у нее все хорошо и она сама представляет функцию из T_0 . Ее мы и назовем f_{A_i} .

Итого, по теореме о замене равных мы имеем полное право заменить наши A_i на $f_{A_i} \in T_0$, и получить $U = f(f_{A_1}(\dots), \dots, f_{A_m}(\dots))$. Мы хотим доказать, что эта формула — тоже из T_0 (типа что из таких функций мы ничего нового составить не сможем, только T_0). Для этого нужно проверить ее значение на нулевом наборе, когда все переменные равны 0.

$$f_U(0, \dots, 0) = f(f_{A_1}(0, \dots, 0), \dots, f_{A_m}(0, \dots, 0))$$

Т.к. все наши f_{A_i} сами лежат в T_0 , то все они равны 0 на нулевом наборе, т.е. $f_U(0, \dots, 0) = f(0, \dots, 0) = 0$. Тогда наша функция — из T_0 , а класс T_0 — замкнутый.

□

Класс T_1 и функции сохраняющие единицу — абсолютно аналогичен, просто вместо $f(0, \dots, 0) = 0$ у нас $f(1, \dots, 1) = 1$. И он тоже замкнутый, что доказывается точно так же.

3.10 Двойственные функции

Для каждой ФАЛ можно определить другую ФАЛ, которая ей *двойственна*. Определяется так: если у нас есть функция $f(x_1, \dots, x_n)$, то двойственная ей функция $f^*(x_1, \dots, x_n)$ такая, что $f^*(x_1, \dots, x_n) = \overline{f(\overline{x_1}, \dots, \overline{x_n})}$. Например, если у нас $f(x_1, x_2) = x_1 \& x_2$, то $f^*(x_1, x_2) = \overline{f(\overline{x_1}, \overline{x_2})} = \overline{\overline{x_1} \& \overline{x_2}} = x_1 \vee x_2$, т.е. дизъюнкция двойственна конъюнкции и наоборот. Но в то же время, если $f(x) = \overline{x}$, то $f^*(x) = \overline{f(\overline{x})} = \overline{\overline{\overline{x}}} = \overline{x} = f(x)$, т.е. отрицание двойственно само себе. Такие функции называются *самодвойственными*.

Можно определить двойственность немного по другому. Пусть у нас есть двоичный набор $(\sigma_1, \dots, \sigma_n)$. Тогда можно сказать, что набор $(\overline{\sigma_1}, \dots, \overline{\sigma_n})$ — *противоположный* ему набор. Например, наборы $(1, 0, 0, 1, 0)$ и $(0, 1, 1, 0, 1)$ — противоположные. Про противоположные наборы можно сказать еще одно: они всегда находятся симметрично относительно центра таблицы. Доказательство этого простое: достаточно просто посчитать расстояния от этого самого центра до противоположных наборов, они будут равны.

Теперь, как построить функцию, двойственную данной. На самом деле, если у нас есть таблица, то ее можно построить намного проще, чем вычислять все значения напрямую. Из-за симметричности противоположных наборов, если мы просто перевернем столбик значений функции, все наборы поменяются на противоположные, т.е. наша функция $f(x_1, \dots, x_n)$ поменяется на $f(\overline{x_1}, \dots, \overline{x_n})$, а если после этого мы еще и инвертируем все значения в нашем столбике, то получим нашу двойственную функцию. Вот так:

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	$f(\overline{x_1}, \overline{x_2}, \overline{x_3})$	$f^*(x_1, x_2, x_3)$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	1	0	1	0	1
1	1	1	0	0	1

Существует одна интересная и полезная теорема о двойственных функциях:

Теорема. Если у нас есть функция h , заданная некоторой формулой $h(x_1, \dots, x_n) = U(f_1, \dots, f_m)$, то чтобы получить двойственную функцию h^* , достаточно просто заменить все функции в формуле на двойственные им:

$$h^*(x_1, \dots, x_n) = U^* = U(f_1^*, \dots, f_m^*)$$

Пример. Пусть $h(x_1, x_2, x_3) = (x_1 \& \overline{x_2}) \vee (\overline{x_3} \& x_4)$. Мы знаем, что конъюнкция двойственна дизъюнкции и наоборот, а отрицание двойственно само себе. Тогда по этой теореме, $h^*(x_1, x_2, x_3) = (x_1 \vee \overline{x_2}) \& (\overline{x_3} \vee x_4)$.

Доказательство. Доказывать будем как всегда индукцией по глубине формулы.

1. *Базис индукции:* $d = 1$, это значит, что наша формула — просто одна функция, т.е. $h(x_1, \dots, x_n) = f(x_1, \dots, x_n)$. Тогда $h^* = \overline{h(\overline{x_1}, \dots, \overline{x_n})} = \overline{f(\overline{x_1}, \dots, \overline{x_n})} = f^*$, т.е. если заменить функцию f на двойственную ей, то h поменяется на h^* (что было достаточно очевидно...)
2. *Индуктивное предположение:* $d \leq k$, предположим, что все формулы глубины $\leq k$ работают именно так: Если мы поменяем все функции в формуле на двойственные, то вся формула будет представлять функцию, двойственную изначальной: $h^*(x_1, \dots, x_n) = U(f_1^*, \dots, f_m^*)$.
3. *Индуктивный переход:* $d = k + 1$. Итак, у нас есть некая функция h , и она представлена формулой U : $h(x_1, \dots, x_n) = U(f_1, \dots, f_m)$, причем глубина U равна $k + 1$. Это значит, что $h(x_1, \dots, x_n) = f(A_1, \dots, A_m)$, где A_i — или символы переменных, или формулы сложности $\leq k$. Как и в доказательстве замкнутости T_0 , нам это дело не нравится, поэтому мы каждому A_i сопоставим функцию f_{A_i} .
 - (а) Если $A_i = x_j$, то $f_{A_i}(x_j) = x_j$. Как нетрудно проверить, $f_{A_i}^*(x_j) = x_j$.
 - (б) Если A_i — формула сложности $\leq k$, то мы можем обозначить за f_{A_i} функцию, которую она представляет. Более того, по индуктивному предположению мы можем получить $f_{A_i}^*$, просто заменив все функции, используемые в ее записи, на двойственные.

Что ж, приступим к проверке главного утверждения теоремы. У нас имеется:

$$\begin{aligned} h(x_1, \dots, x_n) = & f(f_{A_1}(x_{11}, \dots, x_{1l_1}), \\ & f_{A_2}(x_{21}, \dots, x_{2l_2}), \\ & \vdots \\ & f_{A_m}(x_{m1}, \dots, x_{ml_m})) \end{aligned}$$

Насчет индексов: это обозначения Костенко, не мои. Примерное объяснение: В каждой из f_{A_i} используются свои переменные, например, может быть, $f_{A_1}(x_1, x_3, x_5)$, а $f_{A_2}(x_4, x_7)$. Для этого Костенко вводит двойную нумерацию: у функции f_{A_i} все переменные обозначаются как x_{ij} , где i — номер f_{A_i} , а j — номер переменной в ней. Более того, мы не знаем, сколько там всего переменных и какое максимальное j , у каждой функции оно свое. Костенко обозначил его как l_i — количество переменных в функции f_{A_i} . Зачем так переусложнять такую второстепенную вещь, как индексы — вопрос не ко мне.

Итак, нам нужно найти $h^*(x_1, \dots, x_n) = \overline{h(\overline{x_1}, \dots, \overline{x_n})}$, или

$$\overline{f(f_{A_1}(\overline{x_{11}}, \dots, \overline{x_{1l_1}}), f_{A_2}(\overline{x_{21}}, \dots, \overline{x_{2l_2}}), \dots, f_{A_m}(\overline{x_{m1}}, \dots, \overline{x_{ml_m}}))}.$$

Пока все не очень понятно, но будет лучше, когда мы добавим двойное отрицание (которое абсолютно ничего не меняет) к нашим f_{A_i} :

$$\overline{\overline{f(f_{A_1}(x_{11}, \dots, x_{1l_1}), f_{A_2}(x_{21}, \dots, x_{2l_2}), \dots, f_{A_m}(x_{m1}, \dots, x_{ml_m}))}}.$$

Все все еще не очень понятно, но мы можем заметить $\overline{f_{A_i}(x_{i1}, \dots, x_{il_i})}$, а это же и есть $f_{A_i}^*$! Так что-то, что мы написали, равно

$$\overline{f(f_{A_1}^*(x_{11}, \dots, x_{1l_1}), f_{A_2}^*(x_{21}, \dots, x_{2l_2}), \dots, f_{A_m}^*(x_{m1}, \dots, x_{ml_m}))}.$$

Сейчас посмотрим на все выражение в целом: у нас есть $\overline{f(f_{A_1}^*, \dots, f_{A_m}^*)}$. Но это же и есть f^* ! Заменяем:

$$f^*(f_{A_1}^*(x_{11}, \dots, x_{1l_1}), f_{A_2}^*(x_{21}, \dots, x_{2l_2}), \dots, f_{A_m}^*(x_{m1}, \dots, x_{ml_m})).$$

Как мы обнаружили, когда вводили f_{A_i} , $f_{A_i}^* = A_i^*$, т.е. тому, что получается, если заменить все функции в этой формуле на двойственные им. Т.е. $h^*(x_1, \dots, x_n) = f^*(A_1^*, \dots, A_m^*)$, или, если по другому: если мы заменим внешнюю функцию на двойственную, и все функции, которые в ее аргументах (т.е. вообще все функции, которые встречаются в формуле), то общая функция поменяется на двойственную. Т.е. $h^*(x_1, \dots, x_n) = U^* = U(f_1^*, \dots, f_m^*)$, а именно это мы и пытались доказать!

□

3.11 Класс S

Как я уже говорил, самодвойственная функция — это такая, которая двойственна самой себе, типа отрицания. Это значит, что значения на противоположных наборах должны быть разные. Тогда, построив первую половину таблицы мы сможем полностью определить и вторую, просто перевернув ее и сделав отрицание. Например, вот так мы можем получить вторую половину таблицы из первой

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{\neg} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \xrightarrow{\overline{x}} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

и получить самодвойственную функцию со столбиком значений 10001110.

Таким образом, для задания самодвойственной функции нам достаточно только $\frac{2^n}{2} = 2^{n-1}$ строк таблицы. Тогда всего самодвойственных функций будет $2^{2^{n-1}}$.

Множество всех самодвойственных функций называется классом S (да, я знаю, очень неожиданно). А еще этот класс замкнутый (неожиданность

за неожиданностью). Доказательство простое: если у нас есть формула U , состоящая только из самодвойственных функций, и она представляет функцию h , т.е. $h(x_1, \dots, x_n) = U(f_1, \dots, f_m)$, то тогда по нашей лемме $h^* = U(f_1^*, \dots, f_m^*)$. Но т.к. все наши функции — самодвойственные, то эта новая формула — точно такая же, поэтому $h = h^*$, т.е. h — самодвойственная, что и требовалось доказать. Ну а теперь к интересным вещам:

Лемма (о несамодвойственной функции). *Если у нас есть несамодвойственная функция f ($f \notin S$), то если мы сможем подставить x и \bar{x} вместо переменных так, чтобы получилась константа (функция, которая не зависит от x).*

Доказательство. Если $f \notin S$, то значит на каких-то двух противоположных наборах значения функции равны (иначе она была бы самодвойственной):

$$f(\sigma_1, \dots, \sigma_n) = f(\bar{\sigma}_1, \dots, \bar{\sigma}_n).$$

Тогда, мы каждой переменной сопоставим $g_i(x) = x^{\sigma_i}$, т.е. \bar{x} , если $\sigma_i = 0$ или x , если $\sigma_i = 1$. Тогда пусть нашей функцией будет $h(x) = f(g_1(x), g_2(x), \dots, g_n(x))$. Посчитаем ее значение на 0 и на 1:

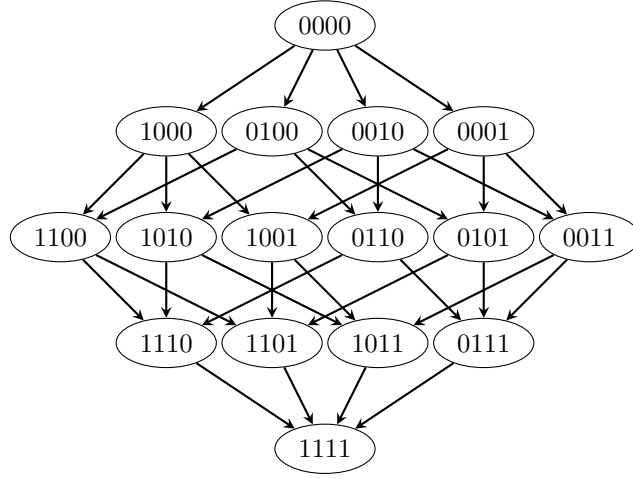
$$\begin{aligned} h(0) &= f(0^{\sigma_1}, 0^{\sigma_2}, \dots, 0^{\sigma_n}) = f(\bar{\sigma}_1, \dots, \bar{\sigma}_n) \\ h(1) &= f(1^{\sigma_1}, 1^{\sigma_2}, \dots, 1^{\sigma_n}) = f(\sigma_1, \dots, \sigma_n) \end{aligned}$$

Т.к. $f(\sigma_1, \dots, \sigma_n) = f(\bar{\sigma}_1, \dots, \bar{\sigma}_n)$, это значит $h(0) = h(1)$, т.е. $h(x)$ — константа. \square

3.12 Класс M

Введем отношение порядка на множестве двоичных наборов. Будем говорить, что набор $\alpha = (\sigma_1, \dots, \sigma_n)$ предшествует набору $\beta = (\delta_1, \dots, \delta_n)$, если для каждого из индексов $\sigma_i \leq \delta_i$ (обозначается $\alpha \preceq \beta$). Это буквально означает, что там, где в α стоят единицы, там в β тоже стоят единицы, а 0 в α могут стать в β как 0, так и 1. Например, $(010001) \preceq (011101)$, но $(010001) \not\preceq (111110)$. Это отношение является отношением порядка, так что для него можно даже нарисовать диаграмму (см. рисунок 3.2). И да, обозначение $f(\alpha)$ значит просто $f(\sigma_1, \dots, \sigma_n)$, в нем нет ничего нового.

Функция называется монотонной, если она «монотонно возрастает» для этого отношения предшествования. Более формально, это значит, что если набор α предшествует набору β , то $f(\alpha)$ должно быть больше или равно $f(\beta)$, что аналогично условию для возрастания обычных функций: если всегда, когда $x_1 < x_2$, выполняется $f(x_1) < f(x_2)$, это значит, что такая функция монотонно возрастает. А если всегда, когда $\alpha \preceq \beta$ выполняется $f(\alpha) \leq f(\beta)$, то это значит, что ФАЛ f — монотонная. Как мы можем догадаться, множество всех монотонных ФАЛ называется классом M . И, как всегда


 Рис. 3.2: Диаграмма отношения предшествования для $n = 4$

Теорема. Класс M замкнут, $[M] = M$.

Доказательство. Как всегда, индукция по глубине формулы.

1. *Базис индукции:* $d = 1$. $U = f(x_1, \dots, x_n)$, где $f \in M$. Тогда если $\alpha \preceq \beta$, то $f_U(\alpha) = f(\alpha) \leq f(\beta) = f_U(\beta)$, значит $f_U \in M$.
2. *Индуктивное предположение:* $d \leq k$. Если U — формула над M глубины $\leq k$, то она представляет монотонную функцию $f_U \in M$.
3. *Индуктивный переход:* $d = k + 1$, $U = f(A_1, \dots, A_m)$, причем $f \in M$ и A_i — или переменные, или формулы над M глубины $\leq k$.
 - (а) Если $A_i = x_j$, то $f_{A_i}(x_j) = x_j$ и тогда очевидно, что $f_{A_i} \in M$.
 - (б) Если A_i — формула над M глубины $\leq k$, то по индуктивному предположению функция, которую она представляет, f_{A_i} , является монотонной.

Если так, то мы можем записать нашу формулу как

$$\begin{aligned}
 f_U(x_1, \dots, x_n) = f(& f_{A_1}(x_{11}, \dots, x_{1l_1}), \\
 & f_{A_2}(x_{21}, \dots, x_{2l_2}), \\
 & \vdots \\
 & f_{A_m}(x_{m1}, \dots, x_{ml_m})).
 \end{aligned}$$

Объяснение индексов см. на странице 35. Проверим эту функцию на наборах $\alpha \preceq \beta$. Так как каждая из f_{A_i} использует только часть переменных, подготовим наборы $\alpha_1, \dots, \alpha_m$ и β_1, \dots, β_m , просто выбрав

из α и β те переменные, которые нужны функции f_{A_i} . Тогда $f_U(\alpha) = f(f_{A_1}(\alpha_1), \dots, f_{A_m}(\alpha_m))$ и $f_U(\beta) = f(f_{A_1}(\beta_1), \dots, f_{A_m}(\beta_m))$. Т.к. каждая из f_{A_i} — монотонная, то каждая из $f_{A_i}(\alpha_i) \leq f_{A_i}(\beta_i)$, а значит, если их объединить, получим наборы $(f_{A_1}(\alpha_1), \dots, f_{A_m}(\alpha_m)) \preceq (f_{A_1}(\beta_1), \dots, f_{A_m}(\beta_m))$. А т.к. $f \in M$, то это значит, что $f_U(\alpha) \leq f_U(\beta)$, т.е. f_U — монотонная. □

Лемма (о немонотонной функции). *Если имеется немонотонная функция f , то подставив вместо переменных 0, 1 или x мы сможем получить функцию $h(x) = \bar{x}$.*

Доказательство. Ну если функция немонотонная, то значит где-то монотонность нарушается, т.е. существуют 2 таких набора $\alpha \preceq \beta$, что $f(\alpha) > f(\beta)$. Т.к. мы работаем только на 0 и 1, это строгое неравенство возможно только если $f(\alpha) = 1$, а $f(\beta) = 0$. По идее, мы можем уже получить отрицание только из них, но мы попытаемся найти такие же 2 набора, но чтобы они отличались только в 1 переменной. Раз $\alpha \preceq \beta$, это значит, что если заменить некоторые 0 в α на 1, мы получим β . Так и сделаем, но будем заменять их по одному и по очереди: $\alpha_0, \alpha_1, \dots, \alpha_r$, причем $\alpha_0 = \alpha$, а $\alpha_r = \beta$. Итак, у нас есть последовательность наборов, где каждый следующий «больше»³ предыдущего, и отличаются они только в одном месте. А еще, $f(\alpha_0) = 1$, а $f(\alpha_r) = 0$. Тогда, если мы будем идти от α_0 к α_r , то мы когда-нибудь встретим 2 таких соседних набора α_i и α_{i+1} , что $f(\alpha_i) = 1$ и $f(\alpha_{i+1}) = 0$, просто потому что когда-то единицы должны будут смениться на нули. Тогда, раз эти наборы соседние в нашей цепочке, они будут отличаться только во 1 переменной, пусть это x_j , т.е. $\alpha_i = (\sigma_1, \dots, \sigma_{j-1}, 0, \sigma_{j+1}, \dots, \sigma_n)$ и $\alpha_{i+1} = (\sigma_1, \dots, \sigma_{j-1}, 1, \sigma_{j+1}, \dots, \sigma_n)$, где все σ — просто константы, или 0, или 1. Тогда мы сможем составить функцию $h(x) = f(\sigma_1, \dots, \sigma_{j-1}, x, \sigma_{j+1}, \dots, \sigma_n)$, и тогда

$$\begin{aligned} h(0) &= f(\alpha_i) = 1 \\ h(1) &= f(\alpha_{i+1}) = 0 \end{aligned}$$

Тогда $h(x) = \bar{x}$, какую мы и пытались найти. □

3.13 Класс L

Если коротко, то функция называется *линейной*, если в ее полиноме Жегалкина отсутствует умножение, т.е. есть только сложение. Например, полином Жегалкина $x_1 + x_3 + 1$ представляет линейную функцию, а $x_1x_2 + x_1 + x_3$ — нет. Класс же линейных функций называется L , и это, пожалуй, — самый простой из всех 5 (да, мы дошли до последнего). Он, как и всегда

³В отношении предшествования, не как двоичное число, ну вы поняли.

замкнут, но доказательство по факту почти отсутствует: если во всех составных функциях есть только сложение, то откуда появиться умножению? Также, насчет количества, каждая из переменных (а еще 1) может или быть в полиноме Жегалкина, а может и не быть, поэтому линейных функций всего 2^{n+1} , что очень мало по сравнению с остальными классами. А теперь, к самому главному:

Лемма (о нелинейной функции). *Если у нас есть нелинейная функция f , то подставляя вместо переменных $0, 1, x_1, \bar{x}_1, x_2, \bar{x}_2$, а еще, возможно, добавив отрицание ко всей функции, мы сможем получить функцию $h(x_1, x_2) = x_1 \& x_2$.*

Доказательство. Если функция f — нелинейная, то значит в ее полиноме Жегалкина присутствует хотя бы одно умножение переменных. Мы можем предположить, что это переменные x_1 и x_2 , потому что иначе мы можем их просто переименовать, нам никто не запрещает. Что ж, тогда, если мы вынесем в полиноме Жегалкина x_1 и x_2 за скобки, мы получим примерно

$$\begin{aligned} f(x_1, \dots, x_n) &= x_1 \cdot x_2 \cdot P_1(x_3, \dots, x_n) \\ &+ x_1 \cdot P_2(x_3, \dots, x_n) \\ &+ x_2 \cdot P_3(x_3, \dots, x_n) \\ &+ P_4(x_3, \dots, x_n), \end{aligned}$$

где P_1, P_2, P_3, P_4 — тоже какие-то полиномы Жегалкина, причем $P_1 \neq 0$, потому что мы знаем, что умножение $x_1 \cdot x_2$ обязательно есть в полиноме Жегалкина нашей функции. Еще, все эти полиномы не должны зависеть от x_1, x_2 , потому что если они зависели, мы бы вынесли их за скобки. Раз наше P_1 не 0, то существует какой-то набор, $(\gamma_3, \dots, \gamma_n)$, на котором он равен 1. Тогда получим, что

$$f(x_1, x_2, \gamma_3, \dots, \gamma_n) = x_1 \cdot x_2 + \alpha x_1 + \beta x_2 + \delta,$$

где

$$\begin{aligned} \alpha &= P_2(\gamma_3, \dots, \gamma_n) \\ \beta &= P_3(\gamma_3, \dots, \gamma_n) \\ \delta &= P_4(\gamma_3, \dots, \gamma_n). \end{aligned}$$

Что нам это дает? Пока ничего. Нам нужно как-то сделать так, чтобы x_1 и x_2 без умножения в нашей функции отсутствовали. Этого можно добиться, если мы заменим x_1 на $x_1 + \beta$, а x_2 на $x_2 + \alpha$. Это прибавление константы — просто возможное отрицание⁴, нам это разрешается, мы можем заменять

⁴ $x + 1 = \bar{x}, x + 0 = x$

x_1 на $\overline{x_1}$ и x_2 на $\overline{x_2}$. Итак, тогда мы получим

$$\begin{aligned} f(x_1 + \beta, x_2 + \alpha, \gamma_3, \dots, \gamma_n) &= (x_1 + \beta)(x_2 + \alpha) + \\ &\quad + \alpha(x_1 + \beta) + \beta(x_2 + \alpha) + \delta \\ &= x_1x_2 + \cancel{\alpha x_1} + \cancel{\beta x_2} + \cancel{\alpha\beta} + \\ &\quad + \cancel{\alpha x_1} + \cancel{\alpha\beta} + \cancel{\beta x_2} + \alpha\beta + \delta \\ &= x_1x_2 + (\alpha\beta + \delta). \end{aligned}$$

Отсюда получим, что если мы возможно добавим к функции отрицание (а именно сделаем $+(\alpha\beta + \delta)$), то мы получим как раз произведение или иначе, конъюнкцию, т.е. $h(x_1, x_2) = f(x_1 + \beta, x_2 + \alpha, \gamma_3, \dots, \gamma_n) + (\alpha\beta + \delta) = x_1 \& x_2$. Так как прибавление константы - просто возможное отрицание, эта функция соответствует условиям теоремы. \square

3.14 Критерий полноты в P_2

Вот мы и дошли до того, зачем все эти классы были нужны. Дело в том, что

Критерий полноты в P_2 . Система ФАЛ B является полной тогда и только тогда, когда для каждого из 5 классов (T_0, T_1, S, M, L) в ней есть такая функция, которая в этом классе не лежит. Другая формулировка: Система ФАЛ B является полной тогда и только тогда, когда она не содержится полностью ни в одном из 5 классов.

Поясню, что это значит: вот у нас есть система $B = \{x_1 \rightarrow x_2, \overline{x}\}$. Изучив эти функции, мы узнаем, что $x_1 \rightarrow x_2$ лежит в классе T_1 и не лежит во всех остальных, а \overline{x} лежит в классах S и L и не лежит в T_0, T_1 и M . Т.е., если мы проверим каждый из классов, в системе найдется функция (импликация или отрицание), которая в нем не лежит. Например, для всех классов, кроме T_1 это может быть импликация, а для T_1 — отрицание. Как видите, одна функция может не лежать сразу в нескольких классах и этого достаточно.⁵ Тогда по нашей теореме эта система является полной.

Доказательство. Из-за «тогда и только тогда» эта теорема на самом деле 2 в 1: с одной стороны, из того, что система полная, следует, что есть эти функции, а с другой: из существования функции следует полнота системы.

- $[B] = P_2 \implies \forall \mathfrak{A} \in \{T_0, T_1, S, M, L\} (B \not\subseteq \mathfrak{A})$ ⁶

Итак, у нас есть полная система B . Чтобы доказать, что она не содержится ни в одном из классов, предположим противное. Пусть у нас все таки есть такой класс $\mathfrak{A} \in \{T_0, T_1, S, M, L\}$, в котором B полностью содержится. Так как $B \subseteq \mathfrak{A}$, то $[B] \subseteq [\mathfrak{A}]$, что достаточно

⁵ Система $B = \{\}$ вообще состоит только из 1 функции, но штрих Шеффера не лежит сразу во всех классах

⁶ Буква \mathfrak{A} — это готическое написание буквы А. Не спрашивайте.

логично: если наша система — подмножество другой системы, то комбинируя функции нашей системы B мы не сможем получить больше, чем если будем комбинировать функции всей системы \mathfrak{A} . Каждый из 5 классов — замкнутый, мы это доказали, и при этом он не является P_2 (очевидно, потому что существуют функции, которые не в нем). Тогда мы получим, что $[B] = P_2$ (по условию) и $[\mathfrak{A}] \neq P_2$, но при этом $[B] \subseteq [\mathfrak{A}]$, что является противоречием. Значит наше предположение неверно и эта часть теоремы доказана.

- $\forall \mathfrak{A} \in \{T_0, T_1, S, M, L\} (B \not\subseteq \mathfrak{A}) \implies [B] = P_2$

Вот тут интереснее, если наша система не содержится ни в одном из классов (для каждого из классов в нашей системе B есть хотя бы 1 функция, которая не в этом классе), то она — полная. Что ж, если так, давайте назовем все эти функции, причем так, что $f_0 \notin T_0, f_1 \notin T_1, f_S \notin S, f_M \notin M, f_L \notin L$, и все они лежат в B . Единственный способ доказать, что эта система полная — это *теорема редукции*, нам нужно при помощи этих 5 (а может и одной, но под 5 разными именами, кто знает) функций выразить все функции некоторой полной системы, например $\{x_1 \& x_2, \bar{x}\}$, и тогда наша система будет считаться полной. Правда нам еще было бы неплохо выразить константы, они были бы полезны. С этого и начнем.

1. Рассмотрим функцию $h_0(x) = f_0(x, x, \dots, x)$. Да, так можно делать, мы просто одну переменную записали во все входы f_0 . По нашей аналогии с микросхемами — мы взяли один входной провод и подключили его ко всем входам f_0 , теперь у нас есть один входной провод и один выходной (от f_0), т.е. функция от одной переменной, $h_0(x)$. Так вот, рассмотрим, чему эта функция равна на 0 и 1:

$$\begin{aligned} h_0(0) &= f_0(0, \dots, 0) = 1 \\ h_0(1) &= f_0(1, \dots, 1) = ? \end{aligned}$$

Так как наша функция *не* сохраняет 0, то на нулевом наборе она обязана быть равна 1. Но чему она равна на единичном наборе — мы не знаем. Если она равна 1, то мы получили $h_0(x) = 1$, нашу искомую константу, а если она равна 0, то $h_0(x) = \bar{x}$, что было бы приятным сюрпризом. Правда нам все равно придется как-то получить константную 1. К счастью, у нас есть *лемма о несамодвойственной функции*, и несамодвойственная функция f_S . Если мы подставим вместо переменных x (что мы всегда можем) и $\bar{x} = h_0(x)$, то мы обязаны получить какую-то константу, но так как у нас есть отрицание, мы сможем получить и другую, просто взяв отрицание того, что получится. Итак, после этого шага у нас обязательно есть константа 1, но нам могло повезти и мы могли получить еще и 0 и \bar{x} . Правда на везение полагать-

ся в математике не стоит, так что будем считать, что у нас есть только 1.

2. Функция $h_1(x) = f_1(x, x, \dots, x)$ работает аналогично.

$$\begin{aligned} h_1(0) &= f_1(0, \dots, 0) = ? \\ h_1(1) &= f_1(1, \dots, 1) = 0 \end{aligned}$$

У нас снова варианты, если $f_1(0, \dots, 0) = 0$, то у нас будет константа 0, а если $f_1(0, \dots, 0) = 1$, то у нас будет $h_1(x) = \bar{x}$, и мы легко сможем получить $0 = \bar{1} = h_1(1)$. Итак, у нас снова возможно есть отрицание, а возможно все еще нет и есть только обе константы. Считаем, что его нет, и идем дальше.

3. У нас есть немонотонная функция f_M и лемма о немонотонной функции, по которой мы можем подставить в нее 0, 1 и x и получить \bar{x} . Так и сделаем, итого у нас есть константы и отрицание, для полного счастья нужна только конъюнкция.
4. У нас есть нелинейная функция f_L и лемма о нелинейной функции, по которой мы можем подставить в нее константы, переменные и отрицание и еще возможно взять отрицание от всей функции, и мы получим $x_1 \& x_2$.

Итак, мы смогли выразить \bar{x} и $x_1 \& x_2$ через наши функции f_0, f_1, f_S, f_M, f_L , которые все лежат в B . Тогда по теореме редукции, наша система — полная.

□

3.15 Предполные классы ФАЛ

А теперь, пожалуй, самое красивое из того, что было до сих пор. Дело в том, что наши классы T_0, T_1, S, M, L — единственные системы функций, являющиеся *предполными*, или, по другому, *классами Поста*. По непонятным причинам, Костенко не упоминает про этого человека, Эмиля Поста, открывшего эти классы и составившего полную структуру, описывающую все замкнутые классы ФАЛ, которая называется *решеткой Поста*. Эта структура крайне красивая и я не понимаю, почему про нее нельзя было хотя бы упомянуть, но, пожалуй, она слишком сложна для 1 курса. Посмотрим, может когда-нибудь Костенко про нее и расскажет.

Так вот, *предполные классы*. Класс B называется *предполным*, если он не полный (логично, правда?) и при этом добавление абсолютно любой функции извне делает его полным. Такие классы, которым для полноты не хватает только одной функции, причем любой.

Теорема. *Предполные классы замкнуты.*

Доказательство. Ну, очевидно, что предполный класс B содержится в собственном замыкании $[B]$ (очевидно, что если нам доступны функции из класса B , то мы можем просто взять их и записать). Предположим, что он незамкнутый, т.е. $[B] \neq B$. Отсюда $B \subsetneq [B]$ (содержится в замыкании, но не равен). Пусть тогда функция f лежит в $[B]$, но не в B . Если мы добавим f к B , то по определению предполного класса, мы должны получить полную систему, так? $[B \cup \{f\}] = P_2$. Но при этом $B \cup \{f\} \subseteq [B]$, а значит по теореме редукции $[B \cup \{f\}] = [B]$, пришли к противоречию, значит у всех предполных классов $[B] = B$. \square

Теорема. Классы T_0, T_1, S, M, L — предполные.

Доказательство. Можно провести доказательство только для T_0 , все остальные аналогичны. То, что класс T_0 — не полный, мы уже доказывали ранее, нужен только второй пункт — при добавлении любой функции не из T_0 мы получим полную систему. Для доказательства этого нужно использовать критерий полноты в P_2 , или *критерий Поста*. У нас имеется одна функция не из T_0 , нужно просто доказать, что в T_0 содержится хотя бы по одной не сохраняющей 1, несамодвойственной, немонотонной и нелинейной функции. Как f_1, f_S, f_M подойдет $x_1 + x_2$, которая лежит в T_0 , а как f_L подойдет $x_1 \vee x_2 = x_1 x_2 + x_1 + x_2$. Итак, в $T_0 \cup \{f_0\}$ есть все 5 нужных функций, значит эта система будет полной, а класс T_0 — предполным. \square

Теорема. Классы T_0, T_1, S, M, L — единственные предполные классы.

Доказательство. Рассмотрим любой предполный класс B . По отношению к классам $\mathfrak{A} \in \{T_0, T_1, S, M, L\}$ он может

1. Лежать в одном из \mathfrak{A} . Если $B \subset \mathfrak{A}$, а $f \in \mathfrak{A} \setminus B$, то $B \cup \{f\} \subseteq \mathfrak{A}$, а значит $[B \cup \{f\}] \subseteq [\mathfrak{A}] = \mathfrak{A} \neq P_2$, т.е. такой класс не может быть предполным.
2. Являться одним из \mathfrak{A} . Возможно, таких случаев ровно 5.
3. Не лежать ни в одном из \mathfrak{A} . Тогда по критерию полноты в P_2 такой класс будет являться полным, значит предполным он быть не может.

Итого, предполными могут быть только один из 5 классов T_0, T_1, S, M, L . \square

3.16 Схемы из функциональных элементов

Я про это уже рассказывал в самом начале, но формулы можно представлять как схемы из «микросхем» или «логических элементов».

- У каждой схемы есть несколько входов (переменных)
- На вход каждого элемента подается ровно 1 сигнал, или от входа всей схемы, или от выхода другого элемента.

- Выходы можно расщеплять (один выход может идти на много разных входов других элементов)
- Неиспользованные выходы элементов становятся общими выходами схемы
- Циклы запрещены (очевидно, с циклами появляются такие сложные вещи, как триггеры)

Так как у любой функции есть СДНФ (кроме 0, но 0 мы можем записать как $x \& \bar{x}$), мы можем ее представить формулой над $\{\&, \vee, \bar{x}\}$, а формулу записать схемой, значит любую функцию можно представить как схему из элементов $\{\&, \vee, \bar{x}\}$. И на этом, пожалуй, все.

Глава 4

Графы

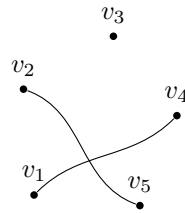
4.1 Определение и способы представления

Что ж, пожалуй самая интересная глава в 1 семестре, на мой взгляд. Графы — это просто набор вершин (точек), которые связаны дугами (линиями) каким-то образом. Причем важно не расположение точек, не форма линий, а только сам факт связи одной точки с другой. А теперь, формально, $G = (V, U)$, где V — множество вершин графа, а U — множество ребер. Это значит, что *графом* называются множество вершин и множество ребер *вместе*, по отдельности они графами не считаются. Вершины графа могут быть чем угодно, хоть числами, хоть буквами, хоть просто точками, но их должно быть конечное количество, значит мы должны быть способны записать их все: $V = \{v_1, v_2, \dots, v_n\}$. Ребер тоже конечное количество: $U = \{u_1, u_2, \dots, u_m\}$, причем $u_i = (a, b)$, где a и b — вершины графа, которое соединяет это ребро. Графы бывают ориентированные и неориентированные, так как эти ребра можно рассматривать по разному: или это значит, что можно двигаться из a в b и из b в a , или только из a в b . Когда можно двигаться в любом направлении, то $u_i = (a, b), (b, a)$ ¹ — неориентированное ребро, а если только в одном, то $u_i = (a, b)$ — ориентированное ребро. Если в графе нет ориентированных ребер, то он неориентированный, если есть хоть одно — ориентированный. Еще между двумя вершинами может быть не больше 1 ребра, иначе это не граф, а *мультиграф*, а мы такие не проходим. Ну и да, степенью вершины называется число ребер, которые из нее выходят, обозначается $d(v_i)$.

Насчет представлений графа:

1. *Геометрическое*: граф можно просто нарисовать. Вершины — просто точки, ребра — просто линии, которые их соединяют, причем пересечения мы игнорируем. Примерно вот так:

¹Без понятия, как это возможно с математической стороны, ну да ладно.



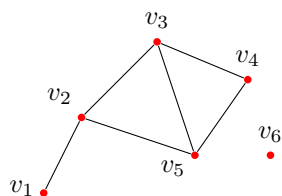
2. *Матрица смежности*: мы можем просто записать все связи в таблице: если есть связь, пишем 1, если нет, пишем 0. Граф выше можно записать вот так:

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	0	1	0
v_2	0	0	0	0	1
v_3	0	0	0	0	0
v_4	1	0	0	0	0
v_5	0	1	0	0	0

У неориентированных графов матрицы смежности симметричны относительно главной диагонали, причем петли, связи вершин с самими собой, располагаются на главной диагонали.

3. *Списки смежности*: мы можем просто записать список всех вершин, с которыми связана каждая вершина, и по такому списку для каждой.

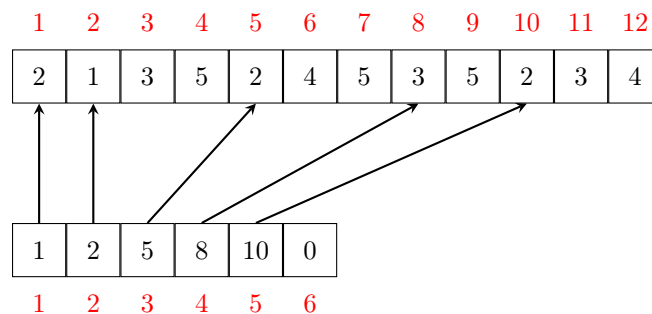
Для примера нам нужен граф посложнее, например



Тогда списками смежности для этого графа будут:

$v_1 : v_2$	$1 : 2$
$v_2 : v_1, v_3, v_5$	$2 : 1, 3, 5$
$v_3 : v_2, v_4, v_5$	$3 : 2, 4, 5$
$v_4 : v_3, v_5$	$4 : 3, 5$
$v_5 : v_2, v_3, v_4$	$5 : 2, 3, 4$
$v_6 :$	$6 :$

Слева — сами вершины, справа — только цифры. На этом собственно и все, но Костенко зачем то предлагает еще и способ хранения таких списков в памяти компьютера. Мы просто записываем все списки подряд в один массив, а во втором размещаем индексы начал этих списков или 0, если список пустой. Вот пример для этого графа:



4.2 Пути и циклы в графах

Этот раздел — в основном куча определений. Итак, если у нас есть граф $G = (V, U)$, $V = \{v_1, \dots, v_n\}$, $U = \{u_1, \dots, u_m\}$, то можно ввести следующие понятия:

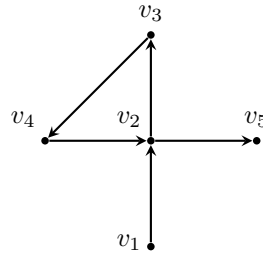
Путь в графе G — последовательность вершин, $W = (v_{i_1}, v_{i_2}, \dots, v_{i_k})^2$, такая, что из каждой вершины можно «перейти» в следующую по ребру, т.е. для всех j (кроме последнего) между v_{i_j} и $v_{i_{j+1}}$ есть ребро. У этого пути длина $k - 1$ (вершин k , но мы считаем ребра, а ребер $k - 1$)

Множество ребер пути W — множество всех ребер, через которые проходит путь. Обозначается $E(W)$. Напоминаю, что путь — это последовательность вершин, и это не то же самое, что множество ребер.

Элементарный путь — такой путь, в котором все вершины разные (не проходит через одну вершину 2 раза).

Простой путь — такой путь, который не проходит ни одно ребро по 2 раза.

Элементарный путь и простой путь — разные понятия. Например, путь $v_1, v_2, v_3, v_4, v_2, v_5$ не является простым, потому что вершина v_2 проходится 2 раза, но каждое из ребер 1-2, 2-3, 3-4, 2-4, 2-5 — уникальное. Рисунок этого пути:



Цикл — такой путь, который начинается и заканчивается в одной вершине (логично, правда?)

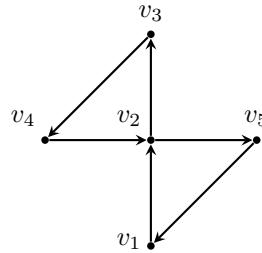
Элементарный цикл — такой цикл, в котором все вершины разные, не учитывая *первую и последнюю*, которые должны быть одинаковыми, чтобы это был цикл.³

Простой цикл — это такой цикл, который простой путь (...да), т.е. такой цикл, который не проходит через ребра по 2 раза.

Элементарный и простой циклы — тоже разные понятия, цикл $v_1, v_2, v_3, v_4, v_2, v_5, v_1$ — простой, но не элементарный:

²Такие индексы значат, что i_1, \dots, i_k — некоторая последовательность чисел, типа 1, 6, 2, 8, тогда путь будет v_1, v_6, v_2, v_8 . Вершины — это не просто числа, это какие-то нумерованные объекты.

³Общий смысл остался тем же, это такой цикл, который проходит через каждую вершину только 1 раз, просто здесь нужна эта поправка, потому что формально если цикл вышел из вершины и вернулся в нее же, то для пути это будет считаться как 2 разных прохода.



Теорема. Если между вершинами есть некоторый путь, то между ними можно провести и элементарный путь.

Доказательство. Ну, если между вершинами есть некоторый путь, то нам может повезти и он и будет элементарным. Тогда все было бы хорошо и теорема была бы доказана, но у нас есть еще один случай.

Пусть между вершинами a и b есть некоторый неэлементарный путь. Что это значит? Это значит, что какая то из вершин в пути проходится дважды: мы к ней пришли, ушли, потом к ней снова вернулись, и пошли дальше. Мы по факту прошли петлю. Если бы мы не пошли по этой петле, а сразу пошли дальше, и сделаем так для всех петель на нашем пути, то мы получим элементарный путь.

А теперь формально, у нас есть путь $W = v_{i_1}, \dots, v_{i_k}$. Раз он неэлементарный, то какую то вершину мы проходим дважды (это начало петли). Пусть эти 2 прохождения нашей вершины будут под номерами s и p , т.е. $v_{i_s} = v_{i_p}$ — начало нашей петли, а все, что между i_s и i_p — и есть наша петля. Если мы удалим из пути все вершины с номерами от i_{s+1} до i_p , то мы удалим петлю. Если путь все еще неэлементарный, мы можем продолжить удалять петли, пока они не закончатся (а они закончатся, все наши пути — конечные). После всего этого удаления мы получим наш элементарный путь между a и b . \square

4.3 Транзитивное замыкание графа

Название говорит само за себя: транзитивным замыканием графа G называется *другой* граф G^* , у которого проведены все возможные дуги по транзитивности. Это значит, что если в графе G между вершинами a и b есть *путь*, то в графе G^* между ними имеется *ребро*.

Попробуем вычислить матрицу смежности графа G^* , если у нас есть матрица смежности G : A . Для этого нужно записать понятие «транзитивной дуги» математически. Элемент A с индексами i и j обозначим σ_{ij} , и если он есть, то между v_i и v_j имеется ребро. Мы должны соединить вершины v_i и v_j , если существует какая то вершина v_k , которая соединена с обоими, т.е. $\sigma_{ik} \& \sigma_{kj}$. Так как ребро будет $(\sigma_{ij} = 1)$ тогда, когда хотя существует хотя бы одно такое k , нам нужно взять дизъюнкцию этих выражений для всех возможных k . И да, если наша матрица $A = A^1$ обозначает пути длины 1

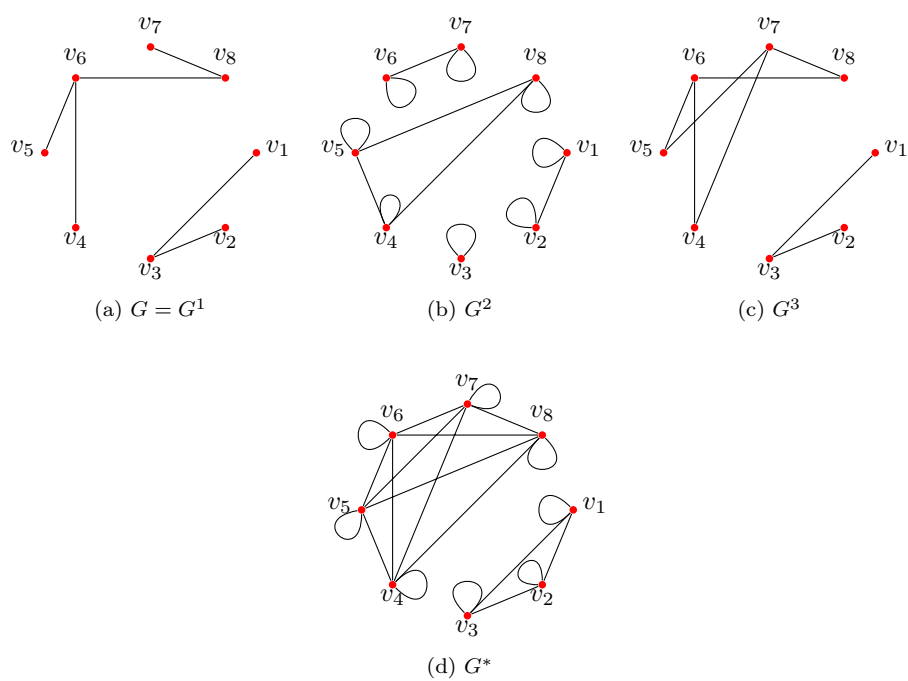


Рис. 4.1: Транзитивное замыкание

(1 ребро), то после одного шага транзитивности мы получим матрицу A^2 , которая обозначает пути длины 2 (2 ребра, ik и kj). Так вот:

$$\sigma_{ij}^2 = \bigvee_{k=1}^n (\sigma_{ik}^1 \& \sigma_{kj}^1).$$

Эта формула очень похожа на

$$a'_{ij} = \sum_{k=1}^n (a_{ik} \cdot a_{kj}),$$

которая представляет возведение обычной матрицы из линейной алгебры в квадрат. Тут то же самое, но $\&$ и \vee вместо \cdot и $+$. Аналогично получим A^3 , воспользовавшись

$$\sigma_{ij}^3 = \bigvee_{k=1}^n (\sigma_{ik}^2 \& \sigma_{kj}^1).$$

Эта матрица представляет собой пути длины 3, т.е. в графе, который она представляет, G^3 , (еще раз, это *не* оригинальный граф G), между вершинами есть ребро, если между ними в графе G есть какой-то путь длины 3. Также можно определить матрицу $A^0 = E$, единичная матрица, из каждой вершины есть путь длины 0 в нее же саму (постоять на месте).

Итак, нам нужна матрица, которая представляет *все* пути. По теореме, если между вершинами есть какой-то путь, то между ними есть и элементарный путь, а длина элементарного пути может быть не больше $n - 1$. Тогда, если мы объединим все матрицы с длинами путей до $n - 1$, мы получим матрицу смежности $A^* = A^0 \vee A^1 \vee A^2 \vee \dots \vee A^{n-1}$, являющейся матрицей смежности нашего искомого транзитивного замыкания (G^*). Пример см. на рисунке 4.1.

4.4 Геометрическая реализация графа

Хоть мы и сказали, что пересечения нам не важны, было бы неплохо так рисовать граф, чтобы в нем не было лишних пересечений ребер, чтобы единственными общими точками были вершины, причем только те, которые нужно. Такое изображение называется *геометрической реализацией графа*.

Теорема. *Любой граф можно реализовать в трехмерном пространстве.*

Доказательство. Раз это просто теорема, мы ищем не самый удобный способ реализации графа, а хоть какой-нибудь. Так что давайте нарисуем прямую и расположим все вершины на ней (см. рисунок 4.2).

Дальше, нам нужно провести ребра так, чтобы они пересекались только в вершинах, и только в своих конечных точках (не проходили через лишние вершины). Проведем какую-то плоскость через нашу прямую, и проведем в ней ребро u_1 так, чтобы оно касалось прямой только в начале и в конце

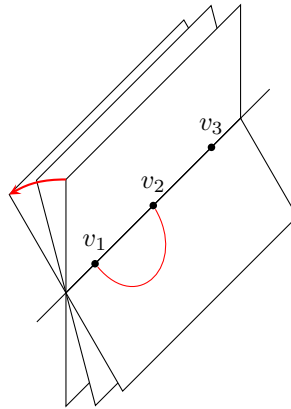


Рис. 4.2: Трехмерная реализация графа

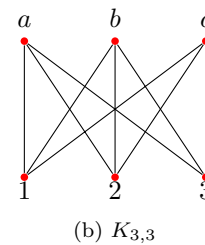
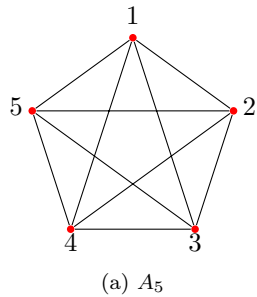


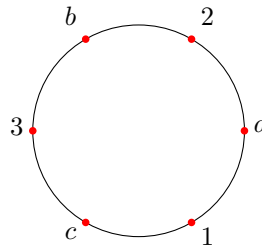
Рис. 4.3: Непланарные графы

(т.е. в начальной и конечной вершине ребра). Чтобы второе ребро точно не пересекалось с этим, повернем нашу плоскость на некоторый угол и нарисуем ребро u_2 . Будем так делать, пока ребра не закончатся, они не будут пересекаться нигде, кроме нашей линии (потому что разные плоскости), а нашу линию они пересекают только в начальных и конечных точках, т.е. это как раз то, что мы искали. Так какой угол? При повороте на 180° наша плоскость вернется в изначальное положение. Всего ребер (и плоскостей) m . Тогда нам нужно поворачивать нашу плоскость на $\frac{180^\circ}{m}$. Мы поворачиваем всего $m - 1$ раз (изначальная плоскость без поворота), значит мы не вернемся в изначальное положение и все хорошо. \square

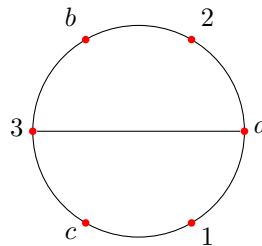
А вот с двухмерным пространством (плоскостью) все намного сложнее. Оказывается, не все графы можно реализовать на плоскости. Те, которые можно, называются *планарными*, те, которые нельзя, — *непланарными*. Существуют 2 основных непланарных графа, так называемые A_5 и $K_{3,3}$ (см. рисунок 4.3)

Теорема. $K_{3,3}$ — непланарный.

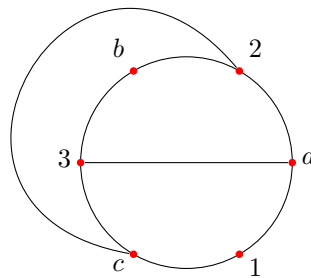
Доказательство. Нам нужно попытаться реализовать этот граф на плоскости и доказать, что это невозможно. Что ж, попробуем. Раз пересечения запрещены, то какой бы цикл на графе мы ни взяли, он не должен быть самопересекающимся, т.е. его можно представить просто как окружность. Возьмем цикл $a, 2, b, 3, c, 1, a$ и нарисует его:



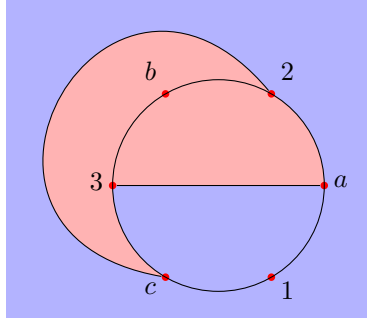
Окей, какие ребра нам остались для завершения $K_{3,3}$? $a3$, $b1$, $c2$. Из-за их полной симметричности, мы можем выбрать любое из них и продолжать далее. Выберем $a3$. Где его провести, внутри или снаружи? Наша окружность разбила плоскость на 2 части, но на самом деле сейчас нет никакой разницы, где именно мы его проведем, так что давайте проведем внутри:



Ребра будут по-прежнему симметричны, так что выберем любое, $c2$. Внутри мы его провести не можем, так что снаружи:



Теперь, если мы попробуем соединить b и 1 , мы обнаружим, что они располагаются в 2 разных областях, ограниченных нашими проведенными ребрами, которые нельзя пересекать:



Значит, этот граф нельзя реализовать на плоскости. \square

Аналогично доказывается и для графа A_5 .

4.5 Критерий планарности графа

Еще больше определений!

Подграф графа G — другой граф $(G' = (V', U'))$, который получается, если удалить из G некоторые вершины и ребра, т.е. $V' \subseteq V, U' \subseteq U$.

Изоморфные графы — два графа $G_1 = (V_1, U_1)$ и $G_2 = (V_2, U_2)$ называются изоморфными, если они «обладают одной структурой», т.е. если мы просто переименуем вершины (и ребра оставим там же) графа G_1 , то получим граф G_2 . Формально, если у нас есть некая «переименовывающая функция» $h : V_1 \rightarrow V_2$, то если между вершинами a и b есть ребро в V_1 , то между $h(a)$ и $h(b)$ (переименованными их версиями) тоже есть ребро в V_1 : $(a, b) \in U_1 \rightarrow (h(a), h(b)) \in U_2$. Пример изоморфных графов см. на рисунке 4.4⁴, функция переименования простая:

$$\begin{aligned} h("a") &= 1 \\ h("b") &= 2 \\ &\vdots \\ h("h") &= 8 \end{aligned}$$

Разбиение ребра графа — операция, которая «разбивает» ребро на 2, ставя вершину посередине ребра. Формально, к множеству вершин добавляется одна новая ($V' = V \cup \{c\}$), а из множества ребер удаляется одно старое и добавляется два новых ($U' = U \setminus \{u\} \cup \{u', u''\}$, причем если $u = (a, b)$, то $u' = (a, c), u'' = (c, b)$).

Разбиение графа — другой граф, который можно получить из данного, разбивая в нем ребра какое то количество раз.



Рис. 4.4: Изоморфные графы

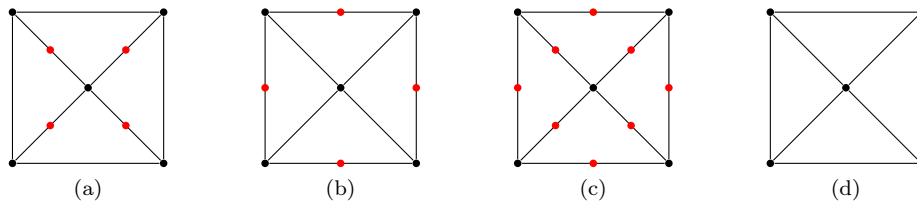


Рис. 4.5: Гомеоморфные графы (все 4)

Гомеоморфные графы — графы, которые можно разбить так, что они будут изоморфными. Т.е. общая структура этих графов одна и та же, если не учитывать все вершины, через которые проходят только 2 ребра. Пример гомеоморфных графов см. на рисунке 4.5

Критерий планарности графа. *Граф является планарным тогда и только тогда, когда он не содержит подграфов, гомеоморфных A_5 или $K_{3,3}$.*

Это, разумеется, замечательная теорема, но доказывать мы ее, конечно, не будем.

4.6 Деревья

Больше определений богу определений!

Связный граф — такой граф, между любыми 2 вершинами которого есть путь.

Компонента связности — связный подграф, который нельзя расширить, в том смысле, что это весь связный «островок» в большом графе, а не маленький кусочек этого «островка».

⁴ «Боевой робот спускается сверху» (К.И. Костенко)

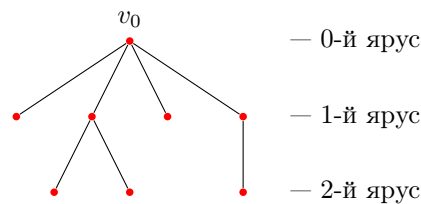
Циклическое ребро — такое ребро, через которое проходит какой-то *цикл длины не меньше, чем 3*. Далее я буду называть такие циклы «нормальными» или «полноценными», потому что циклы длины 0 есть всегда, это просто «постоять на месте и все», циклы длины 1 — это петли, циклы длины 2 — это «пройтись по ребру туда-сюда», есть всегда, когда есть ребра, и только начиная с длины 3 циклы уже больше похожи на циклы, поэтому «нормальные» или «полноценные».⁵

Дальше мы будем говорить исключительно про *неориентированные связные графы без петель*, правда нам все равно надо будет их как-то называть, так что НСГБП.

Дерево — такой НСГБП, в котором нет циклических ребер (т.е. в котором нет «нормальных» циклов)

Корневое представление дерева

Мы можем рисовать деревья не как обычные графы, а более структурированным способом, по ярусам. Выберем одну любую вершину, назовем ее v_0 , она — корень дерева. Мы располагаем ее на *нулевой ярус*. Далее всех ее соседей поместим на 1-й ярус, всех соседей 1-го яруса поместим на 2-й ярус и т.д., пока вершины не закончатся. Примерно так:



Это представление — аналогия с генеалогическим древом. У вершин могут быть дети/потомки, это те вершины, которые связаны с данной и лежат на 1 ярус ниже, и родители/предки, которые лежат на 1 ярус выше.

Свойства корневого представления дерева

1. Во всяком дереве есть листья — вершины, у которых нет потомков, из которых не ведут ребра на следующий ярус. только на ярус вверх.
2. У каждой вершины ровно 1 предок (кроме корня, у корня нет предков)

Доказательство. Доказываем методом от противного. Пусть у нас есть вершина a , и у нее есть 2 предка, b и c . Раз дерево — связное, мы можем провести путь из v_0 в b и путь из v_0 в c (см. рисунок 4.6). Раз они начинаются из одной точки, то хотя бы одна общая вершина у

⁵Не говорите про эту терминологию Костенко, пожалуйста.

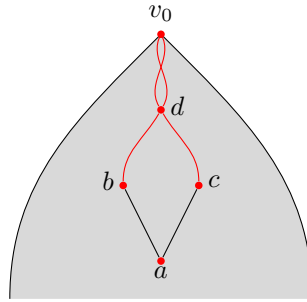


Рис. 4.6: Свойство корневого представления

них будет, но возможно их больше. Пусть d — последняя такая общая вершина этих 2 путей. А теперь посмотрим, у нас появился цикл $abdca$, длины не меньше, чем 4, значит это точно не дерево, противоречие. \square

3. У всех деревьев вершин на 1 больше, чем ребер: $|V| = |U| + 1$, и наоборот, если $|V| = |U| + 1$, то граф — дерево.

(а) G — дерево, тогда $|V| = |U| + 1$

Доказательство. Итак, у нас есть граф $G = (V, U)$ и он дерево. Будем по очереди удалять листья вместе с ребрами, которые в них входят. Каждый шаг мы удаляем 1 вершину и 1 ребро, значит количество удаленных вершин при таких действиях всегда равно количеству удаленных ребер (это называется *инвариантом*, когда что-то неизменно верно во время какого то действия. Инварианты особенно полезны в физике, но и здесь тоже, как видите). После всех этих удалений у нас останется только корень, 1 вершина, но удалили мы столько же вершин, сколько ребер, значит вершин было на 1 больше. \square

(b) Для НСГБП G верно $|V| = |U| + 1$, тогда он - дерево.

Доказательство. Предположим, что этот граф не дерево, тогда в нем есть циклические ребра. Будем по очереди удалять их, пока они не закончатся. При таком действии связность графа не теряется, так как любой путь, проходящий через удаляемое ребро, может это ребро «обойти» через остаток цикла, который проходил через это ребро. После удаления k ребер у нас останется НСГБП G^k без циклических ребер, т.е. дерево. Тогда нам известно, что:

$$\begin{aligned} |V| &= |U| + 1 \text{ (из условия)} \\ |V| &= |U^k| + 1 \text{ (т.к. получилось дерево)} \\ |U^k| &= |U| - k, k > 0. \end{aligned}$$

Такое все вместе невозможно, значит предположение было неверно и G — дерево. \square

4.7 Циклы Эйлера

Если некий цикл проходит абсолютно все ребра в графе по 1 разу, то он называется *циклом Эйлера*. Очевидно, что такой цикл будет являться простым, но не обязательно элементарным. Другое определение: граф называется *четным*, если степени всех его вершин четные, т.е. во все его вершины входит четное количество ребер.

Теорема (Эйлера). *В графе есть цикл Эйлера тогда и только тогда, когда он четный.*

Эта теорема снова 2 в 1:

1. Если есть цикл Эйлера, то граф — четный.

Доказательство. Наш цикл проходит через абсолютно все ребра. Это значит, что если он «заходит» в какую то вершину, то после этого он из нее и «выходит». Таким образом, все ребра каждой из вершин идут парами: одно заходит в вершину, другое выходит. Тогда у всех вершин будут четные степени, т.е. граф будет четным. \square

2. Если граф четный, то там есть цикл Эйлера.

Доказательство. Для начала нужно доказать, что в таком графе вообще есть «нормальные» циклы.

Возьмем произвольную вершину v_0 . Раз граф — связный, то из нее ведет ребро, например, в v_1 . Раз граф четный, то в v_1 ведет четное число ребер, т.е. есть хотя бы еще одно. Это ребро, ведет, например, в v_3 . Аналогично, из v_3 тоже ведет ребро. Оно может вести или в v_1 (тогда у нас есть наш цикл), или в новую v_4 . Из v_4 или в v_1 или в v_2 (тогда нашли цикл), или в новую v_5 . Так как вершин у нас конечное число, мы не можем бесконечно переходить в новую, и тогда мы обязаны когда-нибудь найти наш цикл. Более того, этот цикл будет элементарным, потому что мы прошли через все вершины цикла только по 1 разу.

Чтобы доказать, что цикл Эйлера есть, нужно его построить. Будем строить его индукцией по числу ребер.

- (а) *Базис индукции:* $m = 0$. В таком графе нет ребер, т.е. он — просто 1 вершина. Тогда циклом Эйлера будет просто «постоять на месте», т.к. ребра обходить не надо.

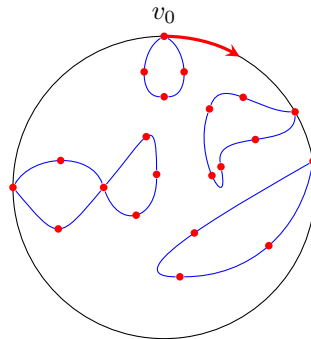


Рис. 4.7: Цикл Эйлера

- (b) *Индуктивное предположение:* $m \leq k$. Мы предполагаем, что мы уже умеем строить циклы Эйлера для четных НСГБП с $\leq k$ ребрами.
- (c) *Индуктивный переход:* $m = k + 1$. Мы уже доказали, что в таком графе будет хотя бы один элементарный цикл длины ≥ 3 . Назовем его C . И давайте возьмем какую-то вершину на нем, v_0 , пригодится. А теперь давайте удалим все ребра цикла из нашего графа. Граф при этом может перестать быть связным, т.е. распасться на кучу кусочков, компоненты связности, G_1, \dots, G_r . Давайте рассмотрим эти компоненты: они все еще остались НСГБП, и в них будет $\leq k$ ребер. Нужно проверить четность. Если какая-то вершина компоненты связности лежала на нашем цикле, то у нее убрались 2 ребра, т.е. ее степень стала на 2 меньше. Т.к. она была четная, то она и останется четной. У вершин, которые не лежали на нашем цикле, степени вершин не поменялись. Это значит, что все компоненты связности — четные НСГБП с $\leq k$ ребрами. Тогда по индуктивному предположению мы можем построить для них циклы Эйлера, C_1, \dots, C_r .

Докажем еще одну вещь, что хотя бы одна вершина каждой из компонент будет лежать на нашем цикле C , который мы удалили. Возьмем любую вершину рассматриваемой компоненты, v_1 . Раз наш граф был связным, то в нем был путь от v_1 к v_0 . Обозначим первую вершину цикла C , которую мы встречаем на этом пути, как v_2 . Таковая обязательно будет, потому что конечный пункт — вершина v_0 , которая сама лежит на цикле. Итак, это значит, что путь от v_1 до v_2 не содержит вершин цикла C (кроме v_2), а значит, в этом пути ничего не удалялось и он остался даже после удаления цикла C . Это значит, что v_1 и v_2 остались в одной компоненте связности и та вершина, которую мы искали — это v_2 .

Итак, у нас есть цикл C , и еще циклы C_1, \dots, C_r , которые явля-

ются циклами Эйлера для компонент связности (см. рисунок 4.7). Эти циклы проходят через все вершины компонент связности (потому что через каждую из вершин проходит хотя бы одно ребро, а цикл Эйлера обязан пройти и через него). Это значит, что они проходят и через ту вершину компоненты связности, которая лежит на C (мы ее обозначили как v_2). Мы можем просто «сдвинуть начала» циклов C_1, \dots, C_r так, чтобы они все начинались и заканчивались на цикле C . Так как наши циклы C_1, \dots, C_r проходят через все ребра графа, получившегося после удаления C , то вместе C, C_1, \dots, C_r проходят через абсолютно все ребра изначального графа G . Тогда мы можем идти по циклу C , начиная с v_0 , и когда встречаем один из маленьких циклов Эйлера, обходить его полностью и идти дальше. Так мы вернемся в v_0 , обойдя абсолютно все ребра графа. Так мы можем получить цикл Эйлера в G .

□

4.8 Циклы Гамильтона

Цикл Гамильтона — это как цикл Эйлера, но обходит все вершины графа, а не все ребра. Только для цикла Эйлера у нас был алгоритм, который их позволял искать (которым мы доказывали предыдущую теорему), а с циклами Гамильтона все не так радужно. Единственный известный способ поиска циклов Гамильтона для графа, про который мы больше ничего не знаем — перебор всех возможных перестановок вершин и проверки, что эта перестановка составляет путь без разрывов. Это очень долго и обладает сложностью $O(n!)$, что, мягко говоря, нехорошо. К счастью, по крайней мере для графов с очень хорошей связностью между вершинами мы всегда можем найти такие циклы:

Теорема. Если для любых 2 вершин НСГБП v_1 и v_2 выполняется $d(v_1) + d(v_2) \geq |V|$, то в графе есть цикл Гамильтона.

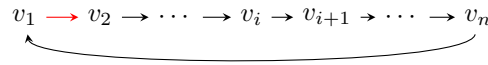
Доказательство. Пусть наше множество вершин $V = \{v_1, v_2, \dots, v_n\}$. Расположим их в некотором порядке, v_1, v_2, \dots, v_n . Будем пытаться сделать из этой последовательности цикл Гамильтона (с учетом того, что первая вершина тоже должна быть связана с последней). В этой перестановке могут быть *разрывы*, а именно такие соседние v_i и v_{i+1} , между которыми нет ребра в графе. Наша цель — такие разрывы устранить.

Если в этой перестановке нет разрывов, то нам повезло и мы сразу нашли наш цикл Гамильтона. Если нам не так повезло и разрывы там есть, то мы будем пытаться их убрать. Раз это цикл, то мы можем сдвинуть начало цикла так, чтобы разрыв был между v_1 и v_2 .

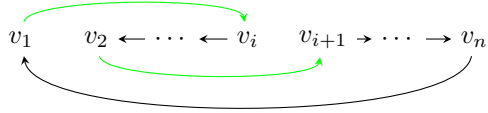
У этого графа есть особое свойство: мы всегда сможем найти такие соседние v_i, v_{i+1} среди v_3, \dots, v_n , чтобы существовали ребра (v_1, v_i) и

(v_2, v_{i+1}) . Предположим, что это не так, т.е. после каждой вершины v_i , которая связана с v_1 идет вершина v_{i+1} , которая *не может* быть связана с v_2 (Если бы была, то мы бы нашли эти v_i и v_{i+1} , а мы предположили, что их нет). Итак, вершин, которые связаны с v_1 всего $d(v_1)$. После каждой такой идет вершина, которая не связана с v_2 , но еще v_2 не связана сама с собой, так что всего вершин, не связанных с v_2 будет не меньше $d(v_1) + 1$. Вершин же, которые связаны с v_2 будет $d(v_2)$. Значит всего вершин должно быть не меньше $d(v_1) + d(v_2) + 1$, но по условию $|V| \leq d(v_1) + d(v_2)$. Такое одновременно невозможно, значит эти вершины v_i, v_{i+1} должны быть.

Тогда мы можем преобразовать наш цикл



в цикл



Т.е. в $v_1, v_i, \dots, v_2, v_{i+1}, \dots, v_n$. Это изменение убрало один разрыв между v_1 и v_2 (красный) и не добавило новых (зеленые линии — единственные новые, и мы их проверили, там ребро есть). Значит разрывов стало меньше и если мы повторим несколько раз эту процедуру, разрывы закончатся и мы получим цикл Гамильтона. \square

4.9 Суммы графов

Если у нас есть некоторый граф G и граф G' с тем же множеством вершин и $U' \subseteq U$. Если множество ребер $U' = \{u_1, \dots, u_m\}$, то мы можем записать U' как $\alpha(G') = \sigma_1 \sigma_2 \dots \sigma_m$, т.е. каждое ребро графа G мы кодируем 1, если это ребро есть и в G' , или 0, если его там нет. Тогда сумма 2 таких подграфов G_1 и G_2 — это другой подграф, ребра которого являются побитовой суммой по модулю 2 (XOR) ребер этих подграфов: $\alpha(G_3) = \alpha(G_1) + \alpha(G_2)$, но все разряды кодируются отдельно, например $00101011 + 10010001 = 10111010$. Часто за граф G берут полный граф, т.е. возможны все ребра. Пример см. на рисунке 4.8.

Теорема. Сумма четных графов — четный граф.

Доказательство. Пусть $G_3 = G_1 + G_2$, а v — любая вершина этого графа. Графы G_1 и G_2 — четные, значит $d(v)$ в обоих графах — четная. Пусть в G_1 $d(v) = 2p$, а в G_2 $d(v) = 2s$. Какие-то из ребер, входящих в вершину v , есть и в графе G_1 , и в графе G_2 . Пусть таких ребер r . Так как такие ребра будут отсутствовать в G_3 ($1 + 1 = 0$), то у вершины v в G_3 будут $2p - r$ ребер из G_1 и $2s - r$ ребер из G_2 , а вместе $2(p + s - r)$ — четное число. Так

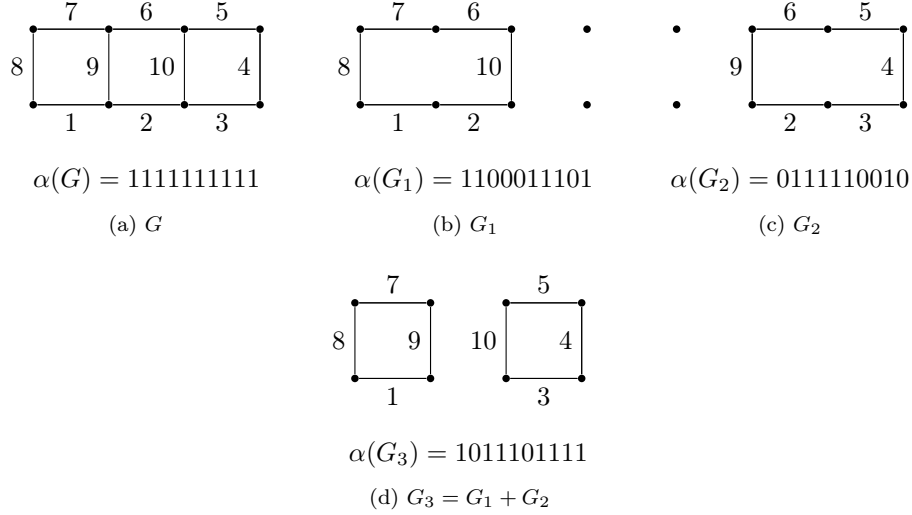


Рис. 4.8: Сумма графов

как мы выбирали v произвольно, то это верно для всех вершин, т.е. G_3 — четный. \square

4.10 Фундаментальное семейство циклов

Для того, чтобы это определить, нам нужно понятие *суммы циклов*. У нас есть сумма графов, но циклы — не графы, а просто последовательность вершин. Тогда для каждого цикла C в графе G определим подграф G_C , который представляет этот цикл (Строится он достаточно очевидным образом, $G_C = (V, E(C))$). Тогда сумма циклов $C = C_1 + C_2$ — это такой цикл, который представляет граф $G_C = G_{C_1} + G_{C_2}$, если этот граф вообще представляет цикл. Например, сумма двух циклов на рисунке 4.8 не представляет единственный цикл (их там 2).

А теперь определим так фундаментальное семейство циклов, ФСЦ. Если у нас есть НСГБП $G = (V, U)$, то множество «нормальных» циклов $\mathcal{F} = \{C_1, \dots, C_k\}$ (возможно пустое) является ФСЦ тогда и только тогда, когда оно удовлетворяет следующим условиям:

1. Ни один из циклов \mathcal{F} нельзя представить в виде суммы *других* циклов из \mathcal{F} . Это означает, что каждый цикл в ФСЦ индивидуален и мы не можем заменить его на сумму других. Единственный способ записать цикл $C_i \in \mathcal{F}$ при помощи циклов \mathcal{F} — это $C_i = C_i$ ⁶.
2. Абсолютно все «нормальные» циклы в графе (элементарные циклы

⁶ Да, сумма из одного слагаемого, вы что-то имеете против?

длины ≥ 3) можно записать как сумму циклов \mathcal{F} (те циклы, которые сами лежат в \mathcal{F} разрешено записывать как $C_i = C_i$, так что насчет них проблем нет).

Фундаментальное семейство циклов очень похоже на фундаментальное/базисное семейство векторов из линейной алгебры. Напоминаю определение: Набор векторов A называется базисом для какого-то большего набора векторов B , если A — линейно независимый (т.е. ни один из векторов A нельзя выразить через другие, эквивалентно условию 1) и в то же время все вектора B — линейные комбинации векторов A (эквивалентно условию 2). Таким образом, ФСЦ — это наименьшее семейство, которого достаточно, чтобы записать все «нормальные» циклы в графе. Нам нужно доказать, что ФСЦ есть во всех графах, а для этого нужно привести алгоритм построения ФСЦ и доказать, что-то, что у нас получилось — действительно ФСЦ.

Для начала построим ФСЦ для некоторого графа G . Если в этом графе нет циклических ребер, то в этом графе нет и «нормальных» циклов в принципе (он — дерево), и выражать нам нечего, так что $\mathcal{F} = \emptyset$. Если же циклические ребра есть, то мы будем их по очереди убирать, пока они не закончатся. При этом, когда удаляем ребро u_i , мы будем добавлять к \mathcal{F} цикл C_i , который проходил через это циклическое ребро u_i . Как только циклические ребра закончатся, мы получим граф G^k , если k — количество удаленных ребер (и, соответственно, циклов в \mathcal{F}). Это число называется *цикломатическим числом графа*, и, независимо от выбора циклов, для каждого графа оно всегда одно и то же и равно $n - m + 1$, где n — число вершин, а m — число ребер, что выводится достаточно легко из свойства, что у деревьев $|V| = |U| + 1$.

Так вот, мы построили вот так вот некое семейство \mathcal{F} , теперь нам надо доказать, что оно — ФСЦ. Для этого нужно доказать 2 условия:

1. Ни один из циклов \mathcal{F} нельзя представить как сумму других циклов из \mathcal{F} .

Доказательство. Предположим, что можно. Тогда существует некий цикл $C_j \in \mathcal{F}$, который равен $C_{i_1} + C_{i_2} + \dots + C_{i_s}$ (i_1, i_2, \dots, i_s — последовательность номеров используемых циклов, расположенная по возрастанию, хотя возможны пропуски, т.е. циклы, которые не используются в сумме). У нас возможны 2 случая:

- (a) $j < i_1$, т.е. цикл C_j был добавлен в \mathcal{F} раньше, чем C_{i_1} . Тогда это значит, что C_j проходит через ребро u_j , а все циклы $C_{i_1}, C_{i_2}, \dots, C_{i_s}$ через него не проходят, так как оно было удалено еще при добавлении C_j , которое было раньше, чем добавление C_{i_1} .
- (b) $j > i_1$, т.е. цикл C_j был добавлен в \mathcal{F} позже, чем C_{i_1} . Тогда это значит, что C_j не содержит u_{i_1} , а среди циклов $C_{i_1}, C_{i_2}, \dots, C_{i_s}$ это ребро имеется только у C_{i_1} , значит в сумме $C_{i_1} + C_{i_2} + \dots + C_{i_s}$

оно имеется. Значит ребра u_{i_1} нет в C_j , но оно есть в $C_{i_1} + C_{i_2} + \dots + C_{i_s}$, так что они не могут быть равны.

Значит наше предположение было неверно и условие 1 доказано. \square

2. Любой цикл в G можно выразить через циклы \mathcal{F} .

Доказательство. Ну давайте попробуем выразить некоторый цикл C через циклы из \mathcal{F} . Этот цикл обязан проходить хотя бы через одно из ребер $\{u_1, u_2, \dots, u_k\}$, потому что иначе он остался бы в дереве G^k . Тогда давайте найдем самое первое такое ребро, через которое проходит цикл C , пусть это u_{i_1} . Тогда $C + C_{i_1}$ — четный неориентированный граф без петель (он может быть несвязным, см. рисунок 4.8). Еще, эта сумма не будет содержать ребер с индексами $\leq i_1$. Если эта сумма содержит еще какие то циклические ребра из $\{u_1, u_2, \dots, u_k\}$, то выберем самое первое, u_{i_2} , и добавим C_{i_2} в сумму: $C + C_{i_1} + C_{i_2}$. Эта сумма — четный НГБП, и она не содержит ребер с индексами $\leq i_2$.

Продолжим так делать, пока не получим четный НГБП $C + C_{i_1} + C_{i_2} + \dots + C_{i_r}$, который больше не содержит циклических ребер из $\{u_1, u_2, \dots, u_k\}$. Раз там нет всех этих ребер, то он подграф G^k , значит там нет циклических ребер в принципе. Но в то же время он четный, значит каждая из его компонент связности содержит цикл Эйлера, т.е. если в этом графе вообще есть ребра, то в нем будет хотя бы один «нормальный» цикл, а значит и циклические ребра, каковых там нет. Значит наш эта сумма не содержит ребер: $C + C_{i_1} + C_{i_2} + \dots + C_{i_r} = \emptyset$. Тогда по правилам сложения по модулю 2 $C = C_{i_1} + C_{i_2} + \dots + C_{i_r}$, т.е. мы выразили C как сумму циклов из \mathcal{F} , что мы и хотели сделать. \square

Значит наше \mathcal{F} и правда ФСЦ.

4.11 Хроматические графы

Как всегда, работаем с НСГБП. Тогда p -раскраска⁷ графа G — это такое разбиение множества вершин V на множества V_1, V_2, \dots, V_p , что все связанные вершины лежат в разных множествах разбиения $((v_1, v_2) \in U \& v_1 \in V_i \& v_2 \in V_j \rightarrow i \neq j)$. Это называется «раскраской», потому что оно возникло из достаточно старой задачи раскраски карт. Мы «красим»⁸ вершины в p разных цветов так, чтобы соседние вершины были «покрашены» в разные цвета. Минимальное необходимое число цветов для графа G называется его хроматическим числом: $\chi(G)$.

Если в графе есть хотя бы одно ребро, то $\chi(G) \geq 2$. Если там есть вообще все ребра (граф полный), то все вершины должны быть покрашены в разные цвета. Тогда $\chi(G) = n$.

⁷ «Но мы на самом деле ничего не красим.» (К.И.Костенко)

⁸ Но на самом деле ничего не красим, разумеется

В отличие от остальных хроматических чисел, мы легко можем проверить, является ли граф 2-хроматичным ($\chi(G) = 2$).

Теорема. $\chi(G) = 2$ тогда и только тогда, когда в G есть хотя бы одно ребро (иначе $\chi(G) = 1$) и не содержится циклов нечетной длины (иначе $\chi(G) > 2$).

Теорема снова состоит из 2 частей, так что

1. Если $\chi(G) = 2$, то в G есть хотя бы одно ребро и не содержится циклов нечетной длины.

Доказательство. Если $\chi(G) = 2$, то есть хотя бы одно ребро, иначе было бы $\chi(G) = 1$, эта часть очевидна.

Предположим, что в G есть цикл нечетной длины, $C = v_{i_1} v_{i_2} \cdots v_{i_{2m}}$, где $i_1 = i_{2m}$, т.к. это цикл. Так как $\chi(G) = 2$, то существует некая 2-раскраска. Тогда эти 2 цвета/множества в C должны чередоваться, иначе там будут 2 подряд идущих (т.е. связанных) вершины одного цвета. Но v_{i_1} и $v_{i_{2m}}$ имеют индексы разной четности, поэтому они должны быть окрашены в разные цвета. Но в то же время это — одна и та же вершина, значит мы пришли к противоречию и в G нет циклов нечетной длины. \square

2. Если в G есть хотя бы одно ребро и не содержится циклов нечетной длины, то $\chi(G) = 2$.

Доказательство. Опять же, раз есть одно ребро, то очевидно, что $\chi(G) \geq 2$. Нам нужно доказать, что существует 2-раскраска и это сделает $\chi(G) = 2$.

Нам нужно разбить V на V_1 и V_2 (покрасить в 2 цвета). Возьмем какую-то вершину v_0 из G и поместим ее в V_1 . Всех ее соседей поместим в V_2 , всех соседей соседей в V_1 и т.д., пока вершины не закончатся. Причем эти множества не пересекаются, потому что от v_0 до каждой v_1 существует некий кратчайший путь и четность этого пути определяет цвет v_1 .

Докажем, что это — 2-раскраска. Предположим противное, т.е. что есть 2 вершины v_1 и v_2 , которые связаны и лежат в одном V_i . Это означает, что кратчайшие пути от v_0 до них — одной четности. Тогда, если W_1 — путь от v_0 до v_1 , а W_2 — от v_0 до v_2 ($\overline{W_2}$ — обратный путь, от v_2 до v_0), тогда можно составить цикл

$$C = \underbrace{v_0 \cdots v_1}_{W_1} \underbrace{v_2 \cdots v_0}_{\overline{W_2}}.$$

Так как W_1 и W_2 — одной четности, то длина этого цикла равна их сумме (четной) плюс 1 (ребро $v_1 v_2$), т.е. нечетная. А по условию циклов нечетной длины в графе нет, получили противоречие. \square

В общем случае найти хроматическое число графа очень сложно, но есть теорема, дающая верхнюю оценку этого числа:

Теорема. Если в графе $G = (V, U)$ степени всех вершин не больше некоторого p ($d(v) \leq p$), то $\chi(G) \leq p + 1$.

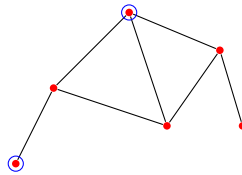
Доказательство. Доказываем индукцией по числу вершин.

1. *Базис индукции:* $n = 1$, граф — одна вершина без ребер. Тогда $d(v) = 0$, $p = 0$, $\chi(G) = 1 \leq p + 1$, т.е. условие выполняется.
2. *Индуктивное предположение:* $n = k$. Предполагаем, что для графов с k вершинами все работает и $\chi(G) \leq p + 1$.
3. *Индуктивный переход:* $n = k + 1$. У нас есть граф G с $k + 1$ вершинами, причем степени всех вершин не больше p . Возьмем произвольную вершину v_0 и удалим ее из графа. Теперь у нас граф G' с k вершинами. Максимальная степень вершины могла уменьшиться, но увеличиться не могла, так что степени всех вершин все еще не больше p . Тогда мы можем использовать индуктивное предположение и заключить, что мы можем раскрасить G' , используя не более $p + 1$ цветов. Сделаем это и вернем v_0 на место. Она связана с не более чем p вершин из G' , т.е. не более, чем с p цветами. Тогда мы всегда можем раскрасить ее в $p + 1$ -й цвет, и она не будет совпадать по цвету ни с одним из связанных цветов. Получили $p + 1$ -раскраску, $\chi(G) \leq p + 1$.

□

4.12 Внутренняя и внешняя устойчивость графов

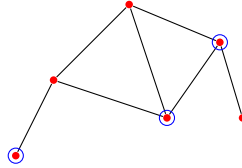
Подмножество вершин $V' \subseteq V$ НСГБП G называется *внутренне устойчивым* подмножеством, если внутри него нет связанных ребром вершин (каждая его вершина не связана ни с одной другой вершиной из V'). *Наибольший* возможный размер внутренне устойчивого подмножества вершин называется числом внутренней устойчивости графа G , обозначается $\alpha(G)$. Пример внутренне устойчивого подмножества.



Это подмножество не максимальное возможное, $\alpha(G) = 3$.

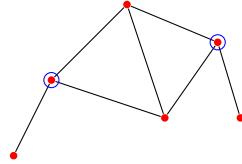
Подмножество вершин $V' \subseteq V$ НСГБП G называется *внешне устойчивым* подмножеством, если его вершины вместе «контролируют» все остальные вершины графа (каждая из вершин графа или сама лежит в V' , или

связана ребром с какой-то из вершин V'). *Наименьший* возможный размер внешне устойчивого подмножества вершин называется числом внешней устойчивости графа G , обозначается $\beta(G)$. Пример внешне устойчивого подмножества:



Это подмножество не минимальное возможное, $\beta(G) = 2$.

Если подмножество одновременно внутренне и внешне устойчивое, то оно называется *ядром*, обычно обозначается K . Пример ядра:



Теорема. В любом НСГВП есть ядро.

Доказательство. Пусть у нас есть НСГВП G , попробуем построить его ядро. Для начала пусть $K = \emptyset$. Возьмем первую попавшуюся вершину v_0 и положим в K . Далее возьмем первую попавшуюся вершину v_1 , которая не связана с v_0 . Так будем делать, пока остаются вершины, не связанные ни с одной из вершин в K . Так как вершин — конечное число, то когда-нибудь мы это случится.

Докажем, что K — ядро G , т.е. что оно внутренне и внешне устойчиво.

1. Оно внутренне устойчиво, потому что каждую из вершин v мы добавляли только если в K не было связанных с ней вершин, и после добавления v не добавляли тех, которые с ней связаны. Тогда v (и любая другая вершина) не может иметь связей внутри K .
2. Оно внешне устойчиво, потому что если бы остались какие-то вершины, которые не связаны ни с одной из вершин K , мы бы их добавили.

Мы нашли ядро, значит оно существует, доказательство окончено. \square

Глава 5

Зачетные задачи

Чтобы получить зачет по дискретке, необходимо решить 9 (хотя мы не успели все разобрать и решали только 7, возможно и у вас так будет) зачетных задач. Я буду описывать решение и оформление задач так, как их объяснял Константин Иванович, но в некоторых задачах буду давать еще и собственные наработки, которые, как мне кажется, облегчают ход решения. Что ж, приступим!

5.1 Задача 1

Необходимые темы: бинарные отношения на множествах (раздел 1.4 на с. 10), отношение эквивалентности (раздел 1.5 на с. 11).

Мы работаем со *словами* из латинских букв. *Слово* в дискретной математике — любая конечная последовательность символов, например *abcabca* — слово. Важный факт, в латинском алфавите ($a \cdots z$) 26 букв. Также существует одно пустое слово, в нем нет букв, и обозначается оно Λ .

Задача. $\alpha\rho\beta \iff$ слово, получаемое удалением из α всех букв, которым предшествует четное число равных им букв, совпадает со словом, получаемым удалением из β всех букв, которым предшествует четное число равных им букв.

Итак, в задаче нам дается некое отношение эквивалентности на множестве слов. Это отношение всегда можно записать коротко как $\alpha\rho\beta \iff F(\alpha) = F(\beta)$, где F — некоторая функция, которая принимает слово и выдает другое слово (чаще всего удаляя какие-то буквы), или, в некоторых задачах, число. Решение задачи состоит из 3 пунктов:

1. Доказать, что отношение — отношение эквивалентности.
2. Найти мощность множества классов эквивалентности (сколько всего классов), достаточно конечное или счетно-бесконечное, точное число не нужно.

3. Найти мощность отдельных классов (сколько элементов в каждом классе), так же конечное или счетно-бесконечное, но в некоторых задачах у разных классов разные мощности, это нужно учесть.

Для начала нужно понять, что именно делает наша F . Она удаляет те буквы, слева от которых есть четное число таких же. Например, если у нас есть слово $abbbbbaaabc$, то $F(\cancel{a}\cancel{b}\cancel{b}\cancel{b}\cancel{b}a\cancel{a}\cancel{b}\cancel{c}) = bbaa$.

Разберем, как это происходит.

1. Перед первой a у нас стоит 0 букв a , т.е. четное число, значит a удаляем.
2. Перед b стоит 0 b , тоже удаляем
3. Перед второй b стоит 1 b (мы считаем буквы даже если их уже удалили), это нечетное число, b оставляем
4. Перед третьей b стоит 2 b , удаляем
5. Перед четвертой b стоит 3 b , оставляем
6. Далее идет вторая a , перед ней была 1 a , нечетное, оставляем, и т.д.
1. Доказательство очень легкое и для всех задач абсолютно одинаковое. Для того, чтобы отношение было отношением эквивалентности, нужно чтобы оно было рефлексивным, симметричным и транзитивным:
 - (а) Рефлексивность, $\forall \alpha (\alpha \rho \alpha)$. В нашем случае это значит $F(\alpha) = F(\alpha)$, что, очевидно, верно.
 - (б) Симметричность, $\forall \alpha, \beta (\alpha \rho \beta \rightarrow \beta \rho \alpha)$. В нашем случае $F(\alpha) = F(\beta) \rightarrow F(\beta) = F(\alpha)$, что верно из-за свойства $=$.
 - (с) Транзитивность, $\forall \alpha, \beta, \gamma (\alpha \rho \beta \ \& \ \beta \rho \gamma \rightarrow \alpha \rho \gamma)$. В нашем случае это можно записать как

$$\begin{array}{ccc} F(\alpha) = F(\beta) & \& & \\ F(\beta) = F(\gamma) & \implies & F(\alpha) = F(\gamma), \end{array}$$

что также является просто свойством знака $=$.

2. Элементы, находящиеся в разных классах эквивалентности не связаны между собой, т.е. $F(\alpha) \neq F(\beta)$. Тогда классов будет столько же, сколько всего разных значений может выдавать наша функция F . Чтобы определить, конечно оно или бесконечно, нужно просто попытаться доказать оба варианта, и только один из них получится. Здесь это количество бесконечно, и чтобы доказать это, нам нужно дать бесконечную последовательность α , которые все дают разные $F(\alpha)$. Такой последовательностью будет

$$\begin{array}{rcl} F(aa) & = & a \\ F(aaaa) & = & aa \\ F(aaaaaa) & = & aaa \\ & \vdots & \end{array}$$

Это доказывает, что множество классов — счетно-бесконечно.

3. Элементы, находящиеся в одном классе эквивалентности связаны между собой, т.е. $F(\alpha) = F(\beta)$. Тогда элементами в классе Q_α будут все слова β , у которых $F(\beta) = \alpha$, и они будут связаны между собой, т.к. $F(\beta_1) = F(\beta_2) = \alpha$. Итого в классе эквивалентности будет столько же элементов, сколько всего различных входов β может выдавать одно и то же значение α . Можно воспользоваться тем фактом, что количество строк какой-то конечной длины n — конечно, и количество строк длины $\leq n$ — тоже конечно. Тогда чтобы доказать, что все классы — конечны, нам потребуется просто найти какое-то ограничение на длину β , т.е. найти такое n , что $|\beta| \leq n$ для всех β , для которых $F(\beta) = \alpha$. Более того, для каждого класса Q_α это ограничение может быть разным, так что на самом деле наше n может как-то зависеть от $|\alpha|$.

Итак, приступим. Мы можем заметить, что букв, перед которыми стоит четное число таких же букв, примерно столько же, сколько букв, перед которыми стоит нечетное число. Это означает, что удаляем мы всегда примерно половину букв или меньше. Если $|\beta|$ — сколько букв было, а $|\alpha|$ — сколько букв осталось после удаления, то $|\beta| - |\alpha|$ — сколько было удалено, и $|\beta| - |\alpha| \leq \frac{1}{2}|\beta|$. Если преобразовать, то $\frac{1}{2}|\beta| \leq |\alpha|$ или $|\beta| \leq 2|\alpha|$. Правда тут есть небольшая неточность, если например $\beta = aaaaa$, то $F(\beta) = \alpha = aa$, т.е. $|\beta| = 5, |\alpha| = 2$ и $5 \not\leq 2 \cdot 2$. Но если мы исправим формулу, $|\beta| \leq 2|\alpha| + 1$, то все будет верно. Итак, мы получили, что β , которая лежит в каком-то классе Q_α , не может быть больше, чем $2|\alpha| + 1$, значит класс Q_α (т.е. все классы) — конечны, задача решена.

Задача. $\alpha\rho\beta \iff$ число символов в α , соседи которых — разные, равно числу символов в β , соседи которых — разные.

Решение. Здесь наша функция F возвращает уже число, а не другое слово. Чтобы понять условие, проверим его на $abbbababcac$. $F(abbbababcac) = 4$. Самую первую и самую последнюю буквы учитывать не будем. У первой b соседи $a \neq b$, ее считаем, у второй b соседи $b = b$, не считаем, и т.д. до самого конца.

1. Доказательство ровно то же, что и в первой задаче.
2. Множество классов счетно-бесконечно, т.к.

$$\begin{aligned} F(abc) &= 1 \\ F(abca) &= 2 \\ F(abcab) &= 3 \\ &\vdots \end{aligned}$$

3. Все отдельные классы тоже счетно-бесконечны. Пусть у нас есть класс Q_n , и в нем есть некая α , у которой $F(\alpha) = n$. Если $n = 3$, то например $\alpha = abcab$. Добавляя к ней 2 последние буквы много раз, мы не будем увеличивать $F(\alpha)$, и мы получим бесконечное число слов в этом классе:

$$\begin{aligned} F(abcab) &= 3 \\ F(abcabab) &= 3 \\ F(abcababab) &= 3 \\ &\vdots \end{aligned}$$

Правда доказать нам это нужно в общем виде. Пусть $\alpha = \sigma_1 \cdots \sigma_{m-1} \sigma_m$. σ_i — одна буква. Тогда повторяя последние 2 буквы, мы будем получать одно и то же:

$$\begin{aligned} F(\sigma_1 \cdots \sigma_{m-1} \sigma_m) &= n \\ F(\sigma_1 \cdots \sigma_{m-1} \sigma_m \sigma_{m-1} \sigma_m) &= n \\ F(\sigma_1 \cdots \sigma_{m-1} \sigma_m \sigma_{m-1} \sigma_m \sigma_{m-1} \sigma_m) &= n \\ &\vdots \end{aligned}$$

Это же можно записать короче, используя обозначение $\sigma_{m-1} \sigma_m \sigma_{m-1} \sigma_m = (\sigma_{m-1} \sigma_m)^2$:

$$\begin{aligned} F(\alpha) &= n \\ F(\alpha \sigma_{m-1} \sigma_m) &= n \\ F(\alpha (\sigma_{m-1} \sigma_m)^2) &= n \\ F(\alpha (\sigma_{m-1} \sigma_m)^3) &= n \\ &\vdots \end{aligned}$$

Эта последовательность доказывает бесконечность всех классов, т.к. даже для $F(\alpha) = 0$ можно подобрать такое α , в котором будет хотя бы 2 буквы (например, $\alpha = aa$)

5.2 Задача 2

Необходимые темы: бинарные отношения на множествах (раздел 1.4 на с. 10).

На этот раз мы работаем с функциями на действительных числах, т.е. с обычными функциями из матанализа. Нам надо некое отношение для этих функций и нам нужно определить, какие из 4 свойств у них есть, а какие нет, и доказать это. Тут может пригодиться вот эта табличка:

	$\forall x$	$\exists x$
да	доказать для всех x	привести пример x
нет	привести пример x	доказать для всех x

Это значит, что если у нас в выражении есть квантор $\forall x(\dots)$, то если он верен, значит нам нужно доказать это для всех x , а если неверен — привести 1 пример x , для которого он неверен. Аналогично для $\exists x(\dots)$, если он верен — привести пример x , для которого верен, а если неверен — доказать, что неверен для всех. Еще одна вещь, если у нас встречается выражение вида $\exists k(P(k)) \& \exists k(Q(k))$, то хоть эти k и обозначаются одной буквой, они могут быть разные, потому что «объявлены» в разных кванторах, поэтому лучше подобное записывать как $\exists k_1(P(k_1)) \& \exists k_2(Q(k_2))$. Тем временем, $\forall k(P(k)) \& \forall k(Q(k))$ мы часто можем «синхронизировать», т.е. так как второе k в любом случае может принимать какие угодно значения, почему не такие же, как первое k ? Еще одна важная вещь — порядок кванторов. $\exists k \forall x$ означает, что для всех разных x существует одно универсальное k , для которого все работает, а $\forall x \exists k$ означает, что k для каждого x может быть свое.

Задача. $f\rho g \iff \exists k > 0 \forall x \in \mathbb{R} (f(x - 2k) + g(x + k) \leq 1)$.

Решение. Нам нужно разобрать 4 свойства:

1. Рефлексивность, $\forall f(f\rho f)$. Если подставить условие задачи, получим $\forall f \exists k > 0 \forall x \in \mathbb{R} (f(x - 2k) + f(x + k) \leq 1)$. Очевидно, это верно не всегда, например, когда f в принципе очень большая, сумма двух f может быть больше 1. Поиск примеров всегда лучше начинать с констант, так что давайте проверим $f(x) = 10$. Когда мы подставляем $x - 2k$ в $f(x)$, мы заменяем все вхождения x на $x - 2k$. Только в константах таких вхождений нет, поэтому $f(x - 2k) = 10$. С другой точки зрения, это происходит потому что график константы — горизонтальная линия, которая не меняется от горизонтального сдвига $f(x - 2k)$. Так вот, $f(x - 2k) + f(x + k) = 10 + 10 = 20 \not\leq 1$. Мы нашли 1 пример f , который не работает ни при каких k , значит *рефлексивность отсутствует*.
2. Симметричность, $\forall f, g(f\rho g \rightarrow g\rho f)$. Тут у нас 2 раза используется отношение ρ , получим

$$\forall f, g (\exists k_1 > 0 \forall x (f(x - 2k_1) + g(x + k_1) \leq 1) \rightarrow \exists k_2 > 0 \forall x (g(x - 2k_2) + f(x + k_2) \leq 1))$$

Заметьте, что k в двух разных «вызовах» ρ — разные, в то время как x можно брать как одно и то же, потому что у нас $\forall x$. На самом деле в таких задачах это очень часто, k в разных «вызовах» — разное, а x — один и тот же. Итак, у нас есть $f(x - 2k_1) + g(x + k_1) \leq 1 \rightarrow g(x - 2k_2) + f(x + k_2)$. Нет очевидных причин, почему это может быть так, поэтому стоит попробовать опровергнуть это. Да, в этих задачах очень многое зависит просто от интуиции, здесь нет общего алгоритма.

Импликация неверна только тогда, когда слева — 1, а справа — 0. Перебираем различные варианты функций:

- (а) Константы. Если f и g — константы, то и слева и справа будет $f + g$, т.е. если выражения будут верны, то одновременно, а мы ищем $1 \rightarrow 0$.
- (б) Простые линейные функции. Полезно пробовать $f(x) = x$, а $g(x) = x + c$, где c — некоторая константа. Получим слева

$$f(x - 2k_1) + g(x + k_1) = (x - 2k_1) + (x + k_1) + c = 2x - k_1 + c.$$

Не особо полезно, потому что в зависимости от x это будет или верно, или нет, а нам нужно, чтобы это было верно при всех x . Было бы хорошо убрать $2x$ отсюда. Как насчет $f(x) = x, g(x) = -x + c$? Это даст слева

$$f(x - 2k_1) + g(x + k_1) = (x - 2k_1) + -(x + k_1) + c = -3k_1 + c \leq 1.$$

А вот это уже лучше. Давайте посчитаем еще выражение справа:

$$g(x - 2k_2) + f(x + k_2) = -(x - 2k_2) + c + (x + k_2) = 3k_2 + c \leq 1.$$

Так как $k_1, k_2 > 0$, если мы выберем $c > 1$ (например $c = 2$), то $-3k_1 + 2 \leq 1$ будет верно, например, при $k_1 = 1$, а $3k_2 + 2 \leq 1$ не будет верно никогда — именно то, что нам нужно. Итак, значит контрпримером будет являться

$$\begin{aligned} f(x) &= x \\ g(x) &= -x + 2, \end{aligned}$$

т.к.

$$f \rho g \Leftrightarrow f(x - 2k_1) + g(x + k_1) \leq 1 \Leftrightarrow -3k_1 + 2 \leq 1, \text{ при } k_1 = 1$$

$$g \rho f \Leftrightarrow g(x - 2k_2) + f(x + k_2) \leq 1 \Leftrightarrow 3k_2 + 2 \leq 1 \text{ — невозможно}$$

(записывать, конечно, необходимо только пример и эти расчеты, а не все предыдущие попытки)

Симметричность отсутствует.

- 3. Транзитивность, $\forall f, g, h (f \rho g \& g \rho h \rightarrow f \rho h)$. Я не буду записывать все кванторы, и так понятно, как они будут располагаться, основной частью будет

$$\begin{cases} f(x - 2k_1) + g(x + k_1) \leq 1 \\ g(x - 2k_2) + h(x + k_2) \leq 1 \end{cases} \stackrel{?}{\implies} f(x - 2k_3) + h(x + k_3) \leq 1.$$

Даже на константах это может не работать, если f и h большие, а g — маленькое. Например $f(x) = 1, g(x) = -1, h(x) = 1$ вполне будет хорошим контрпримером. *Транзитивность отсутствует.*

4. Антисимметричность $\forall f, g(f\rho g \& g\rho f \rightarrow f = g)$. Тоже абсолютно неверно, $f(x) = -1$, $g(x) = -2$, $f + g = -3 \leq 1$, но $f \neq g$. *Антисимметричность отсутствует.*

Задача. $f\rho g \iff \exists k \geq 1 \forall x \in \mathbb{R} (f(x) \geq kg(x))$.

Решение. 1. Рефлексивность, $\forall f(f\rho f)$. $\forall f \exists k \geq 1 \forall x \in \mathbb{R} (f(x) \geq kf(x))$. Для $k = 1$ у нас получается $f(x) \geq f(x)$, что очевидно верно, и мы привели пример k , который будет работать для всех f , значит *рефлексивность есть*.

2. Симметричность, $\forall f, g(f\rho g \rightarrow g\rho f)$. Без кванторов это дает нам $f(x) \geq k_1g(x) \rightarrow g(x) \geq k_2f(x)$. И это доказать не получится, учитывая, что $k \geq 1$, т.е. мы не можем просто сказать, что $k_2 = \frac{1}{k_1}$. Нужно попробовать это опровергнуть, т.е. найти такие $f(x)$ и $g(x)$, у которых $f(x) \geq k_1g(x)$ для какого-то k_1 , но $g(x) < k_2f(x)$ для любых k_2 .

(а) Константы. $f \geq k_1g$ и $g < k_2f$. Давайте попробуем $f(x) = 2$, $g(x) = 1$. Тогда $f(x) \geq k_1g(x)$, например, при $k_1 = 1$ ($2 \geq 1 \cdot 1$), но $g(x) < k_2f(x)$ для всех k_2 , т.к. это дает нам $1 < 2k_2$, т.е. $k_2 > \frac{1}{2}$, что всегда верно, т.к. по условию у нас $k \geq 1$.

У нас все получилось, мы нашли контрпример, значит *симметричность отсутствует*.

3. Транзитивность, $\forall f, g, h(f\rho g \& g\rho h \rightarrow f\rho h)$. Без кванторов

$$\begin{cases} f(x) \geq k_1g(x) \\ g(x) \geq k_2h(x) \end{cases} \xRightarrow{?} f(x) \geq k_3h(x).$$

Если мы домножим второе уравнение на k_1 , то мы получим

$$\begin{cases} f(x) \geq k_1g(x) \\ k_1g(x) \geq k_1k_2h(x) \end{cases}$$

из чего следует $f(x) \geq k_1k_2h(x)$, т.е. при $k_3 = k_1k_2$ все работает и *транзитивность есть*.

4. Антисимметричность, $\forall f, g(f\rho g \& g\rho f \rightarrow f = g)$. Попробуем доказать аналогично с транзитивностью:

$$\begin{cases} f(x) \geq k_1g(x) \\ g(x) \geq k_2f(x) \end{cases} \xRightarrow{?} f(x) = g(x)$$

$$\begin{cases} f(x) \geq k_1g(x) \\ k_1g(x) \geq k_1k_2f(x) \end{cases}$$

Отсюда $k_1k_2 = 1$. Т.к. $k_1, k_2 \geq 1$, то это говорит о том, что $k_1 = k_2 = 1$, т.е. $f(x) \geq g(x)$ и $g(x) \geq f(x)$, из чего следует, что $f(x) = g(x)$. Утверждение доказано, *антисимметричность есть*.

Задача. $f\rho g \iff \exists k > 0 \forall x \in \mathbb{R} (f(x) + k < g(x + k))$.

Решение. А вот это интересная задача, в ней есть много всего необычного.

1. Рефлексивность, $\forall f(f\rho f)$. $f(x) + k < f(x + k)$. Странное заявление, которое не работает даже на константах: $f(x) = 1$, тогда $1 + k < 1$ — неверно, *рефлексивность отсутствует*.
2. Симметричность, $\forall f, g(f\rho g \rightarrow g\rho f)$. $f(x) + k_1 < g(x + k_1) \rightarrow g(x) + k_2 < f(x + k_2)$. Не менее странное заявление, стоит проверить на константах: $f + k_1 < g \rightarrow g + k_2 < f$. Нет, если например $f(x) = 1$, а $g(x) = 3$, то $f + k_1 < g$ например при $k_1 = 1$, но $g + k_2 < f$ — невозможно, *симметричность отсутствует*.
3. Транзитивность, $\forall f, g, h(f\rho g \& g\rho h \rightarrow f\rho h)$.

$$\begin{cases} f(x) + k_1 < g(x + k_1) \\ g(x) + k_2 < h(x + k_2) \end{cases} \xrightarrow{?} f(x) + k_3 \geq h(x + k_3).$$

Чтобы что-то с этим сделать, нужно привести компоненты с $g(x)$ к одному общему виду, например $g(x + k_1) + k_2$. Мы можем прибавить к верхнему неравенству k_2 с обеих сторон, да:

$$\begin{cases} f(x) + k_1 + k_2 < g(x + k_1) + k_2 \\ g(x) + k_2 < h(x + k_2) \end{cases}$$

Но кроме того, т.к. оба этих неравенства обязаны выполняться *для любых* x , мы можем заменить это x в нижнем неравенстве на что угодно, например на $x + k_1$, и оно должно остаться верным:

$$\begin{cases} f(x) + k_1 + k_2 < g(x + k_1) + k_2 \\ g(x + k_1) + k_2 < h(x + k_1 + k_2) \end{cases}$$

Отсюда следует $f(x) + k_1 + k_2 < h(x + k_1 + k_2)$, т.е. при $k_3 = k_1 + k_2$ все работает и *транзитивность есть*.

4. Антисимметричность, $\forall f, g(f\rho g \& g\rho f \rightarrow f = g)$. Это можно записать как

$$\begin{cases} f(x) + k_1 < g(x + k_1) \\ g(x) + k_2 < f(x + k_2) \end{cases} \xrightarrow{?} f(x) = g(x).$$

Если мы применим к этому то же доказательство из транзитивности, мы получим $f(x) + k_3 < f(x + k_3)$, что не говорит нам практически ничего. Нужно попробовать опровергнуть это.

- (а) Константы, $f + k_1 < g$ и $g + k_2 < f$. Если объединить, получим $f + k_1 < g < f - k_2$, что невозможно, т.е. условие $f\rho g \& g\rho f$ для констант не выполняется в принципе, ищем дальше.

(b) Простые линейные функции, $f(x) = x$, $g(x) = x + c$. Получаем

$$\begin{cases} x + k_1 < x + k_1 + c \\ x + c + k_2 < x + k_2, \end{cases}$$

$$\begin{cases} 0 < c \\ c < 0, \end{cases}$$

что не может выполняться одновременно для какого-то c . Ищем дальше, $f(x) = x$, $g(x) = -x + c$:

$$\begin{cases} x + k_1 < -x - k_1 + c \\ -x + c + k_2 < x + k_2 \end{cases}$$

$$\begin{cases} 2x + 2k_1 < c \\ c < 2x, \end{cases}$$

что тоже не может быть верно для любых x . И вообще, при разных коэффициентах при x у нас остается x в неравенствах, т.е. при разных x получается разный ответ, нам такое не подходит. Т.е. коэффициенты при x в $f(x)$ и $g(x)$ должны быть одинаковыми, чтобы выполнялось $f\rho g \& g\rho f$.

(c) Линейные функции в общем виде $f(x) = ax$, $g(x) = ax + c$. Получаем

$$\begin{cases} ax + k_1 < a(x + k_1) + c \\ ax + c + k_2 < a(x + k_2) \end{cases}$$

$$\begin{cases} ax + k_1 < ax + ak_1 + c \\ ax + c + k_2 < ax + ak_2 \end{cases}$$

$$\begin{cases} k_1 < ak_1 + c \\ c + k_2 < ak_2 \end{cases}$$

$$\begin{cases} (a - 1)k_1 > -c \\ (a - 1)k_2 > c \end{cases}$$

чего вполне возможно достичь, например, $a = 2, c = 1, k_1 = 1, k_2 = 2$, тогда

$$\begin{cases} (2 - 1) \cdot 1 > -1 \\ (2 - 1) \cdot 2 > 1 \end{cases}$$

Т.е. функции $f(x) = 2x$ и $g(x) = 2x + 1$ являются контрпримером для антисимметричности, т.е. *антисимметричность отсутствует*.

В принципе, если константы и линейные функции не получились, стоит попробовать $x^2 + c$, а еще функции с разрывами, типа $f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$, такие тоже иногда могут встречаться

5.3 Задача 3

Необходимые темы: правило умножения (раздел 2.1.2 на с. 16), сочетания и размещения (раздел 2.2 на с. 17)

Пожалуй, это самое простое задание из всех. Чтобы решить, нужно просто считать количество объектов, которые остаются доступными в каждой категории, и помнить, что сочетания - это «выбрать столько-то из столько-то», а размещения — это «выбрать столько-то из столько-то *и разместить*». И да, реально считать ничего не нужно, просто записать выражение из кучи C и A . Лучше на примере:

Задача. Посчитать количество троек слов (α, β, γ) , причем

1. В α 2 буквы встречаются по 3 раза и 5 букв по 2 раза.
2. В β есть 2 буквы из α , каждая из которых по 4 раза, и 4 буквы *не* из α по 2 раза.
3. В γ есть 2 из α , но не из β по 2 раза, 1 из β , но не из α , 7 раз, и 5 новых букв (не из α и не из β) по 1 разу.

Решение. Для начала, нужно посчитать, сколько всего букв в каждом слове: $2 \cdot 3 + 5 \cdot 2 = 16$ в α , $2 \cdot 4 + 4 \cdot 2 = 16$ в β и $2 \cdot 2 + 1 \cdot 7 + 5 \cdot 1 = 16$ в γ . В этой задаче во всех 16, но могло бы быть не так.. Чтобы посчитать ответ, нужно просто записывать выражения для α , β , γ по отдельности. И да, всего букв 26, это важно запомнить.

α : Сначала нужно выбрать те 2 буквы, которые встречаются по 3 раза, из 26: C_{26}^2 . Далее нужно выбрать позиции для каждой из этих 2 букв, для каждой из них по 3: $C_{16}^3 C_{13}^3$. После этого нужно выбрать 5 букв из оставшихся 24: C_{24}^5 , и выбрать позиции для всех, по 2 на каждую: $C_{10}^2 C_8^2 C_6^2 C_4^2 C_2^2$. Выбирать позиции мы должны в каком-то заранее определенном нами порядке¹. Этим порядком Константин Иванович всегда выбирает «от младшего к старшему», т.е. от a до z . Это не так важно для решения задачи, но нужно помнить, что мы не должны учитывать в подсчете выбор этого порядка, например $C_{24}^5 \cdot A_5^5 \cdot C_{10}^2 C_8^2 C_6^2 C_4^2 C_2^2$ было бы неправильным, как и выбор «неотличимых» букв по очереди: $C_{24}^1 C_{10}^2 \cdot C_{23}^1 C_8^2 \cdot C_{22}^1 C_6^2 \cdot C_{21}^1 C_4^2 \cdot C_{20}^1 C_2^2$. Пояснение см. после решения этой задачи. Итак, общая запись для α : $C_{26}^2 C_{16}^3 C_{13}^3 \cdot C_{24}^5 C_{10}^2 C_8^2 C_6^2 C_4^2 C_2^2$, в α было задействовано 7 *разных* букв, всего незадействованных букв осталось $26 - 7 = 19$.

β : 2 буквы из α . В α есть 7 разных букв, так что C_7^2 . Чтобы поставить на позиции, сделаем $C_{16}^4 C_{12}^4$. Итого осталось 5 букв, которые

¹Мы же не хотим считать количество разных порядков, в котором можно выбирать позиции для 5 букв? Нам важен только результат.

есть в α , но их нет в β . И мы еще наберем 4 буквы, C_{19}^4 , которые будут в β , но не в α , и разместим их: $C_8^2 C_6^2 C_4^2 C_2^2$. Итого, общая запись для β : $C_7^2 C_{16}^4 C_{12}^4 \cdot C_{19}^4 C_8^2 C_6^2 C_4^2 C_2^2$, осталось 5 букв в α , но не в β , 4 буквы в β , но не в α , и $19 - 4 = 15$ букв не в α и не в β .

γ : 2 из α , но не из β по 2 раза: $C_5^2 C_{16}^2 C_{14}^2$. 1 из β , но не из α , 7 раз: $C_4^1 C_{12}^7$. 5 новых букв по 1 разу: $C_{15}^5 C_5^1 C_4^1 C_3^1 C_2^1 C_1^1$, хотя это же можно записать как $C_{15}^5 A_5^5$, выбрали 5 букв и расставили их в каком-то порядке в 5 оставшихся пропусках, эти 2 записи абсолютно эквивалентны. Итого для γ : $C_5^2 C_{16}^2 C_{14}^2 \cdot C_4^1 C_{12}^7 \cdot C_{15}^5 A_5^5$

Полным ответом тогда будет

$$\begin{aligned} & C_{26}^2 C_{16}^3 C_{13}^3 \cdot C_{24}^5 C_{10}^2 C_8^2 C_6^2 C_4^2 C_2^2 \cdot \\ & C_7^2 C_{16}^4 C_{12}^4 \cdot C_{19}^4 C_8^2 C_6^2 C_4^2 C_2^2 \cdot \\ & C_5^2 C_{16}^2 C_{14}^2 \cdot C_4^1 C_{12}^7 \cdot C_{15}^5 A_5^5, \end{aligned}$$

хотя это можно так и не записывать отдельно от решения.

Что ж, а теперь объяснение «неотличимости». Предположим, мы считаем слова, состоящие из 3 разных букв по 1 разу. Тогда правильным решением будет $C_{26}^3 \cdot C_3^1 C_2^1 C_1^1$, выбираем все 3 буквы одновременно и размещаем их в заранее определенном порядке, от младшего к старшему. Можно попытаться решить это неправильно, а именно: выбрать первую букву, разместить ее, выбрать вторую, разместить, и т.д.: $C_{26}^1 C_3^1 \cdot C_{25}^1 C_2^1 \cdot C_{24}^1 C_1^1$. Это решение будет неправильным.

Предположим, у нас есть слово *cat*. Это слово должно быть посчитано ровно 1 раз, чтобы счет был правильным. Как его посчитает первое решение? Оно выберет 3 буквы одновременно: $\{a, c, t\}$, потом будет размещать их по порядку: *a* идет на 2-е место, *c* — на 1-е, *t* — на оставшееся 3-е. У нас есть ровно один способ получить слово *cat* при помощи этой записи, и это хорошо. Что же сделает второе решение? Оно может сделать так же, сначала выбрать *a*, поставить его на 2-е место, потом *c*, поставить на 1-е, потом выбрать *t*, поставить на 3-е и получить слово *cat*. Хорошо. Но есть еще способ, например, сначала выбрать букву *c*, поставить ее на 1-е место, потом выбрать *a*, поставить на 2-е место, выбрать *t*, поставить на 3-е место, и получить все то же слово *cat*. Этот способ счета различные порядки выбора букв (именно *выбора*, не *записи*) рассматривает как разные, поэтому наше слово *cat* будет посчитано 6 раз, в зависимости от того, в каком порядке выбирались наши 3 буквы. Поэтому единственный правильный способ выбора «неотличимых» объектов — это выбрать все одновременно и рассмотреть в заранее выбранном порядке, например, от младшего к старшему.

Но что такое «неотличимые» объекты? Это те объекты, которые по факту разные, но в условиях задачи они обладают одинаковыми свойствами. Например, в условиях этой задачи, буквы, которые взяты из одного и того же набора букв и записываются одинаковое количество раз — неотличимые.

Например, в условиях 5-й задачи, игроки считаются «неотличимыми», пока им не раздают разные по свойствам карты, несмотря на то, что у этих игроков, скорее всего, есть имена, и мы обязаны выбирать, например, 2 человека из 3, а не просто «взять 2 штуки», они все равно «неотличимы» и мы не должны выбирать «неотличимые» объекты отдельно друг от друга. Все, что было сказано здесь, не так важно для собственно решения задачи, оно скорее важно для понимания причин, почему мы решаем ее именно так, а еще очень полезно для решения дальнейших задач, а именно 5-й.

Все 3-и задачи в целом однотипны, но все равно стоит разобрать еще одну:

Задача. Имеются книги 65 видов по 50 наименований книг всякого вида. Три человека берут книги.

1. Первый — книги 6 видов по 3 *разных* книги, 4 видов по 6 *разных* книг и 3 видов — по 1 книге.
2. Вторым — 4 вида книг первого по 5 книг, 4 вида книг первого по 6 книг и 6 видов книг не первого по 7 книг.
3. Третьим — 3 вида книг первого и не второго по 4 *разных* книги, 4 вида книг второго и не первого по 5 *разных* книг и еще 2 вида книг не первого и не второго по 5 *разных* книг.

Решение. Важная особенность этой задачи — это что вся связь между первым, вторым и третьим заключается только в *видах* книг, не в самих наименованиях. Это значит, что два человека вполне могут брать одну и ту же книгу, т.е. количество книг в виде (50) не уменьшается, когда люди их берут. Другая важная особенность — в одних местах указано, что книги должны быть разные, а в других нет. Это значит, что там, где не указано, возможны повторения и следует использовать \bar{C} вместо C . Что ж, приступим к задаче:

1. Первый человек:

- (а) Книги 6 видов по 3 *разных* книги: $C_{65}^6 C_{50}^3 C_{50}^3 C_{50}^3 C_{50}^3 C_{50}^3$, можно записать коротко как $C_{65}^6 (C_{50}^3)^6$.
- (б) 4 видов по 6 *разных* книг: $C_{59}^4 (C_{50}^6)^4$.
- (в) 3 видов по 1 книге: $C_{55}^3 (C_{50}^1)^3$.

Итого, запись первого человека: $C_{65}^6 (C_{50}^3)^6 \cdot C_{59}^4 (C_{50}^6)^4 \cdot C_{55}^3 (C_{50}^1)^3$, у него имеется $6 + 4 + 3 = 13$ видов книг, неиспользованных осталось $65 - 13 = 52$ вида книг.

2. Вторым человеком:

- (а) 4 вида книг первого по 5 книг: $C_{13}^4 (\bar{C}_{50}^5)^4$.

(b) 4 вида книг первого по 6 книг: $C_9^4 \left(\overline{C}_{50}^6 \right)^4$.

(c) 6 видов книг не первого по 7 книг: $C_{52}^6 \left(\overline{C}_{50}^7 \right)^6$.

Итого, запись второго человека: $C_{13}^4 \left(\overline{C}_{50}^5 \right)^4 \cdot C_9^4 \left(\overline{C}_{50}^6 \right)^4 \cdot C_{52}^6 \left(\overline{C}_{50}^7 \right)^6$, у него имеется $4 + 4 = 8$ видов книг первого, 6 собственных видов книг, у первого осталось $13 - 8 = 5$ собственных видов, неиспользованных осталось $52 - 6 = 46$ видов.

3. Третий человек:

(a) 3 вида книг первого и не второго по 4 *разных* книги: $C_5^3 \left(C_{50}^4 \right)^3$.

(b) 4 вида книг второго и не первого по 5 *разных* книг: $C_6^4 \left(C_{50}^5 \right)^4$.

(c) 2 вида книг не первого и не второго по 5 *разных* книг: $C_{46}^2 \left(C_{50}^5 \right)^2$.

Итого, запись третьего человека: $C_5^3 \left(C_{50}^4 \right)^3 \cdot C_6^4 \left(C_{50}^5 \right)^4 \cdot C_{46}^2 \left(C_{50}^5 \right)^2$.

5.4 Задача 4

Необходимые темы: правило умножения (раздел 2.1.2 на с. 16), сочетания и размещения (раздел 2.2 на с. 17).

Еще одна задача на комбинаторику. В ней также используется один интересный алгоритм перебора разбиений числа, который Костенко, разумеется, не объясняет.

Задача. Имеется 5 видов открыток в количествах 50 разных открыток каждого вида. 2 человека выбирают открытки так, что первый берет от 15 до 17 открыток *всех* видов, а второй от 15 до 16 открыток, не более чем по 9 открыток каждого вида.

Сначала нужно разобраться с первым человеком. Он берет от 15 до 17 открыток, и мы можем выбрать любое число, пусть будет 16. Эти 16 открыток можно разбить на 5 слагаемых многими способами, причем мы сначала разбиваем именно на слагаемые, а не сразу на виды, каждое слагаемое — просто сколько открыток мы будем брать в каком-то (еще неизвестном виде). Например, разбиение $16 = 7 + 3 + 3 + 2 + 1$ — вполне правильное разбиение, но оно не означает, что именно в первом виде будет 7 открыток. Кроме того, этот человек берет открытки всех видов, т.е. каждого вида (а значит, и в каждой группе) должно быть не меньше 1 открытки. Костенко нужно не более 15 вариантов разбиения, но все они должны быть отсортированы «по убыванию», т.е. сначала 73321, потом 73222, потом 66211 и т.д. Алгоритм построения этих разбиений я сейчас опишу.

Для начала, в алгоритме часто необходима операция «сгущивания влево». Не знаю, как ее еще описать, мы просто разбиваем число так, чтобы

разбиение начиналось наибольшим возможным слагаемым, а заканчивалось наименьшим возможным, при этом числа должны монотонно убывать (как и всегда в разбиениях). Например, «скупивание» числа 16 в 5 видов будет выглядеть как $12, 1, 1, 1, 1$, а если максимально возможно только число 6, то это «скупивание» будет выглядеть как $6, 6, 2, 1, 1$. Думаю, понятно, как это работает.

Так вот, самое первое разбиение — это просто «скупивание» общего числа открыток по всем видам, в нашем случае, $12, 1, 1, 1, 1$. Чтобы определить следующее, нам нужно найти *первое справа число, которое больше самого правого хотя бы на 2*. Да, сложная фраза, но пожалуйста, осознай ее. Например, в разбиении $6, 6, 2, 1, 1$ самое правое число — 1, значит мы идем справа налево и ищем хотя бы 3, двойка не подходит, а вот правая 6 подходит (до левой 6 мы так и не доходим). В нашем случае, в разбиение $12, 1, 1, 1, 1$, это 12. Мы уменьшаем это число на 1, а дальше собираем вместе все, что справа от него, добавляем к этому 1 (которую вычли, мы просто ее перенесли вправо) и «скупиваем вправо». Т.е. у нас было $\underline{12}, 1, 1, 1, 1$, мы уменьшаем 12 до 11, справа остается $1 + 4 = 5$, мы это «скупиваем влево» и получаем $11, 2, 1, 1, 1$. Чаще всего эта операция просто увеличивает самое правое значение на 1, но иногда оно при этом будет становиться слишком большим и мы будем должны перенести остаток вправо.

Продолжим так же, самое правое число все еще 1, так что ищем 3, находим только 11: $\underline{11}, 2, 1, 1, 1$. Переносим 1 оттуда вправо, получаем $10, 3, 1, 1, 1$. Теперь 3 уже есть, $10, \underline{3}, 1, 1, 1$, и мы можем перенести 1 уже оттуда, $10, 2, 2, 1, 1$. А вот сейчас уже начинаются отличия от простого переноса 1 вправо: $\underline{10}, 2, 2, 1, 1 \rightarrow 9, 4, 1, 1, 1$. Мы собираем $2 + 2 + 1 + 1 = 6$, добавляем еще 1 от 10-ки, и получаем $4, 1, 1, 1$, а не $3, 2, 1, 1$. А вот следующим шагом у нас получается $9, \underline{4}, 1, 1, 1 \rightarrow 9, 3, 2, 1, 1$. Далее — еще один важный момент. Мы должны уменьшать 3: $9, \underline{3}, 2, 1, 1$, но если мы просто перенесем 1 вправо, мы получим $9, 2, 3, 1, 1$, чего быть не может. Поэтому эта 1 переходит дальше и получается $9, 2, 2, 2, 1$. Вот полная таблица, причем уменьшающиеся числа отмечены \downarrow , а «кучкующиеся влево» отмечены \leftarrow :

1	2	3	4	5
12	1	1	1	1
↓		←	←	←
11	2	1	1	1
↓		←	←	←
10	3	1	1	1
	↓		←	←
10	2	2	1	1
↓		←	←	←
9	4	1	1	1
	↓		←	←
9	3	2	1	1
	↓		←	←
9	2	2	2	1
↓		←	←	←
8	5	1	1	1
	↓		←	←
8	4	2	1	1
	↓		←	←
8	3	3	1	1
		↓		←
8	3	2	2	1
	↓		←	←
7	2	2	2	2
↓		←	←	←
7	6	1	1	1
	↓		←	←
7	5	2	1	1
	↓		←	←
7	4	3	1	1
		↓		

Итак, это сделали. Но каждая из строк этой таблицы на самом деле представляет из себя несколько вариантов, например 7, 2, 2, 2, 2 на самом деле 5 разбиений: (7, 2, 2, 2, 2), (2, 7, 2, 2, 2), (2, 2, 7, 2, 2), (2, 2, 2, 7, 2), (2, 2, 2, 2, 7). Для каждой строки нам необходимо посчитать количество вариантов, точнее просто написать C и A , как в 3-ей задаче. Например, для 8, 3, 2, 2, 1 нужно написать $C_5^1 C_4^1 C_3^2 C_1^1$: сначала выбираем один вид, в котором будет 8, потом один вид, в котором будет 3, потом 2 вида, в которых будут по 2, и последний оставшийся, в котором будет 1. Получаем что-то вроде

1	2	3	4	5	
12	1	1	1	1	$C_5^1 C_4^4$
11	2	1	1	1	$C_5^1 C_4^1 C_3^3$
10	3	1	1	1	$C_5^1 C_4^1 C_3^3$
10	2	2	1	1	$C_5^1 C_4^2 C_2^2$
9	4	1	1	1	$C_5^1 C_4^1 C_3^3$
9	3	2	1	1	$C_5^1 C_4^1 C_3^1 C_2^2$
9	2	2	2	1	$C_5^1 C_4^3 C_1^1$
8	5	1	1	1	$C_5^1 C_4^1 C_3^3$
8	4	2	1	1	$C_5^1 C_4^1 C_3^1 C_2^2$
8	3	3	1	1	$C_5^1 C_4^2 C_2^2$
8	3	2	2	1	$C_5^1 C_4^1 C_3^2 C_1^1$
7	2	2	2	2	$C_5^1 C_4^4$
7	6	1	1	1	$C_5^1 C_4^1 C_3^3$
7	5	2	1	1	$C_5^1 C_4^1 C_3^1 C_2^2$
7	4	3	1	1	$C_5^1 C_4^1 C_3^1 C_2^2$

Далее нужно разобрать 2 варианта, желательно не самых простых. Красная и синяя строки вполне подходят. Для красной строки нам нужно сначала выбрать вид, в котором будет 10: C_5^1 , а потом выбрать эти 10 из 50 (всего открыток в каждом виде), т.е. C_{50}^{10} . Таким образом, запись для красной строки будет $C_5^1 C_{50}^{10} \cdot C_4^2 C_{50}^2 C_{50}^2 \cdot C_2^2 C_{50}^1 C_{50}^1$, а для синей — $C_5^1 C_{50}^8 \cdot C_4^1 C_{50}^3 \cdot C_3^2 C_{50}^2 C_{50}^2 \cdot C_1^1 C_{50}^1$. Далее нам потребуется только один из этих вариантов, пусть это будет синий. Только дело в том, что в этой задаче открытки не бесконечные, и второму придется брать только те открытки, которые не взял первый, а т.к. первый брал из разных видов по разному, теперь в этих видах будет разное количество открыток, и мы не знаем в каких сколько, и это нехорошо. Поэтому нужно как-то обозначить те виды, которые мы выбирали. Пусть α — вид с 8 открытками, β — вид с 3 открытками, γ, δ — виды с 2, и ϵ — вид с одной открыткой. Тогда в видах осталось вот столько открыток:

α	β	γ	δ	ϵ
42	47	48	48	49

Итак, переходим ко второму. У него от 15 до 16 открыток, выберем 15. Еще каждого вида должно быть ≤ 9 , но при этом не сказано, что он берет открытки абсолютно всех видов. Тогда таблица будет такая:

1	2	3	4	5	
9	6	0	0	0	$C_5^1 C_4^1$
9	5	1	0	0	$C_5^1 C_4^1 C_3^1$
9	4	2	0	0	$C_5^1 C_4^1 C_3^1$
9	4	1	1	0	$C_5^1 C_4^1 C_3^2$
9	3	3	0	0	$C_5^1 C_4^2$
9	3	2	1	0	A_5^5
9	3	1	1	1	$C_5^1 C_4^1 C_3^3$
9	2	2	2	0	$C_5^1 C_4^3$
9	2	2	1	1	$C_5^1 C_4^2 C_2^2$
8	7	0	0	0	$C_5^1 C_4^1$
8	6	1	0	0	$C_5^1 C_4^1 C_3^1$
8	5	2	0	0	$C_5^1 C_4^1 C_3^1$
8	5	1	1	0	$C_5^1 C_4^1 C_3^2$
8	4	3	0	0	$C_5^1 C_4^1 C_3^1$
8	4	2	1	0	A_5^5

Нули считать не обязательно, а еще если все числа разные, как например 9, 3, 2, 1, 0, можно просто написать A_5^5 вместо длинного $C_5^1 C_4^1 C_3^1 C_2^1 C_1^1$. Итак, нам снова нужно разобрать 2 строки, например, красную и синюю. Только теперь, когда количества открыток в видах разные, нам потребуется напрямую рассматривать различные перестановки. Их очень много, поэтому достаточно рассмотреть штуки 3. Для красной строки это:

$$\begin{cases} 9 & 4 & 1 & 1 & 0 \\ 0 & 1 & 9 & 1 & 4 \\ 4 & 9 & 1 & 0 & 1 \\ & & & & \vdots \end{cases}$$

Каждый столбик соответствует одному виду из $\alpha, \beta, \gamma, \delta, \epsilon$. Нам нужно рассмотреть одну перестановку, например, среднюю. Для нее выбор будет просто $C_{47}^1 C_{48}^9 C_{48}^1 C_{49}^4$. Для синей строки перестановками будут

$$\begin{cases} 9 & 2 & 2 & 1 & 1 \\ 2 & 1 & 2 & 9 & 1 \\ 1 & 9 & 1 & 2 & 2 \\ & & & & \vdots \end{cases}$$

Выбор для средней строки: $C_{42}^2 C_{47}^1 C_{48}^2 C_{48}^9 C_{49}^1$. Это и будет полным решением задачи.

5.5 Задача 5

Необходимые темы: правило умножения (раздел 2.1.2 на с. 16), сочетания и размещения (раздел 2.2 на с. 17).

Эта задача — самая сложная из всех, и единственный способ решить ее — много практиковаться и уметь думать, при этом держать в голове кучу информации. Я попытаюсь максимально подробно объяснить, как это все должно работать, но большая часть зависит все таки от вас.

Итак, в этой задаче всегда есть 3 человека, которым раздают какие-то объекты (открытки/подарки/документы разных видов, но чаще всего карты разных мастей), и требуется «посчитать» количество способов такой раздачи, чтобы она удовлетворяла некоторым условиям, типа «ровно у 2 человек есть ровно 2 общих масти», т.е. это значит, что нет другой пары, у которой ровно 2 общих масти (пары с 1 или с 3 общими мастями допускаются), или «у всех 3 человек должно быть ровно 1 общее значение». Для этого строится «дерево решений», но не полное, а только одна ветвь. Построение этого дерева должно быть «направленным», т.е. там не должно быть ветвей, которые уходят «в никуда» и выборы ветки не должны быть вида «брать 7-ку червей или не брать», все должно быть в более общем виде.

Что ж, лучше на примере, причем на простом примере:

Задача. 3 человека берут каждый по 7 карт (всего карт 36, 4 масти по 9 величин в каждой), причем

1. Ровно у 2 человек 2 общих масти.
2. Ровно у 2 человек 2 общих величины.
3. Ровно у 2 человек 3 общих величины.

Решение. Сначала необходимо выбрать, какие именно пары людей будут участвовать в каких условиях. Для этого будем разбирать условия по порядку:

1. Ровно у 2 человек 2 общих масти.

Раз все 3 человека на данный момент неотличимы (мы не знаем о них ничего), мы можем просто выбрать 2 любых из них: C_3^2 и назвать их I и II. Также нам нужно узнать, какие 2 масти у них будут общие, мы тоже можем их просто выбрать: C_4^2 и назвать α, β .² Дерево на данный момент:

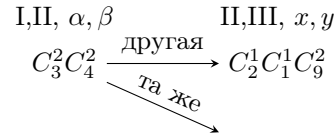
$$\begin{array}{c} \text{I, II, } \alpha, \beta \\ C_3^2 C_4^2 \end{array}$$

2. Ровно у 2 человек 2 общих величины.

Здесь у нас первая развилка, потому что люди перестали быть неотличимыми: У нас есть I и II, которые уже участвуют в 1 условии, а есть

²Люди называются I, II, III, масти/виды называются греческими буквами, а величины — латинскими.

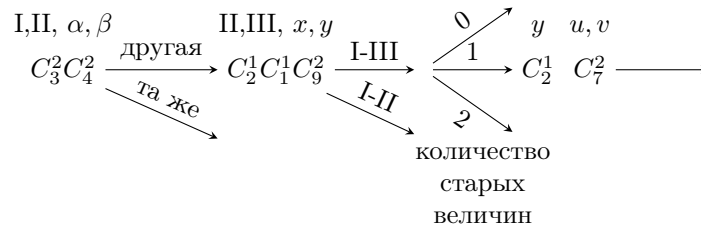
III, который не участвует пока нигде. Эта пара может быть все той же парой I-II, а может быть другой. Вообще, все составление дерева — это задавание вопросов для уточнения информации, а потом выбора самого «интересного» ответа. Здесь мы выберем, что это другая пара. В ней обязательно должен будет присутствовать III, но неизвестно, кто из I-II тоже будет. Так как I и II пока что неотличимы между собой, мы просто выберем C_2^1 и назовем его II. Да, мы можем так делать с неотличимыми объектами, просто выбрать один и назвать его как-то, потому что до этого у нас были просто 2 человека, I и II, но мы не знали, кто из них кто, мы знали только то, что они в паре по 1 условию. Теперь мы знаем, что II еще и в паре по 2 условию. И да, выберем III из оставшихся, C_1^1 , и выберем им 2 общих величины, C_9^2 , и назовем их x, y . Тогда дерево будет выглядеть как



3. Ровно у 2 человек 3 общие величины.

Эта пара не может быть парой II-III, потому что у них 2 общие величины, а не 3. Значит это или I-II, или I-III, причем от этого выбора многое зависит, так что снова ветвление. Пусть это будет I-III.

Следующий вопрос, как эти 3 величины будут соотноситься с x, y ? Это 3 абсолютно новых величины, u, v, t , или это 2 новых и 1 старая, например y, u, v , или это только 1 новая и 2 старых, x, y, u ? От этого выбора тоже будет многое зависеть, поэтому нам нужно сделать еще одно ветвление, по количеству старых величин среди этих 3. Оно может быть или 0, или 1, или 2, и мы выберем 1. Какая из неотличимых величин x, y будет среди 3 новых? Давайте ее выберем, C_2^1 и назовем y . А еще выберем 2 новых величины, u, v , C_7^2 . Тогда полное дерево уточнения условий будет



Итак, условия есть. Что мы имеем? У I и II есть 2 общие масти: α и β , больше про масти ничего не известно. А вот про величины известно многое, поэтому это лучше записать в табличку (см. таблицу 5.1)

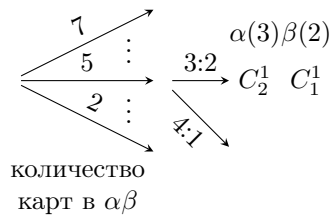
	x	y	u	v
I	×	✓	✓	✓
II	✓	✓	×	×
III	✓	✓	✓	✓

Таблица 5.1: Таблица величин

Далее нужно собственно раздать им карты. Есть 2 варианта, мы можем придумать раздачу заранее (написать на черновике), а можем придумывать ее постепенно. Я всегда делал постепенно, так правильнее с точки зрения Константина Ивановича, но мое дело — рассказать, выбор остается за вами. Раздача отмечается в подобной табличке:

	x	y	u	v					
α									
β									

Итак, первый человек: мы знаем, что у него есть карты мастей α, β , но не знаем сколько. Это нужно узнать, ветвление! Их может быть минимум 2 (одна из α , другая из β) и максимум 7 (все карты I). Выберем примерно посередине, 5. Вопрос, как это 5 разбивается на 2 масти? 3 : 2 или 4 : 1? Ветвление! Пусть будет 3 : 2. Вопрос, в какой из α, β есть 3, а в какой 2? Нет, не ветвление, это решение нам не поменяет, потому что α и β неотличимы на сей момент. Если выбираем один из неотличимых, это делается при помощи C . Итак, выбираем тот, в котором 3, и называем его $\alpha: C_2^1$, а потом выбираем оставшееся $\beta: C_1^1$.³ Итак, дерево пока что:⁴

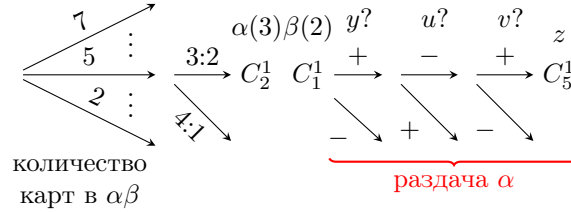


Итак, нам нужно набрать 3 карты из α . У нас есть 4 уже известных и важных для нас величины, нужно узнать про них в первую очередь. x мы не берем в любом случае, нам таблица 5.1. Берем ли мы y ? Пусть да. Берем ли мы u ? Пусть будет нет, мы не обязаны брать ее в α , мы можем взять

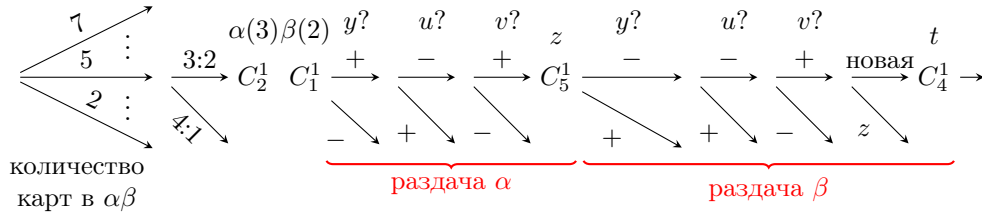
³На самом деле C_1^1 не значит ровным счетом ничего, мы его приписываем просто для понятности, мол «выбираем последний оставшийся», хотя выбора особо и нет.

⁴Я не рисую все дерево полностью, как нужно, иначе оно будет просто огромным, я буду рисовать его частями, так что это — только для I игрока.

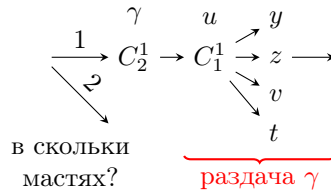
ее где-то еще. Берем ли мы v ? Пусть да. Тогда нам останется взять еще одну карту в α , и раз известные величины закончились, мы берем карту новой величины. Для этого нужно выбрать величину: C_5^1 , и назвать ее как-нибудь, пусть z . Для всех этих выборов нет четких правил, просто делайте то, что вам кажется нужным, и откатывайте назад, если обнаруживаете, что решение невозможно. Вот дерево:



Далее нужно набрать 2 карты в β . Берем y ? Нет. Берем u ? Нет. Берем v ? Да. Берем z или что-то новое? Что-то новое, C_4^1 , назовем его t . (Все карты отмечены в таблице 5.2, см. дальше)



Сейчас вопрос, у нас осталось 2 карты добрать для I, они одной масти или нет? Ветвление. Если бы они были разных мастей, что I бы занимал все 4 масти, и нам было бы довольно сложно дальше, так что пусть обе карты будут в одной масти. В какой? Выберем из оставшихся 2: C_2^1 и назовем γ . Там у нас 2 карты, причем по таблице 5.1 I обязан взять y, u, v . y и v он уже брал, а u еще нет, поэтому обязан ее сейчас взять. А далее у нас осталось много вариантов, или y , или v , или z , или t , или новую. Возьмем z . Тогда, остаток дерева I:⁵



Итак, раздача I завершена, результат см. в таблице 5.2.

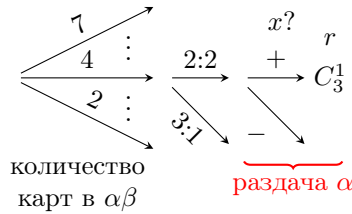
⁵Я не стал рисовать начальную часть, это просто продолжение

	x	y	u	v	z	t			
α		I		I	I				
β				I		I			
γ			I		I				

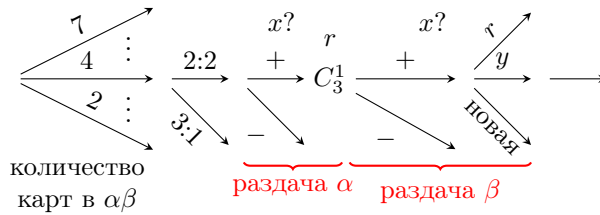
Таблица 5.2: После раздачи I игрока

Приступим к раздаче II, аналогично раздаче I. Количество карт в $\alpha\beta$ тоже от 2 до 7, выберем 4. Это может быть 2 : 2 или 3 : 1, выберем 2 : 2. Итого в α и β по 2. Насчет величин, нам известно, что x, y у II обязаны быть, а u, v там быть не может (из таблицы 5.1). Но что насчет z, t ? У нас уже одна имеется одна общая величина с I, это y , если мы возьмем z, t , то их будет 2 или 3. Но по условию 2 общих величины только у II-III, а 3 только у I-III, значит мы не можем брать и z, t .

Собственно раздача в α : берем x ? Да. y занят, так что обязаны брать новую, C_3^1 , назовем ее r :



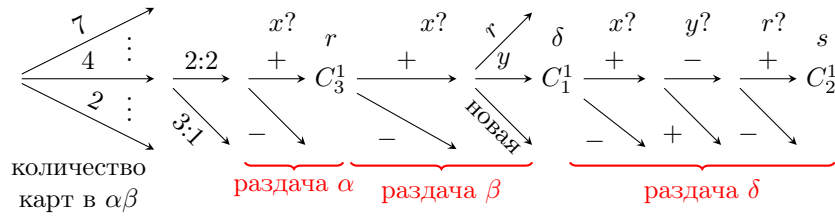
Раздача в β : берем x ? Да. Берем y , или r , или новую? y :



Далее, γ мы брать не можем, потому что иначе она была бы 3-ей общей мастью с I. Вместо этого надо будет добрать 3 карты в δ . x берем? Да. y берем? Нет. r берем? Да. Нам остается одна новая, C_2^1 , назовем ее s . На этом раздача для II закончена:

	x	y	u	v	z	t	r	s
α	II	I		I	I		II	
β	II	II		I		I		
γ			I		I			
δ	II						II	II

Таблица 5.3: После раздачи II игрока



Раздачу см. в таблице 5.3.

Последний, III игрок. Правда насчет него мы не знаем даже в скольких мастях он располагается, в 1, в 2, в 3 или во всех 4. Тут нужно рассмотреть все варианты:

- В 1 масти. Если посмотреть на таблицу 5.3, мы увидим, что не осталось ни одной масти, в которых свободны все 4 величины x, y, u, v , которые III все обязан взять, так что нет.
- В 2 мастях. Тут таких очевидных препятствий нет, нужно рассмотреть все 6 вариантов.
 - $\alpha\beta$ — невозможно, 2 общих масти с I и II
 - $\alpha\gamma$ — невозможно, 2 общих масти с I
 - $\alpha\delta$ — невозможно, 2 общих масти со II
 - $\beta\gamma$ — невозможно, 2 общих масти с I
 - $\beta\delta$ — невозможно, 2 общих масти со II
 - $\gamma\delta$ — возможно

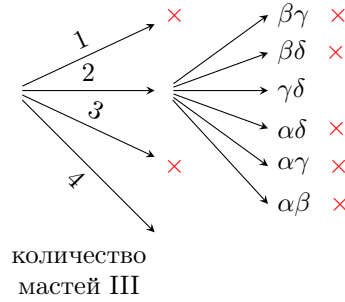
Итого, если рассматривать только 2 масти, осталось только $\gamma\delta$, запомним.

- В 3 мастях. Как бы мы ни взяли, тоже будут 2 общие масти или с I, или со II, так что невозможно.
- Во всех 4 мастях. Возможно, по 3 общих масти с каждым.

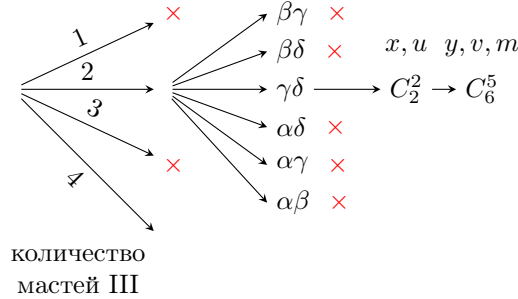
Итого имеем только 2 варианта, $\gamma\delta$ и $\alpha\beta\gamma\delta$. Попробуем сделать с вариантом $\gamma\delta$. Дерево выбора на данный момент:

	x	y	u	v	z	t	r	s	m
α	II	I		I	I		II		
β	II	II		I		I			
γ	III	III	I	III	I				III
δ	II		III	III			II	II	III

Таблица 5.4: После раздачи III игрока



Итак, раздача. В $\gamma\delta$ обязаны быть все 7 карт. Количество общих величин и с I, и со II у нас жестко заданы, и эти самые общие величины тоже. Это значит, что мы не можем брать величины, которые уже есть у I или II, не считая x, y, u, v . Т.е. z, t, r, s нам запрещены. Это оставляет нас только с 8 доступными картами (см. таблицу 5.3), 6 в x, y, u, v и еще 2 неизвестной новой величины. Нам нужно взять 7 из них, так что мы можем себе позволить не брать только одну, причем это должна быть не x и не u , потому что они доступны только в единственном экземпляре и мы обязаны их взять. Эту часть можно решить многими разными способами, допустимо просто сказать, что мы обязательно берем x, u (C_2^2), а остальные 5 выбираем как хотим из 6 доступных: C_6^5 . Можно же разобрать как раньше, x берем, потому что обязательно, y берем или не берем и т.д. Хотя сам Костенко чаще всего пишет именно при помощи C , если замечает такое, так что:



Полная таблица раздачи (одного из вариантов) — таблица 5.4. Решение на этом окончено.

Немного общих рассуждений, прежде чем решить более сложную задачу. Все решение 5-й задачи — это одна ветвь огромного дерева решений, которую мы должны пройти, постепенно задавая вопросы и выбирая «интересный» ответ на них. Не обязательно самый интересный, потому что такие варианты далеко не всегда приводят к решению. В общем, выбирайте варианты примерно так же, как я, или как вам расскажут на практике.

Далее, C . Они — просто способ объединить несколько ветвей дерева в одну, более общую, когда это возможно. Например, когда мы в раздаче I игрока выбирали количество карт в α и β (с. 88), мы не сделали дополнительное разветвление, в котором из них будет 3 карты, а в котором будет 2 карты. Мы эти 2 ветки объединили в одну при помощи C_2^1 , которой мы выбрали ту масть, в которой будет 3 карты, и назвали ее α . Эти 2 решения были бы абсолютно идентичны с точки зрения правильности, но Константин Иванович предпочитает обобщать ветки по максимуму, т.е. буквально со всеми неотличимыми объектами он будет работать при помощи C , а не при помощи ветвлений.

Так как раздача карт во всех 5-х задачах очень похожа, дальше будет описано только уточнение условий задачи, самая сложная их часть.

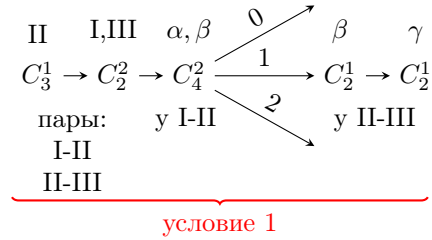
Задача. Имеется 4 вида документов по 10 документов с номерами 1–10 в каждом. 3 человека набирают по 7 документов, так чтобы

1. Ровно у 2 пар людей было по 2 общих вида
2. Ровно у 2 человек было 4 общих номера документов

Решение. Эта задача чуть сложнее, и в ней есть условие на пары людей. К счастью, это единственная сложность в этой задаче. Так как эти 2 пары людей из 1-го условия — неотличимые, мы не можем выбрать их по очереди, т.к. тогда одна из пар будет «первее» другой, а это недопустимо для неотличимых объектов. Нам нужно каким то образом выбрать эти 2 пары одновременно. Для этого выберем того единственного человека, который будет присутствовать в обеих этих парах, пусть он будет II. Тогда парами будут I-II и II-III. Мы можем это сделать просто при помощи C_3^1 , а потом выбрать I и III при помощи C_2^2 . Пусть тогда в паре I-II общими будут виды α, β , которые мы выберем C_4^2 (такое разрешено, потому что хоть мы и не назначили еще, кто будет I, а кто III, как только назначим, все распределение видов и определится).

Далее вопрос, как 2 общих вида пары II-III соотносятся с α, β ?⁶ Это те же самые 2 вида, или это 2 других вида, или это 1 новый вид и 1 старый? Самым интересным определенно будет вариант 1 новый + 1 старый, его и рассмотрим. Старым будет $(C_2^1) \beta$, новым будет $(C_2^1) \gamma$. Вот и получили дерево:

⁶Мы всегда должны задавать такие вопросы про виды/номера, про которые мы уже что-то знаем.



Т.к. пары I-II и II-III у нас неотличимы, для 2-го условия у нас 2 варианта — или пара I-III (причем I и III остаются неотличимыми, что я не рекомендую), или одна из пар I-II, II-III. Это мы решаем ветвлением, выберем одну из I-II, II-III, а потом решим, какую именно, т.е. сделаем C_2^1 и назовем ее I-II, потому что можем за счет неотличимости. Выберем 4 общих номера ей, C_{10}^4 , x, y, z, t , и закончим на этом:

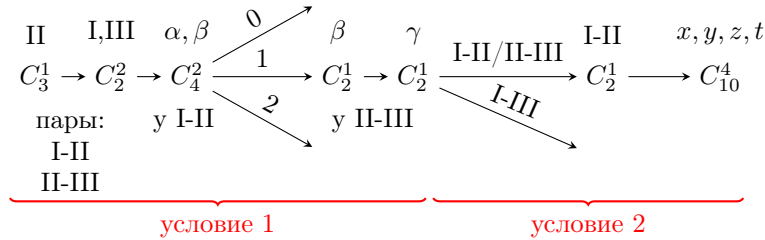


Таблица видов:

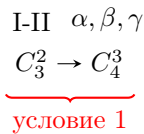
	α	β	γ	δ
I	✓	✓	✗	?
II	✓	✓	✓	?
III	✗	✓	✓	?

Задача. Имеется 4 сорта цветов по 10 наименований с номерами 1–10 в каждом сорте. 3 человека выбирают цветы в количествах 7, 8, 8, так чтобы

1. Ровно 2 человек были 3 общих сорта цветов
2. Ровно 2 номера цветов были общими у всех 3 человек
3. Ровно 2 человек было 3 общих номера цветов

Решение. Здесь 2 сложности: первая — что 3 человека берут цветы в разных количествах, а вторая — условие 2, где 2 номера цветов общие у всех 3. Так как количества цветов у каждого не зависят напрямую от условий и будут только мешаться, мы выберем их потом, а начнем с первого условия.

Первое условие стандартное, его дерево выглядит так:



Второе условие, хоть и необычное, очень простое. Т.к. оно для всех 3 человек, людей нам выбирать вообще не нужно, только номера цветов:

$$\begin{array}{ccc} \text{I-II} & \alpha, \beta, \gamma & x, y \\ C_3^2 \rightarrow C_4^3 & \longrightarrow & C_{10}^2 \\ \underbrace{\hspace{1.5cm}} & & \underbrace{\hspace{1.5cm}} \\ \text{условие 1} & & \text{условие 2} \end{array}$$

Третье условие может относиться или к уже известной паре I-II, или к одной из пар I-III/II-III. Пусть она относится к одной из пар I-III/II-III, а именно к II-III, что мы выберем при помощи C_2^1 . Насчет номеров, у II-III уже есть 2 общих номера из условия 2, так что они обязаны быть общими и по условию 3. Значит эти 3 номера — x, y и некий новый номер z :

$$\begin{array}{ccccccc}
 \text{I-II} & \alpha, \beta, \gamma & & x, y & & \text{II-III} & x, y \quad z \\
 C_3^2 \rightarrow C_4^3 & \longrightarrow & C_{10}^2 & \xrightarrow{\text{I-III/II-III}} & C_2^1 \rightarrow C_2^2 & C_8^1 \\
 & & & \searrow \text{I-II} & & & \\
 \underbrace{\hspace{1.5cm}}_{\text{условие 1}} & \underbrace{\hspace{1.5cm}}_{\text{условие 2}} & \underbrace{\hspace{1.5cm}}_{\text{условие 3}}
 \end{array}$$

Таблица номеров:

	x	y	z
I	✓	✓	✗
II	✓	✓	✓
III	✓	✓	✓

Но не забываем про то, что кто-то один из них должен брать 7, а другие 8, и от этого поменяется решение. Так что:

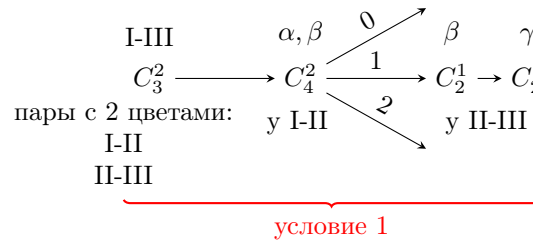
$$\begin{array}{ccccccc}
 \text{I-II} & \alpha, \beta, \gamma & x, y & & \text{II-III} & x, y & z \\
 C_3^2 \rightarrow C_4^3 & \longrightarrow & C_{10}^2 & \xrightarrow{\text{I-III/II-III}} & C_2^1 \rightarrow C_2^2 & C_8^1 & \begin{array}{l} \nearrow 7, 8, 8 \\ \rightarrow 8, 7, 8 \\ \searrow 8, 8, 7 \end{array} \\
 & & & \searrow \text{I-II} & & & \\
 \underbrace{\hspace{10em}}_{\text{условие 1}} & \underbrace{\hspace{10em}}_{\text{условие 2}} & \underbrace{\hspace{10em}}_{\text{условие 3}}
 \end{array}$$

Задача. Имеются шары 4 цветов по 10 шаров с номерами 1–10 в каждом цвете. 3 человека набирают по 7 шаров, так чтобы

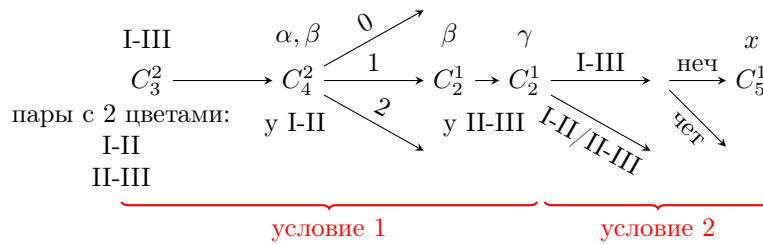
1. Ровно у 2 человек *не* было 2 общих цвета шаров
2. Ровно у 2 человек был 1 общий номер шаров
3. Ровно у 2 человек было 3 общих *четных* номера шаров

Решение. А вот это — очень сложное. Не только тут есть условие 1, где есть *не* (что на самом деле полностью эквивалентно условию 1 задачи с документами), в условии 3 упоминается четность номеров, что означает, что теперь для всех номеров нам нужно дополнительно следить, четные они или нет.

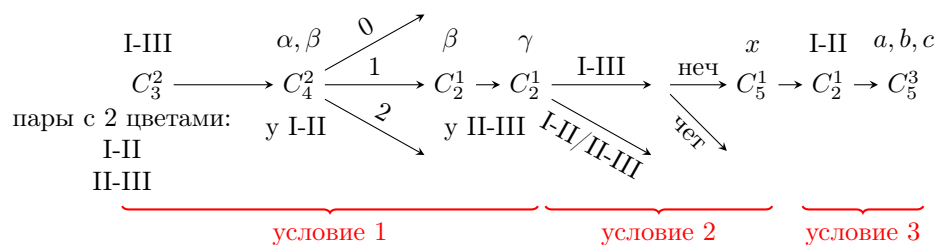
Итак, условие 1: нужно выбрать, у какой пары будет *не* 2 общих цвета. Все люди неотличимы, так что C_3^2 , и пусть это будут I и III. Но в то же время, это значит, что у I-II и II-III будет по 2 общих цвета. Выберем эти цвета точно так же, как в предыдущей задаче:



Второе условие: т.к. в условии 3 нам важна четность шаров, нам нужно будет ее проверять и здесь. Так вот, эта пара — снова или I-III, или одна из I-II, II-III. Пусть будет I-III для разнообразия. Этот 1 общий номер четный или нечетный? Пусть нечетный. Давайте все нечетные номера обозначать x, y, z, r, s , а четные — a, b, c, d, e . Этот нечетный, значит C_5^1 и это x :



Третье условие: мы уже не можем выбрать пару I-III, потому что там всего 1 общий номер, а условие 3 говорит, что должно быть 3, да еще и четных, так что это или I-II, или II-III. Т.к. они неотличимы, выбираем одну из них C_2^1 . И выберем ей 3 общих четных номера, a, b, c , при помощи C_5^3 :



Таблицы:

	α	β	γ	δ		x	a	b	c
I	✓	✓	✗	?	I	✓	✓	✓	✓
II	✓	✓	✓	?	II	?	✓	✓	✓
III	✗	✓	✓	?	III	✓	✗	✗	✗

После этого идет обычная раздача, но каждый раз, когда мы берем новую величину, требуется спрашивать, будет ли она четной или нечетной, и в принципе учитывать четность/нечетность всю задачу.

Задача. Имеется 4 вида открыток по 9 открыток с номерами 1–9 в каждом виде. 3 человека выбирают 8, 7 и 7 открыток, так чтобы

1. Ровно у 2 человек были 2 общие вида открыток
2. Ровно у 2 человек номера открыток одного были между номерами открыток другого
3. Ровно у 2 человек было 3 общих номера открыток

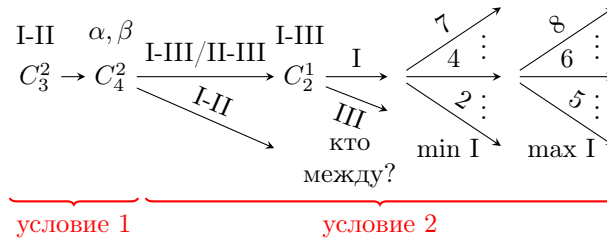
Решение. Самая сложная часть, разумеется — условие 2. Но давайте сначала разберем условие 1. Как обычно, выбираем двух человек и 2 общих вида для них:

$$\begin{array}{c} \text{I-II} \quad \alpha, \beta \\ C_3^2 \rightarrow C_4^2 \\ \hline \text{условие 1} \end{array}$$

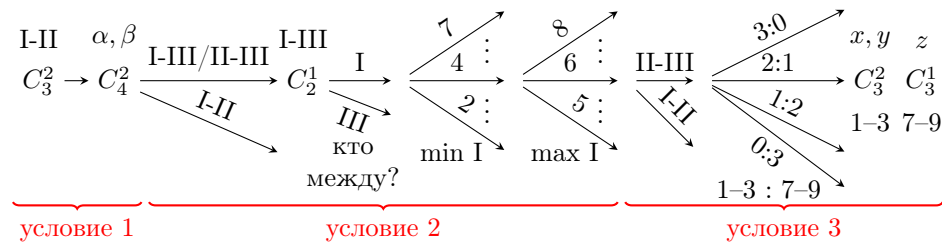
А теперь условие 2. Что значит «номера одного между номерами другого»? Это значит, что номера первого располагаются, например, между a и b (включительно), а номера второго располагаются так, что:

1. Есть хотя бы один номер меньше a .
2. Есть хотя бы один номер больше b .
3. Нет номеров внутри $[a, b]$.

Для начала, нам нужно выбрать, какая пара будет этой. Это или пара I-II, или I-III/II-III. Пусть будет I-III/II-III, а именно I-III. Далее, кто из них будет между номерами другого? I и III уже отличимы, так что для этого потребуется ветвление. Далее нам потребуется выбрать эти самые номера, причем т.к. в зависимости от этих номеров решение меняется значительно. Давайте сначала выберем a . Он не может быть 1, т.к. у III должен быть какой-то номер, который меньше a . Он не может быть 9, потому что должен быть какой-то номер, который больше. Более того, он не может быть 8, потому что открыток с одним и тем же номером 4, а брать нужно минимум 7, т.е. нам потребуется по меньшей мере 2 номера, а a — наименьший. Тогда наши варианты: 2–7. Пусть будет $a = 4$. Мы это выберем ветвлением. По аналогичным причинам b может быть от 5 до 8, пусть будет 6. Итого номера I от 4 до 6 включительно, а номера III или 1–3, или 7–9. Причем нам нужно, чтобы у I были 4 и 6 (5 — неважно), а у III должен быть хотя бы один номер из 1–3 и хотя бы один из 7–9. Дерево:



Итак, третье условие. Эта пара не может быть парой I-III, потому что у них нет общих номеров. Значит это или I-II, или II-III, причем решение будет зависеть от нашего выбора здесь, т.е. ветвление. Пусть это будет II-III. Хорошо. Но номера III располагаются от 1–3 до 7–9 и нам важно, где именно будут эти 3 общих. Как они распределяются по этим 2 компонентам? Возможно 3 : 0, 2 : 1, 1 : 2, 0 : 3. Выберем 2 : 1, например. И выберем эти номера тогда, C_3^2 — номера x, y среди 1–3, и C_3^1 — номер z среди 7–9:



Итого у I и II 2 общих вида: α, β , а табличка номеров следующая:

	x	y	?	4	5	6	z	?	?
I	×	×	×	✓	?	✓	×	×	×
II	✓	✓	?	?	?	?	✓	?	?
III	✓	✓	?	×	×	×	✓	?	?

Номера x, y, z — не обязательно 1, 2, 7 — они расположены просто в интервалах 1–3 и 7–9, но не важно, где именно в них. Далее раздача идет так же, как обычно, но с учетом всех условий (например, II должен иметь хотя бы одну карту из 4–6, потому что иначе номера I будут и между номерами II, а не только III).

5.6 Задача 6

Необходимые темы: геометрическая интерпретация ДНФ (раздел 3.5 на с. 28), максимальные конъюнкции (раздел 3.6 на с. 30).

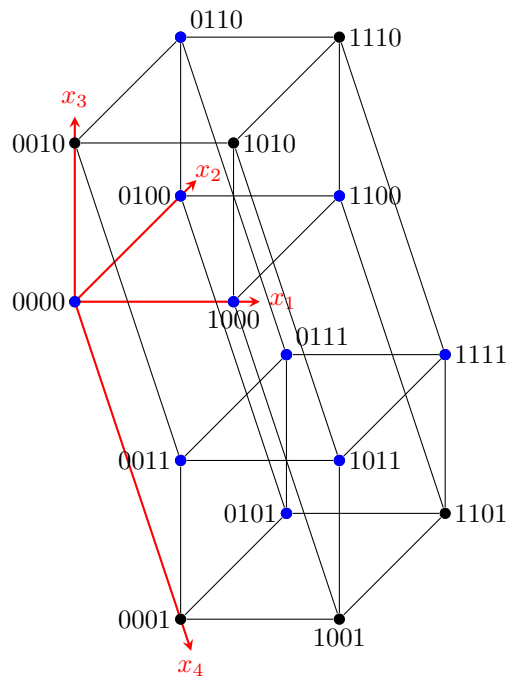
Эта задача значительно проще 5-й, в ней просто требуется найти минимальную ДНФ для некоторой функции, используя геометрическую интерпретацию ДНФ, т.е. 4-х мерный куб. У нас дана некоторая функция от 4-х переменных, мы должны составить для нее таблицу истинности (если она дана формулой, может повезти и она сразу будет дана столбиком значений), нарисовать 4-х мерный куб, отметить на нем те вершины, на которых функция равна 1, а потом покрыть это множество вершин как можно меньшим числом как можно больших граней. Грани могут пересекаться, т.е. можно покрывать одну и ту же вершину несколькими гранями, если это помогает увеличить размер грани. Тем временем, те вершины, которые не отмечены (на которых функция равна 0) нельзя покрывать в принципе. И да, эти грани нужно выделить на самом кубе, так что захватите цветные ручки на зачет. Потом нужно определить, какие конъюнкции соответствуют нашим граням. (Для каждой грани определяем, какие переменные на всех ее вершинах имеют фиксированное значение, и их и записываем в конъюнкцию. Для грани размера n их всегда будет $4 - n$.) Потом просто объединяем их дизъюнкцией и получаем нашу ДНФ.

Задача. Минимизировать геометрически $f = (x_1 \vee \overline{x_2}) \rightarrow (\overline{x_3} + x_4)$.

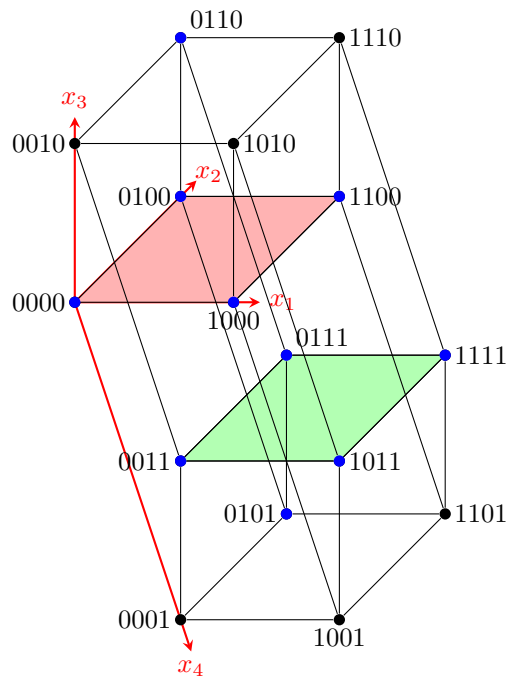
Решение. Построим таблицу истинности:

x_1	x_2	x_3	x_4	$\overline{x_2}$	\vee	$\overline{x_3}$	$+$	f
0	0	0	0	1	1	1	1	1
0	0	0	1	1	1	1	0	0
0	0	1	0	1	1	0	0	0
0	0	1	1	1	1	0	1	1
0	1	0	0	0	0	1	1	1
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	0	0	1
0	1	1	1	0	0	0	1	1
1	0	0	0	1	1	1	1	1
1	0	0	1	1	1	1	0	0
1	0	1	0	1	1	0	0	0
1	0	1	1	1	1	0	1	1
1	1	0	0	0	1	1	1	1
1	1	0	1	0	1	1	0	0
1	1	1	0	0	1	0	0	0
1	1	1	1	0	1	0	1	1

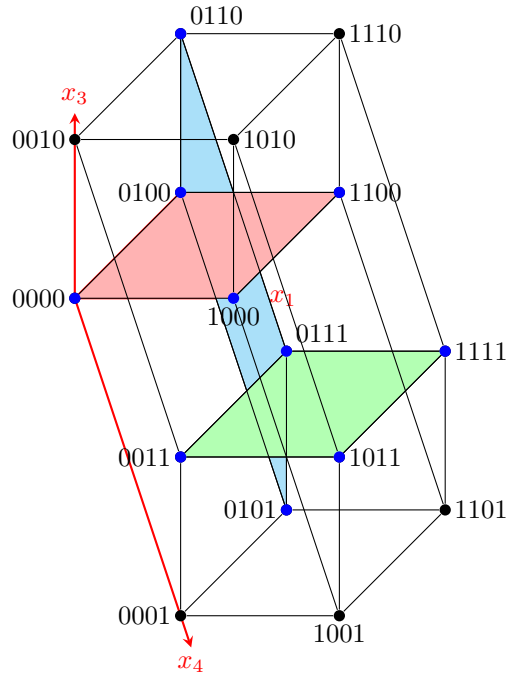
А теперь отметим эти точки на 4-х мерном кубе:



Далее грани. Сразу видно, что нижняя грань верхнего куба и верхняя грань нижнего — полностью закрашены, т.е. их можно считать за двухмерные грани, причем расширить их нельзя:



У нас остались непокрытыми только 2 точки: 0110 и 0101. Мы могли бы покрыть их просто так, точками (0-мерными гранями), но можно справиться получше. Во первых, можно использовать ребра 0100–0110 и 0101–0111, т.к. эти точки тоже использованы, это уже будет лучше. Но есть решение еще лучше — объединить эти 2 ребра в одну двумерную грань: 0100–0101–0111–0110. Да, так тоже можно. Эта грань покрывает обе точки и не выходит на не выделенные, так что так тоже можно делать. Итого получили:



Итого, все наши точки покрыты лишь 3-мя двумерными гранями. Составим конъюнкцию: у красной грани x_3 и x_4 всегда 0, так что ее конъюнкцией будет $\overline{x_3} \& \overline{x_4}$. Аналогично, у зеленой грани они всегда 1, так что ее конъюнкция: $x_3 \& x_4$. У синей грани x_1 всегда 0, а x_2 всегда 1, так что ее конъюнкция $\overline{x_1} \& x_2$. Можно рассматривать это по другому: эта грань «смещена на 0» по x_1 (все вершины не смещены), и «смещена на 1» по x_2 . Итак, общая ДНФ будет $f(x_1, x_2, x_3, x_4) = \overline{x_3} \& \overline{x_4} \vee x_3 \& x_4 \vee \overline{x_1} \& x_2$ и это — ответ. Простая задача, правда?

5.7 Задача 7

Необходимые темы: формулы (раздел 3.2 на с. 24), критерий полноты в P_2 (раздел 3.14 на с. 41).

Нам даны 2 некоторые функции φ_1, φ_2 от 3 переменных (или 1 функция φ от 4), и нам нужно при помощи них записать другую функцию h от 4 переменных, которая задана формулой, в которой используются обычные функции вроде конъюнкции, импликации и т.д. Чтобы сделать это, нужно выразить все нужные функции через φ_1, φ_2 , а это можно сделать, рассматривая таблицы φ_1, φ_2 и находя нужные нам функции внутри них. В первую очередь получаются константы и/или отрицание постановкой x во все переменные, типа $\varphi_1(x, x, x)$ (как в критерии полноты), потом получается более удобное отрицание подстановкой констант вместо переменных (тоже как в критерии), а потом мы уже ищем нужные функции внутри данных.

Задача. Выразить $\overline{(x_1|x_2) \rightarrow (\overline{x_4} + x_3)}$ через функции $\varphi_1 = 01000100$ и $\varphi_2 = 10111011$.

Решение. Итак, нам нужно получить функции $\{ |, \rightarrow, +, \overline{x} \}$. И еще неплохо было бы получить константы 0 и 1. Для начала, нужно рассмотреть таблицы истинности:

x_1	x_2	x_3	φ_1	φ_2
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

Особенно внимательно посмотрим на наборы 000 и 111. Например, $\varphi_1(0,0,0) = \varphi_1(1,1,1) = 0$. Это значит, что $h(x) = \varphi_1(x,x,x) = 0$. Да, мы можем подставлять одну и ту же переменную несколько раз, мы можем делать с переменными и аргументами функций что хотим. Аналогично $\varphi_2(x,x,x) = 1$. Итак, мы получили обе константы, это очень неплохо.

Следующий шаг — отрицание. Нам нужно найти 2 набора, отличающиеся 1 переменной, на которой значения функции — 1 и 0, именно в таком порядке. Например, $\varphi_2(0,0,0) = 1, \varphi_2(0,0,1) = 0$. Это значит, что $h(x) = \varphi_2(0,0,x) = \overline{x}$. Прекрасно, уже есть отрицание. Далее остались $|$, у которого столбик значений равен 1110, \rightarrow , у которого 1101, и $+$, у которого 0110.

Нам не очень повезло, наши функции — просто отрицания друг друга, т.е. они по факту не дают нам ничего нового. Попробуем рассмотреть функцию $\varphi_2(0, x_1, x_2)$:

0	x_1	x_2	$\varphi_2(0, x_1, x_2)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1

Она дает нам нечто, очень похожее на 1110 и 1101, а именно 1011. Нам нужно просто «переместить 0 в другое место». Для этого может пригодиться отрицание переменных, смотрите:

x_1	x_2	$f(x_1, x_2)$	$f(\overline{x_1}, x_2)$	$f(x_1, \overline{x_2})$	$f(\overline{x_1}, \overline{x_2})$
0	0	a	c	b	d
0	1	b	d	a	c
1	0	c	a	d	b
1	1	d	b	c	a

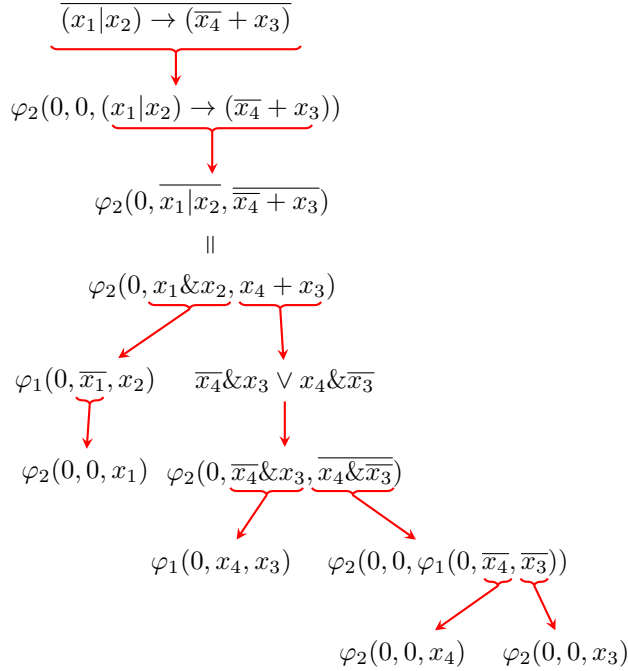
Т.е. отрицание x_1 меняет местами 1-ю строку с 3-й, а 2-ю с 4-й, отрицание x_2 — 1-ю со 2-й, а 3-ю с 4-й. В это же время отрицание обеих меняет местами 1-ю строку с 4-й, а 2-ю с 3-й. Поэтому чтобы получить 1110 из 1011, нужно сделать отрицание x_1 , а чтобы сделать 1101, нужно отрицание обеих переменных:

$$\begin{aligned} x_1 | x_2 &= \varphi_2(0, \overline{x_1}, x_2) \\ x_1 \rightarrow x_2 &= \varphi_2(0, \overline{x_1}, \overline{x_2}) \end{aligned}$$

Правда с $x_1 + x_2$ у нас проблемы, у нас нет подходящего сочетания 1 и 0 ни в одной из функций. Это значит, что нам придется собирать его вручную, используя формулу $x_1 + x_2 = \overline{x_1} \& x_2 \vee x_1 \& \overline{x_2}$. Для нее нам нужно получить еще конъюнкцию и дизъюнкцию. Их мы можем получить аналогичным образом:

$$\begin{aligned} x_1 \& x_2 &= \varphi_1(0, \overline{x_1}, x_2) \\ x_1 \vee x_2 &= \varphi_2(0, x_1, \overline{x_2}) \end{aligned}$$

Все, теперь у нас есть все компоненты формулы. Мы вряд ли сможем собрать всю формулу за раз, так что потребуется собирать ее в виде дерева:



Тогда мы сможем записать общую формулу, просто собрав ее из дерева:

$$\begin{aligned} & \varphi_2(0, 0, \varphi_2(0, \varphi_1(0, \varphi_2(0, 0, x_1), x_2), \\ & \quad \varphi_2(0, \varphi_1(0, x_4, x_3), \varphi_2(0, 0, \varphi_1(0, \varphi_2(0, 0, x_4), \varphi_2(0, 0, x_3)))))) \end{aligned}$$

Она и будет ответом на нашу задачу.

Задача. Выразить $\overline{((x_1 \sim x_2) \rightarrow \overline{x_4}) \& \overline{x_3}}$ через функцию $\varphi = 1000000000010000$.

Решение. Тут задача слегка другая, у нас есть всего лишь одна функция, но от 4 переменных. Таблица истинности:

x_1	x_2	x_3	x_4	φ
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Попробуем использовать на ней старый трюк: $\varphi(x, x, x, x)$. На наборе 0000 она дает 1, а на наборе 1111 она дает 0, т.е. $\varphi(x, x, x, x) = \overline{x}$. Итак, у нас нет даже констант. Зато есть отрицание. Но отрицание очень неудобное, чтобы что-то отрицать, надо повторить это нечто 4 раза. Ничего хорошего в общем. Как найти константы, когда при помощи $\varphi(x, x, x, x)$ не получается? Лемма о несамодвойственной функции! У нас уже есть отрицание, мы можем его использовать. Эта функция φ — обязательно несамодвойственная, т.е. на ней есть противоположные наборы, дающие одно и то же значение. И правда, например, 0001 и 1110 оба дают 0. Это значит, что $h(x) = \varphi(x, x, x, \overline{x}) = 0$, потому что при $x = 0$ у нас получается набор 0001, а при $x = 1$ — набор 1110, оба из которых дают 0. Итого, $\varphi(x, x, x, \varphi(x, x, x, x)) = 0$. 1 же можно получить, просто взяв $1 = \overline{0} = \varphi(0, 0, 0, 0)$, раз 0 у нас уже есть (благо константу 0 уже получили).

Итак, теперь у нас есть все константы и отрицание. Правда отрицание нам еще понадобится, а повторять формулу внутри 4 раза как-то не хочется. Нужно найти отрицание получше. Можно сделать это как обычно делали, например, $\varphi(0, 0, 0, x) = \overline{x}$ — вполне неплохой вариант.

А теперь к собственно тем функциям, которые нам нужны, $\sim, \rightarrow, \&$. Их столбики значений — 1001, 1101, 0001. Попробуем найти \sim . У нас уже есть две единицы на наборах 0000 и 1011, причем между ними одни 0. Тогда, мы можем, например, попробовать подставить переменные так: $\varphi(x_1, 0, x_2, x_2)$. Рассмотрим таблицу истинности этого:

x_1	x_2	x_1	0	x_2	x_2	$\varphi(x_1, 0, x_2, x_2)$
0	0	0	0	0	0	1
0	1	0	0	1	1	0
1	0	1	0	0	0	0
1	1	1	0	1	1	1

Это — как раз то, что нам нужно! Хотя я и использовал x_2 2 раза, так можно, нам главное получить эту функцию хоть в каком то виде, даже в таком.

Идем далее, \rightarrow и ее 1101. У нас даже трех 1 нет, так что нам явно нужно будет потом применить отрицание к тому, что мы найдем. Попробуем найти тогда хотя бы 0010. Как эта единица подходит только та 1, которая на наборе 1011. Мы пытаемся составить функцию 0010, т.е. 1 должна быть при значении переменных 10. Как насчет тогда $\varphi(x_1, x_2, 1, 1)$?

x_1	x_2	x_1	x_2	1	1	$\varphi(x_1, x_2, 1, 1)$	$\overline{\varphi(x_1, x_2, 1, 1)}$
0	0	0	0	1	1	0	1
0	1	0	1	1	1	0	1
1	0	1	0	1	1	1	0
1	1	1	1	1	1	0	1

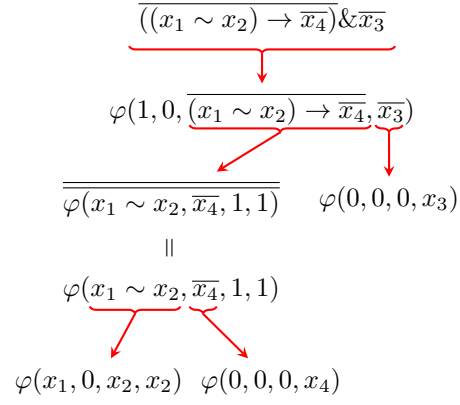
Отрицание этого — как раз то, что нужно! Осталось только $\&$. Как и раньше, единственная подходящая 1 — это на наборе 1011. Только теперь она должна быть при значениях переменных 11. Тогда стоит рассмотреть функцию $\varphi(1, 0, x_1, x_2)$:

x_1	x_2	1	0	x_1	x_2	$\varphi(1, 0, x_1, x_2)$
0	0	1	0	0	0	0
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	0	1	1	1

Работает! Итого, мы получили следующее:

$$\begin{aligned}
 \bar{x} &= \varphi(0, 0, 0, x) \\
 x_1 \sim x_2 &= \varphi(x_1, 0, x_2, x_2) \\
 x_1 \rightarrow x_2 &= \overline{\varphi(x_1, x_2, 1, 1)} \\
 x_1 \& x_2 &= \varphi(1, 0, x_1, x_2)
 \end{aligned}$$

Пора составлять дерево:



И общую формулу:

$$\varphi(1, 0, \varphi(\varphi(x_1, 0, x_2, x_2), \varphi(0, 0, 0, x_4), 1, 1), \varphi(0, 0, 0, x_3))$$

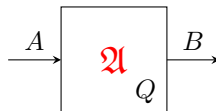
II семестр

Глава 6

Конечные автоматы¹

6.1 Определения

Для начала, что такое *конечный автомат*? Это устройство. Точнее, раз мы на дискретке, это модель устройства, обрабатывающего информацию. Как машина Тьюринга, но проще. Что еще про это устройство можно сказать? У него есть 1 вход и 1 выход. На входы поступают символы из некоторого входного алфавита A , по очереди, по одному, и как только поступает 1 символ на вход, автомат что-то делает и выдает ровно 1 символ на выход (символ из выходного алфавита B). Кроме того, у конечного автомата есть *конечное* (уер) множество состояний. В каждый момент времени он находится в одном из них и на основе того, в каком он состоянии находится сейчас, а так же символа на входе, он принимает 2 решения: первое: какой символ выдать на выход, второе: в какое состояние перейти, чтобы на следующем шаге оказаться уже в нем. И да, раз мы на дискретке, автомат работает в *дискретном времени*, не в непрерывном, т.е. автомат работает по шагам - читает 1 символ, тут же переходит в другое состояние и печатает 1 символ, а потом ждет один «такт» следующего символа, т.е. работает он только в моменты времени t_0 , $t_0 + 1$, $t_0 + 2$ и т.д., не между ними.



Теперь более формально.

Определение. Конечным автоматом называется пятерка объектов $\mathfrak{A} = (A, B, Q, \varphi, \psi)^2$, где

- A — входной алфавит (множество символов, возможных на входе).

¹ «Правильно говорить так, да, а не конченые» — К.И. Костенко

² Да, конечные автоматы обозначаются готическими буквами, например \mathfrak{A} («А») или \mathfrak{T} («Т»).

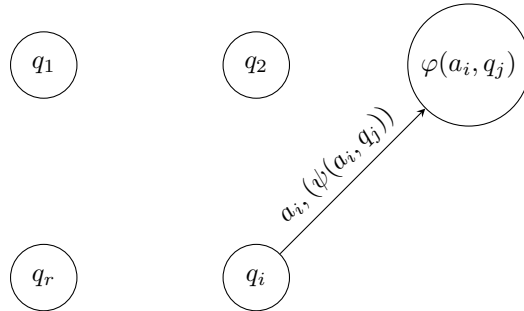
- B — выходной алфавит (множество символов, возможных на выходе).
- Q — множество состояний автомата.
- $\varphi : A \times Q \rightarrow Q$ — функция перехода, 2 аргумента: символ на входе и текущее состояние, а на выходе определяется следующее состояние.
- $\psi : A \times Q \rightarrow B$ — функция выхода, те же 2 аргумента, но выдает символ, который будет на выходе.

Это определение аналогично определению графа — множество вершин и множество ребер, только тут автомат «составляют» 5 объектов: 3 множества и 2 функции.

Как и у графов, мы почти не будем на практике использовать это определение, а в основном будем представлять автоматы более удобными способами.

1. *Диаграмма переходов.*

Самый удобный из всех. Каждое состояние автомата — кружочек с подписью типа q_1, q_2, \dots, q_r . Из каждого кружочка выходит несколько стрелок, каждая стрелка соответствует одному символу входного алфавита. Т.е. когда автомат находится в состоянии q_i и читает символ a_j , он перейдет в следующее состояние $(\varphi(a_j, q_i))$ по стрелке, и напечатает символ $(\psi(a_j, q_i))$, написанный в скобках рядом с a_j . Вот так:



2. *Табличное задание автомата.*

Самый неудобный из всех. Мы рисуем 2 таблицы, одну для φ , другую для ψ . Строки — символы входного алфавита, столбцы — состояния. Итого это выглядит примерно так:

φ	q_1	\dots	q_j	\dots	q_r
a_1			\vdots		
\vdots			\vdots		
a_i	\dots	\dots	$\varphi(a_i, q_j)$		
\vdots					
a_n					

ψ	q_1	\cdots	q_j	\cdots	q_r
a_1			\vdots		
\vdots			\vdots		
a_i	\cdots	\cdots	$\psi(a_i, q_j)$		
\vdots					
a_n					

3. Канонические уравнения.

Самый формальный и точный из всех. Буквально запись автомата при помощи формул. Мы указываем 3 уравнения, описывающих процесс работы автомата, используя следующие функции: $x(t)$ — входной символ в момент t , $y(t)$ — выходной, $q(t)$ — состояние в момент t . Тогда системой уравнений, описывающей автомат будет:

$$\begin{aligned} q(t_0) &= q_0 \\ q(t+1) &= \varphi(x(t), q(t)) \\ y(t) &= \psi(x(t), q(t)) \end{aligned}$$

Они описывают, соответственно, начальное состояние автомата в момент t_0 , как вычисляется следующее состояние (обычно — при помощи φ), и как вычисляется выходной символ (при помощи ψ).

На этом наши способы представлять автоматы закончились. В основном мы будем пользоваться первым, но немного и третьим.

Еще нам потребуется понятие *переработки слов автоматом*. Пусть у нас дан автомат $\mathfrak{A} = (A, B, Q, \varphi, \psi)$, и в начальный момент он находится в состоянии q_0 . Если мы будем ему по очереди давать символы некоторого слова $\alpha = \sigma_1\sigma_2 \cdots \sigma_k$, то он в это же время выведет некоторое слово β . Тогда, если автомат \mathfrak{A} из начального состояния q_0 «перерабатывает» слово α в слово β , то мы это будем обозначать $\beta = f_{\mathfrak{A}}^{q_0}(\alpha)$. И да, иногда мы еще будем употреблять обозначение $\varphi(\alpha, q_0)$ — состояние, в которое автомат попадает после переработки всего слова α , а не просто одного символа, как $\varphi(a, q_0)$.

6.2 Числовые функции, вычисляемые автоматами

Пусть у нас есть некоторая числовая функция $f : \mathbb{N}^k \rightarrow \mathbb{N}$, причем $\mathbb{N} = \{0, 1, 2, \dots\}$, т.е. ноль разрешен, которой подается k чисел как аргументы, и которая выдает натуральное число. Нам требуется как-то приспособить автоматы для вычисления подобных функций. По идее, наши автоматы могут работать в любой системе счисления, но с двоичной работать проще, так что давайте представлять наши числа как двоичную запись $\sigma_1\sigma_2 \cdots \sigma_n$. Еще одна вещь, большинство операций, например, сложение, мы выполняем

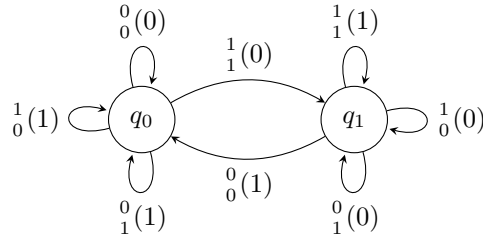


Рис. 6.1: Автомат, выполняющий сложение

от младших разрядов к старшим, т.е. справа налево, но мы определили, что автоматы читают слова слева направо, т.е. чтобы дать число x автомату, нужно будет его предварительно развернуть, что мы будем обозначать как \bar{x} .

И еще одна вещь, если наша функция принимает несколько элементов, как мы будем подавать их все автомату с одним входом? По очереди не получится, потому что, первое, у автомата недостаточно «памяти», чтобы запомнить слово произвольной длины, и второе, нам тогда придется вводить разделительные символы. Поэтому мы их будем давать одновременно, т.е. входным алфавитом автомата будет E_2^k , т.е. один символ — это k бит за раз. Выходным же будет всегда 1 бит, т.е. E_2 . Возникает еще одна проблема, наши числа могут быть разной длины. Тогда мы просто дозаполняем из незначащими нулями, чтобы выровнять длины, и возможно еще немного нулей, чтобы автомат смог закончить работу. Сколько именно нулей добавлять — не так важно. Итак, если нам например требуется подать числа 5, 11, 2, т.е. 101, 1011, 10 в двоичном виде, мы должны их развернуть: 101, 1101, 01, добавить незначащие нули (обычно в начало, но т.к. мы их развернули — в конец): 1010, 1101, 0100, и подавать автомату по 3 бита за раз, т.е. всего будет 4 символа: (1, 1, 0), (0, 1, 1), (1, 0, 0), (0, 1, 0). Пора наконец дать определение числовой функции, вычисляемой автоматом:

Определение. Говорят, что автомат $\mathfrak{A} = (E_2^k, E_2, Q, \varphi, \psi)$ вычисляет функцию $f : \mathbb{N}^k \rightarrow \mathbb{N}$ из некоторого начального состояния q_0 , если $\forall x_1, x_2, \dots, x_k$ автомат \mathfrak{A} из начального состояния q_0 перерабатывает (x_1, \dots, x_k) — преобразованные вышесказанным образом числа в $f(x_1, \dots, x_k)$, т.е. если полученное из автомата слово развернуть, мы получим выходное значение функции на этих аргументах.

Пример. Попробуем составить автомат для сложения, т.е. который вычисляет функцию $f(x, y) = x + y$. Этому автомату на вход будет подаваться по 2 бита за такт, т.е. это будет автомат $\mathfrak{A} = (E_2^2, E_2, Q, \varphi, \psi)$. (см. рисунок 6.1)

Когда мы вычисляем сумму в столбик, мы запоминаем «перенос», автомату тоже это нужно запоминать, поэтому ему нужно 2 состояния для сохранения переноса. Начинаем мы из состояния q_0 , где переноса нет. Далее будем считать в двоичной системе.

1. $\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}$, мы считаем $0 + 0 = 0$, и автомат печатает младший разряд (0), и переноса не возникает, т.е. мы остаемся в q_0 (петля).
2. $\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}$, мы считаем $0 + 1 = 1$, автомат печатает 1 и остается в q_0 .
3. $\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}$, аналогично, $1 + 0 = 1$, автомат печатает 1 и остается в q_0 .
4. $\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}$, $1 + 1 = 10$, автомат печатает 0 и возникает перенос 1, т.е. автомат должен перейти в q_1 .

Далее аналогично для состояния q_1 . Там есть перенос, так что мы должны добавить +1 ко всем вычислениям:

1. $\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}$, мы считаем $0 + 0 + 1 = 1$, автомат печатает 0, и возвращается в q_0 , т.к. перенос не возникает.
2. $\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}$, мы считаем $0 + 1 + 1 = 10$, автомат печатает 0 и остается в q_1 , т.к. снова возникает перенос.
3. $\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}$, аналогично, $1 + 0 + 1 = 10$, автомат печатает 0 и остается в q_1 .
4. $\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}$, $1 + 1 + 1 = 11$, автомат печатает 1 и остается в q_1 .

На этом построение автомата полностью закончено, он не может перейти ни в какие другие, неизвестные нам, состояния.

Невычислимость автоматами умножения

Важным фактом является то, что автоматы не могут считать абсолютно любые функции, только некоторые. Например, умножение 2 произвольно больших чисел не может быть вычислено конечным автоматом.

Теорема. $f(x, y) = x \cdot y$ не может быть вычислено конечным автоматом.

Доказательство. Предположим, что это возможно. Тогда, по предыдущему определению, автомат, который вычисляет умножение, будет выглядеть как $\mathfrak{A} = (E_2^2, E_2, Q, \varphi, \psi)$. Пусть мы умножаем 2 двоичных числа длины n . По правилам математики, доказательство которых не относится к дискретке, длина их произведения будет порядка $2n$, можете проверить сами. Что это значит с точки зрения автомата? Автомат должен выдавать ответ одновременно с подачей входных значений, т.е. когда мы полностью подадим наши числа длины n , автомат уже напишет младшие (правые) n разрядов ответа, и ему останется написать еще n . Напоминаю, сначала вводятся младшие (правые) биты, а потом старшие (левые). Это выглядит примерно так:

$$\begin{array}{rcc}
 & n & n \\
 & \boxed{0 \cdots 0} & \boxed{\sigma_1 \cdots \sigma_n} \quad x \\
 \times & & \vdots \\
 & \boxed{0 \cdots 0} & \boxed{\delta_1 \cdots \delta_n} \quad y \\
 = & & \vdots \\
 & \boxed{2n} &
 \end{array}$$

Не важно как вычисляются правые n бит результата, рассмотрим лучше левые (старшие) n бит. Так как мы можем умножать любые числа длины n , то мы можем получать какие угодно выходные значения, в том числе в этих старших n битах результата. Так как на момент выдачи этих старших n бит результата на вход подаются одни нули, автомат должен уметь выдавать любые последовательности из n бит самостоятельно, не полагаясь на вход, полагаясь только на состояние, в котором он окажется после ввода младших n бит. Что это значит? Всего разных комбинаций старших n бит возможно 2^n , и т.к. каждой такой последовательности должно соответствовать такое состояние, из которого он ее напечатает (когда ему подадут на входы нули, разумеется), то состояний должно быть тоже не меньше 2^n . Итак, для обработки чисел длины n нам нужно $|Q| \geq 2^n$. Значит мы всегда сможем подобрать такое n , чтобы у любого конкретного автомата не хватало состояний для его обработки, значит у любого автомата будет своя максимально допустимое n , и значит умножение чисел произвольной длины при помощи *конечных* автоматов сделать невозможно. \square

6.3 Переработка автоматами периодических сверхслов

У конечных автоматов есть еще много разных ограничений, мы рассмотрим еще одно. Для этого нам потребуется понятие *сверхслова*, это просто *бесконечная* последовательность символов, так же как обычное *слово* было *конечной* последовательностью. Множество всех сверхслов, составленных из алфавита A , обозначается A^∞ . Мы будем рассматривать периодические сверхслова, т.е. сверхслова $\alpha \in A^\infty$ вида $\beta\gamma\gamma\gamma\gamma\cdots$, или $\beta(\gamma)^\infty$, где $\beta, \gamma \in A^*$. И да, $f_{\mathfrak{A}}^q(\alpha)$ — сверхслово, получаемое переработкой автоматом \mathfrak{A} из начального состояния q сверхслова α .

Теорема. Для любого автомата $\mathfrak{A} = (A, B, Q, \varphi, \psi)$, $q \in Q$, если $\alpha \in A^\infty$ — периодическое, то и $f_{\mathfrak{A}}^q(\alpha)$ — периодическое.

Доказательство. Наше слово имеет вид $\beta\gamma\gamma\gamma\gamma\cdots$. Давайте обозначим состояния, в которые попадает автомат *перед* переработкой очередной γ , как q', q'', q''', \dots . Так как наше множество состояний — конечно, когда-нибудь мы встретим какое-то состояние второй раз: $q^{(s)} = q^{(p)}$, где $s < p$, т.е. $q^{(s)}$ встретилось нам раньше, чем $q^{(p)}$. Тогда слово, $f_{\mathfrak{A}}^q(\alpha) = f_{\mathfrak{A}}^q(\beta\gamma^\infty)$ можно выразить как переработку сначала начала $\beta\gamma^{s-1}$ из состояния q , а потом многократную переработку γ^{p-s} из состояния $q^{(s)}$, после которой мы попадаем в состояние $q^{(p)}$ и начинаем эту же переработку заново. Т.е.

$$f_{\mathfrak{A}}^q(\alpha) = f_{\mathfrak{A}}^q(\beta\gamma^\infty) = f_{\mathfrak{A}}^q(\beta\gamma^{s-1}) \left(f_{\mathfrak{A}}^{q^{(s)}}(\gamma^{p-s}) \right)^\infty$$

\square

6.4 Отличимость состояний автомата

Состояния q' и q'' автомата \mathfrak{A} называются неотличимыми, если абсолютно все входные слова дают одинаковый результат, если использовать эти состояния как начальные: $f_{\mathfrak{A}}^{q'}(\alpha) = f_{\mathfrak{A}}^{q''}(\alpha)$. Ну и, очевидно, состояния называются отличимыми, если есть какое-то слово, которое дает разные выходы, если начинать с этих 2 состояний. Это — абсолютно отличимые/неотличимые состояния. Нам пригодится еще понятие k -неотличимых состояний, т.е. состояний, неотличимых на всех словах длины не больше k . Определим отношение абсолютной неотличимости $\varepsilon \subseteq Q \times Q$ между состояниями автомата, а так же отношения k -неотличимости, $\rho_k \subseteq Q \times Q$. Т.е. если состояния k -неотличимы, то $q' \rho_k q''$, а если всегда неотличимы — то $q' \varepsilon q''$. Обнаруживается, что $\varepsilon, \rho_1, \rho_2, \dots$ — это отношения эквивалентности и по известной нам теореме они разбивают Q на классы эквивалентных (неотличимых/ k -неотличимых) состояний. Дальше мы будем доказывать одну большую теорему, и нам потребуется куча лемм для нее, так что начнем:

Лемма. $\rho_1 \supseteq \rho_2 \supseteq \dots \supseteq \rho_i \supseteq \dots \supseteq \varepsilon$.³ Иначе, $\forall i \in \mathbb{N}(\rho_i \supseteq \rho_{i+1})$ и $\forall i \in \mathbb{N}(\rho_i \supseteq \varepsilon)$.

Доказательство. Лемма состоит из 2 частей, значит и доказательство из 2 частей:

1. Достанем из глубин памяти определение \supseteq : это значит, что для любой пары $(q', q'') \in \rho_{i+1}$ выполняется $(q', q'') \in \rho_i$. Докажем это. Если $(q', q'') \in \rho_{i+1}$, то, по определению $i + 1$ -неотличимости, для всех слов длины $\leq i + 1$ автомат дает одинаковые результаты для 2 состояний. Очевидно, что это выполняется и для всех слов длины $\leq i$, т.е. что $(q', q'') \in \rho_i$, что мы и хотели доказать.
2. $(q', q'') \in \varepsilon \stackrel{?}{\implies} (q', q'') \in \rho_i$. Если q' и q'' — неотличимы на любых словах, то и на словах длины $\leq i$ они очевидно неотличимы.

□

Лемма. Если $\rho_i = \rho_{i+1}$, то $\forall k(\rho_i = \rho_{i+k})$ и $\rho_i = \varepsilon$.

Доказательство. Что это значит, для начала? Это значит, что если у нас в какой-то момент при увеличении допустимой для проверки длины слов не теряется ни одна пара неотличимых состояний, т.е. если мы добавили 1 букву к допустимой длине слов, и пары неотличимых состояний остались те же, то они такие и будут при дальнейшем увеличении длины, т.е. эти пары — и есть в принципе неотличимые состояния. Будем доказывать это в несколько шагов:

³Обратите внимание на необычное направление знаков, т.е. оно значит, что ρ_1 включает в себя ρ_2 , т.е. что ρ_1 больше, чем ρ_2 , т.е. что 1-неотличимых состояний больше, чем 2-неотличимых, не наоборот.

1. $\rho_i = \rho_{i+2}$. Т.е. если у нас при переходе от слов длины i к словам длины $i+1$ не потерялась ни одна пара неотличимых состояний, то они не потеряются и при переходе к $i+2$.

Мы уже доказали ранее, что $\rho_i \supseteq \rho_{i+1} \supseteq \rho_{i+2}$, т.е. что пары неотличимых состояний не могут добавляться при увеличении длины, только теряться. Нам осталось доказать, что они и не теряются, т.е. что $(q', q'') \in \rho_i \rightarrow (q', q'') \in \rho_{i+2}$. Итак, что нам дано? Нам дано, что два состояния q' и q'' i -неотличимы, и что $\rho_i = \rho_{i+1}$. Нам нужно доказать, что для любого слова α длины $i+2$ они тоже будут неотличимы: $f_{\mathfrak{A}}^{q'}(\alpha) = f_{\mathfrak{A}}^{q''}(\alpha)$.

Для начала выделим первый символ слова $\alpha = a\beta$, $a \in A, \beta \in A^*$, причем $|\alpha| = i+2, |\beta| = i+1$. Тогда $f_{\mathfrak{A}}^{q'}(\alpha) = f_{\mathfrak{A}}^{q'}(a\beta) = f_{\mathfrak{A}}^{q'}(a)f_{\mathfrak{A}}^{\varphi(a, q')}(\beta)$, т.е. что сначала мы разбираем символ a , а потом уже остальное слово β , но уже из следующего состояния $\varphi(a, q')$. Для q'' аналогично: $f_{\mathfrak{A}}^{q''}(\alpha) = f_{\mathfrak{A}}^{q''}(a\beta) = f_{\mathfrak{A}}^{q''}(a)f_{\mathfrak{A}}^{\varphi(a, q'')}(\beta)$. Поскольку наши q' и q'' i -неотличимы, то очевидно, что 1 символ они будут разбирать одинаково: $f_{\mathfrak{A}}^{q'}(a) = f_{\mathfrak{A}}^{q''}(a)$. Далее, раз они i -неотличимы, а по условию нашей теоремы $\rho_i = \rho_{i+1}$. Это значит, что они и $i+1$ -неотличимы, т.е. они перерабатывают любые слова длины $i+1$ одинаково. Это значит, что любые состояния, в которые можно попасть из q' и q'' по одному символу (например, $\varphi(a, q')$ и $\varphi(a, q'')$), будут i -неотличимы, т.к. они должны будут одинаково перерабатывать любой остаток слова длины i одинаково, т.е. если мы начинаем с них, то любое слово длины i перерабатывается одинаково. А т.к. $\rho_i = \rho_{i+1}$, то они и $i+1$ неотличимы, а это означает, что они переработают β одинаково: $f_{\mathfrak{A}}^{\varphi(a, q')}(\beta) = f_{\mathfrak{A}}^{\varphi(a, q'')}(\beta)$. Тогда $f_{\mathfrak{A}}^{q'}(\alpha) = f_{\mathfrak{A}}^{q''}(\alpha)$ для любого α длины $i+2$, что мы и хотели доказать.

2. $\forall k(\rho_i = \rho_{i+k})$, т.е. что все следующие состояния будут равны нашему ρ_i .

Доказывается элементарно, по индукции: если $\rho_i = \rho_{i+1} = \rho_{i+2}$, то мы можем применить те же рассуждения для $\rho_{i+1} = \rho_{i+2}$, и получить $\rho_{i+1} = \rho_{i+2} = \rho_{i+3}$, а из $\rho_{i+2} = \rho_{i+3}$ получить $\rho_{i+2} = \rho_{i+4}$, и продолжать так сколько угодно.

3. $\rho_i = \varepsilon$, т.е. что все i -неотличимые состояния — и есть всегда неотличимые.

Нам нужно доказать, что $(q', q'') \in \rho_i \rightarrow (q', q'') \in \varepsilon$. Рассмотрим любое слово $\alpha \in A^*$. Если его длина $\leq i$, то из $(q', q'') \in \rho_i$ очевидно следует $f_{\mathfrak{A}}^{q'}(\alpha) = f_{\mathfrak{A}}^{q''}(\alpha)$. Если же его длина будет некое $i+k$, то используя $\rho_i = \rho_{i+k}$ мы получим, что $(q', q'') \in \rho_{i+k}$, т.е. что эти вершины будут и $i+k$ -неотличимы и $f_{\mathfrak{A}}^{q'}(\alpha) = f_{\mathfrak{A}}^{q''}(\alpha)$. Значит для любого слова α мы сможем доказать, что состояния всегда выдают одно и то же, зайдя достаточно далеко по цепочке $\rho_i = \rho_{i+1} = \rho_{i+2} = \dots$. Значит $\rho_i = \varepsilon$.

□

Лемма. Разбиение Q на классы эквивалентности, которое порождает ρ_{i+1} (классы $i+1$ -неотличимости), является подразбиением классов, которые порождает ρ_i (классы i -неотличимости).

Доказательство. Термин «подразбиение» означает разбиение разбиения, т.е. что классы $i+1$ -неотличимости разбивают классы i -неотличимости на еще меньшие множества. Так как мы уже доказали, что $\rho_i \supseteq \rho_{i+1}$, то все классы $i+1$ -неотличимости будут лежать внутри классов i -неотличимости, т.е. это разбиение будет являться подразбиением. □

А теперь, та теорема, которую мы все так ждали:

Теорема. Если нам даны состояния $q', q'' \in Q$ автомата $\mathfrak{A} = (A, B, Q, \varphi, \psi)$, то для доказательства отличимости этих 2 состояний необходимо проверить их только на словах длины $|Q| - 1$, т.е. если они — отличимы, то существует такое слово α длины $|Q| - 1$, что выходные слова автомата будут отличаться: $f_{\mathfrak{A}}^{q'}(\alpha) \neq f_{\mathfrak{A}}^{q''}(\alpha)$.

Доказательство. Из доказанных лемм мы знаем, что отношения неотличимости вложены друг в друга так: $\rho_1 \supset \rho_2 \supset \dots \supset \rho_i = \rho_{i+1} = \dots = \varepsilon$, т.е. что до некоторого ρ_i у нас идут только строгие вложения, количество пар неотличимых состояний автомата строго уменьшаются, а после ρ_i остаются только пары всегда неотличимых состояний.

Мы можем доказать, что ρ_1 разбивает Q по крайней мере на 2 класса, т.к. иначе все состояния будут неотличимы, ведь, все состояния будут перерабатывать одиночные символы одинаково, а после переработки этого одного символа автомат будет снова попадать в два 1-неотличимых состояния, т.е. они будут перерабатывать все слова абсолютно одинаково. Мы же предположили, что там есть хотя бы 2 отличимых состояния, q' и q'' . Далее это будет значить, что все отношения в нашей цепочке до ρ_i будут добавлять хотя бы по 1 классу неотличимости (иначе эти отношения мы совпали, а мы обозначили первую пару совпадающих отношений как $\rho_i = \rho_{i+1}$). Итак, ρ_1 будет разбивать Q на ≥ 2 класса, ρ_2 — на ≥ 3 класса, ρ_3 — на ≥ 4 и т.д., вплоть до ρ_i , разбивающего Q на $i+1$ классов. Так как в каждом классе есть хотя бы одно состояние, то всего классов будет не больше, чем состояний: $i+1 \leq |Q|$, т.е. $i \leq |Q| - 1$. Если q' и q'' — отличимые состояния, то $(q', q'') \notin \varepsilon$, а т.к. $\varepsilon = \rho_i$, то они будут i -отличимыми, т.е. на каком то слове α длины i они будут давать разные результаты, но $i \leq |Q| - 1$, значит существует слово α длины $\leq |Q| - 1$, на котором q' и q'' дают разные результаты $f_{\mathfrak{A}}^{q'}(\alpha) \neq f_{\mathfrak{A}}^{q''}(\alpha)$. □

Пример.

Эта теорема дает нам верхнюю оценку длины слова, необходимую для доказательства отличимости/неотличимости состояний. Если мы хотим доказать неотличимость 2 состояний, нам необходимо проверить их на всех

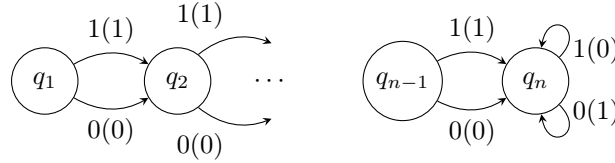


Рис. 6.2: Автомат с отличимыми состояниями

словах длины $|Q| - 1$. Более того, эта оценка является достижимой, т.е. существуют автоматы, в которых есть 2 состояния, ведущих себя одинаково на всех словах длины $|Q| - 2$, но отличающиеся на словах длины $|Q| - 1$. Пример такого автомата см. на рисунке, где q_1, q_2 — отличимые состояния, для проверки которых нужны слова длины $n - 1$.

6.5 Минимизация автомата

Так же, как и с функциями алгебры логики, было бы неплохо как-то упростить автоматы. Это и называется минимизацией. Для начала, нам нужно понятие *эквивалентных* автоматов. Автоматы эквивалентны, если они вычисляют одни и те же функции, т.е. множества вычисляемых ими функций равны. Или, по другому, для каждого состояния первого автомата есть такое состояние второго, которое вычисляет ту же функцию, и наоборот. Ну, это достаточно интуитивно понятно.

Еще нам нужно то, к чему мы собственно будем стремиться, *минимальный автомат*. Это такой автомат, у которого все состояния отличимы. Как мы далее увидим, чтобы минимизировать автомат, нужно объединить всего его неотличимые состояния. Далее как раз последует очень большая и сложная теорема об этом.

Теорема. Для любого автомата $\mathcal{A} = (A, B, Q, \varphi, \psi)$ существует эквивалентный ему минимальный автомат $\mathcal{T} = (A, B, Q', \Phi, \Psi)$. Иначе, любой автомат можно минимизировать, как-то изменив его множества состояний и функции перехода и выхода.

Доказательство. Что ж, приступим. Для начала, если наш автомат \mathcal{A} уже минимальный, то ничего делать не надо, сразу исключим этот вариант. Сначала нам нужно будет построить наш минимальный автомат \mathcal{T} , потом доказать, что он минимальный и что он эквивалентен \mathcal{A} .

1. Построение \mathcal{T} .

Интуитивно, в чем заключается процесс минимизации автомата? Мы ищем неотличимые состояния в \mathcal{A} и «объединяем» их в одно состояние в автомате \mathcal{T} . Тогда наш автомат \mathcal{T} будет минимальным и эквивалентным \mathcal{A} . Теперь нужно доказать, что это всегда возможно и что у нас

не будет возникать проблем в процессе. Для начала, нам нужно ввести кучу обозначений:

Пусть $Q = \{q_1, q_2, \dots, q_n\}$ — состояния исходного автомата \mathfrak{A} . Раз там есть неотличимые состояния, то отношение эквивалентности ε разбивает Q на классы неотличимых состояний, пусть они будут Q_1, Q_2, \dots, Q_d . Напомню, что это значит — все состояния внутри классов неотличимы между собой, но состояния из разных классов — отличимы. Если состояния автомата \mathfrak{A} мы обозначаем нижними индексами, то, для ясности, состояния автомата \mathfrak{T} мы будем обозначать верхними индексами: $Q' = \{q^1, q^2, \dots, q^d\}$. Их будет d , столько же, сколько классов неотличимости в автомате \mathfrak{A} , потому что, напоминая, мы будем все неотличимые состояния, т.е. весь класс неотличимости, объединять в единую вершину автомата \mathfrak{T} , т.е. каждому классу неотличимости Q_i будет соответствовать некое состояние q^i автомата \mathfrak{T} .

Было бы хорошо, если мы могли бы закончить на этом ввод обозначений, но нет. Нам потребуются еще 2 вспомогательные функции.

Первая, $\xi : Q \rightarrow \{1, \dots, d\}$, принимает состояние автомата \mathfrak{A} и выдает номер класса неотличимости, в котором оно лежит. Т.е. если $q_i \in Q_j$, то $\xi(q_i) = j$. Ну и очевидно, что каждое состояние лежит только в одном классе неотличимости, так что эта функция всегда будет выдавать одно значение для любой вершины, и вообще все хорошо.

Вторая, $\eta : \{1, \dots, d\} \rightarrow \{1, \dots, n\}$, в каком-то смысле противоположна: она принимает номер класса неотличимости и выдает номер одного из состояний внутри. Для определенности, пусть она выдает номер минимального состояния там. Т.е. если, например, $Q_3 = \{q_5, q_7, q_{11}, q_{12}\}$, то $\eta(3) = 5$. Более формально, $\eta(i) = \min_{q_t \in Q_i} t$, т.е. минимальный номер состояния в классе Q_i .

На этом наши обозначения все, можно наконец то объяснить, что мы будем делать. Мы будем каждому классу неотличимости Q_i в \mathfrak{A} сопоставлять по состоянию q^i в \mathfrak{T} (см. рисунок 6.3). За прототип этого состояния мы возьмем $q_{\eta(i)}$, т.е. состояние с минимальным номером в этом классе неотличимости.

Далее, вопрос, что делать с функциями, например, с функцией перехода? У нас есть функция $\varphi : A \times Q \rightarrow Q$, нам необходимо получить функцию $\Phi : A \times Q' \rightarrow Q'$. Давайте попробуем посчитать $\Phi(a, q^i)$ (см. рисунок). Прототипом для q^i является $q_{\eta(i)}$. Куда переходит автомат \mathfrak{A} из состояния $q_{\eta(i)}$ при чтении символа a ? В $\varphi(a, q_{\eta(i)})$! Какому состоянию \mathfrak{T} это соответствует? А в каком классе неотличимости это лежит? У нас как раз есть функция для этого, $\xi(\varphi(a, q_{\eta(i)}))$. Итак,

$$\Phi(a, q^i) = q^{\xi(\varphi(a, q_{\eta(i)}))}.$$

Что насчет $\Psi : A \times Q' \rightarrow A$? Она еще проще, прототипом для q^i

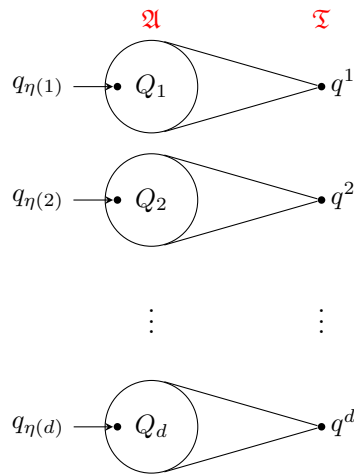


Рис. 6.3: Минимизация автомата

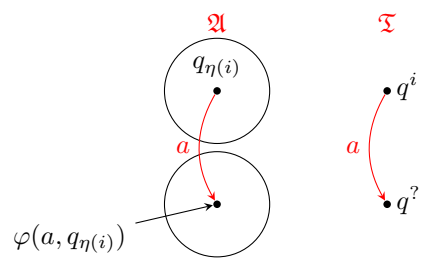


Рис. 6.4: Минимизация автомата

является $q_{\eta(i)}$, так что

$$\Psi(a, q^i) = \psi(a, q_{\eta(i)}).$$

Итак, мы полностью определили автомат \mathfrak{T} , неплохо. Теперь самое сложное, доказать, что мы это делали не просто так.

2. Доказательство, что \mathfrak{T} — минимальный автомат, эквивалентный \mathfrak{A} .

Это доказательство достаточно сложное, так что разобьем его еще дальше:

- (а) Докажем, что $\forall q^i \in Q'$ выполняется $f_{\mathfrak{A}}^{q_{\eta(i)}} \equiv f_{\mathfrak{T}}^{q^i}$, т.е. что состояние $q_{\eta(i)}$ автомата \mathfrak{A} на всех входных словах дает тот же результат, что и состояние q^i автомата \mathfrak{T} .

Мы доказываем, что хотя бы наше состояние q^i соответствует его прототипу $q_{\eta(i)}$. Будем доказывать это дело индукцией по длине входного слова. И еще одно, нам потребуется параллельно доказывать еще один факт, что состояния $\varphi(\alpha, q_{\eta(i)})$ (состояние, куда попал автомат \mathfrak{A} после переработки слова α из состояния $q_{\eta(i)}$) и состояние $q_{\eta(s)}$, где $q^s = \Phi(\alpha, q^i)$ (состояние, которое соответствует тому, которое в которое попадает автомат \mathfrak{T} после переработки α из состояния q^i) — неотличимы. Итак, индукция:

i. Базис индукции, $|\alpha| = k = 1$.

Слово состоит из одного символа, $\alpha = a$, все просто. Слово (тоже из одного символа), которое выдаст автомат \mathfrak{A} , будет $f_{\mathfrak{A}}^{q_{\eta(i)}}(a) = \psi(a, q_{\eta(i)})$. По определению Ψ , это как раз $\Psi(a, q^i)$, а это как раз то, что выдаст автомат \mathfrak{T} , т.е. $f_{\mathfrak{T}}^{q^i}(a)$, все доказано.

У нас правда есть еще одно утверждение, которое надо доказать. Пусть то состояние, куда перейдет автомат \mathfrak{A} , т.е. $\varphi(a, q_{\eta(i)})$, лежит в классе Q_s . Тогда, по определению, $\xi(\varphi(a, q_{\eta(i)})) = s$, а значит $\Phi(a, q^i) = q^s$, как мы и думали. Итак, у нас есть $\varphi(a, q_{\eta(i)}) \in Q_s$ и $q_{\eta(s)} \in Q_s$, лежащие в одном классе неотличимости, т.е. неотличимые, что мы и хотели доказать.

ii. Индуктивное предположение, $|\alpha| = k = l$.

Как обычно, предполагаем, что все хорошо работает для слов длины l , т.е. если $|\alpha| = l$, то $f_{\mathfrak{A}}^{q_{\eta(i)}}(\alpha) = f_{\mathfrak{T}}^{q^i}(\alpha)$, а еще $\varphi(\alpha, q_{\eta(i)})$ и $q_{\eta(s)}$, где $q^s = \Phi(\alpha, q^i)$ — неотличимые состояния.

iii. Индуктивный переход, $|\alpha| = k = l + 1$.

Тут немного проблематичнее, но ладно. Пусть у нас есть некоторое слово α , причем $|\alpha| = l + 1$. Это значит, что мы можем отделить от него последнюю букву, т.е. $\alpha = \beta a$, где $\beta \in A^*$, $a \in A$, причем $|\beta| = l$. Мы пытаемся доказать, что $f_{\mathfrak{A}}^{q_{\eta(i)}}(\alpha) = f_{\mathfrak{T}}^{q^i}(\alpha)$. Приступим.

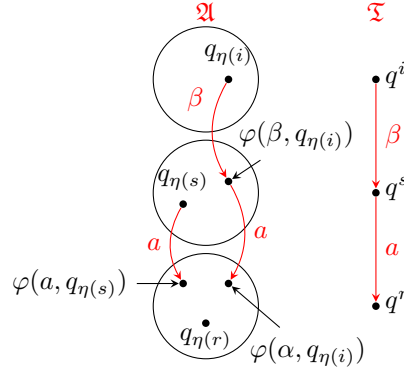


Рис. 6.5: Минимизация автомата

Найдем сначала левую часть, $f_{\mathfrak{A}}^{q_{\eta(i)}}(\alpha) = f_{\mathfrak{A}}^{q_{\eta(i)}}(\beta a)$. Сначала автомат перерабатывает β , а потому a , т.е. $f_{\mathfrak{A}}^{q_{\eta(i)}}(\alpha) = f_{\mathfrak{A}}^{q_{\eta(i)}}(\beta) f_{\mathfrak{A}}^{\varphi(\beta, q_{\eta(i)})}(a)$. Пока закончим на этом, и найдем правую часть, $f_{\mathfrak{T}}^{q^i}(\alpha) = f_{\mathfrak{T}}^{q^i}(\beta a)$. Давайте введем обозначение, что после переработки β автомат \mathfrak{T} переходит в состояние q^s , т.е. $\Phi(\beta, q^i) = q^s$. Тогда $f_{\mathfrak{T}}^{q^i}(\alpha) = f_{\mathfrak{T}}^{q^i}(\beta) f_{\mathfrak{T}}^{q^s}(a)$. По индуктивному предположению, все хорошо работает для слов длины l , таких как β , т.е. $f_{\mathfrak{A}}^{q_{\eta(i)}}(\beta) = f_{\mathfrak{T}}^{q^i}(\beta)$. Осталось доказать то же самое для a . Попробуем преобразовать $f_{\mathfrak{A}}^{\varphi(\beta, q_{\eta(i)})}(a)$. То, что получится после переработки одного символа — просто один выходной символ, т.е. $f_{\mathfrak{A}}^{\varphi(\beta, q_{\eta(i)})}(a) = \psi(a, \varphi(\beta, q_{\eta(i)}))$. Как мы знаем из второй части индуктивного предположения, $\varphi(\beta, q_{\eta(i)})$ и $q_{\eta(s)}$ — неотличимы. Тогда $\psi(a, \varphi(\beta, q_{\eta(i)})) = \psi(a, q_{\eta(s)})$. По определению Ψ , это и есть $\Psi(a, q^s)$, т.е. $f_{\mathfrak{T}}^{q^s}(a)$. Итак, $f_{\mathfrak{A}}^{\varphi(\beta, q_{\eta(i)})}(a) = f_{\mathfrak{T}}^{q^s}(a)$. Значит $f_{\mathfrak{A}}^{q_{\eta(i)}}(\alpha) = f_{\mathfrak{T}}^{q^i}(\alpha)$, первая часть доказана.

Вторая часть, нам надо доказать, что $\varphi(\alpha, q_{\eta(i)})$ и $q_{\eta(r)}$, где $q^r = \Phi(\alpha, q^i)$ — неотличимые состояния (см. рисунок 6.5). Из индуктивного предположения мы знаем, что $\varphi(\beta, q_{\eta(i)})$ и $q_{\eta(s)}$ — неотличимы, т.е. находятся в одном классе Q_s . Отсюда, состояния, куда переходит автомат \mathfrak{A} из состояний $\varphi(\beta, q_{\eta(i)})$ и $q_{\eta(s)}$ после переработки символа a — тоже неотличимы. Это состояния $\varphi(\alpha, q_{\eta(i)})$ и $\varphi(a, q_{\eta(s)})$, соответственно. Если $\Phi(\alpha, q^i) = \Phi(a, q^s) = q^r$, то по определению Φ получим, что $\xi(\varphi(a, q_{\eta(s)})) = r$, т.е. $\varphi(a, q_{\eta(s)}) \in Q_r$. Раз $\varphi(\alpha, q_{\eta(i)})$ и $\varphi(a, q_{\eta(s)})$ неотличимы, то они в одном классе неотличимости, т.е. в Q_r . Там же находится и $q_{\eta(r)}$. Значит, $\varphi(\alpha, q_{\eta(i)})$ и $q_{\eta(r)}$ — неотличимы, что мы и хотели доказать. Индуктивное

доказательство закончено.

- (b) Нужно доказать эквивалентность автоматов \mathfrak{A} и \mathfrak{T} .

Очевидно, что для каждого состояния q^i из \mathfrak{T} есть состояние из \mathfrak{A} , которое вычисляет ту же функцию, это $q_{\eta(i)}$, мы это только что доказали. Значит нужно доказать только, что для каждого состояния $q_i \in Q$ из \mathfrak{A} есть некоторое $q^j \in Q'$ из \mathfrak{T} . Наше q_i лежит в некотором классе, пусть это будет Q_j . Значит из всех состояний в этом классе вычисляется все та же функция, в том числе из состояния $q_{\eta(j)}$. Т.е. $f_{\mathfrak{A}}^{q_i} \equiv f_{\mathfrak{A}}^{q_{\eta(j)}}$. Но мы уже доказали, что из состояния $q_{\eta(j)}$ в \mathfrak{A} вычисляется та же функция, что из q^j в \mathfrak{T} . Значит $f_{\mathfrak{A}}^{q_i} \equiv f_{\mathfrak{T}}^{q^j}$, что мы и хотели доказать.

- (c) Докажем то, что \mathfrak{T} — минимальный автомат.

Рассмотрим 2 разных состояния \mathfrak{T} , $q^i, q^j \in Q', i \neq j$. Это значит, что функции, которые они вычисляют, будут совпадать с функциями $f_{\mathfrak{A}}^{q_{\eta(i)}}$ и $f_{\mathfrak{A}}^{q_{\eta(j)}}$, соответственно. Но так как $q_{\eta(i)}$ и $q_{\eta(j)}$ лежат в разных классах неотличимости (Q_i и Q_j), то они будут отличимы, и их функции будут отличаться: $f_{\mathfrak{A}}^{q_{\eta(i)}} \neq f_{\mathfrak{A}}^{q_{\eta(j)}}$. Значит $f_{\mathfrak{T}}^{q^i} \neq f_{\mathfrak{T}}^{q^j}$, т.е. они будут отличимы. Значит в \mathfrak{T} нет неотличимых состояний, значит он — минимальный.

□

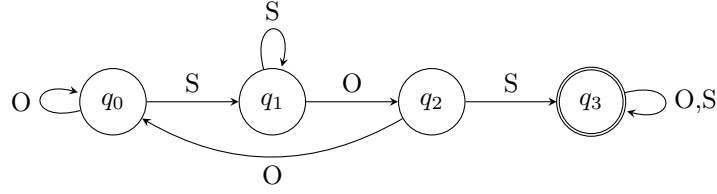
6.6 Распознавание слов конечным автоматом

Ранее мы обращали внимание только на выходное слово автомата и игнорировали то состояние, в котором он оказался после окончания работы. Сейчас мы сделаем ровно наоборот, будем игнорировать выходное значение и обратим особое внимание на состояние, в которое он окажется. Для этого определим некоторое множество состояний $D \subseteq Q$, которое назовем *множеством распознающих состояний*. Будем говорить, что автомат распознал слово α из начального состояния q_0 для множества распознающих состояний D , если он после переработки этого слова оказался в одном из состояний D , т.е. $\varphi(\alpha, q_0) \in D$. На самом деле нам не так важно распознавание отдельных слов, поэтому составим множество $U \subseteq A^*$ всех слов, которое распознается данным автоматом \mathfrak{A} из q_0 при помощи D . Это множество называется автоматным языком.

Пример. Пусть входной алфавит $A = \{S, O\}$. Составим автомат, распознающий слова вида $\alpha' SOS \alpha''$, где $\alpha', \alpha'' \in A^*$, т.е. слова, внутри которых содержится подстрока SOS (см. рисунок).

У любого автомата должно быть начальное состояние, q_0 , так что добавим его⁴. Если на вход приходит S , то автомат должен «насторожиться» и перейти в другое состояние, q_1 , возможно это первая буква нашего слова

⁴ «Уже хорошо, пока ошибок не сделали» — К.И.Костенко


 Рис. 6.6: Автомат, распознающий слово SOS

SOS . Если же приходит O , то это точно не начало, переходить в другое состояние не нужно, остаемся в q_0 . И да, так как нам не важен выход автомата, мы его даже не пишем, просто S или O .

Итак, состояние q_1 . Если мы встретили O , то возможно это следующая буква в SOS , перешли в следующее состояние, q_2 , где мы ждем последнюю S . Если же нам пришло S , то та первая буква точно уже не была началом SOS , но эта новая еще может ей оказаться (например, нам попалось слово $SSOS$), так что мы остаемся в q_1 .

Состояние q_2 , если нам пришло S , то мы нашли наше SOS , переходим в новое состояние, которое будет распознающим (обозначается двойным кружком), q_3 . Если же мы встретили O , то мы прочитали SOO , что точно не может быть началом SOS , так что откатываемся в самое начало, в q_0 . Последнее же состояние, q_3 , является распознающим, и если мы уже нашли SOS , то мы должны в этом состоянии и оставаться, независимо от того, что приходит на вход.

Еще одно, так как нам не нужен выход автомата, то мы можем сделать новое определение для автомата, распознающего слова, а именно $\mathfrak{A} = (A, Q, \varphi, q_0, D)$. Это будет уже не то, что мы изначально называли конечным автоматом, но это новое определение нам будет полезнее в данном контексте. Далее мы докажем теорему, которая позволит комбинировать автоматные языки:

Теорема. Пусть $U_1, U_2 \subseteq A^*$ — множества слов, которые распознают автоматы $\mathfrak{A}_1 = (A, Q_1, \varphi_1, q_0^1, D_1)$ и $\mathfrak{A}_2 = (A, Q_2, \varphi_2, q_0^2, D_2)$. Тогда мы можем составить автомат $\mathfrak{A}_3 = (A, Q_3, \varphi_3, q_0^3, D)$, который будет распознавать слова $U_1 \cup U_2$.

Доказательство. Определим автомат, который работает как два «параллельно» работающих автомата \mathfrak{A}_1 и \mathfrak{A}_2 . Нет, это не просто объединение всех состояний, тут несколько сложнее. Состояниями нового автомата являются пары, состоящие из состояния первого автомата, и состояния второго автомата, т.е. $Q_3 = Q_1 \times Q_2$. Для удобства определим $q_1(t)$ и $q_2(t)$ так, что $q(t) = (q_1(t), q_2(t))$. Начальным состоянием будет $q(t_0) = (q_0^1, q_0^2)$. Т.е. нашим автоматом будет $\mathfrak{A}_3 = (A, Q_1 \times Q_2, \varphi_3, (q_0^1, q_0^2), D)$. Осталось определить φ_3 и D . Раз наши автоматы работают параллельно, то φ должно просто па-

параллельно и независимо менять состояния $q_1(t)$ и $q_2(t)$, т.е.

$$\varphi_3(x(t), (q_1(t), q_2(t))) = (\varphi_1(x(t), q_1(t)), \varphi_2(x(t), q_2(t)))$$

Осталось только D . Нам нужно, чтобы слово было распознано тогда и только тогда, когда или первый автомат распознал слово ($q_1(t) \in D_1$), или второй распознал слово ($q_2(t) \in D_2$). Тогда вполне ясно, что множеством распознающих состояний должно быть $D = D_1 \times Q_2 \cup Q_1 \times D_2$.

Докажем, что $U = U_1 \cup U_2$. Для начала докажем, что $U \subseteq U_1 \cup U_2$. Слово α лежит в U тогда, когда наш автомат его распознал, т.е. $\varphi_3(\alpha, (q_0^1, q_0^2)) \in D$. По определению φ_3 и D это значит, что $(\varphi_1(\alpha, q_0^1), \varphi_2(\alpha, q_0^2)) \in D_1 \times Q_2 \cup Q_1 \times D_2$. Это значит, что или $(\varphi_1(\alpha, q_0^1), \varphi_2(\alpha, q_0^2)) \in D_1 \times Q_2$, или $(\varphi_1(\alpha, q_0^1), \varphi_2(\alpha, q_0^2)) \in Q_1 \times D_2$. В первом случае $\varphi_2(\alpha, q_0^2) \in Q_2$ выполняется всегда, т.е. важно только $\varphi_1(\alpha, q_0^1) \in D_1$, во втором случае важно только $\varphi_2(\alpha, q_0^2) \in D_2$. Т.е. или $\alpha \in U_1$, или $\alpha \in U_2$, т.е. $\alpha \in U_1 \cup U_2$, что мы и хотели доказать. Доказательство другой половины полностью аналогично, просто в обратную сторону. \square

Стоит заметить, что можно сформулировать аналогичные теоремы для $U_1 \cap U_2$, там множеством распознающих состояний будет $D = D_1 \times D_2$, и для $U_1 \setminus U_2$, там $D = D_1 \times (Q_2 \setminus D_2)$.

6.7 Операции над конечными автоматами

Так же, как и другие объекты дискретной математики, конечные автоматы можно комбинировать. Для них определены 2 операции: суперпозиция и операция обратной связи:

1. Суперпозиция

Суперпозиция — просто подаем выход одного автомата на вход другого, как суперпозиция функций. Если нам даны автоматы $\mathfrak{A}_1 = (A, B, Q_1, \varphi_1, \psi_1)$ и $\mathfrak{A}_2 = (B, C, Q_2, \varphi_2, \psi_2)$, то их композицией будет $\mathfrak{A}_3 = (A, C, Q_1 \times Q_2, \varphi_3, \psi_3)$. $Q_3 = Q_1 \times Q_2$, потому что состоянием нашего автомата будут являться пары состояний двух изначальных автоматов. Как и раньше, $q(t) = (q_1(t), q_2(t))$ и $q(t_0) = (q_0^1, q_0^2)$. Первому автомату на вход подается обычное $x(t)$, а второму — выход первого автомата, $\psi_1(x(t), q_1(t))$. Тогда

$$\begin{aligned} \varphi_3(x(t), (q_1(t), q_2(t))) &= (\varphi_1(x(t), q_1(t)), \varphi_2(\psi_1(x(t), q_1(t)), q_2(t))) \\ \psi_3(x(t), (q_1(t), q_2(t))) &= \psi_2(\psi_1(x(t), q_1(t)), q_2(t)) \end{aligned}$$

Чуть внятнее выглядит запись этого автомата в виде канонических

уравнений, разбитым на части:

$$\begin{aligned} q_1(t_0) &= q_0^1 \\ q_2(t_0) &= q_0^2 \\ q_1(t+1) &= \varphi_1(x(t), q_1(t)) \\ q_2(t+1) &= \varphi_2(\psi_1(x(t), q_1(t)), q_2(t)) \\ y(t) &= \psi_2(\psi_1(x(t), q_1(t)), q_2(t)) \end{aligned}$$

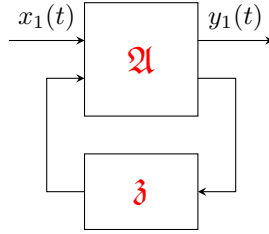
2. Операция обратной связи

Определим некий очень простой автомат, называемый автоматом задержки: $\mathfrak{z} = (A, A, A, \varphi_{\mathfrak{z}}, \psi_{\mathfrak{z}})$, где A — алфавит. Этот автомат, неожиданно, осуществляет задержку входа на один «такт». $a_0 \in A$ — начальное состояние нашего автомата. Он задается достаточно простыми каноническими уравнениями:

$$\begin{aligned} q(t_0) &= a_0 \\ q(t+1) &= x(t) \\ y(t) &= q(t) \end{aligned}$$

т.е. автомат просто запоминает то, что ему дают на вход, и выдает это на следующем «такте».

Пусть нам задан автомат $\mathfrak{A} = (A^2, A^2, Q, \varphi, \psi)$, т.е. на вход ему подается 2 символа алфавита A , и он выдает сразу 2 символа того же алфавита. Пусть $\psi = (\psi_1, \psi_2)$, т.е. функции ψ_1 и ψ_2 по отдельности отвечают за каждый из выходов автомата. Тогда автомат с обратной связью выглядит так:



Это автомат $\mathfrak{A}' = (A, A, Q', \varphi', \psi')$, где $Q' = Q \times A$ (пары состояний \mathfrak{A} и \mathfrak{z}). Мы можем аналогично расписать $q'(t) = (q_1(t), q_2(t))$ и $q'(t_0) = (q_0, a_0)$. Как всегда, лучше определить этот автомат в виде канонических уравнений:

$$\begin{aligned} q_1(t_0) &= q_0 \\ q_2(t_0) &= a_0 \\ q_1(t+1) &= \varphi((x(t), q_2(t)) q_1(t)) \\ q_2(t+1) &= \psi_2((x(t), q_2(t)) q_1(t)) \\ y(t) &= \psi_1((x(t), q_2(t)) q_1(t)) \end{aligned}$$

6.8 Структурный автомат

Помните схемы из функциональных элементов? Для автоматов существует ровно то же самое, для построения такого автомата используются все те же 3 функциональных элемента, $\&$, \vee , \bar{x} , но вместе с ними еще и автомат задержки \mathfrak{z} . И да, мы так можем работать только с двоичной системой, $A = \{0, 1\}$. Правила построения таких схем такие же, как и для схем из функциональных элементов, кроме одного — циклы разрешены, но в них должен быть хотя бы один элемент задержки.

Теперь к вопросу о том, как построить некоторый заданный автомат $\mathfrak{A} = (A, B, Q, \varphi, \psi)$. Для начала, учтем, что наш структурный автомат должен работать исключительно в двоичной системе, т.е. изначальный автомат мы никак не сможем построить точно. Вместо этого мы сможем построить очень похожий на него автомат \mathfrak{A}' , работающий с двоичными кодами символов и состояний. Собственно этим и займемся. Закодируем входной символ $x(t) \in A$ как $x_1(t), \dots, x_p(t)$, выходной символ $y(t) \in B$ как $y_1(t), \dots, y_d(t)$, а состояние $q(t) \in Q$ как $q_1(t), \dots, q_s(t)$, причем $x_i(t), y_i(t), q_i(t) \in \{0, 1\}$. И еще один важный момент, пусть q_0 у нас кодируется как $(0, \dots, 0)$. Традиционные канонические уравнения автомата будут такие:

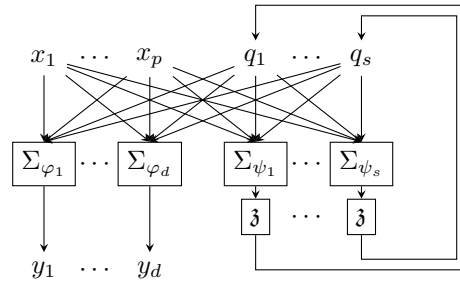
$$\begin{aligned} q(t_0) &= q_0 \\ q(t+1) &= \varphi(x(t), q(t)) \\ y(t) &= \psi(x(t), q(t)) \end{aligned}$$

Мы же при помощи наших кодов преобразуем их в

$$\begin{aligned} q_1(t_0) &= 0 \\ &\vdots \\ q_s(t_0) &= 0 \\ \\ q_1(t+1) &= \varphi_1(x_1(t), \dots, x_p(t), q_1(t), \dots, q_s(t)) \\ &\vdots \\ q_s(t+1) &= \varphi_s(x_1(t), \dots, x_p(t), q_1(t), \dots, q_s(t)) \\ \\ y_1(t) &= \psi_1(x_1(t), \dots, x_p(t), q_1(t), \dots, q_s(t)) \\ &\vdots \\ y_d(t) &= \psi_d(x_1(t), \dots, x_p(t), q_1(t), \dots, q_s(t)) \end{aligned}$$

Типа вместо $x(t)$ мы подаем в функции его двоичный код, и так же с $q(t)$. Кроме того, так как мы теперь получаем тоже не просто состояние, а двоичный код, а так же не просто выходной символ, а его код, то наши функции φ и ψ разбиваются на несколько функций, вычисляющих отдельные биты получающегося кода. При этом $\varphi_1, \dots, \varphi_s, \psi_1, \dots, \psi_d \in P_2^{p+s}$ —

функции алгебры логики. Это значит, что для них мы можем построить схемы из функциональных элементов $\Sigma_{\varphi_1}, \dots, \Sigma_{\varphi_s}, \Sigma_{\psi_1}, \dots, \Sigma_{\psi_d}$. Тогда схемой этого автомата будет:



Глава 7

Рекурсивные функции

7.1 Классы рекурсивных функций

Мы уже видели одну модель вычислений — конечные автоматы. Это была строго «императивная» модель, похожая на императивные языки программирования — автомат в каждый конкретный момент времени находится в каком-то состоянии и переходит в другие по некоторым правилам. Более того, конечные автоматы могут вычислять далеко не все, то же умножение, например, не могут. Для вычисления всех возможных вычислимых функций требуется уже что-то мощнее, например, машина Тьюринга. К ней мы обращаться не будем, а лучше посмотрим на проблему с другой стороны — функциональной.

Существует другая парадигма программирования — функциональное программирование. Программа на таком языке является просто кучей различных функций, похожих на математические, которые все определены друг через друга. Чтобы выполнить такую программу мы просто записываем некоторое изначальное выражение, а потом постепенно раскрываем его и упрощаем, пока не придем к готовому ответу. В такой модели нет «состояния», она полностью состоит из математических преобразований. Так вот, существует функциональная модель, эквивалентная всем известной машине Тьюринга, а именно — рекурсивные функции.

Самым важным для нас классом таких функций являются примитивно рекурсивные функции. Они не особо полезны, но зато помогают определить, какие функции в целом вычислимы, а какие нет. Так как примитивно рекурсивные функции — подкласс всех вычислимых, если мы сможем записать то, что мы хотим, как ПРФ, то это значит, что она вычислима, т.е. что для нее существует алгоритм на машине Тьюринга (и на любом из видов современных компьютеров). Сразу стоит заметить, что далеко не все вычисляемые функции — примитивно рекурсивны, существуют функции, которые вычислимы, но они не ПРФ, такие как функция Аккермана. Правда этот класс все равно намного шире, чем функции, вычисляемые автоматами, все

стандартные математические функции вполне вычислимы на основе ПРФ.

Для начала, определимся, какие именно функции мы изучаем? А вот такие: $f : \mathbb{N}^k \rightarrow \mathbb{N}$, где $\mathbb{N} = \{0, 1, 2, \dots\}$. Мы не будем усложнять все отрицательными и дробными числами, остановимся на натуральных и нуле.

Чтобы подобраться к ПРФ, стоит сначала пройти несколько более простых классов функций, которые мы все назовем *интуитивно вычислимыми*:

1. *Примитивные (простейшие) функции*

Базовые строительные блоки нашего определения, то, что мы умеем их считать, задается как аксиома:

- (a) Нулевая функция, $O(x) = 0$. Просто выдает 0 на всех значениях x .
- (b) Функция следования, $S(x) = x + 1$. Просто прибавляет 1.
- (c) Функции проекции, I_n^m . Функция I_n^m принимает n аргументов и выдает аргумент номер m . Например, функция $I_4^2(x, y, z, t) = y$.

2. *Элементарные функции*

Функции, которые можно получить из примитивных при помощи операции суперпозиции (конечного числа их, разумеется), которая определяется следующим образом:

Определение. Пусть у нас имеются функции $f(x_1, \dots, x_n)$ и $g_1(x_{11}, \dots, x_{1m_1}), g_2(x_{21}, \dots, x_{2m_2}), \dots, g_n(x_{n1}, \dots, x_{nm_n})$. Тогда функция $h(x_1, \dots, x_k)$ называется их *суперпозицией*, если

$$h(x_1, \dots, x_k) = f(g_1(x_{11}, \dots, x_{1m_1}), \dots, g_n(x_{n1}, \dots, x_{nm_n}))$$

Т.е. самая обычная суперпозиция функций, подаем выходы n функций как аргументы функции f . Правда в этом определении мы должны заменять *каждую* переменную функции f . Впрочем, если мы этого не хотим, мы можем просто выбрать $g_i = I_{m_i}^i$, т.е. использовать функцию проекции, чтобы взять нужную переменную из набора.

Очевидно, что эти функции тоже являются интуитивно вычислимыми, ведь если мы можем вычислить все составляющие функции, то мы можем вычислить и их суперпозицию, верно? Так какие функции входят в этот класс? Много раз применяя функцию следования мы получим функцию прибавления константы, например $S_3(x) = S(S(S(x)))$ — функция $x + 3$. Так же можно получить константные функции, например, $C_3(x) = S(S(S(O(x))))$ — функция, которая всегда выдает 3. Также можно комбинировать это различными способами с функциями проекции.

3. *Примитивно рекурсивные функции*

Наконец то мы дошли до них, да? Эти функции получаются из примитивных (или элементарных) при помощи конечного числа операций суперпозиции и *операций примитивной рекурсии*. Определим такую:

Определение. Пусть у нас имеется функции $g(x_1, \dots, x_n)$ и $h(x_1, \dots, x_n, x_{n+1}, x_{n+2})$, причем $n \geq 0$.¹ Тогда функция $f(x_1, \dots, x_n, x_{n+1})$ является результатом применения операции примитивной рекурсии к g и h (давайте обозначать это как $f = P \langle g, h \rangle$, хотя Константин Иванович и не использует никакого обозначения для этого), если выполняются следующие соотношения:

$$\begin{aligned} f(x_1, \dots, x_n, 0) &= g(x_1, \dots, x_n) \\ f(x_1, \dots, x_n, y+1) &= h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)) \end{aligned}$$

Переменные x_1, \dots, x_n — параметры функции, а x_{n+1} — параметр рекурсии. Функция g осуществляет «базис» рекурсии, изначальное значение f на нуле. Когда же мы хотим вычислить f на некотором $y+1$, мы вычисляем значение f на y (рекурсия!) и формируем итоговое значение при помощи h . Таким образом функция h является «шагом» рекурсии, вычисляя значение f при помощи значения этой же f на меньшем аргументе. И да, мы могли бы записать второе правило и как $f(x_1, \dots, x_n, y') = h(x_1, \dots, x_n, y' - 1, f(x_1, \dots, x_n, y' - 1))$, где $y' > 0$, это полностью эквивалентно, просто принято другое определение.

Докажем, что такие функции тоже интуитивно вычислимы. Для этого приведем целых 2 схемы вычисления функций, построенных при помощи операции примитивной рекурсии: одну, основанную на собственно рекурсии, и одну, основанную на циклах.

Пусть нам надо вычислить $f(x_1, \dots, x_n, 10)$. Так как $10 \neq 0$, нам подходит второе соотношение, т.е.

$$f(x_1, \dots, x_n, 10) = h(x_1, \dots, x_n, 9, f(x_1, \dots, x_n, 9))$$

Может так случиться, что функция h может вычислить значение без использования последнего аргумента. Если так и есть, то все прекрасно, это наш ответ. Если же в ней он все таки используется, то мы прекращаем выполнение этой версии h и начинаем считать $f(\dots, 9)$. Для этого нам нужно $h(\dots, 8, f(\dots, 8))$. Если можем вычислить это значение просто так, вычисляем, если нет — считаем $f(\dots, 8)$. Так мы идем или пока h не сможет вычислить значение, не обращаясь снова к рекурсии, или пока не дойдем до $f(\dots, 0)$, которое мы можем вычислить при помощи g . Этот процесс называется прямым ходом рекурсии. В момент же обратного хода рекурсии мы подставляем полученные значения обратно в функции h и постепенно вычисляем итоговое значение.

¹Если $n = 0$, то g — константа, а $h(x_1, x_2)$ — функция двух аргументов.

Вторая схема расчета — мы сразу начинаем с $f(\dots, 0)$, которое вычисляем при помощи g . Потом мы считаем $f(\dots, 1)$ при помощи h , используя уже найденное $f(\dots, 0)$. Далее считаем $f(\dots, 2)$, и так пока не дойдем до $f(\dots, 10)$, или до куда нам там надо.

Значит, если g и h — интуитивно вычислимы, то и f тоже. Правда, вторая схема не учитывает то, что функция h может быть не определена на некоторых значениях y и все еще работать на других — например, если h — константа на $y = 9$, и не определена на нигде больше, то все равно существует $f(x_1, \dots, x_n, 10)$, хотя все предыдущие — нет. Рекурсивная схема это учтет, а циклическая — нет.

Пример. Попробуем определить $f(x, y) = x + y$ как примитивно рекурсивную. Используем y как параметр рекурсии. Тогда

$$\begin{aligned} f(x, 0) &= I_1^1(x) \\ f(x, y + 1) &= x + (y + 1) = (x + y) + 1 = S(f(x, y)) \end{aligned}$$

Нам нужно каким-то образом привести это к более формальному виду, а именно найти такие функции $g(x)$ и $h(x, y, z)$, что $f = P \langle g, h \rangle$. Функция $g(x)$ должна просто давать значение f при $y = 0$. Это просто $g(x) = I_1^1(x)$. Что насчет $h(x, y, z)$? Вспомним, что вообще значат эти аргументы. x и y — те же x и y , что и в $f(x, y + 1)$, тут ничего нового. А вот z у нас слегка по-другому, по определению примитивной рекурсии как последний аргумент h подается $f(x, y)$, т.е. $z = f(x, y)$ и мы можем это использовать. Тогда $h(x, y, z) = S(f(x, y)) = S(z)$. Только вот все не так просто, если у нас подаются 3 переменные, мы должны их все подавать и во внутренние функции, т.е. мы должны получить z как $I_3^3(x, y, z)$. Не такая большая проблема. Итак:

$$\begin{aligned} g(x) &= I_1^1(x) \\ h(x, y, z) &= S(I_3^3(x, y, z)) \end{aligned}$$

Пример. Можно так же определить умножение: $\varphi(x, y) = x \cdot y$

$$\begin{aligned} \varphi(x, 0) &= O(x) \\ \varphi(x, y + 1) &= x \cdot (y + 1) = x + (x \cdot y) = f(x, \varphi(x, y)) \end{aligned}$$

По аналогичным рассуждениям, мы получим умножение, применив операцию примитивной рекурсии к

$$\begin{aligned} g(x) &= O(x) \\ h(x, y, z) &= f(I_3^1(x, y, z), I_3^3(x, y, z)) \end{aligned}$$

Пример. И степень тоже: $p(x, y) = x^y$

$$\begin{aligned} p(x, 0) &= S(O(x)) \\ p(x, y + 1) &= x^{y+1} = x \cdot x^y = \varphi(x, p(x, y)) \end{aligned}$$

То есть мы получим степень, применив операцию примитивной рекурсии к

$$\begin{aligned} g(x) &= S(O(x)) \\ h(x, y, z) &= \varphi(I_3^1(x, y, z), I_3^3(x, y, z)) \end{aligned}$$

К сожалению, существуют вычислимые функции, которые не примитивно рекурсивные. Для того, чтобы их учесть, введем еще один класс:

4. Рекурсивные функции²

Для их определения введем еще один оператор — оператор минимизации.

Определение. Пусть дана $g(x_1, \dots, x_n, x_{n+1})$, где x_{n+1} — переменная минимизации. Тогда функция $f(x_1, \dots, x_n, x_{n+1})$ получится из g , если

$$f(x_1, \dots, x_n, x_{n+1}) = \mu t (g(x_1, \dots, x_n, t) = x_{n+1})$$

Эта запись означает, что t — это наименьшее число, при котором выполняется условие $g(x_1, \dots, x_n, t) = x_{n+1}$. При этом все значения от $g(\dots, 0)$ до $g(\dots, t)$ должны существовать (далее — не обязательно).

Вычисляются такие функции просто последовательной проверкой всех возможных t , пока не встретим ту, на которой условие выполняется. И да, этот оператор позволяет создавать не завершающиеся функции, никогда не выдающие ответа, именно из-за них и были оговорки во всех предыдущих операциях. Например, если мы применим эту операцию к $O(x)$, то $f(0)$ будет равно 0, а на всех остальных значениях переменной функция не завершится, так как $O(x)$ всегда выдает 0 и другие значения оператор минимизации найти там никогда не сможет.

Пример. Посмотрим, что будет, если мы применим эту операцию к $g(x, y) = x + y$. Тогда $f(x, y) = \mu t (x + t = y)$. Да, y в g и y в f — разные переменные. Мы перебираем все возможные значения второй переменной g , до тех пор пока значение g не будет равно второму аргументу f , т.е. y . Для какого t выполняется $x + t = y$? Верно, для $t = y - x$. Причем это единственное значение, на котором это верно, оно и будет выдаваться функцией. Но что будет, если $y < x$? Тогда $x + t = y$ не может быть верно при $t \geq 0$, т.е. эта функция не имеет значения на этих аргументах. Итого:

$$f(x, y) = \begin{cases} y - x & , y \geq x \\ - & , \text{ иначе} \end{cases}$$

²Так то они называются частично-рекурсивными, но по каким-то причинам Константин Иванович не хочет так говорить.

Пример. Пусть у нас есть некоторая $g(x)$, т.е. $g : \mathbb{N} \rightarrow \mathbb{N}$, которая является биекцией (не забыли, что это?). Тогда $f(x) = \mu t(g(t) = x)$, и т.к. это биекция, то $f(x) = g^{-1}(x)$. Т.е. для биективных функций одного аргумента операция минимизации эквивалентна поиску обратной функции.

Ну и да, рекурсивные функции — это те, которые получаются из примитивных после конечного числа суперпозиций, операций примитивной рекурсии и операций минимизации.

7.2 Тезис Чёрча

Мы так и не определили формально, что такое интуитивно-вычислимая функция. Это нечеткое понятие и в этом проблема. Алонзо Чёрч высказал гипотезу, что класс интуитивно-вычислимых функций *совпадает* с классом рекурсивных. Не просто что все рекурсивные функции интуитивно вычислимы, но и что абсолютно все интуитивно вычислимые функции возможно определить как рекурсивные. Эту гипотезу невозможно доказать из-за нечеткости определения интуитивно вычислимых функций, но судя по всему она верна, потому что ученым так и не удалось найти ни одной функции, которую можно называть вычислимой, но которая не попадает под определение рекурсивной. Более того, когда были придуманы другие обобщения алгоритмов при помощи других формальных моделей, таких как машина Тьюринга, система Маркова, tag-система, лямбда исчисление и кучи других, классы вычисляемых ими функций все совпадали с классом рекурсивных, и это все уже вполне доказано. Так что, судя по всему, это — наилучший способ определить вычислимые функции, так что им и воспользуемся.

7.3 Деревья определения рекурсивных функций

Было бы неплохо как-то представлять то, как именно мы определили некую рекурсивную функцию, кроме как словами описывать последовательность всех операций. Оказывается, для этого лучше всего подходят деревья. Чтобы как-то обозначать операции, будем помечать каждую вершину дерева символами из алфавита $\{O, S, I, C, P, M, 1\}$. O, S, I — это наши примитивные функции, C — композиция, P — примитивная рекурсия, M — минимизация, а 1 нам потребуется для обозначения индексов переменных. Нормальные люди пишут это обычными цифрами, но мы — математики, так что будем использовать для этого унарную систему счисления, т.е. просто писать столько единиц, какой индекс. Для краткости будем писать впрочем вещи типа $\langle 4 \rangle$, когда надо было написать 1111. И да, позволим себе писать несколько единиц в одной вершине, т.е. не обязательно по одному символу

на вершину, единиц может быть много. В общем, увидите. Итак, приступим к описанию деревьев.

- Функция $O(x_i)$. Нам нужно указать, что это O , и что в ней используется переменная номер i . Сделаем это вот так:

$$\begin{array}{c} O \\ | \\ \langle i \rangle \end{array}$$

- Функция $S(x_i)$. Абсолютно аналогично:

$$\begin{array}{c} S \\ | \\ \langle i \rangle \end{array}$$

- Функции $I_n^m(x_{s_1}, x_{s_2}, \dots, x_{s_n})$. Тут чуть сложнее, у нас может быть разное число аргументов, поэтому в дереве тоже будет разное число потомков. Самый левый потомок пусть будет $\langle n \rangle$, второй слева — $\langle m \rangle$, а остальные n потомков — номера переменных. Итак:

$$\begin{array}{c} I \\ \swarrow \quad \downarrow \quad \searrow \\ \langle n \rangle \quad \langle m \rangle \quad \langle s_1 \rangle \quad \cdots \quad \langle s_n \rangle \end{array}$$

- Композиция функций. Во первых, композиции на вход подается функция f с n переменными и n функций g_1, \dots, g_n . Нам нужно, чтобы все они были представимы в виде таких деревьев, назовем эти деревья $D_f, D_{g_1}, \dots, D_{g_n}$. Так как мы можем определить число переменных по D_f , нам не обязательно его указывать. Нам в принципе ничего такого не обязательно указывать, мы можем просто составить дерево:

$$\begin{array}{c} C \\ \swarrow \quad \downarrow \quad \searrow \\ D_f \quad D_{g_1} \quad \cdots \quad D_{g_n} \end{array}$$

- Прimitивная рекурсия. Тут все еще проще, у нас есть функции g и h и мы можем сделать деревья D_g и D_h . И будет тогда дерево

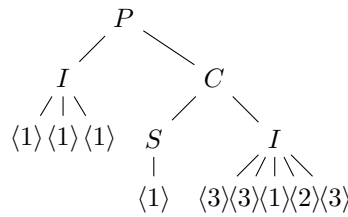
$$\begin{array}{c} P \\ / \quad \backslash \\ D_g \quad D_h \end{array}$$

- Минимизация функций. Еще проще, просто прицепляем D_g к букве M :

$$\begin{array}{c} M \\ | \\ D_g \end{array}$$

И на этом и все. На так уж и сложно, правда? Давайте рассмотрим один пример

Пример. Помните сложение? $f(x_1, x_2) = x_1 + x_2$. Оно получалось при помощи примитивной рекурсии из функций $g(x_1) = I_1^1(x_1)$ и $h(x_1, x_2, x_3) = S(I_3^3(x_1, x_2, x_3))$. Заметим, что h — это композиция функций $S(x_1)$ и $I_3^3(x_1, x_2, x_3)$. Чтобы определить первую функцию $I_1^1(x_1)$, нам нужно написать I с $n + 2 = 3$ потомками. Этими потомками будут $\langle 1 \rangle$ (это n), $\langle 1 \rangle$ (это m) и $\langle 1 \rangle$ (обозначает x_1). Чтобы записать $I_3^3(x_1, x_2, x_3)$ нам потребуется уже 5 потомков, а именно: $\langle 3 \rangle$ (это n), $\langle 3 \rangle$ (это m) и $\langle 1 \rangle$, $\langle 2 \rangle$, $\langle 3 \rangle$ (это x_1, x_2, x_3). Итого, дерево выглядит так:



Деревья это конечно хорошо, но было бы неплохо записать их как-то в более удобном виде. Как насчет слов? Алфавитом для наших слов будет все те же самые $O, S, I, C, P, M, 1$, но мы добавим еще $\$,$ разделитель. Чтобы превратить наше дерево в его код (слово), мы просто начнем с корня и будем обходить его в глубину (идя слева-направо). После каждой вершины мы будем ставить $\$$. Например, для нашего сложения кодом будет

$$P\$I\$1\$1\$1\$C\$S\$1\$I\$111\$111\$1\$11\$111\$$$

Мы сначала пишем P , корень дерева, потом идем в самого левого потомка, пишем I , самый левый потомок — 1 , у 1 нет потомков, поэтому возвращаемся выше. У I осталось еще 2 потомка — две 1 , мы пишем их и возвращаемся обратно в P . После этого мы идем в следующего потомка P , т.е. C , оттуда в S и 1 , а вернувшись в C идем в I и потом по всем 5 его потомкам слева направо. Достаточно логичный порядок обхода для дерева.

Есть только еще одна деталь. А можем ли мы по этому коду восстановить исходное дерево? Оказывается, можем. Количество потомков заранее неизвестно только для I и C , а у них его можно определить, разобрав только самое левое поддереву. Во всех остальных случаях мы можем просто давать каждой вершине столько потомков, сколько ей нужно. Например,

мы знаем, что у S только 1 потомок, и когда мы читаем далее 1, мы знаем, что оно относится к S , а все следующие буквы - к чему-то еще. Даже если у нас будут какие-то сомнения, мы можем просто перебрать все возможные способы разбиения этого на поддеревья и только один из них будет верным! Итак, мы можем каждой функции сопоставить ровно один код и каждому коду ровно одну функцию, т.е. мы имеем биекцию.

Но мы на этом не остановимся! Что может быть еще удобнее слов? Числа!

Давайте составим множество вообще всех возможных слов из символов из алфавита $\{O, S, I, C, P, M, 1, \$\}$, а потом расположим их в определенном порядке. Для начала, они все должны быть по возрастанию длины слова, а если они одинаковой длины, то нужно располагать слова в лексикографическом порядке, т.е.

$$\Lambda, O, S, I, C, P, M, 1, \$, OO, OS, OI, OC, OP, OM, O1, O$, SO, \dots$$

Пронумеруем эту последовательность, $\alpha_0 = \Lambda, \alpha_1 = O, \dots$. Проблема в том, что тут очень много слов, которые не являются легальными кодами деревьев, даже самый первый такой код, $O\$1\$$, является словом α_{1088} , т.е. до него было 1087 слов, не являющимися кодами слов. Давайте просто выкинем эти слова, тогда мы получим новую последовательность β_0, β_1, \dots , состоящую только из тех слов последовательности α , которые на самом деле представляют какую-либо функцию. Т.е. мы просто пропускаем все слова α , пока не найдем первое, которое представляет дерево, называем это слово β_0 , а потом идем дальше. Таким образом мы получим последовательность $\beta_0 = O\$1\$, \beta_1 = S\$1\$, \dots$. И так как каждый код соответствует функции, мы можем по номеру функции i найти код β_i , просто идя по последовательности α (кто говорил, что это будет быстро?), и потом мы можем построить дерево по этому коду, а потом и функцию. И так мы получим последовательность $\mathcal{F} = g_0, g_1, \dots$, содержащую абсолютно все вычислимые функции, что на самом деле очень интересный факт. Это значит, что вычислимых числовых функций всего счетно-бесконечное количество.

А если мы вспомним, что действительных чисел несчетно-бесконечное число, то это будет означать, что далеко не все действительные числа возможно вычислить. Число π например можно, существует куча алгоритмов, которые возвращают двоичное представление этого числа, и вообще все числа, которые были получены и известны людям были вычислимыми, но в это же время вычислимых чисел бесконечно мало по сравнению с невычислимыми. Интересно, правда?

Мы хотим поподробнее изучить функции одного аргумента, так что давайте еще определим последовательность $\mathcal{F}^1 = f_0, f_1, \dots$, где $f_i(x) = g_i(x, x, \dots, x)$, т.е. мы просто отождествляем все переменные, т.е. всегда подставляем туда одно и то же значение. Так как в последовательности α содержатся все возможные сочетания символов, а в последовательности β — все возможные коды, то \mathcal{F} — абсолютно все рекурсивные/вычислимые функции. Но так как существует бесконечно много способов представить

функцию рекурсивно (например просто прибавить к ней 0, это будет другое дерево, но та же функция), то в \mathcal{F} и \mathcal{F}^1 каждая из функций встречается бесконечно много раз.

Нам еще было бы неплохо как-то называть это соответствие. Давайте назовем это *нумерацией* и обозначим ν . Это отображение $\mathcal{F}^1 \rightarrow \mathbb{N}$, причем каждой функции соответствует много номеров. Таких нумераций может быть много, например, одна может преобразовывать математическую функцию в код на C++ (эта нумерация называется программист), а потом код на C++ превращать в двоичный код и записывать как число. Такое тоже разрешено.

Еще мы можем определить так называемую *универсальную функцию* $U(n, x)$, которая способна вычислить любую функцию одного аргумента, т.е. для каждой функции $f_i(x)$ существует некий номер n , что $f_i(x) = U(n, x)$ для всех x . Часто делают так, что $U(n, x) = f_n(x)$ в какой-то нумерации, например нашей нумерации при помощи кодов деревьев. Эта функция является моделью работы компьютера: n — по факту, «программа», а x — входные данные этой «программы».

7.4 Неразрешимые свойства рекурсивных функций

7.4.1 Проблема остановки

Как я уже намекнул в прошлой главе, существуют числа, которые невычислимы, но правда их мало того что нельзя вычислить, их еще и нельзя описать, что не очень хорошо. Правда у нас есть более наглядный пример невычислимых задач, а именно *проблема остановки*.

Для начала опишу ее неформально: пусть у нас есть некая программа (назову ее *тестировщик*), которой на вход поступает *входная программа* и данные для нее, и тестировщик отвечает тем, остановится ли эта программа когда-нибудь или заикнется навсегда. Проблема остановки заключается в том, что *невозможно* создать такой тестировщик, который будет правильно работать и не циклиться для абсолютно всех программ. Для некоторых наборов программ — можно, но для абсолютно всех — нет. В плане, совсем. Даже если мы используем какие-нибудь квантовые компьютеры или еще что-нибудь, невозможно. Да, такую простую задачу невозможно решить никаким способом. Интересно, правда?

Теперь чуть более формально. Нам пригодится понятие *разрешимого множества*³. Это такое множество, для которого *характеристическая функция* вычислима. Что такое характеристическая функция? Все просто, это та функция, которая определяет, лежит ли нечто в нашем множестве. Т.е. пусть у нас есть множество из наборов чисел $A \subseteq \mathbb{N}^k$ (возможно и просто из чисел, не так важно). Тогда характеристической функцией для него

³От того же значения слова «разрешать», что и «разрешение монитора», т.е. «разделять, различать»

будет такая, которая будет давать 1 если набор чисел является элементом A и 0, если не является, т.е.

$$\chi_A(x_1, \dots, x_k) = \begin{cases} 1 & \text{если } (x_1, \dots, x_k) \in A \\ 0 & \text{если } (x_1, \dots, x_k) \notin A \end{cases}$$

Так вот, мы хотим рассмотреть множество всех функций, которые не циклятся:

$$A_1 = \{(n, x) \mid f_n(x) \text{ — определено}\}$$

Теорема. *Множество A_1 неразрешимо, т.е. функция $\chi_{A_1}(n, x)$ не является вычислимой для всех n и x .*

Доказательство. Как всегда от противного, предположим, что она таки вычислима, т.е. ее можно определить как рекурсивную.

Давайте сначала докажем это неформально, а потом запишем это более точно. Если предположить, что таки существует алгоритм, определяющий, завершится ли входная программа, то по идее мы должны прийти к какому-то противоречию. Для этого модифицируем наш тестировщик так, чтобы он вел себя противоположно входной программе: если входная программа успешно завершается, то тестировщик должен циклиться, а если входная программа циклится, то тестировщик что-нибудь выдает, например 0. Пока не потерялись? Хорошо. Тестировщик должен работать на всех программах? Да. Сам тестировщик — программа? Да. Значит он должен работать, если ему подать на вход самого себя. У нас есть 2 варианта — или он при этом заиклится, или нет. Если он заиклится, значит входная программа (сам тестировщик) *не* будет циклиться на тех же входных данных, но он циклится, противоречие. Если он не будет циклиться, то входная программа наоборот заиклится, снова противоречие. Итого это все было одной большой ошибкой и тестировщик невозможен. Теперь формально.

Итак, у нас должна быть функция

$$\chi_{A_1}(n, x) = \begin{cases} 1 & \text{если } f_n(x) \text{ — определено} \\ 0 & \text{если } f_n(x) \text{ — не определено} \end{cases}$$

Она двух аргументов, что не очень удобно, так что давайте определим $g(x) = \chi_{A_1}(x, x)$, т.е. на вход программы будем всегда подавать ее саму же. Ну и очевидно раз мы предположили, что χ_{A_1} вычислима, то и g очевидно тоже. Далее определим нашу «модификацию тестировщика», а именно

$$d(x) = \begin{cases} 0 & g(x) = 0 \\ - & g(x) = 1 \end{cases}$$

Раз эта функция вычислимая, то она должна находиться где-то в нашей \mathcal{F}^1 , т.е. у нее есть какой-то номер, пусть это будет i , т.е. $f_i(x) \equiv d(x)$. А теперь давайте рассмотрим $d(i)$. Если $d(i)$ определено, то $g(i) = 0$ и $f_i(i)$ не определено. Но $f_i(i) = d(i)$, получили противоречие. Если же $d(i)$ не определено, то $g(i) = 1$ и $f_i(i) = d(i)$ определено, снова противоречие. Значит эта функция невычислима и множество неразрешимо. \square

Давайте теперь внимательно рассмотрим доказательство. Если взглянуть в него получше, мы увидим, что оно практически один в один повторяет другое очень знаменитое доказательство — диагональный аргумент Кантора, доказывающий, что действительных чисел несчетное число (в первую очередь имеются в виду числа в промежутке $[0, 1]$). Давайте для начала разберем его: предположим их таки счетное число и мы таки смогли составить их полный список:

- 1) 0.143290...
- 2) 0.746835...
- 3) 0.492349...
- 4) 0.096943...
- ⋮

Давайте теперь рассмотрим цифры на вот такой вот диагонали:

- 1) 0.**1**43290...
- 2) 0.7**4**6835...
- 3) 0.49**2**349...
- 4) 0.096**9**43...
- ⋮

Если мы изменим каждую из этих цифр на что-то еще, например, увеличим на 1 (а все 9-ки уменьшим на 1):

- 1) 0.**2**43290...
- 2) 0.7**5**6835...
- 3) 0.49**3**349...
- 4) 0.096**8**43...
- ⋮

то мы получим число 0.**2538**..., которое отличается от каждого из чисел в списке ровно в одной цифре. Значит самого этого числа нет в списке, и список неполный, значит среди $[0, 1]$ несчетное число действительных чисел.

А теперь аналогия с этим доказательством, составим таблицу функций: строки будут соответствовать разным функциям из \mathcal{F}^1 , а столбцы — разным входным значениям:

	0	1	2	...	i	...
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$...	$f_0(i)$...
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$...	$f_1(i)$...
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$...	$f_2(i)$...
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\ddots
f_i	$f_i(0)$	$f_i(1)$	$f_i(2)$...	$f_i(i)$...
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\ddots

Наша функция $\chi_{A_1}(x, n)$ должна назначать каждому из них единицы или нули в зависимости от того, определено значение в этой таблице или нет. В это же время функция $g(x)$ работает исключительно вот с этими значениями:

	0	1	2	...	i	...
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$...	$f_0(i)$...
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$...	$f_1(i)$...
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$...	$f_2(i)$...
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\ddots
f_i	$f_i(0)$	$f_i(1)$	$f_i(2)$...	$f_i(i)$...
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\ddots

После этого мы определяем функцию $d(x)$, которая *не совпадает* в терминах определенности/неопределенности с каждой из функций f_0, f_1, \dots хотя бы в одном значении, в $f_i(i)$. Тогда этой функции нет в списке, а так у нас там все вычислимые функции, то $d(x)$ невычислима. Но $d(x)$ определена в терминах $g(x)$, которая определена в терминах $\chi_{A_1}(x, n)$, значит $\chi_{A_1}(x, n)$ тоже невычислима. Увидели связь?

7.4.2 Проблема всюду определенности

На этот раз исследуем множество функций, которые определены для всех x :

$$A_2 = \{n \mid \forall x : f_n(x) \text{ — определено}\}$$

Теорема. *Множество A_2 неразрешимо.*

Доказательство. Как всегда, предположим, что разрешима, т.е. что функция

$$\chi_{A_2}(n) = \begin{cases} 1 & \text{если } f_n \text{ — всюду определено} \\ 0 & \text{если } f_n \text{ — не всюду определено} \end{cases}$$

вычислима. Будем снова пользоваться подобием аргумента Кантора, но для этого нам нужно получить «список» всех всюду определенных функций. Легко, пусть $g(n)$ — это n -я всюду определенная функция, т.е. она определяется так (одновременно примитивная рекурсия и оператор минимизации,

да):

$$\begin{aligned} g(0) &= \mu t(\chi_{A_2}(t) = 1) \\ g(n+1) &= \mu t(\chi_{A_2}(t) = 1 \ \& \ t > g(n)) \end{aligned}$$

Это значит, что $g(0)$ — номер первой всюду определенной функции, а $g(n+1)$ — это номер первой всюду определенной функции *после номера* $g(n)$, т.е. следующая после $g(n)$. Тогда пусть $f_{g(0)}, f_{g(1)}, \dots$ — последовательность из всех всюду определенных рекурсивных функций.

Итак, у нас есть некоторая универсальная функция $U(n, x) = f_n(x)$. Определим подобную «универсальную» функцию для всех всюду определенных функций: $H(n, x) = U(g(n), x) = f_{g(n)}(x)$. Очевидно, что эта функция будет сама всюду определена. Эта функция и образует нашу предыдущую таблицу из аргумента Кантора. Далее нам нужно выбрать все диагональные значения этой таблицы, это будет $H(x, x)$. И теперь мы определяем функцию, которая будет отличаться от $H(x, x)$ во всех позициях, например, $h(x) = H(x, x) + 1$. Эта функция всюду определена, так как определялась через всюду определенную функцию $H(n, x)$. Но хотя бы в одной позиции эта функция отличается от каждой из $f_{g(n)}(x)$, а именно в $f_{g(n)}(n)$, т.е. функции $h(x)$ нет в нашем списке всюду определенных функций. Противоречие, значит наша проблема неразрешима. \square

7.4.3 Проблема эквивалентности

Мы знаем, что такое эквивалентные функции? Функции f и g эквивалентны, если на всех значениях x они выдают одно и то же значение. Вопрос только, можем ли мы это всегда определить, просто смотря на текст их программы?

$$A_3 = \{(n, m) \mid f_n \equiv f_m\}$$

Теорема. Множество A_3 неразрешимо.

Доказательство. На этот раз будем доказывать немного по другому. Но все еще от противного, так что предположим, что функция

$$\chi_{A_3}(n, m) = \begin{cases} 1 & \text{если } f_n \equiv f_m \\ 0 & \text{если } f_n \not\equiv f_m \end{cases}$$

Теперь давайте для каждой функции $f_i(x)$ определим вспомогательную функцию $d_i(x)$, которая такая же, как и $f_i(x)$ в терминах определенности, но всегда или не определена, или возвращает 1. То есть

$$d_i(x) = \begin{cases} 1 & \text{если } f_i(x) \text{ — определена} \\ - & \text{если } f_i(x) \text{ — не определена} \end{cases}$$

Такие функции точно являются примитивно рекурсивными, их можно определить просто как

$$d_i(x) = \text{sg}(f_i(x)) + \overline{\text{sg}}(f_i(x))$$

Но раз $d_i(x)$ определена тогда и только тогда, когда $f_i(x)$ определена, то $d_i(x)$ тождественна равна 1 только если $f_i(x)$ определена всегда. Раз наша $d_i(x)$ — вычислимая функция, то у нее должен быть свой номер, давайте назовем этот номер $g(i)$. То есть функции $f_i(x)$ соответствует функция $d_i(x) = f_{g(i)}(x)$. Хорошо. Является ли $g(i)$ вычислимой? Да, является, мы можем взять i , получить код β_i (мы знаем, что это вычислимо), построить по нему дерево D_i (тоже вычислимо), далее мы просто преобразуем это дерево, добавляя к нему сигнумы, получим дерево $D'_i = D_j$ (вычислимо), построим его код β_j (вычислимо), найдем номер j (вычислимо), и вот мы и получили то, что хотели, $g(i) = j$. Итого, эта функция вполне вычислима. Хорошо. Остался последний компонент, функция константной 1 очевидно тоже вычислима, пусть ее номер будет i_0 . Тогда давайте рассмотрим функцию $\chi_{A_3}(i_0, g(n))$. Эта функция будет выдавать 1, если $f_{g(n)} \equiv f_{i_0}$. Если мы раскроем определения этих функций, то получим $d_n \equiv 1$, а мы уже узнали, что это верно только если f_n всюду определена. Итого, мы получили что $\chi_{A_3}(i_0, g(n))$ выдает 1 если f_n всюду определена и 0, если нет, то есть $\forall n : \chi_{A_3}(i_0, g(n)) = \chi_{A_2}(n)$. Но мы уже доказали, что $\chi_{A_2}(n)$ невычислимо, так что если бы χ_{A_3} и правда было бы вычислимо, как мы предположили, то мы могли бы вычислить $\chi_{A_2}(n)$ так и получить противоречие. Этим доказывается, что $\chi_{A_3}(n, m)$ невычислимо и что A_3 — неразрешимо. \square

Глава 8

Вычислительная сложность алгоритмов

8.1 Сложность алгоритмов

Достаточно короткая глава. Итак, мы узнали, что некоторые задачи в принципе не имеют алгоритмического решения. Такие для нас, конечно, интересны, но не особо полезны, так как мы их все равно не сможем решить. А что насчет тех задач, которые можем? Для них всех существует огромное число разных алгоритмов. Как нам узнать, какие из них лучше для наших практических целей? Для этого есть такое понятие, как *вычислительная сложность*. Она может измерять время работы программы или затраты программы по памяти. Особое внимание уделим времени работы, так как оно — более ценный ресурс.

Хоть мы и математики, абстрактная теория алгоритмов — немного не то, что нам сейчас нужно, так что будем рассматривать ее с прикладной стороны. Поэтому никаких точных определений тут не будет.

Пусть у нас есть программа P , которую мы тестируем, и она принимает входные данные из множества A . И пусть, для примера, $a \in A$ — одно такое «входное данное». Тогда $t_P(a)$ — время работы нашей программы на этом «данном». В чем оно измеряется? А не так важно. Мы правда хотим, чтобы оно было *дискретным*, т.е. измерялось в целых числах. Например это может быть число необходимых машинных команд для выполнения программы. За 1 шаг мы можем выбрать что угодно, главное, чтобы эти команды и правда выполнялись за примерно одно время, то есть чтобы за 1 шаг не выполнялись одновременно и операция $+1$ и операция «доказать Великую теорему Ферма». Примерно понятно? Хорошо. Обычно за 1 шаг мы берем что-то типа операции сравнения, или сдвига, или арифметической операции, или присваивание, или еще что-то в таком духе.

Итак, наша функция t_P принимает как аргумент входные данные и выдает время, т.е. $t_P : A \rightarrow \mathbb{N}$. Это конечно хорошо, но нам не особо удобно

работать на конкретных входных данных, да и нам это и не до такой степени важно. От чего в большей степени зависит время работы программы? От размера входных данных. Под таким размером может пониматься что угодно, в зависимости от решаемой задачи. Просто какое-то число, которое мы можем легко измерить для любых конкретных данных. Обозначается $\|a\| \in \mathbb{N}$. Тогда определим новую функцию:

$$\tau_P(n) = \max_{\|a\|=n} (t_P(a))$$

То есть $\tau_P(n)$ — это *максимальное* время работы программы на всех возможных данных размера n . Или время работы в худшем случае. И да, это уже более удобная функция $\tau_P : \mathbb{N} \rightarrow \mathbb{N}$. Только вот еще проблема, эта функция может выглядеть например как $\tau_P(n) = 32n^3 + 31n^2 + 30n + 29$. Нам точно нужны все эти подробности? И правда, не нужны, мы хотим только примерно оценить, займет ли работа нашей программы 10 лет или 5 минут или 3 секунды, или микросекунду. И даже не так, так как мы измеряем скорость работы именно программы, а не компьютера, на котором она выполняется, нам не так важно даже знать точное время. Нам важно только знать, насколько ухудшается время ее работы при увеличении данных. А это уже называется асимптотика и мы это знаем из матанализа. Для того, чтобы получить, мы просто находим самую быстро-растущую компоненту $\tau_P(n)$, отбрасываем все константы и пишем ее. Например на этом примере мы имеем $T_P(n) = O(n^3)$. Да, это та самая вычислительная сложность с программирования. Эта вещь позволяет нам примерно оценивать то, как время увеличивается с ростом данных: если мы увеличим размер данных в 10 раз у такой программы, то ее время работы увеличится в $10^3 = 1000$ раз.

Пример. Давайте рассмотрим крайне важную в программировании задачу — сортировку массива. Пусть у нас есть массив A , с индексами от 1 до n . Да, с 1, в математике так принято. Так вот. Размер исходных данных тогда как раз будет n . Давайте напишем самую простую сортировку, которая придет нам в голову — сортировку пузырьком. Писать будем на псевдо-Паскальном псевдокоде, потому что опять же, так принято.

```
for i = 1 to n-1 do
  for j = i+1 to n do
    if A[i] > A[j] then
      A[i] <-> A[j]
```

Да, это <-> не совсем стандартное, но нас ведь это не смущает, верно? Это просто поменять значения в двух ячейках. Давайте посчитаем, какое время это выполняется. Если мы обозначим выполнение i -го прохода цикла за S_i , то тогда общее время будет $\tau_P(n) = S_1 + S_2 + \dots + S_{n-1}$. В каждом внутреннем цикле по j выполняется, скажем, 6 команд: сравнение j с n , потом сравнение $A[i] > A[j]$, потом обмен $A[i]$ и $A[j]$ (у нас тут самый

худший случай, помните, поэтому считаем, что обмен всегда нужен), который считается за 3 операции (помните стандартную процедуру обмена? $t = A[i]$; $A[i] = A[j]$; $A[j] = t$), а потом увеличение j на 1. Можно было считать как-то не так, это не так важно, но у нас 6. Сколько раз выполняется цикл по j ? $n - i$. То есть $S_1 = 6(n - 1)$, $S_2 = 6(n - 2)$ и т.д. до $S_{n-1} = 6 \cdot 1$. Итого

$$\begin{aligned} \tau_P(n) &= 6(n - 1) + 6(n - 2) + \dots + 6 \cdot 1 = 6(1 + 2 + \dots + (n - 1)) \\ &= 6 \cdot \frac{1 + (n - 1)}{2} (n - 1) = 3n^2 - 3n = O(n^2) \end{aligned}$$

Вот и все, наша пузырьковая сортировка имеет квадратичную сложность, $O(n^2)$, что не то чтобы очень хорошо.

Какие временные сложности вообще часто встречаются? А вот какие:

- $O(1)$ — константная. Очень редкая, во всех интересных задачах невозможная.
- $O(\log n)$ — логарифмическая. Заметьте, основание логарифма не важно, так как оно просто дает константный множитель, а мы их игнорируем.
- $O(n)$ — линейная.
- $O(n \log n)$ — линейно-логарифмическая. Необычная, но очень важная сложность. Это самая лучшая известная сложность для универсальных алгоритмов сортировки.
- $O(n^k)$ — полиномиальная. k здесь какое-то натуральное число, $k \geq 2$. Сюда относятся $O(n^2)$, $O(n^3)$ и т.д.
- $O(2^n)$ — экспоненциальная.

Алгоритмы логарифмической и линейной сложности называются алгоритмами реального времени, так как способны обрабатывать поступающие данные одновременно с тем, как они поступают. Алгоритмы с линейно-логарифмической и полиномиальной сложностью также часто применяются на практике. А вот экспоненциальная сложность — слишком медленно, поэтому стоит всеми силами избегать алгоритмов такой сложности.

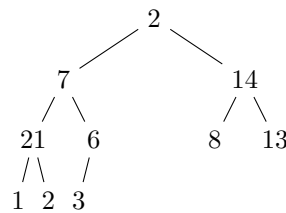
Давайте для примера рассмотрим, как меняется время работы программы на разных n у разных сложностей. Пусть наш компьютер может выполнять 10^6 операций в секунду, тогда время работы программ в секундах будет:

n	10	20	50	1000
$O(\log_2 n)$	$4 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$7 \cdot 10^{-6}$	10^{-5}
$O(n)$	10^{-5}	$2 \cdot 10^{-5}$	$5 \cdot 10^{-5}$	10^{-3}
$O(n^2)$	10^{-4}	$4 \cdot 10^{-4}$	$2.5 \cdot 10^{-3}$	1
$O(2^n)$	10^{-3}	1	$10^9 \approx 27$ лет	$10^{294} \approx 10^{286}$ лет

8.2 Быстрая сортировка

Очень важная задача, как я уже говорил — сортировка массивов. Наш пузырьрек способен справиться со временем $O(n^2)$, но можно лучше, например $O(n \log n)$. Такие алгоритмы сортировок называются *быстрыми* и их существует достаточно много. Не путать с QuickSort, она хоть и относится к этому классу, но она не единственная. Здесь мы разберем алгоритм сортировки heapsort, или пирамидальную сортировку.

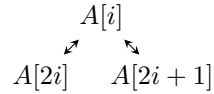
В ней будет использоваться особая структура данных — полное насыщенное бинарное дерево, где каждой вершине сопоставлен элемент массива, отсортированное особым образом, или так называемая «куча». Мы будем представлять наш массив в виде такого дерева таким образом: первый элемент массива — корень, а следующие размещаются по ярусам дерева (сверху вниз), при этом идя по ярусам слева направо. Последний ярус может быть неполным. Например, массив 2, 7, 14, 21, 6, 8, 13, 1, 2, 3 будет представлен в виде



У такого дерева есть 2 интересных свойства. Во-первых, высота этого дерева для n элементов выражается как $O(\log_2 n)$. Доказывается достаточно легко, так что это я делать не буду. Во вторых, у вершины, соответствующей $A[i]$ потомками являются вершины $A[2i]$ и $A[2i + 1]$. Доказать это можно индукцией. Очевидно, что у $A[1]$ потомками будут $A[2]$ и $A[3]$, это базис индукции. Индуктивное предположение — что для i это выполняется, т.е. у $A[i]$ потомки $A[2i]$ и $A[2i + 1]$. Тогда какие потомки у $A[i + 1]$? Очевидно, следующие 2 вершины, из-за того, как мы строим это дерево: $A[2i + 2]$ и $A[2i + 3]$, и правда $2i + 2 = 2(i + 1)$ и $2i + 3 = 2(i + 1) + 1$. Доказали. Что нам это дает? А то, что мы можем даже не строить наше дерево, мы можем по-прежнему работать с нашим массивом, но представлять, что на самом деле он — дерево, и перемещаться по «потомкам» и «предкам» при помощи этих формул. Очень удобно и экономно.

Так как мы сортировать то будем? В 2 этапа. Для начала, мы хотим превратить наш изначальный массив в «кучу», т.е. специальным образом отсортировать. Константин Иванович называет эту часть «пересыпка». Так каким же свойством наша «куча» должна обладать? А вот каким: любой путь от корня до листьев должен монотонно нестрого возрастать (нестрого — значит могут быть равны). То есть самые большие элементы будут в нижней части дерева, а самые маленькие — в верхней. Как мы будем это делать? А вот так. Мы будем идти по дереву в обратном порядке (снизу вверх, справа налево), и корректировать положение каждой вершины. Пусть мы

сейчас в вершине $A[i]$ с потомками $A[2i]$ и $A[2i + 1]$. Если $A[i]$ больше хотя бы одного из его потомков, то мы меняем его местами с самым маленьким из них (т.е. «большой» $A[i]$ опускается вниз). После этого мы снова будем рассматривать то место, куда переместился наш $A[i]$, и будем опускать его еще ниже, если это нужно. Как только уже не нужно будет опускать или мы дойдем до конца дерева, то мы останавливаем эту процедуру и идем дальше, т.е. переходим к $A[i - 1]$ (напоминаю, в обратном порядке).



Как только мы выполним этот процесс для $A[i]$, все поддерево с корнем в $A[i]$ будет «кучей», т.е. все пути от $A[i]$ к листьям будут монотонно нестрого возрастать. Когда мы так дойдем до корня, все дерево станет «кучей», или, как предпочитает говорить Константин Иванович, сортирующим деревом. Какая сложность у этого этапа? Мы перебрали все вершины дерева, это n , и каждую из них опускали вниз по дереву, максимально — на всю его высоту $O(\log_2 n)$. Итого мы имеем сложность $O(n \log_2 n)$. А теперь приступим ко второму этапу.

Мы будем по очереди доставать вершины из корня дерева. Так как все пути из корня в листья возрастают, то корень — самый маленький элемент всего дерева. Если мы его достанем из корня, и поместим в новый массив B , и после чего каким-то образом вернем дерево в состояние «кучи», то мы можем так доставать наименьшие элементы из «кучи», пока она не закончится. Тогда массив B будет отсортированной версией исходного массива A . Но как мы будем превращать дерево в кучу, если мы изъяли элемент из его корня? А вот так: мы переместим в корень меньший из 2-х потомков, а потом то же самое сделаем и с образовавшимся пустым местом. Так мы будем делать, пока не опустимся до низа дерева, т.е. примерно $O(\log_2 n)$ раз. И делать мы это будем, доставая каждую вершину, которых n . Итого общая сложность $O(n \log_2 n)$. Вполне можно показать, что если мы выберем как корень меньшего из 2-х потомков, то свойство «кучи» сохранится. На этом все с нашим алгоритмом сортировки.

Глава 9

Системы Поста

9.1 Определения

Мы уже определили 2 абсолютно разных модели вычисления: конечные автоматы (и основанная на них машина Тьюринга) и рекурсивные функции. Пришло время третьей: системы Поста. Да, того же самого Поста, который придумал предполные классы функций алгебры логики. Эти системы будут основаны на модификации слов по особым правилам.

Мы будем использовать одновременно 3 алфавита: основной алфавит A , вспомогательный алфавит B и алфавит для переменных V . На самом деле, мы почти везде будем использовать объединение алфавитов $A \cup B$, так что разделение $A \cup B$ на два алфавита A и B будет важно только несколько позже. И да, мы хотим, чтобы $|A \cup B| \geq 2$, мы не будем работать с алфавитами из одного символа. Еще мы определим так называемые «образцы» — слова из множества $(A \cup B \cup V)^*$, т.е. слова, которые могут содержать имена переменных в себе. Обычно обозначаются t .

Еще нам потребуются так называемые *подстановки*. Если t — образец, в который входят переменные x_1, \dots, x_n , то мы можем определить подстановку Θ :

$$\Theta = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \end{pmatrix}$$

Эта подстановка ставит каждой из переменных в соответствие некоторое слово $\alpha_i \in (A \cup B)^*$. Мы можем *применить* образец t с помощью подстановки Θ и получить некоторое слово $\alpha = t \cdot \Theta$. Как мы его получаем? Мы просто заменяем все вхождения переменных x_1, \dots, x_n в образец t на соответствующие $\alpha_1, \alpha_2, \dots, \alpha_n$. Множество всех слов, которые мы можем получить подобным образом из образца t мы обозначим U_t , с оговоркой, что для этого все слова $\alpha_1, \alpha_2, \dots, \alpha_n$ должны быть *непустыми*.

Теорема. Для любых t_1 и t_2 , если множества U_{t_1} и U_{t_2} совпадают, то t_1 и t_2 совпадают с точностью до имен переменных, и наоборот.

Доказательство. Докажем это сначала в одну сторону, что если $U_{t_1} = U_{t_2}$, то $t_1 = t_2$ (с точностью до имен переменных). Давайте как-то обозначим отдельные символы t_1 и t_2 : $t_1 = \sigma_1\sigma_2\cdots\sigma_n$, $t_2 = \delta_1\delta_2\cdots\delta_m$. Длина t_1 — n , длина t_2 — m . Нам сначала надо доказать, что они равны. Давайте предположим, что это не так и $n \neq m$. Тогда одно из них короче. Когда получаются самые короткие слова из некоторого образца? Когда мы заменяем все переменные на однобуквенные слова (пустые нельзя). Тогда самые короткие слова в U_{t_1} будут иметь длину n , а в U_{t_2} будут иметь длину m . Если $n \neq m$, то будут слова, которые будут в одном из множеств, но не в другом. Значит $n = m$.

Теперь надо доказать, что они отличаются только в именах переменных. Для этого давайте будем просто рассматривать по очереди пары (σ_i, δ_i) . Возможно 4 варианта:

- $\sigma_i \in A \cup B$, $\delta_i \in A \cup B$. Оба — обычные символы, не переменные. Рассмотрим опять же кратчайшие слова, где все переменные заменяются на однобуквенные слова, и поэтому длина слова и индексы остаются такой же. Тогда, если $\sigma_i \neq \delta_i$, то все кратчайшие слова в U_{t_1} и U_{t_2} будут отличаться в i -м символе. Но $U_{t_1} = U_{t_2}$, так что $\sigma_i = \delta_i$.
- $\sigma_i \in A \cup B$, $\delta_i \in V$. Опять же, рассмотрим кратчайшие слова. Тогда во всех кратчайших словах в U_{t_1} на i -м месте всегда будет стоять σ_i , а в U_{t_2} на i -м месте будет стоять абсолютно что угодно. Так что этот вариант в принципе невозможен, если $U_{t_1} = U_{t_2}$.
- $\sigma_i \in V$, $\delta_i \in A \cup B$. Абсолютно аналогично предыдущему варианту, невозможен.
- $\sigma_i \in V$, $\delta_i \in V$. Оба — переменные. Тут особо нечего сказать, вполне возможный вариант, но нам нужно рассмотреть одну деталь. То есть что переменные будут соответствовать *полностью*, то есть, например, $t_1 = x0x$, а $t_2 = y0y$, но не $t_2 = y0z$. Так, как это записать формально? Пусть наша переменная σ_i встречается еще и в другом месте, в месте j : $\sigma_i = \sigma_j$. Мы хотим доказать, что у второго слова это тоже будет одна и та же переменная: $\delta_i = \delta_j$. Как всегда, если это будет не так $\delta_i \neq \delta_j$, то в U_{t_1} кратчайшие слова в символах i и j будут обязательно содержать одну и ту же букву (напоминаю, заменяются сами переменные, а не вхождения, например, переменная x везде будет заменяться на одну и ту же букву, даже если встречается несколько раз в образце). В это же время в U_{t_2} кратчайшие слова там могут содержать что угодно. Поэтому так нельзя, так как $U_{t_1} = U_{t_2}$. В обратную сторону эта же логика тоже работает.

Отсюда слова t_1 и t_2 и правда одинаковы с точностью до замены имен переменных.

А теперь докажем это в другую сторону. Пусть $t_1 = t_2$ (с точностью до имен переменных), а нам надо доказать, что $U_{t_1} = U_{t_2}$. По определению

равенства множеств, это эквивалентно $U_{t_1} \subseteq U_{t_2} \& U_{t_2} \subseteq U_{t_1}$. Доказательства абсолютно одинаковы, так что докажем только $U_{t_1} \subseteq U_{t_2}$. Для начала, пусть у нас есть некоторое слово $\alpha \in U_{t_1}$. Чтобы доказать, что $U_{t_1} \subseteq U_{t_2}$, необходимо доказать, что $\alpha \in U_{t_2}$. Хорошо. Если $\alpha \in U_{t_1}$, то оно получается применением некоторой подстановки Θ : $\alpha = t_1 \cdot \Theta$ или

$$\alpha = t_1 \cdot \begin{pmatrix} x_1 & \cdots & x_n \\ \alpha_1 & \cdots & \alpha_n \end{pmatrix}$$

Хорошо. Еще, раз у нас t_1 и t_2 отличаются только заменой имен переменных (пусть t_1 использует x_1, \dots, x_n , а t_2 использует y_1, \dots, y_n), то мы можем записать эту замену имен при помощи подстановки

$$\Theta' = \begin{pmatrix} y_1 & \cdots & y_n \\ x_1 & \cdots & x_n \end{pmatrix}$$

Она заменяет переменные y на переменные x , тогда $t_2 \cdot \Theta' = t_1$. Тогда $\alpha = t_1 \cdot \Theta = (t_2 \cdot \Theta') \cdot \Theta = t_2 \cdot (\Theta' \cdot \Theta)$. Мы конечно не определили операцию $\Theta' \cdot \Theta$, но думаю вполне понятно, что она делает — это просто комбинирование подстановок. Сначала применяем Θ' , потом Θ . Тогда

$$\Theta'' = \Theta' \cdot \Theta = \begin{pmatrix} y_1 & \cdots & y_n \\ \alpha_1 & \cdots & \alpha_n \end{pmatrix}$$

И это — новая подстановка, так что $\alpha = t_2 \cdot \Theta''$. Значит $\alpha \in U_{t_2}$, что мы и хотели доказать. \square

Все это конечно жутко интересно (нет), но нам надо бы переходить к собственно системам Поста, а не просто играть с подстановками. Так вот. Основной компонент системы Поста — это так называемая *продукция*. Продукцией называется нечто вида

$$\pi = \frac{t_1 t_2 \cdots t_n}{t_{n+1}}$$

Здесь t_1, t_2, \dots, t_{n+1} — образцы, t_1, \dots, t_n называются посылками, а t_{n+1} называется заключением. Это аналогия с условием «если»: *если* у нас есть среди текущей информации все посылки t_1, \dots, t_n , *то* мы добавляем туда еще и t_{n+1} . Причем если никаких посылок не требуется, то есть $n = 0$, то π называется *аксиомой*.

Продукции это конечно тоже хорошо, но нам надо научиться с ними что-то делать, а именно применять. Пусть у нас есть продукция π , как указано выше, и пусть в посылках t_1, \dots, t_n используются переменные x_1, \dots, x_m . Еще пусть у нас есть n слов (не обязательно разных), $\alpha_1, \dots, \alpha_n \in (A \cup B)^*$. Если существует особая подстановка

$$\Theta = \begin{pmatrix} x_1 & \cdots & x_m \\ \beta_1 & \cdots & \beta_m \end{pmatrix}$$

причем такая, что мы можем получить все α из соответствующих t : $\alpha_i = t_i \cdot \Theta$ (т.е. если слова $\alpha_1, \dots, \alpha_n$ подходят под образцы t_1, \dots, t_n при помощи подстановки Θ), то применением продукции π к нашим словам называется слово $\alpha = t_{n+1} \cdot \Theta$, то есть применение той же подстановки к заключению продукции.

Неплохо. Так что же такое система Поста? Это набор $P = (A, B, V, \Pi)$, где A, B, V — наши алфавиты, а Π — это какое-то множество продукций. Как же она работает? Она выдает последовательность $W = \alpha_1, \alpha_2, \dots$ из разных слов $(A \cup B)^*$. Каждое из слов α_i является или применением некоторой аксиомы (из Π), или некоторой продукции-не-аксиомы, в которой могут использоваться все предыдущие слова вывода W .

У выводов системы Поста есть несколько важных свойств.

- α_1 всегда является применением аксиомы, так как еще не было слов, к которым можно было бы применить какую-то другую продукцию.
- Если α_i является применением продукции π , которая не аксиома, значит, что словам $\alpha_1, \dots, \alpha_{i-1}$ можно так сопоставить посылки продукции, что будет существовать некая подстановка Θ , которая переводит посылки продукции в эти слова из набора $\alpha_1, \dots, \alpha_{i-1}$, и переводит заключение продукции в α_i .
- В выводе может несколько раз повторяться одно и то же слово, это не запрещено.
- Следующее слово вывода нельзя определить однозначно, потому что может существовать много разных возможных вариантов применения посылок.
- Вывод может быть как конечным, так и бесконечным.
- Мы можем создать множество всех возможных выводов, которые порождает некоторая система Поста.

Особенно нам важно знать, какие слова может получить данная система Поста. Поэтому отдельно определим слова $\alpha \in (A \cup B)^*$, выводимые в системе Поста P , как те, которые встречаются хотя бы в одном ее выводе (существует вывод W , который содержит это слово).

9.2 Задачи, решаемые системами Поста

Это все конечно хорошо, но мы бы хотели, чтобы системы Поста решали конкретные задачи. Но как именно мы будем задавать системе Поста условие? А вот так: « t —?». Да, просто образец t и знак вопроса. Система Поста будет генерировать всевозможные слова, и когда встретится слово, подходящее под образец t , то это слово будет решением задачи. Итак, решением задачи « t —?» является слово $\alpha \in (A \cup B)^*$, которое выводимо системой Поста P , и для которого $\alpha = t \cdot \Theta$ для некоторой подстановки Θ . Например,

t может быть «зная основание = 5 и высота = 6, найти площадь треугольника $S = x$ ». Во всей этой записи единственная переменная — это x . Тогда правильная система Поста будет генерировать все возможные решения подобных задач, и то, которое совпадет с условием, даст правильное значение x . Вопрос правда, как именно система Поста будет генерировать все слова?

Далее последует сокращенное описание алгоритма решения. Сокращенное, потому что оригинальное описывает все слишком подробно и медленно. Итак, пусть в нашей системе Поста $P = (A, B, V, \Pi)$ наши множества равны

$$\begin{aligned} A &= \{a_1, \dots, a_m\} \\ B &= \{b_1, \dots, b_l\} \\ V &= \{x_1, \dots, x_d\} \\ \Pi &= \{\pi_1, \dots, \pi_k\} \end{aligned}$$

Пусть у нас дана задача « t —?». Ее решением будет $\alpha = t \cdot \Theta$. Учтем, что у этой задачи может не быть решений, или быть 1 решение, или 2, или даже бесконечное число решений, мы не знаем. Для начала, сможем ли мы вообще заметить, что мы нашли решение? Иными словами, если нам уже дано слово α , сможем ли мы проверить за конечное время, существует ли подстановка Θ , для которой $\alpha = t \cdot \Theta$? Ну, раз мы математики и не ищем оптимальный алгоритм, то будем просто перебирать все возможные Θ . Но их бесконечно много. Тогда немного ограничим область перебора. Слово α у нас имеет известную длину, назовем ее s . Если в Θ встречается переменная, которая заменяется на слово длины $> s$, то эта Θ точно не может быть той самой, потому что тогда слово на выходе будет тоже иметь длину $> s$. Тогда мы можем просто перебирать все Θ , где все переменные заменяются на слова длины $\leq s$. Их, очевидно, конечное число, но давайте все равно посчитаем.

Слов длины 0 у нас \bar{A}_{m+l}^0 , длины 1 — \bar{A}_{m+l}^1 , и т.д. Тогда всего слов длины $\leq s$ будет

$$\bar{A}_{m+l}^0 + \bar{A}_{m+l}^1 + \dots + \bar{A}_{m+l}^s$$

В подстановке d переменных. Тогда всего различных Θ будет

$$\left(\bar{A}_{m+l}^0 + \bar{A}_{m+l}^1 + \dots + \bar{A}_{m+l}^s \right)^d$$

Т.е. очень большое, но *конечное* число. Перебрав их все мы узнаем, подходит ли хотя бы одна (тогда α — решение), или все не подходят — (α — не решение).

Теперь нам нужен механизм генерации всех возможных слов, которые выводимы данной системой Поста. Мы не можем генерировать их как попало, потому что тогда не будет гарантии, что каждое слово встречается через конечное время¹. Генерировать слова сами по себе мы не можем, так

¹В том числе решение нашей задачи, если оно есть. Если не будет этой гарантии, то мы не будем уверены, что сможем решить каждую решаемую задачу за конечное время.

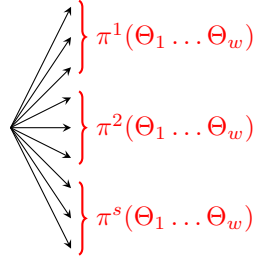
что мы будем генерировать полноценные выводы системы Поста P . Было бы логично сортировать их по возрастанию длины: сначала все выводы длины 1 (состоящие из одного применения продукции, если не помните, что это значит), потом все выводы длины 2, потом все выводы длины 3 и т.д. Проблема в том, что у некоторых систем Поста будет бесконечное число даже выводов длины 1, например, если там будет аксиома, содержащая переменные. Тогда в переменную можно будет подставить слова бесконечным числом способов и мы получим бесконечное число выводов длины 1. Тогда мы так и не доберемся до выводов больших длин, а ведь решение нашей задачи может быть именно там...

Поэтому мы ограничим не только длину вывода, но и разрешенную длину слов, которые можно подставлять в переменные. То есть сначала выводы длины 1, где во все переменные подставляются слова длины ≤ 1 , потом выводы длины 2, где во все переменные подставляются слова длины ≤ 2 и т.д. И так, на этапе r мы рассмотрим все выводы длины r , где в переменные подставляются слова длины $\leq r$.

Давайте сначала рассмотрим $r = 1$. Первое слово вывода всегда аксиома, так что мы обязаны применить одну из аксиом. Пусть π^1, \dots, π^s — все аксиомы. И еще, пусть $\Theta_1, \dots, \Theta_w$ — все подстановки, удовлетворяющие условию на длину слов ≤ 1 . Таких конечное число, как мы уже выяснили ранее, всего лишь

$$\left(\bar{A}_{m+l}^0 + \bar{A}_{m+l}^1 \right)^d$$

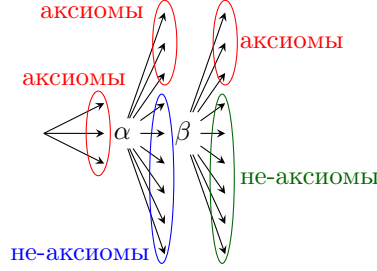
Что далее? Давайте применим по очереди все подстановки к аксиоме π^1 . Получим сколько-то различных выводов длины 1. Потом так же к аксиоме π^2 и так до π^s :



Мы будем строить некий аналог дерева разбора, и это — дерево глубины 1. Так мы разберем все возможные выводы системы Поста для $r = 1$. После этого мы начнем сначала для $r = 2$, построим первый слой дерева, потом второй слой. Для $r = 3$ построим сначала первый слой, потом второй, потом третий и так далее.

Теперь давайте рассмотрим случай $r = i + 1$. Пусть все подходящие подстановки будут $\Theta_1, \dots, \Theta_u$ (подходящие — все переменные заменяются на слова длины $\leq r$). У нас будет 2 варианта — или применить одну из аксиом π^1, \dots, π^s , или одну из других продукций. Для аксиом нам нужны исключительно подстановки $\Theta_1, \dots, \Theta_u$, так что на каждом слое дерева часть вариантов продолжения, соответствующая аксиомам будет всегда одинаковой.

Другой вариант — другие продукции. Для них нужны не только подстановки $\Theta_1, \dots, \Theta_u$, но и некоторые входные слова. Как входные слова мы будем использовать те слова, которые встречались ранее по пути, например, если мы рассматриваем вывод, где на первом шаге мы получили слово α , а на втором β , и сейчас мы рассматриваем третий шаг, то мы можем для всех не-аксиом набирать входные слова для продукций из множества $\{\alpha, \beta\}$:



Продолжаем делать так же, пока мы не получим r слоев, т.е. все возможные выводы длины r (с учетом нашего ограничения на подстановки). Тогда если мы сначала выпишем все выводы с $r = 1$ (конечное число), потом с $r = 2$ и т.д., то получим бесконечную последовательность, но любое выводимое слово в ней встречается под конечным номером. Давайте это докажем.

Пусть у нас есть некоторое выводимое слово α . Выводимое — значит оно есть в некотором выводе $W = \alpha_1 \alpha_2 \dots \alpha_i \dots$. Пусть оно встречается там под номером i . Еще, пусть z — это максимальная длина слова, которое используется во всех подстановках $\Theta_1 \dots \Theta_i$, которые используются при построении W . Тогда наше слово α будет точно выведено на шаге $r = \max(z, i)$. Почему? Потому что так как $r \geq z$, то на шаге номер r разрешено использовать все подстановки $\Theta_1 \dots \Theta_i$, и т.к. $r \geq i$, то там будут присутствовать выводы длины i . Значит наше слово там тоже будет. Вот и все описание нашего алгоритма генерации. Чтобы найти решение задачи, мы будем просто каждое из появляющихся слов проверять на соответствие t , пока не найдем подходящее.

Учтем правда еще одно. Если наше решение существует, то тогда этот метод найдет его за конечное время. Но если решения не существует, то этот метод будет бесконечно генерировать все больше слов и так и не остановится. Ну, вот так вот, по другому никак, иначе это было бы решение проблемы остановки, которое невозможно.

9.3 Свойства слов, выводимых системами Поста

Пусть P — система Поста. Тогда давайте обозначать множество всех слов из алфавита A , которые она выводит, Ω_P . Да, это единственное место, где нам нужно разделение A и B , чтобы в Ω_P входило только то, что нам нужно.

Так вот. Пусть у нас есть некоторые абсолютно разные системы Поста $P_1 = (A_1, B_1, V_1, \Pi_1)$ и $P_2 = (A_2, B_2, V_2, \Pi_2)$. Причем пусть у каждого символа есть только одна «роль», т.е. $A_1 \cup A_2$ (все основные символы), $B_1 \cup B_2$ (все вспомогательные) и $V_1 \cup V_2$ (все переменные) — не пересекаются. Причем A_1 и A_2 вполне могут (и с большой вероятностью будут) пересекаться. Так вот, тогда Ω_{P_1} и Ω_{P_2} — множества слов, которые можно из них получить. Раз это множества, то можно определить и $\Omega_{P_1} \cup \Omega_{P_2}$ и $\Omega_{P_1} \cap \Omega_{P_2}$.

Теорема. *Существует такая система Поста $P_3 = (A_3, B_3, V_3, \Pi_3)$, у которой $\Omega_{P_3} = \Omega_{P_1} \cup \Omega_{P_2}$, т.е. она выводит слова, которые выводит или P_1 , или P_2 . (Также возможно $\Omega_{P_3} = \Omega_{P_1} \cap \Omega_{P_2}$)*

Доказательство. Нам требуется определить такую P_3 , которая это будет сделать. Для A_3 и V_3 мы поступим просто, $A_3 = A_1 \cup A_2$ и $V_3 = V_1 \cup V_2$. Для вспомогательных символов так правда не пройдет, нам потребуется еще 2 вспомогательных символа, назовем их R и S : $B_3 = B_1 \cup B_2 \cup \{R, S\}$. Сейчас объясню, зачем. Давайте на примере, пусть у нас есть 2 системы Поста, которые генерируют цепочки из только нулей и только из единиц соответственно:

$$\begin{aligned} \pi_1 &= _ \\ \pi_2 &= \frac{x}{x0} \\[10pt] \pi'_1 &= _ \\ \pi'_2 &= \frac{x}{x1} \end{aligned}$$

Если мы просто объединим продукции, то мы получим

$$\begin{aligned} \pi''_1 &= _ \\ \pi''_2 &= \frac{x}{x0} \\ \pi''_3 &= \frac{x}{x1} \end{aligned}$$

Но с помощью это системы можно получить любую цепочку из 0 и 1! Нужно просто в нужном порядке применять π''_2 и π''_3 . Не очень хорошо. Это все из за того, что мы позволили системам пересечься, и использовать по очереди то правила из первой системы, то из второй. Нам необходимо разделить их, сделать так, чтобы правила не пересекались до последнего момента. Поэтому сделаем пока так:

$$\begin{aligned} \pi''_1 &= R_ \\ \pi''_2 &= \frac{Rx}{Rx0} \\ \pi''_3 &= S_ \\ \pi''_4 &= \frac{Sx}{Sx1} \end{aligned}$$

Так все слова из первой системы будут помечены R , а из второй — S , и у систем не будет возможности пересечься. Правда нам еще надо как то получить итоговые слова, без R и S , так что добавим еще 2 продукции, которые будут их убирать:

$$\begin{aligned}\pi_5'' &= \frac{Rx}{x} \\ \pi_6'' &= \frac{Sx}{x}\end{aligned}$$

Это не мешает нам, так как все посылки из прошлых продукций требуют слова или начинающиеся с R , или S , поэтому мы не сможем применить слова без них ни для одной другой продукции. Теперь давайте это в общем виде.

Преобразуем Π_1 в Π_1' путем замены всех продукций подобным образом:

$$\pi = \frac{t_1 t_2 \cdots t_n}{t_{n+1}} \Rightarrow \pi' = \frac{Rt_1 Rt_2 \cdots Rt_n}{Rt_{n+1}}$$

Теперь все продукции первой системы используют слова, начинающиеся с R . Аналогично для $\Pi_2 \Rightarrow \Pi_2'$:

$$\pi = \frac{t_1 t_2 \cdots t_n}{t_{n+1}} \Rightarrow \pi' = \frac{St_1 St_2 \cdots St_n}{St_{n+1}}$$

Тогда итоговым множеством продукций будет

$$\Pi_3 = \Pi_1' \cup \Pi_2' \cup \left\{ \frac{Rx}{x}, \frac{Sx}{x} \right\}$$

Итак, построили систему. Давайте докажем, что она работает и $\Omega_{P_3} = \Omega_{P_1} \cup \Omega_{P_2}$. Для этого докажем $\Omega_{P_3} \subseteq \Omega_{P_1} \cup \Omega_{P_2}$, обратное вложение доказывается абсолютно аналогично. Для этого надо доказать, что любое слово $\alpha \in \Omega_{P_3}$ лежит и в $\Omega_{P_1} \cup \Omega_{P_2}$. Для начала, это слово α обязано состоять только из символов из A_3 , т.е. R и S там нет. Так как в заключениях всех посылок, кроме последних двух, есть или R , или S , то это слово получено из одной из двух последних. Это значит, что система может вывести или слово $R\alpha$, или слово $S\alpha$. Благодаря нашему процессу замены это будет означать, что или P_1 может вывести α , или P_2 может вывести α , т.е. или $\alpha \in \Omega_{P_1}$, или $\alpha \in \Omega_{P_2}$, что мы и хотели доказать. Обратно аналогично. \square

Аналогичная же теорема может быть и для $\Omega_{P_3} = \Omega_{P_1} \cap \Omega_{P_2}$, только вместо продукций $\frac{Rx}{x}, \frac{Sx}{x}$ мы добавляем продукцию

$$\frac{Rx Sx}{x}$$

то есть слово будет выведено, только если оно выводится в *обеих* системах Поста. Неожиданно, но $\Omega_{P_1} \setminus \Omega_{P_2}$ мы подобным образом определить не можем, причем совсем, если бы мы так могли для любых систем Поста, то это бы решило проблему остановки. Не очень хорошо.

9.4 Функции, вычислимые системами Поста

В начале главы уже был спойлер, что это абсолютно все вычислимые функции, но это же еще надо доказать, верно? Для начала, с какими функциями мы работаем? Все с теми же, $f : \mathbb{N}^k \rightarrow \mathbb{N}$. И да, 0 включен в \mathbb{N} . Правда наши системы Поста работают со словами, а не с числами, так что нужен способ записи. Можно было бы конечно извратиться с унарной, но лучше не надо, будем с двоичной работать. Итак, все числа записаны в двоичной системе. Тогда функция $f : \mathbb{N}^k \rightarrow \mathbb{N}$ вычислима некоторой системой Поста P , если на любых значениях x_1, \dots, x_k эта функция дает то же значение, что и система Поста (в двоичной системе). Если формально, то $\forall x_1, \dots, x_k \in \mathbb{N} (f(x_1, \dots, x_k) = y \leftrightarrow P \text{ выводит слово } "f(\overline{x_1}, \dots, \overline{x_k}) = \overline{y}").$

Для систем Поста, работающих с двоичными числами, нам часто нужна будет проверка, что нечто — вообще двоичное число. Это обеспечит вот эта маленькая система Поста:

$$\begin{aligned} \pi_1 &= N0 \\ \pi_2 &= N1 \\ \pi_3 &= \frac{N1x}{N1x0} \\ \pi_4 &= \frac{N1x}{N1x1} \end{aligned}$$

То есть если она выдает Nx , то x — двоичная запись натурального числа. Это или 0, или цепочка цифр, начинающаяся с 1.

Так, а теперь докажем нашу теорему.

Теорема. *Класс числовых функций, вычислимых системами Поста, совпадает с классом рекурсивных функций.*

Доказательство. Как всегда со словом «совпадает», на самом деле здесь скрыто 2 теоремы: что все числовые функции, вычислимые системами Поста, вычислимы рекурсивными функциями, и что все рекурсивные функции вычислимы системами Поста. Из этого полноценно доказывать мы будем только второе утверждение, потому что первое вытекает из тезиса Чёрча: так как мы описали вполне интуитивно вычислимый алгоритм работы систем Поста, то его можно реализовать с помощью рекурсивных функций. А вот обратное утверждение менее очевидно: достаточно ли сильны системы Поста, чтобы реализовать все возможные рекурсивные функции? Оказывается да.

Рекурсивные рекурсии, как мы помним, строятся из 4 компонент: набора из 3 примитивных функций, и еще операций суперпозиции, примитивной рекурсии и минимизации. Давайте их по очереди определим.

1. Примитивные функции.

Их всего 3:

$$\begin{aligned} O(x) &= 0 \\ S(x) &= x + 1 \\ I_n^m(x_1, \dots, x_n) &= x_m \end{aligned}$$

Самая сложная из них $S(x)$, с нее и начнем. К четным числам прибавлять 1 в двоичной системе просто, мы просто заменяем последний 0 на 1:

$$\pi_1 = \frac{Nx0}{S(x0) = x1}$$

С нечетными чуть сложнее, тут появляется перенос. Но что такое перенос? Мы просто должны увеличить следующие разряды на 1! То есть если сейчас следующие после самой левой 1 разряды называются x , то мы должны будем написать $(x + 1)0$, или $y0$, если обозначим $y = x + 1$. В виде продукции это будет

$$\pi_2 = \frac{Nx1 \quad S(x) = y}{S(x1) = y0}$$

Хорошо. Но не появились ли у нас граничные ситуации? И правда, появились, если $x = _$. Тогда имеем $S(_) = y$, что явно никогда не сработает. Что ж, тогда добавим случай $S(1)$ (а именно при нем $x = _$) отдельной аксиомой:

$$\pi_3 = S(1) = 10$$

На этом и все. Наши $O(x)$ и $I_n^m(x_1, \dots, x_n)$ определить еще проще:

$$\begin{aligned} \pi &= \frac{Nx}{O(x) = 0} \\ \pi &= \frac{Nx_1 \quad \dots \quad Nx_n}{I_n^m(x_1, \dots, x_n) = x_m} \end{aligned}$$

2. Суперпозиция.

Напоминаю определение суперпозиции для рекурсивных функций. Если у нас есть $f(x_1, \dots, x_n)$ и набор из n функций $g_1(x_{11}, \dots, x_{1m_1}), \dots, g_n(x_{n1}, \dots, x_{nm_n})$ ², в сумме насчитывающих l переменных. Тогда суперпозицией всех этих функций называется функция $h(x_1, \dots, x_l) = f(g_1(\dots), \dots, g_n(\dots))$. Мы хотим, чтобы каждую из этих функций вычисляла какая-то система Поста, а именно $P_f, P_{g_1}, P_{g_2}, \dots, P_{g_n}$. Каждую из их продукций мы преобразуем как

²Напоминаю, индексы такие странные, потому что у каждой из функций g_i может быть разное количество переменных, которое мы обозначаем как m_i . Тогда индексы меняются от 1 до m_i . И так как еще переменные идут не по порядку, мы должны указать, какой функции они принадлежат, так что индексы идут от $i, 1$ до i, m_i .

в прошлой теореме, добавлением вспомогательных символов в начало всех образцов, соответственно F, G_1, G_2, \dots, G_n . На самом деле это может нам даже и не понадобиться, но на всякий случай лучше все равно разделить работу подсистем Поста друг от друга. Итак, теперь построим нашу итоговую продукцию. Если функции g_i дают соответственно y_i , т.е. (с учетом вспомогательных символов) среди множества доступных слов есть $G_i g_i(x_{i1}, \dots, x_{im_i}) = y_i$, и если $f(y_1, \dots, y_n) = z$, то ответом нашей $h(x_1, \dots, x_l)$ будет z . Итого, продукция:

$$\pi = \frac{G_1 g_1(x_{11}, \dots, x_{1m_1}) = y_1 \quad \dots \quad G_n g_n(x_{n1}, \dots, x_{nm_n})}{F f(y_1, \dots, y_n) = z} \quad h(x_1, \dots, x_l) = z$$

3. Прimitивная рекурсия.

Достаточно похоже на суперпозицию. Если нам даны функции $g(x_1, \dots, x_n)$ и $h(x_1, \dots, x_n, x_{n+1}, x_{n+2})$, то мы можем определить результат применения примитивной рекурсии как

$$\begin{aligned} f(x_1, \dots, x_n, 0) &= g(x_1, \dots, x_n) \\ f(x_1, \dots, x_n, y+1) &= h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)) \end{aligned}$$

Очевидно, что нам нужны системы Поста, которые их определяют, а именно P_g, P_h . Правда, видите там еще $y+1$? Для этого нам потребуется та функция $S(x)$, которую мы определяли уже, так что ее систему Поста тоже сюда, P_S . Все эти системы Поста модифицируем добавлением префиксов (вспомогательных символов перед словами), а именно G, H, Σ (Σ потому что S занято). Тогда нам потребуется 2 продукции для двух правил. Первая достаточно просто и логично делается:

$$\pi_1 = \frac{Gg(x_1, \dots, x_n) = y}{f(x_1, \dots, x_n, 0) = y}$$

Вторая чуть сложнее. Мы не можем использовать записи типа $y+1$ в выводе, мы должны называть все переменными. Поэтому давайте назовем $y+1 = v$ (что мы можем проверить как $S(y) = v$), а еще у нас там есть $f(x_1, \dots, x_n, y)$, назовем его w . Итого все второе правило переписывается в

$$\pi_2 = \frac{\Sigma S(y) = v \quad f(x_1, \dots, x_n, y) = w \quad Hh(x_1, \dots, x_n, y, w) = z}{f(x_1, \dots, x_n, v) = z}$$

На этом и все с примитивной рекурсией.

4. Минимизация

Как мы помним, для операции минимизации нам нужны 2 функции с $n+1$ переменными: $f(x_1, \dots, x_n, x_{n+1})$ и $g(x_1, \dots, x_n, x_{n+1})$. При этом мы определяем f как

$$f(x_1, \dots, x_n, x_{n+1}) = \mu t (g(x_1, \dots, x_n, t) = x_{n+1})$$

Мы уже рассматривали то, как это нужно вычислять — перебирать значения t от 0 до бесконечности, пока не встретим то, что нам нужно. И да, напоминая, x_{n+1} в f и x_{n+1} в g — разные переменные. В g это та, которую мы перебираем при помощи t , а в f это итоговый ответ, который мы хотим получить. Итак, какие системы Поста у нас есть? P_g , куда мы добавим префикс G , и еще мы хотим использовать P_S с префиксом Σ . Так вот. Определим вспомогательную функцию $\varphi(x_1, \dots, x_n, x_{n+1}, t)$. Пусть она будет равна 1, когда среди всех $g(x_1, \dots, x_n, i)$ для i от 1 до t только последняя равна x_{n+1} . Если ни одна из них не равна, то пусть будет 0, а если есть несколько равных, мы это не определяем, мы хотим, чтобы она была равна 1 только на первом таком значении.

Итак, для начала определим ее на $t = 0$. Если там мы нашли то, что хотели, тогда $\varphi = 1$:

$$\pi_1 = \frac{Gg(x_1, \dots, x_n, 0) = x_{n+1}}{\varphi(x_1, \dots, x_n, x_{n+1}, 0) = 1}$$

Если же мы получили что-то, но оно не равное x_{n+1} (что мы проверяем при помощи $NE(x, y)$, определение см. на странице 251), то ответом будет 0:

$$\pi_2 = \frac{Gg(x_1, \dots, x_n, 0) = y \quad NE(y, x_{n+1})}{\varphi(x_1, \dots, x_n, x_{n+1}, 0) = 0}$$

Далее нужно определить для произвольного $v = t + 1$. Для начала, нам имеет смысл проверять это только если на всех предыдущих мы не нашли искомого значения t , т.е. $\varphi = 0$. Если мы сейчас увидели, что $g(x_1, \dots, x_n, t + 1) = x_{n+1}$, то мы нашли то, что нужно, и $\varphi = 1$. Иначе $\varphi = 0$. Имеем 2 продукции:

$$\begin{aligned} \pi_3 &= \frac{\varphi(x_1, \dots, x_n, x_{n+1}, t) = 0 \quad \Sigma S(t) = v}{Gg(x_1, \dots, x_n, v) = x_{n+1}} \\ \pi_4 &= \frac{\varphi(x_1, \dots, x_n, x_{n+1}, t) = 0 \quad \Sigma S(t) = v}{Gg(x_1, \dots, x_n, v) = y \quad NE(y, x_{n+1})} \end{aligned}$$

Прекрасно, мы определили φ . Оно равно 1 на первом t , для которого выполняется нужное условие. Теперь мы сможем определить и искомую функцию f и на этом закончить

$$\pi_5 = \frac{\varphi(x_1, \dots, x_n, x_{n+1}, t) = 1}{f(x_1, \dots, x_n, x_{n+1}) = t}$$

Итак, мы определили системы Поста, которые реализуют все возможные виды определений рекурсивных функций. Много раз применяя это мы сможем построить систему Поста для любой рекурсивной функции. Значит

класс функций, вычисляемых системами Поста не уже, чем класс рекурсивных функций. И по тезису Чёрча, так как есть алгоритм, реализующий работу системы Поста, то все функции, которые ими вычислимы, тоже будут вычислимы, т.е. рекурсивны. Тогда классы совпадают. \square

Тогда мы можем ввести и тезис Поста, аналогичный тезису Чёрча: класс вычисляемых функций — это все функции, вычисляемые системами Поста.

Но какие функции невозможно вычислить системами Поста? Помните, ранее мы наткнулись на то, что для двух систем Поста P_1 и P_2 мы можем определить P_3 , которая выводит слова $\Omega_{P_1} \cup \Omega_{P_2}$ или $\Omega_{P_1} \cap \Omega_{P_2}$, но не $\Omega_{P_1} \setminus \Omega_{P_2}$. Почему? А вот сейчас это и докажем.

Доказательство. Мы пытаемся доказать, что можно определить P_3 , которая выводит слова $\Omega_{P_1} \setminus \Omega_{P_2}$ не для любых P_1 и P_2 . Т.е. для некоторых конечно можно, но не для всех. Нам достаточно просто привести пример двух систем Поста, для которых это невозможно.

Ну что, ж, начнем, пусть P_1 — такая система Поста, которая выводит слова $\Omega_{P_1} = \{(m, n) \mid m, n \in \mathbb{N}\}$, то есть множество всех пар. Такую систему Поста можно даже явно определить:

$$\pi = \frac{Nm \quad Nn}{(m, n)}$$

Второй же системой Поста будет нечто сложнее: $\Omega_{P_1} = \{(m, n) \mid f_m(n) \text{ — определено}\}$. Множество всех (m, n) , для которых определено значение $f_m(n)$. Помните, что это? Это та самая последовательность всех рекурсивных/вычисляемых функций. Точно ли это можно определить? Да. Для этого мы воспользуемся нашей универсальной функцией $U(m, n) = f_m(n)$, которую мы определили ранее. Более того, мы доказали, что она вычислима, а значит существует система Поста, которая ее вычисляет. Если мы дополним ее продукцией

$$\pi = \frac{U(m, n) = v}{(m, n)}$$

то мы получим множество всех пар, где функция $U(m, n) = f_m(n)$ определена, т.е. ее вычисление завершается. Именно то, что нужно.

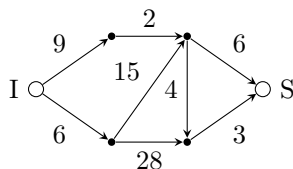
Итак, обе системы существуют. Но существует ли P_3 , у которой $\Omega_{P_3} = \Omega_{P_1} \setminus \Omega_{P_2}$? Что это вообще за множество такое? А просто: $\Omega_{P_3} = \{(m, n) \mid f_m(n) \text{ — не определено}\}$. Множество всех пар, на которых вычисление функции не завершается. Начинаете догадываться, в чем дело? Эта вещь позволила бы нам решить проблему остановки! Предположим, что эта система все таки существует. Дело в том, что любая пара (m, n) лежит или в Ω_{P_2} , или в Ω_{P_3} . Если мы одновременно запустим 2 системы Поста, P_2 и P_3 , то одна из них гарантированно сможет произвести нашу пару (m, n) . А наш метод как раз устроен так, что любое выводимое слово будет выведено через конечное время. То есть наша пара (m, n) обязательно через конечное

время выведется одной из систем. Если она выведена P_2 , то $f_m(n)$ завершает свое вычисление, а если P_3 — то нет. Тогда такой механизм сможет за конечное время различать останавливающиеся и неостанавливающиеся функции, то есть решать проблему остановки, а мы уже доказали, что это невозможно. Значит наше предположение неверно и такой P_3 не существует для конкретно этих P_1 и P_2 , то есть разность множеств возможно определить не для всех систем Поста. \square

Глава 10

Транспортные сети

К сожалению, закончить эту тему на наших лекциях мы не успели, но часть информации все таки есть. Итак, начнем, что же такое сеть в принципе? А это граф. Но *ориентированный* граф. Без петель. И еще мы у него выделили 2 особые вершины, *исток* I и *сток* S , причем из истока ребра могут только выходить, а в сток только входить. Обозначается так: $N = (V, U, \{I, S\})$. Но нам нужны не простые сети, а транспортные. Это такие, где каждому ребру сопоставлен вес, он же — пропускная способность. Это обеспечивается функцией c , которая каждому ребру ставит в соответствие положительное число: $c : U \rightarrow \mathbb{R}^+$. Тогда транспортной сетью будет сеть с такой функцией: $\bar{N} = (V, U, \{I, S\}, c)$. Транспортные сети символизируют модели распространения некоторых ресурсов от истока к стоку, и $c(u)$ — это реальная максимальная пропускная способность каждого ребра. Например, такая сеть \bar{N} может выглядеть так:



Через такие сети может течь много различных потоков. Поток характеризуется тем, сколько ресурсов протекает через каждое из ребер, поэтому математически поток ψ — это функция $\psi : U \rightarrow \mathbb{R}^+$ для какой то конкретной \bar{N} . Эта функция должна правда еще удовлетворять двум соотношениям. Первое, она должна быть везде неотрицательна и не должна нигде превышать максимальную пропускную способность ребра: $\forall u \in U (0 \leq \psi(u) \leq c(u))$. Второе, все вершины кроме истока и стока — «сбалансированы», т.е. все, что втекает в них из них же и вытекает, без вариантов. Если мы обозначим как $E^+(v)$ все входные (+, потому что ресурсы *втекают*) ребра, а $E^-(v)$ — выходные ребра вершины v , то тогда $\forall v \in V \setminus \{I, S\}$

выполняется соотношение:

$$\sum_{u \in E^+(v)} \psi(u) = \sum_{u \in E^-(v)} \psi(u)$$

Какой общий поток течет в сети? Это весь поток, вытекающий из нашего истока I , он называется величиной потока ψ :

$$L(\psi) = \sum_{u \in E^-(I)} \psi(u)$$

Единственные «несбалансированные» вершины в сети — это исток и сток, только в истоке поток может появляться из ниоткуда и только в стоке исчезать в никуда. Поэтому логично предположить, что весь поток, вытекающий из истока I в конце концов попадет в S . Это мы сейчас и докажем

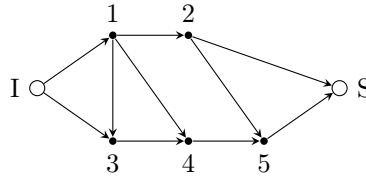
Теорема. Для любой $\bar{N} = (V, U, \{I, S\}, c)$ и любого потока $\psi : U \rightarrow \mathbb{R}^+$ выполняется соотношение

$$\sum_{u \in E^-(I)} \psi(u) = \sum_{u \in E^+(S)} \psi(u)$$

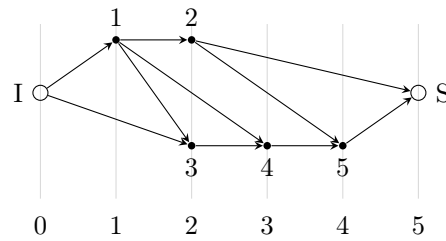
(сколько вытекло из истока, столько и втекло в сток)

Доказательство. Достаточно очевидное заявление, учитывая наш образ «потока», но нам все равно стоит его доказать. Сначала докажем более простой случай, когда в нашей транспортной сети нет циклов (элементарных длины ≥ 3), по крайней мере таких, по которым реально течет какой-то не равный нулю поток.

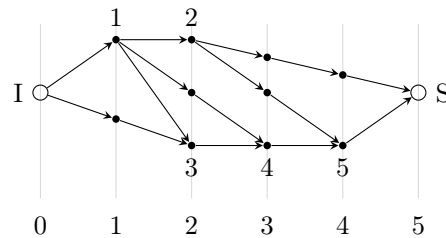
Итак, у нас есть некоторая сеть. Давайте удалим из нее все ребра, через которые ничего не течет, и все вершины, которые таким образом останутся без ребер. Тогда у нас получится, например, вот такая сеть:



Мы хотим расположить вершины по ярусам, причем особым образом, чтобы стрелки шли исключительно на следующие ярусы. Пусть на нулевом ярусе находится только наш исток I . На первом ярусе будут находиться те вершины, до которых *самый длинный путь* из I — длины 1. На втором — длины 2. На третьем — длины 3 и т.д. То есть вершина 1 будет находиться на первом ярусе, вершины 2 и 3 — на втором, 4 на третьем, 5 на четвертом и S на пятом:

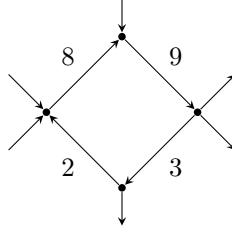


Прекрасно, как видим, теперь все стрелки идут только вперед, а не непонятно как. Проблема в том, что у нас тут некоторые из них скачут через несколько ярусов. Давайте эти длинные ребра разобьем и добавим новые вершины на них. При этом у кусочков разбитого ребра будет все та же пропускная способность и все тот же поток через нее. Получим что то вот такое:

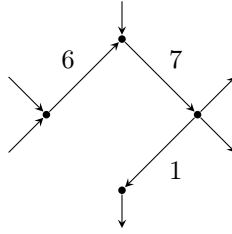


Теперь абсолютно все ребра идут не просто вперед, а ровно на следующий ярус. А теперь посмотрите. Весь поток, выходящий из I , входит в вершины первого яруса. Из-за условия баланса он же обязан выйти из вершин первого яруса. А куда он входит? Правильно, в вершины второго яруса. Если мы так же продолжим, то увидим, что весь поток, выходящий из I , неизменным попадет в вершины последнего яруса. Но вершина последнего яруса всегда одна, S , поэтому для такого частного случая без циклов утверждение теоремы доказано. Но почему в последнем ярусе только S ? А предположим, что там было бы что то еще, пусть была бы цепочка вершин, идущая дальше яруса с S . Так как возвращаться назад нельзя, то последняя вершина этой цепочки не будет иметь выходящих из нее ребер. Но это будет означать, что выходящий поток ее равен 0, а по условию баланса входящий тоже будет 0. Итого вся эта цепочка будет иметь поток 0, так как она не заканчивается в S . Вот и все.

А теперь второй этап, что будет, если в нашем графе есть циклы (элементарные, длиной ≥ 3)? Это будет выглядеть как то так:



Как видите, по этому циклу циркулирует 2 единицы потока. Мы можем эти 2 единицы вычесть из всего цикла:



При этом ничего с точки зрения остальной части сети не меняется, во все вершины все еще входит снаружи и выходит столько же, сколько раньше. При этом наш цикл исчез. Надо повернуть то же самое абсолютно со всеми. Пусть у нас есть некоторый такой цикл C . Ребра, которые в нем есть — $E(C)$. Можно найти минимальное значение потока в них всех, это и будет циркуляция потока в цикле:

$$l = \min_{u \in E(C)} (c(u))$$

Теперь нам нужно вычесть это из всех значений потока в этом цикле, то есть новая функция потока будет:

$$\psi'(u) = \begin{cases} \psi(u) - l & \text{если } u \in E(C) \\ \psi(u) & \text{если } u \notin E(C) \end{cases}$$

Во первых, эта функция — тоже легальная функция потока. Она удовлетворяет условию $\forall u \in U (0 \leq \psi'(u) \leq c(u))$, так как мы вычитали минимальное значение потока в цикле, т.е. в отрицательные числа нигде не попали. Еще она по прежнему удовлетворяет второму условию:

$$\sum_{u \in E^+(v)} \psi'(u) = \sum_{u \in E^-(v)} \psi'(u)$$

потому что для всех вершин, не лежащих в цикле, она и не поменялась, а для лежащих в нем из обеих частей равенства просто вычлось l и равенство осталось верным. Более того, раз I и S гарантированно не лежат в этом цикле, потому что для этого бы потребовалось хотя бы одно входящее и одно выходящее ребро, а у I только выходящие и у S только входящие.

Это значит, что суммарный поток из I и суммарный поток в S после этого преобразования не поменялись.

Итак, мы это сделали. Но что если у нас еще остались циклы? Ничего, будем делать так, пока циклы не кончатся, будем получать $\psi', \psi'', \dots, \psi^{(k)}$. А они кончатся? Да, кончатся, с каждым шагом мы удаляем по одному ребру, а ребер всего конечное число. Как только мы закончим, мы получим поток, у которого в этой транспортной сети нет циклов (элементарных, длины ≥ 3 , с ненулевым потоком), и мы сможем доказать для него наше выражение

$$\sum_{u \in E^-(I)} \psi^{(k)}(u) = \sum_{u \in E^+(S)} \psi^{(k)}(u)$$

так же, как мы это сделали в частном случае. Но при удалении циклов потоки из I и в S не менялись, поэтому это условие выполнялось и для изначального ψ , что и требовалось доказать. \square

Глава 11

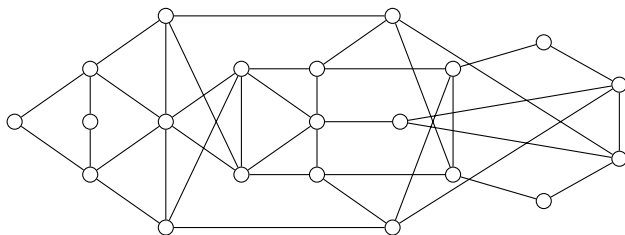
Зачетные задачи

11.1 Задача 1

Необходимые темы: общие сведения о графах, изоморфизм графов.

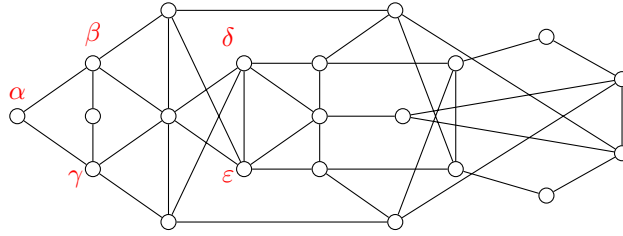
Всего в этом зачете будет 3 задачи на графы, похожие на третью, четвертую и пятую задачи первого зачета. Для их решения необходимо хорошо понимать такую вещь, как неотличимые вершины графа. Помним, что вершины графа не имеют никаких свойств, кроме собственно их положения в графе¹, поэтому если мы можем мысленно поменять 2 вершины местами и общая структура графа не изменится, то эти 2 вершины — неотличимы. Весь смысл задачи — записать через сочетания и размещения количество способов разместить по 1 букве латинского алфавита в каждой вершине графа, аналогично 3-ей задаче. Лучше рассмотреть на примере:

Задача. Сколько существует графов с вершинами из $\{a, \dots, z\}$, изоморфных графу



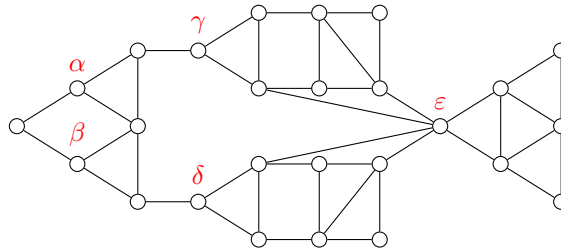
Решение. Начнем раздавать вершины слева направо. На этом рисунке обозначены все важные вершины:

¹ «Лево-право не работает» — К.И.Костенко



Для начала выберем букву для вершины α , это будет C_{26}^1 . Далее, заметим, что 2 связанных с ней вершины, β и γ — неотличимы, а неотличимые объекты мы выбираем одновременно, C_{25}^2 . Вершина между ними не похожа на другие, ее отдельно, C_{23}^1 . Далее, следующие 3 вершины. Одна из букв β, γ будет младше другой, пусть для определенности $\beta < \gamma$. Тогда одна из следующих 3 вершин будет связана только с младшей, β , одна — только со старшей, γ , а третья — с обеими. Значит следующие 3 вершины — все разные, выбираем их по очереди $C_{22}^1 C_{21}^1 C_{20}^1$, или, что то же самое, A_{22}^3 . Следующие 2 вершины тоже отличимы по той же логике, она из них (δ) не связана с той из 3 вершин, которая на стороне β , а другая (ϵ) — не связана с той, которая на стороне γ . Значит выбираем их тоже по очереди, A_{19}^2 . Все остальные вершины тоже становятся отличимыми после выбора β и γ , т.е. их можно выбрать $A_{17}^3 A_{14}^3 A_{11}^2 A_9^2 A_7^2$ или даже A_{17}^{12} . Итого, ответ: $C_{26}^1 C_{25}^2 C_{23}^1 A_{22}^3 A_{19}^2 A_{17}^3 A_{14}^3 A_{11}^2 A_9^2 A_7^2$.

Задача. Сколько существует графов с вершинами из $\{a, \dots, z\}$, изоморфных графу



Решение. Самая левая вершина: C_{26}^1 . Неотличимые вершины α, β : C_{25}^2 . Следующие 3 вершины из за них становятся отличимыми, A_{23}^3 . Пойдем далее по той ветке, которая ближе к младшей из α, β , для определенности, α . Выберем вершину γ : C_{20}^1 . Мы можем пока игнорировать ребра, ведущие в ϵ , поэтому пока что две вершины, соседние с γ — неотличимы: C_{19}^2 . Следующие 2 уже отличимы, потому что одна из них будет связана с младшей из предыдущих 2, а другая — со старшей, A_{17}^2 . Аналогично следующее, A_{15}^2 . Аналогично разбираем другую ветку, с δ : $C_{13}^1 C_{12}^2 A_{10}^2 A_8^2$. Выберем вершину ϵ : C_6^1 . Далее необходимо выбрать, с какой из 2 сторон ветки γ она будет связана (одновременно проводим 3 ребра, два из ϵ и одно диагональное), C_2^1 . Аналогично для ветки δ , C_2^1 . После чего выбираем 2 неотличимых вершины после ϵ , C_5^2 и последние 3 вершины все будут отличимыми, A_3^3 . Итого

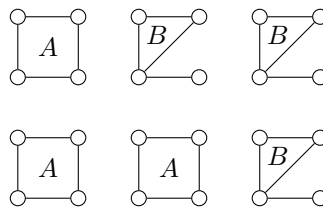
ответом будет $C_{26}^1 C_{25}^2 A_{23}^3 (C_{20}^1 C_{19}^2 A_{17}^2 A_{15}^2) (C_{13}^1 C_{12}^2 A_{10}^2 A_8^2) \cdot C_6^1 C_2^1 C_2^1 \cdot C_5^2 A_3^3$. Заметим, что можно было сразу учесть отличимость 2 вершин после γ , и тогда получим ответ $C_{26}^1 C_{25}^2 A_{23}^3 (C_{20}^1 A_{19}^2 A_{17}^2 A_{15}^2) (C_{13}^1 A_{12}^2 A_{10}^2 A_8^2) \cdot C_6^1 \cdot C_5^2 A_3^3$, тоже являющийся верным.

11.2 Задача 2

Необходимые темы: общие сведения о графах, деревья, изоморфизм графов.

Эта задача уже чуть более сложная, чем прошлая, но все еще не такая, как 5 задача прошлого семестра. Лучше сразу начать с примера:

Задача. Сколько существует связных неориентированных неизоморфных графов, получаемых из заданного графа добавлением минимального числа ребер?



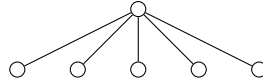
Решение. У нас имеется граф с 6 компонентами связности (частями). 3 из них одного типа, назовем его A , и 3 другого, B . Нам нужно как-то соединить их так, чтобы граф был связным, добавляя минимальное число ребер. Какое минимальное число ребер нужно, чтобы соединить 6 компонент связности? На первом шаге мы можем рассматривать компоненты связности как отдельные вершины. Сколько нужно ребер, чтобы соединить 6 вершин в связный граф? 5, этот граф будет деревом! Раз мы считаем количество таких графов, нам нужно сначала посчитать, сколько существует деревьев с 6 вершинами.

1. Деревья с 6 вершинами.

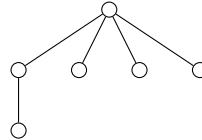
Этот шаг будет в каждом решении 2-й задачи, его можно просто запомнить. Нам нужен какой-то способ классифицировать деревья. Существует 2 таковых: по максимальной степени вершины и по максимальной длине элементарного пути. Здесь я приведу их оба, но в следующих задачах только первый, потому что Константину Ивановичу он нравится больше.

- Классификация по максимальной степени вершины. Какая возможна максимальная степень вершины в графе с 6 вершинами и 5 ребрами? 5, очевидно, все ребра выходят из одной вершины. Начнем же:

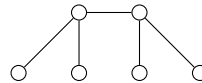
$d = 5$: Существует единственное дерево такого типа:



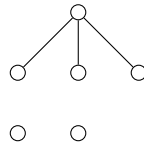
$d = 4$: Тоже единственное, у нас есть вершина степени 4, у нее есть 4 соседа, и еще одна лишняя вершина. Мы не можем подсоединить ее к первой, потому что тогда у нее будет степень 5, значит подсоединяем ее к одному из 4 соседей:



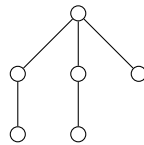
$d = 3$: Тут ситуация становится чуть сложнее. До этого было очевидно, что вершина максимальной степени всего одна, но тут возможно, что их две. Если они не будут соединены напрямую, то всего ребер будет в любом случае 6, а не 5, значит они соединены и у каждой из них есть еще по 2 других соседа:



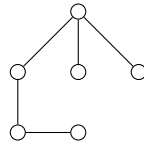
Но возможно, что такая вершина все еще одна, получаем такую ситуацию:



Осталось подсоединить еще 2 вершины. Это можно сделать так:

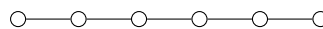


Или так:



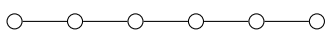
Итого 3 дерева в этом случае

$d = 2$: Последний вариант, тут существует всего одно дерево:

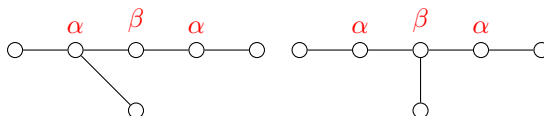


Итого получили 6 разных деревьев.

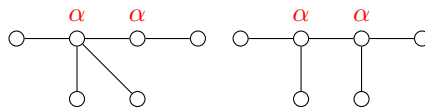
- Классификация по максимальной длине элементарного пути. Какая возможна максимальная длина пути для графа с 5 ребрами? Очевидно, 5:

$l = 5$: 

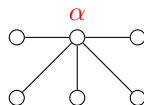
$l = 4$: Нарисуем путь длины 4, останется одна лишняя вершина. Мы можем ее подсоединить или к вершине типа α , или к вершине типа β (к концам нельзя, потому что получим то же, что и в случае $l = 5$), получаем 2 варианта:



$l = 3$: Путь длины 3, остается 2 лишние вершины. Если мы соединим их между собой, а потом к какой-то из вершин основного пути, получим путь длины 4, а так нельзя. Значит всего 2 варианта:



$l = 2$: Единственный вариант:



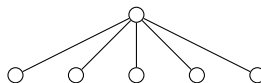
Итого, снова 6 деревьев

Первый шаг закончен, мы узнали, что существует 6 графов с 6 вершинами и 5 ребрами.

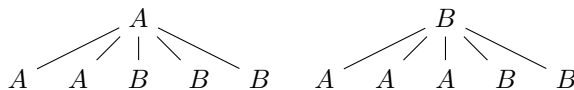
2. Распределение типов компонент.

Вспоминаем, что наши компоненты неоднородны, среди них есть 3 A и 3 B . Нужно как-то распределить эти типы по деревьям и посчитать количество способов распределения. Начнем?

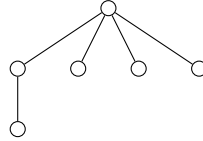
(а) Дерево с $d = 5$:



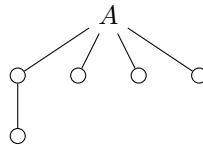
Мы можем дать вершине степени 5 букву A или букву B , все остальные раздать соседям. Так как соседи неотличимы, то существует всего 2 варианта:



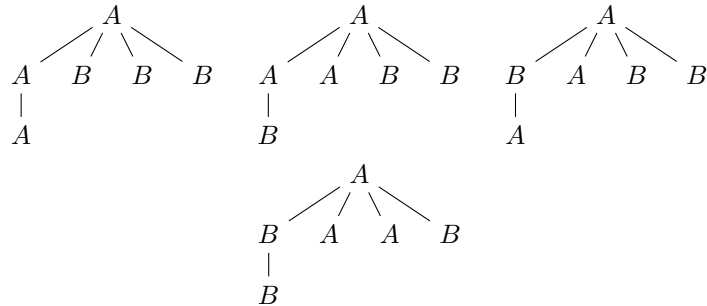
(b) Дерево с $d = 4$:



Воспользуемся полезным фактом, так как у нас поровну A и B , то если мы просто поменяем местами все A и B в раздаче букв, то получим другую легальную раздачу. Т.е. мы можем только разбирать раздачи с верхней вершиной A , а потом умножить на 2, чтобы получить общее число раздач с A или B в верхней вершине. Это и сделаем, сверху A , у нас осталось $AABBB$:

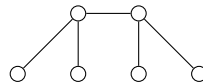


Что может быть в этой ветке из 2 вершин? AA, AB, BA, BB . Каждый из 4 вариантов дает нам по раздаче:

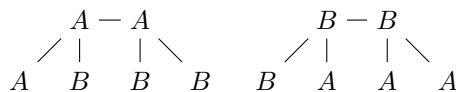


Умножаем на 2, получаем 8 вариантов.

(c) Дерево с 2 вершинами с $d = 3$:

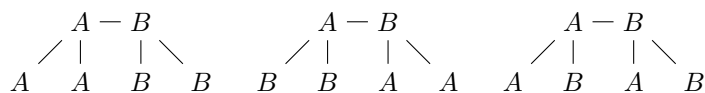


Теперь у нас нет единой вершины с наибольшей степенью, эти 2 надо выбирать одновременно. Возможно 2 варианта, эти 2 вершины $d = 3$ — одного типа, и разного типа. Если одного типа, то есть всего 2 раздачи:



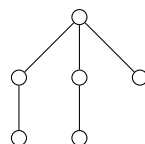
Если же они разные, то у нас возможно 3 случая:

- AA — соседи A , BB — соседи B .
- BB — соседи A , AA — соседи B .
- AB — соседи A , AB — соседи B .

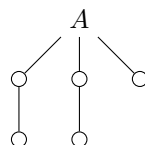


Итого, 5 вариантов.

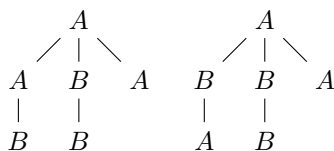
(d) Первое дерево с 1 вершиной $d = 3$:



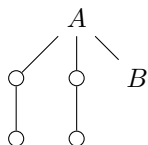
Мы можем снова применить наш трюк с умножением на 2: так что поставим A в верхнюю вершину, и у нас останется $AABBB$.



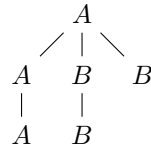
Следующие рассуждения вполне можно представлять в виде дерева решений, как в 5-й задаче первого семестра, но я не буду. Давайте решим, что будет в той одинокой вершине справа, A или B ? Если A , то у нас всего 2 случая:



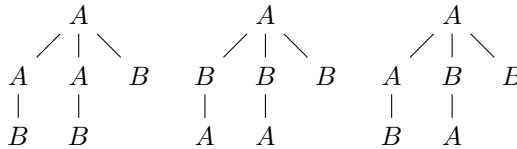
Если же там будет B , то все несколько сложнее. У нас останется $AABB$:



Вопрос, будут ли AA вместе? Если да, получим расстановку

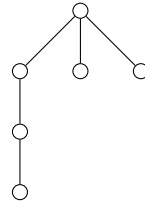


Если нет, то возможно 3 расстановки: $AB + AB$, $BA + BA$, $AB + BA$:

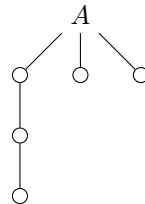


Всего 6 расстановок, но мы умножаем на 2, так что всего 12 вариантов.

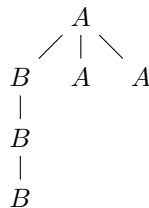
(е) Второе дерево с 1 вершиной $d = 3$:



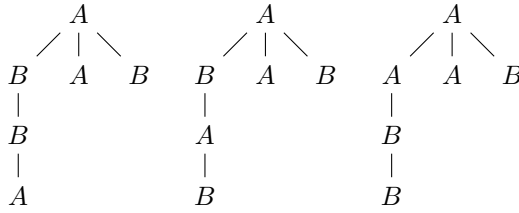
Аналогично, установим в верхнюю вершину A , у нас останется $AABBB$, а потом умножим на 2:



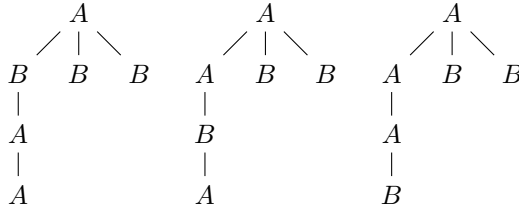
Сколько B будет в левой ветке? Если все 3, то 1 вариант:



Если 2, то 3 варианта:

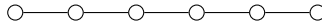


Если 1, то тоже 3 варианта:



Итого 7 вариантов, умножаем на 2, *14 вариантов*.

(f) Дерево с вершинами $d = 2$ (цепочка)



Если мы будем считать его как обычно, то очень быстро устанем, здесь много почти одинаковых вариантов. Но заметим, что этот граф — просто цепочка, поэтому имеет смысл просто посчитать количество слов из 3 букв A и 3 букв B : просто выберем 3 позиции под A из 6, это даст нам C_6^3 . Тут правда есть небольшой нюанс, слова $ABBAVA$ и $ABABBA$ — разные, но графы

$A - B - B - A - B - A$

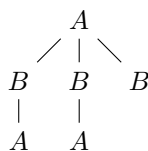
$A - B - A - B - B - A$

— один и тот же граф, просто перевернутый. Значит каждому графу соответствует *2 слова*, его прочтение в одну сторону и в другую. Значит чтобы получить настоящее число графов нам нужно поделить число слов на 2: $C_6^3/2 = 20/2 = 10$. Итого, *10 вариантов*.

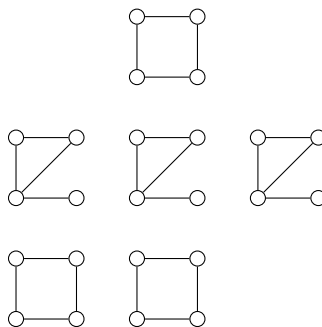
Если объединить все вместе, получим *51 вариант* размещения букв $AAABBB$ по дереву с 6 вершинами.

3. Соединение компонент связности.

Разумеется мы не будем рассматривать все 51 вариант, достаточно 2. Давайте сначала рассмотрим вариант



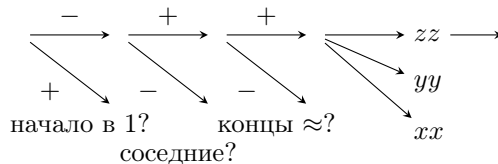
Вернем компоненты связности на место букв, получим



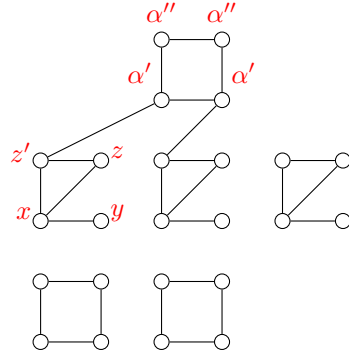
Вот тут нам точно придется строить дерево, как в 5 задаче. Итак, начнем проводить ребра. Ветки BA абсолютно неотличимы, поэтому мы не можем себе позволить выбрать их по очереди, нам придется проводить туда ребра одновременно. Этим и займемся, ребра $A-BB$. Вопрос, эти ребра начинаются из одной точки в A ? Пусть нет. Если они разные, то они соседние или противоположные? Пусть соседние. Так как все вершины в A неотличимы, нам не нужно выбирать, какие именно это пары. Рассмотрим повнимательнее компоненту связности B : в ней есть 3 типа вершин, назовем их x, y, z



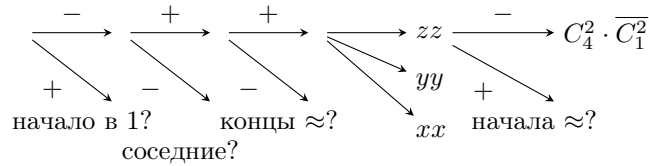
Вопрос, наши ребра заканчиваются в вершинах одного типа или нет? Пусть одного. Какого? Это xx , yy или zz ? Пусть zz . Итого, дерево выбора на данный момент будет



А граф будет выглядеть как



Заметьте, что типы вершин поменялись, теперь в верхнем A у нас есть 2 типа, α' и α'' , а в обеих B все 4 вершины стали разных типов. Идем дальше. Ветки все еще неотличимы, так что ребра $B - A, B - A$ нам придется снова проводить вместе. Ну и ладно. Начала из одного типа вершин? Нет, пусть наши ветки наконец то уже станут разными. Хорошо. Какие это типы? Нам на самом деле это уже не так важно, мы больше не будем работать с этими ветками, мы можем себе позволить терять информацию. Поэтому мы можем их выбрать просто как C_4^2 . Концы однотипные, нам их даже выбирать не нужно, но можно и записать, $\overline{C_1^2}$, выбираем 2 вершины из вершин одного типа (возможны повторы). Итак, дерево и граф:



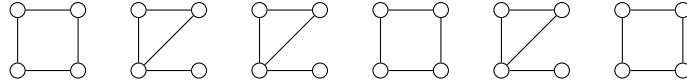
Заметьте, что все 4 вершины верхнего A стали разные, так как теперь они связаны с ветками разной структуры. Осталось связать это

с последней компонентой B , но так как это последний шаг, это можно сделать просто — выбрать 1 из 4 типов для A и 1 из 3 типов для B : $C_4^1 C_3^1$. На этом это решение закончено.

Попробуем аналогично разобрать граф

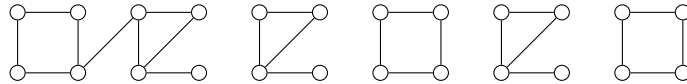
$$A - B - B - A - B - A$$

или



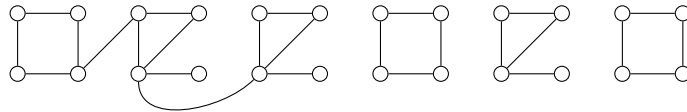
Начнем разбирать слева направо, первое ребро, AB . Начало выбираем просто C_1^1 , а с концом сложнее, у нас будет разное число типов в B , если мы выберем z или x, y . Ветвление! Выберем z .

$$\begin{array}{c} C_1^1 \xrightarrow{z} \\ \searrow xy \end{array}$$

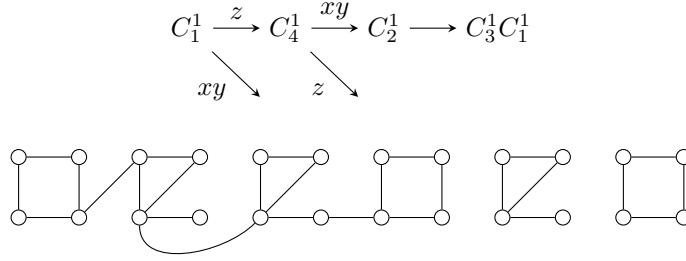


Второе ребро, начало. У нас 4 типа, и так как мы уже никогда не вернемся к этой компоненте, нам не важно, что именно мы сейчас выберем, так что C_4^1 . И снова ветвление из-за конца, но давайте на этот раз выберем xy . Так что именно, x или y ? Не важно, одно из двух, C_2^1 :

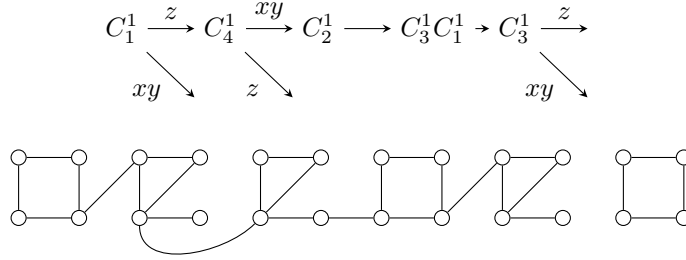
$$\begin{array}{ccccc} C_1^1 & \xrightarrow{z} & C_4^1 & \xrightarrow{xy} & C_2^1 \\ & \searrow xy & & \searrow z & \end{array}$$



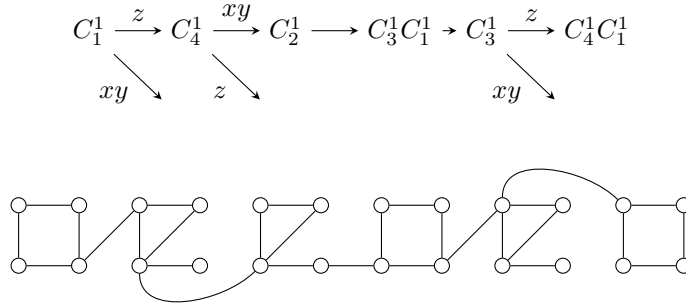
Третье ребро, начало: C_3^1 , потому что z все еще неотличимы. Конец: C_1^1 .



Четвертое ребро, начало: C_3^1 , потому что теперь у нас 3 типа: вершина, куда мы провели ребро, ее соседи, и одна противоположная вершина. Конец — обычное ветвление, выберем z .



Пятое ребро, начало: C_4^1 , конец: C_1^1 . На этом все.



Значительные отличия могут быть только во 2 части задачи. 1 часть полностью идентична всегда и везде, а 3 часть хоть и различается, но общий смысл один и тот же. Отличия в основном 2, если количества наших типов не равны, например $AAABBC$, то мы не сможем использовать наш трюк с умножением на 2, нам придется рассматривать все полностью.

Если же всех типов четное число, например $AABVCC$, то важное отличие появляется в подсчете числа расстановок для «цепочки». Дело в том, что появляются симметричные графы типа $ABCCBA$, которые независимо

от направления дают одно и то же слово. Давайте посчитаем, сколько всего таких слов. В нем 3 буквы, ABC , должны быть как-то расположены в одной половине графа, а другие ABC будут единственным (симметричным) способом располагаться в другой половине. Всего способов расставить ABC в некотором порядке $A_3^3 = 6$, т.е. у нас есть 6 симметричных слов. Сколько всего слов? Мы сначала выбираем 2 позиции под A , а потом 2 позиции под B , и потом оставшиеся 2 позиции остаются C : $C_6^2 C_4^2 C_2^2 = 15 \cdot 6 \cdot 1 = 90$. Значит несимметричных слов у нас $90 - 6 = 84$. Несимметричные слова образуют в 2 раза меньше графов, чем их самих, т.е. $84/2 = 42$. Вместе с симметричными словами, у нас получается ровно $42 + 6 = 48$ расстановок.

11.3 Задача 3

Необходимые темы: общие сведения о графах, критерий планарности, изоморфизм графов.

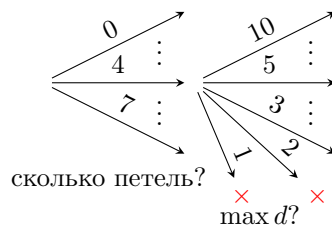
Судя по всему, самая сложная задача этого семестра. Как и в 5, есть простой вариант задачи, не появляющийся в зачете, но необходимый в объяснении, потому что все остальные являются усложнениями этого варианта.

Задача. Сколько существует неизоморфных неориентированных связных графов, содержащих 11 вершин и 17 ребер?

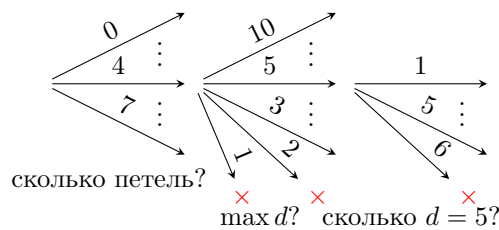
Решение. Да, вот такое вот короткое условие дает очень сложную задачу. Для начала, нам не сказано, что этот граф — граф без петель, поэтому петли тут могут быть. Вопрос, сколько? Их может и не быть, т.е. минимум — 0. А сколько максимум? Сколько нам нужно ребер для обеспечения связности графа из 11 вершин? 10, тогда граф — дерево. В таком случае остается 7 ребер под петли, это максимум. Итого петель от 0 до 7, мы выбираем что-то посередине, это 4. Итого у нас будет 4 петли и 13 обычных ребер.



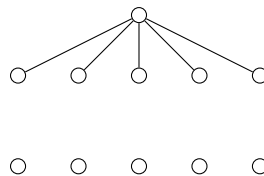
Далее, нам нужна максимальная степень вершины (не считая петель). Максимальная, очевидно, 10, мы можем подсоединить к одной вершине 10 остальных и получим вершину степени 10. А минимум? Чисто теоретически — 1. Практически — 3. Как нам это доказать? Ну во первых, граф с максимальной степенью 1 — просто куча несвязанных отрезков, он не может быть связным, если вершин больше 2. А что с максимальной степенью 2? Если все вершины степени 2, то граф образует один большой цикл, т.е. если в нем 11 вершин, то и ребер будет ровно 11. Если же там есть вершины степени 1, то их будет 2 и это будет просто цепочка, в которой ровно 10 ребер. А нам нужно 13, значит так нельзя. Выберем $d = 5$.



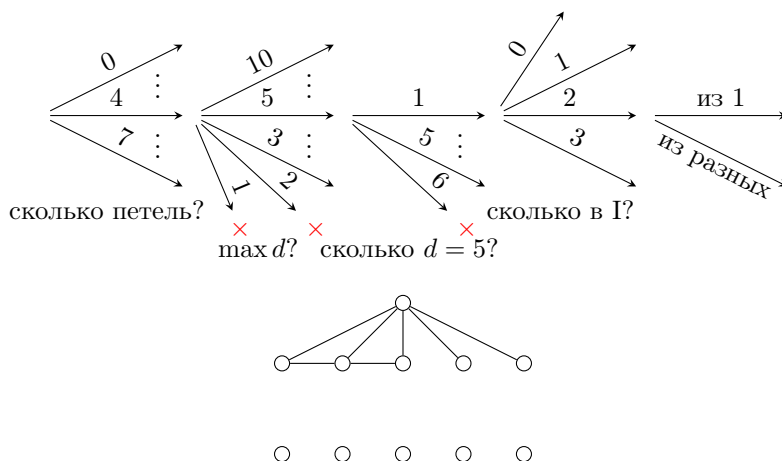
Вопрос, сколько таковых вершин с $d = 5$? Минимум, очевидно, 1. Максимум? Сложнее. Можно чисто комбинаторно посчитать, например, для 6. Если у нас есть 6 вершин степени 5, то из них вместе выходит $6 \cdot 5 = 30$ ребер. Эти ребра могут соединять как раз наши вершины степени 5, тогда число ребер делится пополам: $30/2 = 15$. Все еще больше чем 13, невозможно. А для 5 такой вариант уже не сработает, значит максимум — 5. Но мы выберем 1.



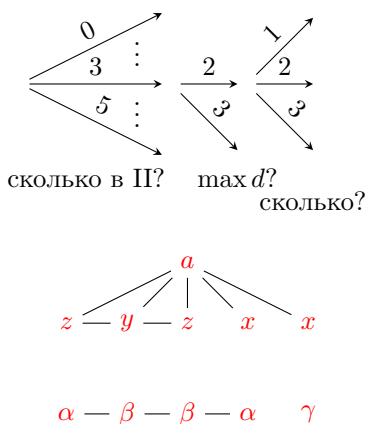
Итого, наш граф выглядит пока так:



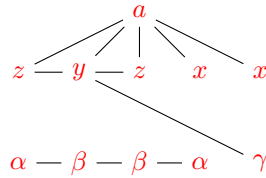
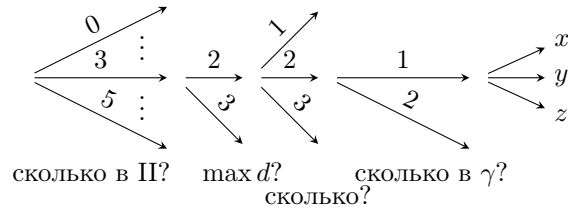
Назовем 5 точек, соседей вершины с $d = 5$, как множество I, и не связанные пока с графом 5 вершин как II. Мы провели уже 5 ребер из 13, осталось 8. Для обеспечения связности нам придется провести хотя бы 5 ребер, т.е. у нас осталось 3 «свободных» ребра, которые можно проводить где угодно. Сколько всего будет ребер между вершинами внутри I? От 0 до 3, выберем 2. Эти 2 ребра будут выходить одной вершины, или они будут соединять несвязанные пары? Давайте из одной. Тогда получим



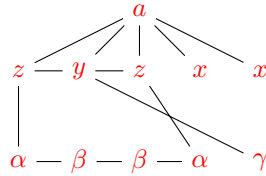
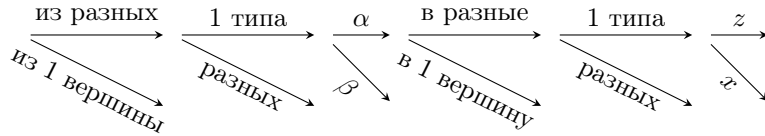
Сколько ребер в II? Эти ребра тоже будут помогать в связности, так что мы можем использовать все 6 оставшихся ребер. Точнее, 5, потому что хотя бы 1 ребро все равно должно соединять I и II. Давайте выберем 3. Какая будет максимальная степень вершины? Или 2, или 3, пусть будет 2. Сколько таких будет? Или 1 (цепочки $\circ - \circ - \circ$ и $\circ - \circ$), или 2 (цепочка $\circ - \circ - \circ - \circ$), или 3 (цикл из 3 вершин). Пусть будет 2.



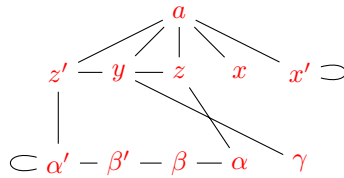
На рисунке каждой вершине сопоставлено название типа, одного из $a, x, y, z, \alpha, \beta, \gamma$. Они нам еще пригодятся. Осталось провести еще 3 ребра между I и II. Давайте сначала разберемся с γ . Сколько ребер соединяет ее с I? Это может быть или 1, или 2. 0 не может быть, иначе γ не связана с основным графом. 3 не может быть, иначе $\alpha\beta\beta\alpha$ не связано с основным графом. Выберем 1. С каким типом вершины из I она связана? Пусть будет с y . Тогда zz и xx останутся неотличимыми.



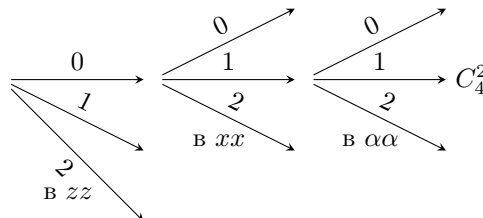
Итак, осталось 2 ребра, и они оба соединяют $\alpha\beta\beta\alpha$ с Γ . Эти 2 ребра в $\alpha\beta\beta\alpha$ начинаются с одной вершины или с разных? Пусть будет с разных. Эти 2 вершины одного типа или нет? Пусть будут одного. Это будет тип α или β ? Пусть будет α . Концы в Γ — это одна вершина? Пусть нет. Это 2 вершины одного типа? Пусть да. Какого, xx или zz ? Пусть будет zz . Закончили на этом с ребрами.



Остались петли, 4 штуки. Было бы хорошо, если бы мы могли их просто C_{11}^4 распределить, но нет, у нас тут есть неотличимости, а неотличимости надо выбирать вместе. Сколько петель будет в zz ? Или 0, или 1, или 2. Пусть будет 0. Сколько в xx ? Пусть будет 1. Это делает xx отличимыми, но на этом все. Сколько в $\alpha\alpha$? Давайте тоже 1. Это делает не только $\alpha\alpha$ отличимыми, но и zz , и, что важно, $\beta\beta$.



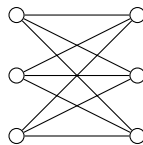
Осталось 2 петли и 4 вершины, про которые мы еще не спрашивали: a, y, β, β' . Так как все они — отличимы, мы можем просто сделать C_4^2 . Итого дерево для выбора петель будет



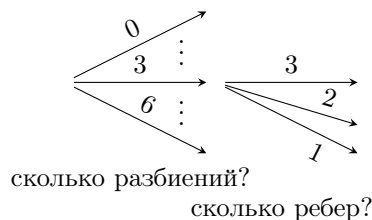
А теперь к усложнениям. Бывают усложнения типа «граф не содержит элементарных циклов длины 6». За этим надо просто следить в процессе построения и исключать ветки, которые их создают. Чуть сложнее с «непланарными графами»

Задача. Сколько существует изоморфных не планарных связных графов, состоящих из 12 вершин и 19 ребер, содержащих разбиение графа $K_{3,3}$ и не содержащих петель и соседних вершин степени 3.

Решение. Для начала, мы уже знаем, что граф содержит $K_{3,3}$, т.е. нам стоит начать с рисования $K_{3,3}$. Еще он не содержит петель, значит все ребра — реальные ребра.

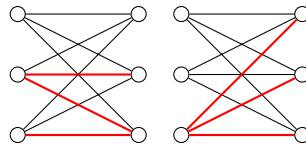


Это сам $K_{3,3}$, а нам нужно некоторое разбиение $K_{3,3}$. Нам нужно разбить сколько-то ребер, сколько? Вполне может быть, что 0, а может быть и 6. 7 нельзя, потому что при разбиении ребра у нас добавляется вершина, 6 вершин всего есть, а максимум может быть 12. Итак, сколько их будет? Давайте 3. Эти 3 разбиения разбивают 1 ребро, 2 или 3? (возможно, что мы просто 3 раза разбили одно и то же ребро $K_{3,3}$). Давайте все 3.

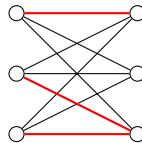


Вопрос, какие это 3 ребра? Их надо выбрать. Причем мы не можем просто выбрать C_9^3 , потому что все ребра разные. И мы не можем их выбирать

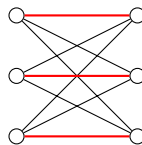
по порядку. И мы не можем даже задавать начала и концы этих ребер. Что же мы можем вообще задавать? Взаимное расположение этих ребер! Давайте рассмотрим подграф $K_{3,3}$, состоящий только из этих 3 ребер (уберем все остальные). Сколько у него компонент связности? Или он связный (1 компонента) и все 3 ребра как-то связаны вместе, или в нем 2 компоненты, одна из 2 ребер, и другая из 1, или в нем 3 компоненты и все 3 ребра не связаны? Давайте по быстрому рассмотрим все варианты, а потом выберем 1. Если у нас 1 компонента связности, то 3 ребра могут или образовывать цепочку, или все выходить из 1 вершины:



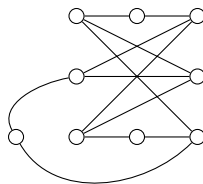
Если 2 компоненты, то 2 ребра в одной компоненте выходят из 1 вершины, и еще одно ребро соединяет еще 2 другие вершины



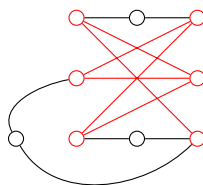
Если же это 3 компоненты, то это 3 абсолютно не связанных ребра:



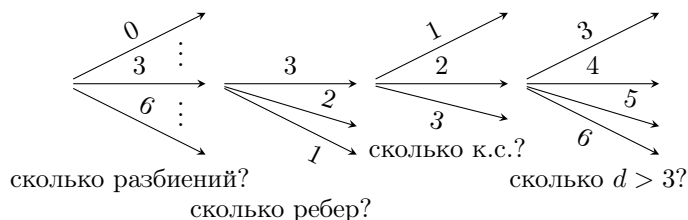
Давайте выберем 2 вариант, и сделаем разбиение этих 3 ребер:



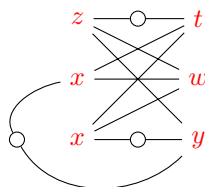
У нас есть еще одно условие, у нас не должно быть соседних вершин степени 3. Давайте выделим все вершины степени 3 и ребра, соединяющие соседние вершины::



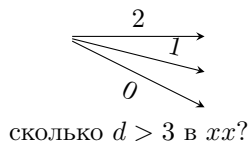
сколько-то из этих вершин должно перестать быть степени 3 (например, к ним добавится 1 ребро и они станут степени 4), тогда это условие будет выполнено. Минимум при таком раскладе — 3, максимум — очевидно, 6. Выберем 4.

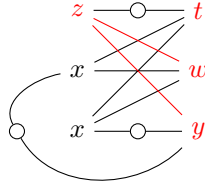


Какие это вершины? Для начала выделим типы этих 6 вершин:

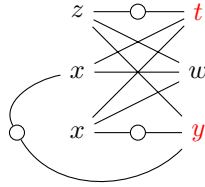
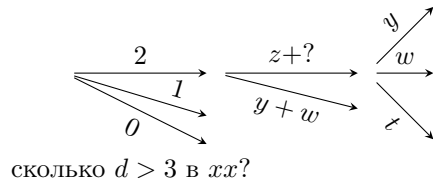


Распределение достаточно простое: y соединено с двумя неотличимыми x теми ребрами, которые мы разбили, z соединено с t , причем z на стороне xx , а t на стороне y , и есть еще одна вершина, из которой не выходят те ребра, которые мы разбивали, w . Для начала решим, сколько из xx находятся среди тех 4, у которых мы будем делать степени > 3 ? Пусть будут обе

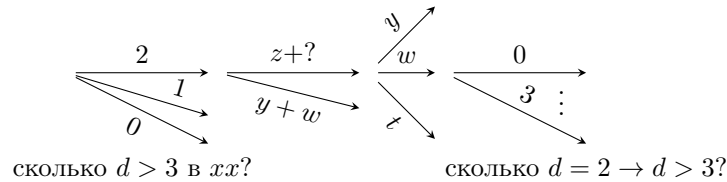




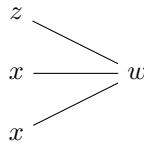
Вершинами с $d = 3$ остались z, t, y, w , причем теперь мы обязаны выбрать или z и что-то еще, или wy . Что именно? Давайте z и что-то еще. Что еще, y, w или $t? w$.



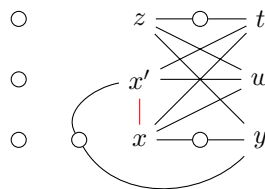
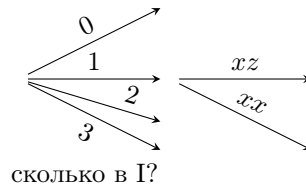
Красные вершины остаются степени 3, мы к ним больше ничего не можем подцеплять. А вот к $xxzw$ мы обязаны подцепить хотя бы 1 ребро. Если же мы будем подцеплять что-то к o , то нужно подцеплять сразу 2 ребра. Сколько у нас вообще ребер осталось? Мы использовали 9 для $K_{3,3}$ и при разбиении добавили еще 3, итого 12. Осталось 7. Достаточно много. А вершин? 3. Не так много. Давайте назовем $xxzw$ — I, а 3 не подсоединенных пока вершины II. Итого 7 ребер могут быть внутри I, внутри II или соединять I и II. Только тут еще одно, есть 3 вершины степени 2 и можно проводить ребра еще из них. Но давайте это исключим:



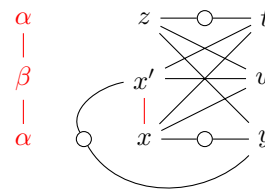
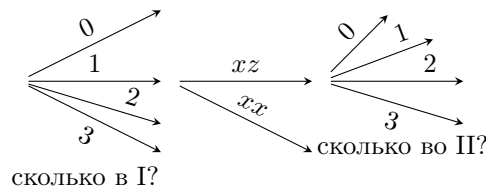
Итак, сколько ребер может быть внутри I? Для начала посмотрим, как выглядит граф, если мы нарисуем только $xxuw$:



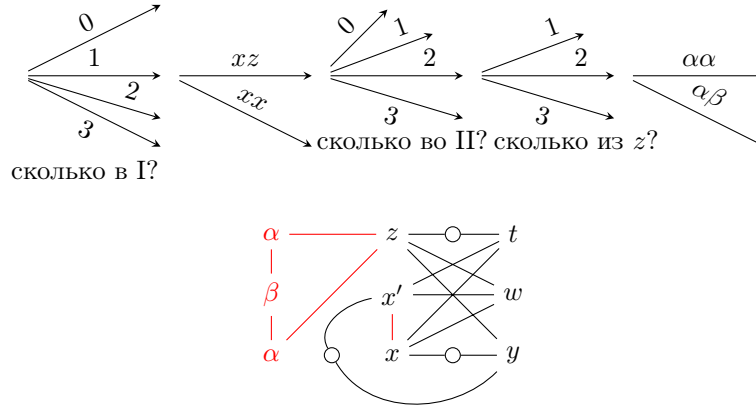
До полного графа нам не хватает только 3 ребер, двух xz и одного xx . Сколько из них будем проводить? От 0 до 3. А конкретно? Пусть будет 1. Какое, xx или xz ? Пусть xz , теперь x -ы у нас отличимы. Окей, это сделали



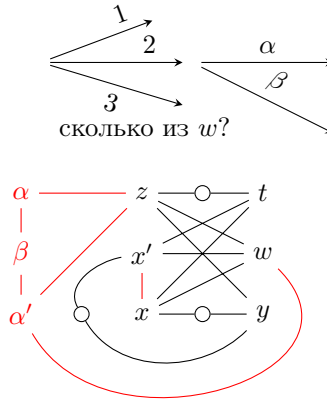
Окей, сколько будет ребер в II? Минимум, очевидно, 0, максимум — 3, там больше некуда. Давайте выберем 2. В 3 вершинах их можно провести только одним способом.



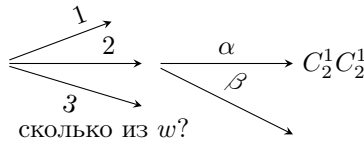
Нам осталось провести 4 ребра между I и II. Причем мы обязаны провести хотя бы по 1 ребру из z и w . Давайте их и рассмотрим. Для начала, сколько ребер может быть из z ? Оттуда может идти даже 3 ребра (одно надо оставить для w). Выберем 2. Эти ребра ведут в $\alpha\alpha$ или в $\alpha\beta$? Пусть в $\alpha\alpha$, они останутся неотличимыми.

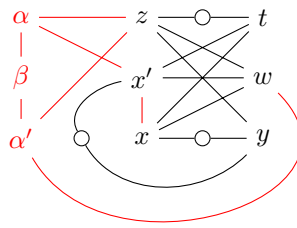


Осталось 2 ребра. Хотя бы одно из них должно быть из w . Одно или оба? Одно. А куда оно пойдет, в α или β ? Пусть α , это делает их отличимыми.



Осталось ровно 1 ребро и на этом наше построение будет закончено. Откуда оно идет? Из x или x' . C_2^1 . А куда? В β оно вести не может, иначе α' и β станут соседними вершинами степени 3. А вот в α или α' может. C_2^1 .





11.4 Задача 4

Необходимые темы: числовые функции, вычисляемые автоматами (раздел 6.2 на с. 111).

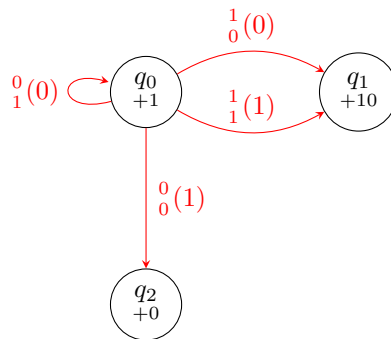
А теперь перешли снова к простым задачам. Нам здесь дана числовая функция формата $f(x, y) = 3x - 2y - 2$, и нам нужно нарисовать диаграмму переходов автомата, который это считает. Для этого мы придерживаемся определенного принципа — каждое состояние автомата отвечает за некоторый перенос. Отрицательный перенос — это заем, как в отрицании. Мы должны вычислить то, что получается у автомата при всех 4 комбинациях (x, y) для каждого из состояний, выделить то, что автомат напечатает сейчас и то, что должен будет запомнить. В соответствии с этим делаем переходы. Насчет константы, которая прибавляется к функции — просто считаем, что это изначальный перенос, т.е. перенос, который соответствует q_0 . Так как автомат работает в двоичной системе, все расчеты и записи будут тоже в двоичной системе, хотя двоичное сложение и вычитание можно для удобства переводить в десятичное и делать сложение/вычитание в уме, а потом переводить обратно. Так вот, приступим сначала к простой задаче без вычитания:

Задача. Составить диаграмму переходов автомата, вычисляющего функцию $f(x) = 3x + y + 1$.

Решение. Начнем. Для начала, переносом q_0 будет +1. Посчитаем то, что у нас будет получаться при (x, y) равных 00, 01, 10, 11:

$$\begin{array}{rclcl}
 + & 1 & + & 1 & + & 1 \\
 + & 0 & + & 0 & + & 1 & 1 \\
 + & 0 & + & 1 & + & 0 & \\
 \hline
 & 1 & \textcolor{red}{1} & 0 & \textcolor{red}{1} & \textcolor{red}{0} & 0 & \textcolor{red}{1} & \textcolor{red}{0} & 1
 \end{array}$$

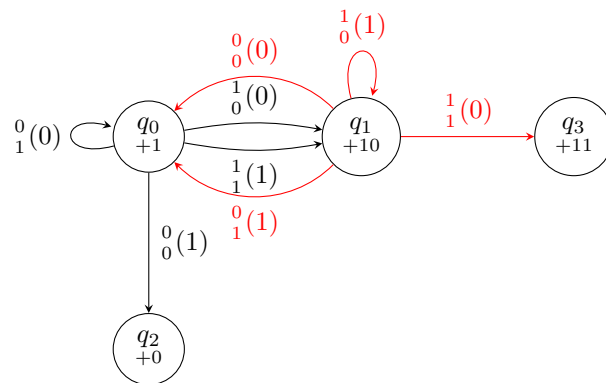
Т.е. при входе 00 автомат печатает 1 и переходит в состояние +0, при входе 01 автомат печатает 0 и остается в +1, при 10 он печатает 0 и переходит в +10, при 11 он печатает 1 и переходит в +10. 2 новых состояния:



Далее, разберем q_1 с переносом +10:

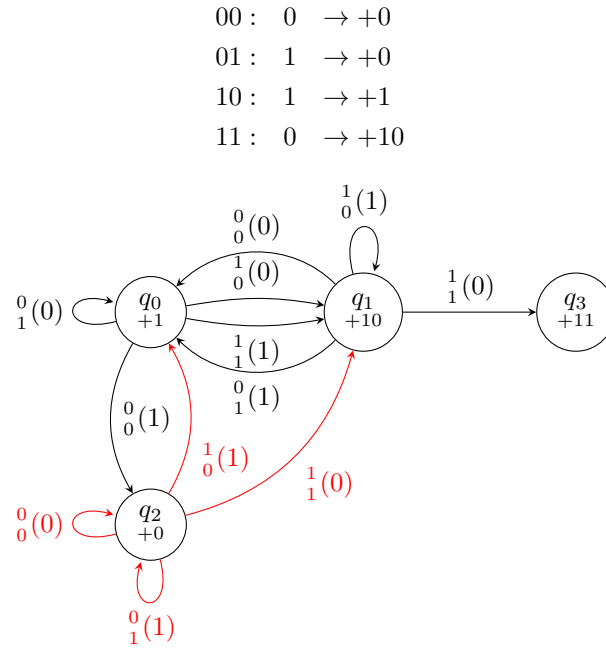
+	1	0	+	1	0	+	1	0	+	1	0
+		0	+		0	+	1	1	+	1	1
+		0	+		1	+		0	+		1
<hr/>			<hr/>			<hr/>			<hr/>		
	1	0		1	1		1	0		1	0

00 :	0	→ +1
01 :	1	→ +1
10 :	1	→ +10
11 :	0	→ +11



Состояние q_2 , с переносом +0

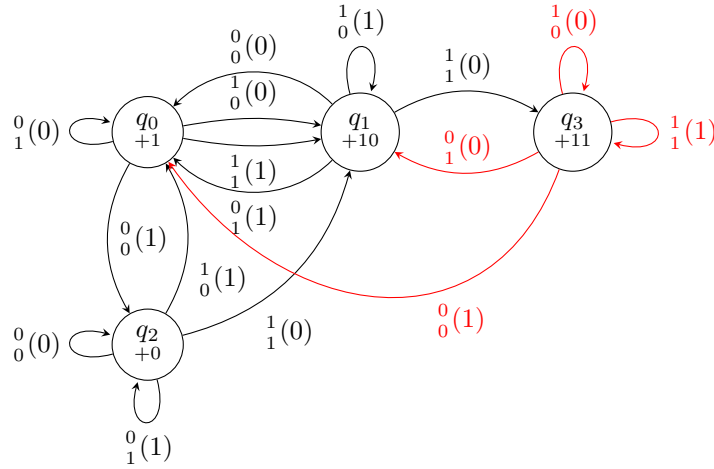
+	0	+	0	+	0	+	0
+	0	+	0	+	1	+	1
+	0	+	1	+	0	+	1
<hr/>		<hr/>		<hr/>		<hr/>	
	0		1		1		0



И состояние q_3 , с переносом +11

+	1	1	+	1	1	+	1	1	+	1	1
+		0	+		0	+	1	1	+	1	1
+		0	+		1	+		0	+		1
<hr/>			<hr/>			<hr/>			<hr/>		
	1	1		1	0		1	1		1	1

$00 : 1 \rightarrow +1$
 $01 : 0 \rightarrow +10$
 $10 : 0 \rightarrow +11$
 $11 : 1 \rightarrow +11$



Простая задача, не правда ли? К сожалению, на зачете таких нет.

Задача. Составить диаграмму переходов автомата, вычисляющего функцию $f(x) = 1x - 5y - 2$.

Решение. Да здравствует вычитание... Начнем с q_0 с переносом $-2_{10} = -10_2$, как обычно

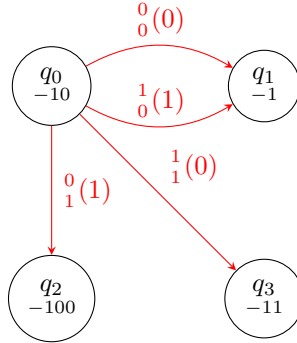
$$\begin{array}{r}
 -10 \\
 +0 \\
 \hline
 -10
 \end{array}
 \begin{array}{r}
 -10 \\
 +0 \\
 \hline
 -10
 \end{array}
 \begin{array}{r}
 -10 \\
 +1 \\
 \hline
 -9
 \end{array}
 \begin{array}{r}
 -10 \\
 +1 \\
 \hline
 -9
 \end{array}$$

Сейчас у вас вероятно вопросы, как такое вообще произошло, да? Для начала, результат этого вычитания — *не* обычный результат вычитания, перенос и самый правый разряд — отдельные, поэтому запятая. Второе, здесь отрицательный только перенос, не самый правый разряд, к нему минус не относится. Третье, как это получилось? Первое, третье и четвертое относительно понятны, а вот со вторым проблемы. $-10 + 0 - 101 = -100, 1$. Смотрим справа налево, по разрядам. Сначала нам нужно сделать $-0 + 0 - 1 = -1$, но наш конечный автомат не умеет печатать -1 , он печатает или 0, или +1. Поэтому делаем заем из левого разряда: оттуда вычитается 1, а к правому мы «прибавляем» 10 (точнее 2, мы в двоичной системе). Итак, справа получается $+10 - 0 + 0 - 1 = 1$, все хорошо. Что слева? У нас там было -1 и -10 , мы еще сделали заем, стало $-1 - 1 - 10 = -100$. Это и осталось. Вот такое вот вычитание.

Впрочем, есть еще один способ считать это, придуманный мной. Просто считаем ответы в обычной десятичной системе (тут получаем $-2, -7, -1, -6$), если оно четное, то пишем справа 0, а слева пишем это же, деленное на 2 (разве что переводим в двоичную систему). Здесь у нас такие -2 и -6 , делим их на 2: $-1, -3$, и записываем в двоичной: $-1, -11$. Если же нечетное, то справа пишем 1, а слева *вычитаем* 1, а потом делим на 2. После

вычитания у нас получается $-8, -2$, после деления $-4, -1$, что в двоичной системе будет $-100, -1$. Итого мы получили $-1, 0; -100, 1; -1, 1; -11, 0$. Все работает, правда? Далее я буду писать оба способа расчетов.

Итак, q_0 . Тут получается, что создаются целых 3 новых состояния, q_1 с переносом -1 , q_2 с переносом -100 и q_3 с переносом -11 .



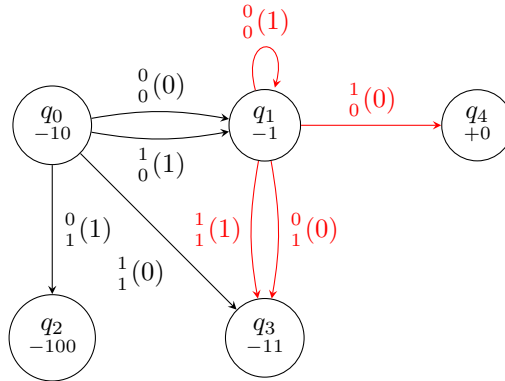
Далее, q_1 с переносом -1 .

$$\begin{array}{r}
 -1 \quad - \quad 1 \quad - \quad 1 \quad - \quad 1 \\
 +0 \quad + \quad 0 \quad + \quad 1 \quad + \quad 1 \\
 -0 \quad -1 \quad 0 \quad 1 \quad -0 \quad -1 \quad 0 \quad 1 \\
 \hline
 -1, 1 \quad -1, 1, 0 \quad 0 \quad -1, 1, 1
 \end{array}$$

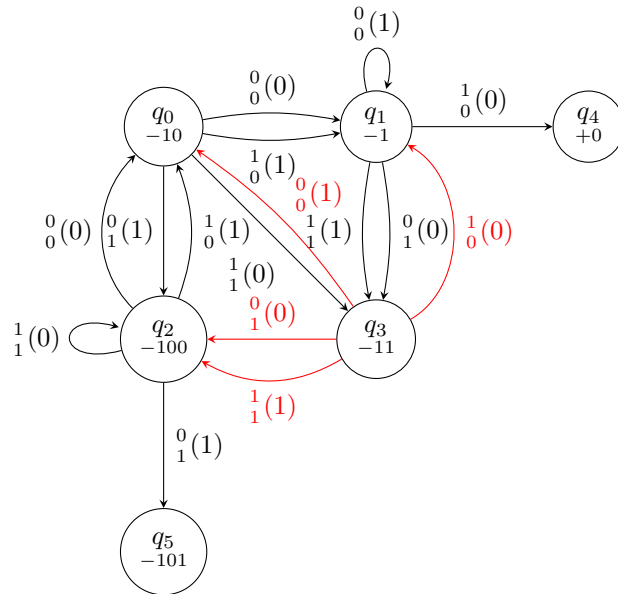
В первом мы просто делаем 1 заем и получается -1 справа и $10 - 1 = 1$ слева. Во втором аналогично, в третьем вообще все хорошо. В четвертом, хоть это и не так легко понять, мы тоже делаем просто 1 заем, как и во втором. Вторым способом, это можно рассчитать как

$$\begin{aligned}
 -1+0-0 &= -1 \rightarrow -2/2, 1 \rightarrow -1, 1 \\
 -1+0-5 &= -6 \rightarrow -6/2, 0 \rightarrow -11, 0 \\
 -1+1-0 &= 0 \rightarrow 0/2, 0 \rightarrow 0, 0 \\
 -1+1-5 &= -5 \rightarrow -6/2, 1 \rightarrow -11, 1
 \end{aligned}$$

Новое состояние, q_4 с переносом $+0$



Новых состояний не возникло, уау!



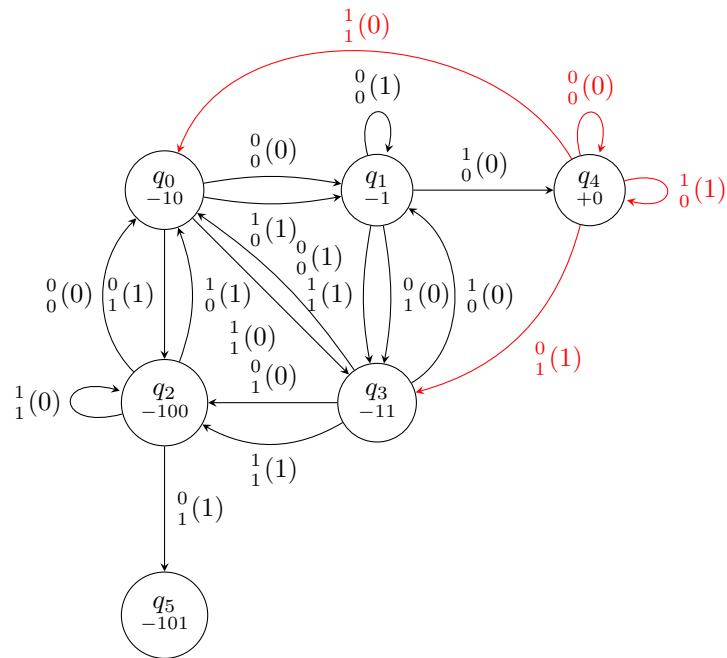
Далее, q_4 с переносом $+0$

$\begin{array}{r} + 0 \\ + 0 \\ - 0 \\ \hline 0 \end{array}$	$\begin{array}{r} + \\ + \\ - 1 \\ \hline - 1 \end{array}$	$\begin{array}{r} 0 \\ 0 \\ 0 \\ \hline 1, 1 \end{array}$	$\begin{array}{r} + 0 \\ + 1 \\ - 0 \\ \hline 1 \end{array}$	$\begin{array}{r} + \\ + \\ - 1 \\ \hline - 1 \end{array}$	$\begin{array}{r} 0 \\ 1 \\ 0 \\ \hline 0, 0 \end{array}$
--	--	---	--	--	---

Или

$$\begin{aligned}
 0+0-0 &= 0 \rightarrow 0/2, 0 \rightarrow 0, 0 \\
 0+0-5 &=-5 \rightarrow -6/2, 1 \rightarrow -11, 1 \\
 0+1-0 &= 1 \rightarrow 0/2, 1 \rightarrow 0, 1 \\
 0+1-5 &=-4 \rightarrow -4/2, 0 \rightarrow -10, 0
 \end{aligned}$$

Новых состояний снова нет:

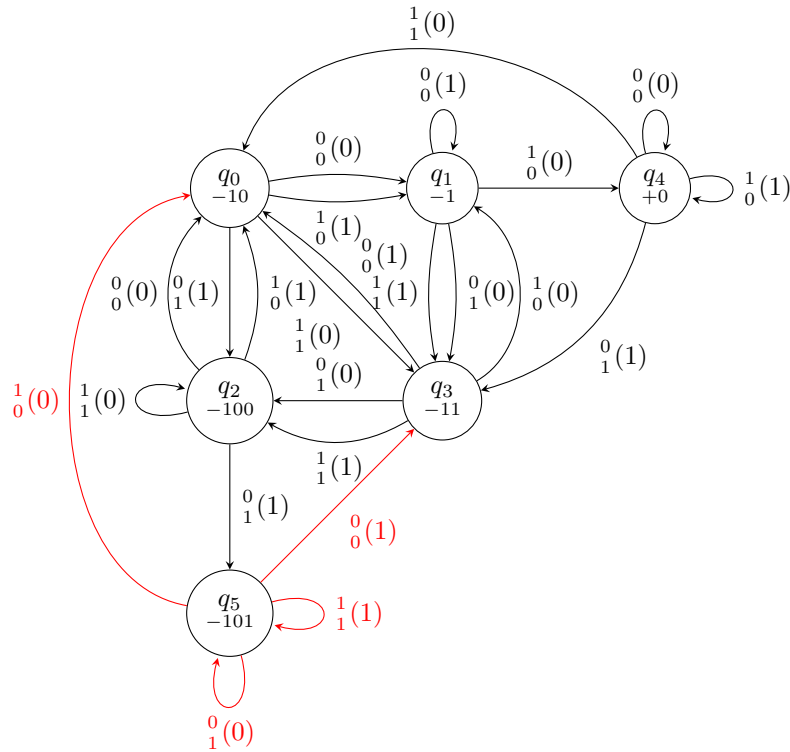


И последнее, q_5 с переносом -101

-	1	0	1	-	1	0	1	-	1	0	1	-	1	0	1
+			0	+			0	+			1	+			1
-			0	-	1	0	1	-			0	-	1	0	1
<hr/>				<hr/>				<hr/>				<hr/>			
-	1	1,	1	-	1	0	1, 0	-	1	0,	0	-	1	0	1, 1

Или

$$\begin{aligned}
 -5+0-0 &= -5 \rightarrow -6/2, 1 \rightarrow -11, 1 \\
 -5+0-5 &= -10 \rightarrow -10/2, 0 \rightarrow -101, 0 \\
 -5+1-0 &= -4 \rightarrow -4/2, 0 \rightarrow -10, 0 \\
 -5+1-5 &= -9 \rightarrow -10/2, 1 \rightarrow -101, 1
 \end{aligned}$$



11.5 Задача 5

Необходимые темы: автоматы, распознающие слова (раздел 6.6 на с. 123).

Достаточная простая задача, хоть и муторная, как только поймешь, как именно работает язык регулярных выражений. Для начала, мы работаем со словами из алфавита $\{0, 1, S\}$. Обозначение $\{A\}^*$ означает нечто A , повторяющееся неизвестное число раз, или, возможно, пустое слово. $[A]^*$ тоже означает повторы A , но хотя бы 1 раз A должно встречаться. Например, $\{0, 1\}^*$ означает любую строку из 0 и 1, возможно пустую, а $[0, 1]^*$ — любую *непустую* строку из 0 и 1. Так же, $\{01\}^*$ означает строку из 01, например 010101, возможно пустую, а $[01]^*$ означает то же самое, но пустые строки запрещены. Еще в заданиях встречается не очень понятное обозначение S^* , но оно, судя по всему, эквивалентно $[S]^*$. Вот и весь язык регулярных выражений.

Пара слов о том, как именно автоматы распознают слова, а точнее, как создавать такие автоматы. Дело в том, что автомат никогда (почти) не может точно знать, где он сейчас находится внутри распознаваемого слова. Более того, некоторые слова разрешают несколько верных вариантов

интерпретации. Возьмем как пример слова вида $\{0\}^*\{1\}^*[01, 0011]^*$. Рассмотрим слово 010101. Можно считать, что это 3 повтора 01 или что это 1 повтор 0, потом 1 повтор 1 и 2 повтора 01. Или слово 00110011, это может быть или просто 2 повтора 0011, или сначала 2 повтора 0, 2 повтора 1 и 1 повтор 0011. В то же время, мы не можем выкинуть ни один компонент из этой записи, ведь должны распознаться в том числе и слова 000000000001 или 111111111101, или даже 0000001111110100110101. Поэтому автомат, не зная, где находится, должен рассматривать все варианты одновременно. Если он видит, что какой-то из вариантов невозможен, то он его вычеркивает, если он видит, что новый символ можно трактовать по-разному, он создает новые варианты и так далее.

Теперь, собственно к задаче:

Задача. Построить диаграмму переходов автомата (15 состояний), распознающего слова вида

$$[0S, 1]^*\{0, SS\}^*[S]^*\{S, S0\}^*[01]^*[S1, S]^*\{S, S0\}^*[0, 11]^*$$

Решение. Во первых, нам не нужно строить диаграмму переходов полностью, она очень большая, достаточно 15, при условии, что мы добрались хотя бы до 1 распознающего состояния. И да, это 15 полностью разобранных, не просто присутствующих на диаграмме. Для начала, отметим важные точки в нашем слове:

$$_0[0S, 1]_1^*\{0, SS\}_2^*[S]_3^*\{S, S0\}_4^*[01]_5^*[S1, S]_6^*\{S, S0\}_7^*[0, 11]_8^*$$

Это — именно точки, это не индексы. Т.е. 0 — мы находимся еще в начале строки, 1 — мы распознали первую часть (или все еще находимся внутри нее, но пока не знаем об этом), 2 — мы распознали вторую часть (или внутри нее) и т.д. Есть еще одно обозначение для положений *внутри* таких групп, например, $\frac{S|0}{4}$ — мы прочитали первый символ из группы $S0$ в четвертой части.

Определимся с правилами переходов. Они достаточно простые — если этот символ может встречаться в текущем месте слова, то автомат оказывается в точке после этого символа, все логично. Чуть сложнее с пустыми словами, но тоже не очень сложно: если автомат после чтения символа оказывается *перед* частью, которая может быть пустой (в фигурных скобках), то мы приписываем еще и точку *после* этой возможно пустой части. Дело в том, что автомат никогда не может знать точно, где именно он сейчас находится в слове, и это — один из тех случаев. Или он все таки встретит эту часть, или она была пустой и он должен сразу находиться в точке после? Неизвестно, поэтому мы рассматриваем все варианты. Еще одно уточнение, если автомат находится *перед* возможно пустой частью, он может читать сразу символы из следующих частей, хотя это и никогда не будет сказываться на конкретном результате.

Каждому состоянию автомата соответствует набор точек слова, где автомат может сейчас находиться. Начальное состояние q_0 соответствует точке 0. Если мы когда-нибудь оказываемся в ситуации, что автомат читает

символ, который ну никак не может быть в текущей точке, мы переходим в особое состояние q_T , «ловушку», которое не распознающее и автомат всегда остается в нем. Буквально, оно значит, что слово уже точно не соответствует искомому и независимо от того, что дальше, мы никогда не попадем в распознающее состояние.

Константин Иванович для каждого состояния считал переходы отдельно, я же предлагаю рассчитать все возможные переходы между частями сразу, без задержек, записать в таблицу, а потом пользоваться.

Итак, точка 0. Если мы читаем 0, то это явно кусочек $0S$, так что мы оказываемся в точке $\frac{0|S}{1}$. Если мы читаем 1, то мы оказываемся в точке 1, т.к. единицы в части 1 идут по отдельности и эта самая первая часть может закончиться сразу после этой 1. Помним про то, что вторая часть — возможно пустая, и мы можем оказаться и в точке 2 после этого. Что же будет, если мы встретим S ? Тогда это не то слово, которое вы ищете. Мы ставим прочерк в таблице. В обозначениях Костенко это выглядит так:

	0	1	S
0	$\frac{0 S}{1}$	1, 2	—

Точка $\frac{0|S}{1}$. Оттуда мы можем прочесть только S (и оказаться в точке 1 или 2), все остальное — «ловушка»:

	0	1	S
0	$\frac{0 S}{1}$	1, 2	—
$\frac{0 S}{1}$	—	—	1, 2

Точка 1 чуть сложнее. Если мы читаем 0 — это новый повтор $0S$, или мы же прочитали 0 из части 2 и теперь переходим в точку 2? Неизвестно. Поэтому мы пишем оба варианта: $\frac{0|S}{1}$ и 2. Чтение 1 — обязательно повтор первой части, так что мы остаемся в той же точке. Чтение S — или начало SS , или S из 3-й части, после которой мы оказываемся или в точке 3, или в точке 4 (благодаря пропуску возможно пустой 4-й части):

	0	1	S
0	$\frac{0 S}{1}$	1, 2	—
$\frac{0 S}{1}$	—	—	1, 2
1	$\frac{0 S}{1}, 2$	1, 2	$\frac{S S}{2}, 3, 4$

Из точки $\frac{S|S}{2}$ мы можем прочесть только S :

	0	1	S
0	$\frac{0 S}{1}$	1, 2	—
$\frac{0 S}{1}$	—	—	1, 2
1	$\frac{0 S}{1}, 2$	1, 2	$\frac{S S}{2}, 3, 4$
$\frac{S S}{2}$	—	—	2

Из точки 2 мы можем прочитать 0 (повтор 2 части) и остаться там же, или S , и оказаться или в $\frac{S|S}{2}$, или в 3, но еще и в 4, т.к. $\{S, S0\}$ может оказаться пустым:

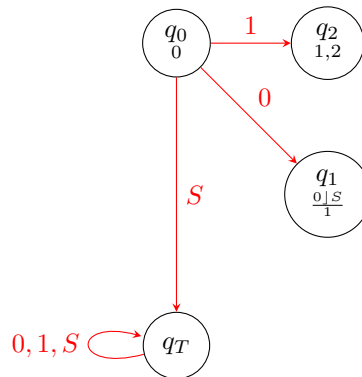
	0	1	S
0	$\frac{0 S}{1}$	1, 2	—
$\frac{0 S}{1}$	—	—	1, 2
1	$\frac{0 S}{1}, 2$	1, 2	$\frac{S S}{2}, 3, 4$
$\frac{S S}{2}$	—	—	2
2	2	—	$\frac{S S}{2}, 3, 4$

Остальная часть таблицы заполняется аналогично:

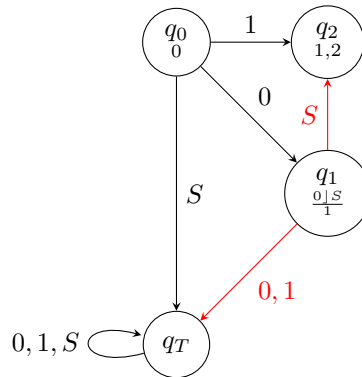
	0	1	S
0	$\frac{0 S}{1}$	1, 2	—
$\frac{0 S}{1}$	—	—	1, 2
1	$\frac{0 S}{1}, 2$	1, 2	$\frac{S S}{2}, 3, 4$
$\frac{S S}{2}$	—	—	2
2	2	—	$\frac{S S}{2}, 3, 4$
3	$\frac{0 1}{5}$	—	3, $\frac{S 0}{4}, 4$
$\frac{S 0}{4}$	4	—	—
4	$\frac{0 1}{5}$	—	$\frac{S 0}{4}, 4$
$\frac{0 1}{5}$	—	5	—
5	$\frac{0 1}{5}$	—	$\frac{S 1}{6}, 6, 7$
$\frac{S 1}{6}$	—	6, 7	—
6	8	$\frac{1 1}{8}$	$\frac{S 1}{6}, 6, 7, \frac{S 0}{7}$
$\frac{S 0}{7}$	7	—	—
7	8	$\frac{1 1}{8}$	7, $\frac{S 0}{7}$
$\frac{1 1}{8}$	—	8	—
8	8	$\frac{1 1}{8}$	—

На самом деле не обязательно заполнять всю таблицу сразу полностью, можно заполнять ее по мере необходимости. Важно просто понимать, что эти значения уникальны для каждой точки слова, но не для каждого состояния автомата, достаточно вычислить их только 1 раз.

Итак, перейдем собственно к построению диаграммы переходов. Изначально автомат находится в состоянии q_0 , которая соответствует точке 0. Если автомат читает 0, то он оказывается в точке $\frac{0|S}{1}$ (смотрим в таблицу), пусть это будет состояние q_1 . Если он читает 1, то он оказывается или в точке 1, или в точке 2. Пусть состояние, отвечающее за набор вариантов 1, 2, будет q_2 . Если же автомат прочитает S в этот момент, то слово заведомо неподходящее, так что мы переходим в ловушку q_T :



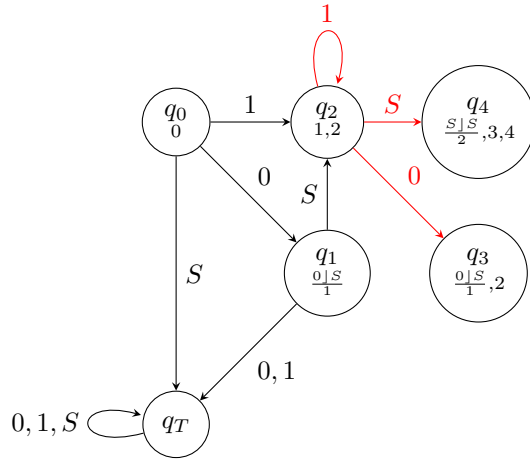
Из состояния q_1 он перейдет в состояние q_2 , если прочтает S , и в состояние q_T , если прочтает 0 или 1:



Состояние q_2 чуть сложнее, в нем автомат может находиться или в точке 1, или в точке 2. Выпишем соответствующие строки таблицы:

q_2	0	1	S
1	$\frac{0 S}{1}, 2$	1, 2	$\frac{S S}{2}, 3, 4$
2	2	—	$\frac{S S}{2}, 3, 4$

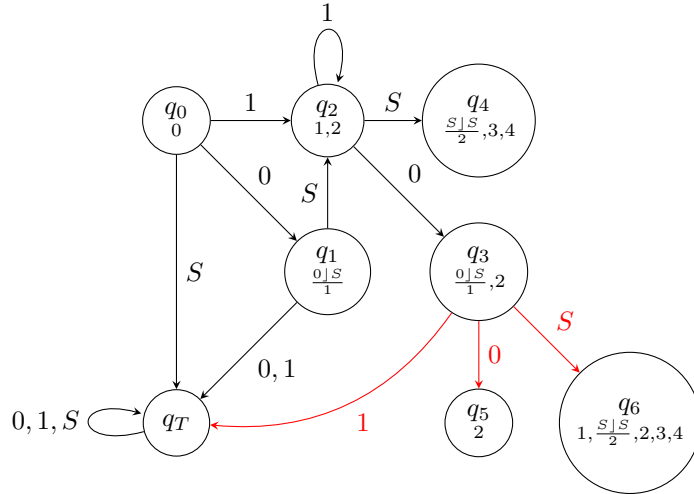
И объединим все в столбиках и получим, что при чтении 0 автомат переходит в набор точек $\frac{0|S}{1}, 2$ (новое состояние, q_3), при чтении 1: остается в 1, 2 (q_2), при чтении S попадает в $\frac{S|S}{2}, 3, 4$ (новое состояние, q_4):



Для q_3 поступаем аналогично:

q_3	0	1	S
$\frac{0 \downarrow S}{1}$	—	—	1, 2
2	2	—	$\frac{S \downarrow S}{2}, 3, 4$

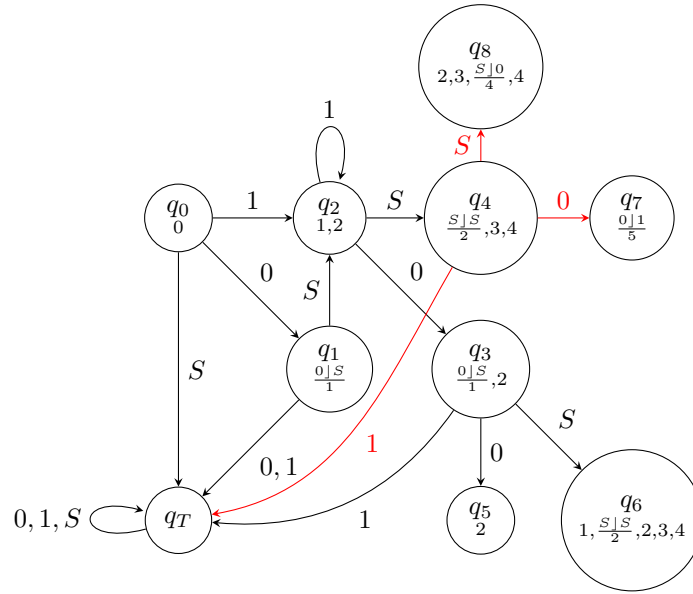
Итого, на 0 получаем 2, это новое состояние, q_5 , на 1 переходим в q_T , на S получаем аж $1, \frac{S \downarrow S}{2}, 2, 3, 4$, это q_6 .



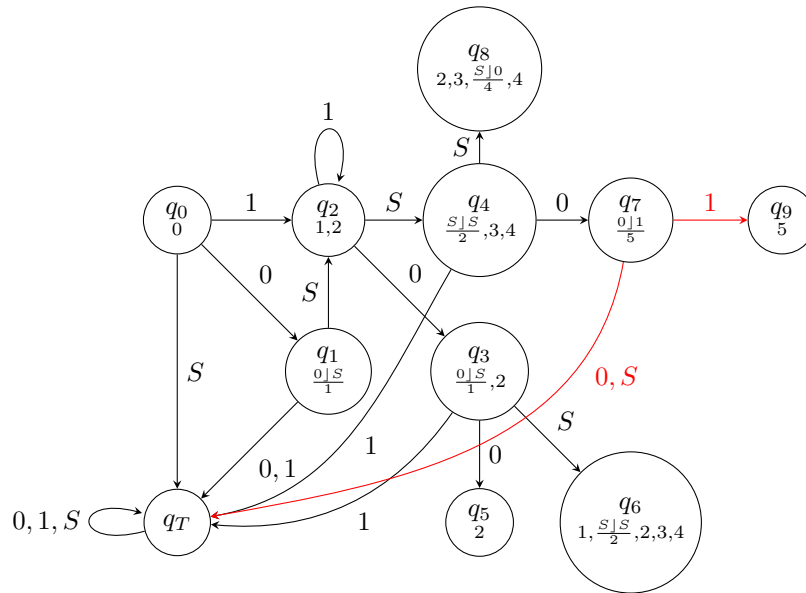
Мы уже разобрали 4 состояния, но сильно ближе к концу не стали. Давайте разбирать далее те, которые продвинулись дальше всего, а только потом доработать старые: пока q_4 .

q_4	0	1	S
$\frac{S S}{2}$	—	—	2
3	$\frac{0 1}{5}$	—	$3, \frac{S 0}{4}, 4$
4	$\frac{0 1}{5}$	—	$\frac{S 0}{4}, 4$

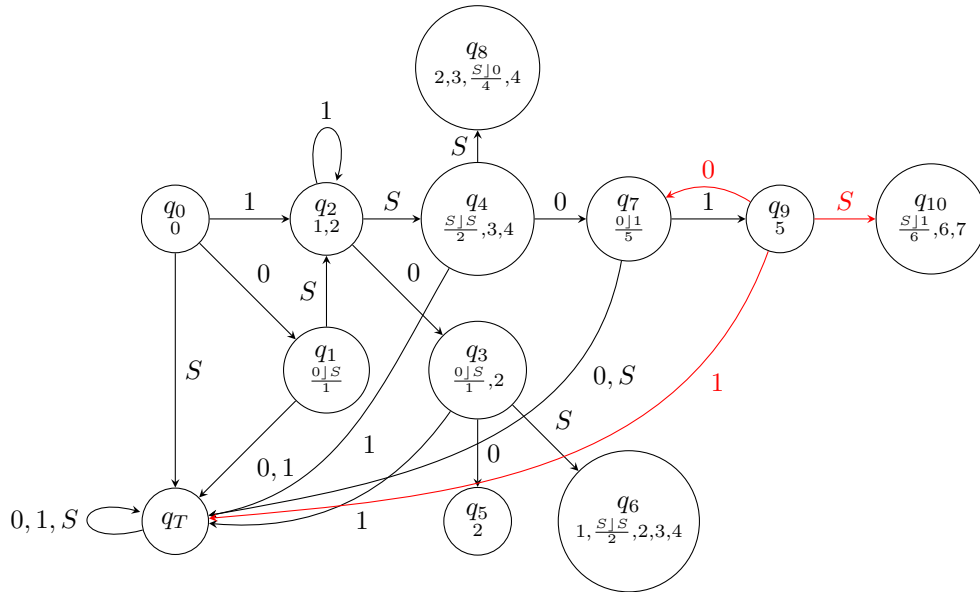
При чтении 0 мы попадаем в $\frac{0|1}{5}$, это q_7 , 1 — сразу в q_T , S — $2, 3, \frac{S|0}{4}, 4$ — q_8 .



5 состояний разобрали, q_7 самая прогрессивная, рассмотрим ее. Тут только один вариант, в 5 (q_9) если 1, и в q_T в противном случае:



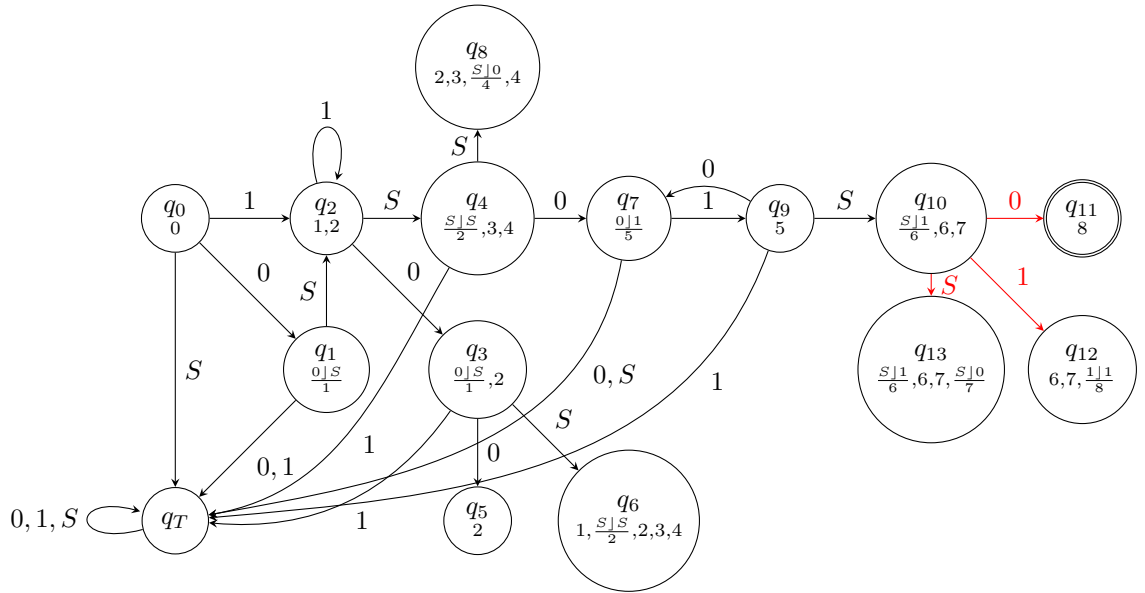
6 состояний. Из q_9 мы возвращаемся обратно в q_7 при 0. При 1 — в «ловушку». При S — в $\frac{S|1}{6}, 6, 7$, в q_{10} .



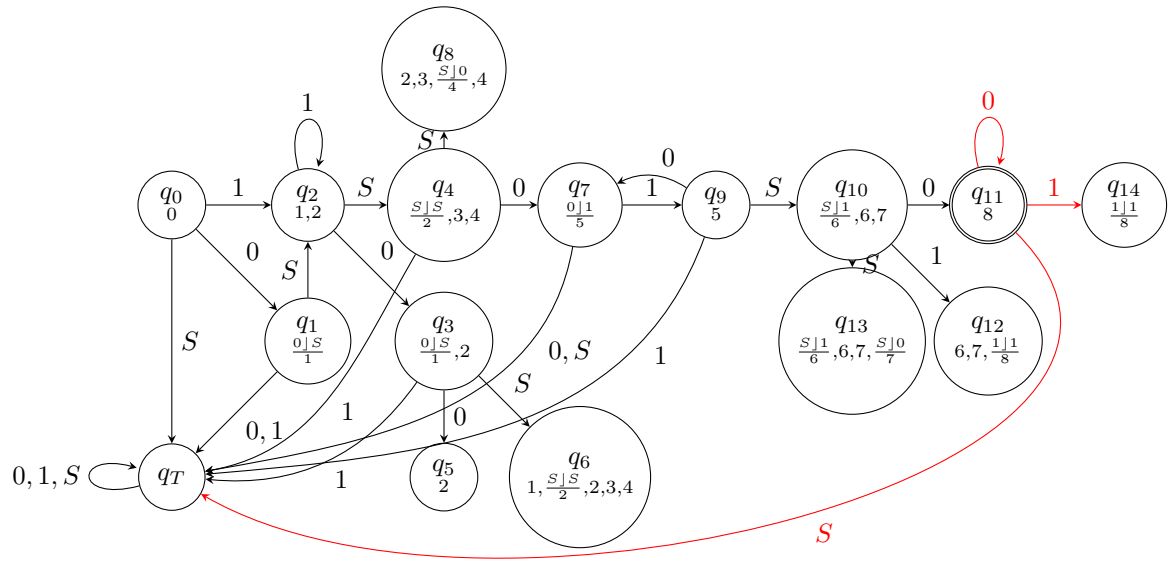
7 состояний, q_{10} :

q_{10}	0	1	S
$\frac{S 1}{6}$	—	6, 7	—
6	8	$\frac{1 1}{8}$	$\frac{S 1}{6}, 6, 7, \frac{S 0}{7}$
7	8	$\frac{1 1}{8}$	$7, \frac{S 0}{7}$

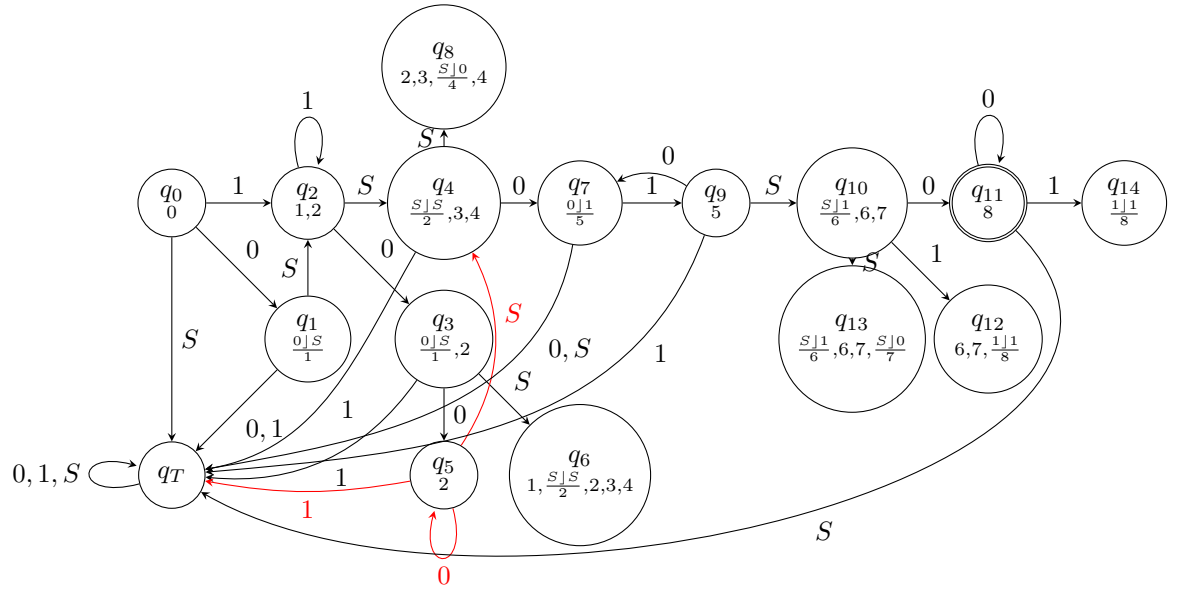
После 0 автомат окажется в 8, q_{11} . Это последняя точка в слове, и если оно закончится сейчас, то оно будет подходить, т.е. q_{11} — распознающее состояние. На самом деле, любое состояние, в котором один из вариантов «трактовки» оказывается в конце слова, будет распознающим. Если 1, то мы оказываемся в 6, 7, $\frac{1|1}{8}$ (q_{12}), если S — в $\frac{S|1}{6}, 6, 7, \frac{S|0}{7}$ (q_{13}).



8 состояний. Разберем q_{11} : если получим 0, то останемся там же, если 1, то попадем в $\frac{1|1}{8}$, q_{14} , если S — уйдем в q_T , потому что его в конце слова быть не должно.



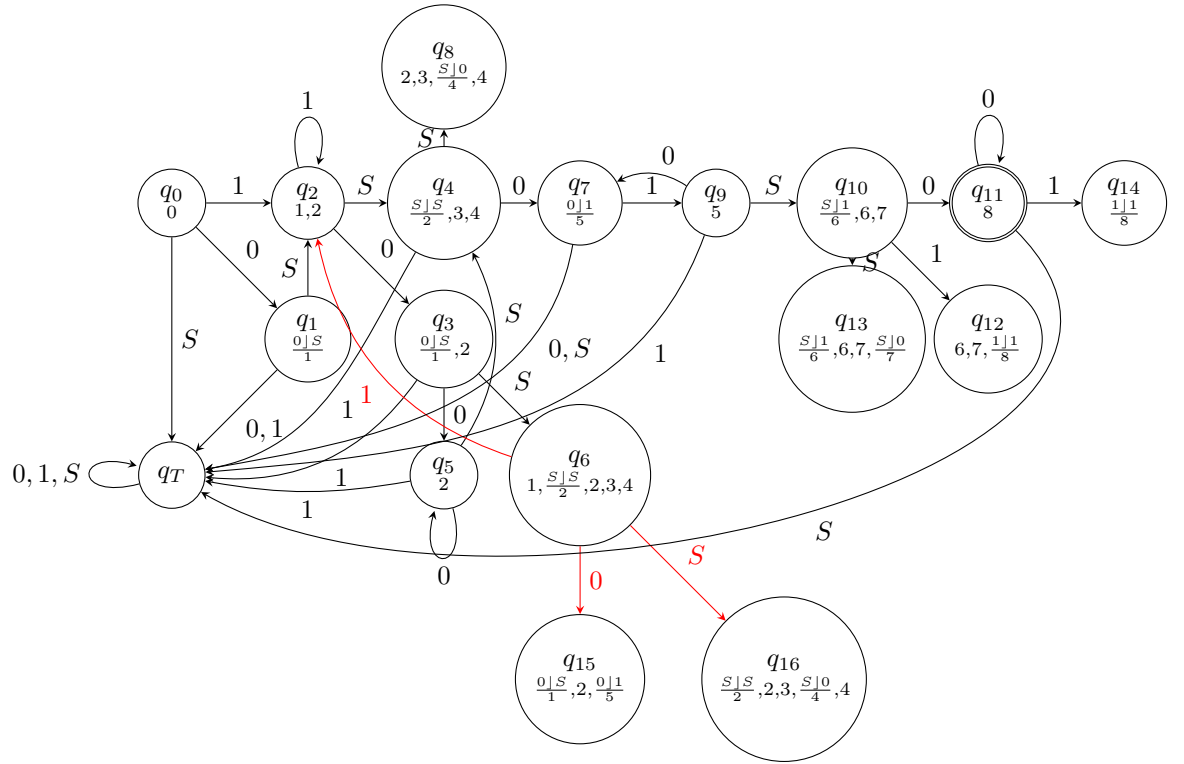
Итак, 9 состояний, осталось еще 6 любых. Давайте разбирать все не разобранные по порядку, например, q_5 . Мы останемся там же при 0, перейдем в q_T при 1 и в q_4 при S :



10 состояний, разберем сложное q_6 :

q_6	0	1	S
1	$\frac{0 S}{1}, 2$	1, 2	$\frac{S S}{2}, 3, 4$
$\frac{S S}{2}$	—	—	2
2	2	—	$\frac{S S}{2}, 3, 4$
3	$\frac{0 1}{5}$	—	$3, \frac{S 0}{4}, 4$
4	$\frac{0 1}{5}$	—	$\frac{S 0}{4}, 4$

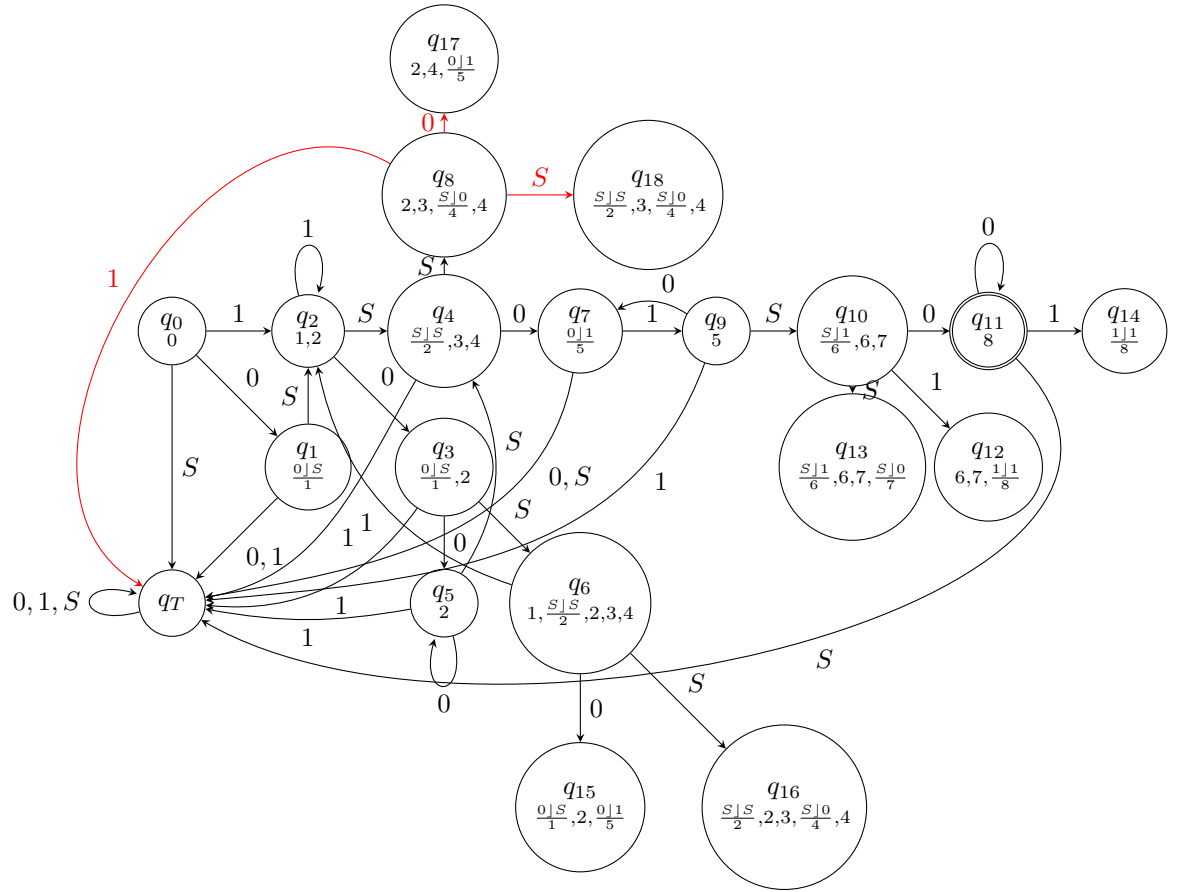
При 0 переходим в $\frac{0|S}{1}, 2, \frac{0|1}{5}$ (q_{15}), при 1 в q_2 , при S переходим в $\frac{S|S}{2}, 2, 3, \frac{S|0}{4}, 4$ (q_{16}).



11 состояний, q_8 ?

q_8	0	1	S
2	2	—	$\frac{S S}{2}, 3, 4$
3	$\frac{0 1}{5}$	—	$3, \frac{S 0}{4}, 4$
$\frac{S 0}{4}$	4	—	—
4	$\frac{0 1}{5}$	—	$\frac{S 0}{4}, 4$

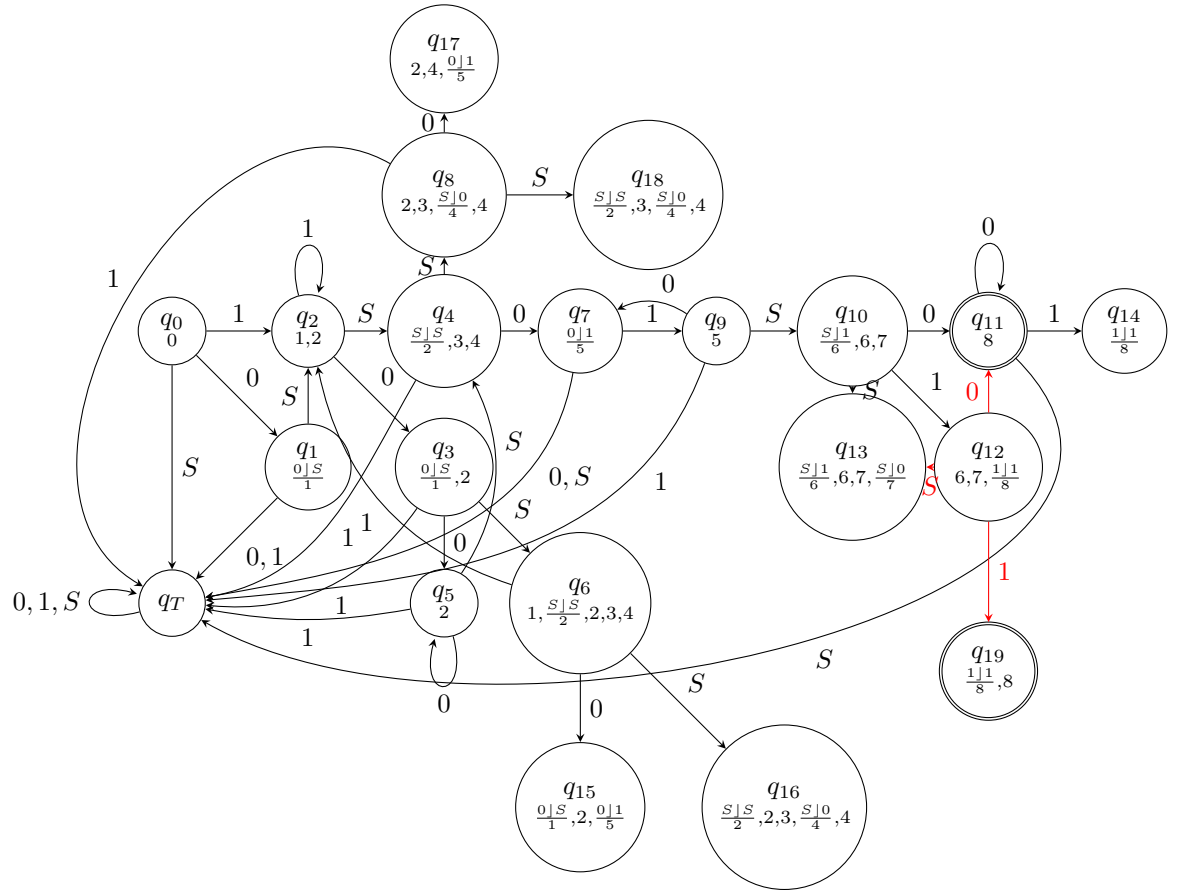
0: 2, 4, $\frac{0|1}{5}$ (q_{17}). 1: q_T . S : $\frac{S|S}{2}, 3, \frac{S|0}{4}, 4$ — нет, это не то же самое, что q_{16} , это q_{18} .



12 состояний. q_{12} ?

q_{12}	0	1	S
6	8	$\frac{1 1}{8}$	$\frac{S 1}{6}, 6, 7, \frac{S 0}{7}$
7	8	$\frac{1 1}{8}$	$7, \frac{S 0}{7}$
$\frac{1 1}{8}$	—	8	—

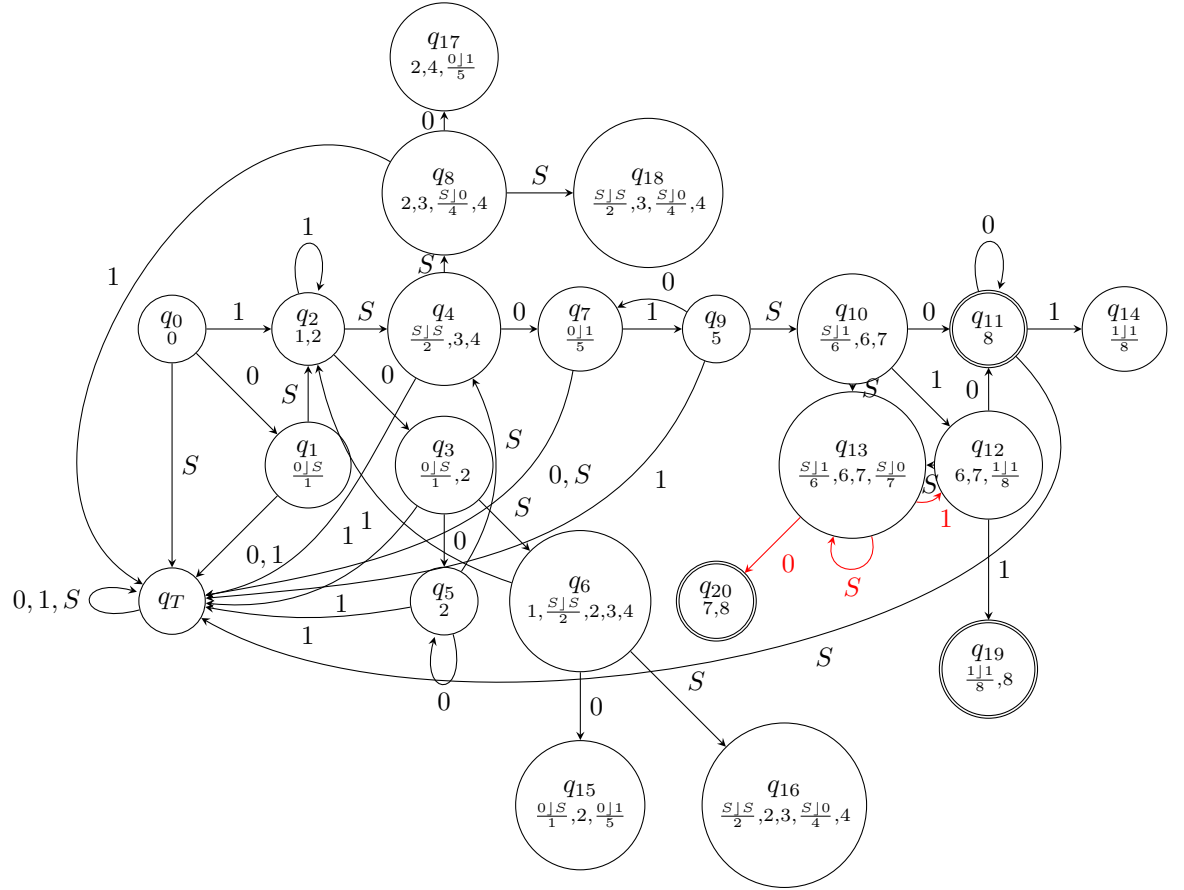
На 0 мы попадаем в q_{11} , на 1 в $\frac{1|1}{8}, 8$ (q_{19}), что тоже распознающее состояние, на S в q_{13} .



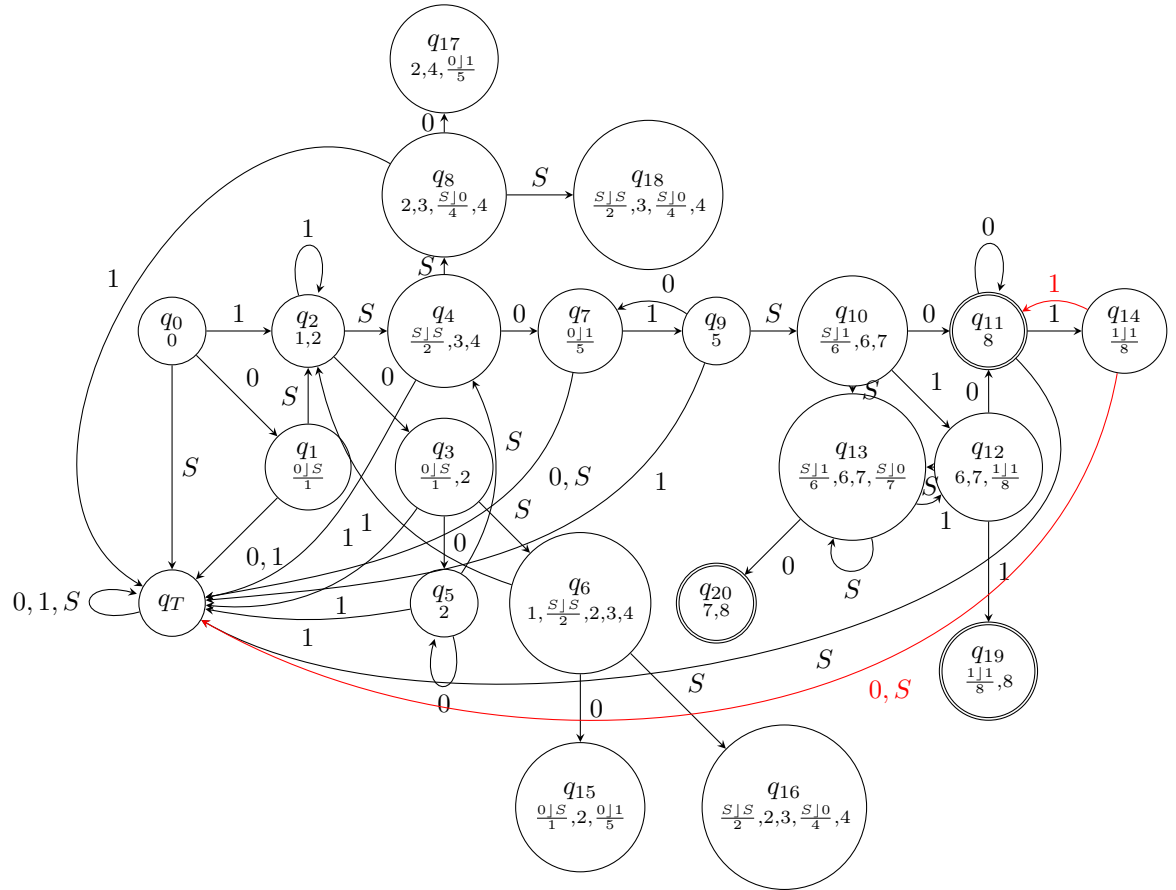
13 состояний, еще 2. q_{13} ?

q_{12}	0	1	S
$\frac{S 1}{6}$	—	6, 7	—
6	8	$\frac{1 1}{8}$	$\frac{S 1}{6}, 6, 7, \frac{S 0}{7}$
7	8	$\frac{1 1}{8}$	7, $\frac{S 0}{7}$
$\frac{S 0}{7}$	7	—	—

На 0 получаем 7, 8 (q_{20}), новое распознающее состояние, на 1 получим обратно q_{12} , а на S получим петлю.



14 состояний, осталось последнее. Пусть будет q_{14} . Оно простое — возвращаемся в q_{11} по 1 и в q_T в остальных случаях:



15 состояний разобраны, задача решена.

11.6 Задача 6

Необходимые темы: автоматы, распознающие слова (раздел 6.6 на с. 123).

Эта задача чем-то похожа на предыдущую, но мы разбираем более «просто» устроенные слова (да и алфавит просто $\{0, 1\}$, S не бывает), но разметка состояний тут становится *намного* сложнее. В задаче нам дается 3 слова из 0 и 1, и нам нужно составить такой автомат, который распознает слова, в которых есть все 3 эти слова. Более того, эти 3 слова не должны «пересекаться», т.е. к примеру, если нам нужно найти подслова $B = \{011, 101, 1110\}$, то слово 011101 нам не подойдет, хотя в нем можно найти и все три слова: **011**101, 01**110**1, 0111**0**1. Дело в том, что эти слова пересекаются. Вместо этого нам придется искать такие слова, как **010**1**10**11011101. Напоминаю, что некоторые слова можно интерпретировать по разному, например, это

слово можно интерпретировать и как 010110111011101. Еще, мы не знаем даже порядок этих подслов в слове, которое нам дают, поэтому автомату нужно искать все 3 одновременно, а как только он какое-то слово находит, ему надо решить, брать его прямо сейчас или попытаться найти там что-то еще (спойлер, автомат это не может решить сходу, поэтому он вынужден рассматривать эти варианты одновременно). Что ж, лучше приступить уже к решению собственно задачи.

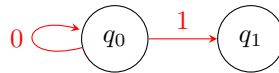
Задача. Составить диаграмму переходов автомата, распознающего слова, которые содержат без пересечений все слова из множества $B = \{111, 10101, 101\}$.

Решение. Сначала мы поступаем достаточно очевидным образом: каждому состоянию мы приписываем текущее (предположительно) положение в искомом слове. Для наглядности в схеме я буду писать только номера состояний и переходы, а разметку состояний буду писать отдельно (возможно, на зачете стоит сделать так же).

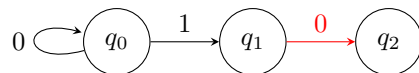
Итак, q_0 . Изначально автомат не прочитал ни одного символа, так что во всех словах (если они конечно начались) он находится в самом начале. Обозначим это вот так:

$$\begin{array}{l} q_0 : \quad |111 \\ \quad \quad |10101 \\ \quad \quad |101 \end{array}$$

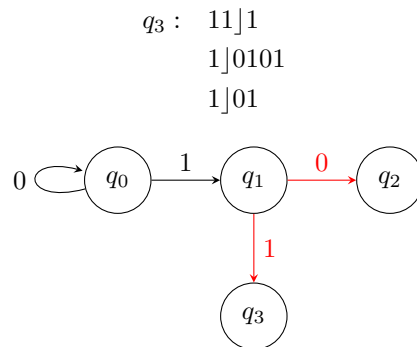
Что будет, если мы прочитаем сейчас 0? Правильно, ничего, все слова должны начинаться с 1. Значит ни одно из слов еще не началось, мы остаемся в q_0 . А что будет, если мы прочитаем 1? Мы переходим в состояние q_1 , где все слова продвинулись на 1 шаг вперед:

$$\begin{array}{l} q_1 : \quad 1|11 \\ \quad \quad 1|0101 \\ \quad \quad 1|01 \end{array}$$


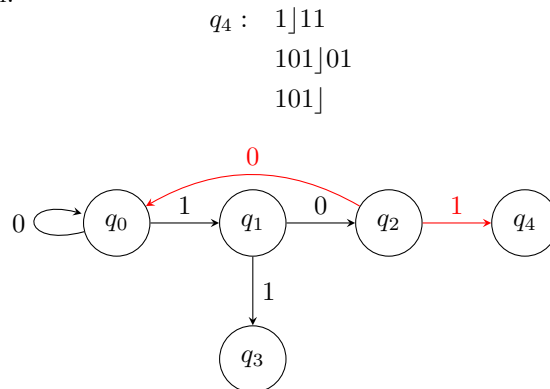
Что будет, если из состояния q_1 мы прочитаем 0? Второе и третье слово продвинулись на шаг, а первое слово сбросится в начало, так как дальше должна была быть 1, а мы прочитали на данный момент 10, чего в слове 111 быть не должно:

$$\begin{array}{l} q_2 : \quad |111 \\ \quad \quad 10|101 \\ \quad \quad 10|1 \end{array}$$


А если мы прочитаем 1? Первое слово просто продвинется на шаг, но что будет со 2 и 3? Там ожидался 0, так что вы могли подумать, что они сбрасываются в самое начало. Тем не менее, это не так. Входной поток может выглядеть вот так: ...1**101**..., причем мы прочитали сейчас первые 2 единицы. Вторая единица может быть первой единицей слова 101 или слова 10101, первая же единицу мы называем «началом слова» и игнорируем. Итак, мы переходим в состояние q_3 такого вида:



В этой задаче нам нужно рассмотреть только 1 ветку автомата до конца, т.е. одно из состояний q_2 и q_3 мы должны будем продолжить, а другое — нет. Давайте продолжим q_2 . Если из q_2 сейчас мы прочитаем 0, то это будет значить, что пока входной поток выглядит так: ...100. Это не может быть началом ни одного из наших 3 слов. Поэтому мы переходим в состояние q_0 . Если же мы читаем 1, то переходим в состояние q_4 , продвинувшись на шаг по всем словам:



Что же у нас происходит с состоянием q_4 ? Мы уже нашли слово 101. Мы можем прямо сразу его забрать и начать читать оставшиеся 2 слова с самого начала. Но есть еще другой вариант.

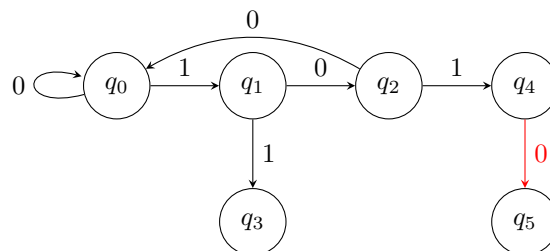
Решение. Давайте для примера рассмотрим входное слово 101]110101010101. На данный момент мы прочитали первые 3 символа. Если мы сейчас «заберем» слово 101, то мы никогда больше не

встретим слово 111, и автомат не распознает это слово. Но он должен был его распознать, так как в нем все таки есть все 3 подслова: 1011101010101. Как видим, в некоторых ситуациях автомату лучше подождать и не забирать слово сразу, а попытаться найти другие слова, как здесь 111. Назовем вариант «забираем сразу слово 101 и ищем 111 и 10101 с начала» как вариант *A*, вариант «пытаемся добрать 111, а как-нибудь потом будем добирать 10101 и 101» вариантом *B*, а вариант «пытаемся добрать 10101, и как-нибудь потом 111 и 101» вариантом *C*. Еще важная вещь — рассматривать варианты *B* и *C* имеет смысл только если записи слова 101 и какого-то еще пересекаются, например как в $\overline{101}11$. Если же записи не пересекаются, как в $\overline{101}111$, то нам ничего не препятствует воспользоваться вариантом *A* и сразу же добирать слово 111. Соответственно оба варианта *B* и *C* работают только *не выходя из пересечения* с 101. В варианте *B* пересечение сейчас составляет один символ: 1 (я буду обозначать это красным цветом, на зачете же лучше писать «1 символ пересечения» или что-то подобное), а в варианте *C*: три символа, 101. Итак, запись всего этого счастья:

$$\begin{aligned}
 q_4 : \quad & A) \begin{array}{l} \text{111} \\ \text{10101} \end{array} \\
 & B) \text{1} \text{11} \\
 & \quad + 10101, 101 \\
 & C) \text{101} \text{01} \\
 & \quad + 111, 101
 \end{aligned}$$

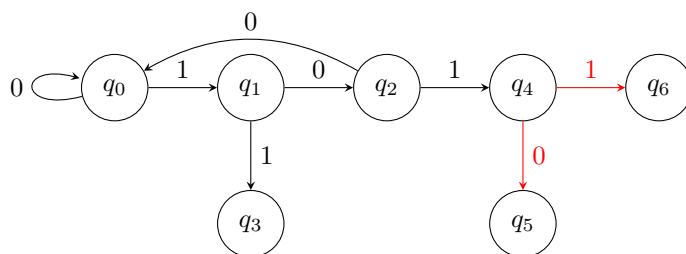
К сожалению, нам придется разбирать все 3 варианта одновременно. Что ж, продолжим? Если мы читаем 0, то оба слова в *A* остаются на месте. Что же происходит с вариантом *B*? Если мы прочитали 0, то значит входной поток сейчас выглядит примерно так: ...1010. Значит слово 111 ну никак не может пересекаться с 101, значит вариант *B* *исчезает*. Вариант *C* же просто продвигается на 1 шаг вперед:

$$\begin{aligned}
 q_5 : \quad & A) \begin{array}{l} \text{111} \\ \text{10101} \end{array} \\
 & B) - \\
 & C) \text{1010} \text{1} \\
 & \quad + 111, 101
 \end{aligned}$$



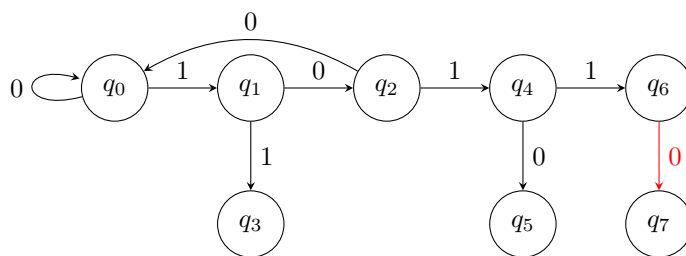
Если же мы читаем 1, то в варианте A оба слова продвигаются на шаг, в варианте B тоже слово на шаг, а в варианте C происходит что-то чуть интереснее. Давайте рассмотрим это что-то подробнее: пусть у нас есть входной поток ...**1011**. Как видим, вариант C тоже исчезает, потому что это не может быть началом 10101.

q_6 : $A)$ 1|11
 1|0101
 $B)$ **1**|1
 + 10101, 101
 $C)$ –



Давайте дальше рассмотрим вариант q_6 . Если мы читаем 0, то в варианте A у нас продвигается второе слово, а первое сбрасывается в начало. Вариант B же просто убирается, потому что если там 0, то 111 не может пересекаться с 101.

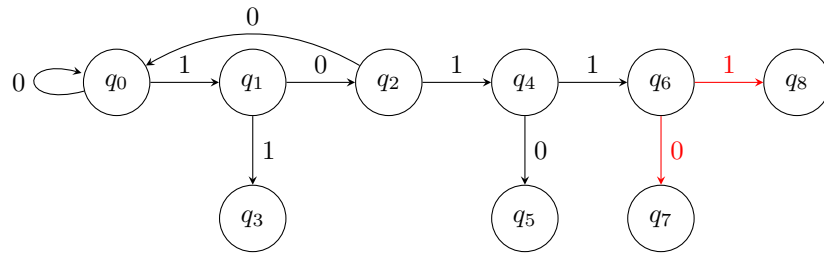
q_7 : $A)$ |111
 10|101
 $B)$ –



Если же мы читаем 1, то в варианте A первое слово перемещается на шаг вперед, а второе остается на месте (первая единица сбрасывается, а следующая сразу же становится началом слова), а вариант B успешно находит слово 111 и переходит к обычному поиску одного из слов 10101 или

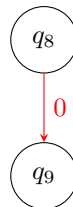
101:

q_8 : $A) 11|1$
 $1|0101$
 $B) |10101$
 $|101$



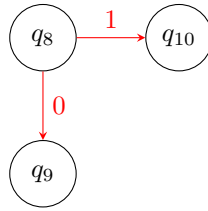
Будем рассматривать далее q_8 . Если мы читаем 0, то в варианте A первое слово сбрасывается, второе делает шаг вперед, в варианте B оба слова остаются на месте:

q_9 : $A) |111$
 $10|101$
 $B) |10101$
 $|101$



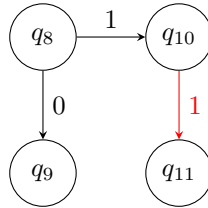
Если же мы читаем 1, то вариант A успешно находит слово 111. У нас правда остается частично прочитанное $1|0101$. Как мы уже знаем, в таком случае у нас происходит деление на варианты: вариант $A1$, где мы сразу читаем 111, и вариант $A2$, где мы пытаемся добрать 10101. В варианте B же оба слова просто делают шаг вперед.

q_{10} : $A1) |10101$
 $A2) 1|0101$
 $+ 111$
 $B) 1|0101$
 $1|01$



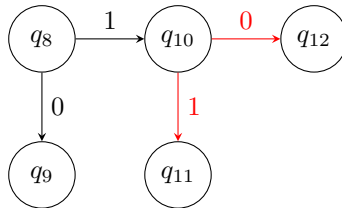
Рассмотрим далее q_{10} . Если мы читаем сейчас 1, то вариант $A1$ делает шаг вперед, вариант $A2$ убирается (потому что больше слова не могут пересекаться), оба слова варианта B остаются на месте.

q_{11} : $A1)$ 1]0101
 $A2)$ —
 $B)$ 1]0101
 1]01



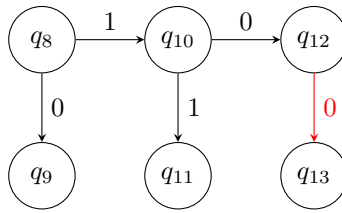
Если же там будет 0, то $A1$ останется на месте, $A2$ сделает шаг вперед, оба слова B тоже шаг вперед:

q_{12} : $A1)$]10101
 $A2)$ **1**0]101
 + 111
 $B)$ 10]101
 10]1



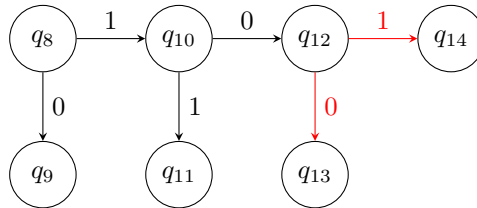
q_{12} . Если читаем 0, вариант $A1$ на месте, вариант $A2$ убирается, оба слова в B сбрасываются в начало.

q_{13} : $A1)$]10101
 $A2)$ —
 $B)$]10101
]101



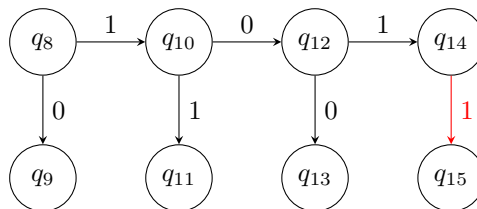
Если читаем 1, то все варианты делают шаг вперед, причем вариант B дочитывает слово 101 до конца и разбивается на варианты $B1$ и $B2$.

q_{14} : $A1) 1|0101$
 $A2) \textcolor{red}{1}01|01$
 $+ 111$
 $B1) |10101$
 $B2) \textcolor{red}{1}01|01$
 $+ 101$



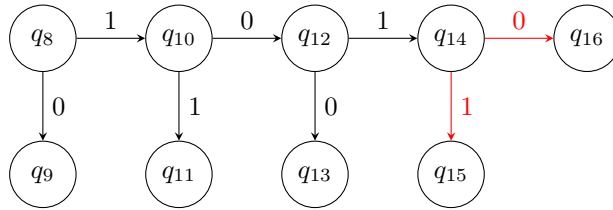
q_{14} . Читаем 1 — $A1$ остается на месте, $A2$ убирается, $B1$ делает шаг вперед, $B2$ убирается.

q_{15} : $A1) 1|0101$
 $A2) -$
 $B1) 1|0101$
 $B2) -$



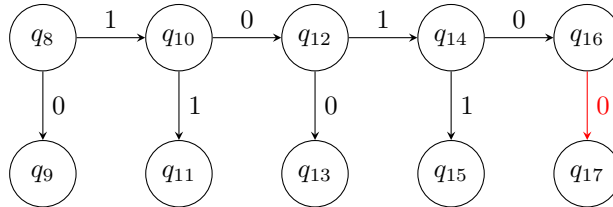
Читаем 0 — $A1$, $A2$ и $B2$ делают шаг вперед, $B1$ на месте.

$$\begin{array}{r} q_{16} : \quad A1) 10 \rfloor 101 \\ \quad \quad A2) \textcolor{red}{1}010 \rfloor 1 \\ \quad \quad \quad + 111 \\ \quad \quad B1) \rfloor 10101 \\ \quad \quad B2) \textcolor{red}{1}010 \rfloor 1 \\ \quad \quad \quad + 101 \end{array}$$



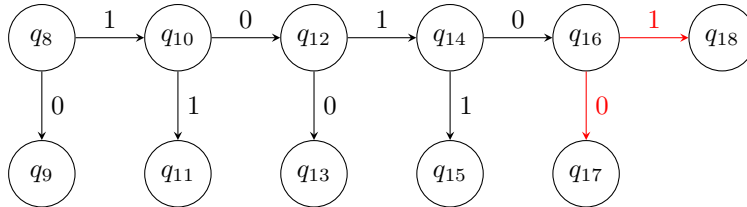
q_{16} . Читаем 0 — $A1$ сбрасывается, $B1$ остается на месте, а $A2$ и $B2$ убираются.

$$\begin{array}{r} q_{17} : \quad A1) \rfloor 10101 \\ \quad \quad A2) - \\ \quad \quad B1) \rfloor 10101 \\ \quad \quad B2) - \end{array}$$



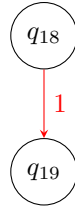
Читаем 1 и слово 10101 в вариантах $A2$ и $B2$ успешно распознается, слова же в вариантах $A1$ и $B1$ делают шаг вперед:

$$\begin{array}{r} q_{18} : \quad A1) 101 \rfloor 01 \\ \quad \quad A2) \rfloor 111 \\ \quad \quad B1) 1 \rfloor 0101 \\ \quad \quad B2) \rfloor 101 \end{array}$$



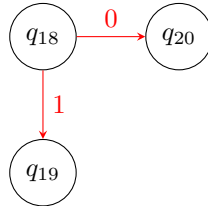
q_{18} . Читаем 1 и шаг вперед делают $A2$ и $B2$. Вариант $A1$ возвращается к первой единице. Вариант $B1$ остается на месте. Заметим, что в этом случае варианты $A1$ и $B2$ идентичны, мы можем их объединить. Более того, если найдется слово 10101 из варианта $A1/B1$, то это будет значить, что слово 101 из $B2$ найдется раньше, а так как мы ждем завершения любого варианта, можно выкинуть вариант $A1/B1$, достаточно только $B2$

q_{18} : $A1) 1|0101$
 $A2) 1|11$ $A2) 1|11$
 $B1) 1|0101 \Rightarrow B2) 1|01$
 $B2) 1|01$



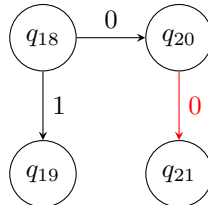
Если же мы читаем 0, то варианты $A1$ и $B1$ продвигаются, а варианты $A2$ и $B2$ — нет.

q_{20} : $A1) 1010|1$
 $A2) |111$
 $B1) 10|101$
 $B2) |101$



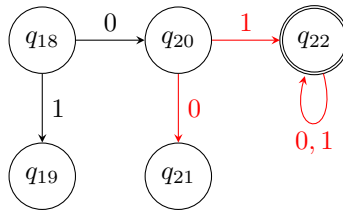
q_{20} . Если мы читаем 0, то все состояния сбрасываются в самое начало, причем состояния $A1$ и $B1$ объединяются, а потом их можно убрать и оставить $B2$:

q_{21} : $A2) |111$
 $B2) |101$



Если же мы читаем 1, то вариант A1 завершается и мы попадаем в распознающее состояние q_{22} , откуда больше не уходим. Задача решена.

q_{22} : A1) 10101]
 A2) 1]11
 B1) 101]01
 B2) 1]01



Есть еще одна особенность, которую не получилось учесть в данной задаче. Пусть у нас есть вариант

?) 111]0
 +???

Т.е. мы почти распознали слово 1110, и на данный момент есть 3 символа пересечения. Для конкретности, давайте предположим, что оно должно пересекаться со словом 0111, т.е. на данный момент вход выглядит как ...0111.... Если далее мы прочитаем 0, то этот вариант успешно будет распознан. Но что будет, если мы прочитаем 1? А вот что: ...01111..., мы все еще останемся в этой же позиции в слове, но у нас будет уже 2 символа пересечения с 0111:

?) 111]0
 +???

Если мы прочитаем здесь 0, то вариант распознается. А если 1? Теперь будет 1 символ пересечения: ...011111....

?) 111]0
 +???

Дальше уже идти нельзя. Если мы теперь прочитаем 1, то пересечения не останется, т.е. слово 1110 (если оно вообще есть) не будет пересекаться с 0111, т.е. этот вариант вычеркивается (подобные варианты работают только если искомое слово пересекается с «распознанным», но пропущенным, ну вы поняли). Если же 0, то он все таки успешно распознается. На этом эта задача все.

Бонусный раздел, определения примитивно рекурсивных функций

Для последующих задач нам потребуются много разных специальных функций, которые мы будем использовать для определения других. Здесь мы определим их все как примитивно рекурсивные. Напоминаю, операция примитивной рекурсии работает так: пусть мы хотим определить функцию некую $f(x_1, \dots, x_n, x_{n+1})$ при помощи примитивной рекурсии по последней переменной, x_{n+1} . Для этого мы определим ее значение при $x_{n+1} = 0$, это будет функция $f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n)$. Далее нам потребуется определить значение этой функции при $x_{n+1} = y + 1$, причем для этого нам разрешено использовать значение функции $f(x_1, \dots, x_n, y)$. И да, нам не разрешено использовать никакие другие рекурсивные значения, типа $f(x_1, \dots, x_n, y - 1)$, только $f(x_1, \dots, x_n, y)$, поэтому это и называется примитивная рекурсия. Формально нам пришлось бы определить функцию $h(x_1, \dots, x_n, y, z)$, где $z = f(x_1, \dots, x_n, y)$, но тут мы чаще всего не будем так писать.

Уже было несколько примеров примитивно-рекурсивных функций, давайте их вспомним:

$$\begin{aligned} f(x, y) = x + y & \quad \begin{cases} f(x, 0) &= I_1^1(x) \\ f(x, y + 1) &= S(f(x, y)) \end{cases} \\ \varphi(x, y) = x \cdot y & \quad \begin{cases} \varphi(x, 0) &= O(x) \\ \varphi(x, y + 1) &= f(x, \varphi(x, y)) \end{cases} \\ p(x, y) = x^y & \quad \begin{cases} p(x, 0) &= 1 = S(O(x)) \\ p(x, y + 1) &= \varphi(x, p(x, y)) \end{cases} \end{aligned}$$

Это функции сложения, умножения и степени соответственно. Мы просто определили их значения на 0 и на $y + 1$, причем довольно похожим образом. Просто применив примитивную рекурсию к этим определениям мы получим то, что мы хотели.

Было бы неплохо еще определить вычитание, но это не так просто. Во-первых, если $x < y$, то $x - y$ будет отрицательным числом, а мы такие не используем. Как быть? Давайте просто сделаем так, что если получилось отрицательное число, мы просто выдавали 0. Вполне можно же сделать и так, верно? Т.е. мы пытаемся определить функцию

$$\text{sub}(x, y) = \begin{cases} x - y & \text{если } x > y \\ 0 & \text{иначе} \end{cases}$$

Давайте попробуем. $\text{sub}(x, 0) = x$, тут все просто. Но $\text{sub}(x, y + 1) = x - (y + 1) = (x - y) - 1 = \text{sub}(x, y) - 1$, т.е. нам придется еще вычесть 1 из $\text{sub}(x, y)$. Вы могли подумать, что это просто $\text{sub}(\text{sub}(x, y), 1)$, но нет, хотя нам и разрешено использовать в примитивной рекурсии $\text{sub}(x, y)$, мы не можем использовать ее никак еще, поэтому такая запись запрещена.

Как же быть? Определить специальную функцию для вычитания 1:

$$d(x) = \begin{cases} x - 1 & \text{если } x > 0 \\ 0 & \text{иначе} \end{cases}$$

Для такой же функции мы можем составить рекурсивное определение значительно проще:

$$\begin{cases} d(0) & = 0 \\ d(x + 1) & = x \end{cases}$$

Да, мы можем использовать в рекурсивных определениях саму переменную рекурсии, так что это определение абсолютно верно. И теперь с помощью этой функции мы можем определить вычитание:

$$\begin{cases} \text{sub}(x, 0) & = x \\ \text{sub}(x, y + 1) & = d(\text{sub}(x, y)) \end{cases}$$

Теперь к более специфичным функциям. Есть одна особенная функция, $\text{sign}(x)$ или $\text{sg}(x)$, «сигнум», которая определяет *знак* числа, которое ей подадут как аргумент. Т.е. для положительных чисел она возвращает 1, для отрицательных -1 , а для 0 она возвращает 0. Правда у нас нету отрицательных чисел, так что мы будем использовать такое определение этой функции:

$$\text{sg}(x) = \begin{cases} 0 & \text{если } x = 0 \\ 1 & \text{если } x > 0 \end{cases}$$

Мы вполне можем определить эту функцию при помощи примитивной рекурсии, а именно вот так:

$$\begin{cases} \text{sg}(0) & = 0 \\ \text{sg}(x + 1) & = 1 \end{cases}$$

Сейчас вопрос, зачем это все. Дело в том, что эта функция будет очень нам полезна для различных логических операций. Если мы будем воспринимать 0 как «ложь», а 1 как «истину», то $\text{sg}(x)$ «нормализует» число, т.е. преобразует его в 0 или 1. 0, если число и было 0, и 1, если нет. Заметим, что $\text{sg}(0) = 0$ и $\text{sg}(1) = 1$.

Раз уж мы приступили к логическим функциям. Для начала нам нужно отрицание, т.е. некая функция, у которой $f(0) = 1$ и $f(1) = 0$. Какие будут все остальные значения? Пусть будут тоже 0, так логичнее. Это функция $\overline{\text{sg}}(x)$, «не-сигнум»:

$$\overline{\text{sg}}(x) = \begin{cases} 1 & \text{если } x = 0 \\ 0 & \text{если } x > 0 \end{cases}$$

Мы можем определить ее примерно так же, как мы определили $\text{sg}(x)$:

$$\begin{cases} \overline{\text{sg}}(0) & = 1 \\ \overline{\text{sg}}(x + 1) & = 0 \end{cases}$$

И действительно, $\overline{\text{sg}}(0) = 1$ и $\overline{\text{sg}}(1) = 0$ — именно то, что мы хотели.

Далее — конъюнкция. Нам нужна функция, обладающая такими свойствами: $f(0, 0) = f(0, 1) = f(1, 0) = 0, f(1, 1) = 1$. Тут все удивительно просто, это умножение! Т.е. наша конъюнкция — обыкновенное умножение, которое мы уже определили, так что $x \& y = x \cdot y$.

Что насчет дизъюнкции? Было бы логично использовать для этого сложение: $f(0, 0) = 0, f(0, 1) = f(1, 0) = 1, f(1, 1) = 2$. Проблема в этой двойке. Но не беда, у нас как раз есть функция, которая превращает все в 0 и 1 — $\text{sg}(x)$. Итого, $x \vee y = \text{sg}(x + y)$. Можете проверить, что это действительно выполняется. Нам даже не понадобилась рекурсия на этот раз, мы просто применили композицию двух уже известных нам функций.

Итак, полный набор. Только мы работаем с числами, и нам еще было бы неплохо иметь операции для сравнения, т.е. $>, <, \geq, \leq, =, \neq$. Как мы это сделаем? У нас уже есть функция, очень близкая к сравнению — это вычитание:

$$\text{sub}(x, y) = \begin{cases} x - y & \text{если } x > y \\ 0 & \text{иначе} \end{cases}$$

Т.е. результат вычитания будет больше нуля, если $x > y$. Что проверяет числа на равенство нулю? $\text{sg}(x)$. Давайте тогда рассмотрим $\text{sg}(x - y)$. Если $x \leq y$, то наше вычитание возвращает 0, и сигнум его и оставляет нулем. Если $x > y$, то вычитание возвращает нечто не меньше 1, что сигнум превращает в ровно 1. Итого

$$\text{sg}(x - y) = \begin{cases} 0 & \text{если } x \leq y \\ 1 & \text{если } x > y \end{cases} = [x > y]$$

Итак, мы получили нашу первую функцию сравнения. Нужно получить остальные. Очевидно, $<$ можно получить просто поменяв аргументы местами: $[x < y] = \text{sg}(y - x)$. Как быть с \geq и \leq ? Эти функции — просто отрицания функций $<$ и $>$ соответственно, а для отрицания у нас используется $\overline{\text{sg}}(x)$:

$$\begin{aligned} [x \geq y] &= \overline{\text{sg}}(y - x) \\ [x \leq y] &= \overline{\text{sg}}(x - y) \end{aligned}$$

Еще мы можем определить $[x = y] = [x \geq y] \& [x \leq y]$ и $[x \neq y] = \overline{\text{sg}}([x = y])$.

Итак, мы почти готовы для более сложных функций, не хватает только одной компоненты. Дело в том, что мы можем вести рекурсию не только по последней переменной, но и по любой другой. Предположим, мы хотим определить некоторую функцию $f(x, y, z, t)$ и мы хотим вести рекурсию по переменной y . Как нам это сделать? Определить при помощи примитивной рекурсии функцию $g(x_1, x_2, x_3, x_4)$, где x_1 соответствует x , x_2 соответствует t , x_3 соответствует z и x_4 соответствует y . Теперь $x_4 = y$ стоит на последнем месте и может являться переменной рекурсии. После этого мы сможем определить функцию f просто как $f(x, y, z, t) = g(x, t, z, y)$, т.е. просто переставить переменные. Таким вот способом мы можем вести рекурсию по любой переменной, и мы будем этим свободно пользоваться.

Определим теперь функцию $\max(x, y)$, которая определяется как

$$\max(x, y) = \begin{cases} x & \text{если } x \geq y \\ y & \text{если } x < y \end{cases}$$

Мы можем определить $[x \geq y]$ как $\overline{\text{sg}}(y - x)$, как мы помним, а $[x < y]$ как $\text{sg}(y - x)$. Эти функции будут равны 1 если условие будет выполняться и 0, если не будет. Тогда:

$$\max(x, y) = x \cdot \overline{\text{sg}}(y - x) + y \cdot \text{sg}(y - x)$$

Когда $x \geq y$, $\overline{\text{sg}}(y - x) = 1$ и $\text{sg}(y - x) = 0$, и $x \cdot \overline{\text{sg}}(y - x) + y \cdot \text{sg}(y - x) = x$. Когда же $x < y$, тогда $x \cdot \overline{\text{sg}}(y - x) + y \cdot \text{sg}(y - x) = y$, чего мы и хотели добиться.

Так же мы можем определять и более сложные функции, например

$$f(x, y) = \begin{cases} x + y & \text{если } x \geq 10 \\ xy & \text{если } x < 10 \ \& \ x \leq y \\ y & \text{иначе} \end{cases}$$

Ветка «иначе» эквивалентна условию $x < 10 \ \& \ x > y$, так что если мы определим

$$\begin{aligned} P_1 &= \overline{\text{sg}}(10 - x) \\ P_2 &= \text{sg}(10 - x) \cdot \overline{\text{sg}}(x - y) \\ P_3 &= \text{sg}(10 - x) \cdot \text{sg}(x - y) \end{aligned}$$

то $f(x, y) = (x + y) \cdot P_1 + xy \cdot P_2 + y \cdot P_3$.

Рассмотрим еще одну важную функцию, $\text{mod}(x, y)$. Правда так как деление на 0 запрещено, если $y = 0$ мы определим $\text{mod}(x, y) = 0$. В этом случае удобнее вести рекурсию по x . Чтобы понять, как определить это при помощи рекурсии, нужно понять, как $\text{mod}(x + 1, y)$ зависит от $\text{mod}(x, y)$. Давайте рассмотрим это на примере $y = 4$:

$$\begin{aligned} \text{mod}(0, 4) &= 0 \\ \text{mod}(1, 4) &= 1 \\ \text{mod}(2, 4) &= 2 \\ \text{mod}(3, 4) &= 3 \\ \text{mod}(4, 4) &= 0 \\ \text{mod}(5, 4) &= 1 \\ \text{mod}(6, 4) &= 2 \\ \text{mod}(7, 4) &= 3 \\ \text{mod}(8, 4) &= 0 \\ &\vdots \end{aligned}$$

Эта функция постепенно возрастает на 1, пока не встречает значение, кратное y , и тогда она сбрасывается на 0. При этом последнее значение функции перед сбросом всегда $y - 1$. Так что значение $\text{mod}(x + 1, y)$ будет равно 0, если $\text{mod}(x, y) = y - 1$ и оно равно $\text{mod}(x, y) + 1$ если $\text{mod}(x, y) < y - 1$. Т.е.

$$\text{mod}(x + 1, y) = 0 \cdot [\text{mod}(x, y) = y - 1] + (\text{mod}(x, y) + 1) \cdot [\text{mod}(x, y) < y - 1]$$

Итак:

$$\begin{cases} \text{mod}(0, y) &= 0 \\ \text{mod}(x + 1, y) &= S(\text{mod}(x, y)) \cdot \text{sg}((y - 1) - \text{mod}(x, y)) \end{cases}$$

Так же определим $\text{div}(x, y)$: оно или остается тем же, чем было, или увеличивается на 1 когда $\text{mod}(x + 1, y) = 0$. Или

$$\begin{cases} \text{div}(0, y) &= 0 \\ \text{div}(x + 1, y) &= \begin{cases} \text{div}(x, y) & \text{если } \text{mod}(x + 1, y) > 0 \\ S(\text{div}(x, y)) & \text{если } \text{mod}(x + 1, y) = 0 \end{cases} \end{cases}$$

Попробуем так же определить логарифм. Правда логарифм дает дробные числа, будем их округлять вверх: $l(x, y) = \lceil \log_y x \rceil$. И да, определим $l(0, y) = 0$. Как видим, здесь снова легче вести индукцию по первой переменной. Давайте рассмотрим некоторые значения функции на $y = 2$:

$$\begin{aligned} l(0, 2) &= 0 \\ l(1, 2) &= 0 \\ l(2, 2) &= 1 \\ l(3, 2) &= 2 \\ l(4, 2) &= 2 \\ l(5, 2) &= 3 \\ l(6, 2) &= 3 \\ l(7, 2) &= 3 \\ l(8, 2) &= 3 \\ l(9, 2) &= 4 \\ &\vdots \end{aligned}$$

Т.е. функция сохраняет свое значение до тех пор, пока не доберется до следующей степени y . До какой? До $y^{l(x, y)}$! Когда значение $l = 3$, то скачок произошел на $x = 2^3 = 8$. То есть функция остается такой же, если $x < y^{l(x, y)}$ и увеличивается на 1, если $x = y^{l(x, y)}$.

$$\begin{cases} l(0, y) &= 0 \\ l(x + 1, y) &= \begin{cases} l(x, y) & \text{если } x < y^{l(x, y)} \\ S(l(x, y)) & \text{если } x = y^{l(x, y)} \end{cases} \end{cases}$$

Есть еще один важный прием, который нужно научиться использовать — *инвариант цикла*. Давайте рассмотрим на примере: попробуем определить функцию $\text{nd}(x)$, которая нам далее тоже будет часто нужна — число делителей числа x . Как вообще к такому приступить? Мы не сможем воспользоваться примитивной рекурсией по переменной x , потому что число делителей очень слабо зависит от числа делителей предыдущего числа.

Как бы мы поступили, если бы писали это обычной программой? Мы бы просто перебрали все числа от 0 до x (это можно оптимизировать, но мы не будем, мы тут математикой занимаемся, оптимальные алгоритмы нам не нужны), что обычно происходит в переменной i , а потом если x делится на i , то i — делитель и мы делаем $+1$ в некоторую переменную-счетчик. Эта переменная и называется *инвариантом цикла*. Так как она меняется не только в зависимости от x , но и от номера прохода цикла (i), то это функция $\varphi(x, i)$. Она представляет из себя «число делителей x , которые мы нашли до проверки i (сама проверка i считается)», или, более формально «число делителей x , которые не превосходят i ». Тогда наша искомая функция будет просто $\text{nd}(x) = \varphi(x, x)$, так как дойдя до $i = x$ мы точно найдем все возможные делители.

Как же определить нашу функцию $\varphi(x, i)$? А просто, примитивная рекурсия, очевидно по i . Во первых, $\varphi(x, 0) = 0$, это начальное значение. Во вторых, на каждом шаге значение функции или увеличивается на 1 (если текущее i — делитель x), или остается таким же. Тогда значение для $\varphi(x, i + 1)$ находится так:

$$\varphi(x, i + 1) = \varphi(x, i) + \begin{cases} 1 & \text{если } \text{mod}(x, i + 1) = 0 \\ 0 & \text{иначе} \end{cases}$$

(да, мы можем прибавлять кусочную функцию, вам что-то не нравится?), что можно так же записать как

$$\varphi(x, i + 1) = \varphi(x, i) + 1 \cdot \overline{\text{sg}}(\text{mod}(x, i + 1)) + 0 \cdot \text{sg}(\text{mod}(x, i + 1))$$

(напоминаю, что $\overline{\text{sg}}$ равен 1 только когда его аргумент *не* равен 0), что мы можем упростить до

$$\varphi(x, i + 1) = \varphi(x, i) + \overline{\text{sg}}(\text{mod}(x, i + 1))$$

Итого наше полное определение $\text{nd}(x)$ будет выглядеть так:

$$\begin{aligned} \text{nd}(x) &= \varphi(x, x) \\ \varphi(x, 0) &= 0 \\ \varphi(x, i + 1) &= \varphi(x, i) + \overline{\text{sg}}(\text{mod}(x, i + 1)) \end{aligned}$$

Далее давайте еще потренируемся, определим так же $\text{nsd}(x)$ — число простых делителей x .² Снова будем использовать инвариант цикла $\varphi(x, i)$ —

² nsd видимо от «Number of Simple Divisors». Не говорите Константину Ивановичу, что «простое число» по-английски prime, а не simple.

число простых делителей x , которые не больше i . Очевидно, что $\varphi(x, 0) = 0$. Как считать $\varphi(x, i + 1)$? Очевидно, что после проверки i число простых делителей или увеличивается на 1 (если $i + 1$ простой делитель), или остается тем же, что мы можем записать как

$$\varphi(x, i + 1) = \begin{cases} \varphi(x, i) + 1 & \text{если } i + 1 \text{ — простой делитель} \\ \varphi(x, i) & \text{иначе} \end{cases}$$

что полностью эквивалентно записи

$$\varphi(x, i + 1) = \varphi(x, i) + \begin{cases} 1 & \text{если } i + 1 \text{ — простой делитель} \\ 0 & \text{иначе} \end{cases}$$

так что мы можем выбирать то, что нам больше нравится. А теперь вопрос, что значит « $i + 1$ — простой делитель»? Это значит, что $i + 1$ — простое число, т.е. $\text{nd}(i + 1) = 2$, и что x делится на $i + 1$, $\text{mod}(x, i + 1) = 0$. Значит мы можем записать это как

$$\varphi(x, i + 1) = \varphi(x, i) + \begin{cases} 1 & \text{если } \text{nd}(i + 1) = 2 \ \& \ \text{mod}(x, i + 1) = 0 \\ 0 & \text{иначе} \end{cases}$$

или

$$\varphi(x, i + 1) = \varphi(x, i) + \overline{\text{sg}}(\text{nd}(i + 1) - 2) \cdot \overline{\text{sg}}(2 - \text{nd}(i + 1)) \cdot \overline{\text{sg}}(\text{mod}(x, i + 1))$$

(далее я не буду так расписывать все, потому что это слишком длинно). И да, очевидно, что все простые делители не больше x , т.е. $\text{nsd}(x) = \varphi(x, x)$.

Далее для зачетных задач нам потребуется еще несколько функций, нужных для *работы со списками*. Да, мы можем представлять списки как числа. У нас есть несколько способов представления, но сначала рассмотрим самый логичный — десятичная запись числа. Мы будем рассматривать x как список из цифр $\sigma_n \sigma_{n-1} \dots \sigma_1$. Заметьте, что нумерация идет справа налево, потому что так намного проще все реализовать. Так вот. Нам нужен способ получать отдельные цифры числа, а именно оператор $[x]_i$, позволяющий получать цифру номер i из числа x . Из программирования мы уже неплохо знаем, как это делается: чтобы получить первую (справа) цифру надо найти $\text{mod}(x, 10)$, а чтобы найти цифру номер i , нужно сначала «сдвинуть» число вправо на $i - 1$ позиций, т.е. чтобы получить вторую цифру, нужно сначала разделить число на 10: $\text{mod}(\text{div}(x, 10), 10)$, третью: $\text{mod}(\text{div}(x, 100), 10)$, а в общем виде:

$$[x]_i = \text{mod}(\text{div}(x, 10^{i-1}), 10)$$

К счастью, все необходимые функции мы уже определили, так что $[x]_i$ — примитивно рекурсивна. Как использовать ее, чтобы, например, найти сумму цифр числа x ? А просто, снова определим $\varphi(x, i)$ — сумму первых i цифр

(начиная справа) числа x . Изначально $\varphi(x, 0) = 0$, а с каждым шагом мы увеличиваем это на цифру в текущем разряде, т.е.

$$\begin{aligned}\varphi(x, 0) &= 0 \\ \varphi(x, i+1) &= \varphi(x, i) + [x]_{i+1}\end{aligned}$$

Вопрос, как определить нашу функцию, до куда идти? Нам еще нужен специальный оператор — «длина» числа x , т.е. количество цифр в нем, $\|x\|$. Как оказывается, $\|x\| = \lfloor \log_{10} x \rfloor + 1$, что можно легко проверить, если есть желание. Тогда наша функция $f(x)$ определяется как $f(x) = \varphi(x, \|x\|)$.

Чуть более сложный пример, давайте найдем число разрядов, совпадающих с симметрично расположенным. Например, в числе $x = 1234521$ получится $f(x) = 4$, а именно 1, 2, 2, 1 — эти симметричные разряды. Как это проверить? Снова инвариант $\varphi(x, i)$, i — номер разряда, до которого мы проверили. С каждым шагом мы просто будем добавлять к нему 1, если цифра в $i+1$ -м разряде совпадает с симметричным. Как найти симметричный? Предположим, что $\|x\| = 10$ для определенности. Тогда цифре 1 симметрична 10, цифре 2 симметрична 9, а цифре i симметрична $11-i$, или $\|x\| - i + 1$. Тогда цифре $i+1$ соответствует $\|x\| - i$, и нам всего лишь нужно проверять $[x]_{i+1} = [x]_{\|x\|-i}$, т.е.

$$\begin{aligned}\varphi(x, 0) &= 0 \\ \varphi(x, i+1) &= \varphi(x, i) + \begin{cases} 1 & \text{если } [x]_{i+1} = [x]_{\|x\|-i} \\ 0 & \text{иначе} \end{cases} \\ f(x) &= \varphi(x, \|x\|)\end{aligned}$$

С цифрами все. Какой еще есть способ представлять списки в одном числе? Разложение на простые числа! Смотрите, число $493920 = 2^5 \cdot 3^2 \cdot 5^1 \cdot 7^3$ соответствует списку 5, 2, 1, 3! Не особо удобно для нас, но вполне удобно для наших функций, кроме того в списке могут быть любые числа больше 0, а не только от 0 до 9. Правда сейчас нам придется определить функции, которые позволят нам обращаться к отдельным элементам этого списка. Например, нам придется уметь определять, на какую максимальную степень числа y делится число x , функция $\deg(x, y)$. Например $\deg(493920, 2) = 5$, а $\deg(493920, 7) = 3$. Понимаете, почему это было бы полезно для извлечения чисел из списка? Правда нам хотелось бы обращаться к ним по индексу, а не по простому числу, так что нам еще будет нужна функция $\text{simple}(x, i)$, которая будет возвращать i -е простое число в разложении x . Например $\text{simple}(2^5 \cdot 3^2 \cdot 5^1 \cdot 7^3, 3) = 5$, а $\text{simple}(2^5 \cdot 7^3, 2) = 7$. Если же этого числа там нет, то пусть возвращает 0. Примерно понятно? А теперь приступим к определениям.

Итак, $\deg(x, y)$. Как определить, на какую максимальную степень y делится x ? Просто делить на все подряд! Делим на y^1, y^2, y^3, \dots , пока не прекратит нацело делиться. Правда нам все равно нужна какая-то степень, на которую x прям точно делиться не будет, чтобы мы могли определить

$\deg(x, y)$ как $\varphi(x, y, ?)$. Нам вполне подойдет x , на y^x точно делиться не будет. Итак φ у нас счетчик, на каждом шаге если число x делится на y^{i+1} , то мы прибавляем к нему 1, а если нет, то не прибавляем. Итого

$$\begin{aligned}\varphi(x, y, 0) &= 0 \\ \varphi(x, y, i+1) &= \varphi(x, y, i) + \begin{cases} 1 & \text{если } \text{mod}(x, y^{i+1}) = 0 \\ 0 & \text{иначе} \end{cases} \\ \deg(x, y) &= \varphi(x, y, x)\end{aligned}$$

Определили. Теперь $\text{simple}(x, i)$ — простой делитель x , который стоит под номером i . Правда для него нам потребуется сначала определить еще одну функцию: $\text{num}(x, y)$ — число простых делителей x , которые не больше y , думаю понятно, как она связана с $\text{simple}(x, i)$. Тут нам даже не потребуются инвариант, ее можно определить обычной рекурсией по y :

$$\begin{aligned}\text{num}(x, 0) &= 0 \\ \text{num}(x, y+1) &= \text{num}(x, y) + \begin{cases} 1 & \text{если } y+1 \text{ — простой делитель } x \\ 0 & \text{иначе} \end{cases}\end{aligned}$$

Ну и да, «простой делитель x » определяется просто как $\text{nd}(y+1) = 2$ и $\text{mod}(x, y+1) = 0$.

Хорошо, теперь наконец то сам $\text{simple}(x, i)$. Тут мы сделаем немного необычный инвариант: $\psi(x, i, y)$ — простой делитель x под номером i , если он не больше y , или 0. Определять мы будем его рекурсией по y , т.е. мы будем просто перебирать все возможные числа от 0 до x и смотреть, является ли оно, во-первых, простым делителем x , а во-вторых, стоит ли оно под номером i . Первое условие мы уже знаем, как писать, а вот для второго мы определяли $\text{num}(x, y)$. Если $\text{num}(x, y) = i$, то это значит, что до y (включительно) стоит i простых делителей x , но так как y — сам простой делитель x , то это значит, что он и есть тот самый искомый i -й делитель. После определения разберем на примере. Итак, если мы наконец то встретили этот делитель, то значение ψ должно стать ему равно, а если нет, то должно остаться таким же как было, или 0, если мы еще не нашли делитель, или что-то еще, если мы его уже нашли. Итак:

$$\begin{aligned}\psi(x, i, 0) &= 0 \\ \psi(x, i, y+1) &= \begin{cases} y+1 & \text{если } y+1 \text{ — простой делитель } x \\ & \& \text{num}(x, y+1) = i \\ \psi(x, i, y) & \text{иначе} \end{cases} \\ \text{simple}(x, i) &= \psi(x, i, x)\end{aligned}$$

Итак, пример. Предположим у нас есть $x = 2 \cdot 3 \cdot 5 \cdot 11 \cdot 17 = 5610$ и мы хотим найти простой делитель номер $i = 4$. Итак, начнем вычислять инвариант начиная с $y = 0$. $\psi(5610, 4, 0) = 0$. Дальше мы просто начнем

перебирать все числа от $y + 1 = 1$ до $y + 1 = 5610$... К счастью нам придется не все. При $y + 1 = 1$ мы видим, что 1 не является даже простым числом, поэтому мы его игнорируем и $\psi(5610, 4, 1) = 0$. Для $y + 1 = 2$ мы действительно имеем простой делитель x , но $\text{num}(5610, 2) = 1$, так как есть только 1 простой делитель 5610, который не больше 2 (и это 2). Идем далее, $y + 1 = 3$. Это простой делитель, но $\text{num}(5610, 3) = 2$, так как есть 2 простых делителя 5610, которые не больше 3 (это 2 и 3). Аналогично $\text{num}(5610, 5) = 3$ и $\text{num}(5610, 11) = 4$, и как только мы добираемся до $y + 1 = 11$, значение $\psi(5610, 4, 11) = 11$. После этого мы проверяем все числа до 5610 и не встречаем больше простых делителей кроме 17, у которого $\text{num}(5610, 17) = 5$, так что значение 11 сохраняется до самого конца. Итак, $\psi(5610, 4, 5610) = 11$, а значит $\text{simple}(5610, 4) = 11$, чего мы и ожидали.

Итак, теперь у нас есть способ обращаться к отдельным степеням в разложении x на простые множители. Например, пусть $x = p_1^{k_1} \cdot p_2^{k_2} \cdot \dots \cdot p_r^{k_r}$, то мы можем получить k_i как $k_i = \deg(x, \text{simple}(x, i))$. Это мы писать каждый раз не будем, просто позволим себе использовать k_i так же, как мы себе позволили использовать $[x]_i$. «Длиной» такого «массива» будет тогда $\text{nsd}(x)$, которую мы определили ранее. Теперь попробуем решить несколько задач с этим:

Задача. $f(x) = \max_{i=1\dots r}(k_i)$, наибольшая степень простого делителя (или наибольшее число в списке)

Решение. Как мы бы решали такое на программировании? Просто бы у нас была некая внешняя переменная (инвариант цикла), и мы бы циклом шли по всем k_i , и обновляли эту переменную. Так и сделаем, $\varphi(x, i)$ — это наибольшая степень простого делителя *среди первых i делителей*. Определяется она очень просто:

$$\begin{aligned}\varphi(x, 0) &= 0 \\ \varphi(x, i + 1) &= \begin{cases} k_{i+1} & \text{если } k_{i+1} > \varphi(x, i) \\ \varphi(x, i) & \text{иначе} \end{cases} \\ f(x) &= \varphi(x, \text{nsd}(x))\end{aligned}$$

Если k_{i+1} больше, чем текущее сохраненное число, то мы сохраняем k_{i+1} , если же нет, то оставляем прошлое значение $\varphi(x, i)$.

Задача. $f(x)$ — число разных k_i в x .

Решение. Опять же, мы математики, так что пока ничего не оптимизируем. Какой самый логичный способ это сделать? Давайте просто для каждого числа проверять, встречалось ли она раньше, и если да, то делаем +1. Разумеется, для этого используем инвариант цикла $\varphi(x, i)$.

$$\begin{aligned}\varphi(x, 0) &= 0 \\ \varphi(x, i + 1) &= \varphi(x, i) + \begin{cases} 1 & \text{если } k_{i+1} \text{ встречается впервые} \\ 0 & \text{иначе} \end{cases} \\ f(x) &= \varphi(x, \text{nsd}(x))\end{aligned}$$

Теперь вопрос, что делать с «встречается впервые». Давайте определим для этого функцию $g(x, i+1)$, которая равна 1, если k_{i+1} уже встречалось среди $k_1 \dots k_i$ и 0, если еще нет. Для этого определим второй инвариант цикла: $\psi(x, i, j) = 1$, если k_i встречалось среди $k_1 \dots k_j$ и 0 если нет. Тогда k_{i+1} будет встречаться впервые, если $g(x, i+1) = \psi(x, i+1, i) = 0$. Но как мы это определим? Да легко, мы будем ставить 1, как только находим то, что нужно, и оставлять старое значение, если не нашли. Это делается так:

$$\begin{aligned}\psi(x, i, 0) &= 0 \\ \psi(x, i, j+1) &= \begin{cases} 1 & \text{если } k_{j+1} = k_i \\ \psi(x, i, j) & \text{если } k_{j+1} \neq k_i \end{cases}\end{aligned}$$

И тогда

$$\begin{aligned}\varphi(x, 0) &= 0 \\ \varphi(x, i+1) &= \varphi(x, i) + \begin{cases} 1 & \text{если } \overline{\text{sg}}(\psi(x, i+1, i)) \\ 0 & \text{если } \psi(x, i+1, i) \end{cases} \\ f(x) &= \varphi(x, \text{nsd}(x))\end{aligned}$$

Вернемся-ка мы обратно к десятичной записи:

Задача. $f(x)$ — наименьшая из наиболее часто встречающихся цифр в десятичной записи x .

Решение. Как бы мы писали программу для решения этого? Нам нужно сначала просто посчитать «массив» из частот всех цифр. Потом мы пройдемся по этому массиву от меньших цифр к большим и найдем ту, которая встречается чаще всего. Причем если какая то из них будет встречаться столько же раз, мы оставим старое значение. Правда массивов у нас тут нет, вместо этого у нас будет функция $\text{count}(x, n)$ — количество цифр n в числе x . Определим ее через инвариант $\psi(x, n, i)$:

$$\begin{aligned}\psi(x, n, 0) &= 0 \\ \psi(x, n, i+1) &= \psi(x, n, i) + \begin{cases} 1 & \text{если } [x]_{i+1} = n \\ 0 & \text{если } [x]_{i+1} \neq n \end{cases} \\ \text{count}(x, n) &= \psi(x, n, \|x\|)\end{aligned}$$

Далее определим наш инвариант для основной функции, $\varphi(x, n)$:

$$\begin{aligned}\varphi(x, 0) &= 0 \\ \varphi(x, n+1) &= \begin{cases} n+1 & \text{если } \text{count}(n+1) > \text{count}(\varphi(x, n)) \\ \varphi(x, n) & \text{если } \text{count}(n+1) \leq \text{count}(\varphi(x, n)) \end{cases} \\ f(x) &= \varphi(x, 9)\end{aligned}$$

То есть мы заменяем старое значение на новое только если оно встречается строго чаще, чем старое. Если ровно столько же, то мы выбираем то, которое меньше, т.е. оставляем старое.

Задача. $f(x)$ — максимальная длина последовательности подряд идущих нулей в десятичной записи x .

Решение. Намного более интересная задача. Итак, снова, как бы мы решали ее, если бы писали программу? Мы бы шли по разрядам, и как только встречали бы 0, увеличивали бы счетчик длины (назовем его $\text{length}(x, i)$), а если бы встречали не 0, то мы бы его сбрасывали. Давайте пока напомним эту часть:

$$\begin{aligned}\text{length}(x, 0) &= 0 \\ \text{length}(x, i + 1) &= \begin{cases} \text{length}(x, i) + 1 & \text{если } [x]_{i+1} = 0 \\ 0 & \text{если } [x]_{i+1} \neq 0 \end{cases}\end{aligned}$$

Правда нам теперь надо найти максимальную длину. Не сложно, мы можем просто перебрать все значения i и найти значение нашей максимальной длины:

$$\begin{aligned}\varphi(x, 0) &= 0 \\ \varphi(x, i + 1) &= \begin{cases} \text{length}(x, i + 1) & \text{если } \text{length}(x, i + 1) > \varphi(x, i) \\ \varphi(x, i) & \text{иначе} \end{cases} \\ f(x) &= \varphi(x, \|x\|)\end{aligned}$$

11.7 Задача 7

Необходимые темы: примитивно рекурсивные функции (раздел 7.1).

Также рекомендуется почитать предыдущий «бонусный» раздел (с. 226)

Задачи 7 и 8 очень похожи, в них требуется описать определение некоторой очень запутанной функции, работающих на «списках». Предупреждаю, они хоть и не особо сложные, но очень затяжные, поэтому набирайтесь терпения. В 7-й задаче это списки, основанные на разложениях на простые числа, а в 8-й — десятичные записи чисел, но это все равно ничего не меняет, только обозначения, общий принцип решения все равно один и тот же. Еще мы будем работать с подпоследовательностями чисел. Задавать мы их будем обычно индексом начала i и длиной подпоследовательности j . Еще важно научиться вычислять все нужные индексы, примеры чего я приведу в решении задач. Пока правда стоит запомнить, что если последовательность начинается в k_i и имеет длину j , то она закончится не в k_{i+j} , а в k_{i+j-1} . Итак, задача:

Задача. Пусть $x = p_1^{k_1} \cdot p_2^{k_2} \cdot \dots \cdot p_r^{k_r}$ и $y = h_1^{d_1} \cdot h_2^{d_2} \cdot \dots \cdot h_s^{d_s}$. Определить примитивно-рекурсивно функцию $f(x, y)$ — число разных симметричных последовательностей подряд идущих элементов из k_1, \dots, k_r , которые (последовательности) входят в d_1, \dots, d_s четное число раз.

Решение. Куча всего и непонятно, с чего начать, да? Давайте сначала найдем ключевые моменты условия. 1) число последовательностей, которые 2) разные, 3) симметричные, 4) входят в d_1, \dots, d_s четное число раз. Будем разбирать это по очереди, для начала, нам нужно найти число последовательностей, которые удовлетворяют условиям 2, 3, 4. Для этого нам надо перебрать все возможные последовательности внутри k_1, \dots, k_r . Как мы это сделаем? Каждая последовательность задается ее началом i и длиной j . Для начала, будем перебирать разные начала (ба-дум-тсс) последовательностей i нашим инвариантом цикла. То есть определим $\varphi(x, y, i)$ — число последовательностей, удовлетворяющих 2, 3, 4, с началами среди k_1, \dots, k_i . Тогда когда i дойдет до $r = \text{nsd}(x)$, мы переберем все: $f(x, y) = \varphi(x, y, \text{nsd}(x))$. Инвариант мы определим как всегда рекурсивно:

$$\begin{aligned}\varphi(x, y, 0) &= 0 \\ \varphi(x, y, i + 1) &= \varphi(x, y, i) + \text{new}(x, y, i + 1)\end{aligned}$$

Число последовательностей с началами среди k_1, \dots, k_{i+1} — это число последовательностей с началами среди k_1, \dots, k_i плюс число последовательностей с началом ровно в k_{i+1} . Это число, как вы догадались, $\text{new}(x, y, i + 1)$. Правда это $i + 1$ — это из-за того, что мы используем ее в примитивной рекурсии, которой определяем φ , так то функция new имеет аргументы $\text{new}(x, y, i)$ и считает число последовательностей с началом в k_i . Запомните эту деталь — чаще всего мы будем подставлять в функцию аргумент $i + 1$ или $j + 1$, но в определении самой функции вместо него будет просто i . Это важно понимать. Так вот, $\text{new}(x, y, i + 1)$ — число последовательностей с началом в k_{i+1} , а $\text{new}(x, y, i)$ — с началом в k_i .

Мы еще не знаем длину. Как мы будем определять $\text{new}(x, y, i)$? Конечно снова инвариант цикла! $\psi(x, y, i, j)$ — число таких последовательностей с началом в k_i и длиной $\leq j$. Как определяем его? Конечно снова примитивная рекурсия!

$$\begin{aligned}\psi(x, y, i, 0) &= 0 \\ \psi(x, y, i, j + 1) &= \psi(x, y, i, j) \\ &+ \begin{cases} 1 & \text{если} \\ & 1) k_i \dots k_{i+j} \text{ встречается впервые} \\ & 2) k_i \dots k_{i+j} \text{ — симметрична} \\ & 3) k_i \dots k_{i+j} \text{ входит в } d_1 \dots d_s \text{ четное число раз} \\ 0 & \text{иначе} \end{cases}\end{aligned}$$

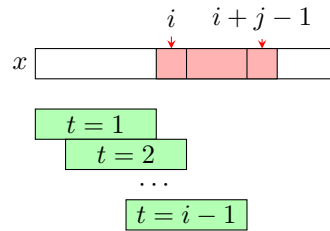
$k_i \dots k_{i+j}$ — просто последовательность с началом в i и длиной $j + 1$. Почему так? А вы посчитайте, если длина 1, то заканчивается в i , если длина 2 — в $i + 1$, если 3 — в $i + 2$ (на один меньше), если j — то в $i + (j - 1)$, а если $j + 1$ — в $i + j$. Назовем ее α . Мы добавляем к нашему счетчику 1, если α удовлетворяет всем нашим требованиям. Теперь стоило бы это записать более формально. Для этого определим кучу функций, каждая из которых

будет проверять одно из условий. α встречается впервые — $\text{first}(x, i, j)$, α симметрична — $\text{sym}(x, i, j)$, α входит четное число раз... нет. Пусть у нас будет функция $\text{number}(x, y, i, j)$ ³, которая считает, сколько раз входит α в $d_1 \dots d_s$, а потом уже будем проверять на четность. Заметьте, что в первых 2-х функциях нет аргумента y , потому что им он и не нужен, они проверяют только последовательность $k_1 \dots k_r$, которая определена в x , а последовательность $d_1 \dots d_s$ использует только последнее условие. Тогда определение принимает вид:

$$\begin{aligned} \psi(x, y, i, 0) &= 0 \\ \psi(x, y, i, j+1) &= \psi(x, y, i, j) \\ &+ \begin{cases} 1 & \text{если } 1) \text{first}(x, i, j+1) \\ & 2) \text{sym}(x, i, j+1) \\ & 3) \text{mod}(\text{number}(x, y, i, j+1), 2) = 0 \\ 0 & \text{иначе} \end{cases} \end{aligned}$$

Так вот, нам надо определить их все. Пойдемте по очереди:

$\text{first}(x, i, j) = 1$, если последовательность $k_i \dots k_{i+j-1}$ (длины j , да, поэтому последний индекс $i + (j - 1)$) не встречалась раньше и 0, если встречалась. Для этого стоит проверить все последовательности (β) такой же длины, которые встречались раньше и вернуть 1, если они все другие. Заметьте, нам не обязательно проверять эти последовательности на условия sym и number , потому что если наша основная α , потому что если α их проходит, то и β тоже, и мы должны считать их обе только один раз. Для понимания происходящего полезно рисовать рисунки:



t — начала последовательностей β (зеленых), которые мы все будем сравнивать с красной последовательностью α . Мы будем перебирать t от 1 до $i-1$. И да, длина и красных, и зеленой последовательности равна j . Начнем? Нам снова нужен инвариант цикла, назовем его $\lambda(x, i, j, t)$. Это логическое значение и оно будет равно 1, если *все* последовательности не будут равны α . Если же хотя бы одна из них нарушила условие, то мы ставим его на 0. Поэтому в таких ситуациях начальное значение инварианта будет 1 — мы еще не просмотрели последовательности, так что пока что условия не

³В оригинале она называлась $\text{col}(x, i, j, y)$, от слова «количество», но мне слишком больно так писать.

нарушены. Как только будут, мы установим это в 0.

$$\begin{aligned}\lambda(x, i, j, 0) &= 1 \\ \lambda(x, i, j, t+1) &= \begin{cases} 0 & \text{если } \alpha = \beta \\ \lambda(x, i, j, t) & \text{иначе} \end{cases}\end{aligned}$$

Как мы их только сравнивать будем? Нужно определить некоторую функцию $\text{equal}(x, i, j, t)$, которая будет проверять последовательности на равенство. Правда давайте заранее посмотрим на 3-е условие, нам надо будет проверять, сколько раз α входит в последовательность, задаваемую y . Так что давайте сразу определим функцию equal так, чтобы она поддерживала сравнение последовательностей из разных «списков»: $\text{equal}(x, y, i, j, t)$. Она будет равна 1, если $k_i \dots k_{i+j-1}$ совпадает с $d_t \dots d_{t+j-1}$. Если же мы будем сравнивать обе последовательности из x , как сейчас, то проблем тоже нет: $\text{equal}(x, x, i, j, t)$, 1, если $k_i \dots k_{i+j-1}$ совпадает с $k_t \dots k_{t+j-1}$. Так вот, тогда наше условие преобразуется в

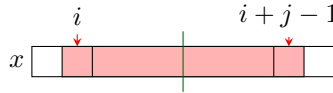
$$\begin{aligned}\lambda(x, i, j, 0) &= 1 \\ \lambda(x, i, j, t+1) &= \begin{cases} 0 & \text{если } \text{equal}(x, x, i, j, t+1) \\ \lambda(x, i, j, t) & \text{иначе} \end{cases}\end{aligned}$$

А как мы будем определять $\text{equal}(x, y, i, j, t)$? Инвариант, а вот и нет. Можем конечно, но можем заметить, что мы можем вести рекурсию просто по длине последовательности j . При $j = 0$ последовательности всегда равны, а далее мы можем просто сравнивать отдельные символы. Так мы иногда можем обойтись без инварианта цикла, не создавая лишних переменных, просто используя те, которые уже есть. Это выглядит так:

$$\begin{aligned}\text{equal}(x, y, i, 0, t) &= 1 \\ \text{equal}(x, y, i, j+1, t) &= \begin{cases} 0 & \text{если } k_{i+j} \neq d_{t+j} \\ \text{equal}(x, y, i, j, t) & \text{иначе} \end{cases}\end{aligned}$$

То есть мы сбрасываем значение функции на 0, если встречаем разные числа, а если они равны, то мы оставляем старое значение (1, если предыдущие числа были равны и 0, если мы уже нашли отличия). Определили. На этом все с нашей функцией $\text{first}(x, i, j)$.

Теперь $\text{sum}(x, i, j)$. Снова, рисунок:



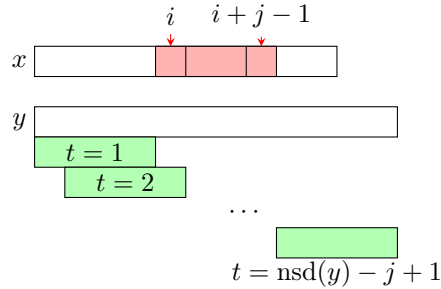
Как всегда, будем идти по t при помощи инварианта цикла, $\mu(x, i, j, t)$, где t — количество проведенных сравнений. Всего нам нужно будет провести j сравнений (на самом деле достаточно просто дойти до середины,

но нам нет смысла оптимизировать это). Так вот, $\text{sym}(x, i, j) = \mu(x, i, j, j)$. Очевидно, что без проверок мы пока считаем наше слово симметричным, потому что симметричность еще не нарушена: $\mu(x, i, j, 0) = 1$. Что делать с $\mu(x, i, j, t+1)$? Нам нужно сравнить $t+1$ -й элемент с начала с $t+1$ -м с конца. Как мы это вычислим? Как всегда, просто выписав последовательность. Для начала, 1-й элемент с начала — это i , 2-й — $i+1$, 3-й — $i+2$, (на 1 меньше), тогда $t+1$ -й — $i+t$. А с конца? 1-й с конца — $i+j-1$, 2-й — $i+j-2$, 3-й — $i+j-3$ (вычитаем то же самое), тогда $t+1$ -й — $i+j-(t+1) = i+j-t-1$. Ну вот и все, нам надо сравнивать элемент $i+t$ с элементом $i+j-t-1$:

$$\begin{aligned}\mu(x, i, j, 0) &= 1 \\ \mu(x, i, j, t+1) &= \begin{cases} 0 & \text{если } k_{i+t} \neq k_{i+j-t-1} \\ \mu(x, i, j, t) & \text{иначе} \end{cases}\end{aligned}$$

С $\text{sym}(x, i, j)$ разобрались, идем дальше.

$\text{number}(x, y, i, j)$ — сколько раз последовательность $k_i \dots k_{i+j-1}$ встречается в $d_1 \dots d_s$. Картинка:



Мы будем просто сравнивать все последовательности длины j в y с нашей α . t — индекс начала этой последовательности. Какой последний нам надо будет проверить? Это даже можно посчитать даже на конкретных числах: пусть $s = \text{nsd}(y) = 20$, т.е. в $d_1 \dots d_s$ у нас 20 чисел. Длина нашей последовательности $j = 5$. Тогда если эта последовательность в самом конце, то до нее будет еще $15 = \text{nsd}(y) - j$ чисел. Тогда какой номер у первого элемента последовательности? 16! Или $t = \text{nsd}(y) - j + 1$. Итак, инвариант цикла, последний, на сей раз: $\eta(x, y, i, j, t)$. Причем мы определяем $\text{number}(x, y, i, j) = \eta(x, y, i, j, \text{nsd}(y) - j + 1)$. Что эта η должна делать? Она должна считать количество таких же последовательностей, как α , в $d_1 \dots d_s$ с началами в $d_1 \dots d_t$. То есть при $t = 0$ их 0, а потом мы просто будем добавлять 1, если эта последовательность равна α . Как раз для этого мы и определили нашу equal так, как определили:

$$\begin{aligned}\eta(x, y, i, j, 0) &= 0 \\ \eta(x, y, i, j, t+1) &= \eta(x, y, i, j, t) + \begin{cases} 1 & \text{если } \text{equal}(x, y, i, j, t+1) \\ 0 & \text{иначе} \end{cases}\end{aligned}$$

Вот и все, на этом наше решение задачи закончено.

11.8 Задача 8

Необходимые темы: примитивно рекурсивные функции (раздел 7.1).

Также рекомендуется почитать «бонусный» раздел (с. 226)

Эта задача очень похожа на предыдущую, так что просто приступим.

Задача. $x = \sigma_n \cdots \sigma_1$ и $y = \delta_m \cdots \delta_1$ — десятичные записи x и y . Определить $f(x, y)$ — максимальную длину последовательности подряд идущих «0»-й в записи x , которые симметрично входят в запись y хотя бы 1 раз.

Решение. Для начала, важные компоненты. 1) максимум длин 2) последовательностей подряд идущих «0»-й 3) встречаются в y симметрично хотя бы 1 раз. Обратим внимание на то, что Константин Иванович понимает под «последовательностью подряд идущих 0-й», например, рассмотрим $x = 1230000040005$. В этом числе *две* последовательности подряд идущих нулей: 00000 и 000. То есть те более маленькие последовательности нулей не учитываются. То есть в его интерпретации если $x = 1230000040005$ и $y = 6007008$ должны выдать $f(x, y) = 0$, потому что в x встречаются только последовательности 00000 и 000, которые ни разу не встречаются в y . Последовательность же 00, которая кучу раз встречается внутри x , но внутри других последовательностей (00000 и 000), не считается. Да, мне тоже немного это странно, но видимо так.

Так вот. В таком случае последовательность задается только ее началом, потому что длину последовательности мы всегда можем потом вычислить по началу, пусть это будет функция $\text{length}(x, i)$, которую мы потом определим. Итак, мы будем просто перебирать начала при помощи инварианта цикла $\varphi(x, y, i)$ — максимум длин 2, 3, которые начинаются в первых i разрядах x :

$$\begin{aligned} \varphi(x, y, 0) &= 0 \\ \varphi(x, y, i+1) &= \begin{cases} \text{length}(x, i+1) \text{ если} \\ \quad \begin{aligned} &1) \text{ в } [x]_{i+1} \text{ начинается новая посл-ть нулей} \\ &2) \text{ length}(x, i+1) > \varphi(x, y, i) \\ &3) \text{ посл-ть нулей длины } \text{length}(x, i+1) \\ &\quad \text{встречается в } y \text{ сим-чно хотя бы 1 раз} \end{aligned} \\ \varphi(x, y, i) \text{ иначе} \end{cases} \end{aligned}$$

Что значит «в $[x]_{i+1}$ начинается новая последовательность нулей»? А то, что $[x]_{i+1}$ — первый 0, т.е. до него нуля не было. То есть $[x]_{i+1} = 0 \ \& \ [x]_i \neq 0$. Хотя есть еще вариант, это вообще самая первая цифра, так что правильно $[x]_{i+1} = 0 \ \& \ ([x]_i \neq 0 \vee i = 0)$ Второе условие мы уже прекрасно разобрали,

а третье давайте просто запишем в функцию $\text{sum}(x, y, i)$:

$$\begin{aligned}\varphi(x, y, 0) &= 0 \\ \varphi(x, y, i+1) &= \begin{cases} \text{length}(x, i+1) & \text{если } 1) \quad [x]_{i+1} = 0 \& ([x]_i \neq 0 \vee i = 0) \\ & 2) \quad \text{length}(x, i+1) > \varphi(x, y, i) \\ & 3) \quad \text{sym}(x, y, i+1) \\ \varphi(x, y, i) & \text{иначе} \end{cases}\end{aligned}$$

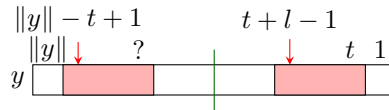
А теперь попробуем определить наш $\text{sum}(x, y, i) - 1$, если последовательность нулей длины $\text{length}(x, i)$ встречается в y симметрично хотя бы 1 раз, и 0 иначе. Как всегда, будем просто перебирать все разряды и смотреть, не начинается ли там такая последовательность при помощи инварианта цикла $\psi(x, y, i, t) - 1$, если такая последовательность встречается симметрично хотя бы 1 раз в *первых t разрядах*, и 0 иначе. Тогда ее можно определить так:

$$\begin{aligned}\psi(x, y, i, 0) &= 0 \\ \psi(x, y, i, t+1) &= \begin{cases} 1 & \text{если} \\ & 1) \text{ в } [y]_{t+1} \text{ начинается посл-ть нулей} \\ & \text{длины } \text{length}(x, i) \\ & 2) \text{ она входит в } y \text{ симметрично} \\ \psi(x, y, i, t) & \text{иначе} \end{cases}\end{aligned}$$

Первое расписать достаточно просто:

$$[y]_{t+1} = 0 \& ([y]_t \neq 0 \vee t = 0) \& \text{length}(y, t+1) = \text{length}(x, i)$$

А вот для второго надо снова обратиться к рисуночку:



Не особо понятный рисуночек, так что давайте лучше обратимся к конкретному числу: $y = 10003450006$. У него $\|y\| = 11$, и мы рассмотрим правую последовательность нулей, начинающуюся в $t+1 = 2$ и заканчивающуюся в 4. Длина у нее $\text{length}(x, i) = 3$. Как мы получили 4? $4 = 2 + (3 - 1) = (t+1) + \text{length}(x, i) - 1 = t + \text{length}(x, i)$. Логично. А что с симметрией? Ну разряду 1 симметричен разряд $\|y\| = 11$, разряду 2 симметричен $\|y\| - 1 = 10$, разряду 3 — разряд $\|y\| - 2 = 9$, значит разряду $t+1$ соответствует $\|y\| - t = 10$. А разряду $t + \text{length}(x, i)$? Разряд $\|y\| - (t + \text{length}(x, i) - 1) = \|y\| - t - \text{length}(x, i) + 1 = 8$, что верно! Итого если наша цепочка начинается в $t+1$, то симметричная ей цепочка должна начинаться в $\|y\| - t - \text{length}(x, i) + 1$ и иметь ту же длину. То есть второе

условие преобразуется в

$$\begin{aligned} [y]_{\|y\| - t - \text{length}(x, i) + 1} &= 0 \& \\ [y]_{\|y\| - t - \text{length}(x, i)} &\neq 0 \& \\ \text{length}(y, \|y\| - t - \text{length}(x, i) + 1) &= \text{length}(x, i) \end{aligned}$$

Итого наша функция будет определена как

$$\begin{aligned} \psi(x, y, i, 0) &= 0 \\ \psi(x, y, i, t + 1) &= \begin{cases} 1, \text{ если} \\ 1) [y]_{t+1} = 0 \& ([y]_t \neq 0 \vee t = 0) \& \\ \quad \text{length}(y, t + 1) = \text{length}(x, i) \\ 2) [y]_{\|y\| - t - \text{length}(x, i) + 1} = 0 \& \\ \quad [y]_{\|y\| - t - \text{length}(x, i)} \neq 0 \& \\ \quad \text{length}(y, \|y\| - t - \text{length}(x, i) + 1) = \text{length}(x, i) \\ \psi(x, y, i, t), \text{ иначе} \end{cases} \end{aligned}$$

Мы не определили только одну функцию — $\text{length}(x, i)$ — длина последовательности нулей, начинающейся в $[x]_i$. Есть много способов ее определить. Тут проблема в том, что подсчет должен остановиться, как только нули закончатся, и не начинаться, даже если они снова начнутся. Для начала, нам явно потребуется инвариант цикла, $\lambda(x, i, j)$, j — число уже пройденных цифр. До i -й цифры уже было $i - 1$ цифр, которые проходить не надо, так что всего нам потребуется пройти $\|x\| - (i - 1) = \|x\| - i + 1$ цифр: $\text{length}(x, i) = \lambda(x, i, \|x\| - i + 1)$. Очевидно, что $\lambda(x, i, 0) = 0$. Давайте пока первую версию напомним:

$$\begin{aligned} \lambda(x, i, 0) &= 0 \\ \lambda(x, i, j + 1) &= \begin{cases} j + 1 & \text{если } [x]_{j+1} = 0 \\ \lambda(x, i, j) & \text{иначе} \end{cases} \end{aligned}$$

То есть мы ставим $j + 1$ (могли бы просто увеличить старое значение на 1, но вы увидите, почему так лучше), если мы встретили 0, и оставляем что было, если нули кончились. Проблема в том, что если нули снова начнутся, то они снова начнут считаться. Но есть один важный момент, заметьте, что $\lambda(x, i, j) = j$ тогда, когда мы еще находимся в пределах последовательности. Тогда мы можем просто добавить это как дополнительное условие:

$$\begin{aligned} \lambda(x, i, 0) &= 0 \\ \lambda(x, i, j + 1) &= \begin{cases} j + 1 & \text{если } [x]_{j+1} = 0 \& \lambda(x, i, j) = j \\ \lambda(x, i, j) & \text{иначе} \end{cases} \end{aligned}$$

И теперь мы и правда определили все функции. Не беспокойтесь, то, как мы определили $\text{length}(x, i)$ — специальный трюк, который не обязательно уметь самому выполнять, просто запомните, что конкретно $\text{length}(x, i)$ можно определить вот так. На этом задача все.

11.9 Бонусный раздел, построение систем Поста

В следующих задачах от нас будет требоваться построить системы Поста, строящие слова с определенными свойствами. Поэтому важно сейчас разобратся, как именно их строить и как они работают. Во первых, работать мы сейчас будем с алфавитом $A = \{0, 1\}$ и алфавитом B каким придется, он не особо важен. Главное, что мы подразумеваем, что мы работаем в двоичной системе. Итак, задача

Задача. Построить систему Поста, которая будет выводить все пары слов (α, β) , где $\alpha, \beta \in \{0, 1\}^*$, и $\beta = \alpha^{-1}$, т.е. β является развернутым словом α .

Система Поста - это просто куча продукций. Каждая продукция — просто правило по преобразованию слов. Система Поста работает так: у нее есть некоторое множество уже построенных слов (изначально пустое), и на каждом шаге она берет какие-то слова из этого множества и применяет к ним одну из своих продукций. После этого появляется новое слово, которое система добавляет в множество уже построенных слов. А теперь к решению.

Решение. Для начала нам обязательно нужна аксиома — какое-то начальное слово/запись, которое всегда можно добавить в систему. Обычно это самые короткие слова, удовлетворяющие условию задачи, то есть в нашем случае — пустые. Давайте их обозначать как $_$. Тогда наша аксиома будет $\pi_1 = (_, _)$. Далее, нам нужен способ получить любые слова, удовлетворяющие условию. Для этого мы будем добавлять к ним символы по одному. Если мы знаем, что в системе есть пара слов (x, y) , т.е. что $y = x^{-1}$ (y — это развернутый x), тогда условию будут удовлетворять и слова $(x0, 0y)$ и $(x1, 1y)$ ($0y$ — это развернутый $x0$, а $1y$ — развернутый $x1$). Это и есть 2 необходимые нам продукции. Тогда системой Поста будет:

$$\begin{aligned}\pi_1 &= (_, _) \\ \pi_2 &= \frac{(x, y)}{(x0, 0y)} \\ \pi_3 &= \frac{(x, y)}{(x1, 1y)}\end{aligned}$$

Задача. Система Поста, выводящая (α, β) , где в α и β одинаковое число нулей.

Решение. Для начала, аксиома $\pi_1 = (_, _)$. Для начала, нам нужен способ добавлять нули в слова. Причем такой, чтобы он всегда сохранял одинаковое число нулей слева и справа. Легко: если в (x, y) одинаковое число нулей, то в $(x0, y0)$ очевидно тоже. Это будет наша продукция:

$$\pi_2 = \frac{(x, y)}{(x0, y0)}$$

А вот теперь будем добавлять и единицы. Единицы можно добавлять свободно, как в слово α , так и в β (учтите, что x, y — просто переменные, они не обязательно левые и правые слова). Это и сделаем, разобьем слово α как xy , а β оставим z . Тогда если в xy столько же нулей, сколько в z , то мы можем спокойно добавить единицу между x и y :

$$\pi_3 = \frac{(xy, z)}{(x1y, z)}$$

Аналогично и со вторым словом в паре:

$$\pi_4 = \frac{(x, yz)}{(x, y1z)}$$

Вот и вся наша система Поста. Правда ее можно и упростить, если мы заметим, что мы можем всегда добавлять цифры в конец слов. То есть мы сначала добавляем сколько надо (разное число) единиц в левое и правое слова, потом сколько надо нулей (одинаковое число), потом сколько надо (разное) единиц и т.д. Тогда подойдет и система Поста

$$\begin{aligned}\pi_1 &= (_, _) \\ \pi_2 &= \frac{(x, y)}{(x0, y0)} \\ \pi_3 &= \frac{(x, y)}{(x1, y)} \\ \pi_4 &= \frac{(x, y)}{(x, y1)}\end{aligned}$$

Давайте на всякий случай рассмотрим пример построения пары $(1001, 1101110)$ с помощью этой системы Поста. Сначала применяем аксиому π_1 , получаем $(_, _)$. Потом применяем π_3 один раз, получаем $(1, _)$. Потом π_4 два раза, получаем $(1, 11)$. Далее применяем π_2 , получаем $(10, 110)$. Дальше цепочка такая:

$$\begin{aligned}(10, 110) &\xrightarrow{\pi_4 \times 3} \\ (10, 110111) &\xrightarrow{\pi_2} \\ (100, 1101110) &\xrightarrow{\pi_3} \\ (1001, 1101110) &\end{aligned}$$

Следующая задача?

Задача. (α, β) , причем число единиц в α больше, чем число единиц в β .

Решение. Мы не можем сделать $(_, _)$, потому что здесь строго больше. Ничего, подойдет $\pi_1 = (1, _)$. Как мы выяснили, мы можем строить слова слева направо, т.е. просто добавлять цифры к концу слова. Давайте

сначала разберемся с единицами: мы можем или оставить разрыв между количествами единиц таким же, либо увеличить его. Вот 2 продукции для этого:

$$\begin{aligned}\pi_2 &= \frac{(x, y)}{(x1, y1)} \\ \pi_3 &= \frac{(x, y)}{(x1, y)}\end{aligned}$$

А с нулями то же самое, ставим сколько хотим:

$$\begin{aligned}\pi_4 &= \frac{(x, y)}{(x0, y)} \\ \pi_5 &= \frac{(x, y)}{(x, y0)}\end{aligned}$$

Задача. Пары (α, β) , где β получается из α сжатием всех групп нулей в один 0.

Решение. Давайте будем собирать слово α посимвольно слева направо. Начнем с пустых слов, $\pi_1 = (_, _)$. Если мы добавляем к α единицу, то она должна добавиться и к β , так как они не меняются:

$$\pi_2 = \frac{(x, y)}{(x1, y1)}$$

А вот с нулями сложнее. То, нужно ли ставить 0 в β , мы можем узнать из предыдущей цифры, которую мы добавили в α . Если прошлая цифра была 0, то новый 0 добавлять в β не нужно, мы находимся в группе нулей. А если прошлая цифра была 1, то нужно добавить 0. Это записывается следующими продуктами:

$$\begin{aligned}\pi_3 &= \frac{(x0, y)}{(x00, y)} \\ \pi_4 &= \frac{(x1, y)}{(x10, y0)}\end{aligned}$$

То есть если $\alpha = x0$, т.е. заканчивается на 0, то при добавлении нуля к α ($\alpha' = x00$) мы не должны добавлять новый ноль к β , потому что один 0 там уже есть, больше не надо. А если же $\alpha = x1$, то тогда нам нужно добавить 0. Это и есть полное решение. Почти. Заметим одну вещь — в продукциях π_3 и π_4 слово α уже должно быть как минимум односимвольным, в нем должен быть или 0, или 1. Но что делать, если мы добавляем первый символ (ноль) к слову α ? Добавим еще одну аксиому: $\pi_5 = (0, 0)$. Так все уже будет работать. Правда приведу еще одно решение: мы можем смотреть на последний символ не α , а β , т.е. если последний символ был 0, то добавлять не нужно, а если 1, то нужно. Тогда получим другую, но абсолютно

эквивалентную систему Поста:

$$\begin{aligned}\pi'_1 &= (_, _) \\ \pi'_2 &= \frac{(x, y)}{(x1, y1)} \\ \pi'_3 &= \frac{(x, y0)}{(x0, y0)} \\ \pi'_4 &= \frac{(x, y1)}{(x0, y10)} \\ \pi'_5 &= (0, 0)\end{aligned}$$

Задача. Пары (α, β) , где α получается из β сжатием всех групп нулей в один 0, а β из α — сжатием всех групп единиц в одну 1.

Решение. Похожая задача. Начнем с $\pi_1 = (_, _)$. Когда мы добавляем нули к β , мы можем как добавить такие же нули к α , так и не добавлять, по тому же принципу, как и раньше. Так что продукции будут аналогичные (но возьмем из второго варианта решения)

$$\begin{aligned}\pi_2 &= \frac{(x0, y)}{(x0, y0)} \\ \pi_3 &= \frac{(x1, y)}{(x10, y0)}\end{aligned}$$

То есть если $\alpha = x0$, то мы к нему ничего не добавляем и оставляем $x0$, а если $x1$, то добавляем к нему 0 и получаем $x10$. Абсолютно аналогично поступаем и с единицами, но в другую сторону:

$$\begin{aligned}\pi_4 &= \frac{(x, y0)}{(x1, y01)} \\ \pi_5 &= \frac{(x, y1)}{(x1, y1)}\end{aligned}$$

Вот и все решение. Почти. Нужны еще аксиомы, все наши продукции требуют как минимум одного символа в хотя бы одном из слов, а у нас тут только пустые. Так что добавим еще 2 аксиомы:

$$\begin{aligned}\pi_6 &= (0, 0) \\ \pi_7 &= (1, 1)\end{aligned}$$

Ну что ж, закончили с задачами. Есть правда еще несколько очень полезных систем Поста, которые мы будем использовать как компоненты в зачетных задачах. Они все используют вспомогательные символы или *маркеры*. Рассмотрим эти системы:

1. Wx — цепочка из 0 и 1. Эта система определяет слова вида Wx , где x — цепочка из 0 и 1, а W — просто маркер, который обозначает, что

это слово является такой цепочкой. У всех других таких цепочек будут другие маркеры. Так вот, если такое слово вдруг появится в множестве уже полученных, то это будет гарантировать, что оно состоит из 0 и 1. Так вот, как определяется? Так:

$$\begin{aligned}\pi_1 &= W_ \\ \pi_2 &= \frac{Wx}{Wx0} \\ \pi_3 &= \frac{Wx}{Wx1}\end{aligned}$$

Буквально 1) «Пустое слово — цепочка из 0 и 1», 2) «Если x — цепочка, то $x0$ — тоже» и 3) «Если x — цепочка, то $x1$ — тоже»

2. Ox — цепочка из нулей. Достаточно очевидно:

$$\begin{aligned}\pi_1 &= O_ \\ \pi_2 &= \frac{Ox}{Ox0}\end{aligned}$$

3. NOx — цепочка не из нулей (т.е. там есть хотя бы одна 1). Не так очевидно, у нас нет отрицания здесь. Но мы можем достаточно легко обойти это: цепочки не из нулей — это те слова, где есть хотя бы одна единица, т.е. слова вида $x1y$. Причем x, y сами должны быть цепочками. Тогда это можно записать одной продукцией:

$$\pi = \frac{Wx \quad Wy}{NOx1y}$$

4. Ix — цепочка из единиц.

$$\begin{aligned}\pi_1 &= I_ \\ \pi_2 &= \frac{Ix}{Ix1}\end{aligned}$$

5. NIx — цепочка не из единиц.

$$\pi = \frac{Wx \quad Wy}{NIx0y}$$

6. $x = y$ — x и y одинаковой длины. Для начала, $_ = _$. А еще мы можем просто добавлять любые символы туда, но в одинаковых количествах:

$$\begin{aligned}\pi_1 &= _ = _ \\ \pi_2 &= \frac{x = y}{x0 = y0} \\ \pi_3 &= \frac{x = y}{x0 = y1} \\ \pi_4 &= \frac{x = y}{x1 = y0} \\ \pi_5 &= \frac{x = y}{x1 = y1}\end{aligned}$$

Сокращенно буду это записывать как

$$\begin{aligned}\pi_1 &= _ = _ \\ \pi_{2-5} &= \frac{x = y}{\begin{array}{l} x0 = y0 \\ x0 = y1 \\ x1 = y0 \\ x1 = y1 \end{array}}\end{aligned}$$

7. $x < y - x$ короче y . Аксиомами являются $_ < 0$ и $_ < 1$. Мы можем безнаказанно добавлять нули и единицы справа, но если мы их добавляем слева, то мы должны добавить столько же справа. Итого получаем:

$$\begin{aligned}\pi_1 &= _ < 0 \\ \pi_2 &= _ < 1 \\ \pi_{3-4} &= \frac{x < y}{\begin{array}{l} x < y0 \\ x < y1 \end{array}} \\ \pi_{5-8} &= \frac{x < y}{\begin{array}{l} x0 < y0 \\ x0 < y1 \\ x1 < y0 \\ x1 < y1 \end{array}}\end{aligned}$$

8. $Sx - x$ — симметричная цепочка из 0 и 1. Аксиомой очевидно является $S_$. Основная продукция — добавление 0 слева и 0 справа, или 1 слева и 1 справа. Правда так мы получим только слова четной длины, поэтому нам еще нужны аксиомы $S0$ и $S1$. Итого получаем:

$$\begin{aligned}\pi_1 &= S_ \\ \pi_2 &= S0 \\ \pi_3 &= S1 \\ \pi_4 &= \frac{Sx}{S0x0} \\ \pi_5 &= \frac{Sx}{S1x1}\end{aligned}$$

9. $NSx - x$ — несимметричная цепочка из 0 и 1. Что значит быть несимметричной цепочкой? Это значит, что где-то на симметричных позициях стоят разные цифры. Например $x0y1z$, где 0 и 1 стоят на симметричных позициях. Что это значит? Что x и z одинаковой длины. Вот и все:

$$\pi_{1-2} = \frac{\begin{array}{cccc} Wx & Wy & Wz & y = z \end{array}}{\begin{array}{l} NSx0y1z \\ NSx1y0z \end{array}}$$

10. $E(x, y)$ — цепочки x и y одинаковы. Очень просто:

$$\pi = \frac{Wx}{E(x, x)}$$

Да, так можно. Если x — цепочка слов, то x равно x и никаких других пар равных слов нет.

11. $NE(x, y)$ — цепочки x и y разные. Ну, очевидно, что если одно меньше другого, то они точно разные:

$$\pi_{1-2} = \frac{x < y}{NE(x, y) \quad NE(y, x)}$$

Два раза, потому что не важно, какое из них стоит первым. А как быть со случаем, когда они одной длины? Что делает слова одной длины разными? Разные символы на одинаковых позициях! Пусть у нас есть 2 слова:

$$\begin{array}{c} u0v \\ z1p \end{array}$$

Они разные, потому что в одном стоит 0, а в другом 1. Но это верно только если длины u и z совпадают, а также длины v и p . Ну что ж, тогда имеем продукции

$$\pi_{3-4} = \frac{Wu \quad Wv \quad Wz \quad Wp \quad u = z \quad v = p}{NE(u0v, z1p) \quad NE(u1v, z0p)}$$

Правда еще одно, $u = z$ — это равные по длине цепочки 0 и 1. Поэтому нам не нужно заново проверять, что они цепочки 0 и 1, это за нас проверит =:

$$\pi_{3-4} = \frac{u = z \quad v = p}{NE(u0v, z1p) \quad NE(u1v, z0p)}$$

12. Mx — x — монотонная цепочка (по возрастанию/не убыванию), т.е. нечто вида $00 \cdots 011 \cdots 1$. Будем строить постепенно, аксиома: $\pi_1 = M_-$. Окей, идем дальше. Пусть у нас есть некое монотонное слово. Если оно заканчивается на 0, то значит мы еще не дошли до единиц и можем добавить что угодно, как 0, так и 1. Т.е. если наше слово имеет вид $x0$, то можно превратить его как в $x00$, так и в $x01$:

$$\pi_{2-3} = \frac{Mx0}{Mx00 \quad Mx01}$$

Но если наше слово заканчивается на 1, то можно продолжать его только единицами:

$$\pi_4 = \frac{Mx1}{Mx11}$$

Правда проблема — аксиома у нас только пустое слово, а все продукции требуют наличия или 0, или 1. Поэтому необходимо добавить еще аксиом:

$$\begin{aligned}\pi_5 &= M0 \\ \pi_6 &= M1\end{aligned}$$

Сноска, можно было сделать это все намного проще, а именно

$$\pi = \frac{Ox \quad Iy}{Mxy}$$

13. $NMx - x$ — не монотонна. Когда это случается? Когда где-то в слове после 1 идет 0. То есть это слова вида $x10y$. Так и запишем:

$$\pi = \frac{Wx \quad Wy}{NMx10y}$$

14. $Y(x, y)$ ⁴— $x = y^n$, т.е. x — сколько-то раз повторенное слово y . В первых, пустые x же считаются, да? Тогда продукция

$$\pi_1 = \frac{Wy}{Y(_, y)}$$

вполне подойдет. Далее нам потребуется просто добавлять слово y к x сколько-то раз, это делает продукция

$$\pi_2 = \frac{Y(x, y)}{Y(xy, y)}$$

15. $NY(x, y)$ ⁵— $x \neq y^n$. Намного сложнее, чем предыдущее. Что значит $x \neq y^n$? Значит, что при попытке замостить x y -ками, мы в какой-то момент наткнемся на отличие, например встретим на месте y слово z такой же длины, но другое: $x = yyy \cdots yz \cdots$. Тогда x будет гарантированно не равно $yyy \cdots y$. Нам не так важно, что идет после z , поэтому назовем его v . А нашу цепочку из y назовем p . Итак, если y , z и v — какие-то слова, p — цепочка из y , z — такой же длины как y , но другое, и $x = pzv$, то $NY(x, y)$. Тогда получим продукцию

$$\pi_1 = \frac{Wy \quad Wz \quad Wv \quad Y(p, y) \quad z = y \quad NE(z, y)}{NY(pzv, y)}$$

⁴От слова уес. Ну вы уже поняли, что это не моя вина, да?

⁵Да, это поуез, мне очень жаль

Как всегда в таких сложных задачах, стоит поискать особые случаи, которые эта продукция не покрывает. Мы предположили, что $x = yyy \cdots yzv$. Что может нарушиться? Ну, цепочки из y может и не быть, это не проблема, как и v . А что будет, если z там есть, но не полностью? Т.е. если z банально не помещается в слово x ? То есть $x = yyy \cdots yz$, но при этом z короче y , что-то такой же длины в слово не помещается. При этом еще надо учесть, что $z = _$ сюда не подойдет, потому что тогда $x = yyy \cdots y$, а это как раз $x = y^n$, чего мы не хотим. Так что мы должны учесть, что $z < y$, но $_ < z$. Соответственно мы получим продукцию

$$\pi_2 = \frac{Wy \quad Wz \quad Y(p, y) \quad z < y \quad _ < z}{NY(pz, y)}$$

16. $\text{ch } x - x$ — цепочка четной длины. Крайне простая задача, просто напишу продукции, думаю будет понятно

$$\begin{aligned} \pi_1 &= \text{ch } _ \\ \pi_{2-5} &= \frac{\text{ch } x}{\begin{array}{l} \text{ch } x00 \\ \text{ch } x01 \\ \text{ch } x10 \\ \text{ch } x11 \end{array}} \end{aligned}$$

17. $\text{nch } x - x$ — цепочка нечетной длины. Аналогично

$$\begin{aligned} \pi_1 &= \text{nch } 0 \\ \pi_2 &= \text{nch } 1 \\ \pi_{3-6} &= \frac{\text{nch } x}{\begin{array}{l} \text{nch } x00 \\ \text{nch } x01 \\ \text{nch } x10 \\ \text{nch } x11 \end{array}} \end{aligned}$$

Еще немножечко задач:

Задача. $\text{Max}(\alpha, \beta)$, где $\alpha \in \{0, 1\}^*$ — любое слово, а β — это самая длинная цепочка нулей в α .

Решение. Попробуем решить это старым методом, добавляя символы к α по одному. Для начала, аксиома: $\pi_1 = \text{Max}(_, _)$. Единицы мы можем безнаказанно добавлять к α , с ними ничего не будет, в y они не считаются:

$$\pi_2 = \frac{\text{Max}(x, y)}{\text{Max}(x1, y)}$$

Проблема с нулями. Когда мы добавляем ноль, должны ли мы удлинять β или нет? Это зависит от того, сколько нулей было в α до этого. Как нам

это найти? Давайте будем считать, что в α есть справа некоторая цепочка нулей v , которая слева ограничена 1: $\alpha = x1v$, где v — цепочка нулей. Тогда то, надо ли добавлять нули, зависит от длины v . Если длина v меньше текущего максимума, то даже если мы добавим к ней 0 ($v0$), то эта длина все равно будет не больше длины β , так что β останется максимальной длиной:

$$\pi_3 = \frac{\text{Max}(x1v, y) \quad Ov \quad v < y}{\text{Max}(x1v0, y)}$$

Например, если $\alpha = 100001100$, то $\beta = 0000$. На данный момент $v = 00$ (последняя цепочка), и даже если мы добавим 0 к α , β не поменяется. А что делать, если длина v равна β ? (Причем, учитывая, что v и β — обе цепочки нулей, тогда они будут полностью равны) Тогда когда мы добавим 0 к α , то там будет цепочка $v0$, которая длиннее β , тогда мы заменим ее на $v0$. Более того, так как мы добавляем по одному символу, мы никогда не окажемся в ситуации, когда v длиннее β , потому что на более ранних шагах мы бы это уже заменили. Итак, продукция:

$$\pi_4 = \frac{\text{Max}(x1v, v)}{\text{Max}(x1v0, v0)}$$

Да, нам не нужно даже проверять, что v — цепочка нулей, так как $\beta = v$ — цепочка нулей по определению. Есть правда еще одно, что мы упустили — когда мы добавляем нули, мы требуем, чтобы слово α содержало хотя бы одну единицу. А как получить α , которые просто цепочка нулей? Добавим еще одну продукцию и на этом все:

$$\pi_5 = \frac{Ox}{\text{Max}(x, x)}$$

Задача. $\text{Del}(\alpha, \beta)$, где $\alpha \in \{0, 1\}^*$ — любое слово, а β — это то, что остается от α после удаления всех цепочек нулей максимальной длины.

Решение. Попробуем снова собирать слово α по одному символу. Есть более простой способ, который мы правда рассмотрим после этого. Итак, для начала, аксиома и добавление единиц простые:

$$\begin{aligned} \pi_1 &= \text{Del}(_, _) \\ \pi_2 &= \frac{\text{Del}(x, y)}{\text{Del}(x1, y1)} \end{aligned}$$

А как с нулями? Снова будем рассматривать α как $x1v$, где v — цепочка нулей. Мы хотим добавить к этому 0, т.е. получить $\alpha = x1v0$. Теперь там будет цепочка нулей $v0$ в конце. У нас 3 варианта, или она короче максимальной цепочки в $x1v$, или такая же, или длиннее (так как мы добавляем символы по одному, то только на 1 символ, т.е. v было максимальной цепочкой).

Давайте по порядку, сначала случай, когда $v0$ — все еще короче максимальной цепочки нулей. Тогда можно спокойно добавлять 0 и к y , мы

удаляем только самые длинные цепочки, а эта точно не самая длинная. И да, назовем самую длинную цепочку нулей z . Тогда продукция:

$$\pi_3 = \frac{\text{Del}(x1v, y) \quad Ov \quad \text{Max}(x1v, z) \quad v < z}{\text{Del}(x1v0, y0)}$$

Второй случай. Наша $v0$ вот прямо сейчас достигает длины максимальной цепочки нулей. Тогда это означает, что эту цепочку не надо записывать в β . Но на всех предыдущих шагах мы же ее записывали, потому что она была меньше... Не очень хорошо вышло, надо удалить это оттуда. Сколько нулей мы уже успели записать в β ? Как раз v . Значит $\beta = yv$ и мы должны убрать оттуда эти нули. И все это происходит при условии, что $v0$ равна максимальной цепочке нулей. Итак, продукция:

$$\pi_4 = \frac{\text{Del}(x1v, yv) \quad \text{Max}(x1v, v0)}{\text{Del}(x1v0, y)}$$

Опять же, можно не проверять, что это реально цепочка нулей, потому что в правой части Max встречаются исключительно цепочки нулей.

Третий случай. Наша v уже была максимальной цепочкой нулей, и теперь $v0$ устанавливает «новый рекорд». Значит все, что мы удаляли раньше — было зря?... Видимо да, необходимо вернуть все на место и удалить единственную цепочку этой длины — $v0$. Таким образом, если $\alpha = x1v0$, то $\beta = x1$.

$$\pi_5 = \frac{\text{Del}(x1v, y) \quad \text{Max}(x1v, v)}{\text{Del}(x1v0, x1)}$$

И снова, мы ожидаем, что в словах есть хотя бы одна единица, так что цепочки нулей надо обработать отдельно:

$$\pi_6 = \frac{Ox}{\text{Del}(x, _)}$$

Решение. А теперь к более приятному способу решения того же самого при помощи лишь 4 простых продукций. Мы добьемся этого, используя вспомогательные слова. Заметьте, что заключения всех наших продукций до этого были словами той формы, которую нам надо было выводить. Теперь давайте же введем вспомогательные слова, которые будут важны только в процессе «работы программы» и сделаем таким образом цикл. Какой самый логичный способ решать данную задачу? Найти максимальную цепочку нулей ($\text{Max}(\alpha, v)$) и удалять постепенно все места, где она встречается в α . Для этого будем использовать слова вида (α, γ) , где γ — вспомогательная «переменная» цикла, в ней будет храниться текущее частично измененное слово α , откуда удаляются цепочки нулей, но возможно удалились еще не все. Как мы ее инициализируем? Так как α может быть любым словом, то вот так:

$$\pi_1 = \frac{Wx}{(x, x)}$$

Далее само тело цикла. Если v — самая длинная цепочка нулей в α , то если слово γ можно записать как uvz (т.е. внутри него есть эта цепочка), то ее надо удалить. Более того, раз это самая длинная цепочка, то ее не будет внутри других цепочек и мы удалим ее целиком. Итак:

$$\pi_2 = \frac{(x, uvz) \quad \text{Max}(x, v)}{(x, uz)}$$

Прекрасно. Много раз применяя эту продукцию мы удалим все нужные цепочки нулей. Но как теперь остановиться и выйти из цикла? Теперь самая длинная цепочка нулей в γ будет меньше, чем была изначально в α , по этому условию и будем выходить из цикла и выдавать результат работы:

$$\pi_3 = \frac{(x, z) \quad \text{Max}(x, v) \quad \text{Max}(z, w) \quad w < v}{\text{Del}(x, z)}$$

Проблема правда в цепочках из единиц. Если там нет нулей в принципе, то удалять нечего, и $w = v = _$ и никогда не станет меньше. Проблема? Просто введем еще одну продукцию специально для цепочек из единиц:

$$\pi_4 = \frac{Ix}{\text{Del}(x, x)}$$

Задача. $\text{del}(\alpha, \beta)$ — где $\alpha \in \{0, 1\}^*$ — любое слово, а β — это то, что остается от α после удаления всех цепочек нулей *не* максимальной длины.

Решение. Даже не будем пытаться решать как раньше, давайте сразу цикл. Инициализация такая же:

$$\pi_1 = \frac{Wx}{(x, x)}$$

А вот удаление чуть сложнее. Проблема в том, что мы не можем просто искать цепочки как uvz , потому что тогда, например, мы можем найти кусочки этой самой цепочки нулей максимальной длины и удалять ее по частям, что не очень хорошо. Поэтому нам надо искать цепочки, ограниченные единицами:

$$\pi_2 = \frac{(x, u1v1z) \quad Ov \quad \text{Max}(x, p) \quad v < p \quad _ < v}{(x, u11z)}$$

Последнее условие на самом деле не обязательно, система Поста будет работать и без него, оно просто для того, чтобы она не пыталась удалять пустые цепочки «из нулей». Но как быть, если эта цепочка нулей стоит в начале слова или в конце, и перед ней/после нее нет единиц? Отдельные продукции!

$$\begin{aligned} \pi_3 &= \frac{(x, v1z) \quad Ov \quad \text{Max}(x, p) \quad v < p \quad _ < v}{(x, 1z)} \\ \pi_4 &= \frac{(x, u1v) \quad Ov \quad \text{Max}(x, p) \quad v < p \quad _ < v}{(x, u1)} \end{aligned}$$

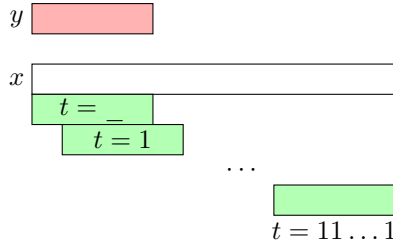
Очевидно, что других вариантов нет, цепочка v не может занимать все слово, потому что она не максимальная. А теперь вывод ответа. Как узнать, что мы удалили все цепочки нулей не максимальной длины? К счастью, мы уже написали функцию $\text{Del}(\alpha, \beta)$, которая удаляет все цепочки максимальной длины. Если мы удалили все цепочки *не* максимальной длины, а потом удалим и максимальной, то у нас не останется цепочек нулей, так? Останутся только единицы. Это и запишем:

$$\pi_5 = \frac{(x, y) \quad \text{Del}(y, z) \quad Iz}{\text{del}(x, y)}$$

На этом и закончим эту задачу.

Задача. $\text{sol}(x, y) = z^6$, где $x, y \in \{0, 1\}^*$, а z — количество вхождений y в x (пересечения разрешены) в унарной системе, т.е. просто столько единиц.

Решение. Для проверки нам потребуется просто перебирать различные варианты расположения y в x :



Будем перебирать значение t (сдвига) в унарной системе счисления. Нам тут потребуются целых 2 дополнительные переменные, t для сдвига и z — счетчик, для хранения результата. Изначально они оба 0.

$$\pi_1 = \frac{Wx \quad Wy}{(x, y, _, _)}$$

Далее цикл. Предположим, что мы нашли там y . То есть $x = uyw$, причем длина u совпадает с длиной нашего сдвига (он у нас здесь чтобы все варианты перебрать). Тогда мы увеличиваем сдвиг и увеличиваем счетчик:

$$\pi_2 = \frac{(uyw, y, t, z) \quad u = t}{(uyw, y, t1, z1)}$$

А если там не y ? Тогда не увеличиваем счетчик. Как проверить, что это не y ? Пусть $x = uvw$, и мы хотим проверить, что v не y . Для начала, $u = t$, у нас для этого сдвиг. Еще v и y должны быть одинаковой длины: $v = y$, но при этом не равны: $NE(v, y)$. Вот и все:

$$\pi_3 = \frac{(uvw, y, t, z) \quad u = t \quad v = y \quad NE(v, y)}{(uvw, y, t1, z)}$$

⁶Это не мое название, претензии не ко мне

И нам еще нужно нечто для извлечения ответа. Как мы узнаем, что t дошел до конца? Когда t вместе с y вылезет за границы x : $x < ty$:

$$\pi_4 = \frac{(x, y, t, z) \quad x < ty}{\text{col}(x, y) = z}$$

На этом все.

11.10 Задачи 9-10

Необходимые темы: системы Поста (глава 9).

Объединяю 2 задачи на системы Поста, потому что они крайне похожи. Для решения этих задач будут использоваться вспомогательные системы и методы решения из предыдущего бонусного раздела, так что прочитать его перед этим было бы хорошей идеей. Эти задачи — просто многоуровневое повторение того, что уже было до этого.

Задача. Определить систему Поста, выводящую (α, β) , где α — любое слово из $\{0, 1\}^*$, а β получается удалением из α первых вхождений, которые не являются самыми длинными, слов вида $(010)^*$, выбираемых целиком.

Решение. Сначала разберемся с условием. α — любое, о нем не беспокоимся, а вот β , 1) получается из α удалением 2) первых вхождений 3) которые не максимальной длины 4) слов вида $(010)^*$. Давайте пример

$$\alpha = 101001011110101111010110100010010$$

В этом слове среди подслов вида $(010)^*$ присутствуют только 2: 010 и 010010. Из них 010010 — самое длинное, его не трогаем. И мы удаляем первое вхождение всех остальных (010) , то есть удаляем первое 010:

$$\beta = 1010010111111010110100010010$$

Решать мы это будем как всегда аналогом цикла. Так как мы будем отыскивать все цепочки, перебирая и все возможные длины, и различные сдвиги, то нам потребуется 2 переменные цикла, для длины (в унарной системе!) и для сдвига (все еще в унарной системе!). Давайте пока напишем общий алгоритм, как мы это собираемся делать.

1. Старт. Тут мы каким-то образом получим α и инициализируем переменные.
2. Найти максимальную $(010)^n$ в α . Раз мы удаляем все, кроме нее, надо бы сначала знать, какая она.
3. Удаляем все первые вхождения $(010)^n$ меньшей длины, чем максимальная.

4. Получим ответ. Слова в цикле не совсем той структуры, так что это будет просто одна продукция, которая форматирует ответ в вид (α, β) .

Начнем? Так как слово α произвольное, мы можем выбрать его просто как Wx . Мы сначала хотим найти максимальную длину последовательности $(010)^n$, и этот поиск можно вести по убыванию — искать сначала самые длинные последовательности, а потом идти к коротким. Как только найдем — переходим на следующий этап. Сначала мы просто возьмем длину цепочки такую же, как x (назовем это y). Еще нам потребуется сдвиг, который пока 0, т.е. пустое слово. Итого первая продукция:

$$\pi_1 = \frac{Wx \quad Iy \quad y = x}{(x, y, _)}$$

Первый параметр — изначальное слово, α , второе — цепочка из 1 такой длины, какие подслова проверяем, а третье — сдвиг от начала. Правда нам нет смысла проверять те длины, которые не кратны 3, и мы могли бы усложнить это, добавив вместо нее 3 продукции, которые бы рассматривали x различной кратности. Мы правда таким сложным путем идти не будем, потому что можем написать и менее эффективную «программу», но рабочую.

Итак, переходим к этапу поиска. Мы будем идти по слову x слева направо, проверяя все цепочки длины y . Если цепочка является повторением 010, то мы нашли, что искали⁷, если же нет, то мы идем дальше. А теперь более формально, сначала, рисунок:

x	u	v	p
	z	y	

z здесь — третья «переменная», сдвиг. Мы разделяем x на 3 части: u, v и p . u соответствует сдвигу z , v — это та цепочка, которую мы проверяем, длины y , а p — просто то, что осталось. Если мы *не* нашли слово, т.е. $v \neq (010)^n$, что мы проверяем при помощи $NY(v, 010)$, то мы идем дальше и увеличиваем z на 1:

$$\pi_2 = \frac{(uvp, y, z) \quad u = z \quad v = y \quad NY(v, 010)}{(uvp, y, z1)}$$

Если же мы нашли наше слово, т.е. $Y(v, 010)$, то это значит, что y — наибольшая длина цепочки, мы ее нашли и должны перейти к этапу удаления. Для этого нам нужно подготовить переменные для него. Мы должны удалять цепочки длинами начиная с $y-3$. Как получить это в унарной системе? А вот так, если $r111 \equiv y$, то r на 3 короче y . Равенство мы записываем как $E(r111, y)$. И тогда мы переходим к новому этапу. Какие там будут переменные? Изначальный x , еще переменная, из которой мы будем удалять

⁷Это цепочка максимальной длины, потому что мы идем от длинных цепочек к коротким, так что самая длинная, потому что мы встретили ее первой

цепочки, а еще снова длина цепочки и сдвиг. 4 переменные. $x = uvr$, то, откуда удаляем, изначально тоже x , длина цепочки изначально r , а сдвиг изначально $_$. Прекрасно, вот и наша продукция:

$$\pi_3 = \frac{(uvr, y, z) \quad u = z \quad v = y \quad Y(v, 010) \quad E(r111, y)}{(uvr, uvr, r, _)}$$

Мы правда кое про что забыли. А что будет, когда сдвиг будет настолько большим, что мы вылезем за пределы слова? Т.е. если x будет короче, чем $z + y$? Тогда нам стоит сбросить z в начало ($_$), а y уменьшить на 1. Почему 1, а не 3? А мы же не добавили ту проверку на кратность, поэтому теперь нам придется перебирать все длины. Но вот так. Как мы будем уменьшать? А снова, $E(r1, y)$:

$$\pi_4 = \frac{(x, y, z) \quad x < zy \quad E(r1, y)}{(x, r, _)}$$

Но мы все еще про кое что забыли. Что будет, если мы такими темпами дойдем до $y = _$? То есть если цепочек вида $(010)^*$ там нет в принципе? Это особый случай. Но если цепочек таких нет, то нам и удалять ничего не надо, верно? Тогда мы сможем сразу выдать ответ как надо:

$$\pi_5 = \frac{(x, _, _)}{(x, x)}$$

На этом поиск все. Теперь переходим к удалению. На входе у нас есть четверки (x, y, r, z) , x — изначальное слово, y — то, откуда удаляем, r — длина, которую удаляем, z — сдвиг. Эта задача — еще далеко не самая сложная, было бы намного сложнее, если бы мы удаляли, например, внутренние вхождения этих цепочек (не первые и не последние). В любом случае, давайте разбивать на случаи. Как всегда, разобьем y на uvr :

y	u	v	p
	z	r	

Сначала рассмотрим разные ситуации, когда v — не та цепочка, которую мы ищем, и ее удалять не нужно. Какой самый простой вариант? Она банально не подходит под $(010)^*$. Тогда мы просто идем дальше и увеличиваем z на 1:

$$\pi_6 = \frac{(x, uvr, r, z) \quad u = z \quad v = r \quad NY(v, 010)}{(x, uvr, r, z1)}$$

Почему еще может оказаться? Дело в том, что мы должны удалять наши цепочки 010 целиком. То есть если мы наткнулись на некоторое $v = (010)^n$, может оказаться, что его брать нельзя, потому что перед ним или после него стоит 010 . Тогда это на самом деле часть более длинной цепочки длины $r + 3$ или даже больше, а мы сейчас проверяем цепочки длины r , так что такое не трогаем.

Сначала рассмотрим, когда слева от v есть 010:

$$\pi_7 = \frac{(x, u010vp, r, z) \quad u010 = z \quad v = r \quad Y(v, 010)}{(x, u010vp, r, z1)}$$

Заметьте, раз у нас v — тоже цепочка из 010, то по идее мы могли бы сдвинуться не на 1, а на длину всей этой цепочки, r , и все было бы тоже нормально. И теперь справа:

$$\pi_8 = \frac{(x, uv010p, r, z) \quad u = z \quad v = r \quad Y(v, 010)}{(x, uv010p, r, z1)}$$

Снова можем сдвинуться намного дальше, аж на $r + 3$, но не будем. Это все ситуации, когда нам ничего не надо удалять.

А когда надо? Когда ни одна из предыдущих не выполняется, т.е. v — и правда $(010)^n$, но при этом слева и справа стоят не 010. А вот сейчас начинаются проблемы. Мы не можем просто сравнить то, что слева, с 010, потому что там, например, может ничего и не быть. Поэтому придется разбивать на случаи. Константин Иванович разбивает «слева не 010» на 6 случаев, а именно:

- 1) $_ = u$
- 2) $_0 = u$
- 3) $q1 = u$
- 4) $_10 = u$
- 5) $q00 = u$
- 6) $q110 = u$

А именно

1. Слева от v ничего нет
2. Слева от v только 0
3. Слева от v стоит нечто, что заканчивается на 1 (а не на 0, как 010)
4. Слева стоит только 10
5. Слева стоит нечто, что заканчивается на 00 (а не на 10, как 010)
6. Слева стоит нечто, что заканчивается на 110

Аналогично и 6 случаев справа:

- 1) $p = _$
- 2) $p = 0_$
- 3) $p = 1s$
- 4) $p = 01_$
- 5) $p = 00s$
- 6) $p = 011s$

Итого вместе нам потребуется 36 продукций с этим. Общий вид такой:

$$\pi_{9-44} = \frac{(x, uvp, r, z) \quad u = z \quad v = r \quad Y(v, 010) \quad E(u, ?) \quad E(p, ?) \quad E(r, t111)}{(x, up, t, _)}$$

Да, 36. Писать мы их конечно не будем. Есть правда одно упущение (не говорите об этом Костенко), число продукций можно сократить до 4. Как? 6 случаев слева можно объединить в 2 и 6 случаев справа тоже в 2. Какие эти 2 случая? Когда справа есть достаточно символов, но они не те (3, 5, 6), и когда слева недостаточно символов (1, 2, 4). Давайте рассмотрим, когда и слева, и справа, достаточно символов. Тогда пусть $y = uu_3vp_3p$, где u_3 и p_3 длины 3, но не 010. Это можно записать продукцией

$$\pi'_9 = \frac{(x, uu_3vp_3p, r, z) \quad u = z \quad v = r \quad Y(v, 010) \quad u_3 = 010 \quad p_3 = 010 \quad NE(u_3, 010) \quad NE(p_3, 010) \quad E(r, t111)}{(x, uu_3p_3p, t, _)}$$

Большая продукция с 9 посылками, но зато она объединяет 9 из 36 продукций в одну. Далее, пусть когда слева недостаточно символов, тогда $y = uvp_3p$, причем длины u меньше длины 010.

$$\pi'_{10} = \frac{(x, uvp_3p, r, z) \quad u = z \quad v = r \quad Y(v, 010) \quad u < 010 \quad p_3 = 010 \quad NE(p_3, 010) \quad E(r, t111)}{(x, up_3p, t, _)}$$

Аналогично если не хватает символов справа:

$$\pi'_{11} = \frac{(x, uu_3vp, r, z) \quad u = z \quad v = r \quad Y(v, 010) \quad p < 010 \quad u_3 = 010 \quad NE(u_3, 010) \quad E(r, t111)}{(x, uu_3p, t, _)}$$

И если не хватает с обеих сторон:

$$\pi'_{12} = \frac{(x, uvp, r, z) \quad u = z \quad v = r \quad Y(v, 010) \quad p < 010 \quad u < 010 \quad E(r, t111)}{(x, up, t, _)}$$

Вот и все, 4 продукции вместо 36. Но мы так делать не будем.

Теперь приступим к случаю, когда мы дошли до конца строки, т.е. когда $y < zr$. В такой ситуации мы уменьшаем r на 3, как делали раньше, и начинаем заново:

$$\pi_{45} = \frac{(x, y, r, z) \quad y < zr \quad E(r, t111)}{(x, y, t, _)}$$

А что делать, если мы дошли до слов длины 0? Это будет конец, мы удалили все, что хотели, можно и вернуть ответ:

$$\pi_{46} = \frac{(x, y, _, _)}{(x, y)}$$

Задача. (α, β) , где α — любое слово из $\{0, 1\}^*$, а β — *всякое* симметричное подслово α не максимальной длины.

Решение. Эта задача слегка отличается от предыдущей, но она и намного проще. Для начала, общий алгоритм. Мы сначала найдем самое длинное симметричное подслово, а потом будем искать и выводить все симметричные подслова, которые короче этого. Для начала, нам нужен механизм для перебора слов, он тот же самый — длина+сдвиг. Перебирать будем начиная с самых длинных, тогда первое симметричное, которое встретим, и будет самым длинным. Первой продукцией тогда будет инициализация:

$$\pi_1 = \frac{Wx \quad Iy \quad y = x}{(x, y, _)}$$

Первая компонента — слово α , вторая — столько единиц, какая длина подпоследовательности, третья — столько единиц, какой сдвиг от начала. Общий механизм перебора опять такой же, перебираем слова и проверяем, как только доходим до конца строки уменьшаем длину и начинаем заново. В этой задаче мы гарантированно это слово найдем, потому что пустое слово — тоже симметричное.

Теперь поиск. Снова будем искать нашу подпоследовательность при помощи uvr . Снова несколько случаев, сначала если наше слово не является симметричным. Тогда мы просто идем дальше:

$$\pi_2 = \frac{(uvr, y, z) \quad u = z \quad v = y \quad NSv}{(uvr, y, z1)}$$

А если симметричное? Тогда это самое длинное, мы его нашли, надо переходить к следующему этапу. Мы хотим запомнить эту самую максимальную длину, так что нам потребуется запомнить (uvr, y) . Правда это — как раз формат ответа, но не ответ, так что чтобы не было путаницы, добавим символ в начале, пусть будет $A(uvr, y)$.

$$\pi_3 = \frac{(uvr, y, z) \quad u = z \quad v = y \quad Sv}{A(uvr, y)}$$

А если вышли за границу слова? То же самое, что раньше:

$$\pi_4 = \frac{(x, y, z) \quad x < zy \quad E(r1, y)}{(x, r, _)}$$

Кстати мы можем записать это немного по другому, если мы сразу будем использовать не y , а $r1$:

$$\pi_4 = \frac{(x, r1, z) \quad x < zr1}{(x, r, _)}$$

А теперь следующий этап, нам нужно найти все симметричные подслова, которые короче y . Вам не кажется, что это идеально подходит под формат посылки продукции? А мне кажется, смотрите, если у нас есть $A(uvp, y)$, то нам необходимо вывести слово v , если оно симметричное и короче чем y . Вот так:

$$\pi_5 = \frac{A(uvp, y) \quad Sv \quad v < y}{(uvp, v)}$$

И все, задача решена. Что делать правда, если вы не заметите такое решение? Или даже хуже, задача не допускает такого? Тогда можно по старинке, цикл. Во-первых, надо поменять нашу продукцию для перехода на 2 этап, нам будет удобнее, если она будет сразу выдавать переменные для последнего этапа: слово $\alpha = uvp$, длину, откуда начинать отсчет и сдвиг:

$$\pi'_3 = \frac{(uvp, y, z) \quad u = z \quad v = y \quad Sv \quad E(t1, y)}{B(uvp, t, _)}$$

Снова нужен символ (B) , чтобы оно не пересекалось с первым этапом. И снова тот же самый обход, если слово несимметрично:

$$\pi'_5 = \frac{B(uvp, y, z) \quad u = z \quad v = y \quad NSv}{B(uvp, y, z1)}$$

И то же самое, если вышли за границу слова:

$$\pi'_6 = \frac{B(x, r1, z) \quad x < zr1}{B(x, r, _)}$$

А если нашли? Во-первых, это будет ответ, а во-вторых, идем дальше. Это будут две продукции:

$$\begin{aligned} \pi'_7 &= \frac{B(uvp, y, z) \quad u = z \quad v = y \quad Sv}{(uvp, v)} \\ \pi'_8 &= \frac{B(uvp, y, z) \quad u = z \quad v = y \quad Sv}{B(uvp, y, z1)} \end{aligned}$$

Вот и все, второй вариант решения. Надо было бы еще кстати рассмотреть, когда эти продукции могут не сработать, например на пустом слове x продукция π'_6 не сработает, но здесь так и надо. Но в другой задаче это стоило бы проверить отдельно.

Задача. (α, β) , где α — любое слово из $\{0, 1\}^*$, а β — *всякая* не самая длинная цепочка нулей, входящая в α симметрично хотя бы один раз

Решение. Какой общий алгоритм? Снова сначала ищем самую длинную цепочку нулей, а потом все остальное, так что начало очень похоже:

$$\begin{aligned}\pi_1 &= \frac{Wx \quad Iy \quad y = x}{(x, y, _)} \\ \pi_2 &= \frac{(uvp, y, z) \quad u = z \quad v = y \quad NOv}{(uvp, y, z1)} \\ \pi_3 &= \frac{(uvp, y, z) \quad u = z \quad v = y \quad Ov}{B(uvp, y)} \\ \pi_4 &= \frac{(x, r1, z) \quad x < zr1}{(x, r, _)}\end{aligned}$$

Сейчас вопрос, как проверить, что наша цепочка нулей входит в α симметрично (хотя бы один раз)? Ну вот пример слова, которое подойдет:

01011100001100000010000100000
s 1 v 1 r 1 v 1 t

Вот что мы имеем в виду под цепочками нулей, расположенными симметрично. Эти цепочки слева и справа ограничены единицами, поэтому они полноценные цепочки, а не просто части цепочек. Итого, если наше слово можно разбить на вид $\alpha = slv1r1v1t$, где s и t одинаковой длины и v — цепочка нулей (причем не самая длинная), то это слово нам подходит. Продукция для этого? Вот:

$$\pi_5 = \frac{B(slvr1v1t, y) \quad s = t \quad Ov \quad v < y}{(slvr1v1t, v)}$$

Но что будет, если v стоят с краю и там нет 1? Если слово вот такое?

00001100000010000
v 1 r 1 v

Тоже должно подойти, и это отдельная продукция:

$$\pi_6 = \frac{B(v1r1v, y) \quad Ov \quad v < y}{(v1r1v, v)}$$

Вам казалось что это все, да? Но нет, мы упустили еще один очень редкий случай, который очень сложно увидеть. Для начала, смотрите, что будет, если мы поставим $r = _$ в первом слове:

0101110000110000100000
s 1 v 1 1 v 1 t

Этот случай вполне покрывается продукцией π_5 . Но обязательно ли между v две единицы? Там может быть одна:

010111000010000100000
s 1 v 1 v 1 t

Для этого тоже нужна продукция:

$$\pi_7 = \frac{B(s1v1v1t, y) \quad s = t \quad Ov \quad v < y}{(s1v1v1t, v)}$$

А что насчет второго случая? Вот такая продукция

$$\pi_8 = \frac{B(v1v, y) \quad Ov \quad v < y}{(v1v, v)}$$

Оказывается, она не нужна. Смотрите, как будет выглядеть такое слово:

000010000
v 1 v

v здесь — гарантированно самая длинная цепочка нулей в α , а их мы не учитываем. Значит такого случая банально не бывает. Вот теперь мы закончили.

Задача. (α, β) , где α — любое слово из $\{0, 1\}^*$, а β — *всякое* не самое длинное слово вида $0^n 1^n$, входящее в α несимметрично четное число раз (≥ 2).

Решение. Общий алгоритм примерно все тот же. Для начала, нам нужно найти самое длинное слово такого вида. Потом будем выводить все подходящие слова с дополнительными проверками. Для начала, что значит «слово вида $0^n 1^n$ »? Возможно несколько интерпретаций, но мне кажется, что самая логичная это такие слова как 000111 или 0000011111, т.е. слова с одинаковым числом 0 и 1. Такие слова можно построить при помощи посылок Ou , Iv , $u = v$, и это будет слово uv . Окей. Наша основная часть:

$$\begin{aligned} \pi_1 &= \frac{Wx \quad Iy \quad y = x}{(x, y, _)} \\ \pi_2 &= \frac{(suvt, y, z) \quad s = z \quad uv = y \quad Ou \quad Iv \quad u = v}{B(suvt, y)} \\ \pi_3 &= \frac{(x, r1, z) \quad x < zr1}{(x, r, _)} \end{aligned}$$

Тут правда нет случая, если это не подходящее слово. Дело в том, что может нарушиться любая из посылок Ou , Iv , $u = v$. Поэтому нам нужны 3 отдельные продукции на их нарушение, вот первые 2:

$$\begin{aligned} \pi_4 &= \frac{(suvt, y, z) \quad s = z \quad uv = r1 \quad NOu \quad u = v}{(suvt, y, z1)} \\ \pi_5 &= \frac{(suvt, y, z) \quad s = z \quad uv = r1 \quad Ou \quad NIv \quad u = v}{(suvt, y, z1)} \end{aligned}$$

А когда нарушается $u = v$? Когда напе y — нечетное, тогда это в принципе невозможно, можно сразу идти к следующей длине. У нас есть система

