DM Eχ 02

isagila @pochtineploho @DUBSTEPHAVEGUN

Собрано 28.01.2024 в 07:24

Содержание

1. Теория графов

1.1. Основные определения.

Def 1.1.1. Граф это упорядоченная пара вида $G = \langle V, E \rangle$, где $V = \{v_1, \ldots, v_n\}$ это множество вершин, а $E = \{e_1, \ldots, e_n\}$ это множество ребер.

Def 1.1.2. Порядком графа называется число его вершин $|V|$.

Def 1.1.3. Размером графа называется количество его ребер $|E|$.

У неориентированного графа $E \subseteq V^{(2)}$, где $V^{(2)}$ это множество всех непустых подмножеств размера не более двух. Таким образом каждое ребро обозначается как $\{u, v\} \in V^{(2)}$ (если $u \neq v$) или $\{v\} \in V^{(2)}$ (если данное ребро это петля). У ориентированного графа $E \subseteq V^2$, т.е. каждое ребро это упорядоченная пара вида $\langle u, v \rangle$. Ребра ориентированного графа также называют дугами.

Def 1.1.4. Петля это ребро, которое соединяет вершину саму с собой.

Def 1.1.5. Мультиребра это ребра, у которых общее начало и общий конец.

Def 1.1.6. Граф называется простым, если в нем нет мультиребер и петель.

Def 1.1.7. Мультиграф это граф с мультиребрами.

Def 1.1.8. Псевдограф это мультиграф с петлями.

Def 1.1.9. Гиперграф это граф, в котором ребро может соединять несколько вершин одновременно.

Def 1.1.10. Нулевой граф это граф, который не содержит вершин.

Def 1.1.11. Пустой граф это граф, которые не содержит ребер.

Def 1.1.12. Граф-синглтон (тривиальный граф) это граф состоящий только из одной вершины.

Def 1.1.13. Полный граф K_n это простой граф, в котором каждая пара различных вершин соединена ребром.

Def 1.1.14. Взвешенный граф $G = \langle V, E, w \rangle$ это граф, в котором каждому ребру сопоставляется некоторое числовое значение (вес), которое определяется весовой функцией $w \colon E \to \mathrm{Num}$.

Def 1.1.15. Подграфом графа $G = \langle V, E \rangle$ называется граф $G' = \langle V', E' \rangle$ такой, что $V' \subseteq V$ и $E' \subseteq E$.

Def 1.1.16. Подграф называется остовным, если он содержит все вершины исходного графа.

Def 1.1.17. Индуцированный подграф это подграф, который получается из некоторого подмножества вершин исходного графа $V' \subseteq V$ и **всех** ребер исходного графа, соединяющих эти вершины.

Def 1.1.18. Две вершины называются смежными (соседними), если между ними есть ребро.

Для иллюстрации отношения смежности обычно используют матрицу смежности. Это квадратная матрица размера $V \times V$, ячейки которой содержат:

- 0 или 1 — для простых графов
- $-1, 0, 1$ — для ориентированных графов
- $\mathbb{N}$ — для взвешенных графов

Однако описанные выше правила не строгие: в разных задачах числа в матрице смежности могу обозначать разные вещи.

Def 1.1.19. Вершина и ребро называются инцидентными, если вершина является одним из концов ребра.

Для иллюстрации отношения инцидентности обычно используют матрицу инцидентности. Это прямоугольная матрица размера $V \times E$, строки которой соответствуют вершинам, а столбцы — ребра. Ячейки этой матрицы могут содержать:

- 1 если вершина и ребро инциденты и 0 в противном случае — для неориентированных графов.
- -1, если ребро выходит из данной вершин, и 1 если входит, 0 если ребро и вершина неинцидентны — для ориентированных графов.

Петля в матрице инцидентности обычно обозначается двойкой. Но, как и в случае с матрицей смежности, описанные правила не являются строгими.

Def 1.1.20. Степенью вершины $\deg u$ называется количество инцидентных ей ребер (петли учитываются дважды).

Со степенью вершины также связны такие понятия как:

- Минимальная степень вершины в графе $\delta(G) = \min\limits_{v \in V} \deg v$.
- Максимальная степень вершины в графе $\Delta(G) = \max\limits_{v \in V} \deg v$.

Lm 1.1.21. Лемма о рукопожатиях.

$$\sum_{v \in V} \deg v = 2\,|E|$$

□ Т.к. каждое ребро инцидентно ровно двум ребрам, то сложив все степени вершин мы учтем каждое ребро дважды (по одному разу для каждой из его концевых вершин). ■

Def 1.1.22. Граф называется r-регулярным, если степень каждой из его вершин равна r.

Def 1.1.23. Граф называется планарным, если его можно изобразить на плоскости без пересечения ребер.

Def 1.1.24. Граф называется плоским, если он *уже* изображен на плоскости без пересечения ребер.

Теорема 1.1.25. Теорема Эйлера для графов
Для планарных графов выполняется соотношение

$$|V| - |E| + |F| = 2$$

где $|V|$ — количество вершин, $|E|$ — количество ребер, а $|F|$ — количество граней.

□ Индукция по количеству граней.
База: $F = 2$. Тогда граф представляет собой многоугольник с $|V|$ вершинами и при этом $|V| = |E| \Longrightarrow$ равенство выполняется.
Переход: пусть теорема верна для графа в котором F граней. Добавим новую грань, для этого проведем по внешней границе некоторую простую цепь, которая будет содержать $r-1$ вершин и r ребер. Подставим эти изменения в формулу и упростим:

$$(|V| + r - 1) - (|E| + r) + (|F| + 1) \stackrel{?}{=} 2$$
$$|V| - |E| + |F| \stackrel{?}{=} 2$$

Это верно по предположению индукции. ■

Def 1.1.26. Клика это множество вершин в графе, которые индуцируют полный подграф.

Def 1.1.27. Независимое (стабильное) множество это множество вершин графа такое, что любые две вершины в нем не смежны.

Def 1.1.28. Паросочетание это множество ребер графа такое, что любые два ребра в нем не смежны.

Def 1.1.29. Совершенное (идеальное) паросочетание это такое паросочетание, что любая вершина в графе инцидента некоторому ребру из него.

Def 1.1.30. Вершинное покрытие это множество вершин таких, что любое ребро в графе инцидентно хотя бы одной вершине из этого множества.

Def 1.1.31. Реберное покрытие это множество ребер таких, что любая вершина инцидентна хотя бы одному ребру их этого множества.

1.2. Морфизмы графов.

Def 1.2.1. Два графа называются изоморфными, если между из вершинами существует биекция, причем две вершины в первом графе смежны тогда и только тогда, когда смежны соответствующие им (по биекции) вершины второго графа.

Def 1.2.2. Автоморфизм графа — это отображение множества вершин графа в себя, сохраняющее смежность.

Замечание 1.2.3. Граф, для которого единственный возможный автоморфизм это тождественное отображение, называется асимметрическим.

Def 1.2.4. Гомоморфизм графа — это отображение одного графа в другой, которое сохраняет смежность вершин.

Def 1.2.5. Подразделением графа G называется граф G', полученный делением ребер графа G новыми вершинами.

Def 1.2.6. Два графа G_1 и G_2 называются гомеоморфными, если существует изоморфизм некоторого подразделения графа G_1 и некоторого подразделения графа G_2

1.3. Пути и расстояния.

Def 1.3.1. Маршрут (*walk*) это чередующаяся последовательность из вершин и ребер.

Def 1.3.2. Путь (*trail*) это маршрут без повторяющихся ребер.

Def 1.3.3. Цепь (*path*) это маршрут без повторяющихся вершин.

Замечание 1.3.4. Для приведенных выше терминов существуют замкнутые версии (т.е. начальная и конечная вершины совпадают):

- Замкнутый маршрут (*closed walk*)
- Замкнутый путь (*closed trail*, *circuit*)
- Замкнутая цепь/цикл (*closed path*, *cycle*)

Def 1.3.5. Длина маршрута (пути, цепи) это количество ребер в нем.

Def 1.3.6. Обхват (*girth*) это длина кратчайшего цикла в графе.

Def 1.3.7. Расстояние $\text{dist}(u, v)$ между двумя вершинами u и v это длина кратчайшего пути $u \rightsquigarrow v$.

Def 1.3.8. Длина наибольшего из кратчайших расстояний от данной вершины v до остальных вершин называется эксцентриситетом $\varepsilon(v) = \max\limits_{u \in V} \text{dist}(u, v)$

Def 1.3.9. Радиусом графа называется минимальный из эксцентриситетов его вершин $\text{rad}(G) = \min\limits_{v \in V} \varepsilon(v)$.

Def 1.3.10. Диаметром графа называется максимальный из эксцентриситетов его вершин $\text{diam}(G) = \max\limits_{v \in V} \varepsilon(v)$.

Def 1.3.11. Множество вершин, эксцентриситет которых равен радиусу называются центром графа $\text{center}(G) = \{v \mid \varepsilon(v) = \text{rad}(G)\}$.

1.4. Эйлеровы графы.

Def 1.4.1. Эйлеров путь это путь, который проходит все ребра графа ровно по одному разу.

Def 1.4.2. Эйлеров цикл это эйлеров путь, у которого начальная вершина совпадает с конечной.

Def 1.4.3. Граф называется эйлеровым, если в нем есть эйлеров цикл, и полуэйлеровым, если в нем есть только эйлеров путь.

Теорема 1.4.4. Условие существования Эйлерова цикла
Эйлеров цикл существует тогда и только тогда, когда степени всех вершин четны.

□ Индукция по количеству ребер
База: $m = 0$, т.к. ребер нет, то считаем, что Эйлеров цикл есть
Переход: пусть теорема верна для графов с менее чем m ребрами. Т.к. все степени четны, то войдя в вершины мы всегда сможет выйти из неё (за исключением стартовой вершины) $\Longrightarrow$ будем идти по ребрам в случайном порядке пока не вернемся в стартовую вершину. Далее возможны две ситуации:

- Если мы обошли все ребра, то теорема верна
- Иначе удалим из графа все пройденные ребра. Он распадется на компоненты связности, в которых будет меньше ребер, чем в исходном графе $\Longrightarrow$ к ним применимо предположение индукции. Найдем в каждой компоненте Эйлеров цикл, после чего 'прикрепим' его к первоначально найденному циклу $\Longrightarrow$ получим Эйлеров цикл.

■

Замечание 1.4.5. Для существования Эйлерова пути необходимо и достаточно, чтобы количество вершин с нечетной степенью не превышало двух. В этом случае можно соединить эти вершины дополнительным ребром и построить Эйлеров цикл (т.к. в графе все степени вершин станут четными). После чего достаточно удалить добавленное ребро, чтобы получить Эйлеров путь.

1.5. Гамильтоновы графы.

Def 1.5.1. Гамильтонов путь это путь, который проходит все вершины в графе ровно по одному разу.

Def 1.5.2. Гамильтонов цикл это гамильтонов путь, у которого начальная вершина совпадает с конечной.
Несмотря на то, что каждая вершина должна быть пройдена ровно один раз, для стартовой вершины делается исключение.

Def 1.5.3. Граф называется гамильтоновым, если в нем есть гамильтонов цикл, и полугамильтоновым, если в нем есть только гамильтонов путь.

Теорема 1.5.4. Теорема Оре.
Если $n \geqslant 3$ и $\deg u + \deg v \geqslant n$ для любых двух **несмежных** вершин неориентированного графа G, то этот граф гамильтонов.

□ От противного: пусть дан граф, удовлетворяющий условиям теоремы, но не являющийся гамильтоновым. Будем добавлять к нем ребра до тех пор, пока он будет будет оставаться не гамильтоновым. Т.к. мы только добавляем ребра, то условия теоремы не нарушатся.
Пусть в полученном графе есть две несмежные вершины u и v такие, что добавление ребра (u, v) приводит к появлению гамильтонова цикла. Это значит, что путь $u \rightsquigarrow v$ гамильтонов. Обозначим вершины на этом пути t_j. Рассмотрим два множества:

- $S = \{i \mid \exists(u, t_{i+1} \in E)\}$
- $T = \{i \mid \exists(t_i, v) \in E)\}$

Тогда $|S| + |T| = \deg u + \deg v \geqslant n$, при этом $|S \cup T| < n$. Из этого следует, что $|S \cap T| = |S| + |T| - |S \cup T| > 0$, т.е. найдутся две смежные вершины t_1 и t_2 такие, что будут существовать ребра (u, t_2) и (t_1, v).
Построим гамильтонов цикл: $u \rightsquigarrow t_1 \to v \rightsquigarrow t_2 \to v$. Получаем противоречие: предполагалось, что граф не гамильтонов, однако мы смогли построить гамильтонов цикл. ■

Теорема 1.5.5. Теорема Дирака.
Если $|V| \geqslant 3$ и $\delta(G) \geqslant \frac{n}{2}$, то неориентированный граф G — гамильтонов.

□ Возьмем две произвольные несмежные вершины u и v, тогда $\deg u + \deg v \geqslant \frac{n}{2} + \frac{n}{2} = n$. Значит по т. Оре (1.5.4) граф G гамильтонов. ■

Замечание 1.5.6. Эти две теоремы являются достаточными, но не обходимыми условиями гамильтоновости графа, т.е. если они не выполняются, то это не значит, что граф не гамильтонов.

1.6. Двудольные графы.

Def 1.6.1. Двудольный граф это граф, вершины которого могут быть разбиты на два множества (две доли) так, что между вершинами каждого из множеств не будет ребер.

Замечание 1.6.2. Об обозначениях.
Двудольный граф обычно обозначается как $G = \langle X, Y, E\rangle$, где X и Y это его доли, а E — множество ребер между этими долями.
Пусть задано некоторое подмножество $S \subseteq X$, тогда $E(S)$ это множество ребер, инцидентных вершинам из множества S.
Обозначение $N(S)$ используется что показать 'соседей' для вершин из множества S, т.е. такие вершины из множества Y, которые смежны хотя бы с одной из вершин из множества S.

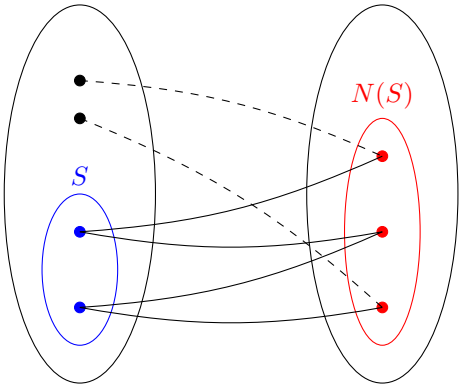
Def 1.6.3. Двудольный граф называется сбалансированным, если размеры его долей совпадают.

Теорема 1.6.4. r-регулярный двудольный граф является сбалансированным.

□ Пусть дан двудольный граф $G = \langle X, Y, E\rangle$. Т.к. он двудольный, то множество ребер может быть представлено в виде $|E| = |X| \cdot r$. С другой стороны оно может быть представлено в виде $|E| = |Y| \cdot r$. Приравниваем и получаем искомое равенство $|X| = |Y|$. ■

Теорема 1.6.5. В любом регулярном двудольном графе существует совершенное паросочетание.

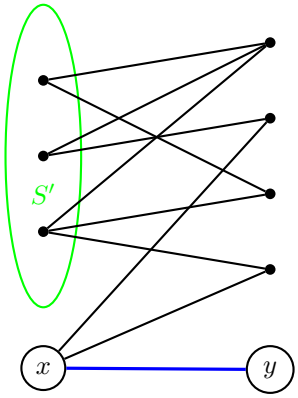
□ Рассмотрим двудольный граф $G = \langle X, Y, E\rangle$. Выберем произвольное $S \subseteq X$. Заметим, что $E(N(S)) \geqslant E(S)$, причем т.к. граф регулярный, то $E(N(S)) = |N(S)| \cdot r$, а $E(S) = |S| \cdot r$.
Подставляя это в полученное ранее неравенство, получаем, что $|N(S)| \geqslant |S|$ причем это выполняется для любого $S \subseteq X$. Значит по т. Холла (1.7.1) в графе существует X-совершенное паросочетание.
Т.к. G — регулярный двудольный граф, то по 1.6.4 он сбалансирован, значит $|X| = |Y|$. Таким образом X-совершенное паросочетание является совершенным паросочетанием. ■



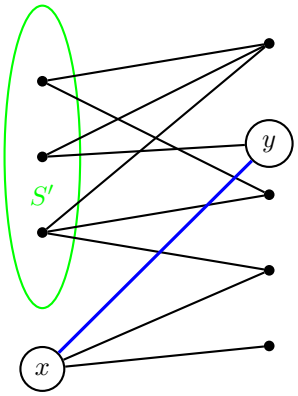
1.7. Теорема Холла.

Теорема 1.7.1. Теорема Холла.
Пусть дан двудольный граф $G = \langle X, Y, E \rangle$. В нем существует X-совершенное паросочетание тогда и только тогда, когда

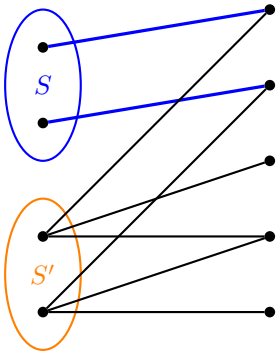
$$\forall S \subseteq X\colon\ |N(S)| \geqslant |S|$$



(a) 1ый случай (a)



(b) 1ый случай (b)



(c) 2ой случай

$\square$ $\Longrightarrow$ Если в графе существует X-совершенное паросочетание, то для любого $x \in X$ существует уникальный сосед в Y, значит количество соседей у любого $S \subseteq X$ будет как минимум $|x|$.
$\Longleftarrow$ Индукция по $|X|$.
База: $|X| = 1$. Очевидно, что X-совершенное паросочетание существует тогда и только тогда, когда у единственной вершины в X будет как минимум один сосед из Y.
Переход: пусть теорема верна при $|X| < n$. Рассмотрим граф, удовлетворяющий условиям теоремы, в котором $|X| = n$. Возможны два случая:

1. $\forall S \subseteq X\colon\ |N(S)| > |S|$

   Выберем произвольный $x \in S$ и поставим ему в пару любого из его соседей. Обозначим этого соседа y, а оставшиеся в S вершины S'.

   Т.к. до выбора соседа для x выполнялось неравенство $|N(S')| > |S'|$, то после выбора соседа выполняется неравенство $|N(S')| \geqslant |S'|$, т.к. количество свободных соседей S' могло уменьшиться не более чем на один (оно уменьшилось на один в случае, если $y \in N(S')$).

   Пары для оставшихся вершин мы можем найти по предположению индукции.

2. $\exists S \subseteq X\colon\ |N(S)| = |S|$

   Каждому $x \in S$ по предположению индукции поставим в пару одного из его соседей. Рассмотрим оставшиеся вершины $S' = X \setminus S$. Покажем, что для них тоже можно найти пару.

   Пусть найдется такое $P \subseteq S'$, что $|N(P)| < |P|$. Тогда рассмотрим множество $S \cup P$. Т.к. $|N(S)| = |S|$ и $|N(P)| < |P|$, то $|N(S \cup P)| < |S \cup P|$. Это противоречит условию теоремы, значит $\forall P \subseteq S'\colon\ |N(P)| \geqslant |P|$, поэтому по предположению индукции мы можем найти пары для всех $x \in S'$.

$\blacksquare$

1.8. Теорема Татта.

Def 1.8.1. 1-фактор графа G это 1-регулярный подграф графа G.

Теорема 1.8.2. Теорема Татта.
Граф G содержит 1-фактор тогда и только тогда, когда

$$\forall S \subseteq V\colon k_0(G - S) \leqslant |S|$$

где $k_0(G - S)$ это число нечетных (т.е. содержащих нечетное количество вершин) компонент графа $G - S$.

$\square$ $\Longrightarrow$ Пусть $S \subseteq V$. Если граф $G - S$ не содержит нечетных компонент, то тогда $k_0(G - S) = 0$ и очевидно, что $k_0(G - S) \leqslant |S|$.
Предположим, что $k_0(G - S) = k \geqslant 1$. Обозначим $G_1, \ldots, G_k$ нечетные компоненты графа $G - s$. Т.к. граф G содержит 1-фактор (обозначим его F), то некоторое ребро из F должно быть инцидентно вершине из G_i и вершине из S. Таким образом $k_0(G - S) \leqslant |S|$.

$\Longleftarrow$ Пусть $\forall S \subseteq V\colon k_0(G-S) \leqslant |S|$. Для $S = \varnothing$ получаем, что $k_0(G-S) = k_0(G) = 0$. Таким образом каждая компонента графа G — четная, а значит и сам граф G содержит четное число вершин. Далее по индукции по числу вершин покажем, что любой граф четного размера, обладающий этим свойством, содержит 1-фактор.
База: K_2 имеет 1-фактор.
Переход: пусть теорема выполнена для всех четных графов, удовлетворяющих условиям, и содержащих менее n вершин. Рассмотрим граф G, удовлетворяющий условиям, и имеющий четный размер n.
Заметим, что любая нетривиальная компонента графа G содержит вершину, которая не является точкой сочленения, поэтому существуют такие $R_j \subset V$ для которой $k_0(G - R_j) = |R_j|$. Пусть S это максимальное по мощности множество R_j. Обозначим $G_1, \ldots, G_k$ нечетные компоненты $G - S$, тогда $|S| = k \geqslant 1$.
Покажем, что $G - S$ содержит только нечетные компоненты, которые мы обозначали $G_1, \ldots, G_k$. Пусть есть четная компонента G_p, содержащая вершину u, которая не является точкой сочленения. Тогда рассмотрим множество $S_p = \cup\{u\}$, мощность которого равна $k + 1$. Получаем, что $k_0(G - S_p) = |S_p| = k + 1$, а это невозможно.
TODO: доказательство не дописано :(■

1.9. Связность.

Def 1.9.1. Две вершины неориентированного графа u и v называются связными, если существует путь из u в v.

Замечание 1.9.2. Связность это отношение эквивалентности, а классы эквивалентности этого отношения называются компонентами связности графа.

Def 1.9.3. Неориентированный граф называется связным, если он состоит из одной компоненты связности.

Def 1.9.4. Две вершины u и v ориентированного графа называются слабо связными, если существует путь из u в v без учета ориентации ребер.

Замечание 1.9.5. Слабая связность это отношение эквивалентности, а классы эквивалентности этого отношения называются компонентами слабой связности.

Def 1.9.6. Ориентированный граф называется слабо связным, если он состоит из одной компоненты слабой связности.

Def 1.9.7. Две вершины u и v ориентированного графа называются сильно связными, если существует путь из u в v и путь из v в u.

Замечание 1.9.8. Сильная связность это отношение эквивалентности, а классы эквивалентности этого отношения называются компонентами сильной связности.

Def 1.9.9. Ориентированный граф называется сильно связным, если он состоит из одной компоненты сильной связности.

Def 1.9.10. Конденсацией ориентированного графа G называется граф G', вершины которого соответствуют компонентам сильной связности графа G, а ребра между вершинами существуют лишь в том случае, если между соответствующими компонентами сильной связности есть ребро.

Def 1.9.11. Вершинная связность $\varkappa(G)$ это минимальное число вершин, которое необходимо удалить для того, чтобы граф стал несвязным или тривиальным.

Def 1.9.12. Вершинная двусвязность это бинарное отношение на ребрах. Два ребра называются двусвязными, если существует два вершиннно-независимых пути между концами этих ребер.

Def 1.9.13. Вершинная двусвязность это отношение эквивалентности, а классы эквивалентности этого отношения называются вершинно-двусвязными компонентами или *блоками*.

Def 1.9.14. Граф называется k-вершинно связным, если после удаления менее чем k вершин он остается связным, т.е. $\varkappa(G) \geqslant k$.

Def 1.9.15. Реберная связность $\lambda(G)$ это минимальное число ребер, которое необходимо удалить для того, чтобы граф стал несвязным или тривиальным.

Def 1.9.16. Реберная двусвязность это бинарное отношение на вершинах. Две вершины называются реберно двусвязными, если между ними существует два реберно-независимых пути.

Def 1.9.17. Реберная двусвязность это отношение эквивалентности, а классы эквивалентности этого отношениями называются реберно-двусвязными компонентами.

Def 1.9.18. Граф называется k-реберно связным, если после удаления менее чем k ребер он остается связным, т.е. $\lambda(G) \geqslant k$.

Замечание 1.9.19. Крайние случаи:

- K_1 не является 1-вершинно связным, но считается связным.
- K_1 не является 2-реберно связным, но считается реберно-двусвязным. Т.е. одна вершина может считаться компонентой реберной двусвязности.

- K_2 не является 2-вершинно связным, но считается вершинно-двусвязным, поэтому K_2 может быть блоком.

Def 1.9.20. Вершина называется точкой сочленения (шарниром), если её удаление увеличивает число компонент связности графа.

Def 1.9.21. Ребро называется мостом, если его удаление увеличивает число компонент связности графа.

Замечание 1.9.22. Таким образом точки сочленения являются 'границами' между блоками (но при этом в блоки они входят), а мосты — между компонентами реберной-двусвязности (но в сами компоненты они не входят).

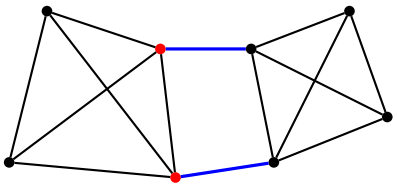
Def 1.9.23. Дерево блоков-точек сочленения это двудольный граф, в котором в одной доле находятся вершины соответствующие блокам, а в другой — точки сочленения.
Если точка сочленения принадлежит блоку, то между ними будет ребро, в противном случае — нет.

1.10. Теорема Уитни.

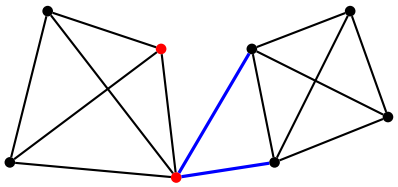
Теорема 1.10.1.

$$\varkappa(G) \leqslant \lambda(G) \leqslant \delta(G)$$

□ Сначала докажем левую часть этого неравенства. Рассмотрим минимальное по включению множество ребер E_d, которые необходимо удалить для того, чтобы граф перестал быть связным.

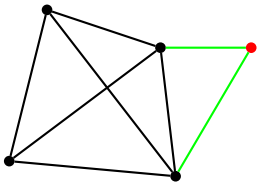


(a) $\varkappa(G) = \lambda(G)$

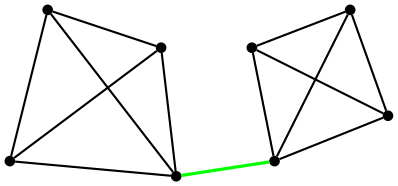


(b) $\varkappa(G) < \lambda(G)$

В худшем случае придется удалить $|E_d| = \lambda(G)$ вершин (по одной вершине на каждое ребро) чтобы граф перестал быть связным. Однако если существует вершины, при удалении которых удалится несколько ребер из E_d, то можно будет удалить меньше вершин для достижения несвязности графа. Значит $\varkappa(G) \leqslant \lambda(G)$.



(a) $\lambda(G) = \delta(G)$



(b) $\lambda(G) < \delta(G)$

Далее докажем правую часть этого неравенства. В худшем случае, чтобы получить несвязный граф, потребуется удалить все ребра инцидентные вершине с наименьшей степенью. В этом случае $\lambda(G) = \delta(G)$. Однако бывают графы, в которых $\lambda(G) < \delta(G)$. ■

1.11. Теорема Менгера.

Def 1.11.1. uv-отделяющее множество это множество вершин, которые необходимо удалить, чтобы между u и v не было пути.

Def 1.11.2. uv-разделяющее множество это множество ребер, которые необходимо удалить, чтобы между u и v не было пути.

Теорема 1.11.3. Для любой пары несмежных вершин u и v в неориентированном графе максимальное число внутренне вершинно-независимых путей из u в v равно минимальной мощности uv-отделяющего множества.

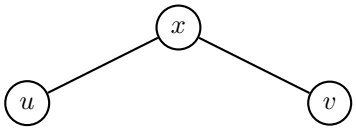
□ В ходе доказательства будем использовать следующие сокращения:

- ВНП := внутренне вершинно независимый путь.
- ОМ := uv-отделяющее множество минимальной мощности.

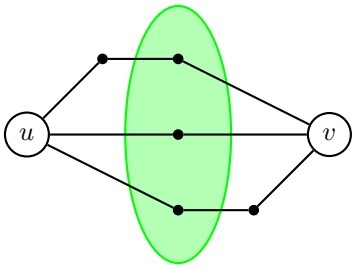
Сразу оговоримся, что если u и v несвязны, то мощность ОМ равна нулю $\Longrightarrow$ теорема верна. Далее будем считать, что u и v связны. Обозначим $S = \{s_1, \ldots, s_n\}$ — ОМ и $P_1, \ldots, P_n$ — множество ВНП.
Индукция по количеству вершин в графе.
База: $n = 3$, тогда мощность ОМ равна единице и существует один ВНП.
Переход: пусть утверждение теоремы верно для графов, у которых менее n вершин. Рассмотрим граф с n вершинами. Возможны три случая:

1. В графе есть вершина x, которая смежная и с u, и с v.

   Для графа без этой вершины будет выполняться предположение индукции. Если же добавить эту вершину в граф, то мощность ОМ увеличится на единицу и при этом появится еще один ВНП $u \to x \to v$.



(a) 1ый случай

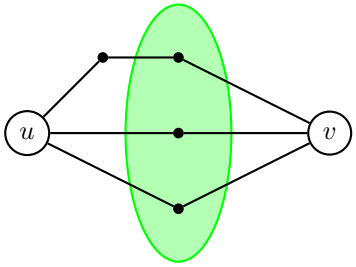


(b) 2ой случай

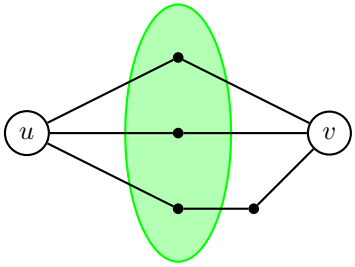
2. Существует ОМ, которое содержит вершину не смежную с u и вершину, не смежную с v.

   Удаление всех вершин из ОМ разобьет его на подграфа A и B. Из графа A построим граф G_1 по следующим правилам:

   - Восстановим все вершины из ОМ
   - Восстановим все ребра, которые соединяли подграф A и ОМ
   - Восстановим вершину v
   - Добавим ребра из v в каждую из вершин ОМ



(a) Граф G_1

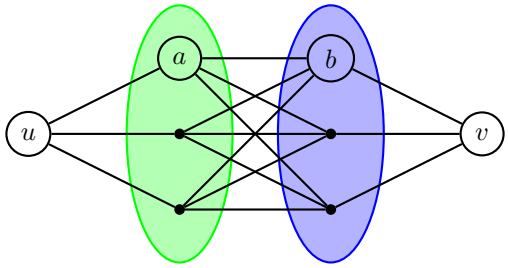


(b) Граф G_2

   Т.к. по условиям пункта в ОМ существовала вершина, не смежная с v, то в полученном графе G_1 будет меньше вершин, чем в изначальном. Применив к нему предположение индукции получим k ВНП (по количеству вершин в ОМ) вида $u \rightsquigarrow s_i \to v$.

   Аналогично из подграфа B можно построить граф G_2 и найти в нем k ВНП вида $u \to s_i \rightsquigarrow v$. Из двух полученных множеств путей очевидным образом составим искомые ВНП вида $u \rightsquigarrow s_i \rightsquigarrow v$.

3. В любой ОМ все вершины смежны с u и не смежны с v или наоборот.



   Если какая либо вершина не входит ни в одно ОМ, то её удаление не повлияет на количество ВНП. Значит её можно удалить и применить предположение индукции, поэтому далее будет считать, что каждая вершина принадлежит некоторому ОМ.

   Среди вершин, смежных с u найдется вершина a, которая будет смежна с вершиной $b \neq u$, причем b будет не смежна с u. Если b не смежна с u, то она смежна с v по условиям пункта. Получем путь $u \to a \to b \to v$.

   Если удалить вершины a и b, то к полученному графу будет применимо предположение индукции. Возвращая вершины a, b в граф мы добавляем один ВНП $u \to a \to b \to v$ и увеличиваем мощность ОМ на один (в него нужно будет взять либо a, либо b).

■

Замечание 1.11.4. Приведенная формулировка и доказательство являются 'вершинной' формой теоремы Менгера. Существует также и реберная форма теоремы Менгера: для любой пары несмежных вершин u и v количество реберно-независимых путей из u в v равно минимальной мощности uv-разделяющего множества.

1.12. Деревья.

Def 1.12.1. Дерево это связный ацикличный граф.

Def 1.12.2. Лес это ацикличный граф, т.е. объединение непересекающихся деревьев.

Def 1.12.3. Укорененное дерево это дерево, в котором одна из вершин обозначена корнем.

Def 1.12.4. Предком вершины называется следующая вершина на кратчайшем пути к корню.

Def 1.12.5. Если u — предок v, то v называется ребенком u.

Def 1.12.6. Если у двух вершин один и тот же родитель, то они называются братьями.

Def 1.12.7. Лист это вершина, у которой нет детей.

Def 1.12.8. Вершины, не являющиеся листьями, называются внутренними.

Def 1.12.9. Дерево называется k-арным, если все его вершины имеют не более k детей. Если $k = 2$, то такое дерево называют бинарным.

Def 1.12.10. Веткой вершины в дереве называется некоторый индуцированный подграф такой, что данная вершина является в нем корнем.

Def 1.12.11. Вес ветки это сумма весов всех входящих в него ребер. Если дерево невзвешенное, то считаем вес каждого ребра равным единице.

Def 1.12.12. Весом вершины называется максимальный из весов веток этой вершины.

Def 1.12.13. Центроид дерева это множество вершин с минимальным весом.

Lm 1.12.14. Центроид дерева содержит либо одну, либо две вершины.

□ Рассмотрим невзвешенное дерево. На каждом шаге будем убирать все листья из текущего дерева, при этом степени оставшихся вершин будут уменьшаться на единицу. Будем так повторять, пока существует вершина со степенью более единицы.
В итоге останется либо одна вершина со степенью ноль, либо две вершины со степенью один. Значит в исходном графе именно у этой вершины (или у этих двух вершин) была наименьшая степень. ■

Def 1.12.15. Дерево называется помеченным, если каждой из его вершин соответствует уникальная метка от 1 до n.

Def 1.12.16. Код Прюфера эт уникальная последовательность меток от 1 до n длины $n-2$, которая соответствует уникальному помеченному дереву на n вершинах.

Lm 1.12.17. Всего существует n^{n-2} помеченных деревьев на n вершинах.

□ Между деревьями на n вершинах и кодами Прюфера длина $n-2$ действует биекция. Существует n^{n-2} кодов Прюфера длины $n-2$. ■

Кодирование [визуализация]:
1. Выбираем лист с наименьшей меткой.
2. Добавляем в ответ его родителя, после чего удаляем этот лист.
3. Повторяем шаги $1-2$ $n-2$ раза.

Декодирование [визуализация]:
1. Выписываем в строку все метки от 1 до n.
2. Среди выписанных меток ищем первую, которой нет в коде Прюфера.
3. Добавляем в граф ребро между найденной меткой и первой меткой в коде Прюфера.
4. Удаляем найденную метку из последовательности меток, а также удаляем первый элемент в коде Прюфера.
5. Повторяем шаги $2-4$ пока не закончится код Прюфера.
6. В последовательности останется 2 метки, добавляем ребро между ними.

Замечание 1.12.18. Кодировать и декодировать код Прюфера можно за $O(n)$. Алгоритм описан здесь.

1.13. Алгоритм Дейкстры.

Цель: ищем кратчайшие пути от одной вершины до всех остальных в графе с неотрицательными весами.
Оценка по времени зависит от реализации приоритетной очереди:

- Наивная $O(V^2 + E)$
- На бинарной куче $O(V + E \log V)$
- На Фибоначчиевой куче $O(V \log V + E)$

Алгоритм:

1. Будем поддерживать множество вершин A, расстояние до которых минимально.
2. Изначально $A = \varnothing$, $\forall v \neq s \in V$: $d_v = +\infty$, $d_s = 0$, где d_x это расстояние от стартовой вершины до вершины x.
3. Из непосещенных вершин выберем вершину u, у для которой d_u минимально.
4. Добавим u в A, причем d_u будет являться кратчайшим расстоянием до вершины u.
5. Далее будем обновлять расстояние до непосещенных вершин, смежных с выбранной:
   $\forall (u, p) \in E$: $d_p = \min(d_p, d_u + w_{up})$, где w_{up} это вес ребра между вершинами u и p.
6. Будем повторять шаги $3 - 5$ пока в A не попадут все достижимые вершины (либо в A не попадет финишная вершина).

О корректности:
Корректность покажем по индукции.
База: согласно шагу 2 расстояние до стартовой вершины s равно нулю.
Переход: пусть $\forall a \in A$: $d_a = \text{dist}(a)$ это действительно кратчайшие расстояния до вершин в A (кратчайшие расстояния будем обозначать $\text{dist}(a)$, а расстояния, найденные в ходе работы алгоритма — d_a). Покажем, что $d_u = \min\limits_{a \in A}(d_a + w_{au})$ это кратчайшее расстояние до u.
Рассмотрим кратчайший путь $s \rightsquigarrow u$. Пусть $a \in A$ это последняя вершина на этом пути, которая принадлежит A, а b — это следующая за ней вершина на пути в u. Тогда $\text{dist}(u) \geqslant d_a + w_{ab}$.
Но т.к. $d_u = \min\limits_{a \in A}(d_a + w_{au})$, то $d_a + w_{ab} \geqslant d_u$, значит $\text{dist}(u) \geqslant d_u$. Итого d_u это кратчайшее расстояние до u.

1.14. Алгоритм Форда-Белмана.

Цель: ищем кратчайшие пути от одной вершины до всех остальных (допускаются отрицательные ребра).
Оценка по времени: $O(VE)$
Алгоритм:

1. Сделаем $|V| - 1$ итерацию (именно столько, т.к. в кратчайшем пути не более $|V| - 1$ ребра).
2. На каждой итерации будем проходить по всем ребрам и релаксировать их.
3. Релаксация ребра выглядит так: пусть дано ребро (u, v). Тогда обновим расстояние до v следующим образом: $d_v = \min(d_v, d_u + w_{uv})$.

О корректности:
Построим следующую динамику: пусть $d[k, v]$ это длина кратчайшего пути из стартовой вершины s в вершину v, содержащего не более k ребер. Тогда получаем:

- Изначально:
  - $\forall v \neq s \in V$: $d[0, v] = +\infty$
  - $d[0, s] = 0$
  - $\forall v \in V, \forall i > 0$: $d[i, v] = +\infty$
- Переход $d[k + 1, v] = \min\left(d[k, v], \min\limits_{u \in V}(d[k, u] + w_{uv})\right)$

  Он вытекает из следующих соображений: пусть мы хотим получить кратчайший путь $s \rightsquigarrow v$, который состоит из не более чем $k + 1$ ребра. Возможно два случая:
  - Можно взять длины кратчайшего пути, который состоит из не более чем k ребер.
  - Можно найти путь в некоторую смежную вершину u, который содержит не более k ребер и присоединить к нему ребро (u, v). Среди всех таких вершин u берем такую, чтобы длина итогового пути была наименьшей.

  Из этих двух случаев мы выбираем наименьший.

Динамика корректна по построению, а алгоритм Форда-Беллмана это следствие из этой динамики: мы убираем индекс k, чтобы оптимизировать затрачиваемую память.

1.15. Алгоритм Флойда-Уоршелла.

Цель: ищем кратчайшие пути от всех вершин до всех.
Оценка по времени: $O(V^3)$
Алгоритм:

1. Рассматриваем каждую из вершин $k \in V$.
2. На каждой итерации проходим по всем возможным парам вершин $\langle i, j \rangle$ и пытаемся прорелаксировать путь $i \rightsquigarrow j$ через вершину k.
3. Релаксация пути через вершину k выглядит так: пусть дана пара вершин $\langle i, j \rangle$. Тогда обновим расстояние от i до j следующим образом: $d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$.

О корректности:
Построим следующую динамику: пусть $d[k, i, j]$ это длина кратчайшего пути из вершины i в вершину j, содержащего только вершины с индексами меньше k. Тогда получаем:

- Изначально:
  - $\forall (i, j) \in E$: $d[0, i, j] = w_{ij}$
  - $\forall i \in V$: $d[0, i, i] = 0$
  - Во всех остальных случаях $d[k, i, j] = +\infty$
- Переход $d[k+1, i, j] = \min\Big(d[k, i, j], d[k, i, k] + d[k, k, j]\Big)$

  Он вытекает из следующих соображений:
  - Путь $i \rightsquigarrow j$ нельзя улучшить через вершину k, тогда $d[k+1, i, j] = d[k, i, j]$.
  - Путь $i \rightsquigarrow j$ можно улучить через вершину k, тогда $d[k+1, i, j] = d[k, i, k] + d[k, k, j]$.

  Из этих двух случаев мы выбираем наименьший.

Динамика корректна по построению, а алгоритм Флойда-Уоршелла это следствие из этой динамики: мы убираем индекс k, чтобы оптимизировать затрачиваемую память.

1.16. Иерархическая кластеризация.

Def 1.16.1. Иерархическая кластеризация это построение иерархии (дерева) кластеров.

Для визуализации результатов используется дендрограмма — дерево, построенное по матрице расстояний между кластерами. В узлах дерева находятся подмножества объектов из обучающей выборки. Объединения узлов между ярусами дендрограммы соответствует объединению кластеров.
Алгоритмы WPGMA/UPGMA:
Задача: построить дерево по матрице расстояний между листьями.

Def 1.16.2. Ультраметричность — следующее свойство: $d_{xy} \leqslant \max(d_{xz}, d_{yz})$, где x, y, z - листья в полученном дереве, а d_{xy} это расстояние между листьями x и y.

Замечание 1.16.3. Полученное дерево уникальное и обладает свойством ультраметричности.

Алгоритм:

1. Ищем две ближайшие вершины x и y.
2. В результирующее дерево добавляем вершину u, которая родителем для двух этих вершин.
3. Вычисляем веса ребер: $\text{dist}(u, x) = \text{dist}(u, y) = \frac{1}{2} \cdot \text{dist}(x, y)$.
4. Обновляем матрицу расстояний: убираем из неё x, y и добавляем u.
5. Теперь необходимо пересчитать расстояния. В зависимости от алгоритма будут разные формулы пересчета:

   (a) WPGMA

   Расстояние от кластера до вершины равно среднему арифметическому расстояний от каждой компоненты кластера до этой вершины:

   $$u = (x, y)$$
   $$\text{dist}(u, z) = \frac{1}{2} \cdot (\text{dist}(z, x) + \text{dist}(z, y))$$

   (b) UPGMA

   Чтобы найти расстояние от кластера до вершины нужно умножить расстояние от вершины до каждой из компонент кластера на размер этой компоненты. Результаты нужно сложить, после чего поделить на количество вершин в кластере

$$u = (xy, z)$$
$$\operatorname{dist}(u, w) = \frac{1}{2+1}(\operatorname{dist}(xy, w) \cdot 2 + \operatorname{dist}(z, w) \cdot 1)$$

6. Будем повторять шаги $1 - 5$ пока матрица не сожмется в одну ячейку.

Замечание 1.16.4. В алгоритме UPGMA можно пересчитывать расстояния от вершин до кластера по-другому: можно взять среднее из расстояний от рассматриваемой вершины до каждой из вершин кластера:

$$u = (xy, z) \Longrightarrow \operatorname{dist}(u, w) = \frac{1}{3}(\operatorname{dist}(x, w) + \operatorname{dist}(y, w) + \operatorname{dist}(z, w))$$

Результат будет такой же, но потребуется дополнительно хранить исходную матрицу расстояний.

2. Комбинаторика

2.1. Упорядоченные размещения. Перестановки и k-перестановки.

Def 2.1.1. Упорядоченным размещением n элементов из Σ называется функция $s\colon [n] \to \Sigma$, где Σ это алфавит, $[n] = \{1, \ldots, n\}$.

- $[n]$ это домен функции
- Σ это кодомен функции
- $s(i)$ это изображение (*image*) для i
- $\operatorname{ran} s = \{x \in \Sigma \mid \exists i \in [n]\colon s(i) = x\}$.

Замечание 2.1.2. Также упорядоченное размещение может быть представлено в виде кортежа $s = (s_1, \ldots, s_n)$, строки $s = s(1)s(2)\ldots s_n$ или последовательности $s(i) = 3i + 2$.

Def 2.1.3. Перестановка это биективная функция $\pi\colon [n] \to \Sigma$.

S_n это множество всех перестановок для $\Sigma = [n]$. $|S_n| = n!$

Def 2.1.4. k-перестановка это упорядоченное размещение k различных элементов из Σ, т.е. это инъективная функция $\pi_k\colon [k] \to \Sigma$.

$P(n, k)$ это множество всех k-перестановок для $\Sigma = [n]$. $|P(n, k)| = \dfrac{n!}{(n-k)!}$

Замечание 2.1.5. Существуют также циклические k-перестановки. π_1 и π_2 называются циклическими эквивалентами, если существует сдвиг $s \in [k]$ такой, что $\forall i, j \in [k]\colon i + s = j(\mod k) \Longrightarrow \pi_1(i) = \pi_2(j)$
Количество циклических k-перестановок в k раз меньше, чем количество обычных k-перестановок:
$P_c(n, k) = \dfrac{n!}{k \cdot (n-k)!}$.

2.2. Мультимножества.

Def 2.2.1. Мультимножество Σ^* это упорядоченная пара вида $\langle \Sigma, r \rangle$, где Σ это множество, а r это функция, показывающая число повторений элемента $r\colon \Sigma \to \mathbb{N}$.

Количество перестановок мультимножества можно найти по формуле (мультиномиальная теорема):

$$|P(\Sigma^*, n)| = \frac{n!}{r_1! \cdot \ldots \cdot r_s!}$$

где $n = |\Sigma^*|$ и $s = |\Sigma|$.

Замечание 2.2.2. О сочетаниях бесконечных мультимножеств
Пусть дано мультимножество $\Sigma^* = \langle \Sigma, r \rangle$, в котором $\forall r_i \geqslant k$, $|\Sigma| = s$ и требуется найти количество сочетаний.
Для решения этой задачи можно воспользоваться методом Stars&Bars. Пусть у нас есть $s-1$ перегородка и k элементов из Σ^*, тогда количество k-сочетаний будет равно

$$\binom{k+s-1}{s-1} = \binom{k+s-1}{k}$$

т.к. нам нужно из $k + s - 1$ позиций выбрать $s - 1$ позицию для перегородок, а остальные позиции займут элементы.

2.3. Сочетания.

Def 2.3.1. Неупорядоченное размещение k элементов множества Σ это мультимножество S размера k.

Def 2.3.2. k-сочетание (k-подмножество) это неупорядоченное размещение k различных элементов из Σ.

Множество всех k-подмножеств обозначается $\binom{\Sigma}{k}$, если $|\Sigma| = n$, то получаем $\binom{n}{k} = C_n^k$.
Для подсчета количества k-подмножеств воспользуемся тем, что каждое k-подмножество имеет $k!$ перестановок:

$$|P(n,k)| = \binom{n}{k} \cdot k! \Longrightarrow \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

2.4. Композиции.

Def 2.4.1. Слабой композицией числа $n \geqslant 0$ на k частей называется решение уравнения $b_1 + \ldots + b_k = n$ при условии $b_i \geqslant 0, b_i \in \mathbb{Z}$.

Для подсчета числа слабых композиций воспользуемся методом Stars&Bars: пусть у нас есть n единиц и $k-1$ перегородка между ними, значит всего получаем

$$\Big|\text{Количество слабых композиций } n \text{ на } k \text{ частей}\Big| = \binom{n+k-1}{k-1}$$

Def 2.4.2. Композицией числа $n \geqslant 0$ на k частей называется решение уравнения $b_1 + \ldots + b_k = n$ при условии $b_i > 0, b_i \in \mathbb{Z}$.

Количество композиций числа можно посчитать следующим образом: возьмем из n k единиц (по одной для каждого слагаемого), а для оставшихся $n-k$ единиц решим задачу о слабой композиции. Итого получим

$$\Big|\text{Количество композиций } n \text{ на } k \text{ частей}\Big| = \binom{n-1}{k-1}$$

Замечание 2.4.3. Число композиций числа n на произвольное число частей (т.е. $k = 1 \ldots n$) равно 2^{n-1}.

2.5. Разбиения множеств. Числа Стирлинга второго рода.

Def 2.5.1. Количество способов разбить множество из n элементов на k непустых непересекающихся подмножеств (так, чтобы они в объединении давали исходное множество) определяется числами Стирлинга второго рода.

Def 2.5.2. Числа Стирлинга второго рода определяются формулой

$$S(n,k) = S(n-1,k-1) + k \cdot S(n-1,k)$$

Эта формула получается следующим образом: пусть требуется разбить множество размера n на k частей. Есть два варианта действий:

- Взять один из элементов в отдельную часть, а оставшиеся $n-1$ элементов разделить на $k-1$ частей.
- Разделить $n-1$ элемент на k частей, а оставшийся элемент добавить в любую из получившихся k частей

Эти два варианта и определяют два слагаемых в формуле выше.

Замечание 2.5.3. О крайних случаях

- $S(n,n) = 1$, причем $S(0,0) = 1$
- $S(n,k) = 0$ если $k \leqslant 0$, т.к. мы не может разделить множество на неположительное число частей.
- $S(n,k) = 0$, если $n < k$, т.к. подмножества должны быть непустыми.

Замечание 2.5.4. Количество способов разбить множество на произвольное число подмножеств (т.е.$k = 1 \ldots n$) называется числами Белла:

$$B(n) = \sum_{k=1}^{n} S(n,k)$$

2.6. Разбиения чисел.

Def 2.6.1. Разбиение числа n на k частей это решение уравнения $b_1 + \ldots + b_k = n$ при условии $b_1 \geqslant \ldots \geqslant b_k \geqslant 1$.

Для подсчета числа разбиений существует специальная функция $p(n, k)$, которая определена как

$$p(n, k) = p(n-1, k-1) + p(n-k, k)$$

Это соотношение построено исходя из следующих соображений: пусть необходимо разбить число n на k слагаемых. Есть две стратегии действий:

- Можно разбить число $n-1$ на $k-1$ слагаемое, а k-тым слагаемым будет единица.
- Можно разбить число $n-k$ на k слагаемых, после чего добавить ко всем слагаемым по единице.

Две эти стратегии определяют два слагаемых в формуле выше.

Замечание 2.6.2. О крайних случаях

- $p(0, 0) = 1$ — полагаем так, чтобы все было хорошо.
- $p(n, k) = 0$, если $n \leqslant 0$, т.к. невозможно представить неположительное число в виде суммы положительных слагаемых.
- $p(n, k) = 0$, если $k \leqslant 0$, т.к число слагаемых должно быть положительным.

2.7. Принцип включения-исключения.

Def 2.7.1. Принцип включений-исключений это комбинаторный принцип, который позволяет вычислить мощность объединения конечного числа множеств, которые могут пересекаться друг с другом.

Введем обозначения:

- X — данное множество элементов
- $|X| = n$
- $P_1, \ldots, P_m$ — 'плохие' свойства
- $X_i = \{x \in X \mid x \text{ содержит свойство } P_i\}$
- $S \subseteq [m]$ — некоторое подмножество свойств
- $N(S) = \bigcap\limits_{i \in S} X_i = \{x \in X \mid x \text{ содержит все свойства из } S\}$

Теорема 2.7.2.

$$|X \setminus (X_1 \cup \ldots \cup X_m)| = \sum_{S \subseteq [m]} (-1)^{|S|} \cdot |N(S)|$$

□ Возьмем произвольный $x \in X$. Возможны два случая:

1. x не содержит ни одного свойства $P_1, \ldots, P_m$, тогда $x \in N(\varnothing)$ и $x \notin N(S)\ \forall S \neq \varnothing$. Значит x 'добавляет' 1 к общей сумме.
2. x содержит некоторый набор свойств из k свойств $T \subseteq [m]$. Значит $x \in S$ только тогда, когда $S \subseteq T$. Получаем

$$\sum_{S \subseteq T} (-1)^{|S|} \cdot 1 = \sum_{i=1}^{k} \binom{k}{i} (-1)^i \tag{1}$$

Заметим, что по биномиальной теореме

$$(1 + (-1))^k = \sum_{i=1}^{k} \binom{k}{i} (-1)^i \cdot 1^{k-i} = \sum_{i=1}^{k} \binom{k}{i} (-1)^i = 0$$

Подставляя это в 1 получаем, что x ничего не вносит в итоговую сумму.

■

Def 2.7.3. Беспорядок — это перестановка чисел от 1 до n, в которой ни один элемент не стоит на своём месте.

Теорема 2.7.4. Количество беспорядков из n элементов равно субфакториалу числа n и вычисляется о формуле:

$$!n = \sum_{i=0}^{n} (-1)^i \cdot \frac{n!}{i!}$$

3. Конечные автоматы

3.1. Формальные языки. Операции над формальными языками.

Def 3.1.1. Формальный язык $\mathcal{L}$ это некоторое подмножества множества всех слов из заданного алфавита Σ.

$$\mathcal{L} \subseteq \Sigma^* \qquad \Sigma^* = \sum_{k=0}^{\infty} \Sigma^k$$

Формальный язык может задан перечислением (как множество слов) либо с помощью грамматики ($\approx$ формулой). Т.к. формальный язык по сути является множеством слов, то с ним можно производить операции характерные для множеств:

- Объединение $\mathcal{L}_1 \cup \mathcal{L}_2$
- Пересечение $\mathcal{L}_1 \cap \mathcal{L}_2$
- Дополнение $\neg\mathcal{L}_1$

Однако помимо этих операций с языками можно производить и другие:

- Конкатенация $\mathcal{L}_1 \cdot \mathcal{L}_2 = \{x + y \mid x \in \mathcal{L}_1, y \in \mathcal{L}_2\}$

  Эта операция порождает новый язык, в котором каждое слово является конкатенацией произвольного слова из первого языка с произвольным словом из второго языка.

  В общем случае она некоммутативна и выполняется равенство $|\mathcal{L}_1 \cdot \mathcal{L}_2| \leqslant |\mathcal{L}_1| \cdot |\mathcal{L}_2|$
- Возведение в степень $\mathcal{L}^k$

  Это сокращенная запись для конкатенации языка самим с собой k раз.

  Стоит отметить, что $L^0 = \{\varepsilon\}$, где ε это специальное 'пустое' слово нулевой длины.
- Звезда Клини $\mathcal{L}^*$

  Это объединение всевозможных степеней языка $\mathcal{L}^* = \bigcup\limits_{k=0}^{\infty} L^k$

3.2. Регулярные языки. Регулярные выражения.

Множество регулярных языков задается рекуррентно. Определим нулевое поколение регулярных языков как

$$Reg_0 = \{\varnothing, \varepsilon, \{c\} \mid c \in \Sigma\}$$

Далее определим переход к следующему поколению:

$$Reg_{i+1} = Reg_i \cup \{\mathcal{L}_1 \cup \mathcal{L}_2, \mathcal{L}_1 \cdot \mathcal{L}_2, \mathcal{L}_1^* \mid \mathcal{L}_1, \mathcal{L}_2 \in Reg_i\}$$

Тогда множество регулярных языков задается как

$$REG = \bigcup_{k=0}^{\infty} Reg_k = Reg_\infty$$

Lm 3.2.1. Множество регулярных языков замкнуто относительно объединения, конкатенации и звезды Клини.

□ Пусть $\mathcal{L}_1 \in Reg_i$, $\mathcal{L}_2 \in Reg_j$, тогда

$$\mathcal{L}_1 \cup \mathcal{L}_1, \mathcal{L}_1 \cdot \mathcal{L}_2, \mathcal{L}_1^* \in Reg_{max(i,j)+1} \subseteq REG$$

■

Def 3.2.2. Регулярные выражения это один из способов задания регулярного языка. Они определяются рекурсивно.

Пусть α и β это регулярные выражения соответствующие регулярным языкам $\mathcal{A}$ и $\mathcal{B}$.

Регулярный язык	Регулярное выражение	Примечание
$\varnothing$	$\varnothing$	Пустое множество
ε	ε	Пустое слово
c	c	Один символ из алфавита
$\mathcal{A} \cup \mathcal{B}$	$\alpha \vert \beta$	Объединение
$\mathcal{A} \cdot \mathcal{B}$	$\alpha\beta$	Конкатенация
$\mathcal{A}^*$	α^*	Звезда Клини

Также для удобства в синтаксис регулярных выражений добавлены:

- Точка соответствует одного любому символу из алфавита.
- Скобки для определения порядка операций.
- Группировка $[abc]$, которая означает 'один любой символ из перечисленных в квадратных скобках'.
- Дополнение $[\hat{}abc]$, которая означает 'один любой символ кроме перечисленных в квадратных скобках'.

3.3. Детерминированный конечный автомат.

Детерминированный конечный автомат (ДКА) это кортеж вида $\langle \Sigma, Q, q_0, F, \delta \rangle$, где

- Σ это алфавит
- Q это множество состояний
- q_0 это начальное состояние ($q_0 \in Q$)
- F это множество принимающих состояний ($F \subseteq Q$)
- δ это функция перехода $\delta \colon Q \times \Sigma \to Q$

Мы рассматриваем только автоматы-детекторы, т.е. им на вход подается слово, после обработки которого автомат оказывается в одном из состояний. Если это принимающее состояние, то говорят, что слово принимается автоматом (автомат допускает слово), в противном случае — слово отвергается автоматом.
Вычисление:

Def 3.3.1. Снимок (*snapshot*) это упорядоченная пара вида $\langle q, s \rangle$, где $q \in Q$ это текущее состояние, а s это еще непросмотренная часть входной строки.

Множество всех снимков обозначается $SNAP$. На этом множество можно ввести бинарное отношение $\vdash$, которое будет показывать, можно ли перейти от одного снимка к другому:

$$\langle q_1, s_1 \rangle \vdash \langle q_2, s_2 \rangle \iff \begin{cases} s_1 = \alpha s_2 \\ \delta(q_1, \alpha) = q_2 \end{cases}$$

Def 3.3.2. Автоматный язык $\mathcal{L}(\mathcal{A})$ это язык, состоящий из слов, принимаемых автоматом:

$$\mathcal{L}(\mathcal{A}) = \{ s \mid \langle a_0, s \rangle \vdash^* \langle f, \varepsilon \rangle, f \in F \}$$

Def 3.3.3. Множество автоматных языков AUT это множество языков, для которых существует автомат их принимающий.

$$AUT = \{ \mathcal{X} \mid \exists \mathcal{A} \colon \mathcal{L}(\mathcal{A}) = \mathcal{X} \}$$

3.4. Недетерминированный конечный автомат.

Недетерминированный конечный автомат (НКА) это кортеж вида $\langle \Sigma, Q, q_0, F, \delta \rangle$, где

- Σ это алфавит
- Q это множество состояний
- q_0 это начальное состояние ($q_0 \in Q$)
- F это множество принимающих состояний ($F \subseteq Q$)
- δ это функция перехода $\delta \colon Q \times \Sigma \to 2^Q$

Таким образом отличие НКА от ДКА заключается в функции перехода: если в ДКА мы переходили от одного состояния к другому по определенному символу, то в НКА по одному символу можно перейти сразу в несколько новых состояний.
Вычисление:
Как и для ДКА, для НКА можно определить снимки:

$$\langle q_1, s_1 \rangle \vdash_{NFA} \langle q_2, s_2 \rangle \iff \begin{cases} s_1 = \alpha s_2 \\ q_2 \in \delta(q_1, \alpha) \end{cases}$$

Тогда язык, принимаемый НКА будет иметь вид

$$\mathcal{L}(\mathcal{A}) = \{ s \mid \langle a_0, s \rangle \vdash^*_{NFA} \langle f, \varepsilon \rangle, f \in F \}$$

3.5. Преобразование НКА в ДКА.

Алгоритм:

1. В начале алгоритма ДКА пустой. Будем постепенно по одной добавлять в него новые вершины.
2. Изначально во множестве S содержится только одна стартовая вершина s.
3. Проходим по всем вершинам в S.
4. Проходим по всем символам алфавита.
5. Для текущей вершины v и текущего символа c смотрим на множество вершин P, в которое мы можем перейти.
6. Создаем новую вершину p, которая будет являться как бы объединением вершин в P.
7. Если такой вершины нет в ДКА, то добавляем её в ДКА и в множество S'.
8. В ДКА добавляем переход $v \xrightarrow{c} p$.
9. После того, как мы прошлись по всем вершинам в S и всем символам в алфавите (шаги $3 - 8$) нужно обновить S: $S = S'$, а также очистить множество S'.

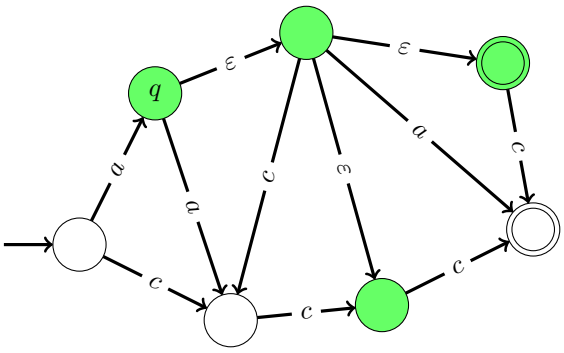
TODO: Визуализация

3.6. ε -НКА. Преобразование ε -НКА в НКА.

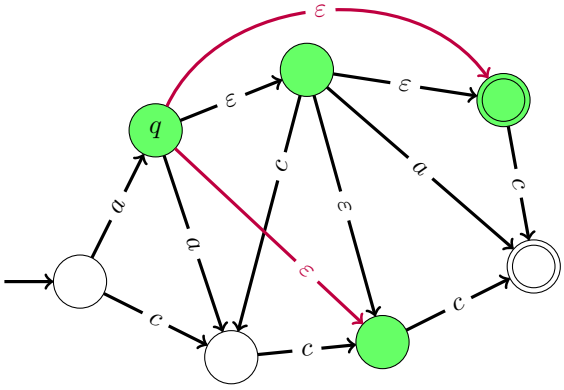
Def 3.6.1. ε-НКА это НКА в котором $\varepsilon \in \Sigma$, т.е. возможны переходы из одного состояния в другое без 'чтения' символов входной строки.

Def 3.6.2. ε-замыканием вершины называется множество вершин, в которое можно добраться из этой вершины только по ε-переходам.

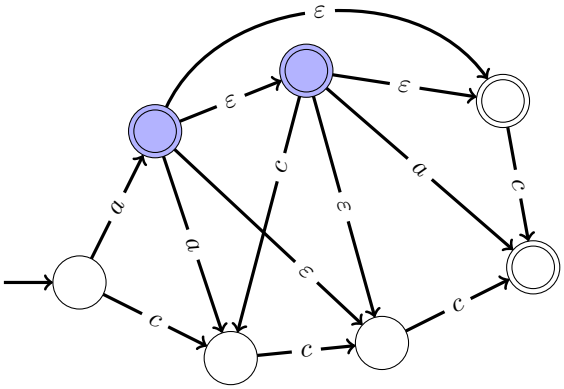
Алгоритм:



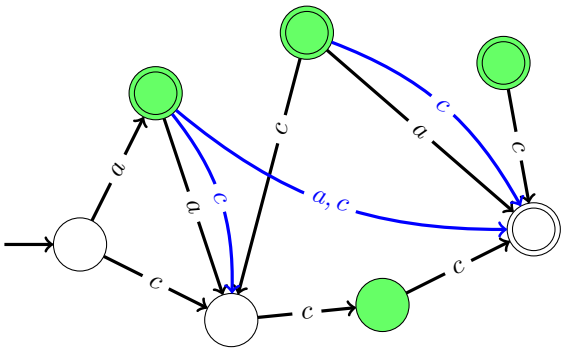
(a) Шаг 1



(b) Шаг 2



(a) Шаг 3



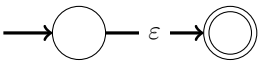
(b) Шаг 4

1. Изначальное состояние ε-НКА
2. Для каждого состояния q добавим ε-переходы в каждое из состояний его ε-замыкания (кроме самого состояния q).
3. Пройдемся по всем состояниям. Если из текущего состояния есть ε-переход в принимающее состояние, то сделаем текущее состояние принимающим.
4. Для каждого перехода вида $q_1 \xrightarrow{\varepsilon} q_2 \xrightarrow{c} q_3$ добавим в НКА переход $q_1 \to cq_3$. Теперь все ε-переходы можно удалить.

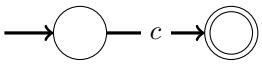
3.7. Построение ε-НКА по регулярному выражению (построение Томпсона).

Регулярное выражение задает регулярный язык. Покажем, что для любого регулярного языка существует автомат, который его принимает. Сделаем это по индукции.
База: для регулярных языков нулевого поколения можно построить следующие автоматы:

(a) $\mathcal{L} = \varnothing$

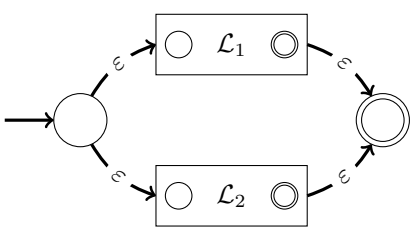


(b) $\mathcal{L} = \{\varepsilon\}$

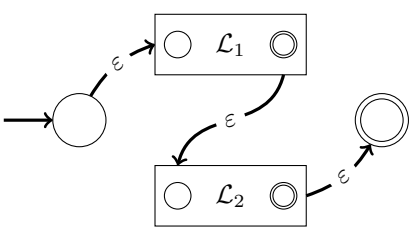


(c) $\mathcal{L} = \{c \in \Sigma\}$

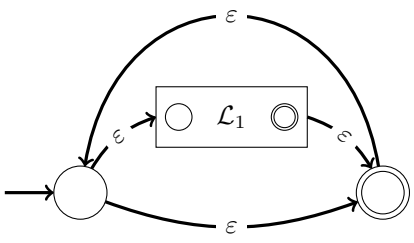
Переход: пусть мы построили автоматы для всех языков менее чем k-ого поколения и хотим построить автомат для языка k-ого поколения. Возможны три случая:



(a) $\mathcal{L}' = \mathcal{L}_1 \cup \mathcal{L}_2$



(b) $\mathcal{L}' = \mathcal{L}_1 \cdot \mathcal{L}_2$



(c) $\mathcal{L}' = \mathcal{L}_1^*$

Таким образом мы построили автомат для регулярного языка k-ого поколения. Значит сможем построить автомат для регулярного языка любого поколения, т.е. для любого регулярного языка.

3.8. Теорема Клини.

Теорема 3.8.1. Множество автоматных языков равно множеству регулярных языков:

$$AUT = REG$$

$\square \implies$ Покажем, что $AUT \subseteq REG$.
Пусть дан НКА. Будем считать, что его 'ребра' являются регулярными выражениями. Таким образом задача состоит в том, чтобы сжать автомат до состояния, в котором останется только начальное и конечные состояния. Будем выполнять две операции:

- Сжатие ребер: заменим переходы $q_1 \xrightarrow{c_1} q_2$ и $q_1 \xrightarrow{c_2} q_2$ на переход $q_1 \xrightarrow{c_1|c_2} q_2$.
- Сжатие вершин: если есть пара переходов вида $q_1 \xrightarrow{c_1} q_2 \xrightarrow{c_3} q_3$, то удалим их и добавим новый переход $q_1 \xrightarrow{c_1 c_2^* c_3} q_3$, где c_2 это символ на петле q_2 (если такой петли нет, то эта часть просто опускается).

Выполняем эти две операции пока не останутся только стартовое и финишные состояния, т.е. $q_0 \xrightarrow{\alpha_i} f_i \ \forall f_i \in F$. Тогда искомым регулярным выражением будет $(\alpha_1| \dots |\alpha_m)$.
$\impliedby$ Покажем, что $REG \subseteq AUT$.
Для любого регулярного выражения можно построить ε-НКА по алгоритму Томпсона (см. 3.7.). ■

4. Рекуррентные соотношения

4.1. Реккурентные соотношения. Характеристические уравнения.

Def 4.1.1. Рекуррентное соотношение это последовательность, в которой текущий член задается через предыдущие.

Def 4.1.2. Рекуррентное соотношение называется однородным, если оно не содержит свободных членов.
Рекуррентное соотношение со свободными членами называется неоднородными.

Def 4.1.3. Характеристическое уравнение это уравнение, построенное по данному рекуррентному соотношению заменой a_n на r^n.

Оно может быть использовано для решения однородных рекуррентных соотношений. Для этого нужно найти его корни, после чего записать общий вид решения. Далее нужно подставить начальные условия, чтобы определить константы и получить замкнутую формулу для рекуррентного соотношения.
Пример:
Пусть требуется решить (т.е. найти замкнутую формулу) следующее рекуррентное соотношение: $a_n = 6a_{n-1} - 9a_{n-2}, a_0 = 1, a_2 = 45$.
Составим характеристическое уравнение и найдем его корни

$$\begin{aligned} r^n &= 6r^{n-1} - 9r^{n-2} \\ r^2 - 6r + 9 &= 0 \\ r_1 = 3, r_2 &= 3 \end{aligned}$$

Значит общее решение будет иметь вид $a_n = c_1 \cdot 3^n + c_2 \cdot n \cdot 3^n$. Найдем константы:

$$\begin{aligned} a_0 = c_1 3^0 + c_2 \cdot 0 \cdot 3^0 = c_1 &= 1 \\ a_2 = c_1 3^2 + c_2 \cdot 2 \cdot 3^2 = 9c_1 + 18c_2 &= 45 \\ c_1 + 2c_2 &= 5 \\ c_2 &= 2 \end{aligned}$$

Ответ: $a_n = 3^n + 2n \cdot 3^n$.

Замечание 4.1.4. Решение однородных рекуррентных соотношений очень похоже на решение линейных однородных дифференциальных уравнений.

4.2. Асимптотический анализ. Алгоритмы разделяй-и-властвуй.

Асимптотический анализ это анализ поведения функции на бесконечности.
Пусть $T(n)$ это функция 'количества работы' (время, память и т.п.) в зависимости от объема входных данных, а $g(n)$ это некоторая функция, тогда:

Def 4.2.1. $g(n)$ называется оценкой сверху для $T(n)$, если

$$T(n) \in O(g(n)) \iff \exists n_0, c > 0 \mid \forall n \geqslant n_0 \colon T(n) \leqslant c \cdot g(n)$$

Def 4.2.2. $g(n)$ называется оценкой снизу для $T(n)$, если

$$T(n) \in \Omega(g(n)) \iff \exists n_0, c > 0 \mid \forall n \geqslant n_0 \colon c \cdot g(n) \leqslant T(n)$$

Def 4.2.3. Если $g(n)$ одновременно является и оценкой сверху, и оценкой снизу, то $g(n)$ называется точной оценкой.

$$T(n) \in \Theta(g(n)) \iff \begin{cases} T(n) \in O(g(n)) \\ T(n) \in \Omega(g(n)) \end{cases}$$

Также можно определять оценки через пределы:

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$	Оценка	Комментарий
$c \neq 0, c \neq \infty$	$f \in \Theta(g)$	f растет так же быстро, как и g
$c \neq \infty$	$f \in O(g)$	f растет не быстрее g
$c \neq 0$	$f \in \Omega(g)$	f растет не медленнее g
$c = 1$	$f \sim g$	f и g эквивалентны
$c = 0$	$f \in o(g)$	f растет медленнее g
$c = \infty$	$f \in \omega(g)$	f растет быстрее g

Некоторые свойства асимптотических оценок:

$$\left.\begin{matrix} f(n) \in O(g(n)) \\ f(n) \in \Omega(g(n)) \end{matrix}\right\} \iff f(n) \in \Theta(g(n)) \qquad f(n) \sim g(n) \implies f(n) \in \Theta(g(n))$$
$$f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n)) \qquad f(n) \in o(g(n)) \implies f(n) \in O(g(n))$$
$$f(n) \in o(g(n)) \iff g(n) \in \omega(f(n)) \qquad f(n) \in \omega(g(n)) \implies f(n) \in \Omega(g(n))$$

Def 4.2.4. Алгоритмы разделяй-и-властвуй основываются на том, что на каждой итерации они дробят текущую задачу на несколько частей, а потом объединяют результаты работы алгоритма для каждой из этих частей. Деление останавливается, когда алгоритм доходит до некоторого базового случая.

На примере сортировки слияниям:

$$T(n) = \underbrace{2T(n/2)}_{\text{рекурсивная работа}} + \underbrace{n}_{\text{работа по разделению и слиянию}}$$

TODO: Скобки пофиксить, либо оформить иначе
Для асимптотической оценки времени работы таких алгоритмов чаще всего используется Мастер теорема или метод Акра-Бацци.

4.3. Мастер теорема.

Мастер теорема позволяет давать асимптотическую оценку некоторым рекуррентным соотношениям (которые чаще всего возникают в алгоритмах разделяй-и-властвуй).
Ограничения:
Функция должна иметь вид

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$
$$a \geqslant 1 \qquad b > 1 \qquad f(n) > 0$$

Обозначим $c_{crit} = \log_b a$. Тогда возможны три случая:

1. $f(n) \in O(n^c), c < c_{crit} \Longrightarrow T(n) \in \Theta(n^{c_{crit}})$
2. $f(n) \in \Theta(n^{c_{crit}} \log^k n)$. В зависимости от k есть три подслучая:
   (a) $k > -1 \Longrightarrow T(n) \in \Theta(n^{c_{crit}} \log^{k+1} n)$
   (b) $k = -1 \Longrightarrow T(n) \in \Theta(n^{c_{crit}} \log\log n)$
   (c) $k < -1 \Longrightarrow T(n) \in \Theta(n^{c_{crit}})$
3. $f(n) \in \Omega(n^c), c > c_{crit} \Longrightarrow T(n) \in \Theta(f(n))$

Пример:

$$T(n) = 2 \cdot T(n/2) + n \log n$$
$$c_{crit} = \log_2 2 = 1$$
$$f(n) \in \Theta(n^{c_{crit}} \log^1 n) \Longrightarrow \text{2ой случай}$$
$$k = 1 \Longrightarrow \text{1ый подслучай}$$
$$T(n) \in \Theta(n \log^2 n)$$

4.4. Метод Акра-Бацци.

Обобщение Мастер теоремы для построение асимптотических оценок рекуррентных соотношений следующего вида

$$T(n) = f(n) + \sum_{i=1}^{k} a_i \cdot T(b_i n + h_i(n))$$
$$a_i > 0 \qquad 0 < b_i < 1 \qquad f(n) \in O(n^c) \qquad h_i(n) \in \Theta\left(\frac{n}{\log^2 n}\right)$$

Для получение асимптотической оценки найдем такое p, что будет верно равенство:

$$\sum_{i=1}^{k} a_i b_i^p = 1$$

Тогда асимптотическая оценка будет иметь вид

$$T(n) \in \Theta\left(n^p \left(1 + \int_1^n \frac{f(t)}{t^{p+1}} \mathrm{d}t\right)\right)$$

4.5. Производящие функции.

Def 4.5.1. Производящая функция это способ задания бесконечной последовательности с помощью коэффициентов степенного ряда.

Таким образом последовательность $a_0, a_1, \ldots, a_n$ будет представлена в виде порождающей функции следующим образом:

$$G(x) = a_0 x^0 + a_1 x^1 + \ldots + a_n x^n = \sum_{n=0}^{\infty} a_n x^n$$

Замечание 4.5.2. Переменная x в порождающих функциях не является переменной в обычно понимании этого слова, т.е. не имеет смысла подставлять вместо неё какое-либо число и вычислять 'значение' производящей функции.

Замечание 4.5.3. С производящими функциями можно проводить арифметические операции, как и с обычными функциями: складывать, вычитать, умножать, дифференцировать, интегрировать и т.п.

Пример #01:
С помощью производящих функций можно решать рекуррентные соотношения.
Пусть требуется найти замкнутую формулу для рекуррентного соотношения $a_n = 4a_{n-1} - 3a_{n-2}, a_0 = -4, a_1 = -2$

1. Записываем условие в виде системы

$$\begin{cases} a_0 = -4 \\ a_1 = -2 \\ a_n = 4a_{n-1} - 3a_{n-2} \end{cases}$$

2. Умножаем каждой из уравнений на z в степени, соответствующий порядковому номеру члена последовательности

$$\begin{cases} a_0 z^0 = -4z^0 \\ a_1 z^1 = -2z^1 \\ a_n z^n = (4a_{n-1} - 3a_{n-2})z^n \end{cases}$$

3. Складываем полученные равенства

$$a_0 z^0 + a_1 z^1 + \sum_{n=2}^{\infty} a_n z^n = -4 - 2z + \sum_{n=2}^{\infty} (4a_{n-1} - 3a_{n-2}) z^n$$

4. То, что получилось в левой части обозначаем $G(z)$. Это порождающая функция. То, что получилось справа разбиваем на отдельные суммы

$$G(z) = -4 - 2z + \sum_{n=2}^{\infty} 4a_{n-1} z^n - \sum_{n=2}^{\infty} 3a_{n-2} z^n$$

5. Далее задача заключается в том, чтобы выразить полученные суммы через производящую функцию. ДЛя этого из каждой суммы вынесем число, а также z в такой степени, чтобы оставшаяся степень совпадала с порядковым номером члена последовательности

$$G(z) = -4 - 2z + 4z \sum_{n=2}^{\infty} a_{n-1} z^{n-1} - 3z^2 \sum_{n=2}^{\infty} a_{n-2} z^{n-2}$$

6. Далее поменяем пределы в полученных суммах

$$G(z) = -4 - 2z + 4z \sum_{n=1}^{\infty} a_n z^n - 3z^2 \sum_{n=0}^{\infty} a_n z^n$$

7. Заметим, что левая сумма это $G(z)$ без первого слагаемого, а правая сумма в точности равна $G(z)$.

$$G(z) = -4 - 2z + 4z(G(z) + 4) - 3z^2 G(z)$$

8. Выразим $G(z)$ в явном виде

$$G(z)(1 - 4z + 3z^2) = -4 + 14z$$
$$G(z) = \frac{14z - 4}{3z^2 - 4z + 1}$$

9. Далее нужно разложить эту дробь на простейшие. Это чисто математическая задача, которую можно решить (например) методом неопределенных коэффициентов либо методом сокрытия Хэвисайда.

$$G(z) = \frac{-5}{1 - z} + \frac{1}{1 - 3z}$$

10. После этого нужно 'свернуть' полученные дроби используя формулу

$$\boxed{\frac{1}{(1-\alpha z)^{k+1}} = \sum_{n=0}^{\infty} \alpha^n \binom{n+k}{k} z^n}$$

Числители дробей можно вынести за скобку, поэтому получаем

$$G(z) = -5 \cdot \sum_{n=0}^{\infty} z^n + \sum_{n=0}^{\infty} 3^n z^n$$

11. По определению производящей функции n-ый член последовательности это коэффициент перед z^n. Смотрим на получившиеся и собираем коэффициенты

$$a_n = -5 \cdot 1 + 3^n$$

Ответ: $a_n = 3^n - 5$
Пример #02:
С помощью производящих функций можно находить количество ограниченных композиций, т.е. число целочисленных решений уравнений вида

$$x_1 + \ldots + x_n = n$$
$$l_i \leqslant x_i \leqslant h_i$$

Для этого нужно для каждого x_i составить производящую функцию вида

$$G_i(z) = z^{l_i} + \ldots + z^{h_i}$$

Полученные функции надо перемножить. Ответом будет являются коэффициент перед z^n. Рассмотрим на примере следующей задачи:
Сколько существует способов разделить 25 одинаковых печенек на четверых, при условии, что каждый должен получить как минимум 3, но не более 7ми печенек?
По описанным выше шагам получим четыре одинаковых порождающих функции:

$$G_i(z) = z^3 + z^4 + z^5 + z^6 + z^7$$

Значит ответом будет являться

$$[z^{25}]\Big(z^3 + z^4 + z^5 + z^6 + z^7\Big)^4$$

Ответ: 20

4.6. Операторы и аннигиляторы.

Def 4.6.1. Оператор это метафункция, которая продуцирует новую функцию.

Свойства операторов:

- Сумма $(f+g)(n) := f(n) + g(n)$
- Растяжение $(\lambda f)(n) := \lambda f(n)$
- Сдвиг $(Ef)(n) = f(n+1)$
- k-сдвиг $E^k f(n) = f(n+k)$
- Композиция $(X \pm Y)f = Xf \pm Yf$, $XY(f) = X(Yf) = Y(Xf)$
- Дистрибутивность $X(f+g) = Xf + Xg$

Операторы можно рассматривать как полиномы относительно E

Def 4.6.2. Аннигилятор это оператор, который при применении к функции f превращает её в константный ноль.

Каждый аннигилятор аннигилирует какой-то класс функций и ничего больше. Для каждой *хорошей* (полином, экспонента) функции можно найти **уникальный** аннигилятор.
Свойства аннигиляторов:

- Если X аннигилирует f, то X также аннигилирует Ef.
- Если X аннигилирует f и Y аннигилирует g, то XY аннигилирует $f \pm g$.

Аннигилятор	Класс функций	Примечание
$E-1$	c_1	
$E-a$	$c_1 a^n$	
$(E-a)(E-b)$	$c_1 a^n + c_2 b^n$	$a \neq b$
$(E-1)^2$	$c_1 n + c_2$	
$(E-a)^{d+1}$	$a^n \sum_{i=0}^{d} c_i n^i$	Полином степени d, умноженный на экспоненту

Таблица аннигиляторов

С помощью аннигиляторов можно решать рекуррентные соотношения
Пример:

1. Пусть требуется найти общую формулу для следующего рекуррентного соотношения

$$R_n = 3R_{n-2} - 2R_{n-3} + 2^n + 1$$

2. Перепишем, прибавив 3 к каждому индексу n

$$R_{n+3} = 3R_{n+1} - 2R_n + 2^{n+3} + 1$$

3. Переведем полученное равенство в операторную форму, для этого заменим R_{n+i} на E^i

$$(E^3 - 3E + 2)R = 2^{n+3} + 1$$

4. Таким образом $E^3 - 3E + 2$ аннигилирует левую часть, а $(E-2)(E-1)$ — правую (для того, чтобы это понять, смотрим в табличку и пользуется свойствами аннигиляторов).

5. Разложим полученную композицию аннигиляторов на множители

$$(E^3 - 3E + 2)(E-2)(E-1)$$
$$(E-2)(E+2)(E-1)^3$$

6. Для каждого из полученных аннигиляторов запишем общее решение

$$(E-2) \longrightarrow c_1 2^n$$
$$(E+2) \longrightarrow c_2(-2)^n$$
$$(E-1)^3 \longrightarrow c_3 n^2 + c_4 n + c_5$$

7. Общим решением исходного рекуррентного соотношения будет композиция общих решений для каждого из аннигиляторов:

$$c_1 2^n + c_2(-2)^n + c_3 n^2 + c_4 n + c_5$$