Дискретная математика. Теормин — II

Содержание

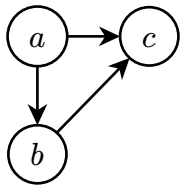
1. Графы

Для начала, стоит дать определение графа. Есть два подхода:

1. Абстрактный. В нем граф — кортеж $G = (V, E, F)$, где:
   - V — конечное множество вершин;
   - E — конечное множество ребер;
   - F — множество функций, задающих структуру графа.
2. Структурный. В нем граф — кортеж $G = (V, E)$, где:
   - V — множество вершин;
   - E — множество ребер.

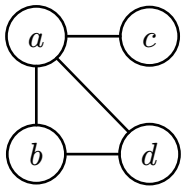
1.1. Directed and undirected graphs

Направленный граф — граф, ребра которого имеют четко отличимые начало и конец ребра.



- В абстрактном подходе, направление задается двумя функциями:
  - $\text{begin} : E \to V$;
  - $\text{end} : E \to V$.
- В структурном подходе, $E \subseteq V \times V$ (то есть ребро просто описывается парой вершин $\langle \text{from}, \text{to} \rangle$).

В *ненаправленном* графе ребро описывается просто двумя равноправными концами.



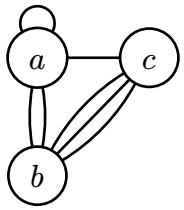
- В абстрактом подходе, ребро описывается функцией $\text{ends} : E \to \binom{V}{2}$
- В структурном подходе, $E \subseteq \binom{V}{2}$.

P.S. $\binom{V}{2} = \{\{u, v\} \mid u, v \in V, u \neq v\}$, то есть просто множество из двух вершин.

1.2. Simple graphs and pseudographs

Простой граф — граф без петель и кратных ребер.

Псевдограф — граф, допускающий петли и кратные ребра.



1.3. Multiedges and multigraphs

- *Кратные ребра* — ребра, соединяющие одну и ту же пару вершин.

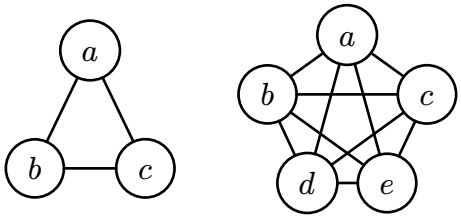
- *Мультиграф* — граф, допускающий кратные ребра, но не допускающий петли.

1.4. Null, empty, singleton graphs

- *Нуль-граф* — граф без вершин.
- *Пустой граф* — граф без ребер.
- *Синглтон-граф* (*тривиальный граф*) — граф с одной вершиной.

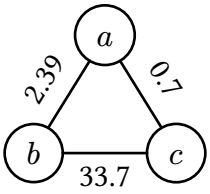
1.5. Complete graph

Полный граф K_n — граф с n вершинами, где каждая пара вершин соединена ребром.



1.6. Weighted graph

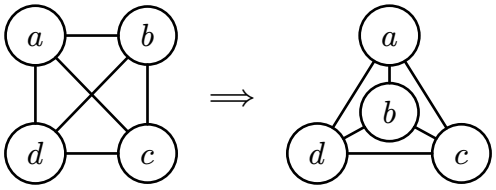
Взвешенный граф — граф, где каждому ребру сопоставлен *вес* — некоторое число.



- В абстрактном подходе, в F добавляется функция $\text{weight} : E \to \mathbb{R}$.
- В структурном подходе, $E = V \times V \times \mathbb{R}$

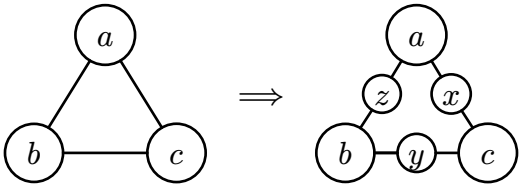
1.7. Planar graphs

Планарный граф — граф, который можно нарисовать на плоскости без пересечений ребер. На примере K_4:



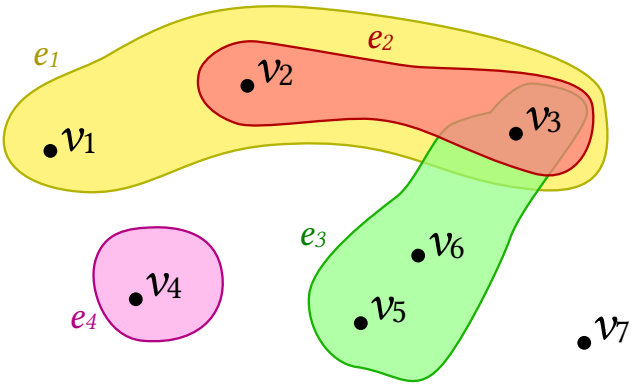
Теорема (Куратовского): граф планарный тогда и только тогда, когда не содержит подграфов-подразделений K_5 или $K_{3,3}$.

Подразделение графа G получается в результате замены ребер путями, то есть в ребро «вставляется» вершина.



1.8. Hypergraphs

Гиперграф — граф, ребра которого соединяют не пару, а произвольное подмножество вершин. Описывается функцией вида incidence : $E \to 2^V$. Либо можно описать с помощью матрицы инцидентности.



1.9. Subgraph, spanning subgraph

$H = (V', E')$ называется *подграфом* графа $G = (V, E)$, если:

$$V' \subseteq V \quad \wedge \quad E' \subseteq E$$

Обозначается как $H \subseteq G$.

Охватывающий (*spanning*) подграф содержит все вершины: $V' = V$

1.10. Induced subgraph

Индуцированный подграф $G[S]$ над подмножеством вершин $S \subseteq V$ — подграф, содержащий все ребра между вершинами из S.

1.11. Adjacency relation

Пара вершин u, v называют *смежными*, если существует ребро $\{u, v\} \in E$.

1.12. Incidence relation

Ребро $e \in E$ называется *инцидентным* вершине $v \in V$, если v является концом e.

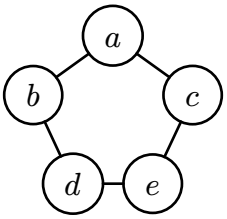
1.13. Vertex degree

Степень вершины v — число инцидентных ему ребер. Обозначается как $\deg(v)$.

- $\delta(G) = \min_{v \in V} \deg(v)$ — *минимальная степень графа* G.
- $\Delta(G) = \max_{v \in V} \deg(v)$ — *максимальная степень графа* G.

1.14. Regular graph

r-регулярный граф — граф, у которого каждая вершина имеет степень r.



2-регулярный граф

1.15. Handshaking lemma

Сумма всех степеней равна удвоенному количеству ребер:

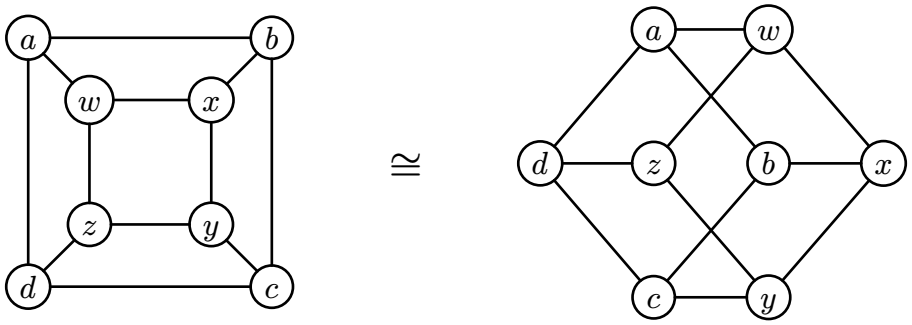
$$\sum_{v \in V} \deg(v) = 2|E|$$

1.16. Graph isomorphism

Два графа $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ изоморфны, если существует биекция между вершинами $\varphi : V_1 \to V_2$, которая сохраняет смежность:

$$\{u, v\} \in E_1 \quad \iff \quad \{\varphi(u), \varphi(v)\} \in E_2$$

Простыми словами, пара графов изоморфна, если у них одинаковая структура, но разные лейблы вершин.



1.17. Walks, paths, trails, cycles

- *Walk* графа — чередующаяся последовательность ребер и вершин:

  $$(v_0, e_1, v_1, e_2, ..., e_k, v_k)$$

  Начинается и заканчивается с вершины. Ребро соединяет левую и правую вершины в последовательности.
- *Trail* — walk без повторяющихся ребер.
- *Path* — walk без повторяющихся вершин.

Циклы:

- *Circuit* — замкнутый trail (то есть вершины в начале и конце последовательности совпадают).
- *Cycle* — замкнутый path.

Длина пути (walk'а ☒) — количество ребер в нем.

1.18. Eulerian path, cycle, graph

- Эйлеров *путь* — путь (правильней сказать trail, ну просто нормального перевода на русский нет), посещающий *каждое* ребро ровно один раз.
- Эйлеров *цикл* — замкнутый эйлеров путь.
- Граф *эйлеров*, если есть эйлеров цикл.

1.19. Eccentricity of a vertex

Расстояние между вершинами u, v — длина кратчайшего пути между ними. Обозначается как $\text{dist}(u, v)$.

Эксцентриситет вершины v — максимально возможное расстояние от вершины v:

$$\operatorname{ecc}(v) = \max_{v \in V} \operatorname{dist}(v, u)$$

1.20. Radius and diameter of a graph

- *Радиус* графа — минимальнй эксцентриситет:

$$\operatorname{rad}(G) = \min_{v \in V} \operatorname{ecc}(v)$$

- Диаметр графа — максимальный эксцентриситет:

$$\operatorname{diam}(G) = \max_{v \in V} \operatorname{ecc}(v)$$

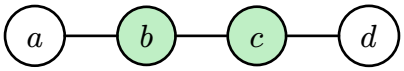
Есть теорема о том, что для всякого связного графа:

$$\operatorname{rad}(G) \leq \operatorname{diam}(G) \leq 2 \operatorname{rad}(G)$$

1.21. Center of a graph

Центр графа — множество вершин, у которых эксцентриситет равен радиусу:

$$\operatorname{center}(G) = \{v \in V \mid \operatorname{ecc}(v) = \operatorname{rad}(G)\}$$



1.22. Centroid of a tree

Центроида дерева — вершина, при удалении которой, каждое дерево из полученного леса будет иметь не более, чем $\frac{|V|}{2}$ вершин.[1] При этом, в любом дереве существует хотя бы один центроид, но не больше двух.

1.23. Clique

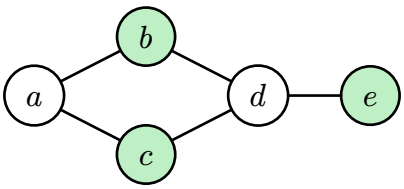
Клика — полный подграф.

- Клика *максимальная*, если в нее нельзя добавить вершин.
- Клика *наибольшая*, если число вершин в ней максимально возможный.
- *Кликовое число* $\omega(G)$ — число вершин в наибольшей клике.

1.24. Independent set

Независимое множество $S \subseteq V$ — множество попарно несмежных вершин.

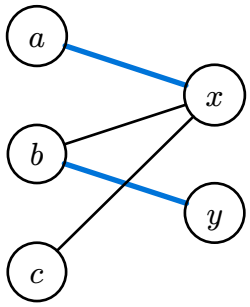
Он же stable set.



1.25. Matching, perfect matching

Паросочетание $M \subseteq E$ — множество попарно несмежных ребер (т.е. между любой парой ребер нет общей вершины).

[1] https://usaco.guide/plat/centroid?lang=cpp#centroids



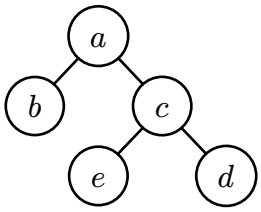
- Паросочетание *максимальное*, если в нее нельзя добавить ребер.
- Паросочетание *наибольшее*, если оно максимально возможного размера.
- *Совершенное паросочетание* покрывает все вершины.

1.26. Vertex cover, edge cover

- *Вершинное покрытие* $R \subseteq V$ — такое множество вершин, что все ребра графа инцидентны какой-либо вершине из R.
- *Реберное покрытие* $F \subseteq E$ — такое множество ребер, что все вершины графа смежны с любым ребром из F.

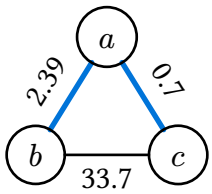
1.27. Tree, forest

- *Дерево* — связный ациклический граф.
- *Лес* — ациклический граф (система непересекающихся деревьев).



1.28. Minimum spanning tree

Минимальное остовное дерево связного графа — подграф-дерево, содержащий все вершины, сумма весов ребер которой минимальна.

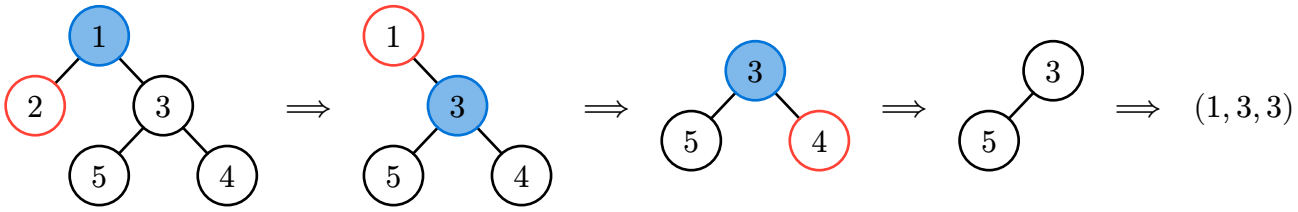


1.29. Prüfer code

Код Прюфера — однозначная кодировка дерева в виде последовательности из $n - 2$ вершин.

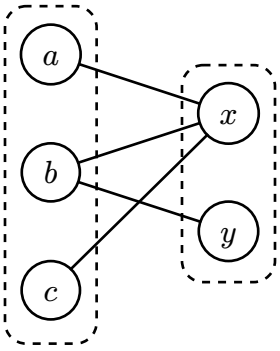
Алгоритм кодирования:

1. Удаляем лист с наименьшим номером;
2. Записываем в последовательность номер вершины, смежной с удаленным листом;
3. Повторяем процесс, пока не останется всего 2 вершины.



1.30. Bipartite graph

Граф $G = (V, E)$ — *двудольный*, если его вершины можно разбить на два непересекающихся множества, чтобы ребра соединяли вершины из этих двух множеств.



Теорема: граф двудольный тогда и только тогда, когда не содержит циклов нечетной длины.

1.31. Connectivity in undirected graphs

- Две вершины в ненаправленном графе называют *связными*, если существует путь между ними.
- Граф называют *связным*, если все вершины попарно связные.
- *Компонента связности* — максимально связный подграф.

1.32. Strong and weak connectivity in directed graphs

- Направленный граф *слабо связный*, если замена ребер на ненаправленные приводит к связному графу.
- Направленный граф *полусвязный*, если для любой пары вершин $u, v \in V$ есть путь $u \to v$ *или* $v \to u$.
- Направленный граф *сильно связный*, если между любыми двумя вершинами есть путь в обе стороны.

Компонента слабой/сильной/полу- связности — максимально соответственно связный подграф.

1.33. Condensation of a directed graph

Конденсация направленного графа — сжатие компонент сильной связности в одну вершину.

1.34. Vertex connectivity

Вершинная связность $\kappa(G)$ — минимальное количество вершин, которое надо удалить, чтобы G перестал быть связным, либо состоял из одной вершины.

1.35. Edge connectivity

Реберная связность $\lambda(G)$ — минимальное количество ребер, которое надо удалить, чтобы G перестал быть связным.

1.36. k-connected graph

- Граф *k-вершинно-связный* (или же просто k-связный), если $\kappa(G) \geq k$.

  Другими словами, у графа больше, чем k вершин, и при удалении менее, чем k вершин сохранится связность.

- Граф *k-реберно-связный*, если $\lambda(G) \geq k$.

  Другими словами, при удалении менее, чем k ребер сохранится связность.

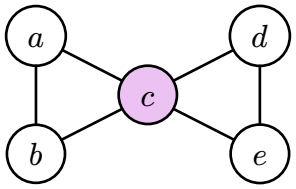
1.37. Biconnectivity

Граф *двусвязный*, если при удалении любой одной вершины сохранится связность.

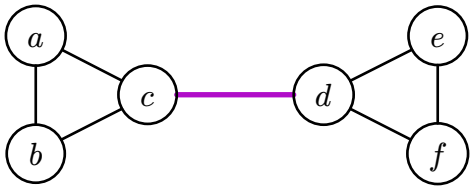
Другими словами, граф *двусвязный*, если он не содержит точек сочленения.

1.38. Articulation point, bridge

Точка сочленения (*шарнир*) — вершина, при удалении которой увеличится количество компонент связности.



Мост — ребро, при удалении которой увеличится количество компонент связности.



1.39. Blocks, block-cut tree

- *Блок* — максимальный связный подграф без точек сочленения. Другими словами, это или двусвязный подграф, или мост, или изолированная вершина.
- *Block-cut дерево* — двудольное дерево, в котором одна доля — блоки, сжатые в одну вершину, а вторая доля — точки сочленения, соединяющие блоки.

2. Чето там выделенное другим цветом

2.1. Euler's theorem for graphs

- Связный граф имеет эйлеров *цикл* тогда и только тогда, когда каждая вершина имеет четную степень.
- Связный граф имеет эйлеров *путь* тогда и только тогда, когда ровно две вершины имеют нечетную степень.

2.2. Hamiltonian path, cycle, graph

- *Гамильтонов путь* — путь, в котором каждая вершина графа посещена ровно один раз.
- *Гамильтонов цикл* — замкнутый гамильтонов путь.
- Граф *гамильтонов*, если содержит гамильтонов цикл.

Проверка графа на гамильтоновость — NP-полная задача (в то время как проверка на эйлеровость решается за $\mathcal{O}(n)$).

2.3. Ore's theorem

Если G имеет $n \geq 3$ вершин и для любой пары несмежных вершин $u, v \in V$ выполняется

$$\deg(u) + \deg(v) \geq n,$$

то граф гамильтонов.

2.4. Dirac's theorem

Если G имеет $n \geq 3$ вершин и $\delta(G) \geq \frac{n}{2}$, то граф гамильтонов.

2.5. Theorem on the balance of regular bipartite graphs

Если двудольный граф $G = (X, Y, E)$ регулярный, то $|X| = |Y|$.[2]

2.6. Theorem on the existence of a perfect matching in a regular bipartite graph

В регулярном двудольном графе существует совершенное паросочетание.

2.7. Hall's theorem (on the existence of an X-perfect matching in a bipartite graph)

Соседство $N(S)$ двудольного графа $G = (X, Y, E)$ — множество всех достижимых вершин из подмножества одной доли ($S \subseteq X$) в другую ($N(S) \subseteq Y$):

$$N(S) = \{y \in Y \mid \exists x \in S.\ \{x, y\} \in E\}$$

Теорема Холла: в двудольном графе существует совершенное паросочетание тогда и только тогда, когда для любого $S \subseteq X$ выполняется $|N(S)| \geq |S|$.

2.8. Whitney's theorem

- *Whitney's theorem*: Граф G, имеющий $n \geq 3$ вершин, является двусвязным тогда и только тогда, когда любые две его вершины лежат на каком-либо общем цикле.
- *Whitney's inequality*: Для любого графа G:

$$\kappa(G) \leq \lambda(G) \leq \delta(G)$$

2.9. Menger's theorem

- *u-v разделитель* (или вершинный разрез) — множество вершин $S \subseteq V \setminus \{u, v\}$, при удалении которых u и v окажутся в разных компонентах связности.
- Два различных пути между u и v называются *вершинно внутренне непересекающимися*, если в этих путях нет общих вершин, кроме u и v.

[2] https://www.youtube.com/watch?v=73u0OQCR2rs

Теорема Менгера для вершин: пусть u, v — пара несмежных вершин в G. Тогда *максимальное* количество вершинно внутренне непересекающихся путей u-v равно *минимальной* мощности u-v разделителя S.

Аналогичная теорема есть и для ребер.

- *u-v реберный разрез* — множество ребер $F \subseteq E$, при удалении которых u и v окажутся в разных компонентах связности.
- Два различных пути между u и v называются *реберно внутренне непересекающимися*, если в этих путях нет общих ребер.

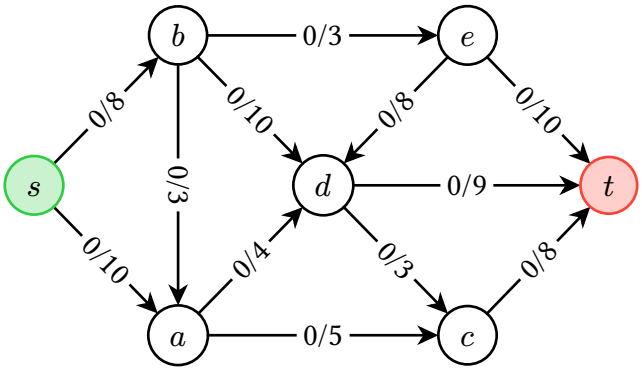
Теорема Менгера для вершин: пусть u, v — пара несмежных вершин в G. Тогда *максимальное* количество реберно внутренне непересекающихся путей u-v равно *минимальной* мощности u-v реберного разреза F.

Данная теорема имеет большую роль в транспортных сетях (в частности, в s-t разрезе).

3. Потоки

3.1. Flow network

Транспортная сеть — кортеж (V, E, s, t, c), где:
- $G = (V, E)$ — направленный граф;
- $s, t \in V$ — *исток* и *сток* ($s \neq t$);
- $c : E \to R_{\geq 0}$ — функция *емкости* ребра, то есть сколько потока может течь по этому ребру.



3.2. Flow value

Поток сети N — функция $f : E \to \mathbb{R}_{\geq 0}$, удовлетворяющая условиям:
1. *Ограничение по емкости*: $0 \leq f(e) \leq c(e)$. Простыми словами, поток не превышает емкость.
2. *Закон сохранения*: для любой вершины $v \in V \setminus \{s, t\}$:

$$\underbrace{\sum_{e \in \text{ in}(v)} f(e)}_{\text{сколько втекло}} = \underbrace{\sum_{e \in \text{ out}(v)} f(e)}_{\text{столько вытекло}}$$

Величина потока $|f|$ — поток через исток:

$$|f| = \underbrace{\sum_{e \in \text{ out}(s)} f(e)}_{\text{сколько вытекло из } s} - \underbrace{\sum_{e \in \text{ in}(s)} f(e)}_{\substack{\text{сколько втекло в } s \\ (=0, \text{ если входных} \\ \text{ребер не было})}}$$

3.3. Residual network

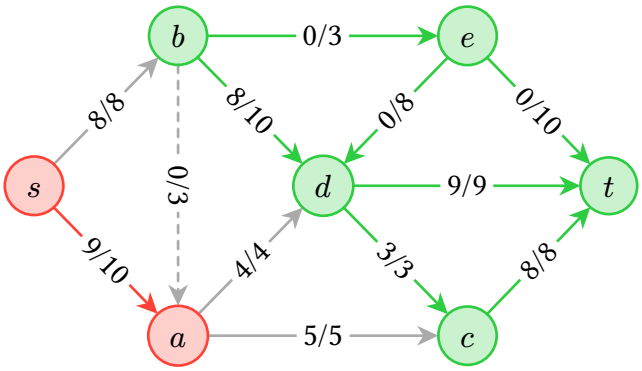
Остаточная сеть N_f — граф, у которой есть два типа ребер:

- Прямое ребро $u \to v$. Его вес равен $c(u, v) - f(u, v)$, то есть сколько еще можно пустить потока.
- Обратное ребро $v \to u$. Его вес равен $f(u, v)$, то есть сколько потока было пущено по ребру. Предназначен для отката потока.

Дополняющий путь P — путь из s в t по остаточной сети. *Узкое горлышко* (боттлнек) дополняющего пути равен минимальному весу ребра из P.

3.4. Flow network cut

s-t разрез сети N — разбиение вершин сети на два таких множества (A, B), что $s \in A$ и $t \in B$.



Емкость s-t разреза (A, B) равен сумме емкостей ребер $A \to B$:

$$c(A, B) := \sum_{\substack{(u,v) \in E, \\ u \in A,\ v \in B}} c(u, v)$$

Поток s-t разреза (A, B) равен сумме потоков $A \to B$ минус сумма потоков $B \to A$:

$$f(A, B) := \sum_{\substack{(u,v) \in E, \\ u \in A,\ v \in B}} f(u, v) - \sum_{\substack{(v,u) \in E, \\ v \in B,\ u \in A}} f(v, u)$$

Теорема: $f(A, B) = |f|$.

3.5. Max-flow Min-cut theorem

В любой транспортной сети N, следующие условия эквивалентны:

1. $|f|$ максимален.
2. Не существует дополняющих путей в остаточной сети N_f
3. Существует s-t разрез (A, B), у которого $|f| = c(A, B)$

Более того, когда эти условия выполняются, разрез (A, B) *минимален*, то есть имеет наименьшую емкость.

4. Формальные языки

Немного полезных определений:

- *Алфавит* Σ — конечное непустое множество символов. Пример: $\Sigma = \{a, b, c\}$.
- *Слово* — конечная последовательность символов из Σ. Особый случай — пустое слово ε.
- *Звезда Клини* — множество всех конечных слов над алфавитом: $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k$.
- *Формальный язык* $L \subseteq \Sigma^*$ — множество конечных слов над конечным алфавитом.

Над формальными языками доступны различные операции:

- Мощность: $|L|$
- Дополнение: $\overline{L} = \{w \mid w \notin L\} = \Sigma^* \setminus L$
- Объединение: $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$
- Конкатенация: $L_1 \cdot L_2 = \{ab \mid a \in L_1, b \in L_2\}$
  - $L^k = L \cdot L \cdot \ldots \cdot L = \{w_1 w_2 \ldots w_k \mid w_i \in L\}$, $L^0 = \{\varepsilon\}$
- Звезда Клини: $L^* = \bigcup_{k=0}^{\infty} L^k$

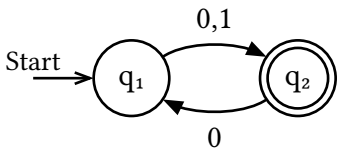
4.1. Deterministic Finite Automaton (DFA)

Детерминированный конечный автомат (ДКА) — 5-кортеж $(Q, \Sigma, \delta, q_0, F)$, в котором:

- Q — конечное множество *состояний*;
- Σ — алфавит;
- $\delta : Q \times \Sigma \to Q$ — *функция перехода*;
- $q_0 \in Q$ — *начальное состояние*;
- $F \subseteq Q$ — множество *принимающих состояний*.

ДКА — машина, которая принимает очередной символ строки и совершает переход из текущего состояния в новое, в зависимости от входного символа. Автомат может *принимать* или *отвергать* строку, в зависимости от состояния, на котором он остановился.

	0	1
$\to q_1$	q_2	q_2
q_2^*	q_1	—



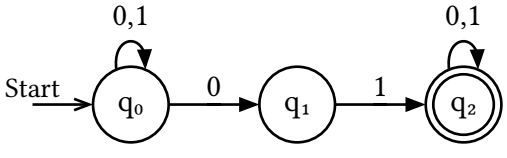
Детерминированный означает, что в любом состоянии, для любого символа есть *ровно один* переход.

4.2. Non-deterministic Finite Automaton (NFA)

Недетерминированный конечный автомат (НКА) — 5-кортеж $(Q, \Sigma, \delta, q_0, F)$, в котором:

- Q — конечное множество *состояний*;
- Σ — алфавит;
- $\delta : Q \times \Sigma \to 2^Q$ — *функция перехода*;
- $q_0 \in Q$ — *начальное состояние*;
- $F \subseteq Q$ — множество *принимающих состояний*.

	0	1
$\to q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2^*	$\{q_2\}$	$\{q_2\}$



Главное отличие от ДКА в том, что для входного символа может быть несколько переходов в разные состояния.

4.3. Regular language

Класс регулярных языков REG (то есть множество всех возможных *регулярных* формальных языков над Σ) определяется индуктивно:

- $\mathrm{Reg}_0 = \{\emptyset, \{\varepsilon\}\} \cup \{\{a\} \mid a \in \Sigma\}$ — *пустой* и *синглтон-* языки;
- $\mathrm{Reg}_{i+1} = \mathrm{Reg}_i \cup \{A \cup B \mid A, B \in \mathrm{Reg}_i\} \cup \{AB \mid A, B \in \mathrm{Reg}_i\} \cup \{A^* \mid A \in \mathrm{Reg}_i\}$

  То есть мы берем все языки из предыдущего *поколения*, объединяем/конкатенируем/применяем звезду Клини над ними, и добавляем их в новое поколение (вместе с предыдущим поколением, конечно).

Итак, $\mathrm{REG} = \bigcup_{k=0}^{\infty} \mathrm{Reg}_k$.

4.4. Regular expression

Регулярное выражение — нотация, описывающая регулярные языки (и операции над ними):

- ε — язык, состоящий только из пустого слова
- `a` — синглтон-язык из символа `a`
- $\alpha\beta$ — конкатенация двух языков.
- $\alpha \mid \beta$ — объединение двух языков.
- α^* — звезда Клини над языком.

Пример регулярного выражения над $\Sigma = \{0, 1\}$: `0*(1|0)`. Описывает все слова, которые начинаются с произвольного количества нулей и заканчиваются либо единицей, либо нулем. Например: `001`, `00000`, `1`.

Этого достаточно, чтобы описать какие-нибудь другие операции, например:

$$\alpha^+ = \alpha\alpha^* \qquad \alpha? = \alpha \mid \varepsilon \qquad \alpha\{n\} = \underbrace{\alpha\alpha\ldots\alpha}_{n \text{ раз}}$$

Если для вас было не очень понятно определение REG, то по сути это множество всех возможных регулярных выражений над каким-то заданным алфавитом.

4.5. Kleene's theorem

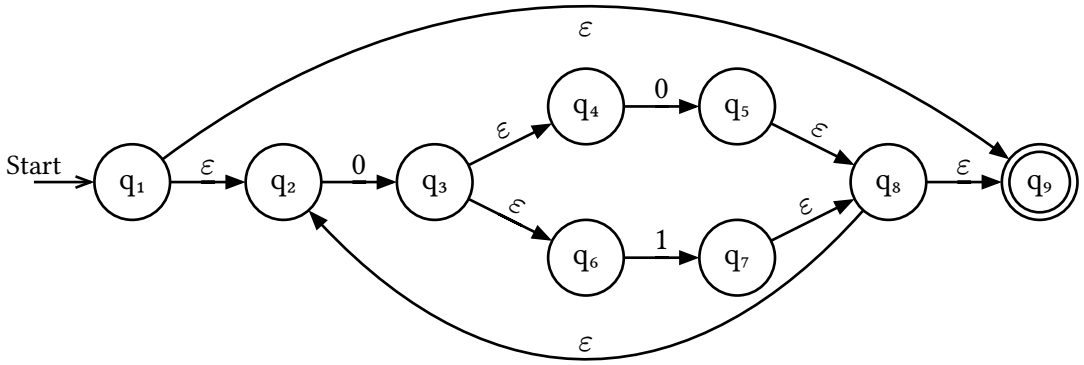
$\mathrm{REG} = \mathrm{AUT}$, где AUT — класс языков, которые *распознаются* с помощью ДКА. Язык *распознается* ДКА, если каждое слово из языка принимается автоматом.

По сути это значит, что язык регулярный тогда и только тогда, когда его можно распознать с помощью ДКА.

4.6. Thompson's construction

Построение Томпсона — алгоритм построения ε-НКА из регулярного выражения.

Суть в том, регулярное выражение преобразуется в синтаксическое дерево, потом делаются автоматы для листьев, и потом из них делаются более крупные автоматы, в зависимости от операции объединения/конкатенации/звезды, и снизу вверх так по дереву.



Пример для регулярного выражения `(0(0|1))*`

4.7. Kleene's algorithm

Алгортм Клини — алгоритм для построения регулярного выражения из ДКА. По своей сути схож с Флойдом-Уоршеллом (но придуман раньше него!).

Пусть R_{ij}^k — регулярное выражение, описываемое переходами из q_i в q_j транзитом через промежуточные состояния $q_1, q_2, ..., q_k$. При $k = 0$ — регулярка для прямого пути из q_i в и q_j. Логично, что базовая регулярка для перехода из состояния в самого себя — ε или петля (если есть такой переход).

Построение регулярки производится по итеративно:

$$R_{ij}^0 = \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\} & i = j \end{cases}$$

$$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1} \left(R_{kk}^{k-1}\right)^* R_{kj}^{k-1}$$

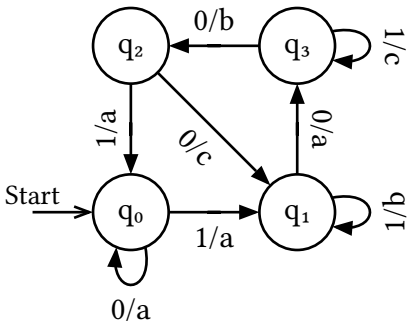
4.8. Mealy machine

Автомат Мили — 6-кортеж $T = (Q, \Sigma, \mathcal{O}, \delta, q_0, \lambda)$, где:

- Q конечное множество состояний;
- Σ алфавит (конечное множество входных символов);
- $\mathcal{O}$ — *выходной алфавит* (конечное множество выходных символов);
- $\delta : Q \times \Sigma \to Q$ — функция перехода;
- $q_o \subseteq Q$ — множество начальных состояний;
- $\lambda : Q \times \Sigma \to \mathcal{O}$ — функция вывода.

По сути, это автомат, в котором, при *переходе* между состояниями, выдается какой-то символ из $\mathcal{O}$.

	0	$\mathcal{O}$	1	$\mathcal{O}$
$\to q_0$	q_0	a	q_1	a
q_1	q_3	a	q_1	b
q_2	q_1	c	q_0	a
q_3	q_2	b	q_3	c

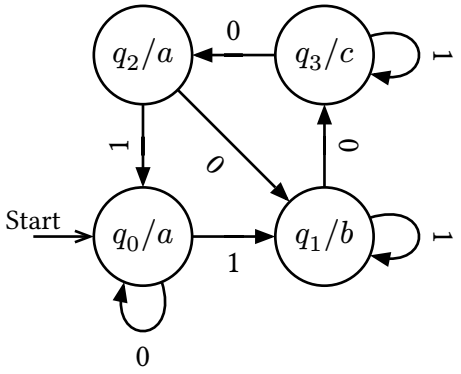


4.9. Moore machine

Автомат Мура — 6-кортеж $T = (Q, \Sigma, \mathcal{O}, \delta, q_0, G)$, где:

- Q конечное множество состояний;
- Σ алфавит (конечное множество входных символов);
- $\mathcal{O}$ — *выходной алфавит* (конечное множество выходных символов);
- $\delta : Q \times \Sigma \to Q$ — функция перехода;
- $q_o \subseteq Q$ — множество начальных состояний;
- $G : Q \to \mathcal{O}$ — функция вывода.

	0	1	$\mathcal{O}$
$\to q_0$	q_0	q_1	a
q_1	q_3	q_1	b
q_2	q_1	q_0	a
q_3	q_2	q_3	c



Отличие от автомата Мили в том, что выходные символы находятся не на переходах, а на состояниях.

4.10. Pumping lemma for regular languages

Полная лемма о накачке: пусть L — регулярный язык. Значит, существует такое $n \in \mathbb{N}, n > 0$, что для любого слова $w \in L$, у которого $|w| \geq n$, существуют строки x, y и z такие, что:

- $w = xyz$;
- $y \neq \varepsilon$;
- $|xy| \leq n$;
- $\forall i \in \mathbb{N} : xy^i z \in L$.

Слабая лемма о накачке отличается тем, что не требует $|xy| \leq n$.

4.11. Non-regular languages

Ну а что тут сказать? Нерегулярные языки — это те, которые... не регулярные. Например, контекстно-свободные, контекстно-зависимые, рекурсивно перечислимые. Для их распознавания нужны более крутые автоматы: pushdown автомат, машина Тьюринга.

Ну например можно опровергнуть предположение о регулярности какого-либо языка, если нарушается лемма о накачке (то есть это только *необходимое* условие регулярности).

А есть еще теорема Майхилла-Нероуда, которая определяет необходимое и достаточное условие регулярности.

4.12. Chomsky hierarchy

Иерархия Хомского — классификация формальных языков по уровню условной сложности:

- Тип 3 — регулярные языки (REG). Самые простые. Для распознавания достаточно конечного автомата.
- Тип 2 — контекстно-свободные языки (CF). Для распознавания достаточно *pushdown-автомата* — автомата, у которой помимо состояний и переходов, есть еще память в виде стека.

- Тип 1 — контекстно-зависимые языки (CS). Для распознавания достаточно машины Тьюринга с ограниченной лентой.
- Тип 0 — рекурсивно-перечислимые языки (RE). Для них нужна полноценная машина Тьюринга.

Иерархия такова:

$$\mathrm{REG} \subset \mathrm{CF} \subset \mathrm{CS} \subset \mathrm{RE}$$

4.13. Closure properties of regular languages

Регулярные языки замкнуты (то есть остаются регулярными) над следующими операциями:

1. Объединение $L_1 \cup L_2$;
2. Пересечение $L_1 \cap L_2$;
3. Дополнение $\overline{L}$;
4. Разница $L_1 \setminus L_2$;
5. Обратное преобразование $L^R = \{w^R \mid w \in L\}$;
6. Звезда Клини L^*;
7. Конкатенация $L_1 \cdot L_2$;
8. Гомоморфизм (т.е. замена символов из Σ на какие-нибудь строки);
9. Обратный гомоморфизм.