

Программа к экзамену по курсу "Алгоритмы и структуры данных".

Содержание

1	Вопросы на неуд.	5
1.1	Понятие графа. Способы хранения графа: список смежности, матрица смежности, список ребер.	5
1.2	Отношение сильной связности.	5
1.3	Обход в глубину DFS. Атрибуты вершин: времена входа и выхода, цвета. Топологическая сортировка. Классификация ребер в обходе DFS.	6
1.4	Отношение реберной двусвязности. Мосты.	8
1.5	Взвешенные графы. Обход в ширину BFS.	9
1.6	Задачи, решаемые алгоритмами Дейкстры и Форда-Беллмана. Время работы.	10
1.7	Постановка задач по поиску минимального остовного дерева. Время ее решения алгоритмом Прима или Крускала.	13
1.8	Система непересекающихся множеств: API, наивная реализация на массиве.	16
1.9	Постановка задачи поиска наименьшего общего предка. Наивное решение.	17
1.10	Понятие сети, потока в сети, разреза в сети. Остаточная сеть. .	20
1.11	Формулировка теоремы о максимальном потоке и минимальном разрезе (Форда-Фалкерсона). Время поиска максимального потока.	22
2	Вопросы на уд.	23
2.1	Лемма о белых путях.	23

2.2	Критерий двудольности графов. Алгоритм поиска разбиения на доли.	24
2.3	Алгоритм Дейкстры.	25
2.4	Алгоритм Форда-Беллмана.	25
2.5	Задача APSP. Алгоритм Флойда-Уоршелла.	25
2.6	Постановка задачи поиска минимального остовного дерева. Лемма о безопасном ребре. Алгоритм Прима.	26
2.7	Постановка задачи поиска наименьшего общего предка. LCA. Наивное решение. Решение с использованием двоичных подъемов.	26
2.8	Алгоритм Куна для поиска максимального паросочетания в двудольном графе. Улучшения алгоритма Куна.	26
2.9	Остаточная сеть. Дополняющий поток. Сложение потоков.	28
2.10	Теорема «о максимальном потоке и минимальном разрезе» (Форда-Фалкерсона). Метод Форда-Фалкерсона. Пример долгой работы, при реализации через dfs. Сведение задачи поиска максимального паросочетания к задаче поиска максимального потока.	28
2.11	Слоистая сеть. Блокирующий поток. Схема Диница. Число итераций в схеме Диница. Жадный поиск блокирующего потока.	28

3 Вопросы на хор. 31

3.1	Отношение сильной связности. Компоненты сильной связности. Алгоритм Косарайю.	31
3.2	Отношение реберной двусвязности. Мосты. Функция t_{up} . Алгоритм поиска мостов.	31
3.3	Вершинная двусвязность. Точки сочленения. Функция t_{up} . Алгоритм поиска точек сочленения.	31

3.4	Взвешенные графы. Обход в ширину BFS. Его модификации: 0-1 BFS, 1-k BFS, 0-k BFS.	32
3.5	Поиск циклов отрицательного веса.	32
3.6	Система непересекающихся множеств. Реализация с использо- ванием леса деревьев. Эвристики. Время работы (б/д).	32
3.7	Постановка задачи поиска минимального остовного дерева. Лемма о безопасном ребре. Алгоритм Крускала.	33
3.8	Алгоритм Борувки.	33
3.9	Постановка задачи поиска наименьшего общего предка LCA. Сведение LCA \iff RMQ.	34
3.10	Двудольность графов. Паросочетания. Увеличивающие цепи и теорема Бержа.	34
3.11	Поток через разрез. Неравенство между величиной произволь- ного потока и пропускной спо- способности произвольного разреза.	35
3.12	Алгоритм Эдмондса-Карпа. Время работы.	35
3.13	Слоистая сеть. Блокирующий поток. Схема Диница. Число ите- раций в схеме Диница. Удаляющий обход.	37
3.14	Алгоритм масштабирования потока.	37
3.15	Теорема о декомпозиции потока.	38
3.16	Биномиальная куча.	40
3.17	Постановка задачи о поиске потока минимальной стоимости. Задача о назначениях и ее сведение к задаче о поиске потока минимальной стоимости.	42

1. Вопросы на неуд.

1.1 Понятие графа. Способы хранения графа: список смежности, матрица смежности, список ребер.

Определение. Ориентированный граф(далее орграф) - это $G = (V, E)$, где V - это множество вершин, а $E \subset V \times V$ - множество ребер.

Определение. Неориентированный граф - это $G = (V, E)$, где V - это множество вершин, а $E = \{\{u, v\} : u, v \in V\}$ - множество ребер, причем $(u, v) \in E \Leftrightarrow (v, u) \in E$.

1.2 Отношение сильной связности.

Определение. Две вершины сильно связны, если между ними есть путь в обе стороны.

Отношение сильной связности является отношением эквивалентности (рефлексивное, симметричное, транзитивное).

Определение. Компоненты сильной связности (КСС) - классы эквивалентности по отношению сильной связности.

Определение. Компонента сильной связности - максимальное подмножество вершин, каждая из которых достижима из любой другой.

Определение. Графом конденсации называют граф, где все компоненты сильной связности сжаты до одной вершины, а ребра между ними получаются как ребра между компонентами.

Алгоритм Косарайю.

1. Запускаем DFS на графе, получить вершины в порядке увеличения времени выхода (почти топологическая сортировка).

2. Транспонируем граф.
3. Запустить на транспонированном графе DFS в порядке уменьшения времени выхода в исходном графе. Каждая найденная компонента является компонентой сильной связности.

Теорема. *Алгоритм Косарайю корректен.*

Доказательство. То есть нам надо доказать, что на шаге 3 каждый запуск посетит одну компоненту сильной связности и только ее. Заметим, что в транспонированном графе компоненты все те же, то есть мы изменили лишь связи между компонентами.

Рассмотрим первый вызов DFS на третьем шаге. Так как это вершина с максимальным временем выхода, то нет ребер в ее компоненту сильной связности (по лемме выше). При этом в транспонированном графе получаем, что из рассматриваемой компоненты нет ребер в другие, а значит DFS посетит только саму компоненту. Ну это именно то, что нам требовалось показать, так как с помощью массива `used` мы отделили всю компоненту.

Рассуждая по индукции, получим, что мы генерируем компоненты сильной связности, при этом в порядке топологической сортировки, так как мы все еще идем по убыванию времени выхода.

□

1.3 Обход в глубину DFS. Атрибуты вершин: времена входа и выхода, цвета. Топологическая сортировка. Классификация ребер в обходе DFS.

Цвета вершин.

- Белый - не посещена.
- Серый - посещена, но не все соседи рассмотрены.

- Черный - посещена, все соседи рассмотрены.

DFS.

1. Красим в серый.
2. Просматриваем соседей.
3. Если сосед белый - вызываем рекурсивно от него.
4. Рассмотрели всех соседей - красим в черный.

Время: $O(|V| + |E|)$

Классификация ребер.

- Ребра деревьев обхода (tree edge) — ребро (u, v) является ребром дерева, если при исследовании ребра была впервые открыта вершина v .
- Обратные ребра (back) — это ребра (u, v) , соединяющие вершину u с ее предком v в дереве поиска в глубину. Петли тоже считаем обратными ребрами.
- Прямые ребра (forward) — это ребра (u, v) , не являющиеся ребрами дерева и соединяющие вершину u с ее потомком v в дереве поиска в глубину.
- Перекрестные ребра (cross) — все остальные ребра, они могут соединять вершины одного и того же дерева поиска в глубину, когда ни одна из вершин не является предком другой, или соединять вершины в разных деревьях поиска в глубину.

Топологическая сортировка.

- *time* — время.

- tin_v — время входа (серый).
- $tout_v$ — время выхода (черный).

1. Используем DFS.
2. При выходе из вершины вносим ее в начало списка.

1.4 Отношение реберной двусвязности. Мосты.

Определение. Деревом обхода DFS называют граф, состоящий из вершин, посещаемых в ходе обхода DFS и следующих ребер:

- Древесное ребро — ребро, по которому DFS переходит напрямую (переходы в белые вершины из серых);
- Обратное ребро — ребро, которое DFS просматривает, но не идет по нему (переходы в серые вершины).

Введем функцию $tup(v)$, определяемую следующим образом:

$$tup(v) = \min \begin{cases} tin(v), \\ \min_u tin(u) \end{cases}$$

Где u — предок v и при этом u достижима по обратному ребру из w — вершины поддерева v .

Определение. Две вершины реберно двусвязны, если между ними есть два реберно непересекающихся пути.

Утверждение. Несложно заметить, что отношение реберной двусвязности является отношением эквивалентности.

Определение. Компонентой реберной двусвязности называют класс эквивалентности по отношению выше.

Определение. Мост — ребро, при удалении которого увеличивается число компонент связности.

Теорема (Критерий моста). *Древесное ребро (t, v) является мостом если и только если $t_{up}(v) \geq t_{in}(v)$.*

Доказательство. $t_{up}(v) \leq t_{in}(v)$ по определению. Осталось рассмотреть случай равенства $t_{up}(v) = t_{in}(v)$. Это равносильно тому, что не найдется вершины u , в которую по обратному ребру можно прыгнуть из поддеревя v выше, чем v (так как иначе это было бы $t_{in}(u) < t_{in}(v)$), что то же самое, что ребро является мостом. \square

1.5 Взвешенные графы. Обход в ширину BFS.

Определение. Взвешенным графом будем называть тройку $G = (V, E, w)$, где V и E уже привычные нам составляющие, а вот $w : E \rightarrow K \subseteq R$ - весовая функция.

Задача 1.1. Найти расстояния от s до всех остальных, если $w : E \rightarrow K = \{1\}$.

Обход в ширину BFS.

1. Заведем очередь. Положим туда s .
2. Извлечем вершину из очереди, пометим как посещенную, добавим смежные.
3. Пока очередь не пуста - повторяем шаг 2 с одним условием: если извлеченная вершина уже посещена, то не добавляем ее в очередь.

0-1 BFS.

Вместо очереди заведем дек. Если $w(from, to) = 0$ — в начало, иначе — в конец.

1-k BFS.

Массив очередей.

Время: $O(k|V| + |E|)$. Память: $O(k|V| + |E|)$.

Оптимизация памяти 1-k BFS.

На самом деле можно заметить, что достаточно использовать только $k + 1$ очередей, так как только k очередей непусты одновременно. Поэтому надо обновлять расстояния до вершин и писать в $at_dist[(d + w) \% (k + 1)]$.

0-k BFS.

Как предыдущий, только вместо очередей — деки.

1.6 Задачи, решаемые алгоритмами Дейкстры и Форда-Беллмана. Время работы.

Алгоритм Дейкстры.

Задача 1.2. Дан взвешенный граф такой, что $w : E \rightarrow \mathbb{R}^+$ и зафиксирована вершина $s \in V$. Нужно найти $\forall v \in V \ dist(s, v)$.

Опишем алгоритм Дейкстры, решающий данную задачу.

Иницилируем множество $S = \{s\}$ — множество вершин, для которых кратчайшее расстояние вычислено корректно на текущий момент времени, также будет массив $d[]$ текущих оценок на вес кратчайшего пути до вершин.

Очевидно, $d[s] = 0$, для $v \in V \setminus S$ $d[v] = \infty$.

Далее повторяем следующий алгоритм:

1. Рассмотрим все вершины v такие, что $v \notin S$, выберем среди них такую, что $d[v]$ минимально.
2. Добавим v в множество S , присвоим $dist(s, v) = d[v]$ (докажем ниже).
3. Рассмотрим ребра вида (v, t) , запишем $d[t] = \min(dist(s, v) + w(v, t), d[t])$, то есть проведем релаксацию.
4. Пока $S \neq V$, то повтори шаги выше.

Время работы алгоритма Дейкстры.

- Уменьшение оценки d для вершины. Каждое ребро уменьшает не более одного раза, значит таких операций $O(|E|)$.
- Получение вершины с минимальной оценкой d не из S . Каждая вершина извлекается не более одного раза, а значит таких операций $O(|V|)$.

Контейнер вершин	Релаксация	Извлечение	Итого
Массив	$O(1)$	$O(V)$	$O(V ^2)$
Дерево поиска	$O(\log V)$	$O(\log V)$	$O(E \log V)$
Фибоначчиева куча	$O^*(1)$	$O(\log V)$	$O(E + V \log V)$

Алгоритм Форда-Беллмана.

Задача 1.3. Найти расстояния от s до всех остальных, если $w : E \rightarrow \mathbb{R}$. Считаем, что циклов отрицательного веса нет.

1. Пусть $dp[v][k]$ равно минимальному весу пути из s в v , состоящему из ровно k ребер.

2. База. $dp[s][0] = 0, dp[:, :] = \infty$.

3. Переход. $dp[v][k] = \min_{(u,v) \in E} (dp[u][k-1] + w(u, v))$.

4. Порядок пересчета. Внешний цикл по k , внутри цикл по ребрам.

5. Ответ. $ans[v] = \min_k dp[v][k]$.

Время: $O(|V||E|)$. Память: $O(|V|^2)$.

Поиск циклов отрицательного веса.

Задача 1.4. Циклом отрицательного веса назовем цикл $v_1, v_2, \dots, v_n, v_1$, у которого $\sum_{i=1}^n w(v_i, v_{(i+1)\%n}) < 0$.

Адаптируем алгоритм Форда-Беллмана так, чтобы он хранил для каждой вершины еще предка, из которого она релаксировалась.

Утверждение. На $|V|$ -й итерации найдется вершина v , до которой расстояние уменьшилось по сравнению с $(|V| - 1)$ -й итерацией тогда и только тогда, когда в графе есть цикл отрицательного веса, достижимый из s .

Доказательство. \Rightarrow Простой кратчайший путь не может быть длиннее $|V| - 1$ ребер, а если произошла релаксация, то существует не простой путь, имеющий вес строго меньший. А значит есть цикл отрицательного веса.

\Leftarrow Рассмотрим цикл отрицательного веса $C = c_1, c_2, \dots, c_k$. Так как $|C| < |V|$, на $|V|$ -й итерации $\exists i : c_i$ будет рассмотрена второй раз, при этом она будет рассмотрена по пути вдоль отрицательного цикла, а значит произойдет релаксация. \square

1.7 Постановка задач по поиску минимального остовного дерева. Время ее решения алгоритмом Прима или Крускала.

Задача 1.5. Есть сеть, состоящая из n городов, и мы хотим соединить города интернетом, чтобы такая сеть была связна, для этого достаточно чтобы полученный граф был деревом. В дереве из n вершин у нас $n - 1$ ребро.

Определение. Матроидом называется пара (X, \mathcal{I}) , где X – множество элементов, называемое носителем матроида, а \mathcal{I} – некоторое множество подмножеств X , называемое семейством независимых множеств. В матроиде должны выполняться следующие свойства:

- $\emptyset \in \mathcal{I}$ – пустое множество независимо;
- если $A \subseteq B$, $B \in \mathcal{I}$, то $A \in \mathcal{I}$ (подмножество независимого множества – независимо);
- если $A, B \in \mathcal{I}$ и $|A| < |B|$, то существует $x \in B \setminus A$ такой, что $A \cup \{x\} \in \mathcal{I}$.

Определение. Матроид называется взвешенным, если на нем задана весовая функция: $\omega(A) = \sum \omega(a_i)$.

Теорема (Радо-Эдмондса). Пусть $A \in \mathcal{I}$ – множество минимального веса среди всех независимых подмножеств X мощности k . Возьмём такой элемент $x \notin A$, что $A \cup \{x\} \in \mathcal{I}$ и вес $\omega(x)$ минимален. Тогда $A \cup \{x\}$ – множество минимального веса среди независимых подмножеств X мощности $k + 1$.

Алгоритм Радо-Эдмондса.

Пусть нам нужно найти независимое множество, которое будет включать как можно больше элементов и при этом иметь как можно меньший вес. Из теоремы Радо-Эдмондса следует, что нам достаточно отсортировать элементы по возрастанию, и в таком порядке добавлять их в ответ.

Алгоритм Крускала.

Применив теорему Радо-Эдмондса к примеру (множеству лесов графа), получим жадный алгоритм:

1. Сортируем ребра по весам от меньшего к большему.
2. Заводим множество T , которое изначально пусто.
3. Рассматриваем ребра по возрастанию веса: если добавление ребра в T не делает множество T циклическим (граф с циклом), добавляем новое ребро в это множество.

Основная проблема алгоритма – добавление элементов в множество T .
Время $O(|E| \log |E|)$. Память $O(|V|)$.

Алгоритм Прима.

Определение. (S, T) – разрез, если $(S \cup T = V) \wedge (T \cap S = \emptyset)$

Определение. (u, v) – пересекает разрез, если u и v лежат в разных частях разреза.

Определение. Пусть $G' = (V, E')$ – подграф некоторого минимального остовного дерева G . Ребро $(u, v) \notin G'$ называется *безопасным*, если при добавлении его в G' , то $G' \cup \{(u, v)\}$ также является подграфом некоторого минимального остовного дерева графа G .

Лемма (О безопасном ребре.). *Рассмотрим связный неориентированный взвешенный граф $G = (V, E)$ с весовой функцией $\omega : E \rightarrow \mathbb{R}$. Пусть $G' = (V, E')$ – подграф некоторого минимального остовного дерева графа G , $\langle S, T \rangle$ – разрез графа G такой, что ни одно ребро из G' не пересекает этот разрез, а (u, v) – ребро минимального веса среди всех рёбер, пересекающих разрез $\langle S, T \rangle$. Тогда ребро $e = (u, v)$ является безопасным для G' .*

Доказательство. Построим E' до некоторого минимального остовного дерева, обозначим его T_{\min} . Если ребро $e \in T_{\min}$, то лемма доказана, поэтому рассмотрим случай, когда ребро $e \notin T_{\min}$. Рассмотрим путь в T_{\min} от вершины u до вершины v . Так как эти вершины принадлежат разным долям разреза, то хотя бы одно ребро пути пересекает разрез, назовём его e' . По условию леммы $w(e) \leq w(e')$.

Заменяем ребро e' в T_{\min} на ребро e . Полученное дерево также является минимальным остовным деревом графа G , поскольку все вершины G по-прежнему связаны и вес дерева не увеличился. Следовательно, $E' \cup \{e\}$ можно дополнить до минимального остовного дерева в графе G , то есть ребро e – безопасное. \square

Алгоритм Прима:

1. Инициализируем множество $S = \{s\}$ – множество вершин, на которых уже построен миностов.
2. Рассмотрим разрез $\langle S, T \rangle$. Найдём безопасное для него ребро $e = (u, v)$, где $u \in S$.
3. Добавим e в миностов и v в S .
4. Добавим в множество ребер, пересекающих разрез ребра, выходящие из v и идущие не в S .
5. Пока $S \neq V$, повтори шаги выше.

Время работы алгоритма Прима.

- Временная сложность при использовании массива — $O(|V|^2)$
- Временная сложность при использовании бинарной кучи/дерева — $O(|E| \log |V|)$

- Временная сложность при использовании фибоначчиевой кучи — $O(|E| + |V| \log |V|)$

1.8 Система непересекающихся множеств: API, наивная реализация на массиве.

Заметим, что в процессе работы алгоритма Крускала компоненты связности графа можно представить как набор множеств, которые необходимо объединять, также проверять в одном ли множестве вершины. Хотим соорудить структуру, которая умеет эффективно выполнять следующие операции:

- Создать систему из n множеств.
- Объединить два множества в одно.
- Проверить для двух элементов в одном или в разных множествах они лежат.
- Этого хватит для реализации алгоритма Крускала.

Такая структура данных носит название СНМ — система непересекающихся множеств.

Наивная реализация.

```

1 // Disjoint Set Union (DSU)
2 void MakeDSU(int cnt) {
3     for (size_t id = 0; id < cnt; ++id) {
4         parent[id] = id;
5     }
6 }
7
8 int FindSet(int a_id) {
9     if (a_id == parent[a_id]) {
```



```

10     return a_id; // root
11 }
12 return FindSet(parent[a_id]);
13 }
14
15 void UnionSets(int a_id, int b_id) {
16     a_id = FindSet(a_id);
17     b_id = FindSet(b_id);
18     if (a_id != b_id) {
19         parent[a_id] = b_id;
20     }
21 }

```

Эвристика сжатия пути.

Заметим что в функции FindSet мы можем сохранять возвращенный путь как предка, это значительно ускорит дальнейшие обращения.

Эвристика объединения по рангу.

Будем помнить размеры деревьев. К большему по размеру дереву будем подвешивать меньшее.

Время работы.

При объединении двух эвристик время работы на один запрос будет составлять $O(\alpha(N))$, где α – обратная функция Аккермана. (Б/Д)

1.9 Постановка задачи поиска наименьшего общего предка. Наивное решение.

Определение. Пусть дано дерево T , подвешенное за вершину r . Тогда назовем наименьшим общим предком двух вершин u, v такую вершину X , что

она лежит одновременно на путях $u \rightarrow r$ и $v \rightarrow r$, при этом такая вершина глубже всех таких вершин.

Обозначение: $X = LCA(u, v)$.

Наивное решение.

1. Найдем глубину двух вершин.
2. Для той вершины, которая лежит глубже поднимемся на разность высот.
3. Далее будем подниматься до тех пор, пока не встретимся.

Предпосчет: $O(|V|)$, запрос: $O(|V|)$.

Двоичные подъемы.

- Перед выполнением предпосчета посчитаем массивы: $parent_v$ и массив глубин d_v .
- Давайте предпосчитаем подъемы вверх, на расстояния степеней двойки.
- $dp[v][i]$ – вершина, в которую мы попадем если поднимемся на 2^i шагов вверх.

•

$$dp[v][i] = \begin{cases} parent[v], & i = 0, \\ dp[dp[v][i-1]][i-1], & i > 0. \end{cases}$$

- Заведем два указателя, один на одну вершину, другой на вторую.
- Разложим разность глубин (вот для чего мы считали d_v) на сумму степеней двойки и поднимемся из более глубокой вершины до уровня менее глубокой вершины.

- Теперь заметим, что если взять $k = \log_2 |V|$, можно уменьшать k до тех пор, пока попадаем в одну вершину, если не попадаем, то делаем подъем для обеих. Продолжаем такие прыжки, пока можем уменьшать k .
- В итоге, когда мы остановимся, возьмем родителя любого из указателей.

Предпосчет: $O(|V| \log |V|)$, запрос: $O(\log |V|)$.

Сведение LCA к RMQ.

- Рассмотрим u, v, w , т.ч. $w = LCA(u, v)$.
- Рассмотрим обход dfs, запущенный из корня.
- Тогда сначала он посетит w , затем u (или v), вернется в w , затем посетит v (или u соответственно). Затем опять посетит w .
- Можно посчитать порядок посещения вершин в dfs, и сохранить в массив $Order$ (вершины записываем как при входе так и при возврате из детей, такой обход называется эйлеровым обходом).
- $First[u], First[v]$ – момент первого посещения вершин u и v .
- $h[i]$ – высота вершины, которая сохранена в $Order[i]$.
- Задача сводится к $id = RMQ_h(First[u], First[v])$.
- Ответ лежит в $Order[id]$.

Сведение RMQ к LCA.

- Будем использовать декартово дерево по неявному ключу.
- Корень - минимальный элемент (то есть значение элемента просто будет приоритетом в дереве).

- В каждом поддереве аналогично.
- $RMQ(l, r) = LCA(A[l], A[r])$.

1.10 Понятие сети, потока в сети, разреза в сети. Остаточная сеть.

Определение. Транспортная сеть $G = (V, E)$ представляет собой ориентированный граф, в котором каждое ребро $(u, v) \in E$ имеет неотрицательную пропускную способность (capacity), $c(u, v) > 0$, если ребро $(u, v) \in E$, то $c(u, v) = 0$. В транспортной сети выделяются две вершины: источник (source) s и сток (sink) t . Для удобства предполагается, что каждая вершина лежит на некоем пути из источника к стоку.

Определение. Поток (flow) в G является действительная функция $f : V \times V \rightarrow \mathbb{R}$, удовлетворяющая трем условиям:

- Ограничение пропускной способности (capacity constraint): $f(u, v) \leq c(u, v), \forall (u, v) \in V$.
- Антисимметричность (skew symmetry): $f(u, v) = -f(v, u), \forall (u, v) \in V$.
- Сохранение потока (flow conservation): $\sum_{v \in V} f(u, v) = 0, \forall u \in V \setminus \{s, t\}$.

Определение. Количество $f(u, v)$, которое может быть положительным, нулевым или отрицательным, называется потоком (flow) из вершины u в вершину v . Величина потока определяется как:

$$|f| = \sum_{v \in V} f(s, v)$$

т. е. это суммарный поток, выходящий из источника.

Определение. Суммарный положительный поток, входящий в вершину:

$$\sum_{v \in V, f(v, u) > 0} f(v, u).$$

Определение. Суммарный чистый поток в некоторой вершине равен разности суммарного положительного потока, выходящего из данной вершины, и суммарного положительного потока, входящего в нее. Одна из интерпретаций свойства сохранения потока состоит в том, что для отличной от источника и стока вершины входящий в нее суммарный положительный поток должен быть равен выходящему суммарному положительному потоку.

Определение. Пусть задана некая транспортная сеть $G = (V, E)$ с источником s и стоком t . Пусть f – некоторый поток в G . Рассмотрим пару вершин $u, v \in V$. Величина дополнительного потока, который мы можем направить из u в v , не превысив пропускную способность $c(u, v)$, является остаточной пропускной способностью ребра (u, v) , и задается формулой $c_f(u, v) = c(u, v) - f(u, v)$.

Лемма. Пусть $G = (V, E)$ – транспортная сеть с источником s и стоком t , а f – поток в G . Пусть G_f – остаточная сеть в G , порождённая потоком f , а f' – поток в G_f . Тогда сумма потоков $f + f'$ является потоком в G , и величина этого потока равна $|f + f'| = |f| + |f'|$.

Определение. Для заданной транспортной сети $G = (V, E)$ и потока f , остаточной сетью в $G_f = (V, E_f)$, порожденной потоком f , является сеть $G_f = (V, E_f)$, где: $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$.

Определение. Для заданных транспортной сети $G = (V, E)$ и потока f увеличивающим путем p является простой путь из s в t в остаточной сети G_f .

Определение. Максимальная величина, на которую можно увеличить поток вдоль каждого ребра увеличивающего пути p , называется остаточной пропускной способностью p и задается формулой: $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$

Определение. Разрезом (S, T) транспортной сети $G = (V, E)$ называется разбиение множества вершин на множества $S, T = V - S$, такие что $s \in S, t \in T$. Если f – поток, то чистый поток через разрез равен: $f(S, T)$. Пропускной способностью разреза является $c(S, T)$. Минимальный разрез сети – разрез, пропускная способность которого среди всех разрезов минимальна.

1.11 Формулировка теоремы о максимальном потоке и минимальном разрезе (Форда-Фалкерсона). Время поиска максимального потока.

Теорема (Форда-Фалкерсона). Если f – некоторый поток в транспортной сети $G = (V, E)$ с источником s и стоком t , то следующие утверждения эквивалентны:

- f – максимальный поток в G .
- Остаточная сеть не содержит увеличивающих путей.
- $|f| = c(S, T)$ для некоторого разреза (S, T) сети G .

Лемма. Пусть f – некоторый поток в транспортной сети G с источником s и стоком t , и пусть (S, T) – разрез G . Тогда чистый поток через разрез равен $f(S, T) = |f|$.

Задача о максимальном потоке.

Задача 1.6. Дана некоторая транспортная сеть G с источником s и стоком t , и необходимо найти поток максимальной величины.

Метод Форда-Фалкерсона.

- Находим на каждой итерации увеличивающий путь и увеличиваем поток вдоль каждого ребра этого пути на величину остаточной пропускной способности $c_f(p)$. Пути ищем при помощи поиска в ширину.
- Не забываем об антисимметричности.
- Перед запуском алгоритма очевидно полагаем поток равным нулю.

Анализ работы.

- Скорость работы зависит от метода поиска увеличивающих путей.
- В случае иррациональных чисел алгоритм может и не сойтись.
- Проанализируем работу алгоритма с целыми числами и использованием dfs для поиска увеличивающих путей.
- В случае целых чисел величина потока увеличивается всегда хотя бы на 1.
- Асимптотика $O(|E||f|)$.

Поиск паросочетаний при помощи потоков.

Хотим наибольшее паросочетание.

- Создадим фиктивный исток, и направим из него ребра веса 1 в каждую из вершин левой доли.
- Создадим фиктивный сток, в него направим ребра веса 1 из правой доли.
- Ориентируем все ребра из левой доли в правую, ставим вес 1.
- Ищем макс. поток.

2. Вопросы на уд.

2.1 Лемма о белых путях.

Лемма. *Рассмотрим момент, когда вершина v была покрашена в серый. Тогда все вершины, достижимые по белым путям из v покрасятся в черный к моменту выхода из v .*

Доказательство. • Черные вершины, очевидно останутся черными.

- Серые вершины, очевидно останутся серыми, так как для того, чтобы они окрасились в черный, необходимо сначала чтобы вершина v окрасилась в черный (они все лежат в стеке рекурсии).
- Осталось показать что все достижимые из v белые вершины станут черными.

Заметим, что ни в какой момент времени не может быть такого, что есть ребро из черной вершины в белую, это условие противоречит самому алгоритму, ведь мы красим вершины в черный только тогда, когда все их дети будут окрашены в черный (посещены).

Допустим $\exists u$, цвет которой отличен от черного и при этом она была достижимой из v по белому пути.

- Если этот цвет — белый, то это значит что вершина u вообще не посещалась, если рассмотреть путь который был белым в момент когда v была серой, то можно заметить, после окраски v в черный у нас появилось ребро из черной вершины в белую, чего быть не может. Противоречие.
- Если этот цвет — серый, то это значит, что мы исследовали не всех ее соседей, но так как мы посетили вершину u после вершины v , то из этого следует что мы не могли обработать и всех соседей вершины v . Противоречие.

Тогда вершина u не могла быть достижимой из v по белому пути. □

2.2 Критерий двудольности графов. Алгоритм поиска разбиения на доли.

Теорема (Критерий двудольности графов). *Граф $G = (V, E)$ является двудольным тогда и только тогда, когда его можно раскрасить в два цвета*

так, что никакие две смежные вершины не будут иметь одинаковый цвет. Эквивалентно: граф является двудольным тогда и только тогда, когда в нем нет циклов нечётной длины.

Проверить можно при помощи DFS или BFS.

2.3 Алгоритм Дейкстры.

См. 1.6.

2.4 Алгоритм Форда-Беллмана.

См. 1.6.

2.5 Задача APSP. Алгоритм Флойда-Уоршелла.

Задача 2.1. Решить задачу APSP, если $w : E \rightarrow \mathbb{R}$. APSP — all pairs shortest paths.

1. Пусть $dp[u][v][k]$ равно минимальному весу пути из u в v , состоящему из вершин с номерами $\leq k$.
2. База. $dp[u][u][0] = 0$, $dp[:, :, 0] = 0$.
3. Переход. $dp[u][v][k] = \min(dp[u][v][k], dp[u][k][k-1] + dp[k][v][k-1])$.
4. Порядок пересчета. Внешний цикл по k , внутри два вложенных цикла по u и v .
5. Ответ. $ans[u][v] = dp[u][v][|V| - 1]$.

Время: $O(|V|^3)$. Память: $O(|V|^3)$.

2.6 Постановка задачи поиска минимального остовного дерева. Лемма о безопасном ребре. Алгоритм Прима.

См. 1.7.

2.7 Постановка задачи поиска наименьшего общего предка. LCA. Наивное решение. Решение с использованием двоичных подъемов.

См. 1.9.

2.8 Алгоритм Куна для поиска максимального паросочетания в двудольном графе. Улучшения алгоритма Куна.

Определение. Паросочетанием в неориентированном графе $G = (V, E)$ называют множество ребер $M \subseteq E$ такое, что не найдется двух ребер из M с общей вершиной.

Определение. Паросочетание M называется максимальным, если не существует паросочетания M' такого, что $|M| < |M'|$.

Определение. Увеличивающая цепь относительно паросочетания M — путь $p = (v_1, \dots, v_{2k})$ такой, что $(v_1, v_2) \notin M, (v_2, v_3) \in M, (v_3, v_4) \notin M, \dots, (v_{2k-1}, v_{2k}) \notin M$, при этом вершины v_1 и v_{2k} не насыщены M .

Теорема (Бержа). Паросочетание M максимально тогда и только тогда, когда относительно M нет увеличивающих цепей.

Лемма. Если в графе степень каждой вершины не превосходит двух, то его ребра разбиваются на непересекающиеся пути и циклы.

Лемма. *Если из вершины v не была найдена увеличивающая цепь, то и далее из нее не найдется увеличивающая цепь.*

Алгоритм Куна.

1. Ищем увеличивающую цепь (при помощи dfs), чередуем ее.
2. Сделаем так для всех вершин.
3. Утверждается, что этого хватит.

Поиск цепи.

Поиск увеличивающей цепи осуществляется с помощью специального обхода в глубину. Изначально обход в глубину стоит в текущей ненасыщенной вершине v левой доли. Просматриваем все рёбра из этой вершины, пусть текущее ребро (v, to) . Если вершина to ещё не насыщена паросочетанием, то, значит, мы смогли найти увеличивающую цепь: она состоит из единственного ребра (v, to) ; в таком случае просто включаем это ребро в паросочетание и прекращаем поиск увеличивающей цепи из вершины v . Иначе, если to уже насыщена каким-то ребром (to, p) , то попытаемся пройти вдоль этого ребра: тем самым мы попробуем найти увеличивающую цепь, проходящую через рёбра (v, to) , (to, p) . Для этого просто перейдём в нашем обходе в вершину p — теперь мы уже пробуем найти увеличивающую цепь из этой вершины.

Таким образом, алгоритм Куна это n запусков DFS или его сложность: $O(|V|(|V| + |E|))$.

Улучшения алгоритма Куна.

- Для запуска алгоритма лучше взять меньшую долю — тогда нужно меньше запусков.

- Можно проинициализировать исходное паросочетание каким-либо не максимальным паросочетанием, например просто жадно набрать ребер, это также потенциально уменьшит число запусков.

2.9 Остаточная сеть. Дополняющий поток. Сложение потоков.

См. 1.10.

2.10 Теорема «о максимальном потоке и минимальном разрезе» (Форда-Фалкерсона). Метод Форда-Фалкерсона. Пример долгой работы, при реализации через dfs. Сведение задачи поиска максимального паросочетания к задаче поиска максимального потока.

См. 1.11.

2.11 Слоистая сеть. Блокирующий поток. Схема Диница. Число итераций в схеме Диница. Жадный поиск блокирующего потока.

Определение. Слоистой сетью N_l для сети N назовём $N_l = (G_l = (V, E_l), c_l, s, t)$, где

$$E_l = \{e = (u, v) \in E \mid d[v] - d[u] = 1\},$$

$$c_l(u, v) = c(u, v) \cdot \mathbf{I}\{(u, v) \in E_l\},$$

$d[v]$ – расстояние в рёбрах от s до v .

Определение. Блокирующий поток — такой поток f , что не найдется пути, вдоль которого поток из s в t можно увеличить.

Схема Диница.

1. Построим сеть N , определим поток $f = 0$. Строим N_f .
2. Строим слоистую сеть из остаточной N_f .
3. Если в N_f , t недостижима из s , то алгоритм окончен. Иначе найдем блокирующий поток f' в N_f .
4. $f = f + f'$, обновляем остаточную сеть вдоль пропущенного потока.

Очевидно, шаги 1, 2 и 4 делаются за $O(|V| + |E|)$. Осталось понять, как искать блокирующий поток и сколько итераций будет, пока не произойдет выход из алгоритма.

Утверждение. *Расстояние между истоком и стоком строго увеличивается после каждой фазы алгоритма, т.е. $d'[t] > d[t]$, где $d'[t]$ — значение, полученное на следующей фазе алгоритма.*

Доказательство. Проведём доказательство от противного. Пусть длина кратчайшего пути из истока в сток останется неизменной после очередной фазы алгоритма. Слоистая сеть строится по остаточной. Из предположения следует, что в остаточной сети будут содержаться только рёбра остаточной сети перед выполнением данной фазы, либо обратные к ним. Из этого получаем, что нашёлся путь из s в t , который не содержит насыщенных рёбер и имеет ту же длину, что и кратчайший путь. Но этот путь должен был быть «заблокирован» блокирующим потоком, чего не произошло. Получили противоречие. Значит длина изменилась. \square

Следствие. Число итераций в алгоритме Диница составляет $|V| - 1$.

Удаляющий обход.

Идея заключается в том, чтобы по одному находить пути из истока s в сток t , пока это возможно. Обход в глубину найдёт все пути из s в t , если из s достижима t , а пропускная способность каждого ребра $c(u, v) > 0$, поэтому, насыщая рёбра, мы хотя бы единожды достигнем стока t , следовательно блокирующий поток всегда найдётся.

Ускорим данный алгоритм. Будем удалять в процессе обхода в глубину из графа все рёбра, вдоль которых не получится дойти до стока t . Это очень легко реализовать: достаточно удалять ребро после того, как мы просмотрели его в обходе в глубину (кроме того случая, когда мы прошли вдоль ребра и нашли путь до стока). С точки зрения реализации, надо просто поддерживать в списке смежности каждой вершины указатель на первое не удалённое ребро, и увеличивать этот указатель в цикле внутри обхода в глубину.

Если обход в глубину достигает стока, насыщается как минимум одно ребро, иначе как минимум один указатель продвигается вперед. Значит один запуск обхода в глубину работает за $O(|V| + K)$, где K — число продвижения указателей. Ввиду того, что всего запусков обхода в глубину в рамках поиска одного блокирующего потока будет $O(P)$, где P — число рёбер, насыщенных этим блокирующим потоком, то весь алгоритм поиска блокирующего потока отработает за $O(P|V| + \sum_i K_i)$, что, учитывая, что все указатели в сумме прошли расстояние $O(|E|)$, даёт асимптотику $O(P|V| + |E|)$. В худшем случае, когда блокирующий поток насыщает все рёбра, асимптотика получается $O(|V||E|)$.

Таким образом, научились искать блокирующий поток за $O(|V||E|)$, а значит алгоритм Диница с удаляющим обходом отработает за $O(|V|^2|E|)$, что «на одну степень $|V|$ быстрее».

3. Вопросы на хор.

3.1 Отношение сильной связности. Компоненты сильной связности. Алгоритм Косарайю.

См. 1.2.

3.2 Отношение реберной двусвязности. Мосты. Функция t_{up} . Алгоритм поиска мостов.

См. 1.4.

3.3 Вершинная двусвязность. Точки сочленения. Функция t_{up} . Алгоритм поиска точек сочленения.

См. 1.4.

Определение. Две вершины вершинно двусвязны, если между ними есть два вершинно непересекающихся пути.

Определение. Точка сочленения — вершина, при удалении которой увеличивается число компонент связности.

Теорема (Критерий точки сочленения). *Рассмотрим древесное ребро (v, to) .*

1. Пусть v — корень дерева обхода, тогда v — точка сочленения \iff у нее хотя бы два ребенка.
2. Пусть v — не корень дерева обхода, тогда v — точка сочленения $\iff t_{up}(to) \geq t_{in}(v)$.

Доказательство. 1. Это то же самое, что есть два древесных ребра из v .

\Rightarrow От противного. У v всего один ребенок в дереве обхода. Но тогда удаление v не ломает связности.

\Leftarrow К моменту обхода первого поддерева второе будет белым. Но в черные вершины ребер нет, значит нет ребер между этими поддеревьями, а единственный путь проходит через корень.

2. $t_{up}(to) \geq t_{in}(v)$ значит, что выше v из поддерева to не прыгнешь.

\Rightarrow Очевидно.

\Leftarrow От противного. Пусть для всех детей v верно, что $t_{up}(to) < t_{in}(v)$, тогда из каждого ребенка можно прыгнуть в наддерево, откуда v — не точка сочленения. Противоречие.

□

3.4 Взвешенные графы. Обход в ширину BFS. Его модификации: 0-1 BFS, 1-k BFS, 0-k BFS.

См. 1.5.

3.5 Поиск циклов отрицательного веса.

См. 1.6.

3.6 Система непересекающихся множеств. Реализация с использованием леса деревьев. Эвристики. Время работы (б/д).

См. 1.8

3.7 Постановка задачи поиска минимального остовного дерева. Лемма о безопасном ребре. Алгоритм Крускала.

См. 1.7.

3.8 Алгоритм Борувки.

Имеем связный граф $G = (V, E)$, с заданной весовой функцией $w : E \rightarrow \mathbb{R}$. Хотим найти минимальное остовное дерево. Мы уже знаем два алгоритма, которые с этим хорошо справляются:

- Алгоритм Крускала.
- Алгоритм Прима.

Познакомимся с еще одним замечательным алгоритмом поиска минимального остовного дерева. Для этого нам понадобится одна простая лемма.

Лемма. *Для любой вершины наименьшее инцидентное к ней ребро является безопасным.*

Доказательство. Рассмотрим некоторую вершину u , обозначим наименьшее инцидентное ей ребро как (u, v) . Предположим, имеется $T_{\min} \subset E$, являющийся минимальным остовным деревом, такое, что $(u, v) \notin T_{\min}$. Поскольку остовное дерево связно, существует путь P из u в v . Покажем, что такое дерево можно улучшить:

У вершины u существует инцидентное ребро $(u, z) \in T_{\min}$. Добавим (u, v) в T_{\min} . Это ребро представляет из себя путь из u в v . Таким образом, мы образовали цикл. Можно удалить (u, z) и получить дерево T_{\min}^2 .

В силу минимальности веса (u, v) : $w(T_{\min}^2) \leq w(T_{\min})$. Противоречие.

□

Алгоритм Борувки:

1. Имеем лес деревьев, изначально это лес из вершин графа.
2. Для каждого дерева найдем наименьшее инцидентное дереву ребро.
3. Добавим эти ребра к деревьям (некоторые могут добавиться более одного раза).
4. После этого некоторые деревья склеиваются в новые деревья, более крупные.
5. Повторяем шаги 2-4, пока не останется одно дерево.

Сложность алгоритма Борувки.

- Временная сложность алгоритма составляет $O(|E| \log |V|)$.
- На каждом шаге число деревьев уменьшается хотя бы в два раза, так как каждое дерево задействовано в объединении.
- Потребление памяти составляет $O(|V|)$.
- Нам нужно поддерживать СНМ, а также минимумы, СНМ содержит не более чем V множеств.

3.9 Постановка задачи поиска наименьшего общего предка LCA. Сведение LCA \iff RMQ.

См. [1.9](#).

3.10 Двудольность графов. Паросочетания. Увеличивающие цепи и теорема Бержа.

См. [2.8](#).

3.11 Поток через разрез. Неравенство между величиной произвольного потока и пропускной способностью произвольного разреза.

См. 1.10.

3.12 Алгоритм Эдмондса-Карпа. Время работы.

В алгоритме Эдмондса-Карпа мы запускаем BFS для поиска увеличивающего пути от истока s к стоку t , а затем пропускаем по найденному пути весь возможный поток, равный минимальной остаточной пропускной способности на этом пути.

Теорема. Алгоритм Эдмондса-Карпа работает за $O(|V||E|^2)$.

Лемма. Если в сети $N = (G, c, s, t)$ увеличение потока производится вдоль кратчайших $s \rightsquigarrow t$ путей в N_f , то $\forall v \in V \setminus \{s, t\}$ длина кратчайшего пути $d_f(s, v)$ в N_f не убывает.

Доказательство. Пусть f, f' – потоки в N , между которыми одна итерация. Пусть v – вершина, $d_f(s, v)$ до которой минимально и $d_{f'}(s, v) < d_f(s, v)$. Рассмотрим путь $p = s \rightsquigarrow u \rightarrow v$, являющийся кратчайшим от s до v в $N_{f'}$. Тогда верно, что $d_{f'}(s, u) = d_{f'}(s, v) - 1$.

По выбору v и из предыдущего утверждения получаем, что $d_{f'}(s, u) \geq d_f(s, u)$.

- Если $(u, v) \in E_f$, тогда

$$d_f(s, v) \leq d_f(s, u) + 1 \leq d_{f'}(s, u) + 1 = d_{f'}(s, v).$$

- Если $(u, v) \notin E_f$, но $(u, v) \in E_{f'}$. Появление (u, v) означает увеличение потока по обратному ребру (v, u) . Увеличение потока производится вдоль

кратчайшего пути, поэтому кратчайший путь из s в u , вдоль которого происходило увеличение, выглядит как $s \rightsquigarrow v \rightarrow u$, откуда

$$d_f(s, v) = d_f(s, u) - 1 \leq d_{f'}(s, u) - 1 = d_{f'}(s, v) - 2.$$

□

Доказательство. Назовём ребро e вдоль кратчайшего пути p в N_f от s до t критическим, если $c_f(e) = \min_{e \in p} c_f(p)$. Покажем, что каждое ребро становится критическим $O(|V|)$ раз.

Заметим, что после увеличения все критические рёбра исчезают из остаточной сети. Рассмотрим $(u, v) \in E$. Увеличение производится вдоль кратчайших путей, поэтому если (u, v) становится критическим в первый раз, то $d_f(s, v) = d_f(s, u) + 1$. Затем оно исчезает из сети и не появится, пока не будет уменьшено по обратному ребру (v, u) .

Пусть в момент перед увеличением поток в сети N составлял f' , тогда $d_{f'}(s, u) = d_{f'}(s, v) + 1$. Согласно лемме $d_f(s, v) \leq d_{f'}(s, v)$, откуда

$$d_{f'}(s, u) = d_{f'}(s, v) + 1 \geq d_f(s, v) + 1 = d_f(s, u) + 2.$$

Получаем, что между итерациями, когда $(u, v) \in E$ становится критическим, расстояние от s до u увеличивается на 2, откуда $O(|V|)$ раз оно могло стать критическим.

Всего рёбер $O(|E|)$, значит, суммарное число итераций составит $O(|V||E|)$.

Время работы каждой итерации – $O(|E|)$.

Тогда итоговое время работы алгоритма Эдмондса-Карпа: $O(|V||E|^2)$.

□

3.13 Слоистая сеть. Блокирующий поток. Схема Диница.

Число итераций в схеме Диница. Удаляющий обход.

См. [2.11](#)

3.14 Алгоритм масштабирования потока.

- Введем $U = \max_{(u,v) \in E} c(u, v)$.
- Рассмотрим $k = \lfloor \log_2 U \rfloor \dots 0$, $\Delta_k = 2^k$.
- На каждой фазе рассматриваем в остаточной сети только ребра с пропускными способностями хотя бы Δ_k и более. Таким образом мы будем сначала пытаться протолкнуть большие потоки и только потом переходить к маленьким.
- $\Delta_0 = 1$, при нем алгоритм вырождается в алгоритм Эдмондса-Карпа, в следствии чего он является корректным.

Лемма. $|f|$ в сети G ограничен сверху значением $|f_k| + 2^k|E|$.

Доказательство. Рассмотрим разрез. Вспомним что поток ограничен пропускной способностью разреза. Поскольку на k фазе мы рассматриваем ребра с пропускными способностями от 2^k , то остаточная возможная пропускная способность будет не более чем $2^k|E|$, в свою очередь f_k — наденый поток на фазе k . □

Лемма. *Время работы алгоритма Эдмондса-Карпа с использованием техники масштабирования потока составляет $O(|E|^2 \log U)$.*

Доказательство. • Всего фаз у алгоритма $\log U$.

- Поиск увеличивающего пути с помощью поиска в ширину работает за $O(|E|)$.

- На каждой фазе у нас не более чем $2|E|$ увеличивающих путей, так как на поток на предыдущей фазе ограничен $2^{k+1}|E|$, а каждый увеличивающий путь имеет пропускную способность как минимум 2^k .

Объединив эти три факта имеем, что время работы алгоритма составляет $O(|E|^2 \log U)$. Отметим, что использование техники оправдано, когда $\log U < |V|$. \square

3.15 Теорема о декомпозиции потока.

Теорема (О декомпозиции). *Любой поток f в сети G можно представить в виде:*

- Набора $s - t$ путей P_1, \dots, P_k с потоками $f_1, \dots, f_k > 0$
- Набора циклов C_1, \dots, C_m с потоками $f_{k+1}, \dots, f_{k+m} > 0$

При этом:

$$1. f(e) = \sum_{i: e \in P_i} f_i + \sum_{j: e \in C_j} f_{k+j}$$

$$2. \text{Суммарный поток путей равен величине потока: } \sum_{i=1}^k f_i = |f|$$

Алгоритм построения декомпозиции:

1. Пока в сети есть ненулевой поток из s :
 - (a) Начинаем из s , выбираем ребро с $f(e) > 0$ в v_1
 - (b) Если $v_1 = t$: найден $s-t$ путь P_i
 - (c) Иначе, по сохранению потока, \exists ребро из v_1 в v_2 с $f > 0$
 - (d) Продолжаем, пока не попадём в t (путь) или в посещённую вершину (цикл)

- (e) Находим минимальный поток f_i по рёбрам пути/цикла
- (f) Добавляем путь/цикл в декомпозицию с потоком f_i
- (g) Вычитаем f_i из f по всем рёбрам пути/цикла
- (h) Повторяем для оставшихся ненулевых потоков (циклы)

2. Повторяем для оставшихся ненулевых потоков (циклы).

Корректность:

- 1. На каждом шаге уменьшается поток хотя бы по одному ребру.
- 2. Максимальное число операций — $|E|$ (по числу рёбер).
- 3. Сохранение потока гарантирует возможность продолжения.
- 4. Величина $|f|$ уменьшается на поток каждого s – t пути.
- 5. После обнуления потока из s остаются только циклы.

Завершение:

- 1. Когда все потоки нулевые, декомпозиция построена.
- 2. Сумма потоков s – t путей равна исходному $|f|$.
- 3. Циклы не влияют на величину потока.

Утверждение. *Время работы алгоритма декомпозиции потока составляет $O(|V||E|)$.*

Доказательство. • Каждый путь или цикл содержит не более $|V|$ вершин
 \Rightarrow поиск одного пути/цикла занимает $O(V)$.

- Декомпозиция содержит не более $|E|$ путей/циклов:
 - На каждом шаге обнуляется хотя бы одно ребро.

– Всего рёбер — $|E|$.

- Каждая вершина рассматривается только при наличии ненулевого потока через неё.
- Итого: $O(V)$ на путь $\times O(E)$ путей = $O(|V||E|)$.

□

3.16 Биномиальная куча.

Определение. Биномиальное дерево B_k — дерево, определяемое для каждого $k = 0, 1, 2, \dots$ следующим образом: B_0 — дерево, состоящее из одного узла; B_k состоит из двух биномиальных деревьев B_{k-1} , связанных вместе таким образом, что корень одного из них является дочерним узлом корня второго дерева.

Определение. Биномиальная пирамида H — представляет из себя множество биномиальных деревьев, которые удовлетворяют следующим свойствам:

1. Каждое биномиальное дерево в H подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родительского узла. Мы говорим, что такие деревья являются упорядоченными в соответствии со свойством неубывающей пирамиды.
2. Для любого неотрицательного целого k имеется не более одного биномиального дерева H , чей корень имеет степень k .

Таким образом каждое дерево содержит самый маленький элемент.

Хранение.

- В списке корней храним деревья в порядке строгого возрастания степеней корней.

- `next_tree` имеет разный смысл для корней и для детей, для корней – это следующий корень в списке, для детей – следующий ребенок.

Поиск минимума.

- Для поиска минимума достаточно найти наименьший корень.
- Функция будет работать за $O(\log N)$.

Слияние.

- При слиянии двух куч сначала сольем их корневые списки, так чтобы степени корней не убывали.
- Далее будем сливать соседние вершины одинаковых степеней.
- Важно что объединении двух деревьев может образоваться три дерева одной степени подряд. Этот случай важно правильно обработать, пропустив первое из трех деревьев.

Вставка.

- Создаем кучу из одного биномиального дерева.
- Эту кучу можно слить с исходной.
- Работать это будет за $O(\log n)$.

Извлечение минимума.

- Ищем `min` корень в списке корней.
- Заметим что его дети – это тоже биномиальные деревья.
- Нужно перевернуть список детей, и это будет биномиальной кучей.

- Сольем две кучи.

3.17 Постановка задачи о поиске потока минимальной стоимости. Задача о назначениях и ее сведение к задаче о поиске потока минимальной стоимости.

Определение. Взвешенной сетью назовем пару (N, w) , где $w : E \rightarrow R$, при этом нет циклов отрицательного веса.

Определение. Стоимость потока — величина, равная

$$\sum_{(u,v) \in E(N)} f(u,v)w(u,v)$$

Задача 3.1. Необходимо найти максимальный поток при этом с минимальной суммарной стоимостью (min cost max flow) или поток величины k минимальной стоимости (min cost k-flow).

Утверждение. Пусть f — максимальный поток. В N_f нет циклов отрицательного веса $\iff f$ имеет минимальную стоимость.

Доказательство. \Rightarrow Пусть f неоптимальный и f^* оптимальный (по стоимости). $f^* - f$ раскладывается в набор простых циклов и $|f^* - f| < 0$, значит есть цикл отрицательного веса.

\Leftarrow Пропустим вдоль цикла отрицательного веса поток, чтобы его убрать. Величина не изменится, а стоимость уменьшится. \square

Задача о назначениях.

Задача 3.2. • Имеется N заказов и N станков

- Для каждого заказа известна стоимость изготовления на каждом станке (A_{ij}) .

- Каждый станок может выполнять только один заказ

Найти распределение заказов по станкам, минимизирующее суммарную стоимость:

$$\min_{\sigma \in S_N} \sum_{i=1}^N A_{i, \sigma(i)}$$

где S_N - множество всех перестановок порядка N .

Сведение задачи к поиску потока минимальной стоимости.

1. Строим ориентированный граф G :

- Исток S и сток T ;
- N вершин для заказов (первая доля);
- N вершин для станков (вторая доля).

2. Добавляем рёбра:

- Из S в каждую вершину-заказ: пропускная способность 1, стоимость 0;
- Из заказов в станки: пропускная способность 1, стоимость A_{ij} ;
- Из каждой вершины-станка в T : пропускная способность 1, стоимость 0.

3. Находим максимальный поток минимальной стоимости.