Билет 58. Остовный подграф, остовное дерево. Минимальный остов. Лемма о безопасном ребре.

Def. $T \subset G$ — **остовное дерево**, если T содержит все вершины G и является деревом

Def. Остовный подграф — подграф, содержащий все вершины.

Def. Минимальным остовным деревом во взвешенном неориентированном графе называется остовное дерево минимального веса.

Def. Ребро $(u, v) \notin G'$ называется безопасным, если при добавлении его в G', $G' \cup \{(u, v)\}$ также является подграфом некоторого минимального остовного дерева графа G.

Def. (S, T) — разрез, если $S \cup T = V, S \cap T = \varnothing$

Def. (u, v) пересекает разрез (S, T), если u и v — в разных частях разреза.

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

Доказательство. Достроим 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, назовем его e'. По условию леммы $w(e) \leqslant w(e')$. Заменим 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. □

Билет 59. Алгоритм Прима: доказательство корректности и реализации за $O(n^2), O(m\ log\ n), O(m + n\ log\ n)$.

Алгоритм Прима – алгоритм поиска минимального остовного дерева.

Идея: пока можем, добавляем к имеющемуся подграфу (C) минимального остовного дерева самое легкое ребро из всех, пересекающих разрез (соединяющих наш подграф с оставшейся частью графа).

В таком виде асимптотика алгоритма $O(nm)$, однако, ее несложно улучшить до $O(n^2)$. А именно для каждой вершины будем хранить минимальное ребро, которое соединяет ее с вершиной из C.

Тогда шагов будет $n - 1$ (в дереве $n - 1$ ребро), на каждом шаге $O(n)$ действий по поиску следующего ребра. Обновление происходит так: для текущей добавленной вершины достаточно обновить минимум по всем ее соседям, не входящим в C. Так что асимптотика будет $O(n^2 + m) = O(n^2)$

Заметим, что есть сходство с алгоритмом Дейкстры. Нам нужно делать extract min, а потом decrease key, причем, O(n) раз. Так что мы можем, используя, например, кучу, добиться лучшей асимптотики.

- Выбираем произвольную стартовую вершину – начало строящегося дерева. Строим кучу с вершинами, до которых есть ребро из стартовой.

- Итеративно добавляем к дереву вершину из кучи с минимальным по весу ребром до дерева. Саму вершину удаляем из кучи. Добавляем новые вершины, доступные из добавленной вершины по ребрам графа, в кучу. А еще обновляем расстояние до вершины из куче $\Big($удаляем пару (старое расстояние; номер вершины), добавляем пару (новое расстояние; номер вершины)$\Big)$, если от добавленной вершины к ней ведет более легкое ребро.

Теорема. *Алгоритм Прима работает за $O(m\ log\ n)$ с очередью на куче и за $O(n\ log\ n + m)$ при очереди на Фибонначиевой куче.*

Доказательство.

- Извлечение вершины из кучи – $O(log\ n)$
- Вставка в кучу $O(log\ n)$

Пункт 1) выполнится n раз, пункт 2) m раз.

□

Билет 60. Система непересекающихся множеств (СНМ). Виды запросов. Эвристика по рангу, эвристика сжатия путей. Асимптотика ответа на запрос при использовании обеих эвристик (б/д).

Def. DSU (СНМ) — система непересекающихся множеств

- Create(u) — создать множество с одним элементом u.
- Find(u) — найти множество по элементу u, чтобы можно было сравнить их (Find(u) == Find(v)). Для удобства возвращает конкретного представителя множества как своеобразный идентификатор.
- Union(u, v) — объединить два множества, одно из которых содержит элемент u, а другое — элемент v.

Удобно воспринимать множества как подвешенные деревья. Корень – представитель или, другими словами, лидер. Соответственно, поиск представителя осуществляется как подъем по дереву к корню. Поэтому чем короче путь, тем лучше. Массив *parent* хранит родителей вершин в наших графах. Значения *parent* для представителей равно -1.

Def. В операции Union будем присоединять дерево с меньшим рангом к дереву с большим рангом. Это называется ранговая эвристика. Есть два варианта ранговой эвристики: в одном варианте рангом дерева называется количество вершин в нём, в другом — глубина дерева (точнее, верхняя граница на глубину дерева, поскольку при совместном применении эвристики сжатия путей реальная глубина дерева может уменьшаться).

Def. Эвристика сжатия пути заключается в следующем: когда после вызова Find(v) мы найдём искомого лидера p множества, то запомним, что у вершины v и всех пройденных по пути вершин — именно этот лидер p. Проще всего это сделать, перенаправив их parent[] на эту вершину p.

Совместное использование эвристик дает асимптотику $O(\alpha(n))$, где $\alpha(n)$ – обратная функция Аккермана, которая растёт очень медленно, настолько медленно, что для всех разумных ограничений n она не превосходит 4 (по крайней мере, для $n \leq 10^{600}$ точно).

Билет 61. Асимптотика ответа на запрос в СНМ при использовании только эвристики по рангу.

Теорема. *Ранговая эвристика дает* $O(\log n))$ *на каждый запрос.*

Доказательство. Рассмотрим ранговую эвристику по глубине дерева. Покажем, что если ранг дерева равен k, то это дерево содержит как минимум 2^k вершин (отсюда будет автоматически следовать, что ранг, а, значит, и глубина дерева, есть величина $O(\log n)$). Доказывать будем по индукции: для $k = 0$ это очевидно. Ранг дерева увеличивается с $k-1$ до k, когда к нему присоединяется дерево ранга $k - 1$; применяя к этим двум деревьям размера $k-1$ предположение индукции, получаем, что новое дерево ранга k действительно будет иметь как минимум 2^k вершин. □

Билет 62. Алгоритм Крускала: корректность, реализация, асимптотика.

Алгоритм Крускала – еще один алгоритм поиска минимального остовного дерева.

- Сортируем все ребра графа по весу.
- Инициализируем лес деревьев. Изначально каждая вершина — дерево.
- Последовательно рассматриваем ребра графа в порядке возрастания веса. Если очередное ребро соединяет два разных дерева из леса, то объединяем эти два дерева этим ребром в одно дерево. Если очередное ребро соединяет две вершины одного дерева из леса, то пропускаем такое ребро.
- Повторяем 3), пока в лесу не останется одно дерево.

Теорема. *Алгоритм Крускала корректен*

Доказательство. Рассмотрим шаг алгоритма. Пусть текущее ребро при добавлении не создает цикл. Тогда оно соединяет два разных поддерева в MST, то есть соединяет разрез. У него минимальный вес, значит оно безопасно. □

Теорема. *Алгоритм Крускала работает за* $O(m \log m)$

Доказательство. Сортировка ребер займет $O(m \log m)$. Работа с СНМ займет $O(m\alpha(n))$, где α — обратная функция Аккермана, которая не превосходит 4 во всех практических приложениях и которую можно принять за константу. Алгоритм работает за $O(m(\log m + \alpha(n))) = O(m \log m)$. □

```
void make_set(int v) {
  parent[v] = v;
  rank[v] = 0;
}

int find_set(int v) {
  if (parent[v] == -1) return v;
  return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
  a = find_set(a);
```

```
  b = find_set(b);
  if (a != b) {
    if (rank[a] < rank[b])
      swap (a, b);
    parent[b] = a;
    if (rank[a] == rank[b]) ++rank[a];
  }
}

sort(edges.begin(), edges.end(), cmp)
\\edges -- массив ребер, сортируем его в порядке неубывания длин ребер
for (auto e : edges) {
  if (find_set(e.u) != find_set(e.v)) {
    union_sets(e.u, e.v);
  }
}
```