7. Алгоритми в графи с тегла на ребрата. Оценки за сложност

1. **Дефиниция на минимално покриващо дърво (МПД) на свързан граф с тегла на ребрата**

Деф: Минимално покриващо дърво

Нека е даден свързан граф $G = (V, E)$. Нека $c: E \to \mathbb{R}$ е функция, която задава цена на всяко ребро. Минимално покриващо дърво на G наричаме покриващо дърво $D = (V, E_0)$, т.ч. за всяко друго покриващо дърво на G, $D' = (V, E')$ е изпълнено:

$$\sum_{e \in E_0} c(e) \le \sum_{e \in E'} c(e)$$

Тегло на дърво наричаме $c(D) = \sum_{e \in E'} c(e)$

2. **Теорема за съгласуваното множество (условия за нарастване на подмножество на МПД)**

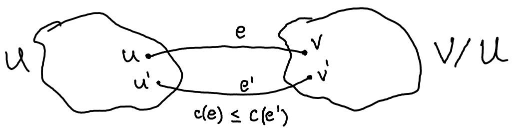
Нека $G = (V, E)$ е граф.

Твърдение:

Нека $U \subseteq V$ и нека $e = \{u, v\} \in E$ е ребро, т.ч. $u \in U, v \in V \setminus U$ и измежду всички ребра от вида $e' = \{u', v'\}$ с $u' \in U, v' \in V \setminus U$, реброто e е с най-ниска цена, т.е. $c(e) \le c(e')$.

Тогава G има МПД, което съдържа e

Д-во:



Допускаме, че не съществува МПД, което съдържа e.

Нека $T = (V, E_0)$ е МПД и $e \notin E_0$.

Тогава T има път от u до v, $u = u_0, u_1, \dots, u_k = v$, в който участва поне едно ребро $e' = \{u_i, u_{i+1}\}$, т.ч. $u_i \in U$ и $u_{i+1} \in V \setminus U$. Нека e' е това ребро.

Така в графа $T' = (V, E_0 \cup \{e\})$ има цикъл $v, u = u_0, u_1, \dots, u_k = v$

Строим дървото $T'' = (V, (E_0 \setminus \{e'\}) \cup \{e\})$, то е покриващо дърво на G, по условие $c(e) \le c(e')$

$c(T'') \le c(T)$

Ако допуснем, че $c(T'') > c(T)$, то противоречи с $c(e) \le c(e')$ и получаваме абсурд.

3. **Алгоритъм на Прим - *построява МПД по зададен начален връх***

Нека е даден $G = (V, E)$ и функция $c: E \to \mathbb{R}$, задаваща цената на ребрата.

В имплементацията по-долу Q е приоритетна опашка на база на key атрубута, тя съдържа всички върхове, които все още не са в дървото.

За всеки връх v, стойността на $v.key$ е минималната тежест на ребро, свързващо v към връх в дървото. Ако няма такова ребро, то $v.key = \infty$. Атрибута $v.\pi$ съдържа родителят на върха v в дървото.

```
MST_Prim(G(V,E), c, r)
for each v ∈ V do:                          } Init
      v.key = ∞
      v.π = NIL
r.key = 0
Q = V

while Q ≠ ∅                      // |V| times
      u = Extract-Min(Q)
            for each v ∈ G.Adj[u] do     // O(E) times
                  if v ∈ Q and c(u,v) < v.key
                        v.π = u
                        v.key = c(u,v)     // Decrease-Key
```

Алгоритъмът поддържа следната инварианта:

Преди всяко изпълнение на *while* цикъла:

1. $A = \{(v, v.\pi): v \in V \setminus (\{r\} \cup Q)\}$
2. Върховете, които вече са поставени в МПД са тези във $V \setminus Q$
3. $\forall v \in Q$, ако $v.\pi \ne NIL$, то $v.key < \infty$ и $v.key$ е тежестта на най-леката ребро $(v, v.\pi)$, свързващо v до друг връх, вече добавен в МПД

Проверката дали елемент е от Q може да се направи константна, като държим в бит информация, която ни казва дали елементът е в Q или не.

Спрямо реализацията на приоритетната опашка:

- Ако се използва binary heap, то Extract-Min операцията отнема $O(\lg V)$, а всички извиквания на операцията отнемат $O(V \lg V)$ време.
  Decrease-Key операцията отнема $O(\lg V)$ време при binary heap. Така цялата сложност на алгоритъма е $O\big(V \lg V + E \lg V\big) = O\big(E \lg V\big)$

- Ако се използва пирамида на Фибоначи, то Extract-Min отнема $O\big(\lg V\big)$ амортизирано, а Decrease-Key отнема $O(1)$ амортизирано време. Времевата сложност за изпълнение на целия алгоритъм е $O\big(E + V \lg V\big)$

4. **Алгоритъм на Крускал** - построява МПД
Алгоритъмът на Крускал е алчен алгоритъм, който избира ребро, което да добави към растящата гора, като избира най-лекото ребро от тези, които свързват две дървета в гората.

Използва се абстрактна структура Union-Find, която поддържа гора от дървета и операциите:

- Make-Set - всеки връх е отделно дърво
- Union - слива две дървета (примерна реализация - пренасочва указателя на единия корен към другия, пази се височината на дърветата и насочваме по-малкото към по-голямото)
- Find-Set - връща корена на дървото, в което се намира елементът

MST_Kruskal(G(V,E), c)

A = ∅ // O(1)
for each $v \in V$ **do**: // |V| times
    Make-Set(v)
Sort(E), подредени в ненамаляващ ред спрямо c(e) // O(E.lgE)
for $(u, v) \in E$ **do** // O(E) times
    if $\mathit{Find\text{-}Set}(u) \neq \mathit{Find\text{-}Set}(v)$
        $\mathrm{A} = \mathrm{A} \cup \{(u, v)\}$
        Union(u, v)
return A

Времевата сложност зависи от конкретната реализация на структурата Union-Find. Ще разгледаме сложността, ако се използва примерната реализация, като се приложи union-by-rank и евристика за компресия на пътя (асимптотично най-бързата известна имплементация).
За сортирането: $E \log E \ \asymp E \log V$.
Общата сложност на операциите, свързани със структурата е $O\big((V + E)\alpha(V)\big)$, където $\alpha(|V|) = O\big(\lg V\big) = O\big(lgE\big)$ - пренебрежимо малка (за практически цели ≤ 5).

Така общата сложност на алгоритъма е $O\big(E \lg E\big)$, но $|E| < |V|^2$, то $\lg|E| = O\ (\lg V)$, така общата сложност е $O(E \lg V)$.

5. **Задача за най-къс път в граф с тегла на ребрата**
Нека $G = (V, E)$ е свързан граф, а $c: E \to \mathbb{R}^+$ е теглова(ценова) функция на ребрата с положителни стойности.

Деф:

Нека p= $(u_0, u_1, \dots, u_k)$ е път от u до v в G. $c\big(p\big) = \sum_{i=1}^{k} c\big(u_{i-1}, u_i\big)$ - *тежест (цена) на пътя* p
Най-къс път от u до v в G е пътят от u до v с най-малко тегло.

Задача: Да се намерят най-късите пътища и теглата им от зададен връх s до всички осттъанали върхове в графа G*.*

6. **Алгоритъм за намиране на дърво на най-къси пътища в граф с константни тегла по ребрата**
Ако тегловата функция е константа, то задачата се свежда до търсене на най-къс път в граф от даден връх до всички останали. Можем да я решим като построим покриващо дърво чрез обхождане в ширина с начален връх - даденият. Сложността на това решение е $O(V + E)$.

7. Алгоритъм на Дейкстра

Алгоритъмът на Дейкстра решава задачата за намиране на най-къси пътища от връх в ориентиран граф с тегла на ребрата $G = (V, E)$, в случая, когато всички тежести са неотрицателни т.е. $c(u, v) \geq 0$ $\forall (u, v) \in E$.

Алгоритъмът поддържа множество от върхове, чиито най-къс път от началният s вече е намерен.

```
Dijkstra(G=(V,E), c, s)
for each v ∈ V do                    // |V| times
    dist(v) = ∞
    parent(v) = NIL
dist(s) = 0
Q = make-queue(V)

while Q ≠ ∅                          // |V| times
    u = Extract-Min(Q)
    S = S ∪ {u}
    for each v ∈ G.Adj[u] do         // In total |E| times
        if dist(v) > dist(u) + c(u,v)
            dist(v) = dist(u) + c(u,v)
            parent(v) = u
            Decrease-Key(Q, v)
```

Сложността зависи от имплементацията на опашката:

- Ако се използва приоритетна опашка, като номерираме върховете от 1 до $|V|$, запазим в масив стойностите на $dist(v)$, то записванията и $Decrease\text{-}Key$ ще бъдат със сложност $O(1)$, а $Extract\text{-}Min$ ще е $O(V)$. Общото време е: $O(V^2 + E) = O(V^2)$
- Ако графът е достатъчно рядък ($E = o(V^2/\ \lg V)$), може да използваме binary heap. Така $Extract\text{-}Min$ и $Decrease\text{-}Key$ имат сложност $O(\lg V)$, сложността за инициализиране на binary heap е O(V). Сложността на алгоритъма е $O((V + E)\lg V)$, което е $O(E \lg V)$, ако всички върхове са достижими от източника .
- Ако се използва Фибоначи heap, амортизираната цена на всяка $Extract\text{-}Min$ операция е $O(\lg V)$. Сложността на $Decrease\text{-}Key$ е амортизирано $O(1)$.
- Така сложността на алгоритъма е $O(V \lg V + E)$

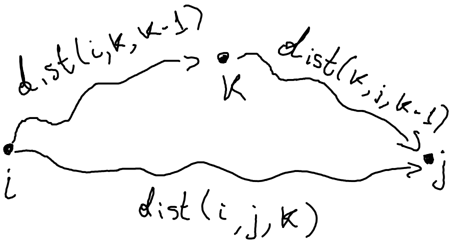
8. Алгоритъм на Флойд за намиране на всички двойки най-кратки пътища

Алгоритъмът на Флойд-Уоршел решава задачата за намиране на всички двойки най-кратки пътища в ориентиран граф $G = (V, E)$. Може да има отрицателни тежести, но предполагаме, че няма отрицателни цикли. Алгоритъмът използва динамично програмиране.
Алгоритъмът разглежда междинните върхове на най-късия път.

$a_{ij}^{(k)}$ – дължина на най-къс път от връх i до връх j, където всички междинни върхове са $\leq k$.
(динамично програмиране по k)

$$a_{ij}^{(0)} = \begin{cases} c(i,j), & \text{ако } \exists \text{ребро между } i \text{ и } j \\ +\infty, & \text{ако не } \exists \text{ ребро между } i \text{ и } j \\ 0, & \text{ако } i = j \end{cases}$$

$$a_{ij}^{(k)} = \min\left\{a_{ij}^{(k-1)}, a_{ik}^{(k-1)} + a_{kj}^{(k-1)}\right\}, \qquad 1 \leq \mathrm{k} \leq \mathrm{n}, \qquad (\mathrm{i} \to \cdots \to \mathrm{k} \to \cdots \to \mathrm{j}) \text{ – прост път}$$



Floyd-Warshall(G=(V,E), c)

1. **for** $i \in \{1, \dots, n\}$ **do**
2.   for $j \in \{1, \dots n\}$ do
3. $dist(i, j, 0) = \infty$
4.
5. for all $(i, j) \in E$
6. $dist(i, j, 0) = c(i, j)$
7. for $k \in \{1, \dots, n\}$ do
8.   for $i \in \{1, \dots, n\}$ do
9.   for $j \in \{1, \dots, n\}$ do
10. $dist(i, j, k) = \min\{dist(i, j, k-1),\ dist(i, k, k-1) + dist(k, j, k-1)\}$

Има три вложени цикъла, а сложността на ред 10. е $O(1)$, така сложността на алгоритъма на Флойд е $\Theta(n^3)$.