Глава 7

Алгоритми за графи

7.1 Обхождане на граф

Нека е даден ориентирания граф $G = (V, E)$. При обхождането на граф, всеки връх може да бъде в едно от три състояния, или както сме ги означили тук, в един от три цвята. Ако един връх v е

- бял, то той още не е срещнат.
- сив, то той е срещнат, но още не е напълно обработен.
- черен, то той е напълно обработен.

7.1.1 Обхождане в широчина

Algorithm 1 Инициализация

```
1:  procedure BFS-INIT(s)
2:      for all v ∈ V \ {s} do
3:          color[v] := WHITE
4:          dist[v] := ∞
5:          pred[v] := NIL
6:      end for
7:      color[s] := GRAY
8:      dist[s] := 0
9:      pred[s] := NIL
10: end procedure
```

За този алгоритъм най-удобно е да имаме масив Adj с дължина $|V|$, като $Adj[u]$ дава списък с наследниците на u, т.е.

$$Adj[u] = \{u \in V \mid (u, v) \in E\}.$$

Algorithm 2 Алгоритъм за обхождането на широчина

```
1:  procedure BFS(s)
2:      BFS-INIT(s)
3:      Q := ∅                          ▷ Опашката Q съдържа точно сивите върхове
4:      put(Q,s)
5:      while Q ≠ ∅ do
6:          u := get(Q)                          ▷ u е премахнат от опашката
7:          for all v ∈ Adj[u] do
8:              if WHITE = color[v] then
9:                  put(Q, v)
10:                 color[v] := GRAY
11:                 pred[v] := u
12:                 dist[v] := dist[u] + 1
13:             end if
14:         end for
15:         color[u] := BLACK
16:     end while
17: end procedure
```

- Алгоритъмът работи както за ориентирани, така и за неориентирани графи.
- Дължина на път (без цикли) е броят на ребрата, които участват в пътя. Например, за пътя $p = (v_0, \ldots, v_k)$ в графа G, неговата дължина е k, защото ребрата, които участват в p са $\{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\}$ и са общо k на брой. Обикновено ще означаваме дължината на пътя p с $|p|$.
- Нека да означим за всеки два върха $u, v \in V$,

$$\delta(u, v) = \begin{cases} \min\{|p| \mid u \overset{p}{\rightsquigarrow} v\}, \text{ ако има път между } u, v \\ \infty, \text{ ако няма път} \end{cases}$$

- Имаме свойството, че за (не)ориентиран граф $G = (V, E)$ и един връх $s \in V$, ако $(u, v) \in E$, то

$$\delta(s, v) \leq \delta(s, u) + 1.$$

- За (не)ориентиран граф $G = (V, E)$, и фиксиран връх $s \in V$, означаваме $G_{pred} = (V_{pred}, E_{pred})$ да бъде

  $V_{pred} = \{v \in V \mid pred[v] \neq NIL\} \cup \{s\}, E_{pred} \quad = \{(u, v) \in E \mid pred[v] = u\}.$

  След изпълнение на BFS(s), G_{pred} представлява дърво с корен s, за всеки достижим в G от s връх v, G_{pred} съдържа единствен прост път $s \overset{p}{\rightsquigarrow} v$, като p е най-къс измежду всички пътища свързващи s с v в G.

Теорема 8. Нека е даден (не)ориентиран граф $G = (V, E)$ и един връх $s \in V$. След изпълнение на BFS(s) получаваме, че

$$(\forall v \in V)[dist[v] = \delta(s, v)].$$

7.1.2 Обхождане в дълбочина

Algorithm 3 Обхождане в дълбочина

```
1: procedure DFS-VISIT(u)
2:     color[u] := GRAY                ▷ Върхът u е посетен, но не е обработен
3:     for all v ∈ Adj[u] do
4:         if WHITE = color[v] then
5:             pred[v] := u
6:             DFS-VISIT(v)
7:         end if
8:     end for
9:     color[u] := BLACK                          ▷ Приключили сме с u
10: end procedure

11: procedure DFS
12:     for all v ∈ V do                                ▷ Инициализация
13:         color[v] := WHITE
14:         pred[v] := NIL
15:     end for
16:     for all v ∈ V do
17:         if WHITE = color[v] then
18:             DFS-VISIT(v)
19:         end if
20:     end for
21: end procedure
```

7.2 Минимално покриващо дърво на граф

- Тук ще разглеждаме само **неориентирани** графи $G = (V, E, w)$ с тегла по ребрата зададени с функцията $w : E \to \mathbb{R}$.
- Един граф $G = (V, E)$ се нарича **свързан**, ако има път между всеки два $v, v' \in V$.
- Един неориентиран граф G се нарича **дърво**, ако G е свързан и без цикли.
- **Покриващо дърво** за свързан неориентиран граф $G = (V, E)$, е дърво $T = (V, E')$, $E' \subseteq E$.
- Тегло на едно подмножество от $U \subseteq E$ е числото

$$w(U) = \sum_{e \in U} w(e).$$

- **Минимално покриващо дърво** свързан неориентиран граф $G = (V, E, w)$ е покриващо дърво T, за което

$$w(T) = \min\{w(T') \mid T' \text{ е покриващо дърво за } G\}.$$

7.2.1 Алгоритъм на Прим

Нека е даден неориентиран граф $G = (V, E, w)$ с тегла по ребрата.

Algorithm 4 Алгоритъм на Прим

```
1:  procedure PRIM(r)
2:      U := {r}                          ▷ Започваме от дърво с корен r и без ребра
3:      S := ∅
4:      while (∃(x, y) ∈ E)[x ∈ U & y ∈ V \ U] do
5:          Избираме (u, v) ∈ E, за което
6:          w(u, v) = min{w(x, y) | x ∈ U & y ∈ V \ U & (x, y) ∈ E}
7:          U := U ∪ {v}
8:          S := S ∪ {(u, v)}
9:      end while
10:     return (U, S)                     ▷ Връщаме като резултат полученото дърво
11: end procedure
```

7.2.2 Алгоритъм на Крускал

Нека е даден неориентиран **свързан** претеглен граф $G = (V, E, w)$.

Algorithm 5 Алгоритъм на Крускал

```
1:  procedure KRUSKAL
2:      X = ∅                             ▷ X ще бъде списък с дървета
3:      for v ∈ V do
4:          Добавяме дървото T = ({v}, ∅) към списъка X
5:      end for
6:      E' :=Sort(E,w) ▷ Сортираме E във възходящ ред относно тегла им

7:      for all (u, v) ∈ E' do
8:          Нека u принадлежи на върховете на дървото T_u ∈ X
9:          Нека v принадлежи на върховете на дървото T_v ∈ X
10:         if T_u ≠ T_v then
11:             V := V_u ∪ V_v
12:             E := E_u ∪ E_v ∪ {(u, v)}
13:             T := (V, E)
14:             Премахваме T_u и T_v от списъка X
15:             Добавяме дървото T към X
16:         end if
17:     end for
18:     return единственото дърво останало в X
19: end procedure
```

7.3 Минимални пътища от даден връх

- С $u \overset{p}{\rightsquigarrow} v$ означаваме, че p е път от u до v.

- Тук ще разглеждаме **ориентирани** графи $G = (V, E)$, като имаме и функция $w : E \to \mathbb{R}$, която задава **тегла** на ребрата на графа.
- **Цена на път** $p = (v_0, \ldots, v_k)$ в графа означаваме

$$w(p) = \sum_{i<k} w(v_i, v_{i+1}).$$

- За всеки два върха $u, v \in V$, означаваме

$$\delta(u,v) = \begin{cases} \min\{w(p) \mid u \overset{p}{\rightsquigarrow} v\}, \text{ ако има път от } u \text{ до } v \\ \infty, \text{ иначе} \end{cases}$$

- **Минимален път** p от u до v е такъв път, за който $w(p) = \delta(u, v)$.
- Имаме следното важно свойство. Нека $u \overset{p}{\rightsquigarrow} v$ и p е **минимален път**. Да означим $p = (v_0, \ldots, v_k)$ и $p_{ij} = (v_i, \ldots, v_j)$ за $0 \leq i \leq j \leq k$. Тогава за всеки $0 \leq i \leq j \leq k$, p_{ij} е **минимален път** от v_i до v_j.
- **Цикъл** е път $p = (v_0, \ldots, v_k)$, където $v_0 = v_k$.
- Също така казваме, че по пътя $p = (v_0, \ldots, v_k)$ има цикъл, ако за някои $0 \leq i < j \leq k$ имаме, че $v_i = v_j$.
- Ако има цикъл с отрицателно тегло по някой път от u до v, то тогава пишем, че $\delta(u, v) = -\infty$.
- Нека $u \overset{p}{\rightsquigarrow} v$ и p е с минимално тегло. Тогава няма цикъл с положително тегло по p.
- Нека $u \overset{p}{\rightsquigarrow} v$ и p е с минимално тегло. Можем без ограничение на общността да приемем, че няма цикли с нулево тегло по p.
- Важно свойство е, че броят на върховете по всички минални пътища е $\leq |V|$.

Нека да фиксираме един връх $s \in V$. Нашата цел е да намерим минимални пътища от s до всички достижими от s върхове, както и тяхната цена. Да отбележим, че ако имаме отрицателен по някой път $s \rightsquigarrow v$, то задачата не е добре дефинирана, защото тогава $\delta(s, v) = -\infty$.

За тази цел въвеждаме два масива, *dist* и *pred*, с дължина $|V|$.

- $dist[v]$ - дава цена на минимален път от s до v. Ако $dist[v] = \infty$, то не е намерен път $s \rightsquigarrow v$.
- $pred[v]$ - дава предшественика на v по този минимален път, т.е. ако $pred[v] = u$, то $s \rightsquigarrow u \to v$. Ако $pred[v] = NIL$, то не е намерен път $s \rightsquigarrow v$.

Algorithm 6 Инициализация

```
1: procedure INIT(s)
2:     for all v ∈ V do
3:         dist[v] := ∞
4:         pred[v] := NIL
5:     end for
6:     dist[s] := 0
7: end procedure
```

Algorithm 7 Търсене на по-добър кандидат

```
1: procedure UPDATE(u,v)
2:     if dist[v] > dist[u] + w(u, v) then
3:         dist[v] := dist[u] + w(u, v)
4:         pred[v] := u
5:     end if
6: end procedure
```

7.3.1 Алгоритъм на Дейкстра

В този алгоритъм, разглеждаме ориентирани графи $G = (V, E, w)$ с *положителни* тегла (или цени) по ребрата, т.е. имаме функция $w : E \to \mathbb{R}^+$.

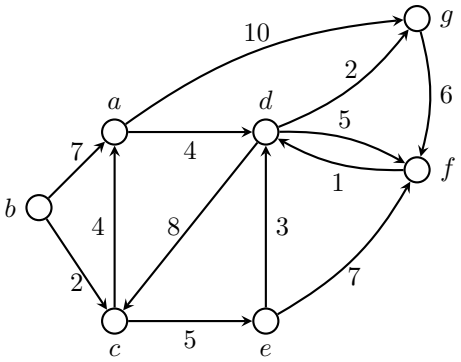
Algorithm 8 Алгоритъм на Дейкстра

```
Require: w : E → ℝ⁺
1: procedure DIJKSTRA(s)
2:     INIT(s)
3:     V' := V
4:     while V' ≠ ∅ do
5:         Избираме u₀ ∈ V', за който dist[u₀] = min{dist[v] | v ∈ V'}
6:         V' := V' \ {u₀}
7:         for all v ∈ Adj[u₀] do
8:             UPDATE(u₀,v)
9:         end for
10:    end while
11: end procedure
```

Ако във V' има останали върхове v, то те имат $\delta(v) = \infty$, т.е. те са недостижими от s и следователно пътят от s до v има дължина ∞.

Фигура 7.1 илюстрира как се променя функцията δ по време на изпълнението на алгоритъма. Освен това, можем да намерим не само стойността на най-късите пътища, но и списък с ребрата, които участват във всеки от тях.

Ще завършим с един прост пример, който показва, че алгоритъмът на Дейкстра не работи при наличието на ребра с отрицателни тегла.

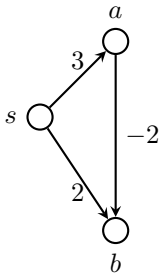


(а) Пример за насочен граф с тегла по ребрата

a	b	c	d	e	f	g
∞	**0**	∞	∞	∞	∞	∞
7	:	**2**	∞	∞	∞	∞
6	:	:	∞	7	∞	∞
:	:	:	10	**7**	∞	16
:	:	:	**10**	:	14	**12**
:	:	:	:	:	14	**12**
:	:	:	:	:	**14**	:
:	:	:	:	:	:	:

(б) Масива *dist* с тегла на пътища с начален връх b

Фигура 7.1: Алгоритъм на Дейкстра



Фигура 7.2: Пример, за който алгоритъмът на Дейкстра не дава верен резултат