ДАА Семинар 12

Алгоритми върху графи

Деф. Тегловен граф

Тегловен граф е наредена двойка (G, w), където $G=(V,E)$ е граф, а $w: E \to \mathbb{R}$ е тегловна/ценова функция

Деф. Тегло на ПД/Минимално покриващо дърво

Нека G е свързан тегловен граф с тегловна функция w и T е ПД на G

Теглото на T се дефинира: $w(T) := \sum_{e \in E(T)} w(e)$

Минимално покриващо дърво T_{min} на G (МПД) е такова ПД, за което е в сила $\forall T \in \{T \mid T \text{ е ПД на } G\}: w(T_{min}) \leq w(T)$

I) Алгоритми за намиране на МПД

1) Алгоритъм на Prim

Класифицираме върховете в следните категории

1) $V(T)$ - дървесни

2) $N(V(T))$ - гранични

3) $V \setminus N[V(T)]$ - неизвестни

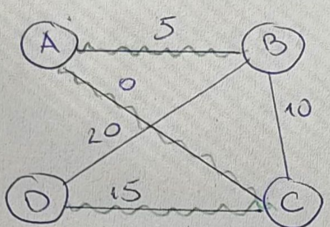
- започва с избран стартов връх и на всяка итерация добавя "най-леко" ребро, единият край на което е дървесен, а другият - граничен

PrimPrQ($G=(V,E)$ неор. св. граф, w-тегл. ф-я на G, s)

```
for v ← 1 to N
    v.key ← ∞
    π[v] ← Nil
vs.key ← 0
create min priority queue Q
Q ← V
While !Q.empty()
    x ← Q.pop() // изкарва върха с мин. ключ
    for y ∈ adj[x]
        if y ∈ Q && w((x,y)) < y.key
            y.key ← w((x,y))
            π[y] ← x
return π[1..n]
```

зад① Дадени са n града и m пътища между тях. Състоянието на някои от пътищата е толкова лошо, че не могат да бъдат използвани. За всеки път е дадена цената за поправката му.
Трябва да се поправят някои от пътищата по такъв начин, че общата цена за поправка да е минимална и след ремонта да има използваем път между всеки два града (не непременно непосредствен).
Градовете, пътищата и цената за поправка са подадени като тегловен граф, представен чрез списъци на съседство.

// Пример: Градовете са 4: A, B, C, D и графът е следният:



↳ най-изгодно е да се оправят пътища (A, C), (A B), (D, C) с общ разход 20.

Решение: Търси се МПД на подадения граф, както и неговата цена.
Ще използваме алгоритъма на Prim, реализиран чрез приоритетна опашка.

```
MSTPrim(G=(V,E), w, s)
for v ← 1 to n
        v.key ← ∞
        π[v] ← Nil
sum ← 0
s.key ← 0
create priority queue Q (min)
Q ← V
while !Q.empty()
        x ← Q.pop  //min
        for y ∈ adj[x]
                if y ∈ Q && w((x,y)) < y.key
                        if y.key != ∞
                                sum ← sum − y.key
                        y.key ← w((x,y))
                        sum ← sum + w((x,y))
                        π[y] ← x
return (sum, π)
```

2) Алгоритъм на Kruskal

11) Алгоритми за намиране на най-къси/леки пътища

Разглеждаме ориентирани теглови графи

Деф. Тегло на път

Тегло на път p е $w(p) := \sum_{e \in E(p)} w(e)$

Наблюдение

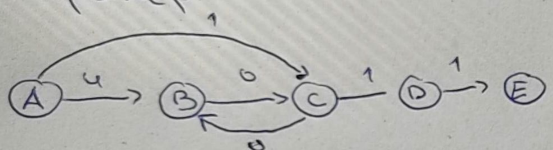
При липса на отрицателни цикли, за всеки два върха u и v, всеки най-къс/лек път от u до v задължително е прост.

Деф. Тегло на най-къс път от u до v

Тегло на най-къс път от u до v е $\delta(u,v) := \begin{cases} \min\{w(p) \mid u \overset{p}{\leadsto} v\}, & \text{ако такъв } \exists \\ \infty, & \text{иначе} \end{cases}$

Th. Най-къс път се състои от най-къси пътища

// Пример



Разглеждаме пътя:

$A \xrightarrow{4} B \xrightarrow{0} C \xrightarrow{1} D \xrightarrow{1} E$ (*) от A до E

с цена 5

1) $A \xrightarrow{4} B$ не е най-лекият/късият път от A до B

& 2) $A \xrightarrow{1} C \xrightarrow{0} B$ е по-къс/лек път

Заменяме 2) с 1) в (*):

$A \xrightarrow{1} C \xrightarrow{0} B \xrightarrow{0} C \xrightarrow{1} D \xrightarrow{1} E$ с цена 3

Новополученият път не е прост. "Изрязваме" повторенията:

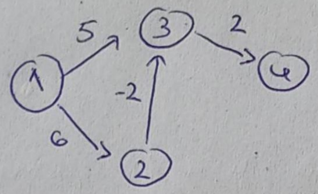
$A \xrightarrow{1} C \xrightarrow{1} D \xrightarrow{1} E$ - най-къс прост път от A до E с цена 3.

1) Алгоритъм на Dijkstra

```
SSShPDijkstra(G=(V,E), w, s-стартов)
for v <- 1 to n
        v.d <- ∞
        v.π <- Nil
s.d <- 0
S <- ∅
create priority min queue Q
Q <- V
while !Q.empty()
        x <- Q.pop() //min
        S <- S ∪ {x}
        for y ∈ adj[x]
            if y.d > x.d + w((x,y))
                y.d <- x.d + w((x,y))
                y.π <- x
```

Алгоритъмът на Dijkstra НЕ работи коректно при наличие на отрицателни тегла

↳ //пример   със старт в връх 1



зад 2 - от изпит 2022г.

Да се докаже/опровергае, че алгоритъмът на Dijkstra остава коректен, ако се имплементира с обикновена опашка (FIFO)

↳ следвайки псевдокода от лекции

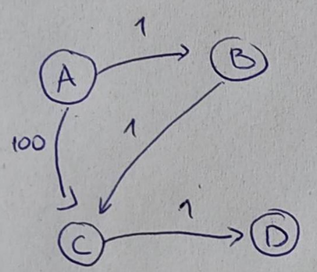
Решение:

Същината на алгоритъма се „разваля", ако се имплементира с обикновена опашка. - алгоритъмът престава да е коректен.

Контрапример:

$G = (V, E)$, където $V = \{A, B, C, D\}$, $E = \{(A,B), (A,C), (B,C), (C,D)\}$, и $w = \{((A,B),1), ((A,C),100), ((B,C),1), ((C,D),1)\}$

Графът изглежда по следния начин:

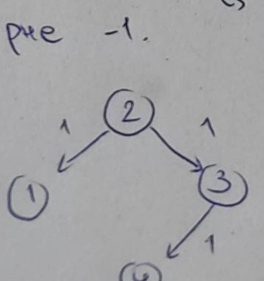


В края на алгоритъма D ще има d-стойност 101, вместо 3

зад 3 Дадена е мрежа, състояща се от N възела, именовани от 1 до N. Дадена е и времева функция - time, която указва времето за достигане на сигнал от u до v, където u,v са възли в мрежата, както и начален възел K, от който ще се пусне сигнал. Да се върне минималното време, което ще отнеме на сигнала да достигне до всичките N възела. Ако последното не е възможно, да се върне -1.

↳ сигналът „пътува" паралелно

//Пример



→ отговорът е 2.

Решение: Търсим максималната стойност от всички минимални разстояния с начало - k - стартовия възел.

Псевдокод:

```
Network(G=(V,E), w, k)
for v←1 to n
      v.d ← ∞
k.d ← 0
create priority queue Q
Q ← V
while !Q.empty()
      x ← Q.pop()
      for y∈adj[x]
            if y.d > x.d + w((x,y))
                  y.d ← x.d + w((x,y))
result ← max(v1.d, ..., vn.d)
if result < ∞
      return result
else
      return -1.
```

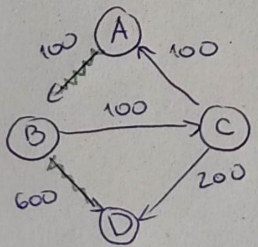
зад 4 Най-евтин полет с най-много К прекачвания

Дадени са N града и M ребра/маршрути (въздушни), които свързват някои от градовете непосредствено. За всеки два града A и B, за които съществува маршрут, е дадена цената на прекия полет от A до B.

При подадени начален и краен град да се намери най-евтин полет от началния до крайния град с най-много K прекачвания

// Пример:

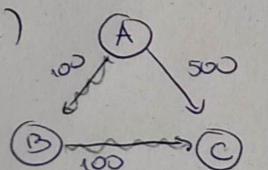
1)



като началният град е A, крайният е D, а k е 1

→ отговор: 700 с едно прекачване в град B

2)



от A до C, където k=1

→ отговор: 200 с прекачване в град B,

ако k е 0, в примера, отговорът ще е 500

Решение: Ще използваме алгоритъма на Bellman-Ford, като, вместо n-1 пъти, ще „релаксираме" ребрата точно K+1 пъти, с което ще намерим най-късите/леките пътища от стартовия връх до всички останали, достижими през най-много $k+1$ ребра.

"Опростен" алгоритъм на Bellman-Ford

```
Bellman-Ford(G, w, s)
for v ← 1 to n
    v.d ← ∞
s.d ← 0
for i ← 1 to m-1
    foreach (x,y) ∈ E(G)
        Relax((x,y)) // if y.d > x.d + w((x,y))
                          y.d ← x.d + w((x,y))
```

Нека v е достижим връх от s и нека $p = (v_0, v_1, \dots, v_t)$, където $v_0 = s$ и $v_t = v$, е най-къс път от s до v. Тъй като най-късите пътища задължително са прости (при липса на отрицателни цикли), то $t \leq n-1$. На всяка от n-те итерации се „релаксират" всичките m ребра. На i-тата итерация, сред релаксираните ребра е и реброто (v_{i-1}, v_i). В контекста на нашата задача търсим най-къс/лек път с най-много $k+1$ ребра - достатъчно е да „релаксираме" всички ребра $k+1$ пъти. // може да има и по-къс/лек път, използвайки повече ребра

```
CheapestFlight(G=(V,E), s, dest, k)
for v ← 1 to n
    v.d ← ∞
s.d ← 0
for i ← 1 to k+1
    foreach (x,y) ∈ E(G)
        if y.d > x.d + w((x,y))
            y.d ← x.d + w((x,y))
if dest.d = ∞
    return -1
return dest.d
```