

Számítógépes Hálózatok

6. gyakorlat

FORGALOMIRÁNYÍTÁS

Dijkstra algoritmus

Óra eleji kisZH

- Elérés:
 - <https://canvas.elte.hu>

☰ 2018/19/1 XK85DZ-IP-08abcSZHG - Számítógépes hálózatok GY. (BSc,08,A) > Kvízek

2018/19/1

Kezdőlap

Hirdetmények

Feladatok

Fórumok

Értékelések

Résztvevők

Oldalak

Fájlok

Tematika

Tanulási eredmények

Kvízek

Modulok


Beállítások

Kvíz keresése

+ Kvíz

⚙️

▼ Gyakorló kvízek



Demo kvíz

Elérhető **Többes határidő** | **Határidő Többes határidő** | 5 pont | 5 kérdés

✓

⚙️

Számítógépes Hálózatok Gyakorlat 6.

3

Forgalomirányítás

- **Definíció:** a hálózati réteg szoftverének azon része, amely azért a döntésért felelős, hogy a bejövő csomag melyik kimeneti vonalon kerüljön továbbításra.

Dijkstra algoritmus

(Ács Zoltán és Laki Sándor korábbi diásorai alapján)

- Statikus (nem adaptív) forgalomirányító algoritmus
- Az eredeti algoritmus célja: két csúcs közti legrövidebb út meghatározása.
- Leggyakrabban használt változatban: egyik csúcsot kinevezzük forrás csúcsnak, és onnan a legrövidebb utakat kiszámítjuk az összes többi csúcshoz.
- Egy v csúcsnak kétféle címkéje lehet: ideiglenes ($ready[v] = false$) vagy állandó ($ready[v] = true$). Kezdetben csak a forrás csúcs állandó, a többi ideiglenes.
- A legrövidebb út megtalálásakor a címke állandó címkévé válik, és továbbá nem változik.
- Van még két segédhalmazunk: E' és Q . Az E' -ben az állandó címkéjűeket tároljuk.

Dijkstra algoritmus

(Ács Zoltán és Laki Sándor korábbi diásorai alapján)

- Q -ban kezdetben a forrás csúcs szomszédjai szerepelnek az odavezető élsúlyoknak megfelelő prioritással (a legkisebb prioritás van Q legtetején)
- Amíg Q ki nem ürül, addig mindig levesszük a tetejéről a csúcsot (a legkisebb költségűt), beletesszük E' -be, és állandó címkét kap. Jelöljük ezt a csúcsot v -vel.
- Vesszük a szomszédjait:
 - Ha egy u szomszédja még nincs Q -ban \rightarrow betesszük, prioritása a v -hez vezető legrövidebb út hosszának és v, u közti élsúlynak az összege lesz
 - Ha benne van, de kisebb költségű úttal is elérhető \rightarrow csökkentjük a prioritását

Dijkstra algoritmus pszeudokódja

(Ács Zoltán és Laki Sándor korábbi diásorai alapján)

Dijkstra(G,s,w)

Output: egy legrövidebb utak fája $T=(V,E')$ G-ben s gyökérrel

```
01  $E' := \emptyset$ ;  
02 ready[s] := true;  
03 ready[v] := false;  $\forall v \in V \setminus \{s\}$ ;  
04 d[s] := 0;  
05 d[v] :=  $\infty$ ;  $\forall v \in V \setminus \{s\}$ ;  
06 priority_queue Q;  
07 forall v  $\in$  Adj[s] do  
08   pred[v] := s;  
09   d[v] := w(s,v);  
10   Q.Insert(v,d[v]);  
11 od  
12 while Q  $\neq \emptyset$  do
```

INICIALIZÁCIÓS FÁZIS

```
13   v := Q.DeleteMin();  
14    $E' := E' \cup \{(pred[v],v)\}$ ;  
15   ready[v] := true;  
16   forall u  $\in$  Adj[v] do  
17     if u  $\in$  Q and d[v] + w(v,u) < d[u] then  
18       pred[u] := v;  
19       d[u] := d[v] + w(v,u);  
20       Q.DecreasePriority(u,d[u]);  
21     else if u  $\notin$  Q and not ready[u] then  
22       pred[u] := v;  
23       d[u] := d[v] + w(v,u);  
24       Q.Insert(u,d[u]);  
25   fi  
26   od  
27 od
```

ITERÁCIÓS LÉPÉSEK

JAVÍTÓ ÚT

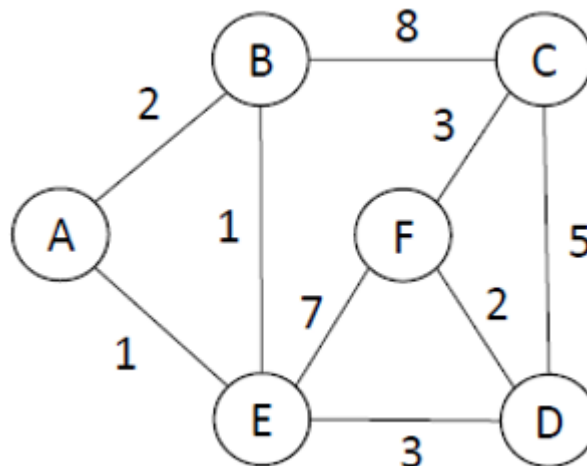
ÚJ ÚT

Feladat 1

Tekintsük a $G = (V;E)$ gráfot az alábbi ábrán egy hálózat reprezentánsának.

Számítsuk ki Dijkstra algoritmusával egy legrövidebb utak fáját D csomópontból minden más csomóponthoz a pseudokód segítségével. Minden iterációnál jelöljük E' halmaz aktuális értékét és adjuk meg minden $u \in V$ csomóponthoz a $d[u]$ értéket egy táblázatban.

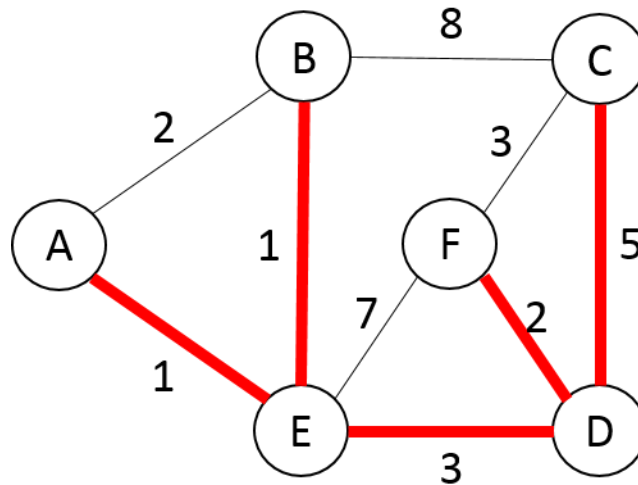
Ezt követően rajzoljuk fel a kiszámított legrövidebb utak fáját.



Feladat 1 megoldása

Iteráció	E' csúcshalmaz	d(.)					
		A	B	C	D	E	F
1	(D,-)	∞	∞	5	0	3	2
2	(D,-),(F,D)	∞	∞	5	0	3	2
3	(D,-),(F,D),(E,D)	4	4	5	0	3	2
4	(D,-),(F,D),(E,D),(A,E)	4	4	5	0	3	2
5	(D,-),(F,D),(E,D),(A,E),(B,E)	4	4	5	0	3	2
6	(D,-),(F,D),(E,D),(A,E),(B,E),(C,D)	4	4	5	0	3	2

Feladat 1 megoldása



Órai feladat (4 pont)

- Adott a graph.json, ami tartalmazza az irányítatlan gráf leírását. A „start-point” a forrás csúcs. A „links” lista tartalmazza az éleket (az él két végpontját: „points”, és az él súlyát: „weight”).
- Készíts egy alkalmazást, amely megvalósítja a Dijkstra algoritmusát a **korábbi dián szereplő pseudokód alapján!**
- Lényegében a Feladat 1 megoldásában látott táblázatot kell kiírni a megoldásnál.
- Az interneten temérdek Python megoldás van fent a Dijkstra algoritmusára. Ne használd! **A „copy-paste” megoldások 0 pontot érnek!**

Órai feladathoz segítség

- Feltettem a forráskódok közé egy heapdict.py fájlt
<https://github.com/DanielStutzbach/heapdict>
- Ez egy prioritásos sor (priority queue) implementáció, amelyben a prioritást lehet csökkenteni/növelni
- Ha a szkripted mellett van, akkor az alábbi paranccsal lehet importálni:

```
from heapdict import heapdict
```

- Az alábbi parancsokkal lehet használni:

```
Q = heapdict()
```

```
Q["A"]=3
```

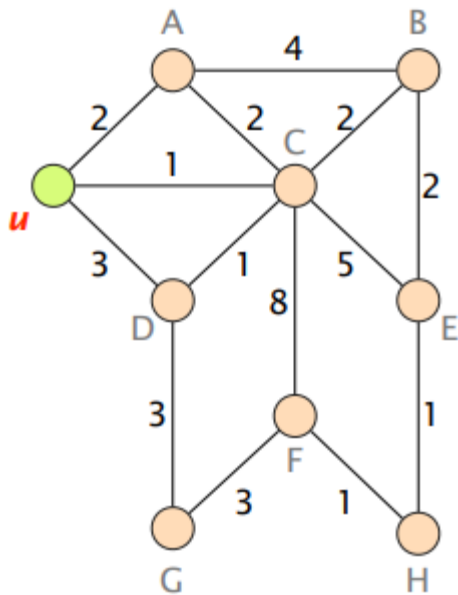
```
Q["A"]=2
```

```
Q.popitem()
```

- Az alábbival pedig azt lehet leellenőrizni, hogy üres-e:

```
if len(Q) > 0:
```

```
...
```



Prof. Laurent Vanbever feladatából

```
{
  "start-point": "u",
  "links": [
    {
      "points": [
        "u",
        "A"
      ],
      "weight": 2
    },
    {
      "points": [
        "u",
        "C"
      ],
      "weight": 1
    },
    {
      "points": [
        "u",
        "D"
      ],
      "weight": 3
    },
    {
      "points": [
        "A",
        "B"
      ],
      "weight": 4
    },
    {
      "points": [
        "A",
        "C"
      ],
      "weight": 2
    },
    {
      "points": [
        "B",
        "C"
      ],
      "weight": 2
    },
    {
      "points": [
        "B",
        "E"
      ],
      "weight": 2
    },
    {
      "points": [
        "C",
        "D"
      ],
      "weight": 1
    },
    {
      "points": [
        "C",
        "E"
      ],
      "weight": 5
    },
    {
      "points": [
        "C",
        "F"
      ],
      "weight": 8
    },
    {
      "points": [
        "D",
        "G"
      ],
      "weight": 3
    },
    {
      "points": [
        "E",
        "H"
      ],
      "weight": 1
    },
    {
      "points": [
        "F",
        "G"
      ],
      "weight": 3
    },
    {
      "points": [
        "F",
        "H"
      ],
      "weight": 1
    }
  ]
}
```

graph.json

```

Iteration: 1
E: [(u'u', u'C')]
[d(u):0', 'd(A):2', 'd(B):2147483647', 'd(C):1', 'd(D):3', 'd(E):2147483647', 'd(F):2147483647', 'd(G):2147483647', 'd(H):2147483647']

Iteration: 2
E: [(u'u', u'C'), (u'C', u'D')]
[d(u):0', 'd(A):2', 'd(B):3', 'd(C):1', 'd(D):2', 'd(E):6', 'd(F):9', 'd(G):2147483647', 'd(H):2147483647']

Iteration: 3
E: [(u'u', u'C'), (u'C', u'D'), (u'u', u'A')]
[d(u):0', 'd(A):2', 'd(B):3', 'd(C):1', 'd(D):2', 'd(E):6', 'd(F):9', 'd(G):5', 'd(H):2147483647']

Iteration: 4
E: [(u'u', u'C'), (u'C', u'D'), (u'u', u'A'), (u'C', u'B')]
[d(u):0', 'd(A):2', 'd(B):3', 'd(C):1', 'd(D):2', 'd(E):6', 'd(F):9', 'd(G):5', 'd(H):2147483647']

Iteration: 5
E: [(u'u', u'C'), (u'C', u'D'), (u'u', u'A'), (u'C', u'B'), (u'B', u'E')]
[d(u):0', 'd(A):2', 'd(B):3', 'd(C):1', 'd(D):2', 'd(E):5', 'd(F):9', 'd(G):5', 'd(H):2147483647']

Iteration: 6
E: [(u'u', u'C'), (u'C', u'D'), (u'u', u'A'), (u'C', u'B'), (u'B', u'E'), (u'D', u'G')]
[d(u):0', 'd(A):2', 'd(B):3', 'd(C):1', 'd(D):2', 'd(E):5', 'd(F):9', 'd(G):5', 'd(H):6']

Iteration: 7
E: [(u'u', u'C'), (u'C', u'D'), (u'u', u'A'), (u'C', u'B'), (u'B', u'E'), (u'D', u'G'), (u'E', u'H')]
[d(u):0', 'd(A):2', 'd(B):3', 'd(C):1', 'd(D):2', 'd(E):5', 'd(F):8', 'd(G):5', 'd(H):6']

Iteration: 8
E: [(u'u', u'C'), (u'C', u'D'), (u'u', u'A'), (u'C', u'B'), (u'B', u'E'), (u'D', u'G'), (u'E', u'H'), (u'H', u'F')]
[d(u):0', 'd(A):2', 'd(B):3', 'd(C):1', 'd(D):2', 'd(E):5', 'd(F):7', 'd(G):5', 'd(H):6']

```

VÉGE
KÖSZÖNÖM A FIGYELMET!