Alkalmazott Modul

ELTE-TTK, Operációkutatási Tanszék

Jüttner Alpár
alpar@cs.elte.hu

ELTE-TTK, Operációkutatási tanszék

Algoritmusok futásideje

Definíció

Algoritmus futásidején vagy

- *egy adott bemeneten megtett lépések számát*

vagy

- *a megtett lépések számának a bemenet méretétől való függését (átlagosan, legrosszabb esetben, gyakorlati példákon)*

értjük.

- Mi a „bemenet mérete?”
- Mi egy „lépés”?

Definíció ($O(\cdot)$ jelölés)

- $f = O(g)$, ha $\exists c \in \mathbb{R}_+, n_0 \in \mathbb{N}$, hogy $\forall n \geq n_0$-ra $f(n) < cg(n)$
- $f = \Omega(g)$, ha $\exists c \in \mathbb{R}_+, n_0 \in \mathbb{N}$, hogy $\forall n \geq n_0$-ra $f(n) > cg(n)$

Definíció

Egy algoritmus polinomiális, ha $\exists k$, hogy futásideje $O(n^k)$.

Algoritmusok futásideje

Definíció

Algoritmus futásidején vagy

- *egy adott bemeneten megtett lépések számát*

vagy

- *a megtett lépések számának a bemenet méretétől való függését (átlagosan, legrosszabb esetben, gyakorlati példákon)*

értjük.

- Mi a „bemenet mérete?”
- Mi egy „lépés”?

Definíció ($O(\cdot)$ jelölés)

- $f = O(g)$, *ha* $\exists c \in \mathbb{R}_+, n_0 \in \mathbb{N}$, *hogy* $\forall n \geq n_0$*-ra* $f(n) < cg(n)$
- $f = \Omega(g)$, *ha* $\exists c \in \mathbb{R}_+, n_0 \in \mathbb{N}$, *hogy* $\forall n \geq n_0$*-ra* $f(n) > cg(n)$

Definíció

Egy algoritmus polinomiális, ha $\exists k$, *hogy futásideje* $O(n^k)$.

Algoritmusok futásideje

Definíció

Algoritmus futásidején vagy

- *egy adott bemeneten megtett lépések számát*

vagy

- *a megtett lépések számának a bemenet méretétől való függését (átlagosan, legrosszabb esetben, gyakorlati példákon)*

értjük.

- Mi a „bemenet mérete?”
- Mi egy „lépés”?

Definíció ($O(\cdot)$ jelölés)

- $f = O(g)$, *ha* $\exists c \in \mathbb{R}_+, n_0 \in \mathbb{N}$, *hogy* $\forall n \geq n_0$*-ra* $f(n) < cg(n)$
- $f = \Omega(g)$, *ha* $\exists c \in \mathbb{R}_+, n_0 \in \mathbb{N}$, *hogy* $\forall n \geq n_0$*-ra* $f(n) > cg(n)$

Definíció

Egy algoritmus polinomiális, ha $\exists k$, *hogy futásideje* $O(n^k)$.

Dinamikus programozás

Az eredeti feladatot „nem túl sok” (polinomiális darab) egymásra épülő részfeladat kiszámítására bontjuk.

Példa (Binomiális együtthatók kiszámítása (csak összeadással))

- $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- $\binom{n}{k} = \begin{cases} 1, & \text{ha } k = 0 \text{ vagy } k = n \\ \binom{n-1}{k} + \binom{n-1}{k-1} & \text{egyébként} \end{cases}$

Házi feladat 1/1. Példa (Hátizsák feladat)

Van p darab tárgyunk amiknek súlya rendre $w_1, w_2, \ldots, w_p$, az érték-k pedig $c_1, c_2, \ldots, c_p$. A hátizsákunkban legfeljebb W súlyt bírunk el. E feltétel mellett szeretnénk a lehető legtöbb értéket magunkkal vinni. Tegyük fel, hogy a súlyok egész számok és W „nem túl nagy”.

$$\max\left\{\sum_{i=1}^{p} c_i x_i : \quad \sum_{i=1}^{p} w_i x_i \leq W \text{ és } \forall i\text{-re } x_i \in \{0,1\}\right\} \tag{1}$$

Dinamikus programozás

Az eredeti feladatot „nem túl sok” (polinomiális darab) egymásra épülő részfeladat kiszámítására bontjuk.

Példa (Binomiális együtthatók kiszámítása (csak összeadással))

- $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- $\binom{n}{k} = \begin{cases} 1, & \text{ha } k = 0 \text{ vagy } k = n \\ \binom{n-1}{k} + \binom{n-1}{k-1} & \text{egyébként} \end{cases}$

Házi feladat 1/1. Példa (Hátizsák feladat)

Van p darab tárgyunk amiknek súlya rendre $w_1, w_2, \ldots, w_p$, az érték-k pedig $c_1, c_2, \ldots, c_p$. A hátizsákunkban legfeljebb W súlyt bírunk el. E feltétel mellett szeretnénk a lehető legtöbb értéket magunkkal vinni. Tegyük fel, hogy a súlyok egész számok és W „nem túl nagy”.

$$\max\left\{\sum_{i=1}^{p} c_i x_i : \quad \sum_{i=1}^{p} w_i x_i \leq W \text{ és } \forall i\text{-re } x_i \in \{0,1\}\right\} \tag{1}$$

Dinamikus programozás

Az eredeti feladatot „nem túl sok” (polinomiális darab) egymásra épülő részfeladat kiszámítására bontjuk.

Példa (Binomiális együtthatók kiszámítása (csak összeadással))

- $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- $\binom{n}{k} = \begin{cases} 1, & \text{ha } k = 0 \text{ vagy } k = n \\ \binom{n-1}{k} + \binom{n-1}{k-1} & \text{egyébként} \end{cases}$

Házi feladat 1/1. Példa (Hátizsák feladat)

Van p darab tárgyunk amiknek súlya rendre $w_1, w_2, \ldots, w_p$, az érték-k pedig $c_1, c_2, \ldots, c_p$. A hátizsákunkban legfeljebb W súlyt bírunk el. E feltétel mellett szeretnénk a lehető legtöbb értéket magunkkal vinni. Tegyük fel, hogy a súlyok egész számok és W „nem túl nagy”.

$$\max\left\{\sum_{i=1}^{p} c_i x_i : \quad \sum_{i=1}^{p} w_i x_i \leq W \text{ és } \forall i\text{-re } x_i \in \{0,1\}\right\} \tag{1}$$

Definíció (Irányított gráf)

- $G = (N, A)$, *ahol* N *a csúcsok (véges) halmaza és* $A \subseteq N \times N$ *az élhalmaz*
  - *Jelölés:* $n := |N|$ *és* $m := |A|$
- $a = (i, j) \in A$ *élnek* i *a farka* $[s(a)]$ *és* j *a feje* $[t(a)]$
- $\rho_G(v) = \rho(v) := \{(i, v) \in A\}$
- $\delta_G(v) = \delta(v) := \{(v, j) \in A\}$
- $G' = (N', A')$, *részgráf, ha* $N' \subseteq N$ *és* $A' \subseteq A$.
- $G' = (N', A')$ *feszített részgráf, ha* $N' = N$ *és* $A' \subseteq A$.
- $G' = (N', A')$ *indukált részgráf, ha* $N' \subseteq N$ *és* $A' := \{(i, j) \in A | i, j \in N'\}$.

Ez a definíció nem engedi meg a párhuzamos éleket. Ettől függetlenül (szinte) minden működni fog párhuzamos éleket tartalmazó gráfokra is.

Gráfok II.

Definíció (Utak, Séták)

- $\{a_1, a_2, \ldots, a_k\} \subseteq A$, *irányított séta*, *ha* $\forall i$*-re* $t(a_i) = s(a_{i+1})$.
- *Egy irányított séta irányított út, ha nem ismétlődik benne egyik csúcs sem.*
- *Egy irányított séta irányított körséta, ha* $t(a_k) = s(a_1)$.
- *Egy irányított körséta irányított kör, ha* $\{a_1, a_2, \ldots, a_{k-1}\}$ *irányított út.*
- *(irányítatlan) séta,(irányítatlan) út,(irányítatlan) körséta,(irányítatlan) kör: mint az előzőek, de az élek mindkét irányban állhatnak.*

Definíció (Összefüggőség)

A $G = (N, A)$ *irányított gráf*

- *összefüggő, ha* $\forall i, j \in N$*-re* $\exists$ *egy* $i \rightsquigarrow j$ *út.*
- *erősen összefüggő, ha* $\forall i, j \in N$*-re* $\exists$ *egy* $i \rightsquigarrow j$ *irányított út.*
- *DAG (Directed Acyclic Graph), ha nem tartalmaz irányított kört.*

Gráf (hálózat) reprezentációk

- Csúcs-csúcs adjacencia mátrix
  - $M \in \{0,1\}^{N \times N}, \quad m_{i,j} := \begin{cases} 1, & \text{ha } (i,j) \in A \\ 0, & \text{egyébként} \end{cases}$
- Csúcs-él incidencia mátrix
  - $M \in \{0,1\}^{N \times A}, \quad m_{i,a} := \begin{cases} 1, & \text{ha } \exists j \in N : a = (i,j) \\ -1, & \text{ha } \exists j \in N : a = (j,i) \\ 0, & \text{egyébként} \end{cases}$
- Éllista reprezentáció (Adjacency list)
- Ki-csillag, be-csillag (Forward and Reverse Star) reprezentáció

Feladatok (dinamikus programozás)

Házi feladat 1/2.

Adott egy $G = (N, A)$ irányított gráf. Adjunk algoritmust, ami eldönti, hogy van-e páratlan hosszú irányított kör a gráfban.

- És ha páratlan helyett párost kérdezünk?

Házi feladat 1/3.

Adott egy $G = (N, A)$ DAG, $s \in N$. Jelölje $\alpha(i)$ a különböző $s \rightsquigarrow i$ utak számát G-ben. Adjunk $O(m)$ idejű algoritmust, ami kiszámolja $\alpha(i)$-t minden $i \in N$-re.

Reklám

http://lemon.cs.elte.hu

- Nyílt forráskódú C++ library gráfokkal (hálózatokkal) kapcsolatos optimalizálási feladatok megoldására
- Cél: Olyan eszköztár létrehozása, amely egyszerre alkalmas kutatási feladatokban *és* ipari felhasználásra.
  - Nyílt forráskód, kereskedelmi felhasználást is biztosító licenc
  - Az eszközök elérhető leghatékonyabb implementációjára törekszünk (a leggyorsabb a piacon)
- Megbízható, kiterjedt implementáció
  - $\approx$ 55000 erősen optimalizált kódsor
  - Windows, Unix támogatás különféle fordítókkal
- Fejlesztő kerestetik!!!

Gráf keresés

```
1:  procedure SEARCH(G, s)
2:      ∀v ∈ N-re m(v) := 0                          ▷ Minden csúcs jelöletlen (eléretlen)
3:      m(s) := 1                                    ▷ kivéve s
4:      pred(s) := 0; order(s) := 1
5:      next := 1
6:      LIST := {s}                                  ▷ Feldolgozandó csúcsok listája
7:      while LIST ≠ ∅ do
8:          Kiválasztunk egy i ∈ LIST csúcsot
9:          if ∃(i, j) ∈ δ_G(i),    m(j) = 0 then
10:             m(j) := 1                            ▷ j-t megjelöljük, mint elértet
11:             pred(j) := i
12:             next := next + 1; order(j) := next
13:             LIST := LIST ∪ {j}
14:         else
15:             LIST := LIST \ {i}
16:         end if
17:     end while
18: end procedure
```

Gráf keresés

```
1: procedure SEARCH(G, s)
2:     ∀v ∈ N-re m(v) := 0                          ▷ Minden csúcs jelöletlen (eléretlen)
3:     m(s) := 1                                    ▷ kivéve s
4:     pred(s) := 0; order(s) := 1
5:     next := 1
6:     LIST := {s}                                  ▷ Feldolgozandó csúcsok listája
7:     while LIST ≠ ∅ do
8:         Kiválasztunk egy i ∈ LIST csúcsot
9:         if ∃(i, j) ∈ δ_G(i),   m(j) = 0 then
10:            m(j) := 1                            ▷ j-t megjelöljük, mint elértet
11:            pred(j) := i
12:            next := next + 1; order(j) := next
13:            LIST := LIST ∪ {j}
14:        else
15:            LIST := LIST \ {i}
16:        end if
17:    end while
18: end procedure
```

Megjegyzés

Van szabadságunk abban, hogy melyik csúcsot válasszuk ki 8-nál.

Gráf keresés II.

Állítás

Futásidő: $O(m)$ (Ha a LIST-műveletek $O(1)$)

Állítás (házi feladat 1/4.)

pred : $N \longrightarrow N$ egy s-gyökerű kifele irányított fát (ki-fenyőt) határoz meg, ami

- pontosan az s-ből irányított úton elérhető csúcsokat feszíti
- irányítatlan gráf esetén feszíti az s-et tartalmazó összefüggő komponenst.

Állítás

Ez a fa minden s-ből irányított úton elérhető t pontra meghatároz egy egyértelmű $s \rightsquigarrow t$ (irányított) utat.

Gráf keresés II.

Állítás

Futásidő: $O(m)$ (Ha a LIST-műveletek $O(1)$)

Állítás (házi feladat 1/4.)

pred : $N \longrightarrow N$ egy s-gyökerű kifele irányított fát (ki-fenyőt) határoz meg, ami

- pontosan az s-ből irányított úton elérhető csúcsokat feszíti
- irányítatlan gráf esetén feszíti az s-et tartalmazó összefüggő komponenst.

Állítás

Ez a fa minden s-ből irányított úton elérhető t pontra meghatároz egy egyértelmű $s \rightsquigarrow t$ (irányított) utat.

Gráf keresés II.

Állítás

Futásidő: $O(m)$ (Ha a LIST-műveletek $O(1)$)

Állítás (házi feladat 1/4.)

pred : $N \longrightarrow N$ egy s-gyökerű kifele irányított fát (ki-fenyőt) határoz meg, ami

- pontosan az *s*-ből irányított úton elérhető csúcsokat feszíti
- irányítatlan gráf esetén feszíti az *s*-et tartalmazó összefüggő komponenst.

Állítás

Ez a fa minden s-ből irányított úton elérhető t pontra meghatároz egy egyértelmű $s \rightsquigarrow t$ (irányított) utat.

Szélességi keresés (Breadth-First Search, BFS)

LIST legyen egy sor (queue)

Állítás

Minden t pontra a BFS-fenyő által meghatározott $s \rightsquigarrow t$ út egy legrövidebb (minimális élszámú) út G gráfban.

Szélességi keresés (Breadth-First Search, BFS)

LIST legyen egy sor (queue)

Állítás

Minden t pontra a BFS-fenyő által meghatározott $s \rightsquigarrow t$ út egy legrövidebb (minimális élszámú) út G gráfban.

Éllistás ábrázolás

Gráf-struktúra

- Élek
  - m: int
  - sources: int[M]
  - targets: int[M]
  - next_out: int[M]
  - next_in: int[M]
- Csúcsok
  - n: int
  - first_out: int[N]
  - first_in: int[N]

```
1: function AddNode
2:     n ← n + 1
3:     first_out[n] ← 0
4:     first_in[n] ← 0
5:     return n
6: end function
```

```
1: function AddEdge(i, j)
2:     m ← m + 1
3:     sources[m] ← i
4:     targets[m] ← j
5:     next_out[m] ← first_out[i]
6:     next_in[m] ← first_in[j]
7:     first_out[i] ← m
8:     first_in[j] ← m
9:     return m
10: end function
```

Éllistás ábrázolás

Gráf-struktúra

- Élek
  - m: int
  - sources: int[M]
  - targets: int[M]
  - next_out: int[M]
  - next_in: int[M]
- Csúcsok
  - n: int
  - first_out: int[N]
  - first_in: int[N]

```
1: function AddNode
2:     n ← n + 1
3:     first_out[n] ← 0
4:     first_in[n] ← 0
5:     return n
6: end function
```

```
1: function AddEdge(i, j)
2:     m ← m + 1
3:     sources[m] ← i
4:     targets[m] ← j
5:     next_out[m] ← first_out[i]
6:     next_in[m] ← first_in[j]
7:     first_out[i] ← m
8:     first_in[j] ← m
9:     return m
10: end function
```

Éllistás ábrázolás

Gráf-struktúra

- Élek
  - m: int
  - sources: int[M]
  - targets: int[M]
  - next_out: int[M]
  - next_in: int[M]
- Csúcsok
  - n: int
  - first_out: int[N]
  - first_in: int[N]

```
1: function AddNode
2:     n ← n + 1
3:     first_out[n] ← 0
4:     first_in[n] ← 0
5:     return n
6: end function
```

```
1: function AddEdge(i, j)
2:     m ← m + 1
3:     sources[m] ← i
4:     targets[m] ← j
5:     next_out[m] ← first_out[i]
6:     next_in[m] ← first_in[j]
7:     first_out[i] ← m
8:     first_in[j] ← m
9:     return m
10: end function
```

BFS

```
 1: procedure BFS(s)
 2:     l : bool[m], pred : int[m]
 3:     ∀i ∈ {1 . . . m}-re l[i] ← false
 4:     l[s] ← true
 5:     pred[s] ← 0
 6:     LIST ← {s}
 7:     while LIST ≠ ∅ do
 8:         Kiválasztjuk a i ∈ LIST csúcsot
 9:         LIST ← LIST \ {i}
10:         e ← first_out[i]
11:         while e ≠ 0 do
12:             j ← targets[e]
13:             if l[j] = false then
14:                 l[j] ← true
15:                 pred[j] ← (i, e)
16:                 LIST ← LIST ∪ {j}
17:             end if
18:             e ← next_out[e]                          ▷ Következő él
19:         end while
20:     end while
21: end procedure
```

BFS

```
 1: procedure BFS(s)
 2:     l : bool[m], pred : int[m]
 3:     ∀i ∈ {1 . . . m}-re l[i] ← false
 4:     l[s] ← true
 5:     pred[s] ← 0
 6:     LIST : int[n],   lh ← 1,   lt ← 1
 7:     LIST[lt] ← s,   lt ← lt + 1
 8:     while lh ≠ lt do
 9:         i ← LIST[lh]
10:         lh ← lh + 1
11:         e ← first_out[i]
12:         while e ≠ 0 do
13:             j ← targets[e]
14:             if l[j] = false then
15:                 l[j] ← true
16:                 pred[j] ← (i, e)
17:                 LIST[lt] ← j,   lt ← lt + 1
18:             end if
19:             e ← next_out[e]                          ▷ Következő él
20:         end while
21:     end while
22: end procedure
```

Mélységi keresés (Depth-First Search, DFS)

LIST legyen egy verem (stack)

Állítás

A DFS fenyőben

- *ha j „leszármazottja” i-nek (azaz van egy $i \rightsquigarrow j$ út a DFS fenyőben és $i \neq j$), akkor $order(j) > order(i)$.*
- *egy pont leszármazottjainak sorrendjei (order értékei) egy intervallumot alkotnak.*

Házi feladat 2/1.

Bizonyítsuk be a fenti állításokat.

Házi feladat 2/2.

Legyen G irányítatlan gráf, T a DFS által megtalált fenyő és $(k, l) \in A \setminus T$. Ekkor vagy k leszármazottja l-nek vagy l leszármazottja k-nak T-ben.

Mélységi keresés (Depth-First Search, DFS)

LIST legyen egy verem (stack)

Állítás

A DFS fenyőben

- *ha j „leszármazottja” i-nek (azaz van egy $i \rightsquigarrow j$ út a DFS fenyőben és $i \neq j$), akkor $order(j) > order(i)$.*
- *egy pont leszármazottjainak sorrendjei (order értékei) egy intervallumot alkotnak.*

Házi feladat 2/1.

Bizonyítsuk be a fenti állításokat.

Házi feladat 2/2.

Legyen G irányítatlan gráf, T a DFS által megtalált fenyő és $(k, l) \in A \setminus T$. Ekkor vagy k leszármazottja l-nek vagy l leszármazottja k-nak T-ben.

Mélységi keresés (Depth-First Search, DFS)

LIST legyen egy verem (stack)

Állítás

A DFS fenyőben

- *ha j „leszármazottja” i-nek (azaz van egy $i \rightsquigarrow j$ út a DFS fenyőben és $i \neq j$), akkor $order(j) > order(i)$.*
- *egy pont leszármazottjainak sorrendjei (order értékei) egy intervallumot alkotnak.*

Házi feladat 2/1.

Bizonyítsuk be a fenti állításokat.

Házi feladat 2/2.

Legyen G irányítatlan gráf, T a DFS által megtalált fenyő és $(k, l) \in A \setminus T$. Ekkor vagy k leszármazottja l-nek vagy l leszármazottja k-nak T-ben.

Topologikus sorrend

Definíció

A $G = (N, A)$ gráf csúcsainak egy $order : N \longrightarrow \mathbb{N}$ sorrendjét topologikus sorrendnek nevezzük, ha minden $(i, j) \in A$ élre $order(i) < order(j)$.

Tétel

Egy gráfnak pontosan akkor van topologikus sorrendje, ha nem tartalmaz irányított kört (DAG).

Algoritmus

```
1: next:=1
2: while G ≠ ∅ do
3:     Válasszunk ki egy i csúcsot, amibe nem megy be él        ▷ Van ilyen?
4:     order(i) := next; next := next + 1
5:     Töröljük ki G-ből i-t (és a belőle kiinduló éleket)
6: end while
```

Házi feladat 2/3.

Adjuk meg a fenti algoritmus egy hatékony ($O(m + n)$ futási idejű) megvalósítását.

Topologikus sorrend

Definíció

A $G = (N, A)$ gráf csúcsainak egy $order : N \longrightarrow \mathbb{N}$ sorrendjét topologikus sorrendnek nevezzük, ha minden $(i, j) \in A$ élre $order(i) < order(j)$.

Tétel

Egy gráfnak pontosan akkor van topologikus sorrendje, ha nem tartalmaz irányított kört (DAG).

Algoritmus

```
1: next:=1
2: while G ≠ ∅ do
3:     Válasszunk ki egy i csúcsot, amibe nem megy be él        ▷ Van ilyen?
4:     order(i) := next; next := next + 1
5:     Töröljük ki G-ből i-t (és a belőle kiinduló éleket)
6: end while
```

Házi feladat 2/3.

Adjuk meg a fenti algoritmus egy hatékony ($O(m + n)$ futási idejű) megvalósítását.

Topologikus sorrend

Definíció

A $G = (N, A)$ gráf csúcsainak egy $order : N \longrightarrow \mathbb{N}$ sorrendjét topologikus sorrendnek nevezzük, ha minden $(i, j) \in A$ élre $order(i) < order(j)$.

Tétel

Egy gráfnak pontosan akkor van topologikus sorrendje, ha nem tartalmaz irányított kört (DAG).

Algoritmus

next:=1
while *$G \neq \emptyset$* ***do***
    Válasszunk ki egy i csúcsot, amibe nem megy be él ▷ *Van ilyen?*
    $order(i) := next$; $next := next + 1$
    Töröljük ki G-ből i-t (és a belőle kiinduló éleket)
end while

Házi feladat 2/3.

Adjuk meg a fenti algoritmus egy hatékony ($O(m + n)$ futási idejű) megvalósítását.

Topologikus sorrend

Definíció

A $G = (N, A)$ gráf csúcsainak egy $order : N \longrightarrow \mathbb{N}$ sorrendjét topologikus sorrendnek nevezzük, ha minden $(i, j) \in A$ élre $order(i) < order(j)$.

Tétel

Egy gráfnak pontosan akkor van topologikus sorrendje, ha nem tartalmaz irányított kört (DAG).

Algoritmus

next:=1
while *$G \neq \emptyset$* ***do***
    Válasszunk ki egy i csúcsot, amibe nem megy be él ▷ *Van ilyen?*
    $order(i) := next$; $next := next + 1$
    Töröljük ki G-ből i-t (és a belőle kiinduló éleket)
end while

Házi feladat 2/3.

Adjuk meg a fenti algoritmus egy hatékony ($O(m + n)$ futási idejű) megvalósítását.

DAG Példa: BKK menetrend

Menetrend gráf

$V := \{(s,t) : s \in \mathcal{S}, t \in \mathcal{T}_s\}$ $A := A_{trip} \cup A_{change}$, ahol

- $\mathcal{S}$ a megállók halmaza,
- $\mathcal{T}_s$ az s megállóba érkező járatok halmaza,
  - Az azonos útvonalon különböző időben haladó járművek külön járatnak számítanak.
- $A_{trip} := \{((s_1,t),(s_2,t)) : s_1$ és s_2 a t két egymást követő megállója$\}$,
- $A_{change} := \{((s,t_1),(s,t_2)) :$ a t_1-ről a t_2-re át lehet szállni az s-ben$\}$
  - gyaloglás éleket is hozzáadhatunk adni a közeli megállók között

Házi feladat 2/3. Írjunk utazástervező programot!

- BKK menetrend:
  `https://lemon.cs.elte.hu/trac/lemon/wiki/AlkMod2013`
- Fel kell fizikailag építenünk a fenti a gráfot a memóriában?
- Célfüggvény: Minimális utazási idő, legkevesebb átszállás, legkevesebb gyaloglás (vagy ezek kombinációja.)
- Alternatív útvonaltervek keresése

DAG Példa: BKK menetrend

Menetrend gráf

$V := \{(s,t) : s \in \mathcal{S}, t \in \mathcal{T}_s\}$ $A := A_{trip} \cup A_{change}$, ahol

- $\mathcal{S}$ a megállók halmaza,
- $\mathcal{T}_s$ az s megállóba érkező járatok halmaza,
  - Az azonos útvonalon különböző időben haladó járművek külön járatnak számítanak.
- $A_{trip} := \{((s_1,t),(s_2,t)) : s_1$ és s_2 a t két egymást követő megállója$\}$,
- $A_{change} := \{((s,t_1),(s,t_2)) :$ a t_1-ről a t_2-re át lehet szállni az s-ben$\}$
  - gyaloglás éleket is hozzáadhatunk adni a közeli megállók között

Házi feladat 2/3. Írjunk utazástervező programot!

- BKK menetrend:
  `https://lemon.cs.elte.hu/trac/lemon/wiki/AlkMod2013`
- Fel kell fizikailag építenünk a fenti a gráfot a memóriában?
- Célfüggvény: Minimális utazási idő, legkevesebb átszállás, legkevesebb gyaloglás (vagy ezek kombinációja.)
- Alternatív útvonaltervek keresése

Legrövidebb utak irányított gráfokban

Definíció (Legrövidebb út probléma)

Adott egy $G = (N, A)$ gráf és egy $c : A \longrightarrow \mathbb{R}$ hosszfüggvény az éleken, továbbá $s, t \in N$ csúcsok.
Keressük a legrövidebb $s \rightsquigarrow t$ utat.

Előfeltevések

- G irányított gráf ($\Longrightarrow$ *irányított utat* keresünk).
- G nem tartalmaz negatív kört (azaz negatív összhosszúságú irányított kört)
- s-ből minden pont elérhető irányított út mentén
- (Néha) A hosszúságértékek egész számok ($c : A \longrightarrow \mathbb{Z}$)

Változatok

- Keressünk utat s-ből az összes többi pontba
  - nemnegatív élhosszok esetén
  - tetszőleges élhosszokkal.
- Keressünk utat mindenhonnan mindenhova.
- Egyéb általánosítások
  - Időkorlátos legrövidebb út, max kapacitású út, fordulások büntetése, ...

Legrövidebb utak irányított gráfokban

Definíció (Legrövidebb út probléma)

Adott egy $G = (N, A)$ gráf és egy $c : A \longrightarrow \mathbb{R}$ hosszfüggvény az éleken, továbbá $s, t \in N$ csúcsok.
Keressük a legrövidebb $s \rightsquigarrow t$ utat.

Előfeltevések

- G irányított gráf ($\Longrightarrow$ *irányított utat* keresünk).
- G nem tartalmaz negatív kört (azaz negatív összhosszúságú irányított kört)
- s-ből minden pont elérhető irányított út mentén
- (Néha) A hosszúságértékek egész számok ($c : A \longrightarrow \mathbb{Z}$)

Változatok

- Keressünk utat s-ből az összes többi pontba
  - nemnegatív élhosszok esetén
  - tetszőleges élhosszokkal.
- Keressünk utat mindenhonnan mindenhova.
- Egyéb általánosítások
  - Időkorlátos legrövidebb út, max kapacitású út, fordulások büntetése, ...

Legrövidebb utak irányított gráfokban

Definíció (Legrövidebb út probléma)

Adott egy $G = (N, A)$ gráf és egy $c : A \longrightarrow \mathbb{R}$ hosszfüggvény az éleken, továbbá $s, t \in N$ csúcsok.
Keressük a legrövidebb $s \rightsquigarrow t$ utat.

Előfeltevések

- G irányított gráf ($\Longrightarrow$ *irányított utat* keresünk).
- G nem tartalmaz negatív kört (azaz negatív összhosszúságú irányított kört)
- s-ből minden pont elérhető irányított út mentén
- (Néha) A hosszúságértékek egész számok ($c : A \longrightarrow \mathbb{Z}$)

Változatok

- Keressünk utat s-ből az összes többi pontba
  - nemnegatív élhosszok esetén
  - tetszőleges élhosszokkal.
- Keressünk utat mindenhonnan mindenhova.
- Egyéb általánosítások
  - Időkorlátos legrövidebb út, max kapacitású út, fordulások büntetése, ...

Legrövidebb utak fája

Állítás

Legyen p egy $s \rightsquigarrow t$ legrövidebb út. Ekkor p minden részútja is egy legrövidebb út annak két végpontja között.

(Emlékeztető: Feltettük, hogy G-ben nincs negatív kör!)

Állítás

Minden $s \rightsquigarrow t$ irányított séta felbontható egy $s \rightsquigarrow t$ út és irányított körök uniójára.

Tétel

Jelölje $d(i)$ a legrövidebb $s \rightsquigarrow i$ út hosszát. Ekkor

- *Minden $(i, j) \in A$ élre $d(j) - d(i) \leq c_{ij}$*
- *Egy p $s \rightsquigarrow u$ út pontosan akkor legrövidebb, ha minden $(i, j) \in p$ élre $d(j) - d(i) = c_{ij}$*

Állítás

Az s-ből kiinduló legrövidebb utak megválaszthatók úgy, hogy egy s gyökerű ki-fenyőt alkossanak.

Legrövidebb utak fája

Állítás

Legyen p egy $s \rightsquigarrow t$ legrövidebb út. Ekkor p minden részútja is egy legrövidebb út annak két végpontja között.

(Emlékeztető: Feltettük, hogy G-ben nincs negatív kör!)

Állítás

Minden $s \rightsquigarrow t$ irányított séta felbontható egy $s \rightsquigarrow t$ út és irányított körök uniójára.

Tétel

Jelölje $d(i)$ a legrövidebb $s \rightsquigarrow i$ út hosszát. Ekkor

- *Minden $(i, j) \in A$ élre $d(j) - d(i) \leq c_{ij}$*
- *Egy p $s \rightsquigarrow u$ út pontosan akkor legrövidebb, ha minden $(i, j) \in p$ élre $d(j) - d(i) = c_{ij}$*

Állítás

Az s-ből kiinduló legrövidebb utak megválaszthatók úgy, hogy egy s gyökerű ki-fenyőt alkossanak.

Legrövidebb utak fája

Állítás

Legyen p egy $s \rightsquigarrow t$ legrövidebb út. Ekkor p minden részútja is egy legrövidebb út annak két végpontja között.

(Emlékeztető: Feltettük, hogy G-ben nincs negatív kör!)

Állítás

Minden $s \rightsquigarrow t$ irányított séta felbontható egy $s \rightsquigarrow t$ út és irányított körök uniójára.

Tétel

Jelölje $d(i)$ a legrövidebb $s \rightsquigarrow i$ út hosszát. Ekkor

- *Minden $(i,j) \in A$ élre $d(j) - d(i) \leq c_{ij}$*
- *Egy p $s \rightsquigarrow u$ út pontosan akkor legrövidebb, ha minden $(i,j) \in p$ élre $d(j) - d(i) = c_{ij}$*

Állítás

Az s-ből kiinduló legrövidebb utak megválaszthatók úgy, hogy egy s gyökerű ki-fenyőt alkossanak.

Legrövidebb út DAG-ban

Tétel

Egy DAG-ban a legrövidebb utak fája $O(m)$ időben meghatározható.

Bemenő éleken dolgozó algoritmus

1: **procedure** DAGSHORTESTPATH(G, c, s)
2: Rendezzük G pontjait topologikus sorrendbe
3: Hagyjuk el az s előtti pontokat ▷ Ezek nem érhetők el s-ből
4: $d(s) := 0, pred(s) := 0$
5: **for** $\forall j \in N, j \neq s$, topologikus sorrendben **do**
6: $(i_{min}, j) := \text{argmin}\{d(i) + c_{ij} : (i, j) \in \rho(j)\}$
7: $d(j) := d(i_{min}) + c_{i_{min}j}$
8: $pred(j) := (i_{min}, j)$
9: **end for**
10: **end procedure**

Legrövidebb út DAG-ban II.

Kimenő éleken dolgozó algoritmus

1: **procedure** DAGSHORTESTPATH(G, c, s)
2: Rendezzük G pontjait topologikus sorrendbe
3: Hagyjuk el az s előtti pontokat ▷ Ezek nem érhetők el s-ből
4: **for** $\forall i \in N, i \neq s$ **do**
5: $d(i) := \infty$
6: **end for**
7: $d(s) := 0, pred(s) := 0$
8: **for** $\forall i \in N, i \neq s$, topologikus sorrendben **do**
9: **for** $j : (i, j) \in \delta(i)$ **do**
10: **if** $d(i) + c_{ij} < d(j)$ **then**
11: $d(j) := d(i) + c_{ij}$
12: $pred(j) := (i, j)$
13: **end if**
14: **end for**
15: **end for**
16: **end procedure**

Legrövidebb út nemnegatív élhosszak esetén

Dijkstra algoritmus

1: **procedure** DIJKSTRA($G = (N, A), c, s$)
2: $S := \emptyset, \overline{S} := N$
3: **for** $\forall i \in N, i \neq s$ **do**
4: $d(i) := \infty$
5: **end for**
6: $d(s) := 0, pred(s) := 0$
7: **while** $\overline{S} \neq \emptyset$ **do**
8: $i := \text{argmin}\{d(j) : j \in \overline{S}\}$
9: $S := S \cup \{i\}, \quad \overline{S} := \overline{S} \setminus \{i\}$
10: **for** $i : (i, j) \in \delta(i)$ **do**
11: **if** $d(i) + c_{ij} < d(j)$ **then**
12: $d(j) := d(i) + c_{ij}$
13: $pred(j) := (i, j)$
14: **end if**
15: **end for**
16: **end while**
17: **end procedure**

Legrövidebb út nemnegatív élhosszak esetén

Dijkstra algoritmus

1: **procedure** DIJKSTRA($G = (N, A), c, s$)
2: $S := \emptyset, \overline{S} := N$
3: **for** $\forall i \in N, i \neq s$ **do**
4: $d(i) := \infty$
5: **end for**
6: $d(s) := 0, pred(s) := 0$
7: **while** $\overline{S} \neq \emptyset$ **do**
8: $i := \text{argmin}\{d(j) : j \in \overline{S}\}$
9: $S := S \cup \{i\}, \quad \overline{S} := \overline{S} \setminus \{i\}$
10: **for** $i : (i, j) \in \delta(i)$ **do**
11: **if** $d(i) + c_{ij} < d(j)$ **then**
12: $d(j) := d(i) + c_{ij}$
13: $pred(j) := (i, j)$
14: **end if**
15: **end for**
16: **end while**
17: **end procedure**

Bizonyítás (Helyesség)

S méretére vonatkozó indukcióval. Indukciós hipotézis:

(1) *Minden $i \in S$-re $d(i)$, helyes azaz a legrövidebb $s \rightsquigarrow i$ út hossza.*

(2) *Minden $i \in \overline{S}$-re $d(i) := \min\{c(p) : p$ egy $s \rightsquigarrow i$ út és $p \subseteq S \cup \{i\}\}$*

Dijkstra algoritmus futásideje

Futásidő (primitív implementáció): $O(n^2 + m)$

- a $d(i)$ értékeket egy tömbben tároljuk
- Csúcs kiválasztás: $n + (n-1) + (n-2) + \cdots + 1 = O(n^2)$
- Távolságértékek ($d(i)$-k) frissítése: $\sum_{i \in N} |\delta(i)| = m$.

Dijkstra algoritmus – heap (kupac) implementáció

Dijkstra algoritmus

```
 1: procedure DIJKSTRA(G = (N, A), c, s)
 2:     create-heap(H)
 3:     for ∀i ∈ N, i ≠ s do
 4:         d(i) := ∞
 5:     end for
 6:     d(s) := 0, pred(s) := 0
 7:     insert(s,H)
 8:     while H ≠ ∅ do
 9:         find-min(i,H), delete-min(H)
10:         for i : (i, j) ∈ δ(i) do
11:             val := d(i) + c_ij
12:             if val < d(j) then
13:                 d(j) := val
14:                 pred(j) := (i, j)
15:                 if j ∈ H then
16:                     decrease-key(val,j,H)
17:                 else
18:                     insert(j,H)
19:                 end if
20:             end if
21:         end for
22:     end while
23: end procedure
```

Dijkstra algoritmus – heap (kupac) implementáció

Dijkstra algoritmus

```
1: procedure DIJKSTRA(G = (N, A), c, s)
2:     create-heap(H)
3:     for ∀i ∈ N, i ≠ s do
4:         d(i) := ∞
5:     end for
6:     d(s) := 0, pred(s) := 0
7:     insert(s,H)
8:     while H ≠ ∅ do
9:         find-min(i,H), delete-min(H)
10:        for i : (i, j) ∈ δ(i) do
11:            val := d(i) + c_ij
12:            if val < d(j) then
13:                d(j) := val
14:                pred(j) := (i, j)
15:                if j ∈ H then
16:                    decrease-key(val,j,H)
17:                else
18:                    insert(j,H)
19:                end if
20:            end if
21:        end for
22:    end while
23: end procedure
```

Tétel (Dijkstra futásideje heap adatstruktúrával)

A Dijkstra algoritmus futásideje $O(nT_M + mT_K)$, ahol T_M egy min. kulcsú elem kiválasztásának, T_K egy kulcscsökkentésnek az ideje.

Dijkstra algoritmus

Futásidő (primitív implementáció): $O(n^2 + m)$

- Min. csúcs kiválasztás: $O(n)$
- Kulcscsökkentés: $O(1)$.

Dijkstra algoritmus – Dial implementáció

$$c : A \longrightarrow \{0, 1, 2, \ldots, C\}$$

Dial-féle implementáció

- Csinálunk $nC + 1$ darab tárolót: $B_k := \{i \in N : d(i) = k\}$.
- Nyilvántartjuk a legkisebb index d_{min} indexet, amire $B_{d_{min}}$ nem üres.
- Minden iterációban frissítjük B_k-kat és d_{min}-t.

Tétel (Futásidő)

A fenti implementáció futásideje $O(nC + m)$.

Állítás

Ha egy iterációban az $i \in N$ csúcsot dolgozzuk fel, akkor minden $j \in N$, $d(j) < \infty$ csúcsra $d(j) \leq d(i) + C$.

Tétel (Ciklikus tárolók)

Elegendő $C + 1$ tárolót fenntartani $(B_{d_{min}}, \ldots, B_{d_{min}+C})$

Dijkstra algoritmus – Dial implementáció

$$c : A \longrightarrow \{0, 1, 2, \ldots, C\}$$

Dial-féle implementáció

- Csinálunk $nC + 1$ darab tárolót: $B_k := \{i \in N : d(i) = k\}$.
- Nyilvántartjuk a legkisebb index d_{min} indexet, amire $B_{d_{min}}$ nem üres.
- Minden iterációban frissítjük B_k-kat és d_{min}-t.

Tétel (Futásidő)

A fenti implementáció futásideje $O(nC + m)$.

Állítás

Ha egy iterációban az $i \in N$ csúcsot dolgozzuk fel, akkor minden $j \in N$, $d(j) < \infty$ csúcsra $d(j) \leq d(i) + C$.

Tétel (Ciklikus tárolók)

Elegendő $C + 1$ tárolót fenntartani $(B_{d_{min}}, \ldots, B_{d_{min}+C})$

Dijkstra algoritmus – Dial implementáció

$$c : A \longrightarrow \{0, 1, 2, \ldots, C\}$$

Dial-féle implementáció

- Csinálunk $nC + 1$ darab tárolót: $B_k := \{i \in N : d(i) = k\}$.
- Nyilvántartjuk a legkisebb index d_{min} indexet, amire $B_{d_{min}}$ nem üres.
- Minden iterációban frissítjük B_k-kat és d_{min}-t.

Tétel (Futásidő)

A fenti implementáció futásideje $O(nC + m)$.

Állítás

Ha egy iterációban az $i \in N$ csúcsot dolgozzuk fel, akkor minden $j \in N$, $d(j) < \infty$ csúcsra $d(j) \leq d(i) + C$.

Tétel (Ciklikus tárolók)

Elegendő $C + 1$ tárolót fenntartani ($B_{d_{min}}, \ldots, B_{d_{min}+C}$)

Dijkstra algoritmus – Dial implementáció

$$c : A \longrightarrow \{0, 1, 2, \ldots, C\}$$

Dial-féle implementáció

- Csinálunk $nC + 1$ darab tárolót: $B_k := \{i \in N : d(i) = k\}$.
- Nyilvántartjuk a legkisebb index d_{min} indexet, amire $B_{d_{min}}$ nem üres.
- Minden iterációban frissítjük B_k-kat és d_{min}-t.

Tétel (Futásidő)

A fenti implementáció futásideje $O(nC + m)$.

Állítás

Ha egy iterációban az $i \in N$ csúcsot dolgozzuk fel, akkor minden $j \in N$, $d(j) < \infty$ csúcsra $d(j) \leq d(i) + C$.

Tétel (Ciklikus tárolók)

Elegendő $C + 1$ tárolót fenntartani $(B_{d_{min}}, \ldots, B_{d_{min}+C})$

Radix heap

Hasonló a Dial implementációhoz, de az egyes tárolókban nem egyetlen távolságértékhez tartozó csúcsokat tárolunk.

Inicializáláskor

- $K := \lceil \log(nC) \rceil$
- $R_0 := [0]$, $R_1 := [1]$, $R_2 := [2,3]$, $R_3 := [4,7]$, $\ldots$, $R_K := [2^{K-1}, 2^K - 1]$.
  Invariáns tulajdonság: R_k intervallumban 2^{k-1} darab szám van, ha $k > 0$.
- $B_k := \{i \in N : d(i) \in R_i\}$
- k_{min} az első nemüres tároló indexe

Műveletek

- Min. csúcs kiválasztása
  - Ha $B_{k_{min}} = \emptyset$, akkor $k_{min} := k_{min} + 1$
  - Ha $|B_{k_{min}}| = 1$, akkor feldolgozzuk a benne levő csúcsot és $k_{min} := k_{min} + 1$
  - Ha $|B_{k_{min}}| > 1$, akkor
    - $R_{k_{min}}$-t felosztjuk $R_1, \ldots, R_{k_{min}-1}$ között, $R_{k_{min}} := \emptyset$
    - $B_{k_{min}}$ elemeti szétosztjuk $B_1, \ldots, B_{k_{min}-1}$-be, $|B_{k_{min}}| := \emptyset$
    - $k_{min} := 0$
- Kulcscsökkentés: Visszafele haladva megkeressük a megfelelő tárolót

Radix heap

Hasonló a Dial implementációhoz, de az egyes tárolókban nem egyetlen távolságértékhez tartozó csúcsokat tárolunk.

Inicializáláskor

- $K := \lceil \log(nC) \rceil$
- $R_0 := [0]$, $R_1 := [1]$, $R_2 := [2,3]$, $R_3 := [4,7]$, $\ldots$, $R_K := [2^{K-1}, 2^K - 1]$.
  Invariáns tulajdonság: R_k intervallumban 2^{k-1} darab szám van, ha $k > 0$.
- $B_k := \{i \in N : d(i) \in R_i\}$
- k_{min} az első nemüres tároló indexe

Futásidő („amortizált futásidő” becslés)

- Kulcscsökkentés összesen: $O(nK)$
- Tároló újraosztás: $O(K)$, összesen: $O(nK)$
- Élek feldolgozása: $O(m)$

Tétel

A radix heap teljes futásideje $O(m + n \log(nC))$.

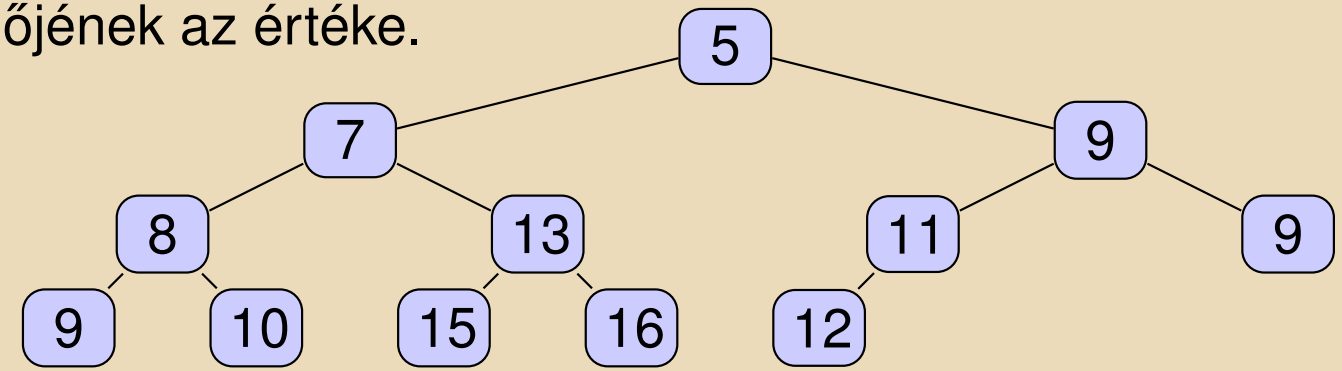
Állítás

A radix heap teljes futásideje $O(m + n \log(C))$. *(Elég* $1 + \lceil \log C \rceil$ *tároló.)*

Bináris és d-Kupac

Egy kiegyensúlyozott teljes bináris (vagy d-leszármazottas) fa a következő invariáns tulajdonsággal:

- A fa minden csúcsában levő érték legalább akkora, mint a szülőjének az értéke.



Tétel

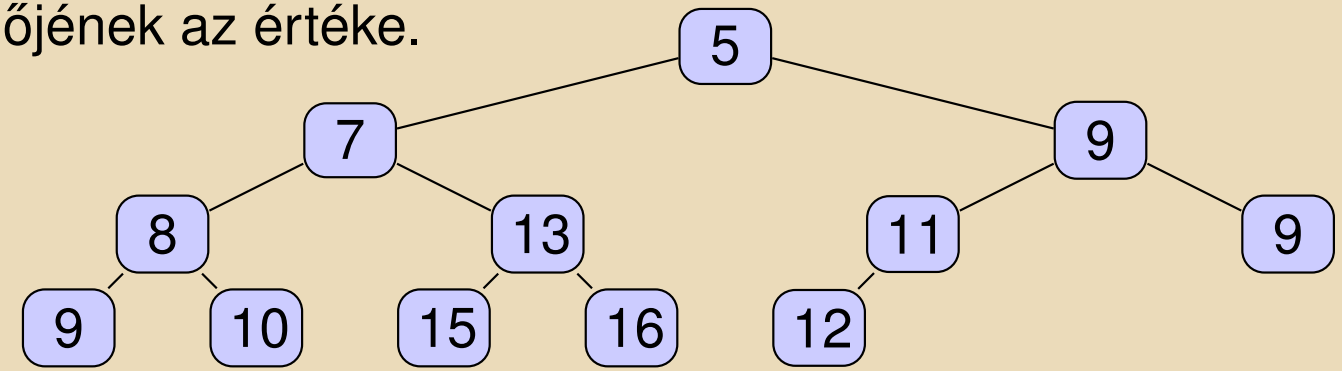
- *Bináris kupacban az* `insert()`, `decrease-min()` *és* `delete-min()` *műveletek* $O(\log n)$ *idejűek.*
- *d-kupacban kupacban az* `insert()` *és* `decrease-min()` *műveletek* $O(\log_d n)$ *idejűek, a* `delete-min()` *pedig* $O(d \log_d n)$.

Egy ilyen adatstruktúra tömbben tárolható!

Bináris és d-Kupac

Egy kiegyensúlyozott teljes bináris (vagy d-leszármazottas) fa a következő invariáns tulajdonsággal:

- A fa minden csúcsában levő érték legalább akkora, mint a szülőjének az értéke.



Tétel

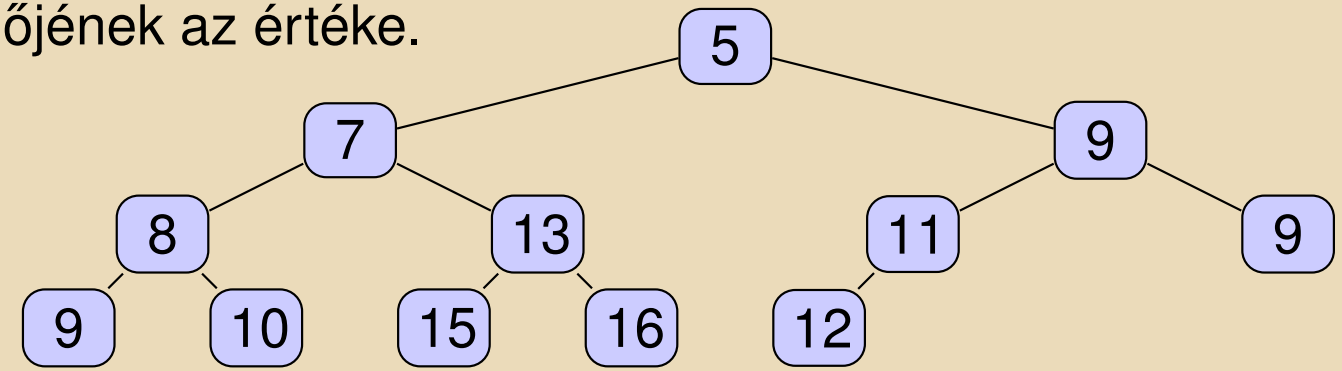
- *Bináris kupacban az* `insert()`, `decrease-min()` *és* `delete-min()` *műveletek* $O(\log n)$ *idejűek.*
- *d-kupacban kupacban az* `insert()` *és* `decrease-min()` *műveletek* $O(\log_d n)$ *idejűek, a* `delete-min()` *pedig* $O(d \log_d n)$.

Egy ilyen adatstruktúra tömbben tárolható!

Bináris és d-Kupac

Egy kiegyensúlyozott teljes bináris (vagy d-leszármazottas) fa a következő invariáns tulajdonsággal:

- A fa minden csúcsában levő érték legalább akkora, mint a szülőjének az értéke.



Tétel

- *Bináris kupacban az* `insert()`, `decrease-min()` *és* `delete-min()` *műveletek* $O(\log n)$ *idejűek.*
- *d-kupacban kupacban az* `insert()` *és* `decrease-min()` *műveletek* $O(\log_d n)$ *idejűek, a* `delete-min()` *pedig* $O(d \log_d n)$.

Egy ilyen adatstruktúra tömbben tárolható!

Dijkstra algoritmus futásideje II.

Tétel (Dijkstra futásideje heap adatstruktúrával)

A Dijkstra algoritmus futásideje $O(nT_M + mT_K)$, ahol T_M egy min. kulcsú elem kiválasztásának, T_K egy kulcscsökkentésnek az ideje.

- *Bináris kupac ($T_M = O(\log n)$, $T_K = O(\log n)$) esetén: $O(m \log n)$*
- *d-kupac ($T_M = O(n \log_d n)$, $T_K = O(\log_d n)$ esetén $O(m \log_d n + nd \log_d n)$*
  - *$d := \max\{2, \lceil m/n \rceil\}$ választással: $O(m \log_d n)$*
  - *Ritka gráfra ($m = O(n)$): $O(n \log n)$*
  - *Sűrű gráfra ($m = \Omega(n^{1+\varepsilon})$): $O(m)$*
- *Fibonacci kupac ($T_M = O(\log n)$, $T_K = O(1)$) esetén: $O(m + n \log n)$*
- *Johnson kupac ($T_M = T_K = O(\log \log C)$) esetén: $O(m \log \log C)$*

Dijkstra alg. kulccsökkentés nélkül – gyakorlati javítás

Dijkstra algoritmus

```
 1: procedure DIJKSTRA(G = (N, A), c, s)
 2:     create-heap(H)
 3:     for ∀i ∈ N, i ≠ s do
 4:         d(i) := ∞
 5:     end for
 6:     d(s) := 0, pred(s) := 0
 7:     insert(s,H)
 8:     while H ≠ ∅ do
 9:         find-min(i,H), delete-min(H)
10:         for i : (i, j) ∈ δ(i) do
11:             val := d(i) + c_ij
12:             if val < d(j) then
13:                 d(j) := val
14:                 pred(j) := (i, j)
15:                 if j ∈ H then
16:                     decrease-key(val,j,H)
17:                 else
18:                     insert(j,H)
19:                 end if
20:                 end if
21:             end for
22:         end while
23:     end procedure
```

Dijkstra alg. kulccsökkentés nélkül – gyakorlati javítás

Dijkstra algoritmus kulccsökkentés nélkül

```
 1: procedure DIJKSTRA(G = (N, A), c, s)
 2:     create-heap(H)
 3:     for ∀i ∈ N, i ≠ s do
 4:         d(i) := ∞
 5:     end for
 6:     d(s) := 0, pred(s) := 0
 7:     insert(s,H)
 8:     while H ≠ ∅ do
 9:         find-min(i,H), delete-min(H)
10:         if pred(i) ≠ 0 then
11:             for i : (i, j) ∈ δ(i) do
12:                 val := d(i) + c_ij
13:                 if val < d(j) then
14:                     d(j) := val
15:                     pred(j) := (i, j)
16:                     insert(j,H)
17:                 end if
18:             end for
19:         end if
20:     end while
21: end procedure
```

Legrövidebb utak általános esetben

Tétel (Legrövidebb út optimalitási feltétel)

Legyen adott $s \in N$, és minden $j \in N$ csúcsra p_j legyen egy tetszőleges $s \rightsquigarrow j$ út, hosszát pedig jelölje $d(j)$. A p_j utak pontosan akkor legrövidebbek, ha

$$d(j) \leq d(i) + c_{ij} \tag{2}$$

teljesül minden $(i, j) \in A$ élre.

Tétel

Legyen $d : N \longrightarrow \mathbb{R}$ függvény és $c_{ij}^d := c_{ij} + d(i) - d(j)$. Ekkor

- *Minden W irányított körre (körsétára) $\sum_{(i,j)\in W} c_{ij}^d = \sum_{(i,j)\in W} c_{ij}$*
- *Ha P egy $k \rightsquigarrow l$ út (séta), akkor $\sum_{(i,j)\in P} c_{ij}^d = \sum_{(i,j)\in P} c_{ij} + d(k) - d(l)$*
- *Ha $d(\cdot)$ éppen a legrövidebb $s \rightsquigarrow \cdot$ utak hossza, akkor $c_{ij}^d \geq 0$ minden $(i, j) \in A$ élre.*

Legrövidebb utak általános esetben

Tétel (Legrövidebb út optimalitási feltétel)

Legyen adott $s \in N$, és minden $j \in N$ csúcsra p_j legyen egy tetszőleges $s \rightsquigarrow j$ út, hosszát pedig jelölje $d(j)$. A p_j utak pontosan akkor legrövidebbek, ha

$$d(j) \leq d(i) + c_{ij} \tag{2}$$

teljesül minden $(i, j) \in A$ élre.

Tétel

Legyen $d : N \longrightarrow \mathbb{R}$ függvény és $c_{ij}^d := c_{ij} + d(i) - d(j)$. Ekkor

- *Minden W irányított körre (körsétára) $\sum_{(i,j)\in W} c_{ij}^d = \sum_{(i,j)\in W} c_{ij}$*
- *Ha P egy $k \rightsquigarrow l$ út (séta), akkor $\sum_{(i,j)\in P} c_{ij}^d = \sum_{(i,j)\in P} c_{ij} + d(k) - d(l)$*
- *Ha $d(\cdot)$ éppen a legrövidebb $s \rightsquigarrow \cdot$ utak hossza, akkor $c_{ij}^d \geq 0$ minden $(i, j) \in A$ élre.*

Cimkejavító algoritmusok

Általános cimkejavító algoritmus

1: **procedure** LABELCORRECTING(G, c, s)
2: $d(s) := 0$, $pred(s) := 0$
3: $d(j) := \infty \quad \forall j \in N$
4: **while** $\exists (i, j) \in A, d(j) > d(i) + c_{ij}$ **do**
5: $d(j) := d(i) + c_{ij}$
6: $pred(j) := i$
7: **end while**
8: **end procedure**

Állítás

Amennyiben az algoritmus megáll, $d(j)$ a legrövidebb $s \rightsquigarrow j$ út hossza lesz minden $j \in N$-re.

Tétel

Az algoritmus futása legfeljebb $2n^2C$ iteráció után véget ér.

Cimkejavító algoritmusok

Általános cimkejavító algoritmus

1: **procedure** LABELCORRECTING(G, c, s)
2: $d(s) := 0$, $pred(s) := 0$
3: $d(j) := \infty \quad \forall j \in N$
4: **while** $\exists (i,j) \in A, d(j) > d(i) + c_{ij}$ **do**
5: $d(j) := d(i) + c_{ij}$
6: $pred(j) := i$
7: **end while**
8: **end procedure**

Állítás

Amennyiben az algoritmus megáll, $d(j)$ a legrövidebb $s \rightsquigarrow j$ út hossza lesz minden $j \in N$-re.

Tétel

Az algoritmus futása legfeljebb $2n^2C$ iteráció után véget ér.

Cimkejavító algoritmusok

Általános cimkejavító algoritmus

1: **procedure** LABELCORRECTING(G, c, s)
2: $d(s) := 0$, $pred(s) := 0$
3: $d(j) := \infty \quad \forall j \in N$
4: **while** $\exists (i,j) \in A, d(j) > d(i) + c_{ij}$ **do**
5: $d(j) := d(i) + c_{ij}$
6: $pred(j) := i$
7: **end while**
8: **end procedure**

Állítás

Amennyiben az algoritmus megáll, $d(j)$ a legrövidebb $s \rightsquigarrow j$ út hossza lesz minden $j \in N$-re.

Tétel

Az algoritmus futása legfeljebb $2n^2C$ iteráció után véget ér.

Cimkejavító algoritmusok

Módosított cimkejavító algoritmus

1: **procedure** MODIFIEDLABELCORRECTING(G, c, s)
2: Legyen
3: $d(s) := 0$, $pred(s) := 0$
4: $d(j) := \infty \quad \forall j \in N$
5: $LIST = \{s\}$
6: **while** $LIST \neq \emptyset$ **do**
7: Legyen $i \in LIST$, $LIST := LIST - i$
8: **for** $(i, j) \in \delta(i)$ **do**
9: **if** $d(j) > d(i) + c_{ij}$ **then**
10: $d(j) := d(i) + c_{ij}$
11: $pred(j) := i$
12: **if** $j \notin LIST$ **then**
13: $LIST := LIST + j$
14: **end if**
15: **end if**
16: **end for**
17: **end while**
18: **end procedure**

Cimkejavító algoritmusok

Módosított cimkejavító algoritmus

```
1: procedure ModifiedLabelCorrecting(G, c, s)
2:     Legyen
3:     d(s) := 0, pred(s) := 0
4:     d(j) := ∞   ∀j ∈ N
5:     LIST = {s}
6:     while LIST ≠ ∅ do
7:         Legyen i ∈ LIST, LIST := LIST − i
8:         for (i, j) ∈ δ(i) do
9:             if d(j) > d(i) + c_ij then
10:                d(j) := d(i) + c_ij
11:                pred(j) := i
12:                if j ∉ LIST then
13:                    LIST := LIST + j
14:                end if
15:            end if
16:        end for
17:    end while
18: end procedure
```

Tétel

Az algoritmus futása $O(nmC)$ lépésben véget ér.

Cimkejavító algoritmusok

Tétel

A fenti algoritmusok futása tetszőleges valós távolságok esetén is véges ($O(2^n)$) lépésben véget ér.

Házi feladat 3/4.

Mutass példát, amin elképzelhető, hogy az fenti algoritmusok $\Omega(2^n)$ iterációt végeznek.

Erősen polinomiális ($O(nm)$) cimkejavító algoritmus

Minden k-ra és i-re számoljuk ki a legrövidebb legfeljebb k élből álló utat.

Általános cimkejavító algoritmus

1: **procedure** LABELCORRECTING(G, c, s)
2: $d(s) := 0$, $pred(s) := 0$
3: $d(j) := \infty \quad \forall j \in N$
4: **while** $\exists (i,j) \in A, d(j) > d(i) + c_{ij}$ **do**
5: $d(j) := d(i) + c_{ij}$
6: $pred(j) := i$
7: **end while**
8: **end procedure**

Erősen polinomiális ($O(nm)$) cimkejavító algoritmus

Minden k-ra és i-re számoljuk ki a legrövidebb legfeljebb k élből álló utat.

Bellman-Ford algoritmus

```
1:  procedure BellmanFord(G, c, s)
2:      d(s) := 0, pred(s) := 0
3:      d(j) := ∞   ∀j ∈ N
4:      for k ∈ {1, ..., n} do
5:          for (i, j) ∈ A do
6:              if d(j) > d(i) + c_ij then
7:                  d(j) := d(i) + c_ij
8:                  pred(j) := i
9:              end if
10:         end for
11:     end for
12: end procedure
```

Erősen polinomiális ($O(nm)$) cimkejavító algoritmus

Minden k-ra és i-re számoljuk ki a legrövidebb legfeljebb k élből álló utat.

Bellman-Ford algoritmus

```
1:  procedure BellmanFord(G, c, s)
2:      d(s) := 0, pred(s) := 0
3:      d(j) := ∞   ∀j ∈ N
4:      for k ∈ {1, ..., n} do
5:          for (i, j) ∈ A do
6:              if d(j) > d(i) + c_ij then
7:                  d(j) := d(i) + c_ij
8:                  pred(j) := i
9:              end if
10:         end for
11:     end for
12: end procedure
```

Tétel

- *Az algoritmus futásaideje: $O(nm)$*
- *a futása végén a $d(\cdot)$ értékek a legrövidebb utak hosszai lesznek.*

Erősen polinomiális ($O(nm)$) cimkejavító algoritmus

Minden k-ra és i-re számoljuk ki a legrövidebb legfeljebb k élből álló utat.

Bellman-Ford algoritmus II.

A módosított cimkejavító algoritmusban a *LIST* adatszerkezet legyen egy FIFO sor (queue).

Erősen polinomiális ($O(nm)$) cimkejavító algoritmus

Minden k-ra és i-re számoljuk ki a legrövidebb legfeljebb k élből álló utat.

Bellman-Ford algoritmus II.

A módosított cimkejavító algoritmusban a *LIST* adatszerkezet legyen egy FIFO sor (queue).

Tétel

- *Az algoritmus „ugyanolyan sorrendben” dolgozza fel az éleket, mint az eredeti változat, így a futásideje $O(nm)$.*

Erősen polinomiális ($O(nm)$) cimkejavító algoritmus

Minden k-ra és i-re számoljuk ki a legrövidebb legfeljebb k élből álló utat.

Bellman-Ford algoritmus II.

A módosított cimkejavító algoritmusban a *LIST* adatszerkezet legyen egy FIFO sor (queue).

Tétel

- *Az algoritmus „ugyanolyan sorrendben” dolgozza fel az éleket, mint az eredeti változat, így a futásideje $O(nm)$.*

Házi feladat 3/5.

Adott $G = (N, A)$ irányított gráf, $c : A \longrightarrow \mathbb{R}$ hosszfüggvény az éleken. Adjunk (erősen polinomiális) algoritmust annak eldöntésére, hogy a gráf tartalmaz-e negatív kört.

Legrövidebb út minden pontpár között

Nemnegatív élek esetén

Futtassuk le minden pontból a Diskstra algoritmust.
Futásidő: $O(nm + n^2 logn)$

Általános esetben

Futtassuk le minden pontból a Bellman-Ford algoritmust.
Futásidő: $O(n^2 m)$

Legrövidebb út minden pontpár között

Nemnegatív élek esetén

Futtassuk le minden pontból a Diskstra algoritmust.
Futásidő: $O(nm + n^2 logn)$

Általános esetben

Futtassuk le minden pontból a Bellman-Ford algoritmust.
Futásidő: $O(n^2 m)$

Legrövidebb út minden pontpár között

Nemnegatív élek esetén

Futtassuk le minden pontból a Diskstra algoritmust.
Futásidő: $O(nm + n^2 logn)$

Általános esetben

Futtassuk le minden pontból a Bellman-Ford algoritmust.
Futásidő: $O(n^2 m)$

Általános esetben: Johnson algoritmus

- Futtassuk le egy pontból a Bellman-Ford algoritmust.
  Ekkor a $c_{ij}^d := c_{ij} + d(i) - d(j)$ nemnegatív
- Futtassuk le a többi pontból a c^d hosszfüggvényre a Dijkstra algoritmust.

Futásidő: $O(nm + n^2 logn)$

Floyd-Warshall Algoritmus

Dinamikus programozás (ez még nem a Floyd-Warshall)

$d^k(i,j)$ a legrövidebb legfeljebb k élből álló $i \rightsquigarrow j$ út hossza. Ezt számítsuk ki minden k-ra.

- Futásidő: $O(n^2 m)$
- „Mátrix-szorzás”: $O(n^3 \log n)$

Floyd-Warshall Algoritmus

Floyd-Warshall Algoritmus

Legyen $d^k(i,j)$ a legrövidebb olyan $i \rightsquigarrow j$ út hossza, ami belső pontként csak az $\{1,\ldots,k-1\}$ csúcsokat tartalmazza.

```
 1: procedure FloydWarshall(G, c)
 2:     d(i,j) := ∞ és pred(i,j) := 0   ∀i,j ∈ N-re
 3:     d(i,i) := 0   ∀i ∈ N-re
 4:     for (i,j) ∈ A do
 5:         d(i,j) := c_ij és pred(i,j) := i
 6:     end for
 7:     for k ∈ {1,...,n} do
 8:         for (i,j) ∈ N × N do
 9:             if d(i,j) > d(i,k) + d(k,j) then
10:                 d(i,j) := d(i,k) + d(k,j)
11:                 pred(i,j) := pred(k,j)
12:             end if
13:         end for
14:     end for
15: end procedure
```

Futásidő: $O(n^3)$

Feladat

Házi feladat 4/1.

Adott $G = (N, A)$ irányított gráf $c_1, c_2 : A \longrightarrow \mathbb{Z}(\mathbb{R})$ súlyfüggvény az éleken, és $s, t \in N$ pontok. Tegyük fel továbbá, hogy nincs c_1 szerint negatív kör a gráfban.

Feladat: Keressünk a c_1-minimális $s \rightsquigarrow t$ utak között egy c_2 minimálisat. Azaz keressük

$$\min\{c_2(p) : p \in \mathcal{P}_{c_1}\}\text{-t},$$

ahol

$$\mathcal{P}_{c_1} := \{p : p \text{ egy } c_1\text{-minimális } s \rightsquigarrow t \text{ út}\}.$$

Feladatok (legrövidebb utak)

Házi feladat 4/2.

Adott egy $G = (N, E)$ irányítatlan (vagy irányított) gráf és egy $c : E \longrightarrow \mathbb{R}$ súlyfüggvény. Egy út *szélességén* az út minimális élsúlyát értjük, azaz

$$\text{szélesség}(P) := \min\{c(e) : e \in P\}$$

Feladat: keressük a maximális szélességű utat

- egy adott $s, t \in N$ pontpár között
- egy adott $s \in N$ pontból az összes többi pontba
- minden pontpárra

Házi feladat 4/3.

Adott egy $G = (N, E)$ irányított gráf és $c : E \longrightarrow \mathbb{R}$ súlyfüggvény az éleken. Tegyük fel, hogy nincs a gráfban negatív kör. Egy $(i, j) \in E$ élre jelölje f_{ij} azt a maximális értéket, amivel az (i, j) él súlyát csökkentve a gráfban továbbra sem lesz negatív kör.
Adjunk hatékony algoritmust az összes f_{ij} érték meghatározására.

Legrövidebb utak „térképeken”

Kétirányú Dijkstra algoritmus

Párhuzamosan futtatjuk a Dijkstra algoritmust s-ből és t-ből visszafele. Megállunk amikor a két futtatás összeér.

Házi feladat 4/4. Meddig futtassuk az algoritmust?

Legyen S_s az s-ből, S_t pedig a t-ből visszafele indított Dijkstra algoritmus által feldolgozott pontok halmaza és tegyük fel, hogy $S_s \cap S_t = \{u\}$. Legyen $p_{s,u}$ és $p_{u,t}$ a legrövidebb $s \rightsquigarrow u$ ill. $u \rightsquigarrow t$ út. Igaz-e, hogy $p_{s,u}p_{u,t}$ egy legrövidebb $s \rightsquigarrow t$ út?

A^* algoritmus

Legyen $c'_{u,v} := c_{u,v} - \pi(u) + \pi(v)$, ahol $\pi(u)$ az $u \longrightarrow t$ távolság légvonalban.

Állítás

$$c'_{u,v} \geq 0 \quad \forall (u,v) \in A$$

Futtassuk a Dijkstra algoritmust a c' súlyfüggvénnyel!

Legrövidebb utak „térképeken”

Kétirányú Dijkstra algoritmus

Párhuzamosan futtatjuk a Dijkstra algoritmust s-ből és t-ből visszafele. Megállunk amikor a két futtatás összeér.

Házi feladat 4/4. Meddig futtassuk az algoritmust?

Legyen S_s az s-ből, S_t pedig a t-ből visszafele indított Dijkstra algoritmus által feldolgozott pontok halmaza és tegyük fel, hogy $S_s \cap S_t = \{u\}$. Legyen $p_{s,u}$ és $p_{u,t}$ a legrövidebb $s \rightsquigarrow u$ ill. $u \rightsquigarrow t$ út. Igaz-e, hogy $p_{s,u}p_{u,t}$ egy legrövidebb $s \rightsquigarrow t$ út?

A^* algoritmus

Legyen $c'_{u,v} := c_{u,v} - \pi(u) + \pi(v)$, ahol $\pi(u)$ az $u \longrightarrow t$ távolság légvonalban.

Állítás

$$c'_{u,v} \geq 0 \quad \forall (u,v) \in A$$

Futtassuk a Dijkstra algoritmust a c' súlyfüggvénnyel!

Törtoptimalizálás

Minimális átlagú körök

Feladat

Adott egy $G = (N, A)$ irányított gráf, ami tartalmaz irányított kört és adott egy tetszőleges $c : A \longrightarrow \mathbb{R}$ súlyozás az éleken.

- Határozzunk meg a minimális körátlagot, azaz

$$\lambda_1^* := \min \left\{ \frac{c(C)}{|C|} : C \subseteq A \text{ egy irányított kör} \right\}$$

- Határozzuk meg azt a minimális λ_2^* értéket, amivel minden él súlyát egységesen csökkentve még éppen nincs negatív kör. (λ_2^* lehet negatív is...)

Állítás

$\lambda_1^* = \lambda_2^*$

Minimális átlagú körök

Ötlet

(Először csökkentsük le minden él költségét annyival, hogy bizonyosan legyen negatív kör)

- Keressünk egy negatív kört
- Növeljük meg az élek súlyát ennek a körnek az átlagával
- Ismételjük a fenti lépéseket addig, amíg van negatív kör

Állítás

A fenti eljárás véges, de exponenciális sok iterációra lehet szükség.

Minimális átlagú körök

Ötlet

(Először csökkentsük le minden él költségét annyival, hogy bizonyosan legyen negatív kör)

- Keressünk egy negatív kört
- Növeljük meg az élek súlyát ennek a körnek az átlagával
- Ismételjük a fenti lépéseket addig, amíg van negatív kör

Állítás

A fenti eljárás véges, de exponenciális sok iterációra lehet szükség.

Minimális átlagú körök

Ötlet

(Először csökkentsük le minden él költségét annyival, hogy bizonyosan legyen negatív kör)

- Keressünk egy minimális súlyú kört
- Növeljük meg az élek súlyát ennek a körnek az átlagával
- Ismételjük a fenti lépéseket addig, amíg van negatív kör

Állítás

- *A fenti n iteráció után véget ér.*
- *A minimális súlyú kör keresése $\mathcal{NP}$-teljes feladat*

Minimális átlagú körök

Ötlet

(Először csökkentsük le minden él költségét annyival, hogy bizonyosan legyen negatív kör)

- Keressünk egy minimális súlyú legfeljebb n hosszú körsétát
- Növeljük meg az élek súlyát ennek a körnek az átlagával
- Ismételjük a fenti lépéseket addig, amíg van negatív kör

Állítás

- *A fenti n iteráció után véget ér.*
- *A minimális súlyú legfeljebb n hosszú körsétát dinamikus programozással polinom időben tudunk keresni.*
- *Futásidő:* $O(n^2 m)$

Minimális átlagú körök

Ötlet

(Először csökkentsük le minden él költségét annyival, hogy bizonyosan legyen negatív kör)

- Keressünk egy minimális súlyú legfeljebb n hosszú körsétát
- Növeljük meg az élek súlyát ennek a körnek az átlagával
- Ismételjük a fenti lépéseket addig, amíg van negatív kör

Állítás

- *A fenti n iteráció után véget ér.*
- *A minimális súlyú legfeljebb n hosszú körsétát dinamikus programozással polinom időben tudunk keresni.*
- *Futásidő:* $O(n^2 m)$

Házi feladat 4/4.

A minimális körátlagot közvetlenül is meg tudjuk dinamikus programozással határozni.

Feladat

Házi feladat 5/1. Segítség az előzőhöz

Adott egy $G = (N, A)$ irányított gráf és egy $c : E \longrightarrow \mathbb{R}$ súlyfüggvény az éleken. Tegyük fel, hogy s-ből minden pont elérhető irányított úton. Jelölje a $d^k(j)$ érték a minimális súlyú pontosan k élből álló $s \rightsquigarrow j$ séta hosszát. Tegyük fel, hogy $d^k(j)$ ismert minden $j \in N$ csúcsra és $k = 1, \ldots n$-re. Határozzuk meg a minimális körátlagot ennek segítségével.

Minimális súlyozott átlagú körök

Feladat

Adott egy $G = (N, A)$ irányított gráf, amit tartalmaz irányított kört és adott egy tetszőleges $c : A \longrightarrow \mathbb{R}$ és egy pozitív $w : A \longrightarrow \mathbb{R}_+$ súlyozás az éleken.

- Határozzunk meg a minimális súlyozott átlagú kört, azaz

$$\lambda_1^* := \min \left\{ \frac{c(C)}{w(C)} : C \subseteq A \text{ egy irányított kör} \right\}$$

- Határozzuk meg a

$$\lambda_2^* := \max \{\lambda \in \mathbb{R} : \forall C \subseteq A \text{ irányított körre } c(C) - \lambda w(C) \geq 0\}$$

értéket

Állítás

$\lambda_1^* = \lambda_2^*$

Algoritmus ötlet

Legyen $c_\lambda(e) := c(e) - \lambda w(e)$.

1. eset *Van egy c_λ negatív C_λ kör.* Ekkor $\boxed{\lambda > \lambda^*}$
2. eset *Nincs c_λ negatív kör, de van 0 hosszú C_λ.* Ekkor $\boxed{\lambda = \lambda^*}$, és ez a kör a súlyozott átlag szerint minimális.
3. eset *Minden kör c_λ szerinti hossza pozitív.* Ekkor $\boxed{\lambda < \lambda^*}$

Következmény

Egy adott λ értékre el tudjuk dönteni, hogy az a min. súlyozott átlagnál kisebb, nagyobb, vagy egyenlő-e vele. Ez utóbbi esetben megkapjuk a min súlyozott átlagú kört is.

Algoritmus ötletek

- Szekvenciális keresés: Ha $\lambda_i > \lambda^*$, akkor legyen $\lambda_{i+1} := \frac{c(C_{\lambda_i})}{w(C_{\lambda_i})}$.
- Bináris keresés (intervallum-felezés)

Tétel

Legyen c és w egészértékű, ekkor a fenti bináris kereséssel polinom időben meghatározhatjuk a súlyozott körátlagot.

Algoritmus ötlet

Legyen $c_\lambda(e) := c(e) - \lambda w(e)$.

1. eset *Van egy c_λ negatív C_λ kör.* Ekkor $\boxed{\lambda > \lambda^*}$
2. eset *Nincs c_λ negatív kör, de van 0 hosszú C_λ.* Ekkor $\boxed{\lambda = \lambda^*}$, és ez a kör a súlyozott átlag szerint minimális.
3. eset *Minden kör c_λ szerinti hossza pozitív.* Ekkor $\boxed{\lambda < \lambda^*}$

Következmény

Egy adott λ értékre el tudjuk dönteni, hogy az a min. súlyozott átlagnál kisebb, nagyobb, vagy egyenlő-e vele. Ez utóbbi esetben megkapjuk a min súlyozott átlagú kört is.

Algoritmus ötletek

- Szekvenciális keresés: Ha $\lambda_i > \lambda^*$, akkor legyen $\lambda_{i+1} := \frac{c(C_{\lambda_i})}{w(C_{\lambda_i})}$.
- Bináris keresés (intervallum-felezés)

Tétel

Legyen c és w egészértékű, ekkor a fenti bináris kereséssel polinom időben meghatározhatjuk a súlyozott körátlagot.

Algoritmus ötlet

Legyen $c_\lambda(e) := c(e) - \lambda w(e)$.

1. eset *Van egy c_λ negatív C_λ kör.* Ekkor $\boxed{\lambda > \lambda^*}$
2. eset *Nincs c_λ negatív kör, de van 0 hosszú C_λ.* Ekkor $\boxed{\lambda = \lambda^*}$, és ez a kör a súlyozott átlag szerint minimális.
3. eset *Minden kör c_λ szerinti hossza pozitív.* Ekkor $\boxed{\lambda < \lambda^*}$

Következmény

Egy adott λ értékre el tudjuk dönteni, hogy az a min. súlyozott átlagnál kisebb, nagyobb, vagy egyenlő-e vele. Ez utóbbi esetben megkapjuk a min súlyozott átlagú kört is.

Algoritmus ötletek

- Szekvenciális keresés: Ha $\lambda_i > \lambda^*$, akkor legyen $\lambda_{i+1} := \frac{c(C_{\lambda_i})}{w(C_{\lambda_i})}$.
- Bináris keresés (intervallum-felezés)

Tétel

Legyen c és w egészértékű, ekkor a fenti bináris kereséssel polinom időben meghatározhatjuk a súlyozott körátlagot.

Törtoptimalizálás

Feladat

Adott egy $\mathcal{X}$ alaphalmaz valamint adott egy tetszőleges $c : \mathcal{X} \longrightarrow \mathbb{R}$ és egy pozitív $w : \mathcal{X} \longrightarrow \mathbb{R}_+$ költségfüggvény.

- Határozzunk meg a

$$\lambda_1^* := \min \left\{ \frac{c(X)}{w(X)} : C \in \mathcal{X} \right\}$$

  értéket
- Határozzuk meg a

$$\lambda_2^* := \max \{ \lambda \in \mathbb{R} : \forall X \in \mathcal{X}\text{-re } c(X) - \lambda w(X) \geq 0 \}$$

  értéket

Állítás

$\lambda_1^* = \lambda_2^*$

Algoritmus ötlet

Legyen $c_\lambda(e) := c(e) - \lambda w(e)$.

1. eset *Van egy c_λ negatív $X_\lambda \in \mathcal{X}$.* Ekkor $\boxed{\lambda > \lambda^*}$
2. eset *Nincs c_λ negatív, de van 0 hosszú $X_\lambda \in \mathcal{X}$.* Ekkor $\boxed{\lambda = \lambda^*}$, és ez a súlyozott átlag szerint minimális.
3. eset *Minden $X \in \mathcal{X}$ c_λ szerinti súlya pozitív.* Ekkor $\boxed{\lambda < \lambda^*}$

Következmény

Egy adott λ értékre el tudnuk dönteni, hogy az a min. súlyozott átlagnál kisebb, nagyobb, vagy egyenlő-e vele. Ez utóbbi esetben megkapjuk a min súlyozott átlagú $X \in \mathcal{X}$-et is.

Algoritmus ötletek

- Szekvenciális keresés: Ha $\lambda_i > \lambda^*$, akkor legyen $\lambda_{i+1} := \frac{c(X_{\lambda_i})}{w(X_{\lambda_i})}$.
- Bináris keresés (intervallum-felezés)

Tétel

Legyen c és w egészértékű, ekkor a fenti bináris kereséssel polinom időben meghatározhatjuk a minimális súlyozott átlagot.

Algoritmus ötlet

Legyen $c_\lambda(e) := c(e) - \lambda w(e)$.

1. eset *Van egy c_λ negatív $X_\lambda \in \mathcal{X}$.* Ekkor $\boxed{\lambda > \lambda^*}$
2. eset *Nincs c_λ negatív, de van 0 hosszú $X_\lambda \in \mathcal{X}$.* Ekkor $\boxed{\lambda = \lambda^*}$, és ez a súlyozott átlag szerint minimális.
3. eset *Minden $X \in \mathcal{X}$ c_λ szerinti súlya pozitív.* Ekkor $\boxed{\lambda < \lambda^*}$

Következmény

Egy adott λ értékre el tudnuk dönteni, hogy az a min. súlyozott átlagnál kisebb, nagyobb, vagy egyenlő-e vele. Ez utóbbi esetben megkapjuk a min súlyozott átlagú $X \in \mathcal{X}$-et is.

Algoritmus ötletek

- Szekvenciális keresés: Ha $\lambda_i > \lambda^*$, akkor legyen $\lambda_{i+1} := \frac{c(X_{\lambda_i})}{w(X_{\lambda_i})}$.
- Bináris keresés (intervallum-felezés)

Tétel

Legyen c és w egészértékű, ekkor a fenti bináris kereséssel polinom időben meghatározhatjuk a minimális súlyozott átlagot.

Algoritmus ötlet

Legyen $c_\lambda(e) := c(e) - \lambda w(e)$.

1. eset *Van egy c_λ negatív $X_\lambda \in \mathcal{X}$.* Ekkor $\boxed{\lambda > \lambda^*}$
2. eset *Nincs c_λ negatív, de van 0 hosszú $X_\lambda \in \mathcal{X}$.* Ekkor $\boxed{\lambda = \lambda^*}$, és ez a súlyozott átlag szerint minimális.
3. eset *Minden $X \in \mathcal{X}$ c_λ szerinti súlya pozitív.* Ekkor $\boxed{\lambda < \lambda^*}$

Következmény

Egy adott λ értékre el tudnuk dönteni, hogy az a min. súlyozott átlagnál kisebb, nagyobb, vagy egyenlő-e vele. Ez utóbbi esetben megkapjuk a min súlyozott átlagú $X \in \mathcal{X}$-et is.

Algoritmus ötletek

- Szekvenciális keresés: Ha $\lambda_i > \lambda^*$, akkor legyen $\lambda_{i+1} := \frac{c(X_{\lambda_i})}{w(X_{\lambda_i})}$.
- Bináris keresés (intervallum-felezés)

Tétel

Legyen c és w egészértékű, ekkor a fenti bináris kereséssel polinom időben meghatározhatjuk a minimális súlyozott átlagot.

Algoritmus ötlet I.

Feltevés I.

Egy adott $\lambda \in \mathbb{R}$-re meg tudjuk határozni

$$\min\{c(X) - \lambda w(X) : X \in \mathcal{X}\} \tag{3}$$

előjelét.

Szekvenciális keresés

Adott egy $X \in \mathcal{X}$. Döntsük el, hogy optimális-e, ha nem, akkor keressünk egy $X' \in \mathcal{X}$-et, amire

$$\frac{c(X')}{w(X')} < \frac{c(X)}{w(X)}. \tag{4}$$

Azaz keresünk egy $X' \in \mathcal{X}$-et, amire

$$c(X') - \frac{c(X)}{w(X)} w(X') < 0 \tag{5}$$

És ezt iteráljuk amíg el nem érjük az optimumot.

Algoritmus ötlet I.

Feltevés I.

Egy adott $\lambda \in \mathbb{R}$-re meg tudjuk határozni

$$\min\{c(X) - \lambda w(X) : X \in \mathcal{X}\} \tag{3}$$

előjelét.

Szekvenciális keresés

Adott egy $X \in \mathcal{X}$. Döntsük el, hogy optimális-e, ha nem, akkor keressünk egy $X' \in \mathcal{X}$-et, amire

$$\frac{c(X')}{w(X')} < \frac{c(X)}{w(X)}. \tag{4}$$

Azaz keresünk egy $X' \in \mathcal{X}$-et, amire

$$c(X') - \frac{c(X)}{w(X)} w(X') < 0 \tag{5}$$

És ezt iteráljuk amíg el nem érjük az optimumot.

Algoritmus ötlet I.

Feltevés I.

Egy adott $\lambda \in \mathbb{R}$-re meg tudjuk határozni

$$\min\{c(X) - \lambda w(X) : X \in \mathcal{X}\} \tag{3}$$

előjelét.

Szekvenciális keresés

Adott egy $X \in \mathcal{X}$. Döntsük el, hogy optimális-e, ha nem, akkor keressünk egy $X' \in \mathcal{X}$-et, amire

$$\frac{c(X')}{w(X')} < \frac{c(X)}{w(X)}. \tag{4}$$

Azaz keresünk egy $X' \in \mathcal{X}$-et, amire

$$c(X') - \frac{c(X)}{w(X)} w(X') < 0 \tag{5}$$

És ezt iteráljuk amíg el nem érjük az optimumot.

Algoritmus ötlet II.

Feltevés II.

Egy adott $\lambda \in \mathbb{R}$-re meg tudjuk határozni a

$$\min\{c(X) - \lambda w(X) : X \in \mathcal{X}\} \tag{6}$$

értéket és egy minimumhelyet.

Javított szekvenciális keresés

Adott egy $X_i \in \mathcal{X}$. Allapítsuk meg a

$$H = \min\{c(X') - \frac{c(X_i)}{w(X_i)} w(X') : X' \in \mathcal{X}\} \tag{7}$$

értéket és egy $X' \in \mathcal{X}$ minimumhelyet. Ha $H < 0$ (azaz X_i nem optimális), akkor $X_{i+1} := X'$-vel iteráljuk az eljárást.

Algoritmus ötlet II.

Feltevés II.

Egy adott $\lambda \in \mathbb{R}$-re meg tudjuk határozni a

$$\min\{c(X) - \lambda w(X) : X \in \mathcal{X}\} \tag{6}$$

értéket és egy minimumhelyet.

Javított szekvenciális keresés

Adott egy $X_i \in \mathcal{X}$. Allapítsuk meg a

$$H = \min\{c(X') - \frac{c(X_i)}{w(X_i)} w(X') : X' \in \mathcal{X}\} \tag{7}$$

értéket és egy $X' \in \mathcal{X}$ minimumhelyet. Ha $H < 0$ (azaz X_i nem optimális), akkor $X_{i+1} := X'$-vel iteráljuk az eljárást.

Newton-Dinkelbach eljárás

Definíció

$$h(\lambda) := \min\{c(x) - \lambda w(x) : x \in \mathcal{X}\}$$
$$x_\lambda := \mathit{argmin}\{c(x) - \lambda w(x) : x \in \mathcal{X}\}$$
$$c_\lambda(x) := c(x) - \lambda w(x)$$

Newton-Dinkelbach eljárás

Definíció

$$h(\lambda) := \min\{c(x) - \lambda w(x) : x \in \mathcal{X}\}$$
$$x_\lambda := \mathit{argmin}\{c(x) - \lambda w(x) : x \in \mathcal{X}\}$$
$$c_\lambda(x) := c(x) - \lambda w(x)$$

Állítás

$h(\lambda)$ szigorúan monoton fogyó, szakaszonként lineáris, konváv függvény.

Newton-Dinkelbach eljárás

Definíció

$$h(\lambda) := \min\{c(x) - \lambda w(x) : x \in \mathcal{X}\}$$
$$x_\lambda := \mathit{argmin}\{c(x) - \lambda w(x) : x \in \mathcal{X}\}$$
$$c_\lambda(x) := c(x) - \lambda w(x)$$

Állítás

$h(\lambda)$ szigorúan monoton fogyó, szakaszonként lineáris, konváv függvény.

Feladat

Keressük a $h(\lambda)$ nullhelyét, azaz a

$$\lambda^* := \min\{\lambda \in \mathbb{R} : h(\lambda) \leq 0\}$$

értéket és a hozzá tartozó x_{λ^*} elemet.

Newton-Dinkelbach eljárás

Definíció

$$h(\lambda) := \min\{c(x) - \lambda w(x) : x \in \mathcal{X}\}$$
$$x_\lambda := \mathit{argmin}\{c(x) - \lambda w(x) : x \in \mathcal{X}\}$$
$$c_\lambda(x) := c(x) - \lambda w(x)$$

1: **procedure** NEWTONDINKELBACH($c, w, \mathcal{X}$)
2: $\lambda_1 \leftarrow \infty$ (vagy $\lambda_1 \leftarrow \frac{c(X_1)}{w(X_1)}$ egy tetszőleges $X_1 \in \mathcal{X}$-re)
3: $i \leftarrow 0$
4: **repeat**
5: $i \leftarrow i + 1$
6: $x_i \leftarrow x_{\lambda_i} = \text{argmin}\{c(x) - \lambda_i w(x) : x \in \mathcal{X}\}$
7: $c_i \leftarrow c(x_i)$
8: $w_i \leftarrow w(x_i)$
9: $h_i \leftarrow h(\lambda_i) = c_i - \lambda_i w_i$
10: $\lambda_{i+1} \leftarrow \frac{c_i}{w_i}$
11: **until** $h_i > 0$
12: **end procedure**

Newton-Dinkelbach eljárás

1: **procedure** NEWTONDINKELBACH($c, w, \mathcal{X}$)
2: $\lambda_1 \leftarrow \infty$ (vagy $\lambda_1 \leftarrow \frac{c(X_1)}{w(X_1)}$ egy tetszőleges $X_1 \in \mathcal{X}$-re)
3: $i \leftarrow 0$
4: **repeat**
5: $i \leftarrow i + 1$
6: $x_i \leftarrow x_{\lambda_i} = \operatorname{argmin}\{c(x) - \lambda_i w(x) : x \in \mathcal{X}\}$
7: $c_i \leftarrow c(x_i)$
8: $w_i \leftarrow w(x_i)$
9: $h_i \leftarrow h(\lambda_i) = c_i - \lambda_i w_i$
10: $\lambda_{i+1} \leftarrow \dfrac{c_i}{w_i}$
11: **until** $h_i > 0$
12: **end procedure**

Newton-Dinkelbach eljárás

1: **procedure** NEWTONDINKELBACH($c, w, \mathcal{X}$)
2: $\lambda_1 \leftarrow \infty$ (vagy $\lambda_1 \leftarrow \frac{c(X_1)}{w(X_1)}$ egy tetszőleges $X_1 \in \mathcal{X}$-re)
3: $i \leftarrow 0$
4: **repeat**
5: $i \leftarrow i + 1$
6: $x_i \leftarrow x_{\lambda_i} = \operatorname{argmin}\{c(x) - \lambda_i w(x) : x \in \mathcal{X}\}$
7: $c_i \leftarrow c(x_i)$
8: $w_i \leftarrow w(x_i)$
9: $h_i \leftarrow h(\lambda_i) = c_i - \lambda_i w_i$
10: $\lambda_{i+1} \leftarrow \frac{c_i}{w_i}$
11: **until** $h_i > 0$
12: **end procedure**

Állítás

- $h_1 < h_2 < \cdots < h_t = 0$
- $\lambda_1 > \lambda_2 > \cdots > \lambda_t = \lambda^*$
- $w_1 > w_2 > \cdots > w_t > 0$

Becslések

Tétel

$$\frac{h_{i+1}}{h_i} + \frac{w_{i+1}}{w_i} \leq 1$$

Következmény

$$\frac{h_{i+1} w_{i+1}}{h_i w_i} \leq \frac{1}{4}$$

Állítás

Minden $i = 1, 2, \cdots, t-2$ *és* $\lambda \leq \lambda_{i+2}$*-re*

$$c_i - \lambda w_i \geq -h_{i+1}$$

Becslések

Tétel

$$\frac{h_{i+1}}{h_i} + \frac{w_{i+1}}{w_i} \leq 1$$

Következmény

$$\frac{h_{i+1} w_{i+1}}{h_i w_i} \leq \frac{1}{4}$$

Állítás

Minden $i = 1, 2, \cdots, t-2$ *és* $\lambda \leq \lambda_{i+2}$*-re*

$$c_i - \lambda w_i \geq -h_{i+1}$$

Becslések

Tétel

$$\frac{h_{i+1}}{h_i} + \frac{w_{i+1}}{w_i} \leq 1$$

Következmény

$$\frac{h_{i+1} w_{i+1}}{h_i w_i} \leq \frac{1}{4}$$

Állítás

Minden $i = 1, 2, \cdots, t-2$ *és* $\lambda \leq \lambda_{i+2}$*-re*

$$c_i - \lambda w_i \geq -h_{i+1}$$

Feladatok

Feladat

Mutassunk olyan $c \in \mathbb{R}^p$-t, és egy „hosszú” $y_1, y_2, \cdots, y_q \in \{-1, 0, 1\}^p$ sorozatot, amire

$$cy_1 \geq cy_2 \geq \cdots \geq cy_q$$

Tétel ((Goemans))

Legyen $c \in \mathbb{R}^p$ és $y_1, y_2, \cdots, y_q \in \{-1, 0, 1\}^p$ olyanok, hogy minden $i \in \{1, \cdots, q-1\}$-re $0 < y_{i+1}c \leq \frac{1}{2} y_i c$ teljesül. Ekkor $q = O(p \log p)$.

Lineáris eset

Legyen E egy véges alaphalmaz $m := |E|$ valamint tegyül fel, hogy $X \subseteq \{0,1\}^E$ és c illetve w lineáris függvények, azaz

$$c(x) := \sum_{e \in E} c_e x_e \quad \text{és} \quad w(x) := \sum_{e \in E} w_e x_e$$

valamilyen $c \in \mathbb{R}^E$ és $w \in \mathbb{R}^E$ vektorokra.

Példák

Minimális tört-optimális utak, körök, feszítőfák, fenyők, párosítások, vágások stb.

Feltéve, hogy a megfelelő paraméteres feladat megoldható!

Tétel (Radzik)

Ebben az esetben a Newton-Dinkelbach eljárás $O(n^2 \log n)$ iteráció után véget ér.

Lineáris eset

Legyen E egy véges alaphalmaz $m := |E|$ valamint tegyül fel, hogy $X \subseteq \{0,1\}^E$ és c illetve w lineáris függvények, azaz

$$c(x) := \sum_{e \in E} c_e x_e \quad \text{és} \quad w(x) := \sum_{e \in E} w_e x_e$$

valamilyen $c \in \mathbb{R}^E$ és $w \in \mathbb{R}^E$ vektorokra.

Tétel (Radzik)

Ebben az esetben a Newton-Dinkelbach eljárás $O(n^2 \log n)$ iteráció után véget ér.

Bizonyítás

Tekintsük az $s_i := -h_{2i} w_{2i-1}$ számsorozatot. Erre

- $s_{i+1} < \frac{1}{4} s_i$
- $s_i = c_{2i} w_{2i-1} - c_{2i-1} w_{2i}$

Alkalmazzuk Goemans lemmáját

Lineáris eset

Legyen E egy véges alaphalmaz $m := |E|$ valamint tegyül fel, hogy $X \subseteq \{0,1\}^E$ és c illetve w lineáris függvények, azaz

$$c(x) := \sum_{e \in E} c_e x_e \quad \text{és} \quad w(x) := \sum_{e \in E} w_e x_e$$

valamilyen $c \in \mathbb{R}^E$ és $w \in \mathbb{R}^E$ vektorokra.

Tétel (Radzik)

Ebben az esetben a Newton-Dinkelbach eljárás $O(n^2 \log n)$ iteráció után véget ér.

Bizonyítás

Tekintsük az $s_i := -h_{2i} w_{2i-1}$ számsorozatot. Erre

- $s_{i+1} < \frac{1}{4} s_i$
- $s_i = c_{2i} w_{2i-1} - c_{2i-1} w_{2i}$

Alkalmazzuk Goemans lemmáját

Tört-legrövidebb út DAG-ban

Tétel

Legyen $G = (N, E)$ egy DAG és $s, t \in N$. Ekkor Newton-Dinkelbach eljárás a $s \rightsquigarrow t$ tört-legrövidebb út problémára alkalmazva $O(m \log n)$ iteráció után megáll.

Definíció

Egy $e \in E$ él fontos az i-ik iterációban, ha $e \in x_i \cup x_{i+1} \cup \ldots$.

Lemma

Végezzünk el $k := \lceil \log n \rceil + 1$ iterációt az algoritmussal. Ekkor a fontos élek halmaza szigorúan csökken.

Feladatok

Házi feladat 6/2. Kereskedelmi hajó útvonaltervezés

Adott egy $G = (N, E)$ irányított gráf, egy $p : E \longrightarrow \mathbb{R}$ profit és egy $t : E \longrightarrow \mathbb{R}$ utazási idő függvény.

- Fentieket felhasználva adjunk erősen polinomiális algoritmust a $\frac{p(C)}{t(C)}$ értéket maximalizáló C irányított kör megtalálására a gráfban.
- Mi a helyzet a férjezett/nős hajóskapitányokkal? Azaz, ha adott egy $f \in N$ férj/feleség, és az olyan C körökön keressük a maximális $\frac{p(C)}{t(C)}$ értéket, amelyek áthaladnak az f ponton.

Tört-legrövidebb út DAG-ban II. (Megiddo módszere)

Ötlet

Próbáljuk lefuttatni a DAG legrövidebb út algoritmust a $c - \lambda^* w$ költségekkel anélkül, hogy λ^* értékét ismernénk, szimbolikusan számolva vele.

($\lambda^* := \max\{\lambda \in \mathbb{R} : \forall X \in \mathcal{X}\text{-re } c(X) - \lambda w(X) \geq 0\}$)

Tört-legrövidebb út DAG-ban II. (Megiddo módszere)

Kimenő éleken dolgozó algoritmus

1: **procedure** DAGSHORTESTPATH(G, c, s)
2: **for** $\forall i \in N, i \neq s$ **do**
3: $d(i) := \infty$
4: **end for**
5: $d(s) := 0, pred(s) := 0$
6: **for** $\forall i \in N, i \neq s$, topologikus sorrendben **do**
7: **for** $j : (i, j) \in \delta(i)$ **do**
8: **if** $d(i) + c_{ij} < d(j)$ **then**
9: $d(j) := d(i) + c_{ij}$
10: $pred(j) := (i, j)$
11: **end if**
12: **end for**
13: **end for**
14: **end procedure**

Tört-legrövidebb út DAG-ban II. (Megiddo módszere)

Kimenő éleken dolgozó algoritmus

1: **procedure** DAGSHORTESTPATH(G, c, s)
2: **for** $\forall i \in N, i \neq s$ **do**
3: $d(i) := \infty$
4: **end for**
5: $d(s) := 0, pred(s) := 0$
6: **for** $\forall i \in N, i \neq s$, topologikus sorrendben **do**
7: **for** $j : (i, j) \in \delta(i)$ **do**
8: **if** $d(i) + (c_{ij} - \lambda^* w_{ij}) < d(j)$ **then**
9: $d(j) := d(i) + (c_{ij} - \lambda^* w_{ij})$
10: $pred(j) := (i, j)$
11: **end if**
12: **end for**
13: **end for**
14: **end procedure**

Tört-legrövidebb út DAG-ban II. (Megiddo módszere)

Kimenő éleken dolgozó algoritmus

```
1: procedure DagShortestPath(G, c, s)
2:     for ∀i ∈ N, i ≠ s do
3:         d(i) := ∞
4:     end for
5:     d(s) := 0, pred(s) := 0
6:     for ∀i ∈ N, i ≠ s, topologikus sorrendben do
7:         for j : (i, j) ∈ δ(i) do
8:             if d(i) + (c_ij − λ* w_ij) < d(j) then
9:                 d(j) := d(i) + (c_ij − λ* w_ij)
10:                pred(j) := (i, j)
11:            end if
12:        end for
13:    end for
14: end procedure
```

- A $d(i)$ tárolókban $a + \lambda^* b$ alakú kifejezéseket (azaz (a, b) számpárokat) tárolunk. Ezekkel szimbolikusan tudunk számolni.
- Összehasonlítás: $a_1 + \lambda^* b_1 < a_2 + \lambda^* b_2 \Leftrightarrow \boxed{\lambda^* > \frac{a_1 - a_2}{b_1 - b_2}}$, ha $b_1 > b_2$.

Tört-legrövidebb út DAG-ban II. (Megiddo módszere)

Házi feladat 6/3.

- Gondoljuk végig a fenti eljárás működését és lássuk be, hogy algoritmus végén kapott $s \rightsquigarrow t$ út tört-minimális lesz.
- Mennyi az algoritmus futási ideje?
- ★ Ötletek futásidő javításra?

Megjegyzések

- Csak a gyengébbik feltevést használtuk, azaz azt, hogy egy adott $\lambda \in \mathbb{R}$-re meg tudjuk határozni $\min\{c(X) - \lambda w(X) : X \in \mathcal{X}\}$ előjelét.
- Ez egy általános módszer, az algoritmusról azt kell feltenni, hogy „nem szoroz”.

Feladat

Házi feladat 6/4.

Tekintsük a következő lineáris programozási feladatot:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = 0 \quad \forall i \in N$$

$$\sum_{(i,j)\in A} w_{ij} x_{ij} = 1$$

$$x_{ij} \geq 0 \quad \forall (i,j) \in A$$

Bizonyítsuk be, hogy, hogy ennek egy optimális megoldása meghatároz körök egy halmazát, amelyek súlyozott átlaga ($\frac{c(C)}{w(C)}$ értéke) egyenlő. Adjunk ennek segítségével eljárást, ami a fenti LP-feladat optimális megoldásának felhasználásával megad egy minimális súlyozott átlagú kört.