Algoritmusok és adatszerkezetek I.
7. Előadás

MST, Prim algoritmus. Legrövidebb utak egy forrásból, Dijkstra algoritmus.

Tartalom

Prim algoritmus

- Kijelölünk a $G = (V, E)$ összefüggő, élsúlyozott, irányítatlan gráfban egy tetszőleges $r \in V$ csúcsot.
- A $T = (\{r\}, \{\})$, egyetlen csúcsból álló fából kiindulva építünk fel egy minimális (V, F) feszítőfát:
  - Minden lépésben a $T = (N, A)$ fához egy újabb biztonságos élet és hozzá tartozó csúcsot adunk:
  - $T = (N, A)$ fa végig ennek a minimális feszítőfának a része marad, azaz végig igaz az $N \subseteq V \wedge A \subseteq F$ invariáns
  - Ehhez mindegyik lépésben egy könnyű élet választunk ki az $(N, V \setminus N)$ vágásban. (Ld. 11. tételt!)
  - A megfelelő könnyű él hatékony meghatározása:
    - egy Q minprioritásos sorban tartjuk nyilván a $V \setminus N$ csúcshalmazt
    - Mindegyik $u \in V \setminus N$ csúcshoz tartozik egy $c(u)$ és egy $p(u)$ címke
    - Ha van él az u csúcs és a T fa N csúcshalmaza között, azaz az $(N, V \setminus N)$ vágásban
      - a $(p(u), u)$ a gráf egyik éle a vágásban
      - $c(u) = w(p(u), u)$
      - és $\forall x$ csúcsra, amelyre (x, u) vágásbeli él, $w(x, u) \geq w(p(u), u)$
      - Ez azt jelenti, hogy $(p(u), u)$ minimális súlyú él azok között, amelyek az u csúcsot a T fához kapcsolják
    - Ha nincs él a u csúcs és a T fa között
      - $p(u) = \infty \wedge c(u) = \infty$

Prim algoritmus

- A Q min-prioritásos sorban a csúcsokat a $c(u)$ értékeik szerint tartjuk nyilván
  - Az $u := Q$.remMin() utasítás egy olyan u csúcsot vesz ki Q-ból, amire a $(p(u), u)$ könnyű él az $(N, V \setminus N)$ vágásban
  - Ezt az élt így a 11. tétel szerint biztonságosan adjuk hozzá a T fához, azaz T továbbra is kiegészíthető minimális feszítőfává, ha még nem az
- Így az eredetileg egyetlen csúcsból álló T fa, $n-1$ db ilyen $(p(u), u)$ él hozzáadásával MST (minimális feszítőfa) lesz
  - Amikor u bekerül a T fába, a Q-beli v szomszédai közelebb kerülhetnek a fához
    - ➢ ezeket meg kell vizsgálni
    - ha némelyikre $w(u, v) < c(v)$ -> $c(v) := w(u, v)$ és a $p(v) := u$ utasításokkal aktualizálni kell a v csúcs címkéit
    - Ilyen módon a c(v) érték csökkenése miatt a Q helyreállítása is szükségessé válhat
      - Ha például Q reprezentációja egy minimum kupac, a v csúcsot lehet, hogy meg kell cserélni a szülőjével, esetleg újra és újra, mígnem a szülőjének c értéke $\leq c(v)$ lesz, vagy v fölér kupac gyökerébe
- Az alábbi algoritmusban a T fa implicite reprezentációja
  - $N = V \setminus Q$
  - T élei: az $N \setminus \{r\}$-beli x csúcsok és $p(x)$ címkéik segítségével $(p(x), x)$

Prim algoritmus stuktogramja és műveletigénye

$\text{Prim}(G : \mathcal{G}_w \; ; \; r : \mathcal{V})$

$\Theta(n)$

$\forall v \in G.V$

$c(v) := \infty \; ; \; p(v) := \emptyset$ // costs and parents still undefined

// edge $(p(v), v)$ will be in the MST where $c(v) = G.w(p(v), v)$

$\Theta(n)$

$c(r) := 0$ // r is the root of the MST where $p(r)$ remaines undefined

// let Q be a minimum priority queue of $G.V \setminus \{r\}$ by label values $c(v)$:

$Q : \text{minPrQ}(G.V \setminus \{r\}, c)$ // $c(v)$ = cost of light edge to (partial) MST

$u := r$ // vertex $u = r$ has become the first node of the (partial) MST

$O(n * \log n)$

$n-1$-szer $\quad \neg Q.\text{isEmpty}()$

// neighbors of u may have come closer to the partial MST

$2m$-szer $\quad \forall v : (u, v) \in G.E \wedge v \in Q \wedge c(v) > G.w(u, v)$ $\quad O(m * \log n)$

$\Theta(1)$ $\quad p(v) := u \; ; \; c(v) := G.w(u, v) \; ; \; Q.\text{adjust}(v)$ $\quad O(\log n)$

$O(\log n)$ $\quad u := Q.\text{remMin}()$ // $(p(u), u)$ is a new edge of the MST

$\sum$: $O((n+m) * \log n)$ a gráf öf -> $m \geq n-1$ -> $MT_{\text{Prim}}(n, m) \in O(m * \log n)$

A Prim algoritmus műveletigénye

- Feltételek
  - Q reprezentációja bináris minimum kupac
  - $n = |G.V| \wedge m = |G.E|$
- Az első ciklus futási ideje $\Theta(n)$
- Az első és a második ciklus közti rész műveletigénye
  - a Q kupac inicializálása határorozza meg -> $\Theta(n)$
- A második ciklus
  - mindegyik iterációja kiveszi a Q legkisebb c-értékű elemét
    - $n-1$-szer fut l
    - a belső ciklustól eltekintve mindegyik iterációja $O(\log n)$ időt igényel
    - ➢ $O(n * \log n)$.
  - + a belső ciklus futási idejét: $O(m * \log n)$

➢ $\Theta(n) + \Theta(n) + O(n * \log n) + O(m * \log n) = O((n+m) * \log n)$

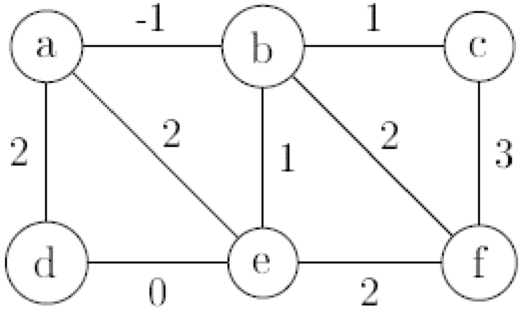
- Mivel a gráf összefüggő -> $m \geq n-1$ -> $MT_{\text{Prim}}(n, m) \in O(m * \log n)$

- A belső ciklus
  - a gráf mindegyik élére legfeljebb kétszer fog lefutni
    - mindegyik élet mindkét végpontja felől megtaláljuk, kivéve azokat, amelyek egyik vépontja a Q-ban utoljára megmaradt csúcs.
    - (A ciklusfejben a $\forall v : (u, v) \in G.E$ rész után következő $v \in Q \wedge c(v) > G.w(u, v)$ szűrő feltétel valójában egy implicit, beágyazott elágazás feltétel része.)
  - Q.adjust(v): $O(\log n)$
  - Többi: $\Theta(1)$
  - ➢ $O(m * \log n)$

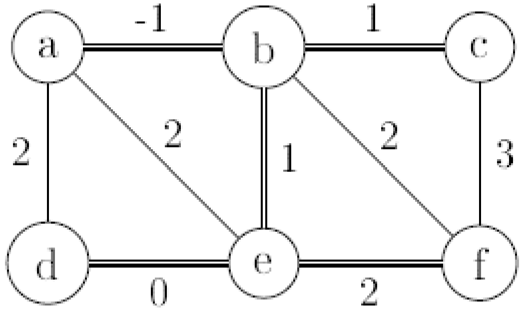
A Prim algoritmus működésének szemléltetése

- A *d* csúcsból indítva

a – b, -1 ; d, 2 ; e, 2.
b – c, 1 ; e, 1 ; f, 2.
c – f, 3.
d – e, 0.
e – f, 2.



in-to MST	changes of c and p labels					
	a	b	c	d	e	f
	∞ ⃠	∞ ⃠	∞ ⃠	0 ⃠	∞ ⃠	∞ ⃠
d	2 d			$\vert$	0 d	
$e \stackrel{0}{-} d$		1 e		$\vert$	$\vert$	2 e
$b \stackrel{1}{-} e$	-1 b	$\vert$	1 b	$\vert$	$\vert$	
$a \stackrel{-1}{-} b$	$\vert$	$\vert$		$\vert$	$\vert$	
$c \stackrel{1}{-} b$	$\vert$	$\vert$	$\vert$	$\vert$	$\vert$	
$f \stackrel{2}{-} e$	$\vert$	$\vert$	$\vert$	$\vert$	$\vert$	$\vert$



Legrövidebb utak egy forrásból

- **Feladat:** A $G : \mathcal{G}_w$ gráf tetszőlegesen rögzített s start csúcsából (source vertex) legrövidebb, azaz optimális utat keresünk mindegyik, az s-ből G-ben elérhető csúcsba.
  - A most következő algoritmusokban az irányítatlan gráfokat olyan irányított gráfoknak tekintjük, ahol a gráf tetszőleges (u, v) élével együtt (v, u) is éle a gráfnak, és $w(u, v) = w(v, u)$.
- A feladat pontosan akkor oldható meg, ha G-ben nem létezik s-ből elérhető negatív kör
  - azaz olyan kör, amely mentén az élsúlyok összege negatív
  - Irányítatlan gráfoknál: ha nincs a gráfban s-ből elérhető negatív él
    - pl. az irányítatlan gráfban (u, v) s-ből elérhető negatív él -> <u, v, u> - s-ből elérhető negatív kör
- Ha a feladat megoldható ->$\forall\ v \in G.V \setminus \{s\}$ csúcsra két lehetőség
  - Ha létezik út s-ből v-be
    - $d(v)$:az optimális út hossza
    - $\pi(v)$: egy ilyen optimális úton a v csúcs közvetlen megelőzője, azaz szülője
  - Ha nem létezik út s-ből v-be
    - $d(v) = \infty$ és $\pi(v) = \emptyset$
  - Az s csúcsra
    - $d(s) = 0$ és $\pi(s) = \emptyset$ (az optimális út csak az s csúcsból áll)

Legrövidebb utakat kereső algoritmusok közös vonásai

- L! $s \rightsquigarrow v$: az adott pillanatig kiszámolt *s*-ből *v*-be vezető legrövidebb út
- Az inicializálásuk után már teljesülő invariáns
  - $d(v)$ ennek a hossza
  - $\pi(v)$ ($v \neq s$ esetén) ezen az úton a *v* csúcs közvetlen megelőzője
- Ha még nem számoltunk ki $s \rightsquigarrow v$ utat ->végtelen hosszúnak tekintjük
  - azaz $d(v) = \infty$ és $\pi(v) = \emptyset$
- Az optimális utakat fokozatosan közelítjük
  - a gráf (u, v) éleit szisztematikusan vizsgáljuk
    - ha $s \rightsquigarrow u \rightarrow v$ rövidebb mint $s \rightsquigarrow v$ -> az előbbi lesz az új $s \rightsquigarrow v$ (*közelítésnek* (relaxation) nevezzük)
    - Kód szinten:

$d(v) > d(u) + G.w(u,v)$	
$\pi(v) := u \ ; \ d(v) := d(u) + G.w(u,v)$	SKIP

- Ismételten kiválasztanak és ki is vesznek az ún. *feldolgozandó* csúcsok halmazából egy csúcsot, és a csúcsból kimenő összes élre végeznek közelítést.
  - Ezeket a közelítéseket együtt a csúcs *kiterjesztésének* nevezzük
  - A csúcs kiválasztását (és kivételét a feldolgozandók közül) a kiterjesztésével együtt a *feldolgozásának* nevezzük.

Legrövidebb utakat kereső algoritmusok különbségei

- Dijkstra algoritmus
  - Kezdetben kiválasztja kiterjesztésre *s*-et
  - A feldolgozandó csúcsok halmazát pedig $G.V \setminus \{s\}$-sel inicializálja
  - Először tehát s-et terjeszti ki
  - Majd a feldolgozandók közül mindig egy olyan csúcsot dolgoz fel, amibe az adott időpontig a többihez képest legrövidebb utat talált
  - Mohó algoritmus
    - Lokálisan optimalizál
      - Mindig a legígéretesebb csúcsot dolgozza fel
  - Kiderül, hogy így mindig olyan csúcsot terjeszt ki, amibe már eddig optimális utat talált, és egy csúcsot sem kell kétszer kiterjesztenie
- DAGshP algoritmus
  - Először meghatározza az *s*-ből elérhető csúcsokat és egyúttal sorbarendezi ezeket
    - Az adott sorrend szerint feldolgozva őket, mindegyik kiválasztásának az időpontjában már megvan bele az optimális út
  - Ilyen módon ez is csak egyszer terjeszti ki, és csak az *s*-ből elérhető csúcsokat.

Legrövidebb utakat kereső algoritmusok különbségei

- Az előbbi két algoritmusnak speciális előfeltételei vannak.
  - Egyik sem tudja a lehető legáltalánosabb esetben megoldani a legrövidebb utak egy forrásból problémát.
  - A Dijkstra algoritmus elégséges előfeltétele
    - A gráfban ne legyen *s*-ből elérhető negatív súlyú él
  - DAGshP algoritmus elégséges előfeltétele
    - A gráf *s*-ből elérhető része DAG legyen
- Cserébe mindkét algoritmus a legrosszabb esetben is meglehetősen hatékony
  - DAGshP: $\Theta(n + m)$
  - Dijkstra: $O(n + m) * \log n$
- QBF algoritmus
  - A többivel szemben a lehető legáltalánosabb esetben oldja meg a feladatot
  - Szükséges és elégséges előfeltétele:
    - Ne legyen az *s*-ből elérhető negatív kör

Legrövidebb utakat kereső algoritmusok különbségei

- QBF algoritmus
  - Cserébe csak O($n * m$) műveletigényt tudunk garantálni
    - ➢ Ugyanazt a csúcsot többször is feldolgozhatja
    - Átlagos esetben ennél sokkal hatékonyabb
      - Implementációikat összehasonlítva sok nemnegatív élsúlyú gráfon a Dijkstra algoritmusnál is gyorsabb
- A DAGshP és a QBF algoritmusok előfeltétele nemtriviális
  - ➢ Ezeknél így az algoritmus része
    - az előfeltételének az ellenőrzése
    - a nem megfelelő bemenetek visszautasítása
  - Módja
    - mindkettő megad egy olyan kört a gráfban, ami miatt nem tudja a feladatot megoldani
    - (A QBF ebben az esetben negatív kört ad meg.)

Dijkstra algoritmusa

- **Előfeltétel:** A $\mathcal{G} : \mathcal{G}_w$ gráfban $\forall(u, v) \in G.E$-re $G.w(u, v) \geq 0$, azaz a gráf mindegyik élének élsúlya nemnegatív.
  - Nincs a gráfban negatív kör -> a legrövidebb utak egy forrásból feladat megoldható.

- $MT_{Dijkstra}(n, m) \in O((n + m) * \lg n)$
  - ~ Prim algoritmus
  - (két alg. közötti hasonlóság)
- $mT_{Dijkstra}(n, m) \in \Theta(n)$
  - A 2. ciklus előtti rész műveletigénye már $\Theta(n)$
  - Ehhez hozzáadódik
    - \+ a 2. ciklus egyetlen iterációja
    - \+ a belső ciklus 0-szori iterációja
  - Ha nincs a gráfban s-nek rákövetkezője
    - = A 2. ciklus műveletigénye $O(\log n)$ (pontosabban $\Theta(1)$)
      - ami nagyságrenddel kisebb, mint $\Theta(n)$

$\text{Dijkstra}(G : \mathcal{G}_w \; ; \; s : \mathcal{V})$

$\forall v \in G.V$

$\quad d(v) := \infty \; ; \; \pi(v) := \emptyset$ // distances are still ∞, parents undefined

$\quad$ // $\pi(v)$ = parent of v on $s \rightsquigarrow v$ where $d(v) = w(s \rightsquigarrow v)$

$d(s) := 0$ // s is the root of the shortest-path tree

// let Q be a minimum priority queue of $G.V \setminus \{s\}$ by label values $d(v)$:

$Q : \text{minPrQ}(G.V \setminus \{s\}, d)$

$u := s$ // going to calculate shortest paths form s to other vertices

$d(u) < \infty \wedge \neg Q.\text{isEmpty}()$

$\quad$ // check, if $s \rightsquigarrow u \to v$ is shorter than $s \rightsquigarrow v$ before

$\quad \forall v : (u, v) \in G.E \wedge d(v) > d(u) + G.w(u, v)$

$\quad\quad \pi(v) := u \; ; \; d(v) := d(u) + G.w(u, v) \; ; \; Q.\text{adjust}(v)$

$\quad u := Q.\text{remMin}()$ // $s \rightsquigarrow u$ is optimal now, if it exists

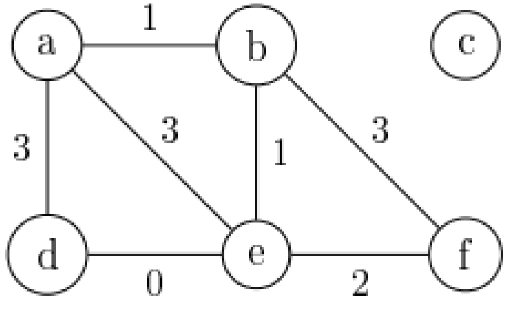
Dijkstra algoritmus tételek

1. Tétel. Amikor az u csúcsot kiválasztjuk kiterjesztésre (először az u := s, később az u := Q.remMin() utasítással), akkor u-ba már optimális utat találtunk, feltéve, hogy d(u) < ∞.

- **Bizonyítás.**
  - u = s-re nyilván igaz az állítás.
  - Tegyük fel indirekt módon, hogy nem mindegyik csúcsra igaz!
    - ∃ v csúcs, amire az algoritmus futása során először lesz igaz, hogy kiválasztjuk kiterjesztésre, de $w(s \overset{opt}{\leadsto} v) < d(v) < \infty$
      - ahol $s \overset{opt}{\leadsto} v$ egy s-ből v-be vezető optimális út, $w(s \overset{opt}{\leadsto} v)$ pedig ennek a hossza
    - ➢ v ≠ s
    - Továbbá, az $s \overset{opt}{\leadsto} v$ úton az s csúcsot már kiterjesztettük, a v csúcsot viszont még nem
      - Van tehát az $s \overset{opt}{\leadsto} v$ úton egy első csúcs, amit még nem terjesztettünk ki
        - L! t az $s \overset{opt}{\leadsto} v$ úton az első csúcs, amit még nem terjesztettünk ki!
        - Szelméletesen: $s \overset{opt}{\leadsto} t \overset{opt}{\leadsto} v$.

Dijkstra algoritmus szemléltetése

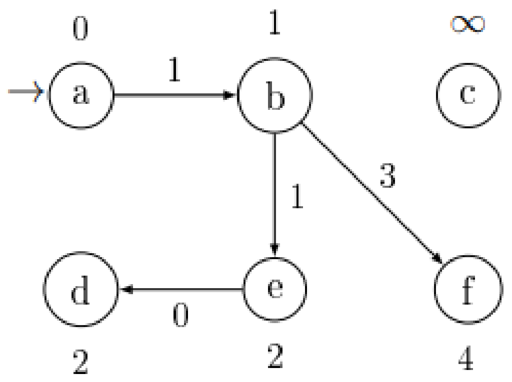
- A Dijkstra algoritmusnál a ki nem terjesztett csúcsokat röviden nyílt csúcsoknak nevezzük.



a – b, 1 ; d, 3 ; e, 3.
b – e, 1 ; f, 3.
c.
d – e, 0.
e – f, 2

ex-pand	changes of d and π labels					
	a	b	c	d	e	f
$d\ \mathcal{V}$	$0\ \oslash$	$\infty\ \oslash$	$\infty\ \oslash$	$\infty\ \oslash$	$\infty\ \oslash$	$\infty\ \oslash$
0 a	|	1 a		3 a	3 a	
1 b	|	|			2 b	4 b
2 e	|	|		2 e	|	
2 d	|	|		|	|	
4 f	|	|		|	|	|

- A legrövidebb utak fája *s* = *a* esetén
- Az s-ből elérhetetlen csúcsot vagy csúcsokat is feltüntetjük, ∞ *d* értékkel



Dijkstra algoritmus tételek

2. Tétel. Ha a kiterjesztésre kiválasztott u csúcsra $d(u) = \infty$, akkor $Q \cup \{u\}$ egyetlen eleme sem érhető már el az s csúcsból.

- **Bizonyítás.**
  - u az első és egyetlen olyan csúcs, amit $d(u) = \infty$ értékkel választottunk ki, mert ezzel megáll a fő ciklus
  - A korábban kiterjesztésre kiválasztott x csúcsokra tehát $d(x) < \infty$
    - ezekbe az előző tétel szerint már optimális utat találtunk
  - Tegyük fel indirekt módon, hogy $Q \cup \{u\}$-nak van olyan v eleme, amely elérhető s-ből
    - Ekkor létezik $s \overset{opt}{\leadsto} v$ is
    - ezen kell lennie egy első t csúcsnak:
      - eleme $Q \cup \{u\}$-nak
      - de az előző tétel bizonyítása szerint erre már $d(t) = w(s \overset{opt}{\leadsto} t) < \infty = d(u)$
      - tehát $d(t) < d(u)$, ami ellentmond annak, hogy az u csúcsot választottuk ki kiterjesztésre

Dijkstra algoritmus tételek

- Mivel *s*-et már kiterjesztettük: $t \neq s$.
- Továbbá:
  - az $s \overset{opt}{\rightsquigarrow} t$ úton *t* közvetlen *x* megelőzőjét is kiterjesztettük már
  - és az *x* kiterjesztésekor még teljesült, hogy $d(x) = w(s \overset{opt}{\rightsquigarrow} x)$
- Akkor viszont az $x \longrightarrow t$ él feldolgozása óta már $d(t) = w(s \overset{opt}{\rightsquigarrow} t)$ is teljesül
- Mivel
  - a $t \overset{opt}{\rightsquigarrow} v$ úton minden él súlya (azaz hossza) nemnegatív
  - *t* az $s \overset{opt}{\rightsquigarrow} v$ úton van, azaz $s \overset{opt}{\rightsquigarrow} t \overset{opt}{\rightsquigarrow} v$

➢ $w(s \overset{opt}{\rightsquigarrow} t) \leq w(s \overset{opt}{\rightsquigarrow} v)$

$\sum$: $d(t) = w(s \overset{opt}{\rightsquigarrow} t) \leq w(s \overset{opt}{\rightsquigarrow} v) < d(v) < \infty$, azaz $d(t) < d(v)$

  - ez ellentmond az indirekt feltételezésnek, ami szerint a *v* csúcsot választottuk kiterjesztésre.

A tétel következménye

- Amíg tehát a kiterjesztésre kiválasztott u csúcsra $d(u) < \infty$
  - addig olyan csúcsot választunk ki, amelyikbe már optimális utat találtunk
- Ha $d(u) < \infty$ mindegyik kiválasztott csúcsra igaz ⇔
  - a fenti algoritmus 2. ciklusa n–1 iterációval kiüríti Q –t
  - megáll, miután a gráf mindegyik csúcsába optimális utat talált
- Ha pedig egyszer a kiterjesztésre kiválasztott u csúcsra már $d(u) = \infty$
  - $Q \cup \{u\}$ egyetlen eleme sem érhető már el az s csúcsból
  - megállhatunk:
    - mindegyik d-értéke végtelen, π-érteke pedig $\varnothing$
    - míg a korábban, véges d-értékkel kiterjesztett csúcsok éppen azok, amelyek elérhetők s-ből, és ezekbe találtunk is optimális utat

Ellenőrző kérdések

1. Implementálja a Dijkstra/Prim algoritmust szomszédossági mátrixos / szomszédossági listás gráfábrázolás esetén! A prioritásos sort rendezetlen tömbbel reprezentálja!
   - Mekkora lesz a műveletigény?
   - Lehet-e az aszimptotikus műveletigényen javítani a prioritásos sor kinomultabb megvalósításával?
2. Mit számol ki a Prim algoritmus?
   - Szemléltesse a működését az a b, 0 ; d, 1. b c, 5 ; d, 2 ; e, 3. c e, 2. d e, 2. irányítatlan gráfon, a d csúcsból indítva!
   - Mondja ki a biztonságos élekről és a minimális feszítőfákról szóló tételt! Definiálja a tételben szereplő vágás és könnyű él fogalmakat! Hogyan következik a Prim algoritmus helyessége ebből a tételből?
   - Mekkora az algoritmusnak az előadásról ismert műveletigénye, és milyen feltételekkel?

Köszönöm a figyelmet!

Pusztai Kinga

A bemutató Ásványi Tibor: Algoritmusok és adatszerkezetek II. eladásjegyzet:Élsúlyozott gráfok és algoritmusaik alapján készült.