ELTE IK - Programtervező Informatikus BSc

Záróvizsga tételek

16. Haladó algoritmusok

Haladó algoritmusok

Elemi gráf algoritmusok: szélességi, mélységi bejárás és alkalmazásai. Minimális feszítőfák, általános algoritmus, Kruskal és Prim algoritmusai. Legrövidebb utak egy forrásból, sor alapú Bellman-Ford, Dijkstra, DAG legrövidebb út. Legrövidebb utak minden csúcspárra: Floyd-Warshall algoritmus. Gráf tranzitív lezártja.

1 Gráfalgoritmusok

1.1 Gráf ábrázolás

Láncolt listás ábrázolás

A gráf csúcsait helyezzük el egy tömbben (vagy láncolt listában). Minden elemhez rendeljünk hozzá egy láncolt listát, melyben az adott csúcs szomszédjait (az esetleges élsúlyokkal) soroljuk fel.

Mátrixos ábrázolás

Legyen a csúcsok elemszáma n. Ekkor egy $A^{n\times n}$ mátrixban jelöljük, hogy mely csúcsok vannak összekötve. Ekkor mind a sorokban, mind az oszlopokban a csúcsok szerepelnek, és az a_{ij} cellában a i csúcsból j csúcsba vezető él súlya szerepel, ha nincs él a két csúcs között, akkor $-\infty$ (súlyozatlan esetben 1 és 0)

Amennyiben a gráf irányítatlan nyilván $a_{ij} = a_{ji}$

1.2 Szélességi bejárás

G gráf (irányított/irányítatlan) s startcsúcsából a távolság sorrendjében érjük el a csúcsokat. A legrövidebb utak feszítőfáját adja meg, így csak a távolság számít, a súly nem.

A nyilvántartott csúcsokat egy sor adatszerkezetben tároljuk, az aktuális csúcs gyerekeit a sor-ba tesszük. A következő csúcs pedig a sor legelső eleme lesz.

A csúcsok távolságát egy d, szüleiket egy π tömbbe írjuk, és ∞ illetve 0 értékekkel inicializáljuk.

Az algoritmus:

1. Az s startcsúcsot betesszük a sorba
2. A következő lépéseket addig ismételjük, míg a sor üres nem lesz
3. Kivesszük a sor legelső (u) elemét
4. Azokat a gyerekcsúcsokat, melyeknek a távolsága nem ∞ figyelmen kívül hagyjuk (ezeken már jártunk)
5. A többi gyerekre (v): beállítjuk a szülőjét ($\pi[v] = u$), és a távolságát ($d[v] = d[u] + 1$). Majd berakjuk a sorba.

1.3 Minimális költségű utak keresése

Dijkstra algoritmus

Egy G irányított, pozitív élsúlyokkal rendelkező gráfban keres s startcsúcsból minimális költségű utakat minden csúcshoz.

Az algoritmus a szélességi bejárás módosított változata. Mivel itt egy hosszabb útnak lehet kisebb a költsége, mint egy rövidebbnek, egy már megtalált csúcsot nem szabad figyelmen kívül hagyni. Ezért minden csúcs rendelkezik három állapottal (nem elért, elért, kész). A d és π tömböket a szélességi bejáráshoz hasonlóan kezeljük.

A még nem kész csúcsokat egy prioritásos sorba helyezzük, vagy minden esetben minimumkeresést alkalmazunk.

Az algoritmus:

1. Az s startcsúcs súlyát 0-ra állítjuk eltároljuk
2. A következő lépéseket addig ismételjük, míg a konténerünk üres nem lesz
3. Kivesszük a sor legjobb (u) elemét, és "kész"-re állítjuk
4. Ha egy gyerekcsúcs (v) nem kész, és a jelenleg hozzávezető út súlya kisebb, mint az eddigi, akkor: a szülőjét u-ra állítjuk ($\pi[v] = u$), és a súlyát frissítjük ($d[v] = d[u] + d(u, v)$).
5. A többi csúcsot kihagyjuk.

Bellman-Ford algoritmus

Egy G élsúlyozott (akár negatív) irányított gráf s startcsúcsából keres minden élhez minimális költségű utakat, illetve felismeri, ha negatív költségű kör van a gráfban. A d és π tömböket az előzőekhez hasonlóan kezeljük.

Az algoritmus:

1. A startcsúcs súlyát állítsuk be 0-ra.
2. $n-1$ iterációban menjünk végig az összes csúcson, és minden csúcsot (u) vessünk össze minden csúccsal (v). Ha olcsóbb utat találtunk akkor v-be felülírjuk a súlyát ($d[v] = d[u] + d(u, v)$), és a szülőjét ($\pi[v] = u$).
3. Ha az n-edik iterációban is történt módosítás, negatív kör van a gráfban

1.4 Minimális költség feszítőfa keresése

A Prim algoritmus egy irányítatlan élsúlyozott (akár negatív) gráf s startcsúcsából keres minimális költségű feszítőfát. A d és π tömböket az előzőekhez hasonlóan kezeljük. Az algoritmus egy prioritásos sorba helyezi a csúcsokat.

Az algoritmus:

1. A startcsúcs súlyát állítsuk be 0-ra.
2. A csúcsokat behelyezzük a prioritásos sorba.
3. A következő lépéseket addig végezzük, míg a prioritásos sor ki nem ürül.
4. Kiveszünk egy csúcsot (u) a sorból.
5. Minden gyerekére (v), amely még a sorban és a nyilvántartott v-be vezető él súlya nagyobb, mint a most megtalált: A v szülőjét u-ra változtatjuk, a nyilvántartott súlyt felülírjuk $d[v] = d(u, v)$. Majd felülírjuk a v állapotát a prioritásos sorban.
6. Azokkal a gyerekekkel, melyek nincsenek a sorban, vagy a súlyukon nem tudunk javítani, nem változtatunk.

1.5 Mélységi bejárás

G irányított (nem feltétlenül összefüggő) gráf mélységi bejárásával egy mélységi fát (erdőt) kapunk. Az algoritmus a következő:

- Az élsúlyok nem játszanak szerepet
- Nincs startcsúcs, a gráf minden csúcsára elindítjuk az algoritmust. (Természetesen ekkor, ha már olyan csúcsot választunk, amin már voltunk, az algoritmus nem indul el.)
- A csúcsokat mohón választjuk, azaz minden csúcs gyerekei közül az elsőt választva haladunk előre, amíg csak lehet. (Olyan csúcsot találunk, amelynek nincs gyereke, vagy minden gyerekén jártunk már.)
- Ha már nem lehet előre haladni visszalépünk.
- Minden csúcshoz hozzárendelünk két értéket. Az egyik a mélységi sorszám, mely azt jelöli, hogy hanyadiknak értük el. A másik a befejezési szám, mely azt jelzi, hogy hanyadiknak léptünk vissza belőle.

A gráf éleit a mélységi bejárás közben osztályozhatjuk. (Inicializáláskor minden értéket 0-ra állítottunk)

- Faél: A következő csúcs mélységi száma 0
- Visszaél: A következő csúcs mélységi száma nagyobb, mint 0, és befejezési száma 0 (Tehát az aktuális út egy előző csúcsára kanyarodunk vissza)
- Keresztél: A következő csúcs mélységi száma nagyobb, mint 0, és befejezési száma is nagyobb, mint 0, továbbá az aktuális csúcs mélységi száma nagyobb, mint a következő csúcs mélységi száma. (Ekkor egy az aktuális csúcsot megelőző csúcsból induló, már megtalált útba mutató éllel van dolgunk)
- Előreél: A következő csúcs mélységi száma nagyobb, mint 0, és befejezési száma is nagyobb, mint 0, továbbá az aktuális csúcs mélységi száma kisebb, mint a következő csúcs mélységi száma. (Ekkor egy az aktuális csúcsból induló, már megtalált útba mutató éllel van dolgunk)

1.6 DAG Topologikus rendezése

Alapfogalmak

- Topologikus rendezés:
  Egy $G(V, E)$ gráf topologikus rendezése a csúcsok olyan sorrendje, melyben $\forall(u \to v) \in E$ élre u előbb van a sorrendben , mint v
- DAG - Directed Acyclic Graph:
  Irányított körmentes gráf.
  Legtöbbször munkafolyamatok irányítására illetve függőségek analizálására használják.
  Tulajdonságok:
  - Ha G gráfra a mélységi bejárás visszaélt talál (Azaz kört talált) $\implies$ G nem DAG
  - Ha G nem DAG (van benne kör) $\implies$ Bármely mélységi bejárás talál visszaélt
  - Ha G-nek van topologikus rendezések $\implies$ G DAG
  - Minden DAG topologikusan rendezhető.

DAG topologikus rendezése

Egy G gráf mélységi bejárása során tegyük verembe azokat a csúcsokat, melyekből visszaléptünk. Az algoritmus után a verem tartalmát kiírva megkapjuk a gráf egy topologikus rendezését.