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.

Alkalmazásai

1. Két u és v csomópont közötti legrövidebb út megkeresése, az út hosszát az élek számával mérve (előnye a mélységi kereséshez képest)
2. (Fordított) Cuthill–McKee-hálószámozás, a szélességi bejárás egy változata
3. Bináris fa szerializációja/deszerializációja versus szerializáció rendezett sorrendben; lehetővé teszi a fa hatékony újrakonstruálását
4. Egy páros gráf kétoldalúságának tesztelése

1.3 Minimális költségű utak keresése

Dijkstra algoritmus

Egy G irányított vagy irányítatlan, 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 költségű feszítőfa keresése

A minimális költségű feszítőfa vagy minimális feszítőfa egy összefüggő, irányítatlan gráfban található legkisebb élsúlyú feszítőfa. A feszítőfa egy olyan fa, ami a gráf összes csúcsát tartalmazza és élei az eredeti gráf élei közül valók. A minimális feszítőfa nem feltétlenül egyértelmű, de annak súlya igen. Egy gráf tetszőleges minimális feszítőfájának keresésére használható Kruskal és Prim algoritmusa. Mindkét algoritmus mohó stratégiát használ.

Kruskal algoritmus

Az éleket súlyuk szerint növekvő sorrendbe rendezzük, és sorra megvizsgáljuk, hogy melyeket vehetjük be a megoldásba. Kezdetben a gráf minden csúcsa egy-egy halmazt alkot. Egy vizsgált él akkor kerül be a megoldásba, ha a két végpontja két különböző halmazban van, mert ebből tudjuk, hogy a vizsgált él hozzáadásával nem keletkezik kör a gráfban. Ekkor a két halmazt egyesítjük. Az algoritmus végére egyetlen halmaz fog maradni.

Az algoritmus:

1. Válasszuk ki a legkisebb súlyú élt.
2. Amennyiben az él a részgráfhoz való hozzáadása kört alkot, dobjuk azt el, különben adjuk hozzá.
3. Addig ismételjük a fenti lépéseket, amíg van meg nem vizsgált él.

Alkalmazásai:

1. A Kruskal algoritmus annak a tételnek az egyszerű bizonyítására készült, hogy a minimális költségű feszítőfa egyértelmű, ha a gráfban nincs két azonos súlyú él.
2. Véletlen labirintust lehet vele generálni. Ebben az esetben a feldolgozandó gráf minden élének súlya megegyezik, ezért nem kell az éleit rendezni.

Prim algoritmus

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 van 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.

MÁSKÉPP

Működési elve, hogy csúcsonként haladva építi fel a fát, egy tetszőleges csúcsból kiindulva és minden egyes lépésben a lehető legolcsóbb élt keresi meg egy következő csúcshoz.

Az algoritmus:

1. Válasszuk ki a gráf egy tetszőleges csúcsát, legyen ez egy egycsúcsú fa.
2. Ameddig van az eredeti gráfnak olyan csúcsa, amelyik nincs benne a fában, addig ismételjük az alábbi lépéseket.
3. Válasszuk ki a fa csúcsai és a gráf többi csúcsa között futó élek közül a legkisebb súlyút.
4. A kiválasztott él nem fabeli csúcsát tegyük át a fába az éllel együtt.

Alkalmazás: Ezt is lehet véletlen labirintus generálására használni.

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)

A mélységi bejárást építőelemként használó algoritmusok

1. Gráf összefüggő komponensének keresése
2. Topológiai rendezés
3. 2- (él vagy csúcs) kapcsolt elemek keresése
4. 3- (él vagy csúcs) kapcsolt elemek megkeresése
5. Szeparáló élek megkeresése egy gráfban
6. Gráf erősen összefüggő komponenseinek keresése
7. Rejtvények megoldása csak egy megoldással, például labirintusokkal
8. A labirintus létrehozása véletlenszerűen elvégzett mélység-előzetes keresést használhat

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) $\Longrightarrow$ G nem DAG
  - Ha G nem DAG (van benne kör) $\Longrightarrow$ Bármely mélységi bejárás talál visszaélt
  - Ha G-nek van topologikus rendezése $\Longrightarrow$ 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.

1.7 Legrövidebb utak minden csúcspárra: Floyd-Warshall algoritmus

A csúcsok távolságát egy d, a szüleiket egy π mátrixba írjuk, melyeket az előbbiekhez hasonlóan ∞ és 0 értékekkel inicializálunk. A végeredmény az lesz, hogy $d[i, j]$ az i indexű csúcsból a j indexű csúcsba vezető optimális út hossza, vagy ∞, ha nincs út i-ből j-be. $\pi[i, j]$ pedig az algoritmus által kiszámolt, az i indexű csúcsból a j indexű csúcsba vezető optimális úton a j csúcs közvetlen megelőzője, ha i $\neq$ j és létezik út i-ből j-be, különben $\pi[i, j] = 0$.

Az algoritmus leírása:

Vegyünk egy G gráfot, V csúcsokkal 1-től N-ig számozva. Továbbá egy függvényt az ún. **legrovidebbUtvonal(i,j,k)**, amely visszatér a legrövidebb útvonallal i és j között adott csúcsok használatával: $1, 2, ..., k$ közbenső pontként a csúcsok mentén. Figyelembe véve az adott függvényt, a célünk az, hogy megtaláljuk a legrövidebb utat minden i-től j-ig bármelyik csúcs használatával $1, 2, ..., N$. A csúcspárok mindegyikénél a **legrovidebbUtvonal(i,j,k)** lehet akár
(1) egy útvonal, amely nem megy át k-n (amely csak az adott csúcsokat használja: $1, ..., k - 1$)
VAGY
(2) egy útvonal, amely végig megy k-n (i-től k-ig és aztán k-tól j-ig, mindkét esetben az adott közbenső csúcsokat használva: $1, ..., k - 1$).

Tudjuk, hogy a legjobb útvonal i-től j-ig az, amely csak azon csúcsokat használja, amik 1-en keresztül $k - 1$-ig vannak meghatározva a **legrovidebbUtvonal(i,j,k-1)** által, és egyértelmű, hogy ha jobb útvonal lenne i-től k-ig, majd onnan j-ig, akkor ezen útvonalaknak a hossza lenne a legrövidebb út láncolata az i-től a k-ig (csak a közbenső csúcsok használatával $1, ..., k - 1$) és a legrövidebb a k-tól j-ig (csak a közbenső csúcsok használatával $1, ..., k - 1$).

Ha $\omega(i, j)$ az él súlya az adott i és j csúcsok között, akkor meghatározhatjuk a **legrovidebbUtvonal(i,j,k)** függvényt a következő rekurzív képlet szerint: **legrovidebbUtvonal(i,j,0)** $= \omega(i, j)$ rekurzív eset:
legrovidebbUtvonal(i,j,k) =
min(legrovidebbUtvonal(i,j,k-1), legrovidebbUtvonal(i,k,k-1) + legrovidebbUtvonal(k,j,k-1)).

Ez a képlet a Floyd-Warshall algoritmus szive. Az algoritmus futása során először kiszámolja a **legrovidebbUtvonal(i,j,k)** alapján az (i, j) párokat a $k = 1$, majd $k = 2$ és így tovább esetekre. Ez a folyamat addig folytatódik, amíg végül $k = N$ teljesül és az N-edik iteráció is lefut, és megtaláltuk a legrövidebb utat mindegyik (i, j) páros számára bármilyen közbenső csúcs használatával.

1.8 Gráf tranzitív lezártja

Első definíció: A $G = (V, E)$ gráf tranzitív lezártja alatt a V × V ⊇ T relációt értjük, ahol (u, v) ∈ T, ha G-ben az u csúcsból elérhető a v csúcs.

Második definíció: Egy $G = (V, E)$ irányított gráf tranzitív lezártja az a $G' = (V, E')$ gráf, amelyben u-ból v-be pontosan akkor vezet él, ha u-ból vezet irányított út v-be az eredeti G gráfban.

Minden irányított körmentes gráfhoz (**DAG**) megfeleltethető a csúcsai egy részbenrendezése, amelyben u $\leq$ v pontosan akkor áll fenn, ha a gráfban létezik u-ból v-be menő irányított út. Ugyanakkor egy ilyen részbenrendezést sok különböző irányított körmentes gráf is reprezentálhat. Ezek közül a legkevesebb élt a tranzitív redukált, a legtöbbet a tranzitív lezárt tartalmazza.

Egy gráf tranzitív lezártja könnyen kiszámítható pl. a Floyd-Warshall-algoritmussal O(n^3) időben, illetve n szélességi kereséssel O(n(n+m)) időben. Ugyanakkor vannak lényegesen hatékonyabb módszerek is, amelyek a gyakorlatban majdnem lineáris időben futnak.

Egy ilyen módszer az erősen összefüggő komponensek meghatározásán, valamint a komponensek gráfjának topologikus rendezésén alapul, de összetett adatszerkezeteket is alkalmaz.