Dinamikus gráfok I

Iván Szabolcs

2016 ősz

ACM feladat

Network

- Adott: $N \sim 100.000$ számítógép.
- Kezdetben nincs köztük semmilyen hálózati összeköttetés.
- Kétféle parancs jöhet az inputon:
  - insert(i, j) – összeköti a két gépet közvetlen (kétirányú) vonallal
  - query(i, j) – elérhető-e i-ből j az eddig felépített hálózatban?

Erre kell adjunk algoritmust. (Parancsból akár $\sim 1.000.000$ is jöhet.)

Klasszikus DFS

Mélységi / szélességi keresés

Ha a gráfot nyilvántartanánk és mindig futtatnánk egy DFS-t:

- insert(i, j) – konstans idő
- query(i, j) – $O(n + m)$ idő (csúcsok + élek száma)

Lassú

Ez ebben az esetben akár 10^{11} össz időigényű is lehet.

Összefüggő komponensek

Ötlet

- Elég az összefüggő komponenseket nyilvántartani valahogy
- insert(i, j) – ha i és j különböző komponensben vannak, akkor ezeket összeolvasztjuk
- query(i, j) – i és j ugyanabban a komponensben vannak-e?

Union-Find Forest

Ezek a műveletek épp az Union-Find adatszerkezet által biztosítottak!

Megoldás

- Inicializálás: makeSets(N) – időigény: $O(N)$
- insert(i, j): union(i, j) – időigény: $O(\alpha(N))$
- query(i, j): find$(i) ==$ find(j) – időigény: $O(\alpha(N))$

Általánosan

Dinamikus gráfproblémák

Általánosan, kiindulunk egy G input gráfból, és a következő kérések jönnek be:

- insert(i, j) – behúzzuk az (i, j) élt
- erase(i, j) – töröljük az (i, j) élt
- query(?) – az aktuális gráfunkra futtatunk egy lekérdezést

A ,,lekérdezés" ebben a példában épp az Elérhetőség volt: az i csúcsból az aktuális példányunkban elérhető-e a j csúcs.

Incremental / Decremental / Fully Dynamic

- Ha nincs erase: inkrementális
- Ha nincs insert: dekrementális
- Ha mindkettő lehet: teljesen dinamikus

Elérhetőség

Inkrementális elérhetőség

Az előző tehát egy inkrementális dinamikus algoritmus volt az irányítatlan gráfok elérhetőség problémájára.

Dekrementális

Próbáljunk meg egy dinamikus algoritmust adni ugyanerre a problémára!

A probléma

Adott egy G gráf, N csúcsú, M élű. Bejön Q darab kérés, a következő alakúak:

- erase(i, j) – töröljük az (i, j) élt
- query(i, j) – elérhető-e i-ből j az aktuális gráfunkban?

A cél $o((N + M) \cdot Q)$ időigényű algoritmust adni.

Erdőkre

Erdőkre

Először adjunk egy megoldást arra az esetre, ha az eredeti G gráfunk egy erdő, vagyis ha nincs benne kör.

Ötlet

Tartsuk nyilván a gráfot és minden csúcs kapja meg a komponensének az azonosítóját!

- Inicializálás: $O(N + M)$ idő alatt, mélységi bejárással.
- Lekérdezés: $O(1)$, csak a címkéket kell összehasonlítani.
- Update: ???

Erdőknél...

...ha egy élt elveszünk, az biztosan elvágja a komponensét.

Komponens-átszínezés

A gond

Ha az $\textsc{erase}(i, j)$ parancsnál (mondjuk) az i csúcs komponensét színezzük át DFS-sel, az rossz esetben akár $\Theta(N)$ lépés is lehet

Egy megoldás

Színezzük át mindig a kisebbet!

Egy kérdés

Honnan tudjuk, melyik a kisebb?

Egy válasz

Indítunk egy-egy DFS-t i-ből is, j-ből is. Párhuzamosan.
Amelyik előbb végez, az a kisebb komponens, a másik bejárását lelőjük.

Komponens-átszínezés

Időigény?

Ha ekkor egy K méretű komponenst egy K_1 és egy K_2 méretűre vágunk szét, ahol $K_1 \leq K_2$, akkor egy ilyen átszínezés költsége $O(K_1)$.

Ez pl. az első vágásnál lehet $N/2$ igényű is...

...de ha két $N/2$ méretű részt kapunk, akkor mikor azokból veszünk el további élt, akkor az azokból törlés már csak $N/4$ időigényű lehet

Amortizált időigény

Nem úgy számoljuk a teljes időigényt, hogy a legelső lépés $O(N)$-es korlátjával becsüljük mindet, hanem egy $T(N)$ időigényt rendelünk az egész számításhoz:
$T(N)$ legyen annak a maximális összköltsége, amennyi ideig az erase műveletekkel egy N-csúcsú fa összes élének eltüntetése tarthat.

Amortizált időigény

Rekurzív összefüggés

Egy N-csúcsú fából minden él eltörlésének az ideje:

- először elveszünk egy élt úgy, hogy egy K- és egy $N-K$-csúcsú fát kapunk, ennek költsége K, ha $K \leq N-K$;
- majd eltöröljük a két fa éleit külön-külön.

Tehát

$$T(N) = \max_{k=1}^{N/2} T(k) + T(N-k) + k.$$

De az mennyi?

Hogy számoljuk ilyennek a nagyságrendjét?

Rekurzió becslése

Ha N páros, akkor lehet, hogy a maximumot $k = N/2$-nél veszi fel.

Mindenesetre $N = 2$-nél biztos. Indukció?

Vegyük $T(2N)$-t. Írjuk fel annak a költségét, ha ezt $N + d$ és $N - d$ méretű fákra vágjuk:

$$T(N + d) + T(N - d) + N - d.$$

Indukció szerint ennek a kettőnek a maximumhelye a felénél van:

$$T(\frac{N+d}{2}) + T(\frac{N+d}{2}) + T(\frac{N-d}{2}) + T(\frac{N-d}{2}) + \frac{N+d}{2} + \frac{N-d}{2} + N - d.$$

Rekurzió becslése

$$T(\frac{N+d}{2}) + T(\frac{N+d}{2}) + T(\frac{N-d}{2}) + T(\frac{N-d}{2}) + \frac{N+d}{2}$$
$$+\frac{N-d}{2} + \frac{N-d}{2} + \frac{N-d}{2}.$$

A piros részek összege $\leq T(N)$. A kékeké is. Tehát ez legfeljebb

$$T(N) + T(N) + N$$

azaz felezni legalább olyan nagy költség $\Rightarrow$ felezni kell.

Páratlanra hasonló módon megy.

A rekurzió, egyszerűsítve

$$T(N) = 2T(\frac{N}{2}) + \frac{N}{2}.$$

Rekurzió becslése

A rekurziónk

$$T(N) = 2T(N/2) + N/2.$$

A mester tétel

Ha $T(n) = a \cdot T(n/b) + f(n)$, akkor

- ha $f(n) = O(n^c)$ egy $c < \log_b a$)-ra, akkor $T(n) = \Theta(n^{\log_b a})$.
- ha $f(n) = \Theta(n^{\log_b a} \log^k n)$, akkor $T(n) = \Theta(n^{\log ba} \log^{k+1} n)$.
- ha $f(n) = \Omega(n^c)$ egy $c > \log_b a$-ra, akkor $T(n) = O(f(n))$.

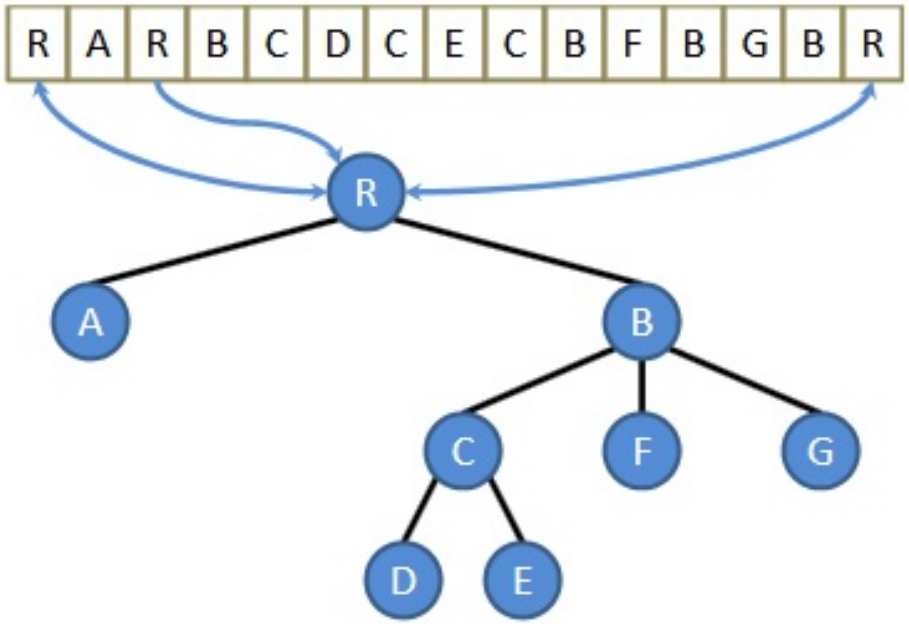
Most $a = 2$, $b = 2$ és $f(n) = n/2$, tehát $\log_b a = 1$, és $f(n) = \Theta(n^1 \log^0 n)$, tehát $T(n) = \Theta(n \log n)$.

Tehát ez a dekrementális elérhetőség-algoritmus erdőkre $O(N \log N + Q)$ időigényű, egy query időigénye $O(1)$, egy erase időigénye pedig amortizáltan $\log N$ (összesen lesz N törlés, az összes költségük $N \log N$).

Tudunk ennél jobbat?

Euler-Tour (ET) fák

Egy fát egy Euler-körsétájával is tárolhatunk.



Minden csúcs bemutat az ő első és utolsó előfordulásába; minden előfordulás bemutat a csúcsba.

ET fák

Támogatott műveletek:

- root(i) – visszaadja a fa gyökerét
- cut(i) – levágja az i csúcsot (részfájával együtt)
- query(i, j) – egy fában van-e i és j

Hogyan?

- A sétát pl. egy kétirányú láncolt listában is tárolhatjuk.
- root(i) visszaadja a lánc fejét, ez $O(1)$ idő.
- query(i, j) visszaadja, hogy root(i) == root(j) teljesül-e. Ez is $O(1)$ idő.

ET fák – vágás

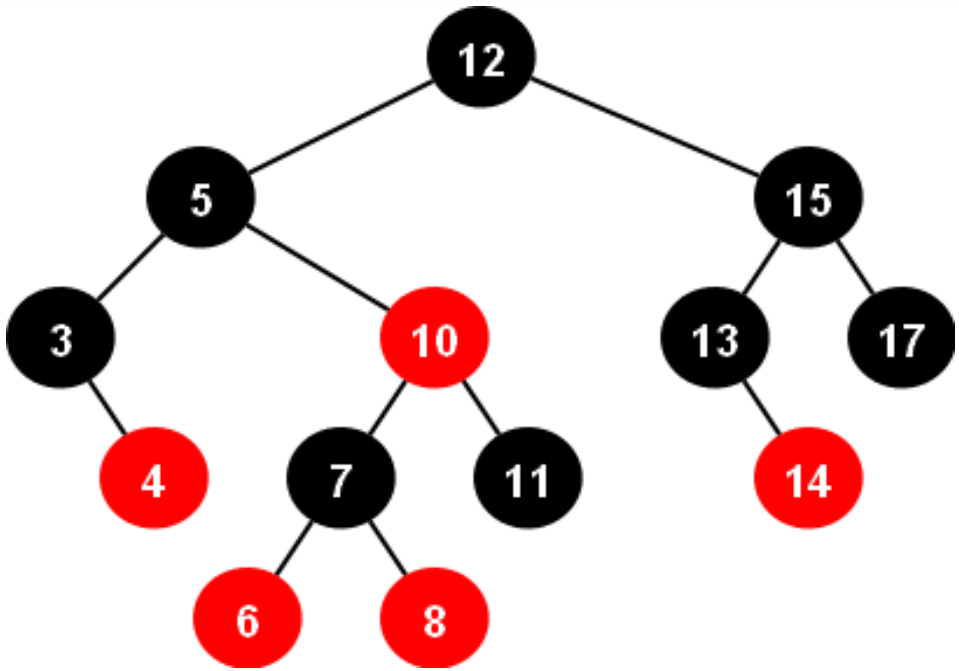
Hogyan?

- cut(i) – levágja az i csúcsot: az Euler-körsétából kivágjuk az i első előfordulásától az utolsóig tartó szakaszt – a láncolt listából ez is $O(1)$ idő, de updatelni kell a gyökeret!
  Az sokáig is eltarthat...

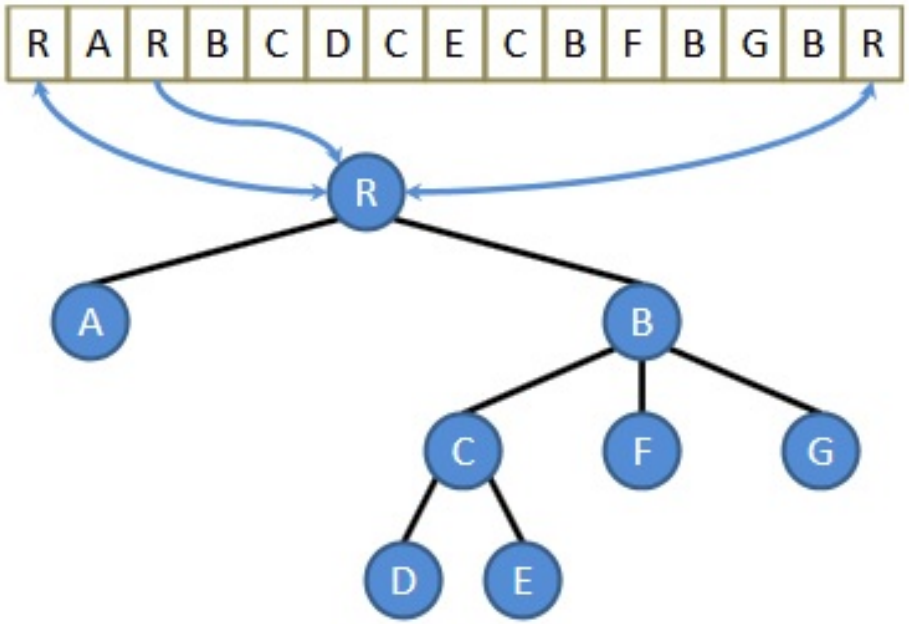
Kiegyensúlyozott bináris keresőfák

BST

- Kulcs-érték párokat tárolnak
- A kulcsokon van egy rendezés
- Ha n elem van, akkor a keresés/minimum/maximum/beszúrás/törlés időigénye $O(\log n)$



Vissza az ET fákhoz



Láncolt lista helyett egy pozíció-node BST-ben is tárolhatunk egy Euler-utat: itt épp $1 \mapsto R$, $2 \mapsto A$, ...

Nem baj, ha pozíciók kimaradnak! $\Rightarrow$ a vágás is gyors lesz, BST-ket splittelni is lehet $O(\log n)$ időben.

Ekkor a fa gyökere a minimális indexen lesz, az is $O(\log n)$.

Általános gráfok

És ha nem csak erdőből indulunk?

Ekkor nem biztos, hogy egy él elvétele elvágja a gráfot.

Párhuzamosan futtatunk dolgokat

Egy erase(i, j) kérésnél párhuzamosan...

- indítunk egy DFS-t i-ből;
- indítunk egy DFS-t j-ből;
- és csinálunk még valamit;

és ha (i, j) elvágja a komponenst, akkor az első két DFS valamelyike a kisebb komponens méretével arányos idő alatt végez (és ekkor átszínezünk, mint az első algoritmusnál), ha pedig nem, azt a harmadik pont fogja detektálni.

Csinálunk még valamit

- A gráfnak nyilvántartjuk egy szélességi bejárását is egy fában.
- Felveszünk fiktív éleket a komponensek gyökerei között.
- Ebben a fában minden i-ből induló él a következők egyike lesz:
  - Visszaél: minden $k > 0$ szintű csúcsból legalább egy él vezet egy $k - 1$ szintű csúcsba;
  - Előreél: minden k szintű csúcsból vezethet tetszőlegesen sok él $k + 1$ szintű csúcsokba;
  - Lokális él: a k szintű csúcsból k szintű csúcsba vezető éleket lokálisnak nevezzük.

Lokális él törlése

Ha lokális élt törlünk, az garantáltan nem vágja szét a komponenst, ez $O(1)$ idő alatt kiderül.

Csinálunk még valamit

Ha (i, j) különböző szintűek, mondjuk i szintje $k-1$, j szintje pedig k, akkor:

- ez az él i-ből előre, j-ből vissza mutat. Töröljük őket az előre/vissza listából.
- Ha j-nek nem üres a vissza listája, akkor még mindig összefüggő ez a komponens, minden rendben, visszatérhetünk.
- Ha j-nek a vissza listája kiürült, megpróbáljuk újraszámolni a szinteket

Szintek újraszámolása

- Betesszük j-t egy Q sorba.
- Amíg Q nem üres, kivesszük az első elemét, legyen ez w
- Megnöveljük w szintjét
- A w eddigi lokális x szomszédjai legyenek w új vissza-szomszédjai és nekik x legyen előre-szomszédja
- Ezek után updateljük w előre-szomszédjait: nekik w már lokális szomszédja lesz (eddig hátra-szomszéd volt), és ha így kiürül a hátra-szomszéd listájuk, tegyük be őket Q-ba.
- A w csúcs eddigi előre-szomszédjai legyenek most már a lokális szomszédai, új előre-szomszédjai pedig ne legyenek.

Időigény

Ha a gráf nem esett szét,

akkor ez az algoritmus $O(N)$ idő alatt újraszámolja a szinteket.

Ha a gráf szétesett,

akkor arra a másik két DFS egyike rájön. Ekkor ne változtassuk meg az eredeti BFS-t mégsem, hanem csak az (i, j) élt váltsuk át fiktívre.

Teljes időigény

- Amikor egy (i, j) élt elveszünk, és nem esik szét komponens, úgy a két végpontjai közül az egyiknek nő a szintje.
- A szint legfeljebb N lehet.
- Így a teljes költség legfeljebb $M \cdot N$ lesz.
- Teljes időigény: $O(Q + N \cdot M)$.