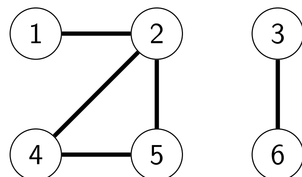
Algoritmusok dinamikus gráfokon

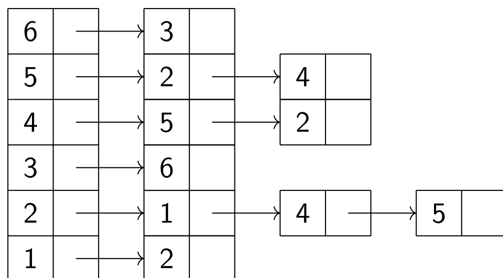
Iván Szabolcs

2022. szeptember 30.

Gráfok



Egy gráf



Számítógépes reprezentációja éllistával

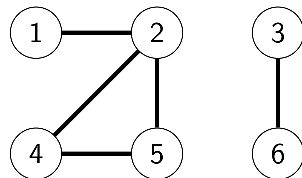
A kérdés: kapunk két csúcsot, van-e köztük út?

pl. $(1, 5) \Rightarrow$ „igen”, $(2, 3) \Rightarrow$ „nem”

Depth-First Search, DFS

- használunk egy vermet (kiknek kell még megnézni a szomszédját) és egy tömböt (kiket láttunk már)
- verembe lerakjuk a kezdőcsúcsot
- tömbben a kezdőcsúcsot 1-re állítjuk, többit 0-ra
- amíg találunk valakit a veremben:
  - kivesszük
  - bejárjuk a szomszédait
  - akit közülük még nem láttunk (0 az értéke a tömbben), azt betesszük a verembe és 1-re állítjuk
- ha így elérjük a célcsúcsot, akkor van út, ha kiürül a verem, akkor nincs

DFS



Kérdés: $(4, 3)$

1	2	3	4	5	6
			1		
	1		1	1	
	1		1	1	
1	1		1	1	
1	1		1	1	

4					
2	5				
2					
1					

„nem”

Futásidő

- hány lépésen belül fut le biztosan az algoritmus?
- attól függ, hogy mekkora a gráf: hány éle és hány csúcsa van
- legyen n a csúcsok és m az élek száma
- akkor a DFS lépésszáma $n + m$-mel arányos lesz max

minden élt max egyszer nézünk meg mindkét végéről

plusz az elején beállítjuk a tömböt és a vermet

- jelben $O(n + m)$

kb. ,,valahányszor $n + m$ lépés már elég lesz biztosan''

Dinamikus gráfok

- kiindulunk egy gráfból
- amit updatelhetünk: beszúrhatunk / törölhetünk egy élt
- és queryzhetünk: megkérdezhetünk két csúcsot, hogy az aktuális gráfban van-e köztük út
- egy megoldás mindig megfelelően változtatni az éllistát update-kor ($O(m)$ idő) és mindig újrafuttatni egy DFS-t query-kor ($O(n+m)$ idő)
- ennél szeretnénk jobbat

általában pl. $O(\log m)$ az, aminek örülnénk

Ha csak élbeszúrás van

- kiindulunk egy üres gráfból
- update: csak élbeszúrás lehet
- query: (mindig) ,,eközött a két csúcs közt van-e út”

Összefüggő komponensek

Ötlet

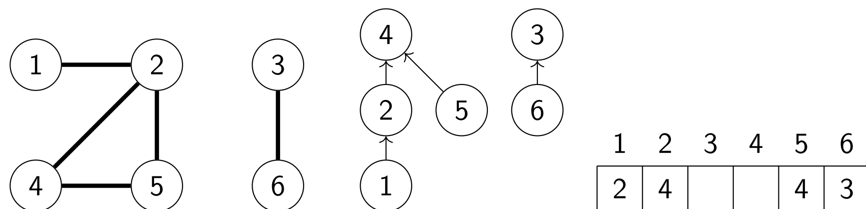
- Elég az összefüggő komponenseket nyilvántartani valahogy
- insert(i, j) – ha i és j különböző komponensben vannak, akkor ezeket összeolvasztjuk
- query(i, j) – i és j ugyanabban a komponensben vannak-e?

Union-Find Forest

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

Union-Find Forest

- minden csúcsnak lehet (max) egy „őse”
- „fákat” alkotnak
- aki egy fában van, az van egy komponensben



- query: felmegyünk a gyökérig, ha ugyanaz, akkor egy komponensben vannak
- insert: ha különböző fában vannak, azokat egyesíteni kell
- az egyik fa gyökerét betesszük a másik fa gyökere alá

Union-Find Forest

- annál gyorsabb, minél ,,alacsonyabbak" a fák
- find: akiket felfelé menet látunk, a végén kössük a gyökér alá közvetlenül

legközelebb gyorsabb lesz

- melyik gyökeret tegyük melyik alá: pl. tároljuk a csúcsszámot és a kisebbet a nagyobb alá

Ezzel k darab egyesítés / lekérdezés összköltsége alig lesz több, mint $O(k)$
Azaz: kb. mintha konstans időben tudnánk updatelni is, lekérdezni is

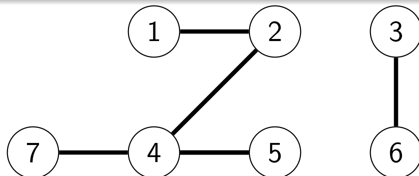
Éltörlés

- most nézzük azt az esetet, amikor az alap gráfunk nem üres
- update: csak törlés
- erre nem lesz jó az Union-Find Forest: hogy ,,szednénk szét''?

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: ???

Komponens-átszínezés

A gond

Ha az ERASE(i,j) update-nél (mondjuk) az i csúcs komponensét színezzük át DFS-sel, az rossz esetben akár n lépés is lehet

Egy megoldás

Színezzük át mindig a kisebbet!

Egy kérdés

Honnan tudjuk, melyik a kisebb?

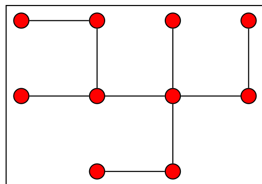
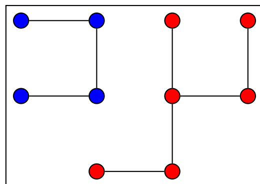
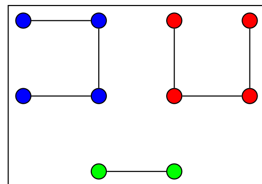
Egy válasz

Indítunk egy-egy DFS-t i-ből is, j-ből is. Párhuzamosan.

pl. mindig azt léptessük tovább, aki eddig kevesebb csúcsot látott.

Amelyik előbb végez, az a kisebb komponens, a másik bejárását lelőjük.

Flood Fill Forest

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 $k \le n-k$;
- 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

Némi számolással kijön, hogy a maximumot felezéskor veszi fel ez a függvény.

A rekurzió, egyszerűsítve

$$T(n) \leq 2T\left(\frac{n}{2}\right) + \frac{n}{2}.$$

A mester tétel

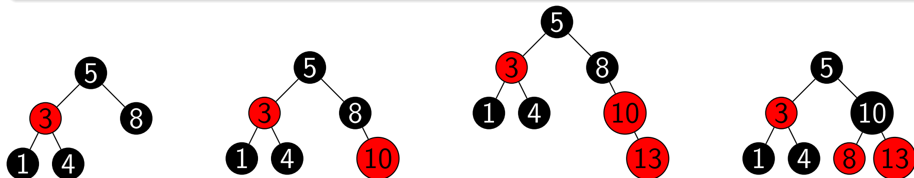
Van egy tétel ilyen alakú rekurzió feloldásra, eszerint $T(n) = O(n \log n)$.

Tehát ha erdőből indulunk, akkor lesz $O(\log n)$ (amortizált) update költségünk és $O(1)$ (tényleges) query költségünk, ha az egész fát eltüntetjük végül.

Tudunk ennél jobbat?

Piros-Fekete Fa (full) (C++ <SET>,<MAP>, JAVA TREESET)

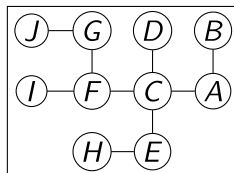
- Kulcs-érték párokat tárol, ahol a kulcsokon van egy rendezés (pl. számok, stringek)
- Támogatott műveletek és időigényük n darab tárolt pár esetén:
  - init: létrehozza üresen, $O(1)$ idő
  - insert(K, V): beszúrja a (K, V) párt, $O(\log n)$ idő
  - remove(K): törli a K kulcsot és a hozzá tartozó értéket, $O(\log n)$ idő
  - min: visszaadja a minimális kulcsot a halmazban, $O(\log n)$ idő
  - next(K): a K után következőt a rendezés szerint, $O(\log n)$ idő
  - split(T, x): szétvág egy fát x-nél, $O(\log n)$ idő

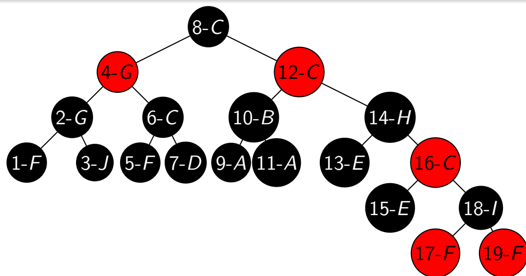


Euler Tour Forest

- Egy n-csúcsú erdőt tárol
- Támogatott műveletek:
  - init: létrehozza üresen, $O(1)$ idő
  - insert(u, v): behúzza az (u, v) élt az erdő két fája közé $O(\log n)$ időben
  - remove(u, v): törli az (u, v) élt $O(\log n)$ időben
  - find(u): visszaadja u fájának egy reprezentánsát $O(\log n)$ időben

FGJGFCDCABACDCEHECFIF





Ha a gráfunk garantáltan erdő marad végig, akkor ez $O(\log n)$ update költség (beszúrásra és törlésre is), és $O(\log n)$ query költség $\Rightarrow$ örülünk

Fully dynamic connectivity

- Input: egy iniciálisan üres G gráf
- Parancsok: insert(u, v), remove(u, v) és query(u, v)

Ötlet

- Nyilvántartjuk G egy feszítőerdőjét (ET fákkal)

  egy olyan erdőt, amiben egy komponens csúcsai egy fába kerülnek, és csak az eredeti gráf éleit használja
- Ekkor könnyű két erdő közé az insert: össze kell őket olvasztani
- Ekkor könnyű egy erdőben az insert: akkor a feszítő erdő nem változik
- Ekkor könnyű egy él törlése, ami nem az erdőben van: akkor a feszítő erdő nem változik
- Nehéz viszont egy erdőbeli él törlése!

Törlés feszítő erdőből

Ötlet

- Minden (i, j) élnek lesz egy $\ell(i, j)$ szintje.
- Egy él szintje 1 és $\log n$ közti egész szám lesz.
- Egy él szintje az idővel csak csökkenhet majd.
- G_i: a G-nek az a részgráfja, melyben a legfeljebb i szintű élek szerepelnek.
- F_i: a G_i-nek egy feszítőerdője.

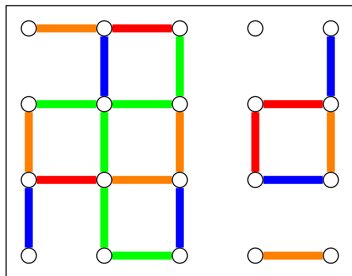
Invariánsok

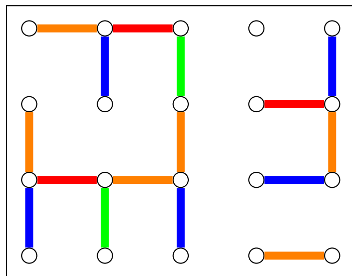
A futás során végig fenn fogjuk tartani a következő tulajdonságokat:

- A G_i gráf minden komponense legfeljebb 2^i méretű.
- $F_1 \subseteq F_2 \subseteq \ldots \subseteq F_{\log N}$.

Példa

$n = 20$ csúcs, az élek szintjei egy, kettő, három vagy négy. (lehetne még 5 is, olyan él itt épp nincs.)
A kék komponensek mérete legfeljebb 2, a kék-narancs komponenseké legfeljebb 4, a kék-narancs-piros komponenseké legfeljebb 8, stb.





Törlés feszítő erdő-hierarchiából

- A G_i gráfok éllistáit is tároljuk. (Így be tudjuk járni hatékonyan az egy csúcsra illeszkedő, adott szintű éleket.)

  él„lista” helyett él„keresőfákat” tárolva, hogy gyorsan lehessen belőlük törölni is

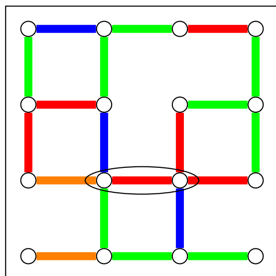
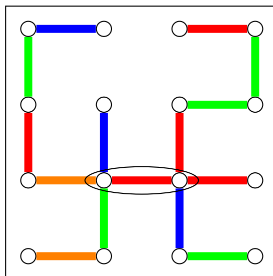
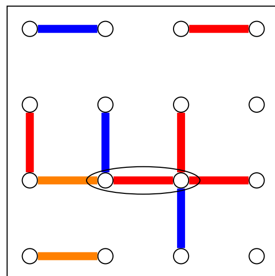
- insert(i,j): beszúrjuk az (i,j) élt, $\ell(i,j) := \log n$.
  Ha i és j az $F_{\log N}$-nek különböző fáiban vannak, akkor egybe mergeljük a két fát és behúzzuk $F_{\log n}$-be az (i,j) élt. Ez $O(\log n)$ idő.
- query(i,j): a két csúcs akkor van egy komponensben, ha $F_{\log n}$-nek egy fájában vannak. Ez $O(\log n)$ idő.
- delete(i,j):
  - Kivesszük az (i,j) élt (G és a G_i-k szomszédsági listáiból).
  - Ha (i,j) nincs benne $F_{\log n}$-ben, nincs gond, megyünk tovább.
  - Ha igen, akkor benne van az összes F_k-ben, amire $k \geq \ell(i,j)$. Töröljük ki belőlük. ez $O((\log n)^2)$ idő lehet
  - Ezek után keresünk egy élt, amivel össze tudjuk kötni az i és j fáját.

Csere-él keresése

- Mivel minden F_k egy minimális feszítőerdő, így $\ell(i,j)$-nél kisebb szintű éllel nem tudjuk összekötni (i,j) helyett a széteső fákat.
- Tehát a $k = \ell(i,j), \ldots, \log n$ szintű élek közt keresünk összekötő élt, ilyen sorrendben.
- Legyen T_i és T_j az i-t és j-t tartalmazó fa F_k-ban úgy, hogy $|T_i| \leq |T_j|$.
- Mivel eredetileg az invariáns miatt $|T_i| + |T_j| \leq 2^k$, így $|T_i| \leq 2^{k-1}$.
- Járjuk be az összes olyan (x,y) élt, amire $x \in T_i$ és $\ell(x,y) = k$.
- Ha $y \in T_j$, akkor adjuk hozzá az (x,y) élt a $T_k, \ldots, T_{\log N}$ fákhoz és végeztünk.
- Különben csökkentsük $\ell(x,y)$-t eggyel.

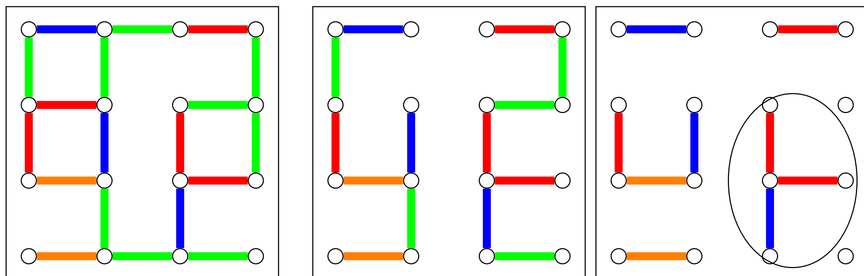
Csere-él példa

$n = 16$ csúcs, az élek szintjei egy, kettő, három vagy négy.
A kék komponensek mérete legfeljebb 2, a kék-narancs komponenseké legfeljebb 4, a kék-narancs-piros komponenseké legfeljebb 8, stb.

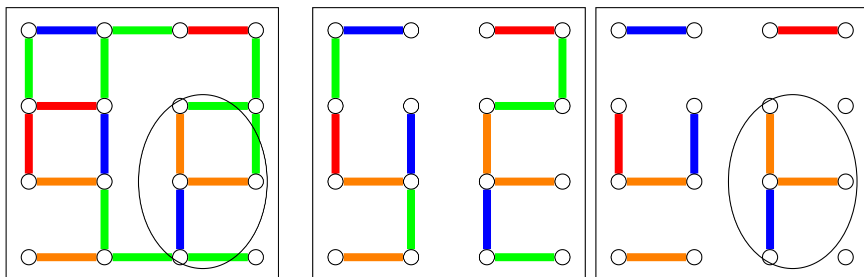
Csere-él példa

$n = 16$ csúcs, az élek szintjei egy, kettő, három vagy négy.
A kék komponensek mérete legfeljebb 2, a kék-narancs komponenseké legfeljebb 4, a kék-narancs-piros komponenseké legfeljebb 8, stb.



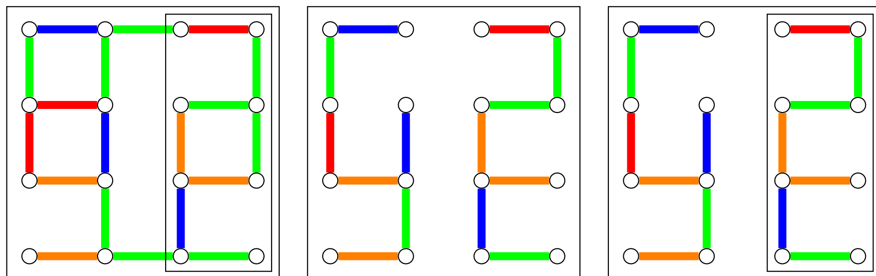
Csere-él példa

$n = 16$ csúcs, az élek szintjei egy, kettő, három vagy négy.
A kék komponensek mérete legfeljebb 2, a kék-narancs komponenseké legfeljebb 4, a kék-narancs-piros komponenseké legfeljebb 8, stb.



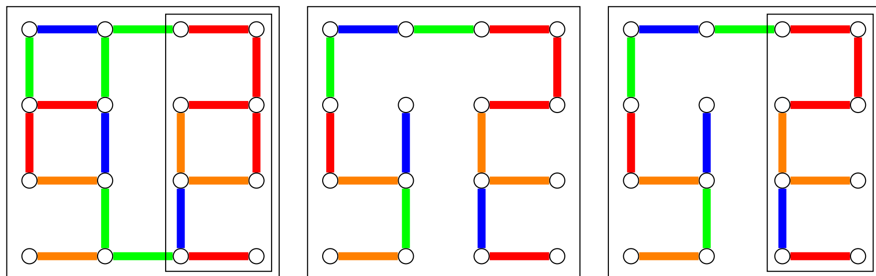
Csere-él példa

$n = 16$ csúcs, az élek szintjei egy, kettő, három vagy négy.
A kék komponensek mérete legfeljebb 2, a kék-narancs komponenseké legfeljebb 4, a kék-narancs-piros komponenseké legfeljebb 8, stb.



Csere-él példa

$n = 16$ csúcs, az élek szintjei egy, kettő, három vagy négy.
A kék komponensek mérete legfeljebb 2, a kék-narancs komponenseké legfeljebb 4, a kék-narancs-piros komponenseké legfeljebb 8, stb.



Vissza az erdőkből törlésre

Tehát: a (kisebb) T_1 fában végignézzük az összes k szintű élt.
Ha az él a fán belül megy: csökkentjük a szintjét. (Ez élenként $\log n$ idő ET forestben).
Egyébként: behúzzuk az ET forest két fája közé és végeztünk a törléssel.
Hagyományosan számolva: egy a törlés költsége a k. szinten lehet $O(e \cdot \log n)$, ahol e a fában scannelt élek száma. (Ez soknak látszik, nem jó felső korlát.)

Ehelyett:

Az él szintjének csökkenésével járó scanneléseket előre kifizetjük az él beszúrásakor!

Az e élt $\log n$ szintűként szúrjuk be $\Rightarrow$ kifizetünk a $\log n$ költségű beszúráson felül pluszban még $(\log n)^2$ költséget. Így minden beszúrt él $(\log n)^2$ „tartalékkal" indul.

Vissza az erdőkből törlésre

Törléskor: a scannelt élek közül aminek csökkentjük a szintjét, azt törléskor nem számoljuk, hanem az él által a beszúráskor felhalmozott tartalékból vonjuk le.
Így szint csökkenéskor ebből a tartalékból vonunk le $\log n$-t.
Az eredeti $\log n$ szintet 1-ig tudjuk csak csökkenteni, annál tovább nem fogjuk, ezért a kezdeti tartalék fedezi az ilyen költségeket végig.
Ha nem csökken a szint, azt a törléshez számoljuk. Abból viszont csak egy lesz legfeljebb törlésenként, így a beszúrás amortizált költsége $O((\log n)^2)$, a törlésé pedig $O(\log n)$, összesen az update amortizált költsége $O((\log n)^2)$.

Összefoglalva

- Láttunk egy gráfproblémát, az összekötöttséget, irányítatlan dinamikus gráfokra
- Beszúrást, törlést $O((\log n)^2)$, lekérdezést $O(\log n)$ költséggel támogató adatszerkezet + algoritmus kombinációt rá
- Van még rengeteg másik gráfprobléma, sok más adatszerkezet, komplikáltabb algoritmusokkal
- Intenzíven kutatott terület

Köszönöm a figyelmet!

A prezentációt a TKP2021-NVA-09 projekt támogatása tette lehetővé