Oblikovanje i analiza algoritama

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

04. prosinca, 2023.

Kruskalov algoritam

- Algoritam za pronalaženje minimalnog razapinjućeg stabla grafa $G = (V, E)$.
- Krećemo od $N = |V|$ disjunktnih komponenti povezanosti grafa $s_i = \{v_i\}$.
- Izabiremo iterativno bridove najkraće duljine.
- Ukoliko brid povezuje dvije nepovezane komponente, stvorimo veću komponentu povezanosti (koja sadrži obje). Inače, odbacujemo brid jer bi zatvorio ciklus.
- Završavamo kada dobijemo samo jednu komponentu povezanosti koja čini MST.

Kruskalov algoritam

- $G = (V, E)$ - graf
- l - funkcija udaljenosti
- T - skup bridova

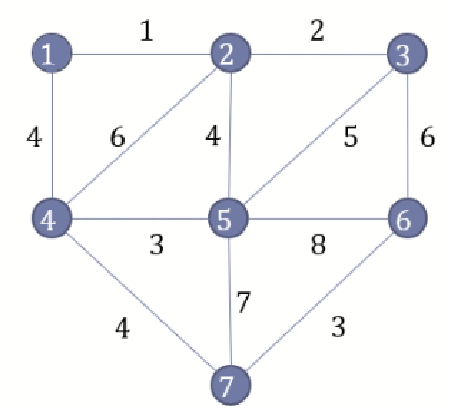
```
void Kruskal(G, l, T){  //inicijalizacija
  sort(E, comp.asc) ;//uzlazno po duljinama l bridova
  T = emptySet(); N = |V|;
  for(i:{1,...,N}) si = {vi};
  do{
      (u,v) = minWeightWhiteEdge()
        ucomp = find(u); //nalazimo komponente povezanosti
        vcomp = find(v); //kojima pripadaju vrhovi u i v
        if(ucomp != vcomp){// prihvati brid (u,v)
         merge(ucomp, vcomp);
         T = setUnion(T, {(u,v)})
        }
        else continue;
  }while(|T|!=N-1);
}
```

Kruskalov algoritam

Teorem: Za povezan usmjeren graf G, Kruskalov algoritam radi korektno.

Dokaz: indukcija po broju izabranih bridova do tog trenutka uz primjenu Leme (definirana kod Primovog algoritma).

Primjer



Primjer

Bridovi grafa **sortirani uzlazno** po duljini:

1	2	3	3	4	4	4	5
$\{1,2\}$	$\{2,3\}$	$\{4,5\}$	$\{6,7\}$	$\{1,4\}$	$\{2,5\}$	$\{4,7\}$	$\{3,5\}$
6	6	7	8				
$\{2,4\}$	$\{3,6\}$	$\{5,7\}$	$\{5,6\}$				

Primjer

korak	promatrani brid	povezane komponente (V, T)
init	–	$\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{7\}$
1	$\{1, 2\}$	$\{1, 2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{7\}$
2	$\{2, 3\}$	$\{1, 2, 3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{7\}$
3	$\{4, 5\}$	$\{1, 2, 3\}$, $\{4, 5\}$, $\{6\}$, $\{7\}$
4	$\{6, 7\}$	$\{1, 2, 3\}$, $\{4, 5\}$, $\{6, 7\}$
5	$\{1, 4\}$	$\{1, 2, 3, 4, 5\}$, $\{6, 7\}$
6	$\{2, 5\}$	odbacuje se (ciklus)
7	$\{4, 7\}$	$\{1, 2, 3, 4, 5, 6, 7\} = V$ (kraj izvođenja)

Zadatak: Graf može imati više minimalnih razapinjućih stabala. Kako se to odražava u Kruskalovom algoritmu?

Zadatak: Što se događa u Kruskalovom algoritmu ako graf G nije povezan?

Trebamo što efikasnije realizirati operacije:

- `find(x)` - nalaženje komponente povezanosti u kojoj se nalazi vrh x
- `merge(a,b)` - spajanje dva disjunktna skupa (dvije disjunktne komponente povezanosti) u jedan skup (unija disjunktnih skupova).

Struktura disjunktnih skupova

- Kako efikasno implementirati funkcije `find` i `merge`?
- Struktura disjunktnih skupova.
  - Imamo N objekata numeriranih brojevima 1 do N. Želimo naći particiju skupa $\{1, 2, \ldots, N\}$.
- Operacije:
  - `find` - za dani objekt nađi skup (klasu ekvivalencije kojem on pripada i vrati oznaku tog skupa).
  - `merge` - spoji dva skupa u jedan (objekte nakon grupiranja više ne razlikujemo).
- Reprezentacija:
  - oznaka skupa - najmanji element skupa (pohlepni pristup)
    - skup $\{6, 2, 1, 9\}$ je "skup 1" (skup$[9] = 1$)
  - inicijalizacija (jednočlanih skupova):

```
void init(){
    for(int k = 1;k<=N;k++) skup[k] = k; }

```

Struktura disjunktnih skupova

Predstavit ćemo nekoliko načina implementacije traženih funkcija.

- funkcija find1

```
    int find1(int x){
        return skup[x];}

```

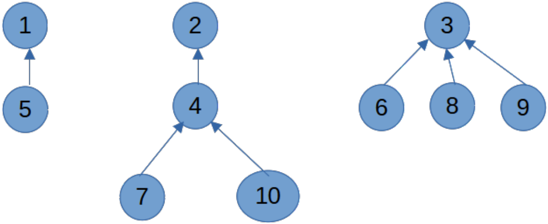
- funkcija merge1

```
    void merge1(int a,int b){
      int i = a, j = b;
      if(i>j) swap(&i,&j);
      for(int k=1;k<=N;k++)
          if(skup[k] == j) skup[k] = i;
    }

```

- Složenost:
  - `find1` - $\mathcal{O}(1)$
  - `merge1` - $\mathcal{O}(N)$
- Niz od n operacija (`find1` ili `merge1`) - $\mathcal{O}(nN)$.
- Promotrimo malo drugačiju reprezentaciju:
  - oznaka skupa - korijen (obrnutog) stabla.

1	2	3	4	5	6	7	8	9	10
1	2	3	2	1	3	4	3	3	4



Struktura disjunktnih skupova

```
void merge2(int a,int b){
        if(a<b) skup[b] = a;
        else skup[a] = b;
    }

```

```
    int find2(int x){
        int i = x;
        while(skup[i]!=i) i = skup[i];
        return i;
    }

```

- Složenost:
  - `find2` - $\mathcal{O}(h_{max})$
  - `merge2` - $\mathcal{O}(1)$
  - Nakon niza od n operacija (`find2` ili `merge2`) - $\sum_{k=1}^{n} ck \in \mathcal{O}(n^2)$.

- Pokušat ćemo ograničiti (kontrolirati) visinu dobivenih stabala.
- Kod spajanja kao korijen ostavimo korijen stabla veće visine (povećamo visinu nižeg stabla)
- **pohlepno kontroliranje visine**
- korijen stabla je oznaka skupa
- kontrola visine određuje oznaku nakon spajanja
- Neka su $h_1,\ h_2$ visine stabala koje spajamo.
- Tada je h - visina stabla dobivenog spajanjem:

$$h = \begin{cases} max\{h_1, h_2\}, & \text{ako je } h_1 \neq h_2 \\ h_1 + 1, & \text{ako je } h_1 = h_2 \end{cases}$$

Zadatak: Dokažite matematičkom indukcijom da ovom tehnikom spajanja startajući iz `init`, nakon proizvoljnog broja operacija `merge`, stablo koje sadrži k objekata ima visinu h: $h \leq \lfloor lgk \rfloor$.

- pamtimo visine stabala:
  - visina[i] = visina objekta (vrha) i u njegovom trenutnom stablu
  - početna visina: visina[i] = 0, $\forall i = 1, \ldots, N$
- korijen ima najveću visinu (obratno korijensko stablo), a listovi imaju visinu nula.

Struktura disjunktnih skupova

```
void merge3(int a,int b){
      if(visina[a] == visina[b]){
             visina[a] = visina[a] + 1;
             skup[b] = a;
        }
      else{
            if(visina[a] < visina[b])
               skup[a] = b;
            else skup[b] = a;
      }
   }

```

- Primijetimo da su funkcije `merge1` i `merge2` radile korektno i ako je na ulazu bilo $a = b$. To ne vrijedi za `merge3` jer bi visina stabla s korijenom a narasla za 1 bez promjene stabla.

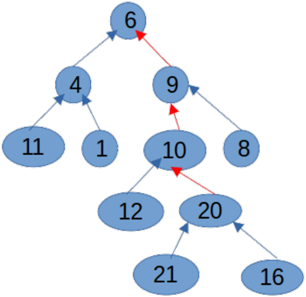
- **Zadatak**: Dokažite da za ukupno vrijeme potrebno za izvođenje niza od n operacija tipa `find2` ili `merge3`, u najgorem slučaju vrijedi:

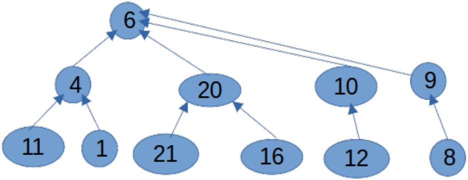
$$T_W(n \times \{find2, merge3\}) \in \mathcal{O}(nlogn)$$

- **Uputa:** koristite prethodni zadatak i činjenicu da se visine vrhova koji nisu korijeni više ne mijenjaju.

Struktura disjunktnih skupova

- Skraćivanje (kompresija puta, eng. *path compression*) - modifikacija funkcije `find2`.
- Smanjenje visine stabla ubrzava kasnije `find` operacije.
- **Primjer**: `find(20)`





Struktura disjunktnih skupova

```
int find3(int x){
      int r = x;

      while(skup[r]!=r)
         r = skup[r];

      int i = x, j;
      while(i!=r){
          j = skup[i];
          skup[i] = r;
          i = j;
      }

```

- Ako koristimo skraćivanje puteva:
  - ne vrijedi da je visina stabla s korijenom *a* dana s `visina[a]`.
  - vrijednost `visina[a]` ostaje gornja ograda za pravu visinu.
  - tu vrijednost zovemo rang stabla.

- Ukomponiramo polje rang u funkcije init i merge3.

```
void init(){
     for(int k = 1; k<=N;k++){
         skup[k] = k;
         rang[k] = 0;
      }
     }
```

```
void merge3(int a, int b){
     if(rang[a] == rang[b]){
        rang[a] = rang[a] + 1;
        skup[b] = a;
     }
     else{
       if(rang[a] < rang[b])
          skup[a] = b;
       else skup[b] = a;
     }
```

Zadatak: Može li se kod skraćivanja puteva efikasno pratiti i korektno postaviti prava visina novodobivenog stabla? (kako to utječe na funkciju `find4` koja bi tako pratila visinu i izvođenje n operacija `find4` ili `merge3`).

- analizu složenosti niza od n operacija tipa `find3` ili `merge3` nećemo detaljno provoditi (analiza je složena).
- može se pokazati da vrijedi

$$T_W(n \times \{find3, merge3\}) \in \mathcal{O}(nlog_2 N)$$

  uz uvjet $n \geq N$ što je i najčešći slučaj u praksi.
- Funkcija $lg^* : \mathbb{N} \mapsto \mathbb{N}$ je definirana s:

$$(1) lg^* N = min\{k | log_2 log_2 \dots log_2 N \leq 0\}$$

$$(2) lg^* N = \begin{cases} 0, & N \leq 1 \\ 1 + lg^*(log_2 N), & N > 1 \end{cases}$$

- Funkcija lg* izuzetno sporo raste.

Struktura disjunktnih skupova

- Vrijedi:

N	lg* N
1	0
$< 1, 2]$	1
$< 2, 4]$	2
$< 4, 16]$	3 (2^{2^2})
$< 16, 2^{16}]$	4
$< 2^{16}, 2^{65536}]$	5

- Za sve praktične potrebe $lg^* N$ možemo smatrati najviše konstantnim, tj. vrijeme za n operacija `find3` ili `merge3` je praktički linearno.
- Precizna analiza koristi Ackermannovu funkciju koja također ne daje linearno vrijeme u najgorem slučaju.
- **Nije poznata** linearna realizacija funkcija `find` i `merge`.
- **Složenost** $(m = |E|,\ N = |V|)$
    - $\mathcal{O}(m lg^* N)$
    - $\mathcal{O}(m\hat{\alpha}(m, N))$

Složenost Kruskalovog algoritma

- $G = (V, E)$, $N = |V|$, $m = |E|$
- Uzlazno sortiranje bridova ima složenost $\mathcal{O}(m log_2 m)$. Zbog $N - 1 \leq m \leq \frac{N(N-1)}{2}$, vrijedi $log_2 m = \Theta(log_2 N)$. Stoga je složenost sortiranja iz $\mathcal{O}(m log_2 N)$.
- Inicijalizacija N disjunktnih skupova je u $\mathcal{O}(N)$.
- Pohlepna petlja radi na strukturi disjunktnih skupova nad skupom od N vrhova. U najgorem slučaju imamo $2m$ `find` operacija i $N - 1$ `merge` operaciju. Ukupan broj operacija je $2m + N - 1$, stoga je složenost operacija iz ove petlje u $\mathcal{O}((2m + N - 1) lg^* N)$, odnosno (zbog $m \geq N$) $\mathcal{O}(m lg^* N)$.
- Preostale operacije imaju složenost najviše $\mathcal{O}(m)$.
- Ukupno potrebno vrijeme je zbroj prije izračunatih vremena. Pošto $lg^* N \in \mathcal{O}(log_2 N)$, ukupna složenost algoritma je iz $\mathcal{O}(m log_2 N)$.

Složenost Kruskalovog algoritma

- Ako je graf G gust, onda $m \approx \frac{N(N-1)}{2}$ i Kruskalov algoritam ima složenost u $\mathcal{O}(N^2 log_2 N)$. Primov algoritam ima složenost u $\mathcal{O}(N^2)$, stoga je njega bolje koristiti kod gustih grafova.
- Ako je graf G rijedak, onda je $m \approx N - 1$, a Kruskalov algoritam ima složenost u $\mathcal{O}(N log_2 N)$. U tom slučaju je Kruskalov algoritam puno brži od Primovog algoritma.