Algoritmi in podatkovne strukture – 2

Grafi

osnove, predstavitve, iskanje / sprehodi

Grafi

Graf je definiran kot:

- Imamo množico vozlišč (*vertex*), ki imajo svoje oznake: $V = \{x_1, x_2, \ldots x_n\}$.
- Obstaja pojem povezave (*edge*) med vozliščema, ki je par: (x_i, x_j), ter imamo množico takšnih parov E. Običajno označimo $|E| = m$.
- Potem je graf definiran kot $G = (V, E)$.

Različne dopolnitve definicije:

- pari, ki predstavljajo povezave, so urejeni – *usmerjen graf* (*directed graf* – DAG)
- množica E je prava množica – nimamo večkratni povezav
- povezave so trojke (x_i, x_j, w_{ij}) , kjer je w_{ij} utež (vrednost, *weight*) povezave – uteženi graf
- vozlišča grafa imajo poleg oznake x_i tudi še utež
- povezave niso (urejeni) pari, ampak p-terice – hipergraf
- ...

Primeri

Z grafi lahko ponazorimo (modeliramo) različne situacije:

- vsako drevo je (usmerjen) graf
- internet je graf: URL naslovi predstavljajo vozlišča in hiper-povezave (HREF) predstavljajo povezave
- omrežje računalnikov je graf, kjer so propustnosti povezav (*bandwidth*) uteži
- cestno omrežje je graf
- osebki so elementi grafa (DNK je njihova oznaka), povezave pa so sorodstveni odnosi med njimi
- končni avtomat
- relacijsko-entiteni model
- načrt izvajanja programa - meniji in obrazci so vozlišča
- ...

Predstavitev

Najprej opazimo, da lahko vozlišča grafa vedno izpeljemo iz množice povezav. Tako imamo dve osnovni predstavitvi grafa:

- s seznamom sosedov: za vsako vozlišče podamo vse sosede tega vozlišča: $x_i : x_j, x_k, \ldots$
- z matriko sosednosti

Seznam sosedov

- Za vsako vozlišče x_i podamo vse sosede tega vozlišča: x_i : $x_j, x_k, \ldots$
- Prostorska zahtevnost je $\Theta(m + n)$, kjer je pri neusmerjenih grafih vsaka povezava naštetea dvakrat.

```
public class vertex {
  public Dictionary neighbours;
  ...
}
```

Matrika sosednosti

- Imamo $n \times n$ matriko C, kjer vsaka vrstica in vsak stolpec predstavljata vozlišče.
- Vrednost elementa $c_{i,j}$ je 1 če in samo če obstaja povezava (x_i, x_j) – različica, je utež povezave.

$$
\begin{array}{ccccc}
 & v_1 & v_2 & \dots & v_n \\
v_1 & w_{1,1} & w_{1,2} & \dots & w_{1,n} \\
v_2 & w_{2,1} & w_{2,2} & \dots & w_{2,n} \\
\vdots & & & & \\
v_n & w_{n,1} & w_{n,2} & \dots & w_{n,1}
\end{array}
$$

- Pri neusmerjenih grafih, je matrika C simetrična.
- Prostorska zahtevnost $O(n^2)$.

Sprehodi / pregledi

- Sistematično pregledamo vsa vozlišča iz danega začetnega vozlišča – zgradimo *vpeto drevo*, ki usteza pregledu.
- Imamo dva osnovna načina:
  1. pregled v širino: ko pridemo v vozlišče, najprej pregledamo vse njegove sosede – nekako metoda deli in vladaj;
  2. pregled v globino: ko pridemo v vozlišče, pogledamo prvega soseda in postopek nadaljujemo, za vsa vozlišča, dokler gre – nekako požrešna metoda

Pregled v širino

- pot v drevesu pregleda v širino od začetnega vozlišča s do vozlišča v ustreza dolžini najkrajše poti od s so v grafu G.
- Vsako vozlišče na razdalji k od s $\gg$obiščemo$\ll$ pred vsemi vozlišči na razdalji $k+1$ od s.

```
public void BFS(Graph G, Vertex s, Function f) {
  Queue fifo;
  Vertex v;
  fifo.insert(s);
  while !fifo.empty() {
    v= fifo.extract();
    v.visited= true;
    forall (u, v.neighbours())
      if !u.visited fifo.insert(u);
    f.operation(v);
  }
}
```

Časovna zahtevnost: $O(n+m)$ – vsako vozlišče se obišče največ dvakrat in vsaka povezava se obišče največ enkrat.

Pregled v globino

1. Vsakič obiščemo vozlišče, ki je še neobiskano in je sosed zadnjega obiskanega vozlišča, ki ima še neobiskanega soseda.
2. Pomemben je vrstni red povezav (sosedov) v vozlišču.

```
public void DFS(Graph G, Vertex s, Function f) {
  Stack filo;
  Vertex v;
  filo.push(s);
  while !filo.empty() {
    v= filo.pop();
    v.visited= true;
    forall (u, v.neighbours())
      if !u.visited filo.push(u);
    f.operation(v);
  }
}
```

Časovna zahtevnost: $O(n + m)$ – vsako vozlišče se obišče največ dvakrat in vsaka povezava se obišče največ enkrat.

Topološko urejanje

- Za graf rečemo, da je *acikličen*, če v njem za nobeno vozlišče v ne obstaja pot, ki nas bi pripeljala nazaj v v.
- Edini aciklični neusmerjen graf je drevo (gozd).
- Topološko urejanje acikličnega usmerjenega grafa G je linearna ureditev njegovih vozlišč tako, da velja: če $(u, v) \in E(G)$, potem $u \leq v$ – nazorneje povedano: če narišemo vozlišča na premici potem so *vse* povezave usmerjene na desno.

```
public Vertex[] TopologicalSort(Graph G, Vertex s) {
  Stack filo;
  Vertex v;
  int indx= 0;
  Vertex[] result= new Vertex[G.V.size()];
  filo.push(s);
  while !filo.empty() {
    v= filo.pop();
    v.visited= true;
    forall (u, v.neighbours())
      if !u.visited filo.push(u);
    result[indx]= v; indx++;
  }
  return result;
}
```

Časovna zahtevnost je enaka časovni zahtevnosti DFS, torej $O(n+m)$.

Nekaj primerov obdelav grafa

- V grafu poiščemo cikle.
- V grafu poiščemo povezane komponente: komponenti grafa sta dva dela grafa, med katerima ni povezave
- V grafu poiščemo močno povezano komponento: močno povezano komponento predstavljajo tista vozlišča $v_1, v_2, \ldots, v_k$, za katera velja, da obstaja za vsak par v v_i, v_j, pot iz v_i do v_j.

Zahtevnost

problem	**zahtevnost**
topološko urejanje	$O(n)$
iskanje ciklov	$O(n)$
povezane komponente	$O(n)$
močno povezane komponente	in koliko bi bilo to

Komentarji:

- Kako je lahko urejanje sedaj $O(n)$?