

Anotace

- Grafové algoritmy.

Definice grafu

Definition

Grafem nazveme uspořádanou dvojici $G = (V, E)$, kde V nazveme množinou vrcholů a $E \subseteq \binom{V}{2}$ nazveme množinou hran.

Definition

Uspořádanou dvojici $G = (V, E)$ nazveme orientovaným grafem s množinou vrcholů V a množinou hran E , jestliže $E \subseteq V \times V$.

- O grafech jste se učili na Diskrétní matematice, měli jste zřejmě i vybrané algoritmy. A algoritmy jsou typicky určeny k naprogramování.
- Definice je pěkná, ale při programování nám nepomůže. Podbízí se otázka:
- Jak graf reprezentovat při programování?

Reprezentace

- Z diskrétní matematiky znáte:
- Matici sousednosti A_G
 - je čtvercová matice obsahující nuly a jedničky, jejíž řádky a sloupce jsou indexovány vrcholy. Jednička odpovídá tomu, že mezi dotýčnými vrcholy hrana vede, nula znamená, že hrana nevede.
- Matici incidence B_G – řádky jsou indexovány vrcholy, sloupce hranami, jednička na pozici $B_G[i, j]$ říká, že hrana j přiléhá k vrcholu i .
- Výhody a nevýhody?
- Jak mezi těmito reprezentacemi převádět?

Převod A_G na B_G a zpět

```
snuluj( $B_G$ );
index_hrany:=1;
for i:=1 to n do begin
  for j:=i+1 to n do begin
    if( $A_G[i,j]=1$ ) then
      begin
         $B_G[i, index\_hrany]:=1$ ;
         $B_G[j, index\_hrany]:=1$ ;
        inc(index_hrany);
      end;
  end;
end;
```

B_G na A_G

Bud'to podobnou analýzou matice incidence, nebo:

$$A_G := B_G \times B_G^T;$$

for $i:=1$ to n do

$$A_G[i, i] := 0;$$

Důkaz.

Snadné cvičení z Kombinatoriky a grafů I.

Další reprezentace grafů

- Seznam vrcholů a k nim přilehlých hran:
- Tedy udržujeme seznam vrcholů a ke každému vrcholu vedeme seznam hran, které z něj vedou.

Funkce (proměnné) potřebné (resp. postačující) pro práci s grafem:

- `najdi_sousedy(v)`,
- `vrcholy`,
- `hrany` nebo `hrana(u,v)`, to ale umíme zjistit pomocí `vrcholy` a `najdi_sousedy`,
- případně další (`vaha_vrcholu(v)`, `vaha_hrany(e)`...).
- Výhody a nevýhody?
- Je-li graf orientovaný, musíme reprezentaci modifikovat.

Sled, tah, cesta, kružnice

Definition

- Sledem délky k nazveme posloupnost hran tvaru $\{v_0, v_1\}, \{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{k-1}, v_k\}$.
- Tahem nazveme sled, v němž se každá hrana vyskytne nejvýše jednou.
- Cestou nazveme tah (nebo sled), ve kterém se každý vrchol vyskytuje nejvýše jednou (přesněji kde se každý vrchol vyskytuje právě ve dvou po sobě jdoucích hranách).
- Tah nazveme kružnicí, pokud začíná a končí v tomtéž vrcholu a pokud se v něm každý vrchol objeví právě jednou.

Souvislost, strom

Definition

- Graf je souvislý, pokud se lze z každého jeho vrcholu dostat do každého jiného (vrcholu).
 - Graf je strom, pokud je souvislý a neobsahuje kružnice.
-
- Definice jsou pěkné, ale pomohou nám při programování?
 - Jak ověříte, zda je graf souvislý?
 - Použijeme vhodné tvrzení.
 - Jak zjistíte, zda je graf strom?
 - Podobně.

Souvislost grafu

Graf je souvislý právě když se lze z jednoho jeho vrcholu dostat do všech ostatních

```
for i in vrcholy do
    nenavstiv(i); {jeste jsme nic nenavstivili}
i:=startovni_vrchol;
fronta:={i};{na dosažitelné vrcholy}
while nonempty(fronta) do begin
    i:=prvni_prvek(fronta);
    navstiv(i);
    fronta:=fronta+nenavstivene_sousedy(i);
end;
souvisly:=true;
for i in vrcholy do begin
    if nenavstiveny(i) then
        souvisly:=false;
```

Analýza algoritmu

- for-cyklus proběhne nejvýš n -krát.
- while-cyklus proběhne pro každý vrchol nejvýše jednou a podívá se na sousedy současného vrcholu.
- Složitost bude záviset na reprezentaci (jak rychle umíme najít sousedy vrcholu).
- Složitost bude $\Omega(m)$ (na každou hranu musíme kouknout).
- Pokud máme seznam hran u vrcholu, bude složitost $O(m)$,
- pokud máme matici sousednosti, bude složitost $O(n^2)$,
- máme-li matici incidence, složitost může být i $\Theta(mn^2)$.
- Při vhodné reprezentaci je tedy složitost $\Theta(m + n)$.

Poznámky

- Použijeme-li frontu, jedná se o algoritmus vlny (prohledávání do šířky) z jednoho vrcholu.
- Můžeme použít i zásobník, v tom případě se jedná o prohledávání do hloubky
- Výhody a nevýhody:
- Při hledání do hloubky můžeme použít rekurzi a nemusíme si aktuálně pamatovat sousedy prohledávaného vrcholu.
- Hledání do šířky navštíví vrchol po nejkratší cestě.

Vyšetření komponent souvislosti

- Naivní algoritmus: Najít komponentu a odstranit (komplikované a nepraktické).
- Lepší algoritmus: Začneme s prázdným grafem a postupně přidáváme hrany.
- Na počátku obarvíme vrcholy každý jinou barvou (reprezentující prozatímní kandidáty na komponenty souvislosti).
- Procházíme hrany a pro každou se podíváme, zda vede uvnitř komponenty. Pokud ne, sluč komponenty (jednu přebarví na barvu druhé).

Hledání kružnice

Graf má kružnici, pokud se při prohledávání grafu vrátíme do už navštíveného vrcholu.

```
kruznice:=false; {zatim zadna}
for i in vrcholy do nenavstiv(i);
for i in vrcholy do
  if nenavstiveny(i) then {nova komponenta}
  begin fronta:={i};
    while(nonempty(fronta)) do
      begin prvni prvek vyrad z fronty a prirad do i;
        if(navstiveny(i)) then
          kruznice:=true;
        else for j in sousedy(i) do
          begin fronta:=fronta+{j};
            smaz_hranu({i,j});
          end;
        end;
      end;
    end;
  end;
end; end;
```

Strom

- Budeme testovat, zda je graf souvislý a bez kružnic, tedy použijeme oba předešlé algoritmy.
- Anebo budeme testovat, zda je bez kružnic a má jen jednu komponentu.
- Anebo otestujeme souvislost (či kružnice) a správný počet hran.

Nejkratší cesta

Hledáme-li v grafu nejkratší cestu z vrcholu do vrcholu, záleží na reprezentaci:

- Prolezeme graf do šířky (máme-li seznam vrcholů a hran),
- mocníme matici sousednosti, pokud máme maticovou reprezentaci.

Theorem

V matici A_G^k hodnota na pozici i, j určuje počet sledů délky k z vrcholu i do vrcholu j .

Corollary

V matici $(A_G + I)^k$ určuje hodnota na pozici i, j počet sledů délky nejvýše k z i do j .

Dijkstrův algoritmus

Hledá nejkratší cestu z daného vrcholu (do všech ostatních)

Vstup: Graf s nezáporně ohodnocenými hranami.

- Udržujeme "frontu" vrcholů seřazenou podle dosud nejkratší cesty do nich.
- Na začátku inicializujeme vzdálenosti do všech vrcholů kromě startovního nekonečnem (tedy dost vysokou hodnotou) a vzdálenost do startu nulou.
- Startovní vrchol přidáme do "fronty" dosažitelných vrcholů.
- Vrchol, do kterého se dostaneme nejkratší cestou z fronty odstraníme a pokusíme se cestu jdoucí z něj rozšířit do jeho sousedů.
- Toto opakuj, dokud je "fronta" neprázdná.

Rozšíření cesty

Rozšíření cesty vypadá tak, že pro vrchol v ve vzdálenosti $d(v)$ zkusíme pro každou hranu $\{v, w\}$, zda

$$d(w) > d(v) + \text{delka}(\{v, w\}).$$

Pokud ano, $d(w) := d(v) + \text{delka}(\{v, w\})$ a oprav pozici vrcholu w ve "frontě".

Analýza

- Algoritmus je popsán a dokázán v mnoha knihách (a skriptech), kupř. Kapitoly z diskrétní matematiky nebo Algebraické algoritmy (Kučera, Nešetřil)...
- Konečnost: V každé iteraci odstraníme z "fronty" jeden vrchol, který se v ní už neobjeví (protože mezi vrcholy ve frontě měl nejmenší vzdálenost od startu a vzdálenosti jsou nezáporné).
- Parciální správnosti pomůže invariant: V každém kroku evidujeme nejkratší cesty ze startu používající pouze vrcholy již odstraněné z "fronty".
- Z invariantu plyne korektnost.
- Jde jen o modifikovaný algoritmus vlny, tedy hledání do šířky!
- Složitost významně závisí na reprezentaci grafu a na reprezentaci "fronty"!