Anotace

- Informace o testu,
- fronta a zásobník v poli,
- předání pole ve struktuře,
- dlouhá čísla a polynomy,
- grafy a jejich reprezentace,
- grafové algoritmy.

Fronta a zásobník

- Definice: Fronta je datová struktura osazená operacemi `insert` a `delete`. Funkce `insert` přidává na konec fronty, `delete` odebírá ze začátku fronty.
- Zásobník je datová struktura osazená operacemi `push` a `pop`. Funkce `push` přidává na konec zásobníku, `pop` odebírá z konce zásobníku.
- Obojí lze reprezentovat v poli (pak vzniká zásobník resp. fronta pevné velikosti a tedy pozor na přetečení).

Zásobník

```
type zasobnik=record
          pole:array[1..MAX] of integer;
          vrchol:integer;
     end;
var zas:zasobnik;
procedure push (a:integer;var zas:zasobnik);
begin
   if(zas.vrchol<MAX) then
   begin
     inc(zas.vrchol);
     zas.pole[zas.vrchol]:=a;
   end;
end;
```

Zásobník

funkce pop

```
function pop(var zas:zasobnik):integer;
begin
   if(zas.vrchol<=0) then
     pop:=-1;
   else begin
     pop:=zas.pole[zas.vrchol];
     dec(zas.vrchol);
   end;
end;
```

Fronta

```
type fronta=record
          pole:array[1..MAX] of integer;
          zacatek,konec:integer;
     end;
var fr:fronta;
procedure insert(i:integer;fr:fronta);
begin
     if nepretece(fr) then
     begin
          inc(fr.konec);
          if(fr.konec>MAX) then fr.konec:=1;
          fr.pole[fr.konec]:=i;
     end;
end;
```

Fronta

funkce delete

```
function delete(fr:fronta);
begin
     if(nepodtece(fr)) then
     begin
          inc(fr.zacatek);
          if(fr.zacatek>MAX) then fr.zacatek:=1;
          delete:=fr.zacatek;
     end;
end;
```

Dlouhá čísla, polynomy

- Dlouhá čísla jsou čísla, která se nevejdou do žádného číselného typu.
- `type dlouhecislo=array [1..MAX] of integer;`
- Můžeme nasadit různé finty (například neudržovat jen jednu číslici, ale víc, například do 10 000) $\Rightarrow$ výhody a nevýhody.
- Polynomy můžeme reprezentovat jako pole koeficientů: `type polynom=array[0..MAX] of integer;`
- Zavřeme-li dotyčné pole do struktury, můžeme si u něj pamatovat délku (resp. stupeň).

Příklad

Sečti polynomy

```
type poly=record
          p:polynom;
          stupen:integer;
     end;
procedure secti(var a:poly;b:poly);
var i:integer;
begin i:=0;
      while(i<=a.stupen) or (i<=b.stupen) do
      begin a.p[i]:=a.p[i]+b.p[i];
            inc(i);
      end;
      a.stupen:=i-1; end;
```

Příklad

Sečti dlouhá čísla

```
type cislo=record
          c:dlouhecislo;
          delka:integer;
     end;
procedure secti(var a:cislo;b:cislo);
var i:integer;
begin i:=1;
      while(i<=a.delka) or (i<=b.delka) do
      begin a.c[i]:=a.c[i]+b.c[i];
            inc(i);
      end;
      a.delka:=i-1;
      for j:=1 to i-1 do
      begin a.c[j+1]:=a.c[j+1]+(a.c[j] div 10);
```

Příklad

Sečti dlouhá čísla

```
procedure secti(var a:cislo;b:cislo);
var i:integer;
begin i:=1;
      while(i<=a.delka) or (i<=b.delka) do
      begin a.c[i]:=a.c[i]+b.c[i];
            inc(i);
      end;
      a.delka:=i-1;
      for j:=1 to i-1 do
      begin a.c[j+1]:=a.c[j+1]+(a.c[j] div 10);
            a.c[j]:=a.c[j] mod 10;
      end;
      if a.c[a.delka+1]<>0 then inc(a.delka);
end;
```

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 dotyčnými 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;
```

B_G na A_G

Buďto 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.
- Jelikož zatím neumíte spojové seznamy, bylo by potřeba nejspíše použít pole.

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 $\{v_0, v_1\}, \{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}$.
- 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.

Faktorová množina

odbočka

- Máme částečně zadanou ekvivalenci (tedy reflexivní, tranzitivní a symetrickou relaci) v podobě grafu (vlastnost být v relaci je určena hranou).
- Problém určení faktorové množiny se vlastně ptá na počet (a strukturu) komponent souvislosti takového grafu.
- Algoritmus: Komponentám (stejně jako vrcholům) přidělíme čísla a pro začátek každému vrcholu přiřadíme jako komponentu jeho číslo (každý vrchol do jiné komponenty).
- Postupně procházíme hrany a podíváme se na čísla komponent koncových vrcholů. Pokud se neshodují, komponenty sloučíme (tedy všechny výskyty jednoho čísla změníme na číslo druhé).

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
      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.

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;
```

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.