

# Anotace

- Aritmetické výrazy, notace a převody mezi nimi,
- grafy a jejich reprezentace,
- grafové algoritmy.

# Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?

# Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4) / 2$  – tzv. infixní notace (operátor je mezi operandy),

# Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4) / 2 -$  tzv. infixní notace (operátor je mezi operandy),
- $/ * + 10 5 - 15 4 2 -$  tzv. prefixní notace (operátor je před operandy),

# Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4) / 2 -$  tzv. infixní notace (operátor je mezi operandy),
- $/ * + 10 5 - 15 4 2 -$  tzv. prefixní notace (operátor je před operandy),
- $/ (* (+ 10 5) (- 15 4)) 2$  (uzávorkování pro větší názornost),

# Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4) / 2 -$  tzv. infixní notace (operátor je mezi operandy),
- $/ * + 10 5 - 15 4 2 -$  tzv. prefixní notace (operátor je před operandy),
- $/ (* (+ 10 5) (- 15 4)) 2$  (uzávorkování pro větší názornost),
- $10 5 + 15 4 - * 2 / -$  postfixní notace (operátor je za operandy),

# Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4) / 2 -$  tzv. infixní notace (operátor je mezi operandy),
- $/ * + 10 5 - 15 4 2 -$  tzv. prefixní notace (operátor je před operandy),
- $/ (* (+ 10 5) (- 15 4)) 2$  (uzávorkování pro větší názornost),
- $10 5 + 15 4 - * 2 / -$  postfixní notace (operátor je za operandy),
- $((10 5 +) (15 4 -) *) 2 /$  (uzávorkováno),

# Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4) / 2 -$  tzv. infixní notace (operátor je mezi operandy),
- $/ * + 10 5 - 15 4 2 -$  tzv. prefixní notace (operátor je před operandy),
- $/ (* (+ 10 5) (- 15 4)) 2$  (uzávorkování pro větší názornost),
- $10 5 + 15 4 - * 2 / -$  postfixní notace (operátor je za operandy),
- $((10 5 +) (15 4 -) *) 2 /$  (uzávorkováno),
- stromem: Ve vrcholu buďto operátor (vrchol má dva syny) nebo operand (číslo, takový vrchol je listem).



# Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4) / 2 -$  tzv. infixní notace (operátor je mezi operandy),
- $/ * + 10 5 - 15 4 2 -$  tzv. prefixní notace (operátor je před operandy),
- $/ (* (+ 10 5) (- 15 4)) 2$  (uzávorkování pro větší názornost),
- $10 5 + 15 4 - * 2 / -$  postfixní notace (operátor je za operandy),
- $((10 5 +) (15 4 -) *) 2 /$  (uzávorkováno),
- stromem: Ve vrcholu buďto operátor (vrchol má dva syny) nebo operand (číslo, takový vrchol je listem).
- Pozor, evaluace bude jen naznačena, na slidech bude pseudokód, který je třeba interpretovat s fantazií!

# Aritmetické výrazy a různé notace

- Výhody a nevýhody jednotlivých notací...
- Lze všechny tyto notace (zápisy) vyhodnotit?
- Lze mezi těmito notacemi převádět?
- Ano a dokonce velmi snadno pomocí stromového zápisu.

# Evaluace prefixní notace

```
Rekurzí: function vyhodnot:integer;
begin
    if (na vstupu je číslo) then
        vyhodnot:=hodnota_čísla_na_vstupu
    else
        begin operator:=na_vstupu();
            arg1:=vyhodnot;
            arg2:=vyhodnot;
            vyhodnot:=proved(operator, arg1, arg2);
        end;
    end;
end;
```

## Strom z prefixní notace

```
function pref_strom:strom;
begin
  if(na vstupu je číslo) then
    pref_strom:=list(hodnota_na_vstupu);
  else
    begin pom:=vrchol(operator);
      pom.arg1:=vyhodnot;
      pom.arg2:=vyhodnot;
      vyhodnot:=pom;
    end;
end;
```

funkce list vytvoří list,

funkce vrchol vytvoří vrchol stupně 2,

vrchol stupně 2 obsahuje prvky arg1 a arg2.

# Jak ze stromu vygenerovat všechny notace?

- Rekurzivně:

# Jak ze stromu vygenerovat všechny notace?

- Rekurzivně:
- Prolezeme strom tak, že v jedné fázi vlezeme do levého syna,

# Jak ze stromu vygenerovat všechny notace?

- Rekurzivně:
- Prolezeme strom tak, že v jedné fázi vlezeme do levého syna,
- v jedné fázi do pravého syna a v jedné fázi vypíšeme údaj o operátoru.

# Jak ze stromu vygenerovat všechny notace?

- Rekurzivně:
- Prolezeme strom tak, že v jedné fázi vlezeme do levého syna,
- v jedné fázi do pravého syna a v jedné fázi vypíšeme údaj o operátoru.
- Všechny tři notace získáme správným uspořádáním těchto tří fází.



# Jak ze stromu vygenerovat všechny notace?

- Rekurzivně:
- Prolezeme strom tak, že v jedné fázi vlezeme do levého syna,
- v jedné fázi do pravého syna a v jedné fázi vypíšeme údaj o operátoru.
- Všechny tři notace získáme správným uspořádáním těchto tří fází.
- Do pravého syna půjdeme vždy až po návštěvě levého syna, **lišit se tedy bude jen okamžik výpisu údajů!**

# Generování prefixní notace

```
procedure gen_pref(v:vrchol);  
begin  
    if(list(v)) then  
        vypis(v);  
    else  
begin vypis(v);  
        gen_pref(v.arg1);  
        gen_pref(v.arg2);  
    end;  
end;
```

Funkce vypis vypíše operátor resp. číslo.

funkce list zjistí, zda je současný vrchol list.

# Generování postfixní notace

```
procedure gen_post(v:vrchol);
begin
    if(list(v)) then
        vypis(v);
    else
    begin gen_post(v.arg1);
        gen_post(v.arg2);
        vypis(v);
    end;
end;
```

Funkce vypis vypíše operátor resp. číslo.

funkce list zjistí, zda je současný vrchol list.

# Generování infixní notace

skoro správně

```
procedure gen_post(v:vrchol);
begin
    if(list(v)) then
        vypis(v);
    else
        begin gen_post(v.arg1);
            vypis(v);
            gen_post(v.arg2);
        end;
    end;
end;
```

Funkce vypis vypíše operátor resp. číslo.

funkce list zjistí, zda je současný vrchol list.

# Generování infixní notace

správně leč ošklivě

```
procedure gen_post(v:vrchol);
begin
  if(list(v)) then
    vypis(v);
  else
begin write('(');
      gen_post(v.arg1);
      vypis(v);
      gen_post(v.arg2);
      write(')');
    end;
end;
```

## K vyhodnocení postfixní notace

Opakování:

**Zásobník** je datová struktura osazená operacemi:

- push – přidej na konec zásobníku,
- pop – uber z konce zásobníku,
- tedy kdo později přijde, ten je dříve odejit.

# Evaluace postfixní notace

```
function eval_post:integer;
begin
  while not eof do
    begin if (na_vstupu_cislo) then
           push(cislo);
          if (na_vstupu_operator) then
            begin arg2:=pop;
                  arg1:=pop;
                  push(operator(arg1,arg2));
            end;
          end;
        writeln(pop);{Výsledek je na vrchu zásobníku}
    end;
end;
```

# Strom z postfixní notace

```
function strom_post:vrchol;  
begin  
  while not eof do  
    begin if (na_vstupu_cislo) then  
           push(list(cislo));  
          if (na_vstupu_operator) then  
            begin pom:=vrchol(operator);  
                  pom.arg2:=pop;  
                  pom.arg1:=pop;  
                  push(pom);  
            end;  
          end;  
    end;  
  strom_post:=pop; {Výsledek na vrchu zásobníku}  
end;
```



# Evaluace stromu

je snad jasná, ale přesto:

```
function eval_tree(v:vrchol);
begin
    if(list(v)) then
        eval_tree:=value(v)
    else
begin arg1:=eval_tree(v.arg1);
    arg2:=eval_tree(v.arg2);
    op:=operator(v);
    eval_tree:=op(arg1,arg2);
    end;
end;
```

# Evaluace infixní notace

aneb Masakr u katova stromu

- Jedna možnost je najít operátor, který se provede jako poslední,
- výraz podle něj rozdělit, zavolat se na každou část zvlášť a nakonec provést operátor.
- Výhoda: Po rozmyšlení, jak najít poslední operátor snadno implementovatelné.
- Nevýhoda: Neustále traverzujeme výrazem a hledáme operátory.
- Jak třeba lze najít poslední provedený operátor:
  - 1 Najdi poslední operátor sčítání nebo odčítání mimo závorky,
  - 2 pokud nebyl nalezen, hledej poslední operátor násobení mimo závorky,
  - 3 pokud koukáme na samotné číslo, vyhodnoť ho,
  - 4 jinak oloupej závorky,

# Pascalův trojúhelník

- Obsahuje kombinační čísla,

# Pascalův trojúhelník

- Obsahuje kombinační čísla,
- $n$ -tý řádek konkrétně obsahuje hodnoty  $\binom{n}{0}, \binom{n}{1}, \dots, \binom{n}{n}$ ,

# Pascalův trojúhelník

- Obsahuje kombinační čísla,
- $n$ -tý řádek konkrétně obsahuje hodnoty  $\binom{n}{0}, \binom{n}{1}, \dots, \binom{n}{n}$ ,
- při výpočtu využíváme toho, že  $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ ,

# Pascalův trojúhelník

- Obsahuje kombinační čísla,
- $n$ -tý řádek konkrétně obsahuje hodnoty  $\binom{n}{0}, \binom{n}{1}, \dots, \binom{n}{n}$ ,
- při výpočtu využíváme toho, že  $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ ,
- pro jednoduchost chceme spočítat jen číslo  $\binom{n}{k}$ .

## Pascalův trojúhelník rekurzivní řešení

- Získali jsme rekurenci  $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ ,
- z této snadno sestojíme rekurzivní program:

```
function kombin(n,k:integer):longint;  
begin  
    if (k=0) or (k=n) then kombin:=1;  
    else kombin:=kombin(n-1,k-1)+kombin(n-1,k);  
end;  
begin  
    kombin(100,50);  
end.
```

Málem stejný problém, pořád počítáme to samé mockrát.  
Opět přidáme cache na výsledky.

# Pascalův trojúhelník

```
const MAX=100;
var cache:array[0..MAX,0..MAX] of longint;
function kom(n,k:integer):longint;
begin
  if cache[n,k]=0 then
    begin if (k=0) or (k=n) then cache[n,k]:=1
          else cache[n,k]:=kom(n-1,k-1)+kom(n-1,k);
    end;
  kom:=cache[n,k];
end;
var n,k:integer;
begin
  read(n,k);
  writeln(kom(n,k));
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.

# 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  $E \subseteq V \times V$ .

# 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 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:

# 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:

# 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.

# 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  $i$ .



# 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?

# 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 $A_G$

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:

## 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.

## 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.

## 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:



## 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)`,

## 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,

## 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`,

## 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)`...).

## 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?

## 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

# Sled, tah, cesta, kružnice

## Definition

- Sledem délky  $k$  nazveme posloupnost hran tvaru  $\{v_0, v_1\}, \{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{k-1}, v_k\}$ .



# 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.

# 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).

# 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

# Souvislost, strom

## Definition

- Graf je souvislý, pokud se lze z každého jeho vrcholu dostat do každého jiného (vrcholu).

# 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.

# 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?

# 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?



# 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.

# 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?

# 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
    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.

# Analýza algoritmu

- for-cyklus proběhne nejvýš  $n$ -krát.
- while-cyklus proběhne pro každý vrchol nejvýše jednou a podívá se na sousedy současného vrcholu.

# 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).

# 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).



# 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  $O(m)$ ,

# 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  $O(n^2)$ ,

# 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  $\Theta(mn^2)$ .

# 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.

# 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

# 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:

# 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.



# 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;
    end;
  end;
end; end;
```

# Strom

- Budeme testovat, zda je graf souvislý a bez kružnic, tedy použijeme oba předešlé algoritmy.

# 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.

# 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),

## 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  $j$ .*

# 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 seříděnou 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)...

# 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).

# Analýza

- Algoritmus je popsán a dokázán v mnoha knihách (a skriptech), kupř. Kapitoly z diskretní 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".

# 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.

# 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!

# 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"!

# Poznámky

- Pokud jde o graf neohodnocený (délka všech hran je 1), potom se z Dijkstrova algoritmu stane obyčejný algoritmus vlny.



# Poznámky

- Pokud jde o graf neohodnocený (délka všech hran je 1), potom se z Dijkstrova algoritmu stane obyčejný algoritmus vlny.
- Aplikace grafových algoritmů: Zde začíná teoretická informatika. :-)

# Poznámky

- Pokud jde o graf neohodnocený (délka všech hran je 1), potom se z Dijkstrova algoritmu stane obyčejný algoritmus vlny.
- Aplikace grafových algoritmů: Zde začíná teoretická informatika. :-)
- Příklady využití grafových algoritmů:

# Poznámky

- Pokud jde o graf neohodnocený (délka všech hran je 1), potom se z Dijkstrova algoritmu stane obyčejný algoritmus vlny.
- Aplikace grafových algoritmů: Zde začíná teoretická informatika. :-)
- Příklady využití grafových algoritmů:
- Theseus a Minotaurus,

# Poznámky

- Pokud jde o graf neohodnocený (délka všech hran je 1), potom se z Dijkstrova algoritmu stane obyčejný algoritmus vlny.
- Aplikace grafových algoritmů: Zde začíná teoretická informatika. :-)
- Příklady využití grafových algoritmů:
- Theseus a Minotaurus,
- Král (či jiné figurky) na šachovnicích různých tvarů s různě pozakazovanými políčky,

# Poznámky

- Pokud jde o graf neohodnocený (délka všech hran je 1), potom se z Dijkstrova algoritmu stane obyčejný algoritmus vlny.
- Aplikace grafových algoritmů: Zde začíná teoretická informatika. :-)
- Příklady využití grafových algoritmů:
- Theseus a Minotaurus,
- Král (či jiné figurky) na šachovnicích různých tvarů s různě pozakazovanými políčky,
- ...

## Další grafové problémy/algoritmy

- Minimální kostra, aneb jak natáhnout elektrické vedení?
- Eulerovský graf, aneb lze všechny hrany grafu nakreslit jedním tahem (abychom žádnou hranou neprojeli dvakrát)?
- Hamiltonskost, aneb kružnice, která navštíví všechny vrcholy (každý právě jednou),
- Klika, aneb maximální úplný podgraf,
- Barevnost, aneb jaký je nejmenší počet barev takový, abychom mohli obarvit vrcholy grafu tak, že sousední vrcholy nedostanou stejnou barvu?
- Hranová barevnost, aneb jaký je nejmenší počet barev takový, abychom mohli obarvit hrany grafu tak, aby žádná dvojice hran přiléhající ke společnému vrcholu neměla stejnou barvu?
- ...

# Konec

Děkuji za pozornost...