

# 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 dotyčnými 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 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  $i$ .



# 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?

# 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=0;  
for(i=0;i<n;i++)  
    for(j=i+1;j<n;j++)  
        if( $A_G[i,j]=1$ ) then  
        {  
             $B_G[i, index\_hrany]=1$ ;  
             $B_G[j, index\_hrany++] = 1$ ;  
        }
```

$B_G$  na  $A_G$ 

Buďto podobnou analýzou matice incidence, nebo:

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

```
for(i=0;i<n;i++)
```

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

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

```
foreach(i in vrcholy)
    nenavstiv(i); {jeste jsme nic nenavstivili}
i=startovni_vrchol;
fronta={i};{na dosažitelné vrcholy}
while(nonempty(fronta))
{
    i=prvni_prvek(fronta);
    navstiv(i);
    fronta=fronta+nenavstivene_sousedy(i);
}
foreach( i in vrcholy)
{
    if(nenavstiveny(i))
        return 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.

# Vyšetření komponent souvislosti

- Naivní algoritmus: Najít komponentu a odstranit (komplikované a nepraktické).

# Vyšetření komponent souvislosti

- Naivní algoritmus: Najít komponentu a odstranit (komplikované a nepraktické).
- Lepší algoritmus: Začneme s prázdným grafem a postupně přidáváme hrany.

# 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).

# 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.

```
foreach(i in vrcholy) nenavstiv(i);
foreach(i in vrcholy do
    if(nenavstiveny(i))//nova komponenta
    {   fronta={i};
        while(nonempty(fronta))
        {   prvni prvek vyrad z fronty a prirad do i;
            if(navstiveny(i)) then
                return true;
            else foreach(j in sousedy(i))
                {   fronta=fronta+{j};
                    smaz_hranu({i,j});
                }
        }
    }
```

# 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 setříděnou 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 diskretní matematiky nebo Algebraické algoritmy (Kučera, Nešetřil)...

# 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).

# 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".



# 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:
- Thesaurus 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:
- Thesaurus a Minotaurus,
- Král (či jiné figurky) na šachovnicích různých tvarů s různě pozakazovanými políčky,
- ...

# Grafově-optimalizační problémy

- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.

# Grafově-optimalizační problémy

- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.
- Hledání nejkratší cesty, minimální kostra,

# Grafově-optimalizační problémy

- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.
- Hledání nejkratší cesty, minimální kostra,
- topologické uspořádání, faktorová množina (vyšetřování komponent).

# Grafově-optimalizační problémy

- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.
- Hledání nejkratší cesty, minimální kostra,
- topologické uspořádání, faktorová množina (vyšetřování komponent).
- Modifikace problémů: Maximální kostra, nejdelší cesta – čím se liší?

# Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).

# Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).
- Povšimněte si, že se vlastně jedná o diskrétní simulaci (rozlijeme vodu na graf a čekáme, kdy voda doteče do jednotlivých vrcholů).

# Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).
- Povšimněte si, že se vlastně jedná o diskrétní simulaci (rozlijeme vodu na graf a čekáme, kdy voda doteče do jednotlivých vrcholů).
- **Bellman-Fordův algoritmus:**  $n - 1$ -krát zopakujeme: Z každého vrcholu zkus "natáhnout" cestu po všech hranách z něj vycházejících (tedy zlepší případnou nalezenou cestu).



# Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).
- Povšimněte si, že se vlastně jedná o diskrétní simulaci (rozlijeme vodu na graf a čekáme, kdy voda doteče do jednotlivých vrcholů).
- **Bellman-Fordův algoritmus:**  $n - 1$ -krát zopakujeme: Z každého vrcholu zkus "natáhnout" cestu po všech hranách z něj vycházejících (tedy zlepši případnou nalezenou cestu).
- Algoritmus funguje i při záporném ohodnocení hran, nesmí ale být přítomna záporná kružnice (kružnice záporné délky).

# Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!

# Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici  $(a, u, v)$ , kde  $u, v$  jsou vrcholy a  $a$  přirozené číslo počítáme nejkratší cestu z  $u$  do  $v$  o nejvýše  $a$  hranách.

# Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici  $(a, u, v)$ , kde  $u, v$  jsou vrcholy a  $a$  přirozené číslo počítáme nejkratší cestu z  $u$  do  $v$  o nejvýše  $a$  hranách.
- Postupujeme pro rostoucí  $a$  (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.

# Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici  $(a, u, v)$ , kde  $u, v$  jsou vrcholy a  $a$  přirozené číslo počítáme nejkratší cestu z  $u$  do  $v$  o nejvýše  $a$  hranách.
- Postupujeme pro rostoucí  $a$  (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...

# Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici  $(a, u, v)$ , kde  $u, v$  jsou vrcholy a  $a$  přirozené číslo počítáme nejkratší cestu z  $u$  do  $v$  o nejvýše  $a$  hranách.
- Postupujeme pro rostoucí  $a$  (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...
- ... jenže jako obvykle velmi neefektivní.

# Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici  $(a, u, v)$ , kde  $u, v$  jsou vrcholy a  $a$  přirozené číslo počítáme nejkratší cestu z  $u$  do  $v$  o nejvýše  $a$  hranách.
- Postupujeme pro rostoucí  $a$  (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...
- ... jenže jako obvykle velmi neefektivní.
- Tedy ji vybavíme cachí v podobě třírozměrného pole...

# Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici  $(a, u, v)$ , kde  $u, v$  jsou vrcholy a  $a$  přirozené číslo počítáme nejkratší cestu z  $u$  do  $v$  o nejvýše  $a$  hranách.
- Postupujeme pro rostoucí  $a$  (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...
- ... jenže jako obvykle velmi neefektivní.
- Tedy ji vybavíme cachí v podobě třírozměrného pole...
- ... a zjistíme, že rekurzi vůbec nepotřebujeme, že stačí čtyři cykly v sobě...



# Floyd – Warshallův algoritmus pseudokód

```
for(a=0;a<n=1;a++)  
  foreach(u in vrcholy)  
    foreach(v in vrcholy)  
      foreach(w in sousedi(v))  
        cache[a,u,v]=min(cache[a,u,v],cache[a-1,u,w]+  
          length(w,v));
```

# Minimální kostra

- Vstup: Graf s ohodnocenými hranami

# Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.

# Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.

# Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Seříd' hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).

# Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Setříd' hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).
- Borůvkův: Spojování komponent souvislosti: Vyber hranu s nejmenší vahou, která vychází z dané komponenty a přidej.

# Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Setříd' hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).
- Borůvkův: Spojování komponent souvislosti: Vyber hranu s nejmenší vahou, která vychází z dané komponenty a přidej.
- Jarníkův (Primův): Pěstování stromu: K dosud postavenému stromu přidej hranu z něj vycházející, která má nejnižší váhu.

# Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Setříd' hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).
- Borůvkův: Spojování komponent souvislosti: Vyber hranu s nejmenší vahou, která vychází z dané komponenty a přidej.
- Jarníkův (Primův): Pěstování stromu: K dosud postavenému stromu přidej hranu z něj vycházející, která má nejnižší váhu.
- Všechny algoritmy počítají to samé. Důkazy korektnosti budou příště.



# Modifikace

- Hledání nejtěžší kostry...

# Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z  $v_i$  na  $-v_i$ .

# Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z  $v_i$  na  $-v_i$ .
- Hledání nejdelší cesty...

# Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z  $v_i$  na  $-v_i$ .
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).

# Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z  $v_i$  na  $-v_i$ .
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).
- Proč? Protože víme, kolik hran má kostra, ale nevíme, kolik hran má cesta.

# Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z  $v_i$  na  $-v_i$ .
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).
- Proč? Protože víme, kolik hran má kostra, ale nevíme, kolik hran má cesta.
- Tudíž i nalezení nejkratší cesty v (potenciálně záporně) ohodnoceném grafu je těžké (NP-těžký problém):

# Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z  $v_i$  na  $-v_i$ .
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).
- Proč? Protože víme, kolik hran má kostra, ale nevíme, kolik hran má cesta.
- Tudíž i nalezení nejkratší cesty v (potenciálně záporně) ohodnoceném grafu je těžké (NP-těžký problém):
- Sedí za ním schovaná Hamiltonskost, tedy hledání cesty délky  $n - 1$ :