Grafové algoritmy

Programovací techniky

Grafy – Úvod - Terminologie

- Graf je datová struktura, skládá se z množiny vrcholů “V” a množiny hran mezi vrcholy “E”
- Počet vrcholů a hran musí být konečný a nesmí být nulový u vrcholů ani u hran
- Grafy
  - Orientované – hrana (u,v) označena šipkou u->v
  - Neorientované – pokud ex. (u,v), existuje také (v,u)
- Souvislost – graf je souvislý, jestliže pro všechny v(i) z V existuje cesta do libovolného v(j) z V, nesouvislý graf je rozdělen na komponenty
- Strom – graf, ve kterém pro každé 2 vrcholy existuje právě jedna cesta

Grafy – Úvod - Terminologie

FIGURE 12.4
Example of a Weighted Graph

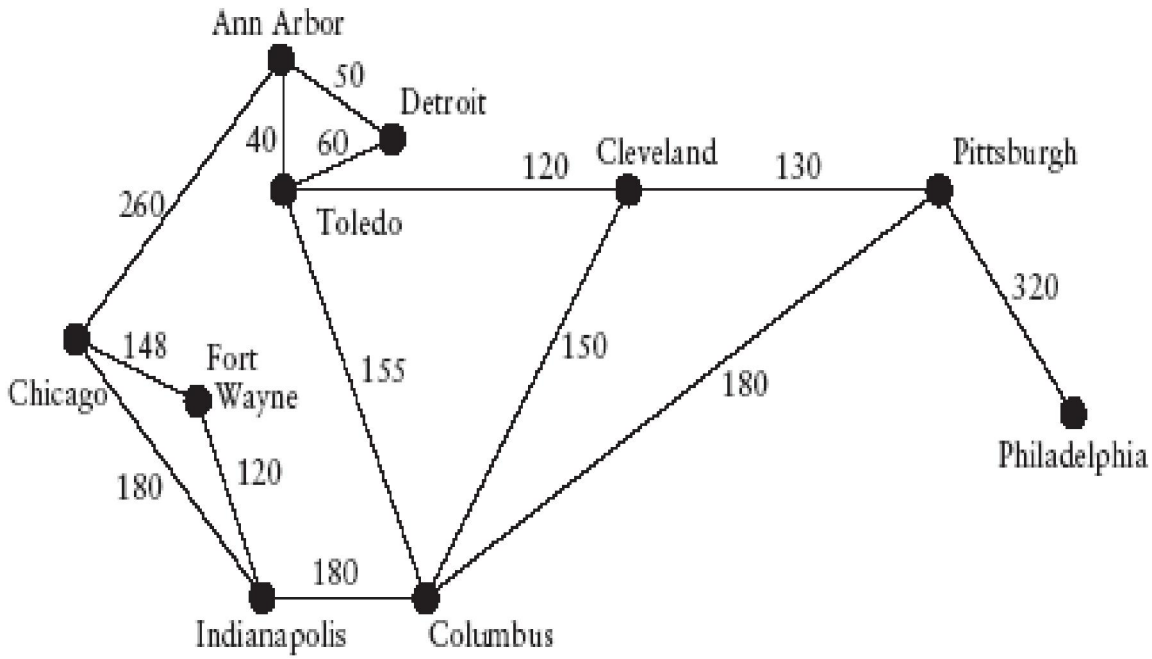
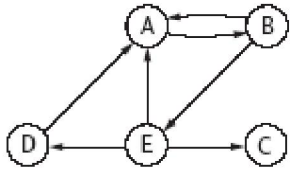


FIGURE 12.3
Example of a Directed Graph



Grafy – Úvod – Reprezentace grafu

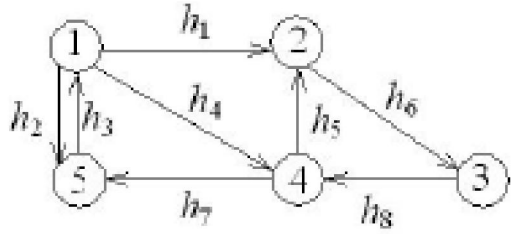
Incidenční matice:

Incidenční matice (Incidence matrix)
– orientovaný graf $G = (V,H,\varepsilon)$

$$a_{ik} = \begin{cases} 1, & \text{jestliže } \exists v \in V : \varepsilon(h_k) = (u_i, v) \\ -1, & \text{jestliže } \exists v \in V : \varepsilon(h_k) = (v, u_i) \\ 0, & \text{jinak} \end{cases}$$

- velikost incidenční matice $|V||H|$
- časová složitost ověření $(i,j) \in H$ je $O(|H|)$

Příklad 3.1

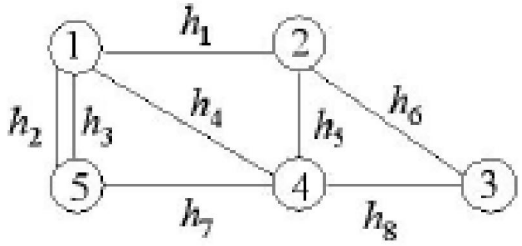


$$\mathbf{A}^{\mathrm{I}} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccccc} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 \end{array} \\ \begin{pmatrix} 1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 & 0 & 1 & -1 \\ 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \end{pmatrix} \end{array}$$

Incidenční matice
– neorientovaný graf $G = (V,H,\varepsilon)$

$$a_{ik} = \begin{cases} 1, & \text{jestliže } h_k \text{ je incidentní s } u_i \\ 0, & \text{jinak} \end{cases}$$

Příklad 3.2



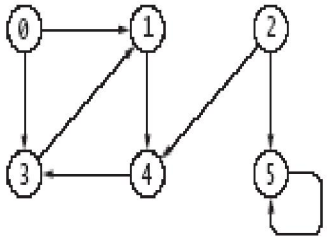
$$\mathbf{A}^{\mathrm{I}} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccccc} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Grafy – Úvod – Reprezentace grafu

Matice sousednosti

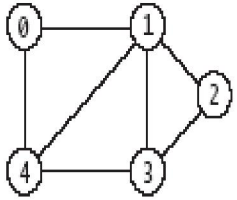
- Dvou dimenzionální pole, hodnoty A(i,j) jsou ohodnocením hrany mezi vrcholy i,j
- Pro neohodnocený graf jsou prvky matice typu Boolean
- Neorientované grafy mají A symetrickou podle diagonály

FIGURE 12.12
A Directed Graph and the Corresponding Adjacency Matrix



Row \ Column	[0]	[1]	[2]	[3]	[4]	[5]
[0]		1.0		1.0		
[1]					1.0	
[2]					1.0	1.0
[3]		1.0				
[4]				1.0		
[5]						1.0

FIGURE 12.13
Undirected Graph and Adjacency Matrix Representation

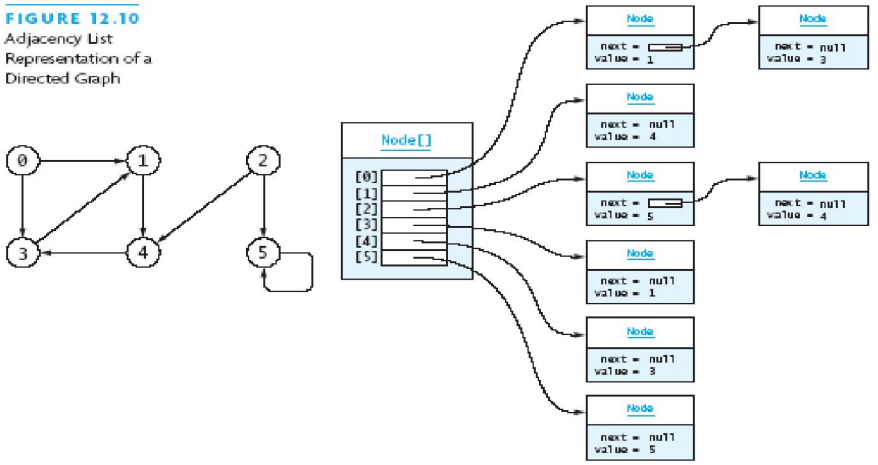


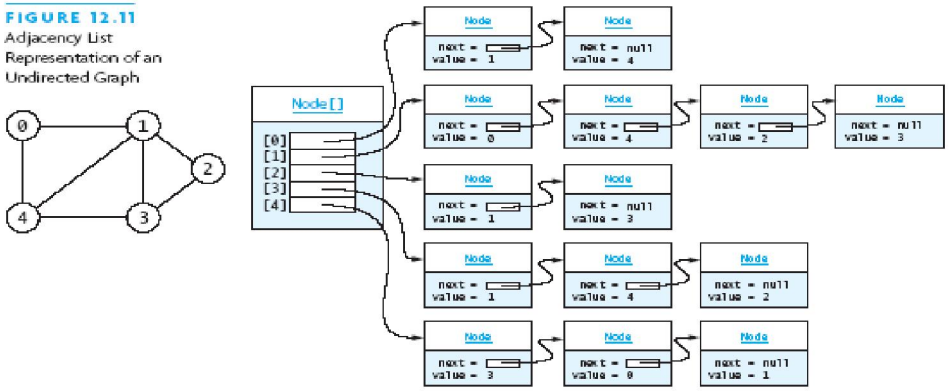
Row \ Column	[0]	[1]	[2]	[3]	[4]
[0]		1.0			1.0
[1]	1.0		1.0	1.0	1.0
[2]		1.0		1.0	
[3]		1.0	1.0		1.0
[4]	1.0	1.0		1.0	

Grafy – Úvod – Reprezentace grafu

Seznamem sousednosti

- Využívá pole seznamů
- Jeden seznam (sousedů) pro každý vrchol
- Vhodné, když chceme měnitelný počet hran





Grafy – Úvod – Reprezentace grafu

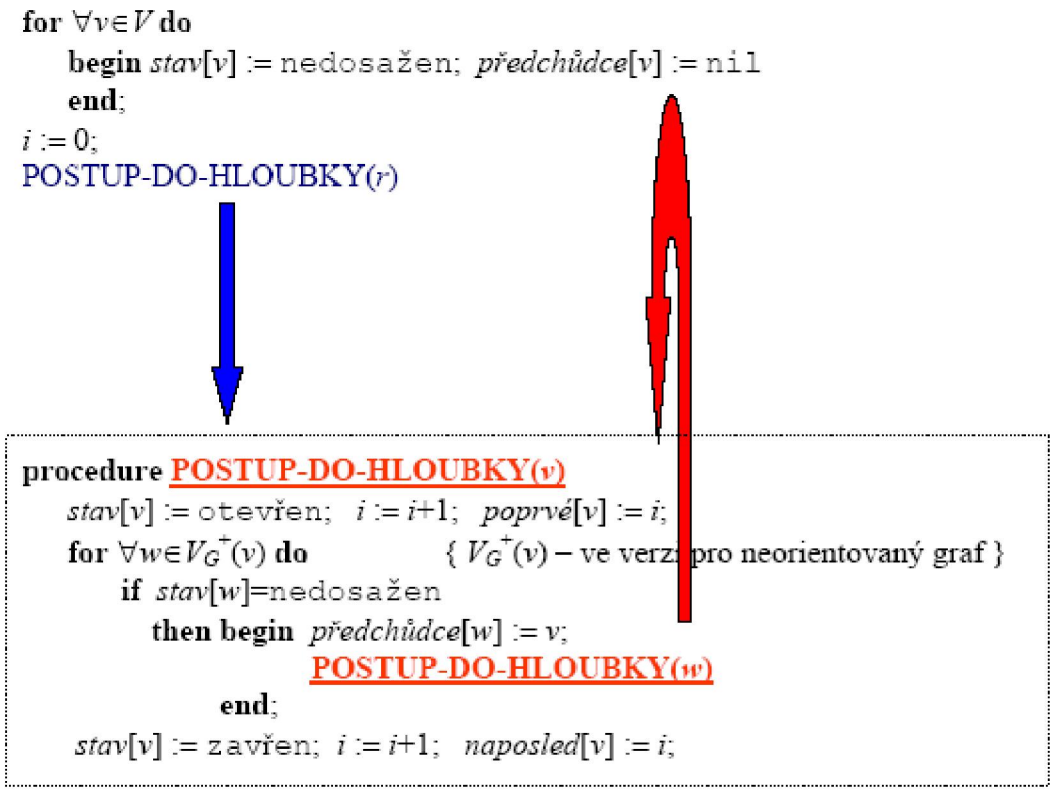
- Plexová struktura
  - Vypadá jako reprezentace seznamem sousednosti, akorát vrcholy nejsou uloženy ve statickém poli, ale v dyn. struktuře (spojový seznam)
    - => Můžeme měnit počet hran i vrcholů

Hledání nejkratší cesty - v neohodnoceném grafu

Prohledávání do šířky

```
1. [Inicializace]
for ∀V−{r} do
    begin dosažen[v] := false; počet-hran-od-r[v] := ∞; předchůdce[v] := nil
    end;
počet-hran [r] := 0; dosažen[r] :=true; FRONTA := (r);
while 2. [Test ukončení] FRONTA ≠ ( ) do
    begin 3. [Volba vrcholu se začátku fronty]
            v := zač-fronty;
          4. [Postup do šířky]
            for ∀V_G^+(v) do    { V_G(v) – ve verzi pro neorientovaný graf }
                if not (dosažen[w])
                   then begin dosažen[w] :=true; předchůdce[w] := v;
                              počet-hran-od-r[w] :=počet-hran-od-r[v]+1;
                              FRONTA := FRONTA + (w)
                        end;
              FRONTA := FRONTA − (v)
    end;
```

Prohledávání do hloubky



Poznámka:

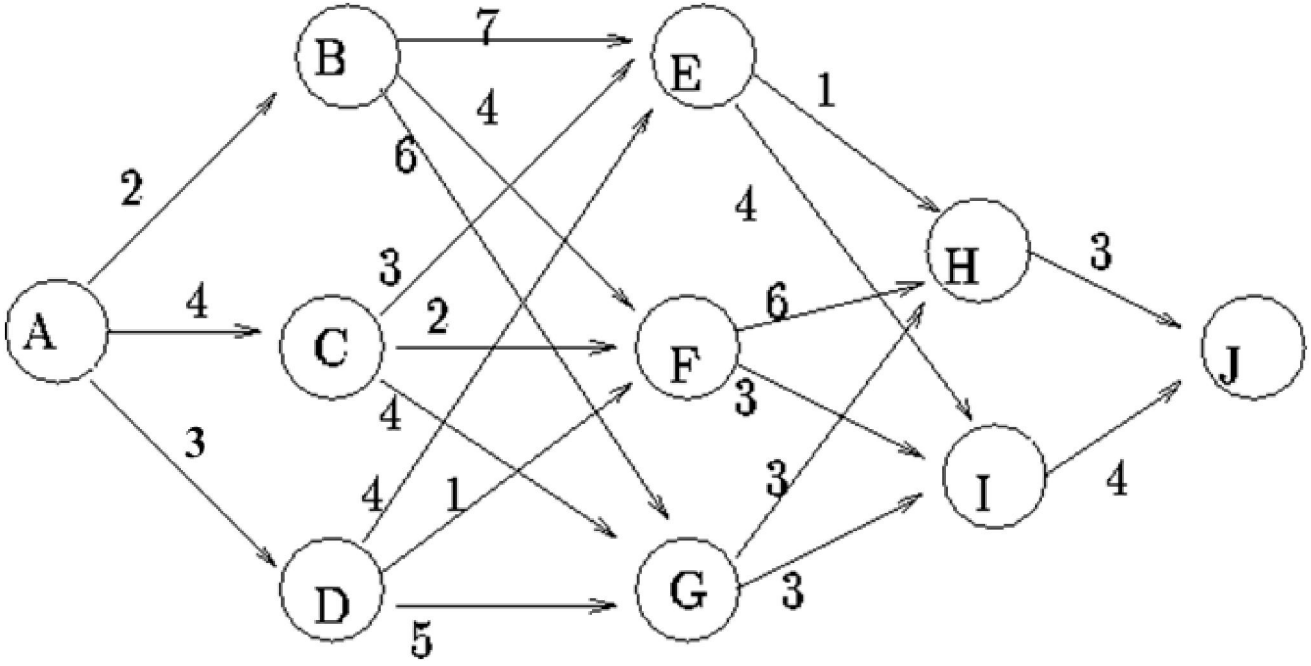
- vrchol v ve stavu `zavřen`
  … všichni následníci (sousedé) vrcholu v už jsou prozkoumáni
- vrcholy ve stavu `otevřen`
  … zásobník rekurzivních volání
- $i \in \langle 1, 2|V| \rangle$
- *poprvé*[v] < *naposled*[v]

Poznámka - Hledání do hloubky (backtracking) - upgrade

- Ořezávání
  - **pokud vidím, že v nějakém uzlu (stavu) prohledáváním jeho následníků URČITĚ nenajdu řešení, neprohledávám dále, ale vrátím se o úroveň výš**
    - Příklad – v grafu projít všechny uzly a a neurazit přitom délku větší než D
- Heuristika
  - **Upřednostním nějakého následníka mezi jinými, podle nějakého kritéria, které mi SNAD urychlí hledání řešení**
    - Příklad(negrafový) – proskákat šachovnici koněm – půjdeme na ta pole, z kterých budeme mít nejméně možností dalšího skoku
- A* algoritmus – zahrnuje prohledávání do šířky i do hloubky s heuristikou i ořezáváním, viz dále.

Hledání nekratší cesty v acyklickém orientovaném grafu

- DAG (directed acyclic graph) shortest path algorithms
- Jaká je nejkratší cesta z A do J v tomto obrázku?



Hledání nekratší cesty v acyklickém orientovaném grafu

- Graf je kvůli své struktuře rozložitelný na etapy

Označíme:

1. etapa: A
2. etapa: B, C, D
3. etapa: E, F, G
4. etapa: H, I
5. etapa: J

Nechť S udává uzel v etapě j a $f_j(S)$ je nejkratší cesta mezi uzly S a J, platí

$$f_j(S) = \min_{\text{nodes } Z \text{ in stage } j+1} \{c_{SZ} + f_{j+1}(Z)\}$$

Kde c_{sz} udává spočtený "oblak hran" *SZ*. Tímto dostáváme rekurentní vztah, který řeší náš problém.

DAG – řešení úlohy

- **Začneme s** $f_5(J)$
- **Etapa 4**
  - Zde se nic nerozhoduje, jen přejdeme do J. Tím, že jdeme do J tedy dostáváme $f_4(H)=3$ a $f_4(I)=4$.
- **Etapa 3 (viz tabulka S_3)**
  - Jak spočítat $f_3(F)$. Z F můžeme jít do H nebo do I.
    - Hrana do H má hodnotu 6, následující je $f_4(H)=3$. Celkem tedy 9.
    - Hrana do I má hodnotu 3, následující je $f_4(I)=4$. Celkem tedy 7.
  - Jakmile se tedy dostaneme do F, je nelepší jít přes I s vydáním = 7.
  - Stejně pro E, G.
- **Pokračujeme takto až do etapy 1**

S_3	$c_{S_3 Z_3} + f_4(Z_3)$		$f_3(S_3)$	Decision
	H	I		Go to
E	4	8	4	H
F	9	7	7	I
G	6	7	6	H

S_2	$c_{S_2 Z_2} + f_3(Z_2)$			$f_2(S_2)$	Decision
	E	F	G		Go to
B	11	11	12	11	E or F
C	7	9	10	7	E
D	8	8	11	8	E or F

S_1	$c_{S_1 Z_1} + f_2(Z_1)$			$f_2(S_1)$	Decision
	B	C	D		Go to
A	13	11	11	11	C or D

DAG shortest path

- Složitost hledání – O(n+m)
- Nejdelší cesta v grafu

$$EC_1 = 0$$

$$EC_w = max_{(v,w)\in E}(EC_v + c_{v,w})$$

Použití:

- např. diagram činností – ukazuje nejkratší čas ukončení projektu
- vzdálenost mezi městy atd.

Dijkstrův algoritmus

- **Hledání nejkratší cesty v ohodnoceném grafu z vrcholu *s* do vrcholu *t***
- **Předpokládáme:**
  - **Graf je souvislý**
  - **Neorientované hrany**
  - **Má nezáporně ohodnocené hrany**
- **Z navštívených vrcholů vytváříme mrak. Zpočátku mrak obsahuje počáteční vrchol S, postupně přidáváme uzly až do nalezení cílového uzlu**
- **V každém kroku – přidáme do mraku vrchol *v* vně mraku s nejmenší vzdáleností d(v) (prioritní fronta)**
- **Opravíme vzdálenosti vrcholů sousedících s v (vede často k relaxaci hran)**

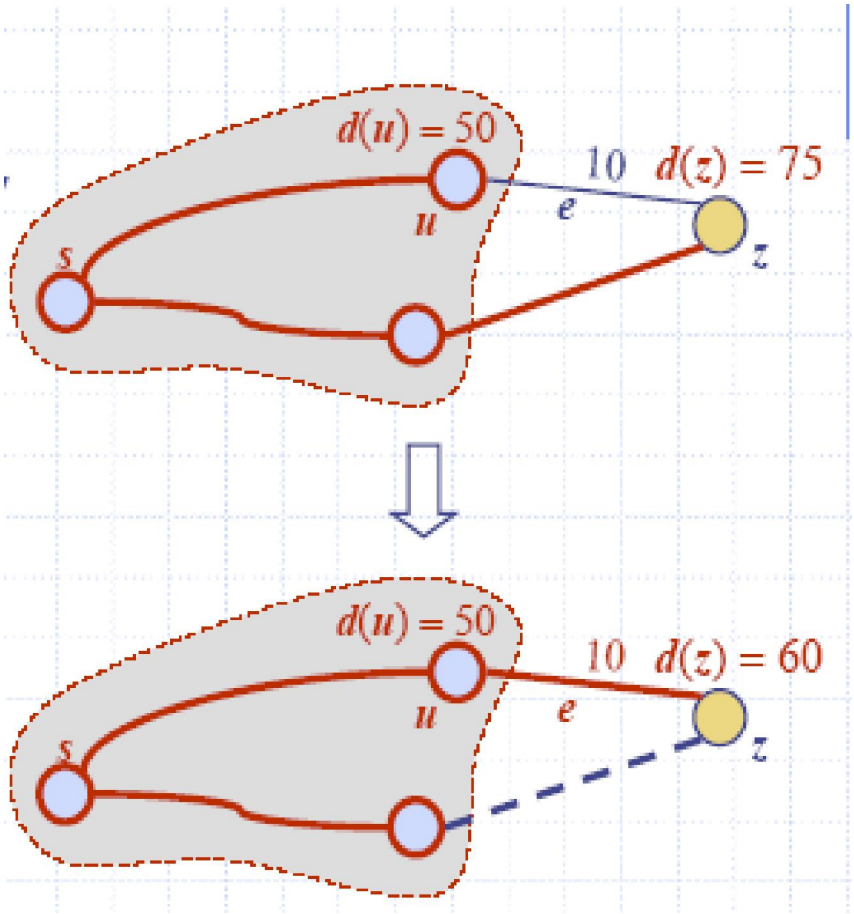
```
for ∀i ∈V do
    begin d[i] := w(s, i);
          označeno[i] := false;
          if w(s,i) < ∞ then před[i] := s
    end;
označeno[s] := true;
konec := false;
celková_délka_cesty := ∞;
repeat imin := 0;
       imin := i: (1) not označeno[i] ∧ (2) d[i] = min {d[u]} < ∞ { hladový postup }
       if imin = 0
         then konec := true { cesta nenalezena, pro všechny neoznačené vrcholy d[i] = ∞ }
         else if imin = t
                then konec := true { cesta nalezena }
                else begin označeno[imin] := true;
                           for ∀i ∈{not označeno[i]} do

                                 if d[imin] + w[imin, i]) < d[i];
                                   then begin d[i] := d[imin] + w[imin, i];
                                              před[i] := imin
                                        end        { relaxace }
                     end
  until konec;
  { if imin = t then celková_délka_cesty := d[imin] }  { Dijkstra s → t }
```

Dijkstrův algoritmus

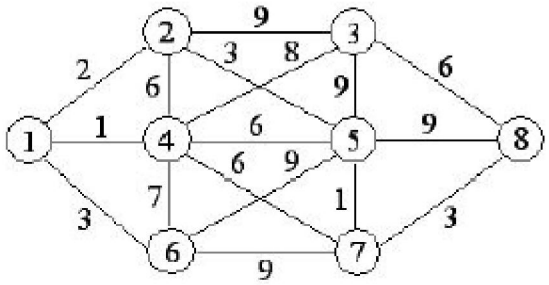
- Relaxace hran
  - Relaxace hrany e upravuje vzdálenost d(z) takto:

$d(z) \leftarrow \min\{d(z), d(u)+\text{weight}(e)\}$



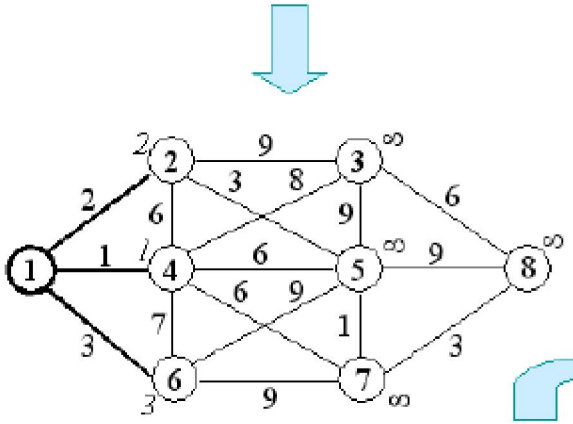
Příklad 7.1

V následujícím grafu najít nejkratší cestu z vrcholu 1 do vrcholu 8.

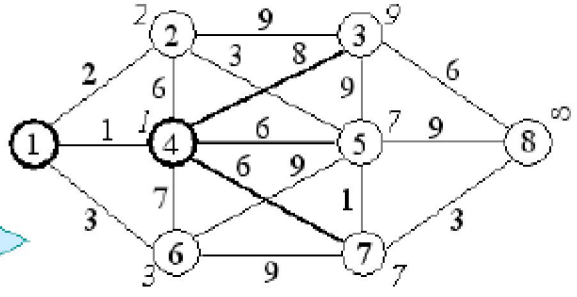


$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{bmatrix} \infty & 2 & \infty & 1 & \infty & 3 & \infty & \infty \\ 2 & \infty & 9 & 6 & 3 & \infty & \infty & \infty \\ \infty & 9 & \infty & 8 & 9 & \infty & \infty & 6 \\ 1 & 6 & 8 & \infty & 6 & 7 & 6 & \infty \\ \infty & 3 & 9 & 6 & \infty & 9 & 1 & 9 \\ 3 & \infty & \infty & 7 & 9 & \infty & 9 & \infty \\ \infty & \infty & \infty & 6 & 1 & 9 & \infty & 3 \\ \infty & \infty & 6 & \infty & 9 & \infty & 3 & \infty \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

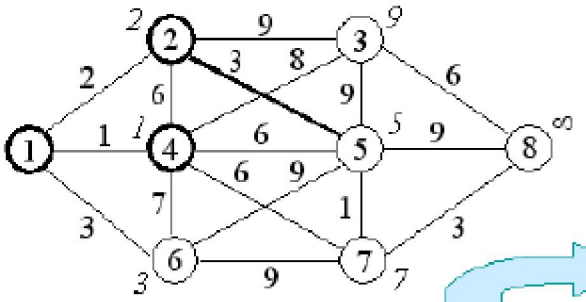
kopírovat vzdálenosti vstupů do vrcholů to 2,4, 6



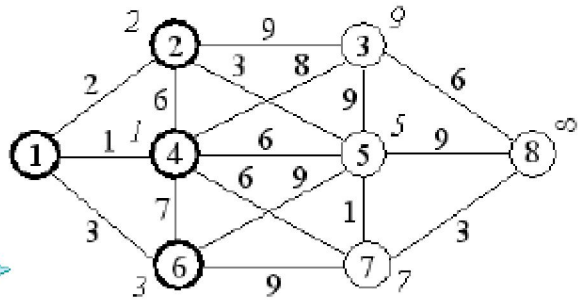
vzít 4 *a označit jej,*
aktualizovat vzdálenost do 3,5,7



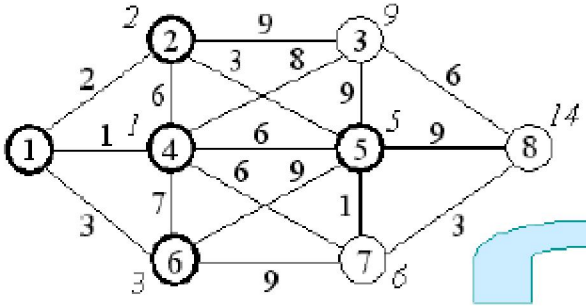
vzít 2 *a označit jej,*
aktualizovat vzdálenost do 5



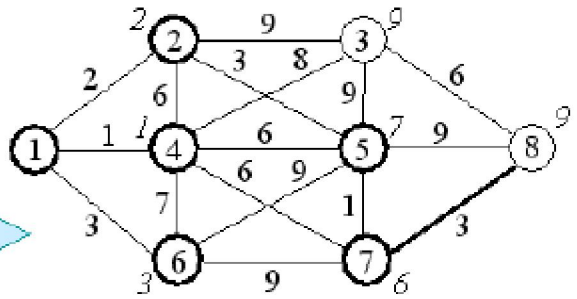
vzít 6 a označit jej,
žádné vzdálenosti se neaktualizují



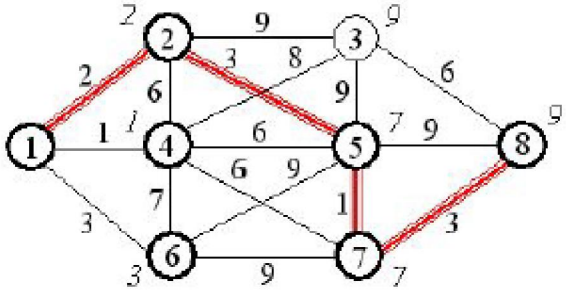
nejmenší vzdálenost z neoznačených
vrcholů má 5, označit jej,
aktualizovat vzdálenost do 7,8



vzít 7 a označit jej,
aktualizovat vzdálenost do 8



8 je cíl, vzít jej a označit



Nejkratší cesta z vrcholu 1 do 8 je: 1 – 2 – 5 – 7 – 8

Floyd-Warshallův algoritmus

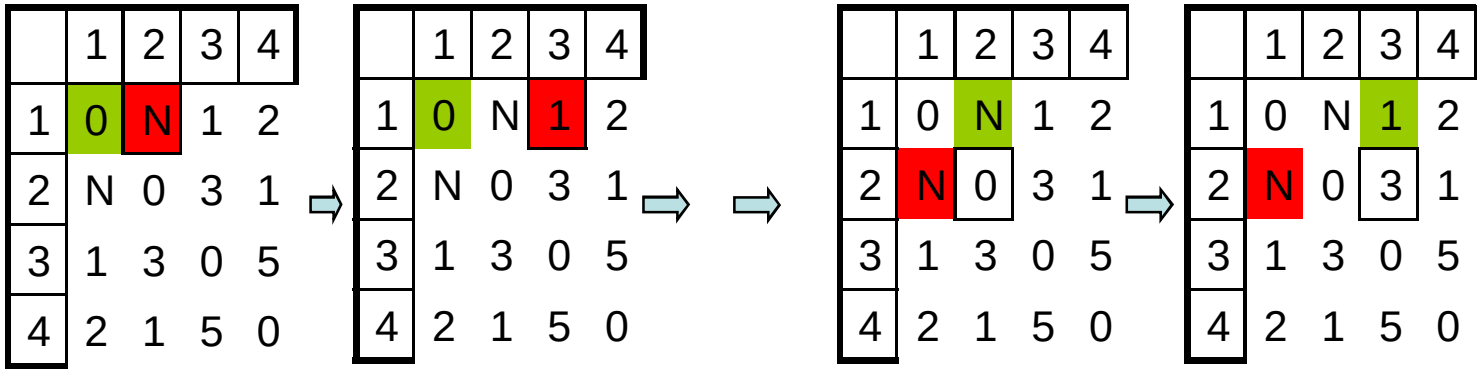
- Algoritmus hledající minimální cestu mezi všemi páry vrcholů (all-pair shortest path algorithm)
- Vhodný pro husté grafy – v tom případě rychlejší než Dijkstra opakovaný pro všechny vrcholy
- Pracuje s maticí sousednosti
- Složitost $O(n^3)$
- Můžeme stanovit max. počet vrcholů, přes které se jde
- Je technikou dynamického programování – viz dále

```
FLOYD-WARSHALL(G,d,mezi)
   vstup  :  souvislý graf G=(V, H) s nezáporným ohodnocením hran  w: H(G) →ℜ+;
   výstup :  d[i,j], i,j ∈V;
             mezi[i,j], i,j ∈V;
   for i := 1 to |V| do
      for j := 1 to |V| do
          begin d[i,j] := w[i,j];
                mezi[i,j] := null
          end;
   for k := 1 to |V| do
      for i := 1 to |V| do
         for j := 1 to |V| do
            if d[i,k] + d[k,j] < d[i,j]
              then begin d[i,j] := d[i,k] + d[k,j];
                         mezi[i,j] := k
                   end;
```

- Matice sousednosti je uložena ve **w**
- **d** je matice aktuálně spočtených nejkratších vzdáleností
- **mezi** je matice nejkratších mezicest, je nainicializována na -1 (null)
- V každém kroku algoritmu zjišťuji, jestli existuje mezi vrcholy i,j kratší cesta přes vrchol k, pokud ano, nastavím vzdálenost v **d[i][j]** na novou velikost a do **path[i][j]** zaznamenám vrchol k

Floyd-Warshallův algoritmus

- **Příklad –** je dána matice sousednosti grafu. FW algoritmem nalezněte nejkratší cestu mezi všemi vrcholy



(k,i,j)=(1,1,2)

0+"N" < "N" ?

•NENÍ

(k,i,j)=(1,1,3)

0+1 < 1 ?

•NENÍ

(k,i,j)=(1,2,2)

"N"+"N" < 0 ?

•NENÍ

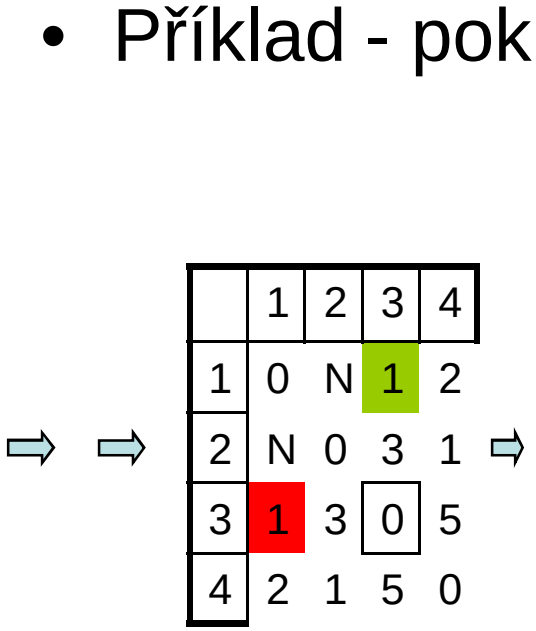
(k,i,j)=(1,2,3)

"N"+1 < 3 ?

•NENÍ

Floyd-Warshallův algoritmus

- Příklad - pokračování

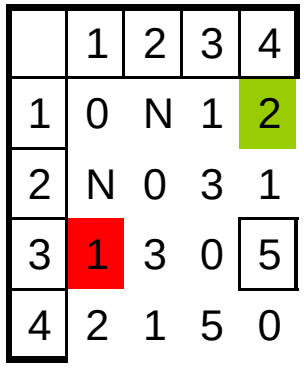


	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	5
4	2	1	5	0

(k,i,j)=(1,3,3)

1+1 < 0 ?

•NENÍ

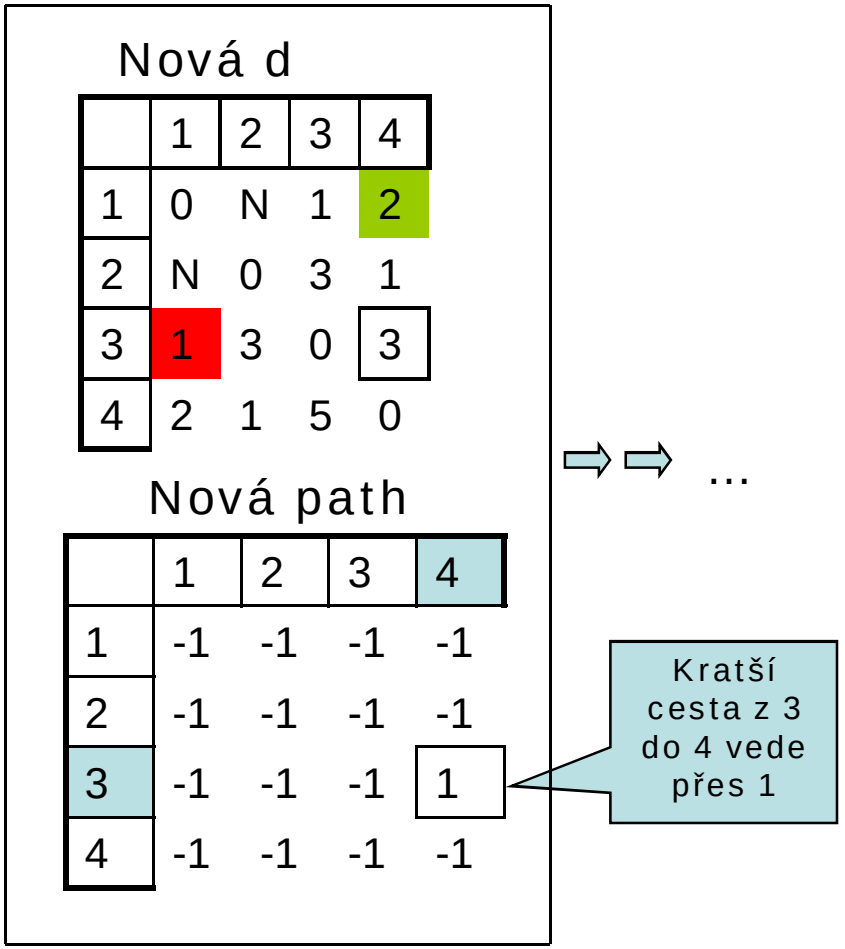


	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	5
4	2	1	5	0

(k,i,j)= (1,3,4)

1+2 < 5 ?

•JE



	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	3
4	2	1	5	0

	1	2	3	4
1	-1	-1	-1	-1
2	-1	-1	-1	-1
3	-1	-1	-1	1
4	-1	-1	-1	-1

Floyd-Warshallův algoritmus

- Rekonstrukce nejkratší cesty mezi i,j
  - V **mezi[i][j]** je index vrcholu k, přes který se má jít. Pokud k není -1(null) podívám se do **mezi** na nejkratší cestu mezi i,k a k,j
  - Toto opakuji rekurzivně dokud nenajdu **mezi[i][j] == -1**.

Minimální kostra grafu

- Algorimus na hledání minimální kostry grafu (Minimum Spannig Tree)

Kostra grafu:

- minimální souvislý podgraf, který obsahuje všechny vrcholy
- neobsahuje cykly – je to strom

Definice: Je dán souvislý graf G, jehož hrany jsou ohodnoceny reálnými čísly, které budeme nazývat cenami. Kostrou grafu, která má nejmenší ohodnocení hran mezi všemi kostrami, nazýváme „nejlevnější (minimální) kostrou grafu“

Aplikace nejlevnější kostry

Elektrifikace území

O. Borůvka algoritmus hledání nejlevnější kostry objevil jako řešení konkrétní praktické úlohy *elektrifikace území*. V této úloze obce tvoří vrcholy grafu, elektrická spojení hrany a ohodnocení hran reprezentuje cenu za natažení vedení mezi obcemi. Úkolem bylo najít takové spojení, aby všechny obce byly připojeny do elektrické sítě a přitom cena spojení byla minimální.

Sjízdnost silnic

Máme silniční síť, z níž chceme udržovat sjízdnou co nejkratší část, která vzájemně propojí všechny obce dané oblasti.

Vybudování železniční sítě

Mezi městy daného regionu máme navrhnout železniční síť tak, aby každá 2 města byla po železnici dosažitelná a přitom náklady na vybudování železniční sítě byly minimální.

Kruskalův algoritmus:

[*Určit nejlevnější kostru v souvislém ohodnoceném grafu* $G=(V,H)$]

1. [*Inicializace kostry* K.]
   $V(K) := V(G)$; $H(K) := \emptyset$; $\{ K := (V , \emptyset) \}$
2. [*Přidání hrany do* K.]
   Nechť h je hrana s minimálním ohodnocením taková, že $h \notin H(K)$

   a $(V, H(K) \cup \{h\})$ je acyklický graf,
   $H(K) := H(K) \cup \{h\}$.
3. [*Test ukončení*.]
   Jestliže $|H(K)|=|V(G)|-1$, pak $K= (V,H(K))$, jinak návrat na krok 2.

Alternativně:

1. $V(K) := V(G)$; $H(K) := H(G)$; $\{ K := G \}$
2. [*Odebrání hrany z* K.]
   Nechť h je hrana s maximálním ohodnocením taková, že $h \in H(K)$
   a $(V, H(K)-\{h\})$ je souvislý graf,
   $H(K) := H(K)-\{h\}$.
3. [*Test ukončení*.]
   Jestliže $|H(K)|=|V(G)|-1$, pak $K= (V,H(K))$, jinak návrat na krok 2.

Borůvkův algoritmus

[*Určit nejlevnější kostru v souvislém ohodnoceném grafu* $G=(V,H)$]

1. [*Inicializace lesa* L.]
   $V(L) := V(G)$; $H(L) := \emptyset$; $\{ L := (V , \emptyset) \}$
2. [*Aktualizace lesa* L.]
   Pro každou komponentu L' lesa L určit nejlevnější hranu z hran vycházejících z L' k jiným komponentám. Nechť S je množina těchto hran. Pak položme
   $H(L) := H(L) \cup S$
   a určeme komponenty souvislosti v lese $L = (V , H(L))$
3. [*Test ukončení*.]
   Jestliže $|H(L)|=|V(L)|-1$, pak $K= (V,H(L))$, jinak návrat na krok 2.
   (Alternativně jestliže počet komponent je roven 1, pak $K= (V,H(L))$,
   jinak návrat na krok 2.)

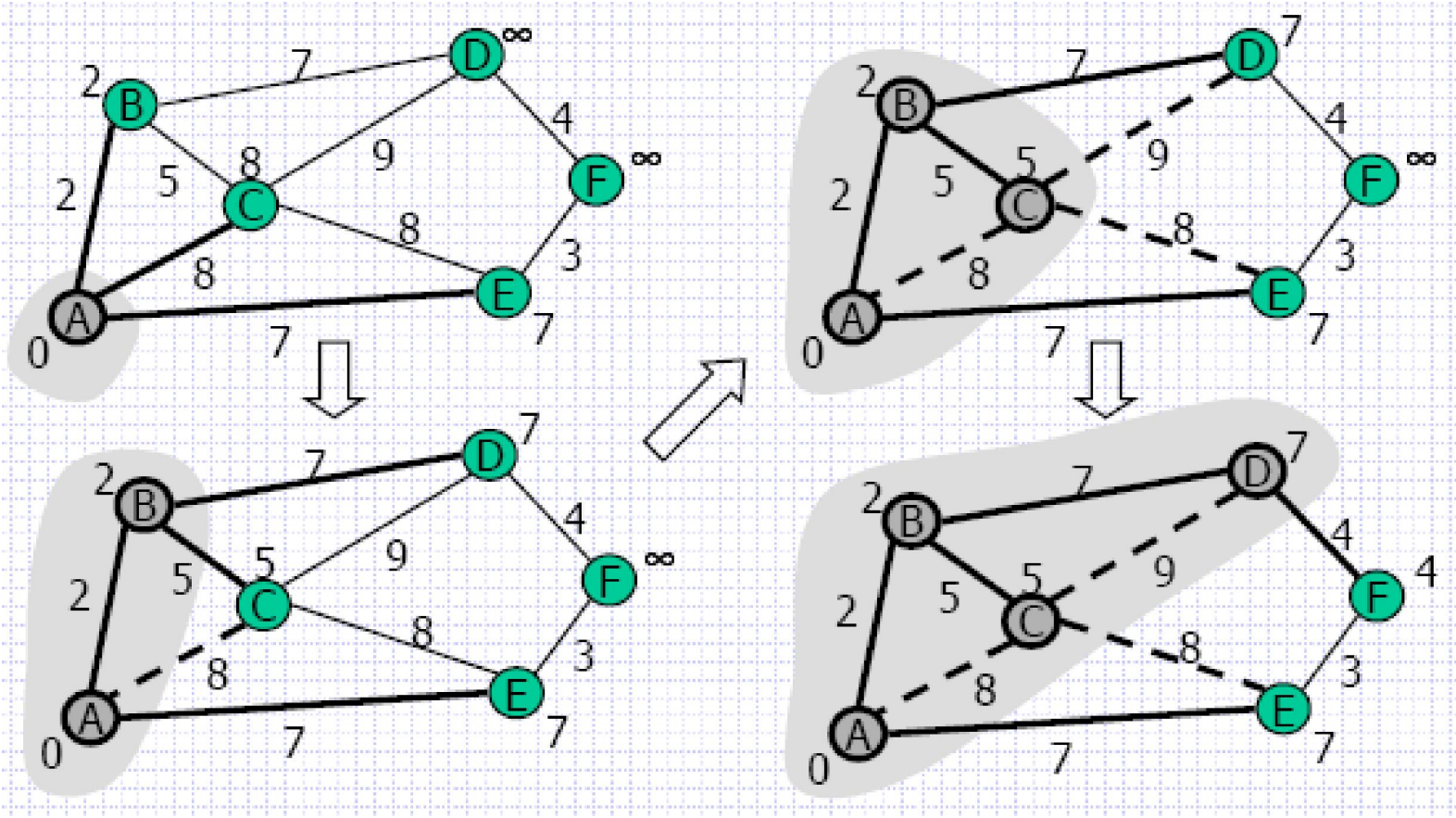
Primův – Jarníkův algoritmus

[*Určit nejlevnější kostru v souvislém ohodnoceném grafu* $G=(V,H)$]

1. [*Inicializace kostry* K.]
   Nechť v je libovolný vrchol grafu G ,
   $V(K) := \{v\}$; $H(K) := \emptyset$; $\{ K := (\{v\},\emptyset) \}$
2. [Aktualizace *kostry*.]
   Nechť h je hrana s minimálním ohodnocením z hran spojujících vrchol $u \in V(K)$ s vrcholem $v \notin V(K)$, pak
   $V(K) := V(K) \cup \{v\}$; $H(K) := H(K) \cup \{h\}$
3. [Test *ukončení*.]
   Jestliže $|H(K)|=|V(G)|-1$, pak $K= (V,H(K))$, jinak návrat na krok 2.

Primův-Jarníkův algoritmus

- Příklad(1)



Primův-Jarníkův algoritmus

- Příklad - pokračování