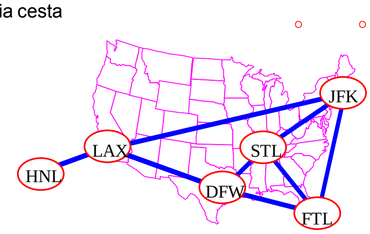
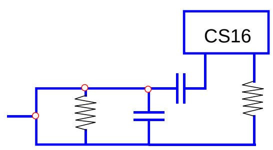
Grafy

aplikácie

- siete:
  - lety, komunikácie
- mapy:
  - plánovanie cesty, najkratšia cesta



aplikácie



- elektronické obvody
  - návrh a kontrola obvodu
  - skrat?
  - dá sa navrhnúť obvod bez kríženia spojov?

aplikácie

- Hypertext
  - dokumenty – vrcholy
  - odkazy – hrany
  - webové prehliadače – nutnosť kontrolovať zacyklenie

graf

- ***Grafom G***, presnejšie neorientovaným, nazývame dvojicu množín *G=(V,H)*, kde *V* je množina vrcholov a *H* je množina hrán daného grafu, ak je pre každú jeho hranu určené, ktorú dvojicu vrcholov spája
  - (zatiaľ) sa nevyžaduje, aby dvojica vrcholov bola usporiadaná

graf

- majme vrcholy *u*, *v* a hranu *h*, ktorá ich spája.
  - zapisujeme: ***h = uv*** alebo ***(u,v)***
  - hovoríme: hrana *h* je ***incidentná*** s vrcholmi *u* a *v*.
  - o vrcholoch *u*, *v* hovoríme, že sú *susedné*.
- Vrcholovú množinu grafu *G* označujeme ***V(G)*** a hranovú množinu grafu *G* označujeme ***H(G)***.

graf

- Ak hrana spája vrchol sám so sebou, $h = uu$, nazývame ju ***slučka.***
- Hranu, ktorá nie je slučka, nazývame ***linka***.
- Vrcholy môžu byť spojené viacerými hranami. Takéto hrany nazývame ***násobné***.
- Graf, ktorý nemá slučky ani násobné hrany, nazývame ***obyčajný***.

graf

- Počet hrán, s ktorými je vrchol ***u*** incidentný je ***stupeň vrcholu*** (označujeme deg(u)).
- Ak je vrchol incidentný so slučkou, počítame ju ako dve hrany. Stupeň vrcholu potom môžeme vypočítať jako **deg(*u*)=*l*+2*s*,** kde l je počet liniek a s je počet slučiek incidentných s vrcholom u.

graf

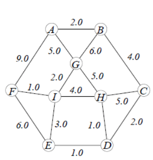
- Ak G je graf, ktorého každý vrchol má rovnaký stupeň k, nazývame ho ***pravidelným*** stupňa ***k***.
- Ak K je obyčajný graf, ktorého každý vrchol je spojený hranou (je incidentný) so všetkými ostatnými vrcholmi grafu, nazývame ho ***úplný*** (*kompletný)* a označujeme K_n, kde n je počet vrcholov úplného grafu.

orientovanosť grafu

- Hranu h z hranovej množiny $H(G)$ grafu G nazývame ***orientovaná***, ak jej priradíme orientáciu. To znamená, že o nej vieme povedať, v ktorom vrchole začína a v ktorom končí.
- Hranu h z hranovej množiny $H(G)$ grafu G nazývame ***neorientovaná***, ak nie je orientovaná.
- Podľa toho, či graf obsahuje orientované alebo neorientované hrany, resp. obidva druhy, môžeme o ňom povedať, že je orientovaný, neorientovaný alebo zmiešaný:
  - Graf G nazývame ***orientovaný***, ak všetky hrany h z jeho hranovej množiny $H(G)$ sú orientované.
  - Graf G nazývame ***neorientovaný*,** ak všetky hrany h z jeho hranovej množiny $H(G)$ sú neorientované.
  - Graf G nazývame ***zmiešaný*,** ak obsahuje orientované aj neorientované hrany.

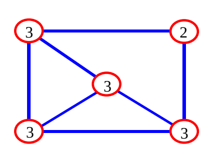
hranovo ohodnotený graf

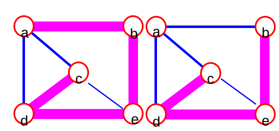
- Graf G nazývame ***hranovo ohodnotený***, ak je každej hrane h z hranovej množiny $H(G)$ grafu G priradené reálne číslo w_h .
- inak: Hranovo ohodnotený graf je dvojica (G, W), pozostávajúca z grafu G=(V,H) a váhovej funkcie W: H $\rightarrow$ Real.
- príklad:



cesta

- cesta z vrcholu x do vrcholu y: postupnosť vrcholov $v_0, v_1, v_2, \ldots v_k$ taká, že $v_0=x$ a $v_k=y$ a hrany $(v_0\,v_1)$, $(v_1\,v_2)$, ... patria do množiny hrán H. Dĺžka tejto cesty je k.

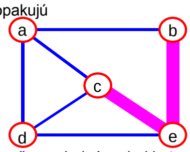




a b e d c   b e d c

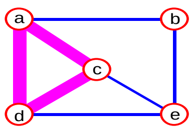
cesta

- jednoduchá cesta: vrcholy sa neopakujú

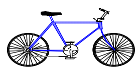


b e c

- cyklus: jednoduchá cesta až na to, že posledný vrchol je ten istý ako prvý

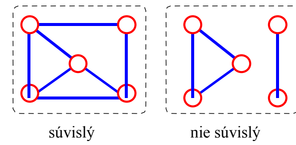


a c d a



súvislý graf

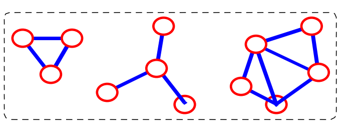
- súvislý graf: ľubovoľné dva vrcholy spája nejaká cesta



- podgraf: podmnožina vrcholov a hrán tvoriaca graf

komponent grafu

- súvislý komponent grafu je maximálna množina vrcholov s vlastnosťou, že každý jej vrchol je dosiahnuteľný z každého iného vrcholu v komponente.
- inak: maximálny súvislý podgraf
- príklad: graf s troma súvislými komponentami.

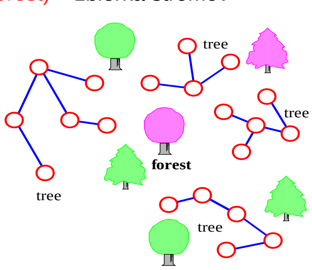


strom ešte raz

- Neorientovaný graf nazývame stromom (angl. tree), ak je acyklický (neobsahuje cykly) a je súvislý.
- Hovoríme, že strom je zakorenený (angl. rooted), ak má vyznačený jeho najvyšší vrchol - koreň (angl. root).

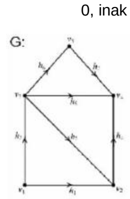
strom a les

- les (forest) – zbierka stromov



matica incidencie

- Majme orientovaný graf G na n vrcholoch $v_1, v_2, \ldots, v_n$ s m hranami $h_1, h_2, \ldots, h_m$. ***Maticou incidencie*** grafu G nazveme maticu $A = (a_{ij})$ typu $n{\times}m$, kde
- $a_{ij} = \begin{cases} -1, & \text{ak hrana } h_j \text{ končí vo vrchole } v_i \\ 1, & \text{ak hrana } h_j \text{ začína vo vrchole } v_i \\ 0, & \text{inak} \end{cases}$

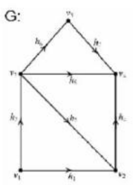


Matica incidencie:

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

matica susednosti

- Majme graf G na n vrcholoch $v_1, v_2, \ldots, v_n$. ***Maticou susednosti*** (adjacenčnou maticou) grafu G nazveme maticu $B = (b_{ij})$ typu $n{\times}n$, kde
- $b_{ij} = \{$ 1, ak existuje hrana, ktorá začína vo vrchole v_i a končí vo vrchole v_j
  0, inak$\}$



Matica susednosti:

$$B = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Zoznamy okolí vrcholov:

v_1 : v_2, v_3;
v_2 : v_4,
v_3 : v_2, v_4, v_5;
v_4 : ;
v_5 : v_4;

ADT Graf

údaj typu graf: neprázdna množina vrcholov a množina neorientovaných hrán, kde každá hrana je dvojca vrholov

operácie: pre všetky *graf* ∈ *Graf*, *v*, v_1 a v_2 ∈ *Vrcholy*

Graf Create()::=vracia prázdny graf

Graf InsertVertex(*graf*, *v*)::= vracia graf s vloženým *v*. *v* nemá s ním incidentnú hranu.

Graf InsertEdge(*graf*, v_1,v_2)::= vracia graf s novou hranou medzi v_1 a v_2

Graf DeleteVertex(*graf*, *v*)::= vracia graf, z ktorého sa odstránil vrchol *v* a všetky s ním incidentné hrany

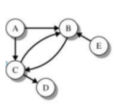
Graf DeleteEdge(*graf*, v_1, v_2)::= vracia graf, z ktorého sa odstráni hrana (v_1, v_2)

Boolean IsEmpty(*graf*)::= if (*graf*==*prázdny graf*) return TRUE
else return FALSE

List Adjacent(*graph*,*v*)::= vracia zoznam všetkých vrcholov susediacich s *v*

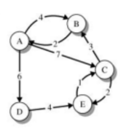
reprezentovanie grafu

- maticou susednosti:
  - prvok:
    - 1 alebo 0 ak ne/existuje hrana z v_i do v_j.
    - váha hrany (v_i,v_j) alebo 0 ak ne/existuje hrana z v_i do v_j (hranovo ohodnotený graf)
      - namiesto 0 hodnota, ktorá nikdy nemôže byť ohodnotením hrany



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	1	0	0
C	0	1	0	1	0
D	0	0	0	0	0
E	0	1	0	0	0

(a)



	A	B	C	D	E
A	0	4	7	6	0
B	2	0	0	0	0
C	0	3	0	0	2
D	0	0	0	0	4
E	0	0	1	0	0

(b)

reprezentovanie grafu

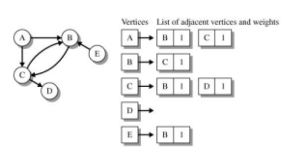
- maticou susednosti:
  - prvky sú 1 alebo 0
  - bežne grafy neobsahujú priveľa hrán, preto matica obsahuje priveľa núl (je riedka)
  - neúsporná reprezentácia z hľadiska pamäti
- maticou incidencie:
  - prvky sú -1, 0, 1, ale
  - v každom stĺpci len len jedna 1, jedna -1 a inak samé nuly (prečo?)
  - ešte neúspornejšia reprezentácia

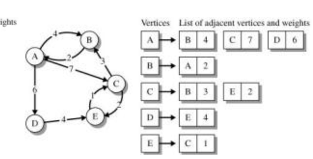
reprezentovanie grafu

- zoznamami okolí vrcholov:
- ***Zoznam vrcholov*** *a* ***zoznam hrán*** je údajová štruktúra, v ktorej sú množiny vrcholov a hrán opísané vymenovaním prvkov. Každá hrana sa zapíše ako usporiadaná dvojica vrcholov (začiatočný a koncový).
- Graf *G* je daný ***zoznamom okolí vrcholov***, ak je ku každému vrcholu v_i priradený zoznam tých vrcholov, do ktorých vedie hrana z v_i. Niekedy je výhodné uvádzať pre každý vrchol *v* dva zoznamy; zoznam hrán, ktoré z neho vychádzajú, G+ (*v*) a zoznam hrán, ktoré do neho vchádzajú, G- (*v*).

Reprezentovanie grafu

- zoznamy susediacich vrcholov



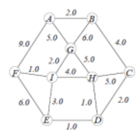


reprezentovanie neorientovaného grafu

- Ak *G* je neorientovaný graf:
  - každá hrana - dvojica opačne orientovaných hrán.
  - matica incidencie grafu *G* -> binárna matica (obsahuje iba jednotky a nuly)
  - matica susednosti grafu *G* -> symetrická binárna matica.
  - Pre zoznamy okolí platí: ak je v_j v zozname vrcholu v_i , tak v_i je v zozname vrcholu v_j.

príklad: reprezentácia neorientovaného grafu

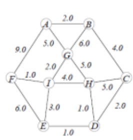
- neorientovaný hranovo ohodnotený graf maticou susednosti:
- všimnime si:
  - symetrická

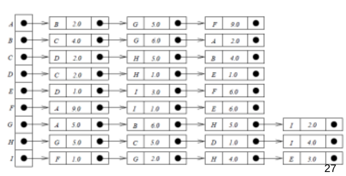


$$\begin{pmatrix} 0 & 2.0 & 0 & 0 & 0 & 9.0 & 5.0 & 0 & 0 \\ 2.0 & 0 & 4.0 & 0 & 0 & 0 & 6.0 & 0 & 0 \\ 0 & 4.0 & 0 & 2.0 & 0 & 0 & 0 & 5.0 & 0 \\ 0 & 0 & 2.0 & 0 & 1.0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 6.0 & 0 & 0 & 3.0 \\ 9.0 & 0 & 0 & 0 & 6.0 & 0 & 0 & 0 & 1.0 \\ 5.0 & 6.0 & 0 & 0 & 0 & 0 & 0 & 5.0 & 2.0 \\ 0 & 0 & 5.0 & 1.0 & 0 & 0 & 5.0 & 0 & 4.0 \\ 0 & 0 & 0 & 0 & 3.0 & 1.0 & 2.0 & 4.0 & 0 \end{pmatrix}$$

príklad: reprezentácia neorientovaného grafu

- neorientovaný hranovo ohodnotený graf zoznamami susediacich vrcholov:
- všimnime si:
  - informačná nadbytočnosť





efektívnosť

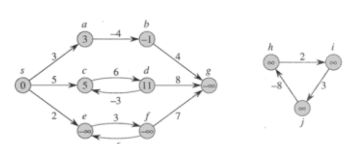
- nech N je počet vrcholov grafu, M je počet hrán grafu a dmax je najväčší zo stupňov vrcholov

Efektívnosť	zoznamy hrán	matica susednosti	zoznamy susednosti
Pamäťové nároky	2M	N^2	2M
Sú dva vrcholy susedné?	M	1	dmax
Susedné vrcholy daného vrcholu	M	N	dmax
Pridaj hranu do grafu	1	1	1
Zmaž hranu z grafu	M	2	2dmax

Najkratšia cesta v orientovanom grafe

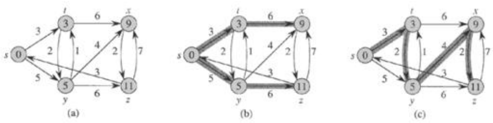
- z jedného východiska:
  - Dijkstra
  - Bellman – Ford
- zo všetkých vrcholov:
  - Floyd - Warshall

záporne ohodnotené hrany v orientovanom grafe



- vnútri vrchola je vpísaná dĺžka najkratšej cesty z východiska s.
- vrcholy e a f tvoria záporný cyklus dosiahnuteľný z východiska s, preto majú obe dĺžku najkratšej cesty z východiska -∞
- vrchol g je dosiahnuteľný z vrchola s -∞, preto aj on má -∞.
- vrcholy h, i, j nie sú dosiahnuteľné z s, preto majú + ∞ napriek tomu, že tvoria záporný cyklus.

príklad orientovaného hranovo ohodnoteného grafu s dĺžkami minimálnych ciest z s

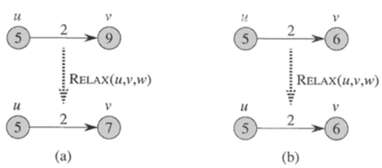


a) orientovaný hranovo ohodnotený graf s dĺžkami minimálnych ciest z s
b) vytieňované hrany tvoria strom najkratších ciest s koreňom vo východisku s.
c) iný strom najkratších ciest s koreňom vo východisku s.

relaxácia

- metóda na opakované znižovanie horného ohraničenia na váhu/dĺžku aktuálnej najkratšej cesty pre každý vrchol, až sa horné ohraničenie bude rovnať váhe najkratšej cesty

relaxácia hrany



- relaxácia hrany (u, v) s ohodnotením w(u,v)=2.
  vnútri vrcholu je vpísaný odhad dĺžky najkratšej cesty.

a) pred relaxáciou platí d[v] > d[u] + w(u,v), preto sa d[v] zmenší
b) pred relaxáciou platí d[v] =< d[u] + w(u,v), preto sa d[v] nezmení

inicializácia

inicializuj-jedno-vychodisko (G, s)
1 for každý vrchol v $\in$ V[G]
2 do d[v] $\leftarrow \infty$
3 π[v] $\leftarrow$ NIL
4 d[s] $\leftarrow$ 0

d – odhad dĺžky najkratšej cesty z východiska
π- vrchol predchodca na ceste z východiska
po inicializácii platí:
π[v] = NIL pre všetky v $\in$ V[G],
d[v] = 0 pre v=s,
d[v] = ∞ pre v $\in$ V[G] – {s} // najhorší možný odhad – už sa môže len zlepšovať

relaxácia

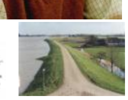
Relax(u,v,w)
1 if d[v] > d[u] + w(u,v)
2 then d[v] $\leftarrow$ d[u] + w(u,v)
3 π[v] $\leftarrow$ u

Edsger Wybe Dijkstra

May 11, 1930, Rotterdam – August 6, 2002, Nuenen

vyštudoval teoretickú fyziku
jeden z prvých programátorov na svete, začiatok 50. rokov 20. storočia
Technická univerzita v Eidhovene
Texaská univerzita v Austine

prínosy:
•Dijkstrov algoritmus hľadania najkratšej cesty v grafe
•bankárov algoritmus
•THE systém multiprogramovania
•semafór
•predložil zničujúcu analýzu príkazu skoku



Dijkstrov algoritmus

- predpoklad:
  - všetky váhy musia byť nezáporné $w(u, v) \geq 0$ pre všetky hrany $(u, v) \in H$
- inicializácia, relaxácia
- udržiava množinu vrcholov S, ktorých definitívne najkratšie dĺžky najkratších ciest sú už určené
- opakovane vyberie vrchol $u \in V - S$ s najmenším odhadom minimálnej dĺžky cesty, pridá ho do S a relaxuje všetky hrany vychádzajúce z u.

Dijkstrov algoritmus

```
Dijkstra(G, w, s)
1 Inicializuj-jedno-vychodisko(G, s)
2 S ← ∅
3 Q ← V[G]
4 while Q ≠ ∅
5  do u ← Extrakt-Min(Q)
6      S ← S ∪ {u}
7      for každý vrchol v ∈ adj[u]
8          do Relax(u, v, w)
```

Q – min-prioritný front

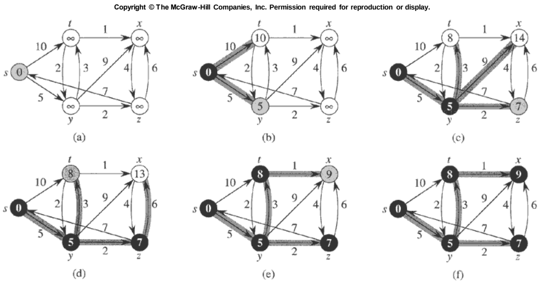


Figure 24.6 The execution of Dijkstra's algorithm. The source s is the leftmost vertex. The shortest-path estimates are shown within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. **(a)** The situation just before the first iteration of the **while** loop of lines 4–8. The shaded vertex has the minimum d value and is chosen as vertex u in line 5. **(b)–(f)** The situation after each successive iteration of the **while** loop. The shaded vertex in each part is chosen as vertex u in line 5 of the next iteration. The d and π values shown in part (f) are the final values.

Dijkstrov algoritmus - ako rýchly je?

- Q vo vektore:
  - Extract-Min O(V), opakuje sa |V| krát, spolu $O(V^2)$
  - každý vrchol sa vkladá do S práve raz. každá hrana sa v cykle 4 skúma práve raz. cyklus sa opakuje |H| krát
  - $O(V^2 + H) = O(V^2)$
- Q v binárnej halde:
  - Extract-Min O(log V), opakuje sa |V| krát
  - vytvorenie binárnej haldy O(V)
  - relaxovanie sa zrealizuje pomocou operácie Decrease-Key O(log V)
  - stále je |H| opakovaní
  - O((V + H). log V) = O(H . log V) ak sú všetky vrcholy dosiahnuteľné z východiska

Bellmanov-Fordov algoritmus

- ohodnotenia hrán môžu byť záporné – všeobecnejší algoritmus
- vracia true, ak v grafe neexistuje záporný cyklus dosiahnuteľný z východiska, inak vracia false (neexistuje riešenie)
- ak true, vracia najkratšie cesty a ich váhy.
- inicializácia,
- relaxácia: |V| - 1 prechodov cez hrany grafu, toľkokrát sa upravujú horné ohraničenia vo vrcholoch
- nakoniec sa urobí test na záporné cykly (cyklus 5 – 7)

Bellmanov-Fordov algoritmus

```
Bellman-Ford(G, w, s)
1 Inicializuj-jedno-vychodisko(G, s)
2 for i ← 1 to |V[G]| - 1
3    do for každú hranu (u, v) ∈ H[G]
4          do Relax(u, v, w)
5 for každú hranu (u, v) ∈ H[G]
6  do if d[v] > d[u]+w(u, v)
7        then return FALSE
8  return TRUE
```

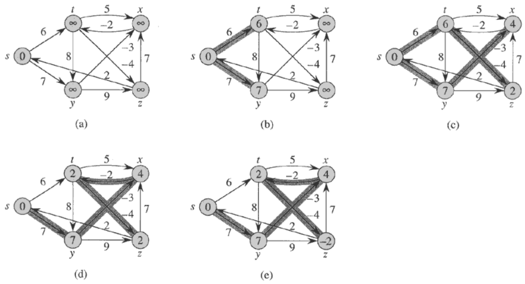


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

Bellmanov-Fordov algoritmus – ako rýchly je?

- inicializácia riadok 1 potrebuje O(V)
- každý z |V| - 1 prechodov na riadkoch 2 – 4 potrebuje O(H)
- záverečný test na riadkoch 5 – 7 potrebuje O(H).
- celkovo O(V.H)


```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u, taken in topologically sorted order
4      do for each vertex v ∈ Adj[u]
5             do RELAX(u, v, w)
```

topologické usporiadanie vrcholov grafu G(V, H): lineárne usporiadanie všetkých vrcholov z V také, že ak H obsahuje hranu (u, v), tak u je v usporadaní pred v. (Ak G nie je acyklický, tak usporiadanie neexistuje.)

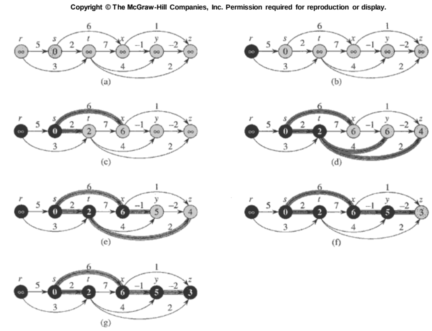


Figure 24.5 The execution of the algorithm for shortest paths in a directed acyclic graph. The vertices are topologically sorted from left to right. The source vertex is s. The d values are shown within the vertices, and shaded edges indicate the π values. **(a)** The situation before the first iteration of the **for** loop of lines 3–5. **(b)–(g)** The situation after each iteration of the **for** loop of lines 3–5. The newly blackened vertex in each iteration was used as u in that iteration. The values shown in part (g) are the final values.

všetky dvojice najkratších ciest v grafe

- graf sa reprezentuje maticou susednosti
- vstup: matica W ohodnotení hrán orientovaného grafu G = (V, H) s n vrcholmi
- w_{ij} = 0, ak i=j,
- w_{ij} = váha orientovanej hrany (i, j), ak $i \neq j$ a $(i, j) \in H$,
- $w_{ij} = \infty$, ak $i \neq j$ a $(i, j) \notin H$
- výstup: nxn matica D
- d_{ij} = dĺžka najkratšej cesty z i do j.
- ale kade vedú tie cesty? matica predchodcov Π
- π_{ij} = NIL ak i=j alebo neexistuje cesta z i do j,
- π_{ij} = nejaký predchodca vrchola j na najkratšej ceste z vrchola i, inak

tlač najkratších ciest všetkých dvojíc vrcholov v grafe

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
if i = j
   then print i
   else if π_ij = NIL
           then print "no path from" i "to" j "exists"
           else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
                print j
```

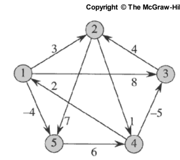
```
EXTEND-SHORTEST-PATHS(L, W)
n ← rows[L]
let L' = (l'_ij) be an n × n matrix
for i ← 1 to n
    do for j ← 1 to n
        do l'_ij ← ∞
           for k ← 1 to n
               do l'_ij ← min(l'_ij, l_ik + w_kj)
return L'
```


súčin matíc

```
MATRIX-MULTIPLY(A, B)
n ← rows[A]
let C be an n × n matrix
for i ← 1 to n
    do for j ← 1 to n
        do c_ij ← 0
           for k ← 1 to n
               do c_ij ← c_ij + a_ik · b_kj
return C
```



```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
n ← rows[W]
L^(1) ← W
for m ← 2 to n − 1
    do L^(m) ← EXTEND-SHORTEST-PATHS(L^(m−1), W)
return L^(n−1)
```

$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Figure 25.1 A directed graph and the sequence of matrices $L^{(m)}$ computed by SLOW-ALL-PAIRS-SHORTEST-PATHS. The reader may verify that $L^{(5)} = L^{(4)} \cdot W$ is equal to $L^{(4)}$, and thus $L^{(m)} = L^{(4)}$ for all $m \geq 4$.

W

1	2	3	4	5
0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

$L^{(1)}$

1	2	3	4	5
0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

$L^{(2)}$

1	2	3	4	5
0	3	8	2	-4
3	0	-4	1	7
∞	4	0	5	11
2	-1	-5	0	-2
8	∞	1	6	0

výpočet v Extend-Shortest-Paths($L^{(1)}$, W)
$l'_{14} = \infty$
for k ← 1 to n // n=5
$l'_{14} = \min(l'_{14}, l_{1k} + w_{k4})$

stopa výpočtu (postupne nadobúdané hodnoty l'_{14}): ∞, min(∞, 0+∞), min(∞,3+1), min(4,8+∞), min(4, ∞+0), min(4, -4+6), 2

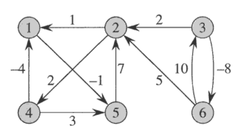


Figure 25.2 A weighted, directed graph for use in Exercises 25.1-1, 25.2-1, and 25.3-1.

FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

$n \leftarrow rows[W]$
$L^{(1)} \leftarrow W$
$m \leftarrow 1$
while $m < n - 1$
  do $L^{(2m)} \leftarrow$ EXTEND-SHORTEST-PATHS$(L^{(m)}, L^{(m)})$
    $m \leftarrow 2m$
return $L^{(m)}$

Robert Floyd

(June 8, 1936 New York – September 25, 2001

- Bc liberálne štúdiá (vo veku 17) Chicago Uni
- Bc fyzika 1958
- docent 1963 Stanford Uni (bez PhD!)
- profesor 1969 Stanford Uni (bez PhD!)

prínosy:
- algoritmus hľadania najkratšej cesty v grafe
- verifikácia programov

Stephen Warshall

1935 New York - December 11, 2006

Bc matematika Harvard
žiadny ďalší titul!

algoritmus pre výpočet tranzitívneho uzáveru
(dokázal prvý správnosť, vyhral fľašu rumu)

je na FB!

Floydov-Warshallov algoritmus

FLOYD-WARSHALL(W)

$n \leftarrow rows[W]$
$D^{(0)} \leftarrow W$
for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

horné indexy matice D: treba n matíc nxn, čiže pamäť rozsahu $\Theta(n^3)$.

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \mathrm{NIL} & 1 & 1 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 2 & 2 \\ \mathrm{NIL} & 3 & \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} \\ 4 & \mathrm{NIL} & 4 & \mathrm{NIL} & \mathrm{NIL} \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 5 & \mathrm{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \mathrm{NIL} & 1 & 1 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 2 & 2 \\ \mathrm{NIL} & 3 & \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} \\ 4 & 1 & 4 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 5 & \mathrm{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \mathrm{NIL} & 1 & 1 & 2 & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 2 & 2 \\ \mathrm{NIL} & 3 & \mathrm{NIL} & 2 & 2 \\ 4 & 1 & 4 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 5 & \mathrm{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \mathrm{NIL} & 1 & 1 & 2 & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 2 & 2 \\ \mathrm{NIL} & 3 & \mathrm{NIL} & 2 & 2 \\ 4 & 3 & 4 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 5 & \mathrm{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \mathrm{NIL} & 1 & 4 & 2 & 1 \\ 4 & \mathrm{NIL} & 4 & 2 & 1 \\ 4 & 3 & \mathrm{NIL} & 2 & 1 \\ 4 & 3 & 4 & \mathrm{NIL} & 1 \\ 4 & 3 & 4 & 5 & \mathrm{NIL} \end{pmatrix}$$

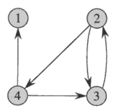
$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \mathrm{NIL} & 3 & 4 & 5 & 1 \\ 4 & \mathrm{NIL} & 4 & 2 & 1 \\ 4 & 3 & \mathrm{NIL} & 2 & 1 \\ 4 & 3 & 4 & \mathrm{NIL} & 1 \\ 4 & 3 & 4 & 5 & \mathrm{NIL} \end{pmatrix}$$

Figure 25.4 The sequence of matrices $D^{(k)}$ and $\Pi^{(k)}$ computed by the Floyd-Warshall algorithm for the graph in Figure 25.1.

Tranzitívny uzáver binárnej relácie

TRANSITIVE-CLOSURE(G)

$n \leftarrow |V[G]|$
for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do if $i = j$ or $(i, j) \in E[G]$
      then $t_{ij}^{(0)} \leftarrow 1$
      else $t_{ij}^{(0)} \leftarrow 0$
for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
return $T^{(n)}$



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Figure 25.5 A directed graph and the matrices $T^{(k)}$ computed by the transitive-closure algorithm.

Floydov-Warshallov algoritmus'

FLOYD-WARSHALL$'(W)$

```
n ← rows[W]
D ← W
for k ← 1 to n
    do for i ← 1 to n
           do for j ← 1 to n
                  do d_ij ← min(d_ij, d_ik + d_kj)
return D
```

jediná zmena: nie sú horné indexy matice D. netreba n matíc nxn, čiže pamäť rozsahu nie $\Theta(n^3)$, ale len $\Theta(n^2)$.

rez

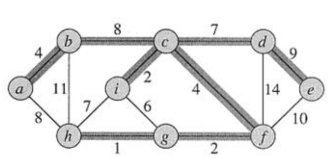
- *Rez grafu* sa nazýva množina prvkov súvislého grafu, po odstránení ktorých sa graf rozpadne na dva komponenty a žiadna podmnožina rezu nemá túto vlastnosť.
- Odstránenie vrcholu z grafu automaticky odstráni aj hrany s ním incidentné.
- Ak je rez vrchol, nazýva sa *artikulácia*, ak je to hrana, nazýva sa *most*.

kostra

- Strom, ktorý obsahuje všetky vrcholy grafu sa nazýva ***kostra grafu.***
- ***minimálna kostra grafu***
  - kostra grafu, ktorej váha nie je väčšia než váha ľubovoľnej inej kostry.
  - váha kostry je daná súčtom váh všetkých jej hrán
- Aplikácie
  - komunikačné siete
  - transportné siete

minimálna kostra grafu

- príklad: hrany kostry sú zvýraznené tieňom. váha kostry je 37(=4+8+7+9+2+4+1+2). táto min kostra nie je jediná (delete (b,c), insert (a,h)).



generický algoritmus

Generický-MST(G, w)

```
1 A ← ∅
2 while A netvorí kostru
3       do nájdi hranu (u, v) takú,
                že je „bezpečná“ (safe) pre A
4          A ← A ∪ {(u, v)}
5 return A
```

všeobecný princíp hľadania minimálnej kostry grafu

- G=(V, H) je súvislý graf; X, Y sú dve množiny hrán
- Invariant cyklu: T=(V, X), X⊆H, X∪Y=H
  - Existuje minimálna kostra T grafu G, ktorá obsahuje všetky hrany z X a žiadnu hranu z Y.
- Nápad: pridávať hrany do X tak, aby sa zachovávala platnosť invariantu

Joseph Bernard Kruskal, Jr.

(January 29, 1928 New York - September 19, 2010)

PhD matematika Princeton Uni

prínosy:
•algoritmus pre minimálnu kostru grafu
•štatistika
•matematická logika
•lingvistika

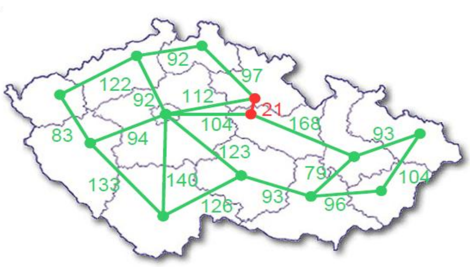
Kruskalov algoritmus

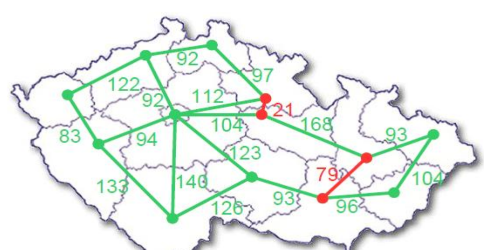
- Využíva vlastnosti cyklu
- Do MST pridáva len hrany, ktoré netvoria s už pridanými cyklus
- 1. Usporiadať hrany grafu vzostupne podľa váhy
- 2. Vytvoriť les MST podstromov (každý podstrom je tvorený jediným vrcholom)
- 3. Postupne prechádzať usporiadané hrany a do MST vkladať len hrany, ktoré spájajú dva oddelené podstromy

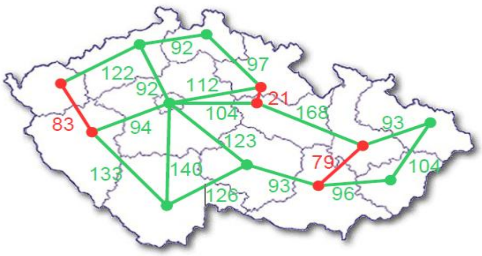
Kruskalov algoritmus

vstup: súvislý graf G=(V,H)
X:=Ø; Y:=Ø; Q:=H
while Q ≠Ø **do**
  vezmi a odstráň z Q hranu {u, v} s najmenšou váhou
  if u a v sú v rôznych súvislých komponentoch grafu (V, X) **then**
    X:=X ∪ {{u,v}}
  else
    Y:=Y ∪ {{u,v}}
return (V, X)

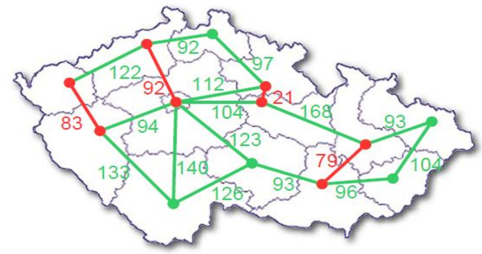
- Q by mal byť prioritný front



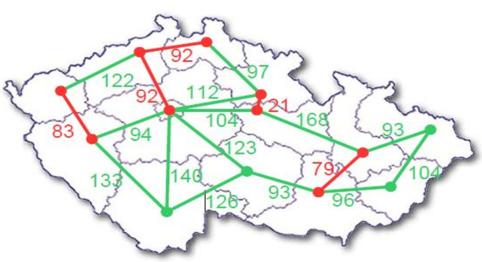




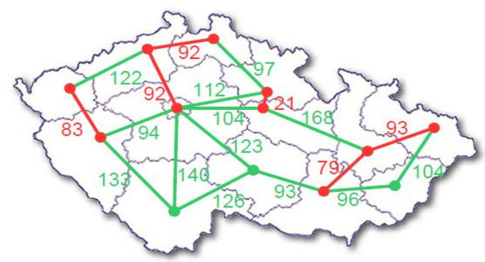
73



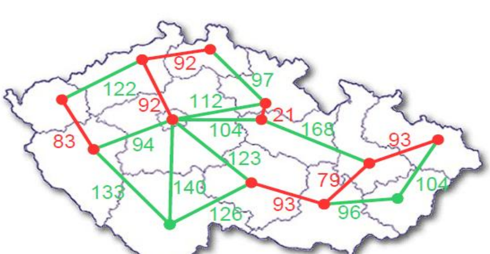
74



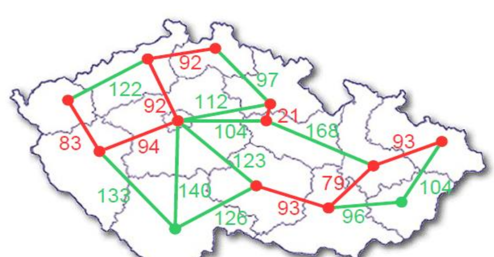
75



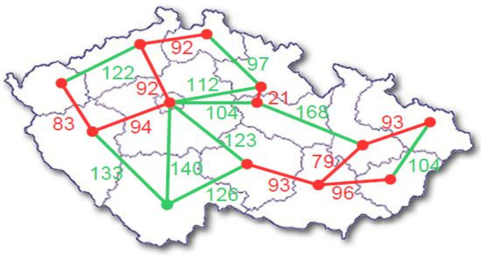
76



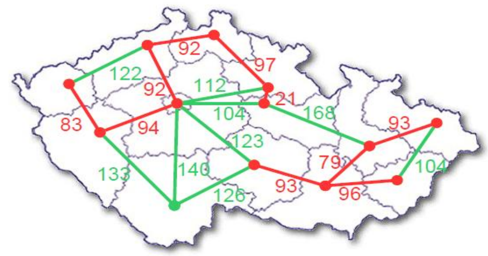
77



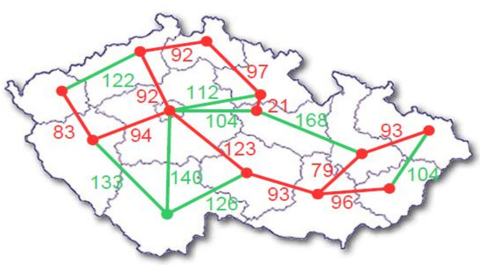
78



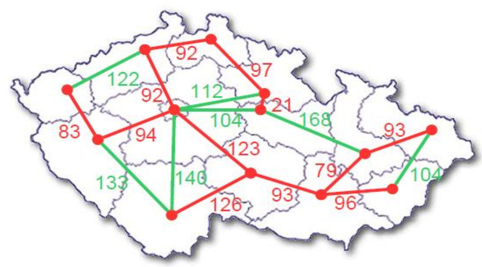
79



80



81



82

Vlastnosti Kruskalovho algoritmu

- zložitosť
- kritické je usporiadanie hrán – O(E log E)

83

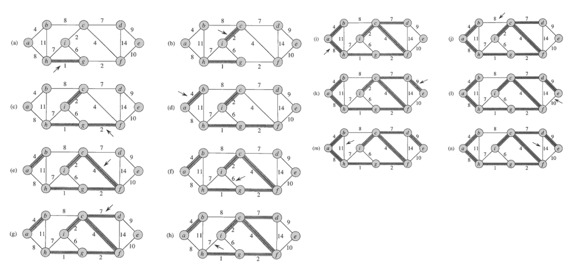


Figure 23.4 The execution of Kruskal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest *A* being grown. The edges are considered by the algorithm in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees.

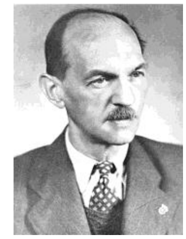
84

Kruskalov algoritmus

MST-Kruskal(G, w)
A ← ∅
for každý vrchol v ∈ V[G]
do Make-Set(v)
usporiadaj hrany v H v neklesajúcom poradí podľa váhy w
for každú hranu (u, v) ∈ H v poradí podľa neklesajúcej váhy
    do if Find-Set(u) ≠ Find-Set(v)
        then A ← A ∪ {(u, v)}
            Union(u, v)
return A

vrcholy – triedy ekvivalencie
množina zvláštny prípad:
Make-Set(v), Find-Set(v), Union(u, v)

Vojtěch Jarník



December 22, 1897 Praha – September 22, 1970 Praha

matematika a fyzika, 1920, Karlova Uni
mim profesor 1929, Karlova U
riadny profesor 1936, Karlova U

prínosy
- teória čísiel
- matematická analýza
- V. Jarník: O jistém problému minimálním, Práce Moravské přírodovědecké společnosti, 6, 1930, pp. 57– 63.

Robert C. Prim

1921 in Sweetwater, Texas

1941 B.S. elektrotechnické inžinierstvo, Princeton Uni
1949 matematika, Princeton Uni

prínosy:
matematika, informatika
1957 znovunavrhol algoritmus pre minimálnu kostru grafu
(1959, Dijkstra ho znovunavrhol nezávisle po tretíkrát)

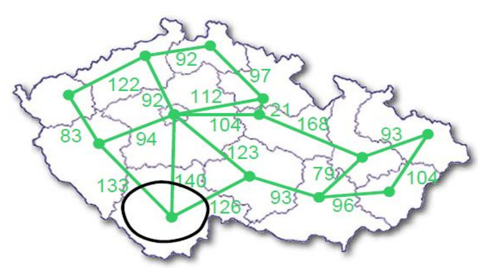
Jarníkov (- Primov – Dijkstrov) algoritmus

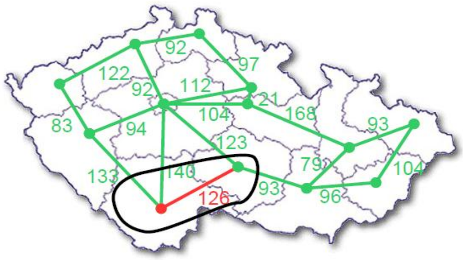
- Využíva vlastnosti rezu
- Pri tvorbe MST sa udržuje rez medzi spracovanými vrcholmi (časťami MST) a ešte nespracovanými vrcholmi
- 1. Vybrať ľubovoľný vrchol a označiť ho ako spracovaný
- 2. Z rezu vybrať minimálnu hranu e a vložiť ju do MST.
- 3. Nespracovaný vrchol hrany e označiť ako spracovaný.
- 4. Opakovať krok 2 dokiaľ nie sú spracované všetky vrcholy.

Jarníkov (- Primov – Dijkstrov) algoritmus

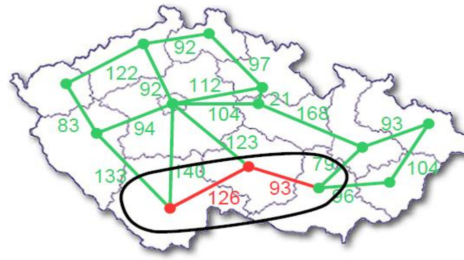
JPM-MST(G, W), G=(V, E)

T= {v}, v ∈ V //Vybrať ľubovoľný vrchol a označiť ho ako spracovaný
MST = { }
while T ≠ V do
    vyber hranu (u,v) s minimálnou váhou w takú, že u ∈ T, ¬ v ∈ T.
    vlož v do T a {u,v} do MST.

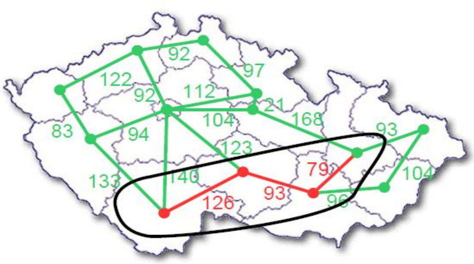




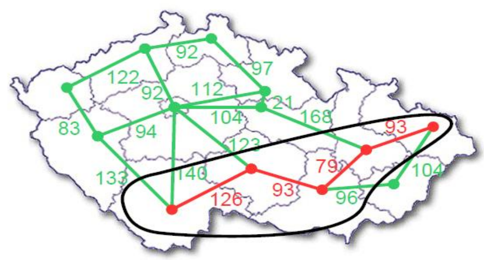
91



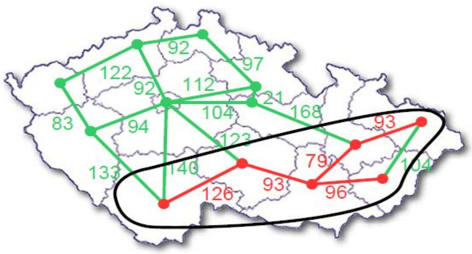
92



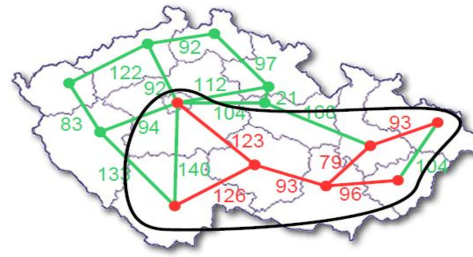
93



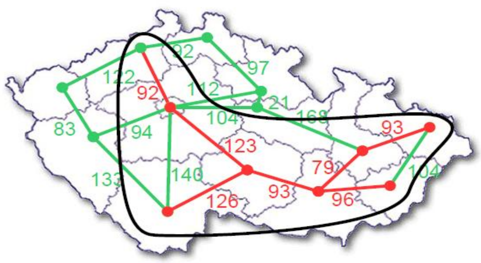
94



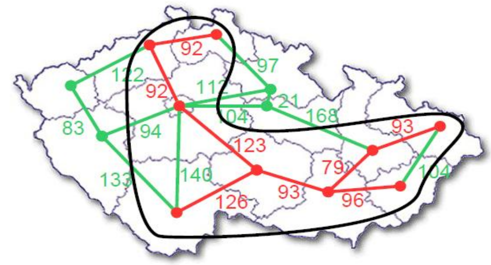
95



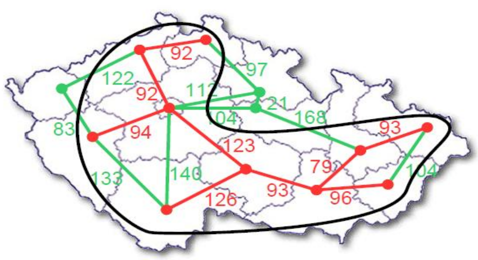
96



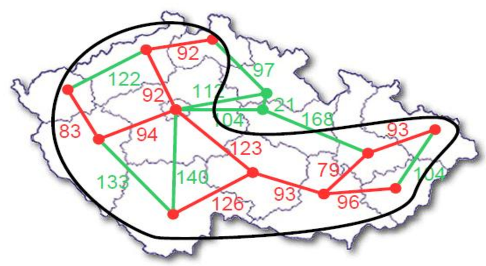
97



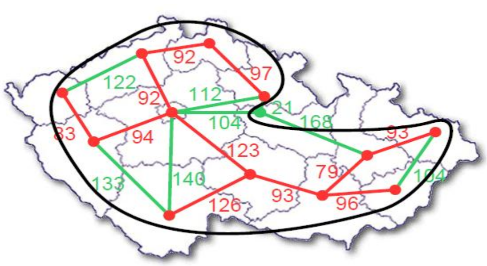
98



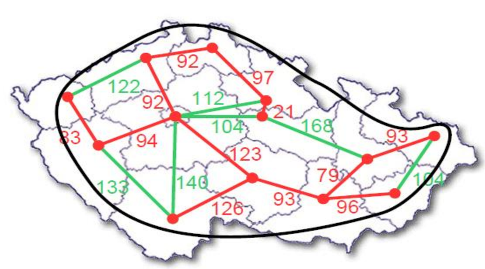
99



100



101



102

Jarníkov (- Primov – Dijkstrov) algoritmus

```
MST-Prim (G, w, r)
-- pre: G je súvislý graf
   --      w matica váh hrán
   --      r koreň
   -- post: A = {(v, v.Π) : v ∈ G.V - {r}}
for každý vrchol u ∈ G.V
  u.key ← ∞
  u.Π ← NIL
Q ← G.V                    -- Q je min-halda
r.key ← 0                  -- úprava hodnoty koreňa r v min-halde
while Q ≠ ∅
  u ← Extract-Min (Q)
  for každý v ∈ G.Adj[ u ]
    if v ∈ Q and w(u, v) < v.key
      v.Π ← u
      v.key ← w(u, v)     -- úprava min-haldy
```

103

Jarníkov (- Primov – Dijkstrov) algoritmus

```
MST-Prim (G, w, r)
-- pre: G je súvislý graf
   --      w matica váh hrán
   --      r koreň
   -- post: A = {(v, v.Π) : v ∈ G.V - {r}}
for každý vrchol u ∈ G.V                                O(|V|)
  u.key ← ∞
  u.Π ← NIL
Q ← G.V                    -- Q je min-halda            O(|V|)
r.key ← 0                  -- úprava hodnoty koreňa r v min-halde
while Q ≠ ∅                                             O(|V|)
  u ← Extract-Min (Q)                                   O(|V| lg |V|)
  for každý v ∈ G.Adj[ u ]                              O(|V|+ |E|)
    if v ∈ Q and w(u, v) < v.key                        O(|V|+ |E|)
      v.Π ← u
      v.key ← w(u, v)     -- úprava min-haldy           O((|V|+|E|)lg |V|)
                                                        alebo O(|V| lg |V|)
```

104

Vlastnosti Jarníkovho algoritmu

- Graf musí byť súvislý
- zložitosť
- bez ďalších vylepšení O(V^2) – pre hustý graf lineárna zložitosť
- použitím prioritného frontu sa dá znížiť na O(E log V)

105

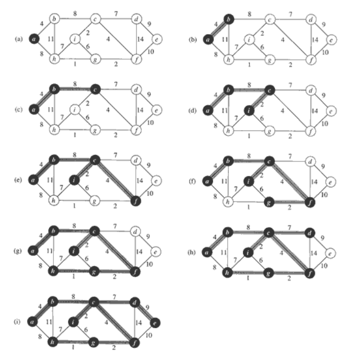


Figure 23.5 The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is *a*. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (*b*, *c*) or edge (*a*, *h*) to the tree since both are light edges crossing the cut.

106

poďakovanie

- Cormen, Leiserson, Rivest & Stein: Introduction to Algorithms Second Edition, McGraw Hill
- Petr Vaneček

107