STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

Grafové algoritmy

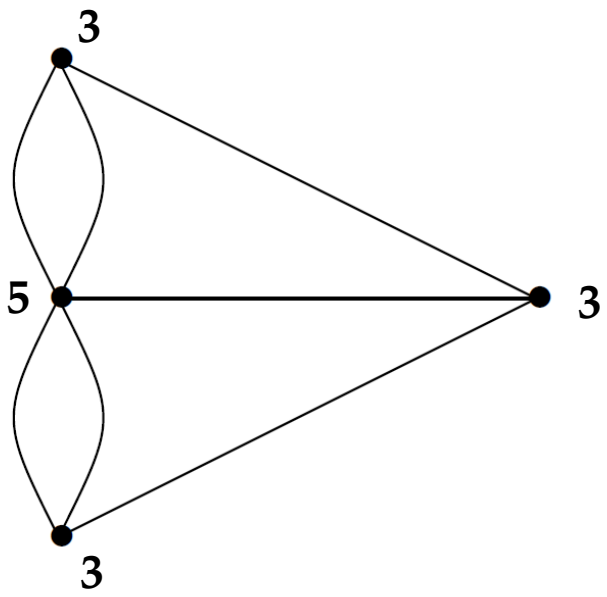
(cesty)

7. 11. 2017

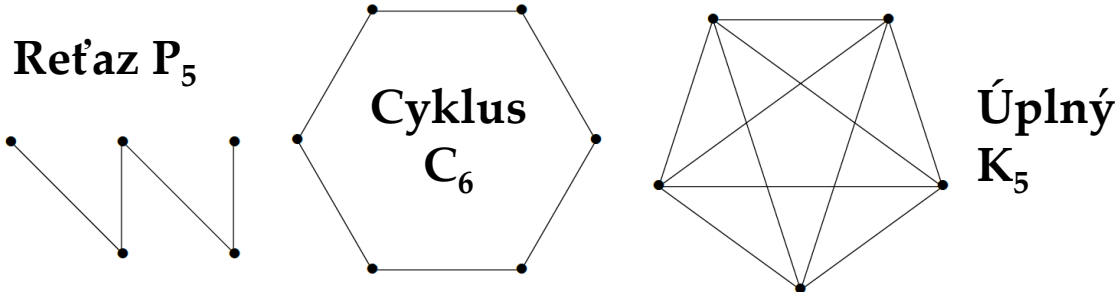
zimný semester 2017/2018

Opakovanie – Teória grafov

- G = (V – vrcholy, E – hrany)
- Kalingradské mosty
  Nájsť prechádzku mestom, ktorá prejde všetky mosty práve raz.
  - **Eulerov cyklus** – ak je počet hrán pri každom vrchole (stupeň) párny
  - **Eulerov ťah** – ak práve dva vrcholy majú nepárny stupeň

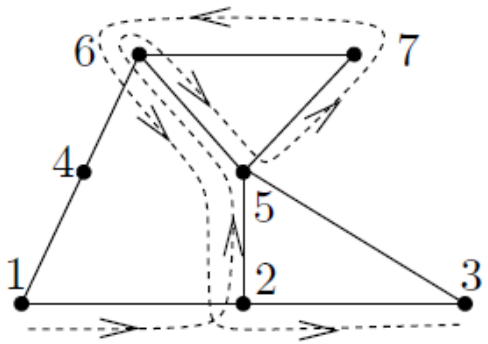


- Jednoduché grafy:

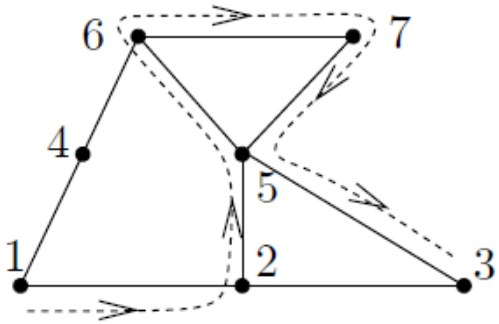


Opakovanie – sled, ťah, cesta

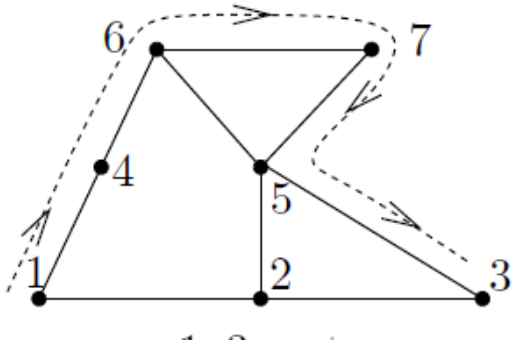
- **Sled** (walk) v grafe je ľubovoľná striedajúca sa postupnosť vrcholov a hrán grafu: $v_1,(v_1,v_2),v_2,(v_2,v_3),v_3, \ldots, (v_{k-1},v_k),v_k$
- **Ťah** (tour) je taký sled, v ktorom sa žiadna hrana neopakuje
- **Cesta** (path) je taký sled, v ktorom sa žiaden vrchol neopakuje



1–3 sled



1–3 ťah



1–3 cesta

1–3 sled: $(1, \{1,2\}, 2, \{2,5\}, 5,$
$\{5,6\}, 6, \{6,5\}, 5, \{5,7\}, 7, \{7,6\}, 6, \{6,5\}, 5, \{5,2\}, 2, \{2,3\}, 3)$.

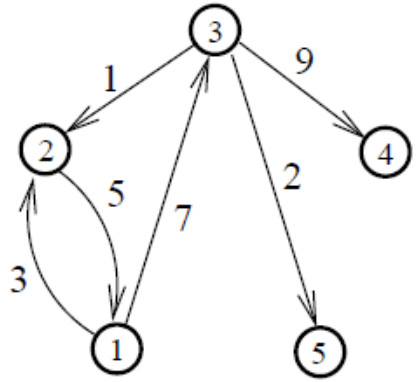
1–3 ťah: $(1, \{1,2\}, 2, \{2,5\}, 5, \{5,6\}, 6, \{6,7\}, 7, \{7,5\}, 5, \{5,3\}, 3)$.

1–3 cesta: $(1, \{1,4\}, 4, \{4,6\}, 6, \{6,7\}, 7, \{7,5\}, 5, \{5,3\}, 3)$.

Teória grafov – ohodnotený graf

- Hranám priradíme ohodnotenie, zobrazenie w
  - w je hranové ohodnotenie w: E $\rightarrow \mathbb{R}$

    w(e) = c (hovoríme, že hrana e má ohodnotenie c)
  - Graf G = (V,E,w) nazývame **ohodnotený graf** (weighted graph)
- Ohodnotenie nazývame aj
  - Váha (weight)
  - Cena (cost)
  - Dĺžka (length)
  - ...



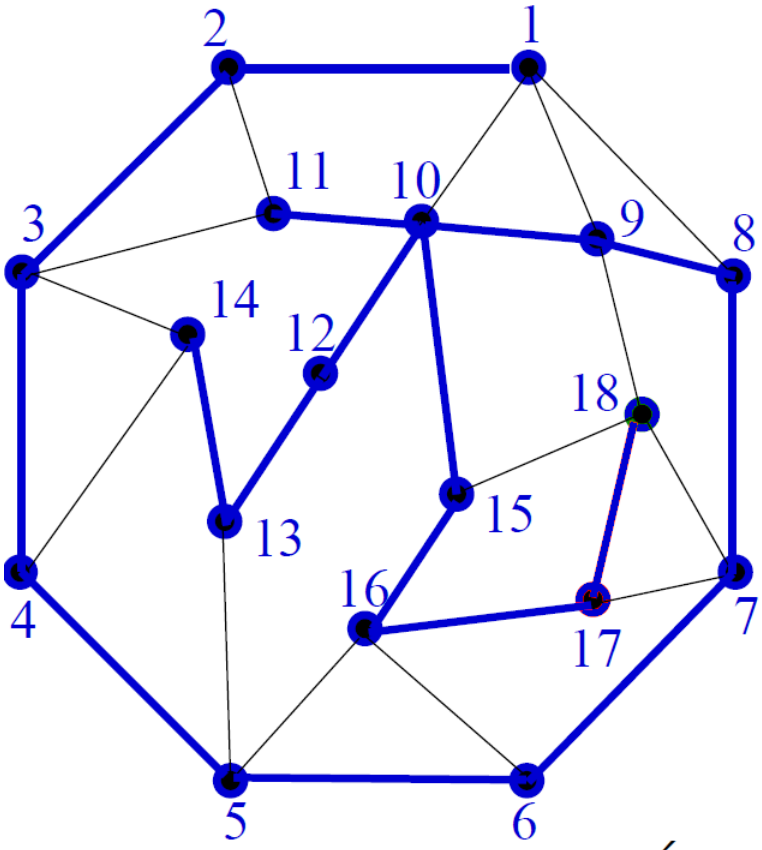
	1	2	3	4	5
1	-	3	7	-	-
2	5	-	-	-	-
3	-	1	-	9	2
4	-	-	-	-	-
5	-	-	-	-	-

- Rôzne typy ohodnotení
  - Vrcholové ohodnotenie (vertex weight)
  - Viacero ohodnotení zároveň (napr. dĺžka hrany a cena hrany)

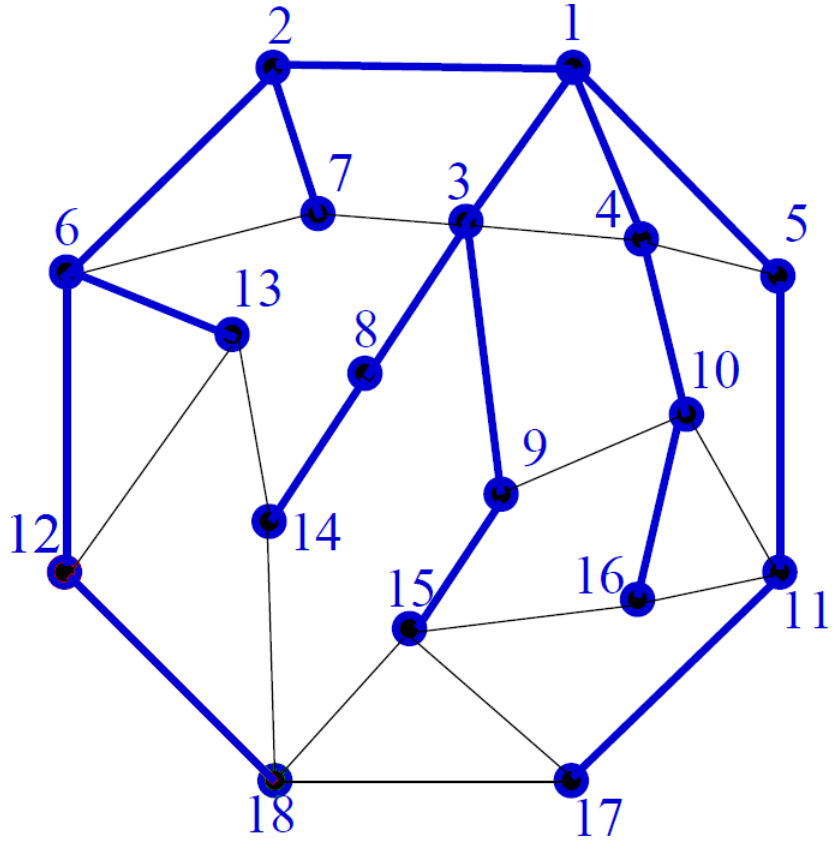
Teória grafov – najkratšia cesta

- Dĺžka sledu, ťahu, cesty v hranovo ohodnotenom grafe nazveme súčet ohodnotení jeho hrán
  - Ak máme sled, tak ohodnotenie každej hrany započítame toľkokrát, koľkokrát sa hrana v slede nachádza
  - Dĺžka sledu, ťahu, cesty ak je to len jeden vrchol je 0.
  - V prípade neohodnoteného grafu, je dĺžka počet hrán.
- **Najkratšia x-y cesta** v hranovo ohodnotenom grafe G je tá zo všetkých x-y ciest v G, ktorá má najmenšiu dĺžku
- Najkratšia x-y cesta v neohodnotenom grafe?
  - Najmenšia dĺžka (počet hrán) cesty z vrcholu x do vrcholu
  - Prehľadávanie do šírky

Najkratšia cesta v neohodnotenom grafe



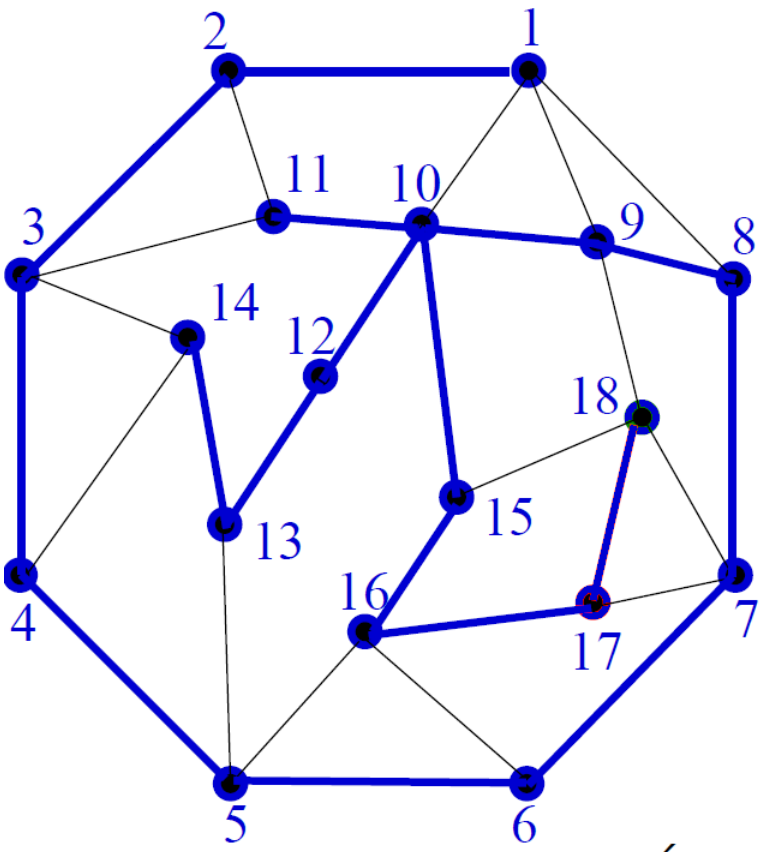
Prehľadávanie do hĺbky



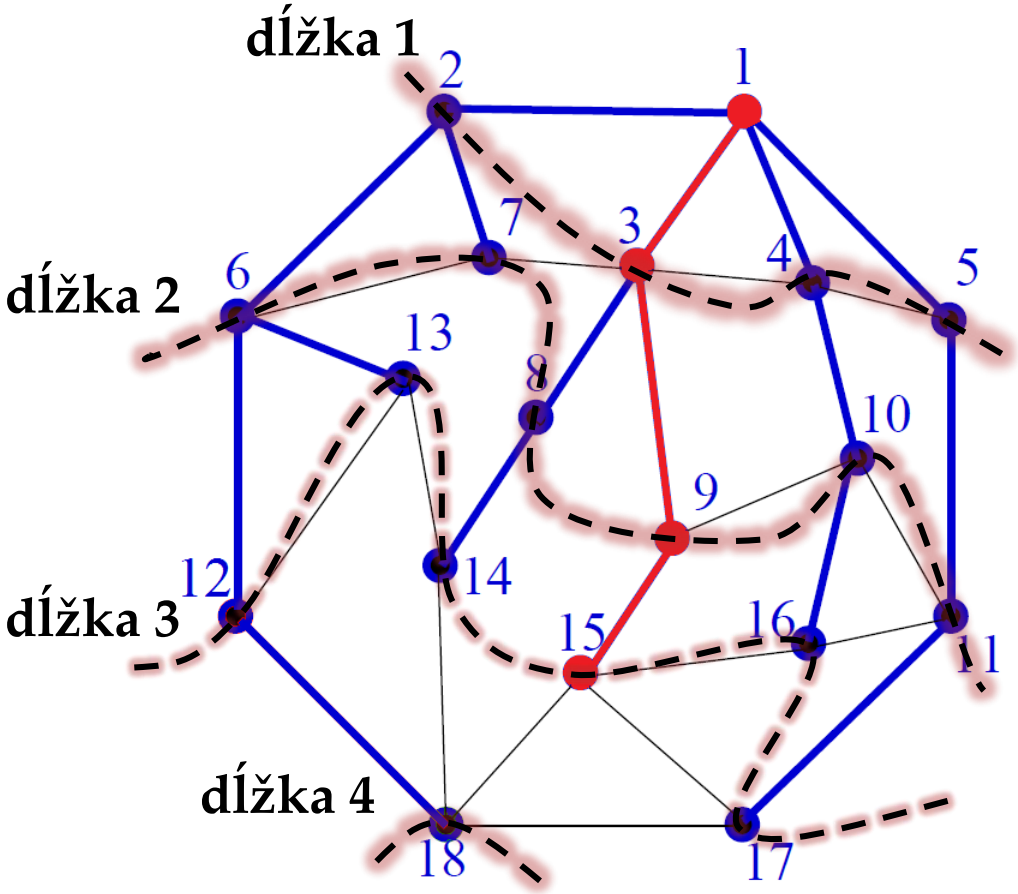
Prehľadávanie do šírky

Strom prehľadávania do šírky (z vrcholu x)
zodpovedá **stromu najkratších ciest** (z vrcholu x)

Najkratšia cesta v neohodnotenom grafe



Prehľadávanie do hĺbky



Prehľadávanie do šírky

Najkratšia cesta v ohodnotenom grafe

- Predpokladajme nezáporné ohodnotenia hrán $w(e) \geq 0$
- Počiatočný vrchol u
- Pre každý ďalší vrchol $i \in V$ nás zaujíma najkratšia cesta.
- Ako si budeme cestu pamätať?
  - **Stačí nám vo vrchole i uchovať predposledný (tzn. predchádzajúci) vrchol na u-i ceste.**
- Celú u-i cestu (postupnosť vrcholov a hrán z vrcholu u do vrcholu i) vieme potom ľahko rekonštruovať využitím predchádzajúceho vrcholu (predchádzajúci vrchol máme uchovaný pre každý vrchol z množiny V)

Najkratšia cesta v ohodnotenom grafe (2)

- Predpokladajme nezáporné ohodnotenia hrán $w(e) \geq 0$
- Počiatočný vrchol u
- Pre každý ďalší vrchol $i \in V$ nás zaujíma najkratšia cesta.
- Ako budeme určovať najkratšiu cestu do vrcholu i?
  - **Začneme s nejakou cestou, ktorú budeme postupne zlepšovať, až sa nebude dať zlepšiť, a bude to skutočne najkratšia u-i cesta.**
- Cesta, ktorú si pre vrchol i pamätáme, teda nemusí byť najkratšia u-i cesta, teda jej dĺžka je horné ohraničenie dĺžky (skutočne) najkratšej cesty.

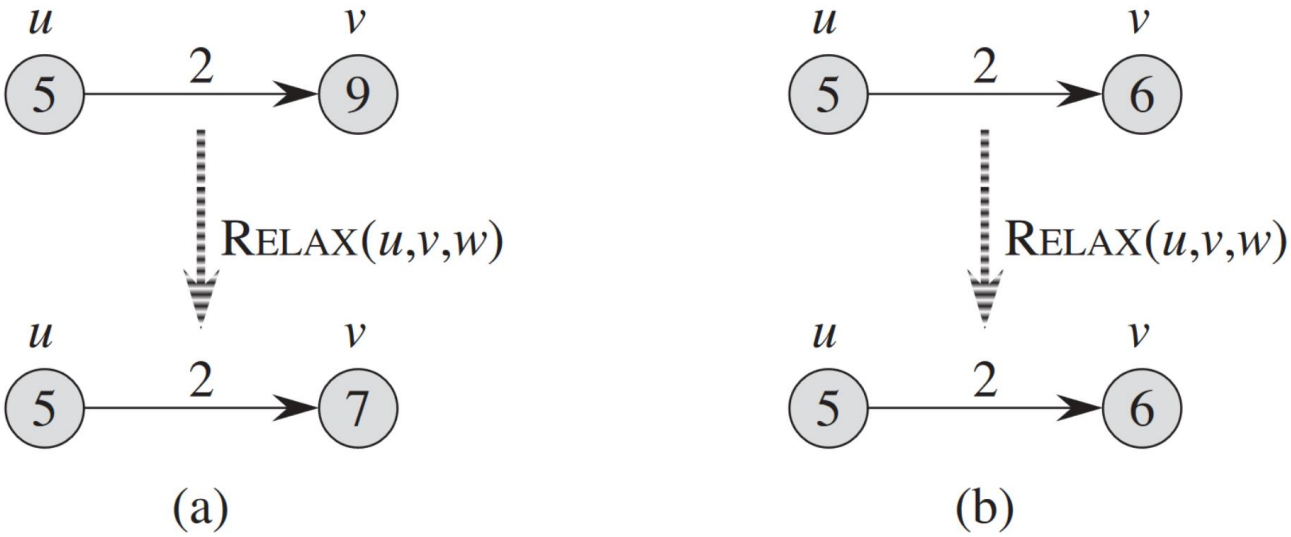
Najkratšia cesta v ohodnotenom grafe (3)

- Predpokladajme nezáporné ohodnotenia hrán $w(e) \geq 0$
- Počiatočný vrchol u
- Pre každý ďalší vrchol $i \in V$ si budeme udržiavať:
  - t(i) – horné ohraničenie na dĺžku (aktuálne) najkratšej u-i cesty
  - p(i) – predposledný vrchol v doteraz nájdenej najlepšej ceste (rodič v strome najkratších ciest)

- Algoritmus hľadania najkratšej cesty:
  opakované znižovanie (relaxácia) horného ohraničenia na dĺžku najkratšej cesty pre každý vrchol, až sa horné ohraničenie bude rovnať váhe najkratšej cesty

Relaxácia hrany

- Napr.



- Relaxácia hrany e=(u,v) s ohodnotením w(e)=2
  - Číslo vnútri vrcholu je horné ohraničenie dĺžky najkratšej cesty

a) Pred relaxáciou platí t(v) > t(u)+w(e), preto sa t(v) zmenší

b) Pred relaxáciou platí t(v) $\leq$ t(u)+w(e), preto sa t(v) nezmení

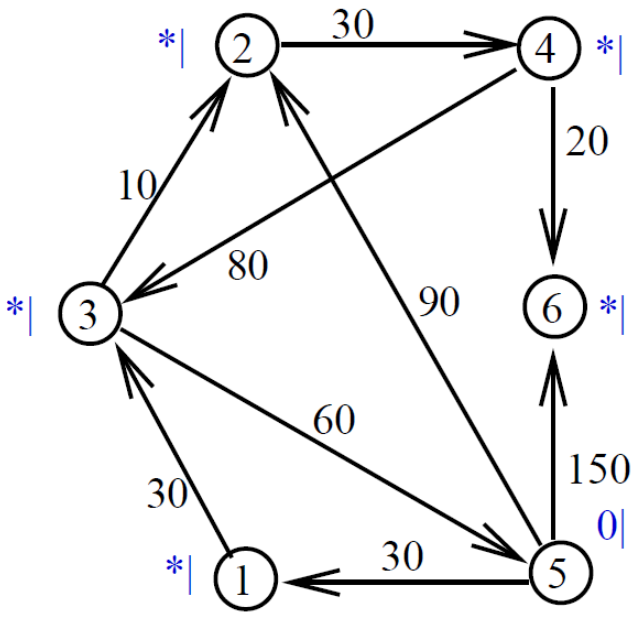
Základná schéma algoritmu

1. Inicializácia (počiatočný vrchol u)
   $t(u) = 0$, $t(v) = \infty$ pre $v \in V$ a $v \neq u$, a $p(v) = 0$
2. Ak existuje orientovaná hrana $e=(x,y) \in E$, pre ktorú platí: $t(y) > t(x)+w(e)$, tak
   - Relaxuj hranu $t(y) = t(x)+w(e)$
   - Zapamätaj predchodcu $p(y) = x$
   - Choď na krok 2.
3. Koniec, našli sme strom najkratších ciest z vrcholu u. (otázka na zamyslenie: je tento strom unikátny?)

- **Ak $t(i) = \infty$, tak vrchol i nie je dosiahnuteľný z vrcholu u, inak najkratšia u-i cesta má dĺžku t(i) a je opačná (spätná) k ceste:**
  - **i, p(i), p(p(i)), p(p(p(i))), …., u**

Ukážka – najkratšie cesty z vrcholu 5

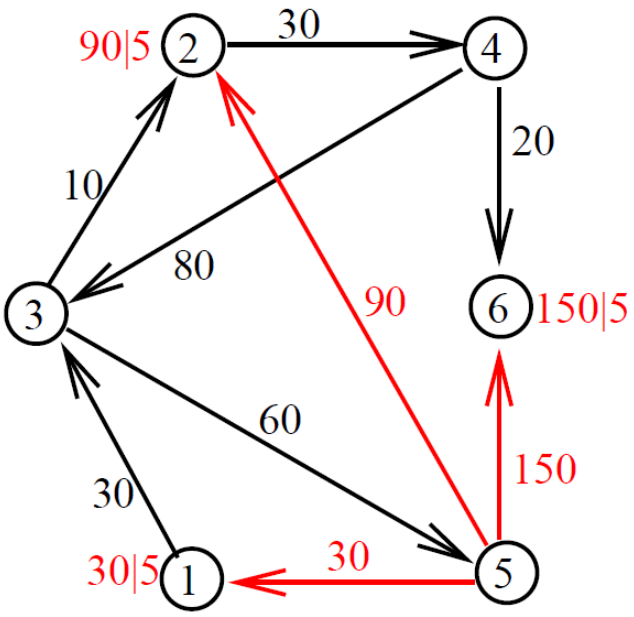
e	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$w(e)$	30	30	10	60	80	20	30	90	150



$e=(i,j)$	$t(i)$	$w(e)$	1	2	3	4	5	6
			$t(v) \mid p(v)$					
-			∞ |	∞ |	∞ |	∞ |	0 |	∞ |
$(5,1)$	0	30	30|5					
$(5,2)$	0	90		90|5				
$(5,6)$	0	150						150|5
$(1,3)$	30	30			60|1			
$(2,4)$	90	30				120|2		
$(3,2)$	60	10		70|3				
$(4,6)$	120	20						140|4
$(2,4)$	70	30				100|2		
$(4,6)$	100	20						120|4

Ukážka – najkratšie cesty z vrcholu 5

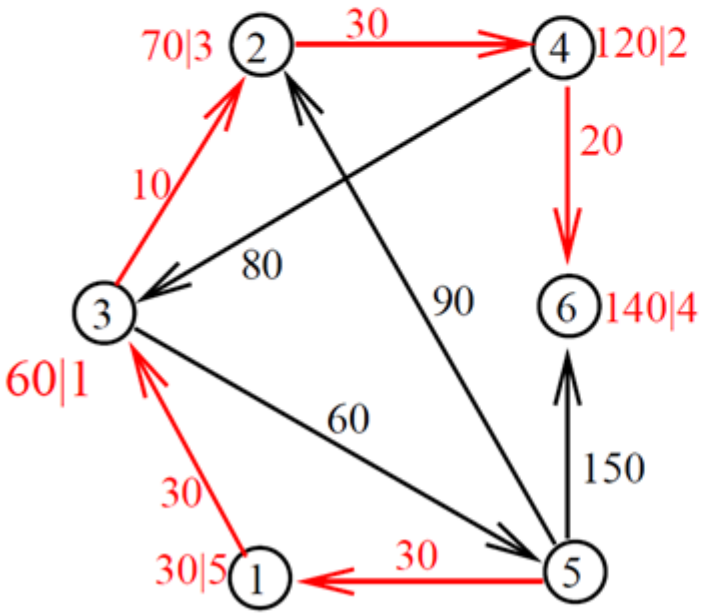
e	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$w(e)$	30	30	10	60	80	20	30	90	150



$e=(i,j)$	$t(i)$	$w(e)$	1	2	3	4	5	6
			$t(v) \mid p(v)$					
-			∞ |	∞ |	∞ |	∞ |	0 |	∞ |
$(5,1)$	0	30	30|5					
$(5,2)$	0	90		90|5				
$(5,6)$	0	150						150|5
$(1,3)$	30	30			60|1			
$(2,4)$	90	30				120|2		
$(3,2)$	60	10		70|3				
$(4,6)$	120	20						140|4
$(2,4)$	70	30				100|2		
$(4,6)$	100	20						120|4

Ukážka – najkratšie cesty z vrcholu 5

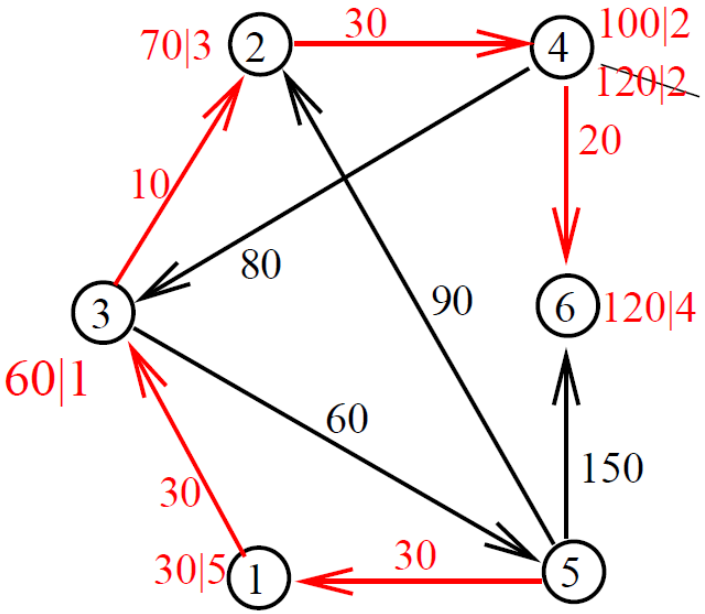
e	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$w(e)$	30	30	10	60	80	20	30	90	150



$e=(i,j)$	$t(i)$	$w(e)$	1	2	3	4	5	6
			$t(v) \mid p(v)$					
-			$\infty\mid$	$\infty\mid$	$\infty\mid$	$\infty\mid$	$0\mid$	$\infty\mid$
$(5,1)$	0	30	$30\mid5$					
$(5,2)$	0	90		$90\mid5$				
$(5,6)$	0	150						$150\mid5$
$(1,3)$	30	30			$60\mid1$			
$(2,4)$	90	30				$120\mid2$		
$(3,2)$	60	10		$70\mid3$				
$(4,6)$	120	20						$140\mid4$
$(2,4)$	70	30				$100\mid2$		
$(4,6)$	100	20						$120\mid4$

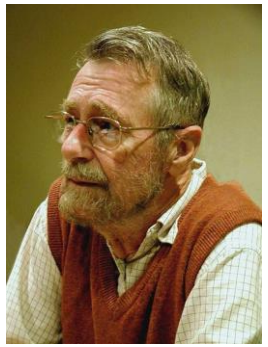
Ukážka – najkratšie cesty z vrcholu 5

e	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$w(e)$	30	30	10	60	80	20	30	90	150



$e=(i,j)$	$t(i)$	$w(e)$	1	2	3	4	5	6
			$t(v) \mid p(v)$					
-			∞ |	∞ |	∞ |	∞ |	0 |	∞ |
$(5,1)$	0	30	30|5					
$(5,2)$	0	90		90|5				
$(5,6)$	0	150						150|5
$(1,3)$	30	30			60|1			
$(2,4)$	90	30				120|2		
$(3,2)$	60	10		70|3				
$(4,6)$	120	20						140|4
$(2,4)$	70	30				100|2		
$(4,6)$	100	20						120|4

Dijkstrov algoritmus



- Najkratšia u-v cesta v orientovanom grafe
- Predpoklad: nezáporné váhy hrán $w(e) \geq 0$
- Inicializácia a relaxácia
- Udržiavame množinu vrcholov $s \in S$, ktorých dĺžky najkratších u-s ciest sú už definitívne určené
- Opakovane vyberieme vrchol $x \in V - S$ s **najmenším** horným ohraničením dĺžky najkratšej cesty:
  - Relaxujeme všetky hrany vychádzajúce z x
  - Vrchol x pridáme do S

Dijkstrov algoritmus – implementácia

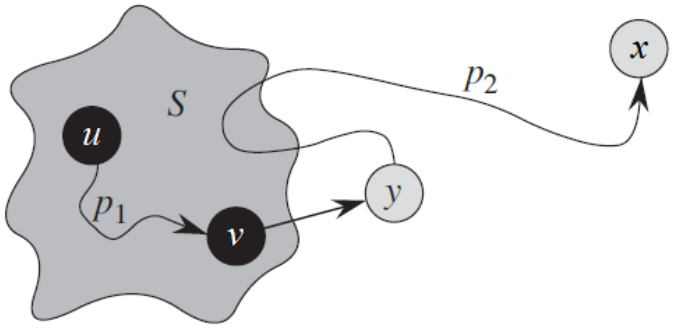
- Najkratšie cesty z u = strom najkratších ciest z u
- Predpoklad: nezáporné váhy hrán w(e) ≥ 0

```
Dijkstra(G, w, u)
1 Inicializuj-počiatok(G, u)
2 S ← ∅
3 PQ ← V(G)
4 while PQ ≠ ∅
5  do x ← Extrakt-Min(PQ)
6       S ← S ∪ {x}
7       for každý vrchol y ∈ Neigh(x)
8            do Relax(x, y, w)
```

PQ – min-prioritný rad

Dijkstrov algoritmus – dôkaz správnosti

- Vždy keď zaraďujeme vrchol x do množiny S (riadok 6), platí, že t(x) je dĺžka najkratšej u-x cesty.
- Predpokladajme, že to neplatí:
  - Nech je x je prvý taký vrchol, že to neplatí (dĺžka t(x) nie je dĺžka najkratšej u-x cesty)
  - Existuje teda nejaká kratšia u-x cesta, a na tejto ceste nájdime prvý vrchol y taký, že $y \notin S$
  - Keďže to je kratšia cesta a platí $w(p_2) \geq 0$ tak $t(y) < t(x)$
  - Tiež platí $t(x) \leq t(y)$ pretože sme vybrali x skôr na pridanie do S.
  - Spor.



Dijkstrov algoritmus – modifikácia

- Najkratšia u-v cesta v orientovanom grafe
- Predpoklad: nezáporné váhy hrán w(e) ≥ 0

```
Dijkstra(G, w, u, v)
1 Inicializuj-počiatok(G, u)
2 S ← ∅
3 PQ ← V(G)
4 while PQ ≠ ∅
5  do x ← Extrakt-Min(PQ)
6     if x = v then STOP.
7     S ← S ∪ {x}
8     for každý vrchol y ∈ Neigh(x)
9          do Relax(x, y, w)
```

PQ – min-prioritný rad

Dijkstrov algoritmus – odhad zložitosti

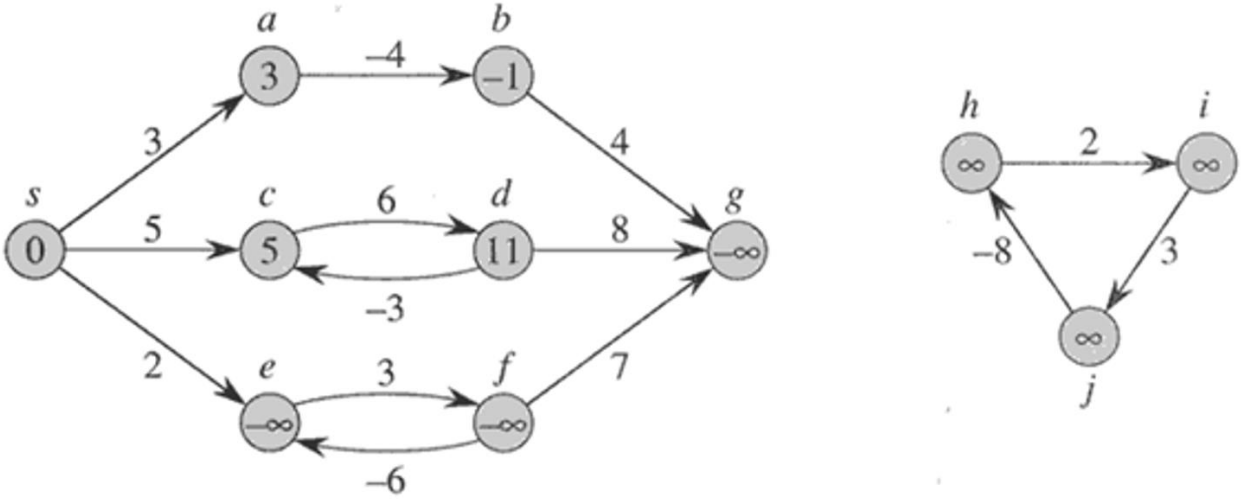
- Počet vrcholov N=|V|, počet hrán M=|E|
- Q vo vektore:
  - Extract-Min O(N), opakuje sa N krát, spolu $O(N^2)$
  - každý vrchol sa vkladá do S práve raz.
  - každá hrana sa relaxuje (v jednom smere) práve raz
  - $O(N^2 + M) = O(N^2)$
- Q v binárnej halde:
  - Extract-Min O(log N), opakuje sa N krát
  - vytvorenie binárnej haldy O(N)
  - relaxovanie sa zrealizuje pomocou operácie Decrease-Key O(log N)
  - stále je M opakovaní
  - $O((N + M) * \log N) = O(M * \log N)$
    ak sú všetky vrcholy dosiahnuteľné z východiska

Teória grafov – vzdialenosť

- **Vzdialenosť vrcholov** u a v
  je dĺžka d(u,v) najkratšej u-v cesty
- **Excentricita vrcholu** x:
  $e(x) = \max \{ d(x,y) \mid y \in V \}$
- **Polomer (rádius) grafu** G=(V,E):
  $r(G) = \min \{ e(x) \mid x \in V \}$
- **Priemer (diameter) grafu** G=(V,E):
  $d(G) = \max \{ e(x) \mid x \in V \}$
  (platí $d(G) = \max \{ d(x,y) \mid x,y \in V \}$)
- **Centrum grafu** G=(V,E) – vrcholy $v \in V$ také, že:
  $e(v) = r(G)$

Čo keď sú ohodnotenia záporné?

- Napr.



- Počiatočný vrchol s
- Vnútri vrcholov je vpísaná dĺžka najkratšieho sledu
  - V prípade ak v grafe nie je záporný cyklus, tak dĺžky najkratších sledov sú konečné, a rovnaké ako dĺžka najkratšej cesty
  - Ak môže byť záporný cyklus, má zmysel uvažovať najkratšiu dĺžku cesty (sledu, v ktorom sa neopakujú vrcholy)?

    Určite má, ale zatiaľ ľudstvo nepozná efektívny algoritmus ako to riešiť, viac neskôr...

Bellman-Fordov algoritmus

- Ohodnotenia hrán môžu byť záporné
- Algoritmus zistí, ak existuje v grafe záporný cyklus dosiahnuteľný z počiatočného vrcholu, inak (ak neexistuje) nájde strom a dĺžky najkratších ciest z počiatočného vrcholu

1. Inicializácia
2. Relaxácia: N-1 prechodov cez všetky hrany grafu
3. Test na záporné cykly

   (ešte jeden pokus o relaxáciu; ak sa podarí, znamená to, že existuje záporný cyklus)

(dôkaz správnosti ako cvičenie: ukázať, že alebo nájde najkratšie cesty alebo vyhlási, že graf obsahuje záporný cyklus)

Bellman-Fordov algoritmus – implementácia

```
Bellman-Ford(G, w, s)
1 Inicializuj-počiatok(G, s)
2 for i ← 1 to N - 1
3   do for každú hranu e=(u,v) ∈ E
4         do Relax(u, v, w)
5 for každú hranu e=(u,v) ∈ E
6   do if t[v] > t[u]+w(e)
7         then return CONTAINS_NEGATIVE_CYCLE
8 return OK
```

- inicializácia (riadok 1) potrebuje O(N)
- každý z N - 1 prechodov (riadky 2–4) potrebuje O(M)
- záverečný test (riadky 5–8) potrebuje O(M).
- celkovo O(N*M)

Najkratšie cesty z každého vrcholu do každého

- Ak graf neobsahuje záporné hrany
  - Spustiť z každého vrcholu Dijkstrov algoritmus
    $N*O(N^2) = O(N^3)$ ak použijeme vektor
    $N*O(M \log N) = O(N*M \log N)$ ak použijeme min-haldu
- Ak obsahuje záporné hrany
  - Spustiť (pomalší) Bellman-Fordov algoritmus z každého vrcholu
    $N*O(N*M) = O(N^2*M)$
    na hustých grafoch to je $O(N^4)$

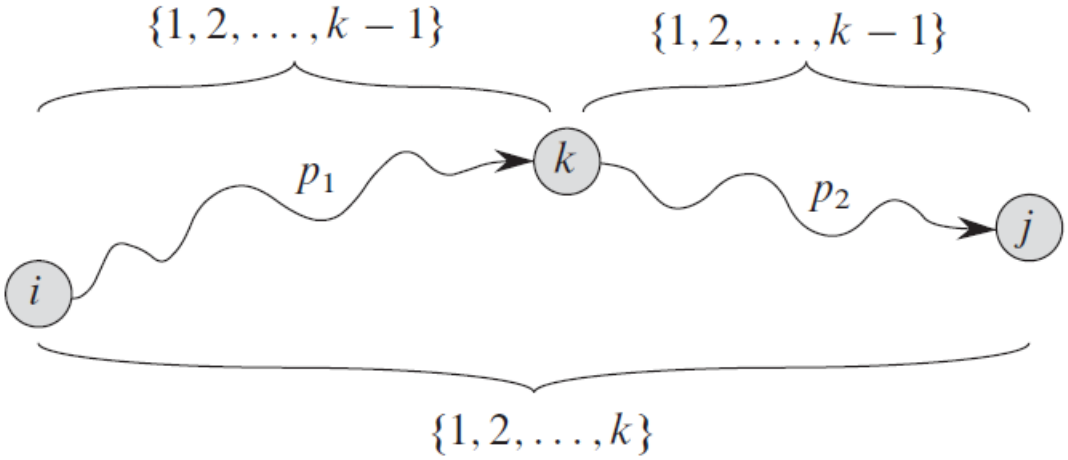
- Chceme to lepšie – $O(N^3)$ aj pre záporné hrany
  - Samozrejme chceme aj detekciu záporného cyklu :)

Floydov algoritmus

- Vzdialenosť (dĺžka najkratšej cesty)
  z každého vrcholu do všetkých ostatných
- Uvažujeme cez ktoré vrcholy ide najkratšia cesta
- **V k-tom kroku: zaujímajú nás najkratšie cesty medzi každou dvojicou vrcholov i,j také že cesta ide po vrcholoch z množiny {1, 2, …, k}**
- Algoritmus postupuje pre k=1, 2, ..., N nakoniec teda budú určené najkratšie cesty medzi každou dvojicou vrcholov také, že idú cez všetky vrcholy {1, …, k=N }

Floydov algoritmus

- Dĺžky najkratších ciest v kroku k určíme induktívne z dĺžok najkratších ciest z kroku k-1 nasledovne:



- Ak vrchol k **nie je** na najkratšej ceste idúcej cez vrcholy {1,...,k}, tak sa použije najkratšia cesta idúca cez {1,...,k-1}
- Ak vrchol k **je** na najkratšej ceste idúcej cez {1,...,k}, a podcesty p_1 a p_2 už môžu obsahovať len {1,...,k-1}.

Floydov algoritmus – rekurzívny zápis

- Označme $d_{ij}^{(k)}$ dĺžku najkratšej cesty z vrcholu i do j takej, že ide len cez vrcholy množiny {1, …, k}

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{ak } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{ak } k \geq 1\,. \end{cases}$$

- Označme $\pi_{ij}^{(k)}$ predposledný vrchol najkratšej cesty z i do j (cez vrcholy z {1,…,k}) ak existuje.

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{ak } i = j \text{ alebo } w_{ij} = \infty\,, \\ i & \text{ak } i \neq j \text{ a } w_{ij} < \infty\,. \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{ak } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\,, \\ \pi_{kj}^{(k-1)} & \text{ak } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\,. \end{cases}$$

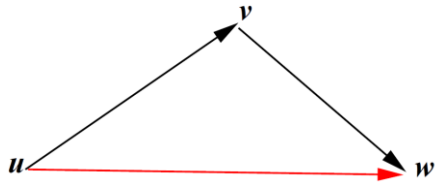
Floydov algoritmus – implementácia

```
Floyd (W)
1 d(0) = W
2 for k ← 1 to N
3   for i ← 1 to N
4     for j ← 1 to N
5       d(k)ij=min(d(k-1)ij, d(k-1)ik+ d(k-1)kj)
6 return d(n)
```

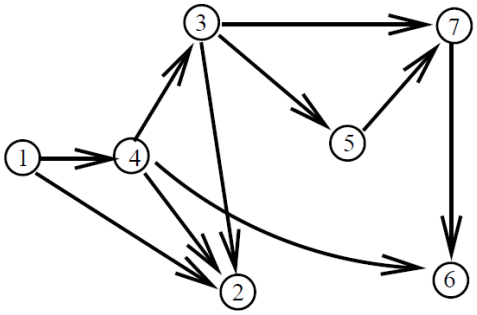
- Zložitosť: $O(N^3)$
  - Pamäťová zložitosť N matíc NxN (cvičenie je to potrebné?)
- Detekcia záporných cyklov
  - Inicializovať $d(0)_{ii} = \infty$
  - Na koniec bude $d(n)_{ii}$ dĺžka najkratšieho i-i cyklu
- Warshallov algoritmus:
  jednoduchý variant na určenie tranzitívneho uzáveru grafu

Teória grafov – tranzitívny uzáver / redukcia

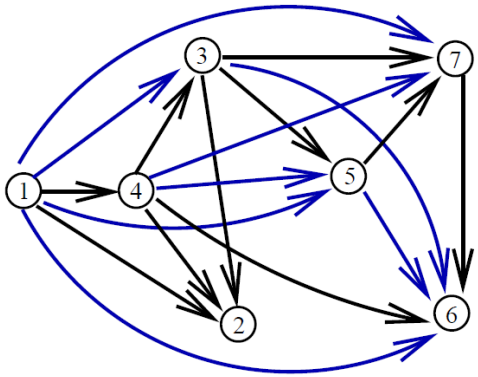
- Digraf je **tranzitívny**, ak platí
  (u,v),(v,w) ∈ E ⇒ (u,w) ∈ E



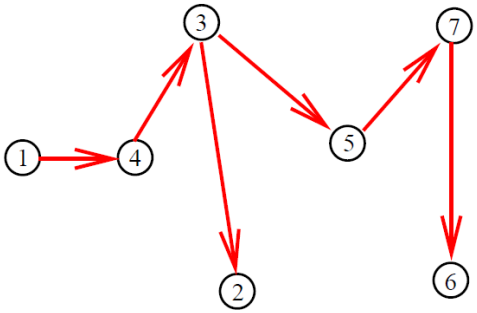
- **Tranzitívny uzáver grafu** G je minimálny tranzitívny digraf obsahujúci ako podgraf graf G
- **Tranzitívna redukcia grafu** G je minimálny podgraf so všetkými vrcholmi, s rovnakou dosiahnuteľnosťou ako G



a) Digraf

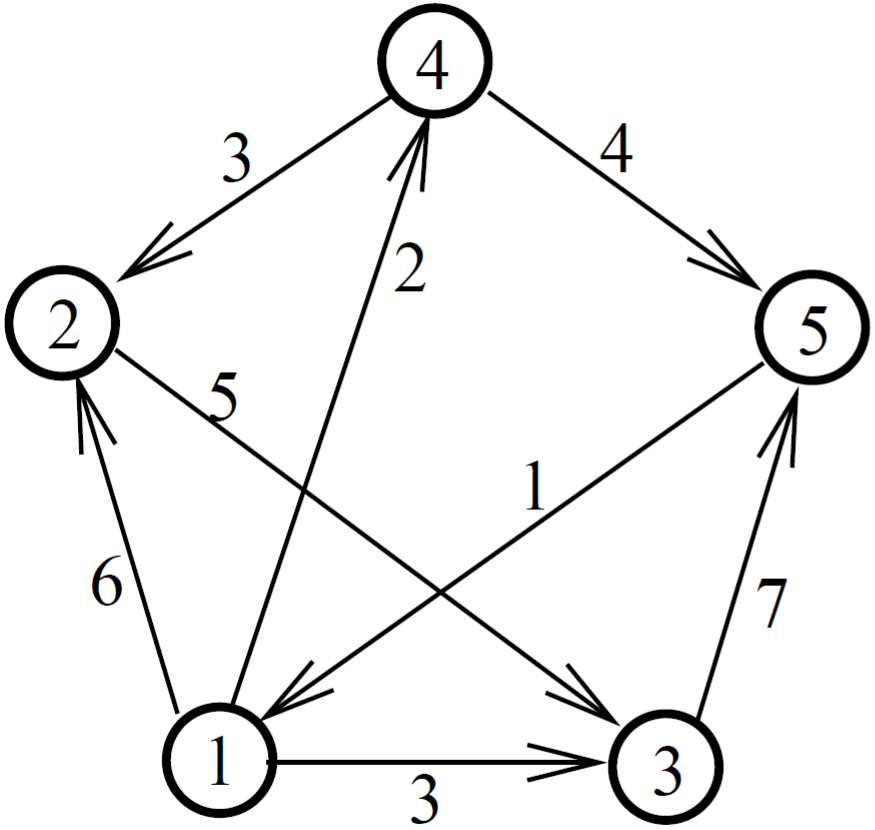


b) Tranzitívny uzáver



c) Tranzitívna redukcia

Floydov algoritmus – ukážka



$d_{ij}^{(0)}$	1	2	3	4	5
1	0	6	3	2	∞
2	∞	0	5	∞	∞
3	∞	∞	0	∞	7
4	∞	3	∞	0	4
5	1	∞	∞	∞	0

$\pi_{ij}^{(0)}$	1	2	3	4	5
1	1	1	1	1	∞
2	∞	2	2	∞	∞
3	∞	∞	3	∞	3
4	∞	4	∞	4	4
5	5	∞	∞	∞	5

Floydov algoritmus – ukážka

	1	2	3	4	5
1	**0**	**6**	**3**	**2**	**∞**
2	**∞**	0	5	∞	∞
3	**∞**	∞	0	∞	7
4	**∞**	3	∞	0	4
5	**1**	∞	∞	∞	0

	1	2	3	4	5
1	**1**	**1**	**1**	**1**	**∞**
2	**∞**	2	2	∞	∞
3	**∞**	∞	3	∞	3
4	**∞**	4	∞	4	4
5	**5**	∞	∞	∞	5

$k = 1$

	1	2	3	4	5
1	0	**6**	3	2	∞
2	**∞**	**0**	**5**	**∞**	**∞**
3	∞	**∞**	0	∞	7
4	∞	**3**	∞	0	4
5	1	**7**	4	3	0

	1	2	3	4	5
1	1	**1**	1	1	∞
2	**∞**	**2**	**2**	**∞**	**∞**
3	∞	**∞**	3	∞	3
4	∞	**4**	∞	4	4
5	5	**1**	1	1	5

$k = 2$

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

	1	2	3	4	5
1	1	1	**1**	1	∞
2	∞	2	**2**	∞	∞
3	**∞**	**∞**	**3**	**∞**	**3**
4	∞	4	**2**	4	4
5	5	1	**1**	1	5

Floydov algoritmus – ukážka

$k = 3$

	1	2	3	4	5
1	0	6	3	2	10
2	∞	0	5	∞	12
3	∞	∞	0	∞	7
4	∞	3	8	0	4
5	1	7	4	3	0

$k = 4$

	1	2	3	4	5
1	0	5	3	2	6
2	∞	0	5	∞	12
3	∞	∞	0	∞	7
4	∞	3	8	0	4
5	1	6	4	3	0

$k = 5$

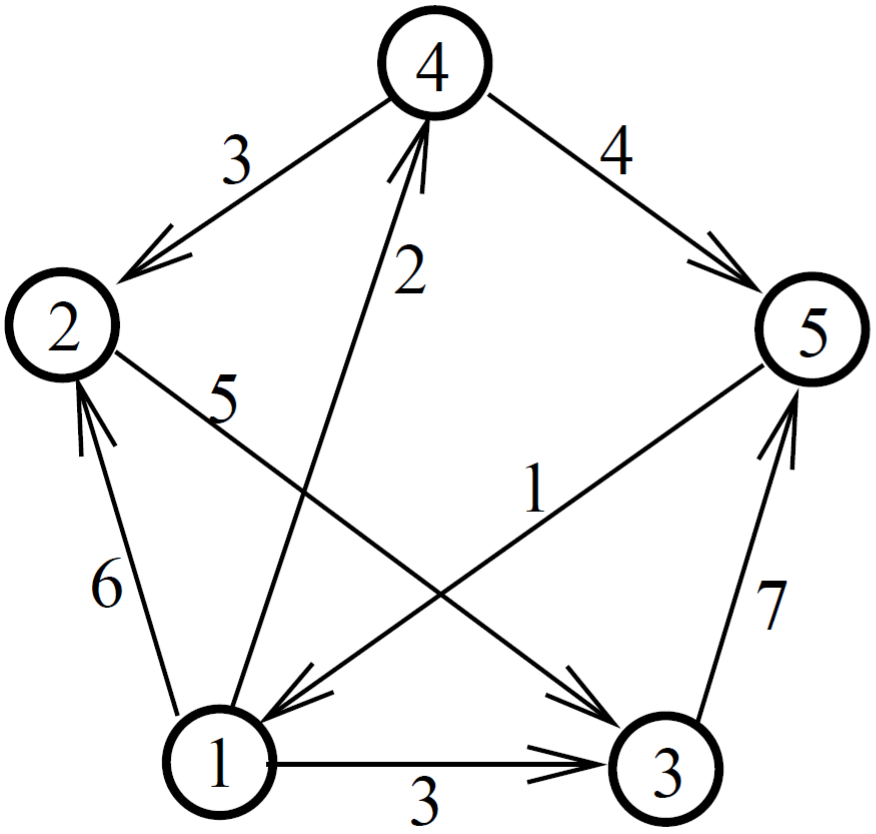
	1	2	3	4	5
1	0	5	3	2	6
2	13	0	5	15	12
3	6	13	0	10	7
4	5	3	8	0	4
5	1	6	4	3	0

	1	2	3	4	5
1	1	1	1	1	3
2	∞	2	2	∞	3
3	∞	∞	3	∞	3
4	∞	4	2	4	4
5	5	1	1	1	5

	1	2	3	4	5
1	1	4	1	1	4
2	∞	2	2	∞	3
3	∞	∞	3	∞	3
4	∞	4	4	4	4
5	5	4	1	1	5

	1	2	3	4	5
1	1	4	1	1	4
2	5	2	2	1	3
3	5	4	3	1	3
4	5	4	2	4	4
5	5	4	1	1	5

Floydov algoritmus – ukážka



$k = 5$

$d_{ij}^{(k)}$	1	2	3	4	5
1	0	5	3	2	6
2	13	0	5	15	12
3	6	13	0	**10**	7
4	5	3	8	0	4
5	1	6	4	3	0

$\pi_{ij}^{(k)}$	1	2	3	4	5
1	1	4	1	1	4
2	5	2	2	1	3
3	**5**	4	3	**1**	**3**
4	5	4	2	4	4
5	5	4	1	1	5

Najkratšia 3-4 cesta je $(\mathbf{3}, (3,5), \mathbf{5}, (5,1), \mathbf{1}, (1,4), \mathbf{4})$ a má dĺžku **10**.

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

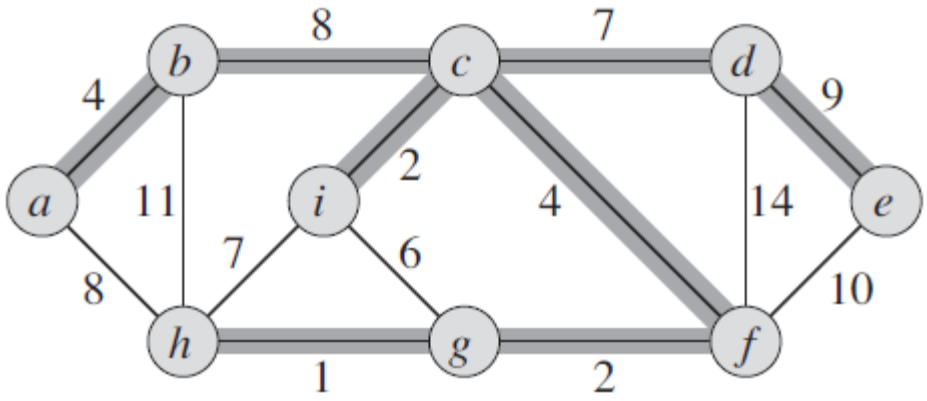
Grafové algoritmy
(kostry)

7. 11. 2017

zimný semester 2017/2018

Teória grafov – kostra

- Kostra (spanning tree) – strom, ktorý obsahuje všetky vrcholy grafu
  - Súvislý podgraf bez cyklov
- **Cena (váha) kostry** je súčet ohodnotení jej hrán
- **Najlacnejšia kostra** (minimum spanning tree) je kostra s najmenšou cenou



- Je táto kostra jediná najlacnejšia?

Všeobecná schéma

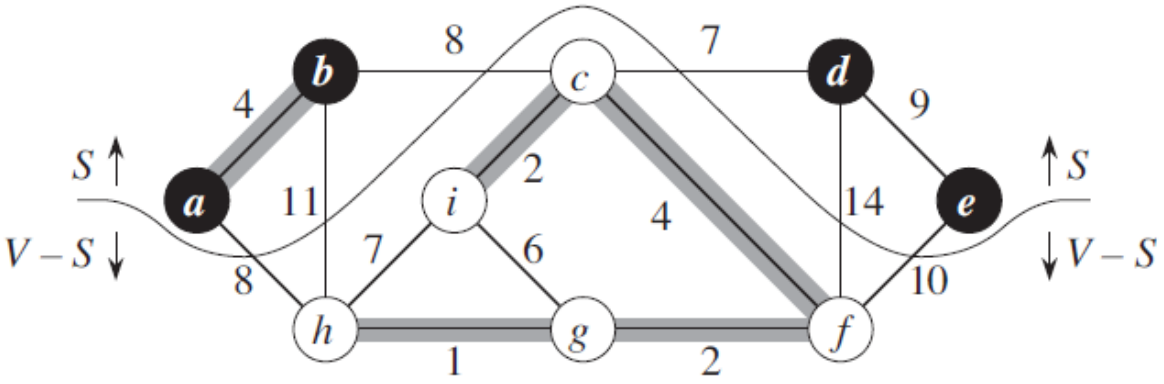
- **Pažravý (greedy) algoritmus**, ktorý pridáva hrany do kostry, až pokým nie je hotová:

```
MST-všeobecný(G, w)
1 A ← ∅
2 while A netvorí kostru
3   do nájdi hranu (u, v) takú,
              že podgraf A ∪ {(u, v)} je
              podgrafom nejakej minimálnej kostry G
4      A ← A ∪ {(u, v)}
5 return A
```

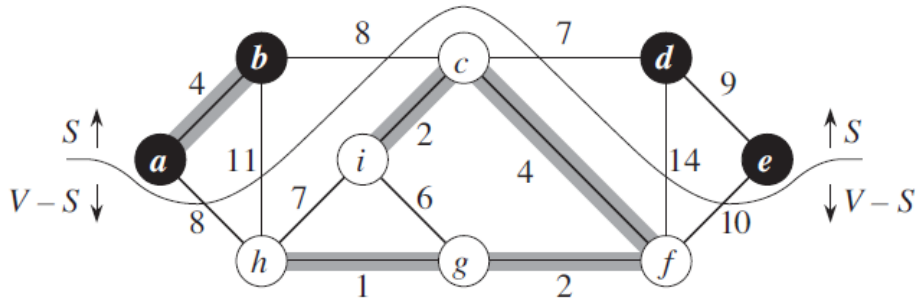
- **V nasledujúcej časti budeme riešiť, ako vybrať takúto hranu**

Všeobecná schéme

- **Rez grafu** sa nazýva množina prvkov súvislého grafu, po odstránení ktorých sa graf rozpadne na dva komponenty (S a V-S) a žiadna podmnožina rezu nemá túto vlastnosť. (ak je rez vrchol = artikulácia, ak hrana = most)
- Rez **rešpektuje** množinu hrán A, ak žiadna hrana z A nepretína rez. **Ľahká hrana** je hrana, ktorá pretína rez, a má najmenšiu váhu spomedzi takých čo pretínajú rez.

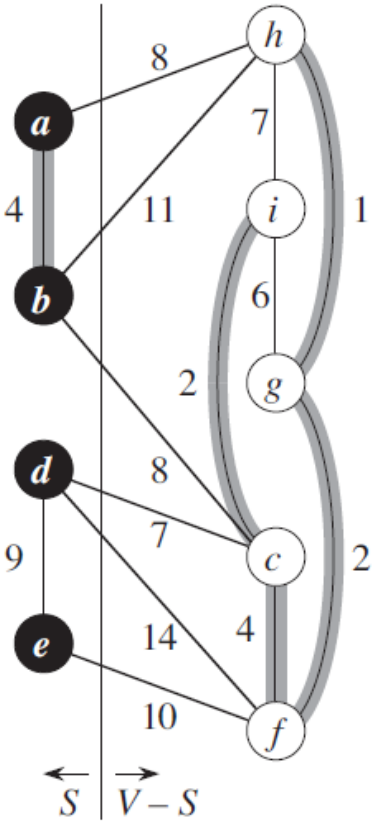


Všeobecná schéme



- Ľahké hrany (medzi S a V-S): (c,d) – cena 7.
- Podmnožina hrán A je šedá.
- Hociktorú z ľahkých hrán môžeme pridať (bude v nejakej min. kostre).

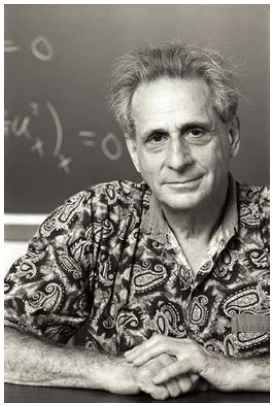
Inak nakreslené: hrany rezu idú len medzi S a V-S



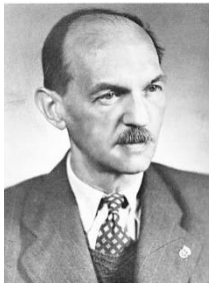
Algoritmy na výpočet minimálnej kostry grafu

- Rôznym spôsobom určujú, ktorú hranu vyberú a pridajú do „vytváranej kostry“
- **Kruskalov algoritmus (1956)**
  - Množina A je les, ktorého vrcholy sú všetky vrcholy grafu G
  - V jednom kroku vyberie najlacnejšiu hranu, spomedzi hrán spájajúcich rôzne komponenty
- **Primov algoritmus (1957)**
  - Množina A je jeden strom
  - V jednom kroku vyberie najlacnejšiu hranu spomedzi hrán spájajúcich vrchol v strome s vrcholom, ktorý nie je v strome
  - Objavil ho už Vojtěch Jarník v roku 1930! **Jarníkov algoritmus**

Martin Kruskal



Robert Prim

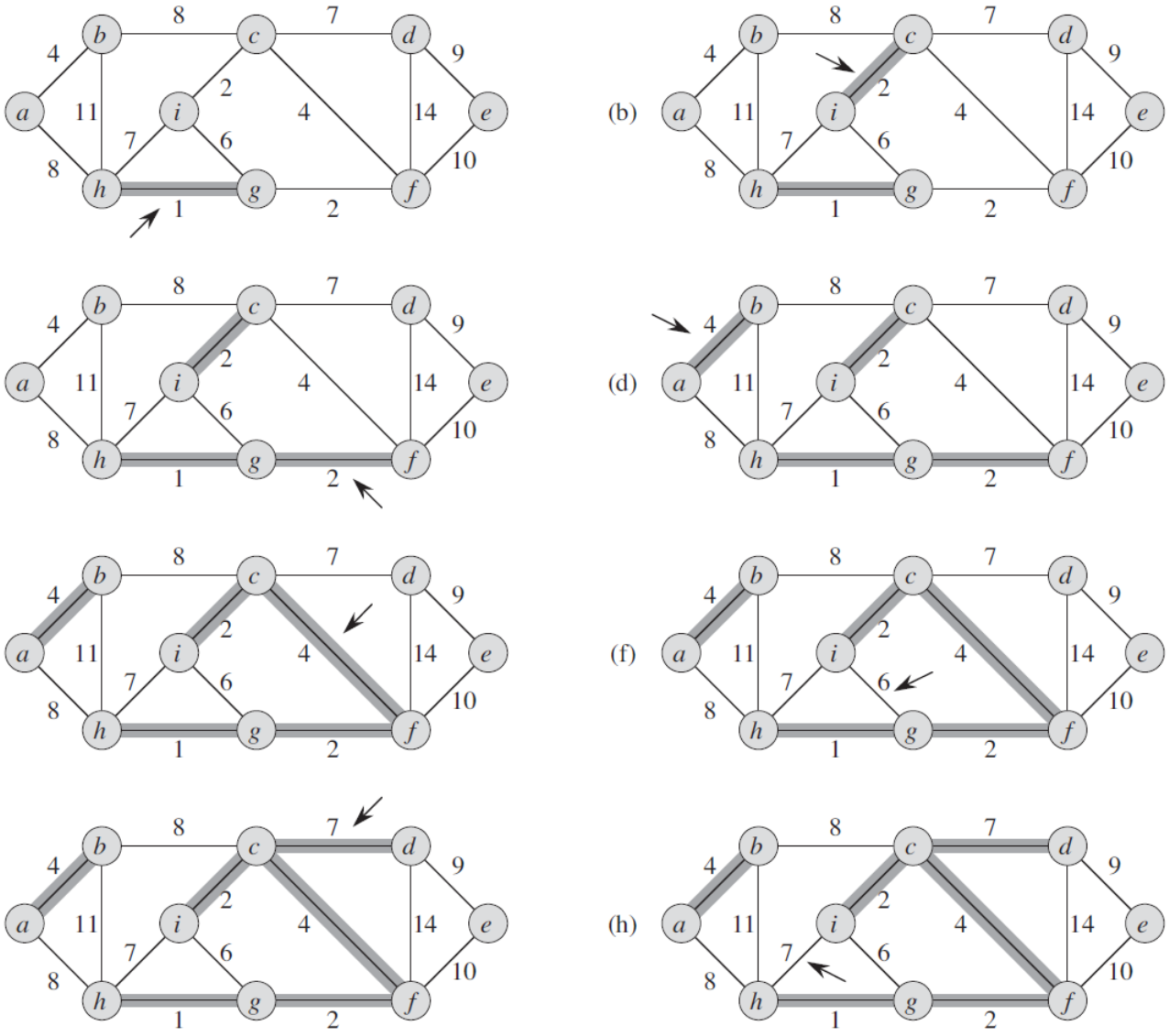


Kruskalov algoritmus

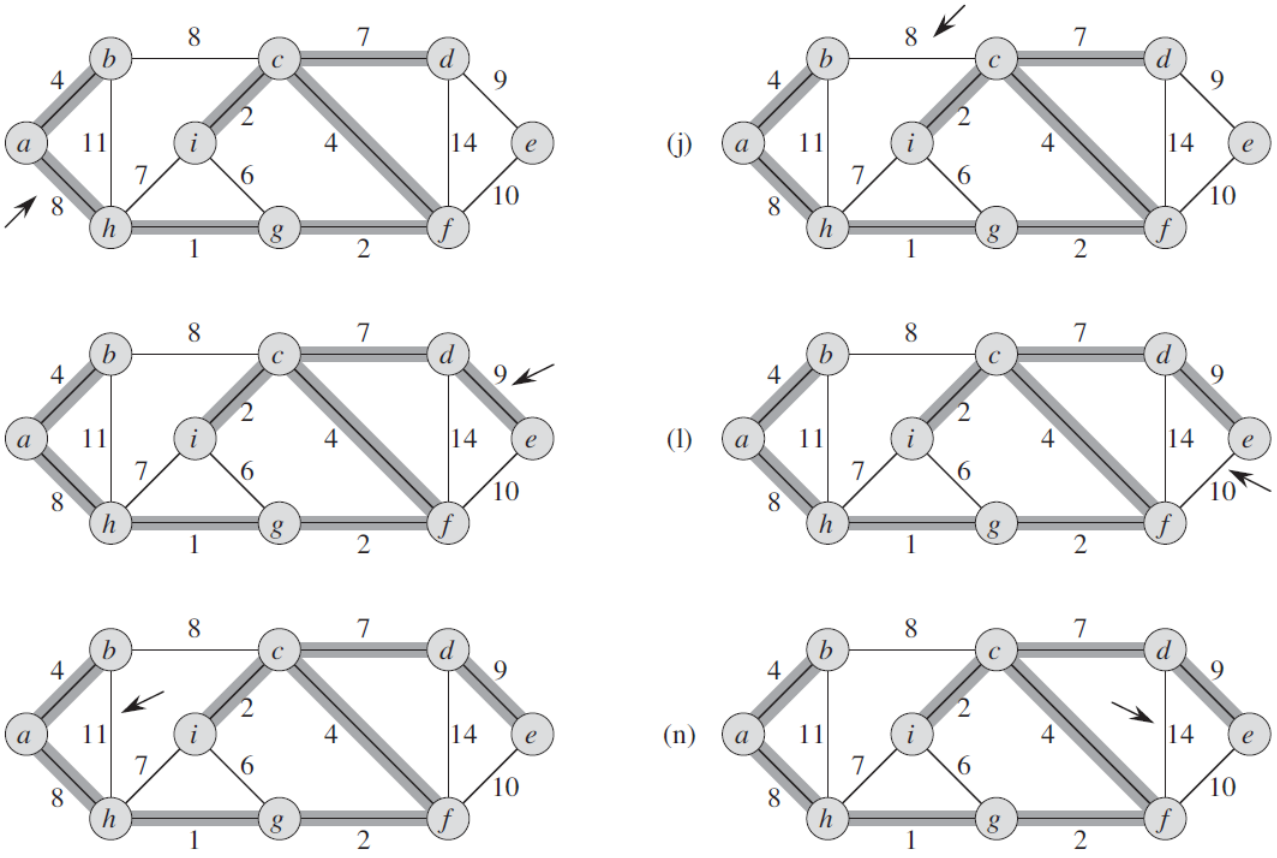
```
MST-Kruskal(G, w)
1 A ← ∅
2 for každý vrchol v ∈ V
3   do Make-Set(v)
4 usporiadaj hrany v H v neklesajúcom poradí podľa váhy w
5 for každú hranu (u, v) ∈ E v poradí podľa neklesajúcej váhy
6  do if Find-Set(u) ≠ Find-Set(v)
7       then A ← A ∪ {(u, v)}
8            Union(u, v)
9  return A
```

- Potrebujeme ešte dátovú štruktúru pre reprezentáciu disjunktných množín:
  - Make-Set(v) vytvoriť triviálnu (jednoprvkovú) množinu {v}
  - Find-Set(v) nájsť identifikátor množiny, v ktorej je prvok v
  - Union(u,v) spojiť množiny v ktorých sú prvky u a v

Kruskalov algoritmus – ukážka



Kruskalov algoritmus – ukážka

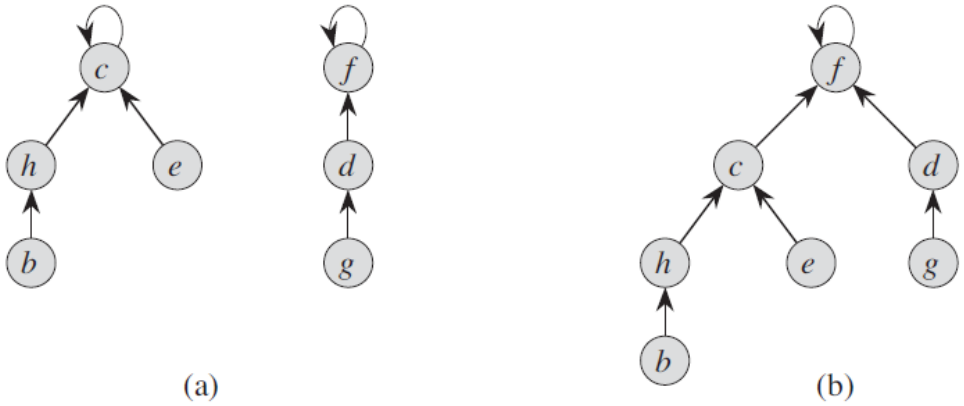


Kruskalov algoritmu – zložitosť

- Kritické je usporiadanie hrán O(M log M)
- Potom M krát voláme operácie union-find
  - Dobrá implementácia union-find má zložitosť operácií „skoro konštantnú“
- Celkovo teda O(M log M)

Dátová štruktúra pre disjunktné množiny

- Reprezentácia lesom



a) Dve disjunktné množiny:
{b,c,h,e} s reprezentantom c
{d,f,g} s reprezentantom f

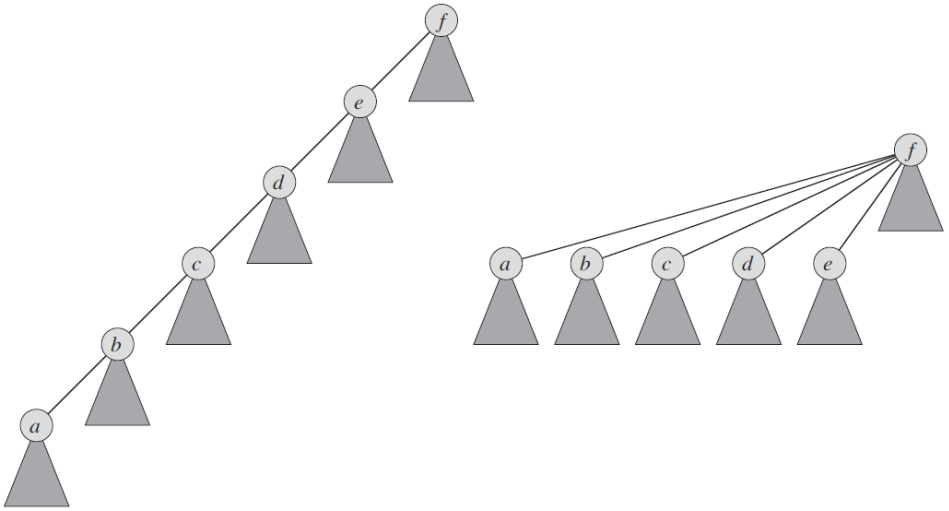
b) Spojenie UNION(e,g):
množina {b,c,d,e,f,g,h}
s reprezentantom f

```
MAKE-SET(x)
1 p_x = x

UNION(x,y)
1 p_FIND-SET(x) = FIND-SET(y)
```

Dátová štruktúra pre disjunktné množiny (2)

- Po viacerých vykonaniach UNION môžu byť cesty príliš dlhé – skrátime (môžeme aj vyvažovať ;)



- Kompresia cesty:
- Zložitosť: $O(\log^* N)$

```
FIND-SET(x)
1 if x ≠ p_x
2   then p_x = FIND-SET(p_x)
3 return p_x
```

Primov (Jarníkov) algoritmus

- Pri tvorbe minimálnej kostry sa udržuje rez medzi spracovanými (kostrou) a ešte nespracovanými vrcholmi

1. Inicializácia:
   Vybrať ľubovoľný vrchol a označiť ho ako spracovaný
2. **Z rezu vybrať najlacnejšiu hranu e a vložiť ju do vytváranej minimálnej kostry.**
3. Nespracovaný vrchol hrany e označiť ako spracovaný.
4. Opakovať krok 2 pokým nie sú spracované všetky vrcholy.

Objavitelia (nezávisle na sebe):
1930 Jarník, 1957 Prim, 1959 Dijkstra

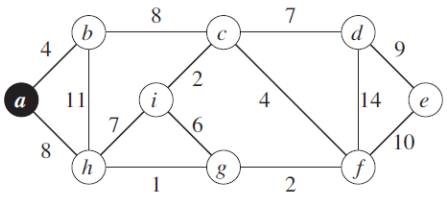
Primov algoritmus – implementácia

G je súvislý graf, w ohodnotenie hrán, r počiatočný vrchol

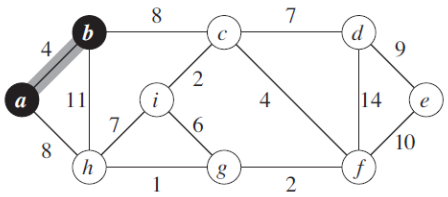
```
MST-Prim (G, w, r)
1 for každý vrchol u ∈ V
2    key_u ← ∞, p_u ← NIL
3 PQ ← V
4 key_r ← 0          (úprava hodnoty koreňa r v min-halde)
5 while PQ ≠ Ø
6    u ← Extract-Min (PQ)
7    for každý v ∈ Neigh(u)
8       if v ∈ PQ and w(u, v) < key_v
9          p_v ← u,
10         key_v ← w(u, v)      (úprava min-haldy)
```

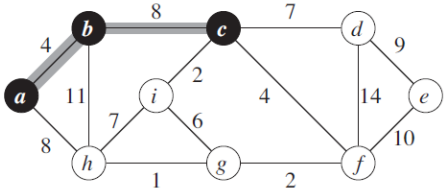
Q je min-halda, key_v priorita vrcholu v min-halde

Primov algoritmus – ukážka

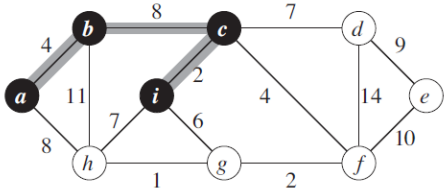


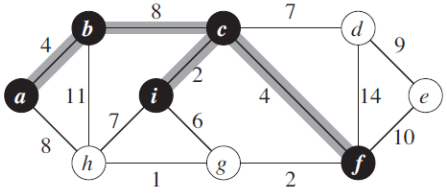
(b)



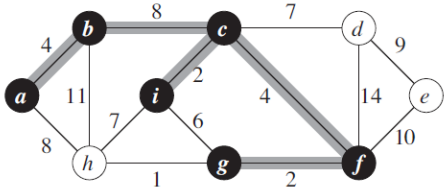


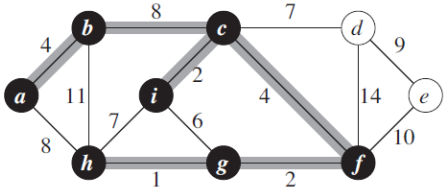
(d)



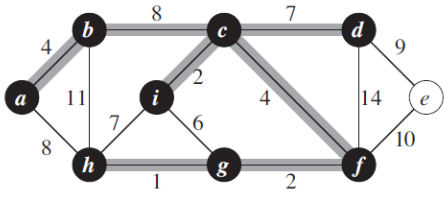


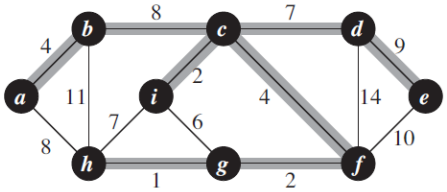
(f)





(h)





Primov algoritmus – zložitosť

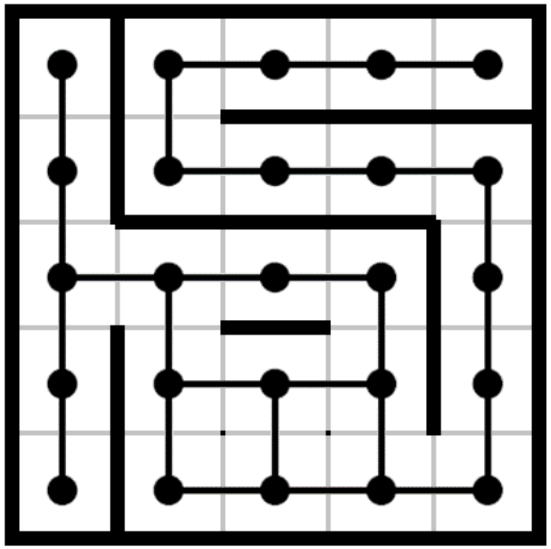
- Využitím min-haldy O(M log N):

```
        MST-Prim (G, w, r)
  O(N)  1 for každý vrchol u ∈ V
        2     key_u ← ∞, p_u ← NIL
  O(N)  3 PQ ← V
        4 key_r ← 0          (úprava priority v min-halde)
  O(N)  5 while PQ ≠ Ø
O(NlogN) 6     u ← Extract-Min (PQ)
 O(N+M) 7     for každý v ∈ Neigh(u)
 O(N+M) 8        if v ∈ PQ and w(u, v) < key_v
        9           p_v ← u,
O(MlogN) 10          key_v ← w(u, v)      (úprava min-haldy)
```

- Využitím vektoru s priamym prístupom $O(N^2)$:
  - Riadok 6 bude O(N)
  - Riadok 10 bude $O(M) = O(N^2)$

Opakovanie – bludiská

- Čo môžeme reprezentovať grafom?
- Mapy (bludisko s miestnosťami prepojenými chodbami)
  - Graf: vrchol = políčko, hrana = dá sa prejsť medzi políčkami



  - Postačuje nám LEN grafová reprezentácia!

Čo robí dobré bludisko?

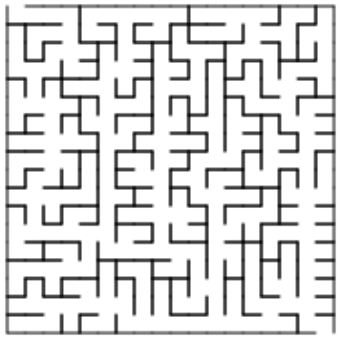
- Kedy je bludisko **náročné**?
  - Dlho blúdim – skúšam nejakú vetvu, keď nakoniec zistím, že nie je správna, musím sa vrátiť a skúsiť inú možnosť
  - **Teória grafov: Medzi dvoma miestami v bludisku existuje len jedna cesta, a teda nie je ľahké (si na križovatke) zvoliť tú správnu**
- Kedy je bludisko **ľahké**?
  - Skoro hocijako idem a hneď prídem do cieľa, nemusím sa vracať a skúšať inú možnosť
  - **Teória grafov: Medzi dvoma miestami v bludisku existuje viacero ciest, a teda sa mi ľahko stane, že si (na križovatke) zvolím nejakú správnu**

Ako vytvoriť dobré bludisko?

- Vizuálne požiadavky nemáme:
  - Bludisko by sa mohlo skladať z nejakých opakujúcich sa vzorov (napr. špirála), ale **náhodný vzor** pôsobí neprekonateľnejšie – dlhšie blúdenie :)
- Budeme teda vytvárať bludisko, v ktorom:
  - Medzi ľubovoľnými dvoma miestami v bludisku **existuje najviac jedna cesta**, a teda nie je ľahké (si na križovatke) zvoliť tú správnu
  - Medzi ľubovoľnými dvoma miestami v bludisku **existuje aspoň jedna cesta**, aby sme využili dostupnú plochu bludiska čo najviac (na blúdenie)
- Čo nato hovorí teória grafov?

Generovanie bludiska

- Bludisko – 2D pole miestností N x N
  - Graf: vrchol = políčko (miestnosť so stenami)
    hrana = dá sa prejsť medzi políčkami



- **Začneme s úplne zamurovaným bludiskom:**
  každá miestnosť má všetky steny
  (hore, doprava, dole, doľava)
  - Graf s N x N vrcholmi, bez hrán
- Úloha:
  Steny prebúrať tak, aby vzniklo bludisko!
  - Pridávať hrany tak, aby nakoniec medzi ľubovoľnými dvoma vrcholmi bola **práve jedna cesta**

Generovanie bludiska

- Začneme s úplne zamurovaným bludiskom.
- Pridávať hrany (prebúrať steny) tak, aby nakoniec medzi ľubovoľnými dvoma vrcholmi bola **práve jedna cesta.**
- **Skúšam hrany v náhodnom poradí, pridám hranu len ak nevznikne cyklus** (detekcia cyklu: union-find):

```
Generate-Maze(N)
1 E ← ∅
2 W ← množina stien (možné hrany)
3 randomShuffle(W)
4 for každú stenu (u,v) ∈ W
6  do if Find-Set(u) ≠ Find-Set(v)
7        then E ← E ∪ {(u, v)}
8          Union(u, v)
9  return E
```

YouTube

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

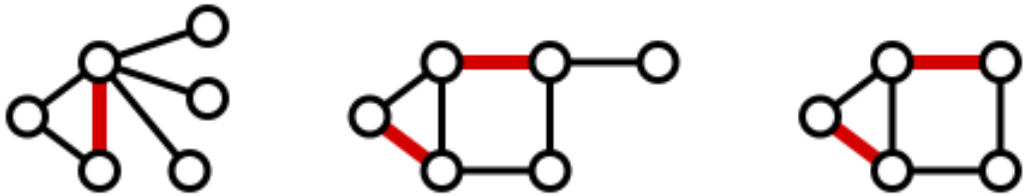
Grafové algoritmy
(párovanie)

7. 11. 2017

zimný semester 2017/2018

Teória grafov – Párovanie

- Daný je (neorientovaný) graf, **párovanie*** (matching) nazývame množinu hrán, ktoré nemajú spoločný vrchol



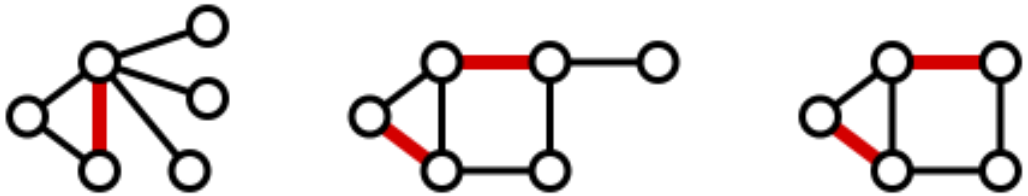
- Množina nezávislých hrán
- Vrchol incidujúci s hranou v párovaní nazývame **spárovaný** (matched) alebo pokrytý, ostatné nespárované (unmatched, free) alebo nepokryté

---

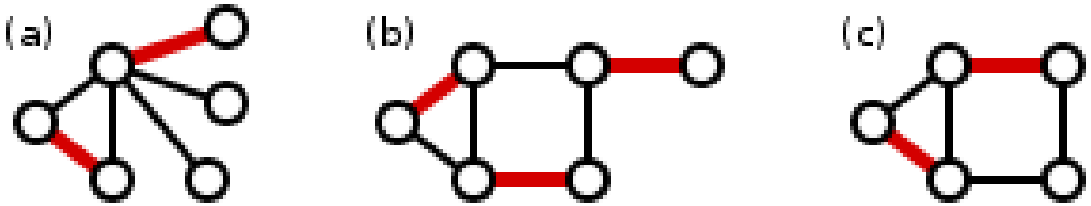
* slovenská literatúra zvyčajne uvádza tento pojem ako „párenie“, čo ale zodpovedá skôr anglickému „mating“ (viď Google Image Search), budeme preto používať vhodnejší český pojem „párovaní“ ...

Teória grafov – Najpočetnejšie párovanie

- Maximálne párovanie – taká množina hrán M, že ak do nej pridám nejakú ďalšiu hranu, už to nebude párovanie:



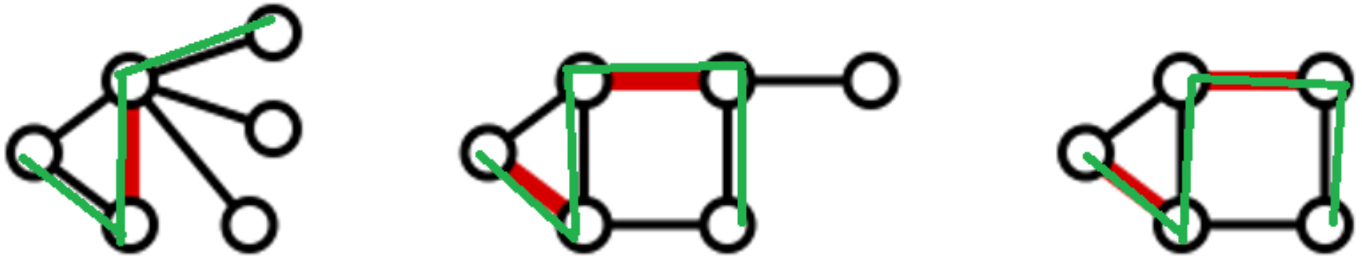
- **Najpočetnejšie** (**maximum**-cardinality) párovanie je párovanie, ktoré obsahuje najväčší možný počet hrán.



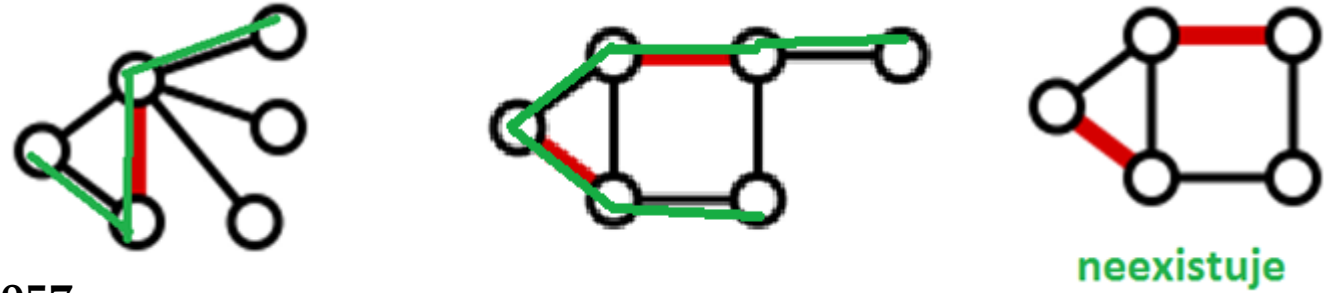
- Ak je každý vrchol spárovaný nazývame ho **perfektné** (alebo úplné) párovanie. Napr. (b)

Teória grafov – Najpočetnejšie párovanie (2)

- **Alternujúca cesta** – postupnosť hrán, v ktorej sa striedajú hrany patriace a nepatriace do párovania:



- **Zväčšujúca** cesta – alternujúca cesta, ktorej krajné vrcholy sú nespárované.

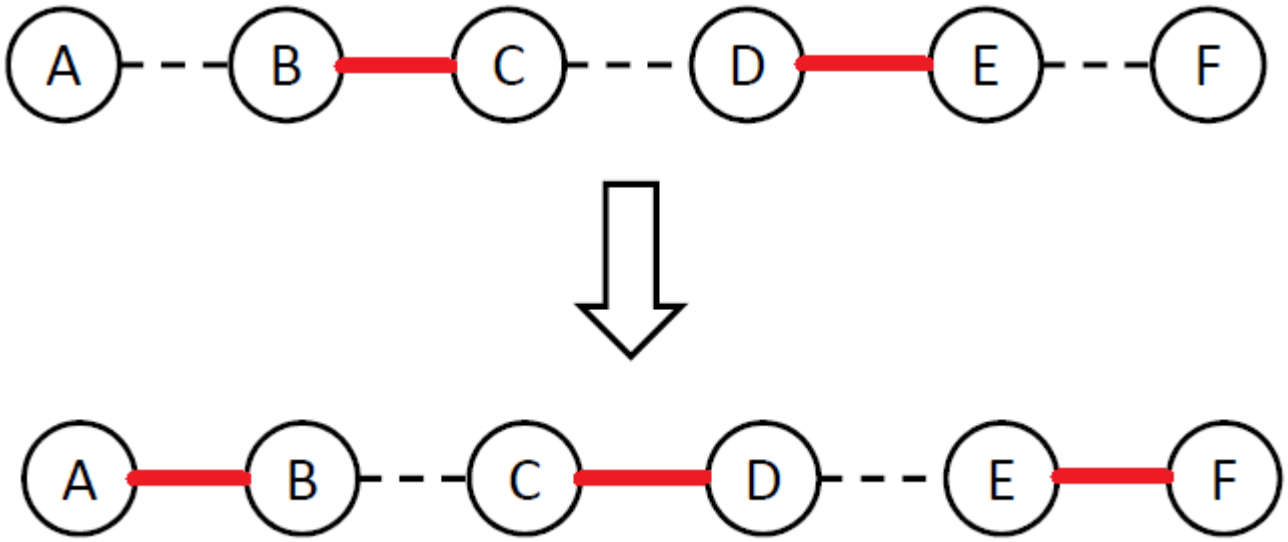


Berge 1957:

Párovanie je najpočetnejšie ⇔ neexistuje zväčšujúca cesta.

Ako nájdem najpočetnejšie párovanie?

- Nájdem zväčšujúcu cestu
- V párovaní vymením hrany na zväčšujúcej ceste:
  - Tie hrany, ktoré patria do párovania odstránim z párovania, a
  - tie hrany ktoré nepatria do párovania pridám do párovania.
- Napr.:



Ako nájdem najpočetnejšie párovanie? (2)

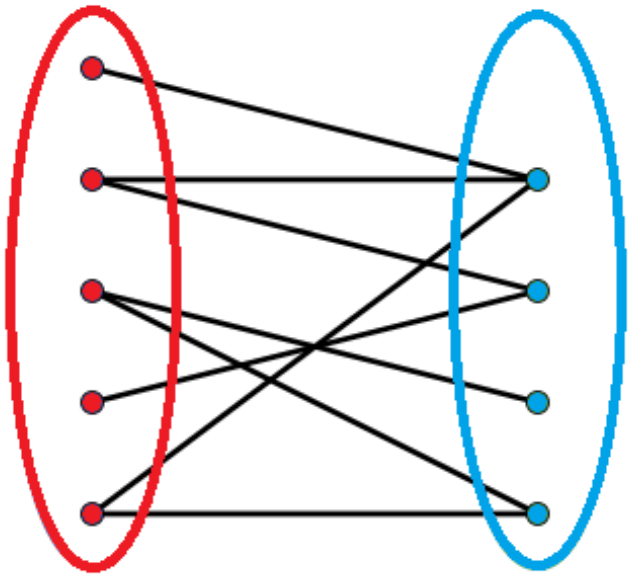
- Uvažujme párovanie M a najpočetnejšie párovanie M'
- Označme k=|M'| – |M|, potom párovanie M obsahuje k vrcholovo nezávislých zväčšujúcich ciest.
- Množina M' $\oplus$ M (symetrická diferencia) je množina hrán, ktoré sú v M' alebo M ale nie v oboch naraz.
- Každý vrchol je incidentný s najviac dvomi hranami v M' $\oplus$ M. Súvislé komponenty podgrafu indukovaného hranami M' $\oplus$ M sú preto (jednoduché) **cesty** a **cykly**.
- Na každej takejto ceste alebo cykle sa striedajú hrany množín M' a M.

Ako nájdem najpočetnejšie párovanie? (3)

- Na každej takejto ceste alebo cykle sa striedajú hrany množín M' a M.
- Každý cyklus obsahuje rovnaký počet hrán z M' ako z M.
- Každá cesta obsahuje rovnaký počet hrán z M' ako z M až na jednu. Cesta, ktorá obsahuje o jednu hranu z M' viac ako z M je zväčšujúca.
- V množine $M \oplus M'$ je práve k hrán viac z M' ako z M, teda podgraf indukovaný hranami $M \oplus M'$ obsahuje k vrcholovo nezávislých zväčšujúcich ciest pre M.
- Dôsledok:
  Pre M existuje zväčšujúca cesta dĺžky najviac n/k.

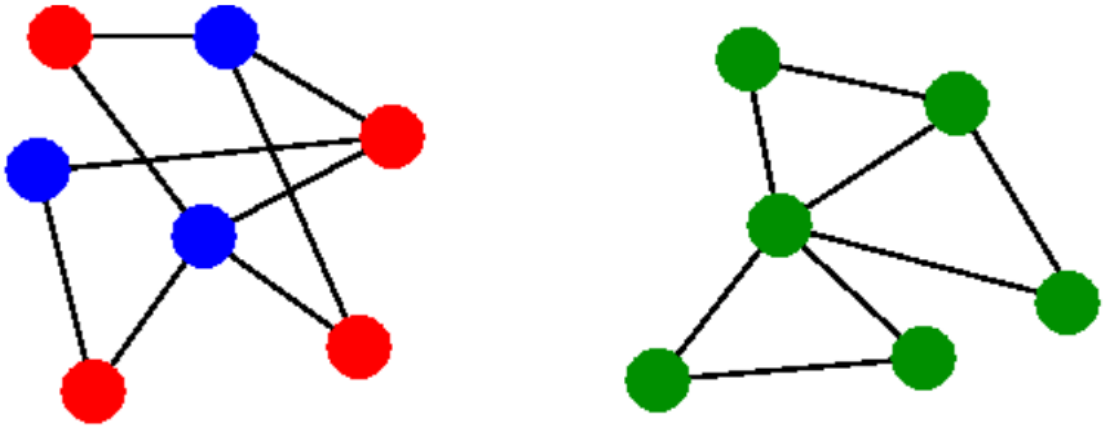
Teória grafov – Bipartitné párovanie

- Dôležitý špeciálny prípad: párovanie v bipartitnom grafe
- **Bipartitný graf** je taký graf, v ktorom sa dajú vrcholy rozdeliť do dvoch (disjunktných) množín tak, aby hrany išli len medzi vrcholmi v rôznych množinách
- Napr.



Ako zistím, či je graf bipartitný?

- Sú tieto grafy bipartitné?



Graf nie je bipartitný ⇔ obsahuje cyklus nepárnej dĺžky.

- Algoritmus:
  Ofarbujem vrcholy dvoma farbami. Začnem v nejakom vrchole, susedov ofarbím inou farbou ...

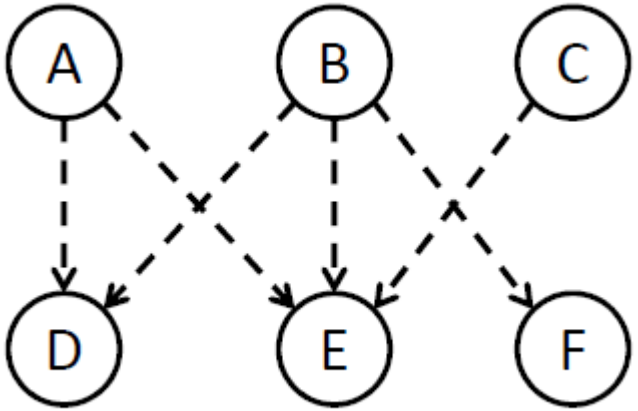
Najpočetnejšie párovanie v bipartitnom grafe

- X, Y partície vrcholov (neorientovaného) grafu
- Počas behu algoritmu budeme meniť smer hrán
  - Ak hrana nepatrí do aktuálneho párovania, smer bude z X do Y
  - Ak hrana patrí do aktuálneho párovania, bude smer z Y do X

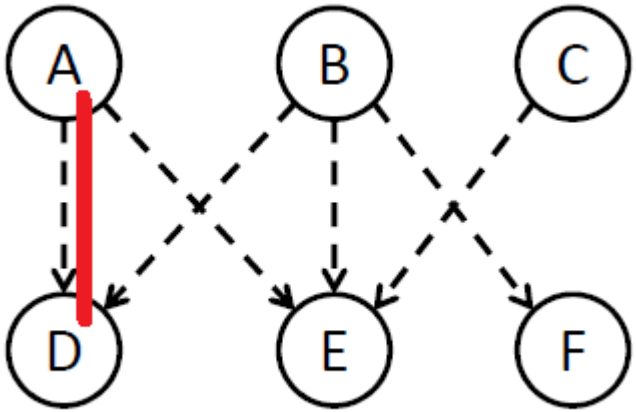
```
BIPARTITE-MATCHING(G):
1 Začni s prázdnym párovaním.
2 Pre všetky hrany nastav smer z X do Y.
3 while ∃nespárovaný vrchol x ∈ X do
4   prehľadávanie z x až do nespárovaného vrcholu v Y
5   if podarilo sa dosiahnuť nespárovaný vrcholov v Y then
6     zväčši párovanie využitím nájdenej zväčšujúcej cesty
7     obráť smer všetkých hrán na zväčšujúcej ceste
8   else
9     odstráň všetky navštívené vrcholy
```

Algoritmus zväčšujúcich polociest – ukážka

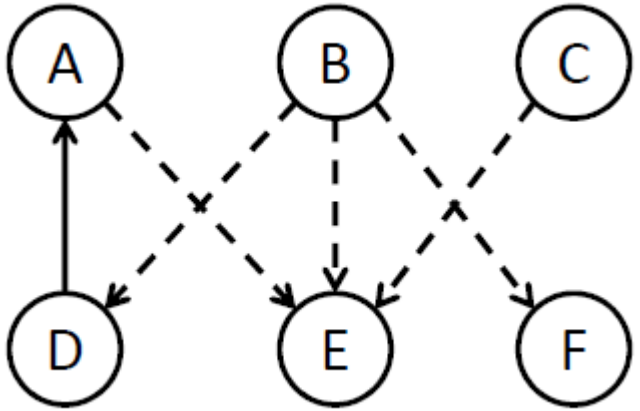
- Prehľadávaj z vrcholu A



Zväčšujúca cesta A → D:

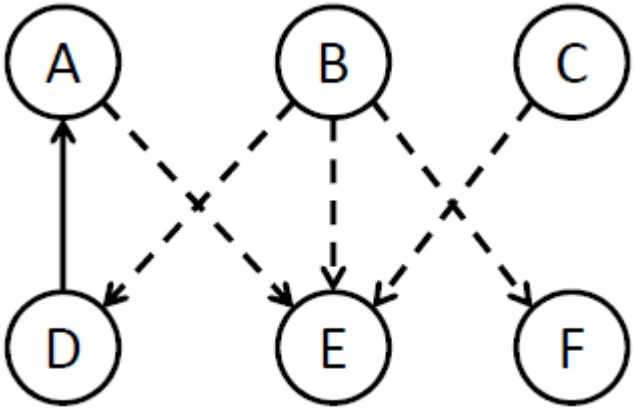


Zmena smeru hrán na zväčšujúcej ceste:

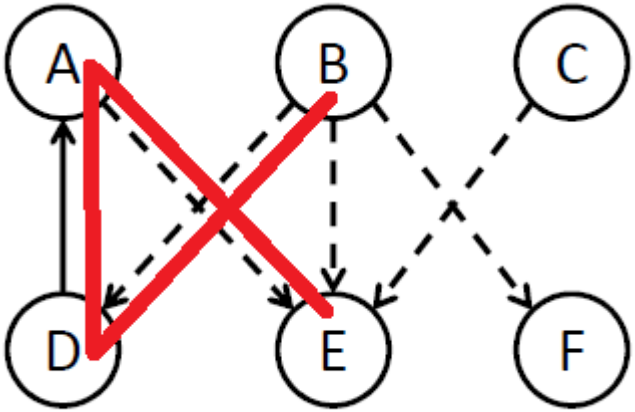


Algoritmus zväčšujúcich polociest – ukážka (2)

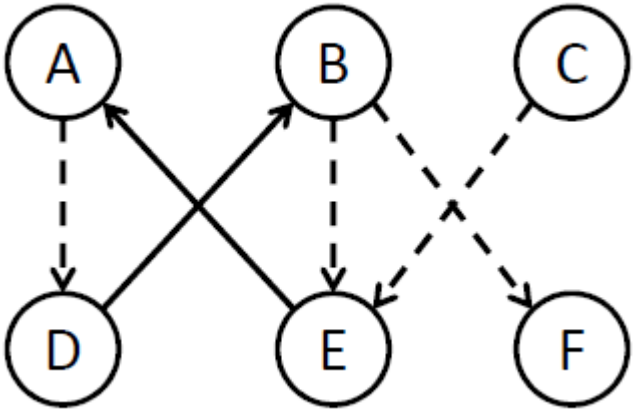
- Prehľadávaj z vrcholu B



Zväčšujúca cesta
B → D → A → E:

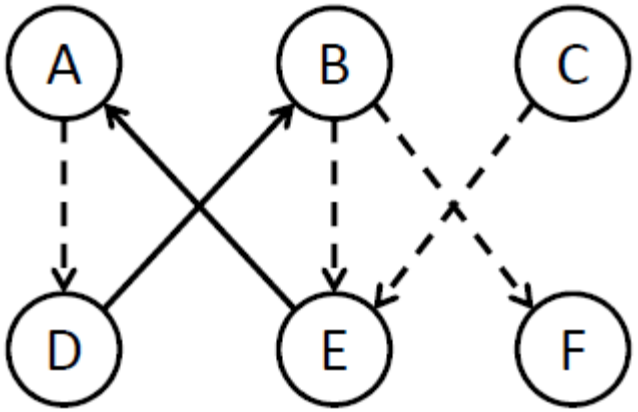


Zmena smeru hrán na zväčšujúcej ceste:

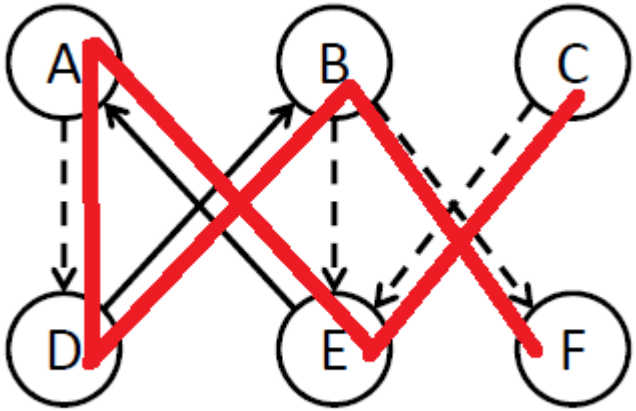


Algoritmus zväčšujúcich polociest – ukážka (3)

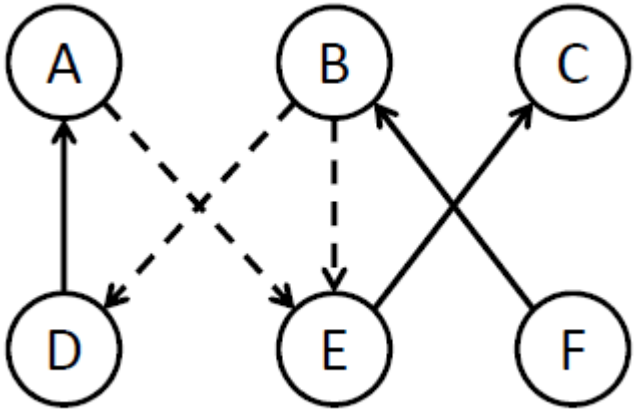
- Prehľadávaj z vrcholu C



Zväčšujúca cesta
C → E → A → D → B → F:



Zmena smeru hrán na zväčšujúcej ceste:



- Zložitosť: O(M) jeden prechod
  Celkovo O(NM)

Otázka z publika

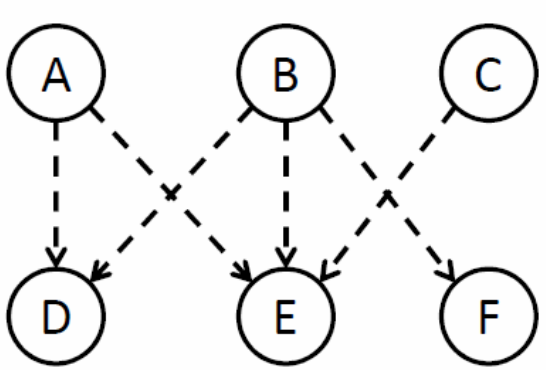
- Prečo v algoritme meníme smer (orientáciu) hrán?
- Odpoveď: Preto, aby každá cesta v takto upravenom grafe bola alternujúca cesta (pre aktuálne párovanie).
- Graf, ktorý priebežne upravujeme (zmenou orientácie hrán) je grafom alternujúcich ciest.
- Okrem toho, že každá cesta v tomto grafe je alternujúca, tak každá x-y cesta ($x \in X, y \in Y$) je zväčšujúca cesta!
- Preto môžeme použiť akýkoľvek algoritmus na hľadanie ciest začínajúcich v X a končiacich v Y nato, aby sme našli nejakú zväčšujúcu cestu.

Hopcroft-Karpov algoritmus

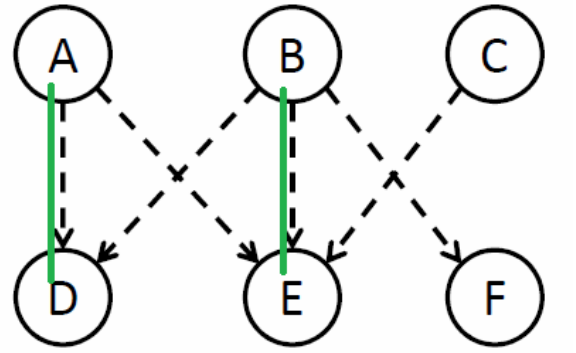
1. Začni prehľadávanie do šírky zo všetkých nespárovaných vrcholov (vlož všetky do počiatočného radu), vytvoríme vrstevný podgraf G' obsahujúci všetky najkratšie zväčšujúce cesty. (prehľadávanie prerušíme na vrstve, v ktorej sa nachádza prvý nespárovaný vrchol)
2. Ak neexistuje zväčšujúca cesta, KONIEC.
3. Nájdi vrcholovo nezávislé cesty v G' pre každý začiatočný vrchol jednu. (Prehľadávanie do hĺbky)
4. Zväčši existujúce párovanie využitím týchto ciest. Choď na Krok 1.

Hopcroft-Karpov algoritmus – Ukážka

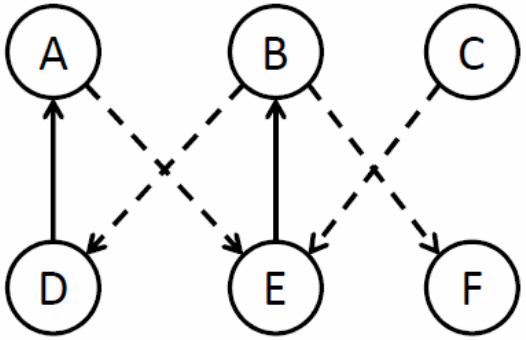
- Prehľadávanie do šírky z A, B, C: G' bude celý graf



- Prehľadávanie do hĺbky nájde cesty: A→D, B→E:

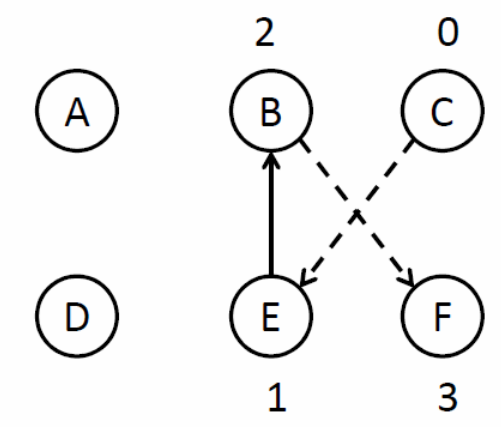


Zväčšíme párovanie:

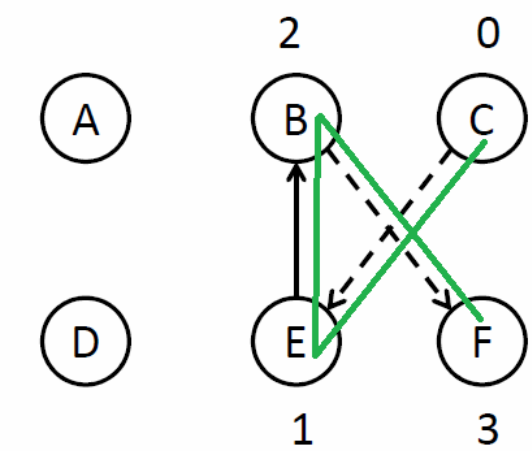


Hopcroft-Karpov algoritmus – Ukážka

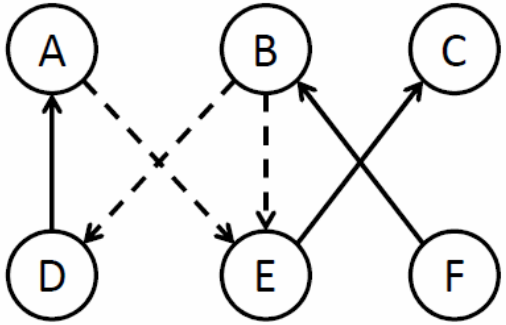
- Prehľadávanie do šírky z C, vznikne podgraf:



- Prehľadávanie do hĺbky nájde cestu: C→E→B→F:



Zväčšíme párovanie:



Hopcroft-Karpov algoritmus – Zložitosť

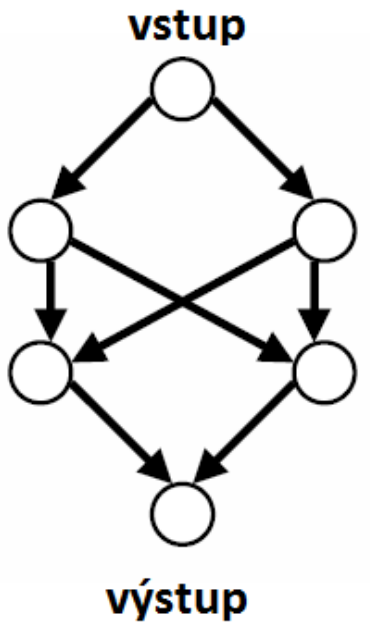
- Jedna fáza: O(M)
  - Konštrukcia podgrafu (prehľadávanie do šírky z každého nespárovaného vrcholu) – navštívime každú hranu najviac raz
  - Určenie vrcholovo nezávislých ciest (prehľadávanie do hĺbky najviac z každého nespárovaného vrcholu podgrafu) – navštívime každú hranu najviac raz
- Počet fáz: najviac $2N^{½} + 1$, celkovo teda $O(N^{½}M)$
  - Po k-tej fáze bude mať ďalšia (ak existuje) zväčšujúca cesta dĺžku aspoň 2k+1, a teda najpočetnejšie párovanie je najviac o n/(2k+1) početnejšie ako existujúce, a teda prebehne ešte naviac n/(2k+1)+1 fáz.
  - Celkovo najviac prebehne k + n/(2k+1)+1 fáz (pre ľubovoľné k), pre $k=(N/2)^{½}$ máme $2N^{½} + 1$

Párovanie – Ohodnotené grafy

- Všeobecnejší problém:
  Ak má každé hrana e=(i,j) váhu (cenu) w(e)
  - Hrana i → j označuje cenu priradenia j-tej úlohy i-temu pracovníkovi.
- Najlacnejšie párovanie v bipartitnom grafe
  - Hľadáme priradenie úloh, ktoré má najnižšiu cenu.
  - Maďarská metóda (Hungarian method)
- Najdrahšie párovanie (s najvyššou celkovou cenou hrán)
  - Trasformácia na úlohu o najlacnejšom párovaní:
    nová cena hrany w'(e) bude maximum z cien mínus w(e)
- Najdrahšie / najlacnejšie perfektné párovanie
- Najdrahšie / najlacnejšie najpočetnejšie párovanie
- Rozšírenie na všeobecné (nie len bipartitné) grafy

Zovšeobecnenie – Toky (Network flow)

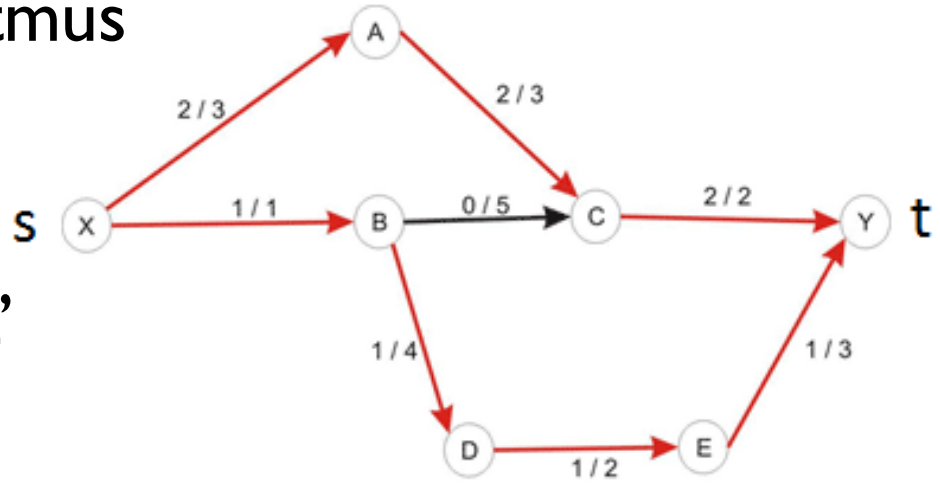
- Transportné siete – abstrakcia fyzických tokov (voda, ropa, materiál všeobecne) modelovanú v grafoch
- Intuitívna predstava:
  - Sústava rúr rôznych veľkostí (kapacitné ohraničenia)
  - Prepojená na križovatkách, v ktorých sú regulátory prietoku a smeru toku
  - Sústava je vyvážená vtedy, keď množstvo, ktoré odteká z uzlu je rovné množstvu, ktoré tam priteká.



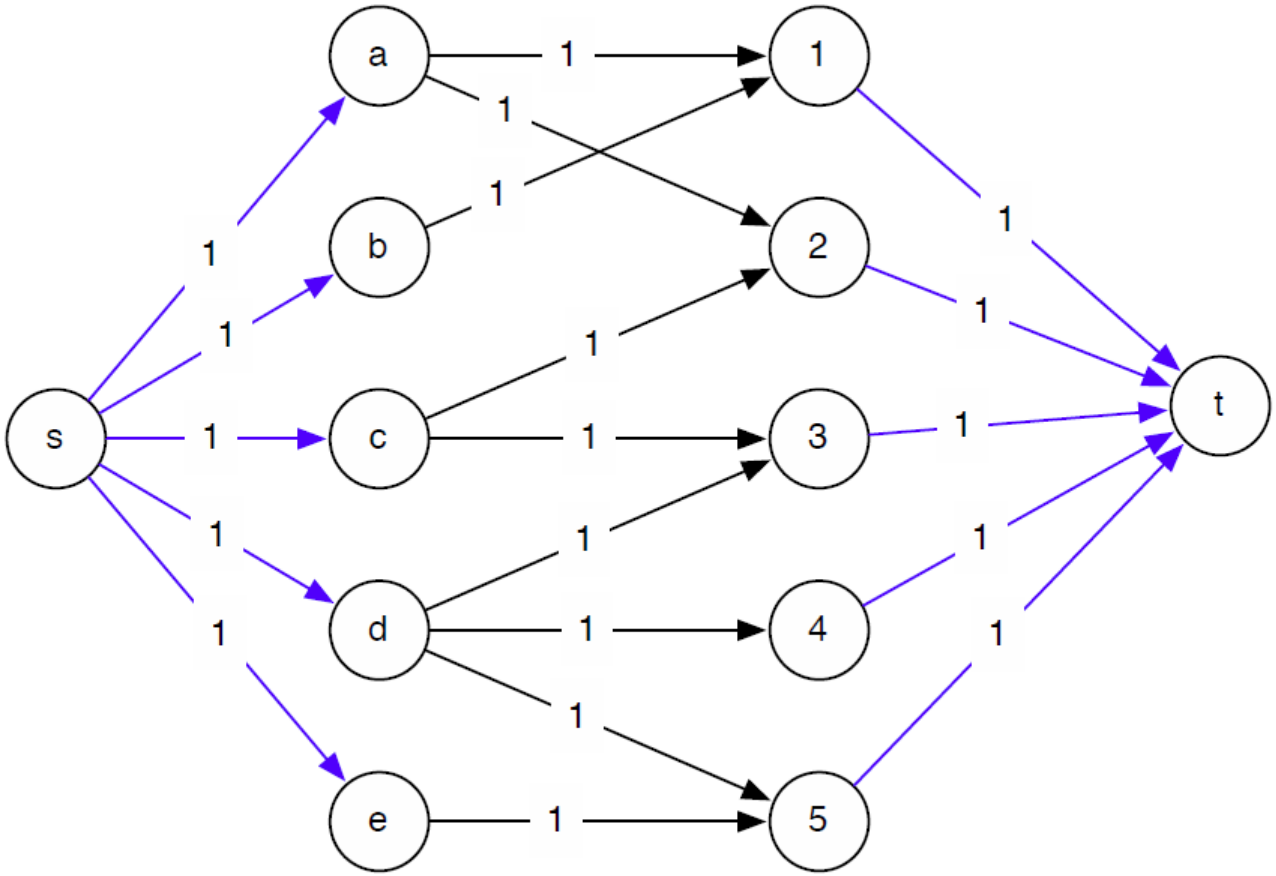
- Teória grafov: každé hrana má dve ohodnotenia c_{ij} kapacita rúry $i \rightarrow j$, a f_{ij} prietok $i \rightarrow j$

Maximálny tok (maxflow)

- Pre daný orientovaný graf G s danými kapacitami hrán a dvoma vyznačenými vrcholmi s a t, úloha je nájsť:
- Priradenie f (tok) – ohodnotenia hrán f_{ij} také, že spĺňajú kapacitné ohraničenia hrán, podmienku kontinuity (tok sa vo vrcholoch nestráca) a množstvo, ktoré tečie z vrcholu s do vrcholu t je čo najväčšie možné.
- Ford-Fulkersonov algoritmus zväčšujúcich polociest
- Dajú sa takto modelovať rôzne typy úloh...



Párovanie v bipartitnom grafe ako tok

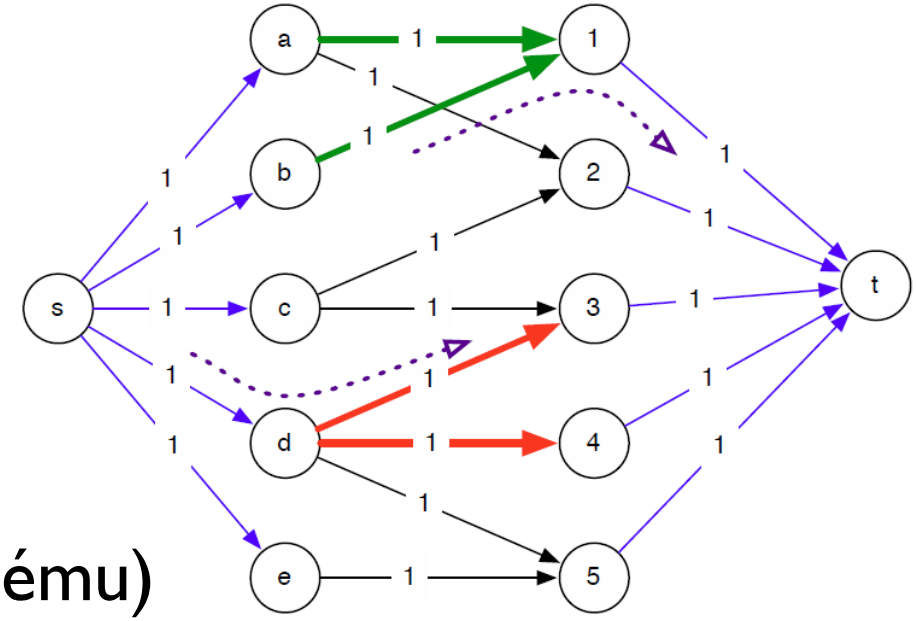


Párovanie v bipartitnom grafe ako tok

- Daný je bipartitný graf $G=(X \cup Y, E)$, vytvor transportnú sieť G' nasledovne:
  - Orientácia hrán bude z X do Y
  - Pridaj nové vrcholy s a t
  - Pridaj hranu z s do každého vrcholu $x \in X$
  - Pridaj hranu z každého vrcholu $y \in Y$ do t
  - Všetky kapacity budú 1
- Vyrieš úlohu o maximálnom toku v sieti G'
  - Hrany v nájdenom (maximálnom) toku budú zodpovedať najpočetnejšiemu párovaniu v G

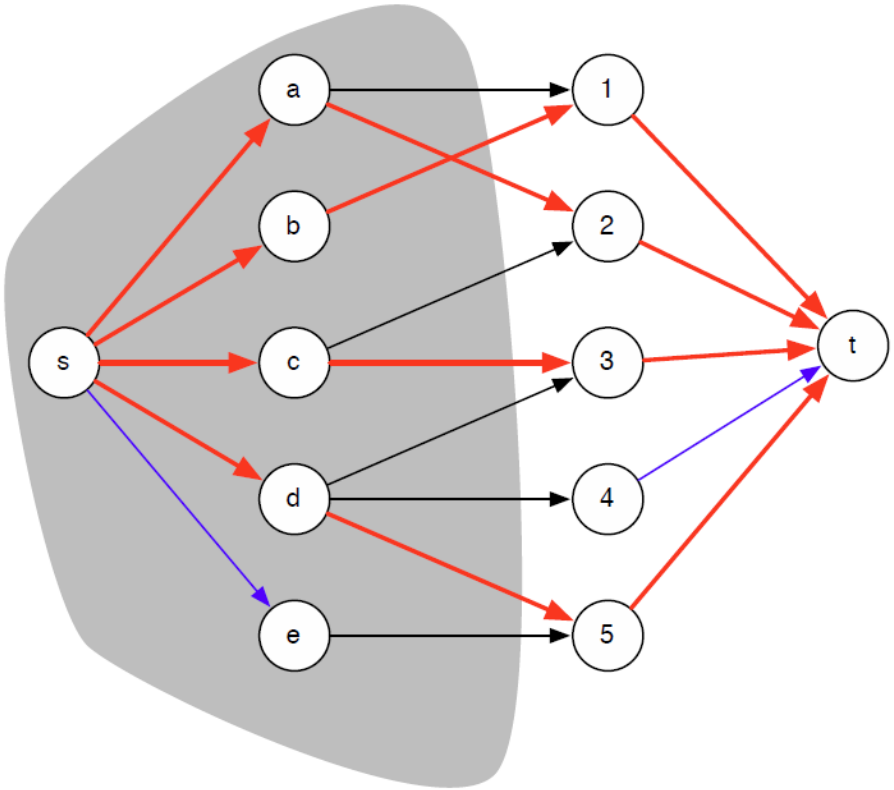
Párovanie v bipartitnom grafe ako tok

- Toky sú celočíselné
- Z každého x ∈ X vyberieme (do toku) najviac jednu hranu (lebo kapacita hrany s,x je $c_{sx} = 1$)
- Do každého y ∈ Y pôjde najviac jedna jednotka toku (lebo $c_{yt} = 1$)
- Ak by sme vybrali viac hrán, nebol by to vyvážený tok
- Resp. môžeme modelovať zložitejšie úlohy (napr. priradenie viacerých k jednému)



Párovanie v bipartitnom grafe ako tok

- Ak existuje párovanie obsahujúce k hrán v G, tak existuje tok veľkosti k v G' a opačne platí tiež.



STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

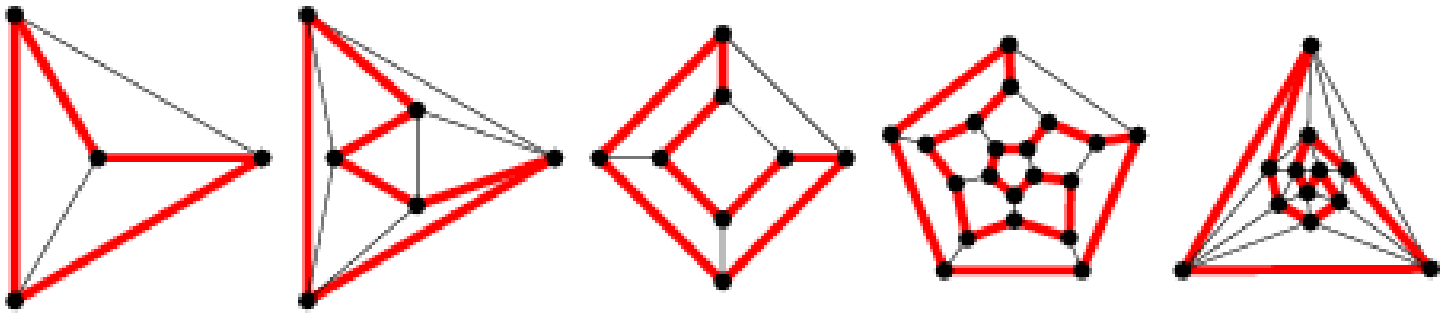
Grafové algoritmy
(Hamiltonovské grafy)

7. 11. 2017

zimný semester 2017/2018

Teória grafov – Hamiltonovský sled, cesta, cyklus

- **Hamiltonovský sled** v grafe G je taký sled, ktorý obsahuje všetky vrcholy grafu G.
- **Hamiltonovská cesta** je taký hamiltonovský sled, ktorý neobsahuje rovnaké hrany
- **Hamiltonovský cyklus** je taký hamiltonovský sled, v ktorom sa okrem prvého a posledného vrcholu žiaden vrchol nevyskytuje viac než raz



Hamiltonovská cesta – Ukážka

- Prechod koňom po šachovnici