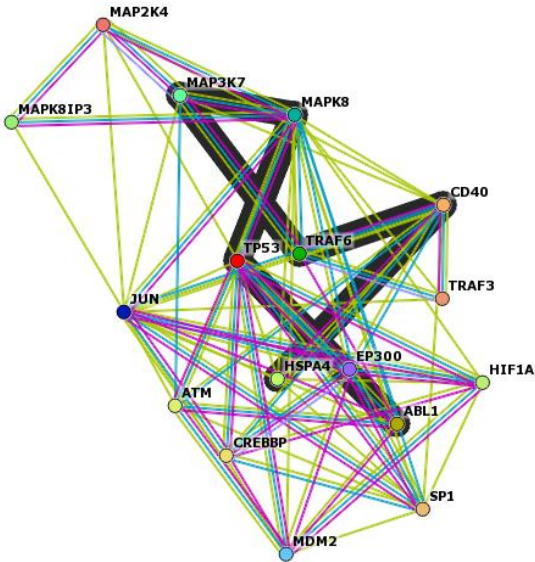




# 9. prednáška

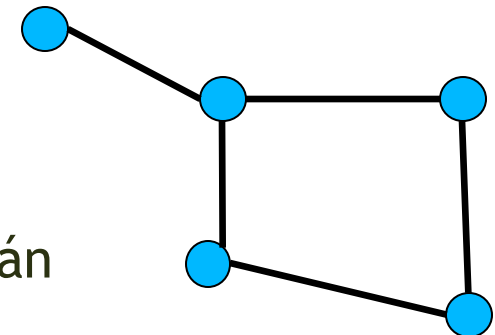
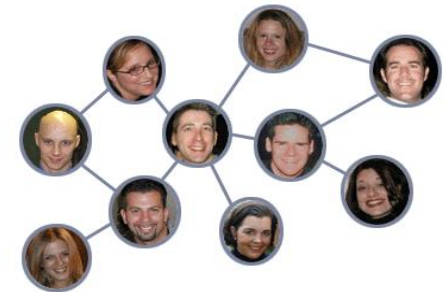
## Najkratšie cesty (v grafe)





# Grafy – čo už vieme...

- Umožňujú **modelovať** relácie medzi objektmi reálneho sveta
- Skladajú sa z **vrcholov** a **hrán**  $G=(V, E)$ 
  - **neorientované grafy** (krúžky a čiary)
    - sociálne siete, dopravné siete
  - **orientované grafy** (krúžky a šípky)
    - dopravné siete s jednosmerkami, následnosť aktivít
- Prehľadávanie grafov:
  - **BFS** (do šírky)
    - bonus: cesty s najmenším počtom hrán
  - **DFS** (do hĺbky)





# Realita nie sú grafy ...

- Reálne vzťahy nie sú „boolean“ - áno/nie
  - máme najlepších priateľov, dobrých priateľov, „iba“ známych...
  - cesta medzi za sebou idúcimi zástavkami MHD trvá rôzne dlho
  - aj keď medzi mestami sú cesty, vzdialenosti medzi mestami sú rôzne

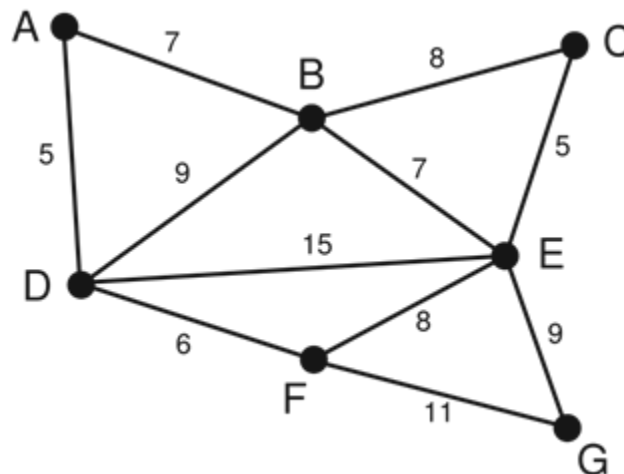


**Ako to modelovať?**



# Ohodnotené grafy

- Na vyjadrenie „kvality“ vzťahu medzi dvoma vrcholmi pridáme ku každej **hrane** grafu **ohodnotenie**.
- Formálne:
  - **ohodnotený graf** je  $G=(V, E, c)$ , kde  $c$  je funkcia z množiny hrán  $E$  do množiny čísel  $R$ , t.j.  $c:E \rightarrow R$





# Interpretácia ohodnotení

## ● Cestná sieť:

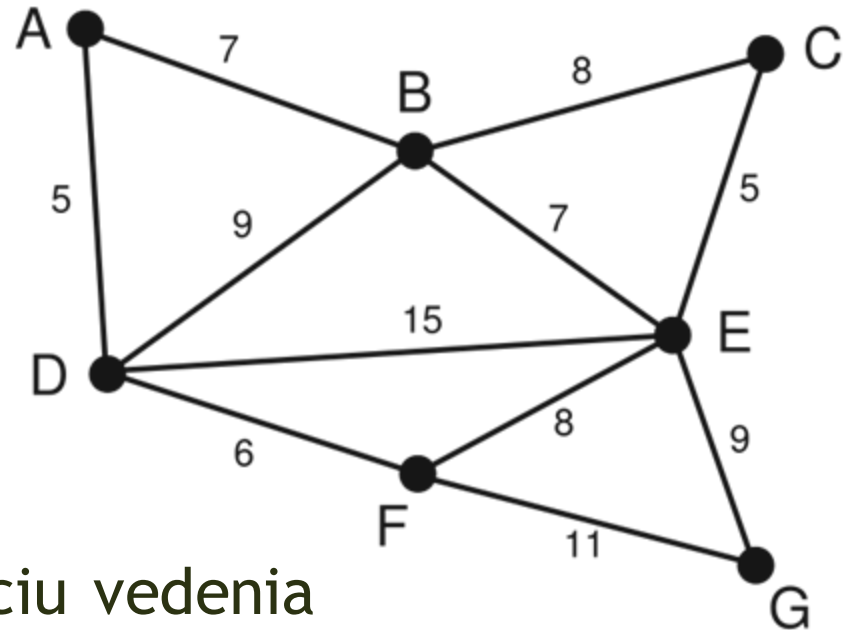
- dĺžka cesty
- čas na jej prejdenie
- šírka cesty

## ● Elektrorozvodná sieť:

- náklady na rekonštrukciu vedenia
- prenosová kapacita

## ● Sociálna sieť:

- ako dlho sa osoby poznajú
- počet vzájomne poslaných správ





# Ako uložit' ohodnotený graf?

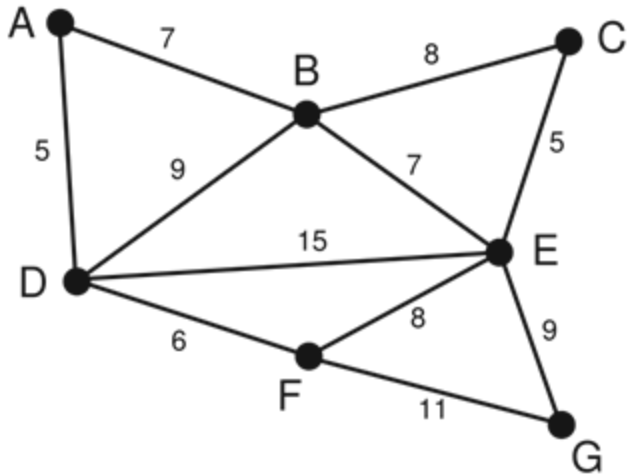
- **Matica susednosti:**
  - dvojrozmerné pole čísel + dohodnutá **hodnota na neprítomnosť hrany**
    - dohodnutá hodnota nemôže byť platným ohodnotením hrany
    - záporné číslo (-1) ak vieme, že ohodnotenia sú len kladné
    - `Double.POSITIVE_INFINITY` ekvivalent pre  $\infty$
- V knižnici `PAZGraphs.jar`
  - metóda `getWeight` triedy `Edge`
- Ďalšie reprezentácie ...



# Matica susednosti

Double.POSITIVE\_INFINITY =  $\infty$

`double[][] graf = new double[7][7]`

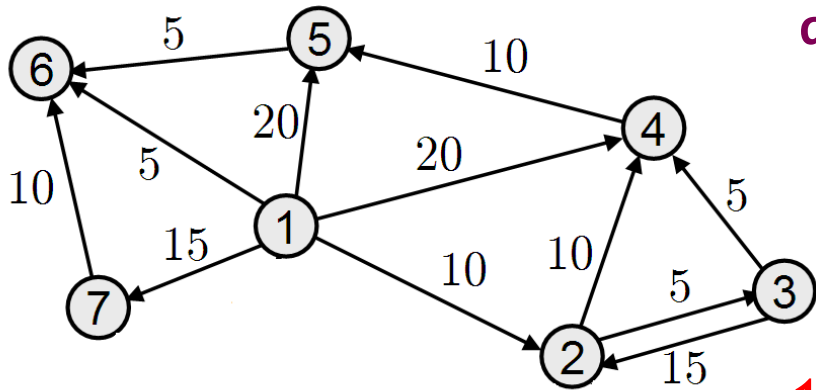


`graf[u][v]` = ohodnotenie hrany medzi `u` a `v`,  
resp.  $\infty$  ak hrany niet

	A	B	C	D	E	F	G
A	$\infty$	7	$\infty$	5	$\infty$	$\infty$	$\infty$
B	7	$\infty$	8	9	7	$\infty$	$\infty$
C	$\infty$	8	$\infty$	$\infty$	5	$\infty$	$\infty$
D	5	9	$\infty$	$\infty$	15	6	$\infty$
E	$\infty$	7	5	15	$\infty$	8	9
F	$\infty$	$\infty$	$\infty$	6	8	$\infty$	11
G	$\infty$	$\infty$	$\infty$	$\infty$	9	11	$\infty$



# Ohodnotené orientované grafy



```
double[][] graf = new double[7][7]
```

	1	2	3	4	5	6	7
1	∞	10	∞	20	20	5	15
2	∞	∞	5	10	∞	∞	∞
3	∞	15	∞	5	∞	∞	∞
4	∞	∞	∞	∞	10	∞	∞
5	∞	∞	∞	∞	∞	5	∞
6	∞	∞	∞	∞	∞	∞	∞
7	∞	∞	∞	∞	∞	10	∞

graf[u][v] = ohodnotenie hrany z u do v, resp. ∞ ak hrany niet





# Ohodnotené cesty v grafoch

- Cesta „neformálne“:

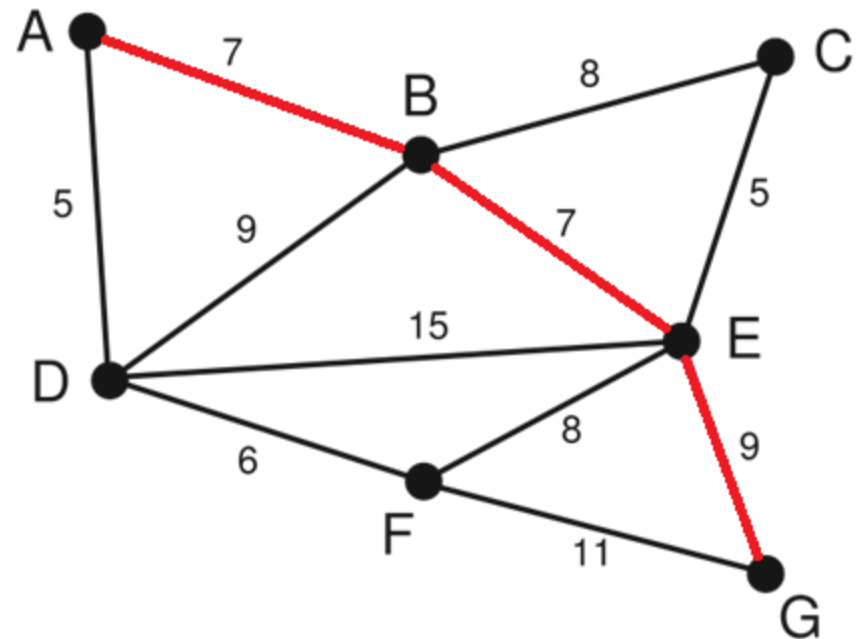
- postupnosť vrcholov grafu bez opakovania, v ktorej každé dva za sebou idúce vrcholy sú spojené hranou

- **Ohodnotenie cesty:**

- súčet ohodnotení hrán tvoriacich cestu
- označujeme aj ako „cena“ cesty alebo „dĺžka“ cesty

- Príklad:

- $c(ABEG) =$   
 $c(AB) + c(BE) + c(EG) =$   
 $7 + 7 + 9 = 23$



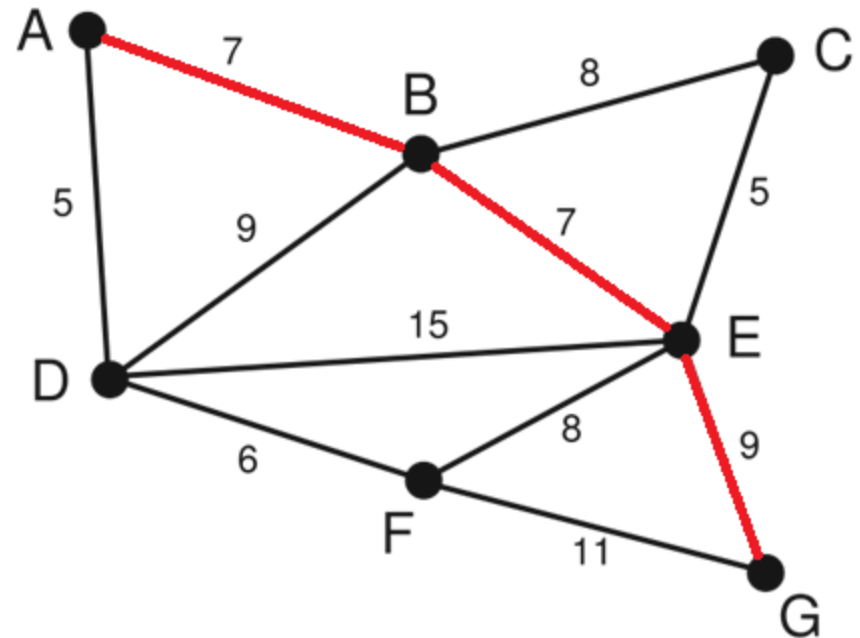


# Interpretácia ohodnotených ciest

- $c(AB)+c(BE)+c(EG) = 7 \text{ km} + 7 \text{ km} + 9 \text{ km} = 23 \text{ km}$
- $c(AB)+c(BE)+c(EG) = 7 \text{ min} + 7 \text{ min} + 9 \text{ min} = 23 \text{ min}$
- $c(AB)+c(BE)+c(EG) = 7 \text{ €} + 7 \text{ €} + 9 \text{ €} = 23 \text{ €}$

Dá sa z A do G  
dostať za menej  
ako 23 minút?

Dá sa z A do G  
dostať za lacnejšie  
ako 23 €?





# Problém najkratšej cesty

## ● Praktické aplikácie:

- GoogleMaps
- GPS navigácia
- Cestovné spojenia (MHD, autobusy, vlaky, lietadlá)

## ● Otázky:

- Aká je najkratšia cesta z Medickej na Jesennú?
- Kde je najbližšia reštaurácia? Ako sa tam dostanem?
- Zoznam všetkých čerpacích staníc do 20 km cesty.
- Aká je aktuálne najrýchlejšia cesta z KE do BA?
- Aká je najlacnejšia letenka z KE do Silicon Valley?



# Problém najkratšej cesty

- **ohodnotený orientovaný graf** popisujúci „cestnú“ sieť
  - ak je neorientovaný graf, každú hranu pridáme ako dva šípky
- počiatočný vrchol - **štart**
- koncový vrchol - **cieľ**
- hľadáme takú **orientovanú cestu** zo štartového vrcholu do cieľového vrcholu, že jej **cena** (ohodnotenie, dĺžka, ...) **je minimálna**
  - v porovnaní s BFS neminimalizujeme počet hrán, ale súčet ohodnotení hrán tvoriacich cestu:
    - priamy autobus Prešov - Londýn (1 hrana) vs. autobus Prešov - Košice, letecky Košice - Viedeň a Viedeň - Londýn (3 hrany)

**Dôležité!!!**

**Ako nájsť najkratšiu (najlacnejšiu) cestu v grafe?**



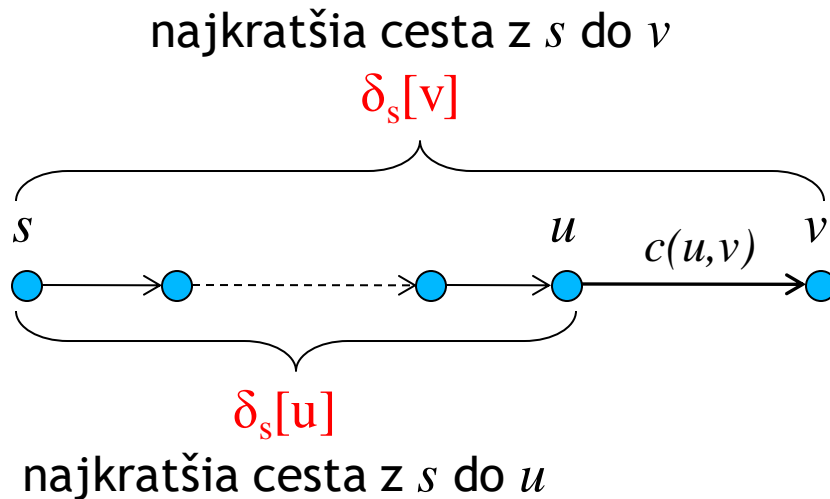
# Ako začať?

- Inšpirácia v dynamickom programovaní?
  - riešime všeobecnejší problém...
  - kopa úvah, ktorá skončí nakoniec pár cyklami...
- **Hľadáme dĺžku najkratšej cesty** zo štartovacieho vrcholu do **všetkých** vrcholov grafu
  - označme  $\delta_s[u]$  dĺžku najkratšej cesty zo štartovacieho vrcholu  $s$  do vrcholu  $u$
  - po vypočítaní  $\delta_s[u]$  pre všetky vrcholy **spätným prehľadávaním** nájdeme cestu (viac na cvičeniach)
    - **najkratších ciest** do vrcholu **môže byť viacero**, všetky ale majú rovnaké ohodnotenie („dĺžku“), ktoré je najmenšie možné



# Pozorovanie 1

- Ak  $sPuv$  je **najkratšia** cesta z  $s$  do  $v$ , potom  $sPu$  je najkratšia cesta z  $s$  do  $u$ .



## Dôsledok:

Ak  $u$  je susedný predchodca  $v$  na najkratšej ceste z  $s$  do  $v$ , potom:

$$\delta_s[u] + c(u, v) = \delta_s[v]$$

**Sporom:** Ak by existovala kratšia cesta z  $s$  do  $u$ , potom jej predĺžením o hranu  $(u, v)$  dostaneme kratšiu cestu z  $s$  do  $v$  - my sme ale vybrali najkratšiu - spor s optimalitou výberu



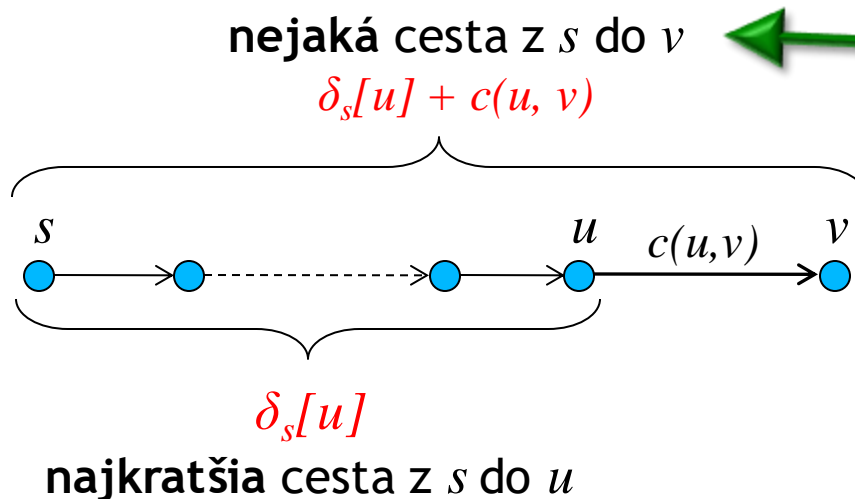
# Pozorovanie 2

- Pre každú hranu  $(u, v)$  platí:

$$\delta_s[u] + c(u, v) \geq \delta_s[v]$$

- Dôkaz:**

- cesta, ktorá vznikne predĺžením najkratšej cesty z  $s$  do  $u$  o hranu  $(u, v)$  je nejaká cesta z  $s$  do  $v$  - preto nemôže byť kratšia ako najkratšia cesta



Každá cesta z  $s$  do  $v$   
 (aj tá cez  $u$ ) je aspoň  
 taká dlhá ako  
 najkratšia cesta:  
 $\delta_s[u] + c(u, v) \geq \delta_s[v]$



# Idea algoritmu

- $\delta_s[v]$  - dĺžka najkratšej cesty z  $s$  do  $v$ 
  - túto hodnotu nepoznáme...
- $d_s[v]$  - „horný odhad“ dĺžky najkratšej cesty z vrcholu  $s$  do  $v$ 
  - nejaké číslo, *ktoré zodpovedá dĺžke nejakého sledu (cesta, kde je dovolené opakovanie vrcholov) z  $s$  do  $v$*
  - udržiavame ho pre každý vrchol grafu
  - chceme ho postupne zlepšovať (zmenšovať)

$$\delta_s[v] \leq d_s[v]$$

Žiaden horný odhad dĺžky najkratšej cesty nemôže byť menší ako skutočná dĺžka najkratšej cesty.





# Idea algoritmu

- $\delta_s[v]$  - dĺžka najkratšej cesty z  $s$  do  $v$
- $d_s[v]$  - „odhad“ dĺžky najkratšej cesty z  $s$  do  $v$ 
  - chceme, aby stále platit' podmienka:

$$\delta_s[v] \leq d_s[v]$$

**Invariant:** podmienka, ktorá stále platí...  
... aj keď jej platnosť nemáme ako overiť.

Na začiatku algoritmu:

$$d_s[s] = 0$$

$$d_s[v] = \infty, \text{ ak } s \neq v$$

Invariant je určite splnený, aj keď nepoznáme  $\delta_s[v]$



# Zlepšovanie odhadu?

- Moje odhady (splňajúce invarianty):
  - Košice -> Liptovský Mikuláš: 2000 km
  - Košice -> Poprad: 800 km

Z Popradu do  
Liptovského Mikuláša je  
diaľnica dlhá 60km



- Moje nové odhady:
  - Košice -> Liptovský Mikuláš: **860** km
  - Košice -> Poprad: 800 km



# Zlepšovanie odhadu?

- Moje odhady (splňajúce invarianty):
  - Košice -> Liptovský Mikuláš: 830 km
  - Košice -> Poprad: 800 km

Z Popradu do  
Liptovského Mikuláša je  
diaľnica dlhá 60km



- Moje nové odhady (informácia nepomohla):
  - Košice -> Liptovský Mikuláš: **830** km
  - Košice -> Poprad: **800** km



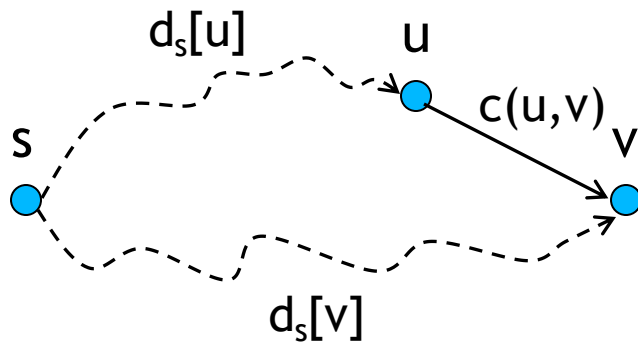
# Relaxácia hrany

- Operácia relaxácie hrany  $e=(u, v)$ :

Relax(u, v):

if ( $d_s[u] + c(u, v) < d_s[v]$ )

$d_s[v] = d_s[u] + c(u, v)$



## Intuícia:

Ak cesta z  $s$  do  $u$  a orientovaná hrana  $(u, v)$  vytvorí kratšiu cestu z  $s$  do  $v$ , ako máme, tak si môžeme zlepšiť odhad najkratšej cesty z  $s$  do  $v$  - máme lepšiu cestu z  $s$  do  $v$ .

Ostane invariant  $\delta_s[v] \leq d_s[v]$  zachovaný?



# Relaxácia zachováva invariant

Pozorovanie 2:  $\delta_s[u] + c(u, v) \geq \delta_s[v]$

Invariant pre vrchol  $u$ :  $\delta_s[u] \leq d_s[u]$

Invariant môže prestať pre vrchol  $v$  platiť, ak sa vykoná if:  
po jeho vykonaní  $d_s[v] = d_s[u] + c(u, v)$

**Tvrdenie:** relaxácia zachováva platnosť invariantu

$$\delta_s[v] \leq \delta_s[u] + c(u, v) \leq d_s[u] + c(u, v) = d_s[v]$$

Pozorovanie 2

Platnosť  
invariantu pre  
vrchol  $u$ :

$$\delta_s[u] \leq d_s[u]$$

Ak nastalo **priradenie**  
relaxácie, potom po jeho  
vykonaní platí:

$$d_s[v] = d_s[u] + c(u, v)$$



# Úvahy pokračujú...

- Vrchol  $v$  nazveme **vybaveným**, ak  $\delta_s[v] = d_s[v]$ 
  - na začiatku je iba štartovací vrchol  $s$  vybavený
- **Tvrdenie:** Ak  $sPuv$  je najkratšia cesta z  $s$  do  $v$  a vrchol  $u$  už je vybavený, tak po relaxácii hrany  $(u, v)$  sa stane vrchol  $v$  vybaveným.

Dôkaz namiesto sľubov...



# Úvahy pokračujú (dôkaz)...

## ● Fakty:

Pozorovanie 1

- $sP_{uv}$  je najkratšia cesta z  $s$  do  $v$ :  $\delta_s[u] + c(u, v) = \delta_s[v]$
- vrchol  $u$  už je vybavený:  $\delta_s[u] = d_s[u]$
- invariant pre  $v$ :  $\delta_s[v] \leq d_s[v]$
- po skončení relaxácie  $(u, v)$ :  $d_s[v] \leq d_s[u] + c(u, v)$

$$d_s[v] \leq d_s[u] + c(u, v) = \delta_s[u] + c(u, v) = \delta_s[v]$$

$$d_s[v] \leq \delta_s[v]$$

$$\delta_s[v] \leq d_s[v]$$

$$\delta_s[v] = d_s[v]$$

$v$  je vybavený



# O správnom poradí relaxácií

- Nech  $(s, v_1), (v_1, v_2), \dots, (v_{k-1}, v)$  je postupnosť hrán najkratšej cesty z  $s$  do  $v$ :
  - $s$  je vybavený, po relaxácií  $(s, v_1)$  je  $v_1$  vybavený
  - po tom, čo  $v_1$  je vybavený, po relaxácii  $(v_1, v_2)$  je  $v_2$  vybavený
  - ...
  - po tom, čo  $v_{k-1}$  je vybavený, po relaxácii  $(v_{k-1}, v)$  je  $v$  vybavený
- Záver: **správna postupnosť** relaxácií hrán garantuje, že všetky vrcholy sa stanú vybavené.







# Ako to všetko využiť?

- **Kľúčový fakt:** Ak  $u$  je vybavený a hrana  $(u, v)$  je na najkratšej ceste z  $s$  do  $v$ , po relaxovaní hrany  $(u, v)$  bude vrchol  $v$  vybavený.
- Vrchol  $s$  je na začiatku vybavený.

Ak zrelaxujeme každú orientovanú hrana v grafe, čo vieme povedať o množine vybavených vrcholov?

?

*Vybavené budú tie vrcholy, do ktorých vedie najkratšia cesta tvorená len jednou hranou (dĺžky 1).*



# Algoritmus Bellman-Ford

```

for (v: vrcholy G) {
    ds[v] = ∞;
}
ds[s] = 0;

```

**Intuícia:**  
 V rámci každej fázy algoritmu skúsime zrelaxovať všetky hrany grafu. Zrealizujeme celkom n fáz.

```

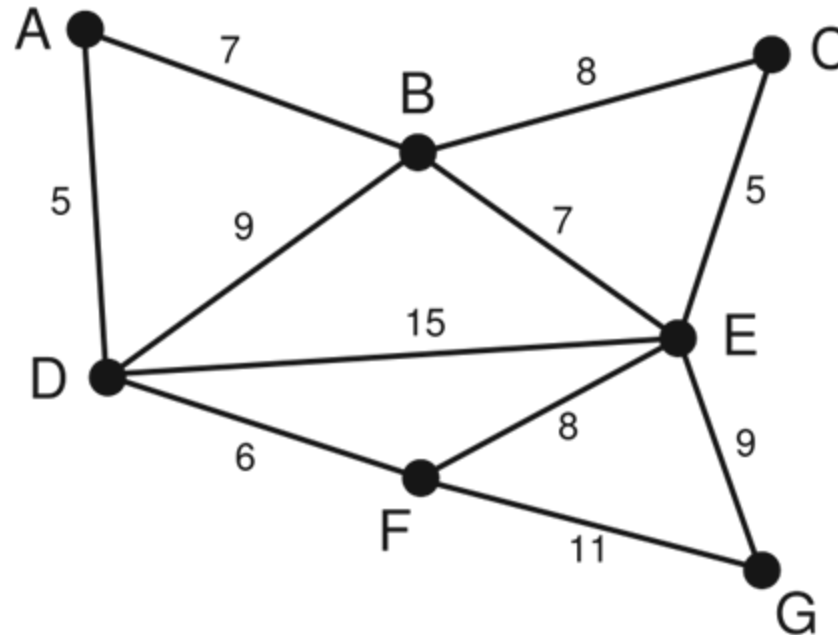
for (int i=0; i<n; i++)
    for (Edge e: graph.getEdges())
        relax(e);

```

**Dôkaz intuitívne:**  
 Po i-tej fáze sa stanú vybavené všetky také vrcholy, že najkratšia cesta zo štartovacieho vrcholu s do nich sa skladá z i hrán. Formálny dôkaz indukciou...



# Bellman-Ford z vrcholu A



A	B	C	D	E	F	G
0	7	15	5	14	11	22



# Bellman-Ford v Java

```
public void relax(Edge e, Map<Vertex, Double> d) {  
    Vertex u = e.getSource();  
    Vertex v = e.getTarget();  
    if (d.get(u) + e.getWeight() < d.get(v))  
        d.put(v, d.get(u) + e.getWeight());  
}
```

```
public Map<Vertex, Double> bellmanFord(Graph g, Vertex s) {  
    Map<Vertex, Double> d =  
        g.createVertexMap(v, Double.POSITIVE_INFINITY);  
    d.put(s, 0d);  
  
    for (int i = 0; i < g.getVertices().size(); i++)  
        for (Edge e : g.getEdges())  
            relax(e, d);  
  
    return d;  
}
```



# Algoritmus Bellman-Ford

## ● Intutívne zdôvodnenie:

- Využitý fakt: ak predposledný vrchol na najkratšej ceste je *vybavený*, tak relaxáciou poslednej hrany sa stane posledný vrchol cesty *vybavený*.
- Po  $i$ -tej relaxácii všetkých hrán grafu sú *vybavené* všetky vrcholy, do ktorých vedie najkratšia cesta skladajúca sa z  $i$  hrán.
- Po  $(i+1)$ -tej relaxácii všetkých hrán grafu sú *vybavené* všetky vrcholy, do ktorých vedie najkratšia cesta skladajúca sa z  $i+1$  hrán. Tieto vrcholy **musia mať suseda**, do ktorého je najkratšia cesta z  $i$  hrán - na základe indukčného predpokladu, je ten už *vybavený*.



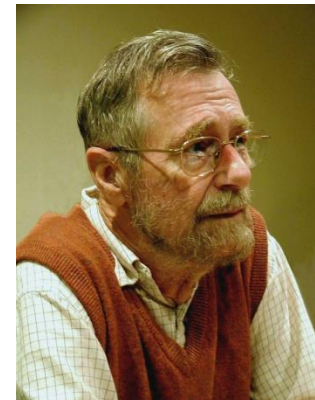
# Algoritmus Bellman–Ford

- **Časová zložitosť'** pri grafe s  $n$  vrcholmi a  $m$  hranami:
  - relaxácia jednej hrany  $O(1)$
  - v každej fáze relaxujeme všetky hrany  $m \cdot O(1) = O(m)$
  - celkovo máme  $n$  fáz:  $n \cdot O(m) = O(n \cdot m)$
- V grafoch je  $m = O(n^2)$ 
  - časová zložitosť' vzhľadom na  $n$ :  $O(n^3)$
- Výhoda:
  - nájdeme najkratšiu cestu **z  $s$  do všetkých vrcholov**



# Dijkstrov algoritmus (1959)

- Funguje **len** v prípadoch, kedy sú **ohodnotenia hrán nezáporné**
- Chytřejším výberom hrán na relaxáciu sa získa časová zložitost'  **$O(n^2)$** 
  - algoritmus bude vybavovat' vrcholy **v rastúcej postupnosti dĺžok** najkratších ciest - možnosť ukončiť algoritmus ihneď, ako máme to, čo hľadáme
- Autor:  
**Edsger Wybe Dijkstra**  
(Holandsko, 1930-2002)





# Dijkstrov algoritmus

$d_s[s] = 0;$

**for** (v: vrcholy G okrem s)

$d_s[v] = \infty;$

Q je množina „zatiaľ nevybavených“ vrcholov

Q = vrcholy G;

**while** (!Q.isEmpty()) {

vyber v z Q taký, že

$d_s[v] = \min\{d_s[u] \mid u \text{ patrí do } Q\}$

**for** (w: susedia vrchoľu v)

**relax**(v, w);

}

Zrelaxujeme všetky hrany vychádzajúce z v





# Invariant Dijkstrovho alg. (1)

## ● Invariant:

- pre každý vrchol mimo  $Q$  platí, že je *vybavený* a všetky z neho vychádzajúce hrany boli relaxované až po tom, čo bol *vybavený*.
- ak dokážeme, že každý vrchol odchádzajúci z  $Q$  je vybavený už pri svojom odchode z  $Q$ , **zelená časť** invariantu vyplýva z algoritmu

## ● Dôkaz (sporom):

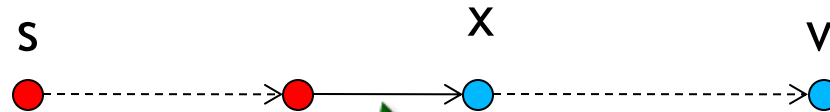
- nech  $v$  je prvý vrchol mimo  $Q$ , pre ktorý neplatí invariant, a v dobe, keď opúšťal  $Q$ , bol nevybavený, t.j.  $\delta_s[v] < d_s[v]$ .



# Invariant Dijkstrovho alg. (2)

**Dôkaz** (pozrime sa na okamih, keď  $v$  opúšťa  $Q$ ):

Keďže  $v$  nie je *vybavený*, existuje **kratšia** (najkratšia) cesta z  $s$  do  $v$ , v ktorej sa nachádza nejaký vrchol z  $Q$  - nech  $x$  je **prvý taký** vrchol na tejto ceste.



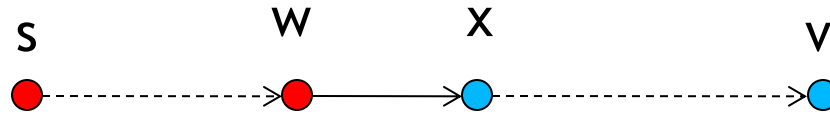
Vrcholy **mimo**  $Q$  - už vybavené ( $s$  je po prvom kroku určite mimo  $Q$ )

Vrcholy **v**  $Q$

Niekde musí byť hrana, kde začiatok je **mimo**  $Q$  a koniec je **v**  $Q$



# Invariant Dijkstrovho alg. (3)



- keďže  $x$  je prvý na ceste platí (invariant pre  $w$  a Pozorovanie 1), že  $x$  je vybavený  $d_s[x] = \delta_s[x]$
- z nezápornosti cien:  $\delta_s[x] \leq \delta_s[v]$
- invariant relaxácie:  $\delta_s[v] \leq d_s[v]$
- minimalita  $v$  medzi vrcholmi z  $Q$ :  $d_s[v] \leq d_s[x]$

$$d_s[x] = \delta_s[x] \leq \delta_s[v] \leq d_s[v] \leq d_s[x]$$

dôsledok:  $\delta_s[v] = d_s[v]$ , t.j.,  $v$  je vybavený



# Poznámky k „Dijkstraovi“

- Časová zložitost'  $O(n^2)$ :
  - **while** cyklus je vykonaný  $n$ -krát
  - v tele cyklu:
    - $O(n)$  na nájdenie vrcholu  $v$  s minimálnym  $d_s[v]$
    - $O(n)$  na relaxáciu hrán vychádzajúcich z  $v$
- Vizualizácia:
  - <http://cam.zcu.cz/~rkuzel/aplety/Dijkstra/Dijkstra.html>
  - <http://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html>
  - <http://www.unf.edu/~wkloster/foundations/DijkstraApplet/DijkstraApplet.htm>



# Floyd-Warshallov algoritmus

- Algoritmus na nájdenie **najkratšej cesty** medzi **každými 2 vrcholmi grafu**
  - založený na dynamickom programovaní
- **Vstup:**
  - ohodnotený orientovaný graf s vrcholmi číslovanými od  $1$  po  $n$
- **Výstup**
  - matica  **$d[i, j]$** , ktorá pre každú dvojicu vrcholov  $i$  a  $j$  obsahuje dĺžku najkratšej cesty medzi týmito vrcholmi



# Floyd-Warshall a dynamika

- $d[i, j, k]$  - dĺžka najkratšej cesty z  $i$  do  $j$ , ktorá používa len vrcholy  $1..k$ 
  - metafora...
- Triviálne prípady:
  - $d[i, j, 0] = c(i, j)$ , ak  $(i, j)$  je hrana grafu
  - $d[i, j, 0] = \infty$ , inak
- Zaujíma nás  $d[i, j, n] = d[i, j]$





# Charakterizácia problému

$$d[i, j, k] = \min\{d[i, j, k-1], d[i, k, k-1] + d[k, j, k-1]\}$$

Dĺžka najkratšej cesty z  $i$  do  $j$  pri ktorej máme dovolené navštíviť len vrcholy **1..k**.

Máme dve možnosti: táto cesta alebo **obsahuje** alebo **neobsahuje** vrchol  $k$

Dĺžka najkratšej cesty z  $i$  do  $j$  pri ktorej máme dovolené navštíviť len vrcholy **1..k-1**

Ak najkratšia cesta z  $i$  do  $j$  cez vrcholy **1..k** obsahuje mesto  $k$ , potom cestu môžeme rozdeliť na dve podcesty:  
z  $i$  do  $k$  cez vrcholy **1..k-1** a  
z  $k$  do  $j$  cez vrcholy **1..k-1**



# Pozorovania

$$d[i, j, k] = \min\{d[i, j, k-1], d[i, k, k-1] + d[k, j, k-1]\}$$

## Pozorovania:

- na zlepšenie hodnoty  $d[*, *, k]$  potrebujeme iba hodnoty tvaru  $d[*, k, k-1]$ ,  $d[k, *, k-1]$
- $d[*, k, k] = d[*, k, k-1]$ ,  $d[k, *, k] = d[k, *, k-1]$ 
  - možnosť použiť vrchol  $k$  ako medzivrchol nepomôže nájsť lepšiu cestu s koncom vo vrchole  $k$

## Dôsledok:

nepotrebujeme 3-rozmerné pole, stačí 2-rozmerné





# Floyd-Warshall v Java

Iniciálne  $d[i, j]$  obsahuje:

- $c(i, j)$  - cenu hrany z  $i$  do  $j$ , ak existuje
- `Double.POSITIVE_INFINITY`, ak neexistuje

Časová zložitosť:  
 $O(n^3)$

```
for (int k=0; k<vrcholy.length; k++)
    for (int i=0; i<vrcholy.length; i++)
        for (int j=0; j<vrcholy.length; j++)
            if (d[i][k] + d[k][j] < d[i][j])
                d[i][j] = d[i][k] + d[k][j];
```



# Sumarizácia

- Ohodnotené grafy na lepšie zachytenie sveta...
- Hľadanie najkratších ciest (B-F, Dijkstra):
  - spoločné myšlienky:
    - postupne zlepšujeme horný odhad dĺžky najkratšej cesty do každého vrcholu
    - operácia relaxácie hrany na zlepšovanie odhadov
- Hľadanie najkratších ciest (Floyd-Warshall):
  - dynamické programovanie - idea:
    - na začiatku môžeme použiť na cestovanie len priame hrany („bez prestupov“)
    - ako sa zmenia dĺžky najkratších ciest, ak nám niekto dovolí používať nejaký nový vrchol ako „prestup“ na cestách?