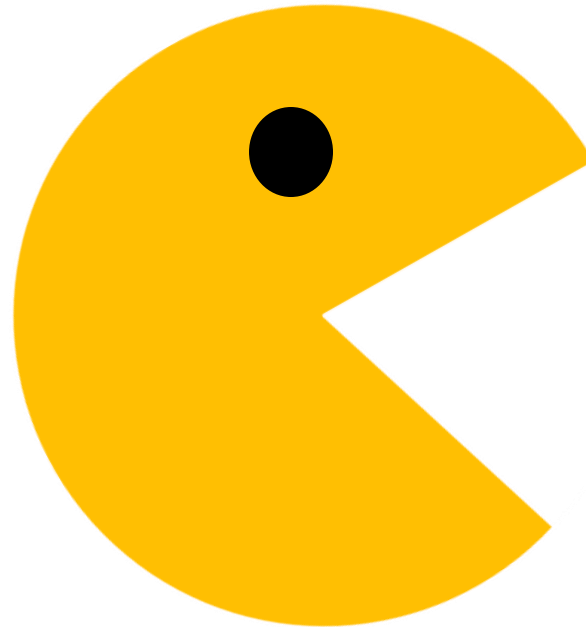




10. prednáška

Greedy algoritmy





Obsah



- **GREEDY**

- **chamtivý**
- **nenásytný**
- **pažravý**



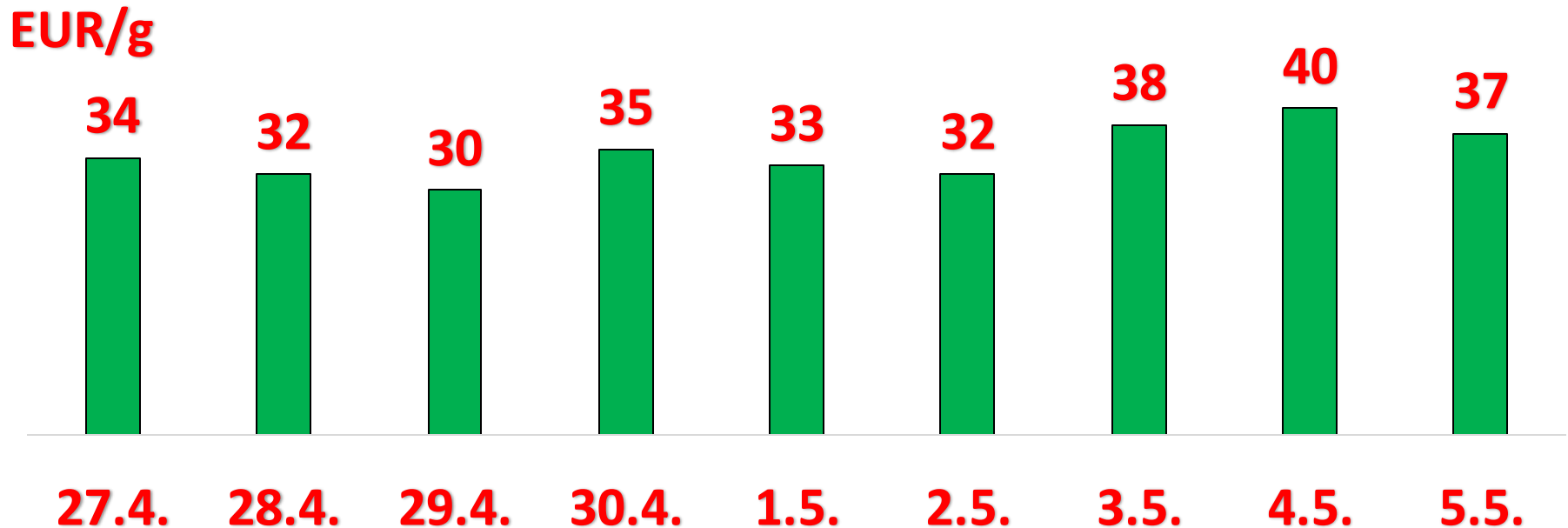
- **Greedy stratégia, greedy algoritmus**
- **Minimálna kostra grafu**
- **Úloha o zastávkach autobusu**
- **Jednoduchý rozvrhový problém**
- **Problém výberu úloh**
- **Problém plnenia batoha**
- **...**



Motivácia

Príklad 1.

Superchytrá umelá inteligencia, ktorá sa nikdy nemýli, nám predpovedala, ako sa bude vyvíjať cena zlata v nasledujúcich dňoch:

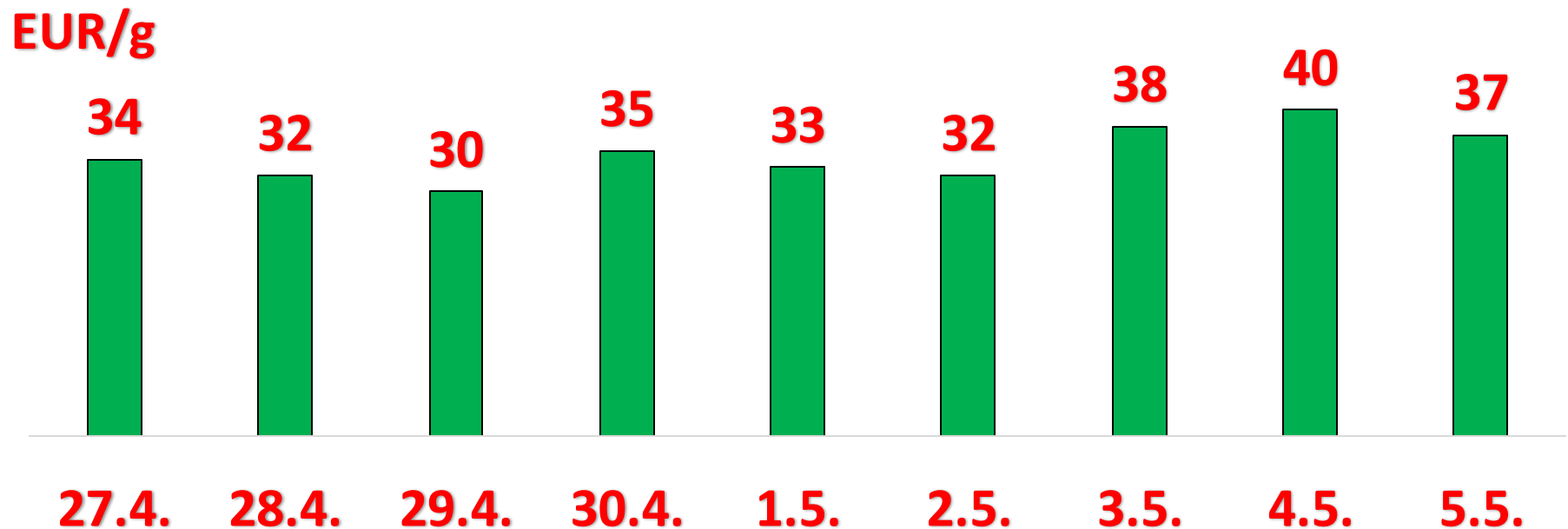




Motivácia

Príklad 1.

- Máme v hotovosti **300 eur**.
- Ako s ním obchodovať, aby sme si čo najviac zarobili?



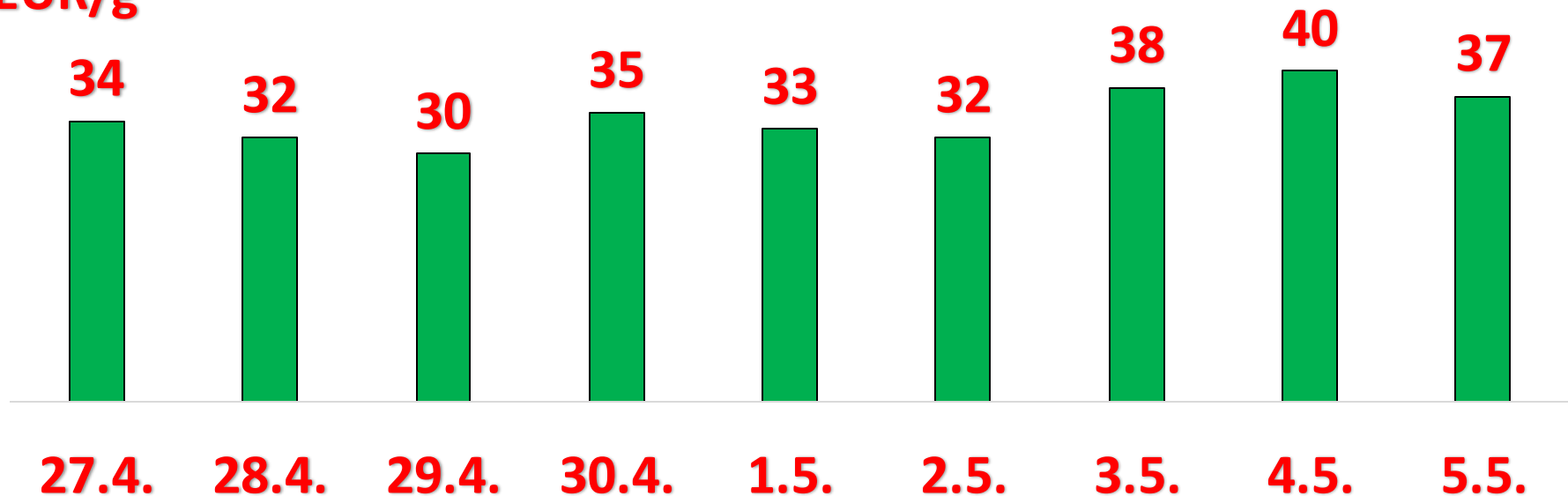


Motivácia

Príklad 1.

- Vieme získať na konci posledného dňa **400 EUR**?
- A dá sa dosiahnuť ešte viac?
- **Ako postupovať optimálne?**

EUR/g

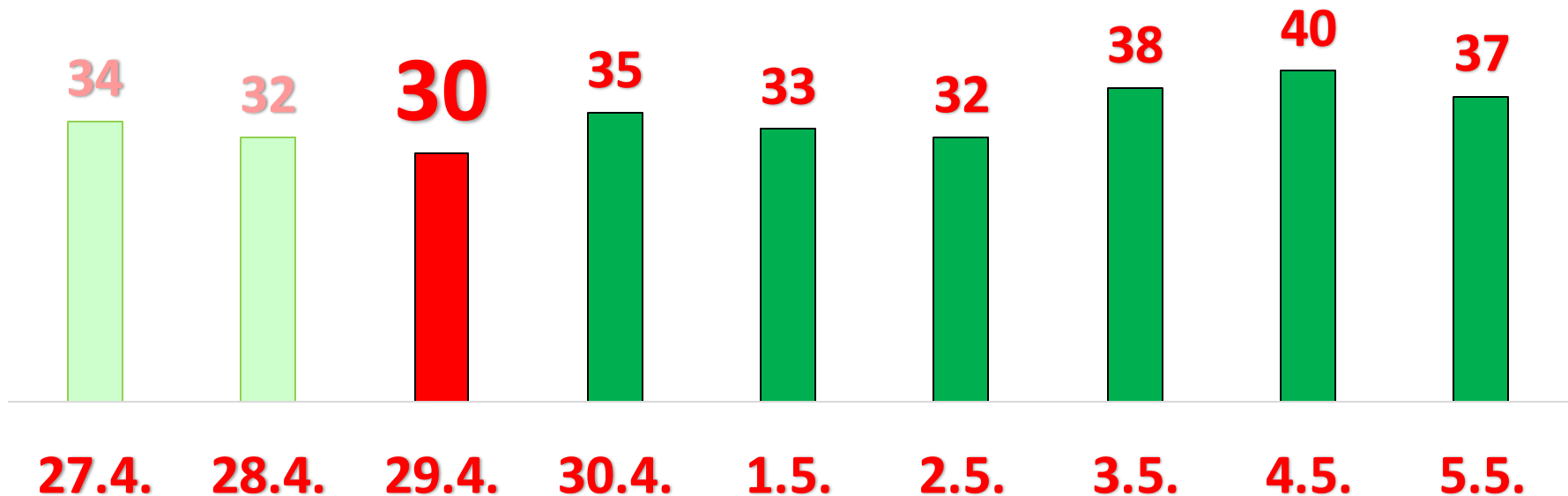




Motivácia

Príklad 1.

- Počkáme, kým cena klesne na 30 EUR/g.
- Vtedy nakúpime za všetky peniaze 10 gramov zlata.
- 300 EUR = 10 g x 30 EUR/g**

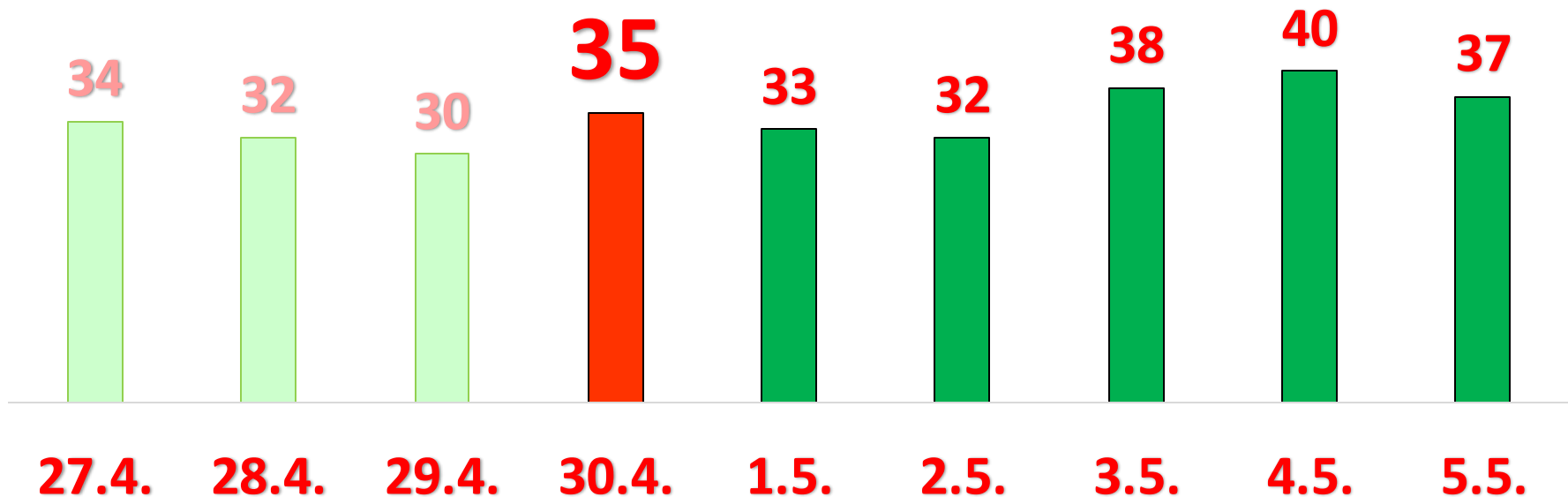




Motivácia

Príklad 1.

- Na druhý deň ho zase všetko predáme.
- **10 g x 35 EUR/g = 350 EUR**
- takže budeme mať 350 eur





Pozorovanie

- V našej stratégii na riešenie úlohy sa striedajú dva kroky:
 - počkáme na lacné zlato a nakúpime;**
 - počkáme na drahé zlato a predáme.**
- Ako exaktne definovať, čo je „lacné zlato“, a teda kedy nakupovať a kedy predávať?



Greedy stratégia

- Cena zlata sa mení v noci.
- Teda každý večer môžeme **pažravo (nenásytne) rozhodnúť**, či chceme zlato alebo peniaze, a podľa toho nakúpiť alebo predat'.

Dôležitý poznatok:

- Stretávame sa so situáciou, kedy sme **globálne optimálne riešenie** zostrojili tak, že sme postupne urobili **niekoľko lokálne optimálnych rozhodnutí**.



Motivácia

Príklad 2.

- Predpokladajme, že budeme platiť mincami.
- Naše mince majú hodnoty

25¢, 10¢, 5¢ a 1¢

máme ich dost'

chceme zaplatiť **63¢**.

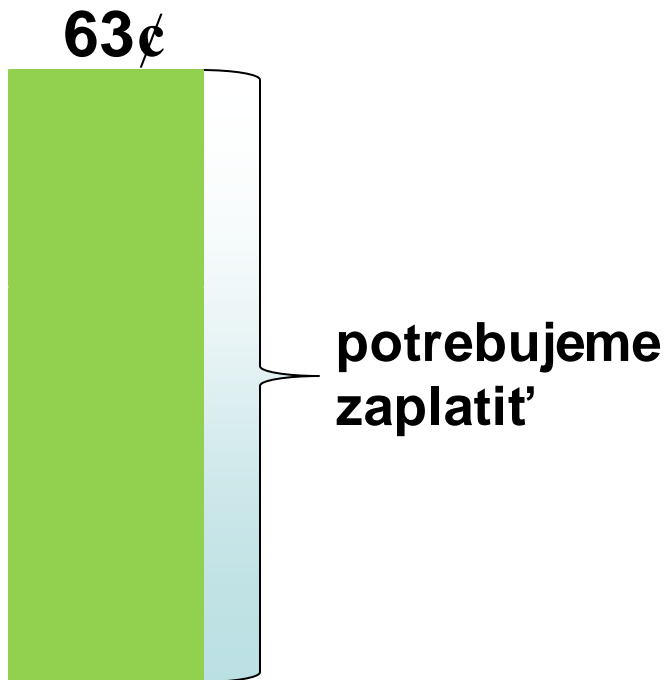


- Pri platbe chceme použiť čo najmenej mincí!



Greedy stratégia

Zvolíme nasledujúci algoritmus:



hodnoty
mincí



Doteraz použité mince:



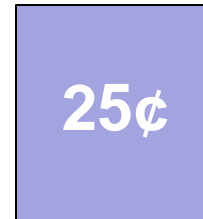
Greedy stratégia

Zvolíme nasledujúci algoritmus:

- Zaplatíme najväčšou mincou nie väčšou ako 63¢, pridáme ju do zoznamu mincí, ktorými platíme a odpočítame jej hodnotu od 63¢ a dostaneme, že nám ostáva nám zaplatiť ešte 38¢;



hodnoty
mincí



Doteraz použité mince:

25¢



Greedy stratégia

Zvolíme nasledujúci algoritmus:

- Potom vyberieme najväčšiu mincu, ktorej hodnota nie je väčšia ako 38¢, pridáme do zoznamu mincí, ktorými platíme a ostáva nám zaplatiť 13¢;

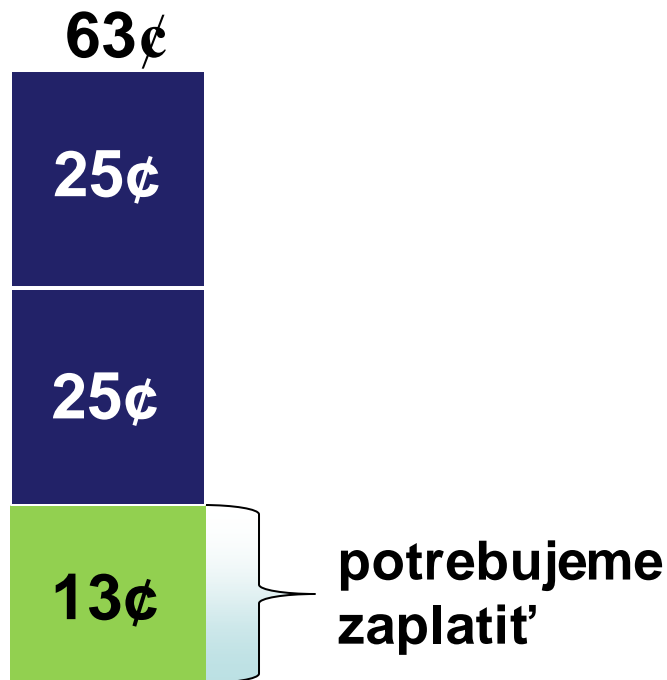
hodnoty
mincí

25¢

10¢

5¢

1¢



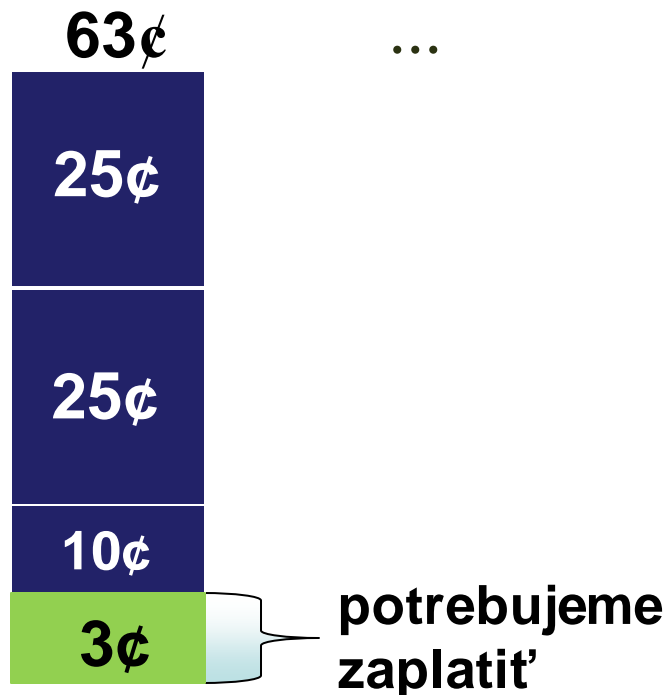
Doteraz použité mince:
25¢, 25¢



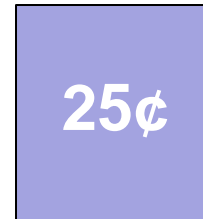
Greedy stratégia

Zvolíme nasledujúci algoritmus:

- Opäť vyberieme najväčšiu mincu, ktorej hodnota nie je väčšia ako 13¢, pridáme do zoznamu mincí, ktorými platíme a ostáva nám zaplatiť ešte 3¢;



hodnoty
mincí



Doteraz použité mince:

25¢, 25¢, 10¢



Greedy stratégia

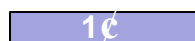
Zvolíme nasledujúci algoritmus:

- ... takýmto postupom následne vyberieme ešte trikrát mincu v hodnote 1¢ a získali sme riešenie.



v tomto prípade greedy stratégia
poskytne optimálne riešenie
 vďaka vhodným hodnotám mincí

hodnoty
mincí



Doteraz použité mince:

25¢, 25¢, 10¢, 1¢, 1¢, 1¢



Greedy stratégia

Zmena !

- Naše mince majú teraz hodnoty

11~~€~~, 5~~€~~ a 1~~€~~

máme ich dosť

chceme zaplatiť **15~~€~~**.

hodnoty
mincí

11~~€~~

5~~€~~

1~~€~~



Greedy stratégia

Zmena !

- podľa tejto stratégie by sme vybrali:

15€

11€

1€

1€

1€

1€

5 mincí

hodnoty
mincí

11€

5€

1€



Greedy stratégia

Zmena !

- podľa tejto stratégie by sme vybrali:

15€

11€

1€

1€

1€

5 mincí

Toto nie je optimálne riešenie

hodnoty
mincí

11€

5€

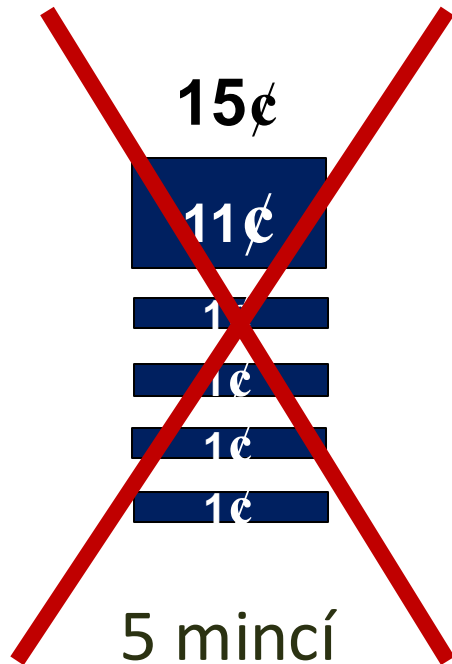
1€



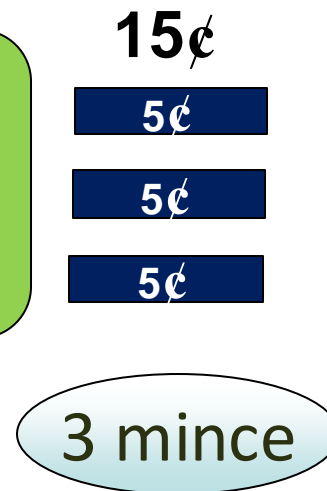
Greedy stratégia

Zmena !

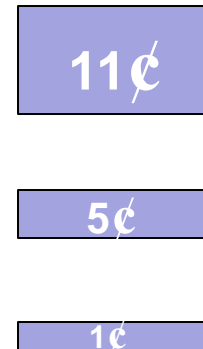
- podľa tejto stratégie by sme vybrali:



Existuje
riešenie
s menším
počtom mincí



hodnoty
mincí





Greedy algoritmy

Greedy algoritmy riešia **optimalizačné problémy** pomocou postupností výberov položiek do riešenia.

Výbery musia byť:

- **Realizovateľné**
 - musia zachovať obmedzenia problému
- **Lokálne optimálne**
 - musia poskytovať najlepší lokálny výber medzi všetkými možnými výbermi v danom kroku
- **Nezrušiteľné**
 - raz urobený výber prvkov do postupnosti je nezmeniteľný, nedá sa zobrať späť a opraviť



Greedy algoritmy

Pre niektoré optimalizačné problémy „greedy“ prístup nedáva optimálny výsledok.

videli sme to v
poslednom príklade

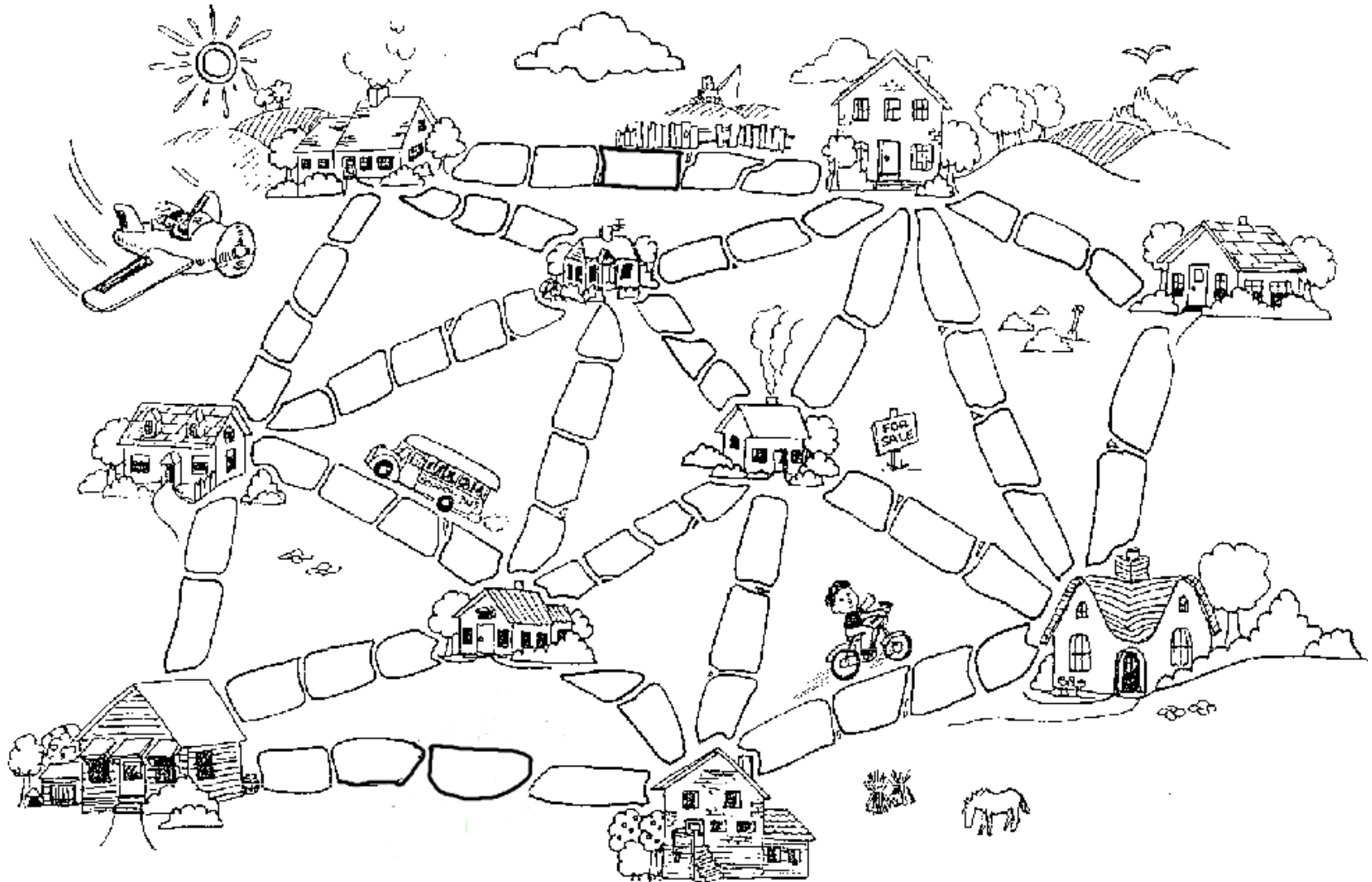


Čo sa deje

- Každá iterácia v greedy algoritme pozostáva z nasledujúcich častí:
 - **Výber** - vyberá ďalšiu položku
 - **Kontrola realizovateľnosti** - výber položky tak, aby spĺňala obmedzenia problému
 - **Kontrola lokálneho optima** - overuje, či výber vytvára lokálne optimálne riešenie
 - **Kontrola riešenia** - zisťuje, či je globálne riešenie už dosiahnuté alebo nie



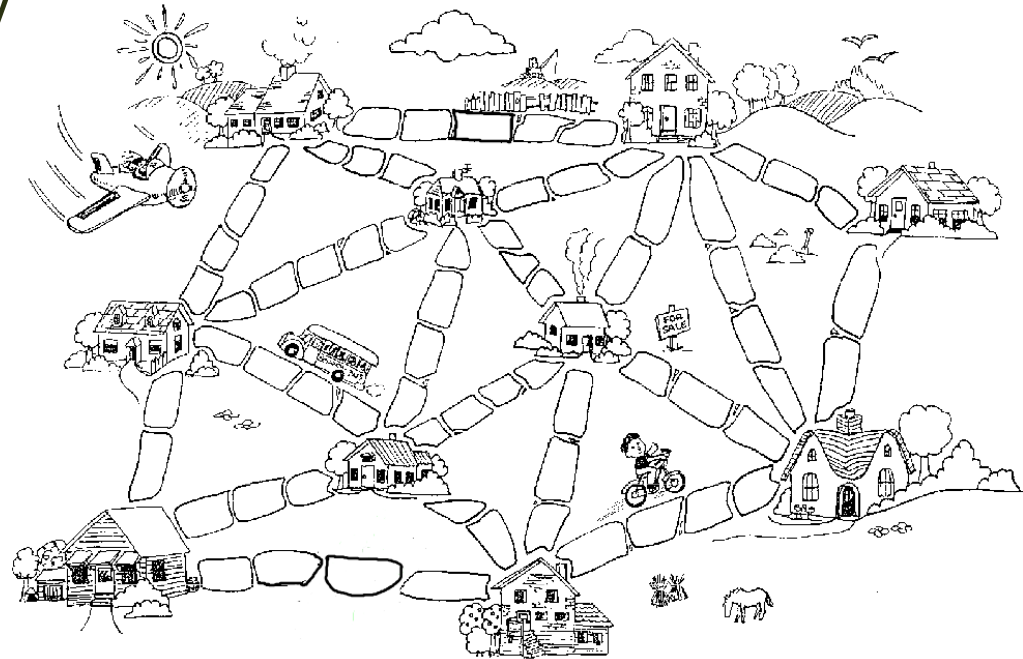
Úloha o dláždení





Úloha o dláždení

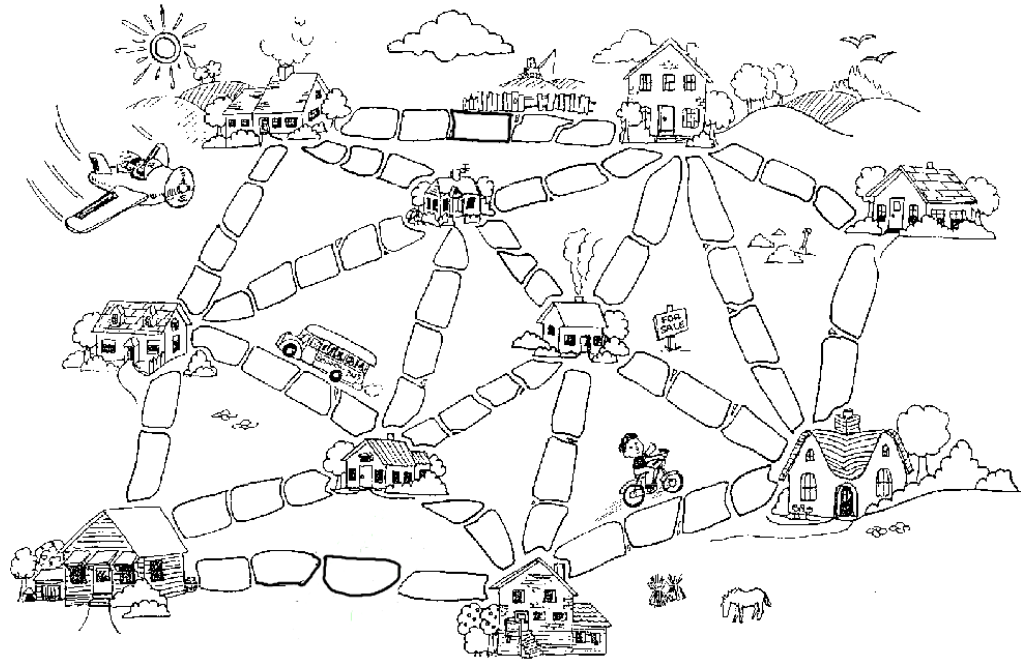
- V odľahlej malej horskej obci chce starosta konečne vyriešiť problém s cestou medzi jednotlivými domami.
 - domy nie sú umiestnené na jednej ulici, ale sú od seba vzdialené rôzne vzdialenosti a cestičky sú len vychodené, bez povrchovej úpravy





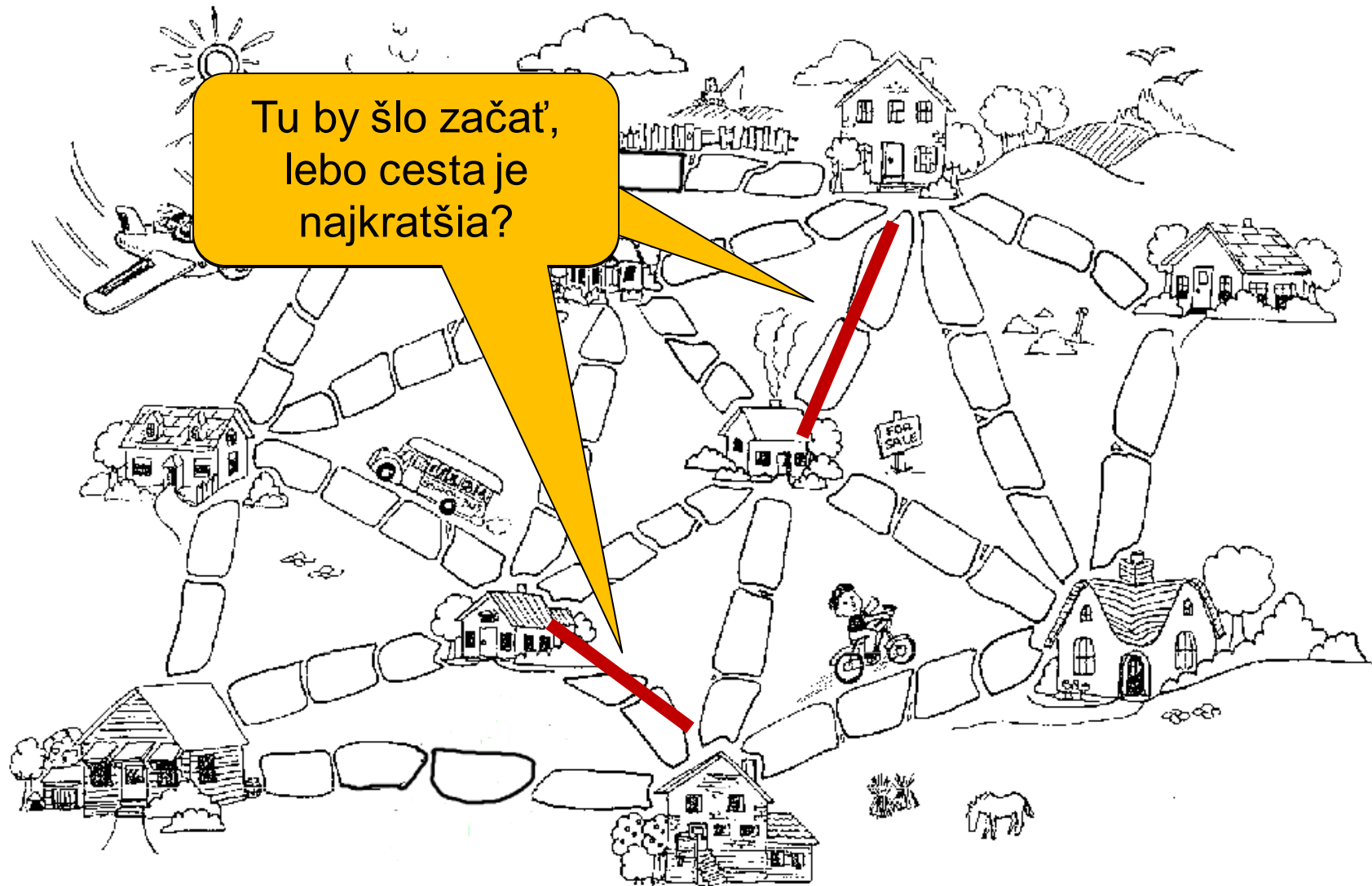
Úloha o dláždení

- Starosta chce vybudovať nové dláždené cesty medzi niektorými domami tak, aby:
 - sa dalo medzi ľubovoľnými dvoma domami prejsť po novej ceste,
 - ale zároveň chce, aby budoval čím menej metrov ciest.
- Stojí pred otázkou medzi ktorými domami ich má vybudovať?





Úloha o dláždení

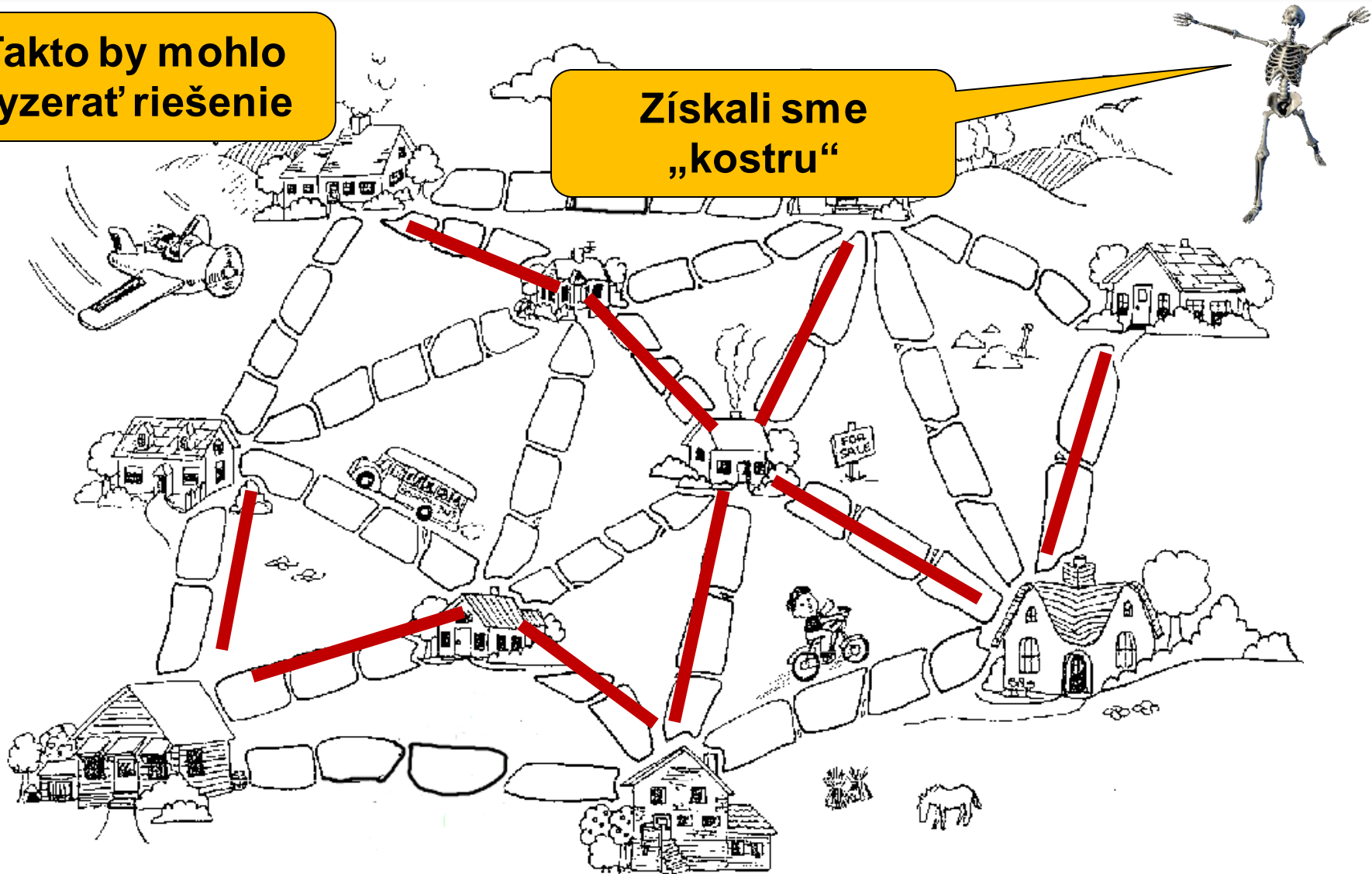




Úloha o dláždení

Takto by mohlo
vyzerat' riešenie

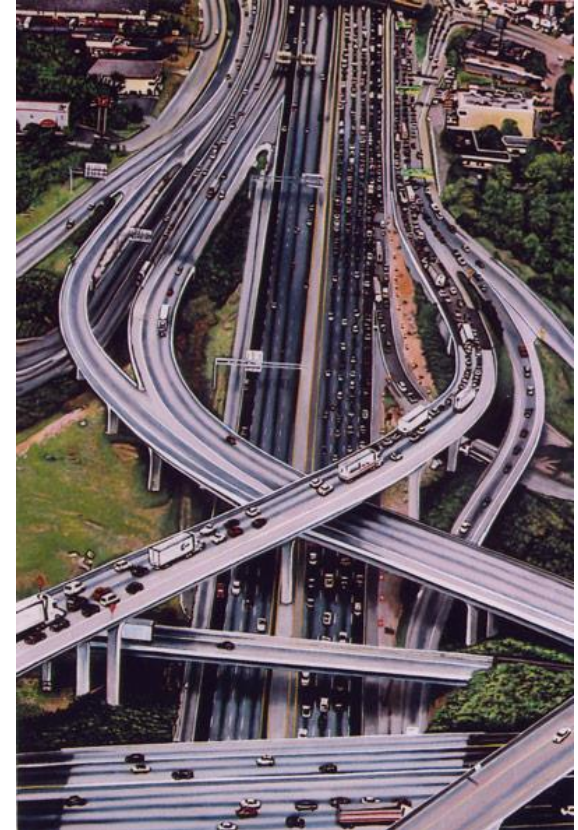
Získali sme
„kostru“





Minimálna kostra – motivácia

- Uvažujme **dopravnú sieť**
- Máme nejaké **peniaze** z EÚ na výstavbu diaľnic
- **Ktoré** cesty máme prerobiť na diaľnice, aby existovalo (nie nutne priame) **spojenie medzi každými 2 mestami** výhradne po diaľniciach a pritom aby sme minuli, čo **najmenej financií**
- **Vstup:** pre každý úsek spájajúci 2 mestá sú dané náklady na jeho prestavbu na diaľnicu





Iné praktické motivácie



- **Telefónna sieť:** ktoré linky prebudovať na optické, aby bolo možné presmerovať všetky hovory po optických linkách a chceme pritom minúť čo najmenej financií?
- Algoritmus na riešenie ako prvý navrhol v roku 1926 Otakar Borůvka, keď riešil **problém efektívnej konštrukcie elektrickej siete** na Morave.

Zdroj:https://encyklopedie.brna.cz/home-mmb/?acc=profil_osobnosti&load=2471



Minimálna kostra grafu

- **Kostra (spanning tree)** súvislého grafu G :
súvislý acyklický podgraf grafu G ,
ktorý obsahuje všetky vrcholy grafu G

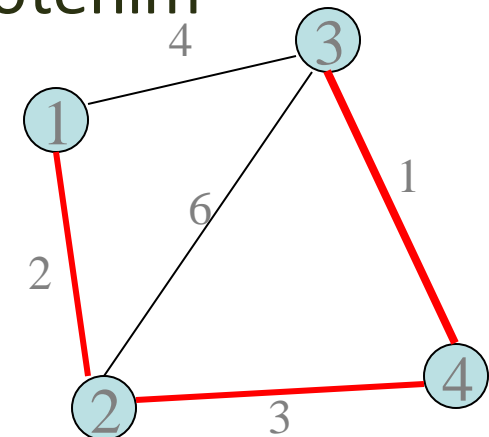


- **Minimálna kostra (minimal spanning tree)** ohodnoteného súvislého grafu G :

kostra grafu G s minimálnym ohodnotením

Príklad:

- minimálna kostra
– hrany s ohodnotením 1, 2, 3





Vlastnosti kostier (bez dôkazu)

- Graf môže mať **veľa kostier**.
- Hrany kostry **nevytvárajú cyklus**.
- Každá kostra grafu s n vrcholmi má práve $n-1$ hrán.
- Pridanie ľubovoľnej nekostrovej hrany ku kostre **vytvorí cyklus**.
- Medzi každými 2 vrcholmi grafu existuje **jediná cesta** využívajúca len kostrové hrany.





Minimálna kostra grafu



- **Vstup:**

súvislý, neorientovaný, ohodnotený graf;

- **Problém:**

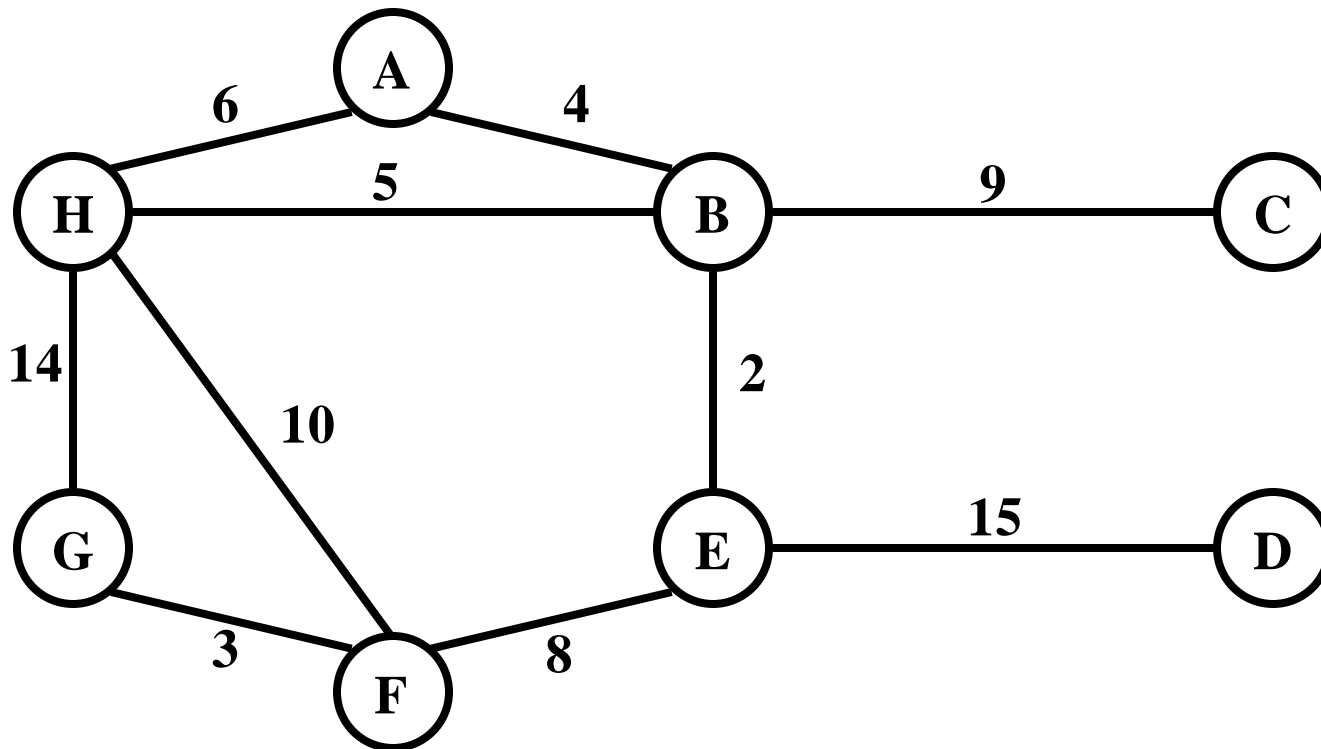
nájsť minimálnu kostru grafu;

použitím hrán, ktoré minimalizujú celkové ohodnotenie.



Minimálna kostra grafu

- Ktoré hrany tvoria minimálnu kostru grafu na tomto obrázku?

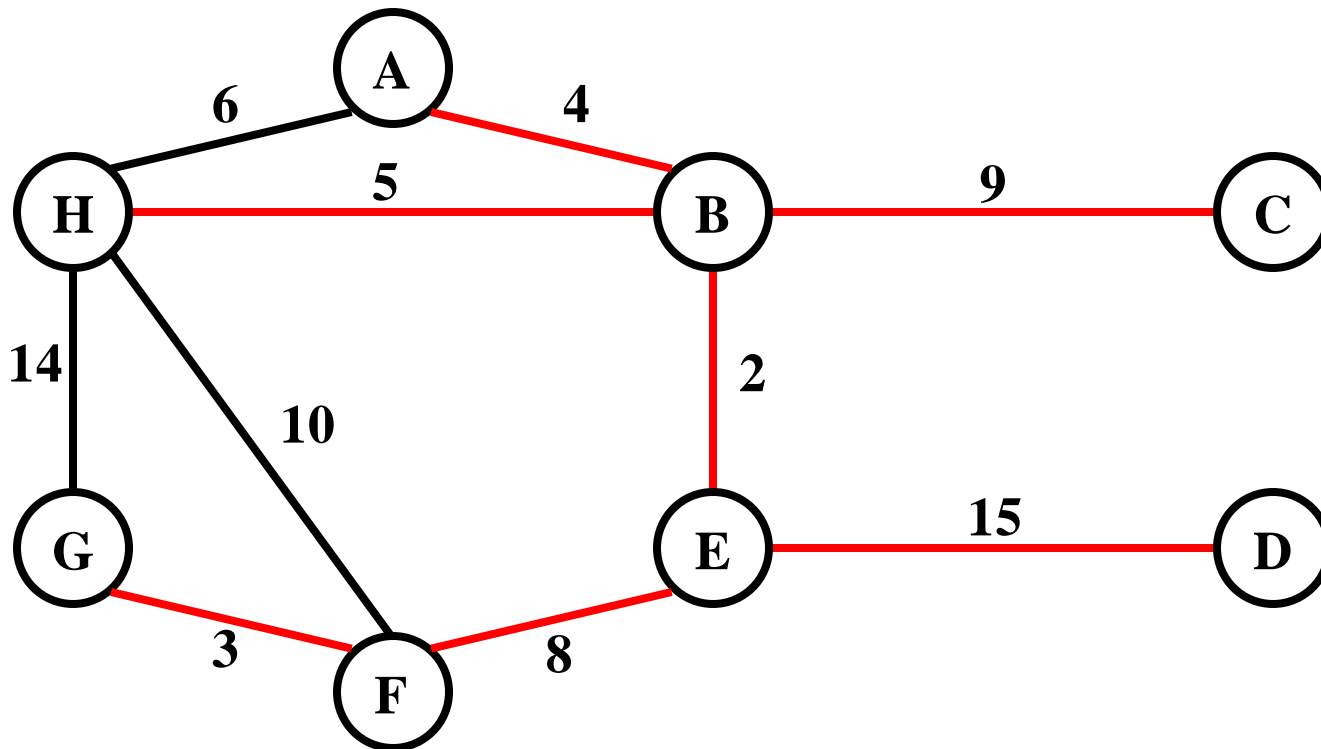




Minimálna kostra grafu

● Odpoveď:

● Cena kostry: ?





Primov (Jarník, Dijkstra) algoritmus

- Tvorbu kostry začneme jedným vrcholom - strom T_1 .
 - V každom kroku skonštruujeme T_{i+1} z T_i :
 - pridáme hranu s minimálnym ohodnotením vychádzajúcu z vrcholu v strome T_i a vedúcu do vrcholu, ktorý ešte nie je v strome
- „greedy” krok:
výber z „okrajových” hrán
s minimálnym ohodnotením**
- Takto konštruujeme postupnosť expandujúcich stromov T_1, T_2, \dots
 - Algoritmus sa zastaví, keď sú do kostry pridané všetky vrcholy.



Primov algoritmus

```

ds[s] = 0;
p[s] = NULL;
for (v: vrcholy G okrem s)
    ds[v] = ∞;
Q = vrcholy G;
while (!Q.isEmpty()) {
    vyber v z Q taký, že
        ds[v] = min{ds[u] | u patrí do Q}
    for (w: susedia vrchoľu v)
        if (w ∈ Q and c(v,w) < ds[w] ) {
            p[w] = v;
            ds[w] = c(v,w);
        }
    }
}

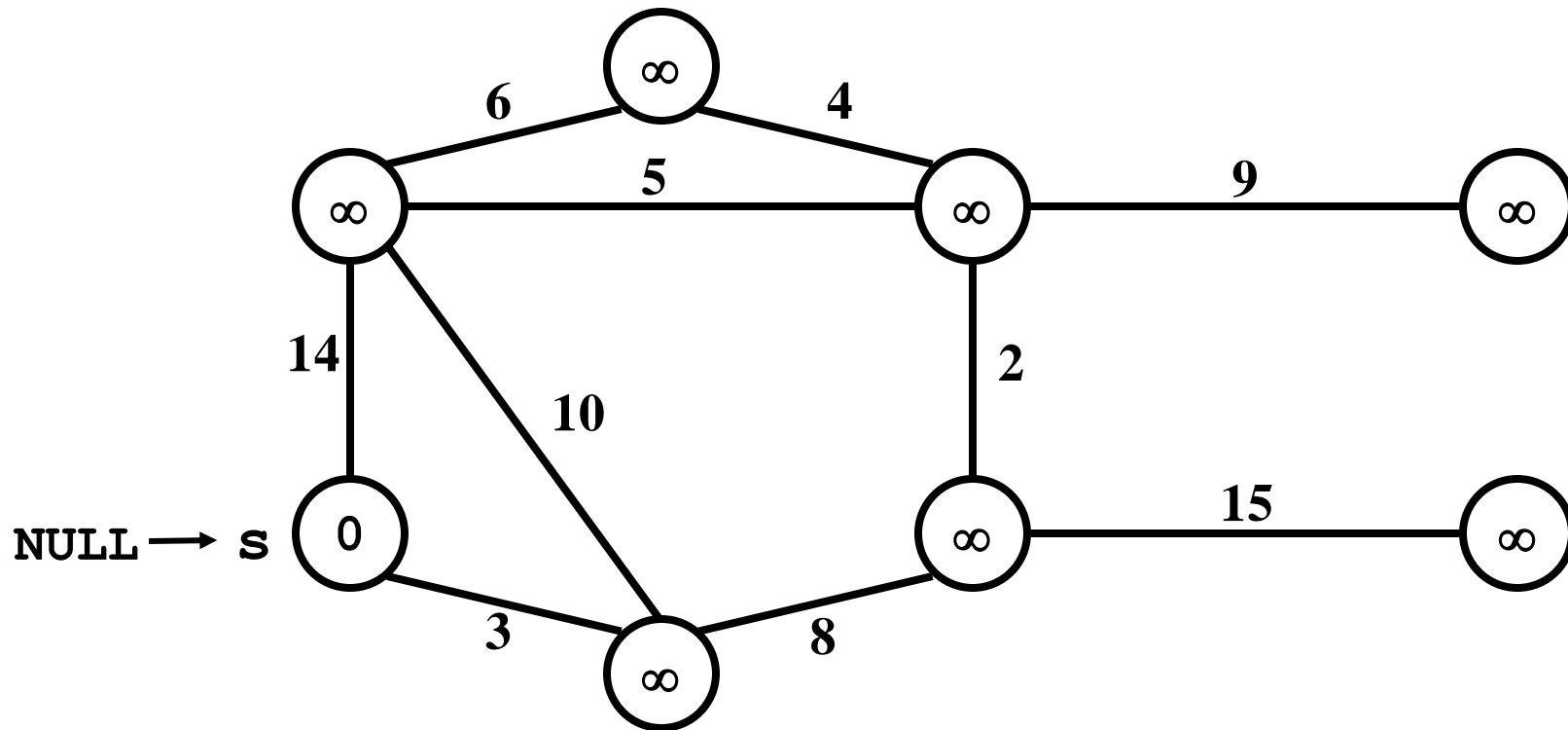
```

Q je množina „zatiaľ nevybavených“ vrcholov

Upravíme hodnoty a predchodcov pre všetky hrany vychádzajúce z v



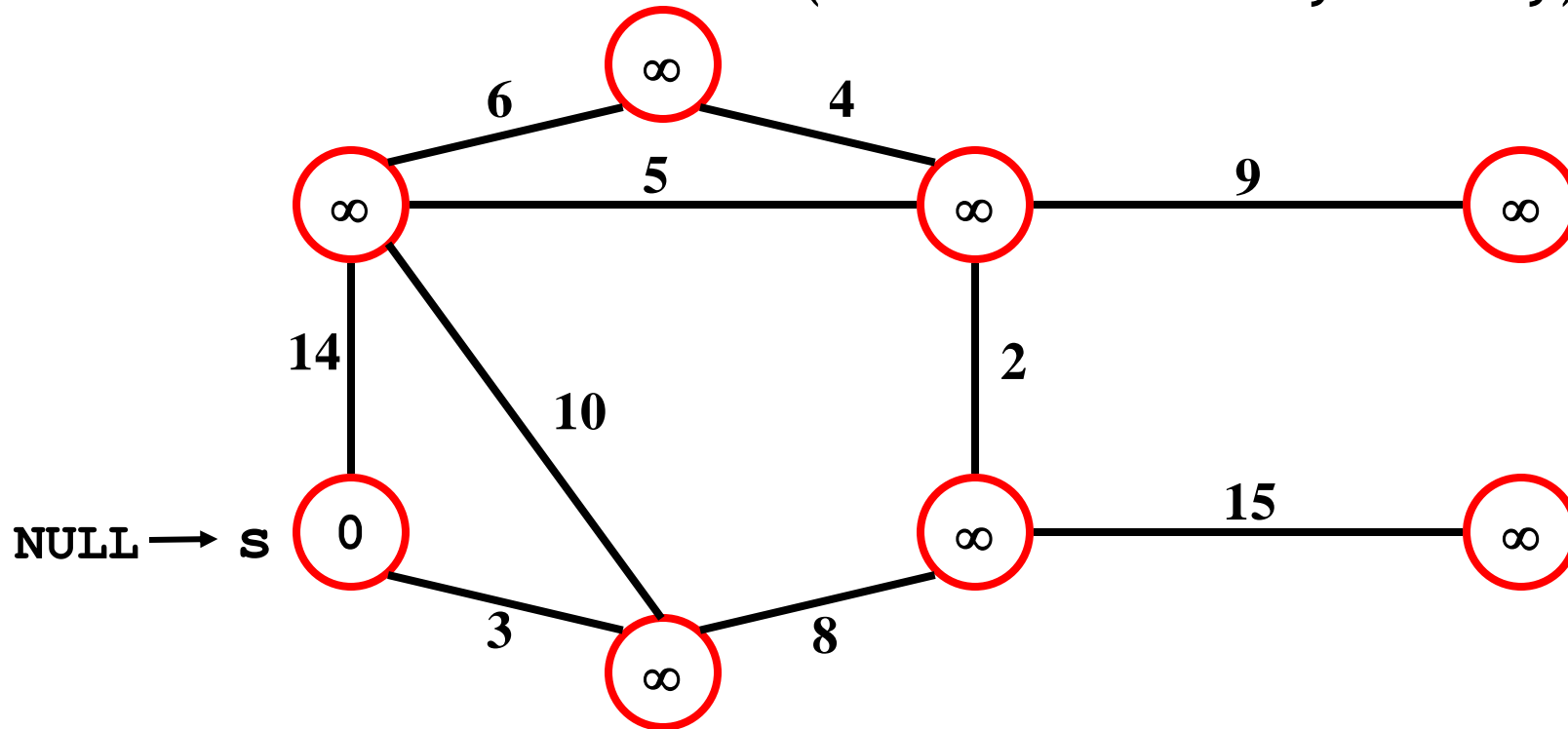
Primov algoritmus - simulácia





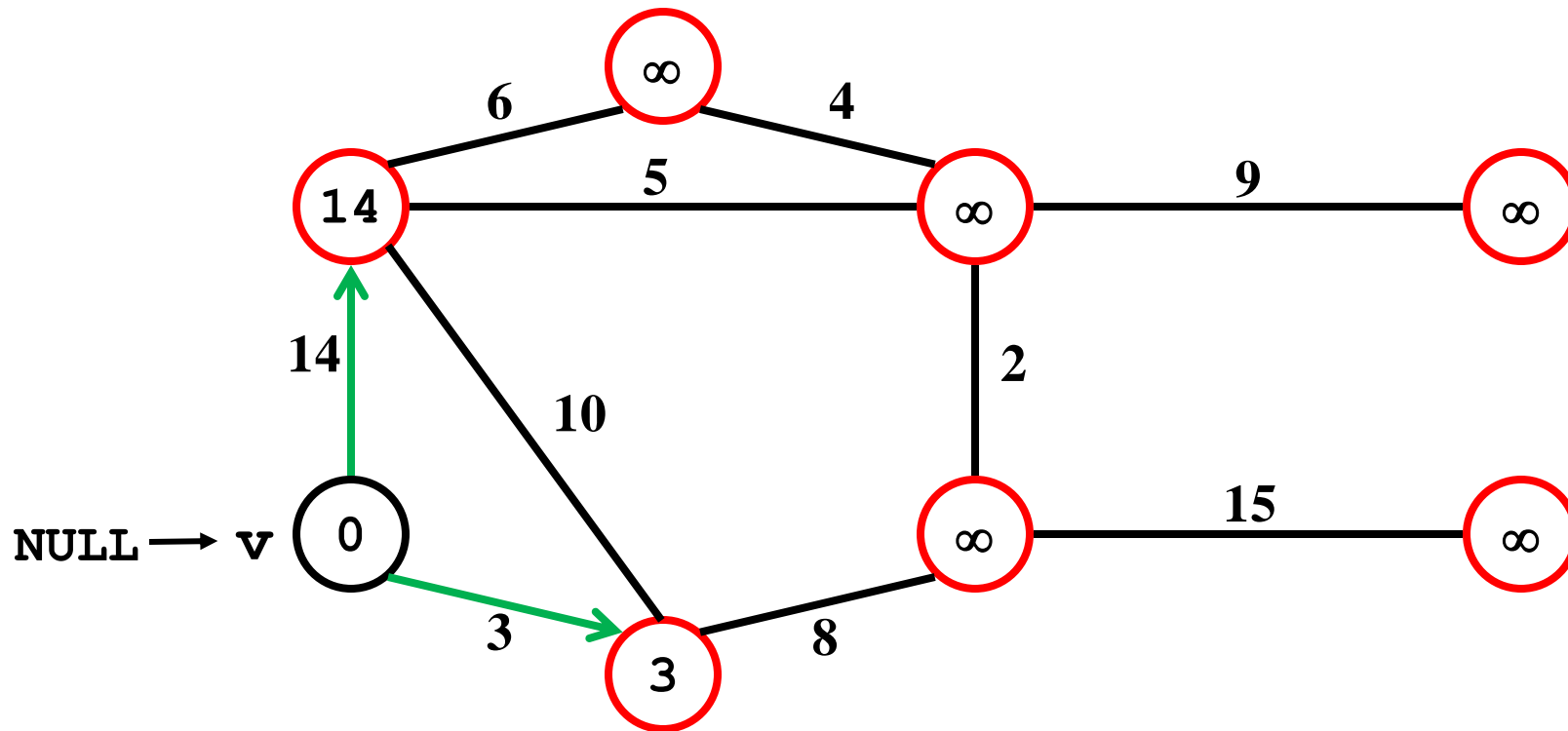
Primov algoritmus - simulácia

Q - množina „zatiaľ nevybavených“ vrcholov
(na začiatku všetky vrcholy)



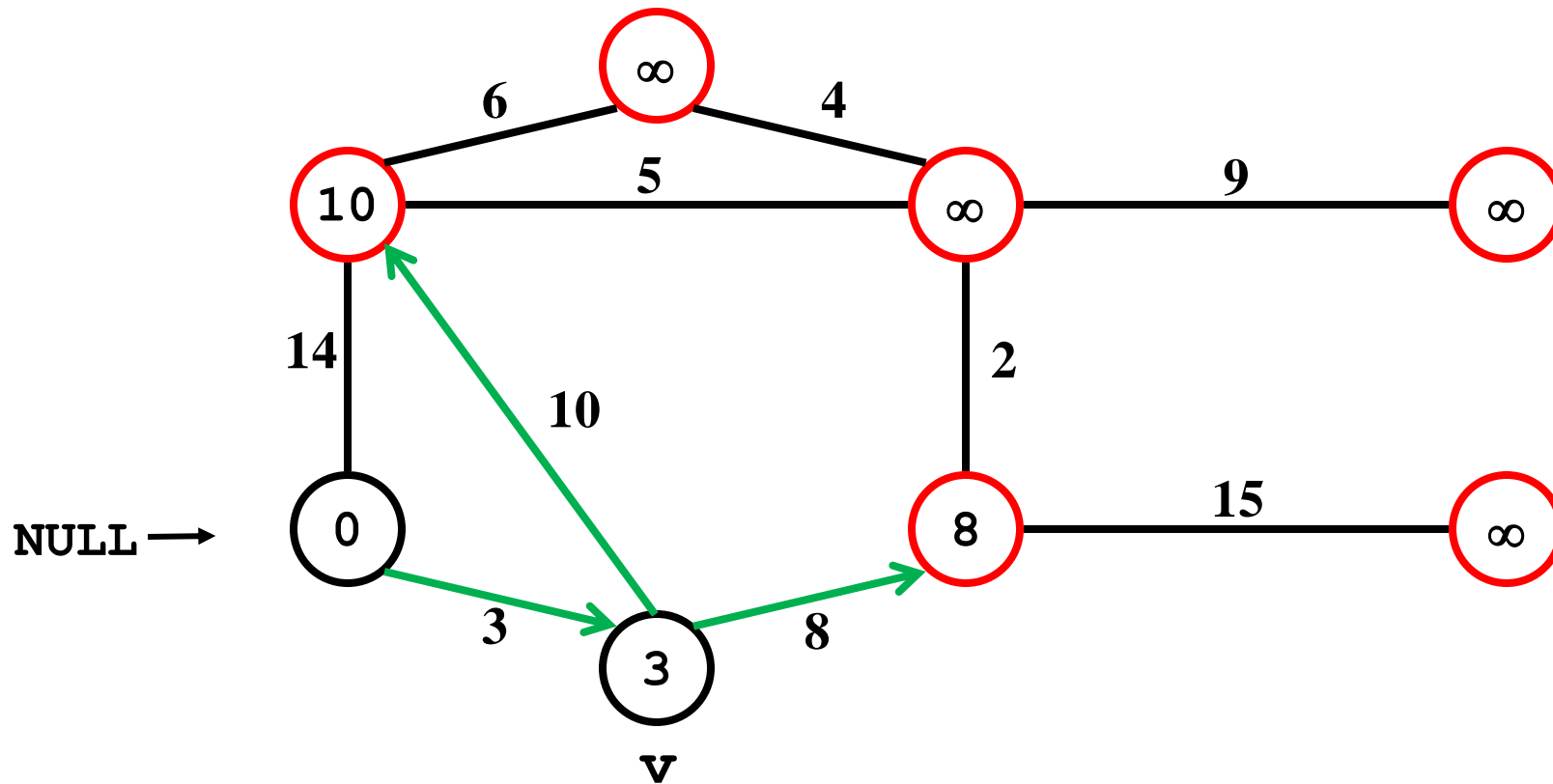


Primov algoritmus - simulácia



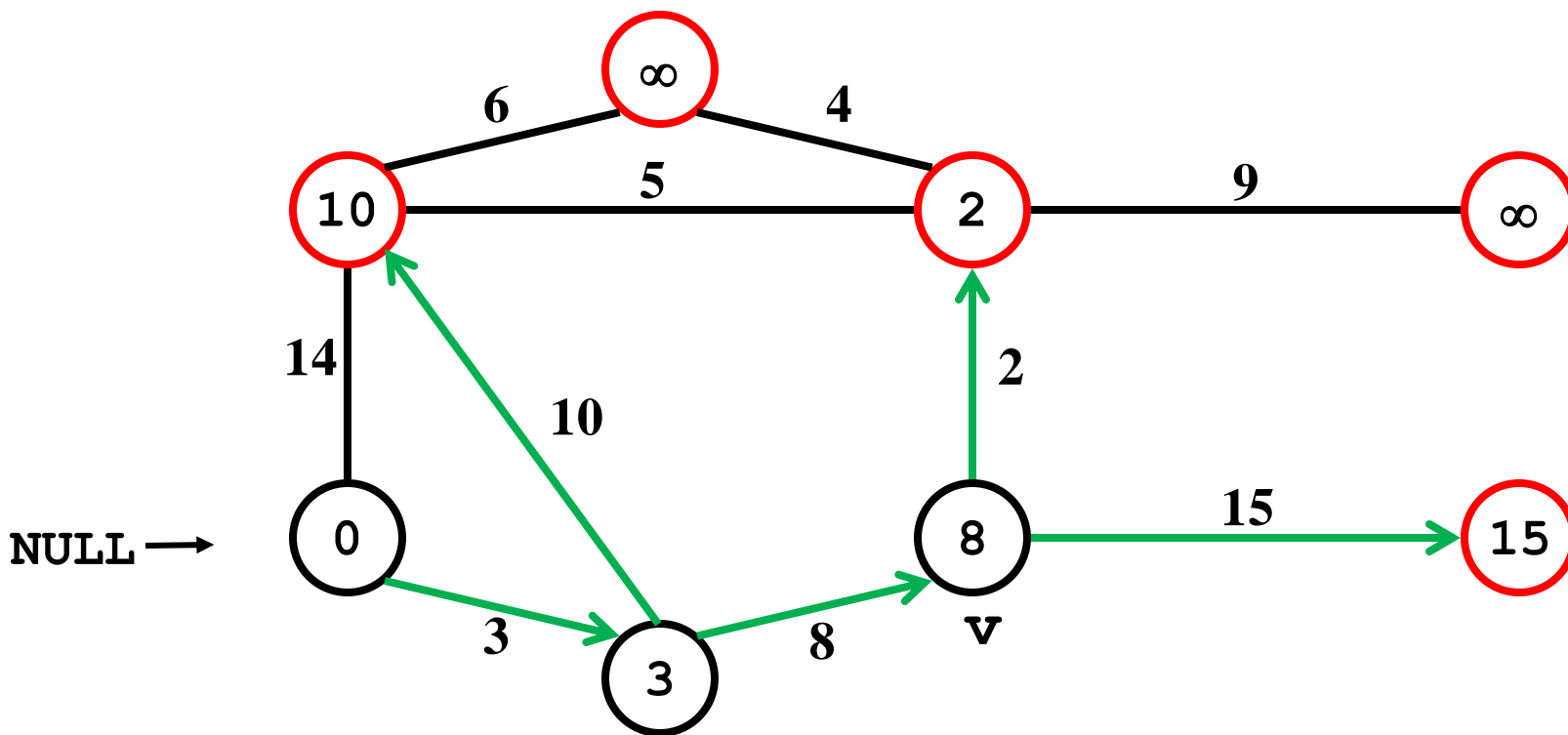


Primov algoritmus - simulácia



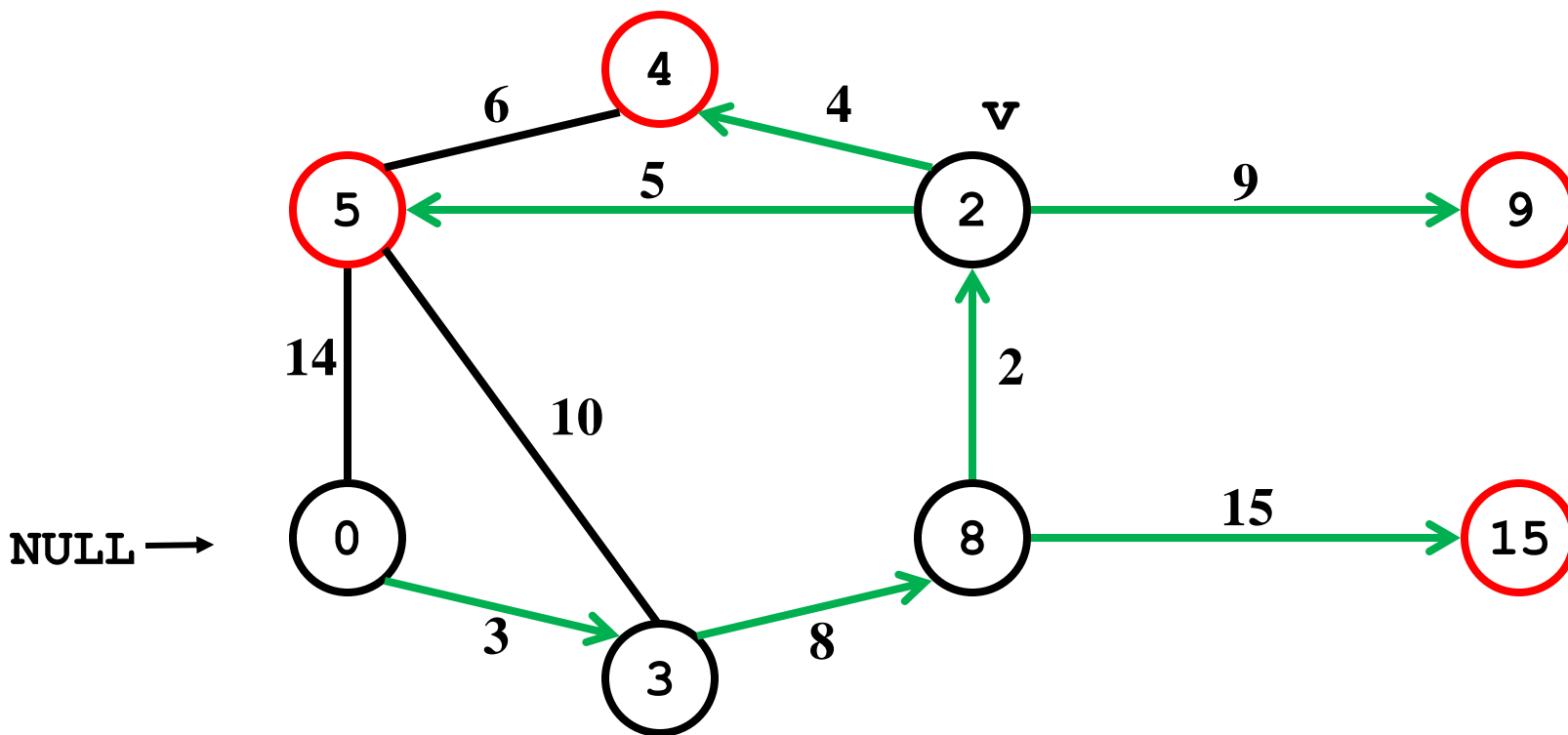


Primov algoritmus - simulácia



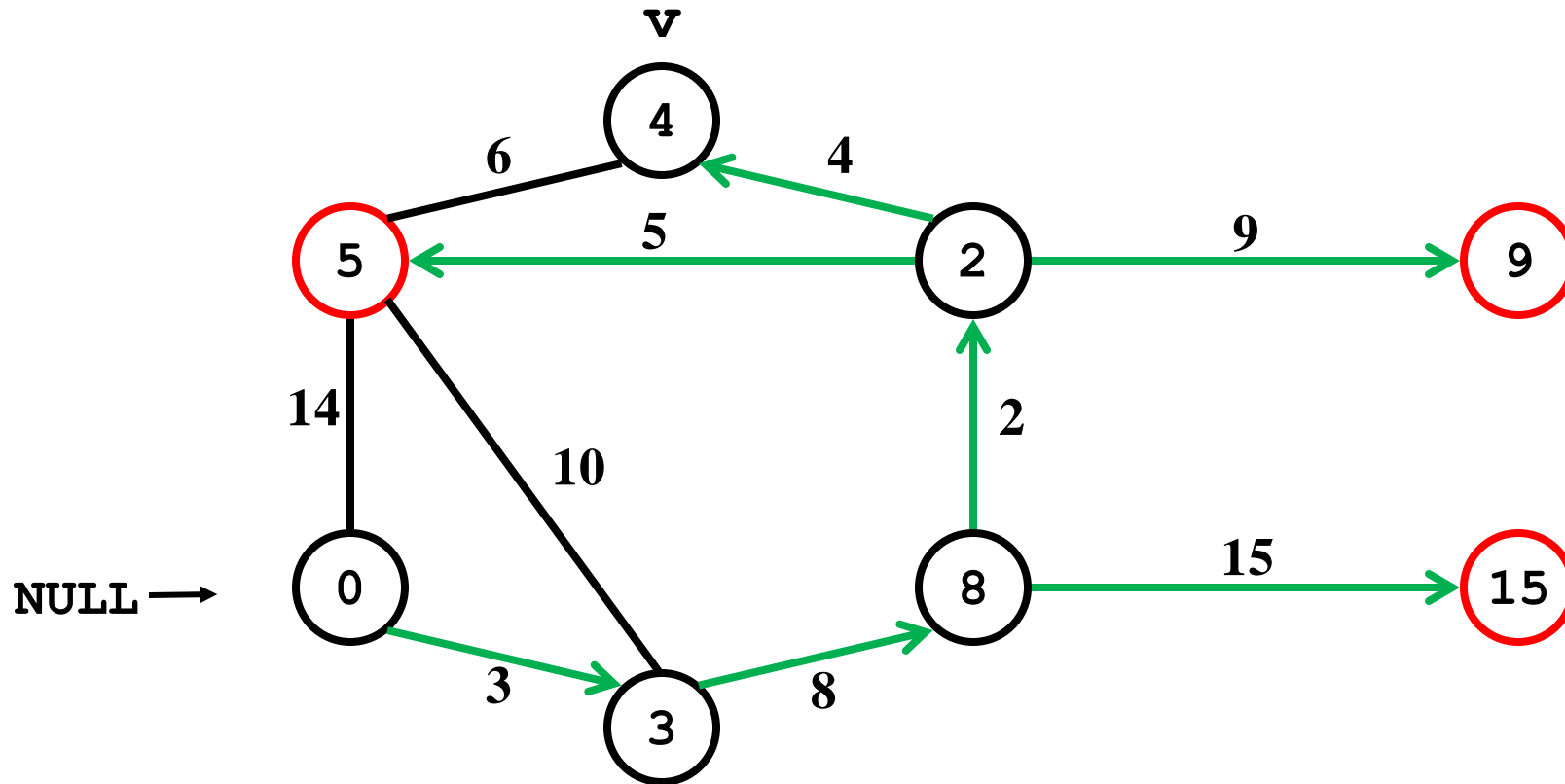


Primov algoritmus - simulácia



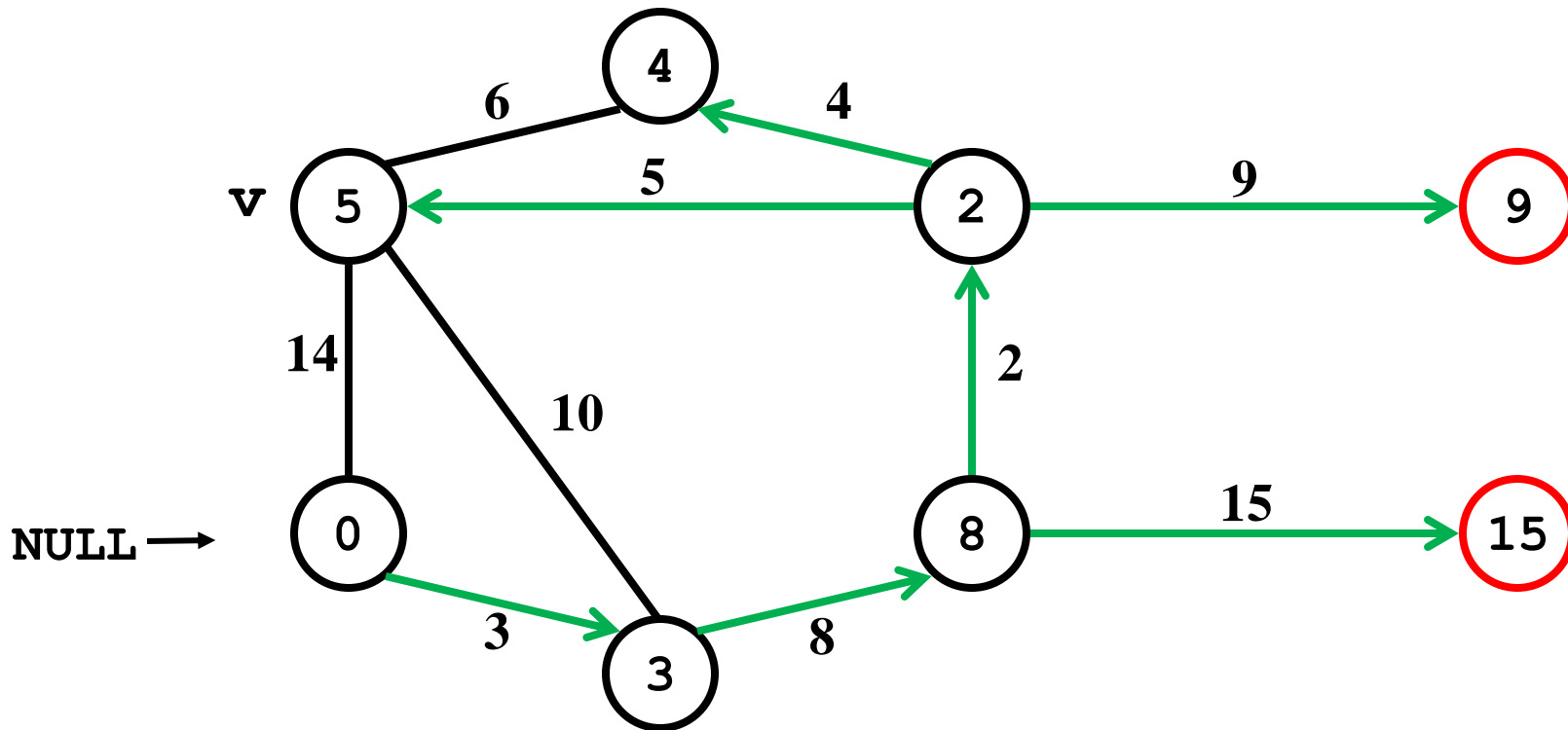


Primov algoritmus - simulácia



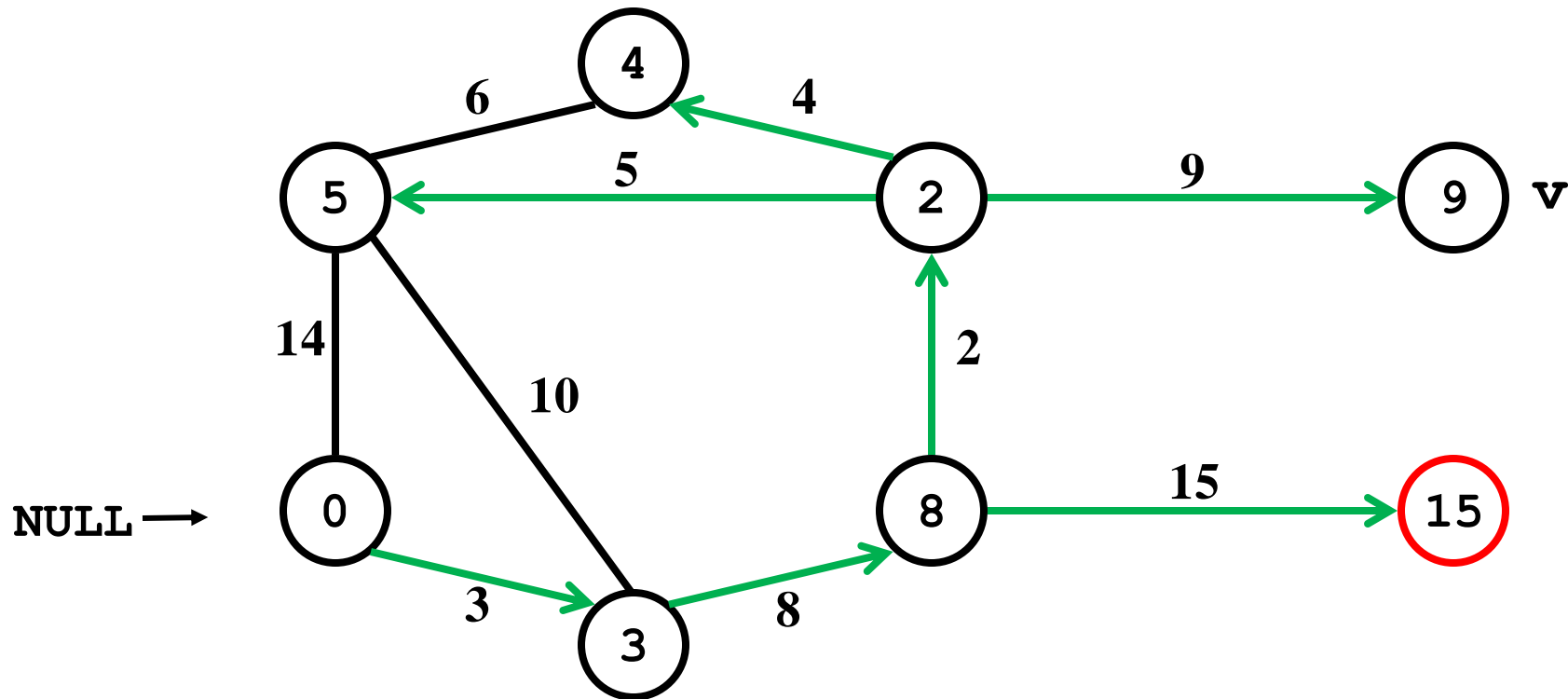


Primov algoritmus - simulácia



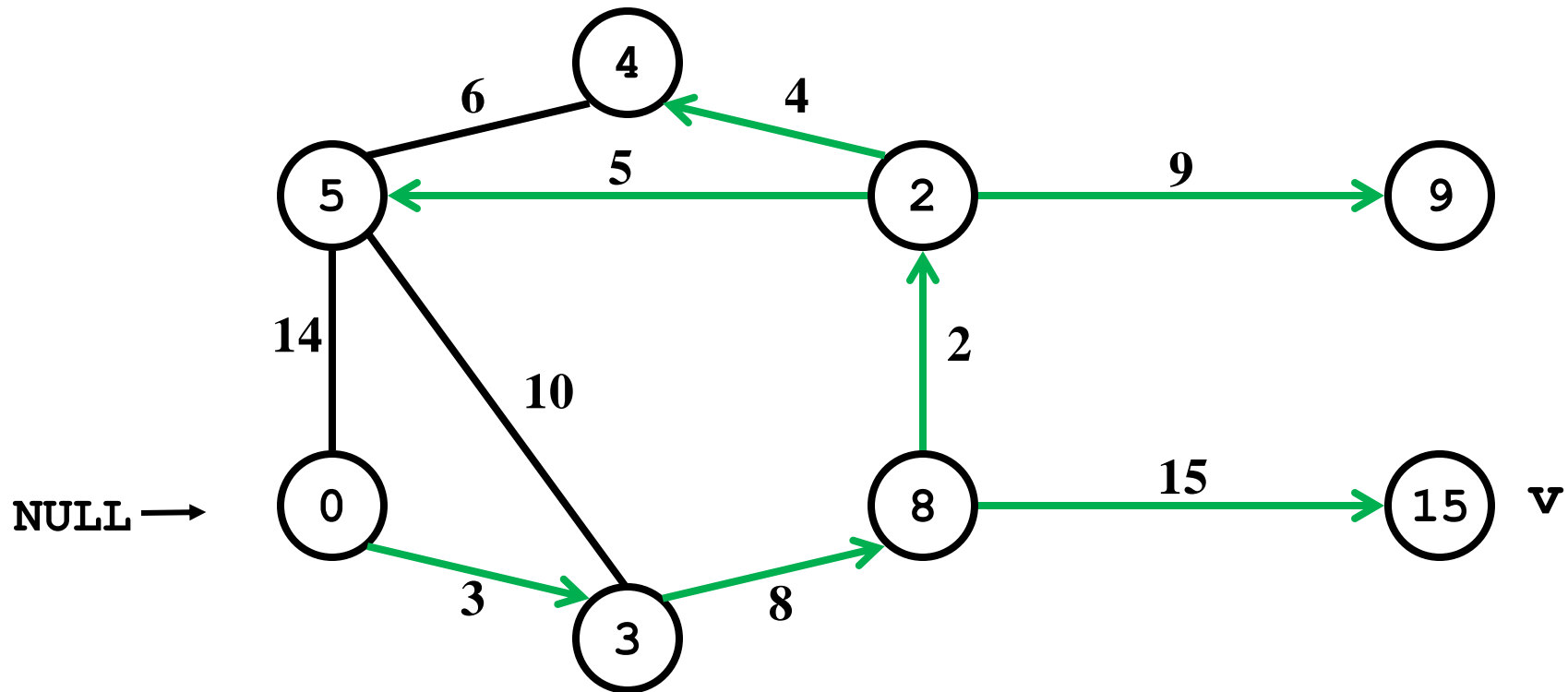


Primov algoritmus - simulácia





Primov algoritmus - simulácia

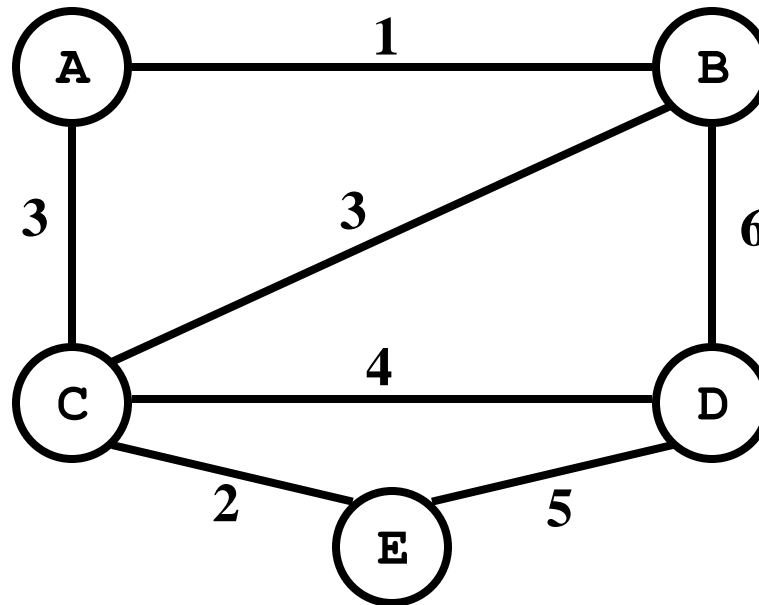




Úloha 1

- Otázka:

Ak začneme vo vrchole D, ktorý vrchol bude odstránený z Q ako prvý a ktorý ako druhý?

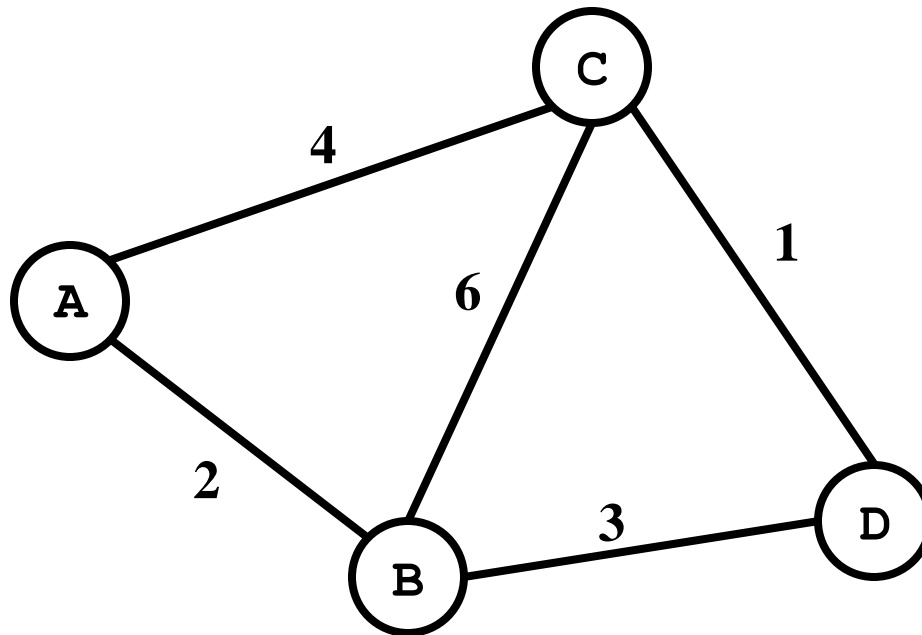




Úloha 2

- Otázka:

Ak začneme vo vrchole B, ktorý vrchol bude odstránený z Q ako prvý a ktorý ako druhý?

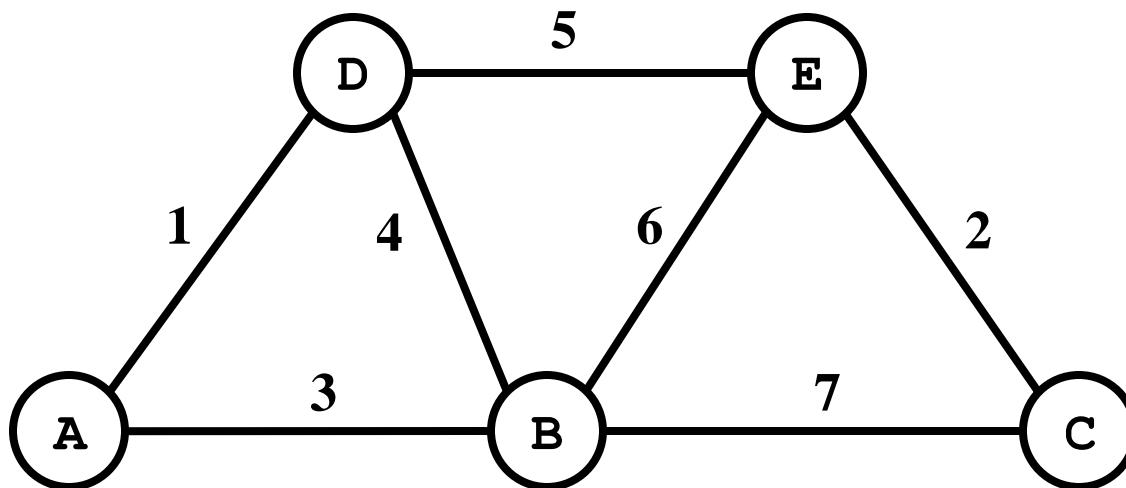




Úloha 3

- Otázka:

Ak začneme vo vrchole B, ktorý vrchol bude odstránený z Q ako prvý a ktorý ako druhý?





Primov algoritmus - poznámky

- **Správnosť (Korektnosť):** Dokázať, že algoritmus skutočne počíta minimálnu kostru grafu. Teda „greedy“ algoritmus vypočíta optimálne riešenie.
- **Efektívnosť:** Ak použijeme prioritný rad na nájdenie okrajovej hrany s najnižším ohodnotením: *min-heap (pre hrany)*, tak pre graf s n vrcholmi a m hranami máme:

$$\underbrace{(n - 1 + m)}_{\text{počet krokov (min-heap vymazania)}} \log n \leftarrow \begin{array}{l} \text{vlozenie/vymazanie z min-heap} \\ \text{počet uvažovaných hrán (min-heap vlozenia)} \end{array}$$

$$\Theta(m \log n)$$

Zložitost' závisí od použitých dátových štruktúr



Primov algoritmus – dôkaz správnosti

- Nech G je súvislý **ohodnotený graf**.
- V každej iterácii Primovho algoritmu **je pridaná hrana**, ktorá má jeden vrchol v podgrafe vytvárajúcom kostru a vrchol mimo tohto podgrafu.
- Pretože G je súvislý, existuje vždy cesta z každého do každého vrcholu.
- Výstup Y Primovho algoritmu je **strom**, pretože vrchol a hrana, ktoré sú pridané do Y sú kontrolované na cyklus v grafe.



Primov algoritmus – dôkaz správnosti

Nech Y_1 je minimálna kostra G .

- Ak $Y_1 = Y$, tak Y (z algoritmu) je minimálna kostra.
- Inak ($Y_1 \neq Y$, teda Y (z algoritmu) nie je minimálna kostra):
 - nech e je prvá hrana pridaná počas konštrukcie Y , ktorá nie je v Y_1 ,
 - a V nech je množina vrcholov spojených hranami pridanými do Y pred pridaním e .
- Potom jeden koncový bod e je vo V a druhý nie je.
- Keďže Y_1 je kostra G , existuje cesta v Y_1 spájajúca tieto dva koncové vrcholy hrany e .
Keď prechádzame pozdĺž tejto cesty, musíme naraziť na hranu f spájajúcu vrchol vo V s vrcholom, ktorý nie je vo V .



Primov algoritmus – dôkaz správnosti

- Teraz, v iterácii, keď je pridávaná hrana e do Y ,
 - f by mohla byť pridaná tiež a mohla by byť pridaná namiesto e
ak by jej ohodnotenie bolo menšie ako e .
- Lenže f **nebola** pridaná,
 - teda ohodnotenie f **nie je menšie** ako ohodnotenie e .



Primov algoritmus – dôkaz korektnosti

- Nech Y_2 je graf získaný z Y_1 odstránením f a pridaním e .
- Je ľahké ukázať, že:
 - Y_2 je súvislý
 - Y_2 má ten istý počet hrán ako Y_1
 - celková váha hrán v Y_2 nie je väčšia ako v Y_1
 preto Y_2 je tiež minimálna kostra G ,
 obsahuje e a všetky hrany pridané pred e počas konštrukcie V
- Opakovaním vyššie uvedených krokov vieme získať minimálnu kostru grafu G , ktorá je identická s Y .
- Toto dokazuje, že Y je **minimálna kostra**.



Kruskalov algoritmus

- les – graf bez kružníc, ktorého komponenty sú stromy
- počet vrcholov grafu uvažujme $n > 0$

- Začiatok Kruskalovho algoritmu:
 - hrany sú utriedené podľa rastúceho ohodnotenia,
 - máme les n disjunktných stromov.

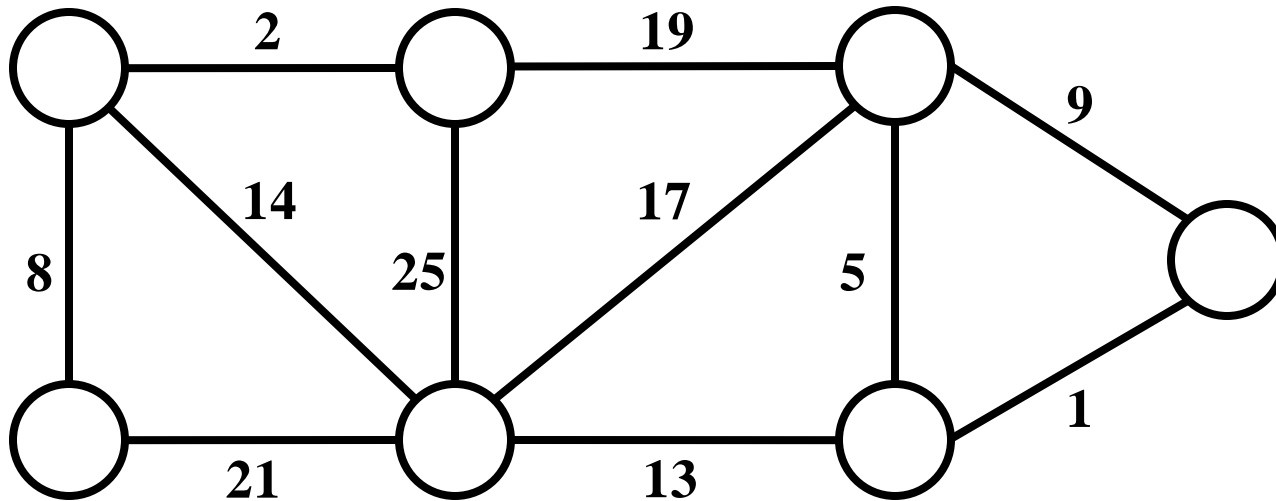


Kruskalov algoritmus

- V každom kroku algoritmu:
 - „porastie” minimálna kostra grafu (minimal spanning tree - MST) o jednu hranu.
 - pridáme hranu s minimálnym ohodnotením do množiny už použitých hrán, ak nevznikne cyklus.
- V priebehu algoritmu jednotlivé kroky vytvárajú obvykle les stromov (nesúvislý graf).



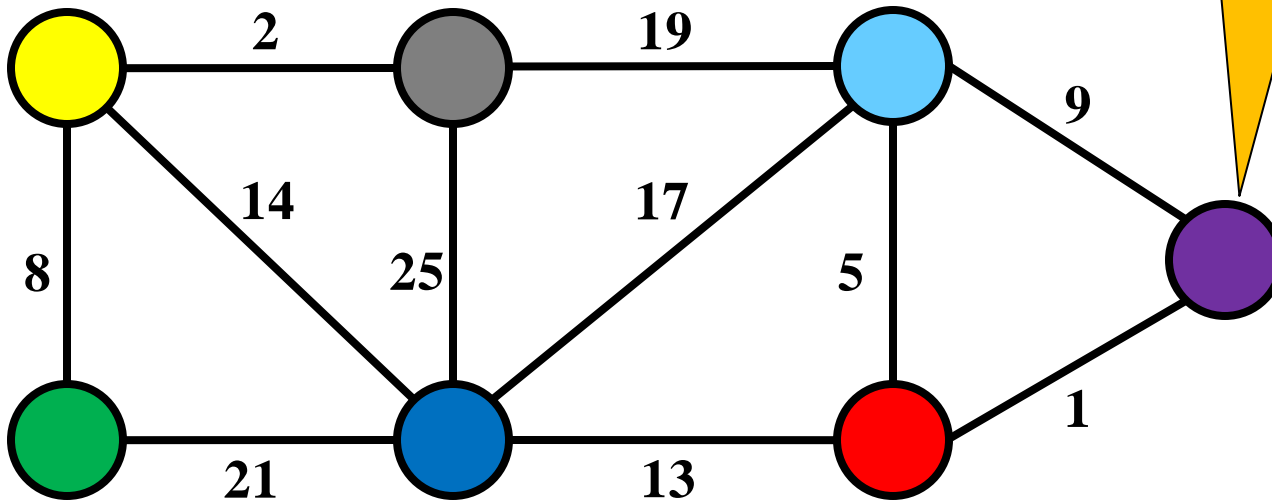
Kruskalov algoritmus - simulácia





Kruskalov algoritmus - simulácia

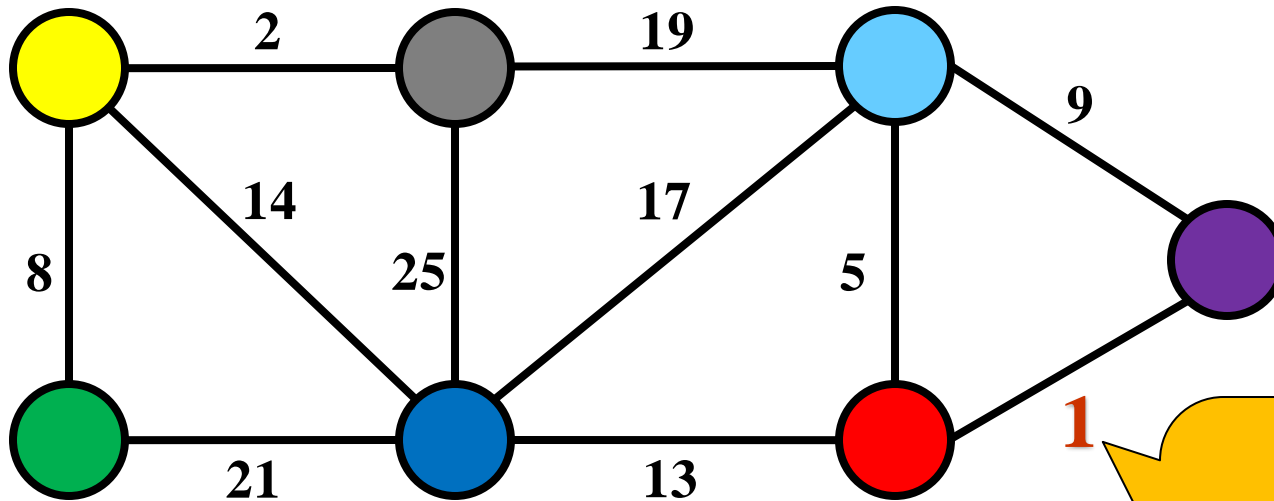
Vrcholy z rovnakého stromu majú spoločnú farbu.
Na začiatku je každý vrchol sám.



Utriedené hrany: 1, 2, 5, 8, 9, 13, 14, 17, 19, 21, 25



Kruskalov algoritmus - simulácia

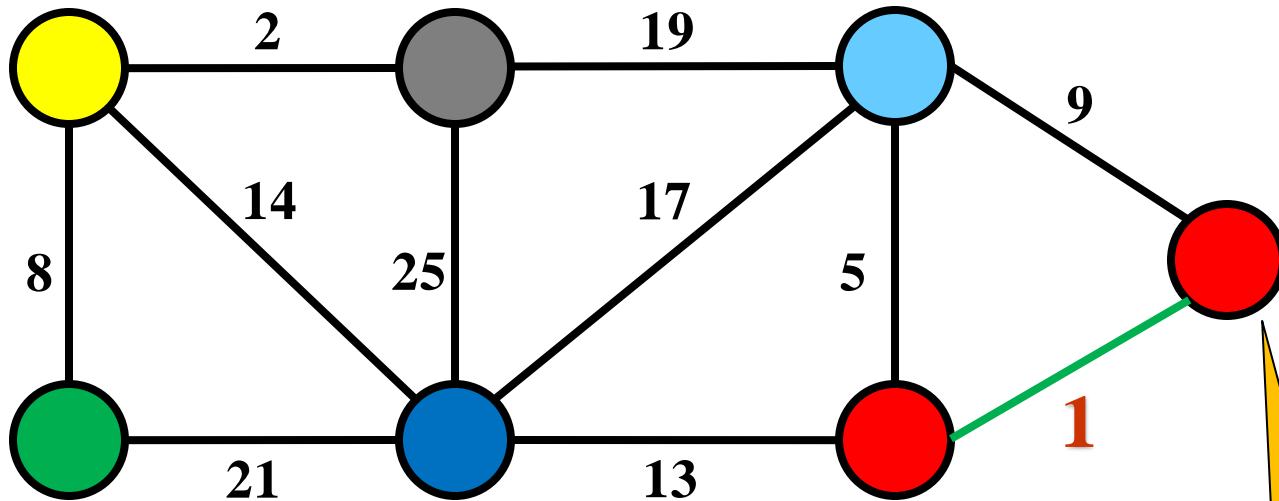


hrany sú
utriedené podľa
ceny a vždy
vyberáme
najlacnejšiu

Utriedené hrany: **1**, 2, 5, 8, 9, 13, 14, 17, 19, 21, 25



Kruskalov algoritmus - simulácia



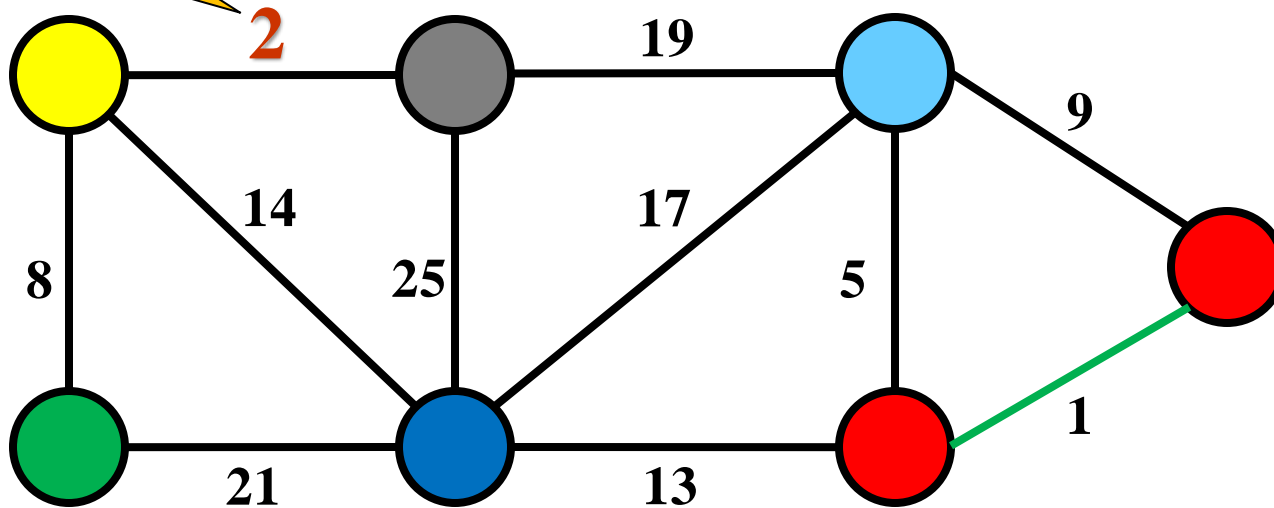
Utriedené hrany: **1**, 2, 5, 8

Spojíme komponenty = vrcholy dostanú rovnakú farbu, lebo sú v spoločnom strome



Kruskalov algoritmus - simulácia

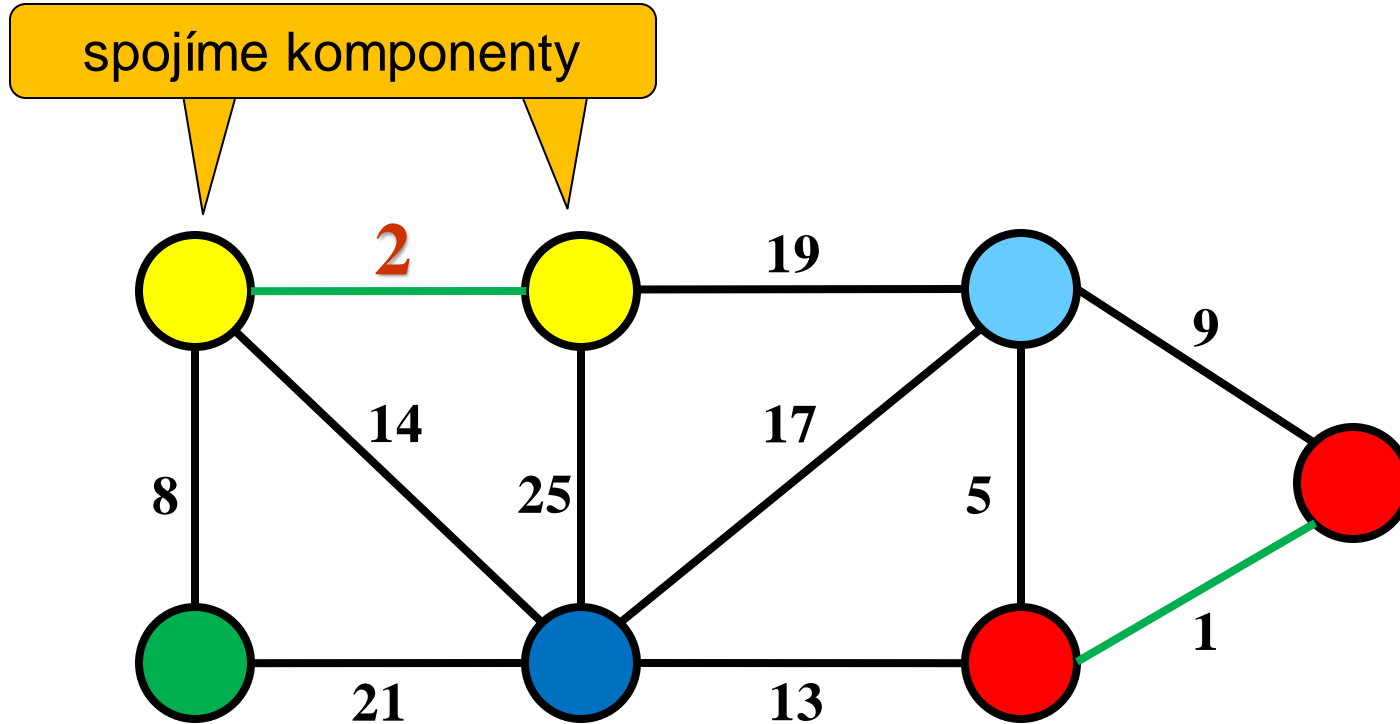
vyberáme ďalšiu najlacnejšiu hranu



Utriedené hrany: 1, 2, 5, 8, 9, 13, 14, 17, 19, 21, 25



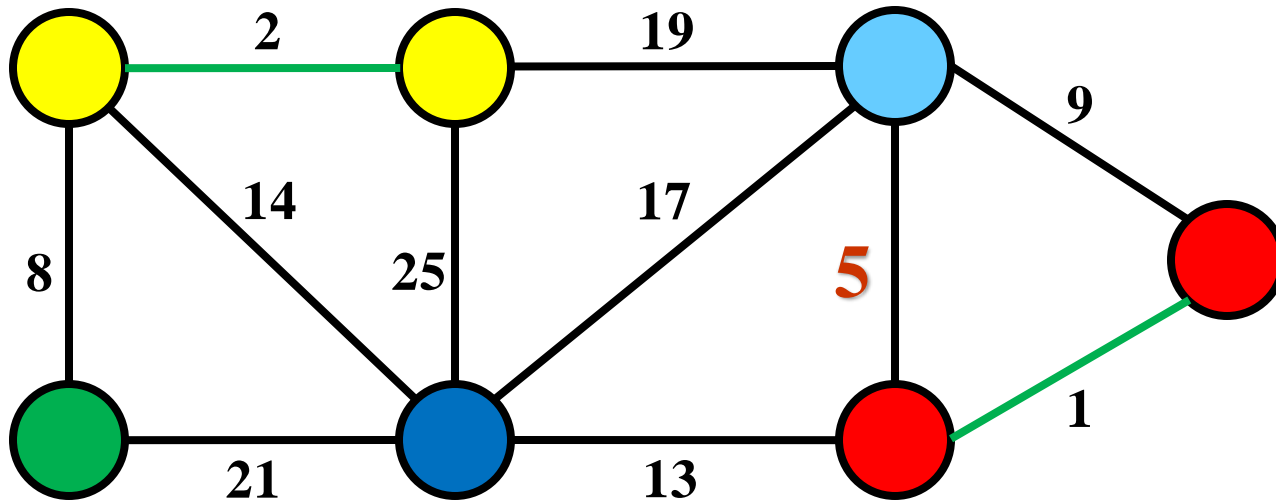
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, 8, 9, 13, 14, 17, 19, 21, 25



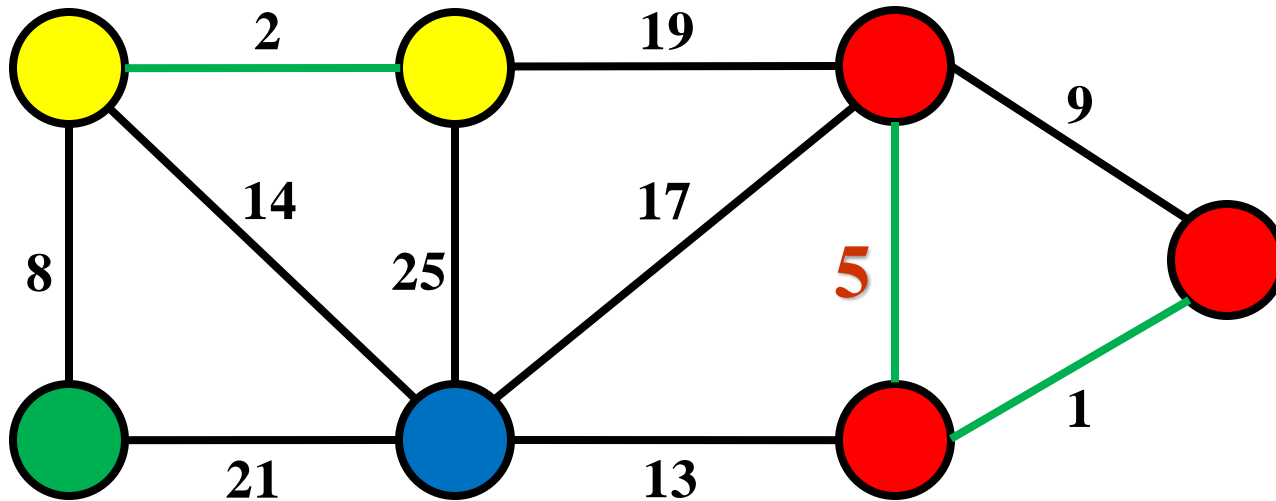
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, **5**, 8, 9, 13, 14, 17, 19, 21, 25



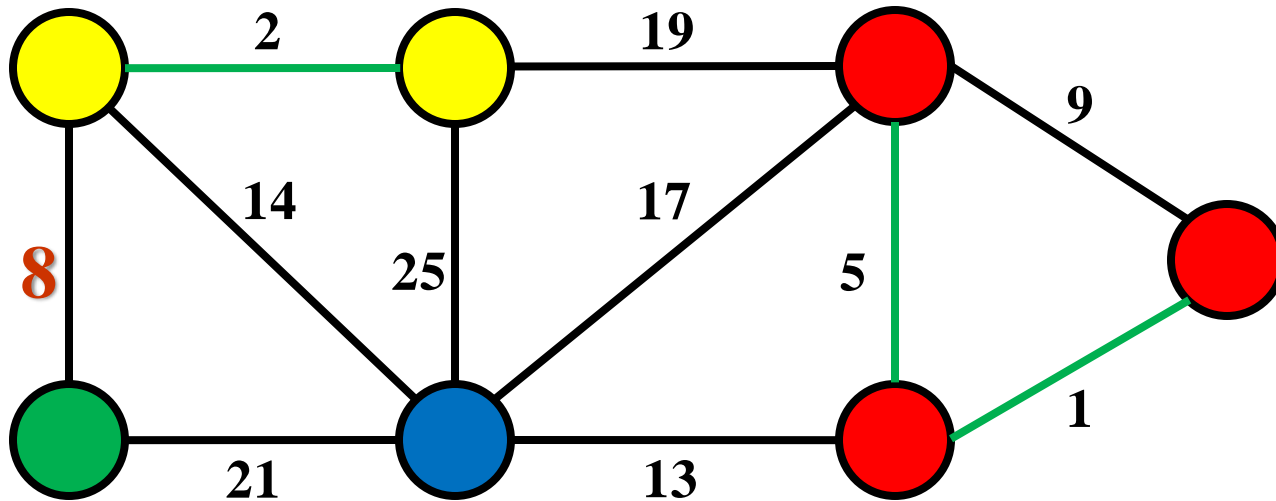
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, **5**, 8, 9, 13, 14, 17, 19, 21, 25



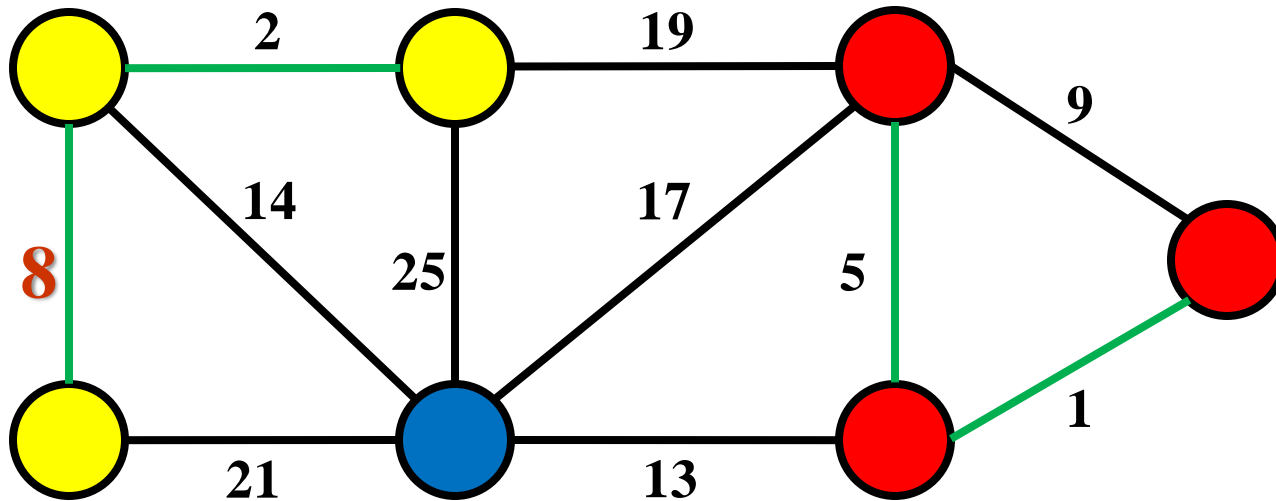
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, **8**, 9, 13, 14, 17, 19, 21, 25



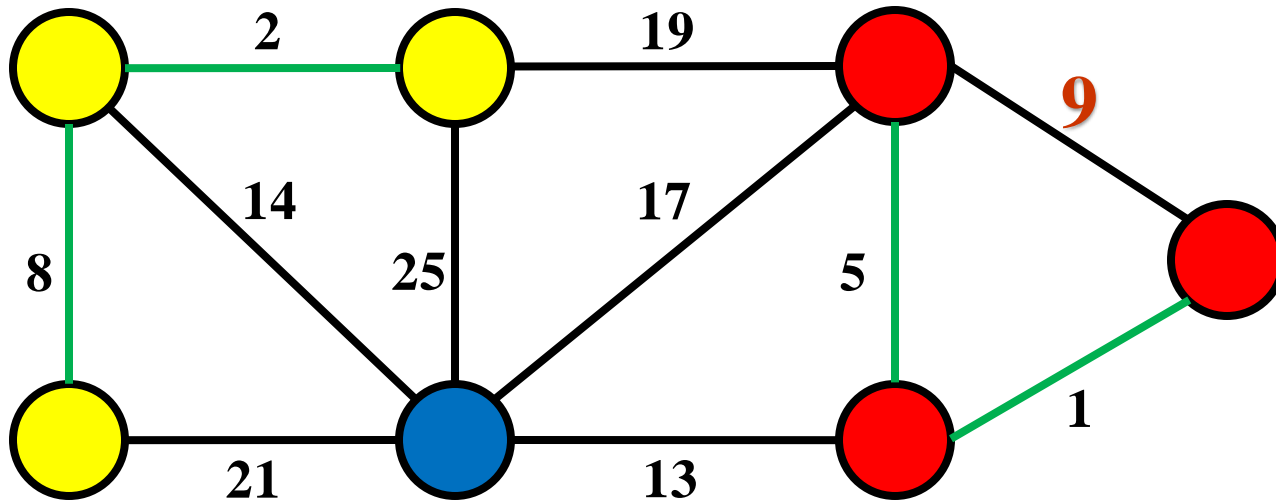
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, 8, 9, 13, 14, 17, 19, 21, 25



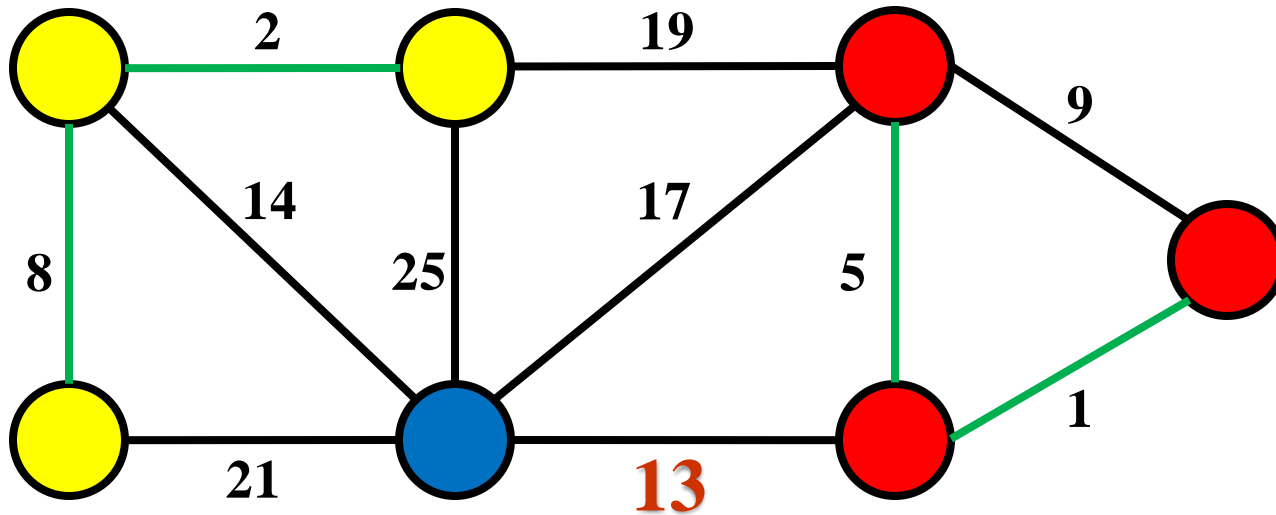
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, 8, **9**, 13, 14, 17, 19, 21, 25



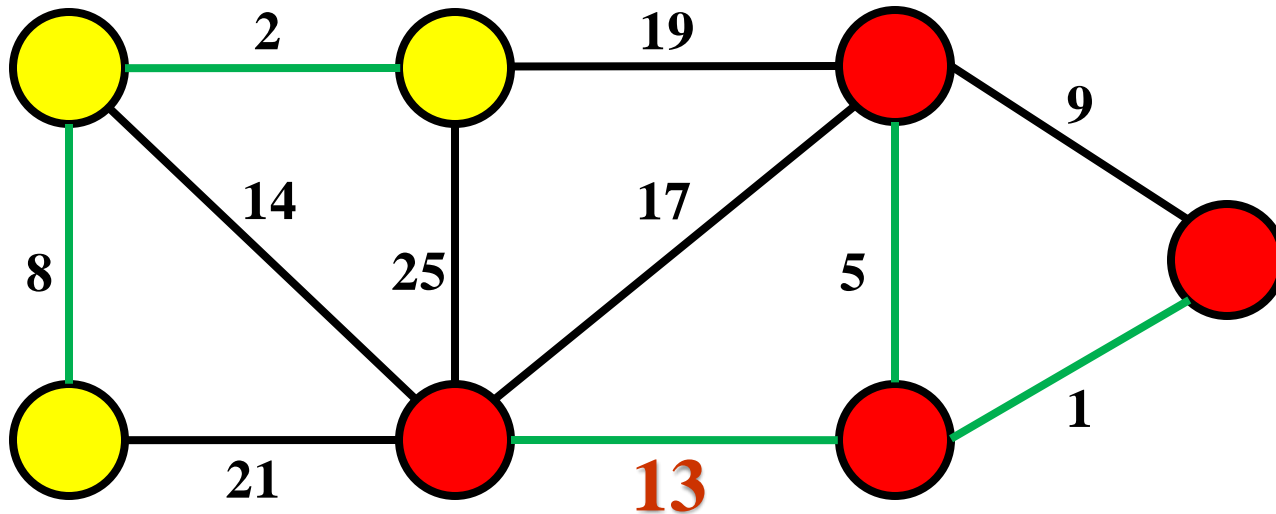
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, 8, 9, **13**, 14, 17, 19, 21, 25



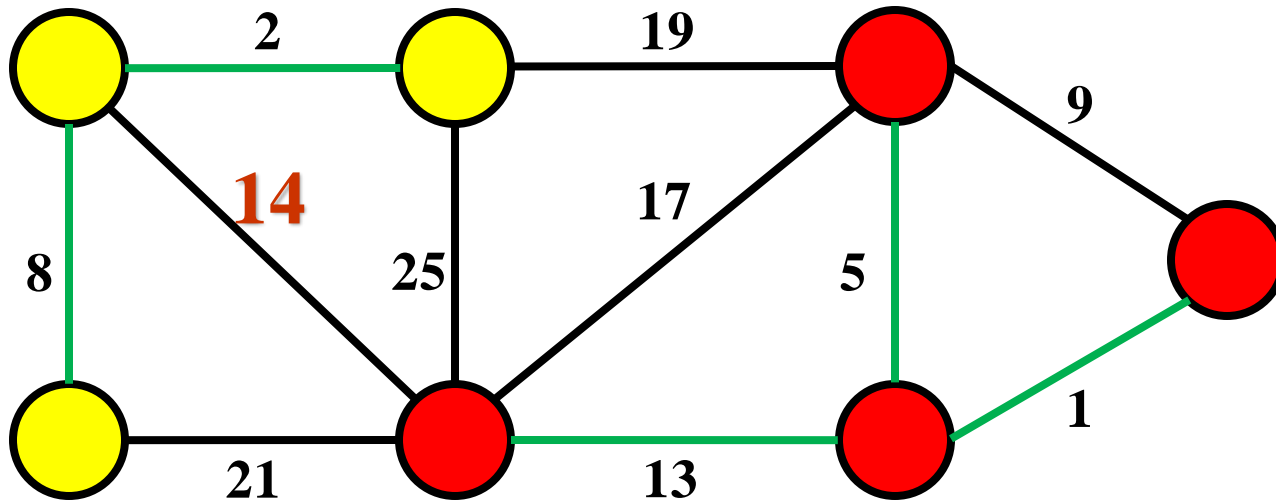
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, 8, 9, **13**, 14, 17, 19, 21, 25



Kruskalov algoritmus - simulácia

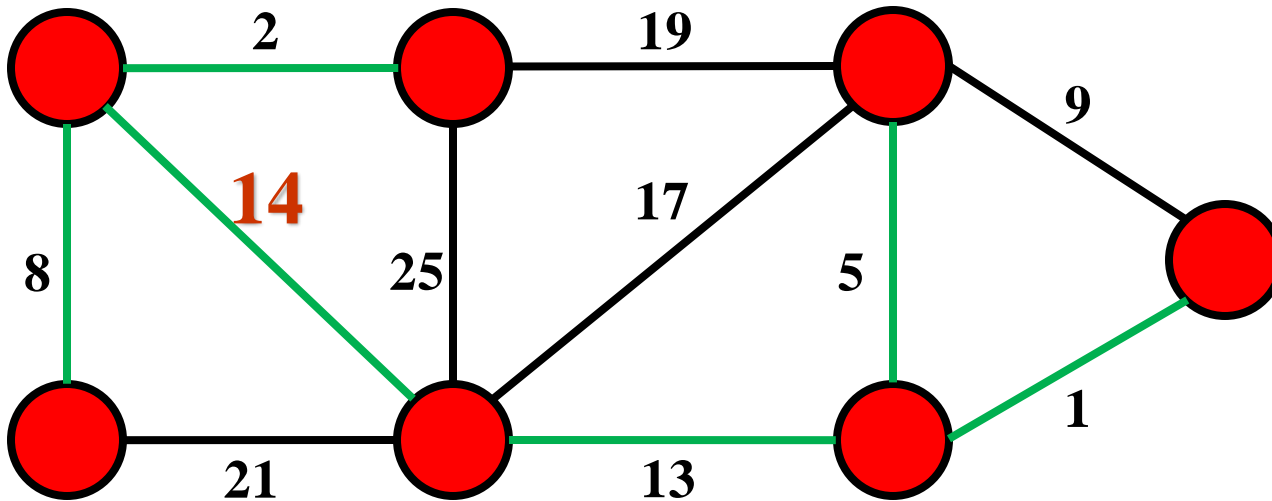


Utriedené hrany: 1, 2, 5, 8, 9, 13, **14**, 17, 19, 21, 25



Kruskalov algoritmus - simulácia

Všetky vrcholy už sú spolu v jednom komponente

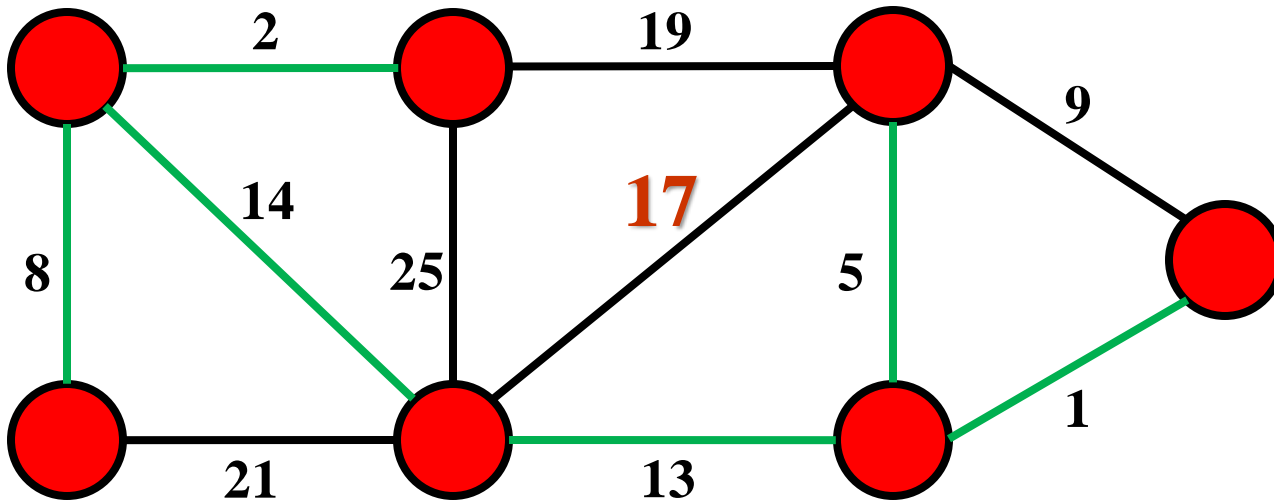


Utriedené hrany: 1, 2, 5, 8, 9, 13, **14**, 17, 19, 21, 25



Kruskalov algoritmus - simulácia

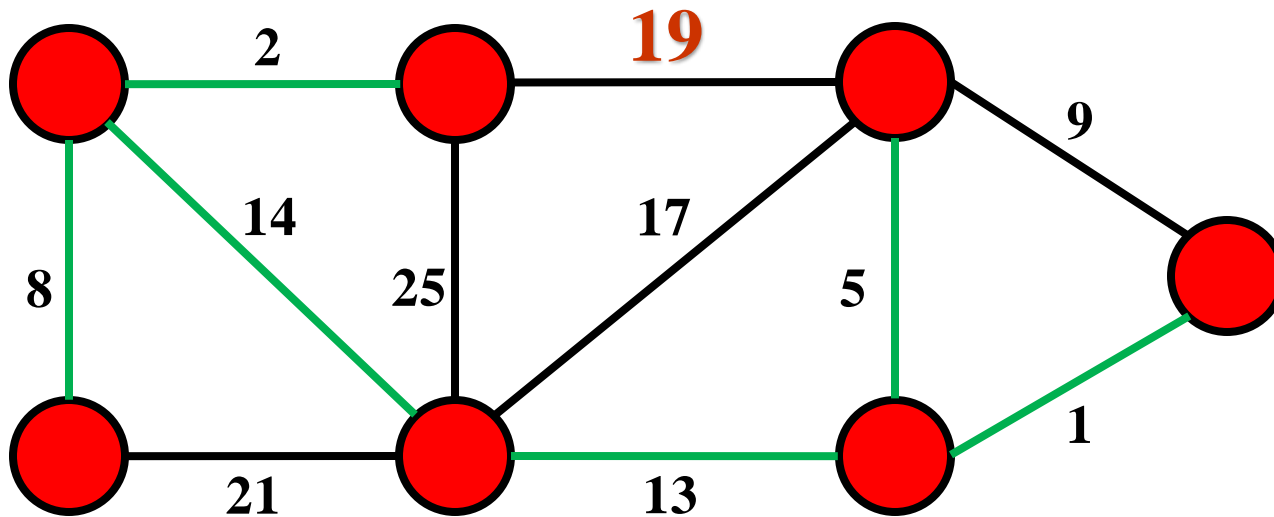
žiadnu ďalšiu hranu už
nevieme pridať



Utriedené hrany: 1, 2, 5, 8, 9, 13, 14, **17**, 19, 21, 25



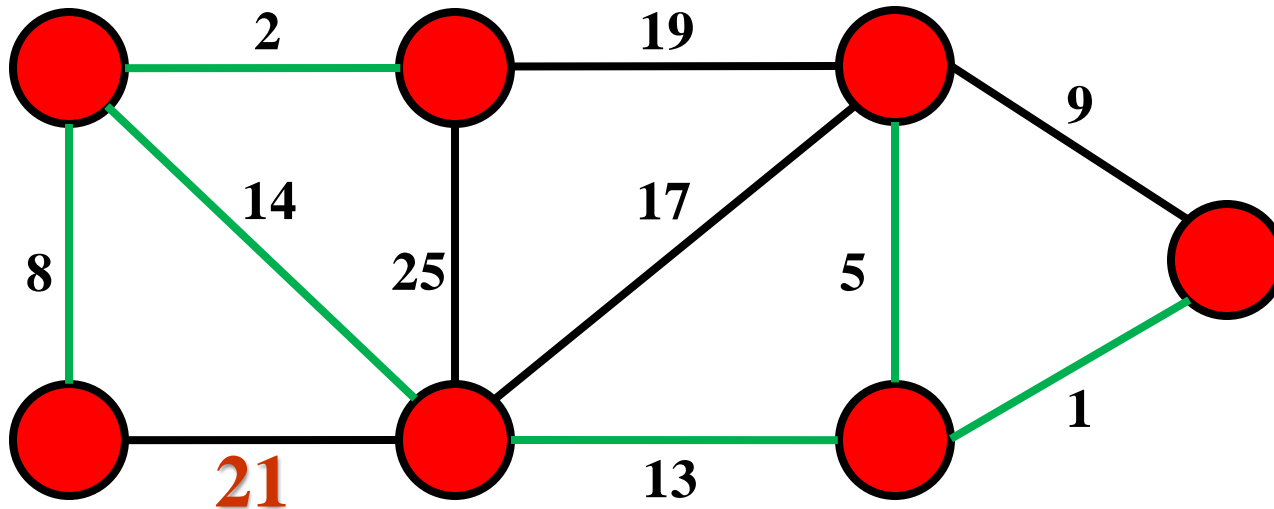
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, 8, 9, 13, 14, 17, **19**, 21, 25



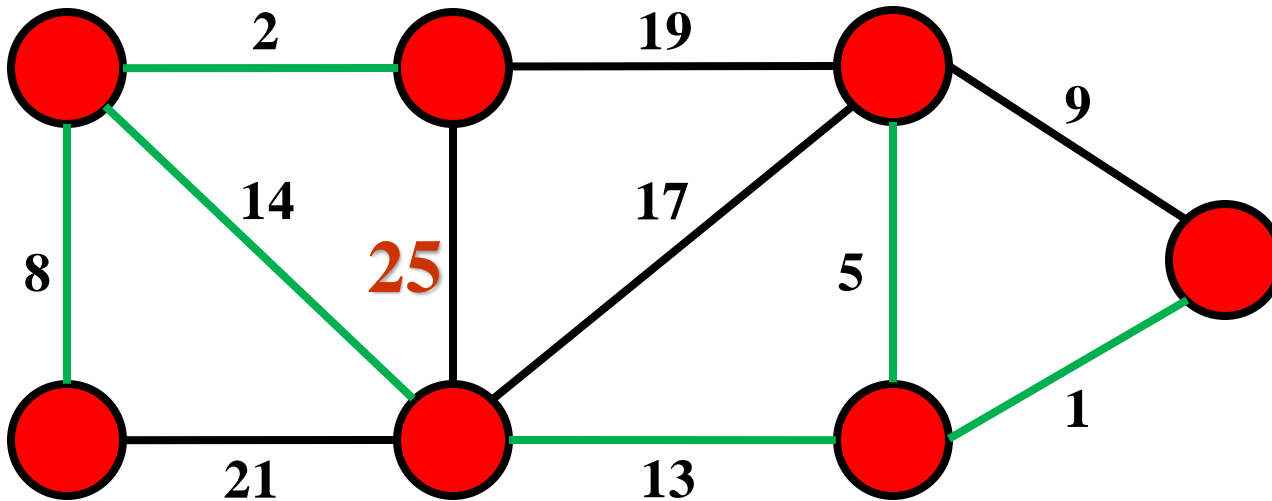
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, 8, 9, 13, 14, 17, 19, **21**, 25



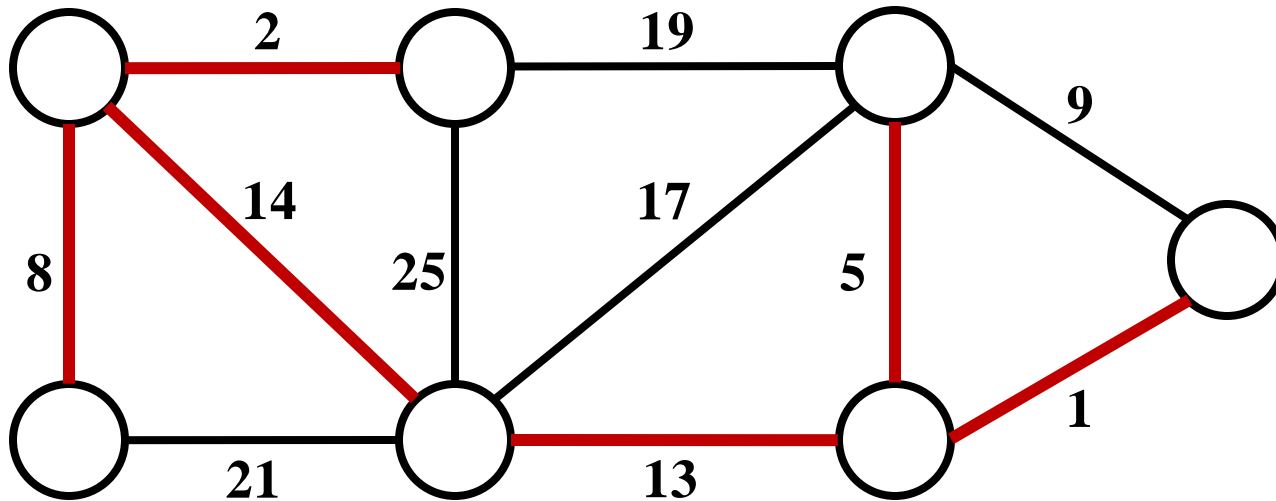
Kruskalov algoritmus - simulácia



Utriedené hrany: 1, 2, 5, 8, 9, 13, 14, 17, 19, 21, **25**



Kruskalov algoritmus - simulácia



získali sme minimálnu
kostru grafu



Kruskalov algoritmus

- V každom kroku môže hrana spoji dva existujúce stromy do jedného stromu
- Je potrebný efektívny spôsob pre určovanie /vyhnutie sa cyklom
- Algoritmus skončí, keď všetky vrcholy sú v jednom strome.

Ak jednovrcholové grafy nepovažujem za stromy

V každom kroku hrana môže:

- zväčšiť nejaký existujúci strom
- spojiť dva existujúce stromy do jedného stromu
- vytvoriť nový strom



Kruskalov algoritmus

kostra = prázdny zoznam hrán;

zotried' množinu hrán v grafe podľa hodnoty;

prirad' vrcholom čísla komponentov od 1 po n;

//kazdy vrchol je na zaciatku vo vlastnom
komponente

for (e: hrany grafu){

if (začiatok a koniec hrany e
sú v rôznych komponentoch){

kostra.pridajHranu(e);

spoj komponenty začiatku a konca hrany;

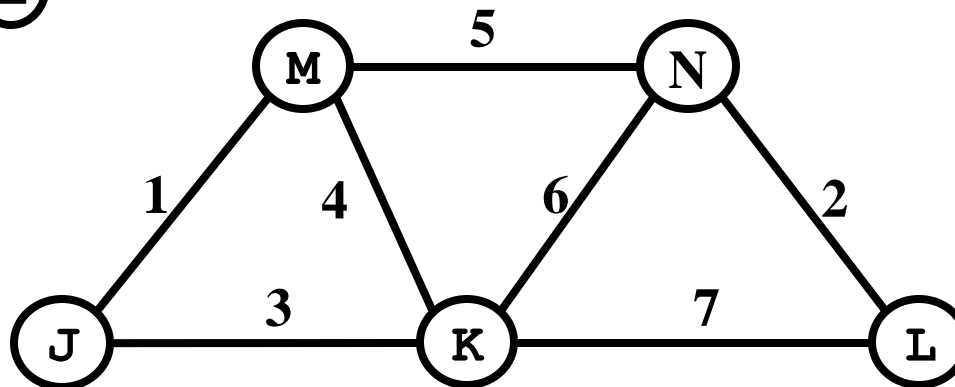
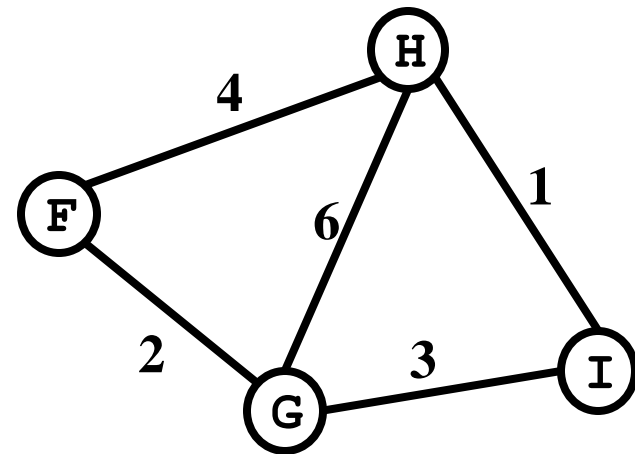
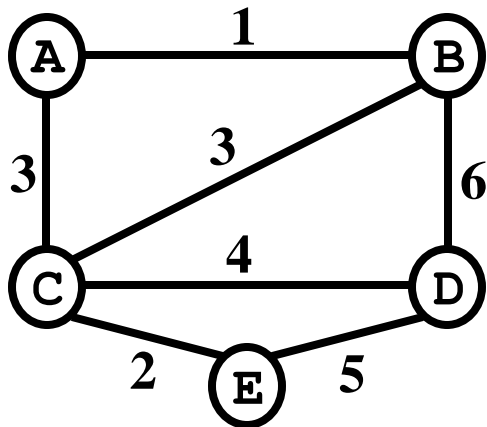
}



Úloha 4

● Otázka:

Ktorou hranou začneme Kruskalov algoritmus?





Kruskalov algoritmus - poznámky

- Algoritmus vyzerá jednoduchší ako Dijkstrov-Primov, ale je
 - Ťažšie implementovateľný (kontrola cyklov v grafe)
 - Menej efektívny $\Theta(m \log m)$ – utriedenie hrán (počet hrán môže byť až n^2)
 - Implementácia spojenia komponentov ovplyvní zložitost'
- **Kontrola cyklov:**
cyklus existuje, práve vtedy ak, hrana spája vrcholy v tom istom komponente.



Primov vs. Kruskalov algoritmus

- Časová zložitost' závisí od použitých dátových štruktúr v implementácii.
- V oboch algoritmoch sa kontroluje, či pridanie prvku nevytvorí cyklus, čo je najťažší krok.
- Ak má graf **veľký počet vrcholov**, tak Primov algoritmus je lepší
- Ak má graf **malý počet hrán**, tak časová zložitost' Kruskalovho algoritmu je lepšia



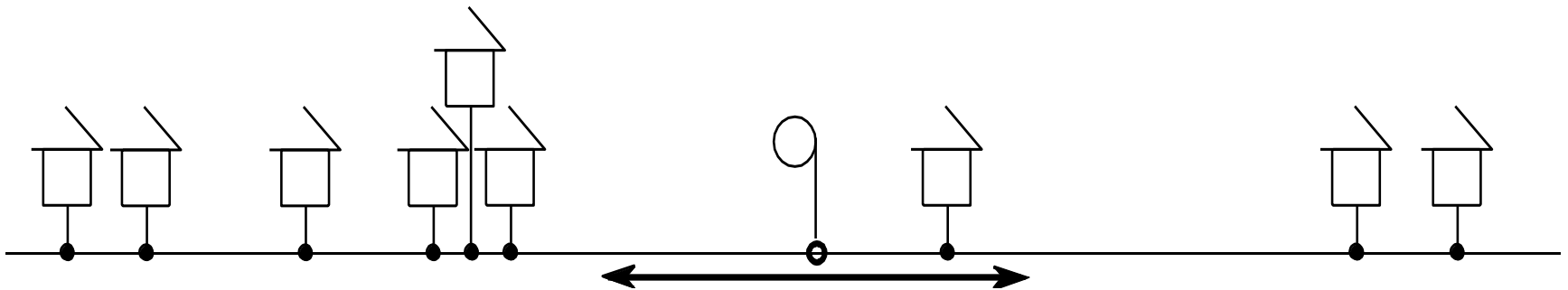
Úloha o zastávkach autobusu

- V obci Kocurkovo stojí niekoľko domov
 - Všetky stoja pri jedinej rovnej ceste, ktorá cez obec prechádza
 - O každom dome vieme, na koľkom metri cesty od začiatku obce má bránu
- Našou úlohou je rozmiestniť na ceste čo najmenej autobusových zastávok tak, aby sme nimi pokryli celú obec
 - musia byť umiestnené tak, aby to nik nemal z domu na zastávku ďalej ako 500 metrov



Úloha o zastávkach autobusu

- V Kocúrkove stoja domy na nasledujúcich súradniciach:
 - 100, 250, 600, 1000, 1100, 1200, 2300, 3300 a 3450 metrov od začiatku obce.
- Nájdite ručne čo najlepšie rozmiestnenie zastávok.
- Situáciu si môžeme znázorniť nasledovne:





Úloha o zastávkach autobusu

- Optimálne riešenie je postaviť štyri zastávky.
- Existuje veľa spôsobov, ako to spraviť
- Napríklad na súradniciach
 - 300 (prvé tri domy)
 - 1100 (štvrtý až šiesty dom)
 - 2800 (siedmy a ôsmy dom)
 - 3350 (posledný dom)
- Iné, rovnako dobré riešenie, na súradniciach
 - 600, 1200, 2300 a 3333.



Úloha o zastávkach autobusu

- Ako ale dokázať, že nám na túto obec nestačia tri zastávky?
- Ako sformulovať všeobecný algoritmus, ktorý bude túto úlohu riešiť a vždy nájde optimálne riešenie?

Optimálne riešenie vieme zostrojiť tak, že stále opakujeme nasledujúce kroky:

1. **Nájdeme prvý dom v obci, ktorý ešte pri sebe nemá zastávku.**
2. **Postavíme zastávku 500 metrov zaň.**



Problém rozvrhovania

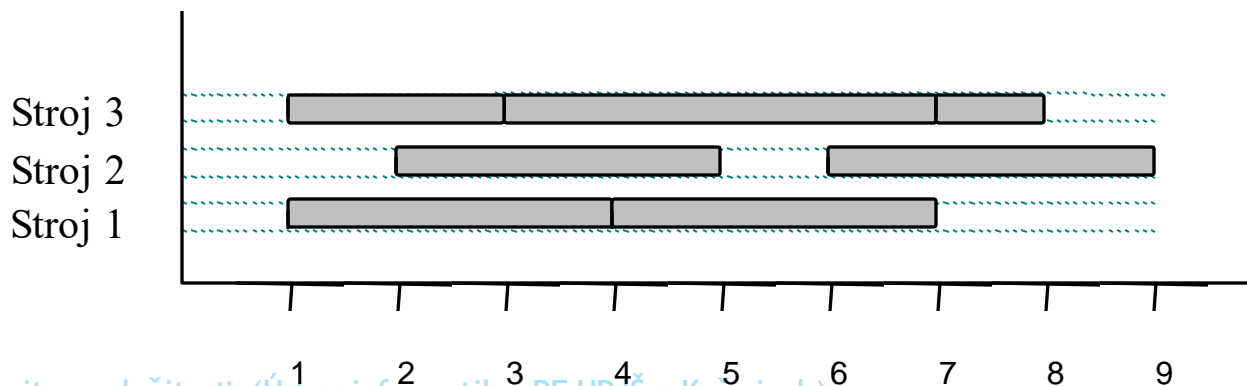
● Vstup:

množina T obsahujúca n úloh, z ktorých každá má:

- s_i – čas začiatku
- f_i - čas dokončenia; $s_i < f_i$
- $\langle s_i, f_i \rangle$ - časová perióda

● **Problém rozvrhovania:**

Vykonať všetky úlohy s minimálnym počtom strojov.





Problém rozvrhovania

- **Greedy stratégia:**

1. Utriedim úlohy podľa času štartu.
2. Zoberiem prvú úlohu a priradím ju stroju ktorý nič nerobí. Ak taky stroj nemám tak pridám stroj. Úlohu zo zoznamu odstránim.

Zložitost': **$O(n \log n)$** .



Problém rozvrhovania - príklad

- Vstup:

množina 7 úloh, ktoré majú časové periódy

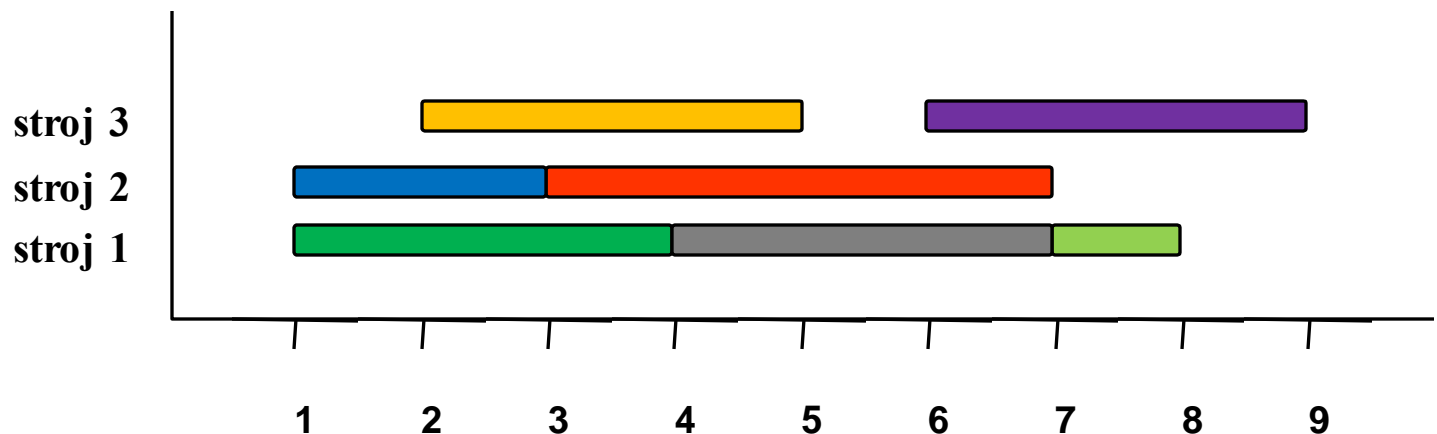
- [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

(usporiadané podľa začiatku)

- Výstup:

Minimálny počet strojov, na ktorých je možné vykonať všetky úlohy

- Riešenie:





Problém rozvrhovania

```
T = množina úloh;  
pocetStrojov = 0;  
while (!T.isEmpty){  
    úloha i = úloha z T, s najmenším  $s_i$ ;  
    if (existuje stroj m pre úlohu i)  
        prirad' úlohu i stroju m;  
    else {  
        pocetStrojov++;  
        prirad' úlohu i stroju pocetStrojov;  
    }  
    odstráň úlohu i z množiny T;  
}
```

Kým sú nejaké úlohy nepriradené



Problém rozvrhovania

● Správnosť:

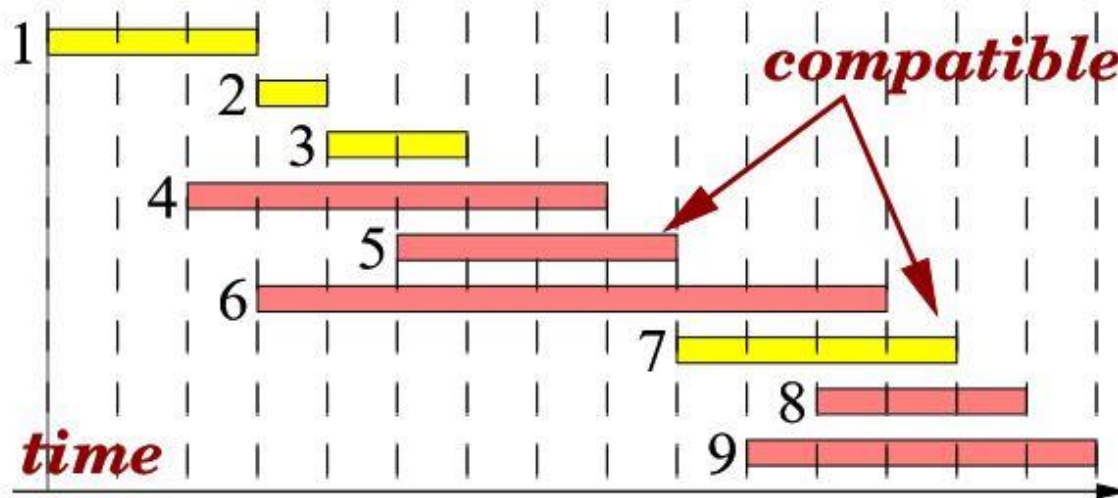
Sporom: Predpokladajme, že existuje lepší rozvrh.

- Predpokladajme, že môžeme použiť $k-1$ strojov
- Algoritmus používa k strojov
- Nech i je prvá úloha na stroji k
- Úloha i musí byť v konflikte s inými $k-1$ úlohami, pretože používame k -ty stroj
- Teda je k navzájom konfliktných strojov
- Ale to znamená, že sa nedá urobiť nekonfliktný rozvrh na $k-1$ strojoch.



Problém výberu úloh

- Vstup:
Množina úloh.
- **Problém výberu úloh**:
Vybrať najväčšiu množinu navzájom kompatibilných úloh.
- Hovoríme, že dve úlohy i a j sú kompatibilné, ak ich časové periódy sú disjunktné.





Problém výberu úloh

● Greedy stratégia:

- Najprv utriedime úlohy podľa času ukončenia úlohy od prvej po poslednú.
- Prechádzame vytvorený zoznam a úlohu pridáme do výberu, **ak je kompatibilná s už urobeným výberom.**
- Aká je efektívnosť tohto algoritmu?
Závisí to od triedenia.



Problém výberu úloh

Optimálnosť algoritmu:

„Greedy“ algoritmus vždy nájde množinu navzájom kompatibilných úloh.

Správnosť algoritmu:

Uvedený „greedy“ algoritmus **optimálne rieši** problém výberu úloh.

Dôkaz: Indukciou vzhľadom na n , počet úloh.

- nech $n = 1$: riešenie je určené jednoznačne a je optimálne.
- nech $n \geq 2$:
 predpokladajme, že tvrdenie platí pre všetky $0 < k < n$
 predpokladajme, že úlohy sú už utriedené podľa koncových časov
- Nech p je počet úloh v optimálnom riešení pre $[1, \dots, n - 1]$ a nech q je počet úloh pre $[1, \dots, n]$



Problém výberu úloh

- Platí $p \leq q$.

Lebo každé optimálne riešenie pre $[1, \dots, n - 1]$ je aj riešením (nie nutne optimálnym) pre $[1, \dots, n]$

- Platí nasledujúca nerovnosť $p \geq q - 1$?

Predpokladajme, že $p \leq q - 2$:

- Nech W je ľubovoľné optimálne riešenie pre $[1, \dots, n]$
- Nech $W_0 = W - \{n\}$, ak W obsahuje n a $W_0 = W$ inak
- Potom W_0 neobsahuje n a W_0 je riešením pre $[1, \dots, n - 1]$
- Toto je spor s predpokladom, že optimálne riešenie pre $[1, \dots, n - 1]$ má p úloh.



Problém výberu úloh

- 1. prípad: Predpokladajme, že $p = q$
 - Potom každé optimálne riešenie pre $[1, \dots, n - 1]$ je optimálne pre $[1, \dots, n]$
 - Podľa nášho indukčného predpokladu: keď $n - 1$ je preskúmané, tak je skonštruované optimálne riešenie pre $[1, \dots, n - 1]$
 - Teda, nebude nič pridané; inak by to bolo riešenie veľkosti $> q$
 - Preto algoritmus dá na výstupe riešenie veľkosti p , ktoré je optimálne



Problém výberu úloh

- 2. prípad: Predpokladajme, že $p = q - 1$
 - Potom každé optimálne riešenie pre $[1, \dots, n]$ obsahuje n .
 - Nech k je najväčšie $i: 1 \leq i \leq n - 1$, také, že $f_i \leq s_n$.
Keďže $f_1 \leq \dots \leq f_n$ potom
pre všetky $i: 1 \leq i \leq k$ platí, že i je kompatibilné s n
a pre všetky $i: k + 1 \leq i \leq n - 1$ platí, že i je nekompatibilné s n .
 - To znamená, že každé optimálne riešenie pre $[1, \dots, n]$ je zjednotením $\{n\}$ a optimálneho riešenia pre $[1, \dots, k]$.
Teda, každé optimálne riešenie pre $[1, \dots, k]$ má p úloh.
Z toho vyplýva, že žiadne optimálne riešenie pre $[1, \dots, k]$ nie je kompatibilné s ľubovoľným $k + 1, \dots, n - 1$.



Problém výberu úloh

- Nech W je množina úloh, ktoré algoritmus už má keď skončil k . Podľa našej indukčnej hypotézy, W je optimálne pre $[1, \dots, k]$. Teda, má p úloh.
- Algoritmus potom nebude pridávať žiadnu úlohu medzi $k + 1$ a $n - 1$ do W , ale pridá n do W . Algoritmus dá na výstupe $W \cup \{n\}$. Tento výstup má $q = p + 1$ úloh, a teda je optimálny pre $[1, \dots, n]$.



Problém plnenia batohu

Knapsack problem

- Zlodej sa vlámал do klenotníctva a chce si naplniť batoh pokladmi (chce ukradnúť čo najviac).
- Vstup:
Daných je n položiek $S = \{položka_1, položka_2, \dots, položka_n\}$, z ktorých každá má svoju váhu w_i a profit p_i
- Výstup:
Ktoré položky by mal zlodej vložiť do svojho batoha, s váhovou kapacitou W , aby dosiahol maximálny profit?



Problém plnenia batohu

- Najjednoduchší prístup by bol vygenerovať všetky možné podmnožiny šperkov a pre každú vypočítať profit. Potom zobrať podmnožinu s najväčším profitom.
- Toto vyžaduje ? času a nazýva sa ? algoritmus.





Problém plnenia batohu

Greedy stratégia

Príklad 1

Položka	Váha	Profit
w_1	25 kg	10 EUR
w_2	10 kg	9 EUR
w_3	10 kg	9 EUR

$W=30$



- Ukradne položku s **najväčším profitom**:
- Profit je **10**, hoci optimálny profit by mohol byť **18**



Problém plnenia batohu

Greedy stratégia

Príklad 2

Položka	Váha	Profit
w_1	5 kg	50 EUR
w_2	10 kg	60 EUR
w_3	20 kg	140 EUR

$W=30$



- Ukradne položku s **najväčším profitom vzhľadom na jednotku váhy**
- Vyberá v poradí [1, 3, 2]
- Profit je **190**, hoci optimálny profit by mohol byť **200**



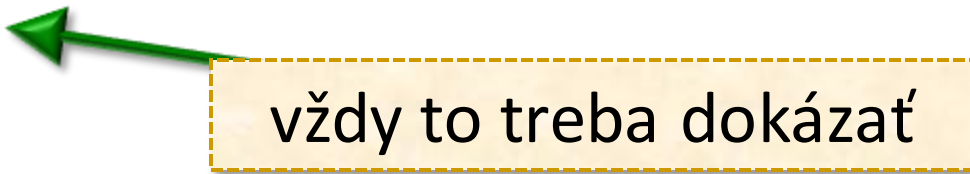
Problém plnenia batohu

Greedy stratégia

- „greedy“ prístup ku riešeniu tohto problému je nepoužiteľný na hľadanie optimálneho riešenia





- Greedy algoritmus je každý algoritmus, ktorý rieši daný problém na základe metaheuristického prístupu, **nájdением najlepšej lokálnej voľby**, pričom dúfame, že sa takto dopracujeme ku globálnemu optimálnemu riešeniu
- Existujú problémy, pre ktoré tento prístup dáva skutočne optimálne riešenie 
- Existujú problémy, pre ktoré tento prístup nedá globálny optimálny výsledok (ale niekedy to stačí)