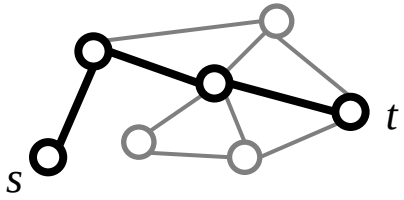
Algoritmy na ohodnoceném grafu

Dvě základní optimalizační úlohy:

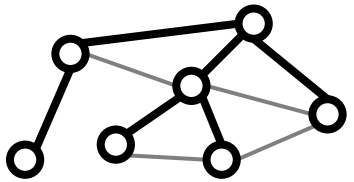
Jak najít nejkratší cestu mezi dvěma vrcholy?

- Dijkstrův algoritmus

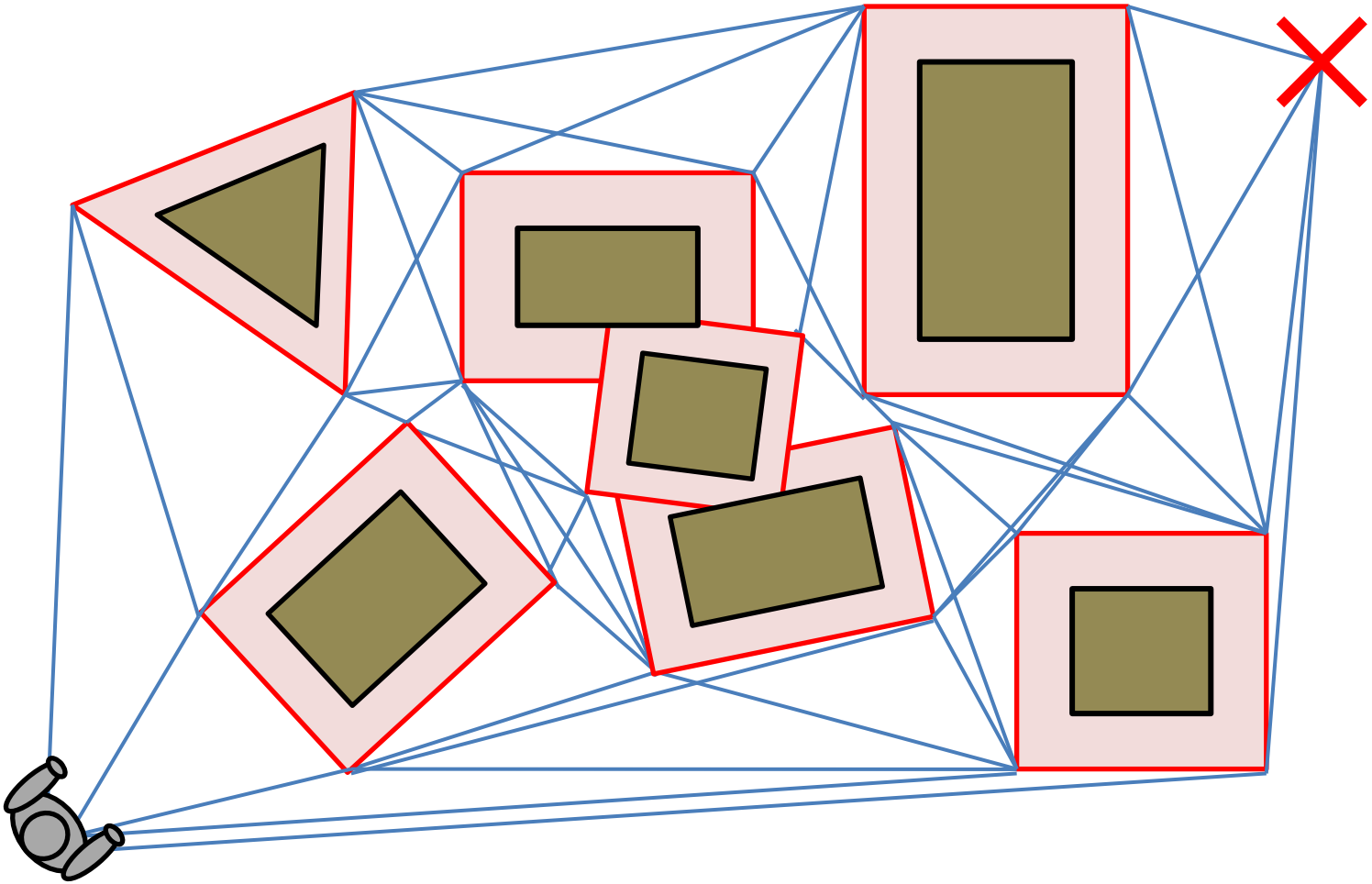


Jak najít minimální kostru grafu?

- Jarníkův a Kruskalův algoritmus



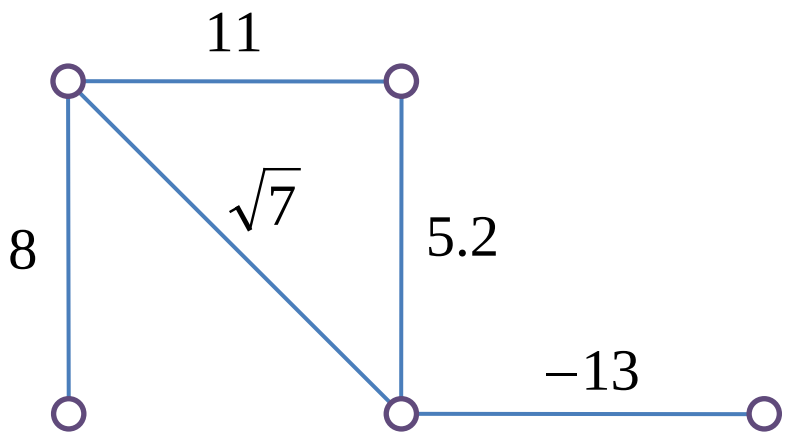
Plánování trasy robotu



Ohodnocený graf

Orientovaný nebo neorientovaný graf $G = (V, E)$, ke kterému přidáme funkci $c : E \to \mathrm{R}$

- funkce každé hraně $e \in E$ přiřazuje hodnotu $c(e)$



Nejkratší cesta v ohodnoceném grafu

$G = (V, E)$ .. ohodnocený orientovaný graf

P .. cesta v G

délka cesty P: $d(P) = \sum_{e \in P} c(e)$

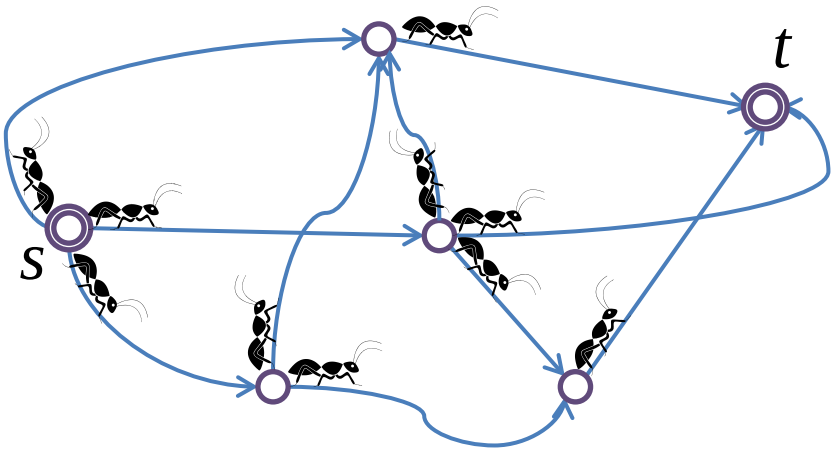
Budeme předpokládat, že ohodnocení je nezáporné.

$$\forall e \in E : c(e) \geq 0$$

Jsou-li dány vrcholy $s, t \in V$, jak nalezneme cestu nejkratší délky z s do t ?

Řešení za pomoci mravenců

- v každém $v \in V$ čeká $\deg_G^{out}(v)$ mravenců, každý na jiné výstupní hraně
- mravenci se pohybují stejnou rychlostí, délka hrany odpovídá jejímu ohodnocení
- v čase T_0 vystartují mravenci z vrcholu s
- jakmile mravenec *M* doběhne po své hraně do koncového vrcholu u, startují všichni mravenci čekající v u a *M* ze závodu odstupuje
- čekáme na prvního mravence, který se v čase T_1 objeví ve vrcholu t



Řešení za pomoci mravenců

Jak zjistíme, kudy vedla nejkratší cesta z s do t ?

- V každém vrcholu sledujeme, který mravenec sem dorazí jako první a poznamenáme si zde, odkud přišel (pokud dorazí jako první více mravenců současně, vybereme si jednoho z nich).
- Nejkratší cestu vystopujeme zpětně z t do s .

Cílem algoritmu bude projít vrcholy grafu v pořadí, které odpovídá času prvního dosažení mravencem (tj. min. vzdálenostem vrcholů od s).

Dijkstrův algoritmus

T .. množina vrcholů, pro které již známe výslednou cestu minimální délky z vrcholu *s*

F .. prioritní fronta

D[v] .. délka doposud zjištěné nejkratší cesty do *v*

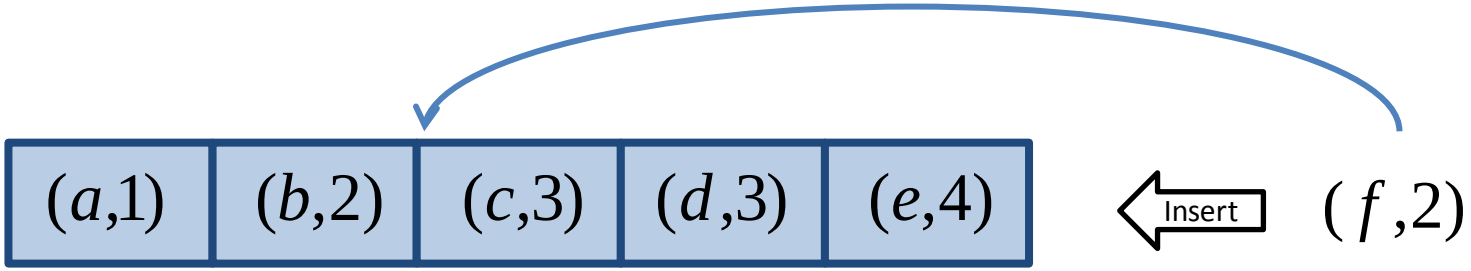
P[v] .. předek vrcholu v doposud zjištěné cestě

Dijkstrův algoritmus

```
Dijkstra(G,s):
    for each v in V do
        P[v]:= NULL; D[v]:= ∞;
    done
    D[s]:= 0; T:= Ø; // inicializace prázdnou množinou
    vytvoř prioritní frontu F z vrcholů ve V s klíči D[v];
    while not F->empty() do
        v := F->deleteMin(); T->add(v);
        for each w in Sousede[v] do
            if D[w] > D[v]+c((v,w)) then
                D[w]:= D[v]+c((v,w));
                P[w]:= v; F->changeKey(w,D[w]);
            endif
        done
    done
```

Prioritní fronta

- Datová struktura podobná frontě
- Každý prvek má navíc definovaný klíč (prioritu) $p \in \mathrm{R}$
- Pokud vložíme na konec nový prvek, předběhne ve frontě všechny prvky s vyšším klíčem



- Operace:

  Insert(v), DeleteMin, ChangeKey(v, newValue)

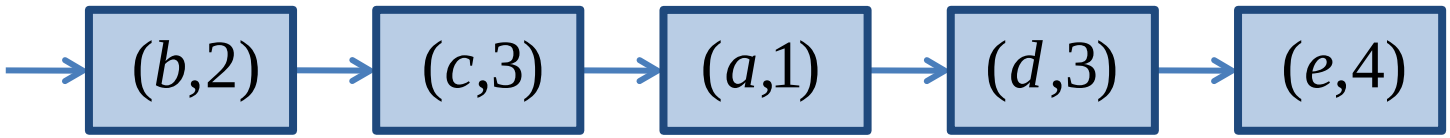
Prioritní fronta – s využitím seznamu

- Jednotlivé prvky ukládáme (neuspořádaně) do spojového seznamu
- Realizace operací:

  DeleteMin – seznam sekvenčně prohledáme $O(n)$

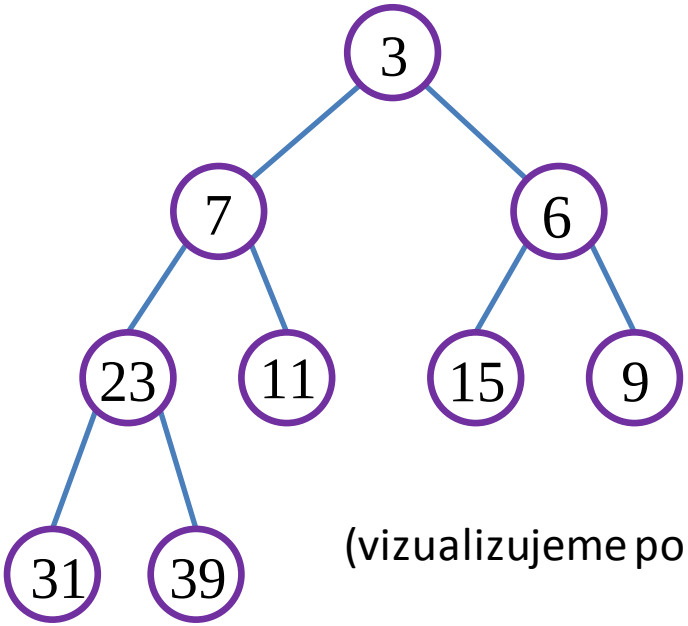
  Insert(v) – vkládáme na konec seznamu $O(1)$

  ChangeKey(v,p) – změníme pouze prioritu prvku v, pořadí v seznamu neměníme $O(1)$



Prioritní fronta – s využitím haldy

- Prvky ukládáme do binární haldy



(vizualizujeme pouze klíče)

- úplný binární strom
- každý vrchol má hodnotu klíče $\leq$ hodnotě klíče potomka

DeleteMin, Insert(v), ChangeKey(v, p) … vše $O(\log n)$

(každá z operací mění strukturu stromu v čase $O(\log n)$)

Dijkstrův algoritmus – časová složitost

Označme $|V| = n, |E| = m$

Čas je úměrný počtu operací provedených na *F*,

těch je max.: n x DeleteMin + m x ChangeKey +

n x Insert

Podle implementace F

- varianta seznam: $O(n^2 + m) = O(n^2)$
- varianta halda: $O((n+m)\log n)$

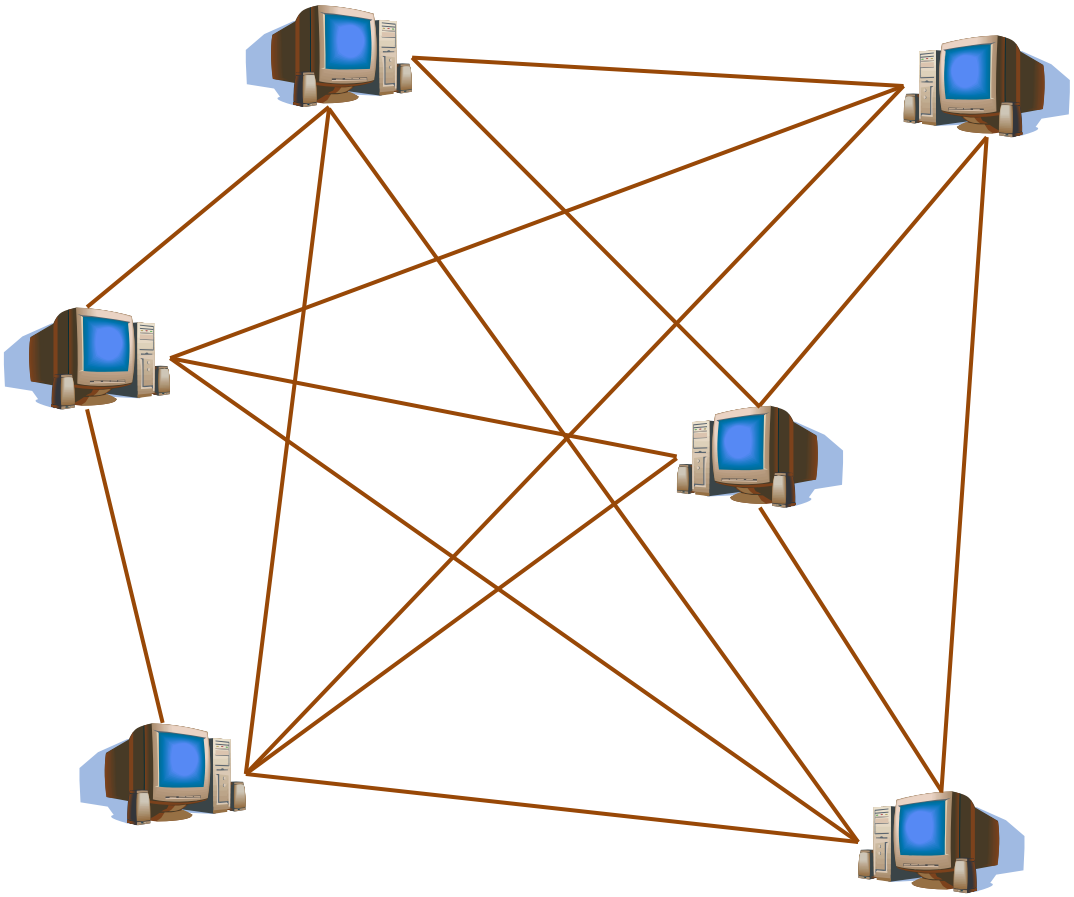
Je-li $m = O(n^2 / \log n)$, vyplatí se použít haldu, jinak je lepší seznam.

Další algoritmy na nejkratší cestu

- Bellman-Fordův
  - funguje pro libovolné ohodnocení hran
  - $O(|V|^3) = O(n^3)$

- Floyd-Warshallův
  - nalezne nejkratší cestu pro všechny dvojice vrcholů
  - hrany mohou být ohodnoceny i záporně, graf ale nesmí obsahovat cyklus záporné délky
  - $O(|V| \cdot |E|) = O(n \cdot m)$

Úloha s počítačovou sítí

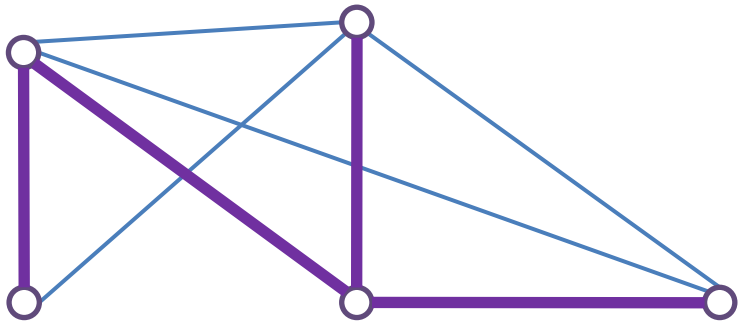


Kostra grafu

• Kostra neorientovaného souvislého grafu je souvislý faktor s minimálním počtem hran, tj. strom na všech vrcholech.

• Jednu z koster v daném grafu můžeme nalézt pomocí DFS nebo BFS.

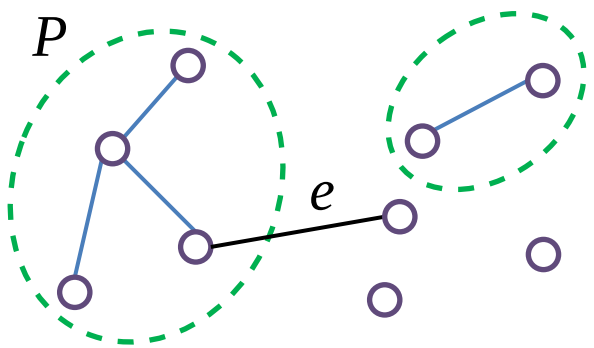
Jak nalezneme minimální kostru *T* v ohodnoceném grafu?



$$c(T) = \sum_{e \in T} c(e)$$

Obecný postup hledání kostry

```
Kostra-Generic(G,c):
    M:= Ø;
    while M není kostra do
        najdi vhodnou hranu a přidej ji do M;
    return M;
```



Předpokládejme:

- *M* je rozšiřitelná do minimální kostry
- *P* je komponenta *M*
- *e* je **nejlevnější** hrana vedoucí z *P* "ven"

Potom:

$M \cup \{e\}$ je též rozšiřitelná do minimální kostry

Union-Find problém

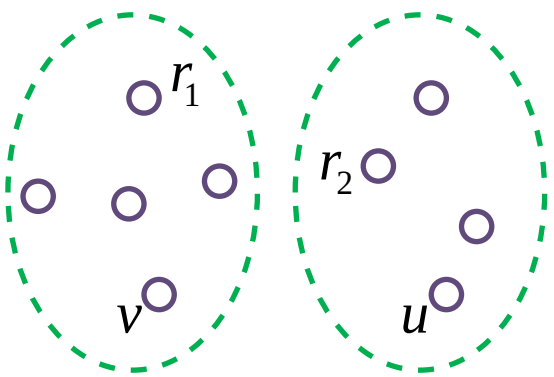
- Cíl: datová struktura pro evidenci komponent grafu.
- Pro každou komponentu vybereme jeden vrchol – její reprezentant.
- Jako r_v označíme reprezentanta komponenty, ve které leží vrchol v.

Požadujeme tyto operace:

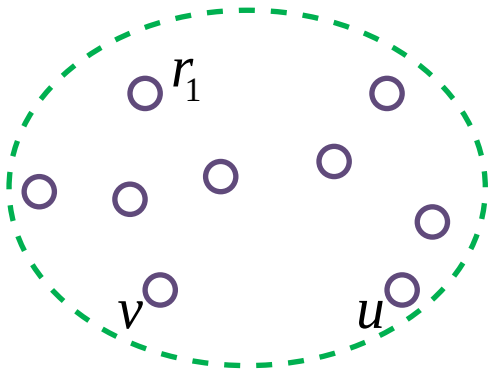
Find(v) – nalezne r_v

Union(u,v) – sjednotí komponenty obsahující vrcholy u a v.

$Find(v) = r_1$ $Find(u) = r_2$



$Union(u,v)$:

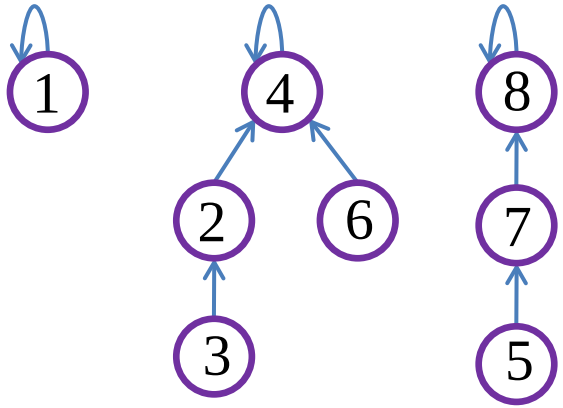


$Find(v) = r_1$

$Find(u) = r_1$

Union-Find problém – implementace

- Reprezentace komponent pomocí orientovaných stromů.



$P[v]$ .. předchůdce vrcholu ve stromu

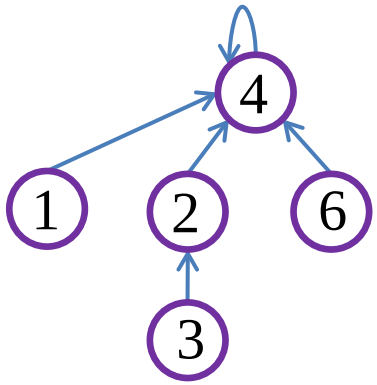
$rank[v]$ .. délka nejdelší orientované cesty vedoucí do v

Find(v):

projdeme z v do kořene

Union(u,v):

1. Nalezneme r_u a r_v
2. Pokud $r_u = r_v$, jsme hotovi
3. Sloučíme stromy, pro r_u a r_v, kořenem je vrchol s větším *rank*



Union-Find problém – časová složitost

Předpokládejme, že začínáme s $|V|$ komponentami, tj. máme graf s izolovanými vrcholy.

Pozorování:

1. Union(u,v) zvětší *rank* kořene maximálně o *1*
2. Je-li *rank* stromu k, má strom alespoň 2^k prvků

   $\Rightarrow$ nejvyšší možný *rank* je $\log|V|$

Důsledek:

Časová složitost Find i Union je $O(\log|V|)$

Kruskalův algoritmus

- Hladový algoritmus

1. Setřiď hrany podle jejich ohodnocení

$$c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$$

2. Inicializuj $M := \emptyset$. Procházej hrany v získaném pořadí. Pokud hrana e_i spojuje různé komponenty v M, přidej e_i do M.
3. Na konci obsahuje M hledanou minimální kostru.

Kruskalův algoritmus

Časová složitost:

1. Setřídění m hran v čase $O(m \log m) = O(m \log n)$
2. Při průchodu m-krát Find a $(n-1)$-krát Union

$$O((m+n-1)\log n) = O(m \log n)$$

Výsledná složitost: $O(m \log n)$

Jarníkův-Primův algoritmus

- Kostru vytváříme souvisle, začínáme s jedním vrcholem a postupně přidáváme nejlevnější hrany.
- Algoritmus je velmi podobný Dijkstrovu algoritmu a má i stejnou časovou složitost.

Jarníkův-Primův algoritmus

```
Jarnik(G,s):
    for each v in V do
        P[v]:= NULL; D[v]:= ∞;
    done
    D[s]:= 0; M:= Ø; // inicializace prázdnou množinou
    vytvoř prioritní frontu F z vrcholů ve V s klíči D[v];
    while not F->empty() do
        v := F->deleteMin(); M->add({v,P[v]});
        for each w in Sousede[v] do
            if D[w] > c({v,w}) then
                D[w]:= c({v,w});
                P[w]:= v; F->changeKey(w,D[w]);
            endif
        done
    done
```