Grafové algoritmy III.

Minimální kostra. Borůvkův/Kruskalův algoritmus. Jarníkův/Primův algoritmus.

Tomáš Bayer | bayertom@natur.cuni.cz

Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta UK.

Obsah přednášky

1. Úvod
2. Kostra grafu
3. Minimální kostra grafu
   - Borůvkův/Kruskalův algoritmus
   - Jarníkův/Primův algoritmus

1. Úvod

Minimální kostra grafu: Minimum Spanning Tree.

Podgraf, faktor, strom, minimální váha hran.

Faktor: podgraf G' obsahující všechny uzly G.

Úloha může mít více řešení.

Často řešeno v dopravě, energetice, telekomunikacích.

Optimální zásobovací síť, udržení sjízdnosti silnic, železniční spojení.

Elektronika, návrh plošných spojů (L1 norma).

Z hlediska spolehlivosti *není optimální*.

Výpadek 1 hrany: rozpad G na 2 nesouvislé grafy.

Důsledek: přerušení dodávky energie, neprůjezdnost (nehoda), atd.

Kritické části infrastruktury nutno posílit: zdvojení.

Předpoklad G neorientovaný, souvislý, nezáporné hrany.

2. Laplaceova matice grafu

Regulární matice $L(\|U\|, \|U\|)$.

Zachycuje stupně uzlů a incidenci.

Laplaceova matice $L = [l_{ij}]$

$$l_{ij} = \begin{cases} deg(u_j), & i = j, \\ -1, & i \neq j \wedge (u_i, u_j) \in G, \\ 0, & \text{jinak.} \end{cases}$$

Alternativně,

$$L = D - V,$$

kde $D(\|U\|, \|U\|)$ matice stupňů uzlu,

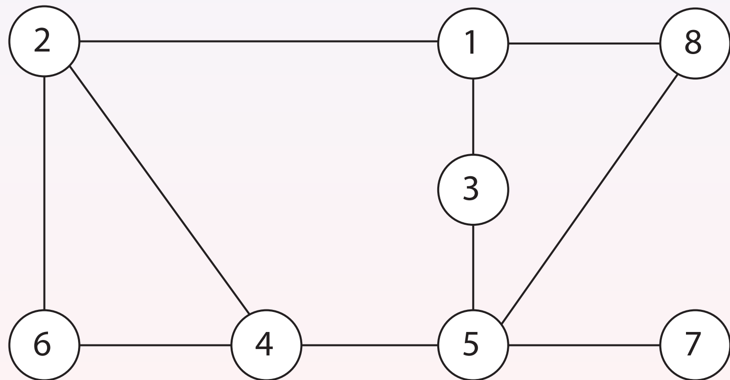
$$d_{ij} = \begin{cases} \sum_{j=1}^{\|U\|} v_{ij}, & i = j, \\ 0, & i \neq j, \end{cases}$$

$V(\|U\|, \|U\|)$ maticí incidence.

Úplný graf

$$L = \begin{bmatrix} n-1 & -1 & & -1 \\ -1 & n-1 & & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & n-1 \end{bmatrix}.$$

3. Výpočet Laplaceovy matice: G



4. Výpočet Laplaceovy matice pro G

$$
\begin{aligned}
L &= D - V, \\
&= \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}, \\
&= \begin{bmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ -1 & 0 & -1 & 3 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{bmatrix}.
\end{aligned}
$$

5. Kostra grafu

Graf $G = (H, U, \rho)$, neorientovaný, $m = \|H\|$, $n = \|U\|$.

$K = (H_k, U, \rho_k)$, faktor G, který je stromem, tvořena n hranami

$$n_k = \|H_k\| = \|U\| - 1 = n - 1.$$

Pro G existuje N_k koster K, výpočet z Laplaceovy matice/přibližně.

Kirchoffův zákon: odstranění $j-$tého řádku/sloupce $L \Rightarrow L_j$

$$N_k = \det(L_j),$$

hodnoty všech subdeterminantů L_j stejné.

Úplný graf G, zjednodušení (Cayley, 1889)

$$N_k = n^{n-2}.$$

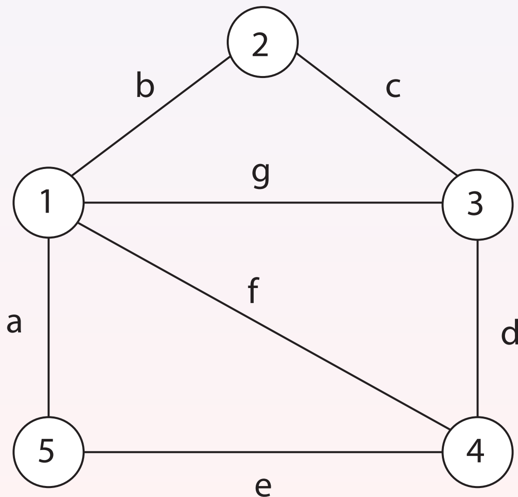
Přibližný vztah (Grimmett, 1976)

$$N_k \leq \frac{1}{n}\left(\frac{2m}{n-1}\right)^{n-1} = \frac{1}{n}\left(\frac{2m}{n_k}\right)^{n_k}.$$

Prohledávání všech koster neefektivní, exponenciální růst!

$n_k(10, 5) \leq 252$, $n_k(20, 10) \leq 184756$, $n_k(40, 20) \leq 137846528820$.

6. Kostry K, $n_k = 4$



7. Počet koster N_k

$$L = \begin{bmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ -1 & 0 & -1 & 3 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{bmatrix},$$

$$L_1 = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, L_2 = \begin{bmatrix} 4 & -1 & -1 & -1 \\ -1 & 3 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}, L_3 = \begin{bmatrix} 4 & -1 & -1 & -1 \\ -1 & 2 & 0 & 0 \\ -1 & 0 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

$$L_4 = \begin{bmatrix} 4 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & 0 \\ -1 & 0 & 0 & 2 \end{bmatrix}, L_5 = \begin{bmatrix} 4 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 3 \end{bmatrix}.$$

Kirchoffův zákon:

$$N_k = \det(L_1) = \det(L_2) = \det(L_3) = \det(L_4) = \det(L_5) = 21.$$

Přibližný výpočet:

$$N_k \leq \frac{1}{5} \left(\frac{2 \cdot 7}{4} \right)^4 \doteq 30.01.$$

8. Minimální kostra grafu

Bez ohodnocení grafu mají všechny K stejnou váhu.

Předpoklad: G souvislý, neorientovaný, bez záporných hran.

Hledáme $K = (H_k, U, \rho_k)$

$$\sum_{h \in H_k} w(h) = \min .$$

Úloha není jednoznačná, existence více minimálních koster.

Přehled algoritmů:

- Borůvkův (Kruskalův): $O(\|H\| \log \|U\|)$.
- Jarníkův (Primův): $O(\|H\| \log \|U\|)$.
- Greedy strategie.

Randomizované algoritmy: lineárním čas.

9. Borůvkův (Kruskalův) algoritmus (1/2)

Objeven prof. Otakarem Borůvkou (1899-1995) v roce 1926.

Připojení každé obce na jižní Moravě energetické sítě.

Minimalizace nákladů na připojení: délka vedení, počet stožárů.

V roce 1956 znovu objeven v USA J. P. Kruskalem.

Heuristika, v každém okamžiku hledáno lokální minimum.

Do kostry přidáváme hranu s minimální vahou spojující 2 podstromy.

Pokračujeme, dokud les není stromem.

Opakované sjednocování: UNION, 2 podstromy.

Strom: souvislý, neobsahuje kružnici.

Les: nesouvislý, neobsahuje kružnici, tvořen posdstromy.

10. Borůvkův (Kruskalův) algoritmus 2/2

Předzpracování: setřídění hran dle w.

Implementace využívá prioritní frontu.

Opakované přidávání hrany hs aktuálně nejnižším w, aby nevznikla kružnice.

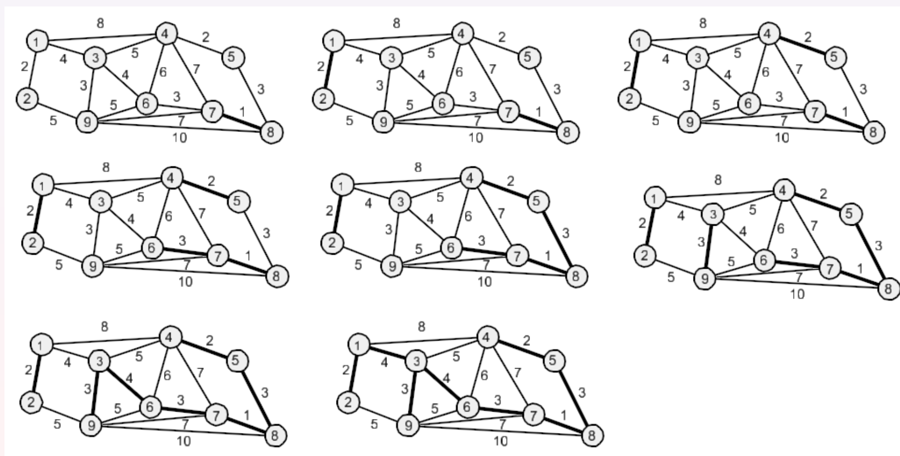
Mohou nastat 4 situace:

1. *Žádný uzel h neleží v jiném podstromu*
   Hrana vytvoří nový podstrom.
2. *Jeden uzel h součástí některého z podstromů*
   Hranu připojíme k podstromu.
3. *Oba uzly h v různých podstromech*
   Oba podstromy spojíme do jednoho.
4. *Oba uzly h v jednom podstromu*
   Přidání h vede ke kružnici, zahodíme.

Implementace množinových operací nad stromem:

- MAKE_SET(x): vytvoření nové množiny s jedním prvkem x (identifikátor).
- FIND_SET(x): vrací identifikátor množiny obsahující prvek x.
- UNION(x, y): sjednocení podmnožin x, y do jedné podmnožiny.

11. Ukázka Borůvkova algoritmu



12. Ukázka operace UNION(x, y)

#	h	$w(h)$	*Disjunktní podmnožiny*								
0	-	-	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$	$\{6\}$	$\{7\}$	$\{8\}$	$\{9\}$
1	$(7,8)$	1	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$	$\{6\}$	$\{7,8\}$		$\{9\}$
2	$(1,2)$	2	$\{1,2\}$		$\{3\}$	$\{4\}$	$\{5\}$	$\{6\}$	$\{7,8\}$		$\{9\}$
3	$(4,5)$	2	$\{1,2\}$		$\{3\}$	$\{4,5\}$		$\{6\}$	$\{7,8\}$		$\{9\}$
4	$(6,7)$	3	$\{1,2\}$		$\{3\}$	$\{4,5\}$		$\{6,7,8\}$			$\{9\}$
5	$(3,9)$	3	$\{1,2\}$		$\{3,9\}$	$\{4,5\}$		$\{6,7,8\}$			
6	$(5,8)$	3	$\{1,2\}$		$\{3,9\}$	$\{6,7,8,4,5\}$					
7	$(3,6)$	4	$\{1,2\}$		$\{6,7,8,4,5,3,9\}$						
8	$(1,3)$	4	$\{6,7,8,4,5,3,9,1,2\}$								

Minimální kostra G:

$$K = \{6,7,8,4,5,3,9,1,2\}, w(K) = 22.$$

13. Pseudokód B/K algoritmu

Na vstupu graf $G = (H, U, \rho)$, ohodnocení $w(h)$.

```
T ← ∅.                                    #Empty tree
for ∀u ∈ U :                              #For each node
    MAKE_SET(u)                           #Create U-F structure
sort(H,w(H))                              #Sort edges by weights
for ∀h = (u, v) :
   if FIND_SET(u) != FIND_SET(v):         #u,v in different sets
      T ← h                               #Add edge to T
      UNION(u, v)                         #Make union of sets containing u,v
```

Pseudokód jednoduchý, implementace obtížnější.

Klíčové faktory:

- efektivita nalezení hrany $h(u, v)$ v podstromu: FIND_SET(u),
- efektivní implementace sjednocení u, v: UNION(u, v).

Buď umíme rychleji hledat nebo rychleji spojovat.

14. Varianty UNION-FIND

UNION-FIND = UNION + FIND_SET (viz dále).

Datová struktura UNION-FIND:

- *Spojový seznam (Array-based)*
  FIND_SET $O(1)$, UNION $O(\log_2(n))$.
- *Strom (Tree-based)*
  FIND_SET $O(\log_2(n))$, UNION $O(1)$.
  Nejrychlejší známá implementace.

V praxi preferována varianta 2.
Téměř všechny implementace Tree-based.

2 heuristiky:

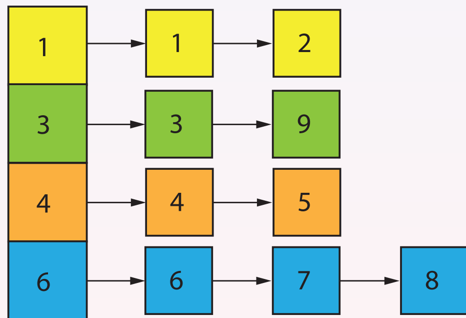
Weighted Union: připojování menšího za větší (AB, TB).

Path Compression: zkracování výšky tromu (TB).

15. UNION-FIND, AB

Linked list uchovává identifikátor, první prvek.

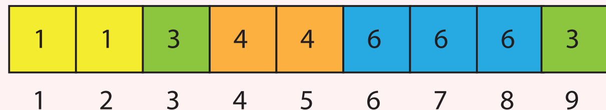
UF: linked list, identificators



UF: sizes

1	2
3	2
4	2
6	3

UF: 1D array, index



16. UNION-FIND, AB, popis operací

MAKE_SET(u):

Vytvoření datové reprezenentace: spojový seznam/pole s délkou 1(+identif.), index.

Setřídění, celkově $O(m \log_2 n)$.

FIND_SET (u).

Velmi efektivní díky indexačnímu poli, $O(1)$.

Celkově $O(2m)$.

UNION(u, v):

Zjištění délky u, v.

Za delší připojíme kratší: *Weighted Union*.

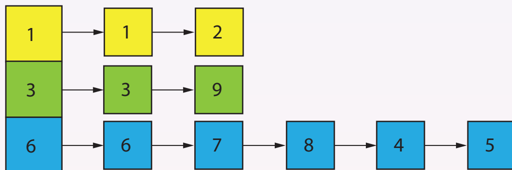
Změna identifikátorů pro kratší seznam.

Delší seznam neměněn.

Složitost $O(\log_2 n)$, celkově $O(n \log_2 n)$.

Celkově: $m \log_2 n + 2m + n \log_2 n$.

17. Ukázka UNION(4,6)



UF: sizes

1	2
3	2
6	5

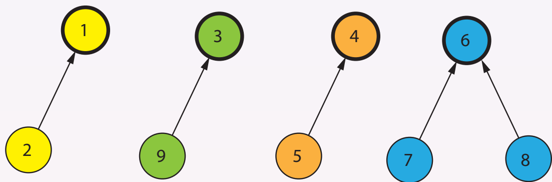
UF: 1D array, index

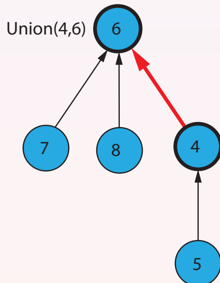
1	1	3	6	6	6	6	6	3
1	2	3	4	5	6	7	8	9

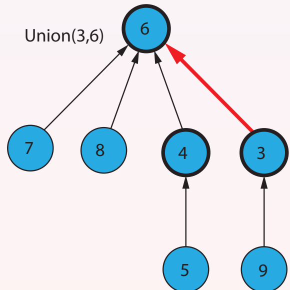
$\|u\| = 2$, $\|v\| = 3$: $v \leftarrow u$.

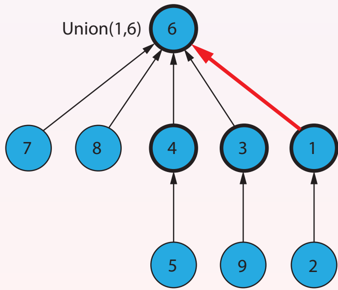
```
if len(u)<len(v):
  for i in u:                   #Process all  elements in shorter arra
    index[i] = v                #Change index to all x elements
    size[v] = size[v] + size[u] #Change size
```

18. Union-Find, TB









19. Union-Find, TB, datové struktury

Každá podmnožina *kořenovým stromem*:

Její prvky odkazují na otce (parent) $\Rightarrow$ kořen.

Aneb pro každý prvek stromu ukládán parent, který je identifikátorem.

Hodnost stromu h (rank):

$$h = \log_2 n_s,$$

odhad výšky stromu, n_s počet uzlu ve stromu, netřeba znát přesně.

Path compression:

Zkracování cesty (stromu) $\Rightarrow$ snižování výšky.

Heuristika, dvojprůchodová, rekurzivní implementace.

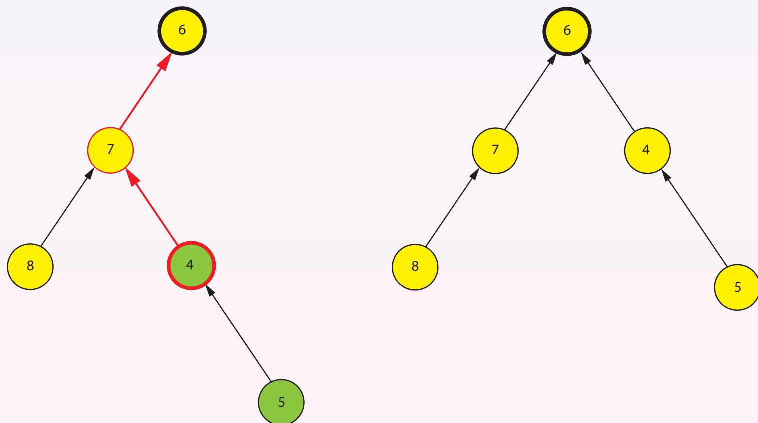
Postup PC:

1. průchod: nalezení kořene stromu.

2. průchod: každému prvek na cestě ke kořenu upraven odkaz na rodiče.

Tyto prvky připojeny na kořen stromu.

20. Path compression (4)



1. Od (4) rekurzivně vzhůru, kořen (6).
2. Přepojení všech uzlů cesty <4,7,6> ke kořeni.

21. Implementace PC

Součást operace FIND_SET.

Velmi jednoduchá implementace, rekurze.

Volána při operaci UNION: proto nazývána UNION-FIND.

Přímý chod: dokud existuje předhůdce, pokračuj do kořene.

Zpětný chod: update předchůdců od kořene.

```
def find(u, p):
    if p[u] != u:               #Node is not root
        p[u] = find(p[u], p)    #Find its predecessor
    return p[u]                 #Return root
```

Časová složitost $O(\log_2 n)$.

22. Union-Find, TB, popis operací

MAKE_SET(u):

Vytvoření datové reprezenentace: stromy tvořené jedním prvkem.

$p[u] = u$, $r[u] = 0$.

Setřídění, celkově $O(m \log n)$

FIND_SET (u).

Path Compression, snižujeme výšku stromu, $O(\log_2 n)$.

Volána při UNIONu.

Celkově $O(2m \log_2 n)$.

UNION(u, v):

Vyvažování stromu při sjednocení.

Kořen "menšího" stromu připojen na kořen "většího" dle r: Weighted Union.

Pokud jsou stejné, vybereme libovolný z nich.

Složitost $O(1)$, celkově $O(n)$.

Celkově: $m \log n + 2m \log_2 m + n$.

23. Operace UNION(u, v)

Pro každý uzel u nutno uchovávat 2 informace:

- parent (rodič) $p[u]$, inicializace $p[u] = u$,
- hodnost (rank) podstromu $r[u]$, inicializace $r[u] = 0$.

Pro p, r použity seznamy.

Připojujeme -li menší podstrom k většímu, rank většího se nemění,

$$r = \max(r[u], r[v]).$$

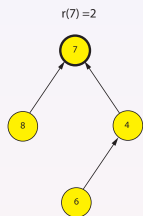
Rank updatován, pokud $r[u] = r[v]$,

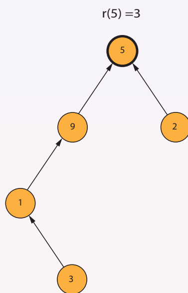
$$r = r[u] + 1 = r[v] + 1.$$

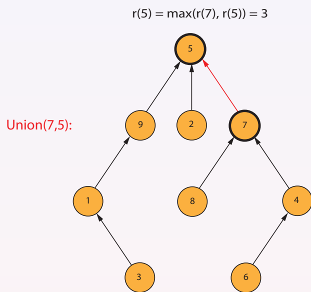
Postup:

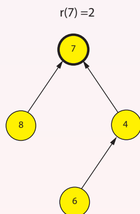
1. Nalezení kořenů R_u, R_v stromů obsahujících u, v + komprese obou cest.
2. Pokud u, v jiném podstomu, porovnáme hodnosti.
   a) Jestliže $r[u] > r[v]$, připojíme v k u: $p[R_v] = R_u$.
   b) Jestliže $r[v] > r[u]$, připojíme u za v: $p[R_u] = R_v$.
   c) Při shodě $r[u] = r[v]$: $p[R_v] = R_u$ (např.).
   Update ranku $r[u] = r[u] + 1$.

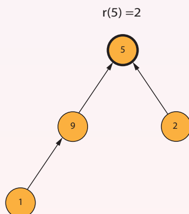
24. Union(u, v), update ranku, ukázka

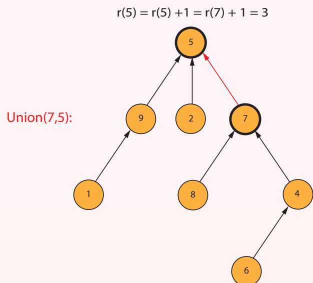












25. Implementace UNION(u, v)

Tree-based varianta.

Kombinace UNION + FIND.

Při hledání uzlu prováděna komprese.

```
def union(u, v, p, r):
    root_u = find(u, p)              #Find root for u + compress
    root_v = find(v, p)              #Find root for v + compress
    if root_u != root_v:             #u, v in different subtrees
        if r[root_u] > r[root_v]:    #u subtree is longer
            p[root_v] = root_u       #Connect v to u
        elif r[root_v] > r[root_u]: #v subtree is longer
            p[root_u] = root_v       #Connect u to v
        elif u!= v:                  #u, v have equal lengths
            p[root_u] = root_v       #Connect u to v
            r[root_v] = r[root_v]+1 #Increment rank
```

25. Datová reprezentace grafu

Kombinovaná reprezentace: seznam vrcholů V a hran E.

Reprezentace vrcholů:

Seznam vrcholů V v G.

```
V = [v1, v2, ...,vk]
```

Reprentace hran:

Hrana: uspořádaná trojice - počáteční uzel u, koncový uzel v, ohodnocení w.

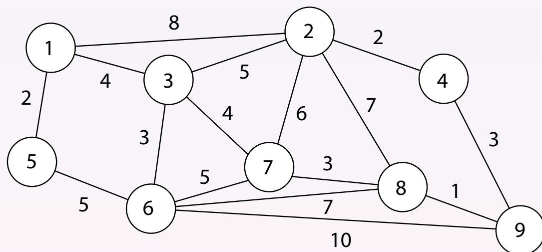
Hrany: spojová reprezentace, seznam hran E.

```
E = [
      [u1, v1, w1],
      [u2, v2, w2],
       ....
      [uk, vk, wk],
    ]
```

Hrany lze snadno setřídit dle w.

Dříve používané reprezentace umožňují snadnou konverzi na smíšenou.

26. Ukázka reprezentace grafu



Reprezentace uzlů:

```
V = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

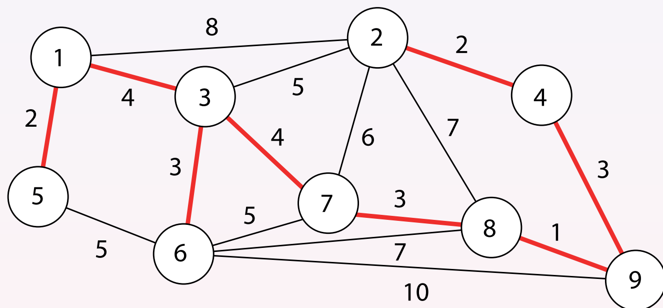
Reprezentace hran:

```
E = [[1, 2, 8], [1, 3, 4], [1, 5, 2], [2, 8, 7], [2, 1, 8], [2, 3, 5],
     [2, 4, 2], [2, 7, 6], [3, 1, 4], [3, 2, 5], [3, 6, 3], [3, 7, 4],
     [4, 9, 3], [4, 2, 2], [5, 1, 2], [5, 6, 5], [6, 8, 7], [6, 9, 10],
     [6, 3, 3], [6, 5, 5], [6, 7, 5], [7, 8, 3], [7, 2, 6], [7, 3, 4],
     [7, 6, 5], [8, 9, 1], [8, 2, 7], [8, 6, 7], [8, 7, 3], [9, 8, 1],
     [9, 4, 3], [9, 6, 10]]
```

27. Implementace Borůvkova-Kruskalova algoritmu

```
def mstk(V, E):
    T=[]                                        #Empty tree
    wt = 0                                      #Sum of weights of T
    p = [inf] * (len(V) + 1)                    #List of predeccessors
    r = [inf] * (len(E) + 1)                    #Rank of the node
    for v in V:                                 #Make set
        make_set(v, p, r)
    ES = sorted(E, key=lambda it:it[2])         #Sort edges by w
    for e in ES:                                #Process all edges
        u, v, w = e                             #Take an edge
        if (find(u, p) != find(v, p)):          #u, v in different trees?
            union(u, v, p, r)                   #Create union
            T.append([u, v, w])                 #Add edge to tree
            wt = wt + w                         #Compute weight of T
    return wt, T
```

28. Výsledky, ukázka




```
wt, T = mst(V,E)
print(T)
>> [[8, 9, 1], [1, 5, 2], [2, 4, 2], [3, 6, 3], [4, 9, 3],
    [7, 8, 3], [1, 3, 4], [3, 7, 4]]
print(wt)
>> 22
```

27. Jarníkův (Primův algoritmus)

Objeven prof. Vojtěchem Jarníkem (1897-1970) v roce 1930.
Prof. Jarník působil v letech 1921-23 na PřF UK.
Znovu objeven: 1957, R. C. Prime, 1959 E. W. Dijkstra.

Borůvkův algoritmus pracuje současně s více podstromy, které sjednocuje.
Jarníkův algoritmus pracuje pouze s jedním podstromem, který rozšiřuje.

Využívá Greedy strategii.
Velmi podobný Dijkstra, liší se *způsobem relaxace*.
K aktuálnímu uzlu u hledáme v minimalizující $d[v] = w(u, v)$.
Vrcholy možno značkovat.

Implementačně jednodušší než Borůvkův/Kruskalův.
Klíčový prvkem rychlost hledání v prioritní frontě Q.
Při implementaci Q haldou $O(n)$, pomalé, nutno značkovat $O(1)$.

28. Popis Jarníkova/Primova algoritmu

Uzly, které nejsou v podstromu, uloženy v prioritní frontě Q setříděné dle $d[v]$.
Pokud nelze v Q hledat $O(n)$, nutno značkovat $O(1)$.
Při inicializaci všem $u \in U$ nastaveny $d[u] = \infty$.

Lze začít od libovolného uzlu.
Jeden označen jako root r, $d[r] = 0$, zpravidla $r = 0$.

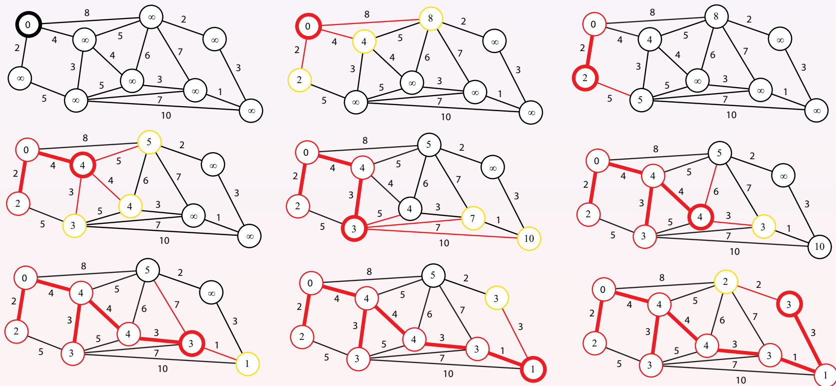
Dokud Q není prázdná vyjmeme uzel $(d[u], u)$.
Pro každé dosud nenavštívené v incidující s u relaxace

$$d[v] > w[u][v] \rightarrow d[v] = w[u][v], p[v] = u.$$

Hodnota $d[v]$ neobsahuje celkovou délku cesty $s \rightarrow u$ (Dijkstra).
Reprentuje vzdálenost od kostry $d[v] = w(u, v)$.

Pro uzel v uložíme předchůdce $p[v]$.
Následně přidáme $(d[v], v)$ do Q:
Rekonstrukce kostry: nalezneme $p[u]$ pro všechna $u \in U$.

29. Ukázka Jarníkova/Primova algoritmu



30. Implementace Jarníkova/Primova algoritmu

Reprezentace G spojovým seznamem.

```
def mstj(G, r):
    s = ['N'] * (len(G) + 1)          #No vertex is in mst
    d = [float('Inf')] * (len(G) + 1) #Set infinite distances
    p = [-1] * (len(G) + 1)           #No predecessors
    Q = queue.PriorityQueue()         #Priority queue
    d[r] = 0                          #Root d[r] = 0
    Q.put((0, r))                     #Add start vertex
    while not Q.empty():
        du, u = Q.get()               #Pop first element
        s[u] = 'C'                    #Node belongs to mst, close
        for v, dv in G[u].items():
            if s[v] == 'N':           #Relaxation, v not in mst
                if d[v] > dv:         #We found a better node
                    d[v] = dv         #Update distance from mst
                    p[v] = u          #Update predecessor
                    Q.put((d[v], v))  #Add to Q
```

Ukázka:

```
mstj(G,1)
>> [-1, 4, 1, 9, 1, 3, 3, 7, 8]
```

31. Výsledky, ukázka