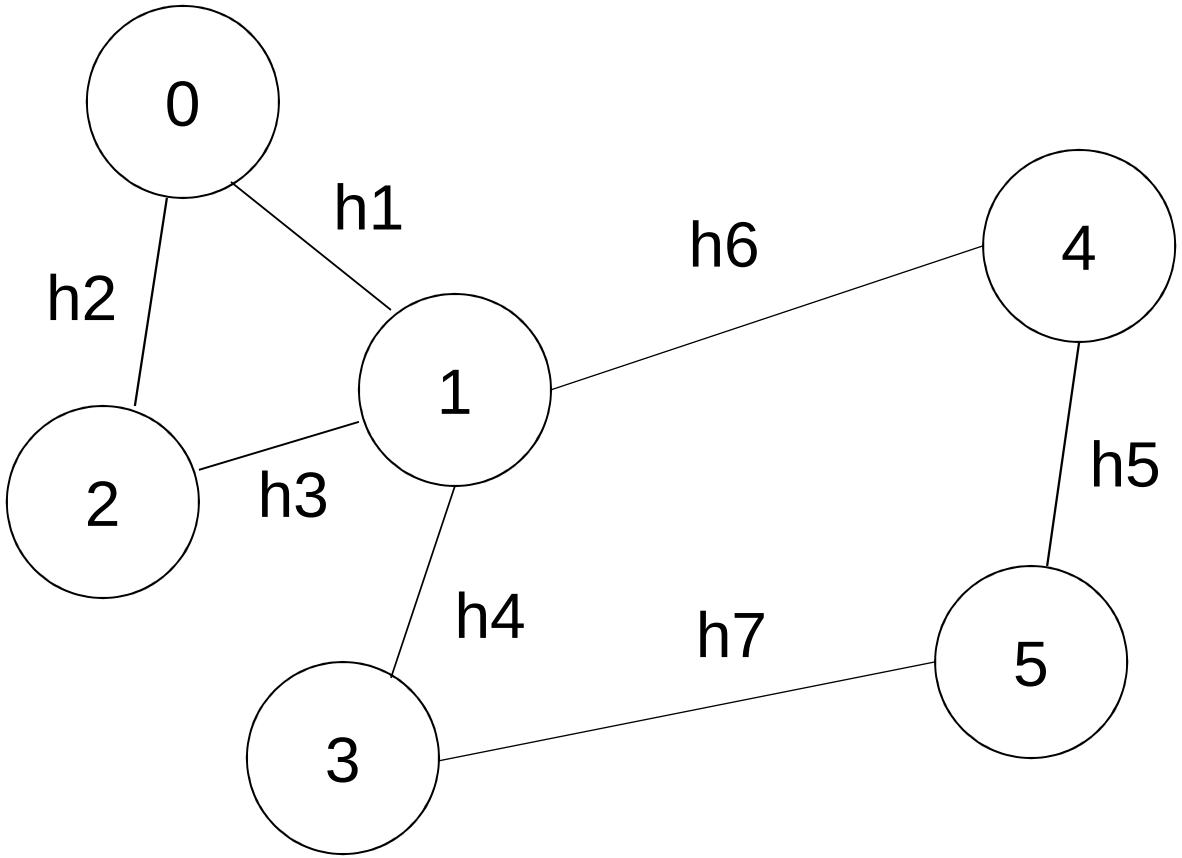
Kostry a minimální kostra

Kostra grafu

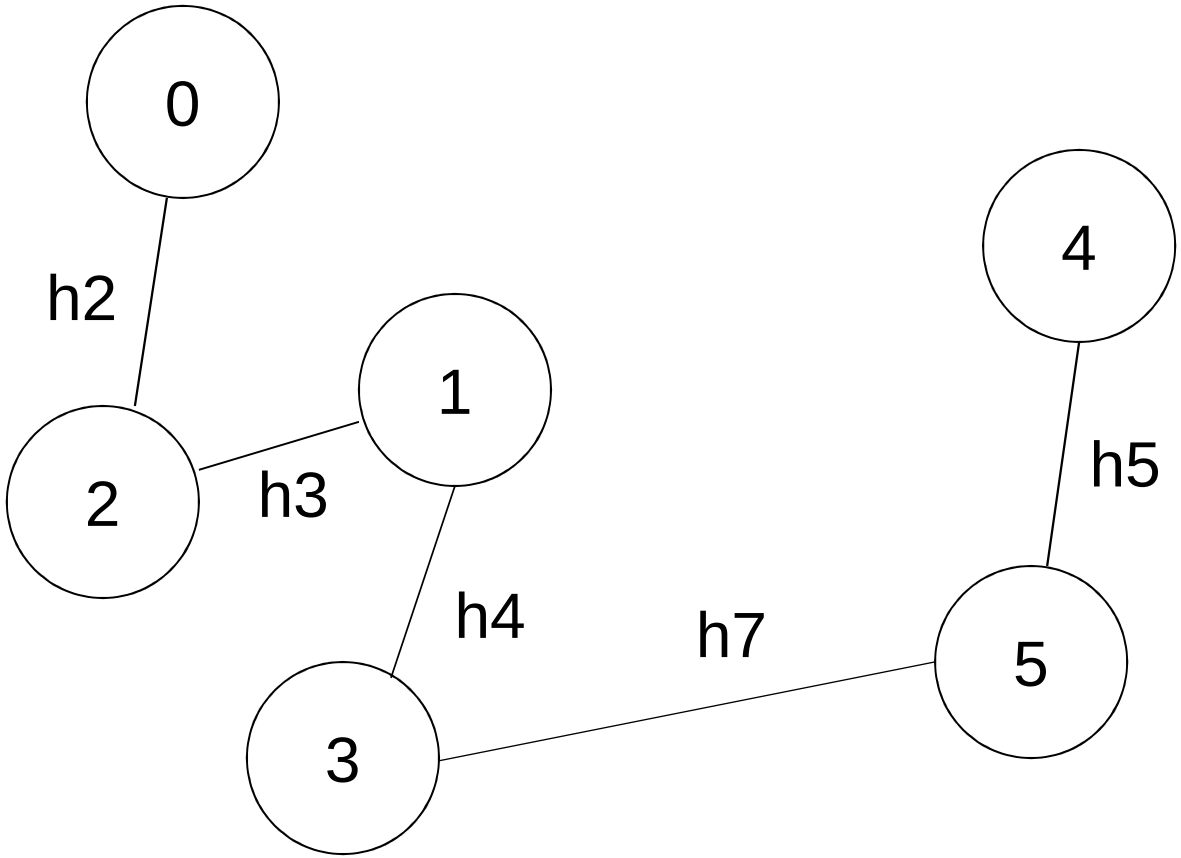
Kostra K (resp. T) souvislého neorientovaného grafu G = (V,E,I)

- takový faktor grafu G, který je stromem
  - faktor grafu G = graf, který má stejnou množinu uzlů jako graf G
  - strom = souvislý graf bez kružnic
  - souvislý graf = mezi každými dvěma uzly existuje cesta
- K = (V,E_v,I), E_v je podmnožinou E
- ke grafu G existuje několik koster

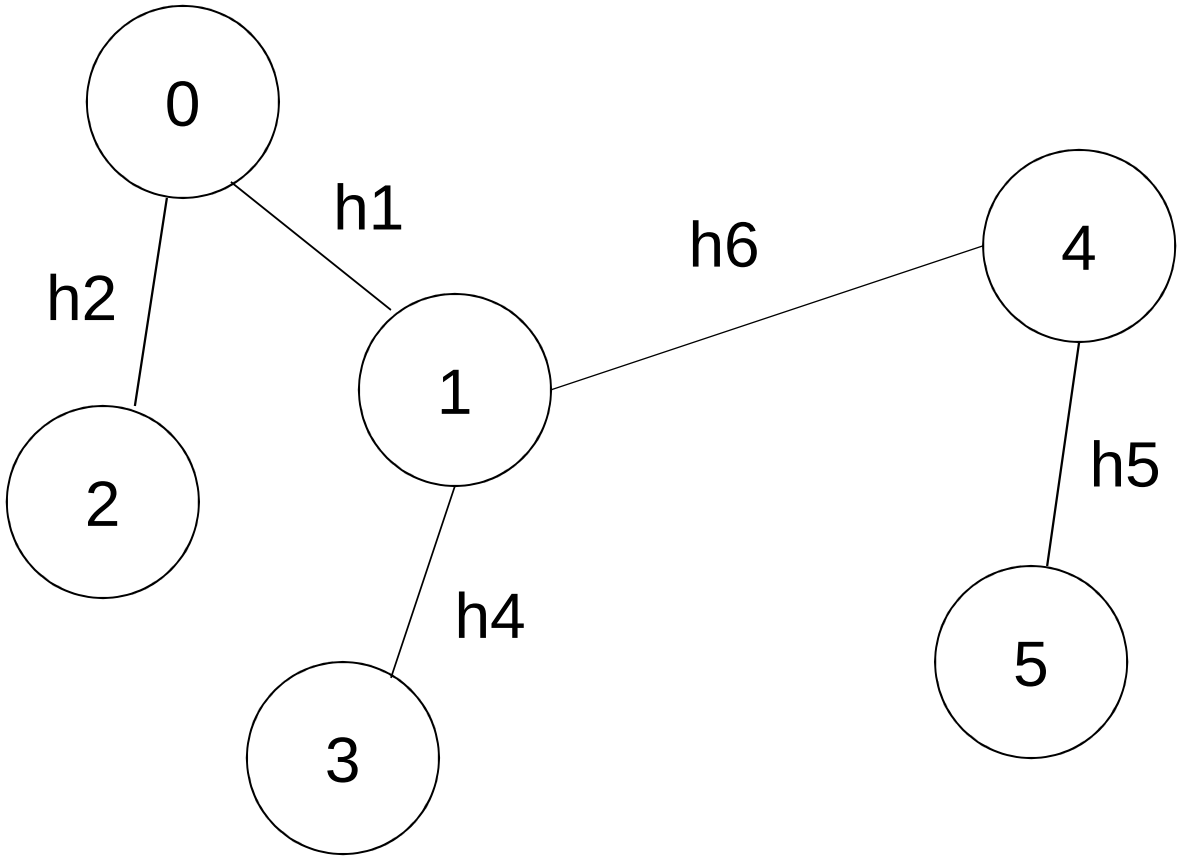
Graf



Kostra K1



Kostra K2



Kolik existuje koster grafu?

- má-li graf $n = |V|$ uzlů, jeho každá kostra má (n-1) hran
  - přidáním jedné hrany vznikne kružnice
- graf G má tolik koster, kolika způsoby lze vybrat (n-1)-tic hran z množiny hran E

$$\binom{|E|}{(n-1)}$$

Příklad: graf ze snímku č. 3

- počet uzlů *n* = |V| = *6*, kostra bude mít 6-1 = 5 hran
- počet hran grafu G |E| = 7
- počet koster grafu G:

$$\binom{7}{5} = \frac{7!}{(7-5)! \cdot 5!} = \frac{7 \cdot 6}{2!} = \frac{7 \cdot 6}{2} = 21$$

Generování všech koster

- algoritmus systematicky generuje faktory postupným přidáváním hran
  - jde o algoritmus s návratem (backtracking)
    - generuje strom řešení prohledáváním prostoru všech kombinací hran
  - testuje, zda přidáním hrany nevznikla kružnice
    - pokud ano, provede se návrat
    - pokud ne
      - je-li počet hran (n-1), uloží kostru a provede návrat
      - jinak zkusí přidat další hranu

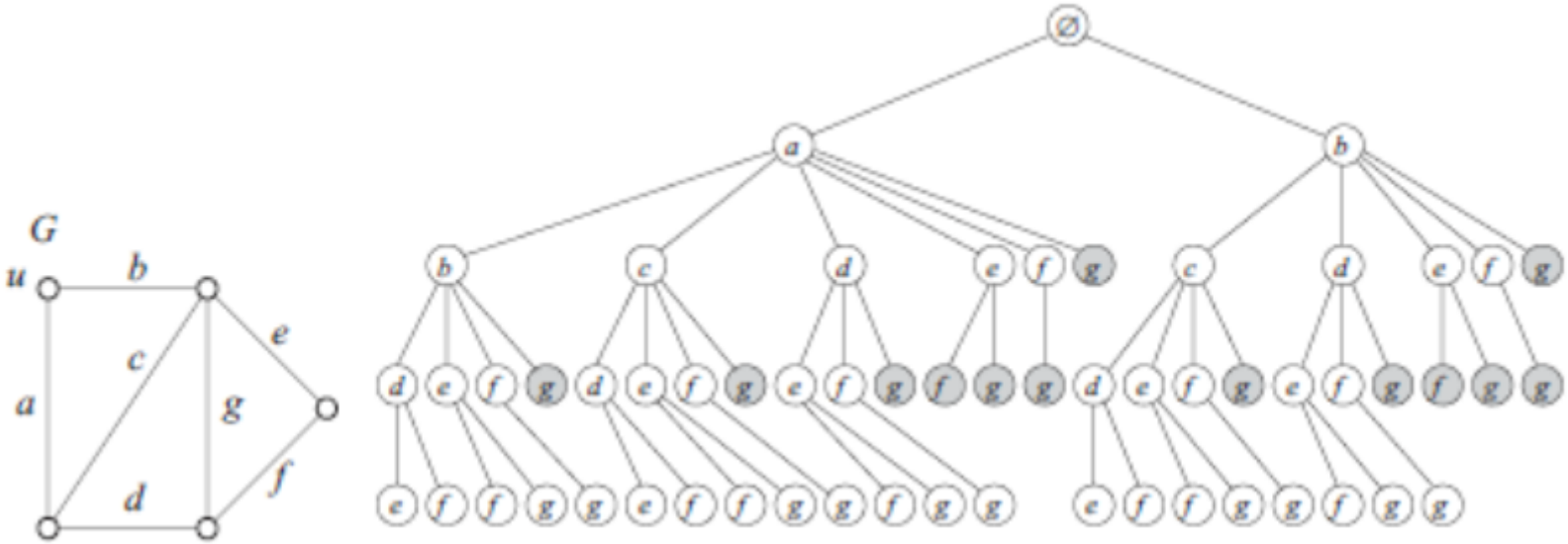
Generování všech koster

```
GSP(G)
1   INIT_STACK(T)                    Nastav prázdný zásobník
2   k := |U| − 1                     a urči počet hran kostry.
3   GenTree(H, T, k)                 Vlastní generování koster

GenTree(H, T, k)
1   if |T| = k then DISPLAY_STACK(T)   Zobraz kostru v zásobníku.
2   else if |H| + |T| ≥ k              Jinak je-li šance na kostru,
3       then h := libovolná hrana z H  vezmi nějakou hranu,
4            if h nevytvoří v T kružnici   a je-li vhodná,
5               then PUSH(T, h)            přidej ji do zásobníku,
6                    GenTree(H − {h}, T, k)   pokračuj v generování
7                    POP(T)                   a pak ji opět vyjmi.
8            GenTree(H − {h}, T, k)     Zkus to i bez hrany h.
```

zdroj: doc. Kolář: Teoretická informatika, FIT ČVUT

Generování všech koster



zdroj: doc. Kolář: Teoretická informatika, FIT ČVUT

Minimální kostra

Je dán souvislý neorientovaný graf G = (V,E,I) s nezáporným ohodnocením hran $w : E \rightarrow R^+$.

Problém minimální kostry grafu G spočívá v nalezení takové jeho kostry $K = (V,E_v,I')$, která splňuje následující podmínku:

$$\sum_{h \in E_v} w(h) \text{ je minimální}$$

Obecný algoritmus hledání minimální kostry

```
GENERIC-MST(G,w)
// G – graf, w – váhy, T - kostra
{
  T := ∅
  while (T netvoří kostru)
  {
    najdi vhodnou hranu [u, v] pro T;
    T := T ∪ {[u, v]};
  }
  return T;
}
```

Algoritmy Borůvky a Jarníka

- algoritmus úspěšně vyřešil v polovině navrhl v polovině 20. let matematik Otakar Borůvka
  - hledal optimální návrh sítě pro rozvod elektřiny na Moravě
- efektivnější metodu řešení navrhl v r. 1930 matematik Vojtěch Jarník
- po r. 1950 v souvislosti s rozvojem výpočetní techniky došlo k oživení zájmu o algoritmické řešení problému minimální kostry, a tak byly algoritmy O. Borůvky i V. Jarníka znovu nezávisle objeveny a publikovány v USA, takže se teprve potom staly všeobecně známými

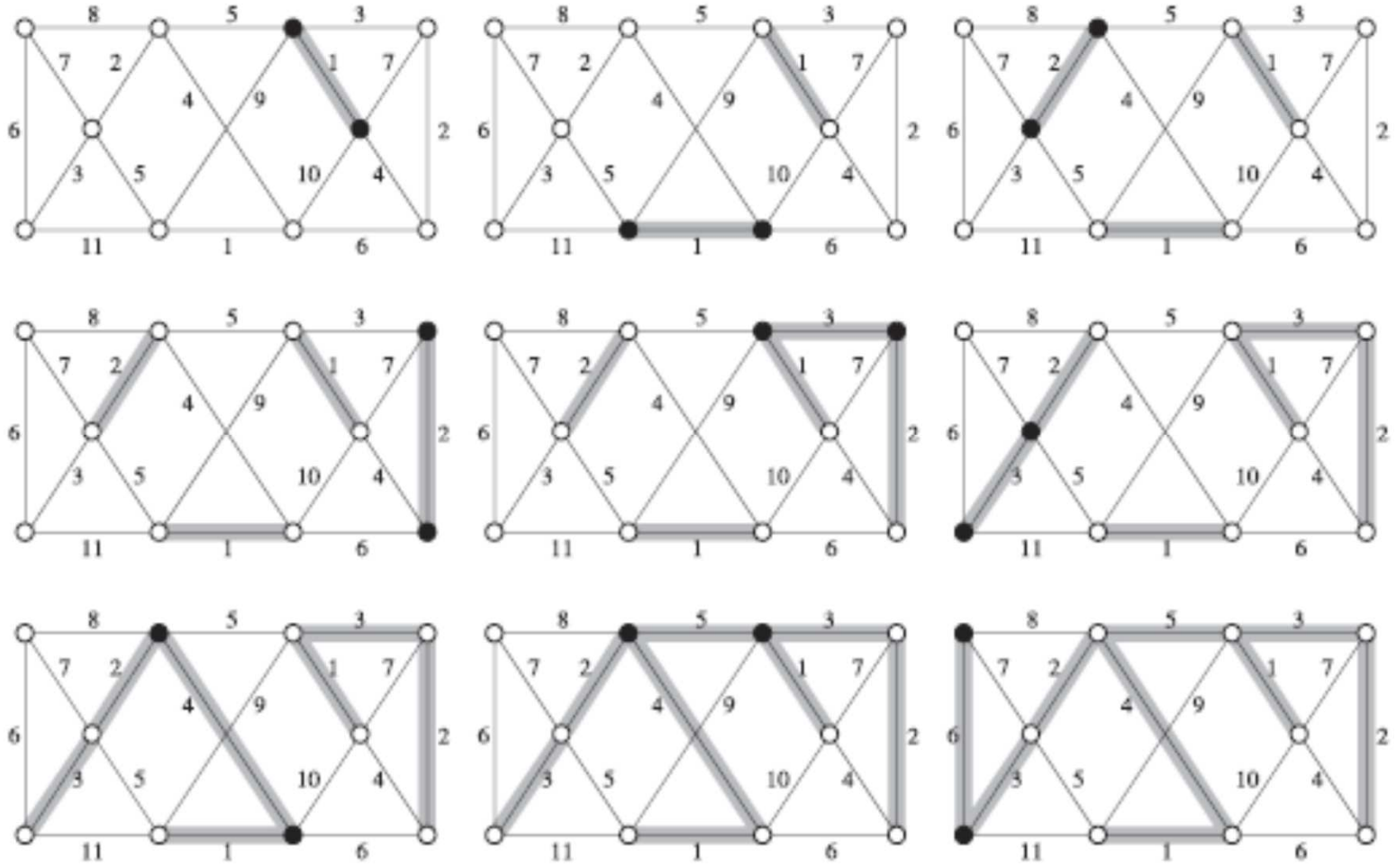
Borůvkův – Kruskalův algoritmus

```
KB-MST(G,w) {
  T := ∅  // Vybraná kostra zatím prázdná
  // vytvoř nnožiny o jednom uzlu (každá množina =
  jeden strom o jednom uzlu)
  for (každý uzel u ∈ V) MAKE-SET(u)
  uspořádej E do nekles. posloupnosti podle váhy w
  for (každou hranu [u, v] ∈ E v pořadí neklesajících
  vah)
  {
    if (FIND-SET(u) != FIND-SET(v)) {
      T := T ∪ {[u, v]} přidej hranu do kostry
      UNION(u, v)  // spoj odpovídající podstromy
    }
  }
return T;
}
```

Borůvkův – Kruskalův algoritmus

- FIND_SET(u) vrátí množinu, do které patří u, tj. podstrom, do kterého patří uzel u
- pokud je FIND_SET(u) != FIND_SET(v), pak uzly u a v patří do různých podstromů, přidáním hrany [u,v] do stromu tedy nemůže vzniknout kružnice, hrana je vhodná
- složitost algoritmu je $O(|E|\log_2|E|)$

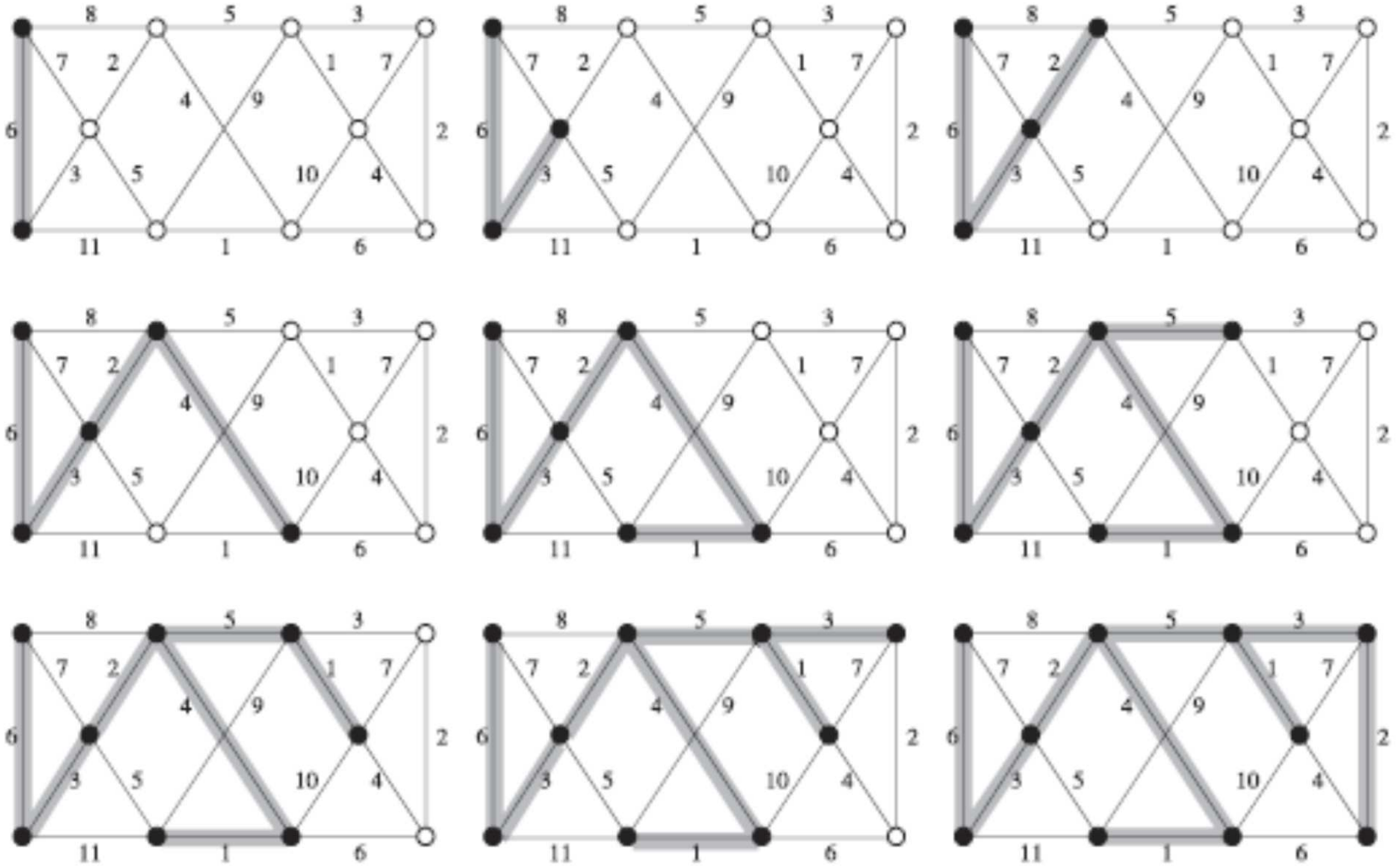
Borůvkův – Kruskalův algoritmus



Jarníkův – Primův algoritmus

```
JP-MST(G, w, r)
1   Q := U                                   Do fronty ulož všechny uzly
2   for každý uzel u ∈ Q                     a urči jim nekonečnou
3       do d[u] := ∞                         vzdálenost od podstromu.
4   d[r] := 0; p[r] := NIL                   Pouze kořen nemá předchůdce.
5   while Q ≠ ∅ do                           Dokud není fronta prázdná,
6       u:=EXTRACT-MIN(Q)                    vyber nejbližší uzel a sousedům z Q
7           for každý uzel v ∈ Adj[u] do     zkus, zda mají blíže k u.
8               if  (v ∈ Q) and (w(u, v) < d[v])
9                   then p[v] := u           Pokud ano, zadej jim předchůdce u
10                       d[v] := w(u, v)     a menší vzdálenost.
```

Jarníkův – Primův algoritmus



Jarníkův - Primův algoritmus

- algoritmus tvoří pouze jeden strom, který rozšiřuje
  - snaží se nalézt uzel, který je neblíže dosud vygenerovanému stromu
- složitost algoritmu je $O(|E|\log_2|V|)$