Použití dalších heuristik

- zkracování cesty při FIND-SET
- UNION podle hodností

Datové struktury ... p[x] - předchůdce uzlu x

hod[x] - hodnost (aprox. výšky)

MAKE-SET(x)

```
p[x] := x
hod[x] := 0
```

UNION(x,y)

```
LINK(FIND-SET(x), FIND-SET(y))
```

LINK(x,y)

```
if hod[x] > hod[y] then p[y] := x
   else p[x] := y
        if hod[x] = hod[y] then hod[y] := hod[y] + 1
```


FIND-SET(x)

```
if x  p[x] then p[x] := FIND-SET(p[x])
return p[x]
```

Složitost m operací MAKE-SET, FIND-SET, UNION, když MAKE-SET je přesně n:

$$O(m \cdot \lg^* n)$$

Minimální kostry

?Jaká kostra je minimální ? Každá má |U|-1 hran?

G = ⟨H,U⟩ souvislý NG s nezáporným ohodnocením hran

w: H → R^+, kostra T = ⟨H_v,U⟩ taková, že

∑w(h) (součet přes h∈ H_v) je minimální

?Jak poznáme minimální kostru?

- prohledáme všechny kostry grafu **???**
- uhádneme kostru a ověříme její minimálnost (**JAK?**)

V: Kostra T grafu G je minimální, pro každou tětivu t je

w(t) ≥ max w(h) pro h∈ K

(K je jediná kružnice v T∪{t})

Hledání minimální kostry

?Praktické možnosti?

- odebírat hrany z původního grafu ???
- najít lib. kostru a postupně ji upravovat ???
- vytvářet minimální kostru přidáváním hran !!!

Generický algoritmus GENERIC-MST(G,w)

```
T := ∅
while T netvoří kostru
   do najdi vhodnou hranu [u,v] pro T
        T := T ∪ {[u,v]}
return T
```

TOHLE je problém

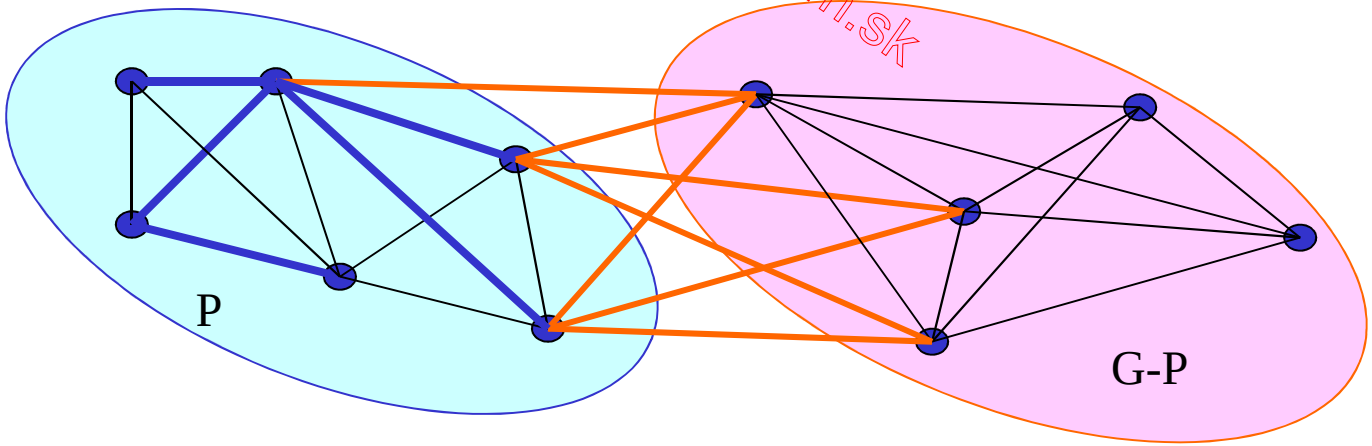
?Jak hledat tu správnou hranu pro přidání do kostry?

V: Nechť P je podstrom vytvořené části minimální kostry grafu G, [p,q] je hrana taková, že

- $p \in P$, $q \notin P$
- $w([p,q]) = \min w(u,v)$

  pro všechny hrany $[u,v]$, $u \in P$, $v \notin P$

Pak lze hranu [p,q] přidat k minimální kostře.



Borůvkův - Kruskalův algoritmus

KB-MST(G, w)

```
1  T:= ∅
2  for každý uzel u ∈ U do MAKE-SET(u)
3  uspořádej H do neklesající posloupnosti podle váhy w
4  for každou hranu [u,v] ∈ H v pořadí neklesajících vah
5       do if FIND-SET(u) ≠ FIND-SET(v)
6                  then T := T {[u,v]}
7                       UNION(u,v)
8  return T
```


O(|H|.lg |H|) řazení, **O(|H|. lg* |U|)** množiny

Jarníkův - Primův algoritmus

JP-MST(G, w, r)

```
1  Q := U
2  for každý uzel u∈Q do d[u] := ∞
3  d[r] := 0; p[r] := nil
4  while Q≠∅ do
5        u := EXTRACT_MIN(Q)
6        for každý uzel v∈Adj[u] do
7              if  (v∈Q) and (w(u,v) < d[v])
8                    then p[v] := u; d[v] := w(u,v)
```


POZOR! SLOŽITÉ

O(|U|) + O(|U| . lg |U|) + O(|H| . lg |U|)

Hladové algoritmy

Požadavky na úlohu

- vlastnost hladového výběru - výběr podproblému a pak jeho řešení
- optimální podstruktura - optimální řešení problému obsahuje optimální řešení podproblému

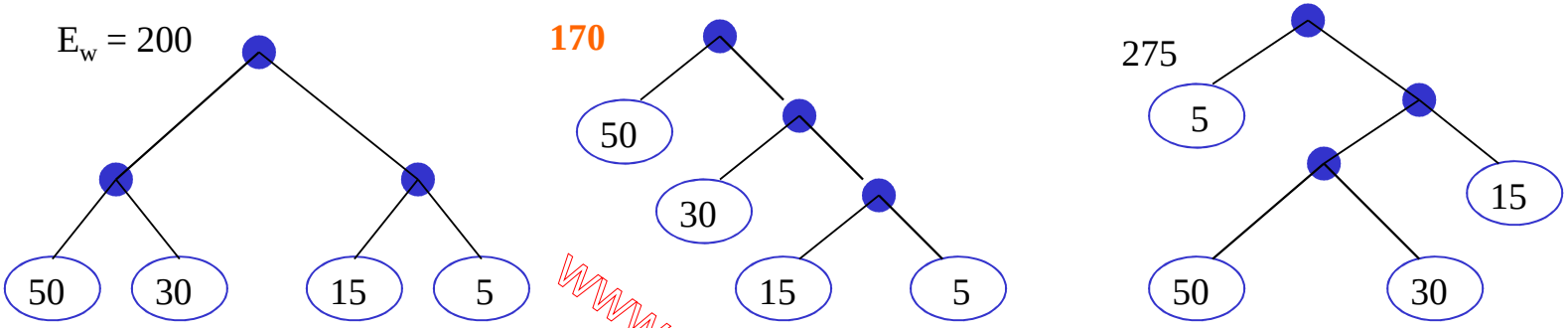
Stromy s minimální w-délkou

Známe tvar stromů s optimální vnější/vnitřní délkou.
? Jak to bude při rozdílných vahách jednotlivých uzlů ?

Pravidelný kořenový strom T_u s ohodnocením uzlů $w_i = w(u_i)$

vnější w-délka $E_w(T_u) = \sum w_i . hl(u_i)$

Huffmanův algoritmus



Huffman(w,n)

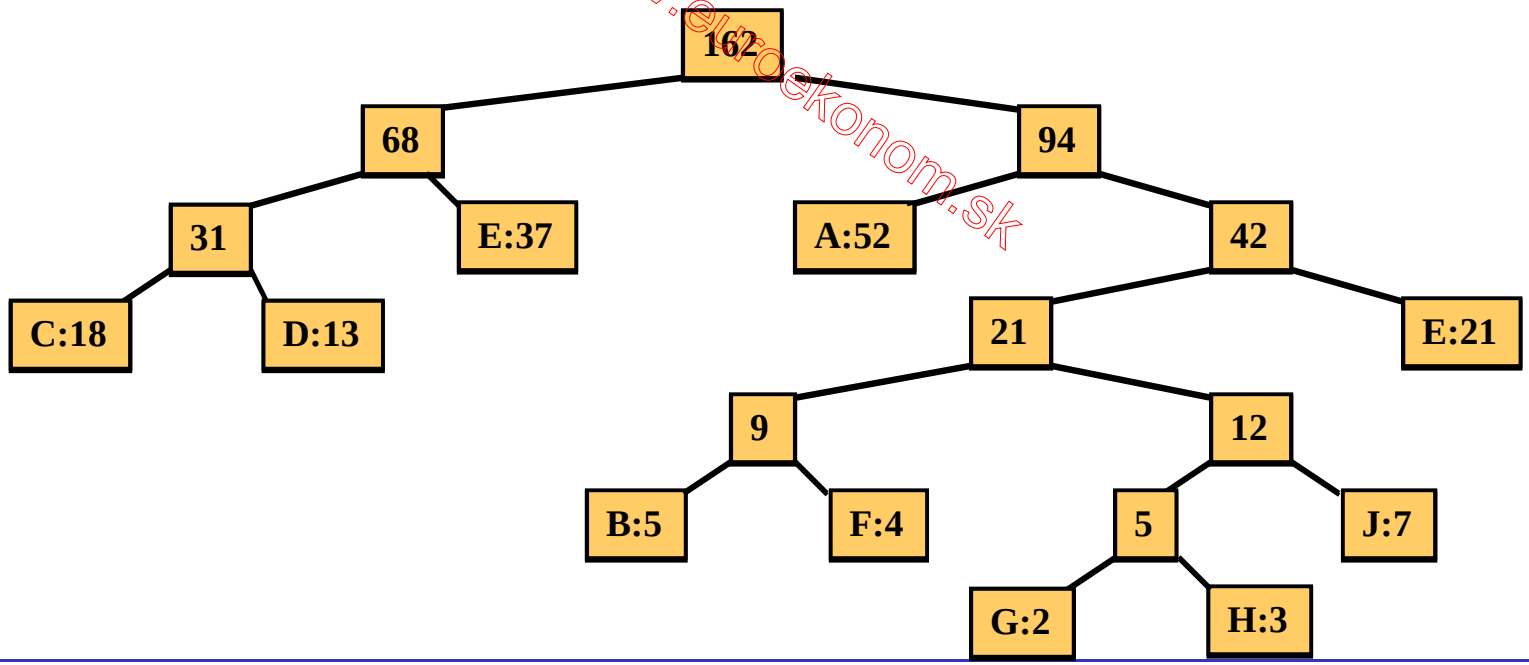
```
1  for i:=1 to n do u:=MakeNode(wi); Insert(u,Q)
2  for i:=1 to n-1
3        do x:=ExtractMin(Q); y:=ExtractMin(Q)
4           z:=MakeNode(w[x]+w[y]); left[z]:=x; right[z]:=y;
5           Insert(z,Q)
6  return ExtractMin(Q)
```

Aplikace - generování optimálního prefixového kódu

prefixový kód - žádné slovo není prefixem jiného slova

Jsou dány znaky a jejich četnosti (absolutní/relativní) v textu

znak	A	B	C	D	E	F	G	H	I	J
četnost	52	5	18	13	21	4	2	3	35	7



Nejkratší cesty z jednoho uzlu

Několik obecných úvah

Uvažujeme nejobecnější případ - ohodnocené OG

- nedostupné uzly - vzdálenost $+\infty$
- spojení se záporným cyklem - vzdálenost $-\infty$
- počítání s nekonečny:

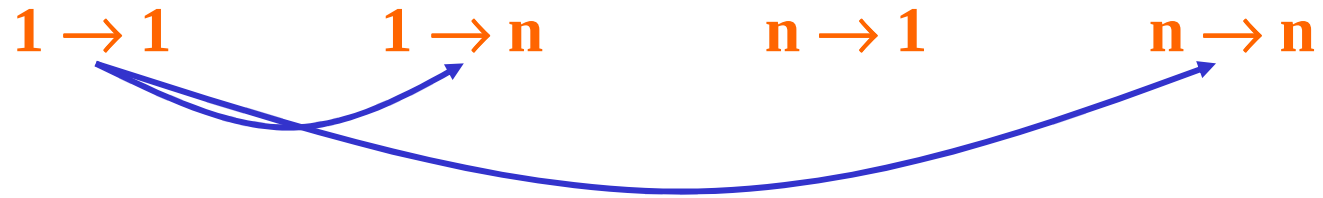
$$a + (-\infty) = (-\infty) + a = -\infty \quad \textbf{pro } a \neq \infty$$
$$a + \infty = \infty + a = \infty \quad \textbf{pro } a \neq -\infty$$

? Které vlastnosti 0 až 4 má taková vzdálenost ?

V: Pro libovolnou hranu $(u,v) \in H$ a uzel $s \in U$ platí

$$d_w(s,v) \leq d_w(s,u) + w(u,v)$$

Varianty úlohy hledání nejkratších cest



Datové struktury

d[u] délka cesty

p[u] předchůdce na min. cestě

Q fronta otevřených uzlů (halda?)

Inicializace InitPaths(G,s,w)

```
1  for každý uzel u  U
2        do d[u]:= ∞ ; p[u]:=nil
3  d[s]:=0
```

Relaxace - (případná) úprava délky nalezené nejkratší cesty

Relax(u,v,w)

```
1  if d[v] > d[u]+w(u,v)
2     then d[v]:= d[u]+w(u,v); p[v]:=u
```

V: Provedeme InitPaths a pak libovolný počet Relax. Potom

- platí $\mathbf{d[u] \geq d_w(s,u)}$
- jakmile d[u] dosáhne hodnoty $d_w(s,u)$, už se nemění
- jakmile se žádné d[u] nemění, máme strom nejkratších cest do všech dosažitelných uzlů z uzlu s

? V jakém pořadí hran a jak dlouho máme provádět relaxaci ?

Dijkstrův algoritmus

Základní předpoklad $w : H \rightarrow R^+$

“Upravený“ algoritmus prohledávání do šířky

Dijkstra(G,s,w)


```
1  InitPaths(G,s)
2  S:=∅; InitQueue(Q)
3  for každý uzel u∈U do Enqueue(Q,u)          O(|U|)
4  while not EmptyQueue(Q)
5        do  u:=ExtractMin(Q); S:=S∪{u}        O(|U| . lg |U|)
6            for uzel v∈Adj[u]
7                do Relax(u,v,w)                O(|H| . lg |U|)
```

Možné ještě **O(|U| . lg |U| + |H|)** nebo **O(|U|**2)**

Bellmanův-Fordův algoritmus

? Co dělat v případě záporně orientovaných hran ?

Bellman-Ford(G,s,w)

```
1  InitPaths(G,s)
2  for i:=1 to |U|-1 do
3        for každou hranu (u,v)∈H do Relax(u,v,w)
4  for každou hranu (u,v)∈H
5     do if d[v] > d[u] + w(u,v) then return false
6  return true
```

Složitost **O(|U| . |H|)**

? Proč má nyní Relax konstantní časovou složitost ?

? Nelze B-F algoritmus nějak upravit / zrychlit ?

Co když **zavedeme frontu uzlů s úspěšným Relax** a bereme jen hrany vycházející z těchto uzlů?

- **ukončení** - při vyprázdnění fronty
- **problém** - co když se fronta nevyprázdní?
- v nejhorším případě zase **O(|U| . |H|)**

Nejkratší cesty pro acyklické grafy

```
1  Topologicky uspořádáme uzly grafu G
2  InitPaths(G,s)
3  for každý uzel u v pořadí podle topologického uspořádání
4     do for každé v∈ Adj[u]
5           do Relax(u,v,w)
```

?? Složitost ??