STROMY A KOSTRY

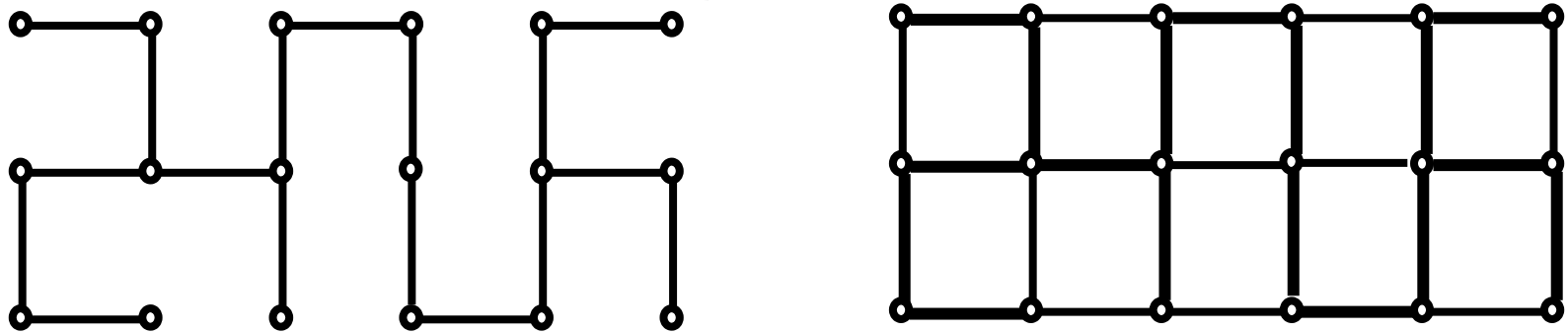
Stromy a kostry

Seznámíme se s následujícími pojmy:

- **kostra grafu, cyklomatické číslo grafu, hodnost grafu**
- (kořenový strom, hloubka stromu, kořenová kostra orientovaného grafu, uspořádaný strom, pravidelný strom stupně r, úplný pravidelný strom), **vnější/vnitřní délka stromu,** binární strom, průchody preorder/inorder/postorder

Skripta odst. 3.2, str. 49 – 58

Připomeňme si nejprve, co je to **strom** ... a co je to **kostra** grafu ...

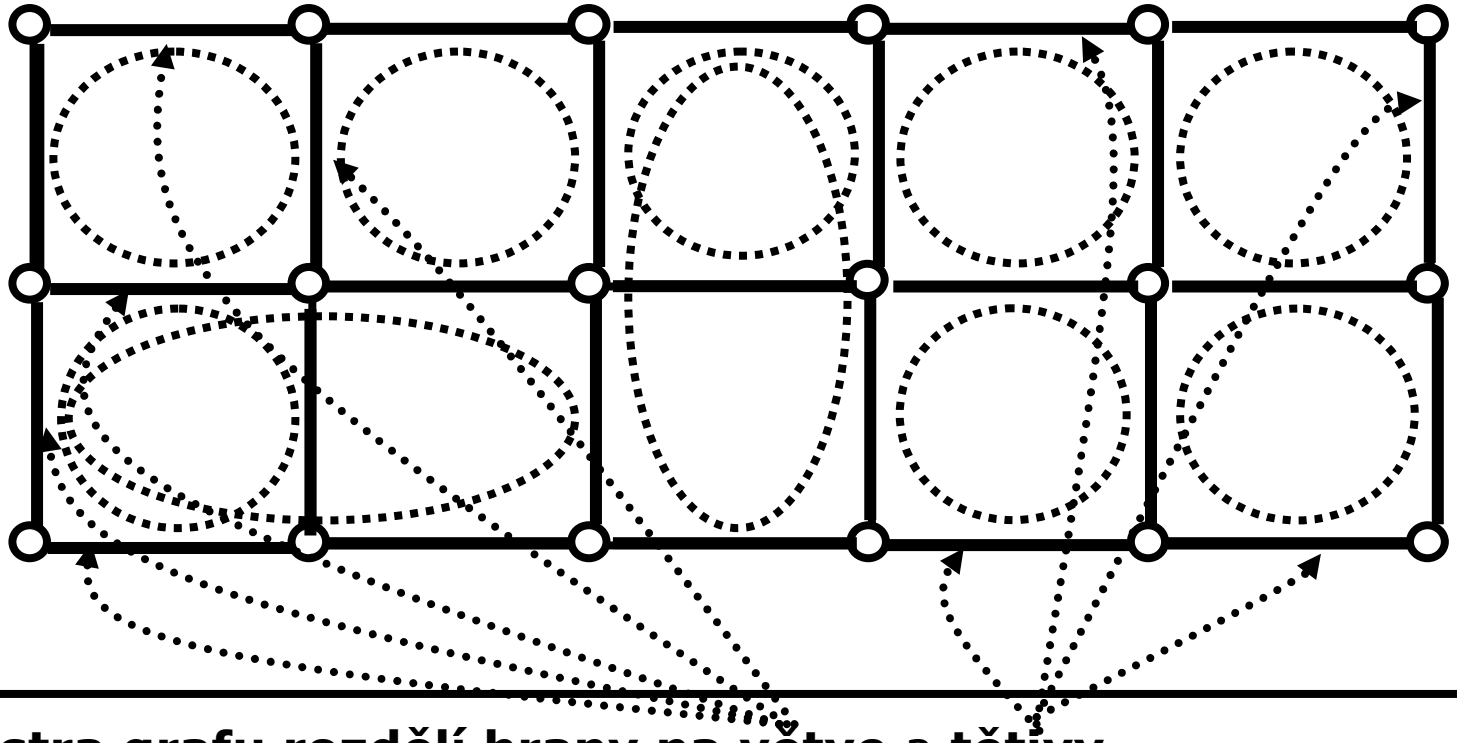


Co lze říci o stromech?

Nechť $G = \langle H,U \rangle$ je prostý graf. Potom

- **a) G je strom**
- **b) $\forall u,v \in U$ existuje právě jedna cesta $u - v$**
- **c) G je souvislý, ale G - {h} není pro libovolnou $h \in H$**
- **d) G je souvislý a platí $|H| = |U| - 1$**
- **e) G neobsahuje kružnice a platí $|H| = |U| - 1$**
- **f) G neobsahuje kružnice, ale $G \cup \{h\}$ ano**

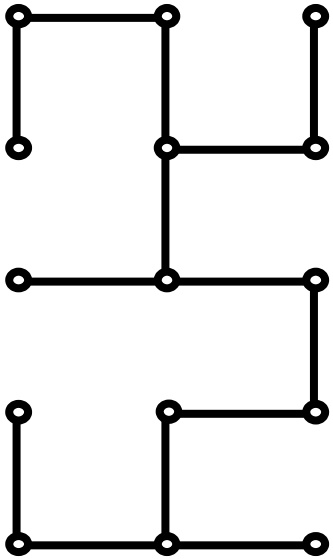
jsou ekvivalentní tvrzení.



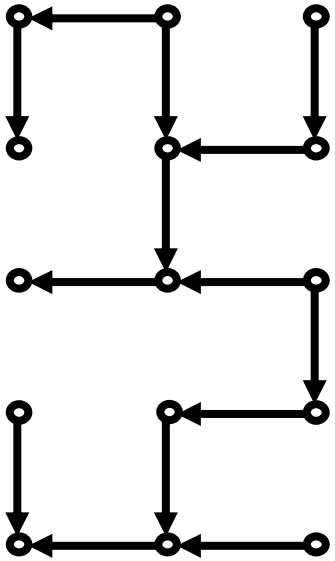
Kostra grafu rozdělí hrany na větve a **tětivy**
Fundamentální soustava kružnic odpovídající dané kostře

Cyklomatické číslo (počet nezávislých kružnic = počet tětiv)
$\mu(G) = |H|-|U|+p$ (**p** je počet komponent)
Hodnost grafu (počet hran kostry nebo lesa grafu)
$h(G) = |U|-p$

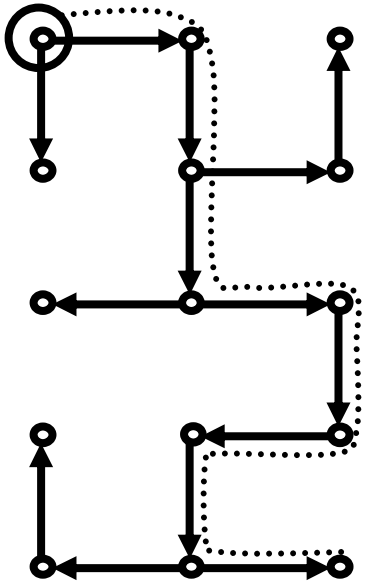
Orientované a speciální stromy



strom

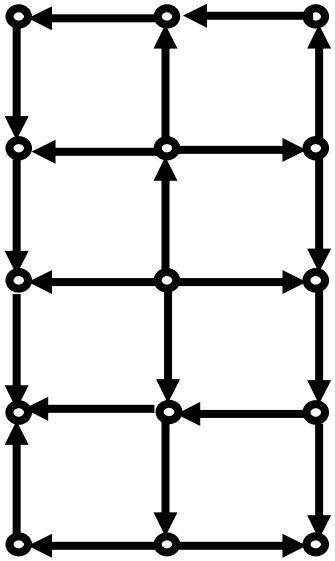


orientovaný strom

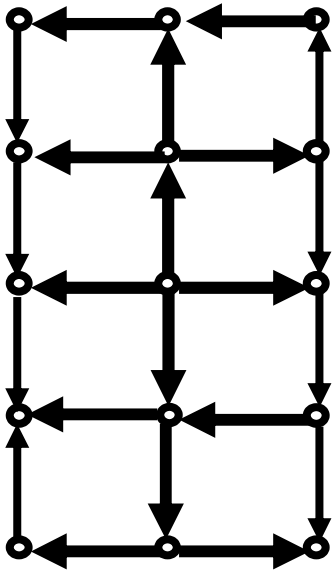


kořenový strom

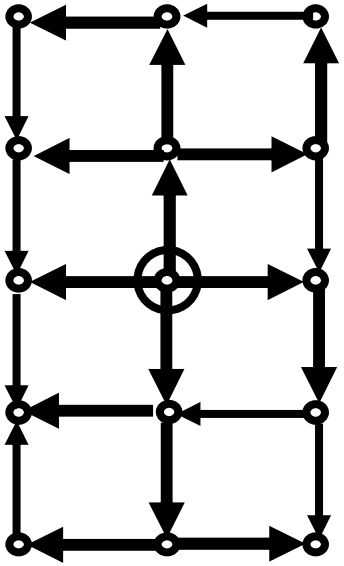
Kořenový strom T_u (s kořenem **u**) - orientovaný strom, kde každá cesta C(**u**,v) je orientovanou cestou.



orient. graf



jeho kostra

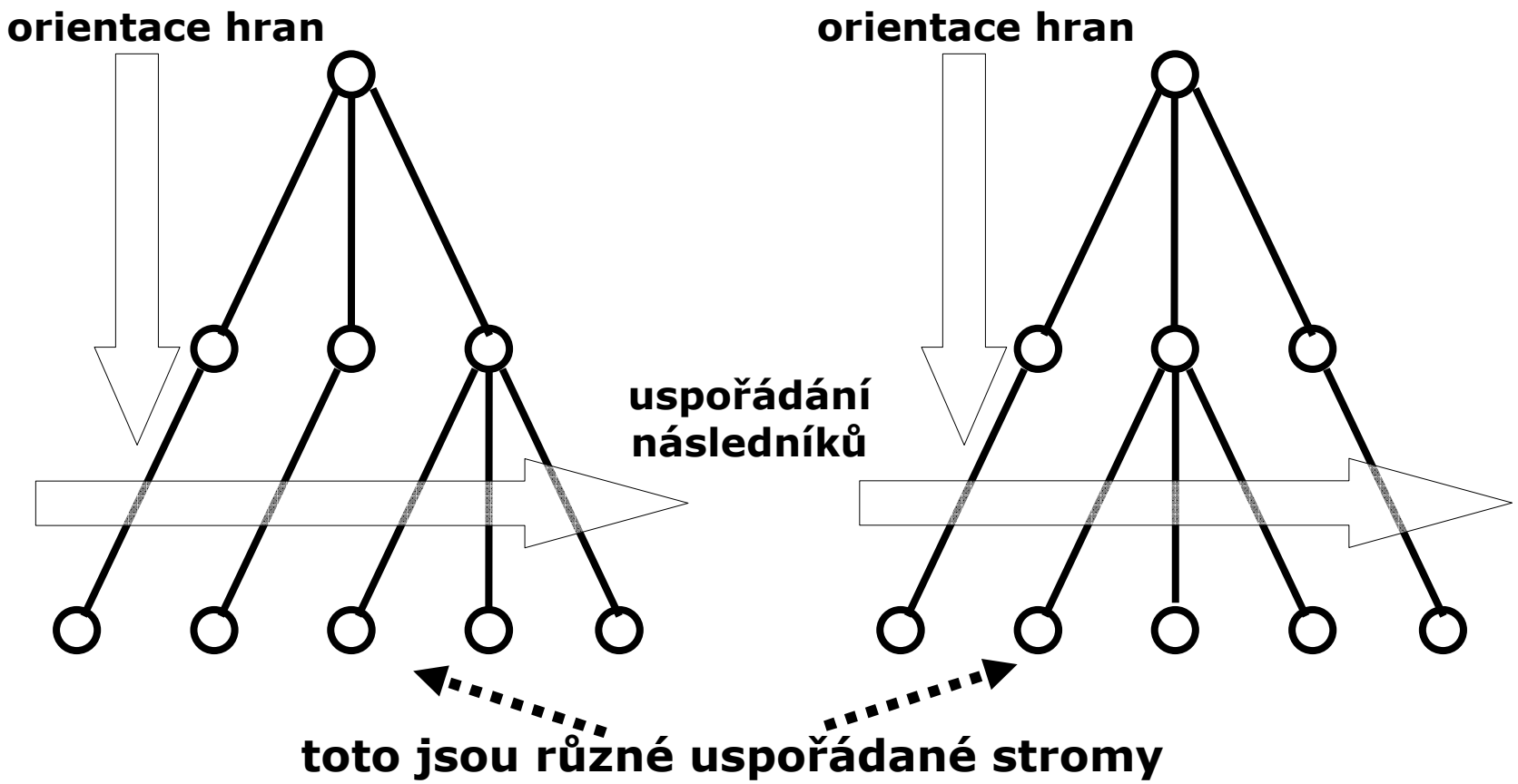


kořenová kostra

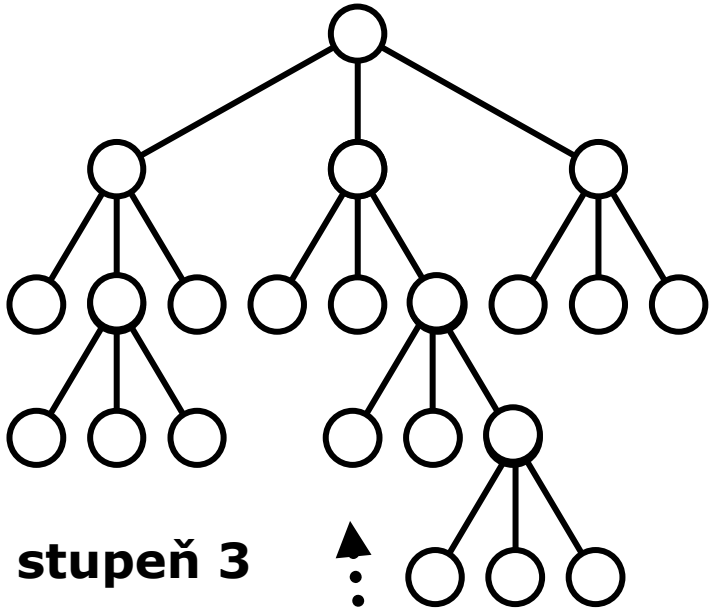
Kořenová kostra - kostra, která je kořenovým stromem.

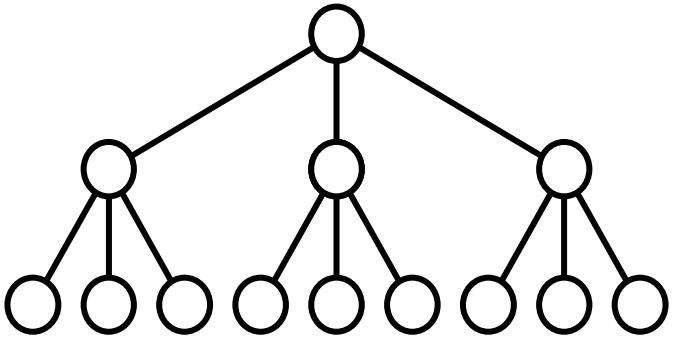
Hloubka hl(x) uzlu x v kořenovém stromu = vzdálenost od kořene

Hloubka kořenového stromu = max hl(x) $x \in U$



Uspořádaný (kořenový) strom – následníci (podstromy každého uzlu jsou pevným způsobem uspořádáni (první, druhý, ...)





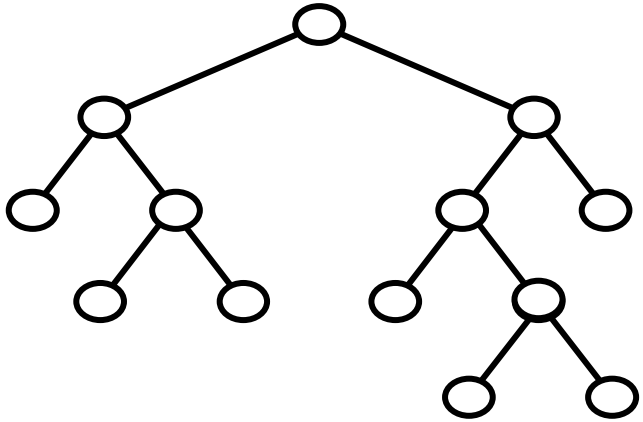
stupeň 3
hloubka 4

stupeň 3 hloubka 2

Pravidelný strom stupně r (≥1) … $\delta^+(u) = 0$ nebo r

Úplný pravidelný strom stupně r hloubky k -

všechny listy jsou v hloubce k od kořene



Statistika:

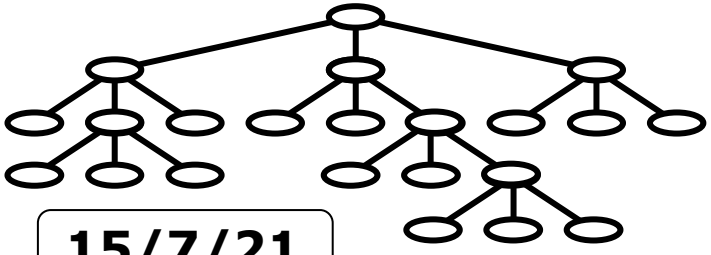
- 7 listů (k)
- 6 vnitřních uzlů (m)
- 12 hran (h)

Zjištění:

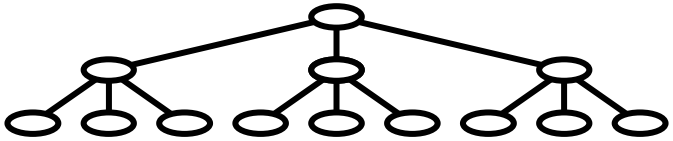
Pro každé n∈N existuje pravidelný strom stupně 2 s n listy

(má **n-1** vnitřních uzlů a **2(n-1)** hran: **n / n-1 / 2(n-1)**)

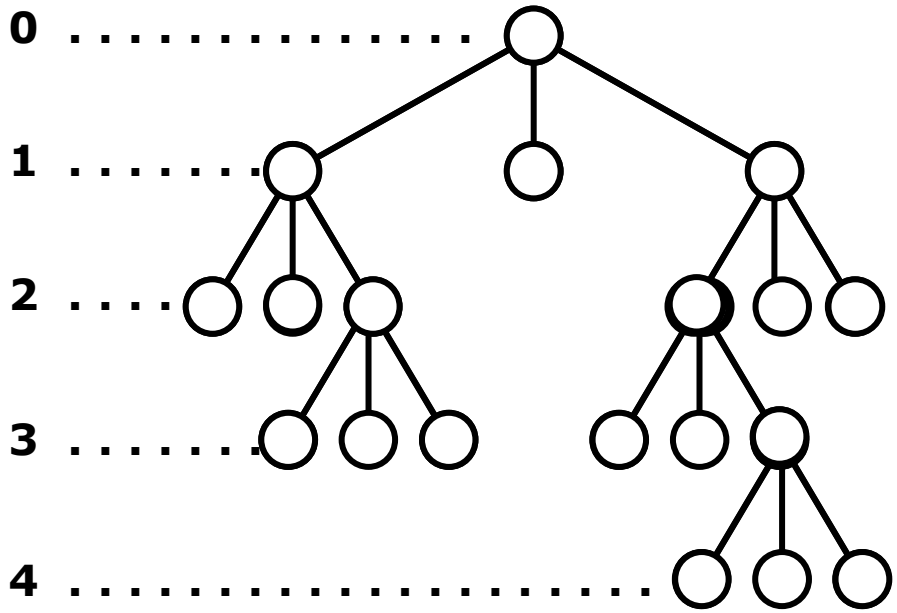
?Jak je to pro jiné stupně? **m.(r-1)+1 / m / m.r**



15/7/21



9/4/12



Statistika (r=2):

- **E(T)=2.4+3.3+2.2= 21**
- **I(T)=1.3+2.2+2.1=9**

Statistika (r=3):

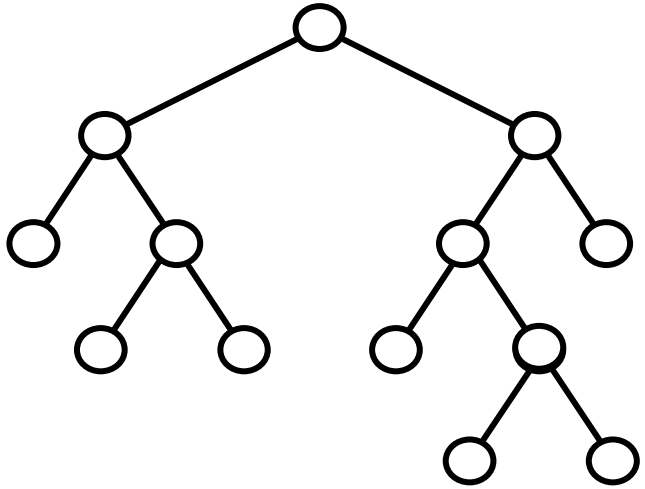
- **E(T)=3.4+5.3+4.2+1.1= 36**
- **I(T)=1.3+2.2+2.1=9**

vnější délka $E(T_u) = \sum hl(v)$ sčítá se přes listy v

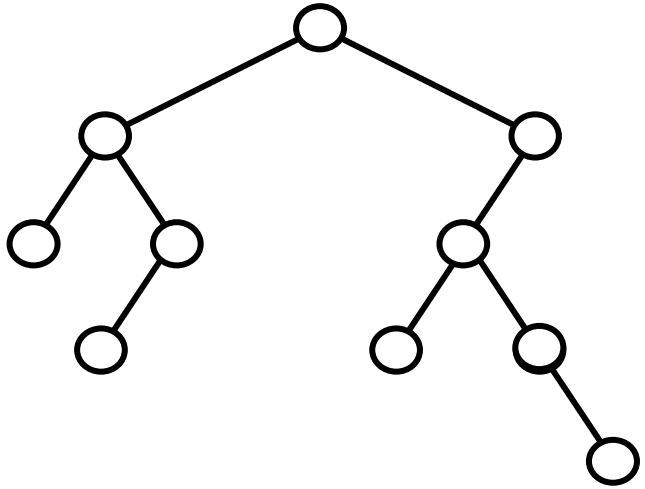
vnitřní délka $I(T_u) = \sum hl(v)$ sčítá se přes vnitřní uzly v

E = I . (r-1) + r . m, kde m je počet vnitřních uzlů

r=2: 21=9.1+2.6 r=3: 36=9.2+3.6



pravidelný strom stupně 2



binární strom

Binární strom

- žádný uzel (prázdný)
- kořen, levý podstrom, pravý podstrom

Průchody binárním stromem

- **preorder**: kořen, LS, PS
- **inorder**: LS, kořen, PS
- **postorder**: LS, PS, kořen

DS 26.3.08

Kontrolní otázky

6.1 Určete, jakou podmínku musí splňovat uzel u souvislého orientovaného grafu, aby existovala kořenová kostra tohoto grafu s kořenem u.

6.2 Pro obecný počet uzlů n ($n \geq 3$) určete, jak vypadá strom s n uzly, který má maximální (resp. minimální) počet listů.

6.3 Kolik různých kružnic vznikne, přidáme-li do kostry grafu dvě tětivy?

6.4 Zdůvodněte, proč se při libovolném z průchodů preorder, inorder, postorder binárního stromu navštíví jeho listy ve stejném relativním pořadí.

6.5 Předpokládejme, že uzly pravidelného stromu stupně 2 jsou nějak očíslovány pořadovými čísly 1 až n. Ukažte, že strukturu tohoto pravidelného stromu lze jednoznačně rekonstruovat, pokud jsou k dispozici alespoň dvě z posloupností získaných průchodem preorder, inorder a postorder tohoto stromu.

6.6 Neorientovaný strom T má k listů a součet stupňů jeho vnitřních uzlů je s. Určete,

- co z toho bezprostředně plyne pro vlastnosti čísel k a s
- počet vnitřních uzlů stromu T v závislosti na k a s.

6.7 Šířkou kořenového stromu se nazývá maximum počtu uzlů nacházejících se ve stejné hloubce. Navrhněte algoritmus pro určení šířky zadaného kořenového stromu.

Minimální kostry

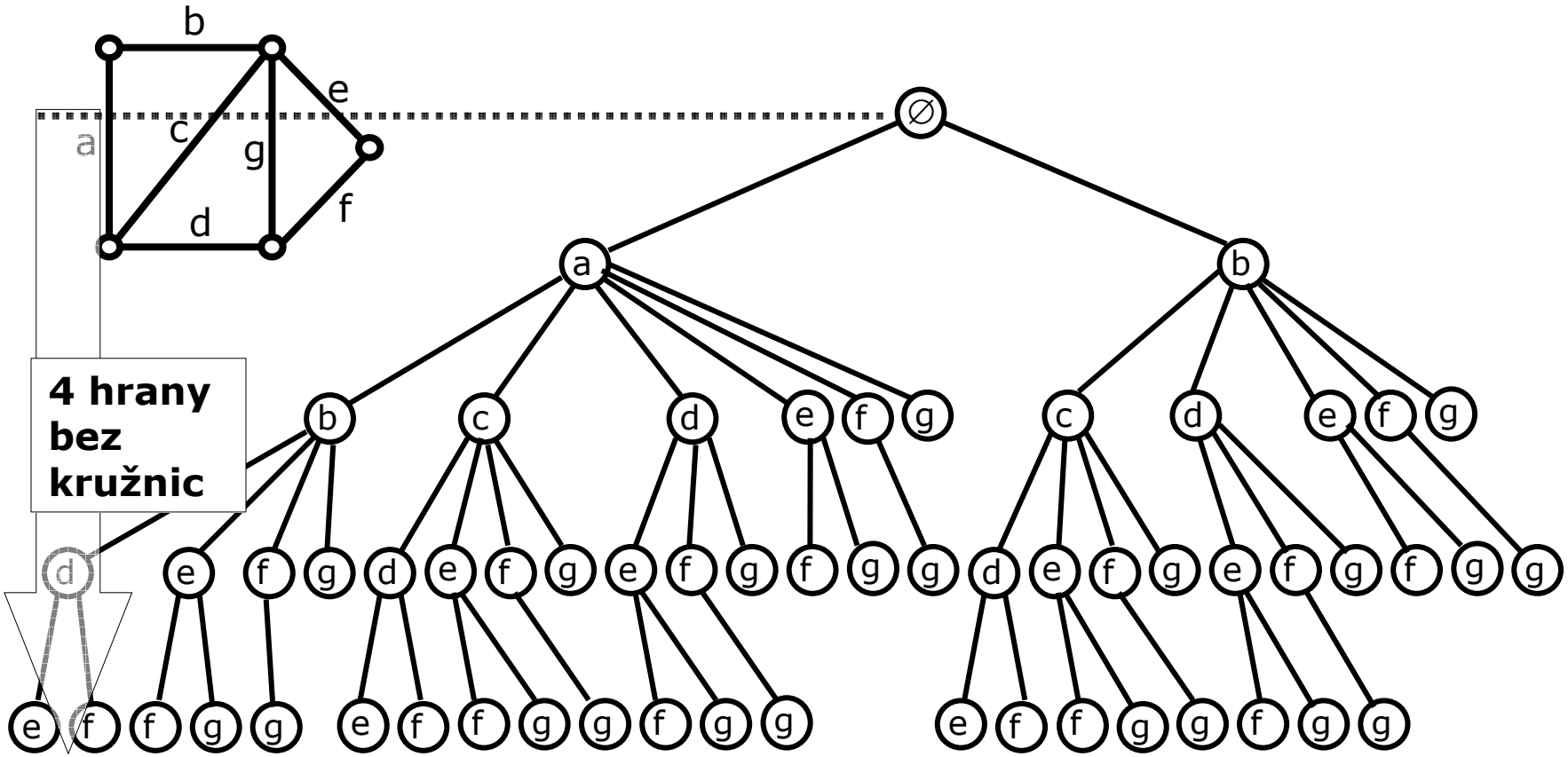
Seznámíme se s následujícími pojmy:

- **generování všech koster grafu, množinový rozklad**
- **minimální kostra grafu, Borůvkův-Kruskalův algoritmus, Jarníkův-Primův algoritmus**
- **hladové algoritmy, Huffmanovo kódování**

Skripta kap. 5, str. 91 - 109

?Jak bychom generovali všechny kostry grafu?

Pomocí stromu všech koster - **efektivní test** vzniku kružnic!



Rekapitulace:

Přidáváme hrany a testujeme vznik kružnice.

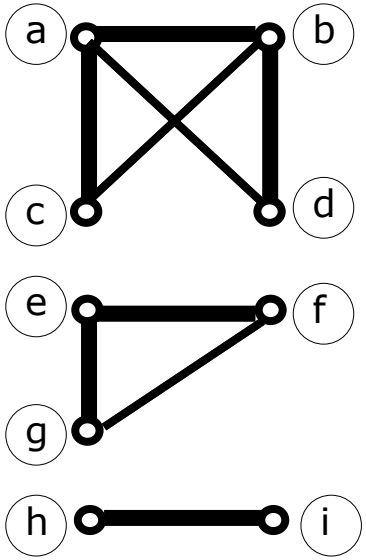
K tomu se hodí ...

Množinový rozklad - dynamický soubor podmnožin dané množiny s operacemi

- **MAKE-SET(x)** vytvoří jednoprvkovou podmnožinu
- **UNION(x,y)** sjednocení podmnožin s prvky **x** a **y**
- **FIND(x)** určí reprezentanta podmnožiny s prvkem **x**

Další použití:

```
void KOMP(Graph G) {      // určení komponent NG
   for (Node u in U(G))  MAKE-SET(u);
   for (Edge (u,v)in H)
     if (FIND(u) != FIND(v))  UNION(u,v);
}
```



	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}
(a,b)	{a,b}		{c}	{d}	{e}	{f}	{g}	{h}	{i}
(e,f)	{a,b}		{c}	{d}	{e,f}		{g}	{h}	{i}
(d,b)	{a,b,d}		{c}		{e,f}		{g}	{h}	{i}
(h,i)	{a,b,d}		{c}		{e,f}		{g}	{h,i}	
(a,c)	{a,b,d,c}				{e,f}		{g}	{h,i}	
(e,g)	{a,b,d,c}				{e,f,g}			{h,i}	
(b,c)	{a,b,d,c}				{e,f,g}			{h,i}	
(f,g)	{a,b,d,c}				{e,f,g}			{h,i}	
(a,d)	{a,b,d,c}				{e,f,g}			{h,i}	

? Složitost : $\Omega(|U| + |H|)$? Výhoda : dynamika!

Jak implementujeme množinový rozklad ?

Pomocí seznamů s odkazy na reprezentanta:

- MAKE-SET, FIND ... **O(1)**
- UNION ... **O(min(m,n))** (pro vyvažování)
  (vyžaduje uchovávat další pomocné údaje)

Agregovaná složitost **m** operací, z toho **n** x MAKE-SET:

O(m + n.lg n)

<u>Při použití na určení kostry provedeme:</u>

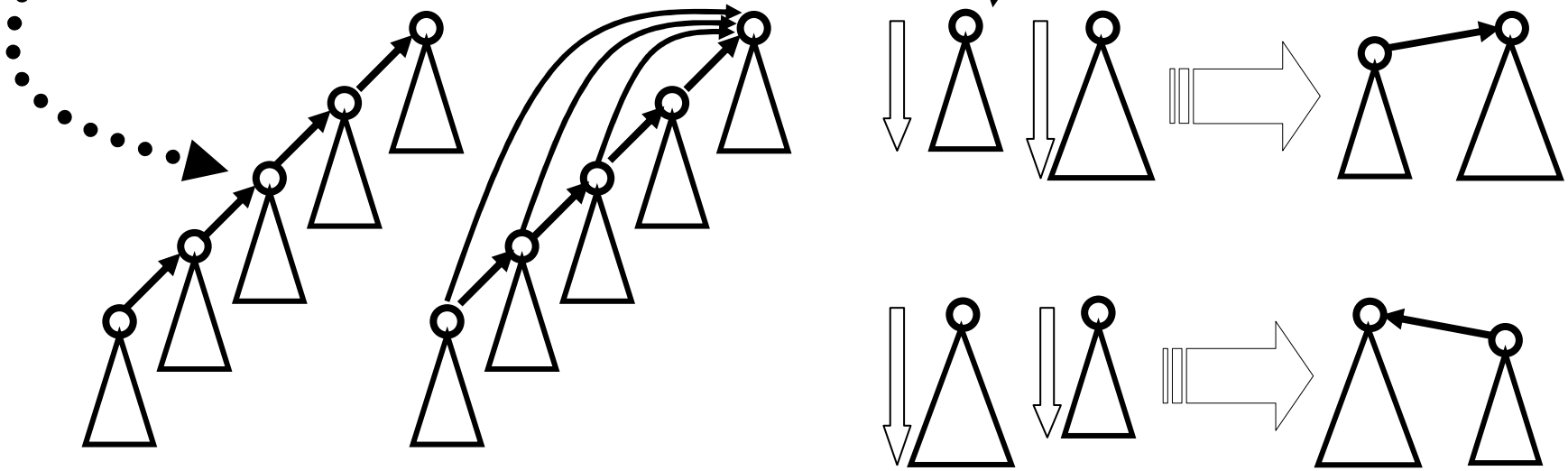
|U|-krát MAKE-SET max. 2.|H|-krát FIND
(|U|-1)-krát UNION ⇒ n = |U|, m ≤ 2.(|U|+|H|) ⇒

O(2|H| + 2|U| + |U|.lg |U|) = O(|H| + |U|.lg |U|)

?Dokážeme to ještě lépe? ANO !

Další heuristiky pro množinový rozklad

- zkracování cesty při FIND
- UNION podle hodností



Datové struktury: **p[x]** - předchůdce uzlu x
hod[x] - hodnost (aproximace výšky)

```
void MAKE-SET (int x) {
  p[x] = x;
  hod[x] = 0;
}
```

```
void UNION (int x, int y) {
  LINK(FIND(x), FIND(y));
}
```

```
void LINK(int x,int y) {
  if (hod[x] > hod[y])  p[y] = x;
  else { p[x] = y;
    if (hod[x] == hod[y])  hod[y]++;
} }
```

```
void FIND(int x) {
  if (x != p[x])  p[x] = FIND(p[x]);
  return p[x];  }
```

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

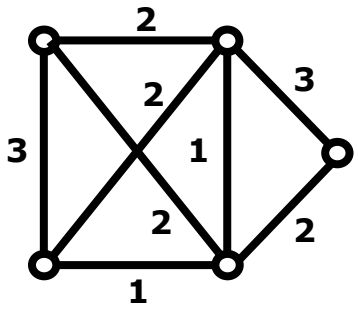
O(m . lg* n) neboli **O((|U|+|H|) . lg* |U|)**

… a teď už konečně ty minimální kostry …

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

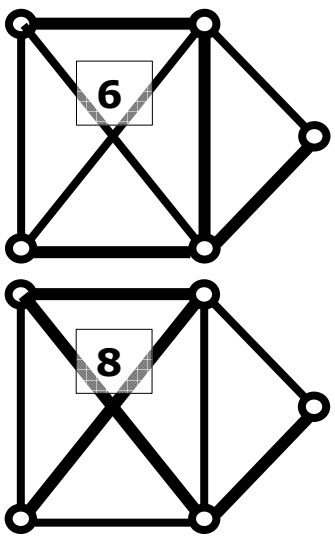
- G = ⟨H,U⟩ - souvislý NG
- s nezáporným ohodnocením hran w: H → R^+,
- kostra T = ⟨H_v,U⟩ taková, že

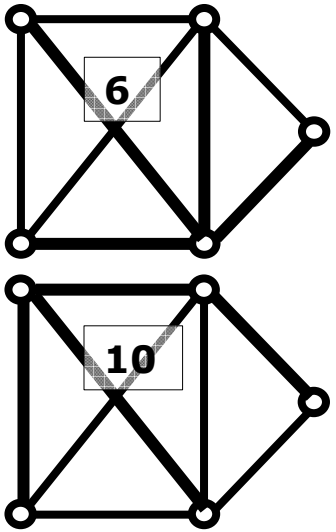
∑w(h) (součet přes h∈H_v) je minimální

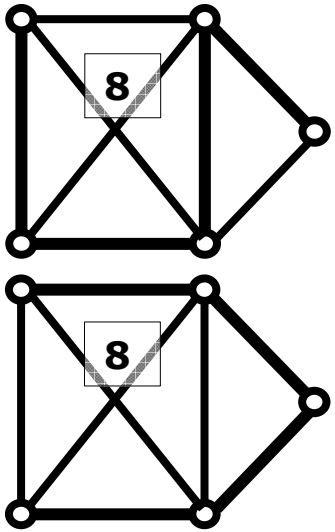


kolik má koster ?

32







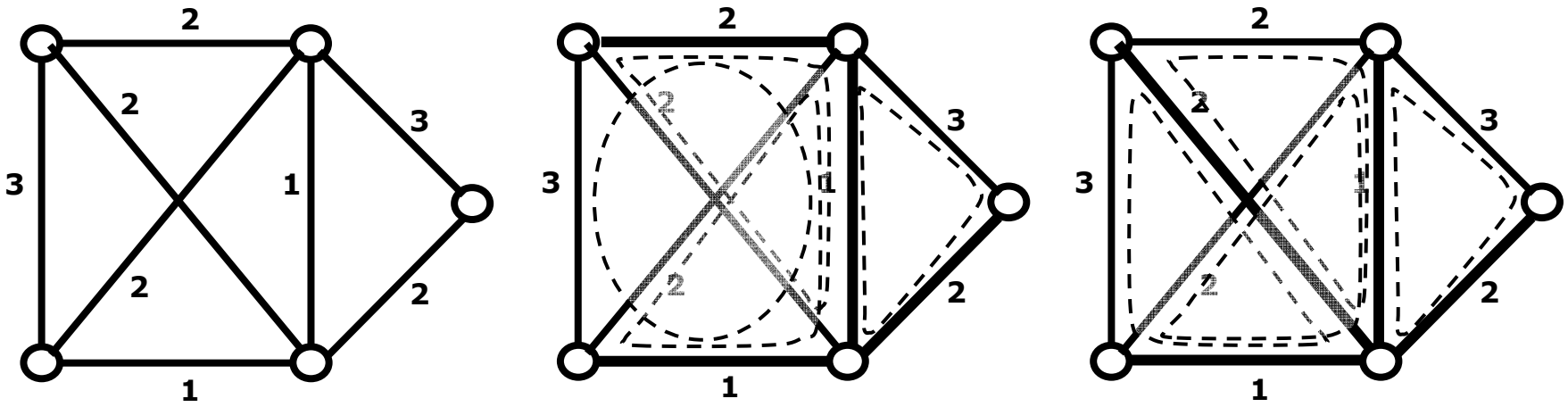
? Jak poznáme, že kostra je minimální ?

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

V: Kostra T grafu G je minimální $\Leftrightarrow$ pro každou tětivu **t** je

$$\mathbf{w(t)} \geq \mathbf{max\ w(h)} \quad \text{pro } h \in K$$

(kde K je jediná kružnice v $T \cup \{\mathbf{t}\}$)



Hledání minimální kostry

? Praktické možnosti ?

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

Generický algoritmus pro minimální kostru

```
void GENERIC-MST(Graph G,Weights w) {
  T = ∅;
  while ("T netvoří kostru") {
    "najdi vhodnou hranu [u,v] pro T"
    T = T ∪ [u,v]
   }
   return T;
}
```

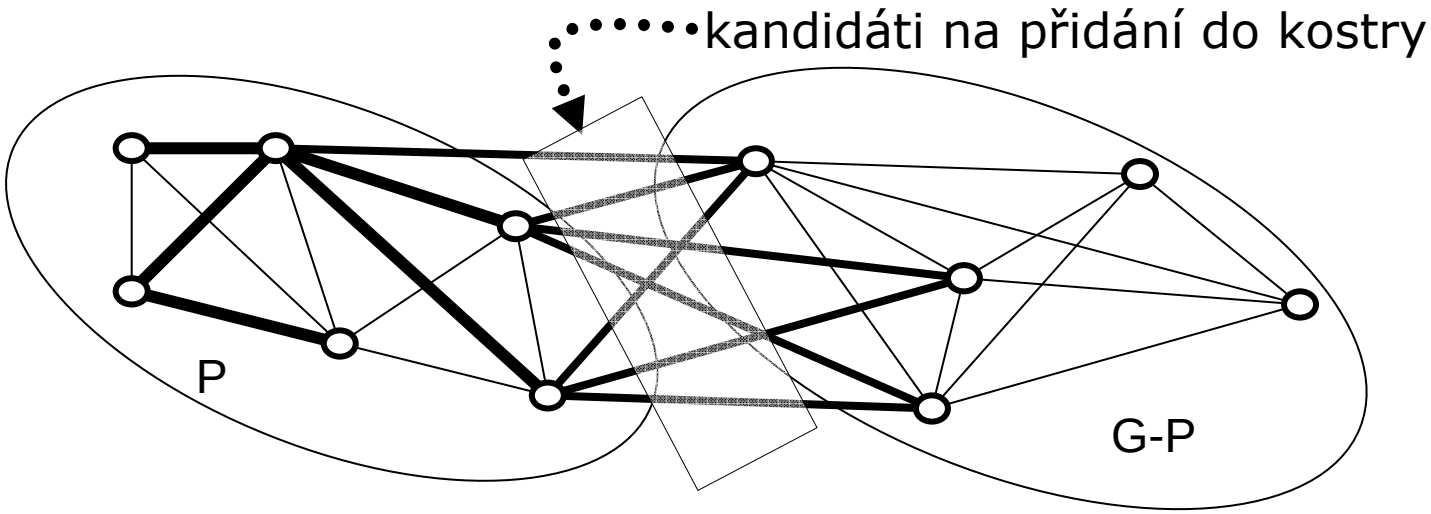
Jak hledat hranu pro přidání do minimální kostry

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

- p∈P, q∉P

- w([p,q]) = min w([u,v]) pro vš. hrany [u,v], u∈P, v∉P .

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



Borůvkův - Kruskalův algoritmus

```
   void BK-MST(Graph G, Weights w) {
1     T = ∅;
2     for (Node u in U(G)) MAKE-SET(u);
3     "seřaď H(G) podle neklesajících vah w(h)"
4     for (Edge [u,v] in H(G)) {  // v daném pořadí
5       if (FIND(u) != FIND(v)) {
6         T = T ∪ [u,v];
7         UNION(u,v);
8     } }
9     return T;
   }
```

O(|H|.lg |H|) řazení; **O(|H|. lg* |U|)** množ. rozklad

Jarníkův - Primův algoritmus

```
void JP-MST(Graph G, Weights w, Node r) {
1     Q = U;
2     for (Node u in Q) d[u] = ∞;
3     d[r] = 0; p[r] = null;
4     while ( Q != ∅ ) {
5       u = Q.ExtMin();
6       for (Node v in Adj[u]) {
7         if ((v∈Q) && (w(u,v) < d[v])) {
8           p[v] = u; d[v] = w(u,v);
9     } } }
}
```

POZOR!
Nemají konstantní složitost !!!

$O(|U|) + O(|U| \cdot \lg |U|) + O(|H| \cdot \lg |U|)$

Kontrolní otázky

6.8 Nechť G je souvislý neorientovaný graf s nezáporným ohodnocením hran w, nechť H_1 je nějaká podmnožina jeho hran, která neobsahuje kružnice. Navrhněte algoritmus nalezení kostry s minimálním ohodnocením hran takové, že obsahuje všechny hrany z množiny H_1.

6.9 Zjistěte, zda jsou následující tvrzení pravdivá či nikoliv. Pravdivá tvrzení dokažte, nepravdivá vyvraťte co nejjednodušším protipříkladem:

- Jsou-li ohodnocení všech hran v souvislém neorientovaném grafu navzájem různá, je jeho minimální kostra určena jednoznačně.
- Jsou-li ohodnocení všech hran v souvislém neorientovaném grafu navzájem různá, pak mají každé dvě jeho různé kostry různá ohodnocení.
- Nechť je ohodnocení hrany h v souvislém neorientovaném grafu menší než ohodnocení každé jiné hrany. Pak je hrana h obsažena v každé jeho minimální kostře.

6.10 Graf G vzniknul tak, že jsme do stromu přidali hranu spojující nějakou dvojici jeho nesousedních uzlů. Čím je určen počet různých koster takto vytvořeného grafu G?

6.11 Ukažte na příkladu, že popsanými algoritmy hledání minimální kostry neorientovaného grafu nelze obecně získat minimální kořenovou kostru orientovaného grafu. V čem spočívá hlavní obtíž?

Hladové algoritmy

Požadavky na úlohu

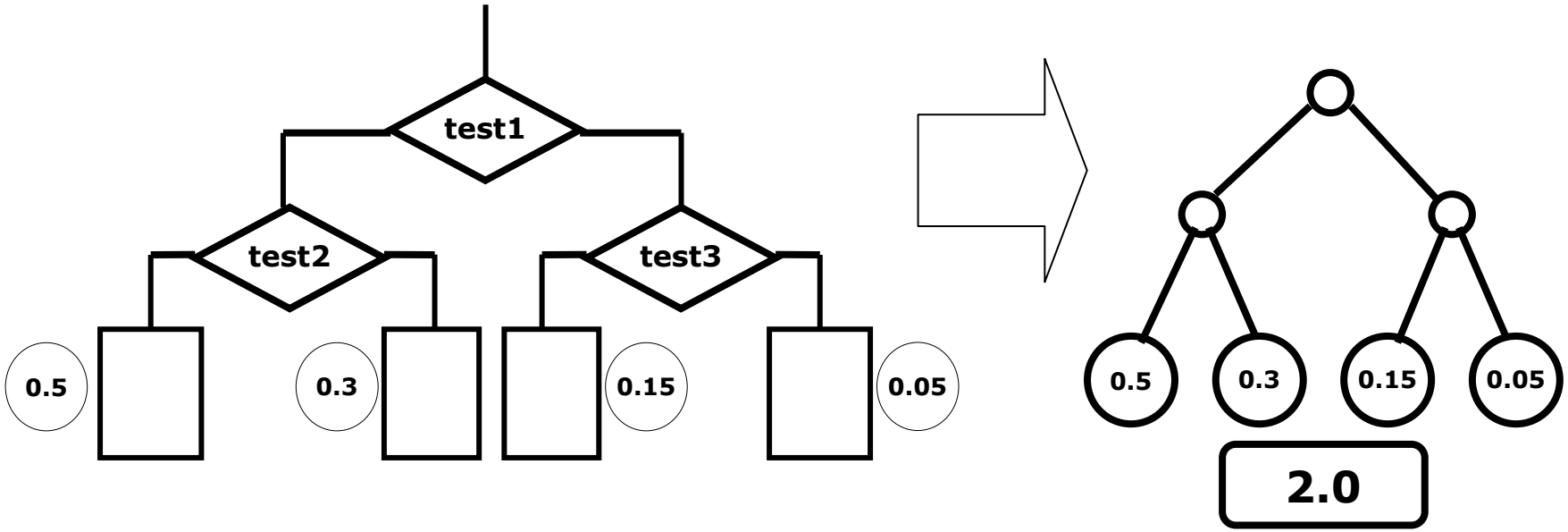
- vlastnost hladového výběru - výběr podproblému a pak jeho řešení
- optimální podstruktura - optimální řešení problému obsahuje optimální řešení podproblému

Postup **shora – dolů** (dynamické programování zdola – nahoru)

Příklad - Jarník-Prim algoritmus:

- **výběr "nejlepšího" z n uzlů mimo dílčí podstrom pro přidání**
- **úprava kritéria pro jeho sousedy**
- **pak totéž pro (n-1) uzlů, atd.**

Příklad – návrh optimální větvící struktury programu



Známe relativní četnost průchodu jednotlivými větvemi

? Jaký bude průměrný počet testů potřebných k větvení?

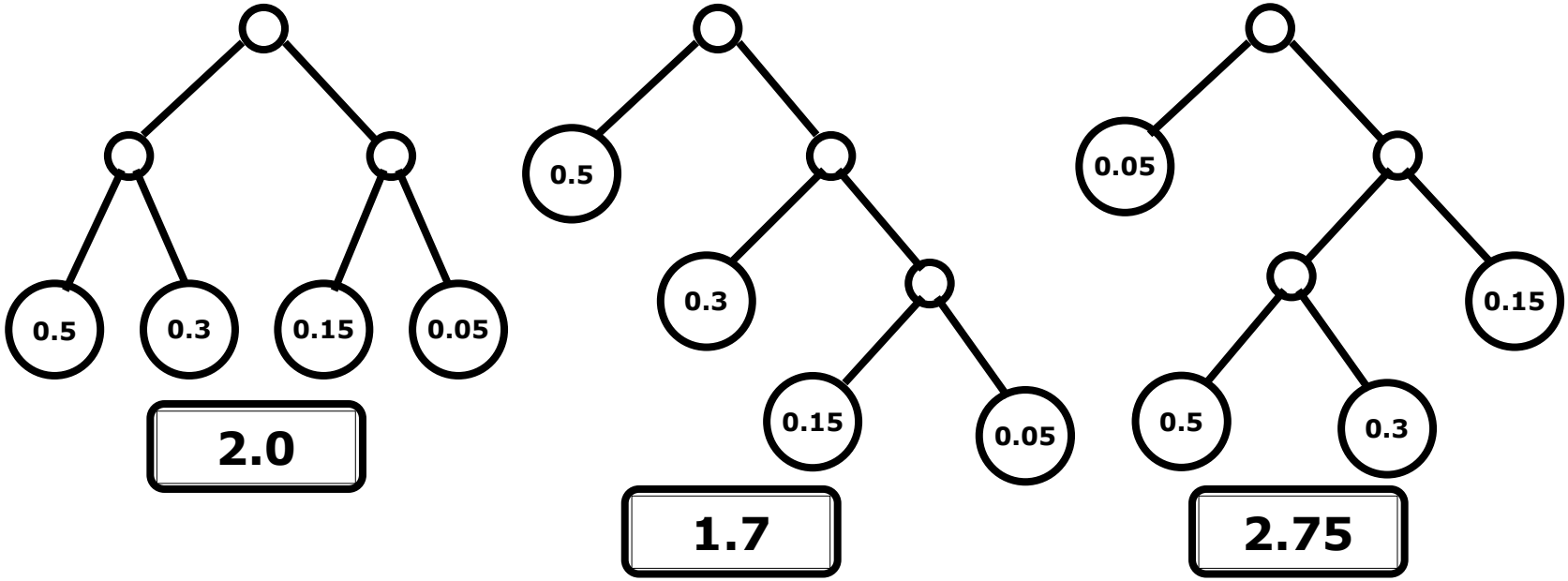
$$\sum w_i . hl(u_i)$$

Obecná formulace:

Pravidelný kořenový strom T_u stupně **r** s **n** listy, které jsou ohodnoceny reálnými čísly $\mathbf{w_i} = w(u_i)$

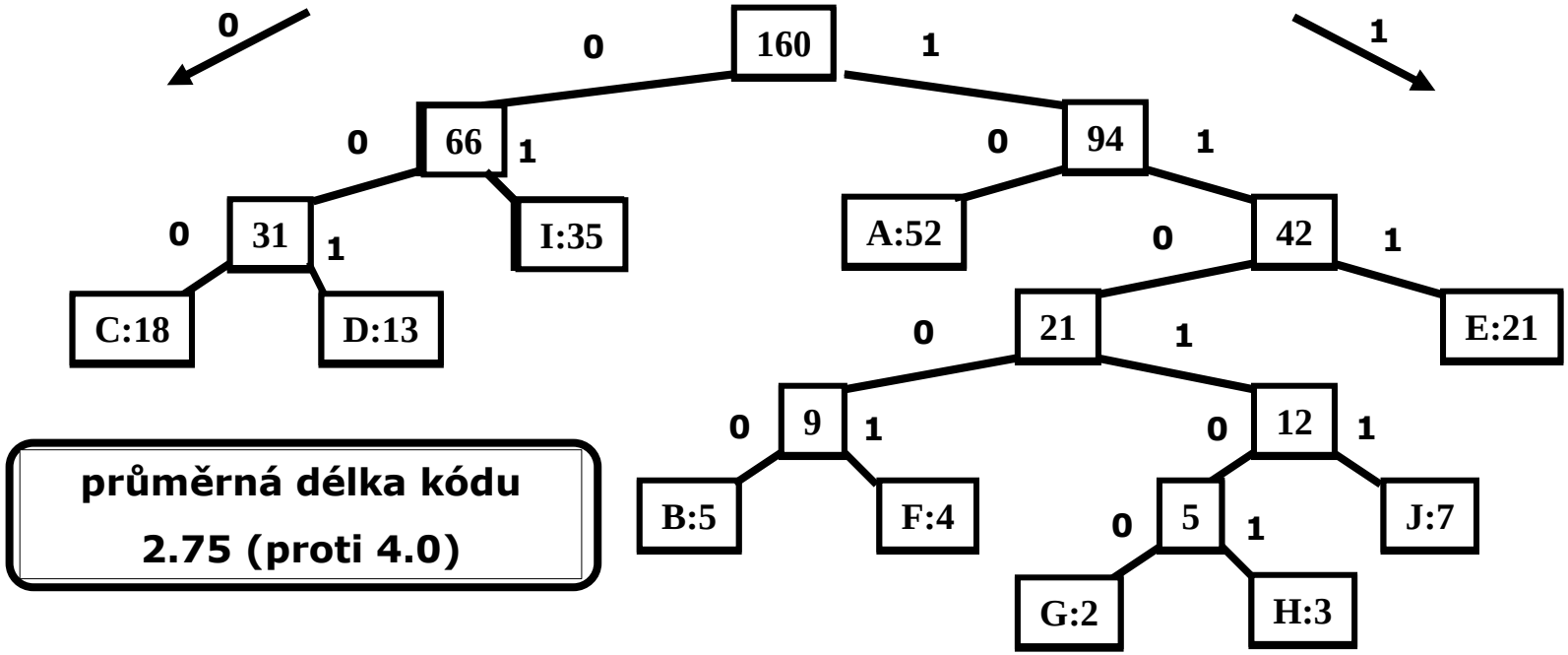
vnější w-délka $\mathbf{E_w(T_u) = \sum w_i . hl(u_i)}$

Jak vypadá strom s minimální vnější w-délkou ?



Jiná 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
kód	10	11000	000	001	111	11001	110100	110101	01	11011



průměrná délka kódu
2.75 (proti 4.0)

Huffmanův algoritmus (pro r=2)

```
void Huffman(Weights w, int n) {
  Q.Init();
  for (int i=1; i<=n; i++)  {
    u = MakeNode(w[i]); Q.Push(u);
  }
  for (int i=1; i<n; i++) {
    x = Q.ExtMin(); y = Q.ExtMin();
    z = MakeNode(w[x]+w[y]);
    z.left = x; z.right = y;
    Q.Push(z);
  }
  return Q.ExtMin();
}
```

Zobecnění pro libovolné **r** (stupeň stromu) je přímočaré ...

Kontrolní otázky

6.12 **Upravte Huffmanův algoritmus tak, aby vytvářel minimální pravidelný strom se zadaným stupněm r.**

6.13 **Dokažte, že hodnotu E_w vnější w-délky pravidelného stromu vytvořeného Huffmanovým algoritmem lze spočítat jako součet ohodnocení všech jeho vnitřních uzlů.**