Stromy, haldy, prioritní fronty

prof. Ing. Pavel Tvrdík CSc.

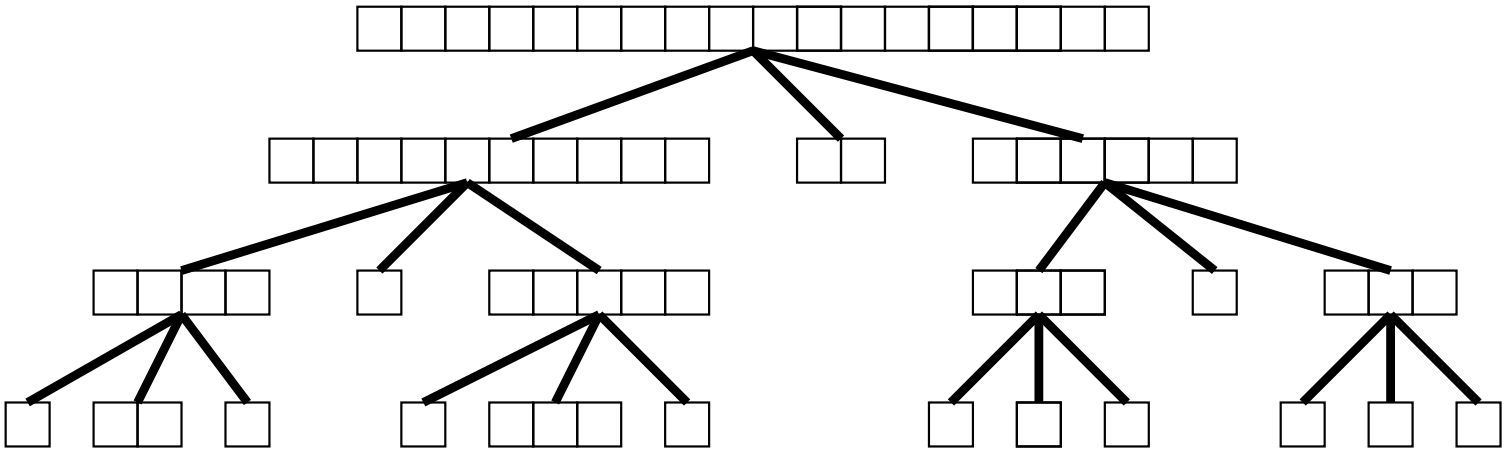
Katedra počítačů FEL
České vysoké učení technické

DSA, ZS 2008/9, Přednáška 6

http://service.felk.cvut.cz/courses/X36DSA/

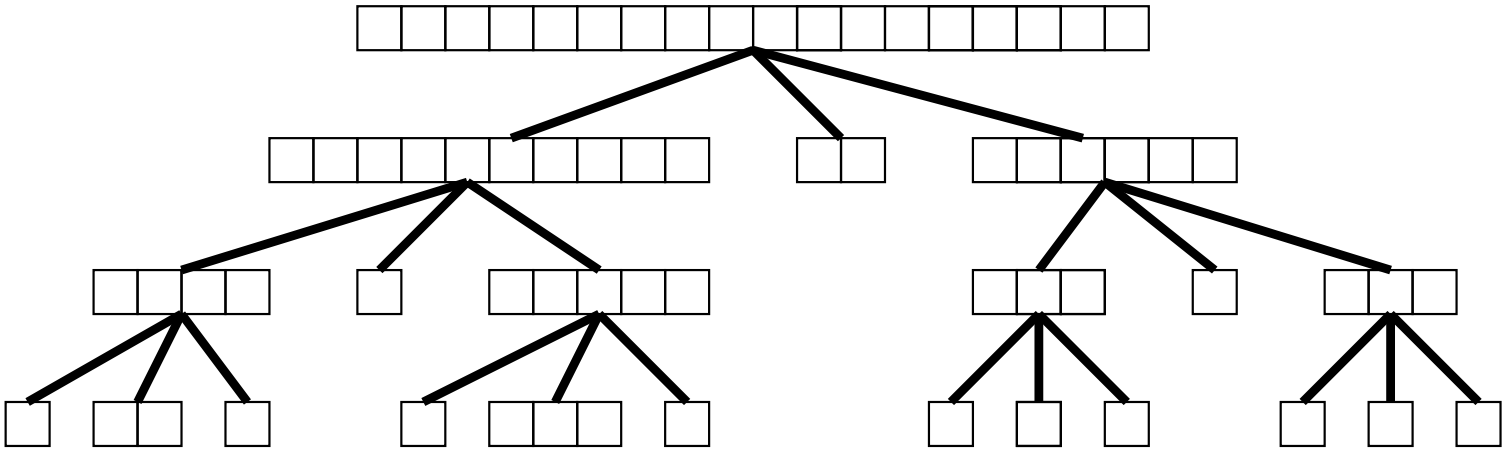
Motivace 1: Volání rekurzivní funkce

- Řazení dat QuickSortem.



Motivace 2: Rekurzivně definova(tel)ná množina dat

- Množina všech permutací dané posloupnosti.

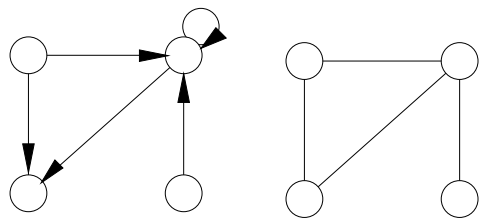


Orientované a neorientované grafy

Definice

- **Orientovaný graf** *o* n *uzlech je dvojice* $G = (V(G), E(G))$, *kde*
  - $V(G)$ *je* **množina uzlů**, $n = |V(G)|$, *a*
  - $E(G) = \{\langle u, v\rangle; u, v \in V(G)\} \subset V(G) \times V(G)$ **množina** $=$ **orientovaných hran**, *čili* **uspořádaných** *dvojic uzlů (graficky šipek).*
  - *Orientovaná hrana* $\langle u, u\rangle$ *se nazývá* **smyčka**.
- **(Neorientovaný obyčejný) graf** *o* n *uzlech je dvojice* $G = (V(G), E(G))$, *kde* $V(G)$ *je* **množina uzlů** *a*
  - $E(G) = \{\{u, v\}; u, v \in V(G), u \neq v\}$ $=$ **množina neorientovaných hran**, *čili* **neuspořádaných** *dvojic uzlů.*
  - *Smyčky nejsou povoleny.*

Poznámka: Nuance v rozdílech pojmů pro orientované a neorientované grafy budeme pro stručnost pomíjet.

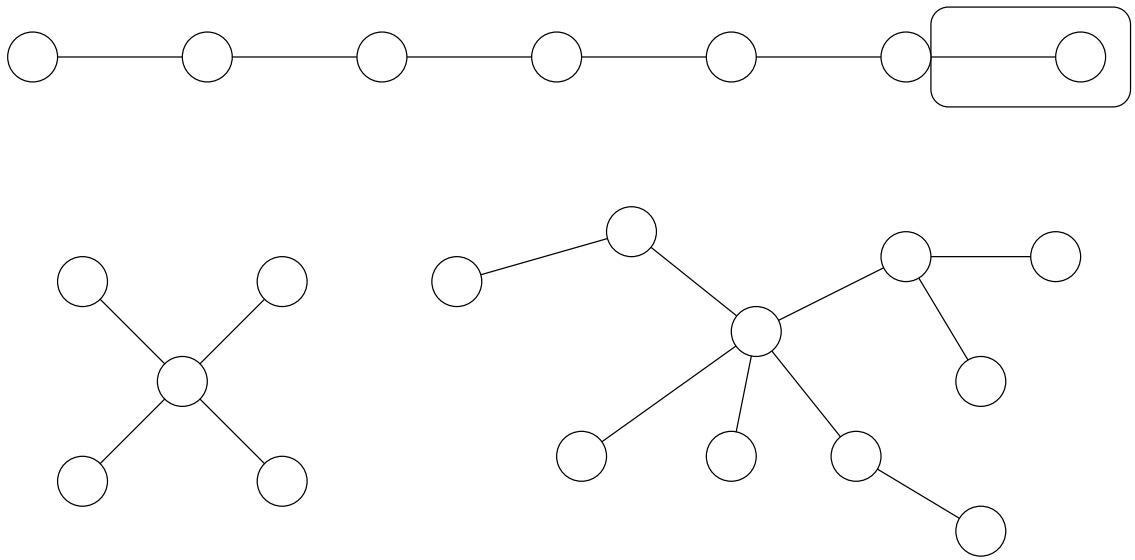


(Volné) stromy

Definice

- **Volný strom** *je každý souvislý acyklický a neorientovaný graf.*
- **Les** *je každý acyklický a neorientovaný graf.*

Poznámka: Většina algoritmů nad stromy funguje i nad lesy.



Vlastnosti volných stromů

Věta

Nechť $G = (V(G), E(G))$ je neorientovaný graf. Pak následující tvrzení jsou ekvivalentní.

1. *G je volný strom.*
2. *Jakékoli 2 uzly v G jsou spojeny jedinečnou jednoduchou cestou.*
3. *G je souvislý, ale pokud vyjmeme jakoukoli hranu z $E(G)$, výsledný graf bude nesouvislý.*
4. *G je souvislý a $|E(G)| = |V(G)| - 1$.*
5. *G je acyklický a $|E(G)| = |V(G)| - 1$.*
6. *G je acyklický, ale pokud přidáme jakoukoli hranu do $E(G)$, výsledný graf bude obsahovat aspoň jeden cyklus.*

Kořenové stromy

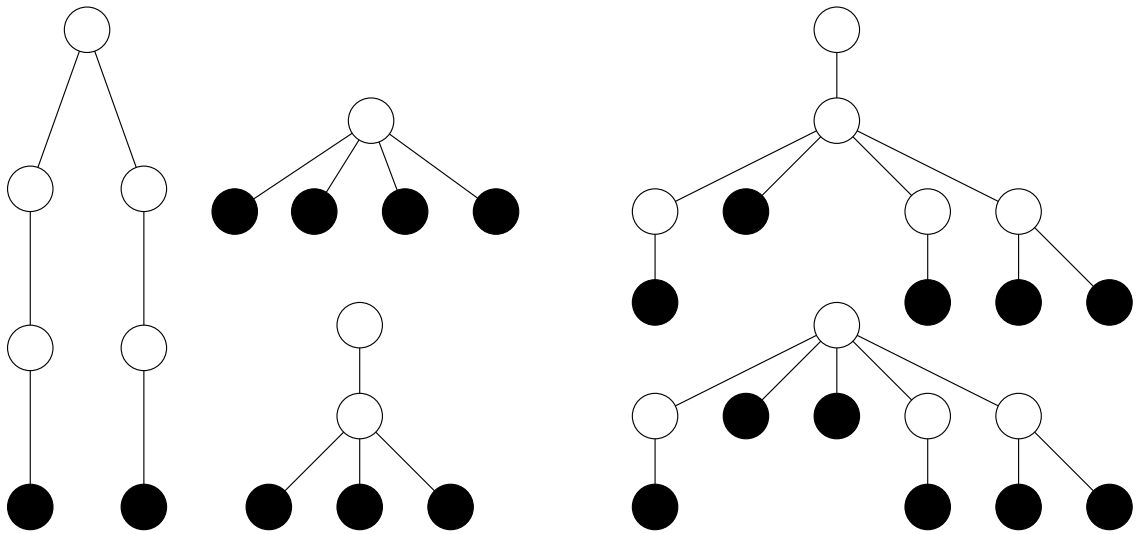
Definice

- **Kořenový strom** *je volný strom, ve kterém jeden z uzlů je odlišen od ostatních jako* **kořen**, r.
- *Uzly* u *ve vzdálenosti* d *od kořenu* r *tvoří hladinu uzlů v* **hloubce** $h(u) = d$. *Kořen má hloubku* $h(r) = 0$.
- *Nechť uzel* u *leží na (jedinečné) cestě z kořene* r *do uzlu* v. *Pak* u *je* **předchůdce** v *a* v *je* **následník** u.
- *Nejbližší předchůdce uzlu je jeho* **rodič** *a nejbližší následník je jeho* **potomek**.

  $\Rightarrow$ *Každý uzel kromě kořenu má jedinečného rodiče.*
- *Uzly se stejným rodičem jsou* **sourozenci**.
- *Uzel, který nemá potomky, se nazývá* **list**. *Ostatní uzly jsou* **vnitřní**.
- **Stupeň** *vnitřního uzlu je počet jeho potomků (rodič se nepočítá)!!!!!!!*
- k**-ární strom**: *každý vniřní uzel má stupeň* **nejvýše** k.

Kořenové stromy

Poznámka: Kořenový strom je rekurzivně definovatelný.



Uspořádané stromy

- k-ární stromy jsou z implementačních důvodů obvykle uspořádané.

Definice

Uspořádaný strom.

- *Potomci každého vnitřního uzlu jsou* **očíslovány** *(zleva doprava) čísly* $\{1, \ldots, \#\text{počet potomků}\}$.
- *Dva kořenové stromy se stejnými uzly v jiném pořadí jsou* **různé** *uspořádané stromy.*

obrazek

Poziční stromy

- k-ární stromy jsou z implementačních důvodů ještě častěji poziční.

Definice

Poziční strom.

- *Potomci každého vnitřního uzlu jsou* **označeny různými** *čísly.*
- *Pokud žádný potomek není označen číslem i, pak i-tý potomek* **chybí** *a příslušný podstrom je* **prázdný** *(*`NIL`*).*
- *Dva kořenové stromy se stejnými podstromy ale jiným označením pozic jsou* **různé** *poziční stromy.*
- k**-ární strom** *je poziční strom, kde každý uzel má potomky označeny čísly* $\leq k$.

obrazek

Binární stromy

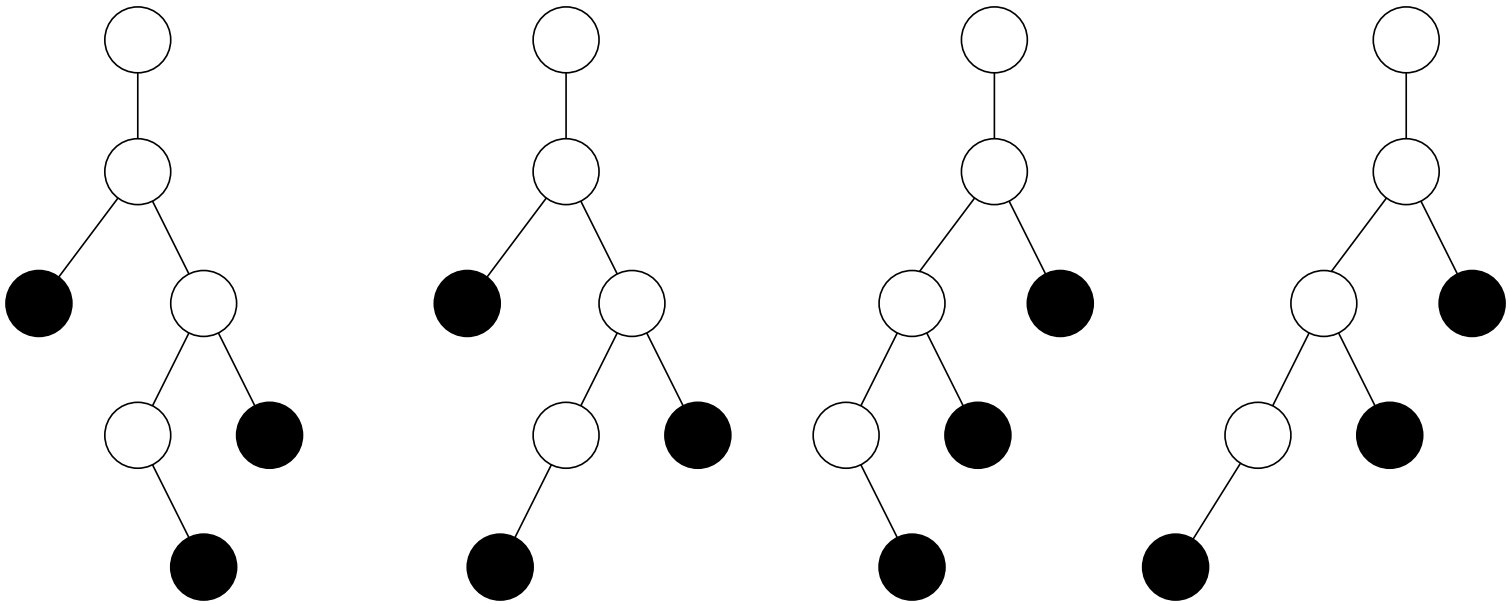
Definice

k-ární strom pro $k = 2$ se nazývá **binární**.

Alternativní **rekurzivní** definice binárního stromu.

Definice

- **Binární strom** *(BS) T je* **datová struktura** *definovaná nad konečnou množinou uzlů, která buď*
  - *neobsahuje žádné uzly nebo*
  - *obsahuje 3 disjunktní množiny uzlů:* **kořen**, *BS zvaný* **levý** *podstrom a BS zvaný* **pravý** *podstrom.*
- *Pokud není levý podstrom prázdný (*`NIL`*), jeho kořen je* **levým** *potomkem kořenu. Podobně pro pravý podstrom.*



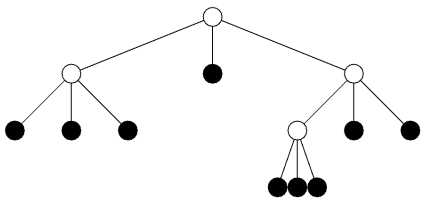
Plné, výškově vyvážené a úplné k-ární stromy

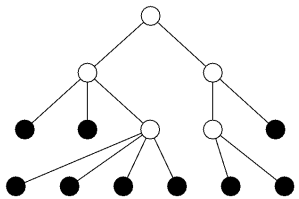
Definice

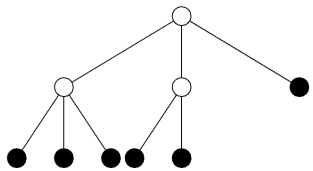
(a) **Plný k-ární strom**: *Každý vnitřní uzel má stupeň právě k.*

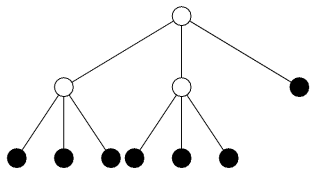
(b) **Výškově vyvážený strom**: *Hloubka libovolných dvou listů se liší nejvýše o 1.*

(c) **Úplný k-ární strom**: *Výškově vyvážený plný k-ární strom, plněný zleva doprava, kde nejvýše 1 vnitřní uzel má stupeň menší než k.*







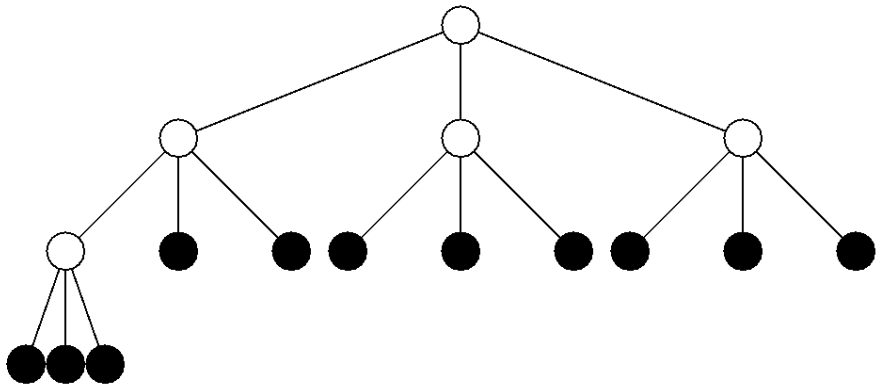


Vlastnosti úplných k-árních stromů

Věta

Nechť T je úplný k-ární strom o n uzlech. Pak

- *Hloubka $h(T) = \lceil \log_k n \rceil$.*
- *Počet uzlů v hloubce $i < h(T)$ je k^i.*
- *Pro $n = \frac{k^{h(T)+1}-1}{k-1}$ mají všechny listy hloubku $h(T)$ a všechny vnitřní uzly stupeň k. Počet listů je pak $k^{h(T)}$ a počet vnitřních uzlů je $1 + k + k^2 + \cdots + k^{h(T)-1} = \frac{k^{h(T)}-1}{k-1}$.*



Vlastnosti úplných binárních stromů (UBS)

Věta

Nechť T je UBS o n uzlech. Pak

(a) *Hloubka $h(T) = \lceil \log n \rceil$.*

(b) *Počet uzlů v hloubce $i < h(T)$ je 2^i.*

(c) *Počet vnitřních uzlů je $\lfloor n/2 \rfloor$ a počet listů je $\lceil n/2 \rceil$.*

(d) *Pro $n = 2^{h(T)+1} - 1$ mají všechny listy hloubku $h(T)$.*

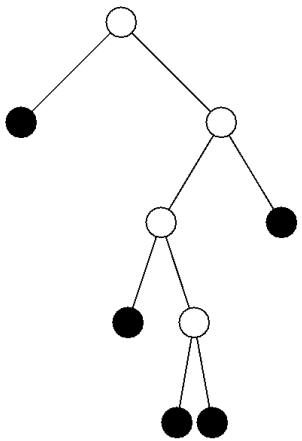
Věta

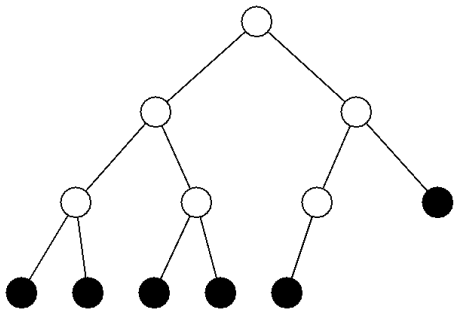
Nechť T je libovolný binární strom o n uzlech. Pak

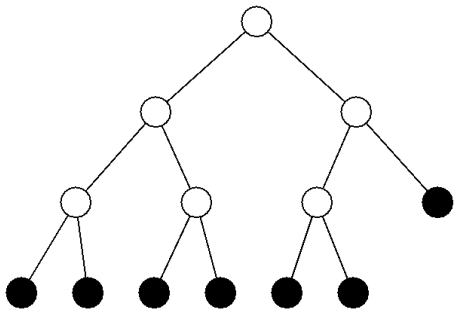
(e) $n - 1 \geq h(T) \geq \log n$.

(f) *Počet uzlů stupně 2 je počet listů minus jedna. (Snadno indukcí.)*

Příklady úplných binárních stromů (UBS)







Implementace binárního stromu pomocí spojových struktur

```
struct node
{
    int key;
    node *parent;
    node *left;
    node *right
}
```

Implementace UBS pomocí pole

Věta

Nechť T je UBS o n uzlech. Předpokládejme, že uzly T jsou číslovány zleva doprava shora dolů, kořen má číslo 1. Pak lze UBS reprezentovat pomocí jednorozměrného pole $A[1, \ldots, n]$ tak, že uzel stromu i je reprezentován prvkem pole $A[i]$. Indexy rodiče, levého potomka a pravého potomka uzlu i v poli A lze spočítat pomocí následujících funkcí:

- $\textsc{Parent}(i) = \lfloor i/2 \rfloor$.
- $\textsc{Left}(i) = 2i$.
- $\textsc{Right}(i) = 2i + 1$.

Poznámka: Implementace těchto funkcí je 1 strojová instrukce.

obrazek

Binární halda (heap)

Definice

Uvažujme pole $A[1, \ldots, Length(A)]$ *implementující UBS pomocí funkcí* PARENT, LEFT, RIGHT. *Pak* **binární halda** *o velikosti* $Heap\_Size(A) < Length(A)$ *je dynamická množina uložená v* $A[1, \ldots, Heap\_Size(A)]$, *která splňuje* **H-vlastnost**:

- *Pro každý prvek s indexem* $1 < i \leq Heap\_Size(A)$ *platí*

$$A[\text{PARENT}(i)] \geq A[i] \tag{1}$$

Důsledek

- *Největší hodnota je v kořeni* $A[1]$.
- *Hloubka haldy o* $Heap\_Size(A)$ *prvcích je* $\Theta(\log(heap\_size(A)))$.

Poznámka: Takto definovaná halda nemá nic společného s haldou ve smyslu dynamické paměti.

Algoritmus udržování H-vlastnosti

Algoritmus

Vstup: *Pole* A *a index* i *takový, že binární podstromy s kořeny v* $A[\textsc{Left}(i)]$ *a* $A[\textsc{Right}(i)]$ *jsou binární haldy (splňují H-vlastnost), ale přitom* $A[i] < A[\textsc{Left}(i)]$ *nebo* $A[i] < A[\textsc{Right}(i)]$.

procedure $\textsc{Heapify}(A, i)$

{

$l \leftarrow \textsc{Left}(i)$;

$r \leftarrow \textsc{Right}(i)$;

if $(i \leq Heap\_Size(A)\ \&\ A[l] > A[i])$

then $Largest \leftarrow l$ **else** $Largest \leftarrow i$;

if $(r \leq Heap\_Size(A)\ \&\ A[r] > A[Largest])$

then $Largest \leftarrow r$;

if $(Largest \neq i)$

then $\{A[i] \leftrightarrow A[Largest]; \textsc{Heapify}(A, Largest)\}$

}

Složitost algoritmu udržování H-vlastnosti

obrazek

Věta

Časová složitost operace HEAPIFY *na podstromu o* n *uzlech (s kořenem* i*) je*

$$t_{\mathrm{HP}}(n) \leq t_{\mathrm{HP}}(2n/3) + \Theta(1) \tag{2}$$

(což znamená $t_{\mathrm{HP}}(n) = \Theta(\log n)$*, viz přednáška za 2 týdny).*

Důkaz. Po provedení $\Theta(1)$ operací, procedura HEAPIFY se volá rekurzivně pro levý nebo pravý podstrom a ten má v nejhorším případě velikost $\frac{2n}{3}$ (případ, kdy poslední hladina stromu je zaplněna přesně z jedné poloviny). ■

Konstrukce haldy

Vstup: libovolné pole $A[1, \ldots, Length(A)]$.

procedure BUILD_HEAP(A)
{
$Heap\_Size(A) \leftarrow Length(A)$;
for $(i = \lfloor length(A)/2 \rfloor$ **downto** $1)$
do HEAPIFY(A, i)
}

Poznámky:

- Prvky v $A[\lfloor n/2 \rfloor + 1, \ldots, n]$ = listy = 1-prvkové haldičky.
- Procedura BUILD_HEAP mapuje postupně na **zbývající** prvky pole operaci HEAPIFY tak, aby postupně směrem nahoru platila H-vlastnost.

Složitost algoritmu konstrukce haldy

Věta

Časová složitost $t_{\mathrm{BH}}(n) = O(n)$.

Důkaz. Protože se $n/2$ krát volá HEAPIFY, které trvá $O(\log n)$, je $t_{\mathrm{BH}}(n) = O(n \log n)$. Protože halda je UBS, platí, že ve výšce h je nejvýše $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ uzlů. HEAPIFY haldy o výšce h trvá $O(h)$. Tedy

$$t_{\mathrm{BH}}(n) = \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \right) = O(2n) = O(n),$$

protože pro libovolné $k > 1$ platí

$$\sum_{h=0}^{k} \frac{h}{2^h} = \sum_{h=1}^{k} \frac{1}{2^h} + \sum_{h=2}^{k} \frac{1}{2^h} + \cdots + \sum_{h=k}^{k} \frac{1}{2^h} =$$

$$\left(1 - \frac{1}{2^k}\right) + \left(\frac{1}{2} - \frac{1}{2^k}\right) + \cdots + \left(\frac{1}{2^{k-1}} - \frac{1}{2^k}\right) = \left(2 - \frac{1}{2^{k-1}}\right) - \frac{k}{2^k} < 2.$$

HeapSort

procedure HeapSort(A)
{
    Build_Heap(A);
    for ($i \leftarrow length(A)$ **downto** 2)
        do { $A[1] \leftrightarrow A[i]$;
            $Heap\_Size(A) \leftarrow Heap\_Size(A) - 1$;
            Heapify(A, i)
        }
}

Věta

Časová složitost $t_{\mathrm{HS}}(n) = O(n \log n)$.

Důkaz. $t_{\mathrm{HS}}(n) = t_{\mathrm{BH}}(n) + \sum_{i=n-1}^{2}(t_{\mathrm{HP}}(i) + \Theta(1)) =$
$O(n) + O(\sum_{i=2}^{n-1} \log i) = O(n \log n)$. ■

Prioritní fronta

Definice

Prioritní fronta *je dynamická množina umožňující efektivní realizaci operací vkládání libovolných nových prvků a jejich vybírání v pořadí jejich velikosti pomocí následujících operací:*

- $\textsc{Get\_Max}(S)$,
- $\textsc{Extract\_Max}(S)$,
- $\textsc{Insert}(S, k)$.

- Jedna z nejužitečnějších aplikací haldy. Např.
  - rozvrhování úloh podle relativních priorit v multitaskingovém systému:
    - ★ při dokončení běžící úlohy se vybere nová s nejvyšší prioritou,
    - ★ nové úlohy jsou průběžně zařazovány.
  - událostmi řízená simulace:
    - ★ prvky haldy jsou události s časovou značkou, které se mají simulovat,
    - ★ simulace se musí provádět v pořadí časových značek,
    - ★ nové události se zařazují podle časových značek,
    - ★ místo operace Max potřebujeme Min.

Operace nad prioritní frontou

```
function GET_MAX(A)
{
    return A[1]
}
function EXTRACT_MAX(A)
{
    if (Heap_Size(A) < 1)
        then error(HeapUnderflow);
    max ← A[1];
    A[1] ← A[Heap_Size[A]];
    Heap_Size(A) ← Heap_Size(A) − 1;
    HEAPIFY(A, 1);
    return max}
}
```

Operace nad prioritní frontou

procedure INSERT(A, k)
{
    $Heap\_Size(A) \leftarrow Heap\_Size(A) + 1;$
    $i \leftarrow Heap\_Size(A);$
    while $(i > 1$ & $A[\text{PARENT}(i)] < k)$
        do {
            $A[i] \leftarrow A[\text{PARENT}(i)];$
            $i \leftarrow \text{PARENT}(i);$
        }
    $A[i] \leftarrow k$
}