Základní datové struktury III: Stromy, haldy, prioritní fronty

prof. Ing. Pavel Tvrdík CSc.

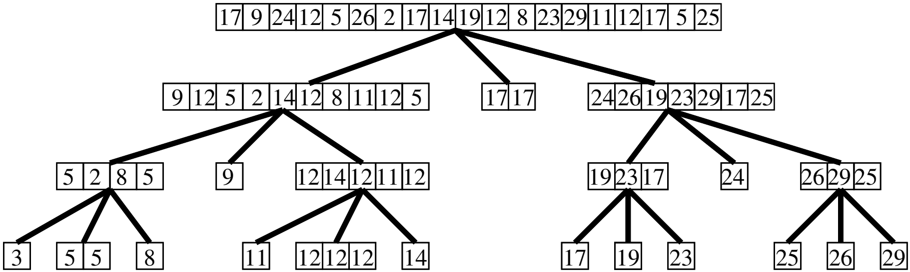
Katedra počítačových systémů FIT
České vysoké učení technické v Praze

DSA, ZS 2009/10, Předn. 5

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

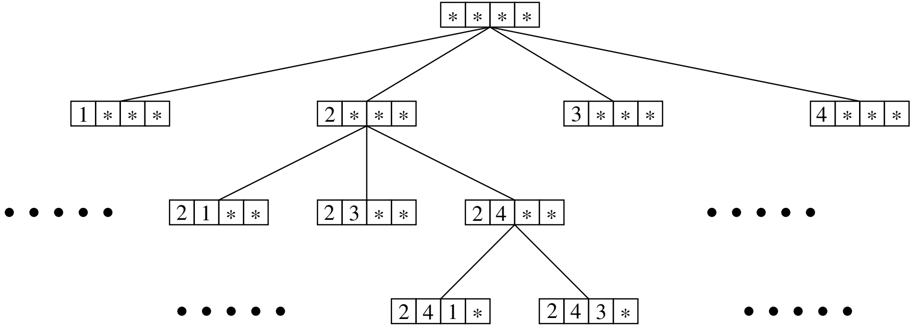
Motivace 1: Volání rekurzivní funkce

- Řazení dat QuickSortem.



Motivace 2: Rekurzivně definova(tel)ná množina dat

- Množina všech permutací dané posloupnosti.

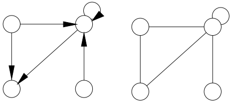


Orientované a neorientované grafy

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 se někdy povolují, někdy ne..*

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

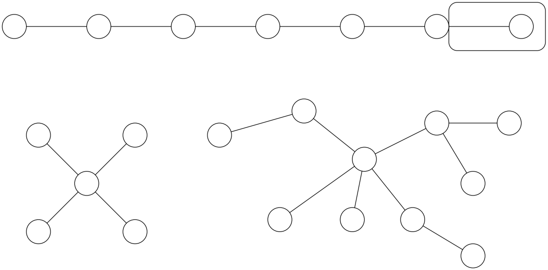


(Volné) stromy

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.*

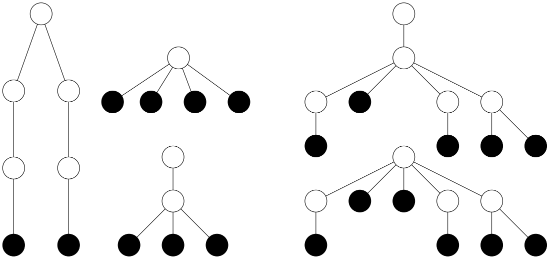
Kořenové stromy

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* **předek** v *a* v *je* **potomek** u.
- *Nejbližší předek uzlu je jeho* **rodič** *a nejbližší potomek je jeho* **syn**.
  $\Rightarrow$ *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 synů (rodič se nepočítá)!!!!!!!*
- k-**ární strom**: *každý vnitřní uzel má stupeň* **nejvýše** k.

Kořenové stromy

Poznámka: Kořenový strom je rekurzivně definovatelný.



Uspořádané stromy

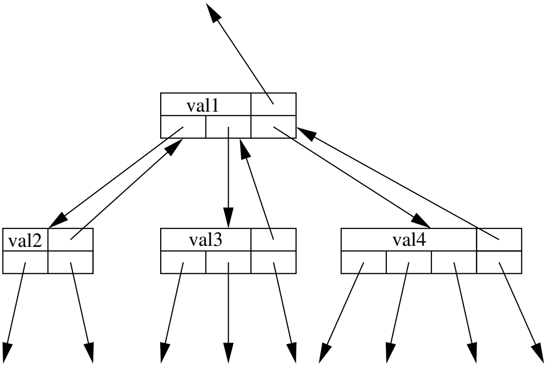
- k-ární stromy jsou z implementačních důvodů obvykle uspořádané.

Definice

Uspořádaný strom.

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

Implementace uspořádaných stromů



Poziční stromy

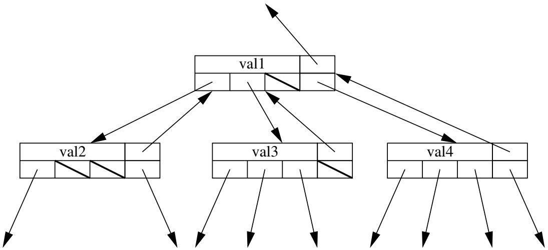
- k-ární stromy jsou z implementačních důvodů ještě častěji poziční.

Definice

Poziční strom.

- *Synové každého vnitřního uzlu jsou* **označeny různými** *čísly.*
- *Pokud žádný syn není označen číslem i, pak i-tý syn* **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 syny označeny čísly $\leq k$.*

Implementace pozičních stromů



Binární stromy

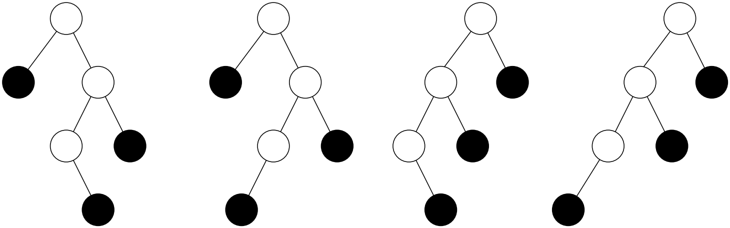
Definice

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** *synem kořenu. Podobně pro pravý podstrom.*



Různé binární stromy s 3 vnitřními uzly a 3 listy.

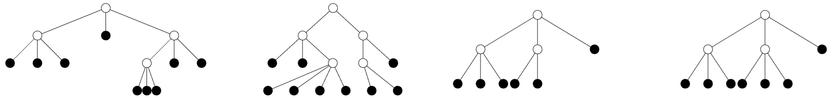
Plné, výškově vyvážené a úplné k-ární stromy

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.*

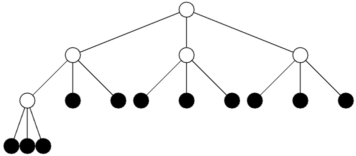


Vlastnosti úplných k-árních stromů

Věta

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

- *Hloubka $h(T)$ je $\lfloor \log_k n \rfloor \leq h(T) \leq \lceil \log_k n \rceil$.*
- *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 $1 + k + k^2 + \cdots + k^{h(T)-1} = \frac{k^{h(T)}-1}{k-1}$.*



Domácí úkol: Najděte přesný výraz pro $h(T)$.

Vlastnosti úplných binárních stromů (UBS)

Věta

Nechť T je UBS o n uzlech. Pak

(a) *Hloubka $h(T) = \lfloor \log n \rfloor$.*

(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

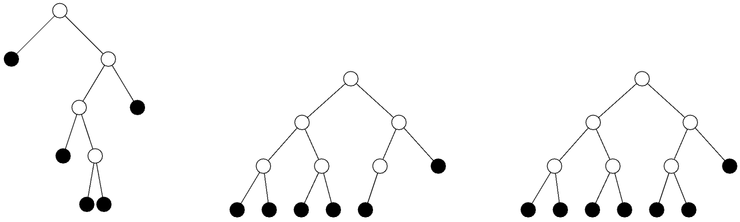
Nechť T je libovolný binární strom o n uzlech. Pak

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

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

Příklad plného a úplných binárních stromů (UBS)

Plný BS s 9 uzly | UBS s 12 uzly | UBS s 13 uzly



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 a že 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 syna a pravého syna 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.

Binární halda (heap)

Definice

Uvažujme pole $A[1,\ldots,Length(A)]$ *implementující UBS pomocí funkcí* $\textsc{Parent}$, $\textsc{Left}$, $\textsc{Right}$. *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*

$$A[\textsc{Parent}(i)] \geq A[i]. \tag{1}$$

Důsledek

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

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

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

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

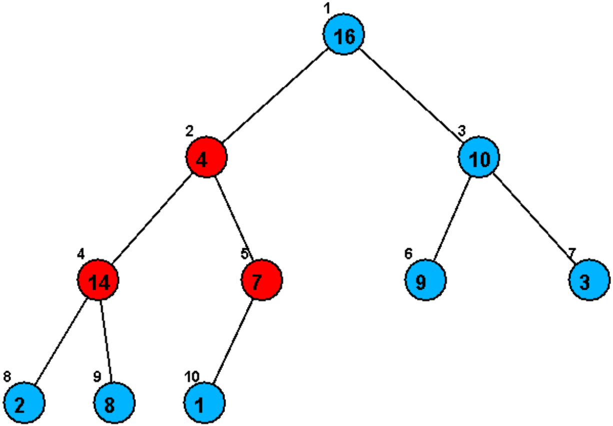
  then $Largest \leftarrow r$;

if $(Largest \neq i)$

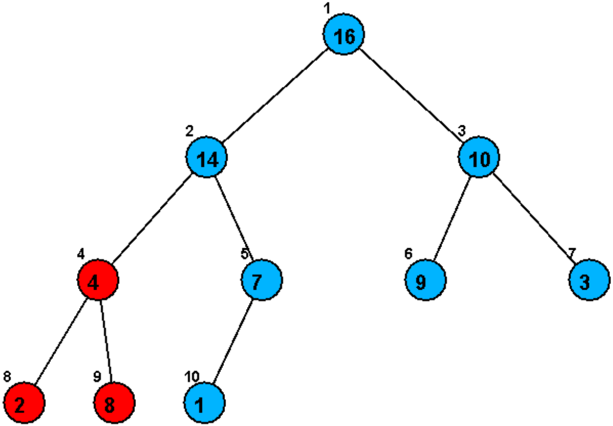
  then $\{A[i] \leftrightarrow A[Largest]; \textsc{Heapify}(A, Largest)\}$

$\}$

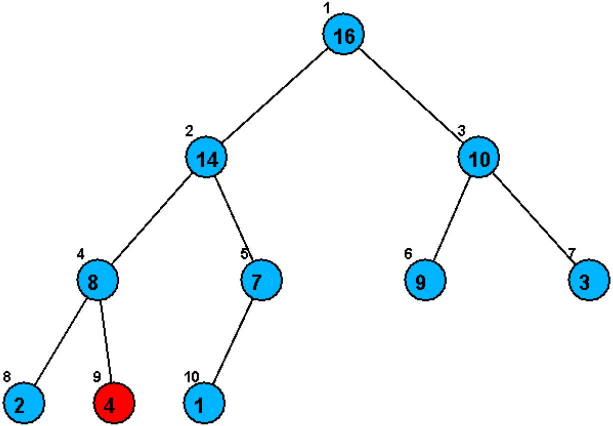
Příklad běhu HEAPIFY



Příklad běhu HEAPIFY



Příklad běhu HEAPIFY



Složitost algoritmu udržování H-vlastnosti

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 $t_{\mathrm{HP}}(n) = O(\log n)$, viz přednáška č.8).

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

Algoritmus

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

procedure $\textsc{Build\_Heap}(A)$

{

(1) $Heap\_Size(A) \leftarrow Length(A)$;

(2) **for** $(i = \lfloor length(A)/2 \rfloor$ **downto** $1)$

(3) **do** $\textsc{Heapify}(A, i)$

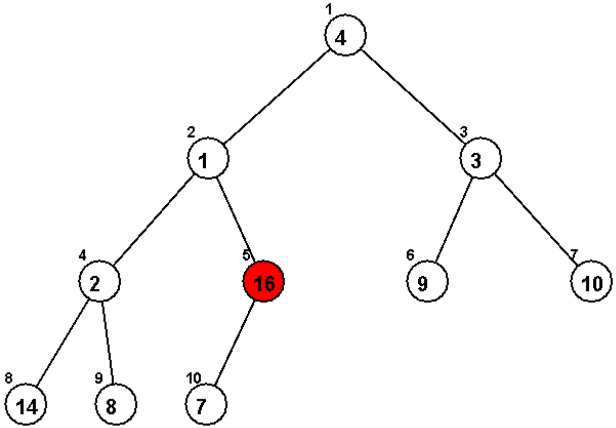
}

Poznámky:

- Prvky v $A[\lfloor n/2 \rfloor + 1, \dots, n]$ = listy = 1-prvkové haldičky.
- Procedura $\textsc{Build\_Heap}$ mapuje postupně na **zbývající** prvky pole operaci $\textsc{Heapify}$, takže směrem nahoru postupně platí H-vlastnost.

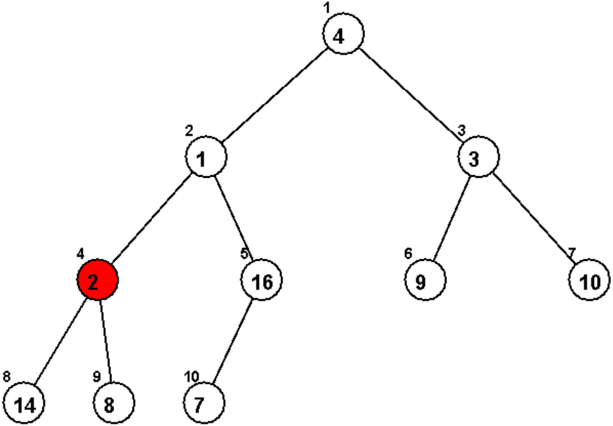
Příklad konstrukce haldy

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7



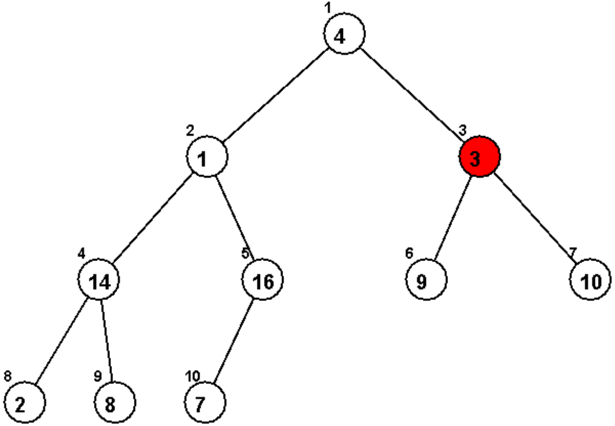
Příklad konstrukce haldy

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7



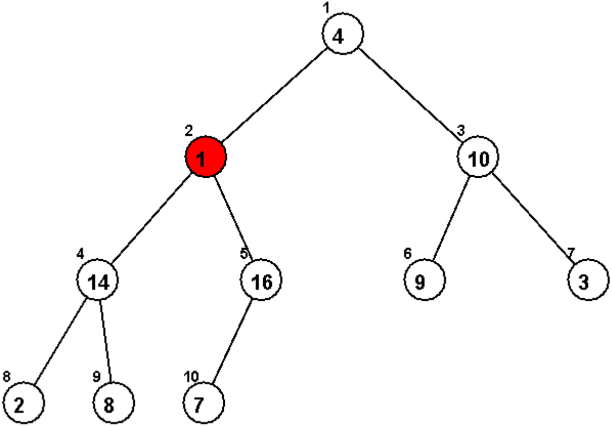
Příklad konstrukce haldy

1	2	3	4	5	6	7	8	9	10
4	1	3	14	16	9	10	2	8	7



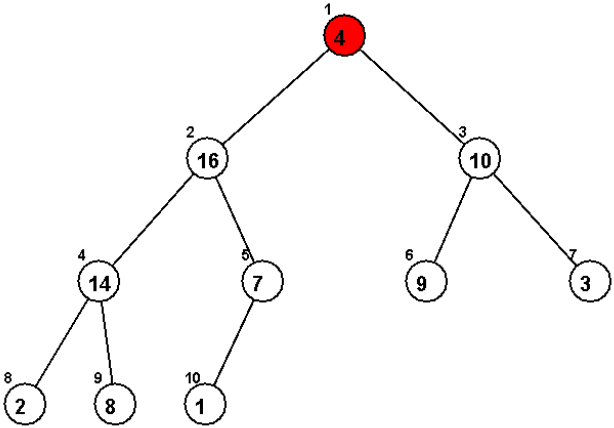
Příklad konstrukce haldy

1	2	3	4	5	6	7	8	9	10
4	1	10	14	16	9	3	2	8	7



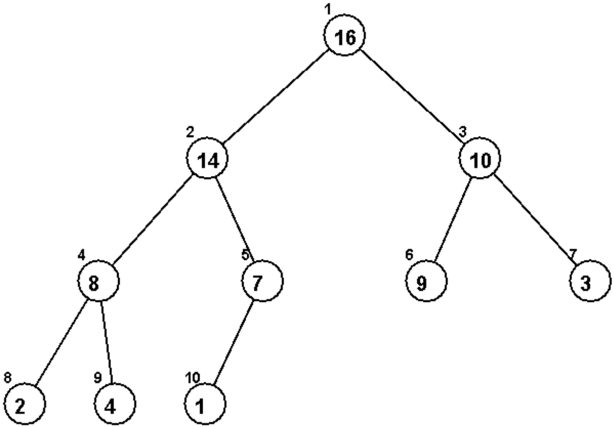
Příklad konstrukce haldy

1	2	3	4	5	6	7	8	9	10
4	16	10	14	7	9	3	2	8	1



Příklad konstrukce haldy

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1



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

Algoritmus

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

procedure $\textsc{HeapSort}(A)$

$\{$

(1) $\textsc{Build\_Heap}(A);$

(2) **for** $(i \leftarrow length(A)$ **downto** $2)$

(3) **do** $\{\ A[1] \leftrightarrow A[i];$

(4) $Heap\_Size(A) \leftarrow Heap\_Size(A) - 1;$

(5) $\textsc{Heapify}(A, 1)$

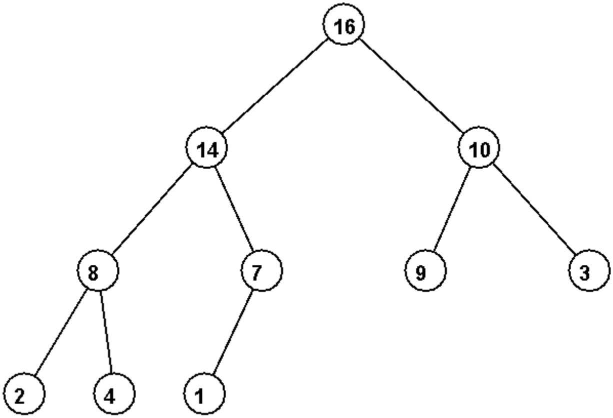
$\}\quad\}$

Věta

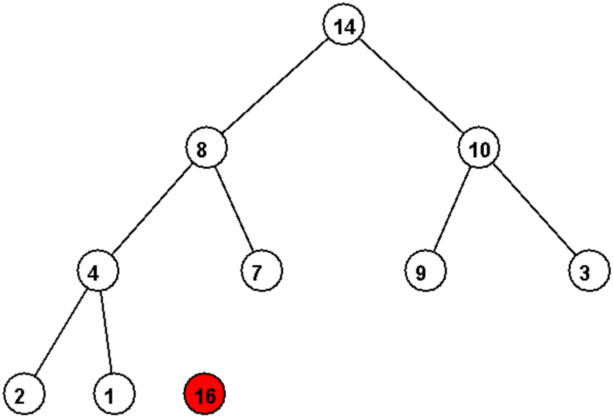
Časová složitost $t_{\mathrm{HS}}(n) = O(n \log n)$.

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

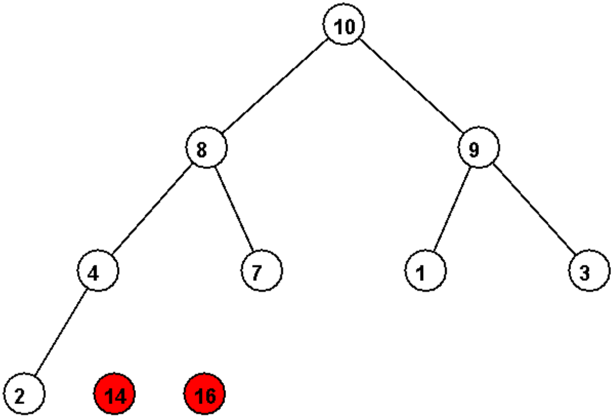
Příklad běhu HeapSortu



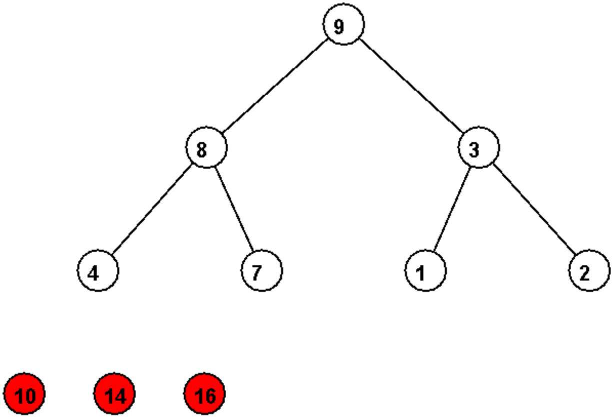
Příklad běhu HeapSortu



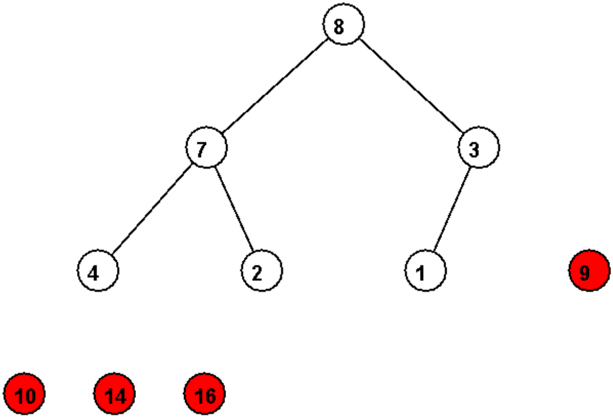
Příklad běhu HeapSortu



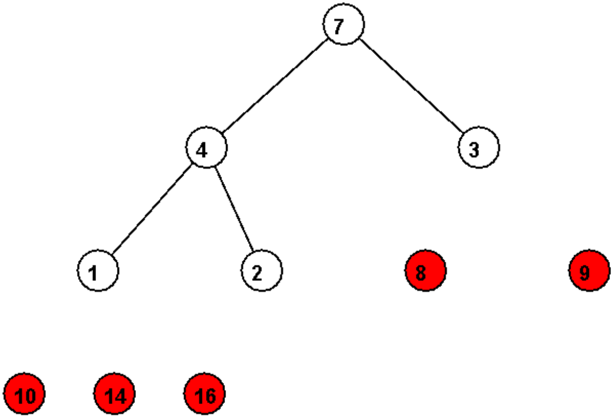
Příklad běhu HeapSortu



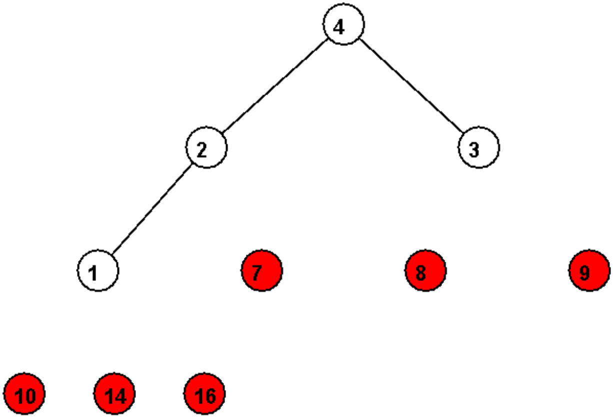
Příklad běhu HeapSortu



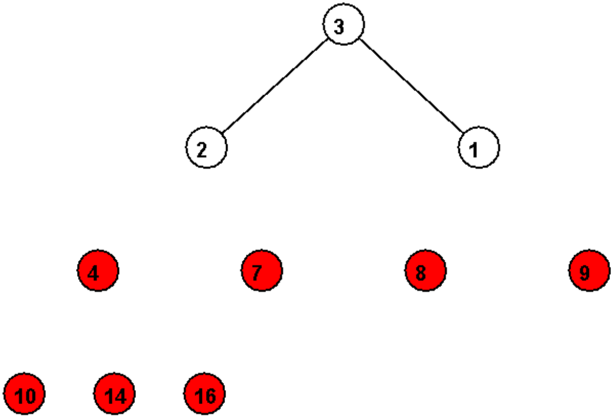
Příklad běhu HeapSortu



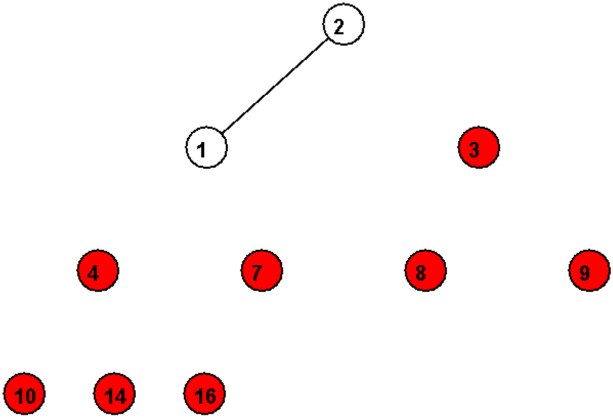
Příklad běhu HeapSortu



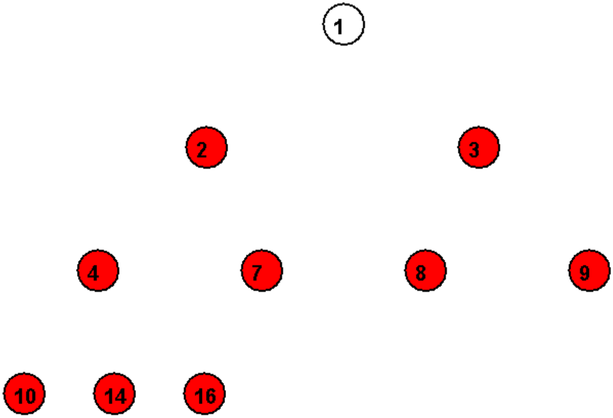
Příklad běhu HeapSortu



Příklad běhu HeapSortu



Příklad běhu HeapSortu



Příklad běhu HeapSortu

1	2	3	4	5	6	7	8	9	10
1	2	3	4	7	8	9	10	14	16

Prioritní fronta

Definice

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

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

- 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.

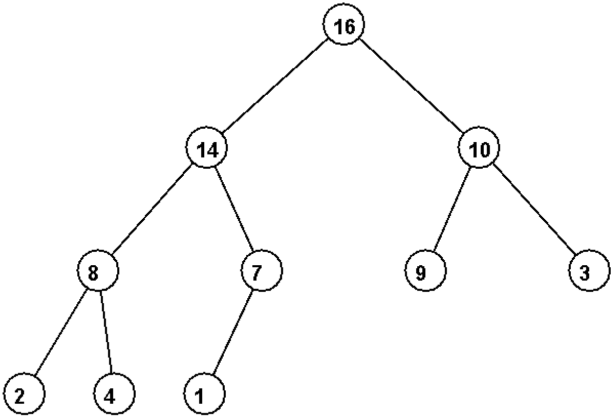
Operace nad prioritní frontou

function $\textsc{Get\_Max}(A)$
{
    return $A[1]$
}
procedure $\textsc{Extract\_Max}(A)$
{
(1) **if** $(Heap\_Size(A) < 1)$
(2)   **then** $error$(HeapUnderflow);
(3) $max \leftarrow A[1]$;
(4) $A[1] \leftarrow A[Heap\_Size[A]]$;
(5) $Heap\_Size(A) \leftarrow Heap\_Size(A) - 1$;
(6) $\textsc{Heapify}(A, 1)$;
(7) **return** max}
}

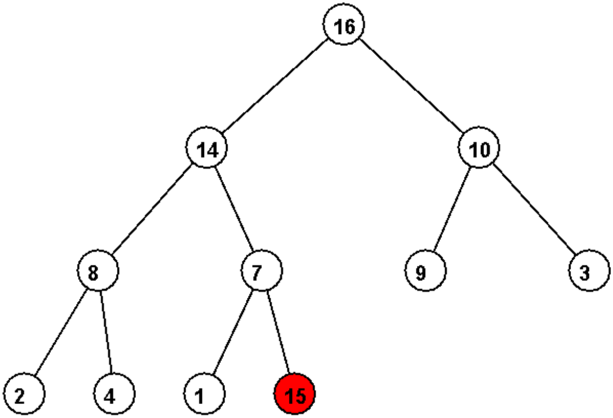
Operace nad prioritní frontou

```
procedure INSERT(A, val)
{
        Heap_Size(A) ← Heap_Size(A) + 1;
        i ← Heap_Size(A);
        while (i > 1 & A[PARENT(i)] < val)
                do {
                        A[i] ← A[PARENT(i)];
                        i ← PARENT(i);
                }
        A[i] ← val
}
```

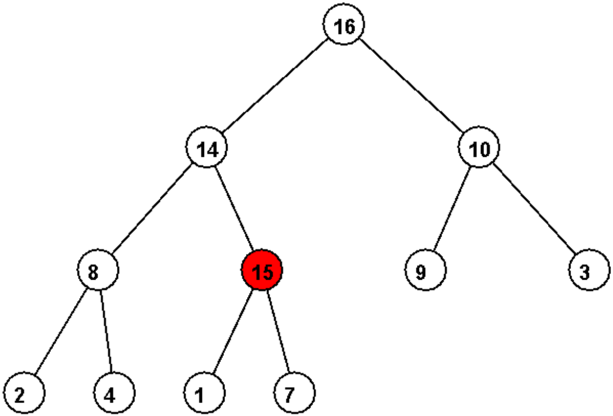
Příklad běhu operace INSERT



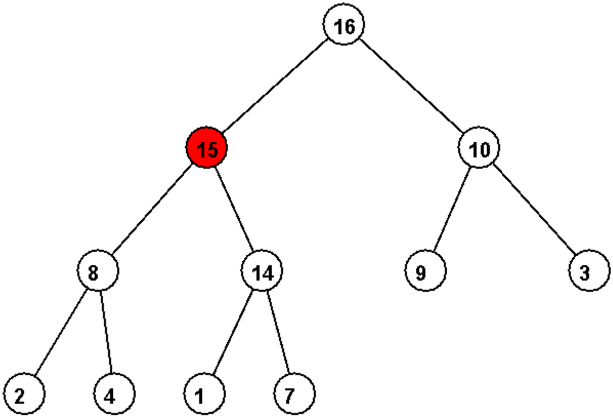
Příklad běhu operace INSERT



Příklad běhu operace INSERT



Příklad běhu operace INSERT



Překladový slovníček

binární strom	binary tree
cesta	path
halda	heap
hrana	edge
kořenový strom	rooted tree
les	forest
list	leaf
potomek	descendat
orientovaný graf	directed graph
plný strom	full tree
syn	child
poziční strom	positional tree
prioritní fronta	priority queue
předchůdce	predecessor
rodič	parent
smyčka	loop
sourozenec	sibling
uspořádaný strom	ordered tree
uzel	node
vnitřní uzel	inner node
volný strom	free tree
výškově vyvážený strom	height balanced tree
úplný strom	complete tree