

# Heap Sort

Bc. Katarína Olejková



KATEDRA INFORMATIKY  
UNIVERZITA PALACKÉHO V OLMOUCI

# Binárny strom terminológia

- Dátova štruktúra, ktorá sa skladá z uzlov a hran
- každý **uzol** má najviac dvoch **potomkov**
- **Potomok** = uzol, do ktorého vstupuje hrana (ľavý, pravý)
- **RODIČ** = uzol, z ktorého vystupuje aspoň jedna hrana
- uzol môže byť zároveň **rodič** aj **potomok**
- Najvyšší uzol = KOREŇ (nemá rodiča)
- Najspodnejšie uzly = LISTY

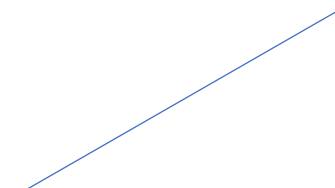



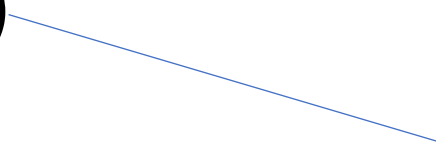
# (Binárna) Halda

- Simuluje uloženie prvkov v binárnom strome
- Dva typy: **Max-Halda**, Min-Halda
- **Max-Halda**: pre všetky uzly v strome musí platiť, že rodič  $\geq$  potomok, čiže v koreni bude najväčší prvok
- Min-Halda: analogicky (rodič  $\leq$  potomok, koreň = najmenší prvok)

# Heap Sort

- Zo vstupného poľa vytvorí Max-Haldu preusporiadaním prvkov
- Opakuje dokým halda nebude obsahovať iba jeden prvok:
  - Vymení koreňový uzol (obsahuje najväčší prvok) s posledným uzlom v halde
  - Zmenšíme veľkosť uvažovanej haldy o 1 (najväčší prvok “odstránime” lebo už je zotriedený)
  - Zaradíme nový koreňový uzol do haldy, tak aby spĺňal podmienku Max-Haldy (rodič  $\geq$  potomok, pre všetky uzly)

## Heap-Sort(A)

1. Build-Max-Heap(A) 
  2. **for**  $i \leftarrow n - 1$  **downto** 1 
  3.     swap(A[0], A[i]) 
  4.     heapsize(A)  $\leftarrow$  heapsize(A) - 1 
  5.     Max-Heapify(A, 0) 
- Zo vstupného poľa vytvorí Max-Haldu preusporiadaním prvkov
  - Opakuje dokým halda nebude obsahovať iba jeden prvok:
    - Vymení koreňový uzol (obsahuje najväčší prvok) s posledným uzlom v halde
    - Zmenší veľkosť uvažovanej haldy o 1 (najväčší prvok “odstránime” lebo už je zotriedený)
    - Zaradí nový koreňový uzol do haldy, tak aby spĺňal podmienku Max-Haldy (rodič  $\geq$  potomok, pre všetky uzly)

## Build-Max-Heap(A)

- Zo vstupného poľa vytvorí Max-Haldu preusporiadaním prvkov

### Build-Max-Heap(A)

1.  $\text{heapsize}(A) \leftarrow n$  Na začiatku uvažujeme haldu ako celé pole, veľkosť haldy = veľkosť poľa
2. **for**  $i \leftarrow \lfloor n / 2 \rfloor - 1$  **downto** 0 Listové uzly spĺňajú podmienku max-haldy, takže ich môžeme preskočiť a začať zaradzovať ďalšie uzly
3.  $\text{Max-Heapify}(A, i)$  Zaradzuje uzly do haldy, tak aby spĺňali podmienku Max-Haldy (rodič  $\geq$  potomok, pre všetky uzly)

## Max-Heapify(A, i)

- Zaradzuje uzly do haldy, tak aby spĺňali podmienku Max-Haldy (rodič  $\geq$  potomok, pre všetky uzly)

### Max-Heapify(A, i)

1.  $L \leftarrow \text{Left}(i)$
2.  $R \leftarrow \text{Right}(i)$
3. if  $L \leq \text{heapsize}(A)$  and  $A[L] > A[i]$
4.      $\text{largest} \leftarrow L$
5. else
6.      $\text{largest} \leftarrow i$
7. if  $r \leq \text{heapsize}(A)$  and  $A[r] > A[\text{largest}]$
8.      $\text{largest} \leftarrow R$
9. if  $\text{largest} \neq i$
10.      $\text{swap}(A[i], A[\text{largest}])$
11.      $\text{Max-Heapify}(A, \text{largest})$

$i$  – index nášho zaradzovaného uzlu

$L$  – index ľavého potomka nášho uzlu

$R$  – index pravého potomka nášho uzlu

$\text{largest}$  – index najväčšieho uzlu

Hľadáme najväčší uzol spomedzi nášho uzla, jeho ľavého a pravého potomka (ak potomkov má)

( $\text{heapsize}(A)$  slúži iba pre kontrolu aby sme neuvažovali za potomkov už zoradené prvky, alebo neexistujúce prvky)

Ak náš uzol nie je najväčší, tak ho s najväčším vymeníme (náš uzol sa dostane nižšie na miesto jeho potomka) a musíme ho ďalej zaradzovať

# Heap Sort - časová zložitosť

- Max-Heapify
  - V najhoršom prípade Max-Heapify zaradí prvok do listu stromu
  - výška celého stromu s  $n$  prvkami je  $O(\log n)$
  - Zložitosť v najhoršom prípade bude  $O(\log n)$
- Build-Max-Heap
  - Volá  $n / 2$  krát funkciu Max-Heapify
  - Zložitosť v najhoršom prípade bude  $O(n/2 * \log n) = O(n * \log n)$
  - Existuje lepší odhad  $O(n)$
- Celková zložitosť Heap Sort
  - Algoritmus prevedie:  $1 * \text{Build-Max-Heap}$  a  $(n - 1) * \text{Max-Heapify}$
  - $O(n) + (n - 1) * O(\log n) = O(n) + O(n - 1 * \log n) = O(n * \log n)$



# Úkol

- Simulácia algoritmu
  - Simulujte kroky algoritmu HeapSort na postupnosti  
 $A = [9, 1, 7, 6, 0, 8, 4, 5]$
- Spôsob odovzdávania – info na [GitHube](#) na konci README