EADŠ - cvičenie 2

29. septembra 2022

Opakovanie - zložitosti

Trieda	Intuitívne	Limita
$f(n) \in O(g(n))$	$f(n) \leq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty$
$f(n) \in \Omega(g(n))$	$f(n) \geq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$
$f(n) \in \Theta(g(n))$	$f(n) \approx g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0, \infty$

Opakovanie - určovanie zložitosti

- zanedbávame konštanty:
  $O(c \cdot f(n)) = O(f(n))$
  (napr. $O(3n) = O(n)$)
- berieme najväčšiu funkciu
  $O(f(n) + g(n)) = O(\max(f(n), g(n)))$
  (napr. $O(n^5 + n^2 + n) = O(n^5)$)
- poradie zložitostí:
  $\forall a > 1 : O(1) \in O(\log^a n) \in O(n^a) \in O(a^n) \in O(n!) \in O(n^n)$
  Intuitívne:
  $\forall a > 1 : 1 \leq \log^a n \leq n^a \leq a^n \leq n! \leq n^n$

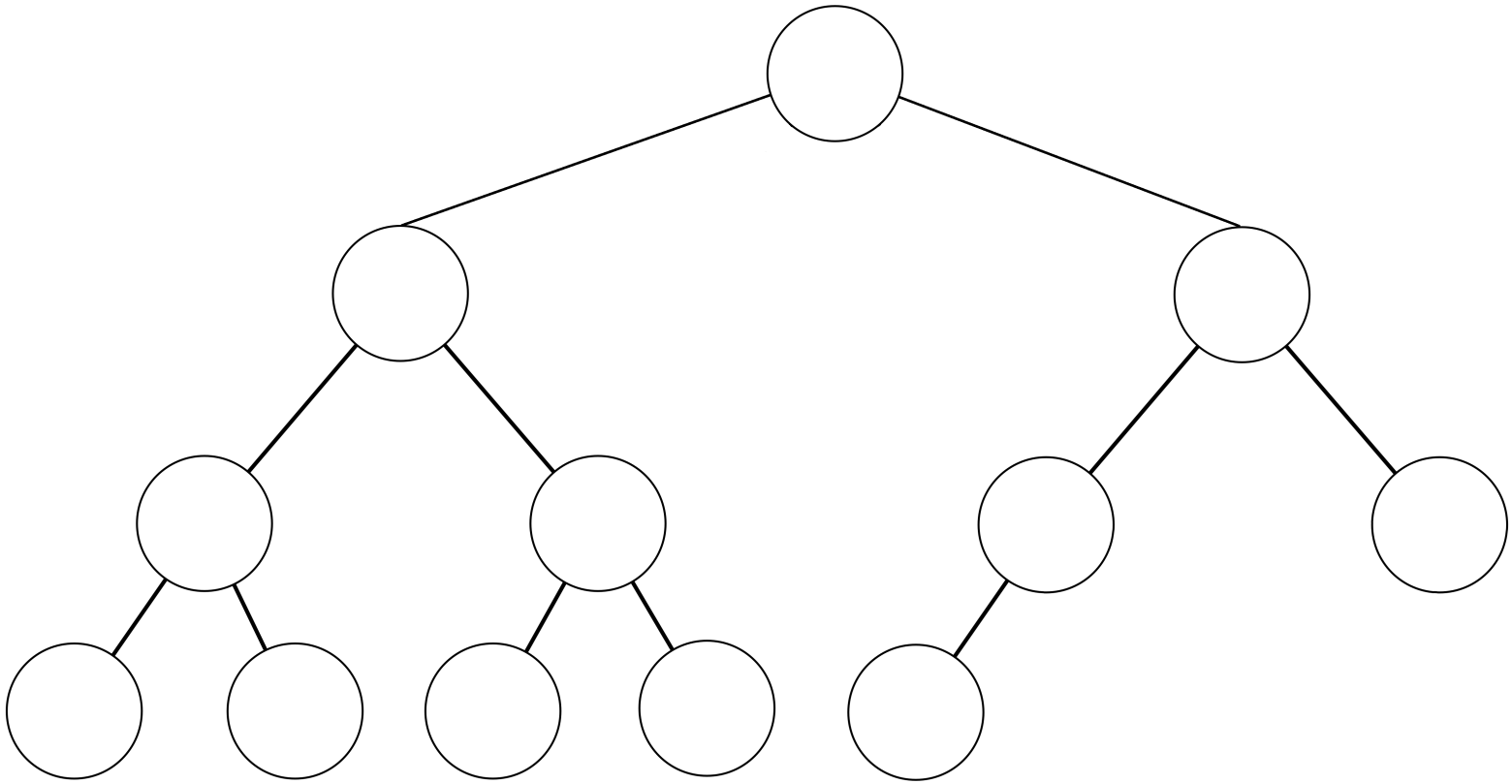
Opakovanie - čo je treba vedieť

- určovanie zložitosti z kódu
- asymptotické porovnávanie dvoch funkcií (O, Ω, Θ)

Halda - opakovanie

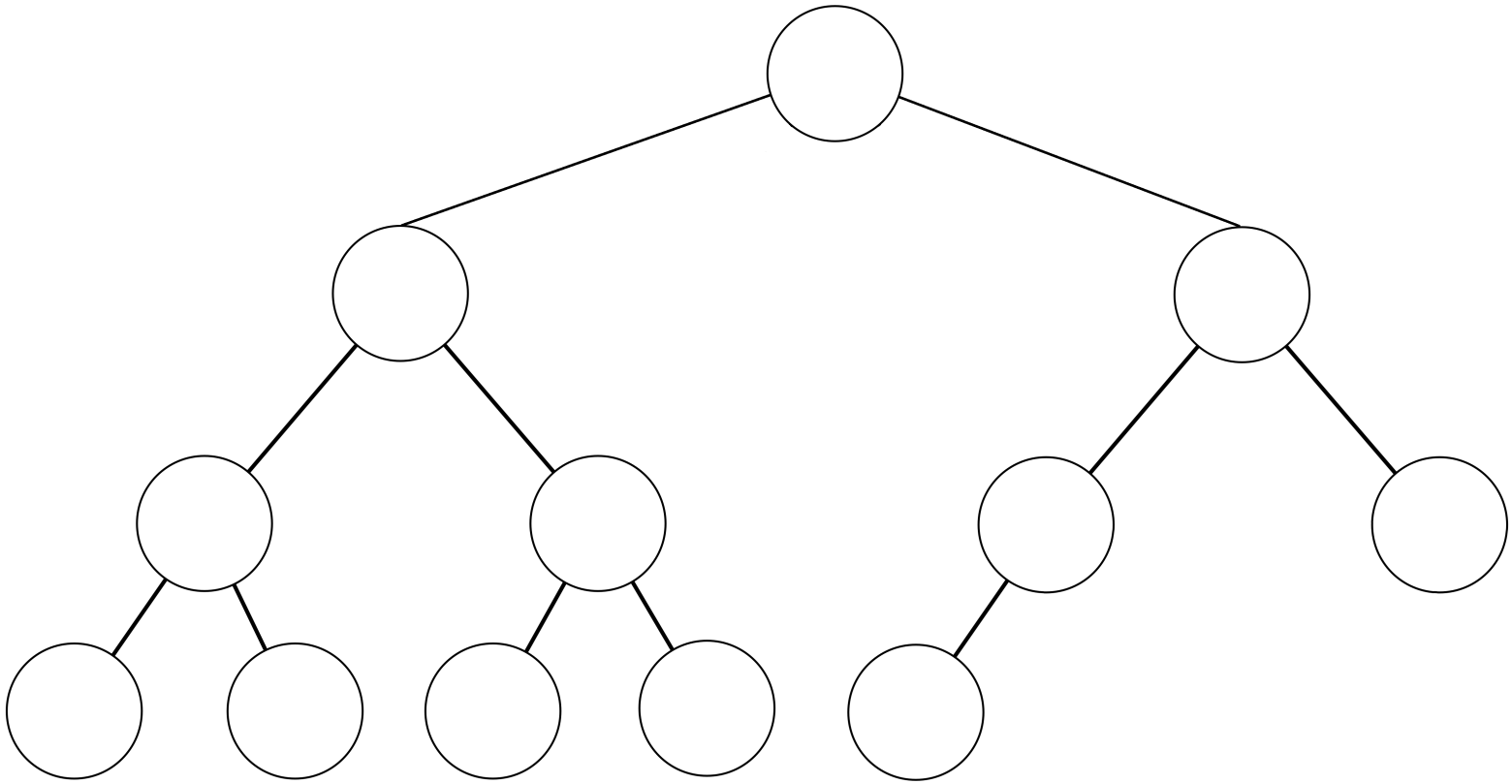
- implementácia dátovej štruktúry *priority queue*
- poloutriedený úplný binárny strom
- minimum v koreni
- rodič má menší kľúč (menšiu prioritu) ako potomok

Halda - 1



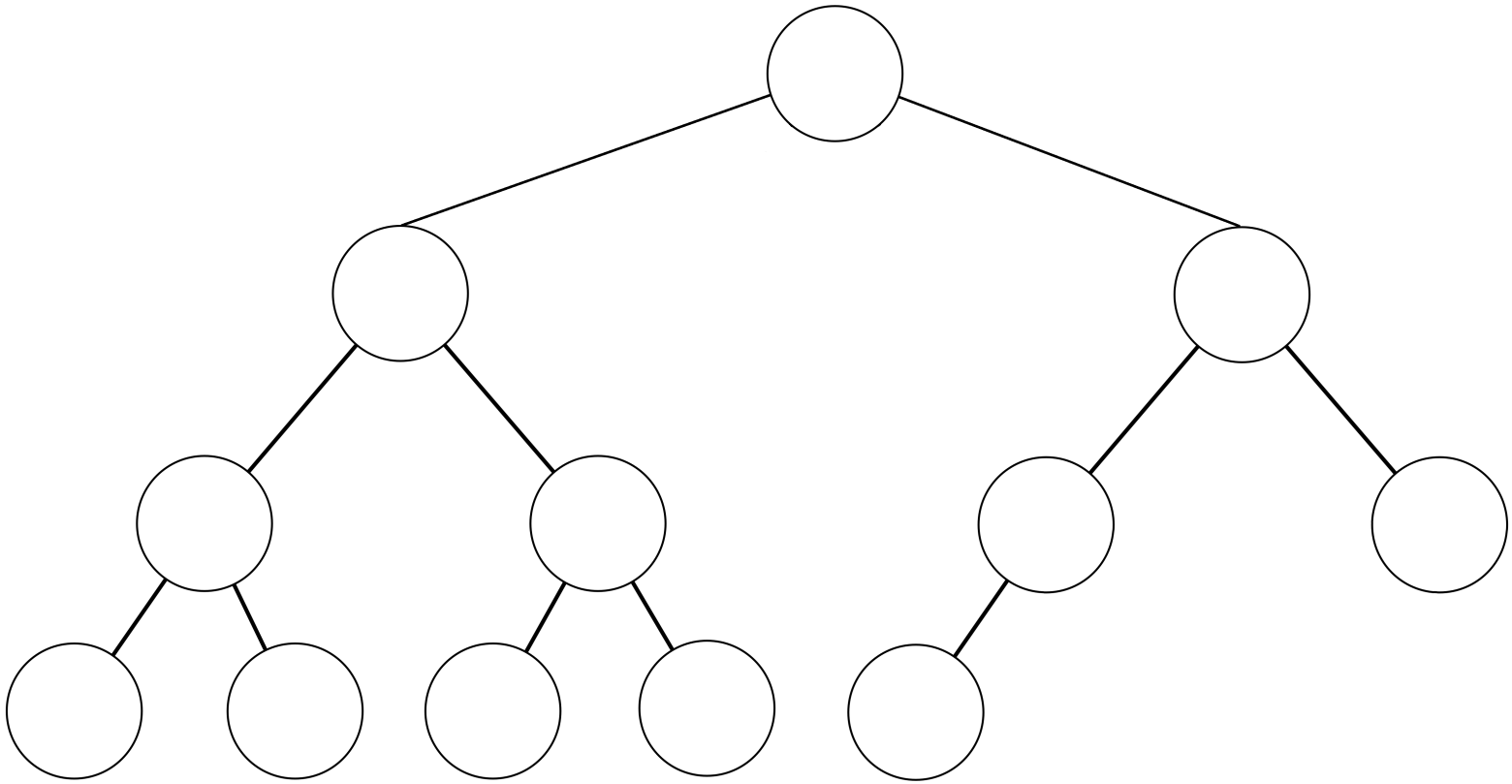
Kde sa môže nachádzať najmenší prvok (prvok s najmenším kľúčom)?

Halda - 2



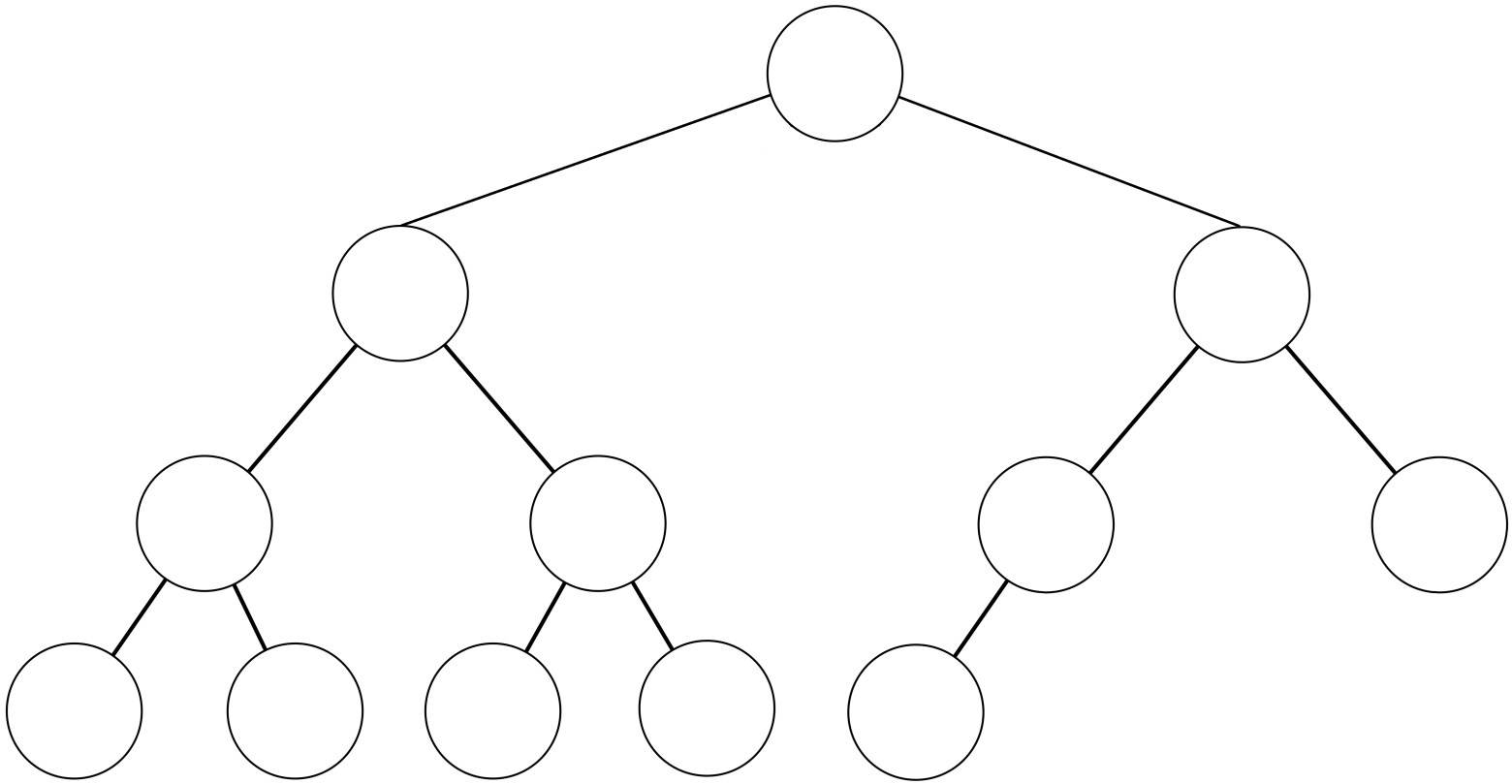
Kde sa môže nachádzať druhý najmenší prvok (prvok s druhým najmenším kľúčom)?

Halda - 3



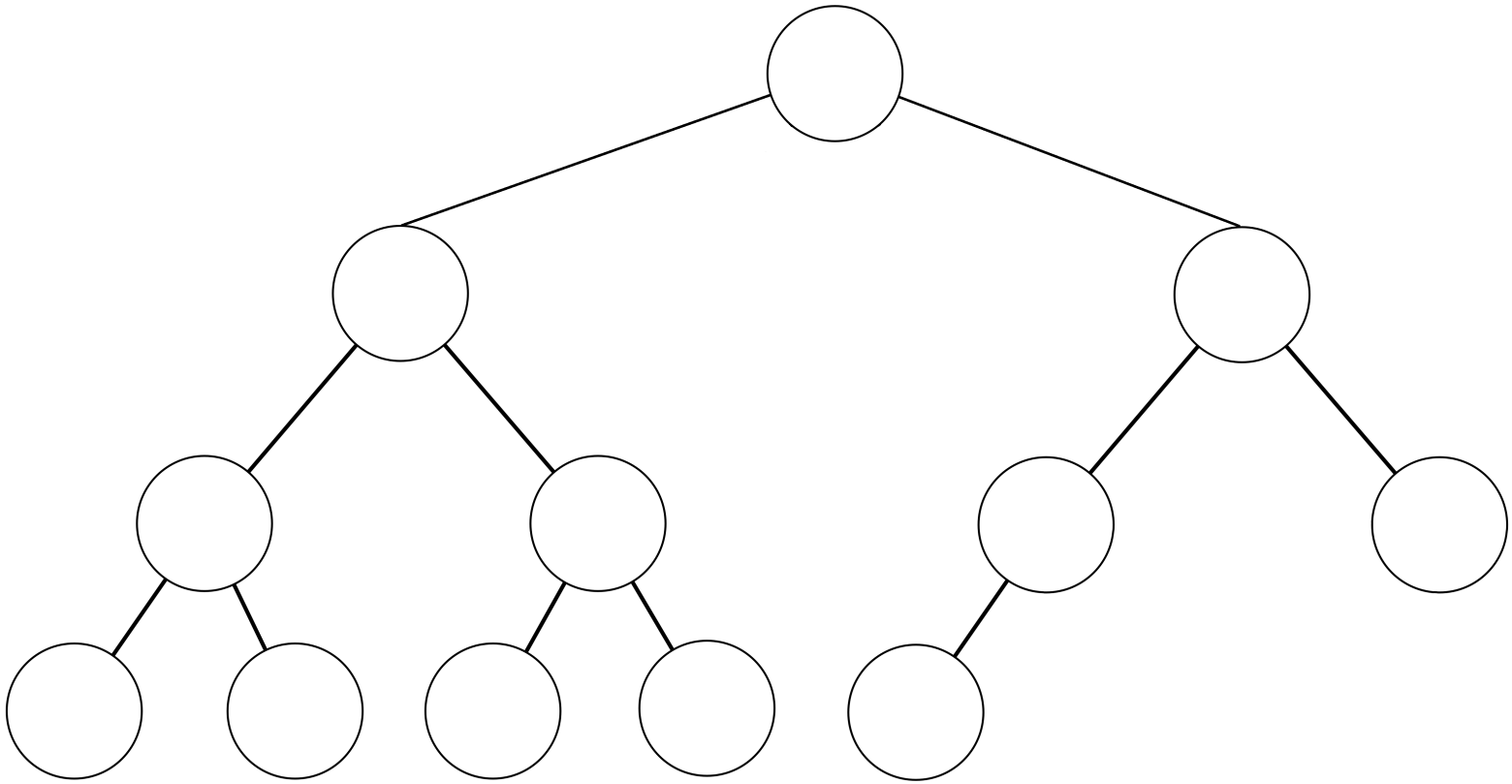
Kde sa môže nachádzať tretí najmenší prvok (prvok s tretím najmenším kľúčom)?

Halda - 4



Kde sa môže nachádzať najväčší prvok (prvok s najväčším kľúčom)?

Halda - 5

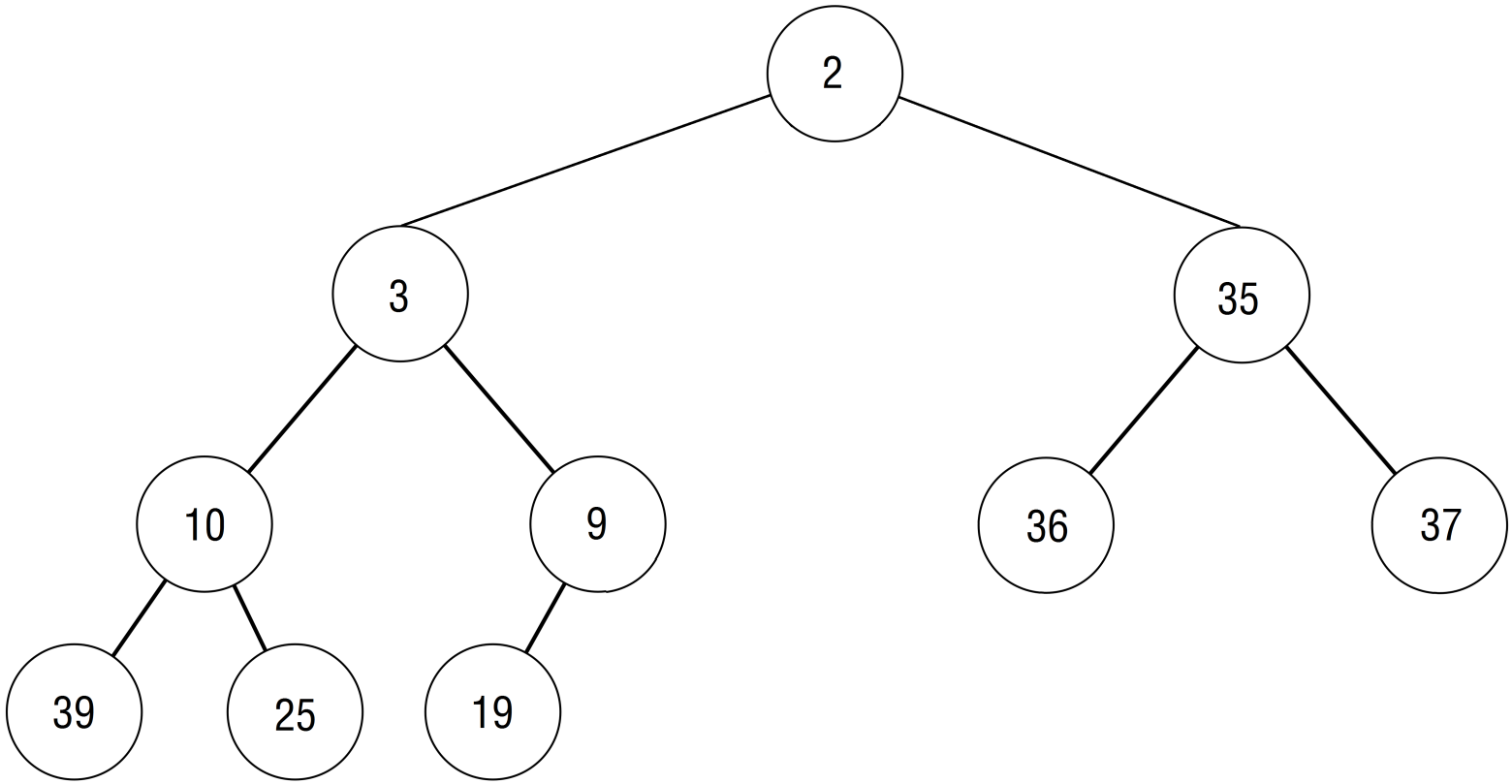


Kde sa môže nachádzať druhý najväčší prvok (prvok s druhým najväčším kľúčom)?

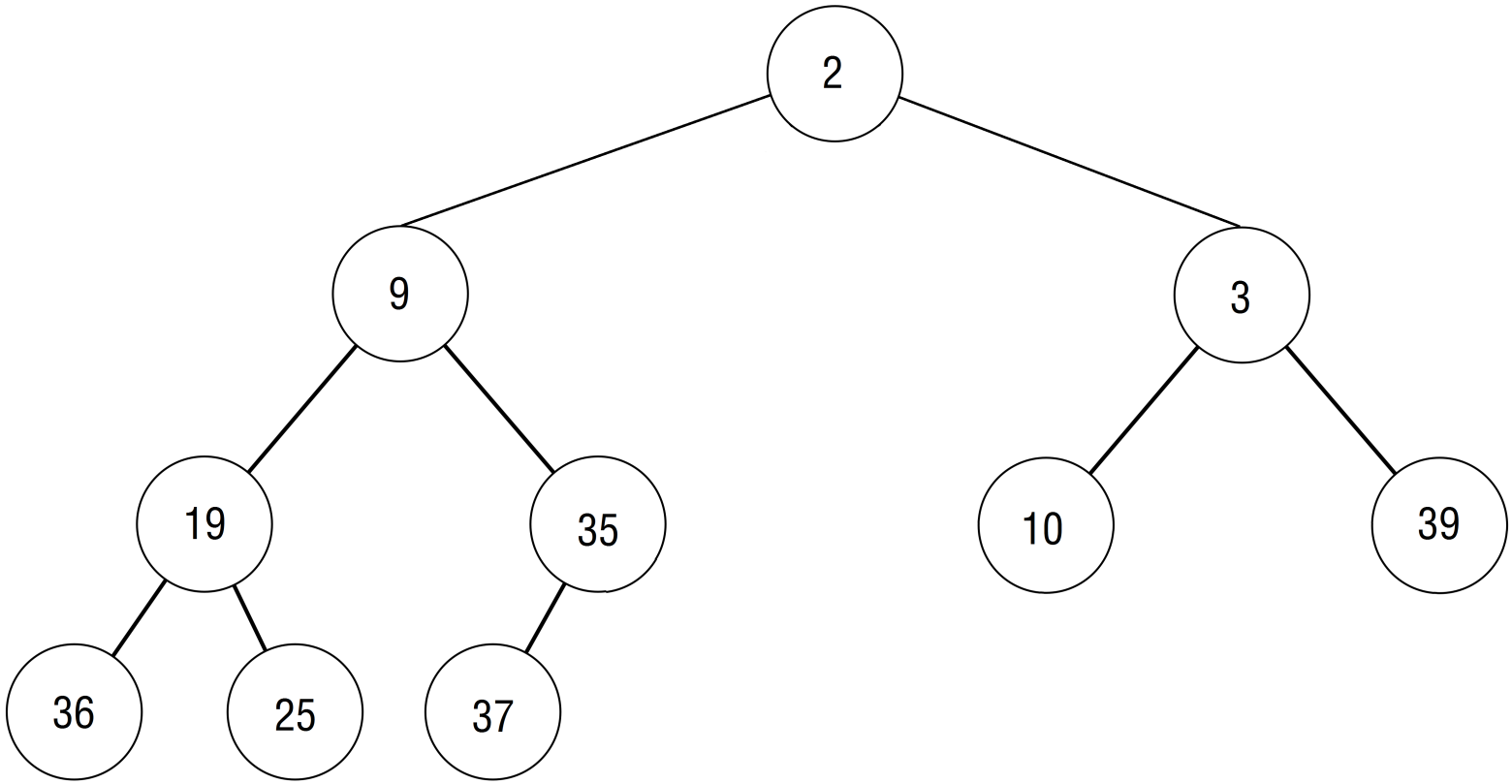
Halda - 6

Vytvorte haldu z týchto čísel: $37, 3, 35, 39, 9, 2, 36, 10, 25, 19$

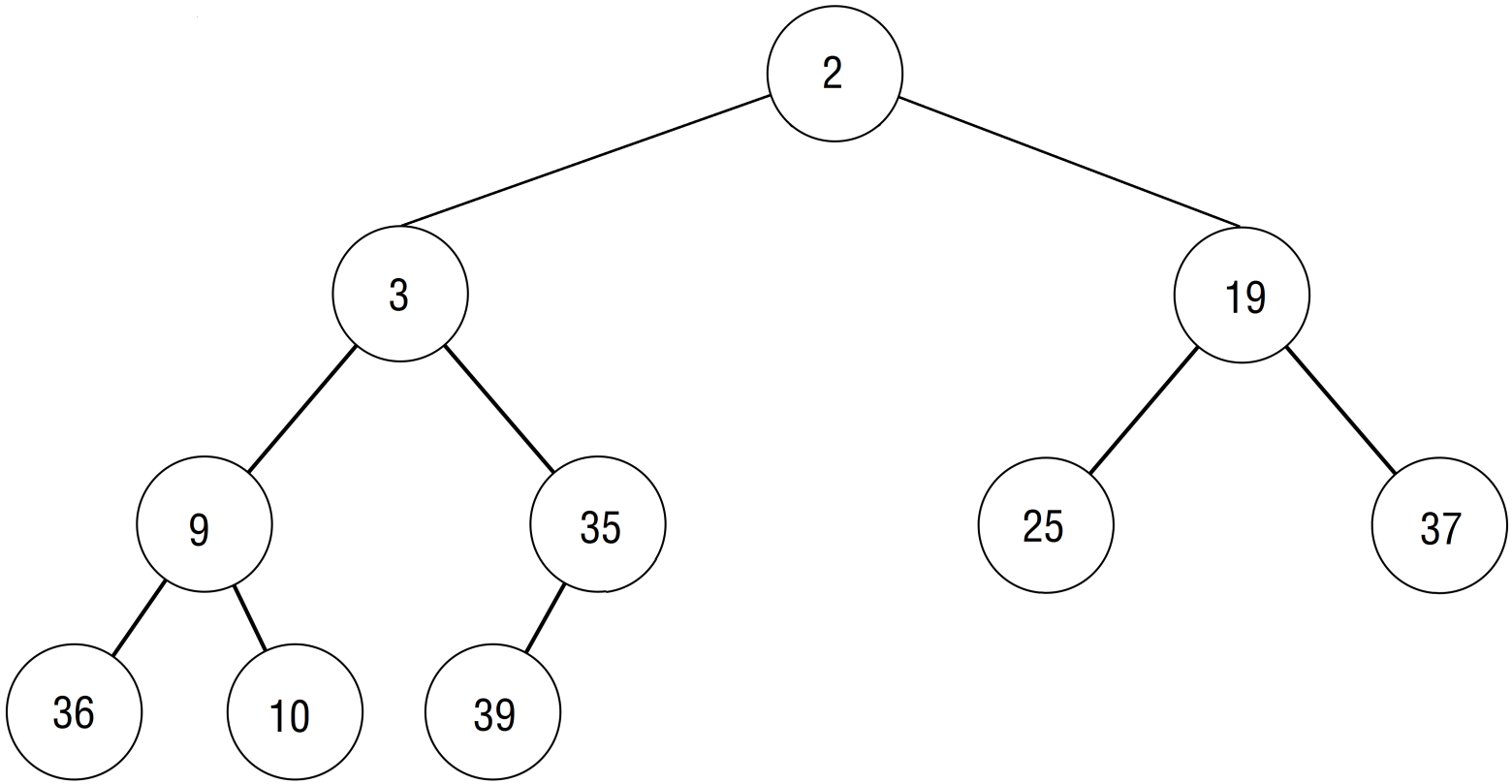
Halda - 6



Halda - 6



Halda - 6



Halda - 7

Bežné operácie:

1. `insert(x)`
2. `extractMin()`

Vedeli by ste, ako naimplementovať operáciu `delete(index)`?

Halda - 8

Halda nemusí byť implementovaná len ako binárny strom, ale pokojne aj ako d-árny. Zložitosti operácií?

Operácia	Binárna halda	d-árna halda
`insert(x)`	$O(\log n)$	?
`extractMin(x)`	$O(\log n)$	?

Halda - 8

Halda nemusí byť implementovaná len ako binárny strom, ale pokojne aj ako d-árny. Zložitosti operácií?

Operácia	Binárna halda	d-árna halda
`insert(x)`	$O(\log n)$	$O(\log_d n)$
`extractMin(x)`	$O(\log n)$	$O(d \log_d n)$

Priority queue - 1

Máme čiernu škatuľku s operáciami:

1. `insert(x)`
2. `extractMin()`

Ako by sme vedeli urobiť čiernu škatuľku, ktorá má operácie:

1. `insert(x)`
2. `extractMax()`

Priority queue - 2

Máme dátovú štruktúru (čiernu škatuľku) s operáciami:

1. `insert(x)`
2. `extractMin()`

Ako by sme vedeli urobiť čiernu škatuľku, s rovnakými operáciami, ale `extractMin` v prípade, že majú prvky rovnakú prioritu ich vracia v poradí podľa času vloženia?

Priority queue - 3

Máme (dátovú štruktúru) čiernu škatuľku s operáciami:

1. `insert((key, value))`
2. `extractMin()`

Ako by sme vedeli pomocou nej urobiť dátovú štruktúru, ktorá má operácie (môžeme použiť maximálne konštantne veľa PQ):

1. `insert(x)`
2. `extractMin()`
3. `extractMax()`

Priority queue - 4

Máme m polí, každé má n prvkov. Prvky sú vrámci jedného poľa utriedené od minimálneho po maximálny. Ako spojiť tieto polia do jedného, ktoré bude tiež utriedené ?

Priority queue - 5

Máme pole, v ktorom pozícia každého prvku je najviac o k políčok vzdialená od pozície tohto prvku, ak by pole bolo utriedené. Ako sa toto pole dá čo najrýchlejšie utriediť?

Priority queue - 6

Dané je pole čísel. Nájdi k-te najväčšie.

Praktické info - Python

Existujú dve knižnice na prácu s haldou/PQ:

1. `import heapq`
2. `from queue import PriorityQueue`

heapq

```
import heapq

halda = [8, 6, 1, 3, 4, 5]
heapq.heapify(halda)
# halda = [1, 3, 5, 6, 4, 8]



heapq.heappush(halda, 9)
heapq.heappush(halda, 15)
heapq.heappush(halda, 0)

while len(halda)!=0:
    heapq.heappop(halda)
```

PriorityQueue

```
from queue import PriorityQueue

q = PriorityQueue()

q.put(4)
q.put(2)
q.put(5)
q.put(1)
q.put(3)

while not q.empty():
    next_item = q.get()
    print(next_item)
```

Praktické info - Python

Rozdiel - `heapq` nie je thread-safe, a teda je rýchlejšia, ak vám ide o rýchlosť/efektívnosť, tak lepšia.

Praktické info - C++

```
#include <iostream>
#include <queue>
using namespace std;

int main()
{
    priority_queue<int> pq;
    pq.push(8);
    pq.push(10);
    pq.push(2);

    cout << "size: " << pq.size() << endl;
    cout << "top: " << pq.top() << endl;

    while (!pq.empty()) {
        cout << "pop() : ";
        gquiz.pop();
    }
}
```