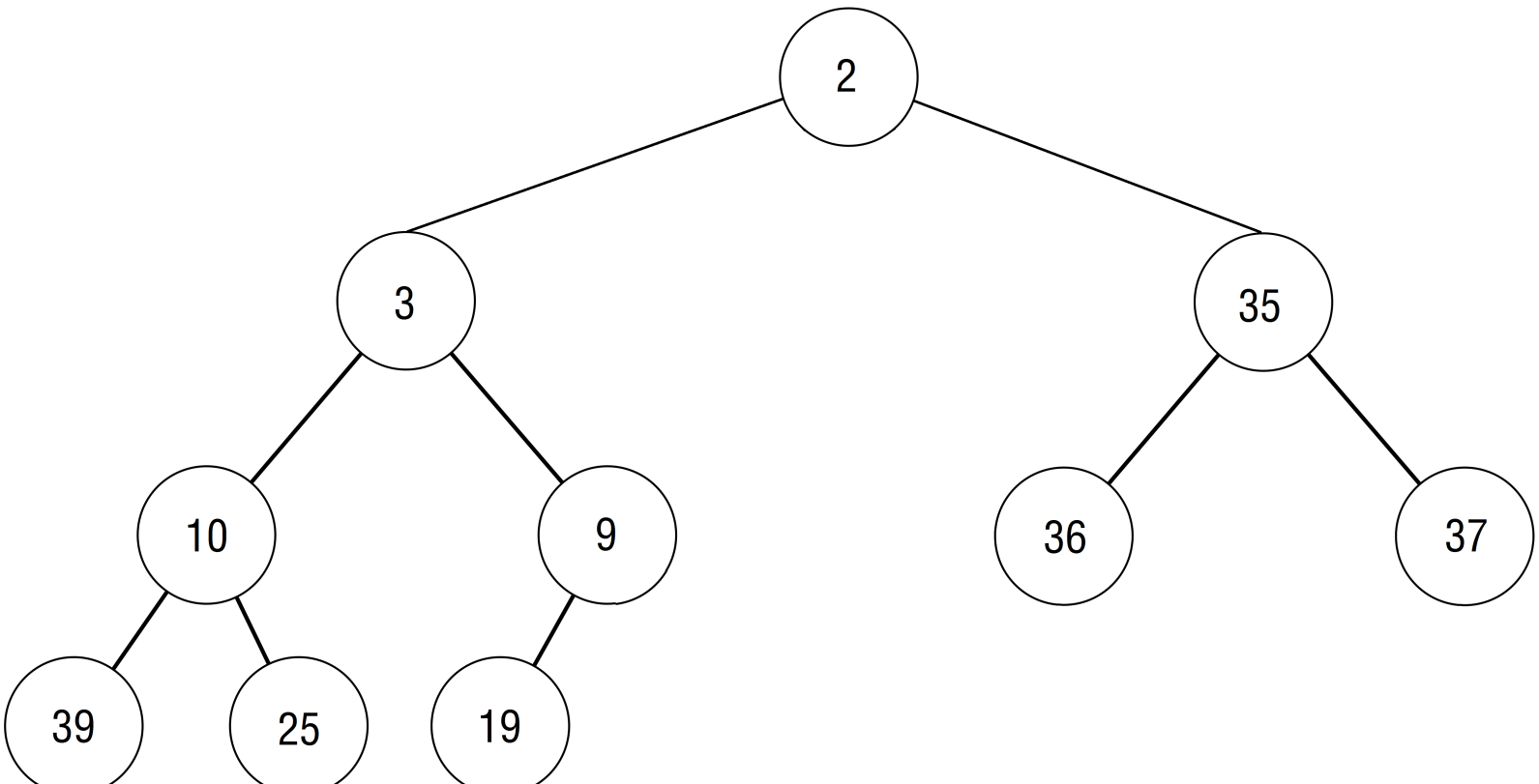
Cvičenie 02: Halda

Efektívne Algoritmy a Dátové Štruktúry
FMFI UK

Princíp haldy (1)

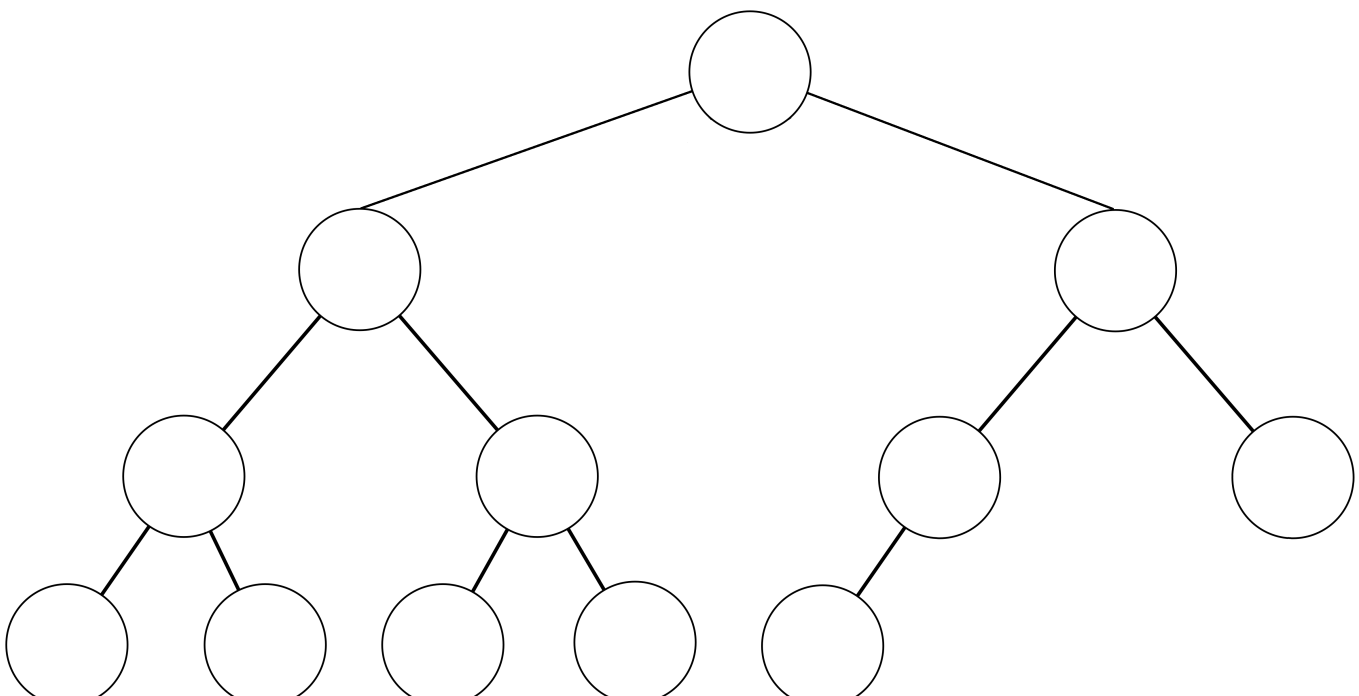
Vytvorte haldu z nasledujúcich čísiel: 37, 3, 35, 39, 9, 2, 36, 10, 25, 19.

Princíp haldy (1)



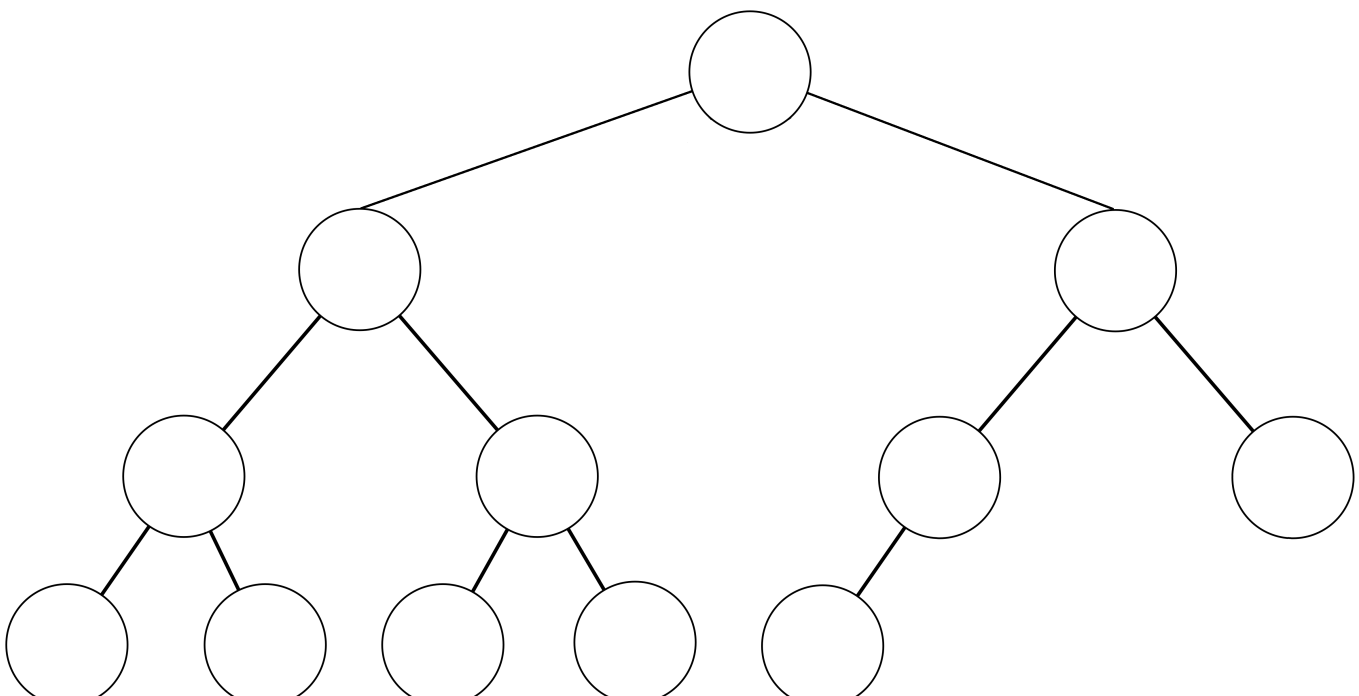
Princíp haldy (2)

Kde sa v halde nachádza najmenší prvok?



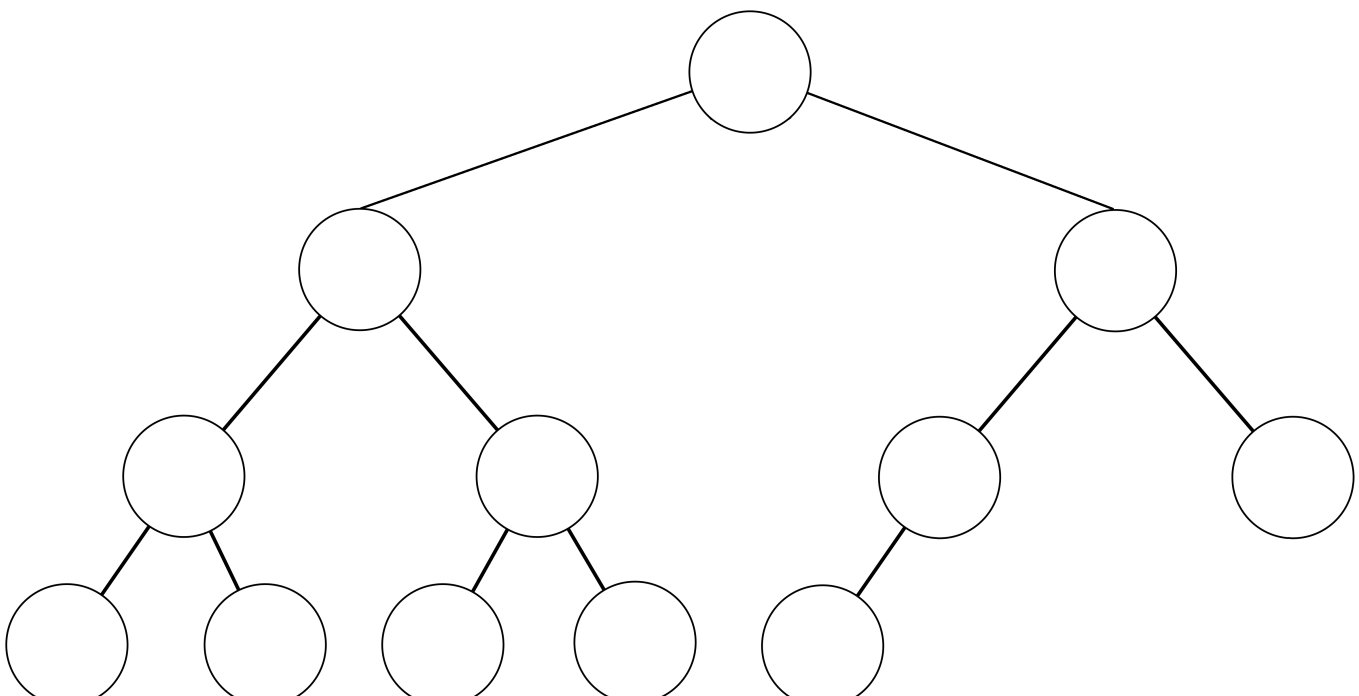
Princíp haldy (3)

Kde sa v halde nachádza 2. najmenší prvok?



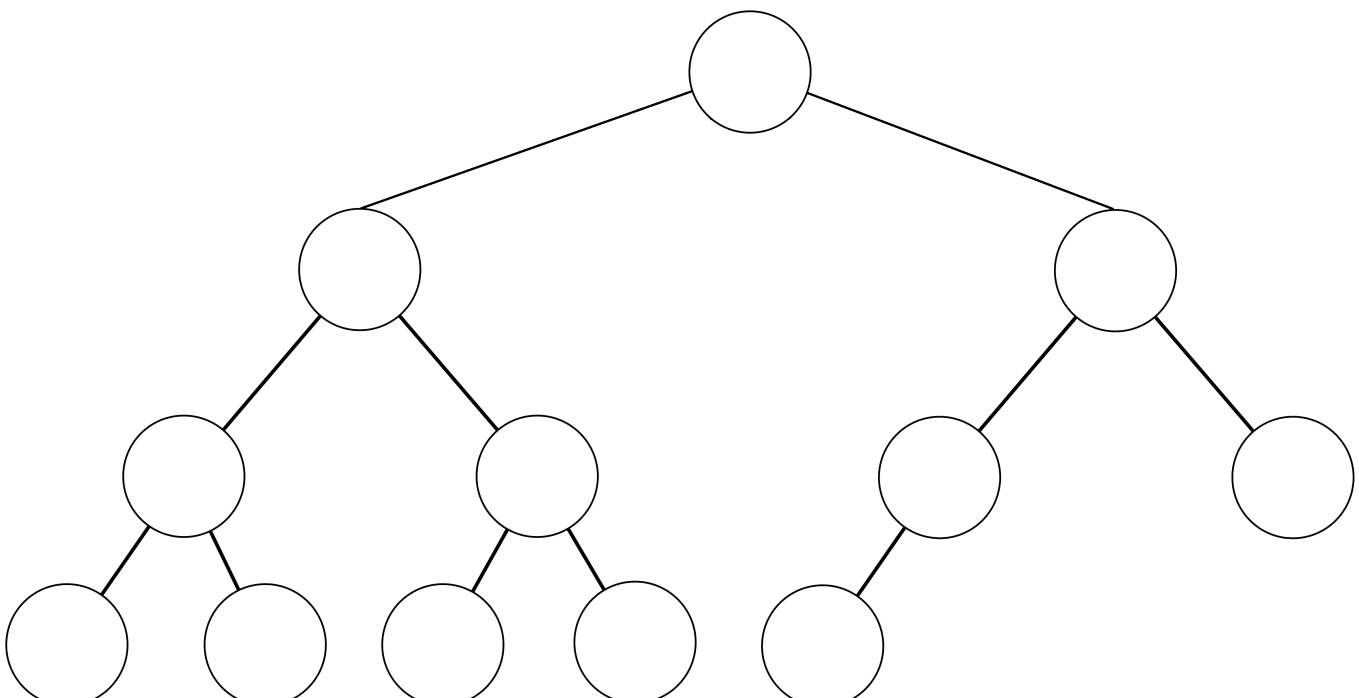
Princíp haldy (4)

Kde sa v halde nachádza 3. najmenší prvok?



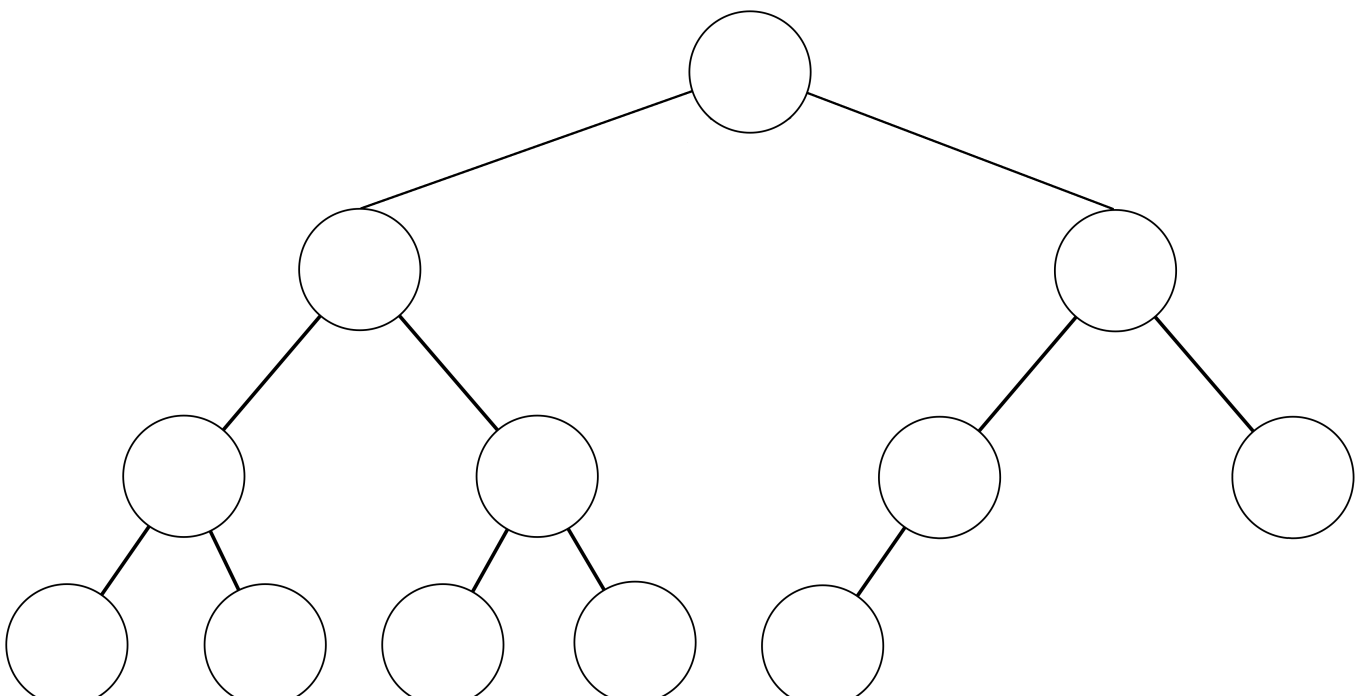
Princíp haldy (5)

Kde sa v halde nachádza najväčší prvok?



Princíp haldy (6)

Kde sa v halde nachádza 2. najväčší prvok?



Modifikácie haldy (1)

Uvažujme prvok, ktorý je uložený v poli haldy na pozícii i. Ako by ste ho odstránili z haldy?

Modifikácie haldy (2)

Vieme tento postup uplatniť aj na implementáciu všeobecnej funkcie `delete(x)`, ktorá z haldy odstráni prvok s hodnotou `x`?

Modifikácie haldy (3)

Ako sa zmení časová zložitosť operácií, ak by sme haldu implementovali pomocou d-árneho stromu namiesto binárneho?

Modifikácie haldy (3)

```
SiftUp(node):
  if (node does not have a parent) done!
  if (node.parent.value > node.value)
    swap(node.value, node.parent.value)
    SiftUp(node)
```

	Binárna halda	d-árna halda
`insert`	$\log n$	$\log_d n$

Modifikácie haldy (3)

```
Heapify(node):
  if (not node.left and not node.right) done!!
  else
    if (node.right and node.right.value < node.left.value)
      k:=node.right
    else
      k:=node.left

    if (node.value <= k.value) done!!
    else
      swap(node.value,k.value);
      Heapify(k);
```

Modifikácie haldy (3)

```
Heapify(node):
  if (node does not have any child) done!!
  else
    k := 0
    for (child in node.children)
      if (k == 0 or child.value < k.value)
        k := child

    if (node.value <= k.value) done!!
    else
      swap(node.value,k.value);
      Heapify(k);
```

Modifikácie haldy (3)

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

Abstraktný dátový typ prioritná fronta (1)

Máme k dispozícii abstraktný dátový typ prioritná fronta.

Ako pomocou neho implementujeme maximovú prioritnú frontu (ktorá má namiesto `extractMin` operáciu `extractMax`)?

Abstraktný dátový typ prioritná fronta (2)

Máme k dispozícii abstraktný dátový typ prioritná fronta.

Vieme pomocou neho implementovať prioritnú frontu s funkciou `delete(x)`?

Abstraktný dátový typ prioritná fronta (3)

Máme k dispozícii abstraktný dátový typ prioritná fronta.

Vieme pomocou nej urobiť abstraktný dátový typ s operáciami: `insert`, `extractMin`, `extractMax`?

m-cestný merge

Máme m utriedených polí, s celkovým počtom prvkov n. Spojte tieto polia do jedného utriedeného poľa.

m-cestný merge

```
merge(a, b):
  create new array M of size n

  k := 1
  m := 1
  for i := 1 ... n
    if a[k] < b[m]
      M[i] := a[k]
      k++
    else
      M[i] := b[m]
      m++

  return M
```

Skoroutriedené pole

Máme skoroutriedené pole, v ktorom sa každý prvok nachádza vo vzdialenosti najviac k od svojej pozície v utriedenom poli. Dotrieďte skoroutriedené pole.

Príklad: $k = 1$, pole `[2, 1, 3, 5, 4]`.

k-ty najväčší

V danom poli nájdite k-ty najväčší prvok.

Knižničné implementácie: Python

```
import heapq

halda = []

heapq.heappush(halda, 8)
heapq.heappush(halda, 10)
heapq.heappush(halda, 2)

while halda:
    minimum = heapq.heappop(halda)
```

Knižničné implementácie: C++

```cpp
#include <queue>

int main() {
  std::priority_queue<int> halda;
  halda.push(8);
  halda.push(10);
  halda.push(2);

  while (!halda.empty()) {
    int minimum = halda.pop();
  }
}
```