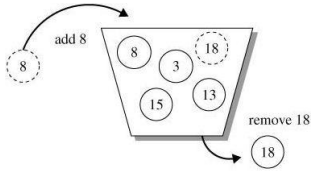
Prioritný front

binárna halda

1

Prioritný front

- Prioritný front je množina prvkov, ktorým je pridelená priorita – je ich možné porovnávať. Prvky je možné vkladať v akomkoľvek poradí s rôznou prioritou avšak pri výbere sa vyberá vždy prvok s najvyššou prioritou.



2

Prioritný front

- každý prvok má prioritu
- prioritný front - front zoradený podľa priority
- nie FIFO, ale vyberie sa prvok s najvyššou prioritou
- príklady:
  - súbory na tlač čakajúce v rade
  - procesy čakajúce na preprocesor

3

Prioritný front - operácie

- Insert (S, x)
  - vloženie prvku x do množiny (prioritného frontu) S
- Maximum (S)
  - vrátenie prvku množiny S s najväčším kľúčom (prvku prioritného frontu s najväčšou prioritou)
- Extract-Max(S)
  - odstránenie a vrátenie prvku S s najväčším kľúčom

4

Prioritný front pomocou spájaného zoznamu

- Pridávanie prvkov na začiatok zoznamu O(1)
- Vymazávanie prvkov - nájdenie prvku s najväčšou prioritou, ten sa vymaže O(n)

5

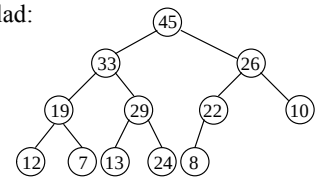
Prioritný front pomocou BVS

- Pridávanie prvkov - zaradenie do stromu podľa priority (priorita je kľúč)
- Vymazávanie prvkov - vymazanie prvku s najväčšou prioritou, t.j. najpravejší vrchol
- Obe operácie - O(log n) - výhodnejšie ako pri spájanom zozname
- Nepotrebujeme všetky vlastnosti BVS
- len také, aby sme našli prvok s najväčšou prioritou

6

Prioritný front pomocou binárnej haldy

- Binárna halda je úplný binárny strom, pre ktorý platí, že hodnota kľúča je väčšia alebo rovná hodnotám kľúčov jeho nasledovníkov
- Príklad:



7

Binárna halda

- Binárna halda má menej striktné pravidlá na umiestnenie prvkov ako BVS
- Neplatí, že ľavý podstrom obsahuje prvky s nižšími hodnotami kľúčov ako pravý podstrom
- platí haldová vlastnosť:
  - A[PARENT(i)] >= A[i] pre všetky vrcholy i okrem koreňa
- dôsledok: koreň stromu má však vždy najväčšiu hodnotu (>= ako ostatné uzly)

8

Prioritný front pomocou binárnej haldy

- Insert (S, x)
  - Heap-Insert (A, x)
- Maximum (S)
  - Heap-Maximum (A)
- Extract-Max(S)
  - Heap-Extract-Max(A)

9

Binárna halda - implementácia vektorom

- Koreň stromu na 1. pozícii heap[1]
- Nasledovníky vrchola zapísaného na i-tej pozícii vektora, ak existujú:
  - left(i) = 2 * i
  - right(i) = 2 * i + 1
- heap[i..j], kde i>=1, je binárna halda práve vtedy, ak každý prvok nie je menší ako jeho nasledovníky.

10

počítanie indexov

- PARENT(i) // index predchodcu vrchola v i
  - return $\lfloor i / 2 \rfloor$ // floor(i/2)
- LEFT(i) // index ľavého nasledovníka vrchola v i
  - return 2*i
- RIGHT(i) // index pravého nasledovníka vrchola v i
  - return 2*i+1

11

vrátenie prvku s najväčšou prioritou

- Heap-Maximum (A)
  - A[1]

O(1)

12

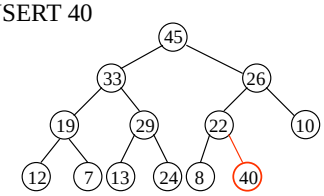
Pridanie prvku do binárnej haldy

- Vytvorí sa nový vrchol na najnižšej úrovni
- Ak hodnota kľúča nového vrchola <= hodnota predchodcu - koniec
- Ak je väčšia, vymení sa nový vrchol so svojím predchodcom
- Ak je hodnota nového vrchola väčšia ako nový predchodca, vymení sa aj s ním, … až pokým nie je strom opäť haldou

13

Pridanie prvku do binárnej haldy - príklad (1)

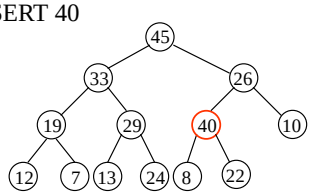
INSERT 40



14

Pridanie prvku do binárnej haldy - príklad (2)

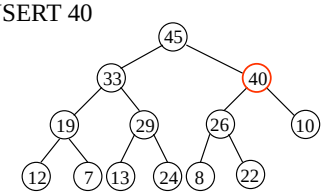
INSERT 40



15

Pridanie prvku do binárnej haldy - príklad (3)

INSERT 40



16

Pridanie prvku do binárnej haldy - implementácia

```
Heap-INSERT(heap, key)
   heap-size (heap) = heap-size(heap) + 1
   i = heap-size (heap)
   while i > 1 and heap[PARENT(i)] < key
        do heap[i] = heap[PARENT(i)]
             i = PARENT(i)
   heap[i] = key
```

O(log n)

17

Odstránenie najväčšieho prvku z binárnej haldy (1)

- Odstráni sa koreň haldy, hodnotu kľúča koreňa označíme R
- Odstráni sa najpravejší vrchol na najnižšej úrovni (jeho hodnotu označme P)
- Pokúsime sa vyplniť hodnotu koreňa hodnotou P
- Ak hodnota P >= R, P sa zapíše do koreňa
- Inak presunieme potomka koreňa s väčšou hodnotou do koreňa,

18

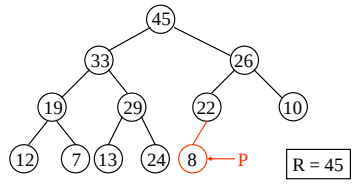
Odstránenie najväčšieho prvku z binárnej haldy (2)

- R= hodnota presunutého vrchola
- Vzniká voľné miesto, kam sa opäť pokúšame umiestniť P (ak hodnota P >= R)
- Takto pokračujeme až pokým nastane hodnota P >= R, kde R je hodnota posledného presunutého vrchola alebo posledný presunutý vrchol je list - tam presunieme P

19

Odstránenie najväčšieho prvku z binárnej haldy - príklad (1)

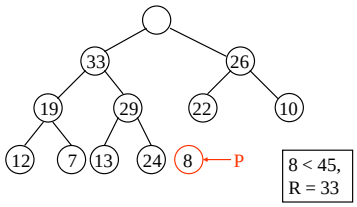
Heap-EXTRACT-MAX



20

Odstránenie najväčšieho prvku z binárnej haldy - príklad (2)

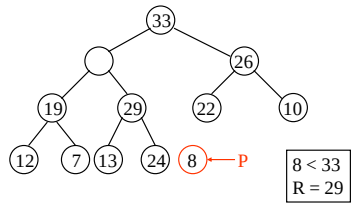
Heap-EXTRACT-MAX



21

Odstránenie najväčšieho prvku z binárnej haldy - príklad (3)

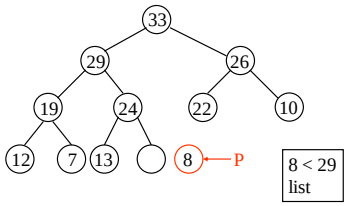
Heap-EXTRACT-MAX



22

Odstránenie najväčšieho prvku z binárnej haldy - príklad (4)

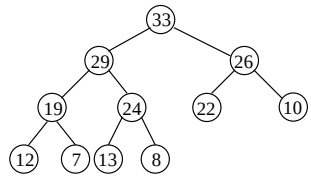
Heap-EXTRACT-MAX



23

Odstránenie najväčšieho prvku z binárnej haldy - príklad (5)

Heap-EXTRACT-MAX



24

Odstránenie najväčšieho prvku z binárnej haldy - implementácia

```
Heap-EXTRACT-MAX(heap)
   if heap-size(heap) < 1
        then error
   max = heap[1]
   heap[1] = heap[heap-size(heap)]
heap-size(heap) = heap-size(heap) -1
   HEAPIFY(heap, 1)
   return max
```

O(log n)

HEAPIFY - implementácia (1)

```
HEAPIFY(heap, i)
   lavy = left(i)
   pravy = right(i)
   if lavy <= heap-size(heap) and heap[lavy] > heap[i]
        then largest = lavy
        else largest = i
% pokračovanie
```


HEAPIFY - implementácia (2)

```
   if pravy <= heap-size(heap) and heap[pravy] >
                                        heap[largest]
        then largest = pravy
   if largest <> i
        then exchange (heap[i], heap[largest])
             HEAPIFY(heap, largest)
```

O(log n)

Vytvorenie haldy

z vektora heap[1..n], kde n=length(heap)

všetky prvky v podvektore heap[(⌊n/2⌋+1..n] sú listy a teda aj 1-prvkové haldy

```
BUILD-HEAP(heap)
   heap-size(heap) = length(heap)
   for i = ⌊length[heap] / 2⌋ downto 1
        do HEAPIFY(heap, i)
```

O(n)

Cvičenie

- Implementujte prioritný rad (binárnu haldu) dynamicky.

  Pomôcka:

```
typedef struct prioritny_rad {
        int hodnota;
        struct prioritny_rad *lavy, *pravy;
} PRIORITNY_RAD;
```