

Aula 3

- Binary Heaps
- Heap Sort
- Priority Queues
- Sorting in linear time
 - Counting Sort
 - Radix Sort

Algoritmo 1 [Max-Heapify / Min-Heapify]

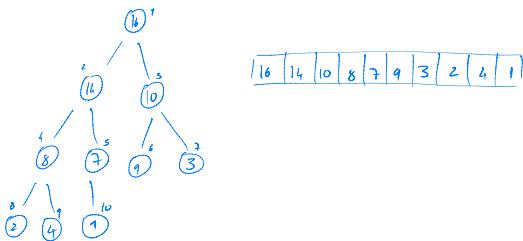
Um array $A[1..n]$ diz-se um max-heap se cessa se

$$A[\text{Parent}(i)] \geq A[i]$$

$$\text{onde Parent}(i) = \lfloor \frac{i}{2} \rfloor.$$

Um array $A[1..n]$ diz-se um min-heap se cessa se

$$A[\text{Parent}(i)] \leq A[i]$$



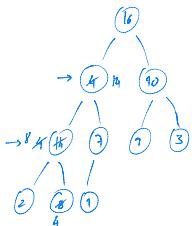
$$\text{Left}(i) = 2 \times i$$

$$\text{Right}(i) = 2 \times i + 1$$

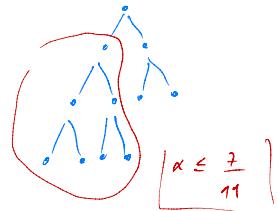
Algoritmo 1 [Max-Heapify]

$\text{Max-Heapify}(A, i)$

- O algoritmo assume \bar{g} as árvores e \bar{g} é maior em $\text{left}(i)$ e $\text{right}(i)$
- seu max-heaps, mas \bar{g} $A[i]$ falso seu menor do que $A[\text{left}(i)]$
- e $A[\text{right}(i)]$.



$$\begin{aligned} & \text{Complexidade} \\ & - T(n) = T(\alpha n) + O(1) \end{aligned}$$



$\text{Max-Heapify}(A, i)$

$$l = \text{left}(i)$$

$$r = \text{right}(i)$$

$$\max = i$$

$$\text{if } l \leq A.\text{size} \text{ and } A[l] > \max \text{ max} = l;$$

$$\text{if } r \leq A.\text{size} \text{ and } A[r] > \max \text{ max} = r;$$

$$\text{if } (\max != i)$$

swap(A, i, max)

$\text{Max-Heapify}(A, \max)$

- O pior caso acontece quando a última nível da árvore está meio cheio.

$$\alpha \leq \frac{2^h - 1}{2^{h+1} - 1} = \frac{2^h (2 - 1/2^{h-1})}{2^{h+1} (2 - 1 - 1/2^{h-1})} = \frac{2 - 1/2^{h-1}}{3 - 1/2^{h-1}}$$

$$- T(n) = T\left(\frac{2^h}{3}\right) + O(1)$$

$$a=1 \quad b=3/2 \quad d=0$$

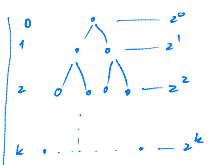
$$\log_{3/2} 1 = 0 \quad O(\log n)$$

Lema 1 [Altura de um heap]

A altura de um heap com n elementos é $\lceil \lg n \rceil$.

Prova:

- Quantas nós tem uma árvore binária completa de altura h .



- TPC: N° máximo e mínimo de elementos de um heap com altura h .

$$\text{mínimo: } 2^h$$

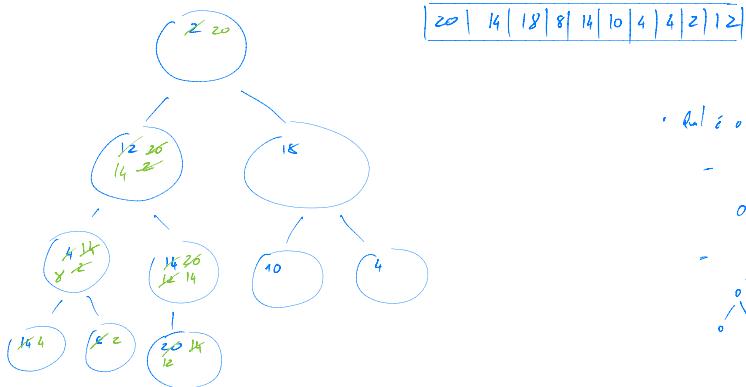
$$\text{máximo: } 2^{h+1} - 1$$

$$\begin{aligned} & 2^h \leq n < 2^{h+1} \\ \Leftrightarrow & h \leq \lceil \lg n \rceil < h+1 \\ \Leftrightarrow & \lceil \lg n \rceil = h \end{aligned}$$

(2)

• Buid-Max-Heap (A) - intuição (task 1 - 2007/2008)

1	2	3	4	5	6	7	8	9	10
2	12	18	4	14	10	4	14	8	20



Lema 2 [O primeiro nó com filhos]

Dado um arranjoado A de n elementos, todos os nós de índice $\leq \lfloor \frac{n}{2} \rfloor$ têm filhos e todos os nós de índice $> \lfloor \frac{n}{2} \rfloor$ não têm filhos.

Prova:

- Suponha que $i \leq \lfloor \frac{n}{2} \rfloor$, segue que $2i \leq \lfloor \frac{n}{2} \rfloor \times 2 \leq n$
 $\text{left}(i) \leq n \rightarrow$ de onde segue que o filho esquerdo de i está no heap.

- Suponha que $i > \lfloor \frac{n}{2} \rfloor$

$$\begin{aligned} \text{• } n \text{ é par: existe } m \text{ tal que } n = 2m \\ i > \lfloor \frac{n}{2} \rfloor \Leftrightarrow i > \lfloor \frac{2m}{2} \rfloor \Leftrightarrow i > m \\ \Leftrightarrow 2i > 2m \\ \Leftrightarrow 2i > n \\ \Leftrightarrow \text{left}(i) > n \end{aligned}$$

$$\begin{aligned} \text{• } n \text{ é ímpar: existe } m \text{ tal que } n = 2m+1 \\ i > \lfloor \frac{n}{2} \rfloor \Leftrightarrow i > \lfloor \frac{2m+1}{2} \rfloor \Leftrightarrow i > \lfloor m + \frac{1}{2} \rfloor \\ \Leftrightarrow i > m \\ \Leftrightarrow 2i > 2m \\ \Leftrightarrow 2i \geq 2m+1 \\ \Leftrightarrow 2i > 2m \vee \underbrace{2i = 2m+1}_{\text{falso}} \\ \Leftrightarrow \text{left}(i) > n \end{aligned}$$

Algoritmo 2 [Build-Max-Heap]

Build-Max-Heap (A)

$$k = \lfloor A.\text{size}/2 \rfloor$$

for $i \leftarrow k$ to 1Max-Heapify (A, i)

Complexidade [Primeira Aproximação]:

$$O(n \lg n)$$

Complexidade [Tight Bound]

$$T(n) = \sum_{h=0}^{\lfloor \log_2 n \rfloor} \text{trees}(n, h) \times O(h) = \sum_{h=0}^{\lfloor \log_2 n \rfloor} \text{trees}(n, h) \times O(h)$$

A questão é:

- Conseguimos majorar $\text{trees}(n, h)$?

(3)

Lema 3 [Árvores de Altura h]

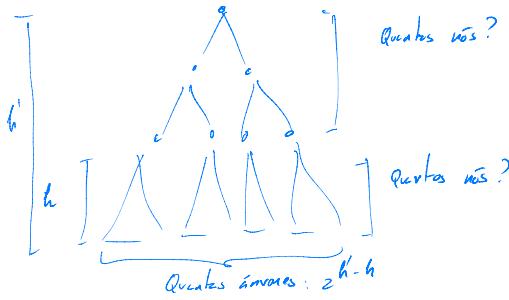
Dado um heap de tamanho n , o número de árvores de altura h contidas no heap, $\text{trees}(n, h)$ é no máximo: $\left\lceil \frac{n}{2^{h+1}} \right\rceil$.

Prova.

- Fazemos a prova para um heap completo, o número de árvores de altura h é inferior ao número de árvores de altura h de um heap completo.

A discutir com o Alexandre

$$n = 2^{h+1} - 1$$



$$\begin{aligned} 2^{h-h} &= \frac{2}{2} \times 2^{(h-h)} \\ &= \frac{2^{h+1} - h}{2} \\ &= \frac{2^{h+1}}{2^{h+1}} = \frac{2^{h+1} - 1 + 1}{2^{h+1}} = \frac{n+1}{2^{h+1}} \\ &= \left\lceil \frac{n}{2^{h+1}} \right\rceil \end{aligned}$$

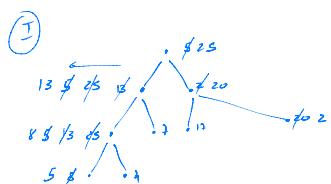
[Complexidade BuildMaxHeap]

$$\begin{aligned} T(n) &= \sum_{h=0}^{\lfloor \lg n \rfloor} \text{trees}(n, h) \times O(h) \\ &\leq \sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) \\ &= O\left(n \cdot \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) \\ &\leq O\left(n \cdot \sum_{h=0}^{\infty} \frac{h}{2^h}\right) \\ &= O\left(n \cdot \frac{\frac{1}{2}}{(1 - \frac{1}{2})^2}\right) \\ &= O(n) \end{aligned}$$

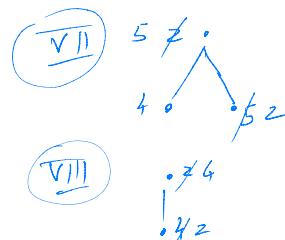
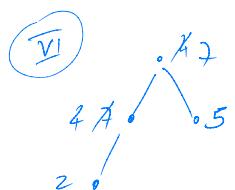
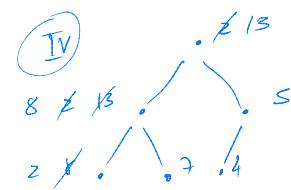
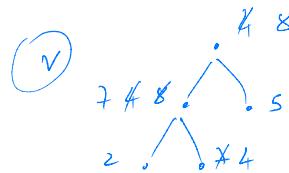
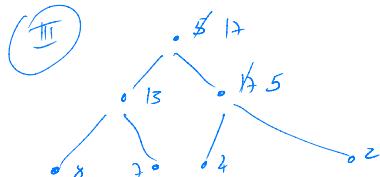
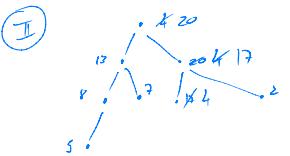
$$\begin{aligned} \sum_{i=0}^{\infty} i a^i &= \frac{1}{1-a} \\ \sum_{i=0}^{\infty} i a^{i-1} &= \frac{0 - (-1) \times 1}{(1-a)^2} \Leftrightarrow \sum_{i=0}^{\infty} i a^{i-1} = \frac{1}{(1-a)^2} \Leftrightarrow \sum_{i=0}^{\infty} i a^i = \frac{a}{(1-a)^2} \end{aligned}$$

Heapsort - Motivação (Exercício 6-4.1 CLRS)

<5, 13, 2, 25, 7, 17, 20, 8, 4>



25
20
17
13
5



Algoritmo 3 [Heap Sort]

```

HeapSort(A)
BuildMaxHeap(A)
for i = 1 to A.size
    swap(A, i, A.size - i + 1)
    MaxHeapify(A, i)

```

Complexidade:

$$\begin{aligned}
 & O(n) + nO(\log n) \\
 & = O(n) + O(n \log n) \\
 & = O(n \log n)
 \end{aligned}$$

Fila de Prioridade

Uma fila de prioridade é um tipo de dados abstrato usado para manter um conjunto de valores que se designam chaves da fila de prioridade.

A fila de prioridade deve suportar as seguintes operações:

- Max(A) \rightarrow retorna a chave máxima na fila
- ExtractMax(A) \rightarrow remove a chave de maior valor e remove a chave da fila de prioridade
- IncreaseKey(A, i, k) \rightarrow altera a chave no índice i por o valor k caso $A[i] \leq k$
- Insert(A, k) \rightarrow insere a chave k na heap

• Max(A)

retorna $A[1]$

$O(1)$

• ExtractMax(A)

$\downarrow (A.size < 1) \text{ erro } \dots$

$max = A[1]$

$O(\log n)$

$A[1] = A[A.size]$

$A.size = A.size - 1$

MaxHeapify(A, 1)

Retorna max

• IncreaseKey(A, i, k)

$\downarrow (A.size < i \text{ || } k < A[i]) \text{ erro } \dots$

$\downarrow (k > A[Parent(i)])$

$A[i] = A[Parent(i)]$

$O(\log n)$

IncreaseKey(A, Parent(i), k)

$\downarrow (A[i] = k)$

• Insert(A, k)

$A.size = A.size + 1$

$A[A.size] = -\infty$

$O(\log n)$

IncreaseKey(A, A.size, k)

(5)

Ordenações em Tempo Linear

(Teorema 8.1 CLRS)

Teorema [Algoritmos de Ordenação Baseados em Comparações]

Qualquer algoritmo de ordenação baseado em comparações tem, no pior caso, complexidade $\Omega(n \log n)$.