① 6ª Aula

Algoritmo Heapsort

Heap: Estrutura de dados útil quando se quer remover o máximo de um conjunto sucessivas vezes

Formalmente, um heap é uma árvore binária em que cada nó contém um elemento do conjunto e vale:

②

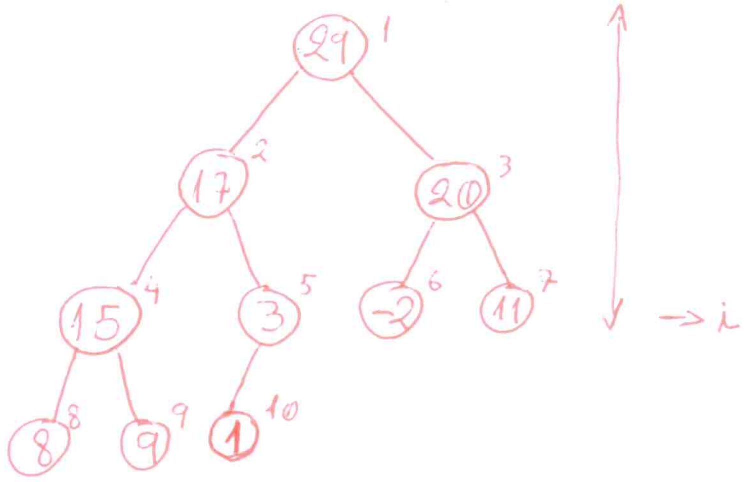
(1) A árvore é completa até o penúltimo nível

(2) No último nível todos os nós estão o mais à esquerda possível

(3) Cada nó interno armazena um valor maior que seus filhos

③ Exemplo

$S = \{20, 15, 17, -2, 8, 11, 3, 9, 19, 1\}$



1	2	3	4	5	6	7	8	9	10
29	17	20	15	3	-2	11	8	9	1

④

Um heap pode ser armazenado em um vetor de tal forma que:

- O pai do elemento da posição i $(i > 1)$ sempre está na posição $\lfloor i/2 \rfloor$
- E os filhos, nas posições $2i$ e $2i+1$ (se tais índices forem $\leq n$)

Note que o valor máximo do conjunto representado pelo heap sempre está na raiz.

⑤

→ Algoritmos de manipulação de heap consomem tempo proporcional à altura do heap

→ Altura de heap com n elementos: $\Theta(\log n)$

```
Desce-Heap(A, n, i)
enquanto (2i ≤ n) faça
    f = 2i
    se (f < n) e (A[f+1] > A[f])
        f = f+1
    se (A[i] < A[f])
        A[i] <-> A[f]
        i = f
    senão
        i = n+1
```

⑥ Remoção do máximo do heap

```
                    Heap
Extrai-Máximo(A, n)
max = A[1]
A[1] = A[n]
Desce-Heap(A, n-1, 1)
retorna max
```

Agora podemos definir o algoritmo Heap-Sort.

(7)

Heap-Sort (A)
1. Constroi-Heap (A) $\Longrightarrow$?
2. para i de n até 2 faça
3. A[1] <-> A[i]
4. Desce-Heap (A, i-1, 1)

$\left.\begin{array}{l} \text{2. para } i \text{ de } n \text{ até } 2 \text{ faça} \\ \left.\begin{array}{l}\text{3. } A[1] \leftrightarrow A[i] \\ \text{4. Desce-Heap}(A, i-1, 1)\end{array}\right] O(\lg n)\end{array}\right] O(n \lg n)$

Qual o custo de Constroi-Heap?

Qual a estratégia de Constroi-Heap?

(8) <u>Construção de um Heap</u>

Podemos utilizar Desce-Heap( ).

Constroi-Heap (A)
1. para i de $\lfloor n/2 \rfloor$ até 1 faça
2. Desce-Heap (A, n, i)

Custo de Constroi-Heap( ): $\Theta(n)$

Mostre que isso é verdade! (Exercício)

9 Complexidade do Heap-Sort

- Pior caso: Vetor em ordem decrescente

  $O(n \log n)$
- Melhor caso: Vetor em ordem crescente

  $O(n \log n)$
- Caso médio: Vetor aleatório

  $O(n \log n)$
- Complexidade de espaço

  $\Theta(n)$

10 Algoritmo Quicksort

- Estratégia de Divisão e Conquista

```
Quick-sort (A, p, r)
  se (p<r)
        q = particiona (A, p, r)   // p ≤ q < r
        Quick-Sort (A, p, q)
        Quick-Sort (A, q+1, r)
```

(11) Particionamento

particiona(A, p, r)	Pior caso
1. $x = A[r]$	$\Theta(1)$
2. $i = p-1$	$\Theta(1)$
3. para j de p até $r-1$ faça	$\Theta(n)$
4. se $A[j] \leq x$ então	$\Theta(n)$
5. $i = i+1$	$O(n)$
6. $A[i] \leftrightarrow A[j]$	$O(n)$
7. $A[i+1] \leftrightarrow A[r]$	$\Theta(1)$
8. retorna $i+1$	$\Theta(1)$
	$\Theta(n)$

(12) Complexidade do Quick-Sort

$$T(n) = \begin{cases} 1 & \text{, se } n = 1 \\ \max\{T(m) + T(n-m)\} + \Theta(n), & \text{se } n > 1 \text{ e } 1 \leq m < n \end{cases}$$

$T(n) = O(n^2)$ (Exercício)

Dica: Utilize indução!

(13)

→ Pior caso: Partições não balanceadas

$T(n) = T(n-1) + T(0) + \Theta(n)$

$T(n) = \Theta(n^2)$

→ Melhor caso: Partições balanceadas

$T(n) = 2T(n/2) + \Theta(n)$

$T(n) = \Theta(n \log n)$

→ Caso médio: Ambas!
$\Theta(n \log n)$

→ Complexidade de espaço
$\Theta(n)$

(14) Quick-Sort aleatorizado

particiona-aleatorizado (A, p, r)
1. i = aleatório (p, r)
2. A[p] <-> A[i]
3. devolva particiona (A, p, r)

Quick-Sort-Aleatorizado (A, p, r)
1. se p < r
2. q = particiona-aleatorizado (A, p, r)
3. Quick-Sort-Aleatorizado (A, p, q-1)
4. Quick-Sort-Aleatorizado (A, q+1, r)

(15) Análise do QuickSort aleatorizado

Recorrência para o tempo esperado $(E(T(n)))$:

$$T(n) = \begin{cases} \Theta(1) & , \text{ se } n = 1 \\ \frac{1}{n}\left(\sum_{K=0}^{n-1} \left(T(K) + T(n-1-K) + \Theta(n)\right)\right) & , \text{ se } n > 1 \end{cases}$$

Podemos manipular o somatório de forma adequada:

$$T(n) = \frac{2}{n} \sum_{K=0}^{n-1} T(K) + c.n$$

(16)

Podemos provar que $T(n) = O(n.\lg n)$. Suponha que $T(n) \leq a.n\lg n + b$, para $n \geq n_0$, com $a, b > 0$ constantes.

$$T(n) = \frac{2}{n} \sum_{K=0}^{n-1} T(K) + cn \leq$$

$$\leq \frac{2}{n} \sum_{K=0}^{n-1} (a.K\lg K + b) + cn =$$

$$= \frac{2a}{n} \sum_{K=1}^{n-1} K\lg K + 2b + cn$$

(17)

Sabemos que $\sum_{k=1}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$.

$$T(n) \leq \frac{2a}{n} \sum_{k=1}^{n-1} k \lg k + 2b + cn \leq$$

$$\leq \frac{2a}{n}\left(\frac{1}{2} n^2 \lg n - \frac{1}{8} n^2\right) + 2b + cn =$$

$$= an \lg n - \frac{a}{4} n + 2b + cn =$$

$$= an \lg n + b + \left(cn + b - \frac{a}{4} n\right) \leq$$

$$\leq an \lg n + b$$

Para $\frac{a}{4} n \geq cn + b,\ n \geq 1$.

(18) Observações:

- Supomos que todos os elementos de A são distintos
- ~~$\frac{1}{n}$ é a chance do elemento A[i] ser o i-ésimo menor elemento de A[p..r]~~
- No particiona-aleatorizado(), para cada j em {p,..,r}, o elemento A[i] tem chance de ser o j-ésimo menor elemento de A[p..r]