HeapSort

Prof. Juliano Foleis

Construção da Heap

A construção da heap é o procedimento usado para converter um vetor $V[1 \ldots n]$ qualquer em uma heap-máxima.

Sabemos que os elementos no subvetor $V[\lfloor \frac{n}{2} \rfloor + 1 \ldots n]$ são folhas, que são heaps unitárias. Portanto, basta percorrer os demais nós da árvore em ordem inversa e executar `MAX-HEAPIFY` em cada um deles:

```
BUILD-MAX-HEAP(V, n)
1. FOR i = CHAO(n/2) DOWNTO 1 DO
2.     MAX-HEAPIFY(V, i, n)
```

Corretude do BUILD-MAX-HEAP

```
1 PROCEDURE BUILD-MAX-HEAP(V, n)
2 FOR i = CHAO(n/2) DOWNTO 1 DO
3   MAX-HEAPIFY(V, i, n)
```

Este algoritmo está correto somente se ao final da execução o vetor $V[1 \ldots n]$ for uma heap máxima, ou seja, se o nó na posição 1 for raíz de uma heap-máxima.

Invariante:

No início de cada iteração do laço FOR das linhas 2–3, todos os nós das posições $i + 1, i + 2, \ldots, n$ são raízes de heaps-máximas.

Corretude do BUILD-MAX-HEAP

```
1 PROCEDURE BUILD-MAX-HEAP(V, n)
2 FOR i = CHAO(n/2) DOWNTO 1 DO
3   MAX-HEAPIFY(V, i, n)
```

Invariante: No início de cada iteração do laço FOR das linhas 2–3, todos os nós das posições $i+1, i+2, \ldots, n$ são raízes de heaps-máximas.

Inicialização: Antes da primeira iteração, $i = \left\lfloor \frac{n}{2} \right\rfloor$. Substituindo na invariante, temos:

Todos os nós das posições $\left\lfloor \frac{n}{2} \right\rfloor + 1, \left\lfloor \frac{n}{2} \right\rfloor + 2, \ldots n$ são raízes de heaps máximas.

Sabemos que todos os nós das posições $\left\lfloor \frac{n}{2} \right\rfloor + 1, \left\lfloor \frac{n}{2} \right\rfloor + 2, \ldots n$ são folhas, e portanto, raízes de heaps-máximas triviais. Portanto, é possível concluir que a invariante é verdadeira.

Corretude do BUILD-MAX-HEAP

```
1 PROCEDURE BUILD-MAX-HEAP(V, n)
2 FOR i = CHAO(n/2) DOWNTO 1 DO
3   MAX-HEAPIFY(V, i, n)
```

Invariante: No início de cada iteração do laço `FOR` das linhas 2–3, todos os nós das posições $i+1, i+2, \ldots, n$ são raízes de heaps-máximas.

Manutenção: Supondo que a invariante é verdadeira no início da iteração i, sabemos que todos os nós das posições $i+1, i+2, \ldots, n$ são raízes de heaps-máximas. Portanto, a pré-condição para invocar `MAX-HEAPIFY` na posição i é satisfeita, uma vez que os filhos do nó estão em posições maiores que i. Sabemos que `MAX-HEAPIFY` torna o nó da posição i a raíz de uma heap máxima, mantendo os nós nas posições subsequentes como raízes de heaps máximas. Desta forma, sabemos que:

Todos os nós das posições *i*, *i*+*1*, *i*+*2*, … , *n* são raízes de heaps-máximas.

Após o decremento do i a invariante é reestabelecida para a próxima iteração:

Todos os nós das posições *i*+*1*, *i*+*2*, *i*+*3*, … , *n* são raízes de heaps-máximas.

Corretude do BUILD-MAX-HEAP

```
1 PROCEDURE BUILD-MAX-HEAP(V, n)
2 FOR i = CHAO(n/2) DOWNTO 1 DO
3   MAX-HEAPIFY(V, i, n)
```

Invariante: No início de cada iteração do laço FOR das linhas 2–3, todos os nós das posições $i+1, i+2, \ldots, n$ são raízes de heaps-máximas.

Término: Após a última iteração do laço FOR, $i = 0$. Pela invariante de laço, todos os nós nas posições $i+1, i+2, i+3, \ldots, n$ são raízes de heaps-máximas. Substituindo i, sabemos que todos os nós nas posições $1, 2, 3, \ldots, n$ são raízes de heaps-máximas.

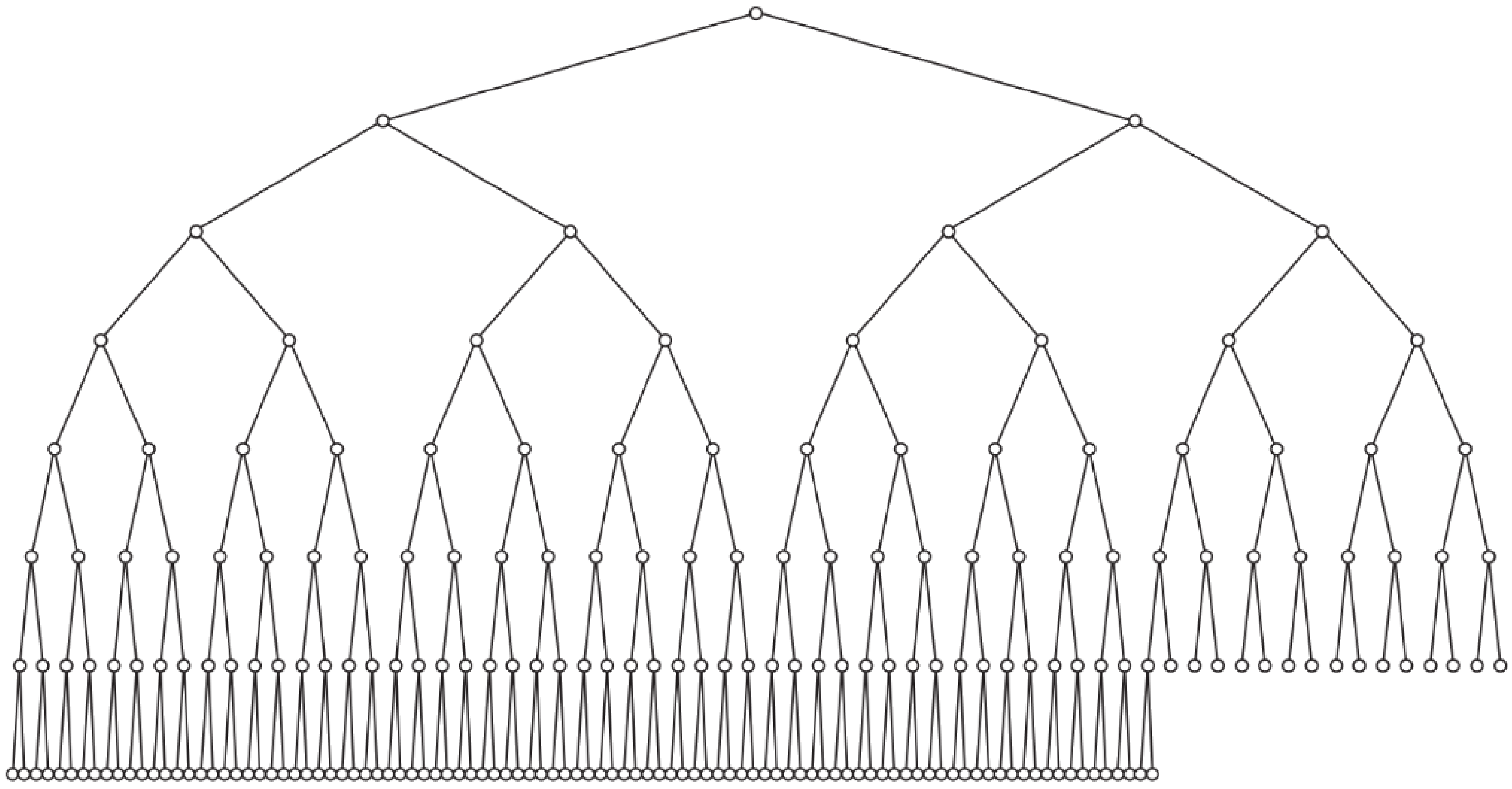
Como o nó na posição 1 é raíz de heap-máxima, então temos que o vetor V tornou-se uma heap máxima. Desta forma, o algoritmo está correto.

Desempenho de `BUILD-MAX-HEAP`

Um limite assintótico superior de `BUILD-MAX-HEAP` pode ser estimado da seguinte forma: cada chamada a `MAX-HEAPIFY` tem custo $O(\lg n)$ e `BUILD-MAX-HEAP` faz $O(n)$ chamadas delas. Desta forma, $T(n) = O(n) \cdot O(\lg n) = O(n \lg n)$ descreve o limite assintótico superior.

Contudo, embora correto, este limite não está ajustado o máximo possível.

Desempenho de `BUILD-MAX-HEAP`



Desempenho de `BUILD-MAX-HEAP`

Podemos encontrar o limite ajustado observando que **o tempo gasto por `MAX-HEAPIFY` varia de acordo com a altura do nó na árvore**, e que a altura da maioria dos nós é pequena (tende a uma constante).

Esta análise mais refinada está embasada nas propriedades que uma heap com com n elementos possui altura $\lfloor \lg n \rfloor$, e no máximo, $\left\lceil \left(\frac{n}{2^{h+1}} \right) \right\rceil$ nós em uma altura h. Assim, o tempo necessário para `MAX-HEAPIFY` executar em um nó de altura h é $O(h)$, portanto o custo de `BUILD-MAX-HEAP` está limitada superiormente por:

$$T(n) = \sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$$

Desempenho de `BUILD-MAX-HEAP`

$$T(n) = \sum_{h=0}^{\lfloor \lg n \rfloor} \left(\frac{n}{2^{h+1}} \right) h$$
$$= \frac{n}{2} \sum_{h=0}^{\lfloor \lg n \rfloor} \left(\frac{h}{2^{h}} \right)$$

sabendo que p/ $|x| < 1, \sum_{k=0}^{\infty} \frac{k}{x^k} = \frac{x}{(1-x)^2}$,

$$T(n) = \frac{n}{2} \sum_{h=0}^{\lfloor \lg n \rfloor} \left(\frac{h}{2^{h}} \right)$$
$$\leq \frac{n}{2} \sum_{h=0}^{\infty} \left(\frac{h}{2^{h}} \right)$$

como

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

$$T(n) \leq \frac{n}{2} \sum_{h=0}^{\infty} \left(\frac{h}{2^{h}} \right)$$
$$\leq \frac{n}{2}(2)$$
$$\leq n$$

e

Portanto, $T(n) = O(n)$.

Bibliografia

[CRLS] CORMEN, T. H. et al. Algoritmos: Teoria e Prática. Elsevier, 2012. 3a Ed. Capítulo 6 (Ordenação por Heap), Seções 6.1–6.4.