Fila de prioridades: Heap binário

Prof. Daniel Kikuti

Universidade Estadual de Maringá

Conteúdo

- Fila de prioridades.
- Estrutura de dados **heap**:
  - Definição.
  - Manutenção.
  - Construção de um heap.
- O algoritmo HEAPSORT.
- Exercícios

Fila de prioridades

Definição

Estrutura de dados abstrata usada para representar um **conjunto dinâmico de elementos** (S), onde cada elemento $x \in S$ está associado a uma **chave** (prioridade).

Operações suportadas:

- $\textsc{Insert}(S, x)$: insere o elemento x no conjunto S.
- $\textsc{Maximum}(S)$: devolve o elemento de S com a maior chave.
- $\textsc{Extract-Max}(S)$: remove o elemento $\textsc{Maximum}(S)$.
- $\textsc{Increase-Key}(S, x, k)$: altera a chave de x para k.

Por que estudar fila de prioridades?

- Diversas aplicações (ex.: escalonamento de processos);
- Útil no desenvolvimento de algoritmos sofisticados.

Como implementar uma fila de prioridades?

Formas básicas

- Lista duplamente encadeada (LDE) e um ponteiro para o máximo.
- Array ordenado (AO).

Exemplo: Considere os valores $\{2, 7, 26, 25, 19, 17, 1, 90, 3, 36\}$.
Quais os custos para efetuar as seguintes operações?

	LDE	AO
MAXIMUM		
EXTRACT-MAX		
INSERT		

Como implementar uma fila de prioridades?

Formas básicas

- Lista duplamente encadeada (LDE) e um ponteiro para o máximo.
- Array ordenado (AO).

Exemplo: Considere os valores $\{2, 7, 26, 25, 19, 17, 1, 90, 3, 36\}$.
Quais os custos para efetuar as seguintes operações?

	LDE	AO
MAXIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MAX	$O(n)$	$\Theta(1)$
INSERT	$\Theta(1)$	$O(n)$

Usando um heap binário

Objetivo: Aproveitar os pontos positivos das duas representações anteriores e efetuar as operações de maneira mais eficiente.

A estrutura de dados **Heap**

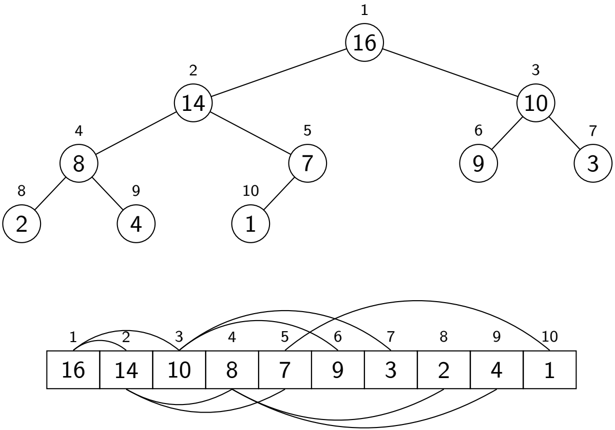
Definindo a estrutura

- **Heap** (binário) é um array que pode ser visto como uma árvore binária praticamente completa.
- A raiz da árvore é $A[1]$.
- Dado o índice i de um nó, as relações pai, filho a esquerda e filho a direita são definidas como:
  - $\textsc{parent}(i) = \lfloor i/2 \rfloor$.
  - $\textsc{left}(i) = 2i$.
  - $\textsc{right}(i) = 2i + 1$.

Tipos de heap e propriedade

- Dois tipos: **heap máximo** e **heap mínimo**.
- Para todo nó i diferente da raiz:
  - $A[\textsc{parent}(i)] \geq A[i]$ em um heap máximo.
  - $A[\textsc{parent}(i)] \leq A[i]$ em um heap mínimo.

A estrutura de dados **Heap** (Figura 6.1 [Cormen])



Exercício – Cormen 6.1-6

O vetor com valores $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12\rangle$ é um heap máximo?

Operações elementares

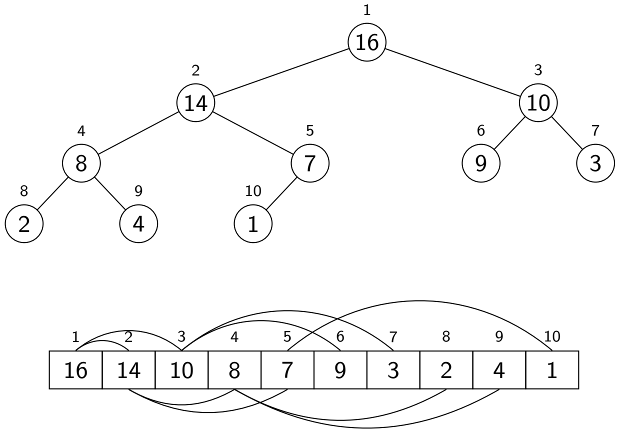
- Como consultar o elemento com maior chave?
- Como remover o elemento de maior chave?
- Como adicionar um novo elemento?
- Como alterar a chave de um elemento?
- Como constuir um heap?

Recurso visual

Para visualizar como as operações são feitas em um heap (e em diversas outras estruturas de dados e algoritmos), consulte o site `https://visualgo.net/en/heap`.

Inserção de elementos em um **Heap**

Considere o heap a seguir. Suponha que desejamos inserir os elementos: 6, 5, 12, 17. Como ficaria o Heap após estas inserções?



Sobre inserções

- Inserções são feitas sempre na posição de índice $n + 1$ (heap-size $+$ 1).
- Ao inserir o elemento, a propriedade do heap pode ser violada (por exemplo, pela inserção dos números 12 e 17 no exemplo anterior).
- Qual o custo para inserir um elemento e fixarmos o heap caso o elemento inserido viole a propriedade do heap?

Manutenção da propriedade de heap

Uma função auxiliar importante – MAX-HEAPFY

- A função MAX-HEAPIFY recebe como parâmetro um array A e um índice i, e requer que:
  - As árvores binárias com raízes em $\text{LEFT}(i)$ e $\text{RIGHT}(i)$ sejam heaps máximos.
- $A[i]$ pode ser menor que seus filhos.
- A função MAX-HEAPIFY deixa que o valor $A[i]$ "flutue para baixo", de maneira que a subárvore com raiz no índice i se torne um heap.

Exemplo

No quadro.

O algoritmo Max-Heapify

Max-Heapify(A, i)

$l \leftarrow$ LEFT(i)
$r \leftarrow$ RIGTH(i)
if $l \leq A.heap\text{-}size$ e $A[l] > A[i]$ **then**
    $largest \leftarrow l$
else
    $largest \leftarrow i$
if $r \leq A.heap\text{-}size$ e $A[r] > A[largest]$ **then**
    $largest \leftarrow r$
if $largest \neq i$ **then**
    Swap$(A[i], A[largest])$
    Max-Heapify$(A, largest)$

$A.heap\text{-}size$ é o número de elementos no heap e $A.length$ é o número de elementos do vetor.

Análise do tempo de execução do MAX-HEAPIFY

Análise em função da altura do nó

$\Theta(1)$ para corrigir os relacionamentos entre os elementos $A[i]$, $A[\text{LEFT}(i)]$ e $A[\text{RIGHT}(i)]$. Portanto, MAX-HEAPIFY em um nó de altura h é $O(h)$ (quantidade de chamadas recursivas).

Outra maneira de ver a complexidade:

- Tempo para executar MAX-HEAPIFY em uma subárvore com raiz em um dos filhos do nó i.
- As subárvores de cada filho têm tamanho máximo igual a $2n/3$ – ocorre quando o último nível da árvore está metade cheia.
- Portanto, o tempo total de execução pode ser descrito pela recorrência $T(n) \leq T(2n/3) + \Theta(1)$.
- Pelo caso 2 do teorema mestre $T(n) = O(\lg n)$.

A construção de um heap

Detalhando o procedimento

- O procedimento MAX-HEAPIFY pode ser usado de baixo para cima para converter um array $A[1..n]$ em um heap máximo.
- Os elementos no subarray $A[(\lfloor n/2 \rfloor + 1)..n]$ são folhas, e cada um é um heap máximo.
- O procedimento BUILD-MAX-HEAP percorre os nós restantes da árvore e executa MAX-HEAPIFY sobre cada um.

Exemplo do funcionamento do BUILD-MAX-HEAP

Considere o vetor: $A = \{3, 5, 4, 2, 1\}$.

A construção de um heap

O algoritmo

BUILD-MAX-HEAP(A)

1 $A.heap\text{-}size = A.length$
2 **for** $i \leftarrow \lfloor (A.length/2) \rfloor$ **to** 1 **do**
3 $\quad$ MAX-HEAPIFY(A, i)

Invariante de laço do BUILD-MAX-HEAP ($n = A.length$)

No começo de cada iteração do laço das linhas 2 e 3, cada nó $i+1, i+2, \ldots, n$ é a raiz de um heap máximo.

Correção do Build-Max-Heap

Demonstração

- **Inicialização**: antes da primeira linha $i = \lfloor n/2 \rfloor$ e cada nó $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ é uma folha, e portanto é a raiz de um heap máximo.
- **Manutenção**: os filhos de i têm índices maiores que i e pelo invariante são raízes de heaps máximos. Esta é a condição exigida para que a chamada $\textsc{max-hepify}(A, i)$ torne i a raiz de um heap máximo. Decrementar i restabelece a invariante para a próxima iteração.
- **Término**: $i = 0$, pela invariante de laço $1, 2, \ldots, n$ são raízes de um heap máximo, particularmente o nó 1 é uma raiz.

Análise de complexidade do BUILD-MAX-HEAP

Limite superior simples

- Cada chamada de MAX-HEAPIFY custa $O(\lg n)$.
- São feitas $\lfloor n/2 \rfloor$ chamadas.
- Portanto, o tempo de execução é $O(n \lg n)$.

Análise de complexidade do BUILD-MAX-HEAP

Limite superior simples

- Cada chamada de MAX-HEAPIFY custa $O(\lg n)$.
- São feitas $\lfloor n/2 \rfloor$ chamadas.
- Portanto, o tempo de execução é $O(n \lg n)$.

Limite restrito (justo)

- O tempo de execução de MAX-HEAPIFY varia com a altura da árvore, a altura da maioria dos nós é pequena.
- Um heap de n elementos tem altura $\lfloor \lg n \rfloor$ e no máximo $\lceil n/2^{h+1} \rceil$ nós de altura h.

Análise de complexidade do Build-Max-Heap

Limite restrito

- O tempo de execução do Build-Max-Heap será:

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

- Usando a fórmula $\sum_{k=1}^{\infty} kx^k = \frac{x}{(1-x)^2}$ com $x = \frac{1}{2}$, obtemos
$$\sum_{h=0}^{\infty} h \left(\frac{1}{2}\right)^h = \frac{1/2}{(1-1/2)^2} = 2.$$
- Portanto, o tempo de execução de Build-Max-Heap é $T(n) = O(n)$.

O algoritmo heapsort

O algoritmo

- Construir um heap usando a função BUILD-MAX-HEAP.
- Trocar o elemento $A[1]$ com $A[n]$, e atualizar o tamanho do heap para $n-1$.
- Corrigir o heap com a função MAX-HEAPIFY e repetir o processo.

Exemplo

Considere o heap: A= 5,3,4,2,1.

O algoritmo heapsort

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i ← A.length to 2 do
    SWAP(A[1], A[i])
    A.heap-size ← A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```

Análise do HEAPSORT

Correção do algoritmo

Exercício (Cormen 6.4-2).

Análise de complexidade

- A chamada a BUILD-MAX-HEAP demora $O(n)$.
- O procedimento MAX-HEAPIFY demora $O(\lg n)$ e é chamado $n - 1$ vezes.
- Portanto, o tempo de execução do HEAPSORT é $O(n \lg n)$.

Exercícios

- Descreva um algoritmo para **remover o elemento de maior chave**. Analise sua complexidade.
- Descreva um algoritmo para **adicionar um novo elemento**. Analise sua complexidade.
- Descreva um algoritmo para **alterar a chave de um elemento**. Analise sua complexidade.

Exercícios

1. Leitura do Capítulo 6 do Livro do Cormen.
2. Construa um heap usando a função BUILD-MAX-HEAP para os valores: $\{2, 7, 26, 25, 19, 17, 1, 90, 3, 36\}$.
3. Simule o algoritmo do HEAPSORT considerando os valores do exercício anterior.

Pensar nos exercícios do Cormen:

- 6.1-1 a 6.1-7
- 6.2-1 a 6.2-6
- 6.3-1 a 6.3-3
- 6.4-1 a 6.4-3