

BCC202 - Estruturas de Dados I

Aula 17: Ordenação: HeapSort

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: silvap@ufop.edu.br

2021



Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

- Inserção

- Remoção

- Implementação

Ordenação usando Heap

Heap Sort

Considerações Finais

Bibliografia

Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

Inserção

Remoção

Implementação

Ordenação usando Heap

Heap Sort

Considerações Finais

Bibliografia

Operações de uma Fila de Prioridade

6. Substitui o maior item por um novo item, a não ser que o novo item seja maior.
7. Altera a prioridade de um item.
8. Remove um item qualquer.
9. Agrupar duas filas de prioridades em uma única.

Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

 Inserção

 Remoção

 Implementação

Ordenação usando Heap

Heap Sort

Considerações Finais

Bibliografia

Heap

A melhor representação de uma fila de prioridade é por meio de uma estrutura de dados chamada **heap**:

Neste caso, Constrói é $O(n)$.

Inserir, Retirar, Substituir e Alterar são $O(\log n)$.

Ordenação por Heap

A *ordenação por heap* introduz outra técnica de projeto de algoritmos: a utilização de uma estrutura de dados, nesse caso uma estrutura que denominamos *heap* para gerenciar informações.

O que é *heap* afinal?

Um **heap** é um vetor que simula uma *árvore binária quase completa*.

Definição

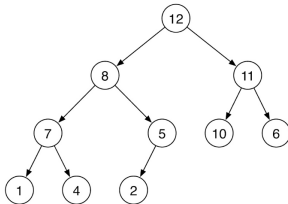
Árvore binária é uma estrutura de dados caracterizada por: Ou não tem elemento algum (árvore vazia). Ou tem um elemento distinto, denominado raiz, com dois apontamentos para duas estruturas diferentes, denominadas sub-árvore esquerda e sub-árvore direita¹.

¹https://pt.wikibooks.org/wiki/Algoritmos_e_Estruturas_de_Dados/Árvores_Binárias

Heap e Árvore Binária

Um **heap** é idealizado como uma **árvore binária quase completa**: cada **nó folha** da **árvore** está no **nível $N-1$** ou **$N-2$** - considerando uma árvore de níveis $0..N-1$.

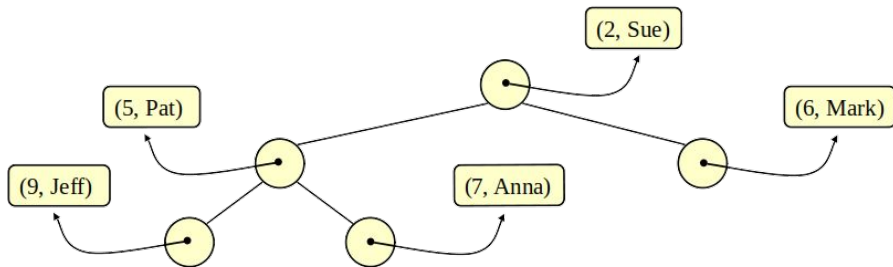
- ▶ as folhas são identificadas da esquerda para a direita
- ▶ a prioridade associada ao nó pai é menor (ou maior) do que as associadas ao(s) seu(s) filho(s)
- ▶ quando dois filhos, não há uma ordenação entre eles



Representação de um Heap

Cada nó possui uma tupla (chave, elemento)

Assim, cada nó do heap armazena todo o item

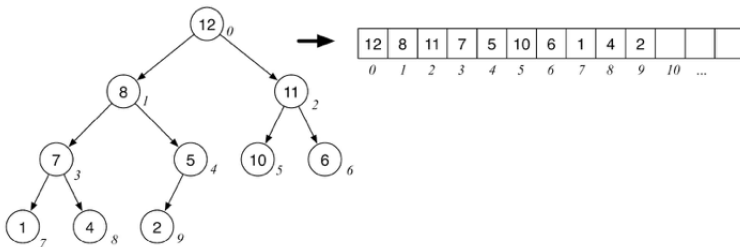


Um **heap** é um **vetor** que simula uma **árvore binária**.

Numeração

Numeramos os nós da árvore de cima para baixo e da esquerda para direita em cada nível.

- ▶ A árvore está completamente preenchida em todos os níveis, exceto possivelmente no último nível, ou nível mais baixo ($N - 1$), preenchido a partir da esquerda até um ponto.



Níveis

A **raiz** da árvore está no **nível zero (N0)**.

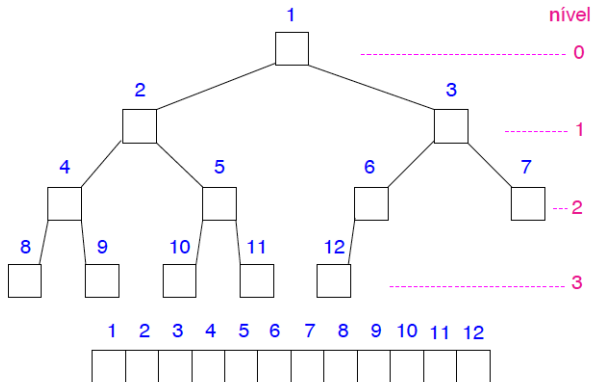


Figura: Níveis de uma árvore.

Considere um vetor $A[0 \dots n - 1]$ representando um **heap**.

Pai e Filho(s) de um Nó

- ▶ Cada posição do vetor corresponde a um nó do *heap*.
 - ▶ O **pai** de um nó i é $\lfloor (i - 1) / 2 \rfloor$ para $0 < i < n$
 - ▶ O nó 0 não tem pai.
 - ▶ O(s) **filho(s)** de um nó i , para $0 \leq i < \lfloor n / 2 \rfloor$
 - ▶ Seu **filho à esquerda** está na posição $2 * i + 1$
 - ▶ Seu **filho à direita** está na posição $2 * i + 2$

Nós Folhas

Os **nós folhas** da *árvore binária* estão nas posições do vetor:

- ▶ $\lfloor n / 2 \rfloor < i \leq n - 1$

Heaps

As chaves na árvore satisfazem a condição do heap.

A chave em cada nó é menor do que as chaves em seus filhos.

A chave no nó raiz é a menor chave do conjunto.

Como dito, uma árvore binária completa pode ser representada por um array.

Heaps II

A representação é extremamente compacta.

Permite caminhar pelos nós da árvore facilmente.

- ▶ Os filhos de um nó i estão nas posições $2i + 1$ e $2i + 2$.
- ▶ O pai de um nó i está na posição $(i - 1)/2$.

Na representação do heap em um vetor, a menor (ou maior) chave está sempre na posição 0 deste vetor.

Construção de um Heap

Um algoritmo elegante para construir o heap foi proposto por Floyd em 1964.

O algoritmo não necessita de nenhuma memória auxiliar.

Dado um vetor $A[0], A[1], \dots, A[n-1]$:

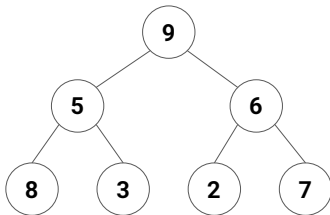
- ▶ Os itens $A[n/2], A[n/2+1], \dots, A[n-1]$ formam um heap válido pois são nós folhas (nós que não possuem filhos).
- ▶ Neste intervalo não existem dois índices i e j tais que $j = 2i + 1$ ou $j = 2i + 2$.

Exemplo da construção de um Heap

0	1	2	3	4	5	6
9	5	6	8	3	2	7

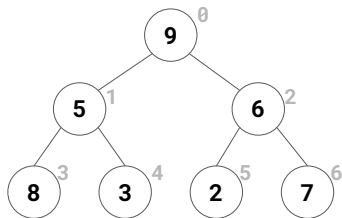
Exemplo da construção de um Heap

0	1	2	3	4	5	6
9	5	6	8	3	2	7



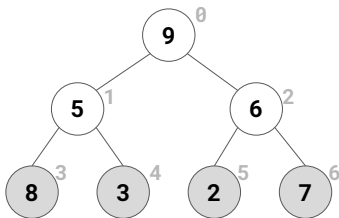
Exemplo da construção de um Heap

0	1	2	3	4	5	6
9	5	6	8	3	2	7

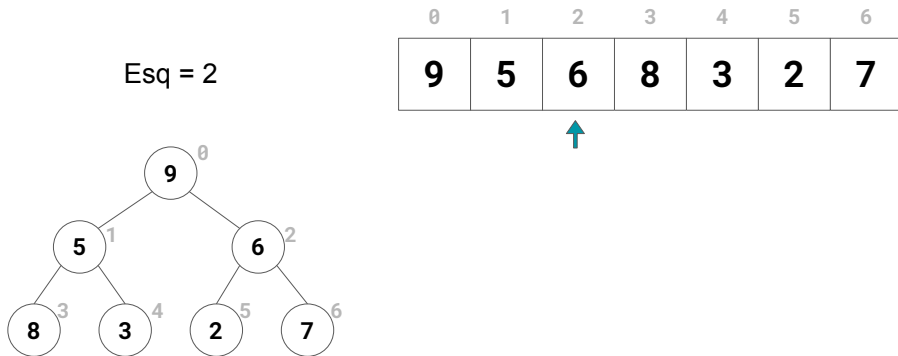


Exemplo da construção de um Heap

0	1	2	3	4	5	6
9	5	6	8	3	2	7

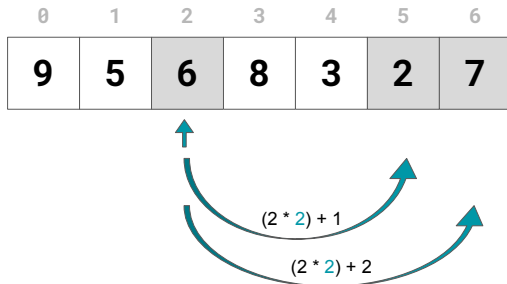
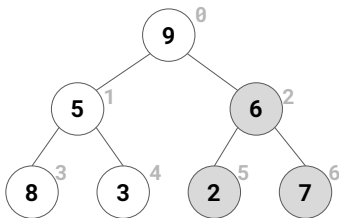


Exemplo da construção de um Heap



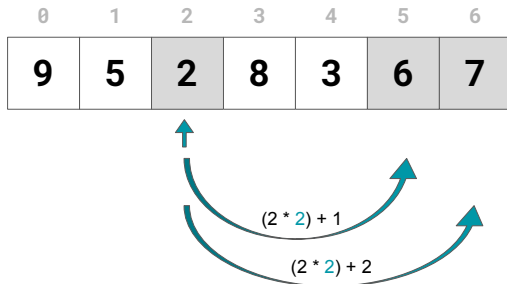
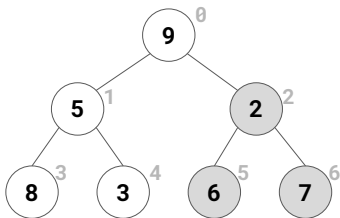
Exemplo da construção de um Heap

Esq = 2



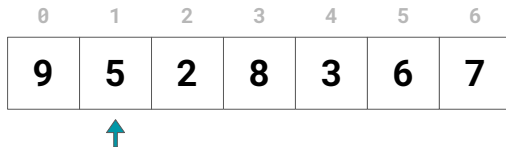
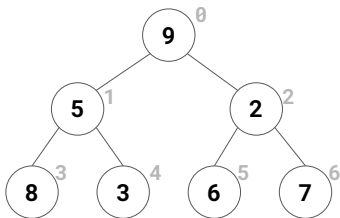
Exemplo da construção de um Heap

Esq = 2



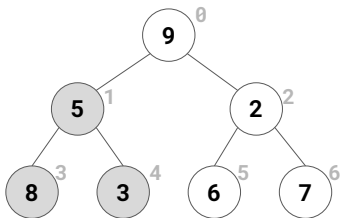
Exemplo da construção de um Heap

Esq = 1

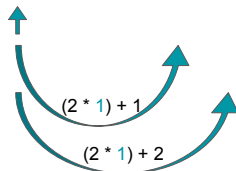


Exemplo da construção de um Heap

Esq = 1

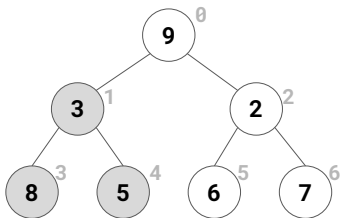


0	1	2	3	4	5	6
9	5	2	8	3	6	7

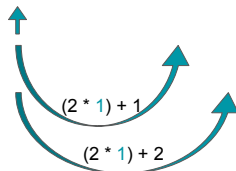


Exemplo da construção de um Heap

Esq = 1

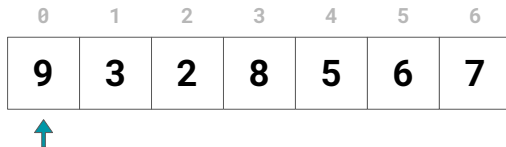
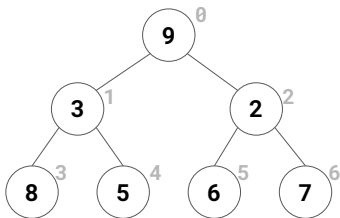


0	1	2	3	4	5	6
9	3	2	8	5	6	7

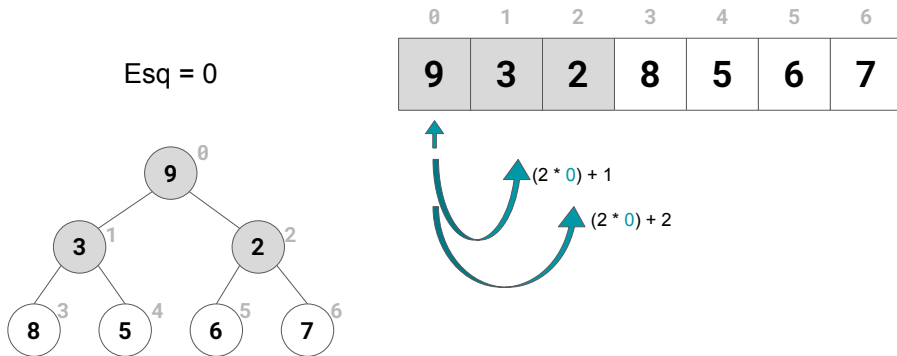


Exemplo da construção de um Heap

Esq = 0

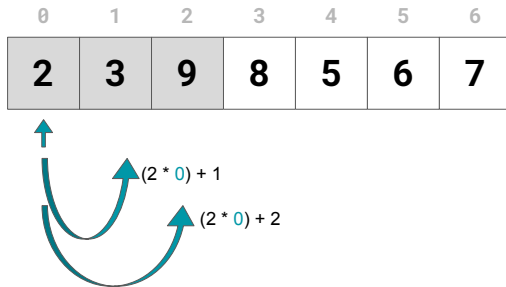
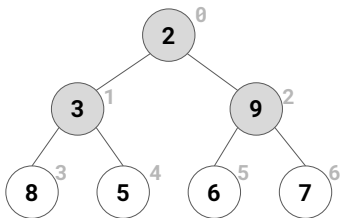


Exemplo da construção de um Heap



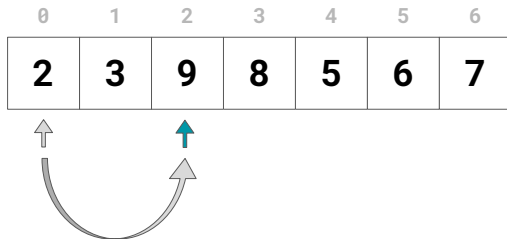
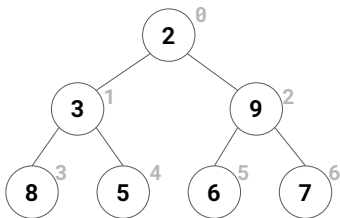
Exemplo da construção de um Heap

Esq = 0

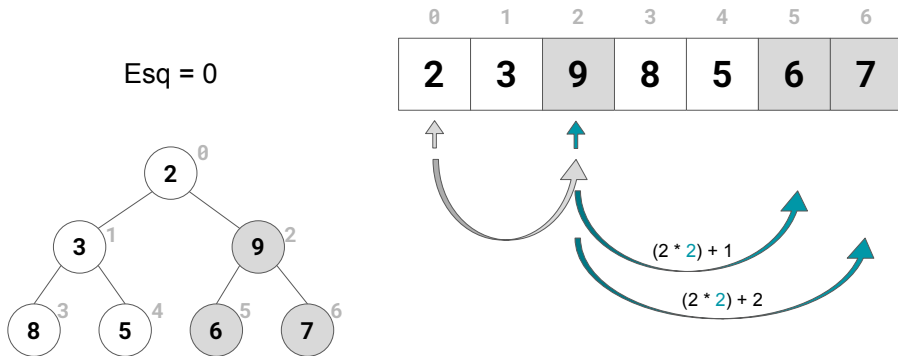


Exemplo da construção de um Heap

Esq = 0

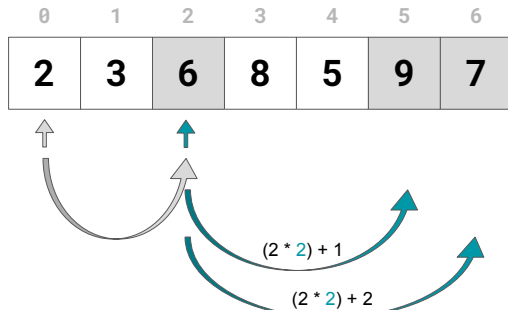
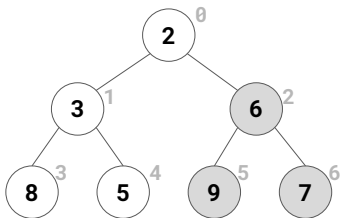


Exemplo da construção de um Heap

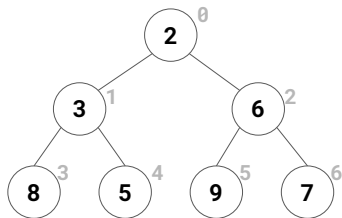


Exemplo da construção de um Heap

Esq = 0



Exemplo da construção de um Heap



0	1	2	3	4	5	6
2	3	6	8	5	9	7

Considerações sobre o exemplo

- ▶ Os itens de $A[3]$ a $A[6]$ formam um heap.
- ▶ O heap é estendido para a esquerda, englobando o item $A[2]$, pai dos itens $A[5]$ e $A[6]$.
- ▶ A condição de heap é violada:
 - ▶ O heap é refeito trocando os itens $A[2]$ e $A[5]$.
- ▶ O item 5 ($A[1]$) é incluindo no heap ($Esq = 1$).
- ▶ A condição de heap é violada:
 - ▶ O heap é refeito trocando os itens $A[1]$ e $A[4]$.

Considerações sobre o exemplo

- ▶ O item 9 ($A[0]$) é incluindo no heap ($Esq = 0$).
- ▶ A condição de heap é violada:
 - ▶ O heap é refeito trocando os itens $A[0]$ e $A[1]$.
- ▶ Como a condição ainda está sendo violada:
 - ▶ O heap é refeito trocando os itens $A[1]$ e $A[4]$.
- ▶ Como resultado, o heap foi construído:

2 3 6 8 5 9 7

Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

Inserção

Remoção

Implementação

Ordenação usando Heap

Heap Sort

Considerações Finais

Bibliografia

Uma *árvore binária* é **heap mínimo** ou **mini-heap** se:

Heap Mínimo ou Mini-Heap

Para todo nó i tal que $0 < i \leq n - 1$:

- ▶ $A[\lfloor (i - 1)/2 \rfloor] \leq A[i]$
- ▶ **o menor elemento está na raiz.**
- ▷ Os valores dos nós satisfazem a **propriedade de heap mínimo.**

Heap Máximo ou Max-Heap

Para todo nó i tal que $0 < i \leq n - 1$:

- ▶ $A[\lfloor (i - 1)/2 \rfloor] \geq A[i]$
- ▶ **o maior elemento está na raiz.**
- ▷ Os valores dos nós satisfazem a **propriedade de heap máximo.**

Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

Inserção

Remoção

Implementação

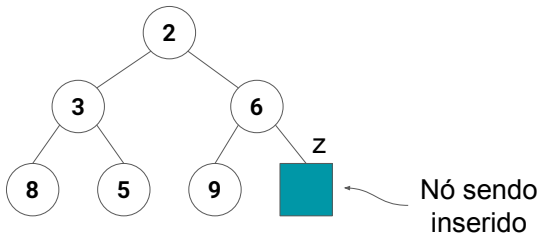
Ordenação usando Heap

Heap Sort

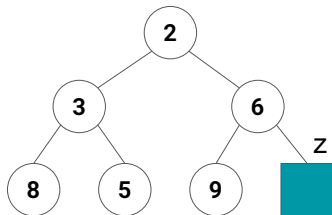
Considerações Finais

Bibliografia

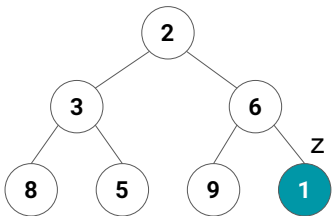
Inserir um Nó no Heap



Inserir um Nó no Heap

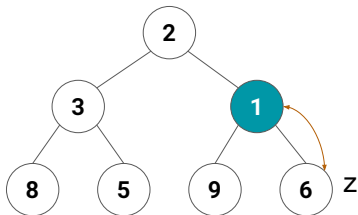


Nó sendo
inserido

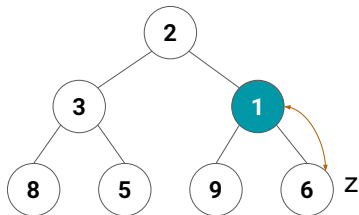


Comparar o nó inserido com os pais e trocar enquanto ele for menor que o pai ou até que ele seja o nó raiz

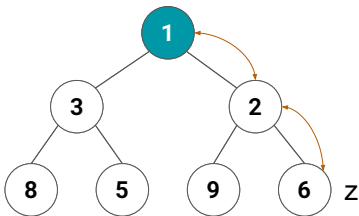
Inserir um Nó no Heap



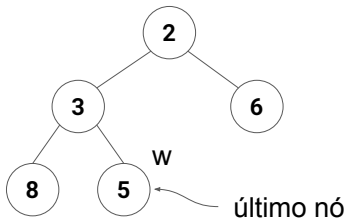
Inserir um Nó no Heap



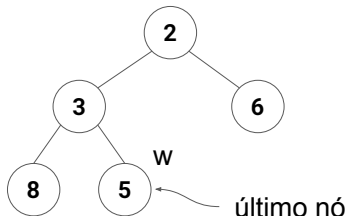
Na pior das hipóteses o custo de uma inserção será $O(\log n)$, equivalente à altura da árvore



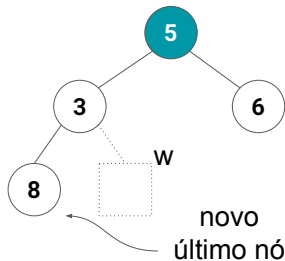
Remover um Nó do Heap



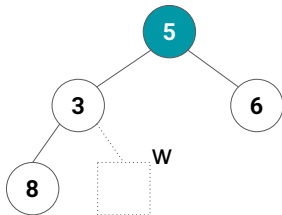
Remover um Nó do Heap



Trocar o nó raiz pelo
último nó do heap e
remover o último nó

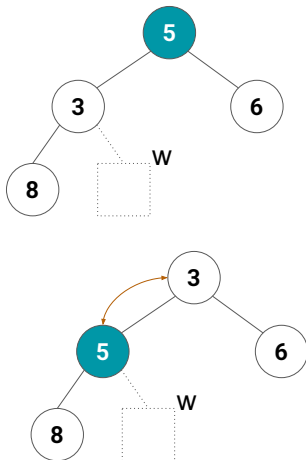


Remover um Nó do Heap



Refazer o heap!

Remover um Nó do Heap



Refazer o heap!

Na pior das hipóteses o custo de uma remoção será $O(\log n)$, equivalente à altura da árvore

1 **Algorithm:** HEAP_CONSTROI

Input: int* v, int n

2 **begin**

3 $esq \leftarrow (n/2) - 1$ // *esq = primeiro no antes do no folha do heap*

4 **while** $esq \geq 0$ **do**

5 HEAP_REFAZ(v, esq, n - 1)

6 $esq --$

7 **end**

8 **end**

1 Algorithm: HEAP_REFAZ**Input:** int* v, int esq, int dir**2 begin****3** $i \leftarrow \text{esq}$ **4** $j \leftarrow i * 2 + 1$ **5** $\text{aux} \leftarrow v[i]$ **6** **while** $j \leq \text{dir}$ **do****7** **if** $j < \text{dir}$ **and** $v[j] < v[j + 1]$ **then****8** $j = j + 1$ *// j recebe o outro filho de i***9** **end****10** **if** $\text{aux} \geq v[j]$ **then****11** break **12** **end****13** $v[i] \leftarrow v[j]$ **14** $i \leftarrow j$ **15** $j \leftarrow i * 2 + 1$ **16** **end****17** $v[i] \leftarrow \text{aux}$ **18 end**

Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

Inserção

Remoção

Implementação

Ordenação usando Heap

Heap Sort

Considerações Finais

Bibliografia

Ordenação usando Heap

- ▶ As operações das filas de prioridades podem ser utilizadas para implementar algoritmos de ordenação.
- ▶ Basta utilizar repetidamente a operação Insere para construir a fila de prioridades.
- ▶ Em seguida, utilizar repetidamente a operação Retira para receber os itens na ordem correta.

Listas vs. Métodos de Ordenação

- ▶ O uso de listas lineares não ordenadas corresponde ao método _____ .
- ▶ O uso de listas lineares ordenadas corresponde ao método _____ .
- ▶ O uso de heaps corresponde ao método _____ .

Listas vs. Métodos de Ordenação

- ▶ O uso de listas lineares não ordenadas corresponde ao método **Select Sort**.
- ▶ O uso de listas lineares ordenadas corresponde ao método _____ .
- ▶ O uso de heaps corresponde ao método _____ .

Listas vs. Métodos de Ordenação

- ▶ O uso de listas lineares não ordenadas corresponde ao método **Select Sort**.
- ▶ O uso de listas lineares ordenadas corresponde ao método **Insert Sort**.
- ▶ O uso de heaps corresponde ao método _____ .

Listas vs. Métodos de Ordenação

- ▶ O uso de listas lineares não ordenadas corresponde ao método **Select Sort**.
- ▶ O uso de listas lineares ordenadas corresponde ao método **Insert Sort**.
- ▶ O uso de heaps corresponde ao método **Heap Sort**.

Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

- Inserção

- Remoção

- Implementação

Ordenação usando Heap

Heap Sort

Considerações Finais

Bibliografia

Heap Sort

- ▶ Para reduzir o número de operações, será utilizado um Heap em que o **menor** valor é armazenado na **raiz**.
 - ▶ Teremos que o valor de um nó pai deverá ser sempre menor ou igual aos valores dos seus filhos.
- ▶ Utilizaremos repetidamente a operação Insere para construir a fila de prioridades.
- ▶ Em seguida, utilizaremos repetidamente a operação Retira para receber os itens na ordem **inversa**.

Heap Sort

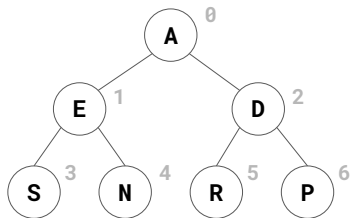
Algoritmo:

1. Construir o heap.
2. Troque o item na posição 0 do vetor (raiz do heap) com o item da posição $n-1$.
3. Use o procedimento Refaz para reconstituir o heap para os itens $A[0]$, $A[1]$, ..., $A[n - 2]$.
4. Repita os passos 2 e 3 com os $n - 1$ itens restantes, depois com os $n - 2$, até que reste apenas um item.

Exemplo de execução

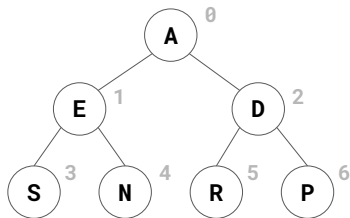
0	1	2	3	4	5	6
A	E	D	S	N	R	P

Exemplo de execução



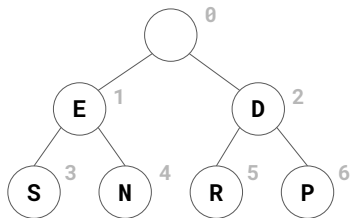
Exemplo de execução

Remover Raiz



Exemplo de execução

Remover Raiz

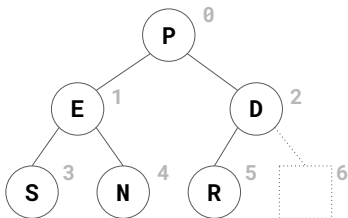


Item removido: A



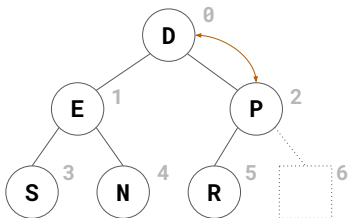
Exemplo de execução

Colocar na raiz
o último item



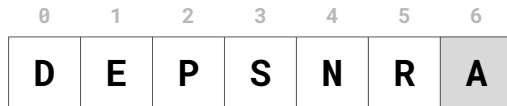
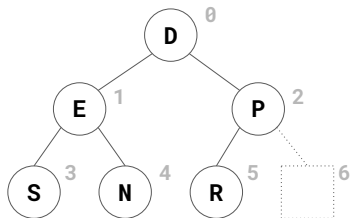
Exemplo de execução

Refazer o Heap



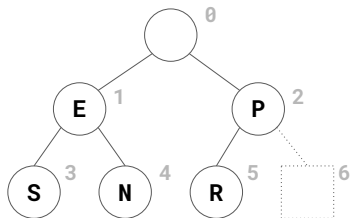
Exemplo de execução

Remover Raiz



Exemplo de execução

Remover Raiz

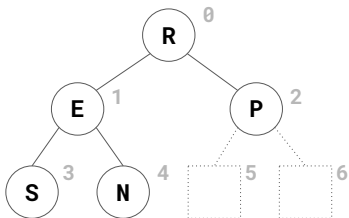


Item removido: D



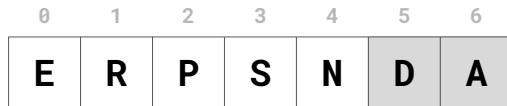
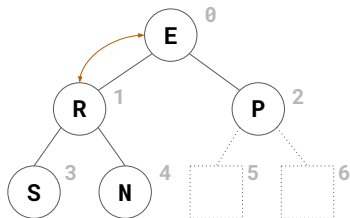
Exemplo de execução

Colocar na raiz
o último item



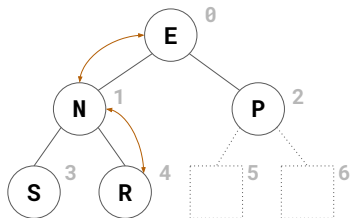
Exemplo de execução

Refazer o Heap



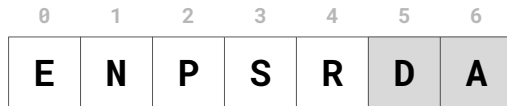
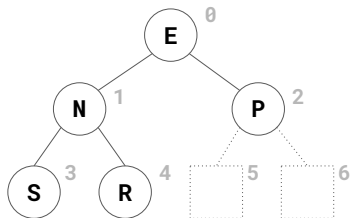
Exemplo de execução

Refazer o Heap



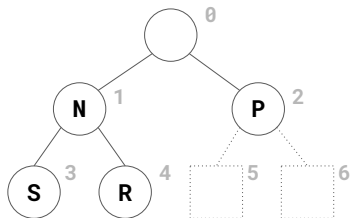
Exemplo de execução

Remover Raiz



Exemplo de execução

Remover Raiz

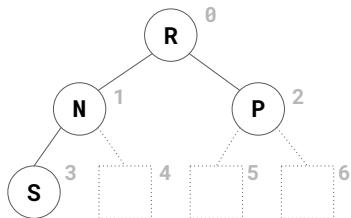


Item removido: E



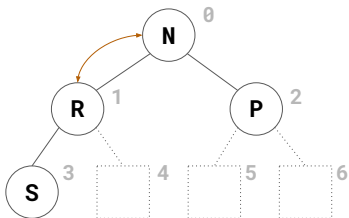
Exemplo de execução

Colocar na raiz
o último item



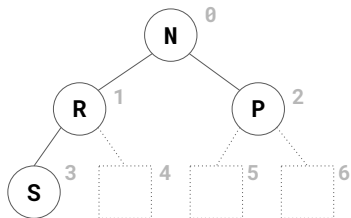
Exemplo de execução

Refazer o Heap



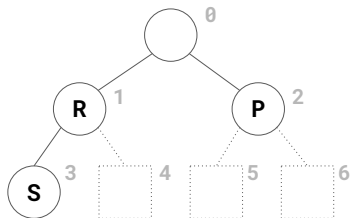
Exemplo de execução

Remover Raiz



Exemplo de execução

Remover Raiz

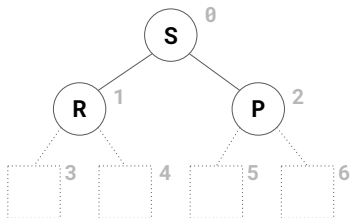


Item removido: N



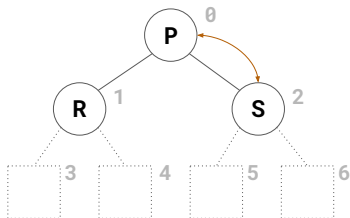
Exemplo de execução

Colocar na raiz
o último item



Exemplo de execução

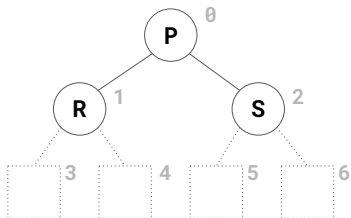
Refazer o Heap



0	1	2	3	4	5	6
P	R	S	N	E	D	A

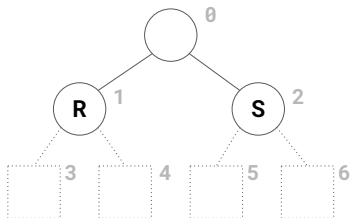
Exemplo de execução

Remover Raiz



Exemplo de execução

Remover Raiz

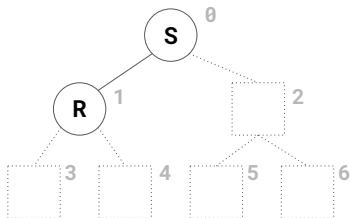


Item removido: P



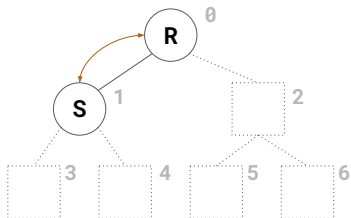
Exemplo de execução

Colocar na raiz
o último item



Exemplo de execução

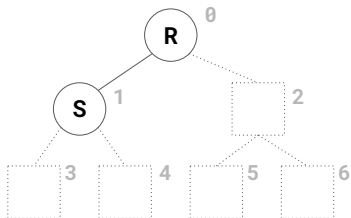
Refazer o Heap



0	1	2	3	4	5	6
R	S	P	N	E	D	A

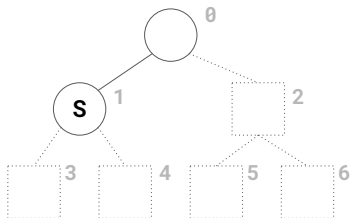
Exemplo de execução

Remover Raiz



Exemplo de execução

Remover Raiz

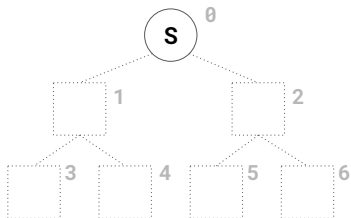


Item removido: R



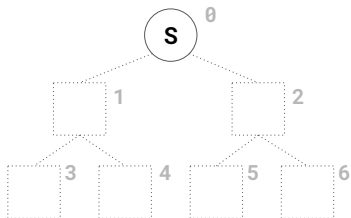
Exemplo de execução

Colocar na raiz
o último item



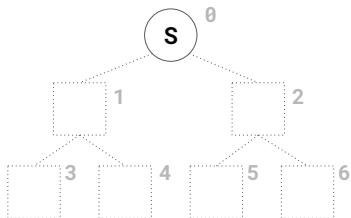
Exemplo de execução

Refazer o Heap



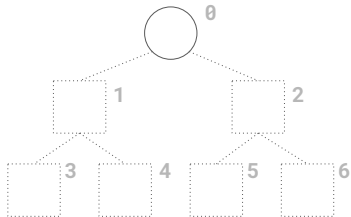
Exemplo de execução

Remover Raiz



Exemplo de execução

Remover Raiz

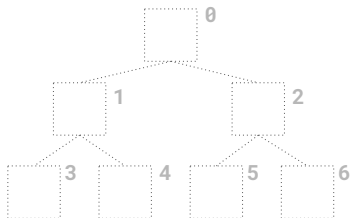


Item removido: S



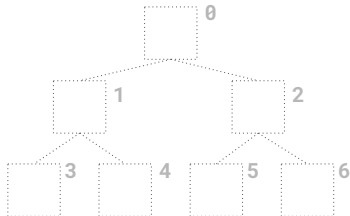
Exemplo de execução

Colocar na raiz
o último item



Exemplo de execução

Heap vazio!!!



Ordenação finalizada.



Vetor na ordem inversa!

Implementação

1 **Algorithm:** HEAP_SORT

Input: int* v, int n

2 **begin**

3 HEAP_CONSTROI(v, n)

4 **while** $n > 1$ **do**

5 $aux \leftarrow v[n-1]$

6 $v[n-1] \leftarrow v[0]$

7 $v[0] \leftarrow aux$

8 $n = n - 1$

9 HEAP_REFAZ(v, 0, n - 1) *// refaz o heap*

10 **end**

11 **end**

Análise do método

- ▶ O procedimento Refaz gasta cerca de $\log n$ operações, no pior caso.
- ▶ Constrói – executa $O(n)$ x **Refaz**.
- ▶ Loop interno – executa $O(n)$ x **Refaz**.
- ▶ Logo, **Heap Sort** gasta um tempo de execução $O(n \log n)$ no pior caso.
- ▶ A altura h da árvore de execução é $O(\log n)$.
- ▶ Refazer o heap, na pior das hipóteses, tem custo igual à altura da árvore. Construir o heap custa $O(n \log n)$.
- ▶ Logo: algoritmo é $O(n \log n)$.

Vantagens:

- ▶ O comportamento do *Heap Sort* é sempre $O(n \log n)$, qualquer que seja a entrada.

Desvantagens:

- ▶ O anel interno do algoritmo é bastante complexo se comparado com o do *Quick Sort*.
- ▶ O *Heap Sort* não é estável.

Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

- Inserção

- Remoção

- Implementação

Ordenação usando Heap

Heap Sort

Considerações Finais

Bibliografia

Conclusão

O Heap Sort é recomendado:

- ▶ Para aplicações que não podem tolerar eventualmente um caso desfavorável.
- ▶ Não é recomendado para arquivos com poucos registros, por causa do tempo necessário para construir o heap.

Ordenação em Tempo Linear

Conteúdo

Introdução

Heap

Heap Mínimo e Heap Máximo

Operações do Heap

- Inserção

- Remoção

- Implementação

Ordenação usando Heap


Heap Sort

Considerações Finais

Bibliografia

Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados de:

 Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford.

Introduction to Algorithms.

The MIT Press, 2011.

Exercício

- ▶ Dada a sequência de números: 3 4 0 2 5 1 3.
- ▶ Ordene em ordem crescente utilizando o algoritmo aprendido em sala (*Heap Sort*), apresentando a sequência dos números a cada passo (Teste de Mesa).