Heaps

Filas de prioridade eficientes

Prof. Marcelo de Souza
marcelo.desouza@udesc.br

Algoritmos e Estruturas de Dados
Bacharelado em Engenharia de Software
Universidade do Estado de Santa Catarina

Material de consulta

Leitura obrigatória:

- Capítulo 2 de Kleinberg and Tardos (2006) – Análise de algoritmos.
  - Seção 2.5 – Filas de prioridade.
- Capítulo 9 de Goodrich et al. (2014) – Filas de prioridade.

Leitura complementar:

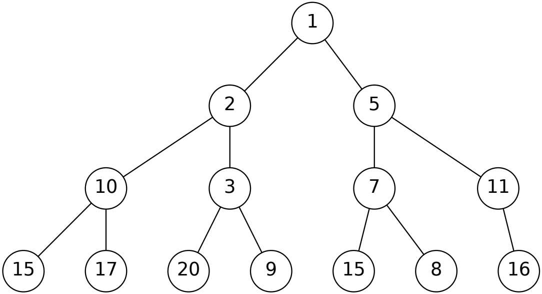
- Capítulo 11 de Preiss (2001) – Heaps e filas de prioridade.

Revisão de filas de prioridade

- Estrutura de dados que armazena elementos com suas prioridades.
- Operações principais:
  1. Inserir elementos.
  2. Recuperar/remover o elemento com maior prioridade.
- Implementação:
  - Lista encadeada não-ordenada: operação (2) em $O(n)$.
  - Lista encadeada ordenada: operação (1) em $O(n)$.
  - Heap (árvore binária balanceada): todas as operações em $O(\log n)$.

Heaps

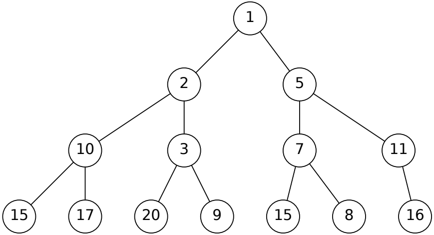
- Trata-se de uma árvore binária balanceada.
- Os elementos são ordenados de acordo com suas chaves (prioridades).
- *Heap order*:
  - Para todo nodo n, cujo pai é m, $\texttt{key}(m) \leq \texttt{key}(n)$.
  - Todo nodo possui maior prioridade (menor chave) que os seus filhos.
  - Nodos de maior prioridade são armazenados mais acima na estrutura.



Implementação

Contiguidade – vetores

- Um vetor armazena todos os elementos da estrutura.
- A raiz é armazenada na primeira posição.
- Os filhos vão sendo armazenados por níveis na árvore.

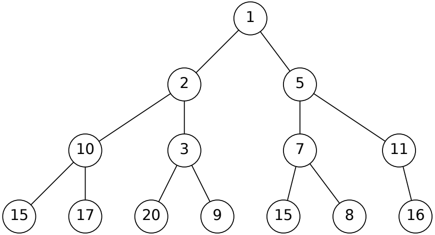


1	2	5	10	3	7	11	15	17	20	9	15	8	16	#

Implementação

Contiguidade – vetores

- Um vetor armazena todos os elementos da estrutura.
- A raiz é armazenada na primeira posição.
- Os filhos vão sendo armazenados por níveis na árvore.



Acesso aos vizinhos na árvore

Seja um elemento i da árvore:

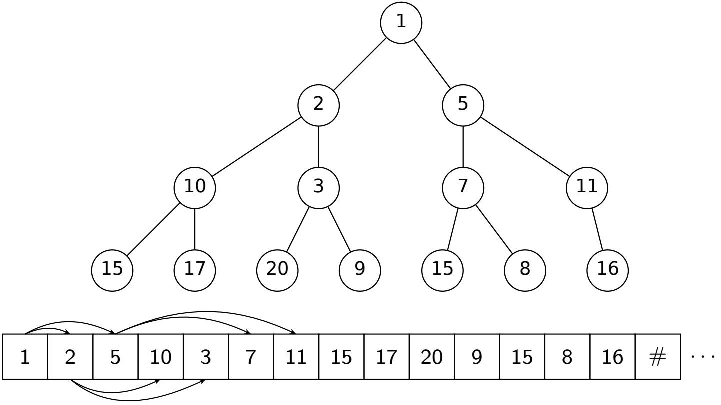
- $\texttt{leftChild}(i) = 2i + 1$
- $\texttt{rightChild}(i) = 2i + 2$
- $\texttt{parent}(i) = \lfloor (i-1)/2 \rfloor$

1	2	5	10	3	7	11	15	17	20	9	15	8	16	#

Implementação

Detalhes

- Chamamos o vetor de H e sua capacidade de N.
- Quando a heap tiver $n < N$ elementos, são usados as primeiras n posições do vetor, o que garante o balanceamento da estrutura.



Operações heap

Inserção de elemento

- **Cenário:** adicionar novo elemento v em uma heap H com $n < N$.
  1. Insere o elemento na primeira célula vaga i de H.
     - Isso quebra a propriedade de uma heap.
     - O elemento adicionado pode ter chave menor que seus ancestrais.
  2. "Conserta" a heap com o procedimento `heapify-up`.

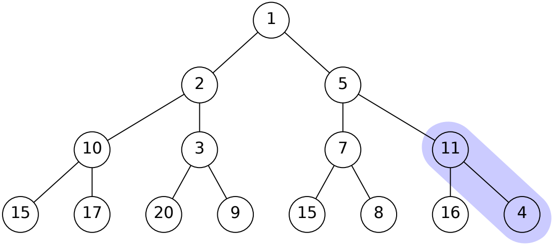
- `Heapify-up`: mover o elemento de maior prioridade para cima, até que a propriedade da heap esteja satisfeita.

Operações heap

Heapify-up

Algorithm: `heapify-up(H, i)`

if $i > 0$ **then**
  Let $j = \texttt{parent}(i) = \lfloor (i-1)/2 \rfloor$
  if $key(H[i]) < key(H[j])$ **then**
    Swap the array entries $H[i]$ and $H[j]$
    `heapify-up(H, j)`

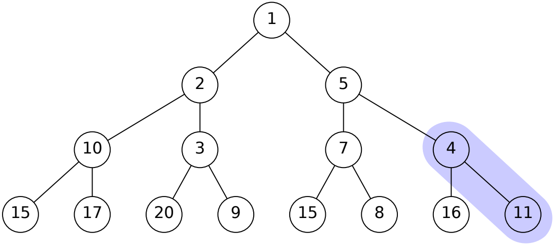


Operações heap

Heapify-up

Algorithm: `heapify-up(H, i)`

if $i > 0$ **then**
- Let $j =$ `parent`(i) $= \lfloor (i-1)/2 \rfloor$
- **if** $key(H[i]) < key(H[j])$ **then**
  - Swap the array entries $H[i]$ and $H[j]$
  - `heapify-up(H, j)`

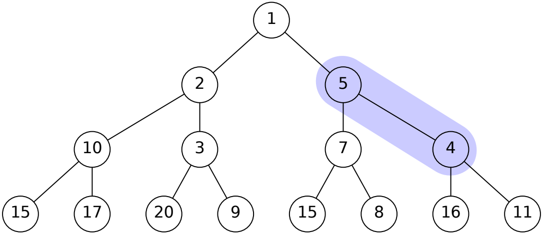


Operações heap

Heapify-up

Algorithm: `heapify-up(H, i)`

if $i > 0$ **then**
- Let $j =$ `parent`$(i) = \lfloor (i-1)/2 \rfloor$
- **if** $key(H[i]) < key(H[j])$ **then**
  - Swap the array entries $H[i]$ and $H[j]$
  - `heapify-up(H, j)`

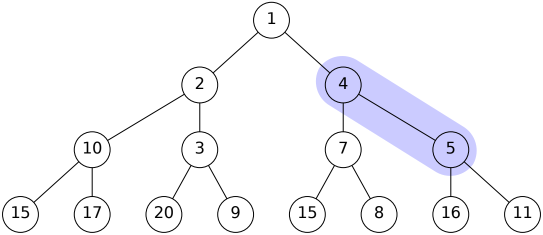


Operações heap

Heapify-up

Algorithm: `heapify-up(H, i)`

if $i > 0$ **then**
- Let $j =$ `parent(`i`)` $= \lfloor (i-1)/2 \rfloor$
- **if** $key(H[i]) < key(H[j])$ **then**
  - Swap the array entries $H[i]$ and $H[j]$
  - `heapify-up(H, j)`



Operações heap

Remoção de elemento

- Em geral, uma fila de prioridades permite a remoção somente do elemento raiz, que possui maior prioridade (operação `ExtractMin`), mas podemos implementar a remoção arbitrária (operação `Delete`).
- **Cenário:** remover um elemento v em uma heap H com n elementos.
  1. Remove o elemento na célula i de H (posição $H[i]$).
     - ▷ Isso resulta em um "buraco" na árvore.
  2. Move o último elemento (folha) da heap para a posição vaga.
     - ▷ Isso quebra a propriedade de uma heap.
     - ▷ O elemento movido pode ter chave menor que seus ascendentes ou maior que seus descendentes.
  3. "Conserta" a heap com o procedimento `heapify-up` ou `heapify-down`, conforme a violação criada.

Operações heap

Heapify-down

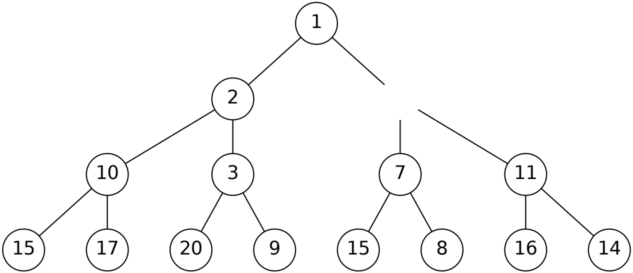
Algorithm: `heapify-down(H, i)`

$n \leftarrow H.size$
if $2i + 1 > n$ **then**
  Terminate with H unchanged
else if $2i + 1 = n$ **then**
  $j \leftarrow 2i + 1$
else if $2i + 1 < n$ **then**
  $left \leftarrow 2i + 1$
  $right \leftarrow 2i + 2$
  $j \leftarrow$ index that minimizes $key[H[left]]$ and $key[H[right]]$
if $key[H[j]] < key[H[i]]$ **then**
  Swap the array entries $H[i]$ and $H[j]$
  `heapify-down(H, j)`

Operações heap

Heapify-down

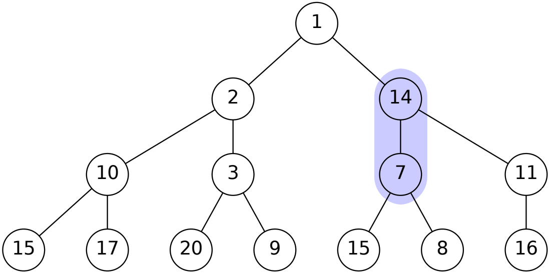
- `Heapify-down`: move o elemento de menor prioridade para baixo, trocando com o menor de seus filhos até que a propriedade da heap seja satisfeita.



Operações heap

Heapify-down

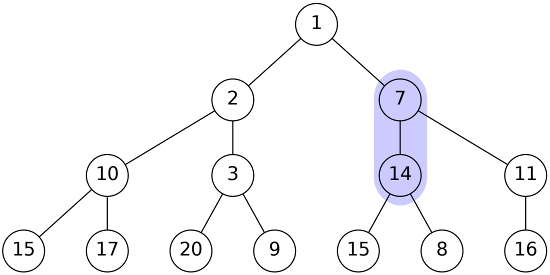
- `Heapify-down`: move o elemento de menor prioridade para baixo, trocando com o menor de seus filhos até que a propriedade da heap seja satisfeita.



Operações heap

Heapify-down

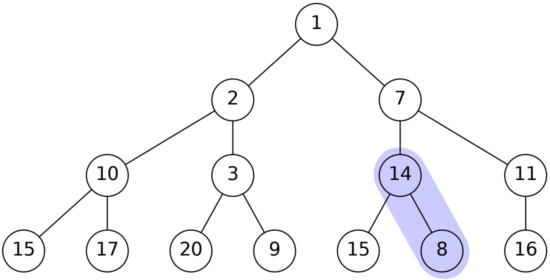
- `Heapify-down`: move o elemento de menor prioridade para baixo, trocando com o menor de seus filhos até que a propriedade da heap seja satisfeita.



Operações heap

Heapify-down

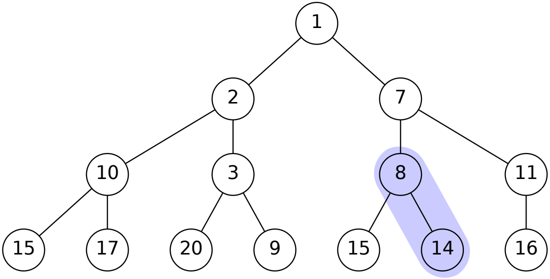
- `Heapify-down`: move o elemento de menor prioridade para baixo, trocando com o menor de seus filhos até que a propriedade da heap seja satisfeita.



Operações heap

Heapify-down

- `Heapify-down`: move o elemento de menor prioridade para baixo, trocando com o menor de seus filhos até que a propriedade da heap seja satisfeita.



Comparação

Complexidade das operações

Método	Lista não-ordenada	Lista ordenada	Heap
`size/isEmpty`	$O(1)$	$O(1)$	$O(1)$
`insert`	$O(1)$	$O(n)$	$O(\log n)$
`min`	$O(n)$	$O(1)$	$O(1)$
`removeMin`	$O(n)$	$O(1)$	$O(\log n)$

Exercício

Implementação com encadeamento

1. Implemente uma heap usando contiguidade para utilizar como fila de prioridades. A estrutura de dados deve fornecer as seguintes operações:
   - inserir elemento
   - encontrar o elemento mínimo
   - extrair elemento mínimo
   - deletar elemento
2. Implemente também um método `changeKey`, que altera a chave (prioridade) de um elemento determinado.

Referências

Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. (2014). *Data structures and algorithms in Java*. John Wiley & Sons, 6th edition.

Kleinberg, J. and Tardos, É. (2006). *Algorithm Design*. Pearson Education India.

Preiss, B. R. (2001). *Estruturas de dados e algoritmos: padrões de projetos orientados a objetos com Java*. Campus.