

ESDC4: Estruturas de Dados I

Aula 10: Estruturas de Dados Não-Lineares - Filas de Prioridades

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

Filas de Prioridades

Contextualização e Aplicação

- As filas de prioridades são usadas quando há a necessidade de se obter chaves em ordem, mas não necessariamente mantê-las numa ordenação completa;
- A ideia é processar os elementos armazenados nela de acordo com a forma que esse ordenação sob demanda está implementada;
- Uma das aplicações mais conhecidas das filas de prioridades é manter os processos do sistema operacional em uma ordem em que os que têm maior prioridade serão escolhidos primeiro para serem executados (*job scheduling*);
- Outro uso é a aplicação em sistemas de simulação em que as chaves correspondem à ordem cronológica de eventos, os quais precisam ter uma ordem para serem processados.

Filas de Prioridades

Heap Binário

➡ Heap (monte) binário (árvore binária completa):

➡ **Armazenamento direto em array:**

➡ Raiz na posição 1;

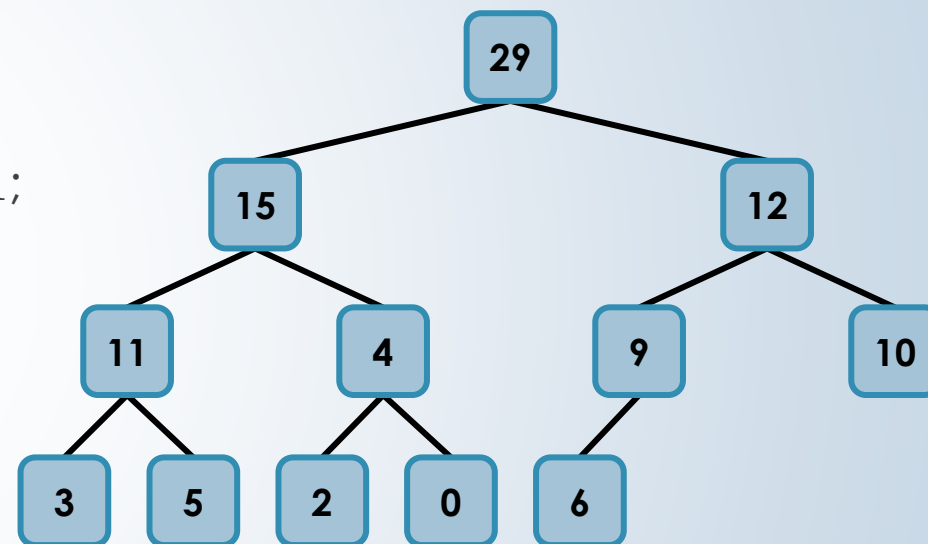
➡ Último elemento na posição $tamanho - 1$;

➡ **Manipulação dos índices:**

➡ **Pai:** $posição\ do\ filho / 2$;

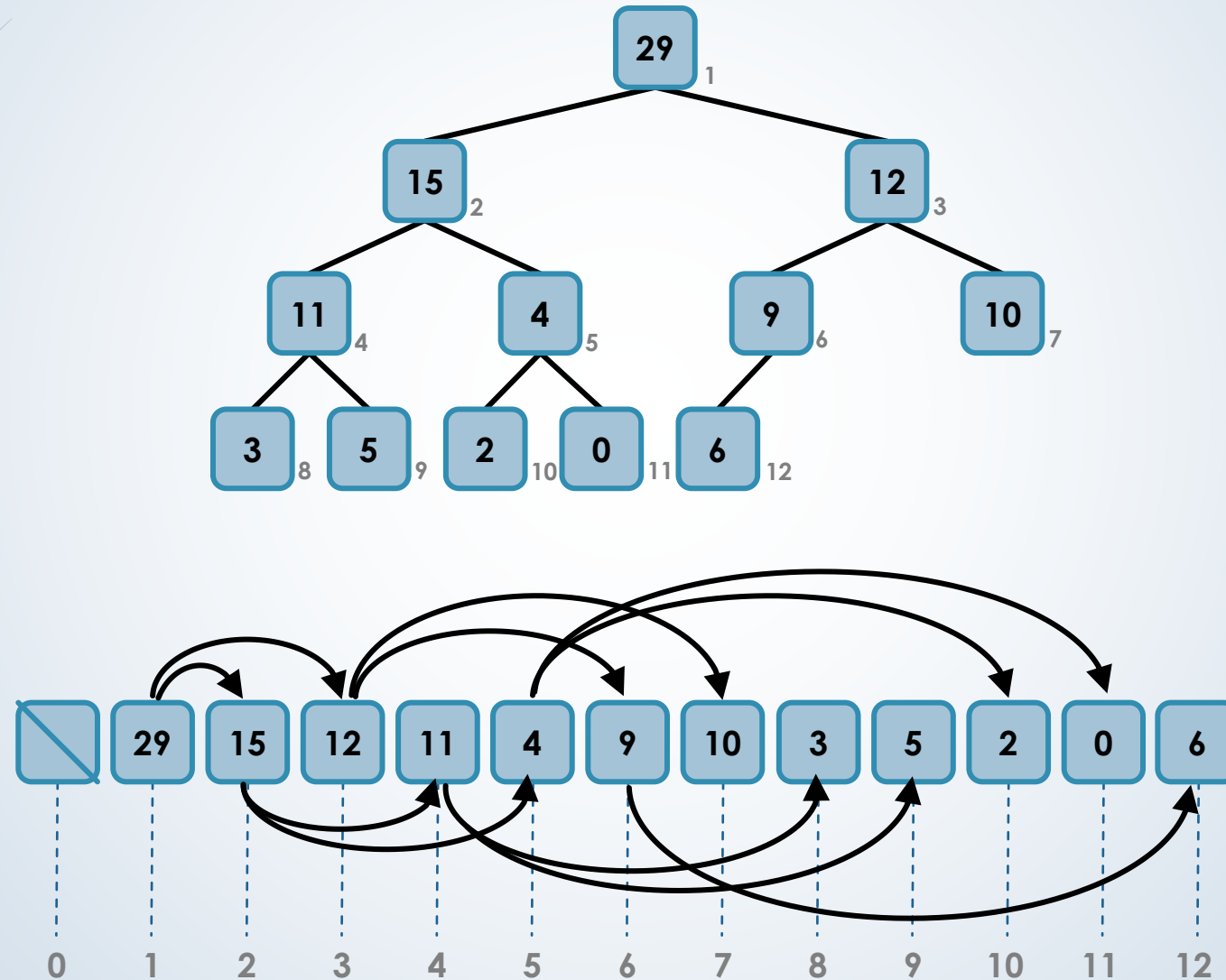
➡ **Filho da esquerda:** $posição\ do\ pai * 2$;

➡ **Filho direita:** $posição\ do\ pai * 2 + 1$.



Filas de Prioridades

Heap Binário



Filas de Prioridades

Implementações

- Iremos estudar duas formas de implementar Filas de Prioridades:

1. Fila de Prioridades Máxima (**MaxPriorityQueue**);
2. Fila de Prioridades Mínima (**MinPriorityQueue**);

- Seguiremos uma API padrão (em inglês):

- **insert**: insere uma chave na fila de prioridades de acordo com a invariante do heap binário usado;
- **peek**: consulta a chave com maior prioridade de acordo com a invariante do heap binário usado;
- **delete**: remove a chave com maior prioridade de acordo com a invariante do heap binário usado;
- **clear**: limpa a fila de prioridades, removendo todos os elementos;
- **isEmpty**: verifica se a fila e prioridades está vazia;
- **getSize**: obtém a quantidade de itens/elementos na fila de prioridades.

Key Comparable Key

PriorityQueue

```
+ insert(key : Key) : void  
+ peek() : Key  
+ delete() : Key  
+ clear() : void  
+ isEmpty() : boolean  
+ getSize() : int
```

Filas de Prioridades Máximas

Heap Binário Máximo

➤ Heap Binário Máximo

➤ **Invariante:** chave do nó pai é sempre maior ou igual às chaves dos nós filhos;

➤ Elemento violando a condição Heap Binário Máximo:

➤ Chave do filho maior que a chave do pai:

- O elemento precisa "subir" na árvore;
- Bottom-up *reheapify* (*swim* → flutuar);

➤ Chave do pai menor que a chave dos filhos (um ou dois):

- O elemento precisa "descer" na árvore;
- Top-down *reheapify* (*sink* → afundar).

Filas de Prioridades Mínimas

Heap Binário Mínimo

- Heap Binário Mínimo

- **Invariante:** chave do nó pai é sempre menor ou igual às chaves dos nós filhos;

- Elemento violando a condição Heap Binário Mínimo:

- **Chave do filho menor que a chave do pai:**

- O elemento precisa "subir" na árvore;
 - Bottom-up *reheapify* (*swim* → flutuar);

- **Chave do pai maior que a chave dos filhos (um ou dois):**

- O elemento precisa "descer" na árvore;
 - Top-down *reheapify* (*sink* → afundar).

Filas de Prioridades Máximas

Heap Binário Máximo – Operação **swim**

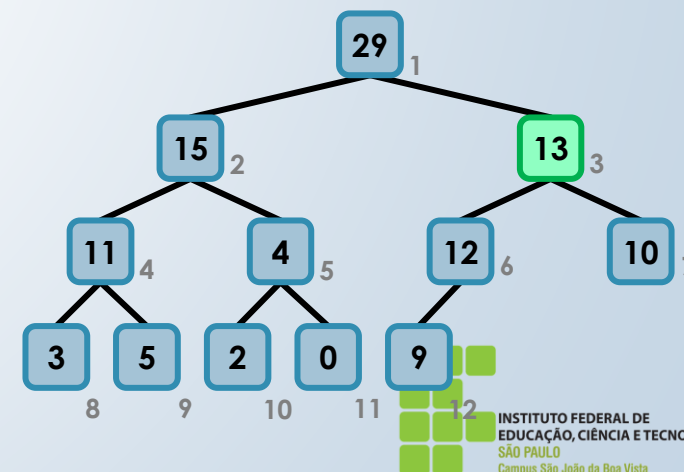
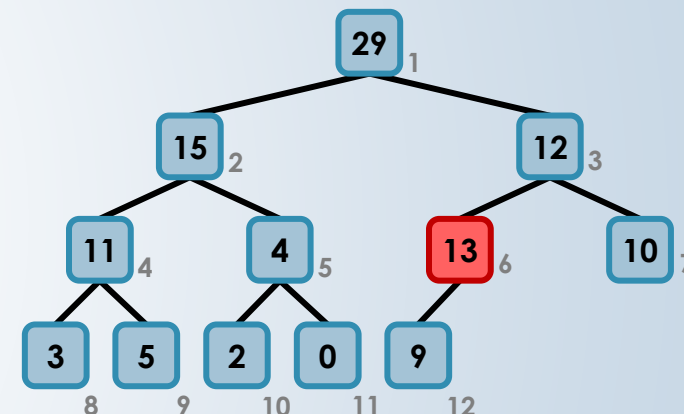
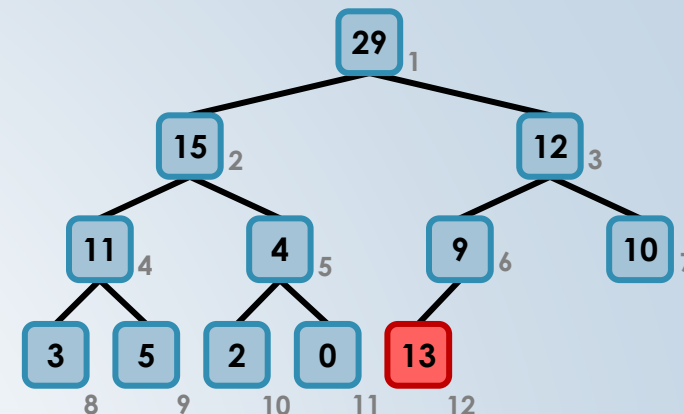
```
private void swim( int k ) {
    while ( k > 1 && less( k / 2, k ) ) {
        exchange( k, k / 2 );
        k = k / 2;
    }
}

private boolean less( int i, int j ) {
    return pq[i].compareTo( pq[j] ) < 0;
}

private void exchange( int i, int j ) {
    Key temp = pq[i];
    pq[i] = pq[j];
    pq[j] = temp;
}
```

cálculo da
posição do pai
 $k/2$, onde k é a
posição do filho

pq: array de
chaves do
heap binário



Filas de Prioridades Máximas

Heap Binário Máximo – Operação **sink**

```
private void sink( int k ) {
    while ( 2 * k <= n ) {
        int j = 2 * k;
        if ( j < n && less( j, j + 1 ) ) {
            j++;
        }
        if ( !less( k, j ) ) {
            break;
        }
        exchange( k, j );
        k = j;
    }
}
```

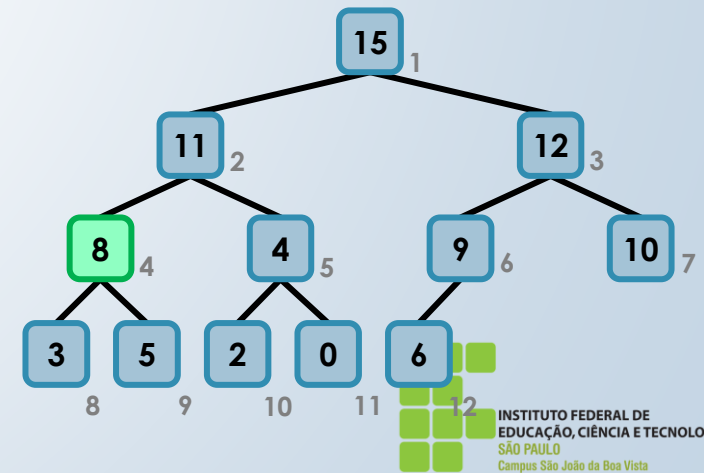
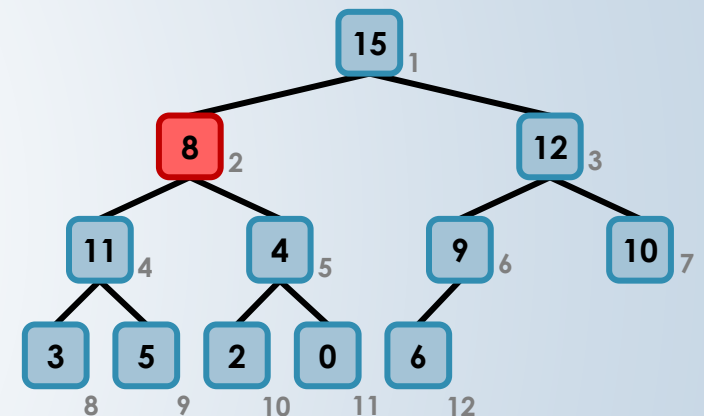
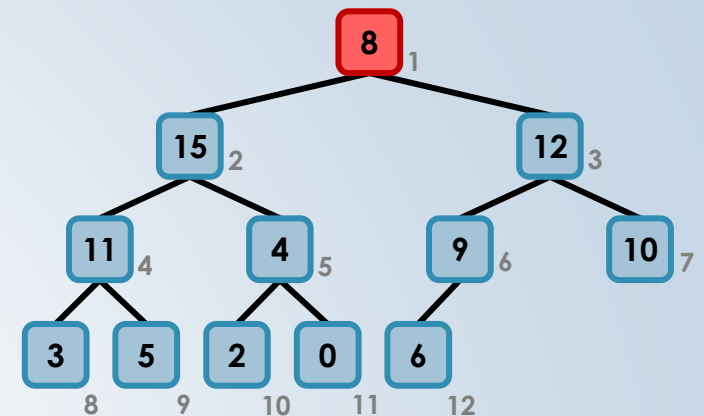
```
private boolean less( int i, int j ) {
    return pq[i].compareTo( pq[j] ) < 0;
}
```

```
private void exchange( int i, int j ) {
    Key temp = pq[i];
    pq[i] = pq[j];
    pq[j] = temp;
}
```

**cálculo da
posição dos filhos**

$k * 2$ (esquerda)
 $k * 2 + 1$ (direita),
 onde k é a
 posição do pai

pq: array de chaves do
 heap binário
n: quantidade de itens
 na fila de prioridades



Filas de Prioridades Máximas

Operações

```

public void insert( Key key ) {
    // insere a chave e a flutua
    pq[++n] = key;
    swim( n );
}

public Key peek() {
    return pq[1];
}

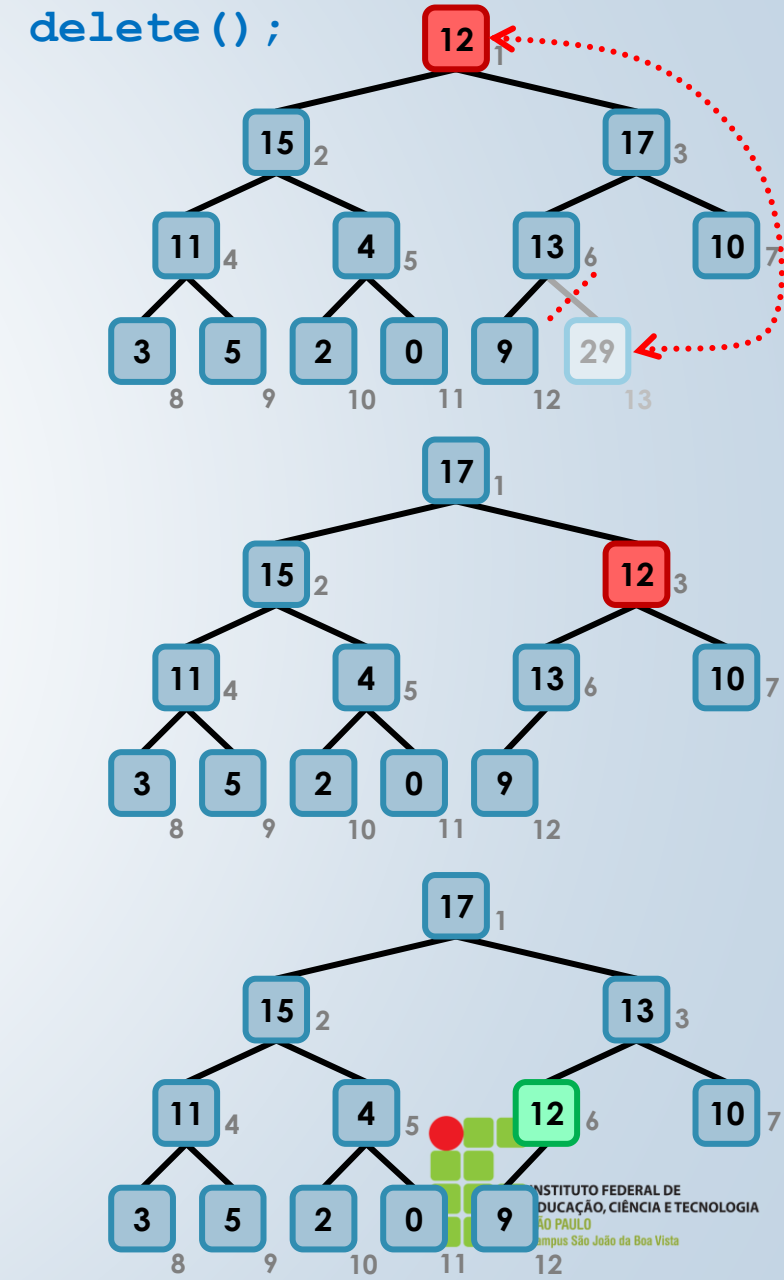
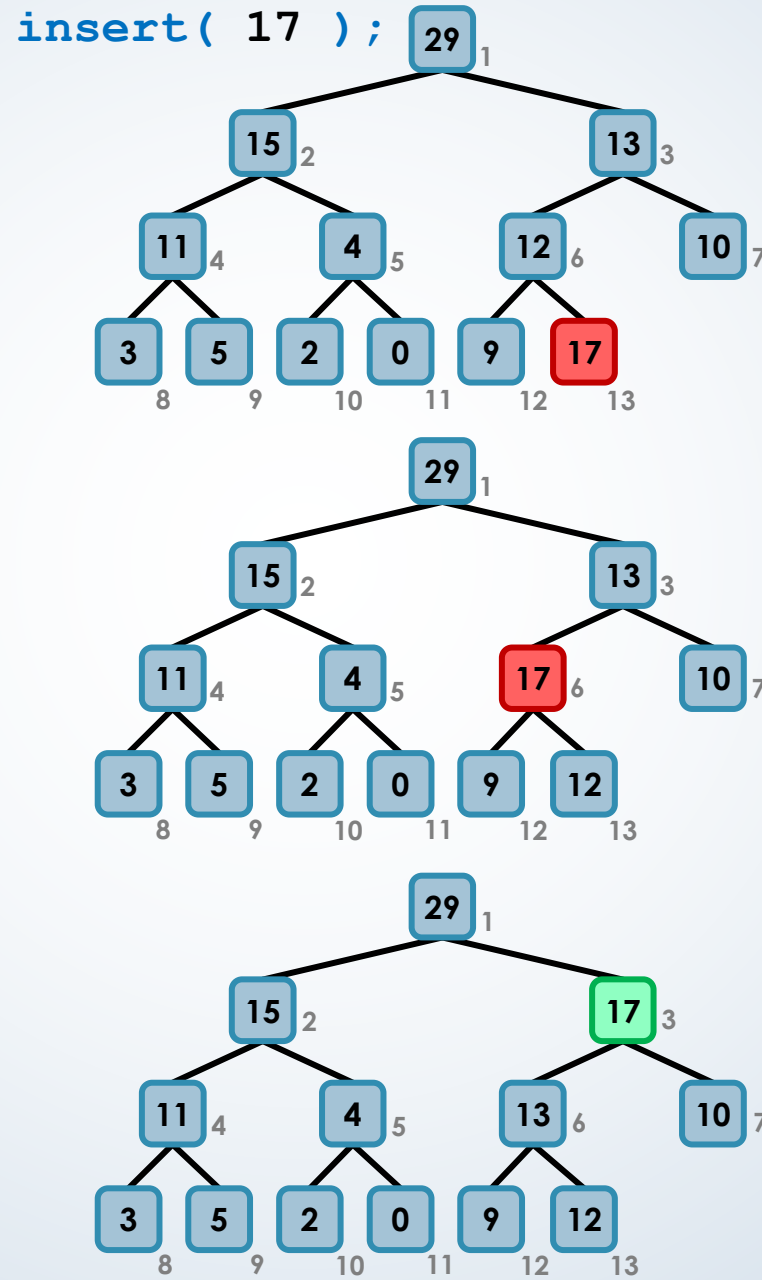
public Key delete() {
    Key max = pq[1];

    // troca a raiz com o último
    exchange( 1, n-- );

    // afunda a nova raiz
    sink( 1 );

    pq[n + 1] = null;
    return max;
}

```



Filas de Prioridades

Exercícios de Implementação (Heaps Multivias)

- **Exercício i10.1:** No projeto **ESDC4Aula10**, é fornecido o esqueleto de uma classe chamada **TernaryMaxPriorityQueue**. Você deve considerar que um Heap Ternário Máximo mantém as chaves da fila de prioridades. Sua tarefa é implementar os algoritmos dos métodos **swim** (flutuar) e **sink** (afundar) do Heap Ternário Máximo, viabilizando assim a utilização dessa classe. Os testes de unidade para a aceitação ou não do que deve ser feito foram implementados. No Heap Ternário Máximo, dado um nó k , seus três filhos estão nas posições $3k - 1$, $3k$, e $3k + 1$ e seu pai na posição $\lfloor (k + 1)/3 \rfloor$ para as posições 1 a $n - 1$ do array, sendo n o tamanho do mesmo.
- **Exercício i10.2:** No projeto **ESDC4Aula10**, é fornecido o esqueleto de uma classe chamada **DAryMaxPriorityQueue**. Você deve considerar que um Heap d -ário Máximo mantém as chaves da fila de prioridades. Sua tarefa é implementar os algoritmos dos métodos **swim** (flutuar) e **sink** (afundar) do Heap d -ário Máximo, viabilizando assim a utilização dessa classe. Os testes de unidade para a aceitação ou não do que deve ser feito foram implementados. No Heap d -ário Máximo, cada nó possui no máximo d filhos. A determinação da posição dos filhos e do pai fica por sua conta. Novamente, as posições válidas dentro do array são 1 a $n - 1$, sendo n o tamanho do mesmo.

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

WEISS, M. A. Data Structures and **Algorithm Analysis in Java**. 3. ed. Pearson Education: New Jersey, 2012. 614 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos – Teoria e Prática**. 3. ed. São Paulo: GEN LTC, 2012. 1292 p.