**Universidade de São Paulo**
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
SCC0202 e SCC0502 – Algoritmos e Estruturas de Dados I

# Projeto 3 - Árvores *Treaps*

**Professor:** Dr. Marcelo Garcia Manzato (mmanzato@icmc.usp.br)
**Estagiários PAE:** André (andrezanon@usp.br) e Luan (luanssouza@usp.br)

# Árvores com Prioridades

## Descrição

Este trabalho consiste na implementação de uma árvore *Treap* (combinação de *Tree* + *Heap*), que são estruturas de dados balanceadas em que os nós contém uma chave e uma prioridade, conforme a Figura 1. As prioridades são geradas aleatoriamente como uma forma de manter o balanceamento e as operações de inserção, remoção e busca em tempo de $O(log(N))$.

Assim como árvores binárias, as *treaps*, mantém a mesma lógica na qual nós com chaves maiores são colocados a direita e nós com chaves menores são colocados a esquerda. No entanto, o balanceamento é realizado conforme a prioridade dos mesmos.

As Figuras 2 e 3 ilustram as operações de remoção e inserção, além das rotações necessárias para manter o balanceamento. Para a operação de remoção, deve-se fazer com que o nó a ser removido vire uma folha ou semi-folha, logo, para que isso aconteça e a estrutura mantenha-se a mesma, as rotações deverão manter um mesmo sentido, assim, diferente do que acontece no exemplo na Figura 3, em que são realizadas remoções sucessivas a direita e esquerda (EE), **seu programa irá rotacionar sempre somente para a esquerda, até o nó a ser removido vire uma folha**. Já para a operação de inserção, assim como na Figura 2, **deve seguir uma *Max Heap*, em que nós com maior valor são priorizados**.

A implementação da sua *Treap* deverá conter as seguintes operações:

- Inserção de um número, em $O(log(N))$
- Busca de um número, em $O(log(N))$
- Remoção de um número, em $O(log(N))$
- Impressão da árvore em-ordem, pós-ordem, pré-ordem ou em largura, em $O(N)$

## Entradas

Seu programa deve ler um conjunto de valores da entrada padrão. A entrada contém um conjunto de $n$ linhas, seguidas por respectivas $n$ operações. Cada linha é formatada da seguinte maneira:

- *insercao chave prioridade*, em que tanto *chave* quanto *prioridade* são números.
- *remocao chave prioridade*, em que tanto *chave* quanto *prioridade* são números.
- *buscar chave* em que deverá ser retornado se existe ou não um número na árvore. Caso exista o retorno será 1, caso contrário 0.
- *impressao modo* em que deverá ser impressa a árvore conforme o modo especificado. Os modos podem ser *ordem*; *posordem*; *preordem* e *largura*.

## Saídas

Seu programa deve imprimir as linhas correspondentes para cada operação, se necessário. Para inserção, quando a chave já existir deverá ser impresso a frase "Elemento ja existente", já quando a chave não existe para a remoção, deverá ser impresso "Chave nao localizada".

A impressão de cada nó deve ser realizada como uma tupla, logo, no formato $(info, prio)$ no qual, $info$ representa a chave e $prio$ a prioridade do nó.

## Observações

- O uso de árvores é obrigatório. Para as funções de impressão, pode ser utilizado também estruturas como pilha e fila.
- Somente as bibliotecas *stdio.h*, *stdlib.h* e *string.h* podem ser utilizadas.

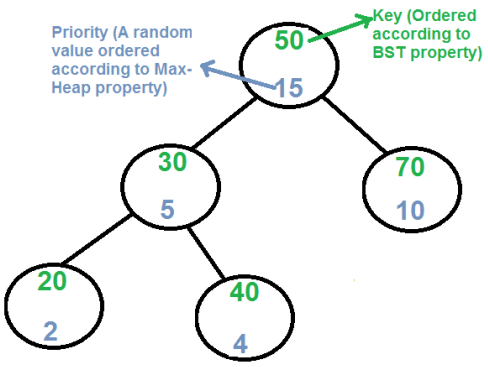
## Entradas

```
10
insercao 30 5
insercao 20 2
insercao 40 4
insercao 70 10
insercao 50 15
insercao 20 18
impressao preordem
impressao largura
remocao 50
impressao ordem
```

## Saídas

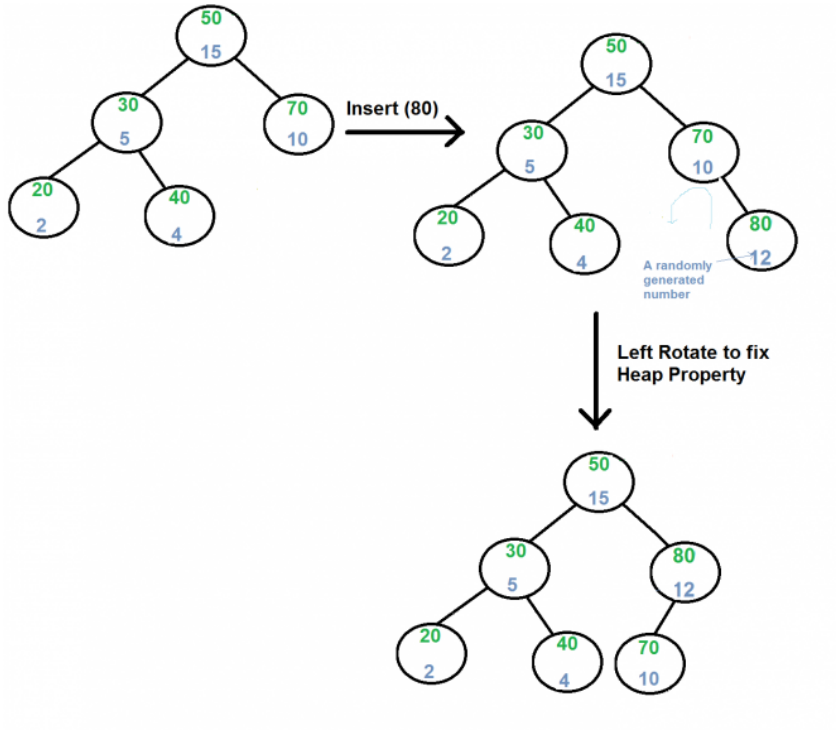
```
Elemento ja existente
(50, 15) (30, 5) (20, 2) (40, 4) (70, 10)
(50, 15) (30, 5) (70, 10) (20, 2) (40, 4)
(20, 2) (30, 5) (40, 4) (70, 10)
```

**Figura 1** – Ilustração de uma árvore *Treap*



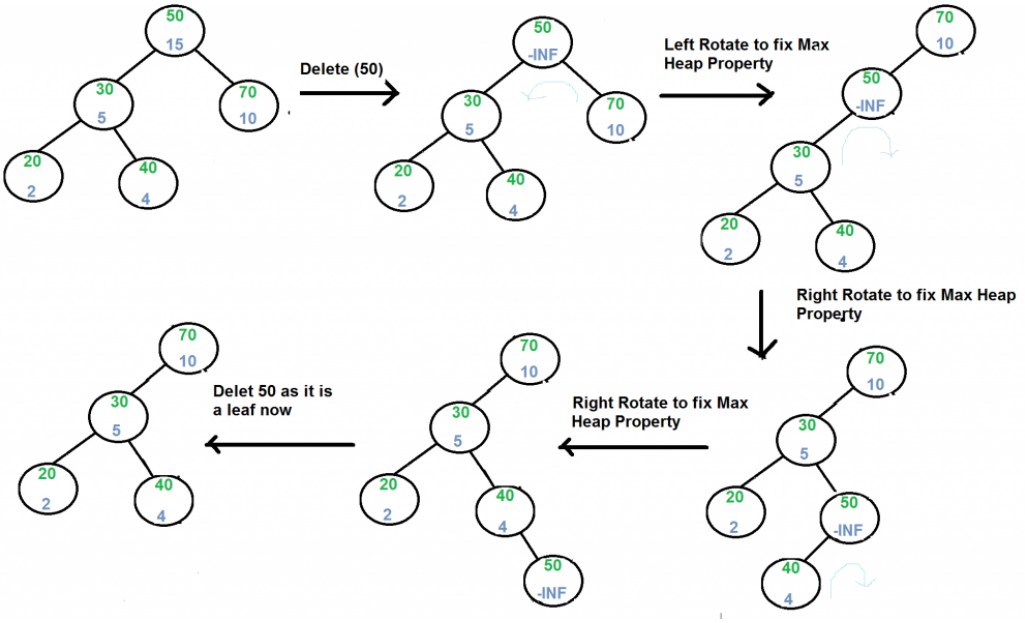

Fonte: https://www.geeksforgeeks.org/treap-a-randomized-binary-search-tree/

**Figura 2** – Ilustração de inserção em uma árvore *Treap*




Fonte: https://www.geeksforgeeks.org/treap-a-randomized-binary-search-tree/

**Figura 3** – Ilustração de remoção em uma árvore *Treap*




Fonte: https://www.geeksforgeeks.org/treap-a-randomized-binary-search-tree/