Parte 1 - Exercícios Heaps e Treaps (tipos especiais de Árvores Binárias)

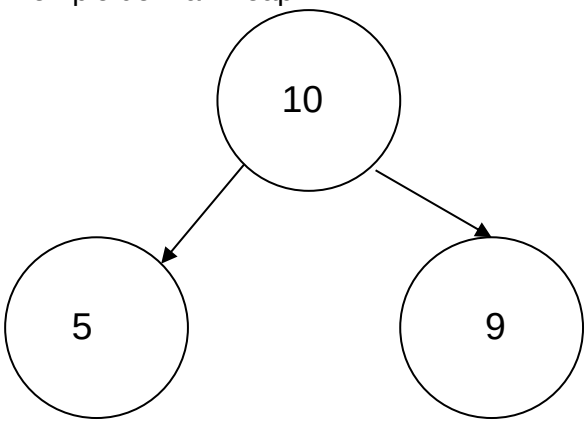
1) Explique e diferencie Heaps e Treaps.
Heaps são um tipo de arvores binarias, porem não um arvore binaria de pesquisa. Na BST, os valores à esquerda de um nó são menores do que ele e os valores à direita são maiores. No Heap, ambos são menores ou iguais. Heaps que seguem essa propriedade são Heaps Máximos porque o maior valor sempre está na raiz. Há também Heaps Mínimos, onde o nó tem valor sempre menor ou igual ao seus filhos. Neste caso, o menor valor sempre está na raiz. Além disso, as Heaps são estruturas balanceadas e completas/quase-completas.

Uma estrutura de dados Treap é basicamente uma combinação de um árvore de pesquisa binária e um heap. A treap e a árvore de busca binária aleatória são duas formas intimamente relacionadas de estruturas de dados de árvore de busca binária que mantêm um conjunto dinâmico de chaves ordenadas e permitem buscas binárias entre as chaves. Após qualquer seqüência de inserções e remoções de chaves, a forma da árvore é uma variável aleatória com a mesma distribuição de probabilidade de uma árvore binária aleatória; em particular, com alta probabilidade, sua altura é proporcional ao logaritmo do número de chaves, de modo que cada operação de pesquisa, inserção ou exclusão leva um tempo logarítmico para ser executada.

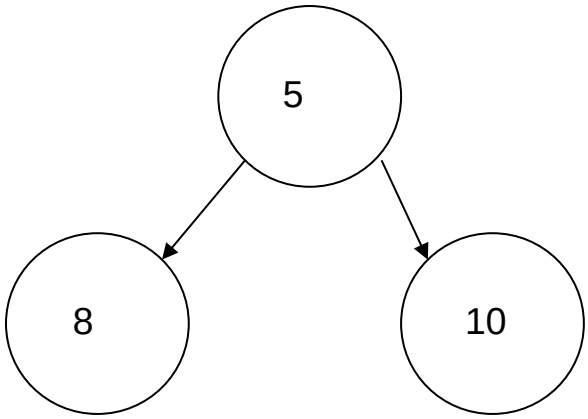
2) O que são Heaps máxima e mínima? Explique cada qual por meio de um exemplo.

A Heaps Maxima são organizadas de modo que os nós pais possuam sempre valores maiores ou iguais que os nós filhos.
Ja na Heaps Minimas, os valores dos nós filhos são maiores que do nó pai.

Exemplo de Max Heap:



Exemplo de Min Heap:



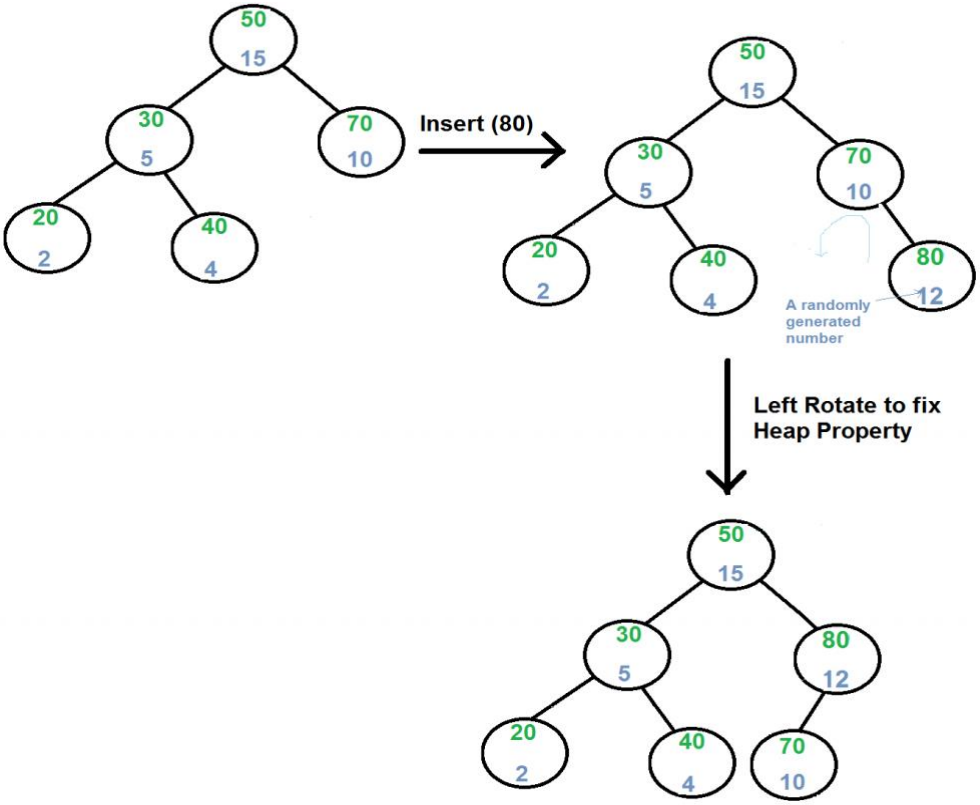
3) Considere na Figura 1 (no fim desta lista) a ordem de inserção de dados no array correspondente a Heap máxima. Logo, faça:
a) Explique como se dá a inserção cuja organização obedece ao método cima-para-baixo?
b) Supondo a mesma sequência de inserção, demonstre como seria formar uma Heap mínima pelo mesmo método cima-para-baixo?

A) No momento de inserir um novo nó, é feito uma comparação entre o nó a ser inserido com nó pai em questão, caso o novo nó seja maior que o pai, ocorre uma troca de posição entre eles. Esse processo se repete até o novo nó ser menor que um nó pai, e, caso ele seja o maior valor da árvore, ele se tornará o root. Essa comparação e troca é chamada de Heapfy.

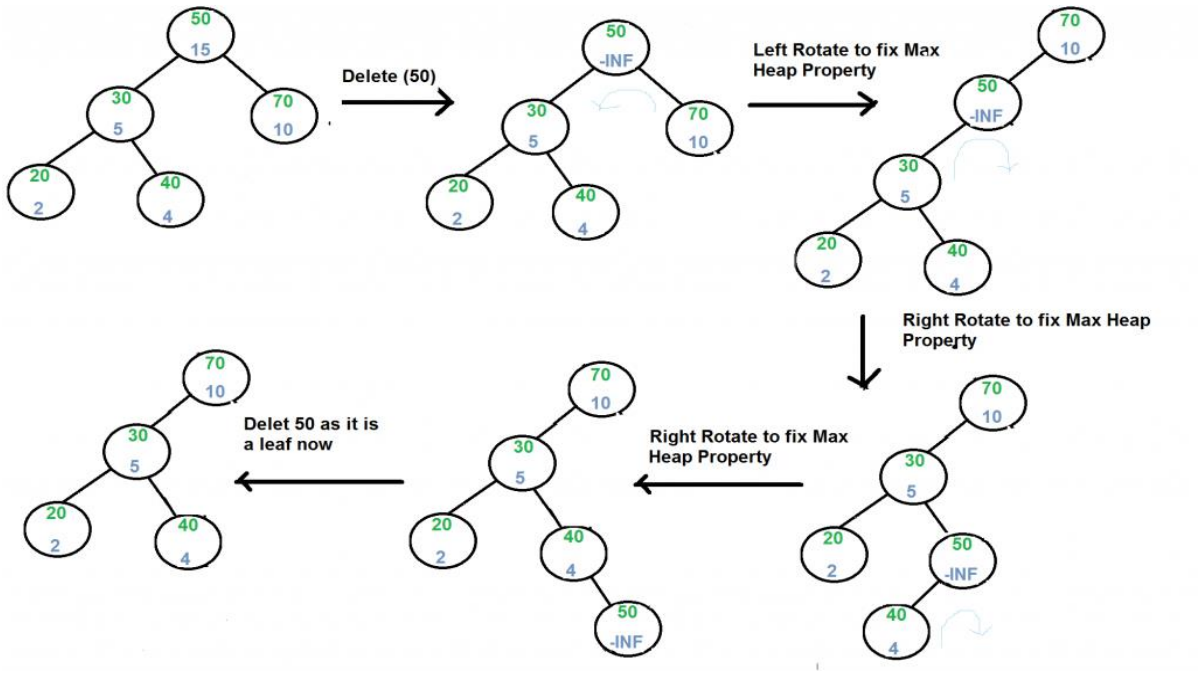
B) Numa Heap Mínima ocorre o mesmo processo de comparação e troca, a única diferença é que a troca entre o novo nó e o nó pai só ocorre caso o primeiro for menor que o pai. Dessa forma, o nó pai será menor que os filhos.

4) Demonstre uma Treap e como se dá a inserção e remoção de dados em um exemplo (dica: leitura recomendada do livro Adam Drozdek, Seção 6.10).

Para **inserir** uma nova chave x no treap, gere uma prioridade aleatória y para x. Pesquisa binária por x na árvore e cria um novo nó na posição da folha onde a pesquisa binária determina que x deve existir um nó para. Então, desde que x não seja a raiz da árvore e tenha um número de prioridade maior que seu pai z, execute uma rotação de árvore que inverta a relação pai-filho entre x e z.



Para excluir um nó x da treap, remova-o se for uma folha da árvore. Se x tiver um único filho, z, remova xda árvore e torne z-o filho do pai de x(ou faça z a raiz da árvore se x não tiver pai). Por fim, se x tiver dois filhos, troca sua posição na árvore com seu sucessor imediato z na ordem ordenada, resultando em um dos casos anteriores. Nesse último caso, a troca pode violar a propriedade de ordenação de heap para z, portanto, rotações adicionais podem precisar ser executadas para restaurar essa propriedade.



Parte 2

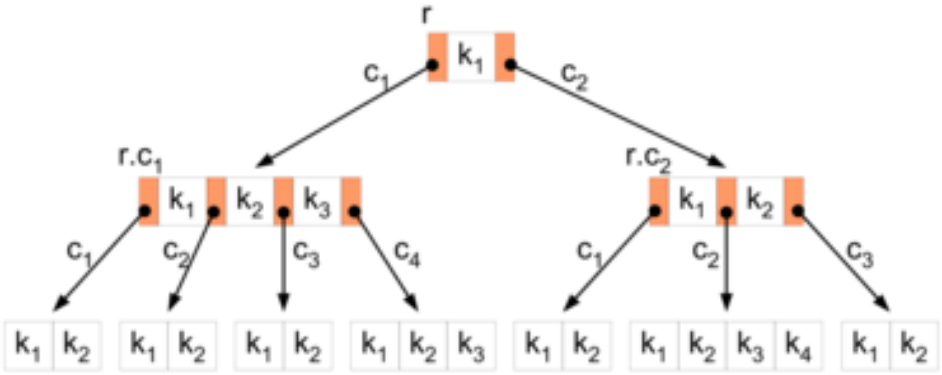
Árvores múltiplas B:

Em ciência da computação, uma **árvore B** é uma estrutura de dados em árvore, auto-balanceada, que armazena dados classificados e permite pesquisas, acesso sequencial, inserções e remoções em tempo logarítmico. A árvore B é uma generalização de uma árvore de pesquisa binária em que um nó pode ter mais que dois filhos. Diferente das árvores de pesquisa binária auto-balanceadas, a árvore B é bem adaptada para sistemas de armazenamento que leem e escrevem blocos de dados relativamente grandes. É normalmente usada em bancos de dados e sistemas de arquivos.

Para qualquer inteiro positivo par M, uma árvore B (B-tree) de ordem M é uma árvore com as seguintes propriedades:

- cada nó contém no máximo M−1 chaves,
- a raiz contém no mínimo 2 chaves e cada um dos demais nós contém no mínimo M/2 chaves,
- cada nó que não seja uma folha tem um filho para cada uma de suas chaves,
- todos os caminhos da raiz até uma folha têm o mesmo comprimento (ou seja, a árvore é *perfeitamente balanceada*).

Exemplo de uma Arvore B:



Exemplo de uma função de busca de valor na arvore B:

```
BTreeNode* BTreeNode::search(int k)
{
    // Find the first key greater than or equal to k
    int i = 0;
    while (i < n && k > keys[i])
        i++;

    // If the found key is equal to k, return this node
    if (keys[i] == k)
        return this;

    // If the key is not found here and this is a leaf node
    if (leaf == true)
```

```
        return NULL;

    // Go to the appropriate child
    return C[i]->search(k);
}
```

Árvores múltiplas B*

Uma árvore B* é uma estrutura de dados na ciência da computação e uma variação da árvore B.Esta apresenta mecanismos de inserção, remoção e busca muito semelhantes aos realizados em árvores B, mas com a diferença em que a técnica de redistribuição de chaves também é empregada durante as operações de inserção. Dessa maneira a operação de split pode ser adiada até que duas páginas irmãs estejam completamente cheias e, a partir daí, o conteúdo dessas páginas irmãs é redistribuído entre três páginas (uma nova criada e as duas páginas irmãs anteriormente cheias).

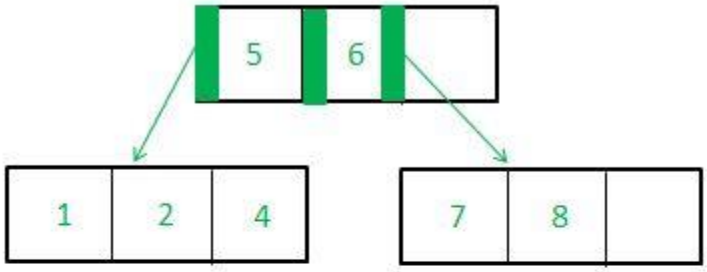
Tal técnica é conhecida como divisão *two-to-three split*, ou no português *divisão de dois para três*, que proporciona propriedades diferentes das árvores B na qual utiliza a divisão usual conhecida como *one-to-two split*. A grande melhoria direta proporcionada por essa abordagem é o melhor aproveitamento de espaço do arquivo, pois, no pior caso, cada página apresenta no mínimo 2/3 do número máximo de chaves que esta pode armazenar, enquanto na divisão usual cada página apresenta, no pior caso, metade do número máximo de chaves.

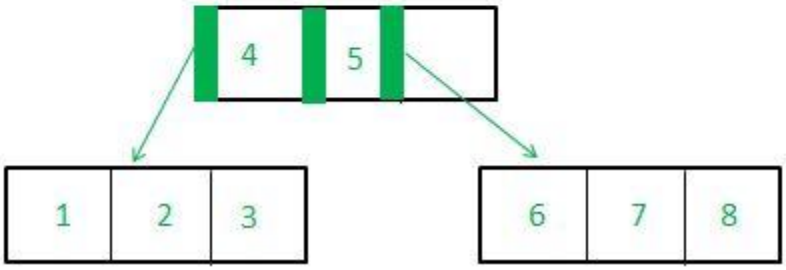
Segundo a definição de uma **árvore B*** de ordem m apresenta as seguintes propriedades:

1. Cada página apresenta no máximo m páginas filhas
2. Uma página folha contém pelo menos ⌊(2m-1)/3⌋ chaves e no máximo m-1
3. Todas as páginas folha estão no mesmo nível
4. Toda página, exceto a raiz e as folhas possuem no máximo (2m-1)/3 descendentes
5. Uma página não folha com k páginas filhas possui k-1 chaves

A vantagem de usar árvores B* em vez de árvores B é um recurso exclusivo chamado divisão "dois para três". Com isso, o número mínimo de chaves em cada nó não é metade do número máximo, mas dois terços dele, tornando os dados muito mais compactos. No entanto, a desvantagem disso é uma operação de exclusão complexa.

Exemplo de uma Arvore Multipla B*:





Exemplo de uma função de busca na Arvore Multipla B*:

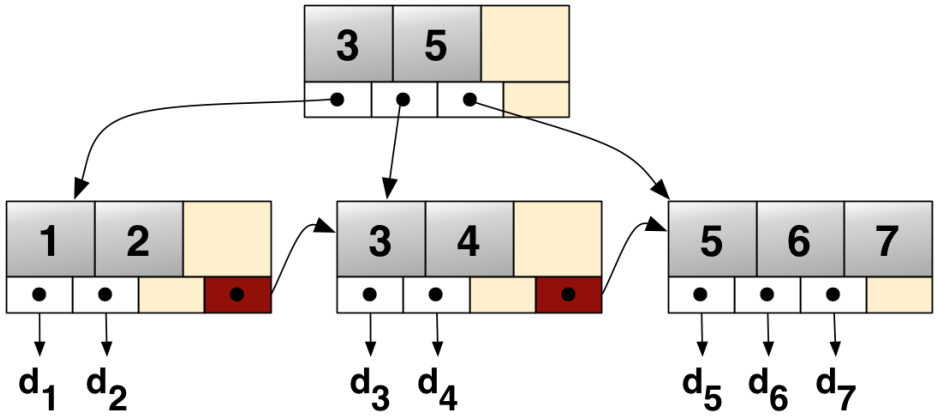
```
struct node* searchforleaf(struct node* root, int k, struct node* parent, int chindex){
    if (root) {
        if (root->isleaf == 1)
            return root;
        else {
            int i;
            if (k < root->key[0])
                root = searchforleaf(root->child[0], k, root, 0);

            else
            {
                for (i = 0; i < root->n; i++)
                    if (root->key[i] > k)
                        root = searchforleaf(root->child[i], k, root, i);
                if (root->key[i - 1] < k)
                    root = searchforleaf(root->child[i], k, root, i);
            }
        }
    }
    else {
        struct node* newleaf = new struct node;
        newleaf->isleaf = 1;
        newleaf->n = 0;
        parent->child[chindex] = newleaf;
        newleaf->parent = parent;
        return newleaf;
    }
}
```

Arvores Multiplas B+:
Na ciência da computação uma árvore B+ é uma estrutura de dados do tipo árvore derivada das árvores B, mas com uma forma diferente de armazenamento de suas chaves.
A idéia inicial desta variação da árvore B é manter todas as chaves de busca em seus nós folha de maneira que o acesso sequencial ordenado das chaves de busca seja um processo mais eficiente do que em árvores B. Obviamente tal acesso sequencial também é possivel nestas, mas, para isso seria necessário algum algoritmo semelhante ao percurso em ordem realizado numa árvore binária.
Vantagens:

- Uma árvore B+ com níveis 'n' pode armazenar mais entradas em seus nós internos em comparação com uma árvore B com os mesmos níveis 'l'. Isso acentua a melhoria significativa feita no tempo de pesquisa de qualquer chave.
- Os dados armazenados em uma árvore B+ podem ser acessados sequencialmente e diretamente.

Porém, suas **desvantagens** são que contém nós não folha maiores, dificultando seu gerenciamento de índices, a inserção de registros pode provocar overflow em um bloco e a remoção de registros pode provocar underflow em um bloco.

Exemplo de uma Arvore Multipla B+:



Exemplo em codigo do metodo de busca:

```
void BPTree::search(int x) {
  if (root == NULL) {
    cout << "Tree is empty\n";
  } else {
    Node *cursor = root;
    while (cursor->IS_LEAF == false) {
      for (int i = 0; i < cursor->size; i++) {
```

```
        if (x < cursor->key[i]) {
          cursor = cursor->ptr[i];
          break;
        }
        if (i == cursor->size - 1) {
          cursor = cursor->ptr[i + 1];
          break;
        }
      }
    }
    for (int i = 0; i < cursor->size; i++) {
      if (cursor->key[i] == x) {
        cout << "Found\n";
        return;
      }
    }
    cout << "Not found\n";
  }
}
```