**Cây B-Tree: bật T**

- Có số khóa: 2\*t-1

- Có số nút: =số khóa +1 =2t

|  |
| --- |
| - Tất cả các khóa trong cây B phải khác biệt.  - Tất cả các nút lá phải ở cùng một cấp độ.  - Một nút có tối đa N khóa thì phải có tối đa N+1 nút con  - Mỗi khóa trong một nút bên trong cây B có một cây con bên trái và một cây con bên phải. Tất cả các khóa phụ bên trái là NHỎ HƠN khóa đó và tất cả các khóa phụ bên phải là LỚN HƠN khóa đó.  - Các khóa trong một nút được lưu trữ theo thứ tự sắp xếp từ nhỏ nhất đến lớn nhất. |

**1. Thêm khóa vào B cây**

- Thêm vào ở nút lá. Để tìm nút lá thì đi từ nút gốc xuống.

- Xuất phát từ root

+ Nếu chưa đầy => Thêm vào nút

/

/

/

+ Ngược lại đã đầy=> Tạo thêm nút mới để thêm vào

/

/

/

26.10 B-Tree

A B-Tree is a self-balancing search tree. A B-Tree maintains the following properties,

* All leaves are on the same level.
* All internal nodes except the root have at most m (nonempty) children and at least ceil(m/2) children.
* The root has at least 2 children if it is not a leaf, and at most m children.

In this exercise, you need to implement the insert() method of a B-Tree where it will maintain the above properties. We will provide you all other additional methods with the main functions.

You may visit the following link to see visually how a B-Tree insertion method works. Note that you need to check the "Preemtive Split / Merge (Even max degree only)" box to match the output with our program. <https://www.cs.usfca.edu/~galles/visualization/BTree.html>

You have the following public methods,

* BTree(unsigned int minNumOfDegree) - the Constructor, takes a parameter for the minimum number of degree. So, a node can have maximum (minNumOfDegree \* 2 - 1) keys.
* ~BTree() - Destructor;
* void insert(T key) - You need to implement this method. You need to insert the key in the proper position of the tree. If the node is full, you may need to create a new node or split your child node.
* bool remove(T key) - Removes the provided key from the tree. True, if the key is found in the tree and delete successfully.
* bool searchKey(T key) - Search a key.
* void traverse() - Preorder traversal of the tree.

You have the following private methods to help to implement those public methods,

* void initializeNode(BNode \*node) - Used for initializing nodes.
* void freeNode(BNode \*node) - Recursive function called by destructor to deallocate all the allocated memory.
* unsigned findIndex(BNode \*node, T key) - Finds the index of a key in a node.
* std::pair<bnode\*, unsigned> search(T key) - Function to find a key in the tree. returnValue.first is the node the item is in. returnValue.second is the correct index in that node's key array.</bnode
* unsigned nodeInsert(BNode \*node, T key) - Inserts a key into a node.
* T nodeDelete(BNode \*node, unsigned index) - Deletes the key at a given index from a node.
* void splitChild(BNode \*node, int split\_index) - Function for splitting nodes that are too full.
* char mergeChildren(BNode \*parent, unsigned index) - Merges two children of a node at a given index into one child.
* char fixChildSize(BNode \*parent, unsigned index) - Makes sure the child of a node at a specified index has >= minDegree items.
* bool lessThan(T a, T b);
* void printNode(BNode \*node);

*Note: You don't need to use all private methods to implement your insert method.*

**Input:**

2

8

10 20 5 6 12 30 7 17

The first input represents the minimum number of degrees. So, each node may have at least 2 degrees or at most 4 degrees (2 \* minDegree), and at most 3 keys (maxDegree -1). The second line represents the total number of keys and the last line represents the corresponding keys that you need to insert.

**Output:**

Preorder traverse:

10 20

5 6 7

12 17

30

So, your root node will have 2 keys (10, 20), and it has 3 child nodes. Check the visualization link to understand clearly.