Lab assessment 7 - B-Tree

Name : Jagtap Mahesh Vilas

Reg No. 24MCS1017

Problem Statement:

You are working on designing a simple file management system where efficient searching and insertion of keys is critical. To optimize the process, your system uses a B-tree to store file identifiers (keys). The file identifiers must be organized such that the tree maintains its balance properties after every insertion, ensuring fast search operations.

Your task is to insert a sequence of file identifiers into a B-tree and then perform a search to locate specific file identifiers.

Instructions:

1. **Insert the following file identifiers in order into an initially empty B-tree**:

   8, 9, 10, 11, 15, 20, 17

   - The B-tree order (degree) is 3, meaning each node can contain at most 2 keys.
   - After every insertion, visualize how the structure of the B-tree evolves, including any necessary splitting of nodes.
   - Carefully describe how nodes split and how the tree maintains its balance after each insertion.

   Search for the following file identifiers in the B-tree:

   - Search for **15** and **17** and narrate how the B-tree traversal process works step-by-step.
   - Show the traversal path taken during the search process and mention the comparisons made at each node.

Answer:

1.Inserting the following file identifiers in order into an initially empty B-tree:

Initial B-tree (Empty)

[ ]

Insert 8:

[8]

Insert 9:

```
[8, 9]
```

Insert 10:

The node is full (2 keys), so we need to split.

Split the node into two nodes and promote the middle key (9).

```
   [9]
  /   \
[8]   [10]
```

Insert 11:

```
   [9]
  /   \
[8]   [10, 11]
```

Insert 15:

The right node is full (2 keys), so we need to split.

Split the node into two nodes and promote the middle key (11).

```
   [9, 11]
  /   |   \
[8]  [10]  [15]
```

Insert 20:

```
   [9, 11]
  /   |   \
```

```
        [8]  [10]  [15, 20]
```

Insert 17:

The right node is full (2 keys), so we need to split.

Split the node into two nodes and promote the middle key (17).

```
   [9, 11, 17]
  /   |   |   \
[8]  [10]  [15]  [20]
```

The root node is now full (3 keys), so we need to split.

Split the root node and promote the middle key (11).

```
    [11]
   /   \
 [9]    [17]
 / \    / \
[8] [10] [15] [20]
```

2.Searching for the following file identifiers in the B-tree:

I. Search for 15:

Start at the root node [11].

15 > 11, move to the right child [17].

15 < 17, move to the left child [15].

Found 15.

Traversal Path: [11] -> [17] -> [15]

II. Search for 17:

Start at the root node [11].

17 > 11, move to the right child [17].

Found 17.

Traversal Path: [11] -> [17]

CODE:

```
#include <iostream>
#include <queue>
using namespace std;

class BTreeNode {
public:
    int *keys;
    int t;
    BTreeNode **C;
    int n;
    bool leaf;
    BTreeNode(int _t, bool _leaf);
    void traverseLevelWise();
    void splitChild(int i, BTreeNode *y);
    void insertNonFull(int k);
    BTreeNode* search(int k);

    friend class BTree;
};

class BTree {
public:
    BTreeNode *root;
    int t;

    BTree(int _t) {
        root = NULL;
        t = _t;
    }

    void traverseLevelWise() {
        if (root != NULL) root->traverseLevelWise();
    }

    void insert(int k);
```

```
    void searchKey(int k);
};

BTreeNode::BTreeNode(int t1, bool leaf1) {
    t = t1;
    leaf = leaf1;
    keys = new int[2 * t - 1];
    C = new BTreeNode *[2 * t];
    n = 0;
}

void BTreeNode::traverseLevelWise() {
    queue<BTreeNode*> q;
    q.push(this);

    while (!q.empty()) {
        int size = q.size();
        for (int i = 0; i < size; i++) {
            BTreeNode* node = q.front();
            q.pop();

            cout << "[";
            for (int j = 0; j < node->n; j++) {
                cout << node->keys[j];
                if (j != node->n - 1) cout << ",";
            }
            cout << "] ";

            if (!node->leaf) {
                for (int j = 0; j <= node->n; j++) {
                    q.push(node->C[j]);
                }
            }
        }
        cout << endl;
    }
}

void BTree::insert(int k) {
    cout << "Inserting " << k << endl;

    if (root == NULL) {
        root = new BTreeNode(t, true);
        root->keys[0] = k;
```

```
        root->n = 1;
    } else {
        if (root->n == 2 * t - 1) {
            BTreeNode *s = new BTreeNode(t, false);
            s->C[0] = root;
            s->splitChild(0, root);
            int i = 0;
            if (s->keys[0] < k)
                i++;
            s->C[i]->insertNonFull(k);
            root = s;
        } else {
            root->insertNonFull(k);
        }
    }

    traverseLevelWise();
}

void BTreeNode::insertNonFull(int k) {
    int i = n - 1;

    if (leaf == true) {
        while (i >= 0 && keys[i] > k) {
            keys[i + 1] = keys[i];
            i--;
        }
        keys[i + 1] = k;
        n = n + 1;
    } else {
        while (i >= 0 && keys[i] > k)
            i--;

        if (C[i + 1]->n == 2 * t - 1) {
            splitChild(i + 1, C[i + 1]);
            if (keys[i + 1] < k)
                i++;
        }
        C[i + 1]->insertNonFull(k);
    }
}

void BTreeNode::splitChild(int i, BTreeNode *y) {
    BTreeNode *z = new BTreeNode(y->t, y->leaf);
```

```
    z->n = t - 1;

    for (int j = 0; j < t - 1; j++)
        z->keys[j] = y->keys[j + t];

    if (!y->leaf) {
        for (int j = 0; j < t; j++)
            z->C[j] = y->C[j + t];
    }

    y->n = t - 1;

    for (int j = n; j >= i + 1; j--)
        C[j + 1] = C[j];

    C[i + 1] = z;

    for (int j = n - 1; j >= i; j--)
        keys[j + 1] = keys[j];

    keys[i] = y->keys[t - 1];
    n = n + 1;

    cout << "Splitting at " << keys[i] << endl;
}

BTreeNode* BTreeNode::search(int k) {
    int i = 0;
    while (i < n && k > keys[i])
        i++;

    if (i < n && keys[i] == k)
        return this;

    if (leaf == true)
        return NULL;

    return C[i]->search(k);
}

void BTree::searchKey(int k) {
    BTreeNode* result = root == NULL ? NULL : root->search(k);

    if (result != NULL)
```

```
        cout << "Found " << k << endl;
    else
        cout << k << " not found" << endl;
}

int main() {
    BTree t(2);
    t.insert(8);
    t.insert(9);
    t.insert(10);
    t.insert(11);
    t.insert(15);
    t.insert(20);
    t.insert(17);
    t.searchKey(15);
    t.searchKey(17);

    return 0;
}
```

OUTPUT:

```
Inserting 8
[8]
Inserting 9
[8,9]
Inserting 10
[8,9,10]
Inserting 11
Splitting at 9
[9]
[8] [10,11]
Inserting 15
[9]
[8] [10,11,15]
Inserting 20
Splitting at 11
[9,11]
[8] [10] [15,20]
Inserting 17
[9,11]
[8] [10] [15,17,20]
Found 15
Found 17
```