#include<iostream>

using namespace std;

//Ds project Group members are Ali ul haq , Usman zahid , Abbas imran , Dawood rafi.

//I will write down my steps according to classes that i will create & above of each class.

//Thank you.

/\*

\* step 1 create a class for btreenode that class will point the node of the tree.

\* Declaring some variable.

\* 1.1 integer of keys this will be the keys of our tree

\* 1.2 t for minimum degree

\* 1.3 c for child pointers

\* 1.4 n for number of total keys in tree.

\* 1.5 leaf it will be true when node is leaf .if its root it will be false.

\* 2. we will create a class constructor with two parameter.

\* 2.1 nonfull function for inserting a new key in tree.

\* 2.2 split child functoin it will split y child of the current node.

\* 2.3 to traverse all nodes connected with the root.

\* 2.4 search method to search key connected with the root node.

\* 2.5 we will make this class friend so we can use this class method in other class or classes.

\*

\*/

class BTreeNode

{

int\* keys;

int t;

BTreeNode\*\* C;

int n;

bool leaf;

public:

BTreeNode(int val1, bool leaf); // Constructor

void insertNonFull(int k);

void splitChild(int i, BTreeNode\* y);

void traverse();

BTreeNode\* search(int k);

friend class BTree;

};

/\*

Btree class

1 create a btreenode pointer as root because it will point to root of our b tree.

1.1 creating a constructor and deaclaring the root with null. because when u create a root it will be null.

1.2 now if the root does not empty we will traverse the root.

1.3 implementing search function. passing 1 argument which is k .later we will use this.

1.4 this insert method will insert new key in our btree.

\*/

class BTree

{

BTreeNode\* root;

int t;

public:

BTree(int \_t)

{

root = NULL; t = \_t;

}

void traverse()

{

if (root != NULL) root->traverse();

}

BTreeNode\* search(int k)

{

return (root == NULL) ? NULL : root->search(k);

}

void insert(int k);

};

/\*

1 we will crete a constructor for btreenode. which we created at first.

1.1 we will pass two argument in this constructor. t1 and leaf1. //==>this will copy leaf property

1.2 n is intiliazing keys to 0.

\*/

BTreeNode::BTreeNode(int t1, bool leaf1)

{

t = t1;

leaf = leaf1;

keys = new int[2 \* t - 1];

C = new BTreeNode \* [2 \* t];

n = 0;

}

/\*

\* 1 traverse all nodes in subtree with this node.

\* 1.1 loop for checking if there is no leaf of child. then traversing the subtree rooted with child.

\* 1.2 printing subtree rooted with last child

\*/

void BTreeNode::traverse()

{

int i;

for (i = 0; i < n; i++)

{

if (leaf == false)

C[i]->traverse();

cout << " " << keys[i];

}

if (leaf == false)

C[i]->traverse();

}

/\*

\* 1 function for searcing key in subtree rooted with node.

\* 1.1 as we now we used "n" for keys. so we will compare it and find first key greater or equal to k.

\* 1.2 if the key found is equal to k .we will return this node.

\* 1.3 if key is not found and if we are finding in on leaf node we have to redirect it to approprite node.

\*/

BTreeNode\* BTreeNode::search(int k)

{

int i = 0;

while (i < n && k > keys[i])

i++;

if (keys[i] == k)

return this;

if (leaf == true)

return NULL;

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

}

/\*

\* 1 this function inserts a new key in this B-Tree

\* 1.1 we will chk first the tree is empty or not.

\* 1.2 then we will alocate memory for root or root node.

\* 1.3 if three is not empty then and root is full then tree will grow in height. and we have to create a new root. and the previous old root will become its child

\* 1.4 split old root and move 1 key to new root.

\* 2 new root have two child. one child will have new key.

\*/

void BTree::insert(int k)

{

if (root == NULL)

{

root = new BTreeNode(t, true);

root->keys[0] = k; // Insert key

root->n = 1; // Update number of keys in root

}

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);

}

}

/\*

\* 1 create and this function will insert key in node

\* 1.1 create a loop to insert new key. and this loop will move greater keys one place ahead.

\* 1.2 inserting key at new find loaction.

\* 1.3finding the child node to have the new key if the node is not leaf.

\* 1.4 findig right child and if full then spliting it.

\*/

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);

}

}

/\*

1 create a function name split child.

1.1 create new node to store keys of y.

1.2 for loop for copying the keys

1.3 copy last children.

2 for loop for space for new child

2.1 also linking new child to the node.

2.2 incrementing keys to the node.

\*/

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 == false)

{

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;

}

/\*

main function

1 creating object of our btree class which is tree.

1.1 inserting value in our b tree.

1.2 traversing it

1.3 searcing it with variable k

\*/

int main()

{

BTree tree(3);

tree.insert(10);

tree.insert(20);

tree.insert(5);

tree.insert(6);

tree.insert(12);

tree.insert(30);

tree.insert(7);

tree.insert(17);

tree.insert(1);

tree.insert(55);

tree.insert(69);

tree.insert(91);

tree.insert(93);

tree.insert(97);

tree.insert(99);

tree.insert(51);

tree.insert(29);

tree.insert(4);

cout << "Traversal of the constucted tree is : ";

tree.traverse();

int search = 6;

(tree.search(search) != NULL) ? cout <<endl<< search <<": Is present" : cout <<endl << search << " : Is not present";

search = 4;

(tree.search(search) != NULL) ? cout << endl << search << ": Is present" : cout << endl << search << " : Is not present";

search = 28;

(tree.search(search) != NULL) ? cout << endl << search << ": Is present" : cout << endl << search << " : Is not present";

search = 56;

(tree.search(search) != NULL) ? cout << endl << search << ": Is present" : cout << endl << search << " : Is not present";

search = 99;

(tree.search(search) != NULL) ? cout << endl << search << ": Is present" : cout << endl << search << " : Is not present";

search = 64;

(tree.search(search) != NULL) ? cout << endl << search << ": Is present" : cout << endl << search << " : Is not present";

search = 93;

(tree.search(search) != NULL) ? cout << endl << search << ": Is present" : cout << endl << search << " : Is not present";

return 0;

}