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

class Node
{
	string *keys;
	int t;
	Node **C;
	int n;
	bool leaf;
public:
	Node(int _t, bool _leaf);

	void insertNonFull(string k);

	void splitChild(int i, Node *y);

	void print();

	Node *search(string k);

friend class BTree;
};

class BTree
{
	Node *root;
	int t;
public:
	BTree(int _t)
	{ root = NULL; t = _t; }

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

	Node* search(string k)
	{ return (root == NULL)? NULL : root->search(k); }

	void insert(string k);
};

Node::Node(int t1, bool leaf1)
{
	t = t1;
	leaf = leaf1;

	keys = new string[2*t-1];
	C = new Node *[2*t];

	n = 0;
}
```

```
void BTree::insert(string k)
{
	if (root == NULL)
	{
		root = new Node(t, true);
		root->keys[0] = k;
		root->n = 1;
	}
	else
	{
		if (root->n == 2*t-1)
		{
			Node *s = new Node(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);
	}
}

void Node::insertNonFull(string 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 Node::splitChild(int i, Node *y)
{
		Node *z = new Node(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;
}

void Node::print()
{
		int i;
		for (i = 0; i < n; i++)
		{
			if (leaf == false)
				C[i]->print();
			cout << " " << keys[i];
		}

		if (leaf == false)
			C[i]->print();
}

Node *Node::search(string 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);
}

int main()
{
	BTree t(2);
	int n;
	cin >> n;
	for (int i = 0; i < n; i++) {
		string x;
		cin >> x;
		t.insert(x);
	}

	cout << "In-order traversal:";
	t.print();

	string k = "a";
	(t.search(k) != NULL)? cout << "\n" << k << " is found" : cout << "\n" << k << " is not
found";

   k = "b";
	(t.search(k) != NULL)? cout << "\n" << k << " is found" : cout << "\n" << k << " is not
found";

	return 0;
}
```