Câu hỏi 3

Chính xác

Chấm điểm của 2,50

In this exercise, you need to complete method **search** for BTree. Method declaration as

```
bool search(const K& key, Node*& outNodePtr, int& outEntryIdx);
```

When implementing, you need to assign appropriate value to **outNodePtr, outEntryIdx** which will be used for checking your code.

Method **toString()** of class Node is hidden. If you want to see the result in your IDE, you need to provide your previous implementation of these methods.

The test code use method insert() but it is already implemented. You don't need to worry about it on this exercise. You may need it to run this exercise in your IDE, in that case, you can take a look at BTree-4.

You can provide your implementation in this exercise directly, using PreCheck button to see the result in some examples without any penalty.

```cpp
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

template <class K, class D, int M> // K: key, D: data, M: order
class BTree {
    /// Convention: Left sub-tree < Root's key <= Right sub-tree

public:
    class Entry;
    class Node;

private:
    Node* root;

public:
    BTree() : root(0) { }
    ~BTree() { }

    ////////////////////////////////////////////////
    ///            CLASS `Entry`                 ///
    ////////////////////////////////////////////////
public:
    class Entry {
    private:
        K key;
        D data; // accept all type
        Node* rightPtr;
        friend class BTree<K, D, M>;

    public:
        Entry(K key = K{}, D value = D{}) : key(key), data(value), rightPtr(0) {}
        ~Entry() { }
        string toString() {
            stringstream ss;
            ss << "<"
               << this->key << ","
               << this->data
               << ">";
            return ss.str();
        }
    };

    ////////////////////////////////////////////////
    ///             CLASS `Node`                 ///
    ////////////////////////////////////////////////
public:
    class Node {
    private:
        Node* firstPtr;
        int numEntries;
        Entry entries[M-1];

        friend class BTree<K, D, M>;

    public:
        Node() : firstPtr(0), numEntries(0) {};

        ~Node() { }
        string toString() {
            // hidden code
            return ss.str();
        }
        bool isFull() {
            return (numEntries >= M-1);
        }
    };

    ////////////////////////////////////////////////////////////
    ///         CLASS `BTree`: method implementation         ///
    ////////////////////////////////////////////////////////////
public:
    string toStringPreOrder();
    void insert(const K& key, const D& data);

    /// BEGIN STUDENT CODE
    /// ================================================
    /// search(key, outNodePtr, outEntryIdx)
    bool search(const K& key, Node*& outNodePtr, int& outEntryIdx) {

    }
```

```
    /// END STUDENT CODE
};
```

For example:

Test	Result

`int keys[] = {78, 21, 14};`
`int size = sizeof(keys) / sizeof(int);`
 	
`BTree<int, int, 5> bTree;`
`for (int idx = 0; idx < size; idx++) {`
`  bTree.insert(keys[idx], idx + 5);`
`}`
 	
`BTree<int, int, 5>::Node* outNode;`
`int outEntryIdx;`
`if (bTree.search(11, outNode, outEntryIdx)) {`
`  cout << "FOUND" << endl;`
`  cout << outNode->toString() << endl;`
`  cout << "Index: " << outEntryIdx;`
`}`
`else {`
`  cout << "NOT FOUND";`
`}`
	`NOT FOUND`

`int keys[] = {78, 21, 14};`
`int size = sizeof(keys) / sizeof(int);`
 	
`BTree<int, int, 5> bTree;`
`for (int idx = 0; idx < size; idx++) {`
`  bTree.insert(keys[idx], idx + 5);`
`}`
 	
`BTree<int, int, 5>::Node* outNode;`
`int outEntryIdx;`
`if (bTree.search(21, outNode, outEntryIdx)) {`
`  cout << "FOUND" << endl;`
`  cout << outNode->toString() << endl;`
`  cout << "Index: " << outEntryIdx;`
`}`
`else {`
`  cout << "NOT FOUND";`
`}`
	`FOUND`
	`[(3)<14,7><21,6><78,5>]`
	`Index: 1`

Answer: (penalty regime: , 5, 10, 15, ... %)

Reset answer

```
1  /// BEGIN STUDENT CODE
2  /// ====================================================
3  // You may need some helping functions here
4
5  bool search(const K& key, Node*& outNodePtr, int& outEntryIdx) {
6      // Start searching from the root node
7      Node* currentNode = root;
8
9      while (currentNode) {
10         // Find the position to insert the key
11         int low = 0;
12         int high = currentNode->numEntries - 1;
13         int mid;
14
15         while (low <= high) {
16             mid = (low + high) / 2;
17             if (key
```

```
                // Key found
                outNodePtr = currentNode;
                outEntryIdx = mid;
                return true;
            } else if (key < currentNode->entries[mid].key) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }

        // Key not found, move to the appropriate child node
        if (key < currentNode->entries[0].key) {
            currentNode = currentNode->firstPtr;
        } else {
            currentNode = currentNode->entries[high].rightPtr;
        }
    }

    // Key not found in the tree
    outNodePtr = nullptr;
    outEntryIdx = -1;
    return false;
}

/// END STUDENT CODE
```

Precheck Kiểm tra

	Test	Expected	Got	

✔	int keys[] = {78, 21, 14};
	int size = sizeof(keys) / sizeof(int);
	
	BTree<int, int, 5> bTree;
	for (int idx = 0; idx < size; idx++) {
	bTree.insert(keys[idx], idx + 5);
	}
	
	BTree<int, int, 5>::Node* outNode;
	int outEntryIdx;
	if (bTree.search(11, outNode, outEntryIdx)) {
	cout << "FOUND" << endl;
	cout << outNode->toString() << endl;
	cout << "Index: " << outEntryIdx;
	}
	else {
	cout << "NOT FOUND";
	}
		NOT FOUND	NOT FOUND	✔

✔	int keys[] = {78, 21, 14};
	int size = sizeof(keys) / sizeof(int);
	
	BTree<int, int, 5> bTree;
	for (int idx = 0; idx < size; idx++) {
	bTree.insert(keys[idx], idx + 5);
	}
	
	BTree<int, int, 5>::Node* outNode;
	int outEntryIdx;
	if (bTree.search(21, outNode, outEntryIdx)) {
	cout << "FOUND" << endl;
	cout << outNode->toString() << endl;
	cout << "Index: " << outEntryIdx;
	}
	else {
	cout << "NOT FOUND";
	}
		FOUND
		[(3)<14,7><21,6><78,5>]
		Index: 1
			FOUND
			[(3)<14,7><21,6><78,5>]
			Index: 1
				✔

Passed all tests! ✔

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn