mgechev / **binary-search-tree-cpp.cpp**

Last active 16 days ago

Simple implementation of binary search tree in C++.

binary-search-tree-cpp.cpp

```cpp
#include <iostream>
#include <math.h>
using namespace std;

template <class T>
struct Node {
    T value;
    Node *left;
    Node *right;

    Node(T val) {
        this->value = val;
    }

    Node(T val, Node<T> left, Node<T> right) {
        this->value = val;
        this->left = left;
        this->right = right;
    }
};

template <class T>
class BST {

    private:
    Node<T> *root;

    void addHelper(Node<T> *root, T val) {
        if (root->value > val) {
            if (!root->left) {
                root->left = new Node<T>(val);
            } else {
                addHelper(root->left, val);
            }
        } else {
            if (!root->right) {
                root->right = new Node<T>(val);
            } else {
                addHelper(root->right, val);
            }
        }
    }

    void printHelper(Node<T> *root) {
        if (!root) return;
        printHelper(root->left);
        cout<<root->value<<' ';
        printHelper(root->right);
    }

    int nodesCountHelper(Node<T> *root) {
        if (!root) return 0;
        else return 1 + nodesCountHelper(root->left) + nodesCountHelper(root->right);
    }
```

```
55
56      int heightHelper(Node<T> *root) {
57          if (!root) return 0;
58          else return 1 + max(heightHelper(root->left), heightHelper(root->right));
59      }
60
61      void printMaxPathHelper(Node<T> *root) {
62          if (!root) return;
63          cout<<root->value<<' ';
64          if (heightHelper(root->left) > heightHelper(root->right)) {
65              printMaxPathHelper(root->left);
66          } else {
67              printMaxPathHelper(root->right);
68          }
69      }
70
71      bool deleteValueHelper(Node<T>* parent, Node<T>* current, T value) {
72          if (!current) return false;
73          if (current->value == value) {
74              if (current->left == NULL || current->right == NULL) {
75                  Node<T>* temp = current->left;
76                  if (current->right) temp = current->right;
77                  if (parent) {
78                      if (parent->left == current) {
79                          parent->left = temp;
80                      } else {
81                          parent->right = temp;
82                      }
83                  } else {
84                      this->root = temp;
85                  }
86              } else {
87                  Node<T>* validSubs = current->right;
88                  while (validSubs->left) {
89                      validSubs = validSubs->left;
90                  }
91                  T temp = current->value;
92                  current->value = validSubs->value;
93                  validSubs->value = temp;
94                  return deleteValueHelper(current, current->right, temp);
95              }
96              delete current;
97              return true;
98          }
99          return deleteValueHelper(current, current->left, value) ||
100                deleteValueHelper(current, current->right, value);
101     }
102
103     public:
104     void add(T val) {
105         if (root) {
106             this->addHelper(root, val);
107         } else {
108             root = new Node<T>(val);
109         }
110     }
111
112     void print() {
113         printHelper(this->root);
114     }
115
116     int nodesCount() {
117         return nodesCountHelper(root);
118     }
119
120     int height() {
121         return heightHelper(this->root);
```

```
    }

    void printMaxPath() {
        printMaxPathHelper(this->root);
    }

    bool deleteValue(T value) {
        return this->deleteValueHelper(NULL, this->root, value);
    }
};

int main() {
    BST<int> *bst = new BST<int>();
    bst->add(11);
    bst->add(1);
    bst->add(6);
    bst->add(-1);
    bst->add(-10);
    bst->add(100);
    bst->print();
    cout<<endl;
    cout<<"Nodes count: "<<bst->nodesCount();
    cout<<endl;
    cout<<"Height: "<<bst->height();
    cout<<endl;
    cout<<"Max path: ";
    bst->printMaxPath();
    cout<<endl;
    bst->deleteValue(11);
    cout<<"11 removed: ";
    bst->print();
    cout<<endl;
    cout<<"1 removed: ";
    bst->deleteValue(1);
    bst->print();
    cout<<endl;
    cout<<"-1 removed: ";
    bst->deleteValue(-1);
    bst->print();
    cout<<endl;
    cout<<"6 removed: ";
    bst->deleteValue(6);
    bst->print();
    cout<<endl;
    cout<<"-10 removed: ";
    bst->deleteValue(-10);
    bst->print();
    cout<<endl;
    cout<<"100 removed: ";
    bst->deleteValue(100);
    bst->print();
    cout<<endl;
    return 0;
}
```


chrisbangun commented on Jul 27, 2015

Hey,
Thanks for sharing your implementation. It is a neat implementation. However, I have a question about the time complexity of your deletion. I saw in your deleteValueHelper function, you returned deleteValueHelper(current, current->left, value) || deleteValueHelper(current, current->right, value);
won't it be redundant? instead, you can check first whether the given value is bigger or lesser than your current.
Thanks

ilyahascyb commented on Nov 20, 2015

лапша

rprtr258 commented on Apr 9, 2016

Добавьте в конструктор Node обнуление указателей
Node(T val) {
this->value = val;
this->left = this->right = nullptr; //C++11
//или this->left = this->right = NULL; //C++98
}

lnrsoft commented on May 26, 2016

well done mate, nice implementation

ralphjoseph commented on Aug 24, 2016

very neat..I concur with chrisbangun

RodionOrets commented on Sep 22, 2016

binary search tree without balancing.....)

seymoutr commented on Oct 10, 2016

very helpful thank you.

seymoutr commented on Oct 12, 2016

One question: What would be the easiest way to implement a deep copy constructor for the BST?

ghrua commented on Oct 19, 2016

Does the line **15** should be this?

```
// add * after Node<T>
Node(T val, Node<T> * left, Node<T> * right)
```

JACKIEZHAOKAI commented on Nov 9, 2016

good implementation

dmakweba commented on Jan 12, 2017

Hi guys, nice to meet you all especially seen the implementation of this BST in c++.
I am very junior on Algorithm and I would like anyone to assist me with basics to implement this BST and others in C++. The main challenge is that I am not good in C++ as well in such away that I don't know how to separate source file, main file and header file. Please, if anyone can redirect/point me where I can have the skills I will appreciate.
Rgds.

jhalakpatel commented on Feb 27, 2017

@dmakweba you can check out my implementation on BST here

xiuquanw commented on Oct 26, 2017

make sure to initialize pointers in the constructor, otherwise, it will throw segmentation fault.
The line 12 should be this

```
Node(int val) : value(val) , left(NULL) , right(NULL) {}
```

suhussai commented on Oct 29, 2017 • edited

To add to **@xiuquanw**'s comment, the root variable in the BST class should be initialized to NULL. So instead of this:

```
Node<T> *root;
```

Add this:

```
Node<T> *root = NULL;
```

This is to ensure that when setting up the BST, the root is properly created in line 108.

supratikn commented on Nov 13, 2017

memory leaks

MarkKoz commented on Dec 7, 2017 • edited

`root` does not have to be initialised in C++ 11. Because no user-defined constructor is given for `BST`, the compiler implicitly defines a default constructor. During value-initialisation, fields will be zero-initialised because of the presence of this implicitly defined default constructor. Because pointers are a scalar type, they will be initialised to 0 i.e. `NULL`