Universidade Federal de Ouro Preto
Lecture Notes
Graphs

Prof. Rodrigo Silva

April 12, 2023

1 Binary Search Trees

1.1 Python

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        if not self.root:
            self.root = Node(data)
        else:
            self._insert(data, self.root)

    def _insert(self, data, current_node):
        if data < current_node.data:
            if not current_node.left:
                current_node.left = Node(data)
            else:
                self._insert(data, current_node.left)
        elif data > current_node.data:
            if not current_node.right:
                current_node.right = Node(data)
            else:
                self._insert(data, current_node.right)
        else:
            print("Value already in tree.")

    def search(self, data):
        if self.root:
            found = self._search(data, self.root)
            if found:
                return True
            return False
        else:
            return None

    def _search(self, data, current_node):
        if data == current_node.data:
            return True
        elif data < current_node.data and current_node.left:
            return self._search(data, current_node.left)
        elif data > current_node.data and current_node.right:
            return self._search(data, current_node.right)
        return False
```

```
48
49      def inorder(self, node):
50          if node is not None:
51              self.inorder(node.left)
52              print(node.data)
53              self.inorder(node.right)
54
55  if __name__ == '__main__':
56
57      bst = BST()
58
59      bst.insert(5)
60      bst.insert(3)
61      bst.insert(7)
62      bst.insert(1)
63      bst.insert(9)
64
65      print(bst.search(1)) # Output: True
66      print(bst.search(10)) # Output: False
67
68      bst.inorder(bst.root)
```

Listing 1: Binary Search Tree in Python

1.2 C++

```
1  #include <iostream>
2  #include <memory>
3
4  struct Node {
5      int data;
6      std::unique_ptr<Node> left;
7      std::unique_ptr<Node> right;
8
9      Node(int data) : data(data), left(nullptr), right(nullptr) {}
10 };
11
12 class BST {
13 private:
14     std::unique_ptr<Node> root;
15
16     void insert(std::unique_ptr<Node>& node, int data) {
17         if (!node) {
18             node = std::make_unique<Node>(data);
19         } else if (data < node->data) {
20             insert(node->left, data);
21         } else if (data > node->data) {
22             insert(node->right, data);
23         } else {
24             std::cout << "Value already in tree." << std::endl;
25         }
26     }
27
28     bool search(const std::unique_ptr<Node>& node, int data) const {
29         if (!node) {
30             return false;
31         } else if (data == node->data) {
32             return true;
33         } else if (data < node->data) {
34             return search(node->left, data);
35         } else {
36             return search(node->right, data);
37         }
38     }
39
```

```
    void inorder(const std::unique_ptr<Node>& node) const {
        if (node) {
            inorder(node->left);
            std::cout << node->data << std::endl;
            inorder(node->right);
        }
    }

public:
    BST() : root(nullptr) {}

    void insert(int data) {
        insert(root, data);
    }

    bool search(int data) const {
        return search(root, data);
    }

    void inorder() const {
        inorder(root);
    }
};


int main() {
    BST bst;

    bst.insert(5);
    bst.insert(3);
    bst.insert(7);
    bst.insert(1);
    bst.insert(9);

    std::cout << bst.search(1) << std::endl; // Output: 1 (true)
    std::cout << bst.search(10) << std::endl; // Output: 0 (false)

    bst.inorder();
    // Output:
    // 1
    // 3
    // 5
    // 7
    // 9

    return 0;
}
```

Listing 2: Binary Search Tree in Python