Data Structures

Binary Search Trees

CS 225

September 18, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Exam 2 September 26-28

Covers material up to and including today

Practice exam will be released this afternoon or tomorrow

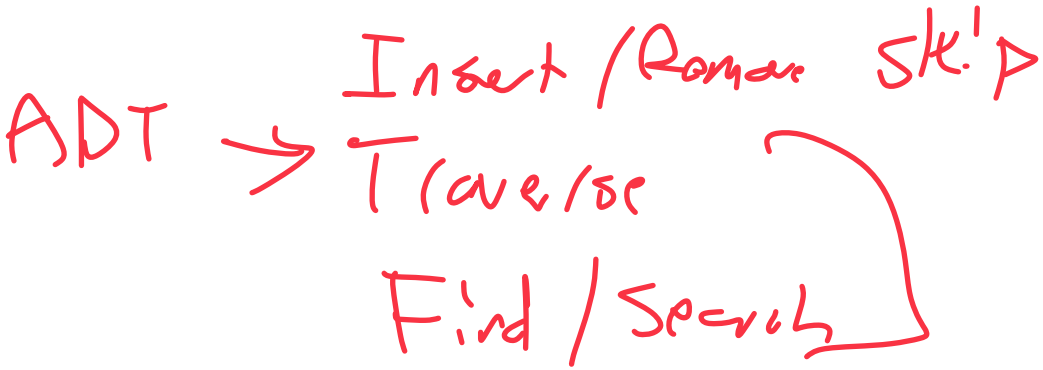
Look at exam page on website

All material!

Learning Objectives

Review binary trees

Extend binary trees into binary *search* trees



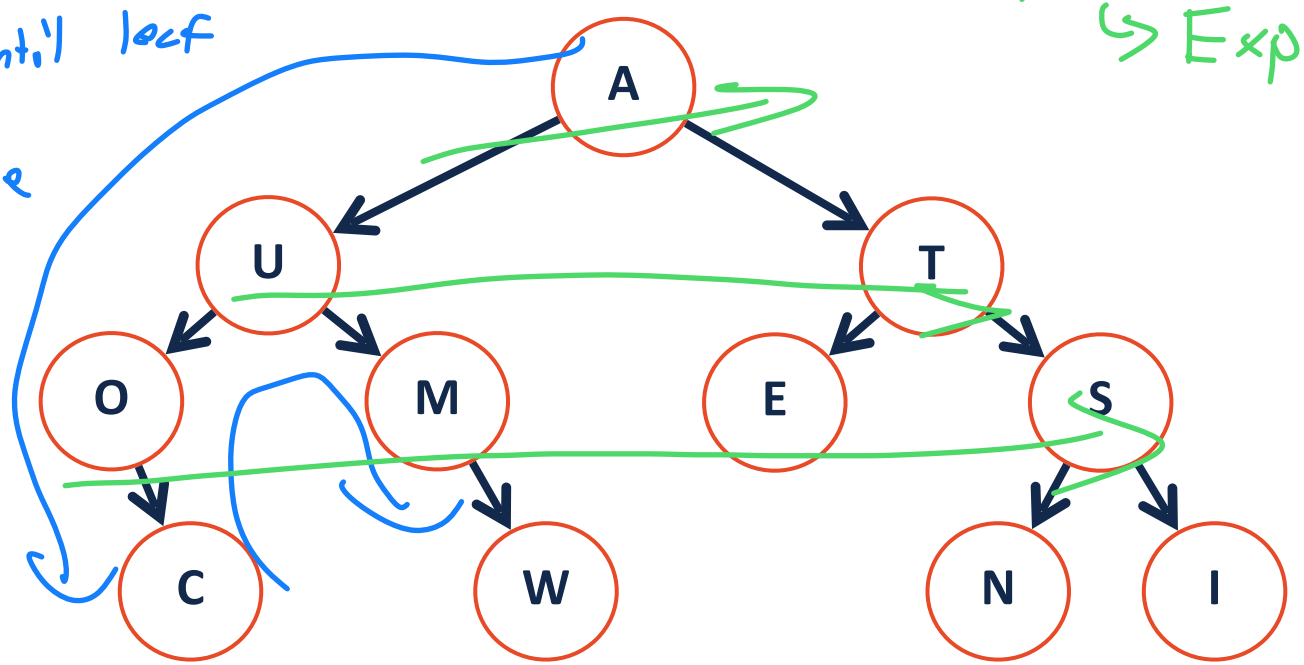
Tree Search

Traversal vs search
↳ visit every node ↳ stop when it find it
O(n)
2 Break case

There are two main approaches to searching a binary tree:

1) Depth-first search
Descend branch until leaf
Recurse somewhere else

2) Breadth first search
↳ Explore on complete level



Depth First Search

Explore as far along one path as possible before backtracking

1) Make a stack initialized to root

2) while stack is not empty

* → tmp = stack.pop() remove top element

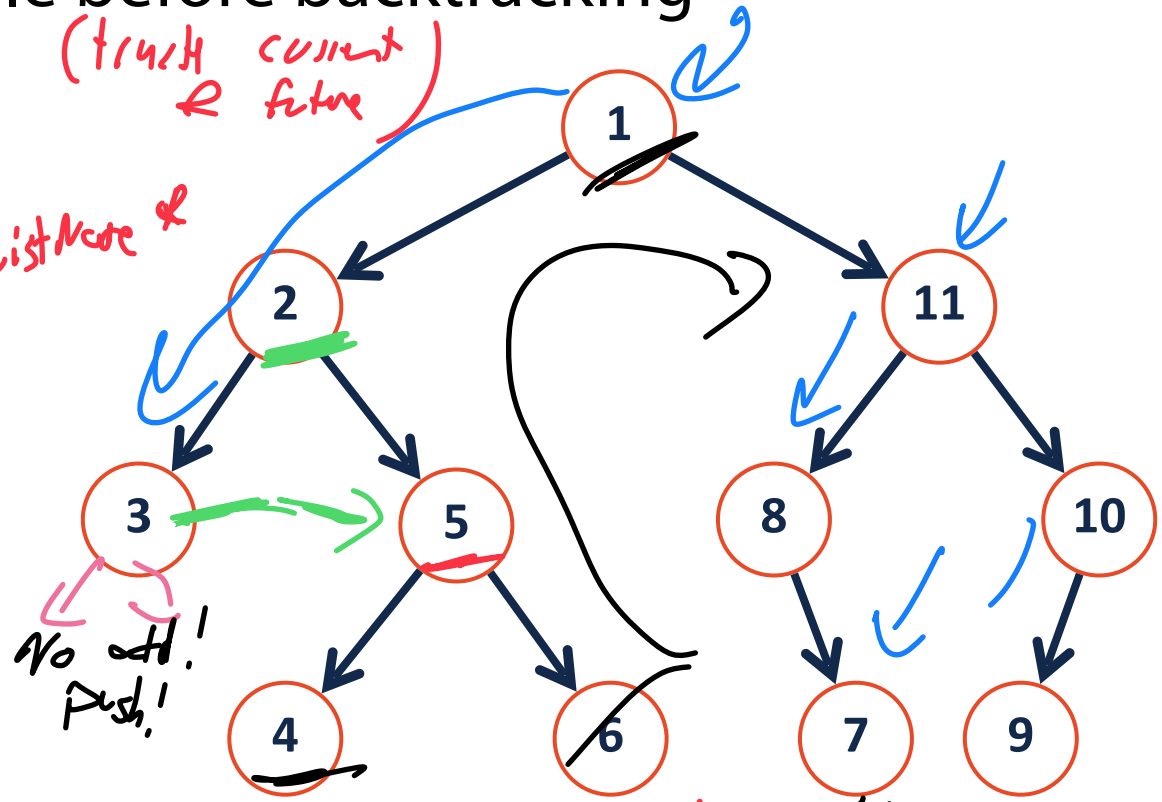
print(tmp)

stack.push(tmp->right)

stack.push(tmp->left)

(track current & future)

ListNode *



pops

1 11 2 5 3 6 4

stack: 1 11 2 5 3 6 8 7 9

print: 1 2 3 5 4 6 11 8 7 10 9 ← preorder

Breadth First Search

Fully explore depth i before exploring depth i+1

pre
recursion
in
recursion
post

1) Make a queue init to root

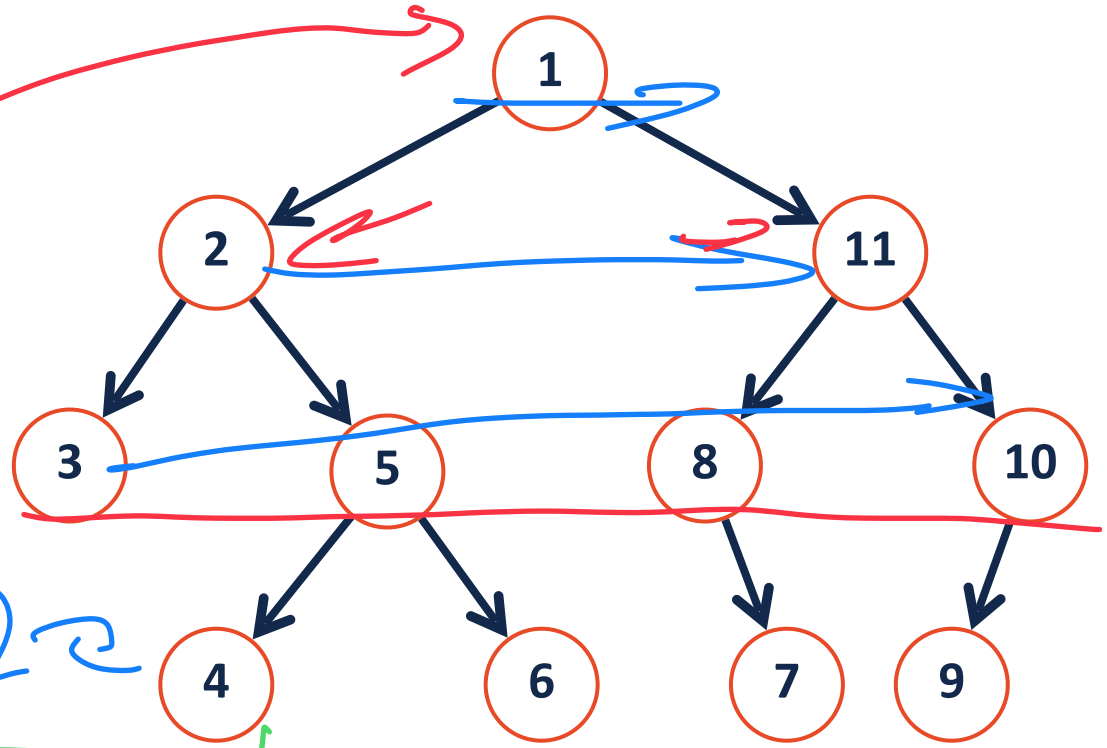
2) while queue is not empty

tmp = queue.dequeue() // free node *

print tmp

queue.enqueue(tmp->left)

queue.enqueue(tmp->right)



Locked entire level

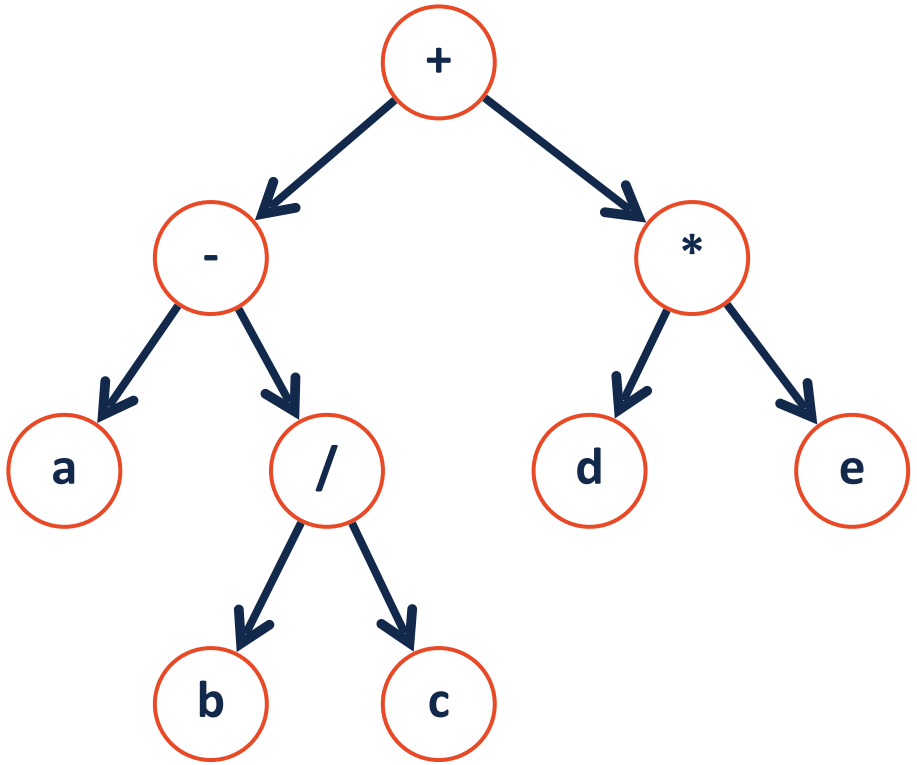
tmp: 2

front

Queue: ~~1~~ ~~2~~ ~~11~~ ~~3~~ ~~5~~ ~~8~~ ~~10~~ 4 6 7 9

print: 1 2 11 3 5 8 10 4 6 7 9

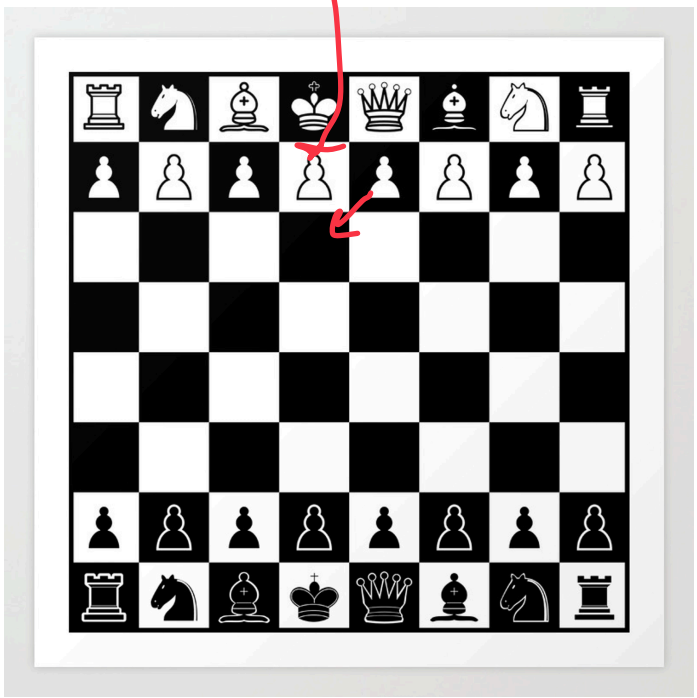
Level-Order Traversal




```
1  template<class T>
2  void BinaryTree<T>::lOrder(TreeNode * root)
3  {
4
5      Queue<TreeNode*> q;
6      q.enqueue(root);
7
8      while( q.empty() == False){
9
10         TreeNode* temp = q.head();
11         process(temp);
12
13         q.dequeue();
14
15         q.enqueue(temp->left);
16         q.enqueue(temp->right);
17
18     }
19 }
```

What search algorithm is best?

The average 'branch factor' for a game of chess is ~31. If you were searching a decision tree for chess, which search algorithm would you use?



DFS

- Most single moves are bad
- We need to look down to pick a good path
- Better space complexity

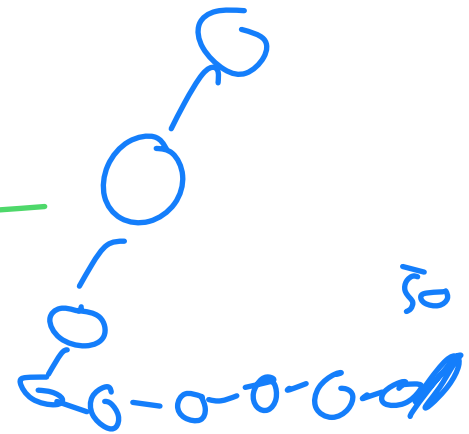
↳ 31 at level 1
31^2 at level 2
31^3 3

BFS

- Consider all moves
- Pick best one
- The only one that works
↳ Get us answer faster

Iterative deepening DFS

1) Do DFS up to depth X
2) If not found, expand X



Tree Search

How can we improve our ability to search a binary tree?

↳ Make it sorted

What do we trade in order to do so?

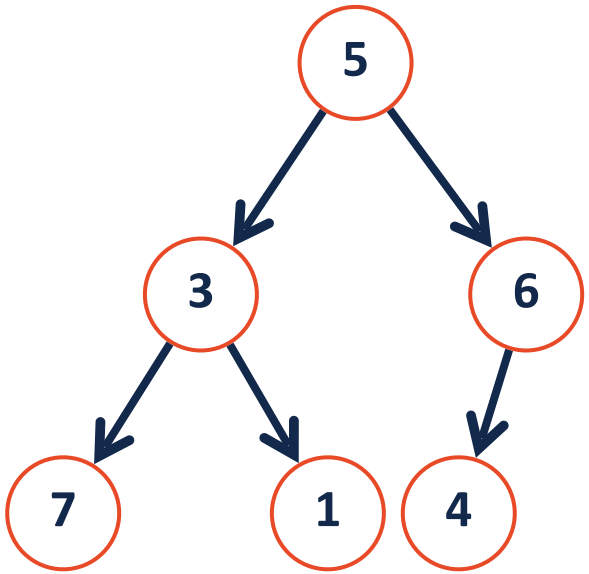
Improved search on a binary tree

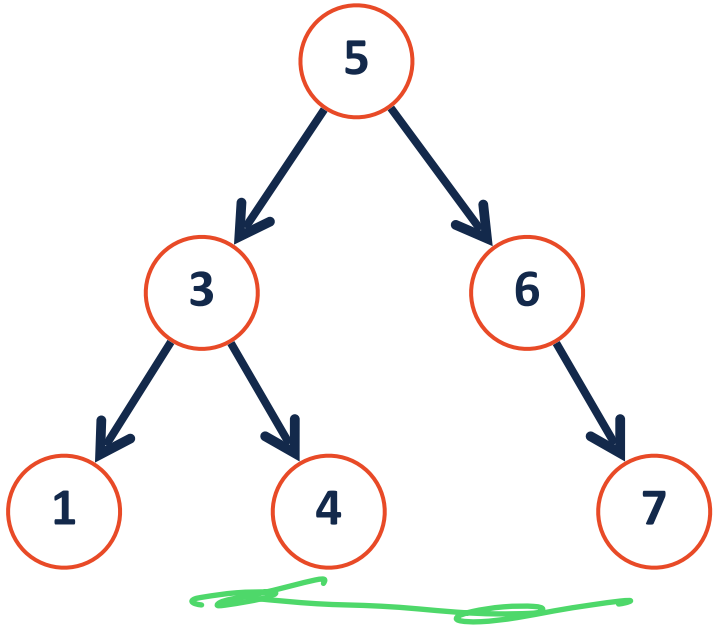
unsorted

5	3	6	7	1	4

sorted

1	3	4	5	6	7





Dictionary ADT

Data is often organized into key/value pairs:

Word → Definition

Course Number → Lecture/Lab Schedule

Node → Incident Edges

Flight Number → Arrival Information

URL → HTML Page

…

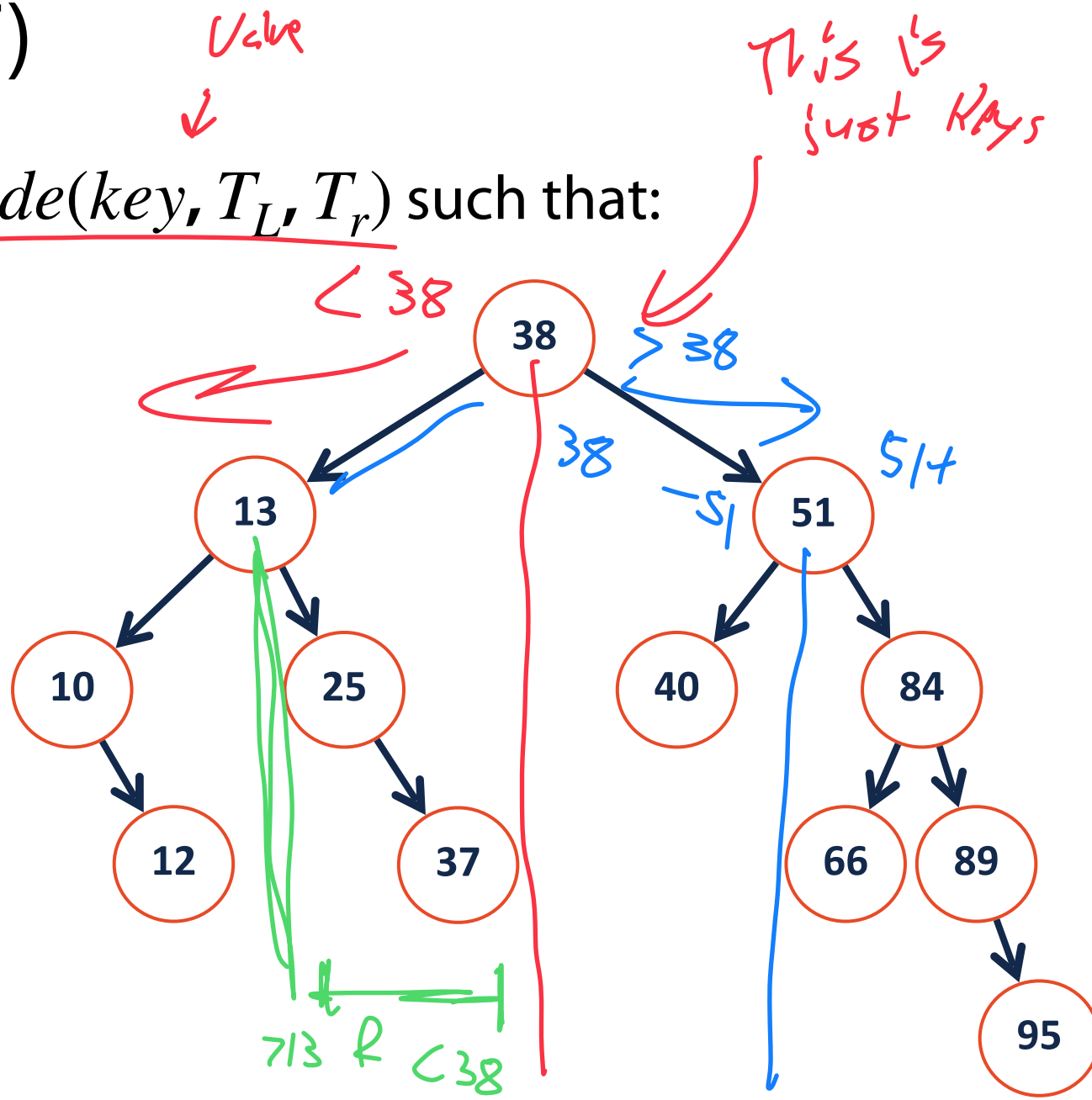
Binary Search Tree (BST)

A **BST** is a binary tree $T = TreeNode(key, T_L, T_r)$ such that:

All nodes to the left of root are:
smaller than the root

right of root are
larger than the root

Assume no duplicates!



BST.h

```
1  #pragma once
2
3  template <typename K, typename V>
4  class BST {
5    public:
6     /* ... */
7    private:
8      class TreeNode {
9         K & key;
10        V & value;
11
12        TreeNode *left, *right;
13
14        TreeNode(K & k, V & v) :
15 key(k), value(v),
16 left(NULL), right(NULL) { }
17     };
18
19     TreeNode *root_;
20    /* ... */
21 };
22
23
```

Tree.h

```
1  #pragma once
2
3  template <typename T>
4  class BinaryTree {
5    public:
6      /* ... */
7    private:
8      class TreeNode {
9        T & data;
10
11       TreeNode * left;
12
13       TreeNode * right;
14
15       TreeNode(T & data) :
16        data(data), left(NULL),
17 right(NULL) { }
18
19     };
20
21     TreeNode *root_;
22     /* ... */
23 };
```

Binary Search Tree ADT

Insert

Modifying our ADT

Remove

Find

- We traversed now we can do both!

Traverse

BST Find

To insert or remove I need find

1) Start at root node

1.5) check if root is null → if so return root;

2) Check value of root->key

↳ if root->key == query

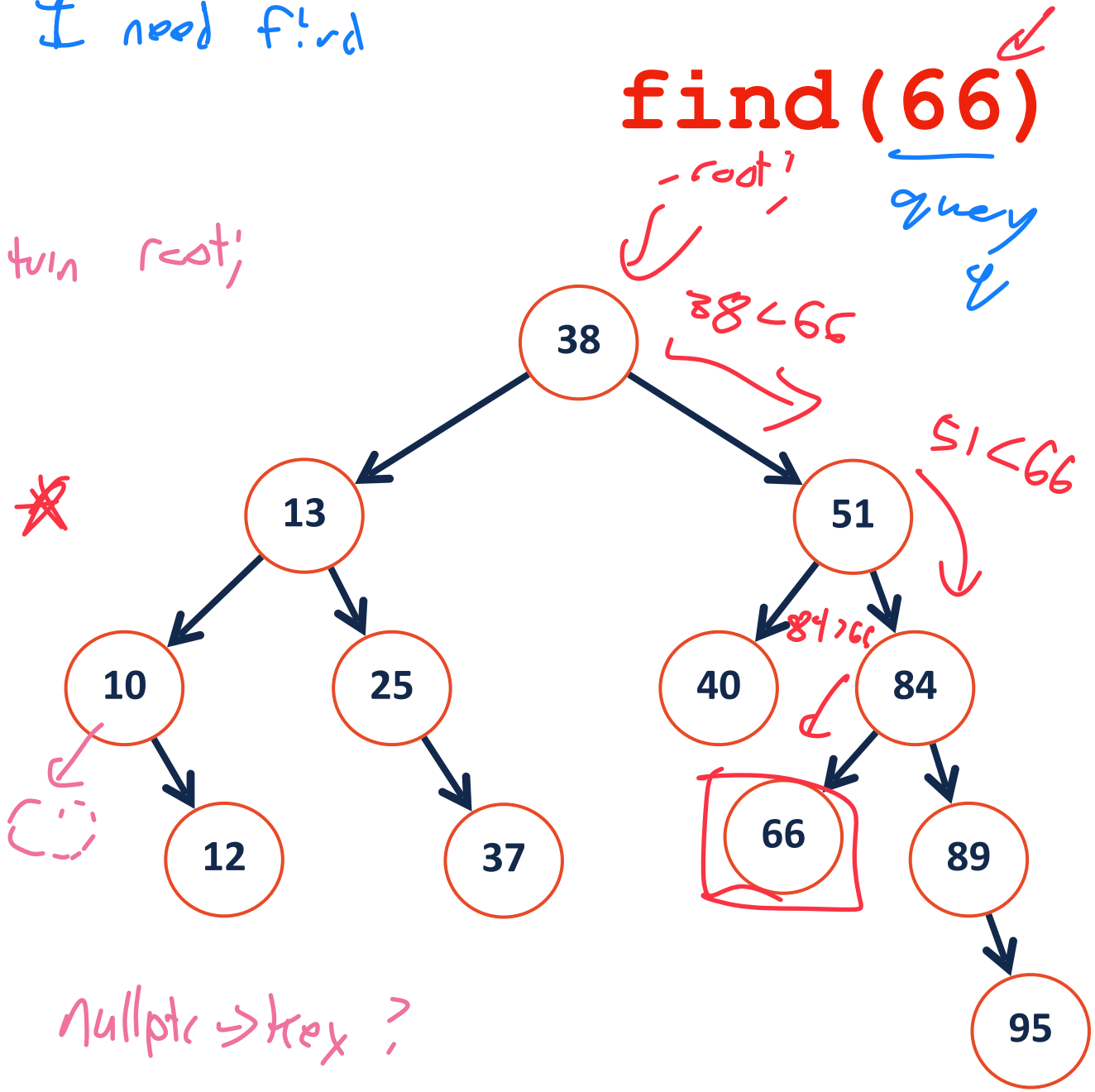
↳ if yes, return root

if root->key < query

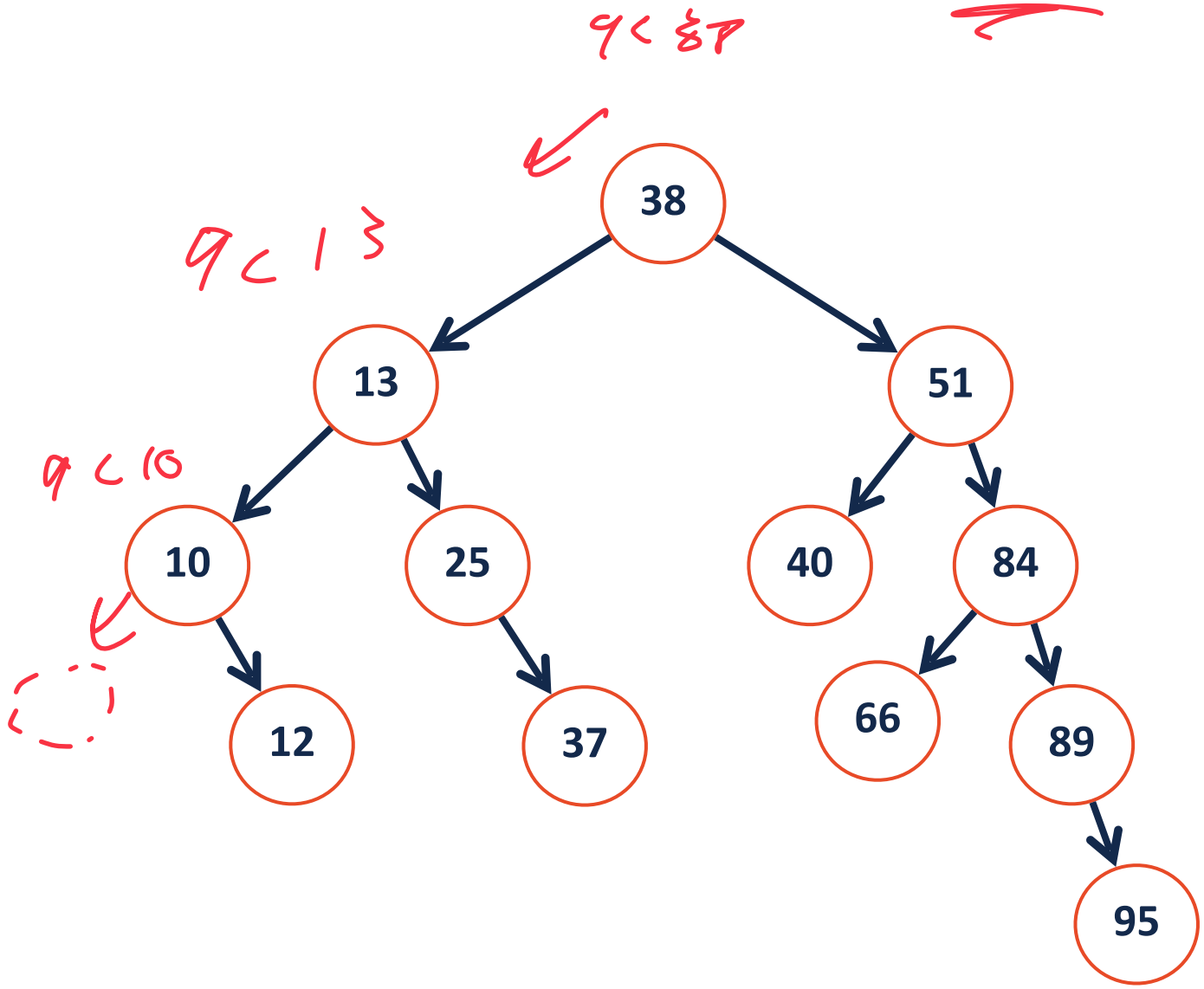
recurse right

if root->key > query

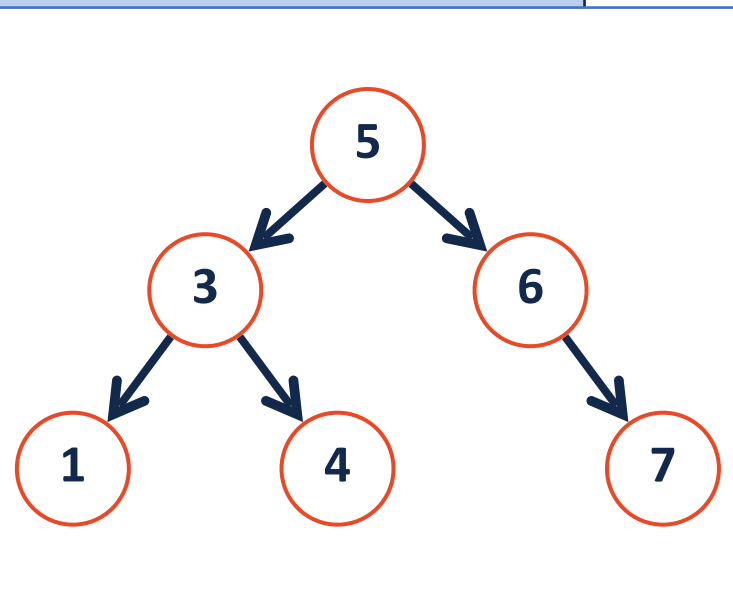
recurse left



BST Find

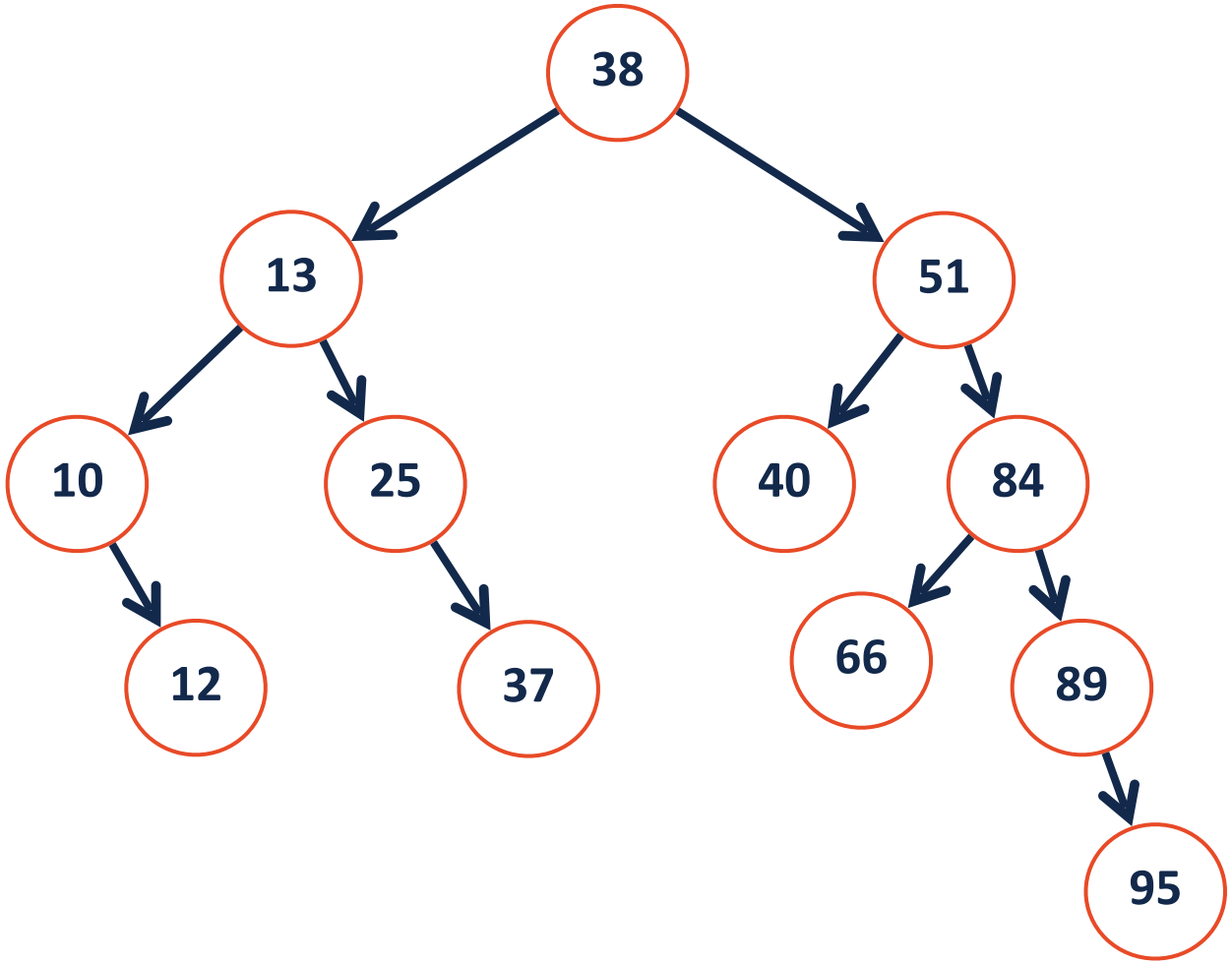


```
1  template<typename K, typename V>
2
3  _____ _find(TreeNode *& root, const K & key) {
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 }
```



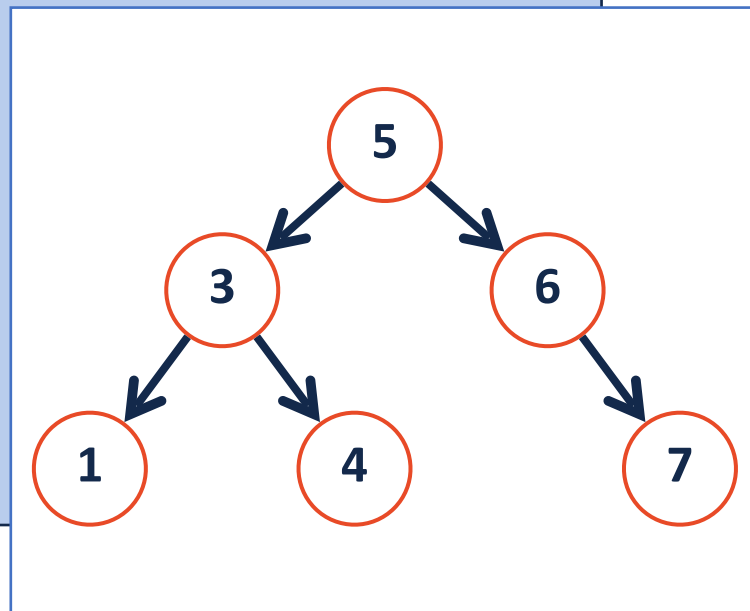
BST Insert

`insert(33, v)`



```
1  template<typename K, typename V>
2
3  void _insert(const K & key, const V & val) {
4
5      return _insert(root, key, val);
6  }
7
```

```
1  template<typename K, typename V>
2
3  void _insert(TreeNode *& root, const K & key, const V & val) {
4
5
6
7
8
9
10
11
12
13
14
15
16 }
```



BST Insert

What binary tree would be formed by inserting the following sequence of integers: `[3, 7, 2, 1, 4, 8, 0]`

BST Remove

What should our tree look like after the following…

```
remove(40)
remove(13)
remove(51)
```