Top Google Questions – Part 1

- Top Google Questions
  - 1. Two Sum
  - 3. Longest Substring Without Repeating Characters
  - 4. Median of Two Sorted Arrays
  - 5. Longest Palindromic Substring
  - 6. ZigZag Conversion
  - 8. String to Integer (atoi)
  - 11. Container With Most Water
  - 14. Longest Common Prefix
  - 19. Remove Nth Node From End of List
  - 23. Merge k Sorted Lists
  - 24. Swap Nodes in Pairs
  - 26. Remove Duplicates from Sorted Array
  - 27. Remove Element
  - 33. Search in Rotated Sorted Array
  - 34. Find First and Last Position of Element in Sorted Array
  - 35. Search Insert Position
  - 38. Count and Say
  - 41. First Missing Positive
  - 46. Permutations
  - 50. Pow(x, n)
  - 51. N-Queens
  - 53. Maximum Subarray
  - 54. Spiral Matrix
  - 55. Jump Game
  - 59. Spiral Matrix II
  - 60. Permutation Sequence
  - 66. Plus One
  - 67. Add Binary
  - 69. Sqrt(x)
  - 70. Climbing Stairs
  - 72. Edit Distance
  - 74. Search a 2D Matrix
  - 75. Sort Colors
  - 77. Combinations

- 78. Subsets
- 79. Word Search
- 80. Remove Duplicates from Sorted Array II
- 81. Search in Rotated Sorted Array II
- 83. Remove Duplicates from Sorted List
- 84. Largest Rectangle in Histogram
- 88. Merge Sorted Array
- 94. Binary Tree Inorder Traversal
- 96. Unique Binary Search Trees
- 101. Symmetric Tree
- 102. Binary Tree Level Order Traversal
- 103. Binary Tree Zigzag Level Order Traversal
- 105. Construct Binary Tree from Preorder and Inorder Traversal
- 109. Convert Sorted List to Binary Search Tree
- 110. Balanced Binary Tree
- 114. Flatten Binary Tree to Linked List
- 118. Pascal's Triangle
- 119. Pascal's Triangle II
- 121. Best Time to Buy and Sell Stock
- 129. Sum Root to Leaf Numbers
- 130. Surrounded Regions
- 136. Single Number
- 137. Single Number II
- 144. Binary Tree Preorder Traversal
- 145. Binary Tree Postorder Traversal
- 146. LRU Cache
- 153. Find Minimum in Rotated Sorted Array
- 162. Find Peak Element
- 163. Missing Ranges
- 169. Majority Element
- 173. Binary Search Tree Iterator
- 174. Dungeon Game
- 198. House Robber
- 200. Number of Islands
- 205. Isomorphic Strings
- 207. Course Schedule
- 208. Implement Trie (Prefix Tree)
- 219. Contains Duplicate II
- 221. Maximal Square
- 222. Count Complete Tree Nodes
- 226. Invert Binary Tree
- 230. Kth Smallest Element in a BST
- 231. Power of Two
- 237. Delete Node in a Linked List
- 242. Valid Anagram
- 246. Strobogrammatic Number

1. Two Sum

Description

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

Example:

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].

Solution

01/02/2020:

```
class Solution {
public:
  vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> m;
    for (int i = 0; i < nums.size(); ++i) {
      if (m.find(target - nums[i]) == m.end()) {
        m[nums[i]] = i;
      } else {
        return {m[target - nums[i]], i};
      }
    }
    return {-1, -1};
  }
};
```

3. Longest Substring Without Repeating Characters

Description

Given a string, find the length of the longest substring without repeating characters.

Example 1:

Input: "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
Example 2:

Input: "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
Example 3:

Input: "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Note that the answer must be a substring, "pwke" is a subsequence and not a substring.

Solution

05/21/2020:

```
class Solution {
public:
  int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> seen;
    int ret = 0, slow = 0, n = s.size();
    for (int fast = 0; fast < n; ++fast) {
      if (seen.count(s[fast]) != 0) slow = max(slow, seen[s[fast]] + 1);
      seen[s[fast]] = fast;
      ret = max(ret, fast - slow + 1);
    }
    return ret;
  }
};
```

4. Median of Two Sorted Arrays

Description

There are two sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

You may assume nums1 and nums2 cannot be both empty.

Example 1:

nums1 = [1, 3]
nums2 = [2]

The median is 2.0
Example 2:

nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5

Solution

01/02/2020:

```
class Solution {
```

```
public:
  double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    int n = nums1.size() + nums2.size();
    vector<int> nums(n, 0);
    auto it1 = nums1.begin();
    auto it2 = nums2.begin();
    auto it = nums.begin();
    for (; it != nums.end(); ++it) {
      *it = it2 == nums2.end() || (it1 != nums1.end() && *it1 < *it2) ? *it1++ :
*it2++;
    }
    if (n % 2 == 0) {
      return ((double) nums[n/2] + nums[n/2-1]) / 2;
    } else {
      return (double) nums[n/2];
    }
  }
};
```

5. Longest Palindromic Substring

Description

Given a string s, find the longest palindromic substring in s. You may assume that the maximum length of s is 1000.

Example 1:

Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.
Example 2:

Input: "cbbd"
Output: "bb"

Solution

01/13/2020 (Dynamic Programming):

```
class Solution {
public:
  string longestPalindrome(string s) {
    int n = s.size(), start, max_len = 0;
    if (n == 0) return "";
```

```
    vector<vector<bool>> dp(n, vector<bool>(n, false));
    for (int i = 0; i < n; ++i) dp[i][i] = true;
    for (int i = 0; i < n - 1; ++i) dp[i][i + 1] = s[i] == s[i + 1];
    for (int i = n - 3; i >= 0; --i) {
      for (int j = i + 2; j < n; ++j) {
        dp[i][j] = dp[i + 1][j - 1] && s[i] == s[j];
      }
    }
    for (int i = 0; i < n; ++i) {
      for (int j = i; j < n; ++j) {
        if (dp[i][j] && j - i + 1 > max_len) {
          max_len = j - i + 1;
          start = i;
        }
      }
    }
    return s.substr(start, max_len);
  }
};
```

01/13/2020 (Expand Around Center):

```
class Solution {
public:
  string longestPalindrome(string s) {
    int n = s.size(), start = 0, max_len = n > 0 ? 1 : 0;
    for(int i = 0; i < n; ++i) {
      for (int l = i - 1, r = i; l >= 0 && r < n && s[l] == s[r]; --l, ++r) {
        if (r - l + 1 > max_len) {
          max_len = r - l + 1;
          start = l;
        }
      }
      for (int l = i - 1, r = i + 1; l >= 0 && r < n && s[l] == s[r]; --l, ++r)
{
        if (r - l + 1 > max_len) {
          max_len = r - l + 1;
          start = l;
        }
      }
    }
    return max_len == 0 ? "" : s.substr(start, max_len);
  }
};
```

6. ZigZag Conversion

Description

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N A
P L S I I GY I
R
```

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string s, int numRows);
```

Example 1:

```
Input: s = "PAYPALISHIRING", numRows = 3
Output: "PAHNAPLSIIGYIR"
```

Example 2:

```
Input: s = "PAYPALISHIRING", numRows = 4
Output: "PINALSIGYAHRPI"
```

Explanation:

```
P     I    N
A   L S  I G
Y A   H R
P     I
```

Solution

05/24/2020:

```
class Solution {
public:
  string convert(string s, int numRows) {
    if (s.empty()) return "";
    if (numRows <= 1) return s;
    int k = 0, increment = 1;
    vector<string> strs(numRows);
    for (int i = 0; i < (int)s.size(); ++i) {
      strs[k].push_back(s[i]);
      if (k == numRows - 1) {
        increment = -1;
```

```
        } else if (k == 0) {
          increment = 1;
        }
        k += increment;
      }
      string ret;
      for (auto& s : strs) ret += s;
      return ret;
    }
};
```

8. String to Integer (atoi)

Description

Implement atoi which converts a string to an integer.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned.

Note:

Only the space character ' ' is considered as whitespace character.
Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−231, 231 − 1]. If the numerical value is out of the range of representable values, INT_MAX (231 − 1) or INT_MIN (−231) is returned.
Example 1:

Input: "42"
Output: 42
Example 2:

Input: "   -42"
Output: -42

Explanation: The first non-whitespace character is '-', which is the minus sign.
Then take as many numerical digits as possible, which gets 42.
Example 3:

Input: "4193 with words"
Output: 4193
Explanation: Conversion stops at digit '3' as the next character is not a numerical digit.
Example 4:

Input: "words and 987"
Output: 0
Explanation: The first non-whitespace character is 'w', which is not a numerical
digit or a +/- sign. Therefore no valid conversion could be performed.
Example 5:

Input: "-91283472332"
Output: -2147483648
Explanation: The number "-91283472332" is out of the range of a 32-bit signed integer.
Thefore INT_MIN (-231) is returned.

Solution

05/24/2020:

```
class Solution {
public:
  int myAtoi(string str) {
    long long  ret = atol(str.c_str());
    ret = ret > INT_MAX ? INT_MAX : ret;
    ret = ret < INT_MIN ? INT_MIN : ret;
    return ret;
  }
};
```

```
class Solution {
public:
  int myAtoi(string str) {
    if (str.empty()) return 0;
    long long ret = 0;
    istringstream iss(str);
    iss >> ret;
    ret = ret < INT_MIN ? INT_MIN : ret; ret
    = ret > INT_MAX ? INT_MAX : ret; return
    ret;
  }
};
```

11. Container With Most Water

Description

Given n non-negative integers a1, a2, ..., an , where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.

The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example:

Input: [1,8,6,2,5,4,8,3,7]
Output: 49

Solution

05/24/2020:

```cpp
class Solution {
public:
  int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1, ret = 0;
    while (left < right) {
      ret = max(ret, min(height[left], height[right]) * (right - left));
      height[left] < height[right] ? left += 1 : right -= 1;
    }
    return ret;
  }
};
```

14. Longest Common Prefix

Description

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Example 1:

Input: ["flower","flow","flight"]
Output: "fl"
Example 2:

Input: ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
Note:

All given inputs are in lowercase letters a-z.

Solution

05/24/2020:

```cpp
class Solution {
public:
  string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty() || strs[0].empty()) return "";
    int minLength = strs[0].size();
    for (auto& s : strs) minLength = min(minLength, (int)s.size());
    string ret;
```

```
    for (int i = 0; i < minLength; ++i) {
      char cur = strs[0][i];
      for (int j = 1; j < (int)strs.size(); ++j) {
        if (strs[j][i] != cur) {
          return ret;
        }
      }
      ret += cur;
    }
    return ret;
  }
};
```

```
class Solution {
public:
  string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty() || strs[0].empty()) return "";
    sort(strs.begin(), strs.end(), [](const string& s1, const string& s2) {
      if (s1.size() == s2.size()) return s1 < s2;
      return s1.size() < s2.size();
    });
    string ret;
    for (int k = (int)strs[0].size(); k >= 0; --k) {
      bool isCommonPrefix = true;
      string prefix = strs[0].substr(0, k);
      for (int i = 1; i < (int)strs.size(); ++i) {
        if (prefix != strs[i].substr(0, k)) {
          isCommonPrefix = false;
          break;
        }
      }
      if (isCommonPrefix) {
        ret = prefix;
        break;
      }
    }
    return ret;
  }
};
```

```
struct Node {
  bool isWord;
  vector<Node*> children;
  Node() { isWord = false; children.resize(26, nullptr); }
  ~Node() { for (auto& c : children) delete c; }
};
```

```
class Trie {
private:
  Node* root;

  Node* find(const string& word) {
    Node* cur = root;
    for (auto i = 0; i < (int)word.size(); ++i) {
      if (cur->children[word[i] - 'a'] == nullptr) {
        break;
      }
      cur = cur->children[word[i] - 'a'];
    }
    return cur;
  }

public:
  Trie() { root = new Node(); }

  void insert(const string& word) {
    Node* cur = root;
    for (auto i = 0; i < (int)word.size(); ++i) {
      if (cur->children[word[i] - 'a'] == nullptr) {
        cur->children[word[i] - 'a'] = new Node();
      }
      cur = cur->children[word[i] - 'a'];
    }
    cur->isWord = true;
  }

  bool contains(const string& word) {
    Node* cur = find(word);
    return cur && cur->isWord;
  }

  bool startsWith(const string& prefix) {
    Node* cur = find(prefix);
    return cur;
  }

  string commonPrefix() {
    string ret;
    Node* cur = root;
    while (true) {
      int cnt = 0;
      char ch = 'a';
      bool isUnique = false;
      for (int i = 0; i < 26; ++i) {
        if (cur->children[i] != nullptr) {
          isUnique = true;
```

```
            if (++cnt > 1) {
              isUnique = false;
              break;
            }
            ch = i + 'a';
          }
        }
        if (isUnique) {
          ret += ch;
          cur = cur->children[ch - 'a'];
        } else {
          break;
        }
      }
      return ret;
    }
};

class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
      Trie t;
      int minLength = INT_MAX;
      for (auto& s : strs) {
        if (s.empty()) return "";
        t.insert(s);
        minLength = min(minLength, (int)s.size());
      }
      return t.commonPrefix().substr(0, minLength);
    }
};
```

19. Remove Nth Node From End of List

Description

Given a linked list, remove the n-th node from the end of list and return its head.

Example:

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.
Note:

Given n will always be valid.

Follow up:

Could you do this in one pass?

Solution

05/23/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* removeNthFromEnd(ListNode* head, int n) {
    ListNode *pre = new ListNode(0, head), *slow = pre, *fast = pre;
    while (fast->next != nullptr && n-- > 0) fast = fast->next;
    while (fast->next != nullptr) {
      fast = fast->next;
      slow = slow->next;
    }
    slow->next = slow->next->next;
    return pre->next;
  }
};
```

23. Merge k Sorted Lists

Description

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

Example:

Input:
[
  1->4->5,
  1->3->4,
  2->6
]
Output: 1->1->2->3->4->4->5->6

Solution

01/02/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
  ListNode* mergeKLists(vector<ListNode*>& lists) {
    vector<int> v;
    for (auto& l : lists) {
      vector<int> tmp = ListNode2vector(l);
      v.insert(v.begin(), tmp.begin(), tmp.end());
    }
    sort(v.begin(), v.end());
    return vector2ListNode(v);
  }

  vector<int> ListNode2vector(ListNode* list) {
    vector<int> v;
    for (; list != nullptr; list = list->next)
      v.push_back(list->val);
    return v;
  }

  ListNode* vector2ListNode(vector<int>& v) {
    ListNode *pre = new ListNode(0), *cur = pre;
    for (auto& n : v) {
      cur->next = new ListNode(n);
```

```
      cur = cur->next;
    }
    return pre->next;
  }
};
```

Mimicking merge_sort 01/02/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
  ListNode* mergeKLists(vector<ListNode*>& lists) {
    int n = lists.size();
    if (n == 0) return nullptr;
    if (n == 1) return lists[0];
    vector<ListNode*> lists1(lists.begin(), lists.begin() + n / 2);
    vector<ListNode*> lists2(lists.begin() + n / 2, lists.end());
    ListNode* l1 = mergeKLists(lists1);
    ListNode* l2 = mergeKLists(lists2);
    if (l1 == nullptr) return l2;
    ListNode* ret = l1;
    while (l2 != nullptr) {
      if (l1->val > l2->val) swap(l1->val, l2->val);
      while(l1->next && l1->next->val < l2->val) l1 = l1->next;
      ListNode* tmp2 = l2->next;
      l2->next = l1->next;
      l1->next = l2;
      l2 = tmp2;
    }
    return ret;
  }
};
```

24. Swap Nodes in Pairs

Description

Given a linked list, swap every two adjacent nodes and return its head.

You may not modify the values in the list's nodes, only nodes itself may be changed.

Example:

Given 1->2->3->4, you should return the list as 2->1->4->3.

Solution

05/22/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* swapPairs(ListNode* head) {
    // pre -> A -> B -> C -> D -> null
    // pre -> B (pre->next = pre->next->next)
    // A -> C (A->next = A->next->next)
    // B -> A (B->next = A):
    // (B -> A -> C -> D -> null)
    ListNode* pre = new ListNode(0, head);
    ListNode* cur = pre;
    while (cur->next && cur->next->next) {
      ListNode* tmp = cur->next;
      cur->next = cur->next->next;
      tmp->next = tmp->next->next;
      cur->next->next = tmp;
      cur = tmp;
    }
    return pre->next;
  }
};
```

26. Remove Duplicates from Sorted Array

Description

Given a sorted array nums, remove the duplicates in-place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

Example 1:

Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively.

It doesn't matter what you leave beyond the returned length.
Example 2:

Given nums = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of nums being modified to 0, 1, 2, 3, and 4 respectively.

It doesn't matter what values are set beyond the returned length.
Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by reference, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

Solution

05/27/2020:

```
class Solution {
public:
  int removeDuplicates(vector<int>& nums) {
    if (nums.size() <= 1) return nums.size();
    int slow = 0, fast = 1, n = nums.size();
    for (; fast < n; ++fast) {
      while (nums[fast] == nums[fast - 1]) if (++fast >= n) break;
      if (fast < n) nums[++slow] = nums[fast];
    }
    return slow + 1;
  }
};
```

27. Remove Element

Description

Given an array nums and a value val, remove all instances of that value in-place and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Example 1:

Given nums = [3,2,2,3], val = 3,

Your function should return length = 2, with the first two elements of nums being 2.

It doesn't matter what you leave beyond the returned length.
Example 2:

Given nums = [0,1,2,2,3,0,4,2], val = 2,

Your function should return length = 5, with the first five elements of nums containing 0, 1, 3, 0, and 4.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned length.
Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by reference, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)
int len = removeElement(nums, val);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

Solution

05/23/2020:

```
class Solution {
public:
  int removeElement(vector<int>& nums, int val) {
    int right = nums.size() - 1, n = nums.size(), cnt = n;
    for (int i = 0; i < n && i <= right; ++i) {
      while (right >= 0 && nums[right] == val) {
        --right;
        --cnt;
      }
      if (nums[i] == val && i <= right) {
        swap(nums[i], nums[right--]);
        --cnt;
      }
    }
    return cnt;
  }
};
```

33. Search in Rotated Sorted Array

Description

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of O(log n).

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1

Solution

04/28/2020:

```
class Solution {
public:
  int search(vector<int>& nums, int target) {
    int lo = 0, hi = nums.size() - 1;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] == target) {
        return mid;
      } else {
        if (nums[mid] < nums[lo]) {
          if (target > nums[lo] && target > nums[hi]) {
            hi = mid - 1;
          } else {
            lo = mid + 1;
          }
        } else {
          if (target > nums[lo] && target > nums[hi]) {
            hi = mid - 1;
          } else {
            lo = mid + 1;
          }
        }
      }
    }
    return -1;
```

```
    }
};
```

34. Find First and Last Position of Element in Sorted Array

Description

Given an array of integers nums sorted in ascending order, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
Example 2:

Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]

Solution

04/28/2020:

```
class Solution {
public:
  vector<int> searchRange(vector<int>& nums, int target) {
    if (nums.empty()) return {-1, -1};
    int n = nums.size();
    int lower = -1, upper = -1;

    // to find lower_bound
    int lo = 0, hi = n - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] >= target) {
        hi = mid;
      } else {
        lo = mid + 1;
      }
    }
    if (nums[lo] < target) ++lo;
```

```
    lower = lo;
    if (lower > hi) return {-1, -1};

    // to find the upper bound
    lo = 0, hi = n - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] > target) {
        hi = mid;
      } else {
        lo = mid + 1;
      }
    }
    if (nums[hi] > target) --hi;
    upper = hi;
    if (lower > upper) return {-1, -1};
    return {lower, upper};
  }
};
```

```
class Solution {
public:
  vector<int> searchRange(vector<int>& nums, int target) {
    if (binary_search(nums.begin(), nums.end(), target)) {
      auto p = equal_range(nums.begin(), nums.end(), target);
      return {int(p.first - nums.begin()), int(p.second - nums.begin() - 1)};
    } else {
      return {-1, -1};
    }
  }
};
```

35. Search Insert Position

Description

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Example 1:

Input: [1,3,5,6], 5
Output: 2

```
Example 2:

Input: [1,3,5,6], 2
Output: 1
Example 3:

Input: [1,3,5,6], 7
Output: 4
Example 4:

Input: [1,3,5,6], 0
Output: 0
```

Solution

04/23/2020:

```
class Solution {
public:
  int searchInsert(vector<int>& nums, int target) {
    int lo = 0, hi = nums.size() - 1;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] == target) {
        return mid;
      } else if (nums[mid] > target) {
        hi = mid - 1;
      } else {
        lo = mid + 1;
      }
    }
    return lo;
  }
};
```

38. Count and Say

Description

```
The count-and-say sequence is the sequence of integers with the first five terms as following:

1.     1
2.     11
3.     21
```

4. 1211
5. 111221
1 is read off as "one 1" or 11.
11 is read off as "two 1s" or 21.
21 is read off as "one 2, then one 1" or 1211.

Given an integer n where 1 ≤ n ≤ 30, generate the nth term of the count-and-say sequence. You can do so recursively, in other words from the previous member read off the digits, counting the number of digits in groups of the same digit.

Note: Each term of the sequence of integers will be represented as a string.

Example 1:

Input: 1
Output: "1"
Explanation: This is the base case.
Example 2:

Input: 4
Output: "1211"
Explanation: For n = 3 the term was "21" in which we have two groups "2" and "1", "2" can be read as "12" which means frequency = 1 and value = 2, the same way "1" is read as "11", so the answer is the concatenation of "12" and "11" which is "1211".

Solution

05/24/2020:

```
class Solution {
public:
  string countAndSay(int n) {
    if (n == 1) return "1";
    string s = countAndSay(n - 1);
    string ret;
    int cnt = 1;
    char cur = s[0];
    for (int i = 1; i < (int)s.size(); ++i) {
      if (s[i] == cur) {
        ++cnt;
      } else {
        ret += to_string(cnt) + cur;
        cur = s[i];
        cnt = 1;
      }
```

```
    }
    if (cnt > 0) {
      ret += to_string(cnt) + cur;
    }
    return ret;
  }
};
```

41. First Missing Positive

Description

Given an unsorted integer array, find the smallest missing positive integer.

Example 1:

Input: [1,2,0]
Output: 3
Example 2:

Input: [3,4,-1,1]
Output: 2
Example 3:

Input: [7,8,9,11,12]
Output: 1
Note:

Your algorithm should run in O(n) time and uses constant extra space.

Solution

05/24/2020:

```
class Solution {
public:
  int firstMissingPositive(vector<int>& nums) {
    unordered_set<int> seen;
    for (auto& i : nums) {
      if (i > 0) {
        seen.insert(i);
      }
    }
    int i = 1, n = nums.size();
    while (i <= n) {
```

```
      if (seen.count(i) == 0) return i;
      ++i;
    }
    return i;
  }
};
```

46. Permutations

Description

Given a collection of distinct integers, return all possible permutations.

Example:

Input: [1,2,3]
Output:
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]

Solution

05/26/2020 (Using next_permutation):

```
class Solution {
public:
  vector<vector<int>> permute(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    vector<vector<int>> ret{nums};
    while (next_permutation(nums.begin(), nums.end())) ret.push_back(nums);
    return ret;
  }
};
```

05/26/2020 (Backtracking):

```
class Solution {
public:
  vector<vector<int>> permute(vector<int>& nums) {
```

```
    int n = nums.size(),numOfChosen = 0;
    vector<bool> chosen(n, false);
    vector<vector<int>> ret;
    vector<int> permutation;
    backtrack(nums, chosen, numOfChosen, permutation, ret);
    return ret;
  }

  void backtrack(vector<int>& nums, vector<bool>& chosen, int numOfChosen,
vector<int>& permutation, vector<vector<int>>& ret) {
    if (numOfChosen == (int)nums.size()) {
      ret.push_back(permutation);
      return;
    }
    for (int i = 0; i < (int)nums.size(); ++i) {
      if (chosen[i] == true) continue;
      chosen[i] = true;
      permutation.push_back(nums[i]);
      backtrack(nums, chosen, numOfChosen + 1, permutation, ret);
      chosen[i] = false;
      permutation.pop_back();
    }
  }
};
```

05/05/2020:

```
class Solution {
public:
  vector<vector<int>> permute(vector<int>& nums) {
    int n = nums.size();
    if (n <= 1) return {nums};
    vector<vector<int>> ret;
    for (int i = 0; i < n; ++i) {
      int cur = nums[i];
      swap(nums[i], nums[n - 1]);
      nums.pop_back();
      vector<vector<int>> sub = permute(nums);
      for (auto& s : sub) {
        s.push_back(cur);
        ret.push_back(s);
      }
      nums.push_back(cur);
      swap(nums[i], nums[n - 1]);
    }
    return ret;
  }
};
```

50. Pow(x, n)

Description

Implement pow(x, n), which calculates x raised to the power n (xn).

Example 1:

Input: 2.00000, 10
Output: 1024.00000
Example 2:

Input: 2.10000, 3
Output: 9.26100
Example 3:

Input: 2.00000, -2
Output: 0.25000
Explanation: 2-2 = 1/22 = 1/4 = 0.25
Note:

-100.0 < x < 100.0
n is a 32-bit signed integer, within the range [-231, 231 - 1]

Solution

04/23/2020:

- Discussion
- Binary Exponentiation: given a positive power n, e.g., n = 22 (binary: 10110) = 16 (binary: 10000) + 4 (binary: 100)+ 2 (binary: 10), then pow(x, n) = x^n = x^22 = x^(16 + 4 + 2) = x^16 * x^4 * x^2. For negative powers, we just flip the sign first, and once we calculate the value and return its reciprocal.
- Time complexity: O(n);
- Space complexity: O(1).

```
class Solution {
public:
  double myPow(double x, int n) {
    long double ret = 1.0, pow_x = x;
    for (long m = n >= 0 ? (long)n : -1 * (long)n; m != 0; m >>= 1) {
      if (m & 1) ret *= pow_x;
      pow_x = pow_x * pow_x;
    }
    return n >= 0 ? ret : 1.0L / ret;
  }
};
```

51. N-Queens

Description

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.

Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

Example:

```
Input: 4
Output: [
 [".Q..",    // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",    // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above.

Solution

05/27/2020:

```
class Solution {
public:
  unordered_set<string> seen;
  vector<vector<string>> solveNQueens(int n) {
    vector<vector<string>> ret;
    vector<string> board(n, string(n, '.'));
    vector<bool> col(n, false);
    vector<bool> diag1(2 * n - 1, false);
    vector<bool> diag2(2 * n - 1, false);
    backtrack(0, n, board, col, diag1, diag2, ret);
    return ret;
  }

  void backtrack(int r, int n, vector<string>& board, vector<bool>& col,
vector<bool>& diag1, vector<bool>& diag2, vector<vector<string>>& ret) {
    if (r == n) {
      ret.push_back(board);
      return;
    }
    for (int c = 0; c < n; ++c) {
      if (!col[c] && !diag1[r + c] && !diag2[r - c + n - 1]) {
        board[r][c] = 'Q';
        col[c] = diag1[r + c] = diag2[r -c + n - 1] = true;
        backtrack(r + 1, n, board, col, diag1, diag2, ret);
        col[c] = diag1[r + c] = diag2[r -c + n - 1] = false;
        board[r][c] = '.';
      }
    }
  }
};
```

53. Maximum Subarray

Description

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Example:

Input: [-2,1,-3,4,-1,2,1,-5,4],
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
Follow up:

If you have figured out the O(n) solution, try coding another solution using the divide and conquer approach, which is more subtle.

Solution

01/13/2020 (Dynamic Programming):

```
class Solution {
public:
  int maxSubArray(vector<int>& nums) {
    int n = nums.size(), max_sum = nums[0];
    for (int i = 1; i < n; ++i) {
      if (nums[i - 1] > 0) nums[i] += nums[i - 1];
      max_sum = max(max_sum, nums[i]);
    }
    return max_sum;
  }
};
```

54. Spiral Matrix

Description

54. Spiral Matrix
Medium

1984

526

Add to List

Share
Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.

Example 1:

Input:
[
 [1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]
]
Output: [1,2,3,6,9,8,7,4,5]
Example 2:

Input:
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]

Solution

04/23/2020:

```
class Solution {
public:
  vector<int> spiralOrder(vector<vector<int>>& matrix) {
    int m = matrix.size();
    if (m == 0) return {};
    int n = matrix[0].size();
    if (n == 0) return {};
    vector<vector<int>> dir{ {0, 1}, {1, 0}, {0, -1}, {-1, 0} };
    vector<int> ret;
    int N = m * n, d = 0, i = 0, j = -1;
    while (ret.size() < N) {
      int ni = i + dir[d][0], nj = j + dir[d][1];
      while (ni < 0 || ni >= m || nj < 0 || nj >= n || matrix[ni][nj] ==
INT_MIN) {
        d = (d + 1) % 4;
        ni = i + dir[d][0], nj = j + dir[d][1];
      }
      ret.push_back(matrix[ni][nj]);
      matrix[ni][nj] = INT_MIN;
      i = ni;
      j = nj;
    }
    return ret;
  }
```

```
};
```

55. Jump Game

Description

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

Example 1:

Input: [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
Example 2:

Input: [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Solution

04/24/2020:

```
class Solution {
public:
  bool canJump(vector<int>& nums) {
    int ret = 0, n = nums.size();
    for (int i = 0; i < n; ++i) {
      if (i <= ret) {
        ret = max(i + nums[i], ret);
      } else {
        break;
      }
    }
    return ret >= n - 1;
  }
};
```

59. Spiral Matrix II

Description

Given a positive integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

Example:

Input: 3
Output:
```
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

Solution

Discussion:

```
class Solution {
public:
  vector<vector<int>> generateMatrix(int n) {
    if (n <= 0) return {};
    vector<vector<int>> dir{ {0, 1}, {1, 0}, {0, -1}, {-1, 0} };
    vector<vector<int>> matrix(n, vector<int>(n, 0));
    int i = 0, j = -1, d = 0, N = n * n, cnt = 0;
    while (cnt < N) {
      int ni = i + dir[d][0], nj = j + dir[d][1];
      while (ni < 0 || ni >= n || nj < 0 || nj >= n || matrix[ni][nj] != 0) {
        d = (d + 1) % 4;
        ni = i + dir[d][0];
        nj = j + dir[d][1];
      }
      i = ni;
      j = nj;
      matrix[i][j] = ++cnt;
    }
    return matrix;
  }
};
```

60. Permutation Sequence

Description

The set [1,2,3,...,n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for n = 3:

"123"
"132"
"213"
"231"
"312"
"321"
Given n and k, return the kth permutation sequence.

Note:

Given n will be between 1 and 9 inclusive.
Given k will be between 1 and n! inclusive.
Example 1:

Input: n = 3, k = 3
Output: "213"
Example 2:

Input: n = 4, k = 9
Output: "2314"

Solution

05/26/2020:

```
class Solution {
public:
  string getPermutation(int n, int k) {
    string s;
    for (int i = 0; i < n; ++i) s += to_string(i + 1);
    while (--k > 0 && next_permutation(s.begin(), s.end()));
    return s;
  }
};
```

66. Plus One

Description

Given a non-empty array of digits representing a non-negative integer, plus one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

Example 1:

Input: [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Example 2:

Input: [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.

Solution

05/23/2020:

```
class Solution {
public:
  vector<int> plusOne(vector<int>& digits) {
    if (digits.empty()) return {1};
    int carry = 1, n = digits.size();;
    for (int i = n - 1; i >= 0; --i) {
      carry += digits[i];
      digits[i] = carry % 10;
      carry /= 10;
    }
    if (carry > 0) digits.insert(digits.begin(), carry);
    return digits;
  }
};
```

67. Add Binary

Description

Given two binary strings, return their sum (also a binary string).

The input strings are both non-empty and contains only characters 1 or 0.

Example 1:

Input: a = "11", b = "1"
Output: "100"
Example 2:

Input: a = "1010", b = "1011"
Output: "10101"

Constraints:

Each string consists only of '0' or '1' characters.
1 <= a.length, b.length <= 10^4
Each string is either "0" or doesn't contain any leading zero.

Solution

05/12/2020:

```
class Solution {
public:
  string addBinary(string a, string b) {
    int i = a.size() - 1, j = b.size() - 1;
    int carry = 0;
    string ret;
    for (; i >= 0 || j >= 0; --i, --j) {
      if (i >= 0) carry += a[i] - '0';
      if (j >= 0) carry += b[j] - '0';
      ret += (carry & 1) + '0';
      carry >>= 1;
    }
    if (carry > 0) ret += carry + '0';
    reverse(ret.begin(), ret.end());
    return ret;
  }
};
```

69. Sqrt(x)

Description

Implement int sqrt(int x).

Compute and return the square root of x, where x is guaranteed to be a non-negative integer.

Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

Example 1:

Input: 4
Output: 2
Example 2:

Input: 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since the decimal part is truncated, 2 is returned.

Solution

04/28/2020:

```
class Solution {
public:
  int mySqrt(int x) {
    int lo = 0, hi = x;
    while (lo <= hi) {
      long long mid = lo + (hi - lo) / 2;
      long long sq = mid * mid;
      if (sq == x) {
        return mid;
      } else if (sq > x) {
        hi = mid - 1;
      } else {
        lo = mid + 1;
      }
    }
    return hi;
  }
};
```

70. Climbing Stairs

Description

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Note: Given n will be a positive integer.

Example 1:

Input: 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
Example 2:

Input: 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

Solution

01/14/2020 (Dynamic Programming):

```
class Solution {
public:
  int climbStairs(int n) {
    int s1 = 1, s2 = 2;
    for (int i = 2; i < n; ++i) {
      s1 = s1 + s2;
      swap(s1, s2);
    }
    return n >= 2 ? s2 : s1;
  }
};
```

72. Edit Distance

Description

Given two words word1 and word2, find the minimum number of operations required to convert word1 to word2.

You have the following 3 operations permitted on a word:

Insert a character
Delete a character
Replace a character
Example 1:

Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
Example 2:

Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')

Solution

05/31/2020:

```
class Solution {
public:
  int minDistance(string word1, string word2) {
    int m = word1.size(), n = word2.size();
    if (m == 0) return n;
    if (n == 0) return m;
    const int INF = 1e9 + 5;
    vector<vector<int>> dp(m + 1, vector(n + 1, INF));
    dp[0][0] = 0;
    // dp[i][j]: the edit distance between word1[0..i] and word2[0..j]
    // dp[i][j] = dp[i - 1][j - 1]       if word1[i] == word2[j]: no operations
needed
    // dp[i][j] = min(                    if word1[i] != word2[j]
    //   dp[i - 1][j - 1] + 1,  replace word1[i] by word2[j]
    //   dp[i - 1][j] + 1,      delete character word1[i]
    //   dp[i][j - 1] + 1       delete character word2[j]
    // )
    for (int i = 1; i <= m; ++i) dp[i][0] = dp[i - 1][0] + 1;
    for (int j = 1; j <= n; ++j) dp[0][j] = dp[0][j - 1] + 1;
```

```
    for (int i = 1; i <= m; ++i) {
      for (int j = 1; j <= n; ++j) {
        if (word1[i - 1] == word2[j - 1]) {
          dp[i][j] = dp[i - 1][j - 1];
        } else {
          dp[i][j] = min(dp[i - 1][j], min(dp[i][j - 1], dp[i - 1][j - 1])) + 1;
        }
      }
    }
    return dp[m][n];
  }
};
```

```
class Solution {
public:
  int minDistance(string word1, string word2) {
    int m = word1.size(), n = word2.size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, INT_MAX));
    for (int i = 0; i <= m; ++i) {
      for (int j = 0; j <= n; ++j) {
        if (i == 0) {
          dp[i][j] = j;
        } else if (j == 0) {
          dp[i][j] = i;
        } else if (word1[i - 1] == word2[j - 1]) {
          dp[i][j] = dp[i - 1][j - 1];
        } else {
          dp[i][j] = min({dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]}) + 1;
        }
      }
    }
    return dp[m][n];
  }
};
```

74. Search a 2D Matrix

Description

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted from left to right.
The first integer of each row is greater than the last integer of the previous row.

```
Example 1:

Input:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 3
Output: true
Example 2:

Input:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 13
Output: false
```

Solution

05/23/2020:

```
class Solution {
public:
  bool searchMatrix(vector<vector<int>>& matrix, int target) {
    if (matrix.empty() || matrix[0].empty()) return false;
    int m = matrix.size(), n = matrix[0].size(); int
    lo = 0, hi = m * n - 1;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      int i = mid / n, j = mid % n;
      if (matrix[i][j] == target) {
        return true;
      } else if (matrix[i][j] < target) { lo
        = mid + 1;
      } else {
        hi = mid - 1;
      }
    }
    return false;
  }
};
```

75. Sort Colors

Description

Given an array with n objects colored red, white or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

Example:

Input: [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
Follow up:

A rather straight forward solution is a two-pass algorithm using counting sort.
First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.
Could you come up with a one-pass algorithm using only constant space?

Solution

05/12/2020:

```
class Solution {
public:
  void sortColors(vector<int>& nums) {
    sort(nums.begin(), nums.end());
  }
};
```

77. Combinations

Description

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

Example:

```
Input: n = 4, k = 2
Output:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

Solution

05/27/2020:

```
class Solution {
public:
  vector<vector<int>> combine(int n, int k) {
    vector<int> combination;
    vector<vector<int>> ret;
    vector<bool> chosen(n + 1, false);
    backtrack(1, n, k, chosen, combination, ret);
    return ret;
  }

  void backtrack(int j, int n, int k, vector<bool>& chosen, vector<int>&
combination, vector<vector<int>>& ret) {
    if (combination.size() == k) ret.push_back(combination);
    for (int i = j; i <= n; ++i) {
      if (chosen[i] == true || (!combination.empty() && i < combination.back()))
continue;
      chosen[i] = true;
      combination.push_back(i);
      backtrack(j + 1, n, k, chosen, combination, ret);
      chosen[i] = false;
      combination.pop_back();
    }
  }
};
```

78. Subsets

Description

Given a set of distinct integers, nums, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

Example:

Input: nums = [1,2,3]
Output:
```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

Solution

05/26/2020:

```
class Solution {
public:
  vector<int> subset;
  vector<vector<int>> s;
  int n;
  vector<vector<int>> subsets(vector<int>& nums) {
    n = nums.size();
    search(0);
    vector<vector<int>> ret;
    for (auto& ss : s) {
      vector<int> tmp;
      for (auto& i : ss) {
        tmp.push_back(nums[i]);
      }
      ret.push_back(tmp);
    }
    return ret;
  }

  void search(int k) {
    if (k == n) {
      s.push_back(subset);
    } else {
      subset.push_back(k);
```

```
      search(k + 1);
      subset.pop_back();
      search(k + 1);
    }
  }
};
```

```
class Solution {
public:
  vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> ret;
    vector<int> subset;
    backtrack(0, nums, subset, ret);
    return ret;
  }

  void backtrack(int k, vector<int>& nums, vector<int>& subset,
vector<vector<int>>& ret) {
    int n = nums.size();
    if (k == n) {
      ret.push_back(subset);
      return;
    }
    subset.push_back(nums[k]);
    backtrack(k + 1, nums, subset, ret);
    subset.pop_back();
    backtrack(k + 1, nums, subset, ret);
  }
};
```

```
class Solution {
public:
  vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> ret{ {} };
    for (auto& n : nums) {
      int sz = ret.size();
      for (int i = 0; i < sz; ++i) {
        ret.push_back(ret[i]);
        ret.back().push_back(n);
      }
    }
    return ret;
  }
};
```

```
public:
```

```
  vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> ret;
    vector<int> subset;
    backtrack(0, nums, subset, ret);
    return ret;
  }

  void backtrack(int k, vector<int>& nums, vector<int>& subset,
vector<vector<int>>& ret) {
    ret.push_back(subset);
    for (int i = k; i < (int)nums.size(); ++i) {
      subset.push_back(nums[i]);
      backtrack(i + 1, nums, subset, ret);
      subset.pop_back();
    }
  }
};
```

79. Word Search

Description

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example:

```
board =
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
```

Given word = "ABCCED", return true.
Given word = "SEE", return true.
Given word = "ABCB", return false.

Constraints:

board and word consists only of lowercase and uppercase English letters.
1 <= board.length <= 200

```
1 <= board[i].length <= 200
1 <= word.length <= 10^3
```

Solution

05/27/2020:

```
class Solution {
public:
  int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
  bool exist(vector<vector<char>>& board, string word) {
    if (word.empty()) return true;
    if (board.empty() || board[0].empty()) return false;
    int m = board.size(), n = board[0].size(), k = word.size();
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        if (backtrack(board, word, i, j, 0))
          return true;
    return false;
  }

  bool backtrack(vector<vector<char>>& board, string word, int i, int j, int k)
{
    int m = board.size(), n = board[0].size(), sz = word.size();
    if (board[i][j] != word[k]) return false;
    if (k == sz - 1) return true;
    board[i][j] = '#';
    for (int d = 0; d < 4; ++d) {
      int ni = i + dir[d][0], nj = j + dir[d][1];
      if (ni < 0 || ni >= m || nj < 0 || nj >= n) continue;
      if (backtrack(board, word, ni, nj, k + 1)) return true;
    }
    board[i][j] = word[k];
    return false;
  }
};
```

80. Remove Duplicates from Sorted Array II

Description

Given a sorted array nums, remove the duplicates in-place such that duplicates appeared at most twice and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

Example 1:

Given nums = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of nums being 1, 1, 2, 2 and 3 respectively.

It doesn't matter what you leave beyond the returned length.
Example 2:

Given nums = [0,0,1,1,1,1,2,3,3],

Your function should return length = 7, with the first seven elements of nums being modified to 0, 0, 1, 1, 2, 3 and 3 respectively.

It doesn't matter what values are set beyond the returned length.
Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by reference, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

Solution

05/27/2020:

```
class Solution {
public:
  int removeDuplicates(vector<int>& nums) {
    if (nums.size() <= 2) return nums.size();
    int slow = 0, fast = 1, n = nums.size(), cnt = 1;
    for (; fast < n; ++fast) {
      while (fast < n && nums[fast] == nums[fast - 1]) {
```

```
        if (cnt <= 1) {
          ++cnt;
          nums[++slow] = nums[fast];
        }
        ++fast;
      }
      if (fast < n && nums[fast] != nums[slow]) {
        cnt = 1;
        nums[++slow] = nums[fast];
      }
    }
    return slow + 1;
  }
};
```

81. Search in Rotated Sorted Array II

Description

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,0,1,2,2,5,6] might become [2,5,6,0,0,1,2]).

You are given a target value to search. If found in the array return true, otherwise return false.

Example 1:

Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
Example 2:

Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
Follow up:

This is a follow up problem to Search in Rotated Sorted Array, where nums may contain duplicates.
Would this affect the run-time complexity? How and why?

Solution

05/27/2020:

```
class Solution {
public:
  bool search(vector<int>& nums, int target) {
    sort(nums.begin(), nums.end());
    return binary_search(nums.begin(), nums.end(), target);
  }
};
```

83. Remove Duplicates from Sorted List

Description

Given a sorted linked list, delete all duplicates such that each element appear only once.

Example 1:

Input: 1->1->2
Output: 1->2
Example 2:

Input: 1->1->2->3->3
Output: 1->2->3

Solution

05/27/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* deleteDuplicates(ListNode* head) {
    if (head == nullptr || head->next == nullptr) return head;
    ListNode *slow = head, *fast = head->next;
    while (fast->next) {
      while (fast && fast->val == slow->val && fast->next) fast = fast->next;
```

```
      if (fast && fast->val != slow->val) {
        slow->next = fast;
        slow = slow->next;
      }
    }
    if (slow->val == fast->val) {
      slow->next = fast->next;
    }
    return head;
  }
};
```

84. Largest Rectangle in Histogram

Description

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.

Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].

The largest rectangle is shown in the shaded area, which has area = 10 unit.

Solution

01/14/2020 (Stack):

```
class Solution {
public:
  // Approach 1: Brute Force
  // int largestRectangleArea(vector<int>& heights) {
  //   int max_area = INT_MIN;
  //   for (int i = 0; i < heights.size(); ++i) {
  //     for (int j = i; j < heights.size(); ++j) {
  //       int min_height = *min_element(heights.begin() + i, heights.begin() +
j + 1);
  //       max_area = max(max_area, min_height * (j - i + 1));
  //     }
  //   }
  //   return max_area;
  // }

  // Approach 2: Better Brute Force
  // int largestRectangleArea(vector<int>& heights) {
  //   int n = heights.size(), max_area = n > 0 ? heights[0] : 0;
```

```
  //    for (int i = 0; i < heights.size(); ++i) {
  //      int l = i, r = i;
  //      for (; l >= 0 && heights[l] >= heights[i]; --l);
  //      for (; r < heights.size() && heights[r] >= heights[i]; ++r);
  //      max_area = max(max_area, heights[i] * (--r - ++l + 1));
  //    }
  //    return max_area;
  // }

  // Approach 3: Divide and Conquer
  // int maxRectangleArea(vector<int>& heights, int left, int right) {
  //    if (left > right) return 0;
  //    int min_height_index = left;
  //    for (int i = left; i <= right; ++i) {
  //      if (heights[min_height_index] > heights[i])
  //        min_height_index = i;
  //    }
  //    return max(max(maxRectangleArea(heights, left, min_height_index - 1),
maxRectangleArea(heights, min_height_index + 1, right)),
heights[min_height_index] * (right - left + 1));
  // }
  //
  // int largestRectangleArea(vector<int>& heights) {
  //    return maxRectangleArea(heights, 0, heights.size() - 1);
  // }

  // Approach 4: Stack
  int largestRectangleArea(vector<int>& heights) {
    stack<int> s;
    s.push(-1);
    int max_area = 0;
    for (int i = 0; i < heights.size(); ++i) {
      while (s.top() != -1 && heights[s.top()] >= heights[i]) {
        int k = s.top();
        s.pop();
        max_area = max(max_area, heights[k] * (i - s.top() - 1));
      }
      s.push(i);
    }
    while (s.top() != -1) {
      int k = s.top();
      s.pop();
      max_area = max(max_area, heights[k] * ((int)heights.size() - s.top() -
1));
    }
    return max_area;
  }
};
```

88. Merge Sorted Array

Description

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:

The number of elements initialized in nums1 and nums2 are m and n respectively. You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2.

Example:

Input:
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6]

Solution

05/23/2020:

```
class Solution {
public:
  void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    copy(nums2.begin(), nums2.end(), nums1.begin() + m);
    inplace_merge(nums1.begin(), nums1.begin() + m, nums1.end());
  }
};
```

94. Binary Tree Inorder Traversal

Description

Given a binary tree, return the inorder traversal of its nodes' values.

Example:

Input: [1,null,2,3]

```
   1
    \
     2
    /
   3
```

Output: [1,3,2]
Follow up: Recursive solution is trivial, could you do it iteratively?

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<int> inorderTraversal(TreeNode* root) {
    if (!root) return {};
    stack<TreeNode*> st;
    st.push(root);
    unordered_set<TreeNode*> visited;
    vector<int> ret;
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      if (cur->left && visited.count(cur->left) == 0) {
        if (cur->right) st.push(cur->right);
        st.push(cur);
        st.push(cur->left);
      } else if (!cur->left) {
        if (cur->right) st.push(cur->right);
        ret.push_back(cur->val);
        visited.insert(cur);
      } else {
        ret.push_back(cur->val);
        visited.insert(cur);
```

```
      }
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> inorderTraversal(TreeNode* root) {
    if (!root) return {};
    vector<int> ret = inorderTraversal(root->left);
    ret.push_back(root->val);
    vector<int> right = inorderTraversal(root->right);
    ret.insert(ret.end(), right.begin(), right.end());
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> inorderTraversal(TreeNode* root) {
    if (!root) return {};
    vector<int> ret;
    TreeNode* cur = root;
    stack<TreeNode*> st;
    while (cur || !st.empty()) {
      while (cur) {
        st.push(cur);
        cur = cur->left;
      }
      cur = st.top(); st.pop();
      ret.push_back(cur->val);
      cur = cur->right;
    }
    return ret;
  }
};
```

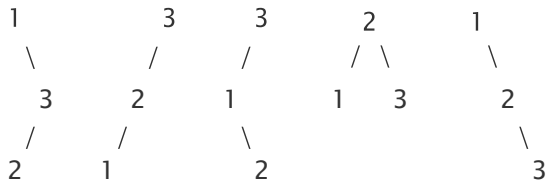
96. Unique Binary Search Trees

Description

Given n, how many structurally unique BST's (binary search trees) that store values 1 ... n?

Example:

Input: 3
Output: 5
Explanation:
Given n = 3, there are a total of 5 unique BST's:



Solution

06/24/2020:

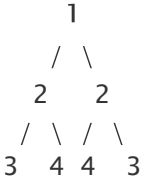
```
class Solution {
public:
  int numTrees(int n) {
    long C = 1;
    for (int i = 0; i < n; ++i) C = C * 2 * (2 * i + 1) / (i + 2);
    return C;
  }
};
```

101. Symmetric Tree

Description

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree [1,2,2,3,4,4,3] is symmetric:



But the following [1,2,2,null,3,null,3] is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

Follow up: Solve it both recursively and iteratively.

Solution

Discussion: This problem is very similar to https://leetcode.com/problems/same-tree/. Non-recursive DFS:

```
class Solution {
public:
  bool isSymmetric(TreeNode* root) {
    stack<pair<TreeNode*, TreeNode*>> st;
    st.emplace(root, root);
    while (!st.empty()) {
      pair<TreeNode*, TreeNode*> cur = st.top(); st.pop();
      if (cur.first == nullptr && cur.second == nullptr) continue;
      if (cur.first != nullptr && cur.second == nullptr) return false;
      if (cur.first == nullptr && cur.second != nullptr) return false;
      if (cur.first->val != cur.second->val) return false;
      st.emplace(cur.first->left, cur.second->right);
      st.emplace(cur.first->right, cur.second->left);
    }
    return true;
  }
};
```

Recursive DFS:

```
class Solution {
public:
  bool isSymmetric(TreeNode* root) {
    if (!root) return true;
    return isSubtreeSymmetric(root->left, root->right);
  }

  bool isSubtreeSymmetric(TreeNode* left, TreeNode* right) {
    if (!left && !right) return true;
    if ((!left && right) || (left && !right)) return false;
    return left->val == right->val && isSubtreeSymmetric(left->left, right->right) && isSubtreeSymmetric(left->right, right->left);
  }
};
```

102. Binary Tree Level Order Traversal

Description

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example:
Given binary tree [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

Solution

04/26/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
```

```
 *      int val;
 *      TreeNode *left;
 *      TreeNode *right;
 *      TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<vector<int>> levelOrder(TreeNode* root) {
    if (root == nullptr) return {};
    vector<vector<int>> ret;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
      int n = q.size();
      vector<int> level;
      for (int i = 0; i < n; ++i) {
        TreeNode* cur = q.front(); q.pop();
        level.push_back(cur->val);
        if (cur->left) q.push(cur->left);
        if (cur->right) q.push(cur->right);
      }
      ret.push_back(level);
    }
    return ret;
  }
};
```

103. Binary Tree Zigzag Level Order Traversal

Description

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example:
Given binary tree [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

Solution

Discussion 04/26/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>> ret;
    if (root == nullptr) return ret;
    queue<pair<TreeNode*, int>> q;
    q.emplace(root, 0);
    while (!q.empty()) {
      int n = q.size();
      vector<int> level;
      int depth = q.front().second;
      for (int i = 0; i < n; ++i) {
        pair<TreeNode*, int> cur = q.front(); q.pop();
        level.push_back(cur.first->val);
        if (cur.first->left) q.emplace(cur.first->left, cur.second + 1);
        if (cur.first->right) q.emplace(cur.first->right, cur.second + 1);
      }
      if (depth % 2 != 0) reverse(level.begin(), level.end());
      ret.push_back(level);
    }
    return ret;
  }
};
```

Better:

```
class Solution {
public:
```

```
  vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>> ret;
    if (root == nullptr) return ret;
    queue<TreeNode*> q;
    q.push(root);
    for (int depth = 0; !q.empty(); ++depth) {
      int n = q.size();
      vector<int> level;
      for (int j = 0; j < n; ++j) {
        TreeNode* cur = q.front(); q.pop();
        level.push_back(cur->val);
        if (cur->left) q.push(cur->left);
        if (cur->right) q.push(cur->right);
      }
      if (depth % 2 != 0) reverse(level.begin(), level.end());
      ret.push_back(level);
    }
    return ret;
  }
};
```

105. Construct Binary Tree from Preorder and Inorder Traversal

Description

Given preorder and inorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

For example, given

preorder = [3,9,20,15,7]
inorder = [9,3,15,20,7]
Return the following binary tree:

```
    3
   / \
  9  20
    /  \
   15   7
```

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder, int p_start
= 0, int p_stop = INT_MAX, int i_start = 0, int i_stop = INT_MAX) {
        if (p_stop == INT_MAX) p_stop = i_stop = inorder.size() - 1;if
        (i_start > i_stop) return nullptr;
        int i = i_start;
        for (; i <= i_stop; ++i)
          if (inorder[i] == preorder[p_start])
            break;
        TreeNode* root = new TreeNode(preorder[p_start]);
        root->left = buildTree(preorder, inorder, p_start + 1, p_start + 1 + (i -
1) - i_start, i_start, i - 1);
        root->right = buildTree(preorder, inorder, p_start + 1 + (i - 1) - i_start
+ 1, p_stop, i + 1, i_stop);
        return root;
    }
};
```

109. Convert Sorted List to Binary Search Tree

Description

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

Example:

Given the sorted linked list: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents the following height balanced BST:

```
      0
     / \
   -3   9
   /   /
 -10  5
```

Solution

05/27/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  TreeNode* sortedListToBST(ListNode* head) {
    if (head == nullptr) return nullptr;
    if (head->next == nullptr) return new TreeNode(head->val);
    ListNode *slow = head, *fast = head, *preSlow = head;
    while (fast && fast->next) {
      preSlow = slow;
      slow = slow->next;
      fast = fast->next->next;
    }
```

```
        preSlow->next = nullptr;
        TreeNode* root = new TreeNode(slow->val);
        root->left = sortedListToBST(head);
        root->right = sortedListToBST(slow->next);
        return root;
    }
};
```

110. Balanced Binary Tree

Description

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as:

a binary tree in which the left and right subtrees of every node differ in height by no more than 1.

Example 1:

Given the following tree [3,9,20,null,null,15,7]:

```
    3
   / \
  9  20
    /  \
   15   7
```

Return true.

Example 2:

Given the following tree [1,2,2,3,3,null,null,4,4]:

```
       1
      / \
     2   2
    / \
   3   3
  / \
 4   4
```

Return false.

Solution

05/27/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  bool isBalanced(TreeNode* root) {
    if (root == nullptr) return true;
    return abs(depth(root->left) - depth(root->right)) <= 1 && isBalanced(root-
>left) && isBalanced(root->right);
  }

  int depth(TreeNode* root) {
    if (root == nullptr) return 0;
    return 1 + max(depth(root->left), depth(root->right));
  }
};
```

114. Flatten Binary Tree to Linked List

Description

Given a binary tree, flatten it to a linked list in-place.

For example, given the following tree:

```
    1
   / \
  2   5
 / \   \
3   4   6
```

The flattened tree should look like:

```
1
```

```
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
```

Solution

05/27/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  void flatten(TreeNode* root) {
    if (root == nullptr) return;
    flatten(root->left);
    flatten(root->right);
    if (root->left) {
      TreeNode* cur = root->left;
      while (cur->right) cur = cur->right;
      cur->right = root->right;
      root->right = root->left;
      root->left = nullptr;
    }
  }
};
```

```
class Solution {
public:
  void flatten(TreeNode* root) {
    if (root == nullptr) return;
```

```
    TreeNode *list = new TreeNode(0), *l = list;
    stack<TreeNode*> st;
    st.push(root);
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      if (cur->right) st.push(cur->right);
      if (cur->left) st.push(cur->left);
      l->right = cur;
      cur->left = nullptr;
      l = l->right;
    }
    root = list->right;
  }
};
```

118. Pascal's Triangle

Description

Given a non-negative integer numRows, generate the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it.

Example:

Input: 5
Output:

```
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

Solution

05/27/2020:

```
class Solution {
public:
  vector<vector<int>> generate(int numRows) {
    if (numRows < 0) return {};
    vector<vector<int>> ret(numRows);
    for (int i = 0; i < numRows; ++i) {
      ret[i].resize(i + 1, 1);
      for (int j = 1; j < i; ++j)
        ret[i][j] = ret[i - 1][j] + ret[i - 1][j - 1];
    }
    return ret;
  }
};
```

119. Pascal's Triangle II

Description

Given a non-negative index k where k ≤ 33, return the kth index row of the Pascal's triangle.

Note that the row index starts from 0.

In Pascal's triangle, each number is the sum of the two numbers directly above it.

Example:

Input: 3
Output: [1,3,3,1]
Follow up:

Could you optimize your algorithm to use only O(k) extra space?

Solution

05/27/2020:

```
class Solution {
public:
  vector<int> getRow(int rowIndex) {
    if (rowIndex <= 1) return vector<int>(rowIndex + 1, 1);
    vector<int> ret(2, 1);
    for (int i = 2; i <= rowIndex; ++i) {
```

```
      vector<int> nextRow(i + 1, 1);
      for (int j = 1; j < i; ++j)
        nextRow[j] = ret[j - 1] + ret[j];
      ret = nextRow;
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> getRow(int rowIndex) {
    int n = rowIndex + 1;
    vector<vector<int>> ret(n);
    for (int i = 0; i < n; ++i) {
      ret[i].resize(i + 1, 1);
      for (int j = 1; j < i; ++j) {
        ret[i][j] = ret[i - 1][j - 1] + ret[i - 1][j];
      }
    }
    return ret.back();
  }
};
```

121. Best Time to Buy and Sell Stock

Description

Say you have an array for which the ith element is the price of a given stock on day **i**.

If you were only permitted to complete at most one transaction (**i.e.**, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one.

Example 1:

Input: [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Not 7-1 = 6, as selling price needs to be larger than buying price.
Example 2:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

Solution

01/13/2020 (Dynamic Programming):

```
class Solution {
public:
  int maxProfit(vector<int>& prices) {
    int n = prices.size();
    if (n <= 1) return 0;
    vector<int> diff(n - 1, 0);
    for (int i = 0; i < n - 1; ++i) {
      diff[i] = prices[i + 1] - prices[i];
    }
    int max_sum = max(0, diff[0]);
    for (int i = 1; i < n - 1; ++i) {
      if (diff[i - 1] > 0) diff[i] += diff[i - 1];
      max_sum = max(diff[i], max_sum);
    }
    return max_sum;
  }
};
```

129. Sum Root to Leaf Numbers

Description

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

Note: A leaf is a node with no children.

Example:

```
Input: [1,2,3]
    1
   / \
```

```
  2   3
Output:    25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.
Example 2:

Input: [4,9,0,5,1]
     4
    / \
   9   0
  / \
 5   1
Output: 1026
 Explanation:
The root-to-leaf path 4->9->5 represents the number 495. The
root-to-leaf path 4->9->1 represents the number 491. The
root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.
```

Solution

06/26/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  int sumNumbers(TreeNode* root) {
    long long ret = 0, num = 0;
    backtrack(num, ret, root);
    return ret;
  }

  void backtrack(long long& num, long long& ret, TreeNode* root) {
    if (!root) return;
    if (!root->left && !root->right) {
```

```
            ret += num * 10 + root->val;
            return;
        }
        num = num * 10 + root->val;
        backtrack(num, ret, root->left);
        backtrack(num, ret, root->right);
        num /= 10;
    }
};
```

130. Surrounded Regions

Description

Given a 2D board containing 'X' and 'O' (the letter O), capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

Example:

X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:

X X X X
X X X X
X X X X
X O X X
Explanation:

Surrounded regions shouldn't be on the border, which means that any 'O' on the border of the board are not flipped to 'X'. Any 'O' that is not on the border and it is not connected to an 'O' on the border will be flipped to 'X'. Two cells are connected if they are adjacent cells connected horizontally or vertically.

Solution

05/08/2020 Discussion:

The idea is borrowed from one of the assignments of the course Algorithms (Part 1):

1. We assume there is a virtual cell (labeled as m * n) that the node on the boundary of the board is

automatically connected to it.

2. We only need to deal with the cells that are 'O': merge these neighboring 'O' cells.
3. Traverse 'O' cells which are stored in visited: if the cell is connected to the virtual cell do nothing; otherwise, change it to 'X'.

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;

public:
  UnionFind (int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool connected(int x, int y) { return find(x) == find(y); }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[j] > sz[i]) {
      swap(i, j);
      swap(sz[i], sz[j]);
    }
    id[j]  = i;
    sz[i] += sz[j];
    return true;
  }
};

class Solution {
public:
  void solve(vector<vector<char>>& board) {
    if (board.empty() || board[0].empty()) return;
    int m = board.size(), n = board[0].size(), bound = m * n;
    UnionFind uf(m * n + 1);
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    unordered_set<int> visited;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (board[i][j] == 'X' || visited.count(i * n + j)) continue;
```

```
        if (i == 0 || i == m - 1 || j == 0 || j == n - 1) uf.merge(i * n + j,
bound);
        for (int d = 0; d < 4; ++d) {
          int ni = i + dir[d][0], nj = j + dir[d][1];
          if (ni >= 0 && ni < m && nj >= 0 && nj < n && board[ni][nj] == 'O' &&
visited.count(i * n + j) == 0) {
            uf.merge(i * n + j, ni * n + nj);
          }
        }
        visited.insert(i * n + j);
      }
    }
    for (auto& p : visited) {
      if (!uf.connected(bound, p)) {
        int i = p / n, j = p % n;
        board[i][j] = 'X';
      }
    }
  }
};
```

Update: Inspired by the above, we can actually use a set uncaptured to store all the cells that areconnected to the bounds. We can always use BFS which starts from the 'O' cells on the boundary to find the inner cells which are connected to the boundary.

```
class Solution {
public:
  void solve(vector<vector<char>>& board) {
    if (board.empty() || board[0].empty()) return;
    int m = board.size(), n = board[0].size();
    unordered_set<int> uncaptured;
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (i == 0 || i == m - 1 || j == 0 || j == n - 1) {
          if (board[i][j] == 'X') continue;
          queue<int> q;
          q.push(i * n + j);
          while (!q.empty()) {
            int sz = q.size();
            for (int s = 0; s < sz; ++s) {
              int cur = q.front(); q.pop();
              if (uncaptured.count(cur) > 0) continue;
              for (int d = 0; d < 4; ++d) {
                int ni = cur / n + dir[d][0], nj = cur % n + dir[d][1];
                if (ni >= 0 && ni < m && nj >= 0 && nj < n && board[ni][nj] ==
'O') {
                  q.push(ni * n + nj);
```

```
                }
              }
              uncaptured.insert(cur);
            }
          }
        }
      }
    }
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (board[i][j] == 'O' && uncaptured.count(i * n + j) == 0) {
          board[i][j] = 'X';
        }
      }
    }
  }
};
```

136. Single Number

Description

Given a non-empty array of integers, every element appears twice except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Example 1:

Input: [2,2,1]
Output: 1
Example 2:

Input: [4,1,2,1,2]
Output: 4

Solution

05/27/2020:

```
class Solution {
public:
  int singleNumber(vector<int>& nums) {
    int ret = 0;
    for (auto& i : nums) ret ^= i;
    return ret;
  }
};
```

137. Single Number II

Description

Given a non-empty array of integers, every element appears three times except for one, which appears exactly once. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Example 1:

Input: [2,2,3,2]
Output: 3
Example 2:

Input: [0,1,0,1,0,1,99]
Output: 99

Solution

05/27/2020:

```
class Solution {
public:
  int singleNumber(vector<int>& nums) {
    unordered_map<int, int> cnt;
    for (auto& i : nums) ++cnt[i];
    for (auto& [k, v] : cnt)
      if (v == 1)
        return k;
    return 0;
  }
};
```

144. Binary Tree Preorder Traversal

Description

Given a binary tree, return the preorder traversal of its nodes' values.

Example:

Input: [1,null,2,3]

```
   1
    \
     2
    /
   3
```

Output: [1,2,3]
Follow up: Recursive solution is trivial, could you do it iteratively?

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
```

```
  vector<int> preorderTraversal(TreeNode* root) {
    if (root == nullptr) return {};
    vector<int> ret;
    stack<TreeNode*> st;
    st.push(root);
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      ret.push_back(cur->val);
      if (cur->right) st.push(cur->right);
      if (cur->left) st.push(cur->left);
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> preorderTraversal(TreeNode* root) {
    if (root == nullptr) return {};
    vector<int> ret;
    ret.push_back(root->val);
    vector<int> l = preorderTraversal(root->left);
    vector<int> r = preorderTraversal(root->right);
    ret.insert(ret.end(), l.begin(), l.end());
    ret.insert(ret.end(), r.begin(), r.end());
    return ret;
  }
};
```

145. Binary Tree Postorder Traversal

Description

Given a binary tree, return the postorder traversal of its nodes' values.

Example:

Input: [1,null,2,3]

```
   1
    \
     2
    /
   3
```

Output: [3,2,1]
Follow up: Recursive solution is trivial, could you do it iteratively?

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<int> postorderTraversal(TreeNode* root) {
    vector<int> ret;
    if (root == nullptr) return ret;
    stack<TreeNode*> st;
    st.push(root);
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      ret.push_back(cur->val);
      if (cur->left != nullptr) st.push(cur->left);
      if (cur->right != nullptr) st.push(cur->right);
    }
    reverse(ret.begin(), ret.end());
    return ret;
  }
};
```

```
class Solution {
public:
```

```
  vector<int> postorderTraversal(TreeNode* root) {
    if (!root) return {};
    stack<TreeNode*> st;
    st.push(root);
    vector<int> ret;
    unordered_set<TreeNode*> visited;
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      if ((!cur->left || visited.count(cur->left)) && (!cur->right ||
visited.count(cur->right))) {
        ret.push_back(cur->val);
        visited.insert(cur);
      } else {
        st.push(cur);
        if (cur->right) st.push(cur->right);
        if (cur->left) st.push(cur->left);
      }
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> postorderTraversal(TreeNode* root) {
    if (!root) return {};
    vector<int> ret = postorderTraversal(root->left);
    vector<int> right = postorderTraversal(root->right);
    ret.insert(ret.end(), right.begin(), right.end());
    ret.push_back(root->val);
    return ret;
  }
};
```

146. LRU Cache

Description

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

The cache is initialized with a positive capacity.

Follow up:
Could you do both operations in O(1) time complexity?

Example:

```
LRUCache cache = new LRUCache( 2 /* capacity */ );

cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // returns 1
cache.put(3, 3);    // evicts key 2
cache.get(2);       // returns -1 (not found)
cache.put(4, 4);    // evicts key 1
cache.get(1);       // returns -1 (not found)
cache.get(3);       // returns 3
cache.get(4);       // returns 4
```

Solution

Discussion:

- Use an unordered_map<int, int> cache to store the key-value pairs.
- Use an unordered_map<int, int> priority to store the current rank for a certain key.
- Use a max-heap (priority_queue) to store the priorities of key's. The higher rank of the key, the earlier that the key will be removed from the cache.
- When we call get(key), we update the rank for the current key by priority[key] = rank and then push this pair {priority[key], key} to the priority_queue. Therefore, the new pair would invalidate the original {rank, key} in the priority_queue. So inside the the put method, we use a while loop to invalidate all those invalid pairs until the first valid pair with the highest rank.
- Time complexity: get: O(1); put: O(m), where m is the number of operations of get.
- Space complexity: O(n + m), where n is the number of key-value pairs and m is s the number of operations of get.

```
class LRUCache {
private:
  unordered_map<int, int> cache;
  unordered_map<int, int> priority;
  priority_queue<pair<int, int>, vector<pair<int, int>>, less<pair<int, int>>>
pq;  // max-heap
  int capacity, rank;

public:
```

```
  LRUCache(int capacity) {
    this->capacity = capacity;
    rank = INT_MAX;
    cache.clear();
    priority.clear();
  }

  int get(int key) {
    if (cache.count(key) == 0) {
      return -1;
    } else {
      priority[key] = rank--;
      priority.emplace(key, priority[key]);
      pq.emplace(priority[key], key);
      return cache[key];
    }
  }

  void put(int key, int value) {
    cache[key] = value;
    priority[key] = rank--;
    pq.emplace(priority[key], key);
    while (cache.size() > capacity) {
      pair<int, int> top = pq.top();
      while (!pq.empty() && priority[top.second] != top.first) {
        pq.pop();
        top = pq.top();
      }
      pq.pop();
      cache.erase(top.second);
      priority.erase(top.second);
    }
  }
};
```

153. Find Minimum in Rotated Sorted Array

Description

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

Find the minimum element.

You may assume no duplicate exists in the array.

Example 1:

Input: [3,4,5,1,2]
Output: 1
Example 2:

Input: [4,5,6,7,0,1,2]
Output: 0

Solution

04/28/2020:

```
class Solution {
public:
  int findMin(vector<int>& nums) {
    if (nums.empty()) return -1;
    int n = nums.size();
    int lo = 0, hi = n - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] > nums[hi]) {
        lo = mid + 1;
      } else  {
        hi = mid;
      }
    }
    return nums[hi];
  }
};
```

162. Find Peak Element

Description

A peak element is an element that is greater than its neighbors.

Given an input array nums, where nums[i] ≠ nums[i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that nums[-1] = nums[n] = -∞.

Example 1:

Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
Example 2:

Input: nums = [1,2,1,3,5,6,4]
Output: 1 or 5
Explanation: Your function can return either index number 1 where the peak element is 2,
or index number 5 where the peak element is 6.
Note:

Your solution should be in logarithmic complexity.

Solution

04/28/2020:

```
class Solution {
public:
  int findPeakElement(vector<int>& nums) {
    if (nums.empty()) return -1;
    int lo = 0, hi = nums.size() - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (mid - 1 >= lo && nums[mid] < nums[mid - 1]) {
        hi = mid - 1;
      } else if (mid + 1 <= hi && nums[mid] < nums[mid + 1]) {
        lo = mid + 1;
      } else {
        return mid;
      }
    }
    return lo;
  }
};
```

163. Missing Ranges

Description

Given a sorted integer array nums, where the range of elements are in the inclusive range [lower, upper], return its missing ranges.

Example:

Input: nums = [0, 1, 3, 50, 75], lower = 0 and upper = 99,
Output: ["2", "4->49", "51->74", "76->99"]

Solution

05/24/2020:

```
class Solution {
public:
  vector<string> findMissingRanges(vector<int>& nums, int lower, int upper) {
    if (nums.empty()) {
      string s = lower == upper ? to_string(lower) : to_string(lower) + "->" +
to_string(upper);
      return {s};
    }
    long long cur = nums.back(); nums.pop_back();
    vector<string> ranges;
    string s = cur + 1 < upper ? to_string(cur + 1) + "->" + to_string(upper) :
to_string(cur + 1);
    if (cur - 1 >= lower) ranges = findMissingRanges(nums, lower, cur - 1);
    if (cur + 1 <= upper) ranges.push_back(s);
    return ranges;
  }
};
```

```
class Solution {
public:
  vector<string> findMissingRanges(vector<int>& nums, int lower, int upper) {
    if (lower > upper) return {};
    if (nums.empty()) {
      if (lower == upper)
        return { to_string(lower) };
      else
        return { to_string(lower) + "->" + to_string(upper) };
    }
    long long cur = nums.back(); nums.pop_back();
    vector<string> ranges;
    if (cur - 1 >= INT_MIN) {
       ranges = findMissingRanges(nums, lower, cur - 1);
    }
```

```
    if (cur + 1 < upper) {
      ranges.push_back(to_string(cur + 1) + "->" + to_string(upper));
    } else if (cur + 1 == upper) {
      ranges.push_back(to_string(cur + 1));
    }
    return ranges;
  }
};
```

169. Majority Element

Description

Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array.

Example 1:

Input: [3,2,3]
Output: 3
Example 2:

Input: [2,2,1,1,1,2,2]
Output: 2

Solution

05/06/2020:

```
class Solution {
public:
  int majorityElement(vector<int>& nums) {
    int n = nums.size();
    if (n == 0) return -1;
    unordered_map<int, int> s;
    int major = nums[0], cnt = 1;
    for (auto& i : nums) {
      if (++s[i] > cnt) {
        cnt = s[i];
        major = i;
      }
    }
```

```
    return cnt > n / 2 ? major : -1;
  }
};
```

173. Binary Search Tree Iterator

Description

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.

Example:

```
BSTIterator iterator = new BSTIterator(root);
iterator.next();    // return 3
iterator.next();    // return 7
iterator.hasNext(); // return true
iterator.next();    // return 9
iterator.hasNext(); // return true
iterator.next();    // return 15
iterator.hasNext(); // return true
iterator.next();    // return 20
iterator.hasNext(); // return false
```

Note:

next() and hasNext() should run in average O(1) time and uses O(h) memory, where h is the height of the tree.
You may assume that next() call will always be valid, that is, there will be at least a next smallest number in the BST when next() is called.

Solution

05/24/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
```

```
 *      int val;
 *      TreeNode *left;
 *      TreeNode *right;
 *      TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *      TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *      TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class BSTIterator {
private:
  stack<TreeNode*> st;
  TreeNode* cur;

public:
  BSTIterator(TreeNode* root) {
    cur = root;
  }

  /** @return the next smallest number */
  int next() {
    while (cur) {
      st.push(cur);
      cur = cur->left;
    }
    cur = st.top(); st.pop();
    int ret = cur->val;
    cur = cur->right;
    return ret;
  }

  /** @return whether we have a next smallest number */
  bool hasNext() {
    return cur || !st.empty();
  }
};

/**
 * Your BSTIterator object will be instantiated and called as such:
 * BSTIterator* obj = new BSTIterator(root);
 * int param_1 = obj->next();
 * bool param_2 = obj->hasNext();
 */
```

```
class BSTIterator {
private:
  vector<TreeNode*> inorder;
```

```
  vector<TreeNode*> inorderTraversal(TreeNode* root) {
    if (root == nullptr) return {};
    vector<TreeNode*> leftTraversal = inorderTraversal(root->left);
    vector<TreeNode*> rightTraversal = inorderTraversal(root->right);
    leftTraversal.push_back(root);
    leftTraversal.insert(leftTraversal.end(), rightTraversal.begin(),
rightTraversal.end());
    return leftTraversal;
  }

public:
  BSTIterator(TreeNode* root) {
    inorder = inorderTraversal(root);
    reverse(inorder.begin(), inorder.end());
  }

  /** @return the next smallest number */
  int next() {
    TreeNode* ret = inorder.back();
    inorder.pop_back();
    return ret->val;
  }

  /** @return whether we have a next smallest number */
  bool hasNext() {
    return !inorder.empty();
  }
};
```

174. Dungeon Game

Description

The demons had captured the princess (P) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (K) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.

Some of the rooms are guarded by demons, so the knight loses health (negative integers) upon entering these rooms; other rooms are either empty (0's) or contain magic orbs that increase the knight's health (positive integers).

In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.

Write a function to determine the knight's minimum initial health so that he is able to rescue the princess.

For example, given the dungeon below, the initial health of the knight must be at least 7 if he follows the optimal path RIGHT-> RIGHT -> DOWN -> DOWN.

```
-2 (K)  -3  3
-5  -10 1
10  30  -5 (P)
```

Note:

The knight's health has no upper bound.
Any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.

Solution

06/21/2020:

```
class Solution {
public:
  const int inf = numeric_limits<int>::max();
  vector<vector<int>> dp;
  int rows, cols;

  int getMinHealth(int curCell, int nextRow, int nextCol) {
    if (nextRow >= rows || nextCol >= cols) return inf;
    int nextCell = dp[nextRow][nextCol];
    return max(1, nextCell - curCell);
  }

  int calculateMinimumHP(vector<vector<int>>& dungeon) {
    rows = dungeon.size();
    cols = dungeon[0].size();
    dp.assign(rows, vector<int>(cols, inf));
    int curCell, rightHealth, downHealth, nextHealth, minHealth;
    for (int row = rows - 1; row >= 0; --row) {
      for (int col = cols - 1; col >= 0; --col) {
        curCell = dungeon[row][col];
        rightHealth = getMinHealth(curCell, row, col + 1);
        downHealth = getMinHealth(curCell, row +  1, col);
```

```
            nextHealth = min(rightHealth, downHealth);
            if (nextHealth != inf) {
              minHealth = nextHealth;
            } else {
              minHealth = curCell >= 0 ? 1 : 1 - curCell;
            }
            dp[row][col] = minHealth;
          }
        }
        return dp[0][0];
    }
};
```

198. House Robber

Description

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

Example 1:

Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
Example 2:

Input: [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.

Solution

01/14/2020 (Dynamic Programming):

```
class Solution {
public:
  int rob(vector<int>& nums) {
    if (nums.size() >= 2) nums[1] = max(nums[0], nums[1]);
    for (int i = 2; i < nums.size(); ++i)
        nums[i] = max(nums[i - 1], nums[i - 2] + nums[i]);
    return nums.size() > 0 ? nums.back() : 0;
  }
};
```

200. Number of Islands

Description

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

Input:
11110
11010
11000
00000

Output: 1
Example 2:

Input:
11000
11000
00100
00011

Output: 3

Solution

05/08/2020 (Union-Find):

```
class UnionFind {
private:
  vector<int> id;
```

```
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
      id[j] = i;
      sz[i] += sz[j];
    } else {
      id[i] = j;
      sz[j] += sz[i];
    }
    return true;
  }
};

class Solution {
public:
  int numIslands(vector<vector<char>>& grid) {
    if (grid.empty() || grid[0].empty()) return 0;
    int m = grid.size(), n = grid[0].size();
    UnionFind uf(m * n);
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == '0') continue;
        for (int d = 0; d < 4; ++d) {
          int ni = i + dir[d][0], nj = j + dir[d][1];
          if (ni >= 0 && ni < m && nj >= 0 && nj < n && grid[ni][nj] == '1') {
            uf.merge(i * n + j, ni * n + nj);
          }
        }
      }
    }
    unordered_set<int> components;
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
```

```
        if (grid[i][j] == '1')
          components.insert(uf.find(i * n + j));
    return components.size();
  }
};
```

```
class Solution {
public:
  int numIslands(vector<vector<char>>& grid) {
    int m = grid.size();
    if (m == 0) return 0;
    int n = grid[0].size(), ret = 0;
    stack<pair<int, int>> st;
    int d[4][2] = { {0, -1}, {0, 1}, {-1, 0}, {1, 0} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == '1') {
          ++ret;
          st.emplace(i, j);
          while (!st.empty()) {
            pair<int, int> cur = st.top(); st.pop();
            grid[cur.first][cur.second] = '0';
            for (int k = 0; k < 4; ++k) {
              int ni = cur.first + d[k][0], nj = cur.second + d[k][1];
              if (ni > -1 && ni < m && nj > -1 && nj < n && grid[ni][nj] == '1')
st.emplace(ni, nj);
            }
          }
        }
      }
    }
    return ret;
  }
};
```

Iterative DFS:

```
class Solution {
public:
  int numIslands(vector<vector<char>>& grid) {
    int m = grid.size();
    if (m == 0) return 0;
    int n = grid[0].size(), ret = 0;
    stack<pair<int, int>> st;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == '1') {
```

```
            ++ret;
            st.emplace(i, j);
            while (!st.empty()) {
              pair<int, int> cur = st.top(); st.pop();
              int ni = cur.first, nj = cur.second;
              grid[ni][nj] = '0';
              if (ni + 1 < m && grid[ni + 1][nj] == '1') st.emplace(ni + 1, nj);
              if (ni - 1 >= 0 && grid[ni - 1][nj] == '1') st.emplace(ni - 1, nj);
              if (nj + 1 < n && grid[ni][nj + 1] == '1') st.emplace(ni, nj + 1);
              if (nj - 1 >= 0 && grid[ni][nj - 1] == '1') st.emplace(ni, nj - 1);
            }
          }
        }
      }
      return ret;
    }
};
```

Recursive DFS:

```
class Solution {
public:
  int numIslands(vector<vector<char>>& grid) {
    int m = grid.size();
    if (m == 0) return 0;
    int n = grid[0].size();
    int ret = 0;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == '1') {
          ++ret;
          dfs(grid, i, j, m, n);
        }
      }
    }
    return ret;
  }

  void dfs(vector<vector<char>>& grid, int i, int j, int m, int n) {
    if (i >= 0 && i < m && j >= 0 && j < n && grid[i][j] == '1') {
      grid[i][j] = '0';
      dfs(grid, i - 1, j, m, n);
      dfs(grid, i + 1, j, m, n);
      dfs(grid, i, j - 1, m, n);
      dfs(grid, i, j + 1, m, n);
    }
  }
};
```

205. Isomorphic Strings

Description

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

Example 1:

Input: s = "egg", t = "add"
Output: true
Example 2:

Input: s = "foo", t = "bar"
Output: false
Example 3:

Input: s = "paper", t = "title"
Output: true
Note:
You may assume both s and t have the same length.

Solution

05/20/2020:

```
class Solution {
public:
  bool isIsomorphic(string s, string t) {
    unordered_map<char, unordered_set<char>> mpST, mpTS;
    for (int i = 0; i < (int)s.size(); ++i) {
      mpST[s[i]].insert(t[i]);
      mpTS[t[i]].insert(s[i]);
    }
    for (auto& m : mpST)
      if (m.second.size() > 1)
        return false;
    for (auto& m : mpTS)
      if (m.second.size() > 1)
        return false;
```

```
    return true;
  }
};
```

207. Course Schedule

Description

There are a total of numCourses courses you have to take, labeled from 0 to numCourses-1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, is it possible for you to finish all courses?

Example 1:

Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
Example 2:

Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should
also have finished course 1. So it is impossible.

Constraints:

The input prerequisites is a graph represented by a list of edges, not adjacency matrices. Read more about how a graph is represented.
You may assume that there are no duplicate edges in the input prerequisites.
1 <= numCourses <= 10^5

Solution

05/29/2020:

```cpp
class Solution {
public:
  bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
    vector<vector<int>> adj(numCourses);
    for (auto& p : prerequisites) adj[p[1]].push_back(p[0]);
    vector<bool> path(numCourses);
    for (int i = 0; i < numCourses; ++i)
      if (isCyclic(i, adj, path))
        return false;
    return true;
  }

  bool isCyclic(int i, vector<vector<int>>& adj, vector<bool>& path) {
    if (path[i]) return true;
    if (adj[i].empty()) return false;
    path[i] = true;
    bool ret = false;
    for (auto& j : adj[i]) {
      ret = isCyclic(j, adj, path);
      if (ret) break;
    }
    path[i] = false;
    return ret;
  }
};
```

208. Implement Trie (Prefix Tree)

Description

Implement a trie with insert, search, and startsWith methods.

Example:

```
Trie trie = new Trie();

trie.insert("apple");
trie.search("apple");   // returns true
trie.search("app");     // returns false
trie.startsWith("app"); // returns true
trie.insert("app");
trie.search("app");     // returns true
```

Note:

You may assume that all inputs are consist of lowercase letters a-z.
All inputs are guaranteed to be non-empty strings.

Solution

05/13/2020:

```
class Trie {
public:
  /** Initialize your data structure here. */
  Trie() {
    root = new Node();
  }

  /** Inserts a word into the trie. */
  void insert(string word) {
    Node* cur = root;
    for (auto& c : word) {
      if (!cur->mp[c])
        cur->mp[c] = new Node();
      cur = cur->mp[c];
    }
    cur->isWord = true;
  }

  /** Returns if the word is in the trie. */
  bool search(string word) {
    Node* cur = find(word);
    return cur != nullptr && cur->isWord;
  }

  /** Returns if there is any word in the trie that starts with the given
prefix. */
  bool startsWith(string prefix) {
    Node* cur = find(prefix);
    return cur != nullptr;
  }

private:
  struct Node {
    bool isWord;
    unordered_map<char, Node*> mp;
    Node() : isWord(false) {}
  };

  Node* root;
```

```
  Node* find(const string& word) {
    Node* cur = root;
    for (auto& c : word) {
      cur = cur->mp[c];
      if (!cur) break;
    }
    return cur;
  }
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```

```
class Trie {
private:
  struct Node {
    bool isWord;
    vector<Node*> children;
    Node() { isWord = false; children.resize(26, nullptr); }
    ~Node() { for(auto& c : children) delete c; }
  };

  Node* root;

  Node* find(const string& word) {
    Node* cur = root;
    for (auto& c : word) {
      cur = cur->children[c - 'a'];
      if (!cur) break;
    }
    return cur;
  }

public:
  /** Initialize your data structure here. */
  Trie() {
    root = new Node();
  }

  /** Inserts a word into the trie. */
  void insert(string word) {
    Node* cur = root;
    for (auto& c : word) {
```

```
      if (!cur->children[c - 'a'])
        cur->children[c - 'a'] = new Node();
      cur = cur->children[c - 'a'];
    }
    cur->isWord = true;
  }

  /** Returns if the word is in the trie. */
  bool search(string word) {
    Node* cur = find(word);
    return cur && cur->isWord;
  }

  /** Returns if there is any word in the trie that starts with the given
prefix. */
  bool startsWith(string prefix) {
    Node* cur = find(prefix);
    return cur;
  }
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```

```
struct Node {
  char character;
  bool isWord;
  vector<Node*> children;
  Node(char c, bool i) : character(c), isWord(i) {}
  ~Node() {
    for (auto& c : children) delete c;
  }
};

class Trie {
private:
  Node *root;

public:
  /** Initialize your data structure here. */
  Trie() {
    root = new Node('r', false);
  }
```

```
  /** Inserts a word into the trie. */
  void insert(string word) {
    int n = word.size();
    Node* cur = root;
    for (int i = 0; i < n; ++i) {
      bool foundCharacter = false;
      for (auto& c : cur->children) {
        if (word[i] == c->character) {
          cur = c;
          foundCharacter = true;
          break;
        }
      }
      if (!foundCharacter) {
        Node* c = new Node(word[i], false);
        cur->children.push_back(c);
        cur = c;
      }
    }
    cur->isWord = true;
  }

  /** Returns if the word is in the trie. */
  bool search(string word) {
    Node* cur = root;
    int n = word.size();
    for (int i = 0; i < n; ++i) {
      bool foundCharacter = false;
      for (auto& c : cur->children) {
        if (c->character == word[i]) {
          foundCharacter = true;
          cur = c;
          break;
        }
      }
      if (!foundCharacter) return false;
    }
    return cur->isWord;
  }

  /** Returns if there is any word in the trie that starts with the given
prefix. */
  bool startsWith(string prefix) {
    Node* cur = root;
    int n = prefix.size();
    for (int i = 0; i < n; ++i) {
      bool foundCharacter = false;
      for (auto& c : cur->children) {
```

```
        if (c->character == prefix[i]) {
          foundCharacter = true;
          cur = c;
          break;
        }
      }
      if (!foundCharacter) return false;
    }
    return true;
  }
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```

219. Contains Duplicate II

Description

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the absolute difference between i and j is at most k.

Example 1:

Input: nums = [1,2,3,1], k = 3
Output: true
Example 2:

Input: nums = [1,0,1,1], k = 1
Output: true
Example 3:

Input: nums = [1,2,3,1,2,3], k = 2
Output: false

Solution

05/20/2020:

```cpp
class Solution {
public:
  bool containsNearbyDuplicate(vector<int>& nums, int k) {
    unordered_map<int, int> mp;
    int n = nums.size();
    for (int i = 0; i < n; ++i) {
      if (mp.count(nums[i]) > 0) {
        if (i - mp[nums[i]] <= k) {
          return true;
        }
      }
      mp[nums[i]] = i;
    }
    return false;
  }
};
```

```cpp
class Solution {
public:
  bool containsNearbyDuplicate(vector<int>& nums, int k) {
    unordered_set<int> visited;
    for (int i = 0; i < (int)nums.size(); ++i) {
      if (visited.count(nums[i]) > 0) return true;
      visited.insert(nums[i]);
      if (i - k >= 0) visited.erase(nums[i - k]);
    }
    return false;
  }
};
```

221. Maximal Square

Description

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.

Example:

Input:

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Output: 4

Solution

04/27/2020 (Dynamic Programming):

```
class Solution {
public:
  int maximalSquare(vector<vector<char>>& matrix) {
    int m = matrix.size();
    if (m == 0) return 0;
    int n = matrix[0].size();
    int ret = 0;
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        dp[i + 1][j + 1] = matrix[i][j] - '0';
        if (dp[i + 1][j + 1] > 0) {
          dp[i + 1][j + 1] = min(dp[i][j], min(dp[i][j + 1], dp[i + 1][j])) + 1;
          ret = max(ret, dp[i + 1][j + 1]);
        }
      }
    }
    return ret * ret;
  }
};
```

More space-efficient Dynamic Programming:

```
class Solution {
public:
  int maximalSquare(vector<vector<char>>& matrix) {
    if (matrix.size() == 0 || matrix[0].size() == 0) return 0;
    int m = matrix.size(), n = matrix[0].size();
    int ret = 0;
```

```
    vector<int> dp(n + 1, 0);
    for (int i = 0; i < m; ++i) {
      vector<int> tmp(dp);
      for (int j = 0; j < n; ++j) {
        dp[j + 1] = matrix[i][j] - 'O';
        if (dp[j + 1] > 0) {
          dp[j + 1] = min(tmp[j], min(tmp[j + 1], dp[j])) + 1;
          ret = max(ret, dp[j + 1]);
        }
      }
    }
    return ret * ret;
  }
};
```

222. Count Complete Tree Nodes

Description

Given a complete binary tree, count the number of nodes.

Note:

Definition of a complete binary tree from Wikipedia:
In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

Example:

Input:

```
    1
   / \
  2   3
 / \  /
4  5 6
```

Output: 6

Solution

06/23/2020:

```
/**
 * Definition for a binary tree node.
```

```
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  int countNodes(TreeNode* root) {
    if (!root) return 0;
    queue<TreeNode*> q;
    q.push(root);
    int ret = 0;
    while (!q.empty()) {
      int sz = q.size();
      ret += sz;
      for (int i = 0; i < sz; ++i) {
        TreeNode* cur = q.front(); q.pop();
        if (cur->right) q.push(cur->right);
        if (cur->left) q.push(cur->left);
      }
    }
    return ret;
  }
};
```

226. Invert Binary Tree

Description

Invert a binary tree.

Example:

Input:

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

Output:

```
     4
   /   \
```

```
  7      2
 / \/     \
9   6 31
```

Trivia:

This problem was inspired by this original tweet by Max Howell:

Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so f*** off.

Solution

06/01/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  TreeNode* invertTree(TreeNode* root) {
    if (root == nullptr) return nullptr;
    swap(root->left, root->right);
    invertTree(root->left);
    invertTree(root->right);
    return root;
  }
};
```

230. Kth Smallest Element in a BST

Description

Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.

Example 1:

```
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
```

Example 2:

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
```

Follow up:
What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

Constraints:

The number of elements of the BST is between 1 to 10^4.
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
```

```
class Solution {
public:
  int kthSmallest(TreeNode* root, int k) {
    stack<TreeNode*> st;
    TreeNode* cur = root;
    while (cur || !st.empty()) {
      while (cur) {
        st.push(cur);  cur
        = cur->left;
      }
      cur = st.top(); st.pop();
      if (--k == 0) return cur->val;
      cur = cur->right;
    }
    return 0;
  }
};
```

```
class Solution {
public:
  int kthSmallest(TreeNode* root, int k) {
    stack<TreeNode*> st;
    st.push(root);
    unordered_set<TreeNode*> visited;
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      if (cur->left && visited.count(cur->left) == 0) {
        if (cur->right) st.push(cur->right);
        st.push(cur);
        st.push(cur->left);
      } else {
        if (!cur->left && cur->right) st.push(cur->right);
        visited.insert(cur);
        if (--k == 0) return cur->val;
      }
    }
    return 0;
  }
};
```

```
class Solution {
public:
  int n, ans;
  int kthSmallest(TreeNode* root, int k) {
    n = k;
    inorder(root);
    return ans;
```

```
  }

  void inorder(TreeNode* root) {
    if (!root) return;
    inorder(root->left);
    if (--n == 0) ans = root->val;
    inorder(root->right);
  }
};
```

```
class Solution {
public:
  int kthSmallest(TreeNode* root, int k) {
    vector<int> list = BST2vector(root);
    return list[k - 1];
  }

  vector<int> BST2vector(TreeNode* root) {
    if (!root) return {};
    vector<int> ret = BST2vector(root->left);
    ret.push_back(root->val);
    vector<int> right = BST2vector(root->right);
    ret.insert(ret.end(), right.begin(), right.end());
    return ret;
  }
};
```

231. Power of Two

Description

Given an integer, write a function to determine if it is a power of two.

Example 1:

Input: 1
Output: true
Explanation: 20 = 1
Example 2:

Input: 16
Output: true
Explanation: 24 = 16
Example 3:

Input: 218
Output: false

Solution

06/07/2020:

```
class Solution {
public:
  bool isPowerOfTwo(int n) {
    return n > 0 && __builtin_popcount(n) == 1;
  }
};
```

237. Delete Node in a Linked List

Description

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Given linked list -- head = [4,5,1,9], which looks like following:

Example 1:

Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.
Example 2:

Input: head = [4,5,1,9], node = 1
Output: [4,5,9]
Explanation: You are given the third node with value 1, the linked list should become 4 -> 5 -> 9 after calling your function.

Note:

The linked list will have at least two elements.
All of the nodes' values will be unique.

The given node will not be the tail and it will always be a valid node of the linked list.
Do not return anything from your function.

Solution

05/10/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
  void deleteNode(ListNode* node) {
    if (node == nullptr) return;
    if (node->next == nullptr) {
      node = node->next;
    } else {
      node->val = node->next->val;
      node->next = node->next->next;
      node = node->next;
    }
  }
};
```

06/02/2020:

```
class Solution {
public:
  void deleteNode(ListNode* node) {
    *node = *(node->next);
  }
};
```

```
class Solution {
public:
  void deleteNode(ListNode* node) {
    node->val = node->next->val;
    node->next = node->next->next;
  }
};
```

242. Valid Anagram

Description

Given two strings s and t , write a function to determine if t is an anagram of s.

Example 1:

Input: s = "anagram", t = "nagaram"
Output: true
Example 2:

Input: s = "rat", t = "car"
Output: false
Note:
You may assume the string contains only lowercase alphabets.

Follow up:
What if the inputs contain unicode characters? How would you adapt your solution to such case?

Solution

05/17/2020:

```
class Solution {
public:
  bool isAnagram(string s, string t) {
    vector<int> cntS(26, 0), cntT(26, 0);
    for (int i = 0; i < (int)s.size(); ++i) ++cntS[s[i] - 'a'];
    for (int i = 0; i < (int)t.size(); ++i) ++cntT[t[i] - 'a'];
    for (int i = 0; i < 26; ++i) {
      if (cntS[i] != cntT[i]) {
        return false;
      }
    }
    return true;
  }
};
```

```
class Solution {
public:
  bool isAnagram(string s, string t) {
    sort(s.begin(), s.end());
    sort(t.begin(), t.end());
    return s == t;
  }
};
```

246. Strobogrammatic Number

Description

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Write a function to determine if a number is strobogrammatic. The number is represented as a string.

Example 1:

Input: "69"
Output: true
Example 2:

Input: "88"
Output: true
Example 3:

Input: "962"
Output: false

Solution

05/18/2020:

```
class Solution {
public:
  bool isStrobogrammatic(string num) {
    unordered_map<char, char> mp{ {'0', '0'}, {'1', '1'}, {'6', '9'}, {'8',
'8'}, {'9', '6'} };
    string stro = "";
    for (auto& n : num) {
      if (mp.count(n) == 0) return false;
      stro = mp[n] + stro;
    }
    return stro == num;
  }
};
```