目录

101. Symmetric Tree 4
102. Binary Tree Level Order Traversal 6
103. Binary Tree Zigzag Level Order Traversal 8
104. Maximum Depth of Binary Tree 10
105. Construct Binary Tree from Preorder and Inorder Traversal 12
106. Construct Binary Tree from Inorder and Postorder Traversal 14
107. Binary Tree Level Order Traversal II 16
108. Convert Sorted Array to Binary Search Tree 18
109. Convert Sorted List to Binary Search Tree★★ 20
110. Balanced Binary Tree 22
111. Minimum Depth of Binary Tree 24
112. Path Sum 26
113. Path Sum II 28
114. Flatten Binary Tree to Linked List★★ 30
115. Distinct Subsequences★★ 32
116. Populating Next Right Pointers in Each Node 34
117. Populating Next Right Pointers in Each Node II★★ 36
118. Pascal's Triangle 38
119. Pascal's Triangle II 40
120. Triangle 42
121. Best Time to Buy and Sell Stock 44
122. Best Time to Buy and Sell Stock II 46
123. Best Time to Buy and Sell Stock III★★ 48
124. Binary Tree Maximum Path Sum★★ 51
125. Valid Palindrome 53
126. Word Ladder II★★ 55
127. Word Ladder 58
128. Longest Consecutive Sequence★★ 60
129. Sum Root to Leaf Numbers 63
130. Surrounded Regions 65
131. Palindrome Partitioning★★ 67
132. Palindrome Partitioning II★★ 69
133. Clone Graph★★ 71
134. Gas Station 74
135. Candy 76
136. Single Number 78
137. Single Number II★★ 79
138. Copy List with Random Pointer 82
139. Word Break 84
140. Word Break II★★ 86
141. Linked List Cycle 89
142. Linked List Cycle II 92

143. Reorder List....95
144. Binary Tree Preorder Traversal....97
145. Binary Tree Postorder Traversal★★....99
146. LRU Cache★★....102
147. Insertion Sort List....104
148. Sort List★★....106
149. Max Points on a Line★★....108
150. Evaluate Reverse Polish Notation....110
151. Reverse Words in a String....112
152. Maximum Product Subarray....114
153. Find Minimum in Rotated Sorted Array....116
154. Find Minimum in Rotated Sorted Array II★★....118
155. Min Stack....120
156.Binary Tree Upside Down....122
157.Read N Characters Given Read4....124
158.Read N Characters Given Read4 II - Call multiple times★★....127
159.Longest Substring with At Most Two Distinct Characters....130
160. Intersection of Two Linked Lists....131
161. One Edit Distance....134
162. Find Peak Element....136
163.Missing Ranges....138
164. Maximum Gap★★....139
165. Compare Version Numbers....141
166. Fraction to Recurring Decimal....144
167. Two Sum II - Input array is sorted....146
168. Excel Sheet Column Title....148
169. Majority Element....150
170. Two Sum III - Data structure design....152
171. Excel Sheet Column Number....154
172. Factorial Trailing Zeroes★★....156
173. Binary Search Tree Iterator....158
174. Dungeon Game★★....160
175. Combine Two Tables(SQL)....162
176. Second Highest Salary(SQL)....164
177. Nth Highest Salary(SQL)....166
178. Rank Scores(SQL)....167
179. Largest Number....169
180. Consecutive Numbers(SQL)....171
181. Employees Earning More Than Their Managers(SQL)....173
182. Duplicate Emails(SQL)....175
183. Customers Who Never Order(SQL)....177
184. Department Highest Salary(SQL)....179
185. Department Top Three Salaries(SQL)....181
186. Reverse Words in a String II....183

187. Repeated DNA Sequences 184
188. Best Time to Buy and Sell Stock IV★★ 186
189. Rotate Array 188
190. Reverse Bits 190
191. Number of 1 Bits 192
192. Word Frequency(Bash) 194
193. Valid Phone Numbers(Bash) 195
194. Transpose File(Bash) 196
195. Tenth Line(Bash) 197
196. Delete Duplicate Emails(SQL) 198
197. Rising Temperature(SQL) 200
198. House Robber 201
199. Binary Tree Right Side View 203
200. Number of Islands 205

101. Symmetric Tree

Easy

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:

```
    1
   /\
  2   2
 /\/\
3  44  3
```

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

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

Note:

Bonus points if you could solve it both recursively and iteratively.

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {

    }
};
```

```cpp
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if (!root) return true;
        stack<TreeNode*> s;
        s.push(root->left);   s.push(root->right);
        while (!s.empty()) {
            TreeNode *p = s.top();  s.pop();
            TreeNode *q = s.top();  s.pop();
            if (!p && !q) continue;
            if (!p || !q) return false;
            if (p->val != q->val) return false;
            s.push(p->left);
            s.push(q->right);
            s.push(p->right);
            s.push(q->left);
        }
        return true;
    }
};
```

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

private:
    bool isSame(TreeNode *l, TreeNode *r) {
        if (!l || !r) return l == r;
        else if (l->val != r->val) return false;
        else return isSame(l->left, r->right) && isSame(r->left, l->right);
    }
};
```

102. Binary Tree Level Order Traversal

Medium

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]
]
```

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {

    }
};
```

```cpp
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if (root == nullptr) return res;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int sz = q.size();
            vector<int> v;
            while (sz--) {
               TreeNode *t = q.front();
               q.pop();
               if (t->left) q.push(t->left);
               if (t->right) q.push(t->right);
               v.push_back(t->val);
            }
            res.push_back(v);
        }
        return res;
    }
};
```

103. Binary Tree Zigzag Level Order Traversal

Medium

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]
]
```

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    }
};
```

```cpp
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        int k = 0;
        if (root == nullptr) return res;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int sz = q.size();
            vector<int> v;
            while (sz--) {
               TreeNode *t = q.front();
               q.pop();
               if (t->left) q.push(t->left);
               if (t->right) q.push(t->right);
               v.push_back(t->val);
            }
            if (k) reverse(v.begin(), v.end());
            k ^= 1;
            res.push_back(v);
        }
        return res;
    }
};
```

104. Maximum Depth of Binary Tree

Easy

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree `[3,9,20,null,null,15,7]`,

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

return its depth = 3.

```
class Solution {
public:
    int maxDepth(TreeNode* root) {

    }
};
```

```cpp
class Solution {
public:
    int maxDepth(TreeNode *root) {
       return !root ? 0 : max(maxDepth(root->left),maxDepth(root->right)) + 1;
    }
};
```

105. Construct Binary Tree from Preorder and Inorder Traversal

Medium

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
```

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {

    }
};
```

```
class Solution {
public:
    TreeNode* buildTree(vector<int> &preorder, vector<int> &inorder) {
        return build(0, 0, preorder.size()-1, preorder, inorder);
    }

 private:
    TreeNode *build(int prel, int inl, int inr, vector<int>& preorder,
                    vector<int>& inorder) {
        if (inl > inr) return nullptr;
        int i = inl;
        while (inorder[i] != preorder[prel]) i++;
        TreeNode *t = new TreeNode(inorder[i]);
        t->left = build(prel+1, inl, i-1, preorder, inorder);
        t->right = build(prel+i+1-inl, i+1, inr, preorder, inorder);
        return t;
    }
};
```

106. Construct Binary Tree from Inorder and Postorder Traversal

Medium

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

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

For example, given

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

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

Return the following binary tree:

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

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
    }
};
```

```cpp
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        return build(postorder.size()-1,0,inorder.size()-1,postorder,inorder);
    }

private:
    TreeNode *build(int postr, int inl, int inr, vector<int>& postorder,
                    vector<int>& inorder) {
        if (inl > inr) return nullptr;
        int i = inl;
        while (inorder[i] != postorder[postr]) i++;
        TreeNode *t = new TreeNode(inorder[i]);
        t->left = build(i-1-inr+postr, inl, i-1, postorder, inorder);
        t->right = build(postr-1, i+1, inr, postorder, inorder);
        return t;
    }
};
```

107. Binary Tree Level Order Traversal II

Easy

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

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

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

return its bottom-up level order traversal as:

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

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {

    }
};
```

```cpp
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> res;
        if (root == nullptr) return res;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int sz = q.size();
            vector<int> v;
            while (sz--) {
               TreeNode *t = q.front();
               q.pop();
               if (t->left) q.push(t->left);
               if (t->right) q.push(t->right);
               v.push_back(t->val);
            }
            res.push_back(v);
        }
        reverse(res.begin(), res.end());
        return res;
    }
};
```

108. Convert Sorted Array to Binary Search Tree

Easy

Given an array 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 array: [-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
```

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {

    }
};
```

```cpp
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return sortedArrayToBST(nums, 0, nums.size()-1);
    }

private:
    TreeNode* sortedArrayToBST(vector<int>& nums, int begin, int end) {
        if (begin > end) return nullptr;
        int mid = begin+(end-begin)/2;
        TreeNode *node = new TreeNode(nums[mid]);
        node->left = sortedArrayToBST(nums, begin, mid-1);
        node->right = sortedArrayToBST(nums, mid+1, end);
        return node;
    }
};
```

109. Convert Sorted List to Binary Search Tree★★

Medium

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
```

```
class Solution {
public:
    TreeNode* sortedListToBST(ListNode* head) {

    }
};
```

```cpp
class Solution {
public:
    TreeNode* sortedListToBST(ListNode* head) {
        ListNode *p = head;
        int len = 0;
        while (p) {
            p = p->next;
            len++;
        }
        return sortedListToBST(head, 0, len-1);
    }
private:
    TreeNode* sortedListToBST(ListNode *&p, int left, int right) {
        if (left > right) return nullptr;
        int mid = left+(right-left)/2;
        TreeNode *left_tree = sortedListToBST(p, left, mid-1);
        TreeNode *node = new TreeNode(p->val);
        p = p->next;
        TreeNode *right_tree = sortedListToBST(p, mid+1, right);
        node->left = left_tree;
        node->right = right_tree;
        return node;
    }
};
```

110. Balanced Binary Tree

Easy

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 depth of the two subtrees of *every* node never differ by 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.

```
class Solution {
public:
    bool isBalanced(TreeNode* root) {

    }
};
```

```cpp
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        return get_height(root) != -1;
    }
    int get_height(TreeNode *p) {
        if (p == nullptr) return 0;
        int l = get_height(p->left), r = get_height(p->right);
        if (abs(l-r) > 1 || l == -1 || r == -1) return -1;
        else return max(l, r)+1;
    }
};
```

111. Minimum Depth of Binary Tree

Easy

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree `[3,9,20,null,null,15,7]`,

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

return its minimum depth = 2.

```
class Solution {
public:
    int minDepth(TreeNode* root) {

    }
};
```

```
//////////////////////////////////BFS//////////////////////////
class Solution {
public:
    int minDepth(TreeNode* root) {
        if (root == nullptr) return 0;
        queue<TreeNode*> q;
        q.push(root);
        int res = 1;
        while (!q.empty()) {
            int sz = q.size();
            while (sz--) {
                TreeNode *t = q.front();
                q.pop();
                if (t->left == nullptr && t->right == nullptr) return res;
                if (t->left) q.push(t->left);
                if (t->right) q.push(t->right);
            }
            res++;
        }
        return -1;
    }
};
```

```
//////////////////////////////////DFS//////////////////////////
class Solution {
public:
    int minDepth(TreeNode* root) {
        if (!root) return 0;
        int L = minDepth(root->left), R = minDepth(root->right);
        return 1 + ((L && R) ? min(L, R) : max(L, R));
    }
};
```

112. Path Sum

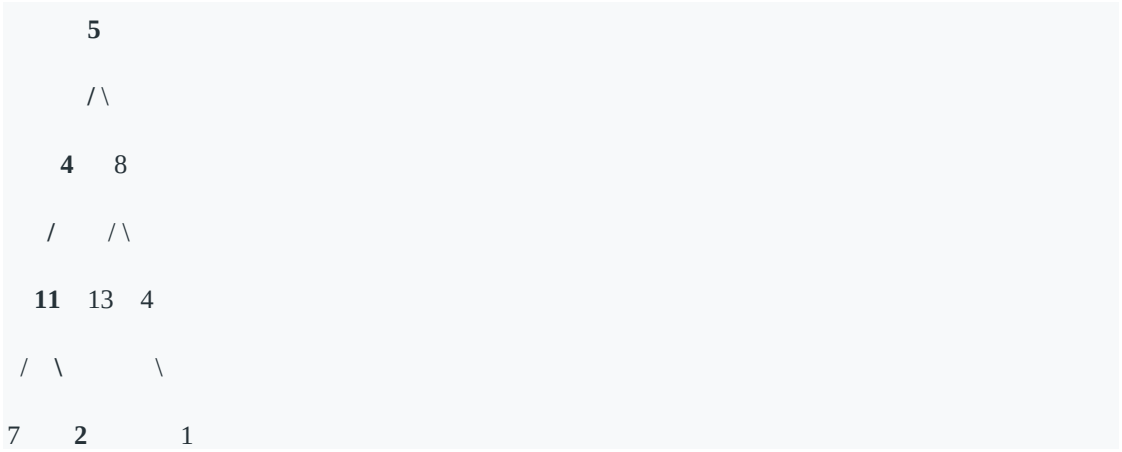
Easy

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22`,



return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

```
class Solution {
public:
    bool hasPathSum(TreeNode* root, int sum) {

    }
};
```

```cpp
class Solution {
public:
    bool hasPathSum(TreeNode* root, int sum) {
        if (!root) return false;
        return dfs(root, 0, sum);
    }
private:
    bool dfs(TreeNode *t, int cur, int &sum) {
        if (!t) return false;
        cur += t->val;
        if (!(t->left) && !(t->right)) return cur == sum;
        return dfs(t->left, cur, sum) || dfs(t->right, cur, sum);
    }
};
```

113. Path Sum II

Medium

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22`,

```
      5
      /\
     4   8
    /     /\
   11  13   4
  /  \       /\
 7    2     5   1
```

Return:

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

```
class Solution {
public:
    vector<vector<int>> pathSum(TreeNode* root, int sum) {
    }
};
```

```
class Solution {
public:
    vector<vector<int>> pathSum(TreeNode* root, int sum) {
        vector<int> cur;
        dfs(root, sum, cur);
        return res;
    }
private:
    vector<vector<int>> res;
     void dfs(TreeNode* root, int sum, vector<int> &cur) {
        if (!root) return;
        cur.push_back(root->val);
        if (!(root->left) && !(root->right) && sum == root->val)
            res.push_back(cur);
        dfs(root->left,  sum-root->val, cur);
        dfs(root->right, sum-root->val, cur);
        cur.pop_back();
    }
};
```

114. Flatten Binary Tree to Linked List★★

Medium

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
```

```
class Solution {
public:
    void flatten(TreeNode* root) {
    }
};
```

```
class Solution {
public:
    void flatten(TreeNode* root) {
        while (root) {
            TreeNode *p = root->left;
            if (p) {
                while (p->right) p = p->right;
                p->right = root->right;
                root->right = root->left;
                root->left = nullptr;
            }
            root = root->right;
        }
    }
};
```

115. Distinct Subsequences★★

Hard

Given a string **S** and a string **T**, count the number of distinct subsequences of **S** which equals **T**.

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, `"ACE"` is a subsequence of `"ABCDE"` while `"AEC"` is not).

Example 1:

Input: S = "rabbbit", T = "rabbit"

Output: 3

Explanation:

As shown below, there are 3 ways you can generate "rabbit" from S.

(The caret symbol ^ means the chosen letters)

rabbbit

^^^^ ^^

rabbbit

^^ ^^^^

rabbbit

^^^ ^^^

Example 2:

Input: S = "babgbag", T = "bag"

Output: 5

Explanation:

As shown below, there are 5 ways you can generate "bag" from S.

(The caret symbol ^ means the chosen letters)

babgbag

^^ ^

babgbag

^^ ^

```
babgbag
^    ^^
babgbag
  ^  ^^
babgbag
    ^^^
```

```
class Solution {
public:
    int numDistinct(string s, string t) {
      int m = s.length(), n = t.length();
      vector<vector<long long>> dp(2, vector<long long>(n + 1, 0));
      dp[0][0] = dp[1][0] = 1;
      int k = 0;
      for (int i = 1; i <= m; i++) {
          for (int j = 1; j <= n; j++) {
              dp[k][j] = dp[k^1][j] + (s[i-1] == t[j-1] ? dp[k^1][j-1] : 0);
          }
          k ^= 1;
      }
      return dp[k^1][n];
    }
};
```

116. Populating Next Right Pointers in Each Node

Medium

Given a binary tree

```
struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to `NULL`.

Initially, all next pointers are set to `NULL`.

Note:

- You may only use constant extra space.
- Recursive approach is fine, implicit stack space does not count as extra space for this problem.
- You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

Example:

```
      1
    /  \
   2    3
  / \  / \
4   5 6   7
```

After calling your function, the tree should look like:

```
     1 -> NULL
   /  \
  2 -> 3 -> NULL
 / \  / \
4->5->6->7 -> NULL
```

```
class Solution {
public:
    void connect(TreeLinkNode *root) {
        TreeLinkNode *dummy = new TreeLinkNode(-1), *p, *q;
        while (root) {
            dummy->next = nullptr; p = dummy; q = root;
            while (q){
                if (q->left) {
                    p->next = q->left;
                    p = p->next;
                }
                if (q->right) {
                    p->next = q->right;
                    p = p->next;
                }
                q = q->next;
            }
            root = dummy->next;
        }
    }
};
```

117. Populating Next Right Pointers in Each Node II★★

Medium

Given a binary tree

```
struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to `NULL`.

Initially, all next pointers are set to `NULL`.

Note:

- You may only use constant extra space.
- Recursive approach is fine, implicit stack space does not count as extra space for this problem.

Example:

Given the following binary tree,

```
    1
  /  \
 2    3
/\     \
4  5    7
```

After calling your function, the tree should look like:

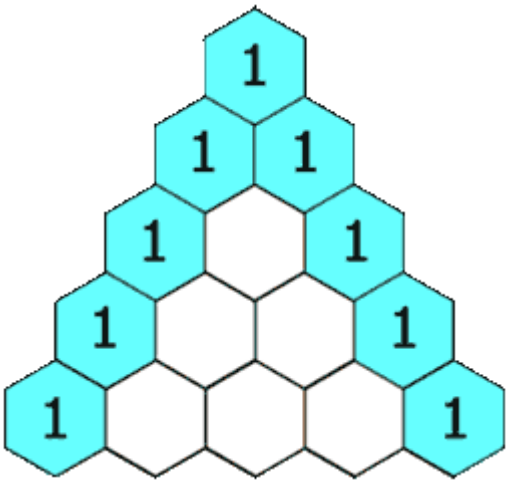
```
     1 -> NULL
   /  \
  2 -> 3 -> NULL
 /\     \
4-> 5 -> 7 -> NULL
```

```
class Solution {
public:
    void connect(TreeLinkNode *root) {
        TreeLinkNode *dummy = new TreeLinkNode(-1), *p, *q;
        while (root) {
            dummy->next = nullptr; p = dummy; q = root;
            while (q){
                if (q->left) {
                    p->next = q->left;
                    p = p->next;
                }
                if (q->right) {
                    p->next = q->right;
                    p = p->next;
                }
                q = q->next;
            }
            root = dummy->next;
        }
    }
};
```

118. Pascal's Triangle

Easy

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]

]

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {

    }
};
```

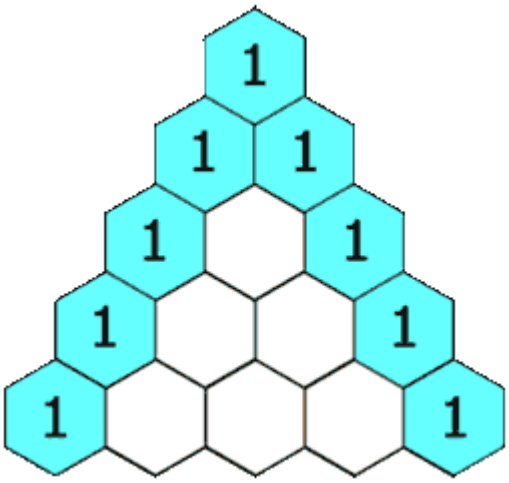
```cpp
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
       if (numRows <= 0) return vector<vector<int>> ();
       vector<vector<int>> res(numRows, vector<int> {1});
       for (int i = 1; i < numRows; ++i) {
          vector<int> &prev = res[i-1], &cur = res[i];
          int sz = prev.size();
          for (int i = 1; i < sz; i++) {
              cur.push_back(prev[i-1]+prev[i]);
          }
          cur.push_back(1);
       }
       return res;
    }
};
```

119. Pascal's Triangle II

Easy

Given a non-negative index k where $k \le 33$, return the k^{th} 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?

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {

    }
};
```

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

120. Triangle

Medium

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

```
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
```

The minimum path sum from top to bottom is `11` (i.e., **2** + **3** + **5** + **1** = 11).

Note:

Bonus point if you are able to do this using only *O*(*n*) extra space, where *n* is the total number of rows in the triangle.

```
class Solution {
public:
    int minimumTotal(vector<vector<int>>& triangle) {

    }
};
```

```cpp
class Solution {
public:
    int minimumTotal(vector<vector<int>>& triangle) {
        vector<int> &f = triangle.back();
        for (int i = triangle.size()-2; i >= 0; i--) {
            for (int j = 0; j <= i; j++) {
                f[j] = min(f[j], f[j+1])+triangle[i][j];
            }
        }
        return f[0];
    }
};
```

121. Best Time to Buy and Sell Stock

Easy

Say you have an array for which the i^{th} 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.

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {

    }
};
```

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int Min = INT_MAX, res = 0;
        for (auto &i : prices) {
            if (i < Min) Min = i;
            else res = max(res, i-Min);
        }
        return res;
    }
};
```

122. Best Time to Buy and Sell Stock II

Easy

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

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times).

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example 1:

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

Output: 7

Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Example 2:

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

Output: 4

Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are

engaging multiple transactions at the same time. You must sell before buying again.

Example 3:

Input: [7,6,4,3,1]

Output: 0

Explanation: In this case, no transaction is done, i.e. max profit = 0.

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {

    }
};
```

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int Buy = INT_MAX, res = 0;
        for (auto &Sell : prices) {
            if (Sell > Buy)   res += Sell-Buy;
            Buy = Sell;
        }
        return res;
    }
};
```

123. Best Time to Buy and Sell Stock III★★

Hard

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

Design an algorithm to find the maximum profit. You may complete at most *two* transactions.

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example 1:

Input: [3,3,5,0,0,3,1,4]

Output: 6

Explanation: Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.

Then buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3.

Example 2:

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

Output: 4

Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are

engaging multiple transactions at the same time. You must sell before buying again.

Example 3:

Input: [7,6,4,3,1]

Output: 0

Explanation: In this case, no transaction is done, i.e. max profit = 0.

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size(), res = 0;
        if (n < 2) return res;
        vector<int> f(n, 0), g(n, 0);
        for (int i = 1, buy = prices[0]; i < n; i++) {
            buy = min(buy, prices[i]);
            f[i] = max(f[i-1], prices[i]-buy);
        }
        for (int i = n-2, sell = prices[n-1]; i >= 0; i--) {
            sell = max(sell, prices[i]);
            g[i] = max(g[i], sell-prices[i]);
        }
        for (int i = 0; i < n; i++) res = max(res, f[i]+g[i]);
        return res;
    }
};
```

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
      int buy1 = INT_MIN, sale1 = 0, buy2 = INT_MIN, sale2 = 0;
      for (int &i : prices) {
          buy1 = max(buy1, -i);                    //left money after buy1
          sale1 = max(sale1, i + buy1);             //left money after sale1
          buy2 = max(buy2, sale1 - i);              //left money after buy2
          sale2 = max(sale2, i + buy2);             //left money after sale2
      }
      return sale2;
    }
};
```

124. Binary Tree Maximum Path Sum★★

Hard

Given a **non-empty** binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain **at least one node** and does not need to go through the root.

Example 1:

Input: [1,2,3]

```
       1
      / \
     2   3
```

Output: 6

Example 2:

Input: [-10,9,20,null,null,15,7]

```
   -10
   / \
  9  20
    /  \
   15   7
```

Output: 42

```
class Solution {
public:
    int maxPathSum(TreeNode *root) {

    }
};
```

```cpp
class Solution {
public:
    int maxPathSum(TreeNode *root) {
        int res = INT_MIN;
        maxToRoot(root, res);
        return res;
    }

private:
    int maxToRoot(TreeNode *root, int &res) {
        if (!root) return 0;
        int l = maxToRoot(root->left, res);
        int r = maxToRoot(root->right, res);
        if (l < 0) l = 0;
        if (r < 0) r = 0;
        res = max(res, l + r + root->val);
        return root->val += max(l, r);
    }
};
```

125. Valid Palindrome

Easy

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Note: For the purpose of this problem, we define empty string as valid palindrome.

Example 1:

Input: "A man, a plan, a canal: Panama"

Output: true

Example 2:

Input: "race a car"

Output: false

```
class Solution {
public:
    bool isPalindrome(string s) {

    }
};
```

```
class Solution {
public:
    bool isPalindrome(string s) {
        if (s.empty()) return true;
        auto p = &s[0], q =  &s[s.length()-1];
        while (p < q) {
            while (p < q && !isalnum(*p)) p++;
            while (p < q && !isalnum(*q)) q--;
            if (tolower(*p) != tolower(*q)) return false;
            p++; q--;
        }
        return true;
    }
};
```

126. Word Ladder II★★

Hard

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find all shortest transformation sequence(s) from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

Note:

- Return an empty list if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

Example 1:

Input:

beginWord = "hit",

endWord = "cog",

wordList = ["hot","dot","dog","lot","log","cog"]

Output:

[["hit","hot","dot","dog","cog"],

["hit","hot","lot","log","cog"]]

Example 2:

Input:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

Output: []

Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.

```
class Solution {
public:
    vector<vector<string> > findLadders(string beginWord, string
endWord, vector<string>& wordList) {
        dict = unordered_set<string>(wordList.begin(),
wordList.end());
        if (!dict.count(endWord)) return {};
        if (beginWord == endWord) return {path};
        words1.insert(beginWord);
        words2.insert(endWord);
        path.push_back(beginWord);

        if (findLaddersHelper(words1, words2))
            getPath(beginWord, endWord);
        return res;
    }
private:
    unordered_set<string> dict, words1, words2;
    unordered_map<string, vector<string> > nexts;
    vector<string> path;
    vector<vector<string>> res;
    bool words1IsBegin = false;

    bool findLaddersHelper(unordered_set<string> &words1,
unordered_set<string> &words2) {
        words1IsBegin = !words1IsBegin;
        if (words1.empty()) return false;
        if (words1.size() > words2.size())
            return findLaddersHelper(words2, words1);
        for (auto it : words1) dict.erase(it);
        for (auto it : words2) dict.erase(it);
        unordered_set<string> words3;
        bool reach = false;
        for (const auto &it : words1) {
            string word = it;
            for (auto &ch : word) {
                char tmp = ch;
                for (ch = 'a'; ch <= 'z'; ++(ch)) if (ch != tmp) {
                    if (words2.count(word)) {
                        reach = true;
                        words1IsBegin ? nexts[it].push_back(word)
                                      : nexts[word].push_back(it);
                    } else if (!reach && dict.count(word)) {
                        words3.insert(word);
```

```
                    words1IsBegin ? nexts[it].push_back(word)
                                  : nexts[word].push_back(it);
                }
            }
            ch = tmp;
        }
    }
    return reach || findLaddersHelper(words2, words3);
}

void getPath(const string &beginWord, const string &endWord) {
    if (beginWord == endWord)
        res.push_back(path);
    else for (auto it : nexts[beginWord]) {
        path.push_back(it);
        getPath(it, endWord);
        path.pop_back();
    }
}
};
```

127. Word Ladder

Medium

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

Example 1:

Input:

beginWord = "hit",

endWord = "cog",

wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

return its length 5.

Example 2:

Input:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

Output: 0

Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.

```cpp
class Solution {
public:
    int ladderLength(string a, string b, vector<string>& v) {
        unordered_set<string> s(v.begin(), v.end());
        if (!s.count(b)) return 0;
        s.erase(a);
        s.erase(b);
        unordered_set<string> nextWords{a}, prevWords{b};
        int cnt = 2, n = a.length();
        while (!nextWords.empty() && !prevWords.empty()) {
            if (nextWords.size() > prevWords.size())
                swap(nextWords, prevWords);
            unordered_set<string> temp;
            for (auto t : nextWords) {
                string word = t;
                for (int p = 0; p < n; p++) {
                    char letter = word[p];
                    for (int j = 0; j < 26; j++) {
                        word[p] = 'a' + j;
                        if (prevWords.count(word))
                            return cnt;
                        if (s.count(word)) {
                            temp.insert(word);
                            s.erase(word);
                        }
                    }
                    word[p] = letter;
                }
            }
            swap(nextWords, temp);
            cnt++;
        }
        return 0;
    }
};
```

128. Longest Consecutive Sequence★★

Hard

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

Your algorithm should run in O(*n*) complexity.

Example:

Input: [100, 4, 200, 1, 3, 2]

Output: 4

Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

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

    }
};
```

```cpp
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        unordered_set<int> s(nums.begin(), nums.end()), searched;
        int longest = 0;
        for (int i: nums) {
            if (searched.find(i) != searched.end()) continue;
            searched.insert(i);
            int j = i - 1, k = i + 1;
            while (s.find(j) != s.end()) searched.insert(j--);
            while (s.find(k) != s.end()) searched.insert(k++);
            longest = max(longest, k-j-1);
        }
        return longest;
    }
};
```

```cpp
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        unordered_set<int> s(nums.begin(), nums.end());
        int longest = 0;
        for (int i: nums) {
            int j = i, k = i + 1;
            while (s.find(j) != s.end()) s.erase(j--);
            while (s.find(k) != s.end()) s.erase(k++);
            longest = max(longest, k-j-1);
        }
        return longest;
    }
};
```

129. Sum Root to Leaf Numbers

Medium

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.

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.

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

```cpp
class Solution {
public:
    int sumNumbers(TreeNode* root) {
        return dfs(root, 0);
    }

private:
    int dfs(TreeNode *t, int sum) {
        if (!t) return 0;
        sum = sum*10+t->val;
        if (t->left == nullptr && t->right == nullptr) return sum;
        else return dfs(t->left, sum) + dfs(t->right, sum);
    }
};
```

130. Surrounded Regions

Medium

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.

```
class Solution {
public:
    void solve(vector<vector<char>>& board) {

    }
};
```

```cpp
class Solution {
public:
    void solve(vector<vector<char>>& board) {
        if (board.empty()) return;
        n = board.size(), m = board[0].size();
        for (int i = 0; i < n; i++) {
            dfs(i, 0, board);
            dfs(i, m-1, board);
        }

        for (int j = 1; j < m-1; j++) {
            dfs(0, j, board);
            dfs(n-1, j, board);
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                board[i][j] = board[i][j] == '+' ? 'O': 'X';
            }
        }
    }

private:
    int n, m;
    void dfs(int i, int j, vector<vector<char>>& board) {
        if (i < 0 || j < 0 || i >= n || j >= m) return;
        else if (board[i][j] != 'O') return;
        board[i][j] = '+';
        dfs(i+1, j, board);
        dfs(i-1, j, board);
        dfs(i, j+1, board);
        dfs(i, j-1, board);
    }
};
```

131. Palindrome Partitioning★★

Medium

Given a string *s*, partition *s* such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of *s*.

Example:

Input: "aab"

Output:

```
[
  ["aa","b"],
  ["a","a","b"]
]
```

```cpp
class Solution {
public:
    vector<vector<string>> partition(string s) {
        const int n = s.size();
        bool d[n][n];
        fill_n(&d[0][0], n*n, false);
        for (int i = n-1; i >= 0; i--) {
            for (int j = i; j < n; j++) {
                d[i][j] = (s[i] == s[j] && (j - i < 2 || d[i+1][j-1]));
            }
        }
        vector<vector<string>> sub_palindrome[n];
        for (int i = n-1; i >= 0; i--) {
            for (int j = i; j < n; j++) {
                if (d[i][j]) {
                    string p = s.substr(i, j-i+1);
                    if (j+1 < n) {
                        for (auto v : sub_palindrome[j+1]) {
                            v.insert(v.begin(), p);
                            sub_palindrome[i].push_back(v);
                        }
                    }
                    else sub_palindrome[i].push_back(vector<string>{p});
                }
            }
        }
        return sub_palindrome[0];
    }
};
```

132. Palindrome Partitioning II★★

Hard

Given a string *s*, partition *s* such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of *s*.

Example:

Input: "aab"

Output: 1

Explanation: The palindrome partitioning ["aa","b"] could be produced using 1 cut.

```
class Solution {
public:
    int minCut(string s) {

    }
};
```

```
class Solution {
public:
    int minCut(string s) {
        if (s.empty()) return 0;
        int n = s.size();
        vector<vector<bool>> p(n, vector<bool>(n,false));
        vector<int> f(n);
        for (int i = n-1; i >= 0; i--) {
            f[i] = n-i-1;
            for (int j = i; j < n; j++) {
                if (s[i] == s[j] && (j - i < 2 || p[i+1][j-1])) {
                    p[i][j] = true;
                    if (j == n-1) f[i]=0;
                    else f[i] = min(f[i], f[j+1]+1);
                }
            }
        }
        return f[0];
    }
};
```

133. Clone Graph★★

Medium

Given the head of a graph, return a deep copy (clone) of the graph. Each node in the graph contains a `label` (`int`) and a list (`List[UndirectedGraphNode]`) of its `neighbors`. There is an edge between the given node and each of the nodes in its neighbors.

OJ's undirected graph serialization (so you can understand error output):
Nodes are labeled uniquely.

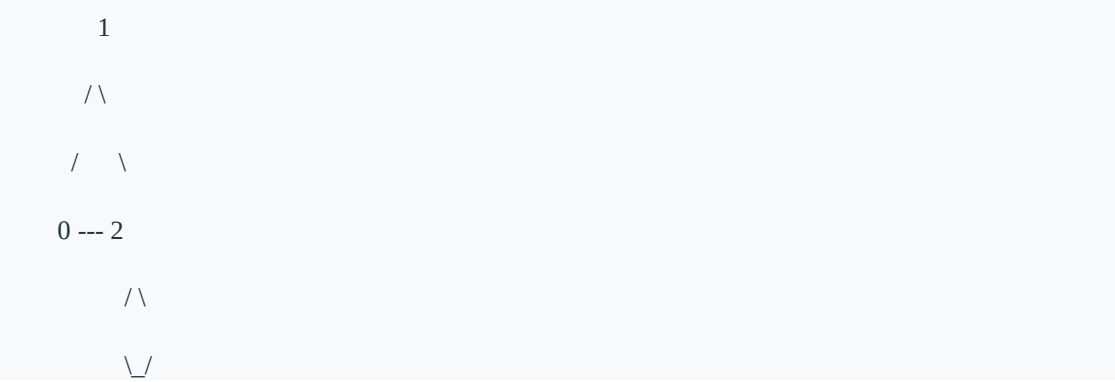
We use `#` as a separator for each node, and `,` as a separator for node label and each neighbor of the node.

As an example, consider the serialized graph `{0,1,2#1,2#2,2}`.

The graph has a total of three nodes, and therefore contains three parts as separated by `#`.

1. First node is labeled as `0`. Connect node `0` to both nodes `1` and `2`.
2. Second node is labeled as `1`. Connect node `1` to node `2`.
3. Third node is labeled as `2`. Connect node `2` to node `2` (itself), thus forming a self-cycle.

Visually, the graph looks like the following:



Note: The information about the tree serialization is only meant so that you can understand error output if you get a wrong answer. You don't need to understand the serialization to solve the problem.

```
/*
class Node {
public:
    int val;
    vector<Node*> neighbors;
    Node() {}
    Node(int _val, vector<Node*> _neighbors) {
        val = _val;
        neighbors = _neighbors;
    }
};
*/

//////////////////////////BFS/////////////////////////////////////
class Solution {
public:
    Node* cloneGraph(Node* node) {
        if (!node)   return nullptr;
        Node *copy = new Node(node->val, {});
        copies[node] = copy;
        queue<Node*> q;
        q.push(node);
        while (!q.empty()) {
            Node *cur = q.front();
            q.pop();
            for (Node *neighbor : cur->neighbors) {
                if (copies.find(neighbor) == copies.end()) {
                    copies[neighbor] = new Node(neighbor->val, {});
                    q.push(neighbor);
                }
                copies[cur]->neighbors.push_back(copies[neighbor]);
            }
        }
        return copy;
    }
private:
    unordered_map<Node*, Node*> copies;
};
```

```
//////////////////////DFS////////////////////////////////
class Solution {
public:
    Node* cloneGraph(Node* node) {
        if (!node)   return nullptr;
        if (copies.find(node) == copies.end()) {
            copies[node] = new Node(node->val, {});
            for (Node *neighbor : node->neighbors) {
                copies[node]->neighbors.push_back(cloneGraph(neighbor));
            }
        }
        return copies[node];
    }
private:
    unordered_map<Node*, Node*> copies;
};
```

134. Gas Station

Medium

There are *N* gas stations along a circular route, where the amount of gas at station *i* is `gas[i]`.

You have a car with an unlimited gas tank and it costs `cost[i]` of gas to travel from station *i* to its next station (*i*+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return -1.

Note:

- If there exists a solution, it is guaranteed to be unique.
- Both input arrays are non-empty and have the same length.
- Each element in the input arrays is a non-negative integer.

Example 1:

Input:

gas = [1,2,3,4,5]

cost = [3,4,5,1,2]

Output: 3

Example 2:

Input:

gas = [2,3,4]

cost = [3,4,3]

Output: -1

```cpp
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        int res = 0, cur = 0, Min_gas = INT_MAX, N = gas.size();
        for (int i = 0; i < N; i++) {
            cur += gas[i]-cost[i];
            if (cur < Min_gas) {
                Min_gas = cur;
                res = (i+1)%N;
            }
        }
        if (cur < 0) return -1;
        else return res;
    }
};
```

135. Candy

Hard

There are *N* children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

Example 1:

Input: [1,0,2]

Output: 5

Explanation: You can allocate to the first, second and third child with 2, 1, 2 candies respectively.

Example 2:

Input: [1,2,2]

Output: 4

Explanation: You can allocate to the first, second and third child with 1, 2, 1 candies respectively.

The third child gets 1 candy because it satisfies the above two conditions.

```
class Solution {
public:
    int candy(vector<int>& ratings) {

    }
};
```

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

136. Single Number

Easy

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

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

137. Single Number II★★

Medium

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: 9

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

    }
};
```

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        const int w = sizeof(int)*8;
        int cnt[w] = {0};
        for (auto i : nums) {
            for (int j = 0; j < w; j++) {
                cnt[j] += (i >> j) & 1;
                cnt[j] %= 3;
            }
        }
        int res = 0;
        for (int i = 0; i < w; i++) {
            res += (cnt[i] << i);
        }
        return res;
    }
};
```

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int a = 0, b = 0;
        for (auto &x : nums) {
            a = a^x & ~b;
            b = b^x & ~a;
        }
        return a; //唯一一个出现一次的数字
        return b; //唯一一个出现两次的数字
    }
};
```

138. Copy List with Random Pointer

Medium

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

```
class Solution {
public:
    RandomListNode *copyRandomList(RandomListNode *head) {

    }
};
```

```
class Solution {
public:
    RandomListNode *copyRandomList(RandomListNode *head) {
        if(head == nullptr) return nullptr;
        RandomListNode *p = head, *q, *head2;
        while (p) {
            q = new RandomListNode(p->label);
            q->next = p->next;
            p->next = q;
            p = q->next;
        }
        p = head;
        while(p) {
            if (p->random) p->next->random = p->random->next;
            p = p->next->next;
        }

        p = head, head2 = head->next;
        while (p) {
            q = p->next;
            p->next = q->next;
            if (q->next) q->next = q->next->next;
            p = p->next;
        }
        return head2;
    }
};
```

139. Word Break

Medium

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, determine if *s* can be segmented into a space-separated sequence of one or more dictionary words.

Note:

- The same word in the dictionary may be reused multiple times in the segmentation.
- You may assume the dictionary does not contain duplicate words.

Example 1:

Input: s = "leetcode", wordDict = ["leet", "code"]

Output: true

Explanation: Return true because "leetcode" can be segmented as "leet code".

Example 2:

Input: s = "applepenapple", wordDict = ["apple", "pen"]

Output: true

Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".

Note that you are allowed to reuse a dictionary word.

Example 3:

Input: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]

Output: false

```
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {


    }
};
```

```cpp
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {
        size_t n = s.size(), max_len = 0, min_len = INT_MAX;
        unordered_set<string> dict(wordDict.begin(), wordDict.end());
        for(auto &i : wordDict) {
            max_len = max(max_len, i.length());
            min_len = min(min_len, i.length());
        }
        vector<bool> f(n+1, false);
        vector<vector<int>> v(n);
        f[n] = true;
        for (int i = n-1; i >= 0; --i){
            for (int j = i+min_len; j <= n && j-i <= max_len; j++) {
                string word = s.substr(i, j-i);
                if (f[j] && dict.count(word)) {
                    f[i] = true;
                    break;
                }
            }
        }
        return f[0];
    }
};
```

140. Word Break II★★

Hard

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, add spaces in *s* to construct a sentence where each word is a valid dictionary word. Return all such possible sentences.

Note:

- The same word in the dictionary may be reused multiple times in the segmentation.
- You may assume the dictionary does not contain duplicate words.

Example 1:

Input:

s = "catsanddog"

wordDict = ["cat", "cats", "and", "sand", "dog"]

Output:

[

  "cats and dog",

  "cat sand dog"

]

Example 2:

Input:

s = "pineapplepenapple"

wordDict = ["apple", "pen", "applepen", "pine", "pineapple"]

Output:

[

  "pine apple pen apple",

  "pineapple pen apple",

  "pine applepen apple"

]

Explanation: Note that you are allowed to reuse a dictionary word.

Example 3:

Input:

s = "catsandog"

wordDict = ["cats", "dog", "sand", "and", "cat"]

Output:

[]

```
class Solution {
public:
    vector<string> wordBreak(string s, vector<string>& wordDict) {

    }
};
```

```cpp
class Solution {
public:
    vector<string> wordBreak(string s, vector<string>& wordDict) {
        size_t n = s.size(), max_len = 0, min_len = INT_MAX;
        unordered_set<string> dict(wordDict.begin(), wordDict.end());
        for(auto &i : wordDict) {
            max_len = max(max_len, i.length());
            min_len = min(min_len, i.length());
        }
        vector<bool> f(n+1, false);
        vector<vector<int>> v(n);
        f[n] = true;
        for (int i = n-1; i >= 0; --i){
            for (int j = i+min_len; j <= n && j-i <= max_len; j++) {
                string word = s.substr(i, j-i);
                if (f[j] && dict.count(word)) {
                    f[i] = true;
                    v[i].push_back(j);
                }
            }
        }
        dfs(0, n, s, v);
        return res;
    }
private:
    vector<string> res, path;

    void dfs(int i, int n, string &s, vector<vector<int>> &v) {
        if (i == n) {
            string s;
            for (auto &i : path) s += (s == "" ? "" : " ") + i;
            res.push_back(s);
        }
        else {
            for (auto j : v[i]) {
                path.emplace_back(s.substr(i, j-i));
                dfs(j, n, s, v);
                path.pop_back();
            }
        }
    }
};
```

141. Linked List Cycle

Easy

Given a linked list, determine if it has a cycle in it.

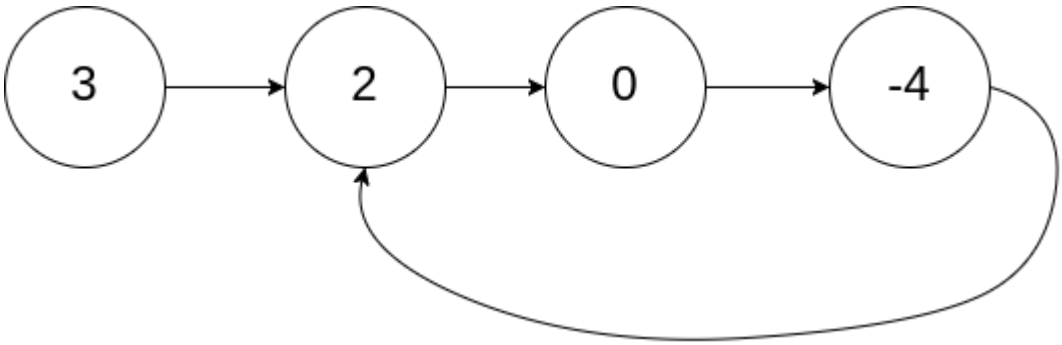
To represent a cycle in the given linked list, we use an integer `pos` which represents the position (0-indexed) in the linked list where tail connects to. If `pos` is `-1`, then there is no cycle in the linked list.

Example 1:

Input: head = [3,2,0,-4], pos = 1

Output: true

Explanation: There is a cycle in the linked list, where tail connects to the second node.

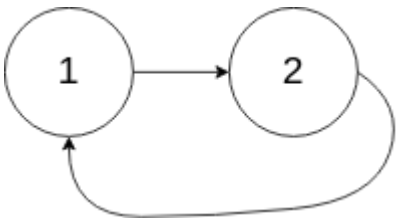


Example 2:

Input: head = [1,2], pos = 0

Output: true

Explanation: There is a cycle in the linked list, where tail connects to the first node.



Example 3:

Input: head = [1], pos = -1

Output: false

Explanation: There is no cycle in the linked list.

Follow up:

Can you solve it using *O(1)* (i.e. constant) memory?

```
class Solution {
public:
    bool hasCycle(ListNode *head) {

    }
};
```

```cpp
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) return true;
        }
        return false;
    }
};
```

142. Linked List Cycle II

Medium

Given a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

To represent a cycle in the given linked list, we use an integer `pos` which represents the position (0-indexed) in the linked list where tail connects to. If `pos` is `-1`, then there is no cycle in the linked list.

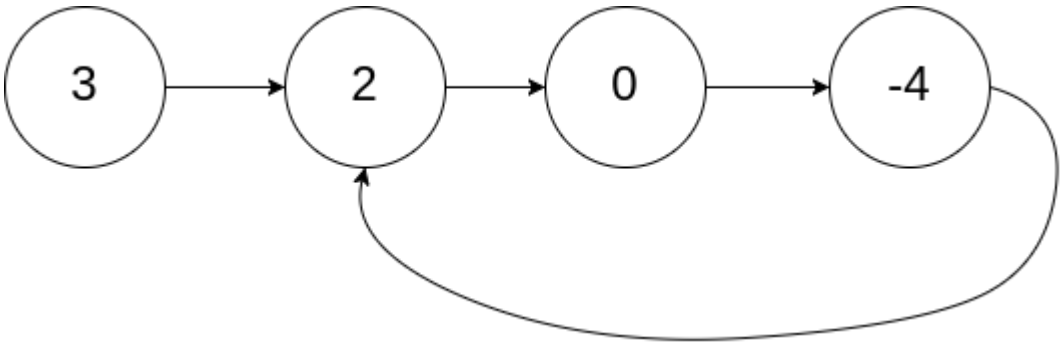
Note: Do not modify the linked list.

Example 1:

Input: head = [3,2,0,-4], pos = 1

Output: tail connects to node index 1

Explanation: There is a cycle in the linked list, where tail connects to the second node.

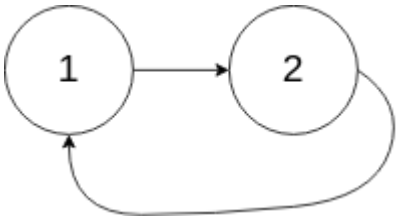


Example 2:

Input: head = [1,2], pos = 0

Output: tail connects to node index 0

Explanation: There is a cycle in the linked list, where tail connects to the first node.



Example 3:

Input: head = [1], pos = -1

Output: no cycle

Explanation: There is no cycle in the linked list.

Follow up:
Can you solve it without using extra space?

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {

    }
};
```

```cpp
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) {
                ListNode *slow2 = head;
                while (slow2 != slow) {
                    slow2 = slow2->next;
                    slow = slow->next;
                }
                return slow;
            }
        }
        return nullptr;
    }
};
```

143. Reorder List

Medium

Given a singly linked list L: $L_0 \rightarrow L_1 \rightarrow \ldots \rightarrow L_{n-1} \rightarrow L_n$,
reorder it to: $L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \ldots$

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

Example 1:

```
Given 1->2->3->4, reorder it to 1->4->2->3.
```

Example 2:

```
Given 1->2->3->4->5, reorder it to 1->5->2->4->3.
```

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    void reorderList(ListNode *head) {

    }
};
```

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    void reorderList(ListNode *head) {
        if (head == nullptr || head->next == nullptr) return;
        ListNode *p = head, *q = p, *head2, *s;
        while (q && q->next) {
            q = q->next->next;
            p = p->next;
        }
        head2 = p;  p = p->next;  head2->next = nullptr;
        while (p) {
            s = p->next;
            p->next = head2->next;
            head2->next= p;
            p = s;
        }
        p = head;  q = head2->next; head2->next = nullptr;
        while (p && q) {
            s = q->next;
            q->next = p->next;
            p->next = q;
            p = p->next->next;
            q = s;
        }
    }
};
```

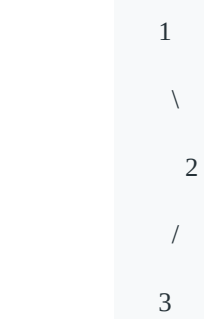
144. Binary Tree Preorder Traversal

Medium

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

Example:

Input: [1,null,2,3]



Output: [1,2,3]

Follow up: Recursive solution is trivial, could you do it iteratively?

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;

/**
 * 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) {

    }
};
```

```cpp
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> res;
        stack<TreeNode*> My_Stack;
        My_Stack.push(root);
        while (!My_Stack.empty()) {
            TreeNode *t = My_Stack.top();
            My_Stack.pop();
            if (t == nullptr) continue;
            res.push_back(t->val);
            My_Stack.push(t->right);
            My_Stack.push(t->left);
        }
        return res;
    }
};
```

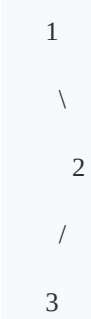
145. Binary Tree Postorder Traversal★★

Hard

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

Example:

Input: [1,null,2,3]



Output: [3,2,1]

Follow up: Recursive solution is trivial, could you do it iteratively?

```
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {

    }
};
```

```cpp
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {
        vector<int> res;
        stack<TreeNode*> My_Stack;
        My_Stack.push(root);
        while (!My_Stack.empty()) {
            TreeNode *t = My_Stack.top();
            My_Stack.pop();
            if (t == nullptr) continue;
            res.push_back(t->val);
            My_Stack.push(t->left);
            My_Stack.push(t->right);
        }
        reverse(res.begin(), res.end());
        return res;
    }
};
```

```cpp
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {
        vector<int> res;
        stack<TreeNode*> stk;
        TreeNode *p = root, *last = nullptr;
        while (p || !stk.empty()) {
            while (p) {
                stk.push(p);
                p = p->left;
            }
            TreeNode *top = stk.top();
            if (top->right && top->right != last) {
                p = top->right;
            } else {
                res.push_back(top->val);
                last = top;
                stk.pop();
            }
        }
        return res;
    }
};
```

146. LRU Cache★★

Hard

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.

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
```

```
class LRUCache {
public:
    LRUCache(int capacity) {}
    int get(int key) {}
    void put(int key, int value) {}
};
```

```
class LRUCache {
public:
    LRUCache(int capacity):capacity(capacity){}

    int get(int key) {
        if (My_map.find(key) == My_map.end()) return -1;
        CacheList.splice(CacheList.begin(), CacheList, My_map[key]);
        return CacheList.begin()->value;
    }

    void put(int key, int value) {
        if (My_map.find(key) == My_map.end()){
            CacheList.emplace_front(key, value);
        }
        else {
            My_map[key]->value = value;
            CacheList.splice(CacheList.begin(),CacheList, My_map[key]);
        }
        My_map[key] = CacheList.begin();
        if (My_map.size() > capacity) {
            My_map.erase(CacheList.back().key);
            CacheList.pop_back();
        }
    }

private:
    struct CacheNode{
        int key, value;
        CacheNode(int k, int v):key(k), value(v){}
    };
    int capacity;
    unordered_map<int, list<CacheNode>::iterator> My_map;
    list<CacheNode> CacheList;
};
```

147. Insertion Sort List

Medium

Sort a linked list using insertion sort.

A graphical example of insertion sort. The partial sorted list (black) initially contains only the first element in the list.

With each iteration one element (red) is removed from the input data and inserted in-place into the sorted list

Algorithm of Insertion Sort:

1. Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.
2. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there.
3. It repeats until no input elements remain.

Example 1:

Input: 4->2->1->3

Output: 1->2->3->4

Example 2:

Input: -1->5->3->4->0

Output: -1->0->3->4->5

```
class Solution {
public:
    ListNode* insertionSortList(ListNode* head) {

    }
};
```

```cpp
class Solution {
public:
    ListNode* insertionSortList(ListNode* head) {
        ListNode *dummy = new ListNode(-1);
        ListNode *cur = head, *next;
        while (cur) {
            next = cur->next;
            ListNode *p = dummy;
            while (p->next != nullptr && p->next->val < cur->val) p = p->next;
            cur->next = p->next;
            p->next = cur;
            cur = next;
        }
        return dummy->next;
    }
};
```

148. Sort List★★

Medium

Sort a linked list in *O*(*n* log *n*) time using constant space complexity.

Example 1:

Input: 4->2->1->3

Output: 1->2->3->4

Example 2:

Input: -1->5->3->4->0

Output: -1->0->3->4->5

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* sortList(ListNode *head) {

    }
};
```

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* sortList(ListNode *head) {
        if (head == nullptr || head->next == nullptr) return head;
        ListNode *mid = find_mid(head), *head2 = mid->next;
        mid->next = nullptr;
        return merge_two_lists(sortList(head), sortList(head2));
    }

private:
    ListNode *merge_two_lists(ListNode *l1, ListNode *l2) {
        ListNode *dummy = new ListNode(-1), *p = dummy;
        while (l1 && l2) {
            if (l1->val > l2->val) swap(l1, l2);
            p->next = l1;
            p = l1;
            l1 = l1->next;
        }
        p->next = l1 == nullptr ? l2: l1;
        return dummy->next;
    }

    ListNode *find_mid(ListNode *head) {
        ListNode *fast = head, *slow = head;
        while (fast && fast->next && fast->next->next) {
            fast = fast->next->next;
            slow = slow->next;
        }
        return slow;
    }
};
```

149. Max Points on a Line★★

Hard

Given *n* points on a 2D plane, find the maximum number of points that lie on the same straight line.

Example 1:

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

Output: 3

Explanation:

```
^
|
|        o
|     o
|  o
+------------>
0  1  2  3  4
```

Example 2:

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

Output: 4

Explanation:

```
^
|
|   o
|    o          o
|        o
| o          o
+------------------>
0  1  2  3  4  5  6
```

```cpp
class Solution {
public:
    int maxPoints(vector<vector<int>>& points) {
        int res = 0;
        for (int i = 0; i < points.size(); ++i) {
            map<pair<int, int>, int> m;
            int duplicate = 1;
            for (int j = i+1; j < points.size(); ++j) {
                if (points[i] == points[j]) {
                    ++duplicate; continue;
                }
                int dx = points[j][0] - points[i][0];
                int dy = points[j][1] - points[i][1];
                int d = gcd(dx, dy);
                ++m[{dx / d, dy / d}];
            }
            res = max(res, duplicate);
            for (auto it = m.begin(); it != m.end(); ++it) {
                res = max(res, it->second + duplicate);
            }
        }
        return res;
    }

private:
    int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b);
    }
};
```

150. Evaluate Reverse Polish Notation

Medium

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are `+`, `-`, `*`, `/`. Each operand may be an integer or another expression.

Note:

- Division between two integers should truncate toward zero.
- The given RPN expression is always valid. That means the expression would always evaluate to a result and there won't be any divide by zero operation.

Example 1:

Input: ["2", "1", "+", "3", "*"]

Output: 9

Explanation: ((2 + 1) * 3) = 9

Example 2:

Input: ["4", "13", "5", "/", "+"]

Output: 6

Explanation: (4 + (13 / 5)) = 6

Example 3:

Input: ["10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+", "5", "+"]

Output: 22

Explanation:

((10 * (6 / ((9 + 3) * -11))) + 17) + 5

= ((10 * (6 / (12 * -11))) + 17) + 5

= ((10 * (6 / -132)) + 17) + 5

= ((10 * 0) + 17) + 5

= (0 + 17) + 5

= 17 + 5

= 22

```cpp
class Solution {
public:
    int evalRPN(vector<string>& tokens) {
        stack<int> My_Stack;
        for(auto &s : tokens) {
            if (!is_operator(s))   My_Stack.push(stoi(s));
            else{
                int b = My_Stack.top(); My_Stack.pop();
                int a = My_Stack.top(); My_Stack.pop();
                switch(s[0]) {
                    case '+' : a += b; break;
                    case '-' : a -= b; break;
                    case '*' : a *= b; break;
                    case '/' : a /= b; break;
                }
                My_Stack.push(a);
            }
        }
        return My_Stack.top();
    }
private:
    bool is_operator(string s){
        return s.length() == 1 && string("+-*/").find(s) != string::npos;
    }
};
```

151. Reverse Words in a String

Medium

Given an input string, reverse the string word by word.

Example:

Input: "the sky is blue",

Output: "blue is sky the".

Note:

- A word is defined as a sequence of non-space characters.
- Input string may contain leading or trailing spaces. However, your reversed string should not contain leading or trailing spaces.
- You need to reduce multiple spaces between two words to a single space in the reversed string.

Follow up: For C programmers, try to solve it *in-place* in *O*(1) space.

```
class Solution {
public:
    void reverseWords(string &s) {

    }
};
```

```cpp
class Solution {
public:
    void reverseWords(string &s) {
        while (!s.empty() && s[0] == ' ') s.erase(0, 1);
        while (!s.empty() && s.back() == ' ') s.erase(s.size()-1);
        if (s.empty()) return;
        auto p = s.begin(), q = p;
        while (1) {
            while (q != s.end() && *q != ' ') q++;
            reverse(p, q);
            if (q == s.end()) break;
            p = ++q;
            while (p != s.end() && *p == ' ') p = s.erase(p);
        }
        reverse(s.begin(), s.end());
    }
};
```

152. Maximum Product Subarray

Medium

Given an integer array `nums`, find the contiguous subarray within an array (containing at least one number) which has the largest product.

Example 1:

Input: [2,3,-2,4]

Output: 6

Explanation: [2,3] has the largest product 6.

Example 2:

Input: [-2,0,-1]

Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

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

    }
};
```

```cpp
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int res = nums[0];
        int MAX = nums[0], MIN = nums[0];
        for (auto i = nums.begin()+1; i != nums.end(); i++) {
            int a = (*i)*MAX, b = (*i)*MIN;
            if (a < b) swap(a, b);
            MAX = max(a, *i);
            MIN = min(b, *i);
            res = max(res, MAX);
        }
        return res;
    }
};
```

153. Find Minimum in Rotated Sorted Array

Medium

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

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

    }
};
```

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

154. Find Minimum in Rotated Sorted Array II★★

Hard

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.

The array may contain duplicates.

Example 1:

Input: [1,3,5]

Output: 1

Example 2:

Input: [2,2,2,0,1]

Output: 0

Note:

- This is a follow up problem to Find Minimum in Rotated Sorted Array.
- Would allow duplicates affect the run-time complexity? How and why?

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

    }
};
```

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

155. Min Stack

Easy

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

Example:

```
MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin();      --> Returns -3.

minStack.pop();

minStack.top();            --> Returns 0.

minStack.getMin();      --> Returns -2.
```

```
class MinStack {
public:
    MinStack() {}
    void push(int x) {
        if (Min_S.empty()) Min_S.push(x);
        else Min_S.push(min(Min_S.top(), x));
        S.push(x);
    }

    void pop() {
        S.pop();
        Min_S.pop();
    }

    int top() {
        return S.top();}

    int getMin() {
        return Min_S.top();
    }

    private:
      stack<int> S, Min_S;
};
```

156.Binary Tree Upside Down

Given a binary tree where all the right nodes are either leaf nodes with a sibling (a left node that shares the same parent node) or empty, flip it upside down and turn it into a tree where the original right nodes turned into left leaf nodes. Return the new root.

Example:

```
Input: [1,2,3,4,5]

    1

   / \

  2   3

 / \

4   5

Output: return the root of the binary tree [4,5,2,#,#,3,1]

   4

  / \

 5   2

    / \

   3   1
```

```
/**
 * 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:
    TreeNode* upsideDownBinaryTree(TreeNode* root) {
        if (!root || !root->left) return root;
        TreeNode *l = root->left, *r = root->right;
        TreeNode *res = upsideDownBinaryTree(l);
        l->left = r;
        l->right = root;
        root->left = root->right = nullptr;
        return res;
    }
};
```

157.Read N Characters Given Read4

Given a file and assume that you can only read the file using a given method `read4`, implement a method to read *n* characters.

Method read4:

The API `read4` reads 4 consecutive characters from the file, then writes those characters into the buffer array `buf`.

The return value is the number of actual characters read.

Note that `read4()` has its own file pointer, much like `FILE *fp` in C.

Definition of read4:

```
    Parameter:  char[] buf

    Returns:    int

Note: buf[] is destination not source, the results from read4 will be copied to buf[]
```

Below is a high level example of how `read4` works:

```
File file("abcdefghijk"); // File is "abcdefghijk", initially file pointer (fp) points to 'a'

char[] buf = new char[4]; // Create buffer with enough space to store characters

read4(buf); // read4 returns 4. Now buf = "abcd", fp points to 'e'

read4(buf); // read4 returns 4. Now buf = "efgh", fp points to 'i'

read4(buf); // read4 returns 3. Now buf = "ijk", fp points to end of file
```

Method read:

By using the `read4` method, implement the method `read` that reads *n* characters from the file and store it in the buffer array `buf`. Consider that you **cannot** manipulate the file directly.

The return value is the number of actual characters read.

Definition of read:

```
    Parameters:  char[] buf, int n

    Returns:     int

Note: buf[] is destination not source, you will need to write the results to
buf[]
```

Example 1:

```
Input: file = "abc", n = 4

Output: 3

Explanation: After calling your read method, buf should contain "abc". We
read a total of 3 characters from the file, so return 3. Note that "abc" is
the file's content, not buf. buf is the destination buffer that you will have
to write the results to.
```

Example 2:

```
Input: file = "abcde", n = 5

Output: 5

Explanation: After calling your read method, buf should contain "abcde". We
read a total of 5 characters from the file, so return 5.
```

Example 3:

```
Input: file = "abcdABCD1234", n = 12

Output: 12

Explanation: After calling your read method, buf should contain
"abcdABCD1234". We read a total of 12 characters from the file, so return 12.
```

Example 4:

```
Input: file = "leetcode", n = 5

Output: 5

Explanation: After calling your read method, buf should contain "leetc". We
read a total of 5 characters from the file, so return 5.
```

Note:

1. Consider that you **cannot** manipulate the file directly, the file is only accesible for `read4` but **not** for `read`.
2. The `read` function will only be called once for each test case.
3. You may assume the destination buffer array, `buf`, is guaranteed to have enough space for storing *n* characters.

```
// Forward declaration of the read4 API.
int read4(char *buf);

class Solution {
public:
    int read(char *buf, int n) {
        int cnt = 0;
        for (int i = 0; i <= n/4; ++i) {
            int cur = read4(cnt + buf);
            if (cur == 0) break;
            cnt += cur;
        }
        return min(cnt, n);
    }
};
```

158.Read N Characters Given Read4 II - Call multiple times★★

Given a file and assume that you can only read the file using a given method `read4`, implement a method `read` to read *n* characters. **Your method `read` may be called multiple times.**

Method read4:

The API `read4` reads 4 consecutive characters from the file, then writes those characters into the buffer array `buf`.

The return value is the number of actual characters read.

Note that `read4()` has its own file pointer, much like `FILE *fp` in C.

Definition of read4:

```
    Parameter:  char[] buf

    Returns:    int

Note: buf[] is destination not source, the results from read4 will be
copied to buf[]
```

Below is a high level example of how `read4` works:

```
File file("abcdefghijk"); // File is "abcdefghijk", initially file pointer
(fp) points to 'a'

char[] buf = new char[4]; // Create buffer with enough space to store
characters

read4(buf); // read4 returns 4. Now buf = "abcd", fp points to 'e'

read4(buf); // read4 returns 4. Now buf = "efgh", fp points to 'i'

read4(buf); // read4 returns 3. Now buf = "ijk", fp points to end of file
```

Method read:

By using the `read4` method, implement the method `read` that reads *n* characters from the file and store it in the buffer array `buf`. Consider that you **cannot** manipulate the file directly.

The return value is the number of actual characters read.

Definition of read:

```
    Parameters:  char[] buf, int n
```

```
    Returns:     int

Note: buf[] is destination not source, you will need to write the results
to buf[]
```

Example 1:

```
File file("abc");

Solution sol;

// Assume buf is allocated and guaranteed to have enough space for storing
all characters from the file.

sol.read(buf, 1); // After calling your read method, buf should contain
"a". We read a total of 1 character from the file, so return 1.

sol.read(buf, 2); // Now buf should contain "bc". We read a total of 2
characters from the file, so return 2.

sol.read(buf, 1); // We have reached the end of file, no more characters
can be read. So return 0.
```

Example 2:

```
File file("abc");

Solution sol;

sol.read(buf, 4); // After calling your read method, buf should contain
"abc". We read a total of 3 characters from the file, so return 3.

sol.read(buf, 1); // We have reached the end of file, no more characters
can be read. So return 0.
```

Note:

1. Consider that you **cannot** manipulate the file directly, the file is only accesible for `read4` but **not** for `read`.
2. The `read` function may be called **multiple times**.
3. Please remember to **RESET** your class variables declared in Solution, as static/class variables are **persisted across multiple test cases**. Please see here for more details.
4. You may assume the destination buffer array, `buf`, is guaranteed to have enough space for storing *n* characters.
5. It is guaranteed that in a given test case the same buffer `buf` is called by `read`.

```cpp
// Forward declaration of the read4 API.
int read4(char *buf);


class Solution {
public:
    int read(char *buf, int n) {
        for (int i = 0; i < n; i++) {
            if (readPos == writePos) {
                writePos = read4(buff);
                readPos = 0;
                if (writePos == 0) return i;
            }
            buf[i] = buff[readPos++];
        }
        return n;
    }

private:
    int readPos = 0, writePos = 0;
    char buff[4];
};
```

159.Longest Substring with At Most Two Distinct Characters

Given a string *s* , find the length of the longest substring *t* that contains **at most** 2 distinct characters.

Example 1:

```
Input: "eceba"

Output: 3

Explanation: t is "ece" which its length is 3.
```

Example 2:

```
Input: "ccaabbb"

Output: 5

Explanation: t is "aabbb" which its length is 5.
```

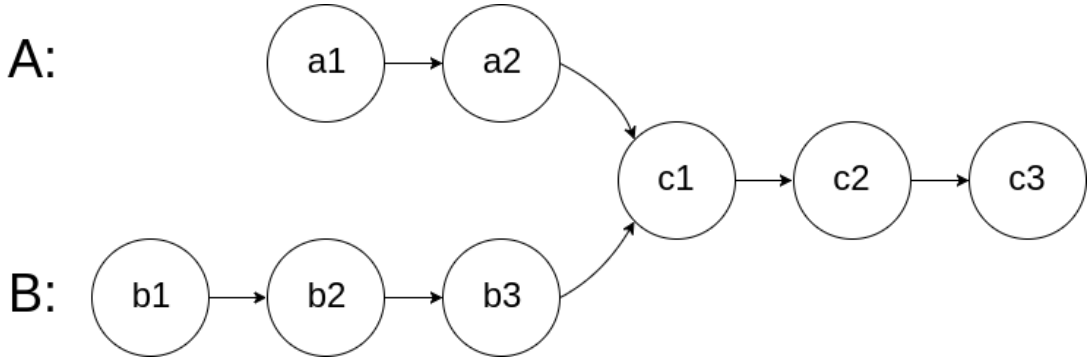
```
class Solution {
public:
    int lengthOfLongestSubstringTwoDistinct(string s) {
        int res = 0, left = 0, n = s.size();
        unordered_map<char, int> m;
        for (int i = 0; i < n; ++i) {
            ++m[s[i]];
            while (m.size() > 2) {
                if (--m[s[left]] == 0) m.erase(s[left]);
                ++left;
            }
            res = max(res, i - left + 1);
        }
        return res;
    }
};
```

160. Intersection of Two Linked Lists

Easy

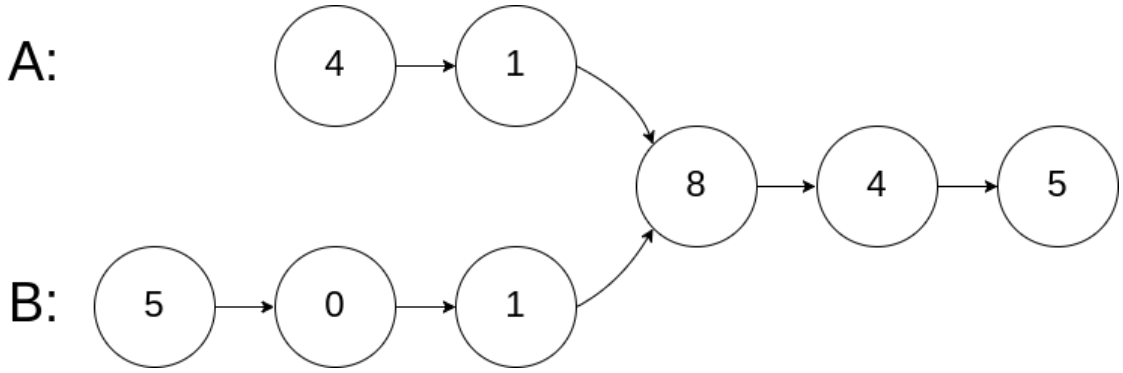
Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:



begin to intersect at node c1.

Example 1:

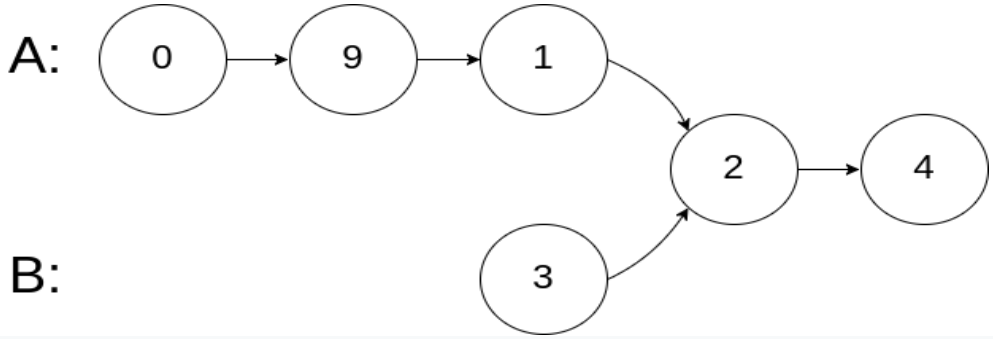


Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3

Output: Reference of the node with value = 8

Input Explanation: The intersected node's value is 8 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,0,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes before the intersected node in B.

Example 2:

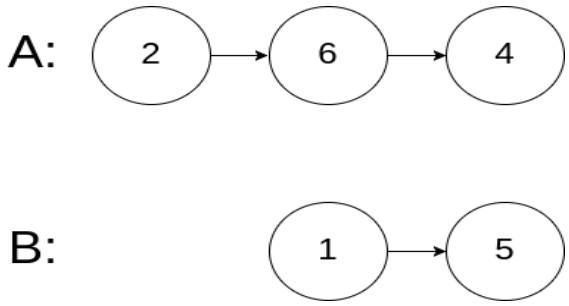


Input: intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1

Output: Reference of the node with value = 2

Input Explanation: The intersected node's value is 2 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [0,9,1,2,4]. From the head of B, it reads as [3,2,4]. There are 3 nodes before the intersected node in A; There are 1 node before the intersected node in B.

Example 3:



Input: intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2

Output: null

Input Explanation: From the head of A, it reads as [2,6,4]. From the head of B, it reads as [1,5]. Since the two lists do not intersect, intersectVal must be 0, while skipA and skipB can be arbitrary values.

Explanation: The two lists do not intersect, so return null.

Notes:

- If the two linked lists have no intersection at all, return `null`.
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        int len1 = 0, len2 = 0;
        ListNode *p = headA, *q = headB;
        while (p) {p = p->next; len1++;}
        while (q) {q = q->next; len2++;}
        if (len1 < len2) {
            swap(len1, len2);
            swap(headA, headB);
        }

        p = headA; q = headB;
        int k = len1-len2;
        while (k--) p = p->next;
        while (p) {
            if (p == q) return p;
            p = p->next;
            q = q->next;
        }
        return nullptr;
    }
}
```

161. One Edit Distance

Given two strings ***s*** and ***t***, determine if they are both one edit distance apart.

Note:

There are 3 possiblities to satisify one edit distance apart:

1. Insert a character into *s* to get *t*
2. Delete a character from *s* to get *t*
3. Replace a character of *s* to get *t*

Example 1:

```
Input: s = "ab", t = "acb"

Output: true

Explanation: We can insert 'c' into s to get t.
```

Example 2:

```
Input: s = "cab", t = "ad"

Output: false

Explanation: We cannot get t from s by only one step.
```

Example 3:

```
Input: s = "1203", t = "1213"

Output: true

Explanation: We can replace '0' with '1' to get t.
```

```cpp
class Solution {
public:
    bool isOneEditDistance(string s, string t) {
        int n = s.size(), m = t.size();
        if (abs(n-m) > 1) return false;
        int len = min(s.size(), t.size());
        for (int i = 0; i < len; i++) {
            if (s[i] != t[i]) {
                if (n == m) return s.substr(i + 1) == t.substr(i + 1);
                else if (n < m) return s.substr(i) == t.substr(i + 1);
                else return s.substr(i + 1) == t.substr(i);
            }
        }
        return true;
    }
};
```

162. Find Peak Element

Medium

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.

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

    }
};
```

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

163.Missing Ranges

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"]
```

```
class Solution {
public:
    vector<string> findMissingRanges(vector<int> &nums,long lower,int upper){
        vector<string> res;
        for (long i : nums) {
            if (i < lower) continue;
            else if (i == lower) lower++;
            else {
                res.push_back(to_string(lower) + (i-lower == 1 ? ""
                                                  : ("->" + to_string(i-1))));
                lower = i+1;
            }
        }
        if (lower - upper != 1) {
            res.push_back(to_string(lower) + (upper == lower ? ""
                                              : ("->" + to_string(upper))));
        }
        return res;
    }
};
```

164. Maximum Gap★★

Hard

Given an unsorted array, find the maximum difference between the successive elements in its sorted form.

Return 0 if the array contains less than 2 elements.

Example 1:

Input: [3,6,9,1]

Output: 3

Explanation: The sorted form of the array is [1,3,6,9], either

(3,6) or (6,9) has the maximum difference 3.

Example 2:

Input: [10]

Output: 0

Explanation: The array contains less than 2 elements, therefore return 0.

Note:

- You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range.
- Try to solve it in linear time/space.

```cpp
class Solution {
public:
   class Bucket {
      public:
      bool used = false;
      int minval = numeric_limits<int>::max();       // same as INT_MAX
      int maxval = numeric_limits<int>::min();       // same as INT_MIN

   };

    int maximumGap(vector<int>& nums) {
        if (nums.size() < 2) return 0;
        int Min = *min_element(nums.begin(), nums.end());
        int Max = *max_element(nums.begin(), nums.end());
        int bucketSize = max(1, (Max - Min) / ((int)nums.size() - 1));
        int bucketNum = (Max - Min) / bucketSize + 1;
  // number of buckets
        vector<Bucket> buckets(bucketNum);

        for (auto &num : nums) {
            int bucketIdx = (num - Min) / bucketSize;
// locating correct bucket
            buckets[bucketIdx].used = true;
            buckets[bucketIdx].minval = min(num, buckets[bucketIdx].minval);
            buckets[bucketIdx].maxval = max(num, buckets[bucketIdx].maxval);
        }

        int prevBucketMax = Min, maxGap = 0;
        for (auto &bucket : buckets) {
            if (!bucket.used) continue;
            maxGap = max(maxGap, bucket.minval - prevBucketMax);
            prevBucketMax = bucket.maxval;
        }
        return maxGap;
    }
};
```

165. Compare Version Numbers

Medium

Compare two version numbers *version1* and *version2*.
If `version1 > version2` return `1;` if `version1 < version2` return `-1;` otherwise return `0`.

You may assume that the version strings are non-empty and contain only digits and the `.` character.

The `.` character does not represent a decimal point and is used to separate number sequences.

For instance, `2.5` is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

You may assume the default revision number for each level of a version number to be `0`. For example, version number `3.4` has a revision number of `3` and `4` for its first and second level revision number. Its third and fourth level revision number are both `0`.

Example 1:

Input: *version1* = "0.1", *version2* = "1.1"

Output: -1

Example 2:

Input: *version1* = "1.0.1", *version2* = "1"

Output: 1

Example 3:

Input: *version1* = "7.5.2.4", *version2* = "7.5.3"

Output: -1

Example 4:

Input: *version1* = "1.01", *version2* = "1.001"

Output: 0

Explanation: Ignoring leading zeroes, both “01” and “001" represent the same number “1”

Example 5:

Input: *version1* = "1.0", *version2* = "1.0.0"

Output: 0

Explanation: The first version number does not have a third level revision number, which means its third level revision number is default to "0"

Note:

1. Version strings are composed of numeric strings separated by dots `.` and this numeric strings **may** have leading zeroes.
2. Version strings do not start or end with dots, and they will not be two consecutive dots.

```
class Solution {
public:
    int compareVersion(string version1, string version2) {

    }
};
```

```cpp
class Solution {
public:
    int compareVersion(string version1, string version2) {
        istringstream iss1(version1), iss2(version2);
        while (iss1 || iss2) {
            char dot;
            long v1 = 0, v2 = 0;
            if (iss1) iss1 >> v1 >> dot;
            if (iss2) iss2 >> v2 >> dot;
            if (v1 != v2) {
                return v1 < v2 ? -1 : 1;
            }
        }
        return 0;
    }
};
```

166. Fraction to Recurring Decimal

Medium

Given two integers representing the numerator and denominator of a fraction, return the fraction in string format.

If the fractional part is repeating, enclose the repeating part in parentheses.

Example 1:

Input: numerator = 1, denominator = 2

Output: "0.5"

Example 2:

Input: numerator = 2, denominator = 1

Output: "2"

Example 3:

Input: numerator = 2, denominator = 3

Output: "0.(6)"

```
class Solution {
public:
    string fractionToDecimal(long a, long b) {

    }
};
```

```cpp
class Solution {
public:
    string fractionToDecimal(long a, long b) {
        if (!a) return "0";
        string s, res = (a > 0 ^ b > 0) ? "-": "";
        a = labs(a), b = labs(b);
        res += to_string(a/b);
        if ((a %= b) == 0) return res;
        unordered_map<long, int> My_map;
        int cnt = 0;
        while (!My_map.count(a) && a) {
            My_map.insert({a, cnt++});
            a = a*10;
            s += a/b+'0';
            a %= b;
        }
        if (!a) return res + '.' + s;
        else return res + '.' + s.substr(0, My_map[a]) + '('
                              + s.substr(My_map[a]) + ')';
    }
};
```

167. Two Sum II - Input array is sorted

Easy

Given an array of integers that is already ***sorted in ascending order***, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.

Note:

- Your returned answers (both index1 and index2) are not zero-based.
- You may assume that each input would have *exactly* one solution and you may not use the *same* element twice.

Example:

Input: numbers = [2,7,11,15], target = 9

Output: [1,2]

Explanation: The sum of 2 and 7 is 9. Therefore index1 = 1, index2 = 2.

```
class Solution {
public:
    vector<int> twoSum(vector<int>& numbers, int target) {

    }
};
```

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& numbers, int target) {
        int l = 0, r = numbers.size()-1;
        while (l < r) {
            if (numbers[l] + numbers[r] == target) return {l+1, r+1};
            else if (numbers[l] + numbers[r] > target) r--;
            else l++;
        }
        return {};
    }
};
```

168. Excel Sheet Column Title

Easy

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

```
1 -> A
2 -> B
3 -> C
...
26 -> Z
27 -> AA
28 -> AB
...
```

Example 1:

```
Input: 1
Output: "A"
```

Example 2:

```
Input: 28
Output: "AB"
```

Example 3:

```
Input: 701
Output: "ZY"
```

```
class Solution {
public:
    string convertToTitle(int n) {

    }
};
```

```cpp
class Solution {
public:
    string convertToTitle(int n) {
        string res;
        while (n) {
            res = char('A' + --n % 26) + res;
            n /= 26;
        }
        return res;
    }
};
```

169. Majority Element

Easy

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

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

    }
};
```

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int res = -1, time = 0;
        for (auto &i : nums) {
            i == res ? time++ : time--;
            if (time <= 0) {
                res = i; time = 1;
            }
        }
        return res;
    }
};
```

170. Two Sum III - Data structure design

Design and implement a TwoSum class. It should support the following operations: `add` and `find`.

`add` - Add the number to an internal data structure.
`find` - Find if there exists any pair of numbers which sum is equal to the value.

Example 1:

```
add(1); add(3); add(5);

find(4) -> true

find(7) -> false
```

Example 2:

```
add(3); add(1); add(2);

find(3) -> true

find(6) -> false
```

```
class TwoSum {
public:
    /** Initialize your data structure here. */
    TwoSum() {}

    /** Add the number to an internal data structure.. */
    void add(int number) {
    }

    /** Find if there exists any pair of numbers which sum is equal to the
value. */
    bool find(int value) {
    }
};

/**
 * Your TwoSum object will be instantiated and called as such:
 * TwoSum* obj = new TwoSum();
 * obj->add(number);
 * bool param_2 = obj->find(value);
 */
```

```
class TwoSum {
public:
    /** Initialize your data structure here. */
    TwoSum() {}

    /** Add the number to an internal data structure.. */
    void add(int number) {
        m[number]++;
    }

    /** Find if there exists any pair of numbers which sum is equal to the value. */
    bool find(int value) {
        for (auto &i : m) {
            int j = value - i.first;
            if (j == i.first && i.second > 1) return true;
            else if (j != i.first && m.count(j)) return true;
        }
        return false;
    }
private:
    unordered_map<int, int> m;
};

/**
 * Your TwoSum object will be instantiated and called as such:
 * TwoSum* obj = new TwoSum();
 * obj->add(number);
 * bool param_2 = obj->find(value);
 */
```

171. Excel Sheet Column Number

Easy

Given a column title as appear in an Excel sheet, return its corresponding column number.

For example:

```
A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

...
```

Example 1:

Input: "A"

Output: 1

Example 2:

Input: "AB"

Output: 28

Example 3:

Input: "ZY"

Output: 701

```cpp
class Solution {
public:
    int titleToNumber(string s) {
        int res = 0;
        for (auto &c : s) {
            res = res*26 + (c-'A') + 1;
        }
        return res;
    }
};
```

172. Factorial Trailing Zeroes★★

Easy

Given an integer *n*, return the number of trailing zeroes in *n*!.

Example 1:

Input: 3

Output: 0

Explanation: 3! = 6, no trailing zero.

Example 2:

Input: 5

Output: 1

Explanation: 5! = 120, one trailing zero.

Note: Your solution should be in logarithmic time complexity.

```
class Solution {
public:
    int trailingZeroes(int n) {

    }
};
```

```cpp
class Solution {
public:
    int trailingZeroes(int n) {
        int res = 0;
        while (n /= 5)   res += n;
        return res;
    }
};
```

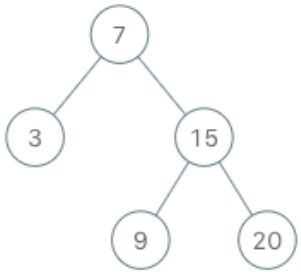
173. Binary Search Tree Iterator

Medium

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.

```
class BSTIterator {
public:
    stack<TreeNode*> s;
    TreeNode *q;
    BSTIterator(TreeNode* root) {
        q = root;
    }

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

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

174. Dungeon Game★★

Hard

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.

```cpp
class Solution {
public:
    int calculateMinimumHP(vector<vector<int>>& dungeon) {
        int n = dungeon.size(), m = dungeon[0].size();
        vector<int> f(m, INT_MAX);
        for (int i = n-1; i >= 0; --i) {
            for (int j = m-1; j >= 0; --j) {
                if (i == n-1 && j == m-1) {
                    f[j] = max(1, 1-dungeon[i][j]);
                } else if (j == m-1) {
                    f[j] = max(f[j]-dungeon[i][j], 1);
                } else {
                    f[j] = max(min(f[j], f[j+1])-dungeon[i][j], 1);
                }
            }
        }
        return f[0];
    }
};
```

175. Combine Two Tables(SQL)

Easy

SQL Schema

Table: `Person`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| PersonId    | int     |
| FirstName   | varchar |
| LastName    | varchar |
+-------------+---------+
```

PersonId is the primary key column for this table.

Table: `Address`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| AddressId   | int     |
| PersonId    | int     |
| City        | varchar |
| State       | varchar |
+-------------+---------+
```

AddressId is the primary key column for this table.

Write a SQL query for a report that provides the following information for each person in the Person table, regardless if there is an address for each of those people:

```
FirstName, LastName, City, State
```

```
SELECT FirstName, LastName, City, State
FROM Person p
LEFT JOIN Address a ON p.PersonId = a.PersonId;
```

176. Second Highest Salary(SQL)

Easy

SQL Schema

Write a SQL query to get the second highest salary from the `Employee` table.

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

For example, given the above Employee table, the query should return `200` as the second highest salary. If there is no second highest salary, then the query should return `null`.

```
+---------------------+
| SecondHighestSalary |
+---------------------+
| 200                 |
+---------------------+
```

```
SELECT
IFNULL(
      (SELECT DISTINCT Salary
       FROM Employee
       ORDER BY Salary DESC
       LIMIT 1 OFFSET 1), NULL)
AS SecondHighestSalary
```

```
SELECT
      (SELECT DISTINCT Salary
       FROM Employee
      ORDER BY Salary DESC
      LiMIT 1, 1)
AS SecondHighestSalary
```

177. Nth Highest Salary(SQL)

Medium

Write a SQL query to get the n^{th} highest salary from the `Employee` table.

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

For example, given the above Employee table, the n^{th} highest salary where $n = 2$ is `200`. If there is no n^{th} highest salary, then the query should return `null`.

```
+------------------------+
| getNthHighestSalary(2) |
+------------------------+
| 200                    |
+------------------------+
```

```
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  DECLARE M INT;
  SET M = N-1;
  RETURN (
      # Write your MySQL query statement below.
      SELECT DISTINCT Salary
      FROM Employee
      ORDER BY Salary DESC
      LIMIT M, 1
  );
END
```

178. Rank Scores(SQL)

Medium

SQL Schema

Write a SQL query to rank scores. If there is a tie between two scores, both should have the same ranking. Note that after a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no "holes" between ranks.

```
+----+-------+
| Id | Score |
+----+-------+
| 1  | 3.50  |
| 2  | 3.65  |
| 3  | 4.00  |
| 4  | 3.85  |
| 5  | 4.00  |
| 6  | 3.65  |
+----+-------+
```

For example, given the above `Scores` table, your query should generate the following report (order by highest score):

```
+-------+------+
| Score | Rank |
+-------+------+
| 4.00  | 1    |
| 4.00  | 1    |
| 3.85  | 2    |
| 3.65  | 3    |
| 3.65  | 3    |
| 3.50  | 4    |
+-------+------+
```

```
select S.Score, count(*) 'Rank'
  from Scores S
  join (select distinct Score FROM Scores) S2 on S.Score <= S2.Score
  group by S.Id
  order by S.Score desc;
```

```
select Score,(select count(distinct Score) from Scores where Score >= S.Score) 'Rank'
from Scores S
order by Score desc;
```

179. Largest Number

Medium

Given a list of non negative integers, arrange them such that they form the largest number.

Example 1:

Input: [10,2]

Output: "210"

Example 2:

Input: [3,30,34,5,9]

Output: "9534330"

Note: The result may be very large, so you need to return a string instead of an integer.

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

    }
};
```

```cpp
class Solution {
public:
    string largestNumber(vector<int>& nums) {
        vector<string> v;
        for_each (nums.begin(), nums.end(), [&](const int &i){
            v.push_back(to_string(i));
        });
        sort (v.begin(), v.end(), [](const string &a, const string &b){
            return a+b > b+a;
        });
        string res;
        for (auto &s : v) res += s;
        while (res.size() > 1 && res[0] == '0') res.erase(res.begin());
        return res;
    }
};
```

180. Consecutive Numbers(SQL)

Medium

SQL Schema

Write a SQL query to find all numbers that appear at least three times consecutively.

```
+----+-----+
| Id | Num |
+----+-----+
| 1  |  1  |
| 2  |  1  |
| 3  |  1  |
| 4  |  2  |
| 5  |  1  |
| 6  |  2  |
| 7  |  2  |
+----+-----+
```

For example, given the above `Logs` table, `1` is the only number that appears consecutively for at least three times.

```
+-----------------+
| ConsecutiveNums |
+-----------------+
| 1               |
+-----------------+
```

```
select distinct l1.num as 'ConsecutiveNums'
from Logs l1
join Logs l2 on l1.Id = l2.Id - 1 and l1.Num = l2.Num
join Logs l3 on l2.Id = l3.Id - 1 and l2.Num = l3.Num;
```

181. Employees Earning More Than Their Managers(SQL)

Easy

SQL Schema

The `Employee` table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.

```
+----+-------+--------+-----------+
| Id | Name  | Salary | ManagerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | NULL      |
| 4  | Max   | 90000  | NULL      |
+----+-------+--------+-----------+
```

Given the `Employee` table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

```
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```

```
select a.Name as 'Employee'
from Employee a
join    Employee b on a.ManagerId = b.Id and a.Salary > b.Salary;
```

182. Duplicate Emails(SQL)

Easy

SQL Schema

Write a SQL query to find all duplicate emails in a table named `Person`.

```
+----+---------+
| Id | Email   |
+----+---------+
| 1  | a@b.com |
| 2  | c@d.com |
| 3  | a@b.com |
+----+---------+
```

For example, your query should return the following for the above table:

```
+---------+
| Email   |
+---------+
| a@b.com |
+---------+
```

Note: All emails are in lowercase.

```
select Email
from    (select Email, count(Email) as num
           from Person
           group by Email) a
where a.num > 1;
```

```
select Email
from Person
group by Email
having count(Email) > 1;
```

183. Customers Who Never Order(SQL)

Easy

SQL Schema

Suppose that a website contains two tables, the `Customers` table and the `Orders` table. Write a SQL query to find all customers who never order anything.

Table: `Customers`.

```
+----+-------+
| Id | Name    |
+----+-------+
| 1    | Joe      |
| 2    | Henry |
| 3    | Sam      |
| 4    | Max      |
+----+-------+
```

Table: `Orders`.

```
+----+------------+
| Id | CustomerId |
+----+------------+
| 1    | 3                |
| 2    | 1                |
+----+------------+
```

Using the above tables as example, return the following:

```
+-----------+
| Customers |
+-----------+
| Henry         |
| Max             |
+-----------+
```

```
select Name as 'Customers'
from Customers
left join Orders on Customers.Id = Orders.CustomerId
where CustomerId is null
```

184. Department Highest Salary(SQL)

Medium

SQL Schema

The `Employee` table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Henry | 80000  | 2            |
| 3  | Sam   | 60000  | 2            |
| 4  | Max   | 90000  | 1            |
+----+-------+--------+--------------+
```

The `Department` table holds all departments of the company.

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, Max has the highest salary in the IT department and Henry has the highest salary in the Sales department.

```
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Max      | 90000  |
| Sales      | Henry    | 80000  |
+------------+----------+--------+
```

```
select Department.Name as 'Department', Employee.Name Employee, Salary
from Employee
Join Department on Employee.DepartmentId = Department.Id
where
    (Employee.DepartmentId, Salary) IN
    (
        SELECT DepartmentId, MAX(Salary)
        FROM Employee
        GROUP BY DepartmentId
    )
```

185. Department Top Three Salaries(SQL)

Hard

SQL Schema

The `Employee` table holds all employees. Every employee has an Id, and there is also a column for the department Id.

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Henry | 80000  | 2            |
| 3  | Sam   | 60000  | 2            |
| 4  | Max   | 90000  | 1            |
| 5  | Janet | 69000  | 1            |
| 6  | Randy | 85000  | 1            |
+----+-------+--------+--------------+
```

The `Department` table holds all departments of the company.

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who earn the top three salaries in each of the department. For the above tables, your SQL query should return the following rows.

```
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Max      | 90000  |
```

```
| IT          | Randy    | 85000  |
| IT          | Joe      | 70000  |
| Sales       | Henry    | 80000  |
| Sales       | Sam      | 60000  |
+-------------+----------+--------+
```

```
select d.Name Department, e1.Name Employee, e1.Salary
from Employee e1
join Department d on e1.DepartmentId = d.Id
where 3 > (select count(distinct(e2.Salary))
                from Employee e2
                where e2.Salary > e1.Salary
                        and e1.DepartmentId = e2.DepartmentId);
```

```
select d.Name Department, e1.Name Employee, e1.Salary
from Employee e1
join Department d on e1.DepartmentId = d.Id
join Employee e2 on e2.Salary >= e1.Salary and e1.DepartmentId = e2.DepartmentId
group by e1.Id
having count(distinct e2.Salary) <= 3
```

186. Reverse Words in a String II

Given an input string , reverse the string word by word.

Example:

```
Input:  ["t","h","e"," ","s","k","y"," ","i","s"," ","b","l","u","e"]

Output: ["b","l","u","e"," ","i","s"," ","s","k","y"," ","t","h","e"]
```

Note:

- A word is defined as a sequence of non-space characters.
- The input string does not contain leading or trailing spaces.
- The words are always separated by a single space.

Follow up: Could you do it *in-place* without allocating extra space?

```
class Solution {
public:
    void reverseWords(vector<char>& str) {
        vector<char>::iterator p = str.begin(), q = p;
        while (p != str.end()) {
            while (q != str.end() && *q != ' ') q++;
            reverse(p, q);
            if (q == str.end()) break;
            p = ++q;
        }
        reverse(str.begin(), str.end());
    }
};
```

187. Repeated DNA Sequences

Medium

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.

Example:

Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"

Output: ["AAAAACCCCC", "CCCCCAAAAA"]

```
class Solution {
public:
    vector<string> findRepeatedDnaSequences(string s) {

    }
};
```

```cpp
class Solution {
public:
    vector<string> findRepeatedDnaSequences(string s) {
        unordered_map<char, int> ch{{'A',0}, {'C',1}, {'G',2}, {'T',3}};
        unordered_map<int, int> My_map;
        vector<string> res;
        int t = 0, i = 0, n = s.length();
        while (i < 9) t = t << 2 | ch[s[i++]];
        while (i < n) {
            t = t << 2 & 0xfffff | ch[s[i++]];
            if (My_map[t]++ == 1) res.push_back(s.substr(i-10, 10));
        }
        return res;
    }
};
```

188. Best Time to Buy and Sell Stock IV★★

Hard

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

Design an algorithm to find the maximum profit. You may complete at most **k** transactions.

Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Example 1:

Input: [2,4,1], k = 2

Output: 2

Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 = 2.

Example 2:

Input: [3,2,6,5,0,3], k = 2

Output: 7

Explanation: Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6-2 = 4.

Then buy on day 5 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.

```cpp
class Solution {
public:
    int maxProfit(int k, vector<int> &prices) {
        if (prices.empty()) return 0;
        int N = prices.size();
        if (k >= N) return maxProfit(prices);
        vector<int> buy(k + 1, INT_MIN);
        vector<int> sell(k + 1, 0);
        //buy 代表交易 k 次且付钱买入一件
        for (int i = 0; i < N; ++i) {
            for (int j = k; j >= 1; --j) {
                buy[j] = max(buy[j], sell[j - 1] - prices[i]);
                sell[j] = max(sell[j], buy[j] + prices[i]);
            }
        }
        return sell[k];
    }
private:
    int maxProfit(vector<int>& prices) {
        int Buy = INT_MAX, res = 0;
        for (auto &Sell : prices) {
            if (Sell > Buy)   res += Sell-Buy;
            Buy = Sell;
        }
        return res;
    }
};
```

189. Rotate Array

Easy

Given an array, rotate the array to the right by *k* steps, where *k* is non-negative.

Example 1:

Input: [1,2,3,4,5,6,7] and *k* = 3

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

Explanation:

rotate 1 steps to the right: [7,1,2,3,4,5,6]

rotate 2 steps to the right: [6,7,1,2,3,4,5]

rotate 3 steps to the right: [5,6,7,1,2,3,4]

Example 2:

Input: [-1,-100,3,99] and *k* = 2

Output: [3,99,-1,-100]

Explanation:

rotate 1 steps to the right: [99,-1,-100,3]

rotate 2 steps to the right: [3,99,-1,-100]

Note:

- Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
- Could you do it in-place with O(1) extra space?

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {

    }
};
```

```cpp
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        if (nums.empty()) return;
        k = nums.size() - k % nums.size();
        reverse(nums, 0, k-1);
        reverse(nums, k, nums.size()-1);
        reverse(nums, 0, nums.size()-1);
    }

    void reverse(vector<int>& nums, int l, int r) {
        while (l < r)  swap(nums[l++], nums[r--]);
    }
};
```

190. Reverse Bits

Easy

Reverse bits of a given 32 bits unsigned integer.

Example 1:

Input: 00000010100101000001111010011100

Output: 00111001011110000010100101000000

Explanation: The input binary string **00000010100101000001111010011100** represents the unsigned integer 43261596, so return 964176192 which its binary representation is **00111001011110000010100101000000**.

Example 2:

Input: 11111111111111111111111111111101

Output: 10111111111111111111111111111111

Explanation: The input binary string **11111111111111111111111111111101** represents the unsigned integer 4294967293, so return 3221225471 which its binary representation is **10101111110010110010011101101001**.

Note:

- Note that in some languages such as Java, there is no unsigned integer type. In this case, both input and output will be given as signed integer type and should not affect your implementation, as the internal binary representation of the integer is the same whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 2** above the input represents the signed integer `-3` and the output represents the signed integer `-1073741825`.

Follow up:

If this function is called many times, how would you optimize it?

```cpp
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t res = 0;
        for (int i = 0; i < 32; i++) {
            res <<= 1;
            res += n & 0x00000001;
            n >>= 1;
        }
        return res;
    }
};
```

```cpp
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        n = (n >> 16) | (n << 16);
        n = ((n & 0xff00ff00) >> 8) | ((n & 0x00ff00ff) << 8);
        n = ((n & 0xf0f0f0f0) >> 4) | ((n & 0x0f0f0f0f) << 4);
        n = ((n & 0xcccccccc) >> 2) | ((n & 0x33333333) << 2);
        n = ((n & 0xaaaaaaaa) >> 1) | ((n & 0x55555555) << 1);
        return n;
    }
};
```

191. Number of 1 Bits

Easy

Write a function that takes an unsigned integer and return the number of '1' bits it has (also known as the Hamming weight).

Example 1:

Input: 00000000000000000000000000001011

Output: 3

Explanation: The input binary string **00000000000000000000000000001011** has a total of three '1' bits.

Example 2:

Input: 00000000000000000000000010000000

Output: 1

Explanation: The input binary string **00000000000000000000000010000000** has a total of one '1' bit.

Example 3:

Input: 11111111111111111111111111111101

Output: 31

Explanation: The input binary string **11111111111111111111111111111101** has a total of thirty one '1' bits.

Note:

- Note that in some languages such as Java, there is no unsigned integer type. In this case, the input will be given as signed integer type and should not affect your implementation, as the internal binary representation of the integer is the same whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 3** above the input represents the signed integer `-3`.
-

Follow up:

If this function is called many times, how would you optimize it?

```cpp
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int cnt = 0;
        while (n) {
            n &= (n - 1);
            cnt++;
        }
        return cnt;
    }
};
```

192. Word Frequency(Bash)

Medium

Write a bash script to calculate the frequency of each word in a text file `words.txt`.

For simplicity sake, you may assume:

- `words.txt` contains only lowercase characters and space `' '` characters.
- Each word must consist of lowercase characters only.
- Words are separated by one or more whitespace characters.

Example:

Assume that `words.txt` has the following content:

```
the day is sunny the the

the sunny is is
```

Your script should output the following, sorted by descending frequency:

```
the 4

is 3

sunny 2

day 1
```

Note:

- Don't worry about handling ties, it is guaranteed that each word's frequency count is unique.
- Could you write it in one-line using Unix pipes?

193. Valid Phone Numbers(Bash)

Easy

Given a text file `file.txt` that contains list of phone numbers (one per line), write a one liner bash script to print all valid phone numbers.

You may assume that a valid phone number must appear in one of the following two formats: (xxx) xxx-xxxx or xxx-xxx-xxxx. (x means a digit)

You may also assume each line in the text file must not contain leading or trailing white spaces.

Example:

Assume that `file.txt` has the following content:

```
987-123-4567

123 456 7890

(123) 456-7890
```

Your script should output the following valid phone numbers:

```
987-123-4567

(123) 456-7890
```

194. Transpose File(Bash)

Medium

Given a text file `file.txt`, transpose its content.

You may assume that each row has the same number of columns and each field is separated by the `' '` character.

Example:

If `file.txt` has the following content:

```
name age
alice 21
ryan 30
```

Output the following:

```
name alice ryan
age 21 30
```

195. Tenth Line(Bash)

Easy

Given a text file `file.txt`, print just the 10th line of the file.

Example:

Assume that `file.txt` has the following content:

```
Line 1

Line 2

Line 3

Line 4

Line 5

Line 6

Line 7

Line 8

Line 9

Line 10
```

Your script should output the tenth line, which is:

```
Line 10
```

Note:

1. If the file contains less than 10 lines, what should you output?
2. There's at least three different solutions. Try to explore all possibilities.

196. Delete Duplicate Emails(SQL)

Easy

Write a SQL query to **delete** all duplicate email entries in a table named `Person`, keeping only unique emails based on its *smallest* **Id**.

```
+----+------------------+
| Id | Email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
| 3  | john@example.com |
+----+------------------+
```

Id is the primary key column for this table.

For example, after running your query, the above `Person` table should have the following rows:

```
+----+------------------+
| Id | Email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
+----+------------------+
```

Note:

Your output is the whole `Person` table after executing your sql. Use `delete` statement.

```
/*
delete p1 from Person p1, Person p2
where p1.Email = p2.Email and p1.Id > p2.Id
*/

delete from Person where Id not in (
    select id from (
        select min(Id) as id from Person
        group by Email
    ) as _          # it needs an alias for set
)
```

197. Rising Temperature(SQL)

Easy

SQL Schema

Given a `Weather` table, write a SQL query to find all dates' Ids with higher temperature compared to its previous (yesterday's) dates.

```
+---------+------------------+------------------+
| Id(INT) | RecordDate(DATE) | Temperature(INT) |
+---------+------------------+------------------+
|       1 |       2015-01-01 |               10 |
|       2 |       2015-01-02 |               25 |
|       3 |       2015-01-03 |               20 |
|       4 |       2015-01-04 |               30 |
+---------+------------------+------------------+
```

For example, return the following Ids for the above `Weather` table:

```
+----+
| Id |
+----+
|  2 |
|  4 |
+----+
```

```
select a.Id
from Weather a
join Weather b on a.Temperature > b.Temperature and datediff(a.RecordDate, b.RecordDate)
= 1
```

198. House Robber

Easy

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.

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

    }
};
```

```cpp
class Solution {
public:
    int rob(vector<int>& nums) {
        int res[2] = {0, 0};   // res[0]: 不偷    res[1]: 偷
        for (auto &i : nums) {
            int temp = res[1];
            res[1] = res[0] + i;
            res[0] = max(res[0], temp);
        }
        return max(res[0], res[1]);
    }
};
```

199. Binary Tree Right Side View

Medium

Given a binary tree, imagine yourself standing on the *right* side of it, return the values of the nodes you can see ordered from top to bottom.

Example:

Input: [1,2,3,null,5,null,4]

Output: [1, 3, 4]

Explanation:

```
   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
```

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {

    }
};
```

```cpp
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> res;
        if (root == nullptr) return res;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int sz = q.size();
            while (sz--) {
               TreeNode *t = q.front();
               q.pop();
               if (t->left) q.push(t->left);
               if (t->right) q.push(t->right);
               if (!sz) res.push_back(t->val);
            }
        }
        return res;
    }
};
```

200. Number of Islands

Medium

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

```
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {

    }
};
```

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

private:
    const int dx[4] = {1, -1, 0, 0};
    const int dy[4] = {0, 0, 1, -1};

    void dfs(int x, int y, int n, int m, vector<vector<char>>& grid) {
        grid[x][y] = '0';
        for (int k = 0; k < 4; k++) {
            int xx = x+dx[k], yy = y+dy[k];
            if (xx >= n || yy >= m || xx < 0 || yy < 0) continue;
            if (grid[xx][yy] != '0') dfs(xx, yy, n, m, grid);
        }
    }
};
```