Solutions of leetcode

hustsxh@gmail.com

February 7, 2015

1 Solutions of leetcode algorithm problems

1.1 3Sum Closest

Problem Description

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

```
For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

Solution

水题

Code

```
class Solution {
public:
    int threeSumClosest(vector<int> &num, int target) {
        if (num.size() < 3) return 0;
        int ans = num[0] + num[1] + num[2], temp;
        sort(num.begin(), num.end());
        for (int i = 0; i < num.size(); ++i) {
            int j = i + 1, k = num.size() - 1;
            while(j < k) {
                while (j < k && num[i] + num[j] + num[k] >= target) {
                    if (abs(ans - target) > abs((temp = num[i] + num[j] + num
                        [k]) - target)) {
                        ans = temp;
                    }
                    -- k;
                }
                while (j < k && num[i] + num[j] + num[k] <= target) {
                    if (abs(ans - target) > abs((temp = num[i] + num[j] + num
                        [k]) - target)) {
                        ans = temp;
                    }
                    ++ j;
                }
            }
        }
        return ans;
    }
};
```

1.2 3Sum

Problem Description

Given an array S of n integers, are there elements a, b, c in S such that $a + b + c = 0$? Find all unique triplets in the array which gives the sum of zero.

Note:

- Elements in a triplet (a,b,c) must be in non-descending order. (ie, $a \leqslant b \leqslant c$)
- The solution set must not contain duplicate triplets.

```
For example, given array S = {-1 0 1 2 -1 -4},

A solution set is:
(-1, 0, 1)
(-1, -1, 2)
```

Solution

水题

Code

```
class Solution {
public:
    vector<vector<int> > threeSum(vector<int> &num) {
        vector<vector<int> > ans;
        vector<int> triplet(3);
        sort(num.begin(), num.end());
        int i = 0;
        do {
            int j = i + 1, k = num.size() - 1;
            do {
                while(j < k && num[i] + num[j] + num[k] > 0) {
                    -- k;
                }
                if (j < k && num[i] + num[j] + num[k] == 0) {
                    triplet[0] = num[i],
                    triplet[1] = num[j],
                    triplet[2] = num[k],
                    ans.push_back(triplet);
                }
                ++ j;
                while(j < k && num[j] == num[j - 1]) ++j;
            } while (j < k);
            ++ i;
            while (i < num.size() && num[i] == num[i - 1]) ++i;
        } while (i < num.size());

        return ans;
    }
};
```

1.3 4Sum

Problem Description

Given an array S of n integers, are there elements a, b, c, and d in S such that $a + b + c + d$ = target? Find all unique quadruplets in the array which gives the sum of target.

Note:

- Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, $a \leqslant b \leqslant c \leqslant d$)
- The solution set must not contain duplicate quadruplets.

```
For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

A solution set is:
(-1,  0, 0, 1)
(-2, -1, 1, 2)
(-2,  0, 0, 2)
```

Solution

水题

Code

```
class Solution {
public:
    vector<vector<int> > fourSum(vector<int> &num, int target) {
        int n = num.size();
        vector<vector<int> > ans;
        if (n < 4) return ans;
        vector<int> next(n, n), pre(n, -1);
        sort(num.begin(), num.end());
        for (int i = n - 2; i >= 0; --i) {
            if (num[i] == num[i + 1]) {
                next[i] = next[i + 1];
            } else {
                next[i] = i + 1;
            }
        }
        for (int i = 1; i < n; ++i) {
            if (num[i] == num[i - 1]) {
                pre[i] = pre[i - 1];
            } else {
                pre[i] = i - 1;
            }
        }
        for (int i = 0; i < n; i = next[i]) {
            for (int j = i + 1; j < n; j = next[j]) {
                int p = j + 1, q = n - 1;
                while (p < q) {
                    while (p < q && num[i] + num[j] + num[p] + num[q] !=
                        target) {
                        if (num[i] + num[j] + num[p] + num[q] < target) {
                            p = next[p];
                        } else {
                            q = pre[q];
                        }
                    }
                    if (p < q) {
                        ans.push_back(quadruplets(num[i], num[j], num[p], num
                            [q]));
                        p = next[p];
                    }
                }
            }
        }
        return ans;
    }
private:
    vector<int> quadruplets(int a, int b, int c, int d) {
        vector<int> replets;
        replets.push_back(a);
        replets.push_back(b);
        replets.push_back(c);
        replets.push_back(d);
        return replets;
    }
};
```

1.4 Add Binary

Problem Description

Given two binary strings, return their sum (also a binary string).
For example,
a = **"11"**
b = **"1"**
Return **"100"**.

Solution

水题

Code

```
class Solution {
public:
    string addBinary(string a, string b) {
        string sum;
        int c = 0;
        for (int i = 0; i < a.size() || i < b.size(); ++i) {
            if (i < a.size()) {
                c += a[a.size() - 1 - i] - '0';
            }
            if (i < b.size()) {
                c += b[b.size() - 1 - i] - '0';
            }
            sum.push_back(c % 2 + '0');
            c /= 2;
        }
        if (c) {
            sum.push_back('1');
        }
        return string(sum.rbegin(), sum.rend());
    }
};
```

1.5 Add Two Numbers

Problem Description

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Solution

水题

Code

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *addTwoNumbers(ListNode *l1, ListNode *l2) {
        if (l1 == NULL && l2 == NULL) return NULL;
        ListNode head(0), *p;
        int c = 0;
        p = &head;
        while(l1 || l2) {
            if (l1) {
                c += l1->val;
                l1 = l1->next;
            }
            if (l2) {
                c += l2->val;
                l2 = l2->next;
            }
            p->next = new ListNode(c % 10);
            p = p->next;
            c /= 10;
        }
        if (c) {
            p->next = new ListNode(c);
        }
        return head.next;
    }
};
```

1.6 Anagrams

Problem Description

Given an array of strings, return all groups of strings that are anagrams.
Note: All inputs will be in lower-case.

Solution

水题

Code

```cpp
class Solution {
public:
    vector<string> anagrams(vector<string> &strs) {
        vector<string> ans;
        map<string, int> cnt;
        for (int i = 0; i < strs.size(); ++i) {
            string s = strs[i];
            sort(s.begin(), s.end());
            if (cnt.find(s) == cnt.end()) {
                cnt[s] = 1;
            } else {
                ++ cnt[s];
            }
        }
        for (int i = 0; i < strs.size(); ++i) {
            string s = strs[i];
            sort(s.begin(), s.end());
            if (cnt[s] >= 2) {
                ans.push_back(strs[i]);
            }
        }
        return ans;
    }
};
```

1.7 Balanced Binary Tree

Problem Description

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

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

Solution

水题

Code

```cpp
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isBalanced(TreeNode *root) {
        int depth;
        return _isBalanced(root, depth);
    }
private:
    bool _isBalanced(TreeNode *root, int &depth) {
        if (root == NULL) {
            depth = 0;
            return true;
        }
        int leftdepth, rightdepth;
        if (_isBalanced(root->left, leftdepth) && _isBalanced(root->right,
            rightdepth)) {
            depth = max(leftdepth, rightdepth) + 1;
            return abs(leftdepth - rightdepth) <= 1;
        }
        return false;
    }
};
```

1.8 Best Time to Buy and Sell Stock II

Problem Description

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 (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Solution

水题

Code

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

1.9 Best Time to Buy and Sell Stock III

Problem Description

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 (ie, you must sell the stock before you buy again).

Solution

水题

Code

```cpp
class Solution {
public:
    int maxProfit(vector<int> &prices) {
        int n = prices.size();
        if (n <= 1) return 0;
        vector<vector<int> > f(n + 1, vector<int>(5, 0));
        f[0][1] = f[0][3] = -prices[0];
        for (int i = 1; i <= prices.size(); ++i) {
            f[i][0] = f[i - 1][0];
            f[i][1] = max(f[i][0] - prices[i - 1], f[i - 1][1]);
            f[i][2] = max(f[i][1] + prices[i - 1], f[i - 1][2]);
            f[i][3] = max(f[i][2] - prices[i - 1], f[i - 1][3]);
            f[i][4] = max(f[i][3] + prices[i - 1], f[i - 1][4]);
        }
        return f[n][4];
    }
};
```

1.10 Best Time to Buy and Sell Stock

Problem Description

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

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

Solution

水题

Code

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

1.11 Binary Search Tree Iterator

Problem Description

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

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

Note: next() and **hasNext()** should run in average O(1) time and uses O(h) memory, where h is the height of the tree.

Credits:

Special thanks to @ts for adding this problem and creating all test cases.

Solution

水题

Code

```
class BSTIterator {
public:
    BSTIterator(TreeNode *root) {
        while (root) {
            path.push_back(root);
            root = root->left;
        }
    }

    /** @return whether we have a next smallest number */
    bool hasNext() {
        return path.size() != 0;
    }

    /** @return the next smallest number */
    int next() {
        TreeNode *ptr = path[path.size() - 1];
        int value = ptr->val;
        if (ptr->right) {
            ptr = ptr->right;
            path.push_back(ptr);
            while (ptr->left) {
                ptr = ptr->left;
                path.push_back(ptr);
            }
        } else {
            while (path.size() >= 2 && path[path.size() - 2]->right == path[
                path.size() - 1]) {
                path.pop_back();
            }
            path.pop_back();
        }
        return value;
    }
private:
    vector<TreeNode*> path;
};

/**
 * Your BSTIterator will be called like this:
 * BSTIterator i = BSTIterator(root);
 * while (i.hasNext()) cout << i.next();
 */
```

1.12 Binary Tree Inorder Traversal

Problem Description

Given a binary tree, return the *inorder* traversal of its nodes' values.
For example:
Given binary tree **{1,#,2,3}**,

```
1
```

```
    \
     2
    /
   3
```

return [**1,3,2**].

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

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

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

The above binary tree is serialized as **"{1,2,3,#,#,4,#,#,5}"**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode *root) {
        vector<int> ans;
        inorderTraversal(root, ans);
        return ans;
    }
private:
    void inorderTraversal(TreeNode *root, vector<int> & ans) {
        if (root == NULL) return ;
        inorderTraversal(root->left, ans);
        ans.push_back(root->val);
        inorderTraversal(root->right, ans);
    }
};
```

1.13 Binary Tree Level Order Traversal II

Problem Description

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,#,#,15,7}**,

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

return its bottom-up level order traversal as:

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

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

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

The above binary tree is serialized as **"{1,2,3,#,#,4,#,#,5}"**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int> > levelOrderBottom(TreeNode *root) {
        vector<vector<int> > ans;
        if (root == NULL) return ans;
        vector<int> nowval;
        vector<TreeNode *> up, now;
        now.push_back(root);
        while (now.size()){
            nowval.clear();
            for (int i = 0; i < now.size(); ++i) {
                nowval.push_back(now[i]->val);
            }
            ans.push_back(nowval);
            up = now;
            now.clear();
            for (int i = 0; i < up.size(); ++i) {
                if (up[i]->left) {
                    now.push_back(up[i]->left);
                }
                if (up[i]->right) {
                    now.push_back(up[i]->right);
                }
            }
        }
        return vector<vector<int> > (ans.rbegin(), ans.rend());
    }
};
```

1.14 Binary Tree Level Order Traversal

Problem Description

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

For example:

Given binary tree **{3,9,20,#,#,15,7}**,

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

return its level order traversal as:

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

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

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

The above binary tree is serialized as **"{1,2,3,#,#,4,#,#,5}"**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int> > levelOrder(TreeNode *root) {
        vector<vector<int> > ans;
        if (root == NULL) return ans;
        vector<int> nowval;
        vector<TreeNode *> up, now;
        now.push_back(root);
        while (now.size()){
            nowval.clear();
            for (int i = 0; i < now.size(); ++i) {
                nowval.push_back(now[i]->val);
            }
            ans.push_back(nowval);
            up = now;
            now.clear();
            for (int i = 0; i < up.size(); ++i) {
                if (up[i]->left) {
                    now.push_back(up[i]->left);
                }
                if (up[i]->right) {
                    now.push_back(up[i]->right);
                }
            }
        }
        return ans;
    }
};
```

1.15 Binary Tree Maximum Path Sum

Problem Description

Given a binary tree, find the maximum path sum.
The path may start and end at any node in the tree.
For example:
Given the below binary tree,

```
   1
  / \
 2   3
```

Return **6**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int maxPathSum(TreeNode *root) {
        if (root == NULL) return 0;
        int ans = root->val;
        maxPath(root, ans);
        return ans;
    }
private:
    int maxPath(TreeNode *root, int &ans) {
        if (root == NULL) return 0;
        int leftPath = max(maxPath(root->left, ans), 0);
        int rightPath = max(maxPath(root->right, ans), 0);
        ans = max(leftPath + rightPath + root->val, ans);
        return max(leftPath, rightPath) + root->val;
    }
};
```

1.16 Binary Tree Postorder Traversal

Problem Description

Given a binary tree, return the *postorder* traversal of its nodes' values.
For example:
Given binary tree **{1,#,2,3}**,

```
1
 \
  2
 /
3
```

return [**3,2,1**].
Note: Recursive solution is trivial, could you do it iteratively?

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> postorderTraversal(TreeNode *root) {
        vector<int> nodes;
        postorderTraversal(root, nodes);
        return nodes;
    }
private:
    void postorderTraversal(TreeNode *root, vector<int> &nodes) {
        if (root == NULL) return;
        postorderTraversal(root->left, nodes);
        postorderTraversal(root->right, nodes);
        nodes.push_back(root->val);
    }
};
```

1.17 Binary Tree Preorder Traversal

Problem Description

Given a binary tree, return the *preorder* traversal of its nodes' values.
For example:
Given binary tree **{1,#,2,3}**,

```
 1
  \
   2
  /
 3
```

return [**1,2,3**].
Note: Recursive solution is trivial, could you do it iteratively?

Solution

水题

Code

```
/**
 * Definition for binary tree
 * 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) {
        vector<int> nodes;
        preorderTraversal(root, nodes);
        return nodes;
    }
private:
    void preorderTraversal(TreeNode *root, vector<int> &nodes) {
        if (root == NULL) return;
        nodes.push_back(root->val);
        preorderTraversal(root->left, nodes);
        preorderTraversal(root->right, nodes);
    }
};
```

1.18 Binary Tree Zigzag Level Order Traversal

Problem Description

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

For example:

Given binary tree **{3,9,20,#,#,15,7}**,

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

return its zigzag level order traversal as:

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

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

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

The above binary tree is serialized as **"{1,2,3,#,#,4,#,#,5}"**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int> > zigzagLevelOrder(TreeNode *root) {
        vector<vector<int> > ans;
        if (root == NULL) return ans;
        vector<int> nowval;
        vector<TreeNode *> up, now;
        now.push_back(root);
        int level = 0;
        while (now.size()){
            nowval.clear();
            for (int i = 0; i < now.size(); ++i) {
                nowval.push_back(now[i]->val);
            }
            ++ level;
            if (level % 2 == 1) {
                ans.push_back(nowval);
            } else {
                ans.push_back(vector<int> (nowval.rbegin(), nowval.rend()));
            }
            up = now;
            now.clear();
            for (int i = 0; i < up.size(); ++i) {
                if (up[i]->left) {
                    now.push_back(up[i]->left);
                }
                if (up[i]->right) {
                    now.push_back(up[i]->right);
                }
            }
        }
        return ans;
    }
};
```

1.19 Candy

Problem Description

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:
What is the minimum candies you must give?

Solution

水题

Code

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

1.20 Climbing Stairs

Problem Description

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

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

Solution

水题

Code

```
class Solution {
public:
    int climbStairs(int n) {
        int f[n + 1];
        f[0] = f[1] = 1;
        for (int i = 2; i <= n; ++i) {
            f[i] = f[i - 1] + f[i - 2];
        }
        return f[n];
    }
};
```

1.21 Clone Graph

Problem Description

Clone an undirected graph. Each node in the graph contains a **label** and a list of its **neighbors**.

Nodes are labeled uniquely.

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 #.

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

- First node is labeled as **0**. Connect node **0** to both nodes **1** and **2**.
- Second node is labeled as **1**. Connect node **1** to node **2**.
- 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:

```
       1
      / \
     /   \
    0 --- 2
         / \
         \_/
```

Solution

水题

Code

```
/**
 * Definition for undirected graph.
 * struct UndirectedGraphNode {
 *     int label;
 *     vector<UndirectedGraphNode *> neighbors;
 *     UndirectedGraphNode(int x) : label(x) {};
 * };
 */
class Solution {
public:
    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node) {
        if (node == NULL) return NULL;
        pmap.clear();
        return _cloneGraph(node);
    }
private:
    UndirectedGraphNode * _cloneGraph(UndirectedGraphNode *node) {
        if (pmap.find(node) != pmap.end()) {
            return pmap[node];
        }
        UndirectedGraphNode *newnode = new UndirectedGraphNode(node->label);
        pmap[node] = newnode;
        for (int i = 0; i < node->neighbors.size(); ++i) {
            newnode->neighbors.push_back(_cloneGraph(node->neighbors[i]));
        }
        return newnode;
    }
    map<UndirectedGraphNode *, UndirectedGraphNode *> pmap;
};
```

1.22 Combination Sum II

Problem Description

Given a collection of candidate numbers ($\boldsymbol{C}$) and a target number ($\boldsymbol{T}$), find all unique combinations in $\boldsymbol{C}$ where the candidate numbers sums to $\boldsymbol{T}$.

Each number in $\boldsymbol{C}$ may only be used **once** in the combination.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination ($a1$, $a2$, $\cdots$, ak) must be in non-descending order. (ie, $a1 \leqslant a2 \leqslant \cdots \leqslant ak$).
- The solution set must not contain duplicate combinations.

For example, given candidate set **10,1,2,7,6,1,5** and target **8**,
A solution set is:
[**1, 7**]
[**1, 2, 5**]
[**2, 6**]
[**1, 1, 6**]

Solution

水题

Code

```
class Solution {
public:
    vector<vector<int> > combinationSum2(vector<int> &num, int target) {
        int n = num.size();
        ans.clear();
        select.clear();
        if (n == 0) return ans;
        sort(num.rbegin(), num.rend());
        int c[n];
        for (int i = 0; i < n; ++i) {
            c[i] = num[i];
        }
        _gen(c, 0, n, target);
        return ans;
    }
private:
    void _gen(int c[], int k, int n, int target, bool last = true) {
        if (n == k) {
            if(target == 0) {
                vector<int> onec(select.rbegin(), select.rend());
                ans.push_back(onec);
            }
            return ;
        }

        if (!(last == false && c[k] == c[k - 1]) && c[k] <= target) {
            select.push_back(c[k]);
            _gen(c, k + 1, n, target - c[k], true);
            select.pop_back();
        }

        _gen(c, k + 1, n, target, false);
    }
    vector<vector<int> >  ans;
    vector<int> select;
};
```

1.23 Combination Sum

Problem Description

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The **same** repeated number may be chosen from C unlimited number of times.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination ($a1$, $a2$, $\cdots$, ak) must be in non-descending order. (ie, $a1 \leqslant a2 \leqslant \cdots \leqslant ak$).
- The solution set must not contain duplicate combinations.

For example, given candidate set **2,3,6,7** and target **7**,
A solution set is:
[**7**]
[**2, 2, 3**]

Solution

水题

Code

```cpp
class Solution {
public:
    vector<vector<int> > combinationSum(vector<int> &candidates, int target)
        {
        int n = candidates.size();
        sort(candidates.rbegin(), candidates.rend());
        int c[n], m = 0;
        c[m++] = candidates[0];
        for (int i = 1; i < n; ++i) {
            if (candidates[i] != candidates[i - 1]) c[m++] = candidates[i];
        }
        ans.clear();
        select.clear();
        _gen(c, m, target);
        return ans;
    }
private:
    void _gen(int c[], int n, int target) {
        if (n == 0) {
            if(target == 0) {
                vector<int> onec(select.rbegin(), select.rend());
                ans.push_back(onec);
            }
            return ;
        }
        int times = 0;

        while(c[0] * ++times <= target) {
            select.push_back(c[0]);
            _gen(c + 1, n - 1, target - c[0] * times);
        }

        for (int i = 1; i < times; ++i) {
            select.pop_back();
        }
        _gen(c + 1, n - 1, target);
    }
    vector<vector<int> >  ans;
    vector<int> select;
};
```

1.24 Combinations

Problem Description

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.
For example,
If $n = 4$ and $k = 2$, a solution is:

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

Solution

水题

Code

```
class Solution {
public:
    vector<vector<int> > combine(int n, int k) {
        ans.clear();
        select.clear();
        _gen(k, n, 0);
        return  ans;
    }
private:
    void _gen(int k, int n, int last) {
        if (select.size() == k) {
            ans.push_back(select);
            return ;
        }
        for (int i = last + 1; i <= n; ++i) {
            select.push_back(i);
            _gen(k, n, i);
            select.pop_back();
        }
    }
    vector<int> select;
    vector<vector<int> > ans;
};
```

1.25 Compare Version Numbers

Problem Description

Compare two version numbers *version1* and *version1*.
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.

Here is an example of version numbers ordering:
Credits:
Special thanks to @ts for adding this problem and creating all test cases.

```
0.1 < 1.1 < 1.2 < 13.37
```

Solution

水题

Code

```cpp
class Solution {
public:
    int compareVersion(string version1, string version2) {
        int sub = vecsub(version2vec(version1), version2vec(version2));
        return min(1, max(-1, sub));
    }
private:
    vector<int> version2vec(const string& v) {
        vector<int> vec;
        size_t dot1 = -1, dot2;
        while ((dot2 = v.find('.', dot1 + 1)) != string::npos) {
            vec.push_back(atoi(v.substr(dot1 + 1, dot2 - dot1 - 1).c_str()));
            dot1 = dot2;
        }
        vec.push_back(atoi(v.substr(dot1 + 1).c_str()));
        return vec;
    }

    int vecsub(const vector<int>& v1, const vector<int>& v2) {
        for (size_t i = 0; i < v1.size() || i < v2.size(); ++i) {
            if (!(i < v1.size())) {
                if (v2[i] != 0) return -1;
            } else if (!(i < v2.size())) {
                if (v1[i] != 0) return 1;
            } else if (v1[i] != v2[i]) return v1[i] - v2[i];
        }
        return 0;
    }
};
```

1.26 Construct Binary Tree from Inorder and Postorder Traversal

Problem Description

Given inorder and postorder traversal of a tree, construct the binary tree.
Note:
You may assume that duplicates do not exist in the tree.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode *buildTree(vector<int> &inorder, vector<int> &postorder) {
        if (inorder.size() == 0) return NULL;
        int n = inorder.size();
        int post[n], in[n];
        for (int i = 0; i < n; ++i) {
            post[i] = postorder[i];
            in[i] = inorder[i];
        }
        return _bulidTree(post, in, n);
    }
private:
    TreeNode *_bulidTree(int post[], int in[], int n) {
        if (n == 0) return NULL;
        if (n == 1) {
            return new TreeNode(post[n - 1]);
        }
        int k = 0;
        while(k < n && in[k] != post[n - 1]) ++ k;
        TreeNode *root = new TreeNode(post[n - 1]);
        root->left = _bulidTree(post, in, k);
        root->right = _bulidTree(post + k, in + 1 + k, n - k - 1);
        return root;
    }
};
```

1.27 Construct Binary Tree from Preorder and Inorder Traversal

Problem Description

Given preorder and inorder traversal of a tree, construct the binary tree.
Note:
You may assume that duplicates do not exist in the tree.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
```

```
class Solution {
public:
    TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder) {
        if (preorder.size() == 0) return NULL;
        int n = preorder.size();
        int pre[n], in[n];
        for (int i = 0; i < n; ++i) {
            pre[i] = preorder[i];
            in[i] = inorder[i];
        }
        return _bulidTree(pre, in, n);
    }
private:
    TreeNode *_bulidTree(int pre[], int in[], int n) {
        if (n == 0) return NULL;
        if (n == 1) {
            return new TreeNode(pre[0]);
        }
        int k = 0;
        while(k < n && in[k] != pre[0]) ++ k;
        TreeNode *root = new TreeNode(pre[0]);
        root->left = _bulidTree(pre + 1, in, k);
        root->right = _bulidTree(pre + 1 + k, in + 1 + k, n - k - 1);
        return root;
    }
};
```

1.28 Container With Most Water

Problem Description

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

Note: You may not slant the container.

Solution

水题

Code

```
class Solution {
public:
    int maxArea(vector<int> &height) {
        int n = height.size();
        int s1[n], top1 = 0, s2[n], top2 = 0, area = 0;
        for(int i = 0; i < n; ++i) {
            if (top1 == 0 || height[s1[top1 - 1]] < height[i]) {
                s1[top1++] = i;
            }
        }
        for (int i = n - 1; i >= 0; --i) {
            if (top2 == 0 || height[s2[top2 - 1]] < height[i]) {
                s2[top2++] = i;
            }
        }
        int p1 = 0, p2 = 0;
        while(p1 < top1 && p2 < top2 && s1[p1] < s2[p2]) {
            int newarea = min(height[s1[p1]], height[s2[p2]]) * (s2[p2] - s1[
                p1]);
            area = max(area, newarea);
            if (height[s1[p1]] < height[s2[p2]]) {
                ++ p1;
            } else {
                ++ p2;
            }
        }
        return area;
    }
};
```

1.29 Convert Sorted Array to Binary Search Tree

Problem Description

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode *sortedArrayToBST(vector<int> &num) {
        int a[num.size()], n = num.size();
        for (int i = 0; i < n; ++i) {
            a[i] = num[i];
        }
        return _gen(a, n);
    }
private:
    TreeNode *_gen(int a[], int n) {
        if (n == 0) return NULL;
        TreeNode *root = new TreeNode(a[n / 2]);
        root->left = _gen(a, n / 2);
        root->right = _gen(a + n / 2 + 1, n - n / 2 - 1);
        return root;
    }
};
```

1.30 Convert Sorted List to Binary Search Tree

Problem Description

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

Solution

水题

Code

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode *sortedListToBST(ListNode *head) {
        vector<int> vals;
        for (ListNode *p = head; p; p = p->next) {
            vals.push_back(p->val);
        }
        return _gen(vals, 0, vals.size());
    }
private:
    TreeNode *_gen(vector<int> &vals, int left, int right) {
        if (left >= right) {
            return NULL;
        }
        int mid = (left + right) >> 1;
        TreeNode *root = new TreeNode(vals[mid]);
        root->left = _gen(vals, left, mid);
        root->right = _gen(vals, mid + 1, right);
        return root;
    }
};
```

1.31 Copy List with Random Pointer

Problem Description

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.

Solution

水题

Code

```
/**
 * Definition for singly-linked list with a random pointer.
 * struct RandomListNode {
 *     int label;
 *     RandomListNode *next, *random;
 *     RandomListNode(int x) : label(x), next(NULL), random(NULL) {}
 * };
 */
class Solution {
public:
    RandomListNode *copyRandomList(RandomListNode *head) {
        map<RandomListNode*, RandomListNode*> keyword;
        keyword[NULL] = NULL;
        RandomListNode newhead(0), *tail;
        tail = &newhead;
        for (RandomListNode *p = head; p; p = p->next) {
            tail->next = new RandomListNode(p->label);
            tail = tail->next;
            keyword[p] = tail;
        }
        for (RandomListNode *p = head; p; p = p->next) {
            keyword[p]->random = keyword[p->random];
        }
        return newhead.next;
    }
};
```

1.32 Count and Say

Problem Description

The count-and-say sequence is the sequence of integers beginning as follows:
1, 11, 21, 1211, 111221, ...
1 is read off as **"one 1"** or **11**.
11 is read off as **"two 1s"** or **21**.
21 is read off as **"one 2**, then **one 1"** or **1211**.
Given an integer n, generate the nth sequence.
Note: The sequence of integers will be represented as a string.

Solution

水题

Code

```
class Solution {
public:
    string countAndSay(int n) {
        string ans = "1";
        for (int k = 2; k <= n; ++k) {
            string read;
            ans.push_back('e');
            int cnt = 1;
            for (int i = 1; i < ans.size(); ++i) {
                if (ans[i - 1] == ans[i]) {
                    ++ cnt;
                } else {
                    char cs[32];
                    itoa(cnt, cs);
                    read += string(cs);
                    read.push_back(ans[i - 1]);
                    cnt = 1;
                }
            }
            ans = read;
        }
        return ans;
    }
private:
    int itoa(int n, char s[]) {
        if (n < 10) {
            s[0] = n + '0';
            s[1] = 0;
            return 1;
        }
        int len = itoa(n / 10, s);
        s[len++] = n % 10 + '0';
        s[len] = 0;
        return len;
    }
};
```

1.33 Decode Ways

Problem Description

A message containing letters from **A-Z** is being encoded to numbers using the following mapping:
Given an encoded message containing digits, determine the total number of ways to decode it.
For example,
Given encoded message **"12"**, it could be decoded as **"AB"** (1 2) or **"L"** (12).
The number of ways decoding **"12"** is 2.

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Solution

水题

Code

```
class Solution {
public:
    int numDecodings(string s) {
        if (s.size() == 0) return 0;
        vector<int> f(s.size() + 1, 0);
        f[0] = 1;
        for (int i = 0; i < s.size(); ++i) {
            if ('1' <= s[i] && s[i] <= '9') {
                f[i + 1] += f[i];
            }
            if (i) {
                int p = (s[i - 1] - '0') * 10 + s[i] - '0';
                if (10 <= p && p <= 26) {
                    f[i + 1] += f[i - 1];
                }
            }
        }
        return f[s.size()];
    }
};
```

1.34 Distinct Subsequences

Problem Description

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

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).

Here is an example:

S = "rabbbit", **T = "rabbit"**

Return **3**.

Solution

水题

Code

```
class Solution {
public:
    int numDistinct(string S, string T) {
        vector<vector<int> > f(T.size() + 1, vector<int>(S.size() + 1, 0));
        for (int j = 0; j <= S.size(); ++j) {
            f[0][j] = 1;
        }
        for (int i = 1; i <= T.size(); ++i) {
            for (int j = 1; j <= S.size(); ++j) {
                f[i][j] = f[i][j - 1];
                if (T[i - 1] == S[j - 1]) {
                    f[i][j] += f[i - 1][j - 1];
                }
            }
        }
        return f[T.size()][S.size()];
    }
};
```

1.35 Divide Two Integers

Problem Description

Divide two integers without using multiplication, division and mod operator.

If it is overflow, return MAX_INT.

Solution

水题

Code

```
class Solution {
public:
    int divide(int dividend, int divisor) {
        long long remind, d1 = dividend, d2 = divisor;
        if (d2 == 0) return 0;
        if (d1 < 0 && d2 < 0) {
            return _divide(-d1, -d2, remind);
        }
        if (d1 < 0 && d2 > 0) {
            return -_divide(-d1, d2, remind);
        }
        if (d1 >= 0 && d2 < 0) {
            return -_divide(d1, -d2, remind);
        }
        if (d1 >= 0 && d2 > 0) {
            return _divide(d1, d2, remind);
        }
    }
private:
    long long _divide(long long dividend, long long divisor, long long &
        remind) {
        if (dividend < divisor) {
            remind = dividend;
            return 0;
        }
        long long ans = _divide(dividend >> 1, divisor, remind) << 1;
        remind = (remind << 1) | (dividend & 1);
        if (remind >= divisor) {
            remind -= divisor;
            ++ans;
        }
        return ans;
    }
};
```

1.36 Dungeon Game

Problem Description

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

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

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

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

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

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

Notes:

- 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.

Credits:

Special thanks to @stellari for adding this problem and creating all test cases.

Solution

水题

Code

```
class Solution {
public:
    int calculateMinimumHP(vector<vector<int> > &dungeon) {
        if (dungeon.size() == 0 || dungeon[0].size() == 0) return 0;
        int n = dungeon.size(), m = dungeon[0].size();
        for (int j = m - 2; j >= 0; --j) dungeon[n - 1][j] += min(0, dungeon[
            n - 1][j + 1]);
        for (int i = n - 2; i >= 0; --i) {
            dungeon[i][m - 1] += min(0, dungeon[i + 1][m - 1]);
            for (int j = m - 2; j >= 0; --j) {
                dungeon[i][j] += min(0, max(dungeon[i + 1][j], dungeon[i][j +
                    1]));
            }
        }
        return max(-dungeon[0][0], 0) + 1;
    }
};
```

1.37 Edit Distance

Problem Description

Given two words *word1* and *word2*, find the minimum number of steps required to convert *word1* to *word2*. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

a) Insert a character

b) Delete a character

c) Replace a character

Solution

水题

Code

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

1.38 Evaluate Reverse Polish Notation

Problem Description

Evaluate the value of an arithmetic expression in Reverse Polish Notation.
Valid operators are +, -, *, /. Each operand may be an integer or another expression.
Some examples:

```
["2", "1", "+", "3", "*"] -$>$ ((2 + 1) * 3) -$>$ 9
["4", "13", "5", "/", "+"] -$>$ (4 + (13 / 5)) -$>$ 6
```

Solution

水题

Code

```
class Solution {
public:
    int evalRPN(vector<string> &tokens) {
        stack<int> numbers;
        int a, b;
        for (int i = 0; i < tokens.size(); ++i) {
            if (tokens[i] == "+") {
                getab(numbers, a, b);
                numbers.push(a + b);
            } else if (tokens[i] == "-") {
                getab(numbers, a, b);
                numbers.push(a - b);
            } else  if (tokens[i] == "*") {
                getab(numbers, a, b);
                numbers.push(a * b);
            } else  if (tokens[i] == "/") {
                getab(numbers, a, b);
                numbers.push(a / b);
            } else {
                numbers.push(atoi(tokens[i].c_str()));
            }
        }
        return numbers.top();
    }
private:
    void getab(stack<int> &numbers, int &a, int &b) {
        b = numbers.top();
        numbers.pop();
        a = numbers.top();
        numbers.pop();
    }
};
```

1.39 Excel Sheet Column Number

Problem Description

Related to question Excel Sheet Column Title
Given a column title as appear in an Excel sheet, return its corresponding column number.
For example:
Credits:
Special thanks to @ts for adding this problem and creating all test cases.

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

Solution

水题

Code

```
class Solution {
public:
    int titleToNumber(string s) {
        int result = 0;
        for (size_t i = 0; i < s.size(); ++i) {
            result = result * 26 + (s[i] - 'A') + 1;
        }
        return result;
    }
};
```

1.40 Excel Sheet Column Title

Problem Description

Given a positive integer, return its corresponding column title as appear in an Excel sheet.
For example:
Credits:
Special thanks to @ifanchu for adding this problem and creating all test cases.

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

Solution

水题

Code

```
class Solution {
public:
    string convertToTitle(int n) {
        string result;
        for (; n; n = (n - 1) / 26) {
            result += char('A' + (n - 1) % 26);
        }
        reverse(result.begin(), result.end());
        return result;
    }
};
```

1.41 Factorial Trailing Zeroes

Problem Description

Given an integer n, return the number of trailing zeroes in $n!$.
Note: Your solution should be in logarithmic time complexity.
Credits:
Special thanks to @ts for adding this problem and creating all test cases.

Solution

水题

Code

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

1.42 Find Minimum in Rotated Sorted Array II

Problem Description

Follow up for "Find Minimum in Rotated Sorted Array":
What if *duplicates* are allowed?
Would this affect the run-time complexity? How and why?
Suppose a sorted array 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.

Solution

水题

Code

```
class Solution {
public:
    int findMin(const vector<int>& num) {
        return findMin(num, 0, int(num.size()) - 1);
    }
private:
    int findMin(const vector<int>& num, int left, int right) {
        if (left == right) return num[left];
        int mid = (left + right) / 2;
        if (num[left] > num[mid] || num[mid] < num[right]) return findMin(num
            , left, mid);
        if (num[mid] > num[right]) return findMin(num, mid + 1, right);
        return min(findMin(num, left, mid), findMin(num, mid + 1, right));
    }
};
```

1.43 Find Minimum in Rotated Sorted Array

Problem Description

Suppose a sorted array 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.

Solution

水题

Code

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

1.44 Find Peak Element

Problem Description

A peak element is an element that is greater than its neighbors.
Given an input array where **num[i]** $\neq$**num[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 **num[-1]** = **num[n]** = **-∞**.

For example, in array **[1, 2, 3, 1]**, 3 is a peak element and your function should return the index number 2.

Your solution should be in logarithmic complexity.

Credits:

Special thanks to @ts for adding this problem and creating all test cases.

Solution

水题

Code

```
class Solution {
public:
    int findPeakElement(const vector<int> &num) {
        int left = 0, right = num.size() - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            if ((mid - 1 < left || num[mid - 1] < num[mid]) &&
                (mid + 1 > right || num[mid + 1] < num[mid])) {
                return mid;
            }
            if (mid - 1 >= left && num[mid - 1] > num[mid]) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
        return left;
    }
};
```

1.45 First Missing Positive

Problem Description

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

For example,

Given **[1,2,0]** return **3**,

and **[3,4,-1,1]** return **2**.

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

Solution

水题

Code

```
class Solution {
public:
    int firstMissingPositive(int A[], int n) {
        if (n == 0) return 1;
        for (int i = 0, temp; i < n; ++i) {
            while(0 <= A[i] && A[i] < n && A[i] != i && A[i] != A[A[i]]) {
                swap(A[i], A[A[i]]);
            }
        }
        for (int i = 1; i < n; ++i) {
            if (A[i] != i) {
                return i;
            }
        }
        return n + (A[0] == n);
    }
private:
    void swap(int &a, int &b) {
        a ^= b ^= a ^= b;
    }
};
```

1.46 Flatten Binary Tree to Linked List

Problem Description

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

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

If you notice carefully in the flattened tree, each node's right child points to the next node of a pre-order traversal.

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

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution {
```

```
public:
    void flatten(TreeNode *root) {
        TreeNode *begin, *end;
        _flatten(root, begin, end);
    }
private:
    void _flatten(TreeNode *root, TreeNode * &begin, TreeNode * &end) {
        if (root == NULL) {
            begin = end = NULL;
            return;
        }
        TreeNode *lbegin, *lend, *rbegin, *rend;
        _flatten(root->left, lbegin, lend);
        _flatten(root->right, rbegin, rend);
        root->left = NULL;
        begin = end = root;
        if (lbegin) {
            end->right = lbegin;
            end = lend;
        }
        if (rbegin) {
            end->right = rbegin;
            end = rend;
        }
    }
};
```

1.47 Fraction to Recurring Decimal

Problem Description

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.

For example,

- Given numerator = 1, denominator = 2, return "0.5".
- Given numerator = 2, denominator = 1, return "2".
- Given numerator = 2, denominator = 3, return "0.(6)".

Credits:
Special thanks to @Shangrila for adding this problem and creating all test cases.

Solution

水题

Code

```
class Solution {
public:
    string fractionToDecimal(int numerator, int denominator) {
        if (numerator < 0 && denominator < 0) {
            return fractionToDecimalU(-numerator, -denominator);
        }
        if (numerator < 0 && denominator > 0) {
            return "-" + fractionToDecimalU(-numerator, denominator);
        }
        if (numerator > 0 && denominator < 0) {
            return "-" + fractionToDecimalU(numerator, -denominator);
        }
        return fractionToDecimalU(numerator, denominator);
    }

private:
    string fractionToDecimalU(unsigned numerator, unsigned denominator) {
        unsigned integer = numerator / denominator;
        numerator %= denominator;
        std::string result = std::to_string(integer);
        if (numerator == 0) return result;
        std::vector<unsigned> bits;
        std::map<unsigned, size_t> remainder;
        while (numerator && remainder.find(numerator) == remainder.end()) {
            remainder[numerator] = bits.size();
            bits.push_back(unsigned((unsigned long long)numerator * 10 /
                denominator));
            numerator = unsigned((unsigned long long)numerator * 10 %
                denominator);
        }
        result += '.';
        for (size_t i = 0; i < bits.size(); ++i) {
            if (numerator && i == remainder[numerator]) {
                result += '(';
            }
            result += char('0' + bits[i]);
        }
        if (numerator) result += ')';
        return result;
    }
};
```

1.48 Gas Station

Problem Description

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, otherwise return -1.

Note: The solution is guaranteed to be unique.

Solution

水题

Code

```
class Solution {
public:
    int canCompleteCircuit(vector<int> &gas, vector<int> &cost) {
        int n = gas.size(), left = 0, right = 0;
        vector<int> leftgas(n * 2), que(n * 2);
        leftgas[0] = 0;
        for (int i = 1; i < leftgas.size(); ++i) {
            leftgas[i] = leftgas[i - 1] + gas[(i - 1) % n] - cost[(i - 1) % n
                ];
        }
        for (int i = 0; i < n; ++i) {
            while (left < right && leftgas[que[right - 1]] >= leftgas[i]) {
                -- right;
            }
            que[right++] = i;
        }
        for (int i = n; i < leftgas.size(); ++i) {
            if (que[left] == i - n) ++left;
            while (left < right && leftgas[que[right - 1]] >= leftgas[i]) {
                -- right;
            }
            que[right++] = i;
            if (leftgas[que[left]] >= leftgas[i - n]) {
                return i - n;
            }
        }
        return -1;
    }
};s
```

1.49 Generate Parentheses

Problem Description

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given $n = 3$, a solution set is:

"((()))", "(()())", "(())()", "()(())", "()()()"

Solution

水题

Code

```
class Solution {
public:
    vector<string> generateParenthesis(int n) {
        vector<string> ans;
        if (n <= 0) return ans;
        vector<vector<string> > ansSet(n + 1);
        ansSet[0].push_back("");
        for (int k = 1; k <= n; ++k) {
            for (int i = 0; i < ansSet[k - 1].size(); ++i) {
                ansSet[k].push_back("(" + ansSet[k - 1][i] + ")");
            }
            for (int i = 1; i < k; ++i) {
                for (int p = 0; p < ansSet[i - 1].size(); ++p) {
                    for (int q = 0; q < ansSet[k - i].size(); ++q) {
                        ansSet[k].push_back("(" + ansSet[i - 1][p] + ")" +
                            ansSet[k - i][q]);
                    }
                }
            }
        }
        return ansSet[n];
    }
};
```

1.50 Gray Code

Problem Description

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.

For example, given $n = 2$, return **[0,1,3,2]**. Its gray code sequence is:

Note:

For a given n, a gray code sequence is not uniquely defined.

For example, **[0,2,3,1]** is also a valid gray code sequence according to the above definition.

For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

```
00 - 0
01 - 1
11 - 3
10 - 2
```

Solution

水题

Code

```
class Solution {
public:
    vector<int> grayCode(int n) {
        vector<int> ans;
        ans.push_back(0);
        if (n <= 0) return ans;
        ans.push_back(1);
        for (int k = 2; k <= n; ++k) {
            vector<int> extended;
            for (int i = 0; i < ans.size(); ++i) {
                if (i % 2 == 0) {
                    extended.push_back(ans[i] << 1);
                    extended.push_back(ans[i] << 1 | 1);
                } else {
                    extended.push_back(ans[i] << 1 | 1);
                    extended.push_back(ans[i] << 1);
                }
            }
            ans = extended;
        }
        return ans;
    }
};
```

1.51 Implement strStr()

Problem Description

Solution

水题

Code

```cpp
class Solution {
public:
    char *strStr(char *s, char *t) {
        int n = strlen(t);
        if (n == 0) return s;
        int next[n], pos = 0;
        next[0] = pos;
        for (int i = 1; i < n; ++i) {
            while(pos && t[next[pos]] != t[i]) {
                pos = next[pos - 1];
            }
            next[i] = pos = pos + (t[pos] == t[i]);
        }
        pos = 0;
        for (int i = 0; s[i]; ++i) {
            while (pos && t[pos] != s[i]) {
                pos = next[pos - 1];
            }
            pos += t[pos] == s[i];
            if (pos == n) return s + i - n + 1;
        }
        return NULL;
    }
};
```

1.52 Insert Interval

Problem Description

Given a set of *non-overlapping* intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1:

Given intervals **[1,3]**,**[6,9]**, insert and merge **[2,5]** in as **[1,5]**,**[6,9]**.

Example 2:

Given **[1,2]**,**[3,5]**,**[6,7]**,**[8,10]**,**[12,16]**, insert and merge **[4,9]** in as **[1,2]**,**[3,10]**,**[12,16]**.

This is because the new interval **[4,9]** overlaps with **[3,5]**,**[6,7]**,**[8,10]**.

Solution

水题

Code

```cpp
/**
 * Definition for an interval.
 * struct Interval {
 *     int start;
 *     int end;
 *     Interval() : start(0), end(0) {}
 *     Interval(int s, int e) : start(s), end(e) {}
 * };
 */
class Solution {
public:
    vector<Interval> insert(vector<Interval> &intervals, Interval newInterval
        ) {
```

```
        vector<Interval> ans;
        bool newin = false;
        for (int i = 0; i < intervals.size(); ++i) {
            if (intervals[i].end < newInterval.start) {
                ans.push_back(intervals[i]);
            } else if (intervals[i].start > newInterval.end) {
                if (!newin) {
                    ans.push_back(newInterval);
                    newin = true;
                }
                ans.push_back(intervals[i]);
            } else {
                newInterval.start = min(newInterval.start, intervals[i].start
                    );
                newInterval.end = max(newInterval.end, intervals[i].end);
            }
        }
        if (!newin) ans.push_back(newInterval);
        return ans;
    }
};
```

1.53 Insertion Sort List

Problem Description

Sort a linked list using insertion sort.

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *insertionSortList(ListNode *head) {
        ListNode *newhead = NULL, *p = head, *q;
        while (p) {
            q = p;
            p = p->next;
            insert(newhead, q);
        }
        return newhead;
    }

private:
    void insert(ListNode* &head, ListNode *node) {
        node->next = NULL;
        if (head == NULL) {
            head = node;
        } else if (node->val < head->val) {
            node->next = head;
            head = node;
        } else {
            for (ListNode *p = head; p; p = p->next) {
                if (p->next == NULL || p->next->val >= node->val) {
                    node->next = p->next;
                    p->next = node;
                    return ;
                }
            }
        }
    }
};
```

1.54 Integer to Roman

Problem Description

Given an integer, convert it to a roman numeral.
Input is guaranteed to be within the range from 1 to 3999.

Solution

水题

Code

```
class Solution {
public:
    string intToRoman(int num) {
        char r1 = 'I', r5 = 'V', r10 = 'X', r50 = 'L', r100 = 'C', r500 = 'D'
            , r1000 = 'M';
        string roman;
        roman += digital2roman(num / 1000, r1000, '␣', '␣');
        roman += digital2roman(num % 1000 / 100, r100, r500, r1000);
        roman += digital2roman(num % 100 / 10, r10, r50, r100);
        roman += digital2roman(num % 10, r1, r5, r10);
        return roman;
    }

private:
    string digital2roman(int x, char c1, char c5, char c10) {
        string roman;
        if (1 <= x && x <= 3) {
            switch (x) {
                case 3:
                    roman.push_back(c1);
                case 2:
                    roman.push_back(c1);
                case 1:
                    roman.push_back(c1);
            }
        } else if (x == 4) {
            roman.push_back(c1);
            roman.push_back(c5);
        } else if (5 <= x && x <= 8) {
            roman.push_back(c5);
            switch (x) {
                case 8:
                    roman.push_back(c1);
                case 7:
                    roman.push_back(c1);
                case 6 :
                    roman.push_back(c1);
            }
        } else if (x == 9) {
            roman.push_back(c1);
            roman.push_back(c10);
        }
        return roman;
    }
};
```

1.55 Interleaving String

Problem Description

Given *s1*, *s2*, *s3*, find whether *s3* is formed by the interleaving of *s1* and *s2*.
For example,
Given:
s1 = **"aabcc"**,
s2 = **"dbbca"**,
When *s3* = **"aadbbcbcac"**, return true.
When *s3* = **"aadbbbaccc"**, return false.

Solution

水题

Code

```
class Solution {
public:
    bool isInterleave(string s1, string s2, string s3) {
        if (s1.size() + s2.size() != s3.size()) return false;
        vector<vector<bool> > f(s1.size() + 1, vector<bool>(s2.size() + 1,
            false));
        f[0][0] = true;
        for (int i = 0; i <= s1.size(); ++i) {
            for (int j = 0; j <= s2.size(); ++j) {
                if (f[i][j]) {
                    if (i != s1.size() && s1[i] == s3[i + j]) {
                        f[i + 1][j] = true;
                    }
                    if (j != s2.size() && s2[j] == s3[i + j]) {
                        f[i][j + 1] = true;
                    }
                }
            }
        }
        return f[s1.size()][s2.size()];
    }
};
```

1.56 Intersection of Two Linked Lists

Problem Description

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.
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.

Credits:
Special thanks to @stellari for adding this problem and creating all test cases.

```
A:          a1 → a2

                     c1 → c2 → c3

B:     b1 → b2 → b3
```

Solution

水题

Code

```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) {
        size_t lena = ListSize(headA);
        size_t lenb = ListSize(headB);
        while (lena && lenb) {
            if (headA == headB) return headA;
            if (lena >= lenb) {
                -- lena;
                headA = headA->next;
            } else {
                -- lenb;
                headB = headB->next;
            }
        }
        return nullptr;
    }
private:
    size_t ListSize(ListNode *head) {
        size_t size = 0;
        while (head) {
            ++ size;
            head = head->next;
        }
        return size;
    }
};
```

1.57 Jump Game II

Problem Description

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

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

Your goal is to reach the last index in the minimum number of jumps.

For example:

Given array A = **[2,3,1,1,4]**

The minimum number of jumps to reach the last index is **2**. (Jump **1** step from index 0 to 1, then **3** steps to the last index.)

Solution

水题

Code

```
class Solution {
public:
    int jump(int A[], int n) {
        priority_queue<Point> que;
        Point last;
        que.push(Point());
        for (int i = 0; i < n; ++i) {
            last = que.top();
            while (last.x < i) {
                que.pop();
                last = que.top();
            }
            if (i == n - 1) return last.value;
            que.push(Point(i + A[i], last.value + 1));
        }
    }
private:
    struct Point {
        int x, value;
        Point(): x(0), value(0){};
        Point(int x, int value):x(x), value(value){};
        bool operator<(const Point &p) const {
            return value > p.value;
        }
    };
};
```

1.58 Jump Game

Problem Description

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

Each element in the array represents your maximum jump length at that position.
Determine if you are able to reach the last index.
For example:
A = [**2,3,1,1,4**], return **true**.
A = [**3,2,1,0,4**], return **false**.

Solution

水题

Code

```cpp
class Solution {
public:
    bool canJump(int A[], int n) {
        if (n <= 1) return true;
        int maxJump = A[0];
        for (int i = 0; i < n; ++i) {
            if (maxJump < i) return false;
            maxJump = max(maxJump, A[i] + i);
        }
        return true;
    }
};
```

1.59 Largest Number

Problem Description

Given a list of non negative integers, arrange them such that they form the largest number.
For example, given [**3, 30, 34, 5, 9**], the largest formed number is **9534330**.
Note: The result may be very large, so you need to return a string instead of an integer.
Credits:
Special thanks to @ts for adding this problem and creating all test cases.

Solution

水题

Code

```cpp
class Solution {
public:
    static bool cmp(const string& a, const string& b) {
        return a + b > b + a;
    }

    string largestNumber(vector<int> &num) {
        vector<string> numstr(num.size());
        for (size_t i = 0; i < num.size(); ++i) {
            numstr[i] = std::to_string(num[i]);
        }
        std::sort(numstr.begin(), numstr.end(), cmp);
        string result;
        size_t i = 0;
        while (i < numstr.size() && numstr[i] == "0") ++i;
        for (; i < numstr.size(); ++i) {
            result += numstr[i];
        }
        return result == ""? "0": result;
    }
};
```

1.60 Largest Rectangle in Histogram

Problem Description

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

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

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

For example,

Given height = **[2,1,5,6,2,3]**,

return **10**.

Solution

水题

Code

```
class Solution {
public:
    int largestRectangleArea(vector<int> &height) {
        int area = 0, n = height.size();
        int h[n + 2];
        for (int i = 1; i <= n; ++i) {
            h[i] = height[i - 1];
        }
        h[0] = h[n + 1] = -1;

        vector<int> less(n + 2);
        stack<int> min1, min2;
        min2.push(n + 1);
        for (int i = n; i; --i) {
            while(h[min2.top()] >= h[i]) {
                min2.pop();
            }
            less[i] = min2.top();
            min2.push(i);
        }
        min1.push(0);
        for (int i = 1; i <= n; ++i) {
            while(h[min1.top()] >= h[i]) {
                min1.pop();
            }
            area = max(area, h[i] * (less[i] - min1.top() - 1));
            min1.push(i);
        }
        return area;
    }
};
```

1.61 Length of Last Word

Problem Description

Given a string s consists of upper/lower-case alphabets and empty space characters **' '**, return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example,

Given s = **"Hello World"**,

return **5**.

Solution

水题

Code

```
class Solution {
public:
    int lengthOfLastWord(const char *s) {
        int lastlen = 0;
        for(int i = 0; s[i]; ++i) {
            if (s[i] != '␣') {
                int p = i + 1;
                while (s[p] && s[p] != '␣') {
                    ++ p;
                }
                lastlen = p - i;
                i = p - 1;
            }
        }
        return lastlen;
    }
};
```

1.62 Letter Combinations of a Phone Number

Problem Description

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.

Note:

Although the above answer is in lexicographical order, your answer could be in any order you want.

```
<b>Input:</b>Digit string "23"
<b>Output:</b> ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

Solution

水题

Code

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        string keys[] = {"␣", "1", "abc", "def", "ghi", "jkl", "mno", "pqrs",
             "tuv", "wxyz"};
        vector<string> ans;
        string let = "";
        _gen(digits, 0, ans, let, keys);
        return ans;
    }
private:
    void _gen(string &digits, int k, vector<string> &ans, string &let, string
         keys[]) {
        if (k >= digits.size()) {
            ans.push_back(let);
            return;
        }
        for (int i = 0; i < keys[digits[k] - '0'].size(); ++i) {
            let.push_back(keys[digits[k] - '0'][i]);
            _gen(digits, k + 1, ans, let, keys);
            let.pop_back();
        }
    }
};
```

1.63 Linked List Cycle II

Problem Description

Given a linked list, return the node where the cycle begins. If there is no cycle, return **null**.
Follow up: Can you solve it without using extra space?

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        set<ListNode*> pset;
        for (ListNode *p = head; p; p = p->next) {
            if (pset.find(p) != pset.end()) {
                return p;
            }
            pset.insert(p);
        }
        return NULL;
    }
};
```

1.64 Linked List Cycle

Problem Description

Given a linked list, determine if it has a cycle in it.
Follow up:
Can you solve it without using extra space?

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        set<ListNode*> pset;
        for (ListNode *p = head; p; p = p->next) {
            if (pset.find(p) != pset.end()) {
                return true;
            }
            pset.insert(p);
        }
        return false;
    }
};
```

1.65 Longest Common Prefix

Problem Description

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

Solution

水题

Code

```
class Solution {
public:
    string longestCommonPrefix(vector<string> &strs) {
        if (strs.size() == 0) return "";
        string ans;
        for (int k = 0; k < strs[0].size(); ++k) {
            bool diff = false;
            for (int i = 1; i < strs.size(); ++i) {
                if (strs[i].size() <= k || strs[i][k] != strs[0][k]) {
                    diff = true;
                    break;
                }
            }
            if (diff) break;
            ans.push_back(strs[0][k]);
        }
        return ans;
    }
};
```

1.66 Longest Consecutive Sequence

Problem Description

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.
For example,
Given **[100, 4, 200, 1, 3, 2]**,
The longest consecutive elements sequence is **[1, 2, 3, 4]**. Return its length: **4**.
Your algorithm should run in O(n) complexity.

Solution

水题

Code

```
class Solution {
public:
    int longestConsecutive(vector<int> &num) {
        if (num.size() == 0) return 0;
        sort(num.begin(), num.end());
        num.resize(unique(num.begin(), num.end()) - num.begin());
        int res = 1, cnt = 1;
        for (int i = 1; i < num.size(); ++i) {
            if (num[i] == num[i - 1] + 1) {
                ++ cnt;
            } else {
                cnt = 1;
            }
            res = max(res, cnt);
        }
        return res;
    }
};
```

1.67 Longest Palindromic Substring

Problem Description

Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

Solution

水题

Code

```
bool f[1010][1010];

class Solution {
public:
    string longestPalindrome(string s) {
        int n = s.size();
        if (n == 0) return "";
        int lx = 0, ly = 1;
        for (int len = 0; len <= n; ++len) {
            for (int i = 0; i + len <= n; ++i) {
                int j = i + len;
                if (len == 0 || len == 1) {
                    f[i][j] = true;
                } else {
                    f[i][j] = (f[i + 1][j - 1] && s[i] == s[j - 1]);
                    if (f[i][j] && ly - lx < len) {
                        lx = i, ly = j;
                    }
                }
            }
        }
        return s.substr(lx, ly - lx);
    }
};
```

1.68 Longest Substring Without Repeating Characters

Problem Description

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

Solution

水题

Code

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        vector<int> cnt(256, 0);
        int ans = 0;
        for (int i = 0, begin = 0; i < s.size(); ++i) {
            ++ cnt[s[i]];
            while (cnt[s[i]] >= 2) {
                -- cnt[s[begin++]];
            }
            ans = max(ans, i - begin + 1);
        }
        return ans;
    }
};
```

1.69 Longest Valid Parentheses

Problem Description

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

Solution

水题

Code

```cpp
class Solution {
public:
    int longestValidParentheses(string s) {
        stack<int> stk;
        stk.push(-1);
        int ans = 0;
        for (int i = 0; i < s.size(); ++i) {
            if (stk.size() >= 2 && s[stk.top()] == '(' && s[i] == ')') {
                stk.pop();
                ans = max(ans, i - stk.top());
            } else {
                stk.push(i);
            }
        }
        return ans;
    }
};
```

1.70 LRU Cache

Problem Description

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

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

Solution

水题

Code

```cpp
struct LinkedListNode {
    LinkedListNode(): pre(nullptr), next(nullptr) { }
    int key, value;
    LinkedListNode *pre, *next;
};

class LRUCache{
public:
    LRUCache(int capacity): capacity(capacity) {
        head.next = &tail;
        tail.pre = &head;
    }

    ~LRUCache() {
        for (LinkedListNode *node = head.next; node != &tail; node = head.
            next) {
            head.next = node->next;
            delete node;
        }
    }

    int get(int key) {
        if (hashtable.find(key) != hashtable.end()) {
            LinkedListNode* node = hashtable[key];
            erase(node), push(node);
            return node->value;
        }
        return -1;
    }

    void set(int key, int value) {
        LinkedListNode* node;
        if (hashtable.find(key) != hashtable.end()) {
            node = hashtable[key];
            node->key = key, node->value = value;
            erase(node), push(node);
            return ;
        }
        if (hashtable.size() == capacity) {
            node = tail.pre;
            erase(node);
            hashtable.erase(node->key);
        } else {
            node = new LinkedListNode();
        }
        node->key = key, node->value = value;
        hashtable[key] = node;
        push(node);
    }

private:

    void push(LinkedListNode* node) {
        node->pre = &head;
        node->next = head.next;
        head.next->pre = node;
        head.next = node;
    }

    void erase(LinkedListNode* node) {
        node->pre->next = node->next;
        node->next->pre = node->pre;
    }

    LinkedListNode head, tail;
    unordered_map<int, LinkedListNode*> hashtable;
    int capacity;
```

```
};
```

1.71 Majority Element

Problem Description

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

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

Credits:

Special thanks to @ts for adding this problem and creating all test cases.

Solution

水题

Code

```
class Solution {
public:
    int majorityElement(vector<int> &num) {
        std::stack<int> stk;
        for (size_t i = 0; i < num.size(); ++i) {
            if (!stk.empty() && stk.top() != num[i]) {
                stk.pop();
            } else {
                stk.push(num[i]);
            }
        }
        return stk.top();
    }
};
```

1.72 Max Points on a Line

Problem Description

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

Solution

水题

Code

```
/**
 * Definition for a point.
 * struct Point {
 *     int x;
 *     int y;
 *     Point() : x(0), y(0) {}
 *     Point(int a, int b) : x(a), y(b) {}
 * };
 */
class Solution {
public:
    int maxPoints(vector<Point> &points) {
        int ans = min(2, (int)points.size());
        for (int i = 0; i < points.size(); ++i) {
            for (int j = i + 1; j < points.size(); ++j) {
                ans = max(ans, pointsInLine(points, points[i], points[j]));
            }
        }
        return ans;
    }

private:
    int pointsInLine(const vector<Point> &points, const Point &p1, const
       Point &p2) {
        int count = 0;
        for (int i = 0; i < points.size(); ++i) {
            if (!equal(p1, p2) && inLine(p1, p2, points[i]) || equal(p1, p2)
                && equal(points[i], p1)) {
                ++ count;
            }
        }
        return count;
    }

    bool equal(const Point &p1, const Point &p2) {
        return p1.x == p2.x && p1.y == p2.y;
    }

    bool inLine(const Point &p1, const Point &p2, const Point &p3) {
        return (p2.y - p1.y) * (p3.x - p1.x) == (p2.x - p1.x) * (p3.y - p1.y)
            ;
    }
};
```

1.73 Maximal Rectangle

Problem Description

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

Solution

水题

Code

```
class Solution {
public:
    int maximalRectangle(vector<vector<char> > &matrix) {
        if (matrix.size() == 0) return 0;
        int n = matrix.size(), m = matrix[0].size();
        vector<vector<int> > len(n, vector<int> (m + 2));
        for (int j = 0; j < m; ++j) {
            for (int i = n - 1; i >= 0; --i) {
                len[i][j + 1] = (matrix[i][j] == '0'? 0: (i == n - 1? 1: len[
                    i + 1][j + 1] + 1));
            }
        }
        int area = 0;
        for (int k = 0; k < n; ++k) {
            len[k][0] = len[k][m + 1] = -1;
            vector<int> &height = len[k], less(m + 2);
            stack<int> min1, min2;
            min2.push(m + 1);
            for (int i = m; i; --i) {
                while(height[min2.top()] >= height[i]) {
                    min2.pop();
                }
                less[i] = min2.top();
                min2.push(i);
            }
            min1.push(0);
            for (int i = 1; i <= m; ++i) {
                while(height[min1.top()] >= height[i]) {
                    min1.pop();
                }
                area = max(area, height[i] * (less[i] - min1.top() - 1));
                min1.push(i);
            }
        }
        return area;
    }
};
```

1.74 Maximum Depth of Binary Tree

Problem Description

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.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode *root) {
        if (root == NULL) return 0;
        return max(maxDepth(root->left), maxDepth(root->right)) + 1;
    }
};
```

1.75 Maximum Gap

Problem Description

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

Try to solve it in linear time/space.

Return 0 if the array contains less than 2 elements.

You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range.

Credits:

Special thanks to @porker2008 for adding this problem and creating all test cases.

Solution

水题

Code

```cpp
class Solution {
public:
    int maximumGap(vector<int> &num) {
        int n = int(num.size());
        if (n <= 1) return 0;
        int maxvalue = *(max_element(num.begin(), num.end()));
        int minvalue = *(min_element(num.begin(), num.end()));
        vector<pair<int, int>> interval(n, make_pair(maxvalue, -1));
        int valsize = (maxvalue - minvalue + 1) / int(num.size()) + 1;
        for (int i = 0; i < n; ++i) {
            int idx = (num[i] - minvalue) / valsize;
            interval[idx].first = min(interval[idx].first, num[i]);
            interval[idx].second = max(interval[idx].second, num[i]);
        }
        int result = 0, x = interval[0].first;
        for (int i = 0; i < n; ++i) {
            if (interval[i].second != -1) {
                result = max(result, interval[i].first - x);
                x = interval[i].second;
            }
        }
        return result;
    }
};
```

1.76 Maximum Product Subarray

Problem Description

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array **[2,3,-2,4]**,

the contiguous subarray **[2,3]** has the largest product = **6**.

Solution

水题

Code

```cpp
class Solution {
public:
    int maxProduct(int A[], int n) {
        if (n <= 0) return 0;
        int maxa = A[0], mina = A[0], maxp = A[0];
        for (int i = 1; i < n; ++i) {
            int maxt = maxa * A[i];
            int mint = mina * A[i];
            maxa = std::max(std::max(maxt, mint), A[i]);
            mina = std::min(std::min(maxt, mint), A[i]);
            maxp = std::max(maxa, maxp);
        }
        return maxp;
    }
};
```

1.77 Maximum Subarray

Problem Description

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array **[2,1,3,4,1,2,1,5,4]**,

the contiguous subarray **[4,1,2,1]** has the largest sum = **6**.

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

Solution

水题

Code

```
class Solution {
public:
    int maxSubArray(int A[], int n) {
        if (n <= 0) return 0;
        int ans = A[0], sum = A[0];
        for (int i = 1; i < n; ++i) {
            sum = max(sum, 0) + A[i];
            ans = max(ans, sum);
        }
        return ans;
    }
};
```

1.78 Median of Two Sorted Arrays

Problem Description

There are two sorted arrays A and B of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Solution

水题

Code

```

class Solution {
public:
    double findMedianSortedArrays(int A[], int m, int B[], int n) {
        double ans = findKth(A, m, B, n, (n + m) / 2 + 1);
        if ((n + m) % 2 == 0) {
            ans = ans * 0.5 + findKth(A, m, B, n, (n + m) / 2) * 0.5;
        }
        return ans;
    }
private:
    int findKth(int A[], int m, int B[], int n, int k) {
        if (n == 0) return A[k - 1];
        if (m == 0) return B[k - 1];
        int mid = (m - 1) / 2;
        int pos = lower_bound(B, B + n, A[mid]) - B;
        if (mid + 1 + pos > k) {
            return findKth(A, mid, B, pos, k);
        }
        if (mid + 1 + pos < k) {
            return findKth(A + mid + 1, m - mid - 1, B + pos, n - pos, k -
                mid - 1 - pos);
        }
        return A[mid];
    }
};
```

1.79 Merge Intervals

Problem Description

Given a collection of intervals, merge all overlapping intervals.
For example,
Given **[1,3]**,**[2,6]**,**[8,10]**,**[15,18]**,
return **[1,6]**,**[8,10]**,**[15,18]**.

Solution

水题

Code

```
/**
 * Definition for an interval.
 * struct Interval {
 *     int start;
 *     int end;
 *     Interval() : start(0), end(0) {}
 *     Interval(int s, int e) : start(s), end(e) {}
 * };
 */

class Solution {
public:
    vector<Interval> merge(vector<Interval> &intervals) {
        vector<Interval> ans;
        if (intervals.size() == 0) return ans;
        sort(intervals.begin(), intervals.end(), cmp);
        Interval node(intervals[0].start, intervals[0].end);
        for (int i = 1; i < intervals.size(); ++i) {
            if (node.end < intervals[i].start) {
                ans.push_back(node);
                node = intervals[i];
            } else {
                node.end = max(node.end, intervals[i].end);
            }
        }
        ans.push_back(node);
        return ans;
    }
private:
    static bool cmp(const Interval &a, const Interval &b) {
        return a.start < b.start;
    }
};
```

1.80 Merge k Sorted Lists

Problem Description

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

Solution

水题

Code

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

struct Pointer {
    ListNode *p;
    Pointer() {}
    Pointer(ListNode *p): p(p) {}
    bool operator < (const Pointer &other) const {
        if (p && other.p) return p->val > other.p->val;
        return p < other.p;
    }
};

class Solution {
public:
    ListNode *mergeKLists(vector<ListNode *> &lists) {
        priority_queue<Pointer> que;
        ListNode head(0), *tail;
        tail = &head;
        for (int i = 0; i < lists.size(); ++i) {
            que.push(Pointer(lists[i]));
        }
        while(!que.empty()) {
            Pointer now = que.top();
            que.pop();
            if (now.p) {
                tail->next = now.p;
                tail = tail->next;
                que.push((now.p)->next);
            }
        }
        return head.next;
    }
};
```

1.81 Merge Sorted Array

Problem Description

Given two sorted integer arrays A and B, merge B into A as one sorted array.
Note:
You may assume that A has enough space (size that is greater or equal to $m + n$) to hold additional elements from B. The number of elements initialized in A and B are m and n respectively.

Solution

水题

Code

```
class Solution {
public:
    void merge(int A[], int m, int B[], int n) {
        for (int i = m + n - 1; i >= 0; --i) {
            int k;
            if (m && n) {
                if (A[m - 1] >= B[n - 1]) {
                    k = A[--m];
                } else {
                    k = B[--n];
                }
            } else if (m && !n) {
                k = A[--m];
            } else if (!m && n) {
                k = B[--n];
            }
            A[i] = k;
        }
    }
};
```

1.82 Merge Two Sorted Lists

Problem Description

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *mergeTwoLists(ListNode *l1, ListNode *l2) {
        ListNode head(0), *p;
        p = &head;
        while (l1 || l2) {
            if (l1 && l2) {
                if (l1->val < l2->val) {
                    p->next = l1, l1 = l1->next;
                } else {
                    p->next = l2, l2 = l2->next;
                }
            } else if (!l1 && l2) {
                p->next = l2, l2 = l2->next;
            } else if (l1 && !l2) {
                p->next = l1, l1 = l1->next;
            }
            p = p->next;
        }
        return head.next;
    }
};
```

1.83 Min Stack

Problem Description

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.

Solution

水题

Code

```
class MinStack {
public:
    void push(int x) {
        if (stk.size() == 0 || x <= minvalue) {
            stk.push(minvalue);
            minvalue = x;
        }
        stk.push(x);
    }

    void pop() {
        int popvalue = stk.top();
        stk.pop();
        if (popvalue == minvalue) {
            minvalue = stk.top();
            stk.pop();
        }
    }

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

    int getMin() {
        return minvalue;
    }

private:
    stack<int> stk;
    int minvalue;
};
```

1.84 Minimum Depth of Binary Tree

Problem Description

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.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int minDepth(TreeNode *root) {
        if (root == NULL) return 0;
        if (root->left == NULL && root->right == NULL) return 1;
        int ans = LONG_MAX;
        if (root->left) {
            ans = min(ans, minDepth(root->left) + 1);
        }
        if (root->right) {
            ans = min(ans, minDepth(root->right) + 1);
        }
        return ans;
    }
};
```

1.85 Minimum Path Sum

Problem Description

Given a *m* x *n* grid filled with non-negative numbers, find a path from top left to bottom right which *minimizes* the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Solution

水题

Code

```
class Solution {
public:
    int minPathSum(vector<vector<int> > &grid) {
        int n = grid.size(), m = grid[0].size();
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (i && j) {
                    grid[i][j] += min(grid[i - 1][j], grid[i][j - 1]);
                } else if (!i && j) {
                    grid[i][j] += grid[i][j - 1];
                } else if (i && !j) {
                    grid[i][j] += grid[i - 1][j];
                }
            }
        }
        return grid[n - 1][m - 1];
    }
};
```

1.86 Minimum Window Substring

Problem Description

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

For example,

S = "ADOBECODEBANC"

T = "ABC"

Minimum window is **"BANC"**.

Note:

If there is no such window in S that covers all characters in T, return the emtpy string "".

If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

Solution

水题

Code

```
class Solution {
public:
    string minWindow(string S, string T) {
        vector<int> cnt(256, 0);
        int cnttype = 0;
        for (int i = 0; i < T.size(); ++i) {
            if (cnt[T[i]] == 0) {
                -- cnttype;
            }
            -- cnt[T[i]];
        }
        int ansx = 0, anslen = S.size() + 1, x = 0;
        for (int i = 0; i < S.size(); ++i) {
            ++ cnt[S[i]];
            if (cnt[S[i]] == 0) {
                ++ cnttype;
            }
            while(cnttype == 0) {
                if (cnt[S[x]] == 0) {
                    -- cnttype;
                }
                if (anslen > i - x) {
                    ansx = x;
                    anslen = i - x + 1;
                }
                -- cnt[S[x++]];
            }
        }
        if (anslen == S.size() + 1) return "";
        return S.substr(ansx, anslen);
    }
};
```

1.87 Multiply Strings

Problem Description

Given two numbers represented as strings, return multiplication of the numbers as a string.
Note: The numbers can be arbitrarily large and are non-negative.

Solution

水题

Code

```
class Solution {
public:
    string multiply(string num1, string num2) {
        string ans;
        num1 = string(num1.rbegin(), num1.rend());
        num2 = string(num2.rbegin(), num2.rend());

        for (int i = 0; i < num1.size(); ++i) {
            ans = add(ans, multiply(num2, num1[i] - '0'), i);
        }
        while(ans.size() >= 2 && ans[ans.size() - 1] == '0') ans.pop_back();
        return string(ans.rbegin(), ans.rend());
    }

private:
    string multiply(string num1, int n2) {
        int c = 0;
        for (int i = 0; i < num1.size(); ++i) {
            c += (num1[i] - '0') * n2;
            num1[i] = c % 10 + '0';
            c /= 10;
        }
        if (c) {
            num1.push_back(c + '0');
        }
        return num1;
    }

    string add(string num1, string num2, int pow) {
        string ans;
        int c = 0;
        for (int i = 0, j = -pow; i < num1.size() || j < num2.size(); ++i, ++
            j) {
            if (i < num1.size()) {
                c += num1[i] - '0';
            }
            if (0 <= j && j < num2.size()) {
                c += num2[j] - '0';
            }
            ans.push_back(c % 10 + '0');
            c /= 10;
        }
        if (c) {
            ans.push_back(c + '0');
        }
        return ans;
    }
};
```

1.88 N-Queens II

Problem Description

Follow up for N-Queens problem.
Now, instead outputting board configurations, return the total number of distinct solutions.

Solution

水题

Code

```
class Solution {
public:
    int totalNQueens(int n) {
        return _DFS(0, n, 0, 0, 0);
    }
private:
    int _DFS(int k, int n, int left, int right, int up) {
        if (k == n) {
            return 1;
        }
        int ans = 0;
        for (int i = 0; i < n; ++i) {
            if (!(left & 1<<i) && !(right & 1<<i) && !(up & 1<<i)) {
                ans += _DFS(k + 1, n, (left | 1<<i) << 1, (right | 1<<i) >>
                    1, (up | 1<<i));
            }
        }
        return ans;
    }
};
```

1.89 N-Queens

Problem Description

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

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

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

For example,

There exist two distinct solutions to the 4-queens puzzle:

```
[
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

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

Solution

水题

Code

```
class Solution {
public:
    vector<vector<string> > solveNQueens(int n) {
        ans.clear();
        now.clear();
        _DFS(0, n, 0, 0, 0);
        return ans;
    }
private:
    void _DFS(int k, int n, int left, int right, int up) {
        if (k == n) {
            _gen(n);
        }
        for (int i = 0; i < n; ++i) {
            if (!(left & 1<<i) && !(right & 1<<i) && !(up & 1<<i)) {
                now.push_back(i);
                _DFS(k + 1, n, (left | 1<<i) << 1, (right | 1<<i) >> 1, (up |
                     1<<i));
                now.pop_back();
            }
        }
    }
    void _gen(int n) {
        vector<string> solve;
        string line(n, '.');
        for (int i = 0; i < n; ++i) {
            line[now[i]] = 'Q';
            solve.push_back(line);
            line[now[i]] = '.';
        }
        ans.push_back(solve);
    }
    vector<vector<string> > ans;
    vector<int> now;
};
```

1.90 Next Permutation

Problem Description

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

1,2,3 →1,3,2
3,2,1 →1,2,3
1,1,5 →1,5,1

Solution

水题

Code

```
class Solution {
public:
    void nextPermutation(vector<int> &num) {
        if (next_permutation(num.begin(), num.end()) == false) {
            sort(num.begin(), num.end());
        }
    }
};
```

1.91 Palindrome Number

Problem Description

Determine whether an integer is a palindrome. Do this without extra space.

Could negative integers be palindromes? (ie, -1)

If you are thinking of converting the integer to string, note the restriction of using extra space.

You could also try reversing an integer. However, if you have solved the problem "Reverse Integer", you know that the reversed integer might overflow. How would you handle such case?

There is a more generic way of solving this problem.

Solution

水题

Code

```
class Solution {
public:
    bool isPalindrome(int x) {
        if (x < 0) return false;
        char s[16], n = 0;
        while(x) {
            s[n++] = x % 10;
            x /= 10;
        }
        for (int i = 0; i < n / 2; ++i) {
            if (s[i] != s[n - i - 1]) return false;
        }
        return true;
    }
};
```

1.92 Palindrome Partitioning II

Problem Description

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.
For example, given $s =$ **"aab"**,
Return **1** since the palindrome partitioning [**"aa","b"**] could be produced using 1 cut.

Solution

水题

Code

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

1.93 Palindrome Partitioning

Problem Description

Given a string s, partition s such that every substring of the partition is a palindrome.
Return all possible palindrome partitioning of s.
For example, given $s =$ **"aab"**,
Return

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

Solution

水题

Code

```
class Solution {
public:
    vector<vector<string>> partition(string s) {
        int n = s.size();
        vector<vector<bool> > pali(s.size(), vector<bool> (s.size() + 1,
            false));
        for (int i = 0; i < n; ++i) {
            pali[i][i] = pali[i][i + 1] = true;
        }
        for (int len = 2; len <= n; ++len) {
            for (int i = 0; i + len <= n; ++i) {
                int j = i + len;
                if (s[i] == s[j - 1] && pali[i + 1][j - 1]) {
                    pali[i][j] = true;
                }
            }
        }
        vector<vector<string> > ans;
        vector<string> now;
        _gen(s, 0, pali, ans, now);
        return ans;
    }
private:
    void _gen(string &s, int k, vector<vector<bool> > &pali, vector<vector<
        string>> &ans, vector<string> &now) {
        if (k == s.size()) {
            ans.push_back(now);
            return ;
        }
        for (int i = k + 1; i <= s.size(); ++i) {
            if (pali[k][i]) {
                    now.push_back(s.substr(k, i - k));
                    _gen(s, i, pali, ans, now);
                    now.pop_back();
                }
            }
        }
};
```

1.94 Partition List

Problem Description

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,

Given **1->4->3->2->5->2** and $x = 3$,

return **1->2->2->4->3->5**.

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *partition(ListNode *head, int x) {
        ListNode h1(0), *p1, h2(0), *p2;
        p1 = &h1, p2 = &h2;
        for (ListNode *p3 = head; p3; p3 = p3->next) {
            if (p3->val < x) {
                p1->next = p3;
                p1 = p1->next;
            } else {
                p2->next = p3;
                p2 = p2->next;
            }
        }
        p1->next = h2.next;
        p2->next = NULL;
        return h1.next;
    }
};
```

1.95 Pascal's Triangle II

Problem Description

Solution

水题

Code

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {
        vector<int> line(rowIndex + 1, 1);
        long long c = 1;
        for (int i = 1; i < rowIndex; ++i) {
            c = c * (rowIndex - i + 1) / i;
            line[i] = c;
        }
        return line;
    }
};
```

1.96 Pascal's Triangle

Problem Description

Solution

水题

Code

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

1.97 Path Sum II

Problem Description

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

return

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

```
              5
             / \
            4   8
```

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

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int> > pathSum(TreeNode *root, int sum) {
        vector<vector<int> > ans;
        vector<int> path;
        getPathSum(root, sum, path, ans);
        return ans;
    }
private:
    void getPathSum(TreeNode *root, int sum, vector<int> &path, vector<vector
        <int> > &ans) {
        if (root == NULL) return ;
        if (root->left == NULL && root->right == NULL) {
            if (root->val == sum) {
                path.push_back(root->val);
                ans.push_back(path);
                path.pop_back();
            }
            return ;
        }
        path.push_back(root->val);
        getPathSum(root->left, sum - root->val, path, ans);
        getPathSum(root->right, sum - root->val, path, ans);
        path.pop_back();
    }
};
```

1.98 Path Sum

Problem Description

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.

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

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

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool hasPathSum(TreeNode *root, int sum) {
        if (root == NULL) return false;
        if (root->left == NULL && root->right == NULL) {
            return root->val == sum;
        }
        if (hasPathSum(root->left, sum - root->val) || hasPathSum(root->right
            , sum - root->val)) {
            return true;
        }
        return false;
    }
};
```

1.99 Permutation Sequence

Problem Description

The set [**1,2,3,**···,$\boldsymbol{n}$] contains a total of n! unique permutations.
By listing and labeling all of the permutations in order,
We get the following sequence (ie, for $n = 3$):
"123" "132" "213" "231" "312" "321"

- **"123"**
- **"132"**
- **"213"**
- **"231"**
- **"312"**
- **"321"**

Given n and k, return the kth permutation sequence.
Note: Given n will be between 1 and 9 inclusive.

Solution

水题

Code

```
class Solution {
public:
    string getPermutation(int n, int k) {
        vector<bool> select(n + 1, false);
        vector<int> fac(n + 1, 1);
        string s;
        for (int i = 2; i <= n; ++i) {
            fac[i] = fac[i - 1] * i;
        }
        -- k;
        for (int i = n - 1; i >= 0; --i) {
            s.push_back(_get(select, k / fac[i]) + '0');
            k %= fac[i];
        }
        return s;
    }
private:
    int _get(vector<bool> &select, int k) {
        int cnt = 0;
        for (int i = 1; i < select.size(); ++i) {
            if (select[i] == false) {
                ++ cnt;
            }
            if (cnt == k + 1) {
                select[i] = true;
                return i;
            }
        }
        return -1;
    }
};
```

1.100 Permutations II

Problem Description

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,
[**1,1,2**] have the following unique permutations:
[**1,1,2**], [**1,2,1**], and [**2,1,1**].

Solution

水题

Code

```
class Solution {
public:
    vector<vector<int> > permuteUnique(vector<int> &num) {
        vector<vector<int> > ans;
        sort(num.begin(), num.end());
        do {
            ans.push_back(num);
        } while (next_permutation(num.begin(), num.end()));
        return ans;
    }
};
```

1.101 Permutations

Problem Description

Given a collection of numbers, return all possible permutations.
For example,
[1,2,3] have the following permutations:
[1,2,3], **[1,3,2]**, **[2,1,3]**, **[2,3,1]**, **[3,1,2]**, and **[3,2,1]**.

Solution

水题

Code

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

1.102 Plus One

Problem Description

Given a non-negative number represented as an array of digits, plus one to the number.
The digits are stored such that the most significant digit is at the head of the list.

Solution

水题

Code

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

1.103 Populating Next Right Pointers in Each Node II

Problem Description

Follow up for problem "*Populating Next Right Pointers in Each Node*".
What if the given tree could be any binary tree? Would your previous solution still work?
Note:

- You may only use constant extra space.

For 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
```

Solution

水题

Code

```cpp
/**
 * Definition for binary tree with next pointer.
 * struct TreeLinkNode {
 *  int val;
 *  TreeLinkNode *left, *right, *next;
 *  TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
 * };
 */
class Solution {
public:
    void connect(TreeLinkNode *root) {
        if (root == NULL) return ;
        TreeLinkNode *head = NULL, *tail = NULL, *p = root;
        for (; p; p = p->next) {
            insertLink(head, tail, p->left);
            insertLink(head, tail, p->right);
        }
        connect(head);
    }
private:
    void insertLink(TreeLinkNode * &head, TreeLinkNode * &tail, TreeLinkNode
        * node) {
```

```
        if (node != NULL) {
            if (head == NULL) {
                head = tail = node;
            } else {
                tail->next = node;
                tail = tail->next;
            }
        }
    }
};
```

1.104 Populating Next Right Pointers in Each Node

Problem Description

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.
- You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

For example,
Given the following perfect binary tree,

```
     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
```

Solution

水题

Code

```
/**
 * Definition for binary tree with next pointer.
 * struct TreeLinkNode {
 *  int val;
 *  TreeLinkNode *left, *right, *next;
 *  TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
 * };
 */
class Solution {
public:
    void connect(TreeLinkNode *root) {
        if (root == NULL) return ;
        queue<TreeLinkNode *> nodes;
        queue<int> depths;
        nodes.push(root); depths.push(0);
        while (nodes.size()) {
            root = nodes.front();
            int d = depths.front();
            nodes.pop(); depths.pop();
            if (nodes.size() && depths.front() == d) root->next = nodes.front
                ();
            else root->next = NULL;
            if (root->left) {
                nodes.push(root->left); depths.push(d + 1);
            }
            if (root->right) {
                nodes.push(root->right); depths.push(d + 1);
            }
        }
    }
};
```

1.105 Pow(x, n)

Problem Description

Solution

水题

Code

```
class Solution {
public:
    double pow(double x, int n) {
        long long longn = n;
        if (n < 0) return 1 / _pow(x, -longn);
        return _pow(x, n);
    }
private:
    double _pow(double x, long long n) {
        double ans = 1.0;
        while (n) {
            if (n % 2) ans *= x;
            x *= x;
            n >>= 1;
        }
        return ans;
    }
};
```

1.106 Recover Binary Search Tree

Problem Description

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

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

The above binary tree is serialized as **"{1,2,3,#,#,4,#,#,5}"**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void recoverTree(TreeNode *root) {
        if (root == NULL) return ;
        TreeNode *mistakeroot, *minp, *maxp, *lminp, *lmaxp, *rminp, *rmaxp;
        mistakeroot = _mistake(root, minp, maxp);

        _mistake(mistakeroot->left, lminp, lmaxp);
        _mistake(mistakeroot->right, rminp, rmaxp);

        if ((lmaxp && lmaxp->val > mistakeroot->val) && (rminp && rminp->val
            < mistakeroot->val)) {
            _swap(lmaxp, rminp);
            return ;
        }
        if (lmaxp && lmaxp->val > mistakeroot->val) {
            _swap(lmaxp, mistakeroot);
        }
        if (rminp && rminp->val < mistakeroot->val) {
            _swap(rminp, mistakeroot);
        }
    }
private:
    void _swap(TreeNode *p, TreeNode *q) {
        int temp = p->val;
        p->val = q->val;
        q->val = temp;
    }

    TreeNode* _mistake(TreeNode *root, TreeNode *&minp, TreeNode *&maxp) {
        if (root == NULL) {
            minp = maxp = NULL;
            return NULL;
        }

        TreeNode *lminp, *rminp, *lmaxp, *rmaxp, *lmistake, *rmistake;

        lmistake = _mistake(root->left, lminp, lmaxp);
        rmistake = _mistake(root->right, rminp, rmaxp);

        minp = maxp = root;
        if (lminp && lminp->val < minp->val) minp = lminp;
        if (rminp && rminp->val < minp->val) minp = rminp;
        if (lmaxp && lmaxp->val > maxp->val) maxp = lmaxp;
        if (rmaxp && rmaxp->val > maxp->val) maxp = rmaxp;


        if ((lmaxp && lmaxp->val > root->val) || (rminp && rminp->val < root
            ->val)) return root;
        if (lmistake) return lmistake;
        if (rmistake) return rmistake;
        return NULL;
    }
};
```

1.107 Regular Expression Matching

Problem Description

Implement regular expression matching with support for '.' and '*'.

```
'.' Matches any single character.
'*' Matches zero or more of the preceding element.

The matching should cover the <b>entire</b> input string (not partial).

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "a*") → true
isMatch("aa", ".*") → true
isMatch("ab", ".*") → true
isMatch("aab", "c*a*b") → true
```

Solution

水题

Code

```
class Solution {
public:
    bool isMatch(const char *s, const char *p) {
        int n = strlen(s), m = strlen(p);
        vector<vector<bool> > f(n + 1, vector<bool> (m + 1, false));
        f[0][0] = true;
        for (int i = 0; i <= n; ++i) {
            for (int j = 0; j <= m; ++j) {
                if (i && j && p[j - 1] == '.') {
                    f[i][j] = f[i - 1][j - 1];
                } else if (j >= 2 && p[j - 1] == '*') {
                    if (p[j - 2] == '.') {
                        f[i][j] = (f[i][j - 2] || (i && f[i - 1][j]));
                    } else {
                        f[i][j] = (f[i][j - 2] || (i && s[i - 1] == p[j - 2]
                            && f[i - 1][j]));
                    }
                } else if (i && j){
                    f[i][j] = (f[i - 1][j - 1] && s[i - 1] == p[j - 1]);
                }
            }
        }
        return f[n][m];
    }
};
```

1.108 Remove Duplicates from Sorted Array II

Problem Description

Follow up for "Remove Duplicates":
What if duplicates are allowed at most *twice*?
For example,
Given sorted array A = [**1,1,1,2,2,3**],
Your function should return length = **5**, and A is now [**1,1,2,2,3**].

Solution

水题

Code

```
class Solution {
public:
    int removeDuplicates(int A[], int n) {
        if (n <= 2) return n;
        int m = 2;
        for (int i = 2; i < n; ++i) {
            if (A[m - 2] != A[i]) {
                A[m++] = A[i];
            }
        }
        return m;
    }
};
```

1.109 Remove Duplicates from Sorted Array

Problem Description

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

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,

Given input array A = **[1,1,2]**,

Your function should return length = **2**, and A is now **[1,2]**.

Solution

水题

Code

```
class Solution {
public:
    int removeDuplicates(int A[], int n) {
        if (n == 0) return 0;
        int m = 1;
        for (int i = 1; i < n; ++i) {
            if (A[m - 1] != A[i]) A[m++] = A[i];
        }
        return m;
    }
};
```

1.110 Remove Duplicates from Sorted List II

Problem Description

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only *distinct* numbers from the original list.

For example,

Given **1->2->3->3->4->4->5**, return **1->2->5**.

Given **1->1->1->2->3**, return **2->3**.

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *deleteDuplicates(ListNode *head) {
        ListNode newhead(0);
        newhead.next = head;

        ListNode *pre = &newhead, *p = head;
        while (p && p->next) {
            bool needdelete = false;
            while(p->next && p->val == p->next->val) {
                p->next = p->next->next;
                needdelete = true;
            }
            if (needdelete) {
                pre->next = p->next;
                p = pre->next;
            } else {
                pre = p;
                p = p->next;
            }
        }
        return newhead.next;
    }
};
```

1.111 Remove Duplicates from Sorted List

Problem Description

Given a sorted linked list, delete all duplicates such that each element appear only *once*.
For example,
Given **1->1->2**, return **1->2**.
Given **1->1->2->3->3**, return **1->2->3**.

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *deleteDuplicates(ListNode *head) {
        if (head == NULL) return head;
        ListNode *p = head, *q;
        while(p->next) {
            if (p->val == p->next->val) {
                q = p->next;
                p->next = q->next;
                delete q;
            } else {
                p = p->next;
            }
        }
        return head;
    }
};
```

1.112 Remove Element

Problem Description

Given an array and a value, remove all instances of that value in place and return the new length.
The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Solution

水题

Code

```
class Solution {
public:
    int removeElement(int A[], int n, int elem) {
        int m = 0;
        for (int i = 0; i < n; ++i) {
            if (A[i] != elem) A[m++] = A[i];
        }
        return m;
    }
};
```

1.113 Remove Nth Node From End of List

Problem Description

Given a linked list, remove the nth node from the end of list and return its head.
For example,
Note:
Given n will always be valid.
Try to do this in one pass.

```
Given linked list: <b>1->2->3->4->5</b>, and <b><i>n</i> = 2</b>.

After removing the second node from the end, the linked list becomes <b>1->2->3->5</b>.
```

Solution

水题

Code

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

class Solution {
public:
    ListNode *removeNthFromEnd(ListNode *head, int n) {
        if (head == NULL || n <= 0) return head;
        ListNode newhead(0), *p, *q = head;
        int diff = 0;
        newhead.next = head;
        p = &newhead;
        for (; q; q = q->next) {
            if (diff == n) {
                p = p->next;
            } else {
                ++ diff;
            }
        }
        q = p->next;
        p->next = q->next;
        delete q;
        return newhead.next;
    }
};
```

1.114 Reorder List

Problem Description

Given a singly linked list L: $L0 \rightarrow L1 \rightarrow \cdots \rightarrow Ln\text{-}1 \rightarrow Ln$, reorder it to: $L0 \rightarrow Ln \rightarrow L1 \rightarrow Ln\text{-}1 \rightarrow L2 \rightarrow Ln\text{-}2 \rightarrow \cdots$

You must do this in-place without altering the nodes' values.

For example, Given **{1,2,3,4}**, reorder it to **{1,4,2,3}**.

Solution

水题

Code

```
/**
 * 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 == NULL || head->next == NULL) return;

        ListNode *h1, *h2;
        cut(head, h1, h2);
        h2 = reverse(h2);
        for (ListNode *p, *q; h2; h1 = p, h2 = q) {
            p = h1->next;
            q = h2->next;
            h1->next = h2;
            h2->next = p;
        }
    }
private:
    ListNode *reverse(ListNode *head) {
        ListNode *p1 = NULL, *p2 = head, *p3;
        for (; p2; p1 = p2, p2 = p3) {
            p3 = p2->next;
            p2->next = p1;
        }
        return p1;
    }
    void cut(ListNode *head, ListNode* &h1, ListNode* &h2) {
        ListNode* p;
        p = h1 = h2 = head;
        for (int i = 0; p != NULL; ++i, p = p->next) {
            if (i && i % 2 == 0) {
                h2 = h2->next;
            }
        }
        p = h2, h2 = h2->next, p->next = NULL;
    }
};
```

1.115 Restore IP Addresses

Problem Description

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example:

Given **"25525511135"**,

return [**"255.255.11.135", "255.255.111.35"**]. (Order does not matter)

Solution

水题

Code

```
class Solution {
public:
    vector<string> restoreIpAddresses(string s) {
        ans.clear();
        now.clear();
        if (s.size() > 12) return ans;
        _DFS(s, 0);
        return ans;
    }
private:
    void _DFS(string &s, int k) {
        if (now.size() == 4 || k == s.size()) {
            if (now.size() == 4 && k == s.size()) {
                ans.push_back(now[0] + "." + now[1] + "." + now[2] + "." +
                    now[3]);
            }
            return ;
        }
        if (s[k] == '0') {
            now.push_back("0");
            _DFS(s, k + 1);
            now.pop_back();
        } else {
                string block = "";
                int value = 0;
                for (int i = k; i < s.size(); ++i) {
                    value = value * 10 + s[i] - '0';
                    if (value < 256) {
                        block.push_back(s[i]);
                        now.push_back(block);
                        _DFS(s, i + 1);
                        now.pop_back();
                    }
                }
            }

        }
        vector<string> ans, now;
};
```

1.116 Reverse Integer

Problem Description

Reverse digits of an integer.

Here are some good questions to ask before coding. Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases?

For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

Update (2014-11-10): Test cases had been added to test the overflow behavior.

Solution

水题

Code

```
class Solution {
public:
    int reverse(int x) {
        long long y = x;
        if (y >= 0) {
            return _reverse(y);
        }
        return - _reverse(-y);
    }
private:
    long long _reverse(long long x) {
        int a[32], n = 0;
        long long ans = 0;
        while (x) {
            a[n++] = x % 10;
            x /= 10;
        }
        for (int i = 0; i < n; ++i) {
            ans = ans * 10 + a[i];
        }
        return ans;
    }
};
```

1.117 Reverse Linked List II

Problem Description

Reverse a linked list from position m to n. Do it in-place and in one-pass.
For example:
Given **1->2->3->4->5->NULL**, $m = 2$ and $n = 4$,
return **1->4->3->2->5->NULL**.
Note:
Given m, n satisfy the following condition:
$1 \leqslant m \leqslant n \leqslant$ length of list.

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *reverseBetween(ListNode *head, int m, int n) {
        ListNode newhead(0), *pre, *p, *q;
        newhead.next = head;
        pre = &newhead;
        p = head;
        for (int i = 1; i < m; ++i) {
            pre = p, p = p->next;
        }
        for (int i = m; i < n; ++i) {
            q = p->next;
            p->next = q->next;
            q->next = pre->next;
            pre->next = q;
        }
        return newhead.next;
    }
};
```

1.118 Reverse Nodes in k-Group

Problem Description

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.
If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.
You may not alter the values in the nodes, only nodes itself may be changed.
Only constant memory is allowed.
For example,

Given this linked list: **1->2->3->4->5**
For $k = 2$, you should return: **2->1->4->3->5**
For $k = 3$, you should return: **3->2->1->4->5**

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *reverseKGroup(ListNode *head, int k) {
        ListNode newhead(0), *pre;
        newhead.next = head;
        pre = &newhead;
        for (ListNode *p = head; p; pre = p, p = p->next) {
            ListNode *tail = p;
            for (int i = 1; i < k && tail; ++i) tail = tail->next;
            if (tail == NULL) break;
            tail = p->next;
            for (int i = 1; i < k; ++i) {
                p->next = tail->next;
                tail->next = pre->next;
                pre->next = tail;
                tail = p->next;
            }
        }
        return newhead.next;
    }
};
```

1.119 Reverse Words in a String

Problem Description

Given an input string, reverse the string word by word.
For example, Given s = "**the sky is blue**", return "**blue is sky the**".

- What constitutes a word? A sequence of non-space characters constitutes a word.
- Could the input string contain leading or trailing spaces? Yes. However, your reversed string should not contain leading or trailing spaces.
- How about multiple spaces between two words? Reduce them to a single space in the reversed string.

Solution

水题

Code

```
class Solution {
public:
    void reverseWords(string &s) {
        int n = s.length();
        reverse(s);
        for (int i = 0; i < n; ++i) {
            if (isalpha(s[i])) {
                int p = i + 1;
                while (p < n && isalpha(s[p])) ++p;
                reverse(s, i, p - i);
                i = p;
            }
        }
        int m = 0;
        for (int i = 0; i < n; ++i) {
            if (s[i] != '␣' || s[i] == '␣' && m && s[m - 1] != '␣') {
                s[m ++] = s[i];
            }
        }
        if (s[m - 1] == '␣') -- m;
        s.resize(m);
    }
private:
    bool isalpha(char c) {
        return c != '␣';
    }
    void reverse(string &s, int start = 0, int len = INT_MAX) {
        len = std::min(len, int(s.length()) - start);
        for (int i = 0, p = start, q = start + len - 1; i < len / 2; ++i, ++p
            , --q) {
            s[p] ^= s[q] ^= s[p] ^= s[q];
        }
    }
};
```

1.120 Roman to Integer

Problem Description

Given a roman numeral, convert it to an integer.
Input is guaranteed to be within the range from 1 to 3999.

Solution

水题

Code

```
class Solution {
public:
    int romanToInt(string s) {
        int ans = 0;
        const char r1 = 'I', r5 = 'V', r10 = 'X', r50 = 'L', r100 = 'C', r500
            = 'D', r1000 = 'M';
        for (int i = 0; i < s.size(); ++i) {
            switch (s[i]) {
                case r1000:
                    ans += 1000;
                    break;
                case r500:
                    ans += 500;
                    break;
                case r100:
                    if (i + 1 < s.size() && (s[i + 1] == r1000 || s[i + 1] ==
                         r500)) {
                        ans -= 100;
                    } else {
                        ans += 100;
                    }
                    break;
                case r50:
                    ans += 50;
                    break;
                case r10:
                    if (i + 1 < s.size() && (s[i + 1] == r100 || s[i + 1] ==
                        r50)) {
                        ans -= 10;
                    } else {
                        ans += 10;
                    }
                    break;
                case r5:
                    ans += 5;
                    break;
                case r1:
                    if (i + 1 < s.size() && (s[i + 1] == r10 || s[i + 1] ==
                        r5)) {
                        ans -= 1;
                    } else {
                        ans += 1;
                    }
            }
        }
        return ans;
    }
};
```

1.121 Rotate List

Problem Description

Given a list, rotate the list to the right by k places, where k is non-negative.
For example:
Given **1->2->3->4->5->NULL** and $k = \mathbf{2}$,
return **4->5->1->2->3->NULL**.

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *rotateRight(ListNode *head, int k) {
        if (head == NULL) return head;
        int n = 1;
        ListNode *tail = head;
        for (; tail->next; tail = tail->next) {
            ++ n;
        }
        tail->next = head;
        k = n - k % n;
        for (int i = 0; i < k; ++i) {
            tail = tail->next;
        }
        head = tail->next;
        tail->next = NULL;
        return head;
    }
};
```

1.122 Same Tree

Problem Description

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isSameTree(TreeNode *p, TreeNode *q) {
        if (p == NULL || q == NULL) {
            return  p == q;
        }
        if (isSameTree(p->left, q->left) && isSameTree(p->right, q->right)) {
            return p->val == q->val;
        }
        return false;
    }
};
```

1.123 Scramble String

Problem Description

Given a string *s1*, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of *s1* = **"great"**:

To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node **"gr"** and swap its two children, it produces a scrambled string **"rgeat"**.

We say that **"rgeat"** is a scrambled string of **"great"**.

Similarly, if we continue to swap the children of nodes **"eat"** and **"at"**, it produces a scrambled string **"rgtae"**.

We say that **"rgtae"** is a scrambled string of **"great"**.

Given two strings *s1* and *s2* of the same length, determine if *s2* is a scrambled string of *s1*.

```
    great
   /    \
  gr    eat
 / \    /  \
g   r  e   at
           / \
          a   t

    rgeat
   /    \
  rg    eat
 / \    /  \
r   g  e   at
           / \
          a   t

    rgtae
   /    \
  rg    tae
 / \    /  \
r   g  ta  e
       / \
      t   a
```

Solution

水题

Code

```
class Solution {
public:
    bool isScramble(string s1, string s2) {
        return isScramble(s1.c_str(), s2.c_str(), s1.size());
    }
private:
    bool isScramble(const char *s1, const char *s2, int n) {
        if (n == 0) return true;
        if (n == 1) return s1[0] == s2[0];
        vector<int> cnt(256, 0);
        for (int i = 0; i < n; ++i) {
            ++ cnt[s1[i]];
        }
        for (int i = 0; i < n; ++i) {
            -- cnt[s2[i]];
            if (cnt[s2[i]] < 0) return false;
        }
        for (int i = 1; i < n; ++i) {
            if (isScramble(s1, s2, i) && isScramble(s1 + i, s2 + i, n - i)) {
                return true;
            }
            if (isScramble(s1, s2 + n - i, i) && isScramble(s1 + i, s2, n - i
                )) {
                return true;
            }
        }
        return false;
    }
};
```

1.124 Search a 2D Matrix

Problem Description

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

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

For example,
Consider the following matrix:
Given **target** = **3**, return **true**.

```
[
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
```

Solution

水题

Code

```
class Solution {
public:
    bool searchMatrix(vector<vector<int> > &matrix, int target) {
        int n = matrix.size(), m = matrix[0].size();
        int left = 0, right = n * m - 1;
        while (left <= right) {
            int mid = (left + right) >> 1;
            if (matrix[mid / m][mid % m] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left < n * m && matrix[left / m][left % m] == target;
    }
};
```

1.125 Search for a Range

Problem Description

Given a sorted array of integers, find the starting and ending position of a given target value.
Your algorithm's runtime complexity must be in the order of $O(\log n)$.
If the target is not found in the array, return [**-1, -1**].
For example,
Given [**5, 7, 7, 8, 8, 10**] and target value 8,
return [**3, 4**].

Solution

水题

Code

```
class Solution {
public:
    vector<int> searchRange(int A[], int n, int target) {
        vector <int> ans;
        int begin = findBegin(A, n, target);
        int end = findEnd(A, n, target);
        if (begin <= end) {
            ans.push_back(begin);
            ans.push_back(end);
        } else {
            ans.push_back(-1);
            ans.push_back(-1);
        }
        return ans;
    }
private:
    int findBegin(int A[], int n, int target) {
        int left = 0, right = n - 1;
        while (left <= right) {
            int mid = (left + right) >> 1;
            if (A[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left;
    }

    int findEnd(int A[], int n, int target) {
        int left = 0, right = n - 1;
        while (left <= right) {
            int mid = (left + right) >> 1;
            if (A[mid] <= target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return right;
    }
};
```

1.126 Search in Rotated Sorted Array II

Problem Description

Follow up for "Search in Rotated Sorted Array":
What if *duplicates* are allowed?
Would this affect the run-time complexity? How and why?
Write a function to determine if a given target is in the array.

Solution

水题

Code

```
class Solution {
public:
    bool search(int A[], int n, int target) {
        if (n <= 0) return false;
        if (n == 1) return A[0] == target;
        int left = 0, right = n - 1, mid = (left + right) / 2;
        if (A[left] == target || A[right] == target || A[mid] == target)
            return true;
        if (A[left] < A[mid]) {
            if (A[left] < target && target < A[mid]) {
                return search(A + left + 1, mid - left - 1, target);
            } else {
                return search(A + mid + 1, right - mid - 1, target);
            }
        }
        if (A[mid] < A[right]) {
            if (A[mid] < target && target < A[right]) {
                return search(A + mid + 1, right - mid - 1, target);
            } else {
                return search(A + left + 1, mid - left - 1, target);
            }
        }
        return search(A + left + 1, mid - left - 1, target) || search(A + mid
             + 1, right - mid - 1, target);
    }
};
```

1.127 Search in Rotated Sorted Array

Problem Description

Suppose a sorted array is rotated at some pivot unknown to you beforehand.
(i.e., **0 1 2 4 5 6 7** might become **4 5 6 7 0 1 2**).
You are given a target value to search. If found in the array return its index, otherwise return -1.
You may assume no duplicate exists in the array.

Solution

水题

Code

```
class Solution {
public:
    int search(int A[], int n, int target) {
        int pivot = findPivot(A, n), idx;
        if (target >= A[0]) {
            idx = find(A, pivot, target);
            if (idx == pivot || A[idx] != target) {
                idx = -1;
            }
        } else {
            idx = find(A + pivot, n - pivot, target) + pivot;
            if (idx == n || A[idx] != target) {
                idx = -1;
            }
        }
        return idx;
    }
private:
    int findPivot(int A[], int n) {
        int left = 0, right = n - 1;
        while (left <= right) {
            int mid = (left + right) >> 1;
            if (A[mid] >= A[0]) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left;
    }
    int find(int A[], int n, int target) {
        int left = 0, right = n - 1;
        while (left <= right) {
            int mid = (left + right) >> 1;
            if (A[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left;
    }
};
```

1.128 Search Insert Position

Problem Description

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

You may assume no duplicates in the array.

Here are few examples.

[**1,3,5,6**], 5 →2
[**1,3,5,6**], 2 →1
[**1,3,5,6**], 7 →4
[**1,3,5,6**], 0 →0

Solution

水题

Code

```
class Solution {
public:
    int searchInsert(int A[], int n, int target) {
        int left = 0, right = n - 1;
        while (left <= right) {
            int mid = (left + right) >> 1;
            if (A[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left;
    }
};
```

1.129 Set Matrix Zeroes

Problem Description

Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do it in place.
Did you use extra space?
A straight forward solution using O(mn) space is probably a bad idea.
A simple improvement uses O($m + n$) space, but still not the best solution.
Could you devise a constant space solution?

Solution

水题

Code

```
class Solution {
public:
    void setZeroes(vector<vector<int> > &matrix) {
        if (matrix.size() == 0) return ;
        int n = matrix.size(), m = matrix[0].size();
        vector<bool> row(n, false), col(m, false);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (matrix[i][j] == 0) {
                    row[i] = col[j] = true;
                }
            }
        }
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (row[i] || col[j]) {
                    matrix[i][j] = 0;
                }
            }
        }
    }
};
```

1.130 Simplify Path

Problem Description

Given an absolute path for a file (Unix-style), simplify it.
For example,
path = ”**/home/**”, => ”**/home**”
path = ”**/a/./b/../../c/**”, => ”**/c**”
Did you consider the case where **path** = ”**/../**”?
In this case, you should return ”/”. Another corner case is the path might contain multiple slashes ’/’ together, such as ”**/home//foo/**”.
In this case, you should ignore redundant slashes and return ”**/home/foo**”.

- Did you consider the case where **path** = ”**/../**”?

  In this case, you should return ”/”.

- Another corner case is the path might contain multiple slashes ’/’ together, such as ”**/home//foo/**”.

  In this case, you should ignore redundant slashes and return ”**/home/foo**”.

Solution

水题

Code

```cpp
class Solution {
public:
    string simplifyPath(string path) {
        vector<string> files = split(path);
        vector<string> newfiles;
        for (int i = 0; i < files.size(); ++i) {
            if (files[i] == "..") {
                if (newfiles.size()) newfiles.pop_back();
            } else if (files[i] == "" || files[i] == ".") {

            } else {
                newfiles.push_back(files[i]);
            }
        }
        if (newfiles.size() == 0) return "/";
        string newpath;
        for (int i = 0; i < newfiles.size(); ++i) {
            newpath.push_back('/');
            for (int j = 0; j < newfiles[i].size(); ++j) {
                newpath.push_back(newfiles[i][j]);
            }
        }
        return newpath;
    }
private:
    vector<string> split(string path, char c = '/') {
        vector<string> ans;
        string dir;
        for (int i = 0; i < path.size(); ++i) {
            if (path[i] == c) {
                ans.push_back(dir);
                dir = "";
            } else {
                dir.push_back(path[i]);
            }
        }
        ans.push_back(dir);
        return ans;
    }
};
```

1.131 Single Number II

Problem Description

Given an array of integers, every element appears *three* times except for one. Find that single one.

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

Solution

水题

Code

```
class Solution {
typedef long long LL;
public:
    int singleNumber(int A[], int n) {
        LL ans = 0, x;
        for (int i = 0; i < n; ++i) {
            x = A[i];
            ans = xor3(ans, abs(x));
        }
        int cnt = 0;
        for (int i = 0; i < n; ++i) {
            if (A[i] == ans) {
                ++ cnt;
            }
        }
        return cnt == 1? ans: -ans;
    }
private:
    LL xor3(LL x, LL y) {
        if (x == 0 || y == 0) {
            return x ^ y;
        }
        return xor3(x / 3, y / 3) * 3 + (x % 3 + y % 3) % 3;
    }
};
```

1.132 Single Number II2

Problem Description

Solution

水题

Code

```
class Solution {
typedef long long LL;
public:
    int singleNumber(int A[], int n) {
        return single((unsigned*)(A), n);
    }

private:
    int single(unsigned A[], int n) {
        if (n == 1) return A[0];
        unsigned remind = 0;
        for (int i = 0; i < n; ++i) {
            remind = (remind + A[i] % 3) % 3;
        }
        int m = 0;
        for (int i = 0; i < n; ++i) {
            if (A[i] % 3 == remind) {
                A[m++] = A[i] / 3;
            }
        }
        return single(A, m) * 3 + remind;
    }
};
```

1.133 Single Number II3

Problem Description

Solution

水题

Code

```
class Solution {
public:
    int singleNumber(int A[], int n) {
        int cnt0 = ~0, cnt1 = 0, cnt2 = 0;
        for (int i = 0; i < n; ++i) {
            int cnt2_ = cnt2;
            cnt2 = (cnt2 & ~A[i]) | (cnt1 & A[i]);
            cnt1 = (cnt1 & ~A[i]) | (cnt0 & A[i]);
            cnt0 = (cnt0 & ~A[i]) | (cnt2_ & A[i]);
        }
        return cnt1;
    }
};
```

1.134 Single Number

Problem Description

Given an 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?

Solution

水题

Code

```
class Solution {
public:
    int singleNumber(int A[], int n) {
        int ans = 0;
        for (int i = 0; i < n; ++i) {
            ans ^= A[i];
        }
        return ans;
    }
};
```

1.135 Sort Colors

Problem Description

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

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

Note:

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

Follow up:

A rather straight forward solution is a two-pass algorithm using counting sort.

First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

Solution

水题

Code

```
class Solution {
public:
    void sortColors(int A[], int n) {
        int count[3] = {0};
        for (int i = 0; i < n; ++i) {
            ++ count[A[i]];
        }
        for (int i = 0, k = 0; i < 3; ++i) {
            for (int j = 0; j < count[i]; ++j, ++k) {
                A[k] = i;
            }
        }
    }
};
```

1.136 Sort List

Problem Description

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

Solution

水题

Code

```
/**
 * 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 == NULL || head->next == NULL) return head;
        ListNode* h1, *h2;
        cut(head, h1, h2);
        h1 = sortList(h1);
        h2 = sortList(h2);
        return merger(h1, h2);
    }

private:
    void cut(ListNode *head, ListNode* &h1, ListNode* &h2) {
        ListNode* p;
        p = h1 = h2 = head;
        for (int i = 0; p != NULL; ++i, p = p->next) {
            if (i && i % 2 == 0) {
                h2 = h2->next;
            }
        }
        p = h2, h2 = h2->next, p->next = NULL;
    }

    ListNode* merger(ListNode* h1, ListNode* h2) {
        ListNode *head, *tail, *p;
        if (h1->val < h2->val) {
            tail = head = h1;
            h1 = h1->next;
        } else {
            tail = head = h2;
            h2 = h2->next;
        }
        while (h1 || h2) {
            if (h1 && h2) {
                if (h1->val < h2->val) {
                    p = h1, h1 = h1->next;
                } else {
                    p = h2, h2 = h2->next;
                }
            } else if (!h1 && h2) {
                p = h2, h2 = h2->next;
            } else if (h1 && !h2) {
                p = h1, h1 = h1->next;
            }
            tail->next = p;
            tail = p;
        }
        tail->next = NULL;
        return head;
    }
};
```

1.137 Spiral Matrix II

Problem Description

Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.
For example,
Given n = **3**,

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

Solution

水题

Code

```
int dir[][2] = {0, 1, 1, 0, 0, -1, -1, 0};

class Solution {
public:
    vector<vector<int> > generateMatrix(int n) {
        if (n == 0) return ans;
        ans = vector<vector<int> >(n, vector<int>(n, 0));
        DFS(0, n, 0, 0, 0);
        return ans;
    }
private:
    void DFS(int k, int n, int x, int y, int d) {
        if (k == n * n) {
            return ;
        }
        ans[x][y] = k + 1;
        int nx = x + dir[d][0], ny = y + dir[d][1];
        if (!(0 <= nx && nx < n && 0 <= ny && ny < n) || ans[nx][ny]) {
            d = (d + 1) % 4;
            nx = x + dir[d][0], ny = y + dir[d][1];
        }
        DFS(k + 1, n, nx, ny, d);
    }
    vector<vector<int> > ans;
};
```

1.138 Spiral Matrix

Problem Description

Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.
For example,
Given the following matrix:
You should return [**1,2,3,6,9,8,7,4,5**].

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

Solution

水题

Code

```cpp
int dir[][2] = {0, 1, 1, 0, 0, -1, -1, 0};

class Solution {
public:
    vector<int> spiralOrder(vector<vector<int> > &matrix) {
        ans.clear();
        if (matrix.size() == 0) return ans;
        visted = vector<vector<bool> >(matrix.size(), vector<bool>(matrix[0].
            size(), false));
        _gen(matrix, 0, 0, 0);
        return ans;
    }
private:
    void _gen(vector<vector<int> > &matrix, int x, int y, int d) {
        int n = matrix.size(), m = matrix[0].size();
        ans.push_back(matrix[x][y]);
        visted[x][y] = true;
        if (ans.size() == n * m) return ;
        int nx = x + dir[d][0], ny = y + dir[d][1];
        if ((0 <= nx && nx < n && 0 <= ny && ny < m) && !visted[nx][ny]) {
            _gen(matrix, nx, ny, d);
        } else {
            d = (d + 1) % 4;
            _gen(matrix, x + dir[d][0], y + dir[d][1], d);
        }
    }
    vector<vector<bool> > visted;
    vector<int> ans;
};
```

1.139 Sqrt(x)

Problem Description

Solution

水题

Code

```cpp
class Solution {
public:
    int sqrt(int x) {
        long long left = 0, right = x, mid;
        while(left <= right) {
            mid = (left + right) / 2;
            if (mid * mid <= x) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return right;
    }
};
```

1.140 String to Integer (atoi)

Problem Description

Solution

水题

Code

```
class Solution {
public:
    int atoi(const char *str) {
        long long ans = 0;
        int sign = 1;
        char *p = (char *)str;
        while (*p == '␣' || *p == '\t' || *p == '\n') ++p;
        if (*p == '-') {
            sign = -1;
            ++p;
        } else if (*p == '+') {
            ++p;
        }
        while (*p && '0' <= *p && *p <= '9') {
            ans = ans * 10 + (*p++ - '0');
            if (ans >= 2147483648) {
                return sign == 1? 2147483647: -2147483648;
            }
        }
        return ans * sign;
    }
};
```

1.141 Subsets II

Problem Description

Given a collection of integers that might contain duplicates, S, return all possible subsets.
Note:

- Elements in a subset must be in non-descending order.
- The solution set must not contain duplicate subsets.

For example,
If $\boldsymbol{S}$ = **[1,2,2]**, a solution is:

```
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

Solution

水题

Code

```
class Solution {
public:
    vector<vector<int> > subsetsWithDup(vector<int> &S) {
        ans.clear();
        now.clear();
        sort(S.begin(), S.end());
        _gen(S, 0, true);
        return ans;
    }
private:
    void _gen(vector<int> &S, int k, bool last) {
        if (k == S.size()) {
            ans.push_back(now);
            return ;
        }

        _gen(S, k + 1, false);

        if (!(last == false && S[k] == S[k - 1])) {
            now.push_back(S[k]);
            _gen(S, k + 1, true);
            now.pop_back();
        }
    }
    vector<vector<int> > ans;
    vector<int> now;
};
```

1.142 Subsets

Problem Description

Given a set of distinct integers, S, return all possible subsets.
Note:

- Elements in a subset must be in non-descending order.
- The solution set must not contain duplicate subsets.

For example,
If $\boldsymbol{S}$ = [**1,2,3**], a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

Solution

水题

Code

```
class Solution {
public:
    vector<vector<int> > subsets(vector<int> &S) {
        vector<vector<int> > ans;
        vector<int> now;
        sort(S.begin(), S.end());
        _gen(S, 0, now, ans);
        return ans;
    }
private:
    void _gen(vector<int> &S, int k, vector<int> &now, vector<vector<int> > &
       ans, bool c = true) {
        if (k == S.size()) {
            ans.push_back(now);
            return ;
        }
        _gen(S, k + 1, now, ans, false);
        if (c == false && S[k - 1] == S[k]) return;
        now.push_back(S[k]);
        _gen(S, k + 1, now, ans, true);
        now.pop_back();
    }
};
```

1.143 Substring with Concatenation of All Words

Problem Description

You are given a string, **S**, and a list of words, **L**, that are all of the same length. Find all starting indices of substring(s) in S that is a concatenation of each word in L exactly once and without any intervening characters.

For example, given:
S: **"barfoothefoobarman"**
L: [**"foo", "bar"**]
You should return the indices: [**0,9**].
(order does not matter).

Solution

水题

Code

```
#define maxn 100000

struct _TreeNode {
    _TreeNode * c[26], *fail;
    int Lid;
    _TreeNode() {
        init();
    }
    void init() {
        for (int i = 0; i < 26; ++i) {
            c[i] = NULL;
        }
        fail = NULL;
        Lid = -1;
    }
}nodes[maxn], *que[maxn];

class Solution {
public:
    vector<int> findSubstring(string S, vector<string> &L) {
        vector<int> ans;
        if (S.size() == 0 || L.size() == 0) return ans;
        int len = L[0].size(), n = 0;
        vector<int> cnt(L.size(), 0);

        sort(L.begin(), L.end());
        cnt[n++] = 1;
        for (int i = 1; i < L.size(); ++i) {
            if (L[i] == L[n - 1]) {
                ++ cnt[n - 1];
            } else {
                L[n++] = L[i];
                cnt[n - 1] = 1;
            }
        }
        vector<int> f(S.size());

        cntnode = 0;
        nodes[cntnode].init();
        _TreeNode *root = &nodes[cntnode++];
        for (int i = 0; i < n; ++i) {
            insert(root, L[i].c_str(), i);
        }

        createAC(root);

        _TreeNode *pos = root;
        for (int i = 0; i < S.size(); ++i) {
            while (pos != root && pos->c[S[i] - 'a'] == NULL) {
                pos = pos->fail;
                assert(pos != NULL);
            }
            if (pos->c[S[i] - 'a'] != NULL) {
                pos = pos->c[S[i] - 'a'];
            }
            f[i] = pos->Lid;
        }

        for (int i = 0; i < len; ++i) {
            vector<int> visted(n, 0);
            int cntL = 0;
            for (int j = i; j < S.size(); j += len) {
                if (f[j] != -1) {
                    ++ visted[f[j]];
                    if (visted[f[j]] == cnt[f[j]]) {
                        ++ cntL;
                    }
                }
                int out = j - len * L.size();
                if (out >= 0 && f[out] != -1) {
                    -- visted[f[out]];
                    if (visted[f[out]] == cnt[f[out]] - 1) {
                        -- cntL;
                    }
                }
                if (cntL == n) {
                    ans.push_back(out + 1);
                }
            }
        }
        sort(ans.begin(), ans.end());
        return ans;
    }

private:
    void insert(_TreeNode *root, const char s[], int Lid) {
        for (int i = 0; s[i]; ++i) {
            int k = s[i] - 'a';
            if (root->c[k] == NULL) {
                nodes[cntnode].init();
                root->c[k] = &nodes[cntnode++];
            }
            root = root->c[k];
        }
        root->Lid = Lid;
    }

    void createAC(_TreeNode *root) {
        int left = 0, right = 0;
        que[right++] = root;
        root->fail = root;
        while (left < right) {
            _TreeNode *now = que[left++], *pos;
            for (int i = 0; i < 26; ++i) {
                if (now->c[i] == NULL) continue;
                if (now == root) {
                    now->c[i]->fail = root;
                } else {
                    pos = now->fail;
                    while (pos != root && pos->c[i] == NULL) {
                        pos = pos->fail;
                    }
                    if (pos->c[i] != NULL) {
                        pos = pos->c[i];
                    }
                    now->c[i]->fail = pos;
                }
                que[right++] = now->c[i];
            }
        }
    }
    int cntnode;
```

```
};
```

1.144 Sudoku Solver

Problem Description

Write a program to solve a Sudoku puzzle by filling the empty cells.
Empty cells are indicated by the character **'.'**.
You may assume that there will be only one unique solution.
A sudoku puzzle...
...and its solution numbers marked in red.

Solution

水题

Code

```
class Solution {
public:
    void solveSudoku(vector<vector<char> > &board) {
        for (int i = 0; i < 9; ++i) for (int j = 0; j < 9; ++j) row[i][j] =
            col[i][j] = blk[i][j] = false;
        for (int i = 0; i < 9; ++i) {
            for (int j = 0; j < 9; ++j) {
                if (board[i][j] != '.') {
                    int c = board[i][j] - '1';
                    row[i][c] = col[j][c] = blk[i / 3 * 3 + j / 3][c] = true;
                }
            }
        }
        solve(board, 0);
    }

private:
    bool solve(vector<vector<char> > &board, int k) {
        if (k == 81) return true;
        int x = k / 9, y = k % 9;
        if (board[x][y] != '.') return solve(board, k + 1);
        for (int i = 0; i < 9; ++i) {
            if (!(row[x][i] || col[y][i] || blk[x / 3 * 3 + y / 3][i])) {
                row[x][i] = col[y][i] = blk[x / 3 * 3 + y / 3][i] = true;
                board[x][y] = i + '1';
                if(solve(board, k + 1)) return true;
                board[x][y] = '.';
                row[x][i] = col[y][i] = blk[x / 3 * 3 + y / 3][i] = false;
            }
        }
        return false;
    }
    bool row[9][9], col[9][9], blk[9][9];
};
```

1.145 Sum Root to Leaf Numbers

Problem Description

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

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

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

```
    1
   / \
  2   3
```

The root-to-leaf path **1->2** represents the number **12**.
The root-to-leaf path **1->3** represents the number **13**.
Return the sum = 12 + 13 = **25**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int sumNumbers(TreeNode *root) {
        int sum = 0;
        _sumNumbers(root, 0, sum);
        return sum;
    }
private:
    void _sumNumbers(TreeNode *root, int number, int &sum) {
        if (root == NULL) return;
        number = number * 10 + root->val;
        if (root->left == NULL && root->right == NULL) {
            sum += number;
        }
        _sumNumbers(root->left , number, sum);
        _sumNumbers(root->right, number, sum);
    }
};
```

1.146 Surrounded Regions

Problem Description

Given a 2D board containing **'X'** and **'O'**, capture all regions surrounded by **'X'**.
A region is captured by flipping all **'O'**s into **'X'**s in that surrounded region.
For 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
```

Solution

水题

Code

```
class Solution {
public:
    void solve(vector<vector<char> > &board) {
        if (board.size() == 0 || board[0].size() == 0) return ;
        int n = board.size(), m = board[0].size(), dir[][2] = {1, 0, -1, 0,
            0, -1, 0, 1};

        vector<vector<bool> > visted(n, vector<bool> (m, false));
        for (int i = 0; i < n; ++i) {
            if (!visted[i][0]) {
                DFS(board, visted, i, 0, dir);
            }
            if (!visted[i][m - 1]) {
                DFS(board, visted, i, m - 1, dir);
            }
        }
        for (int i = 0; i < m; ++i) {
            if (!visted[0][i]) {
                DFS(board, visted, 0, i, dir);
            }
            if (!visted[n - 1][i]) {
                DFS(board, visted, n - 1, i, dir);
            }
        }
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (!visted[i][j]) {
                    board[i][j] = 'X';
                }
            }
        }
    }
private:
    void DFS(vector<vector<char> > &board, vector<vector<bool> > &visted, int
         x, int y, int dir[][2]) {
        int n = board.size(), m = board[0].size();
        if (!(0 <= x && x < n && 0 <= y && y < m)) return ;
        if (board[x][y] == 'X' || visted[x][y]) return;
        visted[x][y] = true;
        for (int i = 0; i < 4; ++i) {
            DFS(board, visted, x + dir[i][0], y + dir[i][1], dir);
        }
    }
};
```

1.147 Swap Nodes in Pairs

Problem Description

Given a linked list, swap every two adjacent nodes and return its head.
For example,
Given **1->2->3->4**, you should return the list as **2->1->4->3**.
Your algorithm should use only constant space. You may **not** modify the values in the list, only nodes itself can be changed.

Solution

水题

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *swapPairs(ListNode *head) {
        ListNode newhead(0);
        newhead.next = head;
        for (ListNode *p = head, *pre = &newhead; p && p->next; pre = p, p =
            p->next) {
            pre->next = p->next;
            p->next = p->next->next;
            pre->next->next = p;
        }
        return newhead.next;
    }
};
```

1.148 Symmetric Tree

Problem Description

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

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

But the following is not:

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

Note:

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

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

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

The above binary tree is serialized as **"{1,2,3,#,#,4,#,#,5}"**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isSymmetric(TreeNode *root) {
        if (root == NULL) return true;
        return _isSymmetric(root->left, root->right);
    }
private:
    bool _isSymmetric(TreeNode *root1, TreeNode *root2) {
        if (root1 == NULL || root2 == NULL) {
            return root1 == root2;
        }
        if (root1->val != root2->val) return false;
        return _isSymmetric(root1->left, root2->right) && _isSymmetric(root1
            ->right, root2->left);
    }
};
```

1.149 Text Justification

Problem Description

Given an array of words and a length L, format the text such that each line has exactly L characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces **' '** when necessary so that each line has exactly L characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.
For example,
words: [**"This", "is", "an", "example", "of", "text", "justification."**]
L: **16**.
Return the formatted lines as:

```
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
```

Note: Each word is guaranteed not to exceed L in length.
A line other than the last line might contain only one word. What should you do in this case?
In this case, that line should be left-justified.

- A line other than the last line might contain only one word. What should you do in this case?

  In this case, that line should be left-justified.

Solution

水题

Code

```cpp
class Solution {
public:
    vector<string> fullJustify(vector<string> &words, int L) {
        vector<string> ans;
        int p = 0, n = words.size();
        while (p < n) {
            int q = p + 1, tot_len = words[p].size();
            while (q < n && tot_len + words[q].size() + q - p <= L) {
                tot_len += words[q++].size();
            }
            int cntword = q - p;
            int cntspace = cntword - 1;
            string line = words[p];
            if (cntword == 1) {
                for (int i = 0; i < L - words[p].size(); ++i) {
                    line.push_back('␣');
                }
            } else if (q < n) {
                for (int i = 1; i <= cntspace; ++i) {
                    int space = (L - tot_len) / cntspace + (i <= (L - tot_len
                        ) % cntspace);
                    for (int j = 0; j < space; ++j) {
                        line.push_back('␣');
                    }
                    for (int j = 0; j < words[p + i].size(); ++j) {
                        line.push_back(words[p + i][j]);
                    }
                }
            } else {
                for (int i = p + 1; i < q; ++i) {
                    line.push_back('␣');
                    for (int j = 0; j < words[i].size(); ++j) {
                        line.push_back(words[i][j]);
                    }
                }
                int left = L - line.size();
                for (int i = 0; i < left; ++i) {
                    line.push_back('␣');
                }
            }
            ans.push_back(line);
            p = q;
        }
        return ans;
    }
};
```

1.150 Trapping Rain Water

Problem Description

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example,

Given **[0,1,0,2,1,0,1,3,2,1,2,1]**, return **6**.

The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. **Thanks Marcos** for contributing this image!

Solution

水题

Code

```
class Solution {
public:
    int trap(int A[], int n) {
        if (n <= 2) return 0;
        int stk[n], top = 0, ans = 0;
        for (int i = 0; i < n; ++i) {
            while (top && A[stk[top - 1]] <= A[i]) {
                if (top >= 2) {
                    ans += (min(A[stk[top - 2]], A[i])  - A[stk[top - 1]]) *
                        (i - stk[top - 2] - 1);
                }
                -- top;
            }
            stk[top++] = i;
        }
        return ans;
    }
};
```

1.151 Triangle

Problem Description

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.

Solution

水题

Code

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

1.152 Two Sum

Problem Description

Given an array of integers, 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. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

Solution

水题

Code

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int> &numbers, int target) {
        vector<int> ans;
        for (int i = 0; i < numbers.size(); ++i) {
            for (int j = i + 1; j < numbers.size(); ++j) {
                if (numbers[i] + numbers[j] == target) {
                    ans.push_back(i + 1), ans.push_back(j + 1);
                    return ans;
                }
            }
        }
    }
};
```

1.153 Unique Binary Search Trees II

Problem Description

Given n, generate all structurally unique **BST's** (binary search trees) that store values 1...n.
For example,
Given $n = 3$, your program should return all 5 unique BST's shown below.

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

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

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

The above binary tree is serialized as **"{1,2,3,#,#,4,#,#,5}"**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<TreeNode *> generateTrees(int n) {
        return _gen(1, n);
    }
private:
    vector<TreeNode *> _gen(int left, int right) {
        vector<TreeNode *> roots;
        if (left > right)  {
            roots.push_back(NULL);
            return roots;
        }
        for (int k = left; k <= right; ++k) {
            vector<TreeNode *> lefttree = _gen(left, k - 1);
            vector<TreeNode *> righttree = _gen(k + 1, right);
            for (int i = 0; i < lefttree.size(); ++i) {
                for (int j = 0; j < righttree.size(); ++j) {
                    TreeNode *root = new TreeNode(k);
                    root->left = lefttree[i];
                    root->right = righttree[j];
                    roots.push_back(root);
                }
            }
        }
        return roots;
    }
};
```

1.154 Unique Binary Search Trees

Problem Description

Given n, how many structurally unique **BST's** (binary search trees) that store values 1...n?
For example,
Given $n = 3$, there are a total of 5 unique BST's.

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

Solution

水题

Code

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

1.155 Unique Paths II

Problem Description

Follow up for "Unique Paths":
Now consider if some obstacles are added to the grids. How many unique paths would there be?
An obstacle and empty space is marked as **1** and **0** respectively in the grid.

For example,
There is one obstacle in the middle of a 3x3 grid as illustrated below.
The total number of unique paths is **2**.
Note: m and n will be at most 100.

```
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
```

Solution

水题

Code

```cpp
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int> > &obstacleGrid) {
        if (obstacleGrid.size() == 0) return 0;
        int n = obstacleGrid.size(), m = obstacleGrid[0].size();
        vector<vector<int> > f(n, vector<int>(m, 0));

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) if (obstacleGrid[i][j] == 0) {
                if (i && j) {
                    f[i][j] = f[i - 1][j] + f[i][j - 1];
                } else if (i && !j) {
                    f[i][j] = f[i - 1][j];
                } else if (!i && j) {
                    f[i][j] = f[i][j - 1];
                } else {
                    f[i][j] = 1;
                }
            }
        }
        return f[n - 1][m - 1];
    }
};
```

1.156 Unique Paths

Problem Description

A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).
The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).
How many possible unique paths are there?
Above is a 3 x 7 grid. How many possible unique paths are there?
Note: m and n will be at most 100.

Solution

水题

Code

```cpp
class Solution {
public:
    int uniquePaths(int m, int n) {
        return C(n + m -2, n - 1);
    }
private:
    int C(int n, int m) {
        m = min(m, n - m);
        long long ans = 1;
        for (int i = 1; i <= m; ++i) {
            ans = ans * (n - i + 1) / i;
        }
        return ans;
    }
};
```

1.157 Valid Number

Problem Description

Validate if a given string is numeric.
Some examples:
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true
Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.
Update (2014-12-06): New test cases had been added. Thanks **unfounder**'s contribution.

Solution

水题

Code

```cpp
class Solution {
public:
    bool isNumber(const char *s) {
        int left = 0, right = strlen(s) - 1;
        while(left <= right && (s[left] == '␣' || s[left] == '\t' || s[left]
            == '\n')) {
            ++ left;
        }
        while(left <= right && (s[right] == '␣' || s[right] == '\t' || s[
            right] == '\n')) {
            -- right;
        }
        if (left > right) return false;
        int cnt_dot = 0, cnt_e = 0, pos_e = 0;
        for (int i = left; i <= right; ++i) {
            if (!('0' <= s[i] && s[i] <= '9')) {
                if (s[i] == '.') {
                    ++ cnt_dot;
                } else if (s[i] == 'e' || s[i] == 'E'){
                    ++ cnt_e;
                    pos_e = i;
                } else if (s[i] != '-' && s[i] != '+'){
                    return false;
                }
            }
        }
        if (cnt_e > 1 || cnt_dot > 1) {
            return false;
        }
        if ((cnt_e && isFloat(s, left, pos_e - 1) && isInt(s, pos_e + 1,
            right)) || (!cnt_e && isFloat(s, left, right))) {
            return true;
        }
        return false;
    }
private:
    bool isFloat(const char *s, int left, int right) {
        if (left <= right && (s[left] == '-' || s[left] == '+')) {
            ++left;
        }
        if (left > right) return false;
        if (s[left] == '.' && left == right) {
            return false;
        }
        for (int i = left; i <= right; ++i) {
            if (!('0' <= s[i] && s[i] <= '9' || s[i] == '.')) {
                return false;
            }
        }
        return true;
    }

    bool isInt(const char *s, int left, int right) {
        if (left <= right && (s[left] == '-' || s[left] == '+')) {
            ++ left;
        }
        if (left > right) return false;
        for (int i = left; i <= right; ++i) {
            if (!('0' <= s[i] && s[i] <= '9')) {
                return false;
            }
        }
        return true;
    }
};
```

1.158 Valid Palindrome

Problem Description

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

For example,

"A man, a plan, a canal: Panama" is a palindrome.

"race a car" is *not* a palindrome.

Note:

Have you consider that the string might be empty? This is a good question to ask during an interview.

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

Solution

水题

Code

```
class Solution {
public:
    bool isPalindrome(string s) {
        string charset;
        for (int i = 0; i < s.size(); ++i) {
            if ('A' <= s[i] && s[i] <= 'Z') {
                charset.push_back(s[i] - 'A' + 'a');
            } else if ('a' <= s[i] && s[i] <= 'z' || '0' <= s[i] && s[i] <= '
                9') {
                charset.push_back(s[i]);
            }
        }
        for (int i = 0; i < charset.size() / 2; ++i) {
            if (charset[i] != charset[charset.size() - i - 1]) {
                return false;
            }
        }
        return true;
    }
};
```

1.159 Valid Parentheses

Problem Description

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

Solution

水题

Code

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> stk;
        for (int i = 0; i < s.size(); ++i) {
            switch(s[i]) {
                case '(':
                case '[':
                case '{':
                    stk.push(s[i]);
                    break;
                case ')':
                    if (stk.empty() || stk.top() != '(') return false;
                    stk.pop();
                    break;
                case ']':
                    if (stk.empty() || stk.top() != '[') return false;
                    stk.pop();
                    break;
                case '}':
                    if (stk.empty() || stk.top() != '{') return false;
                    stk.pop();
                    break;
            }
        }
        return stk.empty();
    }
};
```

1.160 Valid Sudoku

Problem Description

Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.
The Sudoku board could be partially filled, where empty cells are filled with the character **'.'**.
A partially filled sudoku which is valid.
Note:
A valid Sudoku board (partially filled) is not necessarily solvable. Only the filled cells need to be validated.

Solution

水题

Code

```
class Solution {
public:
    bool isValidSudoku(vector<vector<char> > &board) {
        vector<vector<bool> > row(9, vector<bool> (9, false)), col(9, vector<
            bool> (9, false)), blk(9, vector<bool> (9, false));
        //bool row[9][9] = {false};
        //bool col[9][9] = {false};
        //bool blk[9][9] = {false};

        for (int i = 0; i < 9; ++i) {
            for (int j = 0; j < 9; ++j) {
                if (board[i][j] != '.') {
                    int c = board[i][j] - '1';
                    if (row[i][c] || col[j][c] || blk[i / 3 * 3 + j / 3][c])
                        {
                         return false;
                    }
                    row[i][c] = col[j][c] = blk[i / 3 * 3 + j / 3][c] = true;
                }
            }
        }
        return true;
    }
};
```

1.161 Validate Binary Search Tree

Problem Description

Given a binary tree, determine if it is a valid binary search tree (BST).
Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.
Here's an example:

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

The above binary tree is serialized as **"{1,2,3,#,#,4,#,#,5}"**.

Solution

水题

Code

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isValidBST(TreeNode *root) {
        int rmax, rmin;
        return isValidBST(root, rmax, rmin);
    }
private:
    bool isValidBST(TreeNode *root, int &rmax, int &rmin) {
        if (root == NULL) return true;
        rmax = rmin = root->val;
        if (root->left) {
            int cmax, cmin;
            if (isValidBST(root->left, cmax, cmin) == false || cmax >= root->
                val) {
                return false;
            }
            rmax = max(rmax, cmax), rmin = min(rmin, cmin);
        }
        if (root->right) {
            int cmax, cmin;
            if (isValidBST(root->right, cmax, cmin) == false || cmin <= root
                ->val) {
                return false;
            }
            rmax = max(rmax, cmax), rmin = min(rmin, cmin);
        }
        return true;
    }
};
```

1.162 Wildcard Matching

Problem Description

Implement wildcard pattern matching with support for '?' and '*'.

```
'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).

The matching should cover the <b>entire</b> input string (not partial).

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "*") → true
isMatch("aa", "a*") → true
isMatch("ab", "?*") → true
isMatch("aab", "c*a*b") → false
```

Solution

水题

Code

```
class Solution {
public:
    bool isMatch(const char *s, const char *p) {
        vector<char *> blk;
        int lenp = strlen(p), lens = strlen(s);
        char cp[lenp + 1], *head = (char *)s, *tail;
        strcpy(cp, p);
        for (int i = 0; i <= lenp; ++i) {
            if (cp[i] == '*') {
                cp[i] = 0;
                if (i == 0) {
                    blk.push_back(cp + i);
                }
            } else {
                if (i == 0 || cp[i - 1] == 0) {
                    blk.push_back(cp + i);
                }
            }
        }

        if (blk.size() == 1 && lenp != lens) return false;

        tail = head + lens - strlen(blk[blk.size() - 1]);
        if (strStr(head, blk[0]) != head || strStr(tail, blk[blk.size() - 1]) != tail) {
            return false;
        }

        for (int i = 0; i < blk.size(); ++i) {
            head = strStr(head, blk[i]);
            if (head == NULL) return false;
            head += strlen(blk[i]);
        }
        return true;
    }
private:
    char *strStr(char *s, char *t) {
        int n = strlen(t);
        if (n == 0) return s;
        int next[n + 1], pos = 0;
        next[0] = pos;
        for (int i = 1; i < n; ++i) {
            while(pos && t[pos] != t[i] && t[i] != '?' && t[pos] != '?') {
                pos = next[pos - 1];
            }
            next[i] = pos = pos + (t[pos] == t[i] || t[pos] == '?' || t[i] == '?');
        }

        pos = 0;
        for (int i = 0; s[i]; ++i) {
            while (pos && t[pos] != '?' && t[pos] != s[i]) {
                pos = next[pos - 1];
            }
            pos += (t[pos] == s[i] || t[pos] == '?');
            if (pos == n) return s + i - n + 1;
        }
        return NULL;
    }

};
```

1.163 Word Break II

Problem Description

Given a string *s* and a dictionary of words *dict*, add spaces in *s* to construct a sentence where each word is a valid dictionary word.

Return all such possible sentences.

For example, given *s* = **"catsanddog"**, *dict* = [**"cat"**, **"cats"**, **"and"**, **"sand"**, **"dog"**].

A solution is [**"cats and dog"**, **"cat sand dog"**].

Solution

水题

Code

```
class Solution {
public:
    vector<string> wordBreak(string s, unordered_set<string> &dict) {
        vector<bool> f(s.size() + 1, false);
        vector<string> dicts;
        for (unordered_set<string>::iterator it = dict.begin(); it != dict.
            end(); ++it) {
            dicts.push_back(*it);
        }
        f[0] = true;
        for (int i = 1; i <= s.size(); ++i) {
            for (int j = 0; j < dicts.size(); ++j) {
                int cutlen = dicts[j].length();
                if (i >= cutlen && s.substr(i - cutlen, cutlen) == dicts[j])
                    {
                    f[i] = f[i] | f[i - cutlen];
                }
            }
        }
        vector<string> ans;
        _gen(s, dicts, f, s.size(), "", ans);
        return ans;
    }
private:
    void _gen(const string &s, vector<string> &dicts, vector<bool> &f, int k,
         string now, vector<string> &ans) {
        if (k == 0) {
            ans.push_back(now);
            return ;
        }
        if (f[k] == false) return ;
        for (int i = 0; i < dicts.size(); ++i) {
            int cutlen = dicts[i].length();
            if (k >= cutlen && s.substr(k - cutlen, cutlen) == dicts[i]) {
                _gen(s, dicts, f, k - cutlen, (now == "" ? dicts[i] + now:
                    dicts[i] + "␣" + now), ans);
            }
        }
    }
};
```

1.164 Word Break

Problem Description

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

For example, given s = **"leetcode"**, *dict* = [**"leet", "code"**].

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

Solution

水题

Code

```
class Solution {
public:
    bool wordBreak(string s, unordered_set<string> &dict) {
        vector<bool> f(s.size() + 1, false);
        vector<string> dicts;
        for (unordered_set<string>::iterator it = dict.begin(); it != dict.
            end(); ++it) {
            dicts.push_back(*it);
        }
        f[0] = true;
        for (int i = 1; i <= s.size(); ++i) {
            for (int j = 0; j < dicts.size(); ++j) {
                int cutlen = dicts[j].length();
                if (i >= cutlen && s.substr(i - cutlen, cutlen) == dicts[j])
                    {
                    f[i] = f[i] | f[i - cutlen];
                }
            }
        }
        return f[s.size()];
    }
};
```

1.165 Word Ladder II

Problem Description

Given two words (*start* and *end*), and a dictionary, find all shortest transformation sequence(s) from *start* to *end*, such that:

For example,

Given:

start = **"hit"**

end = **"cog"**

dict = [**"hot","dot","dog","lot","log"**]

Return

```
[
  ["hit","hot","dot","dog","cog"],
  ["hit","hot","lot","log","cog"]
]
```

Note:

- All words have the same length.
- All words contain only lowercase alphabetic characters.

Solution

水题

Code

```
class RotationStr {
public:
    char str[10];
    int id, len;
    RotationStr() {}
    RotationStr(string s, int id) {
        len = s.size();
        this->id = id;
        for (int i = 0; i < len; ++i) {
            str[i] = s[i];
        }
    }
    void rotate() {
        char temp = str[0];
        for (int i = 1; i < len; ++i) {
            str[i - 1] = str[i];
        }
        str[len - 1] = temp;
    }

    bool operator < (const RotationStr &other) const {
        return strncmp(str, other.str, len - 1) < 0;
    }

    bool operator == (const RotationStr &other) const {
        return strncmp(str, other.str, len - 1) == 0;
    }
};

class Solution {
public:
    vector<vector<string> > findLadders(string start, string end,
       unordered_set<string> &dict) {
        allstr.clear();
        res.clear();

        if (dict.size() == 0) return res;
        if (start == end) {
            vector<string> justone(1, start);
            res.push_back(justone);
            return  res;
        }

        allstr.push_back(make_pair(start, 1));
        allstr.push_back(make_pair(end, 0));

        for (unordered_set<string>::iterator pos = dict.begin(); pos != dict.
            end(); ++ pos) {
            if (*pos != start && *pos != end) {
                allstr.push_back(make_pair(*pos, 0));
            }
        }

        _makeLinks();

        int step = _ladderLen();

        stk.clear();
        stk.push_back(1);
        _DFS(step);
        return res;
    }

private:
    int _ladderLen() {
        queue<int> que;
        que.push(0);
        while (!que.empty()) {
            int now_id = que.front();
            que.pop();
            int now_step = allstr[now_id].second;
            if (now_id == 1) return now_step;
            for (int i = 0; i < links[now_id].size(); ++i) {
                if (allstr[links[now_id][i]].second == 0) {
                    allstr[links[now_id][i]].second = now_step + 1;
                    que.push(links[now_id][i]);
                }
            }
        }
        return 0;
    }

    void _makeLinks() {
        vector<RotationStr> dicts;
        links.clear();
        links.resize(allstr.size());
        int len = allstr[0].first.size();
        for (int i = 0; i < allstr.size(); ++i) {
            dicts.push_back(RotationStr(allstr[i].first, i));
        }
        for (int k = 0; k < len; ++k) {
            for (int i = 0; i < dicts.size(); ++i) {
                dicts[i].rotate();
            }
            sort(dicts.begin(), dicts.end());
            for (int i = 0; i < dicts.size(); ++i) {
                for (int j = i + 1; j < dicts.size() && dicts[i] == dicts[j];
                     ++j) {
                    links[dicts[i].id].push_back(dicts[j].id);
                    links[dicts[j].id].push_back(dicts[i].id);
                }
            }
        }
    }

    void _DFS(int step) {
        int now = stk[stk.size() - 1];
        if (allstr[now].second != step + 1 - stk.size()) return ;
        if (stk.size() == step) {
            if (now == 0) {
                vector<string> oneres;
                for (int i = stk.size() - 1; i >= 0; --i) {
                    oneres.push_back(allstr[stk[i]].first);
                }
                res.push_back(oneres);
            }
            return ;
        }
        for (int i = 0; i < links[now].size(); ++i) {
            stk.push_back(links[now][i]);
            _DFS(step);
            stk.pop_back();
        }
    }

    vector<int> stk;
    vector<vector<string> > res;
    vector<pair<string, int> > allstr;
    vector<vector<int> > links;
};
```

1.166 Word Ladder

Problem Description

Given two words (*start* and *end*), and a dictionary, find the length of shortest transformation sequence from *start* to *end*, such that:

For example,
Given:
start = **"hit"**
end = **"cog"**
dict = [**"hot"**,**"dot"**,**"dog"**,**"lot"**,**"log"**]
As one shortest transformation is **"hit" -> "hot" -> "dot" -> "dog" -> "cog"**,
return its length **5**.
Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.

Solution

水题

Code

```
struct TrieTreeNode {
    int c[26];
    bool visted;
    void reset() {
        for (int i = 0; i < 26; ++i) c[i] = 0;
        visted = false;
    }
}nodes[1000000];

class Solution {
public:
    int ladderLength(string start, string end, unordered_set<string> &dict) {
        queue<pair<string, int> > que;

        TrieInit();

        for (unordered_set<string>::iterator pos = dict.begin(); pos != dict.
            end(); ++ pos) {
            TrieInsert(pos->c_str());
        }

        inTrie(start);
        que.push(make_pair(start, 1));

        while (!que.empty()) {
            pair<string, int> nowpair = que.front();
            que.pop();
            string &now_str = nowpair.first;
            int now_step = nowpair.second;

            if (now_str == end) return now_step;

            for (int i = 0; i < now_str.size(); ++i) {
                char oc = now_str[i];
                for (char c = 'a'; c <= 'z'; ++c) {
                    now_str[i] = c;
                    if (inTrie(now_str)) {
                        que.push(make_pair(now_str, now_step + 1));
                    }
                }
                now_str[i] = oc;
            }
        }
        return 0;
    }

private:

    void TrieInit() {
        nodes[0].reset();
        cntnodes = 1;
    }
    void TrieInsert(const char *s, int root = 0) {
        if (*s == 0) {
            return ;
        }
        int id = *s - 'a';
        if (nodes[root].c[id] == 0) {
            nodes[cntnodes].reset();
            nodes[root].c[id] = cntnodes++;

        }
        TrieInsert(s + 1, nodes[root].c[id]);
    }

    bool inTrie(string s) {
        int root = 0;
        for(int i = 0; i < s.size(); ++i) {
            if (nodes[root].c[s[i] - 'a'] == 0) {
                return false;
            }
            root = nodes[root].c[s[i] - 'a'];
        }
        return nodes[root].visted? false: (nodes[root].visted = true);
    }

    int cntnodes;
};
```

1.167 Word Search

Problem Description

Given a 2D board and a word, find if the word exists in the grid.
The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are

those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,

Given **board** =

```
[
  ["ABCE"],
  ["SFCS"],
  ["ADEE"]
]
```

word = **"ABCCED"**, -> returns **true**,
word = **"SEE"**, -> returns **true**,
word = **"ABCB"**, -> returns **false**.

Solution

水题

Code

```
int dir[][2] = {0, 1, 0, -1, 1, 0, -1, 0};

class Solution {
public:
    bool exist(vector<vector<char> > &board, string word) {
        if (board.size() == 0) return false;
        int n = board.size(), m = board[0].size();
        visted = vector<vector<bool> >(n, vector<bool>(m, false));
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (DFS(i, j, board, word, 0)) {
                    return true;
                }
            }
        }
        return  false;
    }

private:
    bool DFS(int x, int y, vector<vector<char> > &board, string &word, int k)
         {
        if (k == word.size()) return true;
        int n = board.size(), m = board[0].size();
        if (!(0 <= x && x < n && 0 <= y && y < m && word[k] == board[x][y] &&
            !visted[x][y])) {
            return false;
        }
        visted[x][y] = true;
        bool found = false;
        for (int i = 0; i < 4; ++i) {
            if (found = DFS(x + dir[i][0], y + dir[i][1], board, word, k + 1)
                ) {
                break;
            }
        }
        visted[x][y] = false;
        return found;
    }
    vector<vector<bool> > visted;
};
```

1.168 ZigZag Conversion

Problem Description

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

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

And then read line by line: **"PAHNAPLSIIGYIR"**
Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string text, int nRows);
```

convert("PAYPALISHIRING", 3) should return **"PAHNAPLSIIGYIR"**.

Solution

水题

Code

```
class Solution {
public:
    string convert(string s, int nRows) {
        if (nRows == 1) return s;
        vector<string> row(nRows);
        for (int i = 0; i < s.size(); ++i) {
            int remind = i % (2 * nRows - 2);
            if (remind < nRows) {
                row[remind].push_back(s[i]);
            } else {
                row[2 * nRows - 2 - remind].push_back(s[i]);
            }
        }
        string ans;
        for (int i = 0; i < nRows; ++i) {
            ans += row[i];
        }
        return ans;
    }
};
```