22. Generate Parentheses 递归

```
1.class Solution {
public:

    void generateParenthese(int left, int right, string s, vector<string>& vs){
        if(left > right )
            return;
        if(left == 0 && right == 0) vs.push_back(s);
        else{
            if(left > 0){
                /*不能这样写因为s改变后也会影响到下面那种情况
                s = s + '(';
                generateParenthese(left-1, right, s, vs);*/
                generateParenthese(left-1, right, s+'(', vs);
            }
            if(right > 0){
                generateParenthese(left, right-1, s+')', vs);
            }
        }
    }

    vector<string> generateParenthesis(int n) {
        vector<string> vs;
        string s = "";
        generateParenthese(n, n, s, vs);
        return vs;
    }
};
```

48. Rotate Image

```
1.class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int m = matrix.size();
        for(int i = 0;  i < m; i++){
            for(int j = i+1; j < m; j++){
                swap(matrix[i][j], matrix[j][i]);
            }
            reverse(matrix[i].begin(), matrix[i].end());
        }

        return;
    }
};
```

49. Group Anagrams

```
1.class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        vector<vector<string>> vvi;
        map<string, vector<string>> m;
        for(int i = 0; i < strs.size(); i++){
            string s = strs[i];
            sort(s.begin(), s.end());
            m[s].push_back(strs[i]);
        }

        for(auto ele : m){
            vvi.push_back(ele.second);
        }
        return vvi;
    }
};
```

50. Pow(x, n)

```
1.class Solution {
public:
    double myPow(double x, int n) {
        if(n < 0){
            return 1.0/Power(x, -n);
        }
        else{
            return Power(x, -n);
        }
    }

    double Power(double x, int n){
        if(n == 0)
            return 1;
        if(n%2 == 0){
            return Power(x, n/2)*Power(x, n/2);
        }
        else {
            return x*Power(x, n/2)*Power(x, n/2);
        }
    }
};
```

47. Permutation II 递归-DFS

1.交换法(主要还是利用了set的特性)

在Permutations的基础上稍加修改，我们用set来保存结果，利用其不会有重复项的特点，然后我们再递归函数中的swap的地方，判断如果i和start不相同，但是nums[i]和nums[start]相同的情况下跳过，继续下一个循环

```
class Solution {
```

3

```
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        set<vector<int>> res;
        permute(nums, 0, res);
        return vector<vector<int>> (res.begin(), res.end());
    }
    void permute(vector<int> &nums, int start, set<vector<int>> &res) {
        if (start >= nums.size()) res.insert(nums);
        for (int i = start; i < nums.size(); ++i) {
            if (i != start && nums[i] == nums[start]) continue;
            swap(nums[i], nums[start]);
            permute(nums, start + 1, res);
            swap(nums[i], nums[start]);
        }
    }
};
```

2.数组标记法

在递归函数中要判断前面一个数和当前的数是否相等，如果相等，前面的数必须已经使用了，即对应的visited中的值为1，当前的数字才能使用，否则需要跳过，这样就不会产生重复排列了

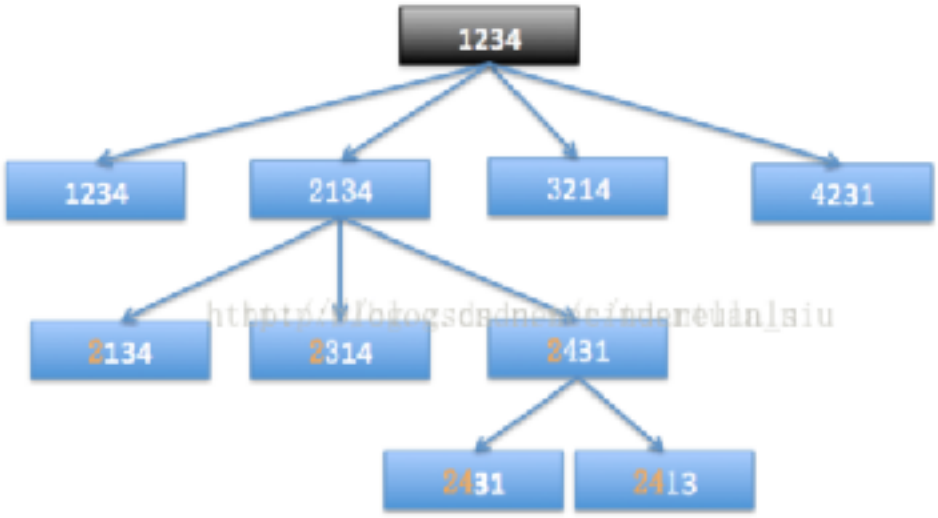
```
class Solution {
public:
    vector<vector<int> > permuteUnique(vector<int> &num) {
        vector<vector<int> > res;
        vector<int> out;
        vector<int> visited(num.size(), 0);
        sort(num.begin(), num.end());
        permuteUniqueDFS(num, 0, visited, out, res);
        return res;
    }
    void permuteUniqueDFS(vector<int> &num, int level, vector<int> &visited,
vector<int> &out, vector<vector<int> > &res) {
        if (level >= num.size()) res.push_back(out);
        else {
            for (int i = 0; i < num.size(); ++i) {
                if (visited[i] == 0) {
                    if (i > 0 && num[i] == num[i - 1] && visited[i - 1] == 0)
continue;
                    visited[i] = 1;
                    out.push_back(num[i]);
                    permuteUniqueDFS(num, level + 1, visited, out, res);
                    out.pop_back();
                    visited[i] = 0;
                }
            }
        }
    }
};
```

46. Permutations(全排列) 递归-DFS(深度优先搜索)

题意：

Input: [1,2,3]
Output:

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



1.交换法

```
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> vvi;
        Permutation(vvi, nums, 0);
        return vvi;
    }

    void Permutation(vector<vector<int>>& vvi, vector<int>& nums, int begin){
        if(begin >= nums.size()-1){
            vvi.push_back(nums);
            return;
        }
        for(int i = begin; i < nums.size(); i++){
```

```
            swap(nums[begin], nums[i]);
            Permutation(vvi, nums, begin+1);
            //nums已经被改变，所以要变回上一步的
            swap(nums[begin], nums[i]);
        }
    }
};
```

参考资料：
https://blog.csdn.net/cinderella_niu/article/details/42930281

2.数组标记法

```
class Solution {
public:
    vector<vector<int> > permute(vector<int> &num) {
        vector<vector<int> > res;
        vector<int> out;
        vector<int> visited(num.size(), 0);
        permuteDFS(num, 0, visited, out, res);
        return res;
    }
    void permuteDFS(vector<int> &num, int level, vector<int> &visited, vector<int>
&out, vector<vector<int> > &res) {
        if (level == num.size()) res.push_back(out);
        else {
            for (int i = 0; i < num.size(); ++i) {
                if (visited[i] == 0) {
                    visited[i] = 1;
                    out.push_back(num[i]);
                    permuteDFS(num, level + 1, visited, out, res);
                    out.pop_back();
                    visited[i] = 0;
                }
            }
        }
    }
};
```

参考资料：
http://www.cnblogs.com/grandyang/p/4358848.html

40. Combination Sum II(回溯)

1.结果不能有重复项，但输入的数可以重复

```
class Solution {
public:
    void Combination(vector<int>& candidates, int target, vector<vector<int>>&
result, vector<int>& vi, int begin){
        if(target == 0){
            result.push_back(vi);
            return;
        }
        for(int i = begin; i < candidates.size(); i++){
            if(candidates[i] > target)
                return;
            if(i > begin && candidates[i] == candidates[i-1])
                continue;
            vi.push_back(candidates[i]);
            Combination(candidates, target-candidates[i], result, vi, i+1);
            vi.pop_back();
        }
    }

    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        vector<vector<int>> result;
        vector<int> vi;
        sort(candidates.begin(), candidates.end());
        Combination(candidates, target, result, vi, 0);
        return result;
    }
};
```

39. Combination Sum(回溯)

1.题目给了我们一个candidates array 和一个 target， 让我们从array 里找到所有的组合，它的和是等于target的。任何组合只要是它的和等于target的话，都需要找到，但是不需要重复的。这道题中是可以重复利用一个数字的，那么我们就需要每次都代入同一个数字，直到它之和达到target 或者它超过了target， 然后在倒退回去一个数字，继续找下一个数字，这种情况肯定是要用递归了。这里是backtracking，每次倒退回一个数字，需要继续递归下去，在倒退，一直重复直到搜索了所有的可能性。

我们可以看原题中的例子：

[2,3,6,7] target 7

2 选2, sum = 2

2+2 选2, sum = 4

2+2+2 选2, sum = 6

7

2+2+2+2 选2，sum = 8 这时候 sum已经超出target，需要返回到上一个数字

2+2+2+3 选3，sum = 9， 也超出了target， 这里我们可以发现，如果是sorted array的话，从小到大，只要一次超出，后面的数字必然也会超出target，所以可以在第一次超出的时候就直接跳出这一个迭代

2+2+3 选3，sum = 7，等于target， 此时返回并且跳出这一个迭代，因为后面的数字肯定超出（array里不会有重复的数字）

2+3 选3，sum = 5，小于target，继续递归同一个数字

2+3+3 选3，sum = 8，超出target，返回上一个数字

2+6 选6，sum = 8，超出target，返回上一个数字

3 选3，这里继续从3开始递归

...

...

...

我们可以看出，一开始有一个迭代从2，一直到7，然后把每一个数字递归下去，包括它自己，每次递归下去的数字，会继续有一个迭代，一旦超出或者等于了，返回前面一个数字继续递归。所以把array sort一下就可以提早跳出那一轮的迭代。

```
class Solution {
public:
    void Combination(vector<int>& candidates, int target, vector<vector<int>>&
result, vector<int>& vi, int begin){
        if(target == 0){
            result.push_back(vi);
            return;
        }
        for(int i = begin; i < candidates.size(); i++){
            if(candidates[i] > target)
                return;
            vi.push_back(candidates[i]);
            Combination(candidates, target-candidates[i], result, vi, i);
            vi.pop_back();
        }
    }

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> result;
        vector<int> vi;
        sort(candidates.begin(), candidates.end());
        Combination(candidates, target, result, vi, 0);
        return result;
    }
};
```

41. First Missing Positive

1.既然不能建立新的数组，那么我们只能覆盖原有数组，我们的思路是把1放在数组第一个位置nums[0]，2放在第二个位置nums[1]，即需要把nums[i]放在nums[nums[i] - 1]上，那么我们遍历整个数组，如果

nums[i] != i + 1，而nums[i]为整数且不大于n，另外nums[i]不等于nums[nums[i] - 1]的话，我们将两者位置调换，如果不满足上述条件直接跳过，最后我们再遍历一遍数组，如果对应位置上的数不正确则返回正确的数

```
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {
        int n = nums.size();
        for(int i = 0; i < n; ++i){
            while(nums[i] > 0 && nums[i] <= nums.size() && nums[nums[i] - 1] != nums[i]){
            //swap函数参数是引用
            swap(nums[nums[i] - 1], nums[i]);
            }
        }

        for(int i = 0; i < n; ++i){
            if(nums[i] != i+1)
                return i+1;
        }

        return n+1;
    }
};
```

小明1，2，3步，总共6步，多少种走法

动态规划：

```
int main() {
    int a[3] = {1, 2, 3};
    int dp[100] = {};
    int n;
     cin >> n;
    dp[0] = 1;
    for(int j =1; j <= n; j++){
        for(int i =0; i <3; i++)
            if(a[i] <= j)dp[j] += dp[j-a[i]];
     }


     cout << dp[n];
    return 0;
}
```

36. Valid Sudoku

```
1.class Solution {
public:
    bool isValidSudoku(vector<vector<char>>& board) {
        for(int i = 0; i < 9; i++){
            int a[9] = {0};
            int b[9] = {0};
            int c[9] = {0};
```

```
            for(int j = 0; j < 9; j++){
                //检查每行
                if(board[i][j] != '.'){
                    if(a[board[i][j]-'1'] == 1)
                        return false;
                    else
                        a[board[i][j]-'1'] = 1;
                }

                //检查每列
                if(board[j][i] != '.'){
                    if(b[board[j][i]-'1'] == 1)
                        return false;
                    else
                        b[board[j][i]-'1'] = 1;
                }

                //检查每个九宫格
                if(board[i/3*3+j/3][i%3*3+j%3] != '.'){
                    if(c[board[i/3*3+j/3][i%3*3+j%3]-'1'] == 1)
                        return false;
                    else
                        c[board[i/3*3+j/3][i%3*3+j%3]-'1'] = 1;
                }
            }
        }
        return true;
    }
};
```

38. Count and Say

```
1.class Solution {
public:
    string CS(string s){
        string ret;
        int c = 1;
        for(int i = 1; i < s.size(); i++){
            if(s[i] != s[i-1]){
                ret += to_string(c) + s[i-1];
                c = 1;
            }else{
                c++;
            }
        }
        ret += to_string(c) + s[s.size()-1];
        return ret;
    }

    string countAndSay(int n) {
        string s = "1";
        while(--n){
```

```
            s = CS(s);
        }
        return s;
    }
};
```

35. Search Insert Position

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

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

O(log(n))二分法

```
1.class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        vector<int> result{-1, -1};
        int low = 0, high = nums.size()-1;
        if(nums.size() == 0 || target < nums[0] || target > nums[nums.size()-1])
            return result;
        while(low <= high){
            int mid = low + (high-low)/2;
            if(nums[mid] < target){
                low = mid+1;
            }else if(nums[mid] > target){
                high = mid-1;
            }else{
                low = mid;
                high = mid;
                while(low-1 >= 0 && nums[low-1] == target)
                    low--;
                while(high+1 < nums.size() && nums[high+1] == target)
                    high++;
                break;
            }
        }
        if(nums[low] == target){
```

```
            result[0] = low;
            result[1] = high;
        }
        return result;
    }
};
```

33. Search in Rotated Sorted Array

这道题让在旋转数组中搜索一个给定值，若存在返回坐标，若不存在返回-1，我们还是考虑二分搜索法，但是这道题的难点在于我们不知道原数组在哪旋转了，我们还是用题目中给的例子来分析，对于数组[0 1 2 4 5 6 7] 共有下列七种旋转方法：

0	1	2	4	5	6	7
7	0	1	2	4	5	6
6	7	0	1	2	4	5
5	6	7	0	1	2	4
4	5	6	7	0	1	2
2	4	5	6	7	0	1
1	2	4	5	6	7	0

二分搜索法的关键在于获得了中间数后，判断下面要搜索左半段还是右半段，我们观察上面红色的数字都是升序的，由此我们可以观察出规律，如果中间的数小于最右边的数，则右半段是有序的，若中间数大于最右边数，则左半段是有序的，我们只要在有序的半段里用首尾两个数组来判断目标值是否在这一区域内，这样就可以确定保留哪半边了，代码如下：

O(log(n))二分法

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

```
                else{
                    left = mid+1;
                }
            }
        }
        return -1;
    }
};
```

32. Longest Valid Parentheses

1.循环1

```
class Solution {
public:
    int longestValidParentheses(string s) {
        int Max = 0;
        int m = s.length();
        stack<int> stk;
        stk.push(-1);
        for(int i = 0; i < m; i++){
            switch(s[i]){
                case ')': if(stk.top() != -1 && s[stk.top()] == '('){stk.pop();
Max = max(Max, i-stk.top()); break;}
                default: stk.push(i);
            }
        }
        return Max;
    }
};
```

2.循环2

```
class Solution {
public:
    int longestValidParentheses(string s) {
        int res = 0, start = 0;
        stack<int> m;
        for(int i = 0; i < s.size(); ++i){
            if(s[i] == '(')
                m.push(i);
            else if(s[i] == ')'){
                if(m.empty())
                    start = i+1;
                else{
                    m.pop();
                    res = m.empty() ? max(res, i-start+1) : max(res, i-m.top());
                }
            }
        }
        return res;
    }
};
```

31. Next Permutation-Pre Permutation

Next Permutation原理：
1.124653

从后往前找到第一个降序 6->4
交换 653 中第一个大于 4 的数字
（如果是6553 那么交换第一个5）
125 643

反转643 接到 125
125346

2.7654321

由于已经是降序排列，所以直接全部变为升序即可

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int size = nums.size();
        for(int i = size-1; i > 0; --i){
            if(nums[i] > nums[i-1]){
                int val = nums[i-1];
                int j = size-1;

                //找到第一个大于val的位置i
                for(; j>= i; --j){
                    if(nums[j] > val)
                        break;
                }

                //交换val和从右到左第一个大于val的值
                swap(nums[i-1], nums[j]);
                int l = i;
                int r = size-1;


                //逆序处理
                while(l < r){
                    swap(nums[l], nums[r]);
                    l++;
                    r--;
                }
                return;
            }
        }
        //走到此说明if里面的条件一直不成立，所以并没有找到val，也就是说nums本来就是按照降序排列的
        int start = 0;
        int last = size - 1;
        while (start < last)
```

```
        {
            swap(nums[start], nums[last]);
            start++;
            last--;
        }
        return;
    }
};


Pre Permutation原理
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int size = nums.size();
        for(int i = size-1; i > 0; --i){
            if(nums[i] < nums[i-1]){
                int val = nums[i-1];
                int j = size-1;

                //找到第一个小于val的位置i
                for(; j>= i; --j){
                    if(nums[j] < val)
                        break;
                }

                //交换val和从右到左第一个大于val的值
                swap(nums[i-1], nums[j]);
                int l = i;
                int r = size-1;


                //逆序处理
                while(l < r){
                    swap(nums[l], nums[r]);
                    l++;
                    r--;
                }
                return;
            }
        }
        //走到此说明if里面的条件一直不成立，所以并没有找到val，也就是说nums本来就是按照降序
排列的
        int start = 0;
        int last = size - 1;
        while (start < last)
        {
            swap(nums[start], nums[last]);
            start++;
            last--;
        }
        return;
```

```
    }
};
```

28. Implement strStr

```
1.class Solution {
public:
    int strStr(string haystack, string needle) {
        if(needle.size() == 0)
            return 0;
        int i = 0, j = 0;
        for(; i < haystack.size() && j < needle.size(); ){
            if(haystack[i] == needle[j]){
                i++;
                j++;
            }else{
                //以防出现这种情况："lllp" "llp"

                i -= j-1;
                j = 0;
            }
        }
        if(j == needle.size())
            return i - j;
        else
            return -1;
    }
};
2.class Solution {
public:
    int strStr(string haystack, string needle) {
        if(needle.length() == 0)
            return 0;
        if(haystack.find(needle) != haystack.npos)
            return haystack.find(needle);
        else
            return -1;
    }
};
```

27. Remove Element

```
1.class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        if(nums.size() == 0)
            return 0;
        int j = 0;
        for(int i = 0; i < nums.size(); i++){
```

```
            if(nums[i] != val){
                nums[j] = nums[i];
                j++;
            }
        }
        nums.erase(nums.begin()+j, nums.end());
        return j;
    }
};
```

26. Remove duplicates from sorted array

1.利用index遍历并且同时改变原来的vector

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if(nums.size() == 0)
            return 0;
        //默认已经保留了第一个元素
        int j = 1;
        for(int i = 1; i < nums.size(); i++){
            if(nums[i] != nums[i-1]){
                nums[j] = nums[i];
                j++;
            }
        }
        nums.erase(nums.begin()+j, nums.end());
        return j;
    }
};
```

2.利用set数据结构

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        set<int> si;
        for(int i = 0; i < nums.size(); i++){
            si.insert(nums[i]);
        }
        nums.clear();
        for(int ele : si){
            nums.push_back(ele);
        }
        return si.size();
    }
};
```

24. Swap Nodes in Pairs

```
/**
* 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) {
        //添加虚拟节点

        ListNode* pre = new ListNode(0);
        ListNode* ret = pre;
        pre->next = head;
        ListNode* cur = head;

        while(cur && cur->next){
            pre->next = cur->next;
            cur->next = pre->next->next;
            pre->next->next = cur;

            //下一轮
            pre = cur;
            cur = pre->next;
        }

        return ret->next;
    }
};
```

23. Merge K sorted lists

```
1./**
* Definition for singly-linked list.
* struct ListNode {
*     int val;
*     ListNode *next;
*     ListNode(int x) : val(x), next(NULL) {}
* };
*/
struct PtrComparator {
    bool operator() (const ListNode* a, ListNode* b) const {
        return a->val < b->val;
    }
};


class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*> &lists) {
        ListNode* head(NULL);
```

```
        if(lists.size() == 0){
             return NULL;
        }

        priority_queue<ListNode*, vector<ListNode*>, PtrComparator> tokens;


        for(int i = 0; i< lists.size(); i++){
            ListNode* ptr = lists[i];
            while(ptr){
                tokens.push(ptr);
                ptr = ptr->next;
            }
        }

        while(!tokens.empty()){
            ListNode* p(tokens.top());
            tokens.pop();
            p->next = head;
            head = p;
        }

        return head;
    }
};
```

数据结构: priority_queue 优先队列

```
1.#include<stdio.h>
#include<stdlib.h>
#include<queue>
#include<iostream>
using namespace std;


//方法1:直接用定义好的priority_queue
vector<int> GetLeastNumbers_Solution(vector<int> input, int k){
    int size = input.size();
    vector<int> ret;


    if(size == 0 || k <= 0 || k > size){
        return ret;
    }
    //priority_queue默认为大顶堆
     /*
      *如果要使用小顶堆的话需要增加使用两个参数:
      * priority_queue<int, vector<int>, greater<int>> q; 小顶堆
      * priority_queue<int, vector<int>, less<int>> q; 大顶堆
      * */
    priority_queue<int> q;
    for(int i = 0; i < size; i++){
```

```
        if(q.size() < k)
            q.push(input[i]);
        else{
            if(input[i] < q.top()){
                q.pop();
                q.push(input[i]);
            }
        }
    }


    while(!q.empty()){
        ret.push_back(q.top());
        q.pop();
    }


    reverse(ret.begin(), ret.end());
    return ret;
}


//自定义的数据结构1
/*struct Node{
    int x, y;
    Node(int a = 0, int b = 0):x(a), y(b){}
};


struct cmp{
    bool operator()(const Node& a, const Node& b){
        if(a.x == b.x)
            return a.y > b.y;
    }
};
priority_queue<Node, vector<Node>, cmp> q;


*/


//自定义的数据结构2
struct Node{
    int x, y;
    Node(int a = 0, int b = 0):x(a), y(b){}
};


bool operator< (const Node& a, const Node& b){
    if(a.x == b.x)
        return a.y > b.y;
    return a.x > b.x;
```

```
}


priority_queue<Node> q;




int main(){
    vector<int> vi{4,5,1,6,2,7,3,8};
    vector<int> result = GetLeastNumbers_Solution(vi, 4);
    for(int ele : result){
         cout << ele;
     }
     cout << endl;
    return 0;
}
```

21. Merge Two Sorted Lists

```
/**
* 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) {
        if(!l1)
            return l2;
        if(!l2)
            return l1;

        ListNode* pre1 = l1;
        ListNode* cur1 = l1;
        ListNode* cur2 = l2;

        while(cur1 && cur2){
            if(cur1->val <= cur2->val){
                pre1 = cur1;
                cur1 = cur1->next;
            }
            else{
                if(cur1 != l1){
                    pre1->next = cur2;
                    cur2 = cur2->next;
```

```
                    pre1->next->next = cur1;
                    pre1 = pre1->next;
                }
                else{
                    ListNode* tmp = cur2->next;
                    cur2->next = cur1;
                    l1 = cur2;
                    pre1 = l1;
                    cur2 = tmp;
                }
            }
        }
        if(cur2){
            pre1->next = cur2;
        }
        return l1;
    }
};
```

20. Valid Parentheses(栈)

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

push()会将一个元素放入stack中。
top()会返回stack中的栈顶元素，返回的是reference，可以就地修改值。
pop()移除栈顶元素，无返回值。
size()返回stack长度。
empty()返回stack是否为空。

19. 删除链表倒数第N个节点 设置虚拟节点通用处理问题

```
/**
```

```
* 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) {
        int length = 0;
        ListNode* first = head;
        while(first){
            length++;
            first = first->next;
        }

        length -= n;

        //设置虚假节点，就是为了能通用的处理单节点问题
        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* pre = dummy;
        ListNode* cur = head;

        while(length > 0){
            pre = cur;
            cur = cur->next;
            length--;
        }
        pre->next = cur->next;
        cur->next = nullptr;
        //不论是否为单个节点，如果是单个节点被删除，也能实现返回为NULL
        return dummy->next;
    }
};
```

18. 4Sum

```
1.class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        vector<vector<int>> vvi;
        sort(nums.begin(), nums.end());
        if(nums.size() < 4) return {};
        for(int i = 0; i < nums.size()-3; i++){
            //不能有该局，因为负数的话还有可能变小

            /*if(nums[i] > target)
                break;*/
            //防止第一个数字重复 避免重复项
```

```
            if(i > 0 && nums[i] == nums[i-1]) continue;
            for(int j =i+1; j < nums.size()-2; j++){
                //防止第二个数字重复

                if(j > i+1 && nums[j] == nums[j-1]) continue;

                int tar = target - nums[i] - nums[j];
                int l = j+1;
                int u = nums.size()-1;

                while(l < u){
                    if(nums[l] + nums[u] == tar){
                        vvi.push_back({nums[i], nums[j], nums[l], nums[u]});
                        //跳过重复项

                        while(l < u && nums[l] == nums[l+1]) l++;
                        while(l < u && nums[u] == nums[u-1]) u--;
                        l++;
                        u--;
                    }else if(nums[l] + nums[u] < tar)
                        l++;
                    else
                        u--;
                }
            }
        }
        return vvi;
    }
};
```

15. 3sum

```
1.class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> vvi;
        sort(nums.begin(), nums.end());
        if(nums.empty() || nums.back() < 0 || nums.front() > 0) return {};
        for(int i = 0; i < nums.size(); i++){
            if(nums[i] > 0) break;
            //保证j能取到重复序列的第一个字符

            if(i > 0 && nums[i] == nums[i-1]) continue;
            int target = 0-nums[i];

            int l = i+1;
            int u = nums.size()-1;

            while(l < u){
                if(nums[l]+nums[u] == target){
                    vvi.push_back({nums[i], nums[l], nums[u]});
                    while(l < u && nums[l] == nums[l+1]) l++;
```

```
                    while(l < u && nums[u] == nums[u-1]) u--;
                    l++;
                    u--;
                }
                else if(nums[l]+nums[u] < target) l++;
                else u--;
            }
        }
        return vvi;
    }
};
```

16. 3SumCloset

```
1.class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        int result = nums[0] + nums[1] + nums[2];
        int closet = abs(nums[0] + nums[1] +nums[2] - target);
        for(int i = 0; i < nums.size(); i++){
            int l = i+1;
            int u = nums.size()-1;
            while(l < u){
                if(closet == 0)
                    return result;
                else{
                    if(abs(nums[i]+nums[l]+nums[u]-target) < closet){
                        result = nums[i]+nums[l]+nums[u];
                        closet = abs(nums[i]+nums[l]+nums[u]-target);
                    }
                    if(nums[i]+nums[l]+nums[u] > target)
                        u--;
                    else
                        l++;
                }
            }
        }
        return result;
    }
};
```

14.Longest common prefix

1.加入sort后运行速度明显变快

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        string result = "";
        string findWord = "";
        if(strs.size() == 0)
            return result;
        sort(strs.begin(), strs.end());
```

```
        for(int i = 1; i <= strs[0].size(); i++){
            findWord = strs[0].substr(0, i);
            for(int j = 1; j < strs.size(); j++){
//不能用find_first_of:find_first_of 函数最容易出错的地方是和find函数搞混。它最大的区别就是如果在一个字符串str1中查找另一个字符串str2，如果str1中含有str2中的任何字符，则就会查找成功，而find则不同；
                if(strs[j].find(findWord) != 0)
                    return result;
            }
            result = findWord;
        }
        return result;
    }
};
```

11. Container with most water

```
1.class Solution {
public:
    int maxArea(vector<int>& height) {
        int Max = 0;
        for(int i = 0; i < height.size(); i++){
            for(int j = i; j < height.size(); j++){
                if(min(height[i], height[j]) * (j-i) > Max){
                    Max = min(height[i], height[j]) * (j-i);
                }
                else
                    continue;
            }
        }
        return Max;
    }
};
```

2.复杂度优化后：

```
class Solution {
public:
    int maxArea(vector<int>& height) {
        int i = 0, Max = 0;
        int j = height.size()-1;
        while(i < j){
            int Min = min(height[i], height[j]);
            Max = max(Max, Min*(j-i));
            while(height[i] <= Min && i < j){
                i++;
            }
            while(height[j] <= Min && i < j){
                j--;
            }
        }
        return Max;
```

```
    }
};
```

3.最优化的方法：

```
class Solution {
public:
    int maxArea(vector<int>& height) {
        int Max = 0;
        int i = 0, j = height.size()-1;
        while(i < j){
            int Min = min(height[i], height[j]);
            Max = max(Max, Min*(j-i));
            if(height[i] < height[j])
                i++;
            else
                j--;
        }
        return Max;
    }
};
```

10. DP解决正则匹配 递归,回溯,动态规划区别联系

1.递归(慢)

```
class Solution {
public:
    bool isMatch(string s, string p) {
        if(p.empty()) return s.empty();
        if(p.size() == 1){
            return (s.size() == 1 && (s[0] == p[0] || p[0] == '.'));
        }
        if(p[1] != '*'){
            if(s.empty()) return false;
            return (s[0] == p[0] || p[0] == '.') && isMatch(s.substr(1),
p.substr(1));
        }
        while(!s.empty() && (s[0] == p[0] || p[0] == '.')){
            if(isMatch(s, p.substr(2))) return true;
            s = s.substr(1);
        }
        return isMatch(s, p.substr(2));
    }
};
```

2.动态规划(快)

```
class Solution {
public:
   bool isMatch(string s, string p) {
```

```
        int m = s.length();
        int n = p.length();
        vector<vector<bool>> a(m+1, vector<bool>(n+1, false));
        a[0][0] = true;
        for(int i = 0; i <= m; i++){
            for(int j = 1; j <= n; j++){
                //j-1才是正常字符串中的字符位置
                //a[i][j]表示s的前i个字符和p的前j个字符match
                if(p[j-1] == '*'){
                    a[i][j] = a[i][j-2] || (i > 0 && (s[i-1] == p[j-2] || p[j-2]
== '.') && a[i-1][j]);
                }else{
                    a[i][j] = i > 0 && a[i-1][j-1] && (s[i-1] == p[j-1] ||
p[j-1] == '.');
                }
            }
        }
        return a[m][n];
    }
};
```

//递归(Recursive):为了描述问题的某一状态，必须用到该状态的上一个状态；而如果要描述上一个状态必须用到上一个状态的上一个状态...这样用来自己定义自己的方法就是递归

回溯(BackTrack):是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或者达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法。(三要素：选择，限制，结束条件)

//动态规划(DP):动态规划就是记忆化了的递归程序。动态规划把很多递归问题的解存储下来，这样就省去了求许多子问题的解，从而达到了快速求解的目的。(三要素:问题的阶段，每个阶段的状态，前后阶段的递推关系)

5. Longest Palindromic Substring

1.动态规划

```
string longestPalindrome(string s) {
        int max_len = 1;
        int start = 0;
        bool a[s.size()][s.size()];
        //a[i][j]表示i和j之间是回文
        //fill_n三个参数:迭代器，计数器，值
        fill_n(&a[0][0],s.size()*s.size(),false);
        for(int i = 0; i < s.size(); i++){
            for(int j = 0; j <= i; j++){
                if(i - j < 2)
                    a[j][i] = (s[i] == s[j]);
                else{
                    a[j][i] = (s[j] == s[i] && a[j+1][i-1]);
                }
```

```
                if(a[j][i] && max_len < (i-j+1)){
                    max_len = i-j+1;
                    start = j;
                }
            }
        }
        return s.substr(start, max_len);
}


int main() {
    string s = "babad";
    cout << longestPalindrome(s) << endl;
    return 0;
}
```

运行结果：

```
bab
```

```
2.class Solution {
public:

    void Palindrome(string s, int left, int right, int &start, int& maxLen){
        int length = 0;
        while(left >= 0 && right < s.size() && s[left] == s[right]){
            left--;
            right++;
        }
        if(right - left - 1 > maxLen){
            maxLen = right - left - 1;
            start = left + 1;
        }
    }

    string longestPalindrome(string s) {
        int left = 0, right = 0, start = 0, maxLen = 0;
        if(s.size() < 2)
            return s;
        for(int i = 0; i < s.size(); i++){
            Palindrome(s, i, i, start, maxLen);
            Palindrome(s, i, i+1, start, maxLen);
        }
        return s.substr(start, maxLen);
    }
};
```

8. myAtoi

```
1.class Solution {
```

```
public:
    int myAtoi(string str) {
        int result = 0;
        int signal = 1;
        int i = 0;
        while(i < str.size() && str[i] == ' '){
            i++;
        }
        if(str[i]  == '-'){
            i++;
            signal = -1;
        }
//如果没有else的话"-+3"就不会被排除(会输出1而不是0)
        else if(str[i]  == '+'){
            i++;
            signal = 1;
        }
        for(; i < str.size(); i++){
            if('0' > str[i] || str[i] > '9')
                return signal*result;
            if(signal*(10.0*result + (str[i]-'0')) < INT_MIN)
                return INT_MIN;
            if(signal*(10.0*result + (str[i]-'0')) > INT_MAX)
                return INT_MAX;
            result = 10*result + (str[i]-'0');
        }
        return signal*result;
    }
};
```

9.Palindrome Number

```
1.-121不是Palindrome Number, 121是Palindrome Number
class Solution {
public:
    bool Palindrome(string s, int i, int j){
        if(j - i >= 2){
            if(s[i] != s[j])
                return false;
            else{
                i++;
                j--;
                return Palindrome(s, i, j);
            }
        }
//j-i<2用来处理中心为一个数字或是两个数字(即奇数位回文数字还是偶数位回文数字)
        else if(j - i < 2){
            if(s[i] == s[j])
                return true;
            else
                return false;
        }
```

```
    }

    bool isPalindrome(int x) {
        if(x < 0)
            return false;
        string s = to_string(x);
        int left = 0, right = s.size()-1;
        return Palindrome(s, left, right);
    }
};
```

7. Reverse Integer

```
1.class Solution {
public:
    int reverse(int x) {
        long long result = 0;
        long long y = abs((long long)x);
        while(y != 0){
            result = 10*result + y%10;
            y /= 10;
            //INT_MAX整型上限
            if( result > INT_MAX)
                return 0;
        }
        return x > 0 ? (int)result : ((-1)*(int)result);
    }
};
```

4. Median of Two Sorted Arrays

```
1.直接将nums1变为合并后重新排好序的数列
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int i = 0, j = 0, k = 0;
        while(i != nums1.size() && j != nums2.size()){
            if(nums1[i] > nums2[j]){
                nums1.insert(nums1.begin()+i, nums2[j]);
                j++;
            }else{
                i++;
            }
        }
        for(k = j; k < nums2.size(); k++){
            nums1.push_back(nums2[k]);
        }
        if(nums1.size() % 2 == 1)
            return (double)nums1[nums1.size() / 2];
        else
            return (double)(nums1[nums1.size() / 2 - 1] + nums1[nums1.size() /
2])/2;
    }
```

```
};
```

运行结果：

```
2.5
```

2.二分法 时间复杂度：O(log(m+n))

如果某个数组没有第K/2个数字，那么我们就淘汰另一个数组的前K/2个数字即可(也就是说第K个数字不在该前K/2个数字中)。

比较这两个数组的第K/2小的数字midVal1和midVal2的大小，如果第一个数组的第K/2个数字小的话，那么说明我们要找的数字肯定不在nums1中的前K/2个数字，所以我们可以将其淘汰。

```
class Solution {
public:
    int findKth(vector<int>& nums1, int i, vector<int>& nums2, int j, int k){
        if(i >= nums1.size())
            return nums2[j+k-1];
        if(j >= nums2.size())
            return nums1[i+k-1];
        if(k == 1)
            return min(nums1[i], nums2[j]);
        int midVal1 = (i+k/2-1 < nums1.size()) ? nums1[i+k/2-1] : INT_MAX;
        int midVal2 = (j+k/2-1 < nums2.size()) ? nums2[j+k/2-1] : INT_MAX;
        if(midVal1 < midVal2){
            return findKth(nums1, i+k/2, nums2, j, k-k/2);
        }else{
            return findKth(nums1, i, nums2, j+k/2, k-k/2);
        }
    }

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int m = nums1.size();
        int n = nums2.size();
        int left = (m+n+1)/2;
        int right = (m+n+2)/2;
        return (findKth(nums1, 0, nums2, 0, left) + findKth(nums1, 0, nums2, 0,
right))/2.0;
    }
};
```

3.最长无重复子串

```
1.class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int max = 0;
        string re;
        for(int i = 0;  i< s.size(); i++){
```

```
            for(int j = i; j < s.size(); j++){
                if(re.find(s[j]) == re.npos){
                    re += s[j];
                    max = re.length() > max ? re.length() : max;
                }
                else{
                    re.clear();
                    break;
                }
            }
        }
        return max;
    }
};
```

2.滑动窗口法

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int Max = 0, pre = -1;
        map<char, int> book;
        for(int i = 0; i < s.size(); i++)
            book[s[i]] = -1;
        for(int i = 0; i < s.size(); i++){
            pre = max(pre, book[s[i]]);
            Max = max(Max, i - pre);
            book[s[i]] = i;
        }
        return Max;
    }
};
```

2. Add Two Numbers

1.main.cpp

```
#include "LinkList.h"
#include <iostream>
using namespace std;


int main() {
   LinkList l1;
   l1.Insert(0, 3);
   l1.Insert(1, 9);
   l1.Insert(2, 9);
   l1.Insert(3, 9);
   l1.Insert(4, 9);
   l1.Insert(5, 9);
   l1.Insert(6, 9);
   l1.Insert(7, 9);
   l1.Insert(8, 9);
```

```
    l1.Insert(9, 9);
    LinkList l2;
    l2.Insert(0, 7);


    ListNode* r = addTwoNumbers(l1.head, l2.head);
    ListNode* rr = r;
    while(rr){
        cout << rr->val << "->" << endl;
        rr = rr->next;
    }
}
```

2.LinkList.h

```
#ifndef LINKEDLIST_LINKLIST_H
#define LINKEDLIST_LINKLIST_H


struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};


class LinkList{
public:
    ListNode *head;
public:
    LinkList(){head = nullptr;}
    //~LinkList();
    bool Insert(int i, int e);
};
ListNode * addTwoNumbers(ListNode* l1, ListNode* l2);


#endif //LINKEDLIST_LINKLIST_H
```

3.LinkList.cpp

```
#include "LinkList.h"
#include <iostream>
using namespace std;
#include <deque>
#include <math.h>


/*LinkList::~LinkList(){
    ListNode *p = head;
    while(p){
        p = head;
```

```
    head = head->next;
    delete p;
  }
}*/


bool LinkList::Insert(int i, int e){
  ListNode *p = head;
  ListNode *s;
  int j = 0;
  if(i == 0){
    s = (ListNode *)new ListNode(e);
    s->next = p;
    head = s;
    return true;
  }
  if(p == nullptr)
    return false;
  while(p && j < i-1){
    p = p->next;
    j++;
  }
  s = (ListNode *)new ListNode(e);
  s->next = p->next;
  p->next = s;
  return true;
}


ListNode * addTwoNumbers(ListNode* l1, ListNode* l2) {
  long int result1 = 0;
  int size_moins_11 = 0;
  ListNode* r1;
  ListNode* p1 = l1;
  ListNode* q1 = l1->next;
  l1->next = nullptr;
  while(q1){
    r1 = q1->next;
    q1->next = p1;
    p1 = q1;
    q1 = r1;
    size_moins_11++;
  }
  l1 = p1;
  int size_iter1 = size_moins_11;
  while(p1){
    result1 += p1->val*(int)pow(10, size_iter1);
    p1 = p1->next;
    size_iter1--;
  }
```

```
    long int result2 = 0;
    int size_moins_12 = 0;
    ListNode* r2;
    ListNode* p2 = l2;
    ListNode* q2 = l2->next;
    l2->next = nullptr;
    while(q2){
        r2 = q2->next;
        q2->next = p2;
        p2 = q2;
        q2 = r2;
        size_moins_12++;
    }
    l2 = p2;
    int size_iter2 = size_moins_12;
    while(p2){
        result2 += p2->val*(long int)pow(10, size_iter2);
        p2 = p2->next;
        size_iter2--;
    }


    long int result3 = result1 + result2;
    deque<int> v3;
    if(result3 == 0)
        v3.push_front(0);
    while(result3 != 0){
        v3.push_front(result3%10);
        result3 = result3/10;
    }


    ListNode* head = nullptr;
    for(auto ele : v3){
        ListNode* p = new ListNode(ele);
        p->next = head;
        head = p;
    }
    return head;
}
```

运行结果:

```
8->
0->
4->
5->
6->
0->
0->
1->
4->
```

36

1->

通过数:**1553 / 1562** test cases passed.

问题: long int 范围不够，long long int 编译器不支持

65. Plus One

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

64. Minimum Path Sum

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

```
        }
        return sum[m-1][n-1];
    }
};
```

63. Unique Paths II

路中间有障碍物

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

        if(obstacleGrid[0][0] == 1)
        {
            return 0;
        }
        else if(m == 1 && n == 1)
        {
            return 1;
        }

        int paths[m][n];

        for(int i = 0; i < m; ++i)
        {
            if(obstacleGrid[i][0] == 1)
            {
                while(i < m)
                {
                    paths[i][0] = 0;
                    ++i;
                }
                break;
            }
            else
            {
                paths[i][0] = 1;
            }
        }

        for(int j = 1; j < n; ++j)
        {
            if(obstacleGrid[0][j] == 1)
            {
                while(j < n)
                {
                    paths[0][j] = 0;
                    ++j;
                }
                break;
```

```
        }
        else
        {
            paths[0][j] = 1;
        }
    }

    for(int i = 1; i < m; ++i)
        for(int j = 1; j < n; ++j)
        {
            if(obstacleGrid[i][j] == 1)
            {
                paths[i][j] = 0;
            }
            else
            {
                paths[i][j] = paths[i][j - 1] + paths[i - 1][j];
            }
        }

    return paths[m - 1][n - 1];
  }
};
```

62. Unique Paths(DP)


```
class Solution {
public:
  int uniquePaths(int m, int n) {
    vector<vector<int>> path(m, vector<int>(n , 1));
    for(int i = 1; i < m; i++){
        for(int j = 1; j < n; j++){
            path[i][j] = path[i-1][j] + path[i][j-1];
        }
    }
    return path[m-1][n-1];
```

```
    }
};
```

59. Spiral Matrix II

```
class Solution {
public:
    vector<vector<int>> generateMatrix(int n) {
        vector<vector<int>> res(n, vector<int>(n, 0));
        int k = 1;
        int left = 0, right = n-1, top = 0, bottom = n-1;
        while(left < right && top < bottom){
            for(int j = left; j < right; j++){
                res[top][j] = k++;
            }
            for(int i = top; i < bottom; i++){
                res[i][right] = k++;
            }
            for(int j = right; j > left; j--){
                res[bottom][j] = k++;
            }
            for(int i = bottom; i > top; i--){
                res[i][left] = k++;
            }
            left++;
            right--;
            top++;
            bottom--;
        }
        if(n%2 != 0){
            res[n/2][n/2] = k;
        }
        return res;
    }
};
```

56. Merge Intervals

```
class Solution {
public:
    static bool comp(const Interval& a, const Interval& b){
        return (a.start < b.start);
    }

    vector<Interval> merge(vector<Interval>& intervals) {
        vector<Interval> res;
        if(intervals.empty())
            return res;
        sort(intervals.begin(), intervals.end(), comp);//sort函数的用法:sort(words.begin(), words.end(), isShorter)
        res.push_back(intervals[0]);
        for(int i = 0; i < intervals.size(); i++){
            if(res.back().end >= intervals[i].start)
                res.back().end = max(res.back().end, intervals[i].end);//如果区间有重叠侧进行区间融合
```

```
            else
                res.push_back(intervals[i]);
        }
        return res;
    }
};
```

54. Spiral Matrix

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        if(matrix.empty()) return {};
        int m = matrix.size();
        int n = matrix[0].size();
        vector<int> spiral(m*n);
        int u = 0, d = m-1, l = 0, r = n-1, k = 0;
        while(true){
            //up
            for(int col = l; col <= r; col++)
                spiral[k++] = matrix[u][col];
            if(++u > d) break;
            //right
            for(int row = u; row <= d; row++)
                spiral[k++] = matrix[row][r];
            if(--r < l) break;
            //down
            for(int col = r; col >= l; col--)
                spiral[k++] = matrix[d][col];
            if(--d < u) break;
            //left
            for(int row = d; row >= u; row--)
                spiral[k++] = matrix[row][l];
            if(++l > r) break;
        }
        return spiral;
    }
};
```

55. Jump Game

```
//贪心算法
    bool canJump(vector<int>& nums) {
        int Max = nums[0];
        for(int i = 0; i < nums.size() - 1; i++){
            Max = max(Max, i+nums[i]);
            if(Max < i+1)
                return false;
            if(Max >= nums.size() - 1)
                return true;

        }
        if(nums.size() == 0)
```

```
            return false;
        if(nums.size() == 1)
            return true;
        return false;
    }
```

参考资料:https://www.cnblogs.com/271934Liao/p/7053406.html

53. Maximum Subarray(动态规划)

```
int maxSubArray(vector<int>& nums) {
        int s=0,p=nums[0];
        for(int i=0;i<nums.size();i++){
            s=max(nums[i],s+nums[i]);
            p=max(p,s);
        }
        return p;
    }
```

反转单链表

```
NODE *LinkList::Reverse(){
    if(head == nullptr || head->next == nullptr)
        return head;
    NODE *p = head;
    NODE *q = head->next;
    NODE *r;
    head->next = nullptr;
    while(q){
        r = q->next;
        q->next = p;
        p = q;
        q = r;
    }
    head = p;
    return head;
}
```

11. Container With Most Water

```
class Solution {
public:
    int maxArea(vector<int>& height) {
        int capa = 0;
        int i = 0;
        int j = height.size()-1;
        while(i < j){
            int h = min(height[i], height[j]);
            capa = max(capa, h*(j-i));
            while(height[i] <= h && i < j)
                i++;
            while(height[j] <= h && i < j)
```

```
            j--;
        }
        return capa;
    }
};
```

7. Reverse Integer

```
class Solution {
public:
    int reverse(int x) {
        long long y = abs((long long)x);
        long long result = 0;
        while(y != 0){
            result = result * 10 + y%10;
            if(result > INT_MAX)
                return 0;
            y /= 10;
        }
        return x > 0 ? (int)result : ((-1)*(int)result);
    }
};
```

先都转化为long long，将大于INT_MAX的返回0，最终的返回值要变为int

50. Pow(x, n)

```
double myPow(double x, int n) {
        double ans = 1;
        unsigned long long p;
        if (n < 0) {
            p = -n;
            x = 1 / x;
        } else {
            p = n;
        }
        while (p) {
            if (p & 1)
                ans *= x;
            x *= x;
            p >>= 1;
        }
        return ans;
```

49. Group Anagrams

Input: `["eat", "tea", "tan", "ate", "nat", "bat"]`,
Output:

```
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
```

43

]

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        vector<vector<string>> result;
        map<string, vector<string>> m;
        for(int i = 0; i < strs.size(); i++){
            string str = strs[i];
            sort(str.begin(), str.end());
            m[str].push_back(strs[i]);
        }
        for(map<string, vector<string>>::iterator it = m.begin(); it != m.end(); it++){
            result.push_back(it->second);
        }
        return result;
    }
};
```

42. Trapping Rain Water

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.

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!

Example:

```
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

```
class Solution {
public:
    int trap(vector<int>& height) {
        int result = 0;
        vector<int> containLeft(height.size(), 0);//必须要初始化后面才能用下标调用
        vector<int> containRight(height.size(), 0);
        int left_max = 0;
        int right_max = 0;
```

```
        for(int i = 0; i < height.size(); i++){
            containLeft[i] = left_max;
            left_max = max(left_max, height[i]);
        }
        for(int i = height.size()-1; i >=0; i--){
           containRight[i] = right_max;
           right_max = max(right_max, height[i]);
        }
        for(int i = 0; i < height.size(); i++){
           int x = min(containLeft[i], containRight[i]);
           int a = x - height[i] > 0 ? x - height[i] : 0;
           result += a;
        }
        return result;
    }
};
```

运行结果: 6

第i块地方的存水量 = min(第i块左边最高的bar高度, 第i块右边最高的bar的高度) - 第i块地方bar的高度.

参考资料: https://blog.csdn.net/cinderella_niu/article/details/43482897

33. Search in Rotated Sorted Array(旋转数组中的二分搜索)

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

```
            }else{
                max = mid-1;
            }
        }
    }
    return -1;
  }
};
```

参考资料:https://segmentfault.com/a/1190000003811864

28. Implement strStr()

```
int strStr(string haystack, string needle) {
    int m = haystack.length(), n = needle.length();
    if (!n) return 0;
    for (int i = 0; i < m - n + 1; i++) {
        int j = 0;
        for (; j < n; j++)
            if(haystack[i + j] != needle[j])
                break;
        if (j == n) return i;
    }
    return -1;
```

记住: string.length()返回的是无符号型，要保证准确性，要将其赋给int型

22. Generate Parentheses(回溯)

1.我的错误写法:

```
class Solution {
public:
  void addpar(vector<string>& vec, string s, int n, int m){
    if(n == 0 && m == 0){
        vec.push_back(s);
        return;
    }
    if(m > 0){
        m-=1;
        s+=")";
        addpar(vec, s, n, m);
    }
    if(n > 0){
        n-=1;
        m+=1;
        s+="(";
        addpar(vec, s, n, m);
    }
  }

  vector<string> generateParenthesis(int n) {
```

```
        vector<string> v;
        addpar(v, "", n, 0);
        return v;
    }
};
```

2.正确写法:

```
classSolution {
public:
    vector<string> generateParenthesis(int n) {
        vector<string> res;
         addingpar(res, "", n, 0);
        return res;
     }
    voidaddingpar(vector<string> &v, string str, int n, int m){
        if(n==0 && m==0) {
             v.push_back(str);
            return;
         }
        if(m > 0){ addingpar(v, str+")", n, m-1); }
        if(n > 0){ addingpar(v, str+"(", n-1, m+1); }
     }
};
```

总结:

由于if(m > 0)和if(n > 0)又相互包含的情况，而该程序要求各自情况间所用变量不能相互影响，所以不能在回溯括号之外改变变量。

6. ZigZag Conversion

```
#include <iostream>
using namespace std;


string convert(string s, int numRows) {
    string s1 = "";
    int p = 1;
    int i = 0;
    char** a = new char*[numRows];
    for(int m = 0; m < numRows; m++){
        a[m] = new char[s.size()];
    }


    for(int w = 0; w < numRows; w++){
        for(int x = 0; x < s.size(); x++){
            a[w][x] = '0';
        }
```

```
    }


    for(; i < numRows-1; i++){
        a[i][0] = s[i];
    }


    while(i<s.size()){
        for(int j = numRows-1; j >= 0; j--){
            a[j][(numRows-1)*p-j] = s[i];
            i++;
        }
        for(int j = 1; j <= numRows-2; j++){
            a[j][(numRows-1)*p] = s[i];
            i++;
        }
        p++;
    }


    for(int i = 0; i < numRows; i++){
        for(int j = 0; j < s.size(); j++){
            if(a[i][j] != '0' && a[i][j] != '\0')
                s1.push_back(a[i][j]);
        }
    }
    return s1;
}


int main() {
    string s = "PAYPALISHIRING";
    cout << convert(s, 4) << endl;
    return 0;
}
```

运行结果:
PINALSIGYAHRPI
通过率:
179 / 1158