Table of Contents

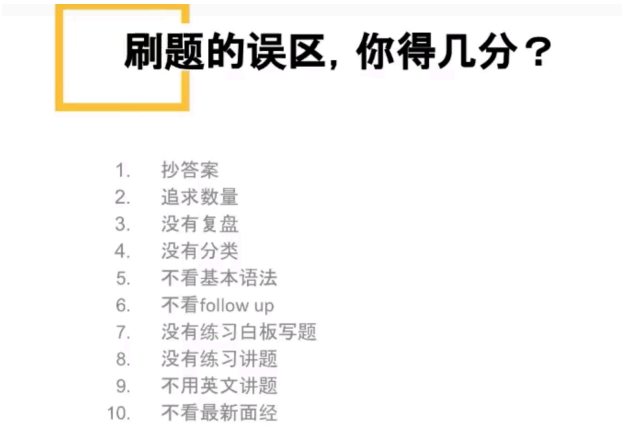
To do list

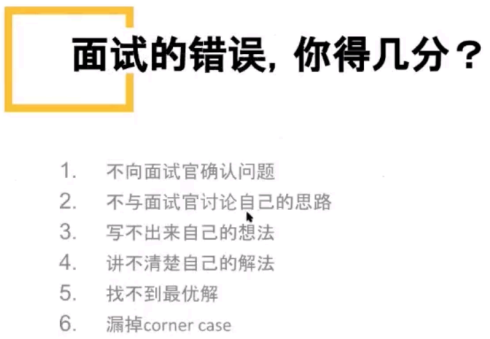
445
54,4,10

To review list

17 backtracking using DFS
19 has question

Overhead





L170

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

```
};

for(int i=nums.size()-1;i>=0;i--){
        if(nums[i]<target)
           return i+1;
    }
    return 0;
}
```

```
        int i=0;
        for(;nums[i]<target&&i<nums.size();i++);
        return i;
```

L205

```
public boolean isIsomorphic(String s1, String s2) {
        Map<Character, Integer> m1 = new HashMap<>();
        Map<Character, Integer> m2 = new HashMap<>();

        for(Integer i = 0; i < s1.length(); i++) {

            if(m1.put(s1.charAt(i), i) != m2.put(s2.charAt(i), i)) {
                return false;
            }
        }
        return true;
    }
class Solution {
public:
    Solution() : s_table(256, -1), t_table(256, -1) {}
    bool isIsomorphic(string s, string t) {
        for (int i = 0; i != s.size(); ++i) {
            if (s_table[s[i]] != t_table[t[i]]) return false;
            s_table[s[i]] = t_table[t[i]] = i;
        }
        return true;
    }
private:
    vector<int> s_table, t_table;
};
```

L290

A. Easy

1. Two Sum//vector/map

Unordered_map, insert, find(return interator), get value from key, vector, push back

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

        unordered_map<int,int> map;
        vector<int> res;
        for(int i=0;i<nums.size();i++){
            if(map.find(nums[i])!=map.end()){
                res.push_back(map[nums[i]]);
                res.push_back(i);
            }
            else map.insert(make_pair(target-nums[i], i));
        }
        return res;

    }


};
```

7. Reverse Integer//check overflow

```
class Solution {
public:
    int reverse(int x) {
        cout<<pow(2,31)-1;
        int res=abs(x);
        int sum=0;
        while(res>0){
            if(sum>(pow(2,31)-1)/10) return 0;//check overflow
            sum*=10;
            sum+=res%10;
            res/=10;
        }
        return x>0?sum:-sum;
    }
```

9. Palindrome Number//string

```
class Solution {
public:
    bool isPalindrome(int x) {

        if(x<0) return false;
        string res=to_string(x);//change to string
        for(long i=0,j=res.size()-1;j>=i;i++,j--){//iterate from both sides at the same time
            if(res[i]!=res[j]) return false;
        }
        return true;
        //or inverse the number to compare;

    }

};
```

13. Roman to Integer//map

```
class Solution {
public:
    int romanToInt(string s) {
        unordered_map<char,int> T=
        {{'I',1},{'V',5},{'X',10},{'L',50},{'C',100},{'D',500},{'M',1000}};
        int sum=T[s.back()];
        for(int n=s.length()-2; n>=0;n--){
            if(T[s[n]]<T[s[n+1]])
                sum-=T[s[n]];
            else
                sum+=T[s[n]];
        }
        return sum;
    }

};
```

14. Longest Common Prefix// string/ recursion

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if(strs.size()<1) return "";
        int l=0;
        int r=strs.size()-1;
        return longestCommonPrefixH(strs,l,r);
    }
    string longestCommonPrefixH(vector<string>& strs,int l,int r){
        int m=l+(r-l)/2;
        if(l<r)
            return
com(longestCommonPrefixH(strs,l,m),longestCommonPrefixH(strs,m+1,r));
        return strs[l];//pay attention here!!!
    }
    string com(string A, string B){
        string res="";
        for(int i=0;i<min(A.size(),B.size());i++){
            if(A[i]==B[i])
                res+=A[i];
            else return res;

        }return res;
    }
};
```

20. Valid Parentheses//stack/switch

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> p;
        for(char &c:s){
            switch(c){
                case'(':p.push(c);break;
                case'{':p.push(c);break;
                case'[':p.push(c);break;
                case')':if(p.empty()||p.top()!='(') return false; else p.pop();break;
                case'}':if(p.empty()||p.top()!='{') return false; else p.pop();break;
                case']':if(p.empty()||p.top()!='[') return false; else p.pop();break;
                default:;
            }
        }
        return p.empty();

};
```

21. Merge Two Sorted Lists//LIst

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2)  {
        if(l1==NULL) return l2;
        if(l2==NULL) return l1;
        //ListNode* res=l1;
        if(l1->val<l2->val){
            l1->next=mergeTwoLists(l1->next,l2);
            return l1;
        }
        else{
            l2->next=mergeTwoLists(l2->next,l1);
            return l2;
        }
    }

};
int main(int argc, const char * argv[]) {
    Solution s1;
    ListNode l1(1);
    ListNode l3(2);
    ListNode l5(4);
    ListNode l2(1);
    ListNode l4(3);
    ListNode l6(4);
    l1.next=&l3;
    l3.next=&l5;
    l2.next=&l4;
    l4.next=&l6;
   // 124   134
    cout<<l1.next->next->val;
    ListNode* t=s1.mergeTwoLists(&l1, &l2);
    cout<<t->val;
```

26. Remove Duplicates from Sorted Array// ！ ！ ！ ！ ！inplace as two pointers/vector

```
using namespace std;
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}

};
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1)  {
        ListNode* h=l1;ListNode* t=l1;
        while(t->next){
            while(t->val==h->val)
                t=t->next;
            h->next=t;
            h=h->next;
        }
        return h;
    }

};
```

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if(nums.size()==0) return 0;
        int i=0;
        for (int j=1;j<nums.size();j++ ){
            if(nums[j]!=nums[i]){
                i++;
                nums[i]=nums[j];
            }

        }
        return i+1;
    }
};
```

27. Remove Element//vector

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

        }
        return i;
    }
};
```

28. Implement strStr()//string compare/ double iterate/boundary case

Brute-Force(暴力解法)O（M*N） and KMP O（M+N）(not required for interview)

```
class Solution {
public:
    int strStr(string haystack, string needle) {
        int h=haystack.length();
        int n=needle.length();
        if(n==0) return 0;
        if(h==0||h<n) return -1;
        for(int i=0;i<h-n+1;i++){
            int j=0;
            //if(haystack.substr(i,n).compare(needle)==0) return i;
            while(j<n){
                if(haystack[i+j]==needle[j]) j++;
                else break;
            }
            if(j==n) return i;
            }
        return -1;
        }
};
```

35. Search Insert Position//brute force/binary search/

```
for(int i=0;i<nums.size();i++){
    if (nums[i]>=target)
        return i;

}return nums.size();
```

```
int l=0;
int r=nums.size()-1;
int m=0;
while(l<=r){
    m=l+(r-l)/2;
    if(nums[m]>target)
        r=m-1;
    else if(nums[m]==target) return m;
    else l=m+1;
}
return l;
```

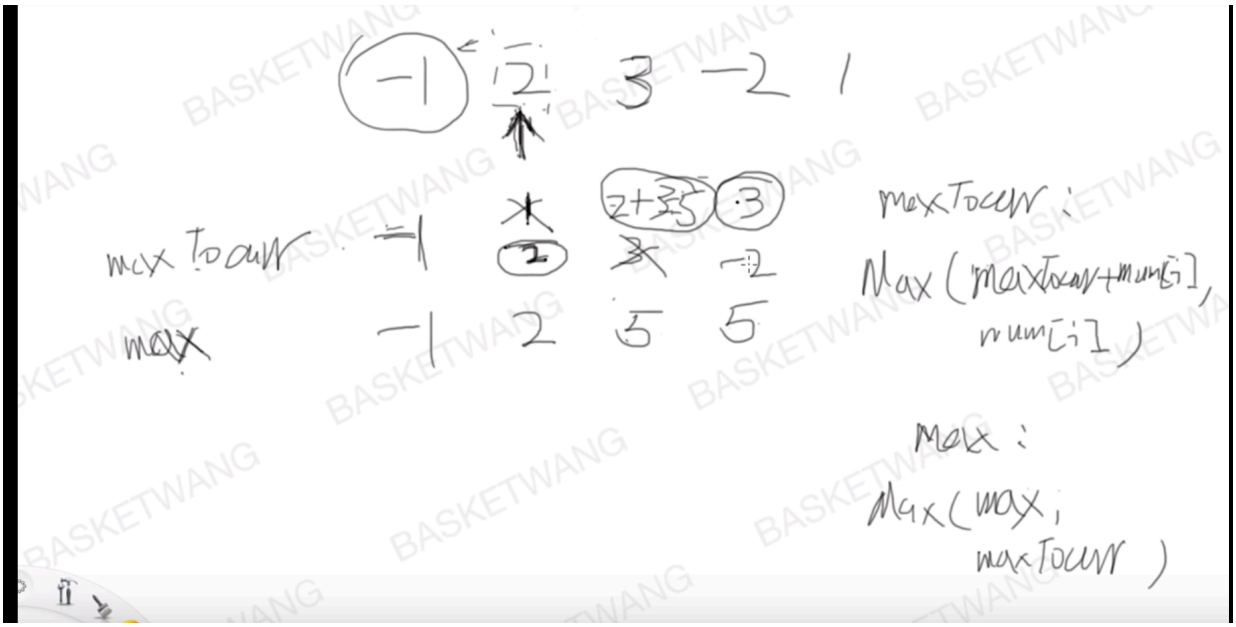
38. Count and Say//

```
        string str="1";
         for(int i=2;i<n+1;i++){

             int count=1;
             char pre=str[0];
             string tmp="";
             for(int idx=1;idx<str.length();idx++){
                 if(str[idx]==pre){
                     count++;
                 }
                 else{
                     tmp+=to_string(count);
                     tmp+=pre;
                     pre=str[idx];
                     count=1;
                 }
             }
             tmp+=to_string(count);
             tmp+=pre;
             str=tmp;
         }

         return str;
    }
};
```

```cpp
class Solution {
public:
    string countAndSay(int n) {
        if(n<=0) return"";

        string str="1";

        for(int i=1;i<n;i++){
            int count=0;
            char prv='.';
            string tmp="";
            for(int idx=0;idx<str.length();idx++){//tranverse str
                if(prv==str[idx]||prv=='.'){//equal to previous
                    count++;
                }
                else {//not equal to previous
                    tmp+=to_string(count);
                    tmp+=prv;
                    count=1;//reset count to 1
                }
                prv=str[idx];     //reset prv
            }
            tmp+=to_string(count);//add the last
            tmp+=prv;
            str=tmp;
        }
        return str;
    }
};
```

```java
 * @param n
 * @return
 */
public String countAndSay(int n) {
    int i = 1;
    String res = "1";
    while (i < n) {
        int count = 0;
        StringBuilder sb = new StringBuilder();
        char c = res.charAt(0);
        for (int j = 0; j <= res.length(); j++) {
            if (j != res.length() && res.charAt(j) == c) {
                count++;
            } else {
                sb.append(count);
                sb.append(c);
                if (j != res.length()) {
                    count = 1;
                    c = res.charAt(j);
                }
            }
        }
        res = sb.toString();
        i++;
    }
    return res;
```

53. Maximum Subarray// Kadane's Algorithm //DP Algorithm



```java
*/
public class MaximumSubarray {
    public int maxSubArray(int[] nums) {
        if (nums == null || nums.length == 0) return 0;
        int max = Integer.MIN_VALUE;
        int prev = Integer.MIN_VALUE;
        for (int n : nums) {
            n = prev > 0 ? n + prev : n;
            max = Math.max(n, max);
            prev = n;
        }
        return max;
    }

    public static void main(String[] args) {
        MaximumSubarray sol = new MaximumSubarray();
        System.out.println(sol.maxSubArray(new int[] {-2,1,-3,4,-1,2,1,-5,4}));
    }
}
```

int[] nums

```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxcur=nums[0];
        int sum=nums[0];

        for(int i=1;i<nums.size();i++){
            maxcur=std::max(nums[i],maxcur+nums[i]);
            sum=std::max(sum,maxcur);
        }
        return sum;
    }

};
```

58. Length of Last Word

Using pointer of C, same in C++, different signature// way to count!!
To do another method

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int len = 0, tail = s.length() - 1;
        while (tail >= 0 && s[tail] == ' ') tail--;
        while (tail >= 0 && s[tail] != ' ') {
            len++;
            tail--;
        }
        return len;
    }
};
```

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int len=0;
        char* c=(char*)s;//???????????????
        while(*c){
            if(*c++!=' ')//here c has been updated to c++
                ++len;
            else if(*c&&*c!=' ')//so have to check *c again
                len=0;

        }
        return len;
    }
};
```

```
int lengthOfLastWord(const char* s) {
        int len = 0;
        while (*s) {
            if (*s++ != ' ')
                ++len;
            else if (*s && *s != ' ')
                len = 0;

        }
        return len;
    }
```

66. Plus One

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

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.
You may assume the integer does not contain any leading zero, except the number 0 itself.

Example 1:

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

Example 2:

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

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        for(int i=digits.size()-1;i>=0;i--){
            if(digits[i]<9){// when came across the first non-9, plus 1 and return the new digits
                digits[i]++;
                return digits;
            }
            else//when came across 9, then change it to 0;
                digits[i]=0;
        }
        //if all 9(change to 0 but not return yet), change them to 0 and add 1 ahead
        digits.insert(digits.begin(), 1);
        return digits;
    }

};
```

67. Add Binary//carry on c

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

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

Example 1:

```
Input: a = "11", b = "1"
Output: "100"
```

Example 2:

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

```cpp
class Solution
{
public:
    string addBinary(string a, string b)
    {
        string s = "";

        int c = 0, i = a.size() - 1, j = b.size() - 1;
        while(i >= 0 || j >= 0 || c == 1)
        {
            c += i >= 0 ? a[i --] - '0' : 0;
            c += j >= 0 ? b[j --] - '0' : 0;
            s = char(c % 2 + '0') + s;
            c /= 2;
        }

        return s;
    }
};
```

69. Sqrt(x)//binary search/similar to 278,35

```
class Solution {
public:
    int mySqrt(int x) {

        int l=1, h=x;// from 1, if x==1, mid=1;works
        while(l<=h){

           long  mid=l+(h-l)/2;//mid should be long as we have mid*mid later
            cout<<mid;
            if(mid*mid==x)
                return mid;
            else if(mid*mid<x)
                l=(int)mid+1;
            else
                h=(int)mid-1;
        }
        return h;// if l and h cross then return h;h is the smaller one
    }
};
```

278. First Bad Version

```
bool isBadVersion(int version);

class Solution {
public:
    int firstBadVersion(int n) {
        int l=1;
        int r=n;
        int m=0;
        while(l<=r){
            m=l+(r-l)/2;
            if(isBadVersion(m)&&!isBadVersion(m-1))
               return m;
            else if(isBadVersion(m)&&isBadVersion(m-1))
                r=m-1;
            else if(!isBadVersion(m)&&!isBadVersion(m-1))
                l=m+1;

        }
        return l;
    }

};
```

70. Climbing Stairs//DP!! /3methods/fibonacci

```
class Solution {
public:
    int climbStairs(int n) {
        vector<int> f(n+1,0);//from 0 to n.
        f[0]=1;
        f[1]=1;
        for(int i=2;i<=n;i++)
            f[i]=f[i-1]+f[i-2];//fibonacci
        return f[n];
    }
};
```

```
    //recurrsion 递归
    int climbStairs(int n) {
        f_=vector<int>(n+1,0);
        return climb(n);
    }
private:
    vector<int> f_;
    int climb(int n){
        if(n<2) return 1;
        if(f_[n]>0) return f_[n];
        f_[n]=climb(n-1)+climb(n-2);
        return f_[n];
    }

};
```

```
  //save space, just use two variables to record the prvious two step
  */
  int climbStairs(int n) {
      int one=1;
      int two=1;
      int curr=1;
      for(int i=2;i<=n;i++){
          curr=one+two;
          two=one;
          one=curr;
      }
      return curr;
  }
```

83. Remove Duplicates from Sorted List//linked list/

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* cur=head;
        while(cur&&cur->next){
            if(cur->val==cur->next->val){
                cur->next=cur->next->next;
            }else
                cur=cur->next;

        }
        return head;
    }
};
```

88. Merge Sorted Array//vector/merge to exist

From end to begin. Pay attention to the first while loop. Decrement at the same time using one line.

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i=m-1;
        int j=n-1;
        int k=m+n-1;
        while(i>=0&&j>=0){
            nums1[k--]=nums1[i]>=nums2[j]?nums1[i--]:nums2[j--];
        }
        while(j>=0)
            nums1[k--]=nums2[j--];

    }
};
```

100. Same Tree//tree/recursion

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(!p&&!q) return true;
        if(!p||!q) return false;
        if(p->val!=q->val) return false;
        else return isSameTree(p->left,q->left)&&isSameTree(p->right,q->right);

        }
};
```

101. Symmetric Tree// tree/recursion

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if(!root) return true;
        return SHelper(root->left,root->right);4
    }
    bool SHelper(TreeNode* left,TreeNode* right){
        if(!left&&!right) return true;
        if(!left||!right) return false;
        if(left->val!=right->val) return false;
        else return SHelper(left->right,right->left)&&SHelper(left->left,right->right);
    }
};
```

104. Maximum Depth of Binary Tree //tree/recursion

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(!root) return 0;//if root==NULL, which is !root
        return std::max(maxDepth(root->left),maxDepth(root->right))+1;
    }
};
```

107. Binary Tree Level Order Traversal II// compare to 102/two dimensional vector, add a row first then add elements

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

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

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

return its bottom-up level order traversal as:

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

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>>res;
        DFS(root,res,0);
        reverse(res.begin(),res.end());// same to 102 except here
        return res;
    }
    void DFS(TreeNode* root, vector<vector<int>>&res,int depth){
        if(!root) return;
        while(res.size()<=depth)//add row
            res.push_back({});//add ele to this row
        res[depth].push_back(root->val);
        DFS(root->left,res,depth+1);
        DFS(root->right,res,depth+1);



    }
};
```

102. Binary Tree Level Order Traversal//BFS

Method1: BFS call private method BFS. No recursion
Method2: DFS, recursion

```
    }
private:
    vector<vector<int>> BFS(TreeNode* root) {
        if(!root) return {};
        vector<vector<int>> ans;
        vector<TreeNode*> curr,next;
        curr.push_back(root);
        while(!curr.empty()) {
            ans.push_back({});
            for(TreeNode* node : curr) {
                ans.back().push_back(node->val);
                if(node->left) next.push_back(node->left);
                if(node->right) next.push_back(node->right);
            }
            curr.swap(next);
            next.clear();
        }
        return ans;
    }
```

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>>res;
        DFS(root,res,0);
        reverse(res.begin(),res.end());// same to 102 except here
        return res;
    }
    void DFS(TreeNode* root, vector<vector<int>>&res,int depth){
        if(!root) return;
        while(res.size()<=depth)
            res.push_back({});

        DFS(root->left,res,depth+1);
        DFS(root->right,res,depth+1);
        res[depth].push_back(root->val);


    }
```

108. Convert Sorted Array to Binary Search Tree//DFS

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return Helper(nums,0,nums.size()-1);
    }
private:
    TreeNode* Helper(vector<int>& nums,int l, int h){
        if(l>h) return NULL;
        int mid=l+(h-l)/2;
        TreeNode* r=new TreeNode(nums[mid]);//choose mid as root for every time
        r->left=Helper(nums,l,mid-1);
        r->right=Helper(nums,mid+1,h);
        return r;
    }
};
```

110. Balanced Binary Tree//DFS

```
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        if(!root) return true;
        if(std::abs(Helper(root->left)-Helper(root->right))<=1)
            return isBalanced(root->left)&&isBalanced(root->right);
        return false;// have to be false
    }
private:
    int Helper(TreeNode* root){
        if(!root) return 0;
        return max(Helper(root->left),Helper(root->right))+1;

    }
};
```

111. Minimum Depth of Binary Tree//DFS

```
class Solution {
public:
    int minDepth(TreeNode* root) {
        if(!root) return 0;
        if(root->left&&!(root->right)) return minDepth(root->left)+1;
        if(!(root->left)&&root->right) return minDepth(root->right)+1;
        return std::min(minDepth(root->left)+1,minDepth(root->right)+1);
    }
};
```

112. Path Sum//Tree/DFS

Method 1: recursion

```
public:
    bool hasPathSum(TreeNode* root, int sum) {
        if(!root) return false;
        if(!(root->left)&&!(root->right)) return sum==root->val;
        return hasPathSum(root->left,sum-(root->val))||hasPathSum(root->right,sum-(root->val));
    }
};
```

Method2: iteration

```
bool hasPathSum(TreeNode* root, int sum) {//iteration
    stack<TreeNode*> node;stack<int> sm;node.push(root);sm.push(sum);
    while(!node.empty()&&root){// pay attention to root!=NULL
        TreeNode* c=node.top();
        node.pop();
        int csum=sm.top();
        sm.pop();
        if(!(c->left)&&!(c->right)&&c->val==csum) return true;
        if(c->right){//can not be !(c->left),why????????
            node.push(c->right);
            sm.push(csum-c->val);
        }
        if(c->left){
            node.push(c->left);
            sm.push(csum-c->val);
        }
    }
    return false;
```

118. Pascal's Triangle//2D array/resize()/insert

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> num;
        num.resize(numRows);
        for(int i=0;i<numRows;i++){
            if(i!=0)
                num[i].push_back(1);//add to the end of a row;
            num[i].insert(num[i].begin(),1);//insert to a position of a row;
            if(i>1){
                for(int j=1;j<i;j++)
                    num[i].insert(num[i].begin()+j,num[i-1][j-1]+num[i-1][j]);
            }
        }
        return num;
    }
};
```

119. Pascal's Triangle II//array

Given a non-negative index *k* w

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {
      vector<int> pre;
        for(int i=0;i<=rowIndex;i++)
        {
            vector<int> tmp(i+1, 1);
 // only when j<i, assign pre addition to tmp, else assign tmp to pre;
            for(int j=1;j<i;j++)
                tmp[j] = pre[j]+pre[j-1];
            pre = tmp;
        }
        return pre;
    }
};
```

here $k \leqslant 33$, return the k^{th} index row of the Pascal's triangle.

121. Best Time to Buy and Sell Stock//array/DP

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        if (prices.size()==0) return 0;//can not be ==NULL, must return 0here.
        int min=prices[0];//INT_MIN
        int max=0;
      for(int i=0;i<prices.size();i++){
          if(prices[i]<min)
              min=prices[i];
          else if(prices[i]-min>max)
              max=prices[i]-min;
      }
        return max;
    }
};
```

122. Best Time to Buy and Sell Stock II//array/greedy

Greedy, do the transaction if have profit

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

125. Valid Palindrome//array/two pointers

```
class Solution {
public:
    bool isPalindrome(string s) {
    int start=0, end=s.size()-1;
        while (start < end) {
            while (!isalpha(s[start])&&!isdigit(s[start])) start++;
            while (!isalpha(s[end])&&!isdigit(s[end])) end--;
            if (start > end) break;
            if (tolower(s[start]) != tolower(s[end])) return false;
            start++, end--;
        }

        return true;
    }
};
```

136. Single Number//hash map/set/bit

```
class Solution {
public:
    //method 1:unordered_map
    int singleNumber(vector<int>& nums) {
        unordered_map<int, int> mp;
        for(auto& it:nums){
            if(mp.find(it)==mp.end())//do not exist
                mp.insert(make_pair(it,0));
            else
                mp[it]++;
        }
        for(auto& it:nums){
            if(mp[it]==0)
                return it;
        }
        return 0;
    };
    //method 2:set
    int singleNumber(vector<int>& nums) {
        set<int> nset;
        for(int n:nums){
            if(nset.find(n)==nset.end())
                nset.insert(n);
            else
                nset.erase(n);//erase is very slow, but
                //can not use :remove(nset.begin(),nset.end(),n);if you know why please
leave a message below.

        }
        return *nset.begin();//has to add *
    }
};
```

Method 3: Bit Manipulation

If we take XOR of zero and some bit, it will return that bit

a $\oplus 0=a$

If we take XOR of two same bits, it will return 0

a $\oplus a=0$

$a \oplus b \oplus a=(a \oplus a) \oplus b=0 \oplus b=b$

```
int singleNumber(vector<int>& nums) {
    int t=0;
    for(int n:nums)
        t^=n;
    return t;
}
```

141. Linked List Cycle//linked list/two pointers

Method1:count :check if exist. is 1 if exist, 0 otherwise

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        unordered_set<ListNode *>  hset;
        while(head){
            if(hset.count(head))
                return true;
            else hset.insert(head);
            head=head->next;
        }
        return false;
    }
};
```

Method2:

```
        auto fast=head;
        auto slow=head;
        while(fast){
            if(!fast->next)
                return false;
            fast=fast->next->next;
            slow=slow->next;
            if(fast==slow)
                return true;
        }
        return false;
```

155. Min Stack//JAVA solution is on leetcode

```
class MinStack {
public:
    void push(int x) {
        s.push(x);
        if (mins.empty() || x<=mins.top()) {
            mins.push(x);
        }
    }

    void pop() {
        int temp = s.top();
        s.pop();
        if (temp == mins.top()) {
            mins.pop();
        }
    }

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

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

private:
    stack<int> s;
    stack<int> mins;
};

/**
 * Your MinStack object will be instantiated and called as such:
```

973. K Closest Points to Origin

```
class Compare{//for priority queue
    public:
    bool operator() (pair<float,vector<int>> A,pair<float,vector<int>> B){
        return A.first>B.first;//compare distance of pair
    }
};
class Solution {
public:
    vector<vector<int>> kClosest(vector<vector<int>>& points, int K) {
        vector<vector<int>> res;// res to return
        priority_queue<pair<float,vector<int>>, vector<pair<float,vector<int>>>,Compare
> q;//to contain distance and point. parameters are data type, container and Compare
        for(auto& point: points){
            float d=cal_dis(point);
            q.push(make_pair(d,point));
        }
        int i=0;
        while(i<K){
            res.push_back(q.top().second);
            q.pop();
            K--;
        }
        return res;
    };

    float cal_dis(vector<int> point){
        return pow(point[0],2)+pow(point[1],2);
    }
};
```

JAVA solution// slow

```
class Solution {
    public int[][] kClosest(int[][] points, int K) {
        if(points.length==0)
            return new int[0][0];// boundary case for 0
        int[][] res=new int[K][2];
        TreeMap<Double, Integer> map=new TreeMap<Double, Integer>();//sorted map
        for(int i=0;i<points.length;i++ ){
            double d=Math.pow(points[i][0],2)+Math.pow(points[i][1],2);
            map.put(d,i);// keep distance and row number
        }
        int i=0;
        for(Map.Entry<Double, Integer> e:map.entrySet()){
            if(i<K){
                res[i][0]=points[e.getValue()][0];
                res[i][1]=points[e.getValue()][1];
                i++;
            }else break;
        }
        return res;
    }
}
```

160. Intersection of Two Linked Lists

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        if(!headA||!headB) return NULL;
        int disa=1,disb=1;
        ListNode *tmpA=headA,*tmpB=headB;
        //get the length of each list
        while(tmpA->next){
            tmpA=tmpA->next;
            disa++;
        }
        while(tmpB->next){
            tmpB=tmpB->next;
            disb++;
        }
       //start from the same length
        if(disa>disb){
            while(disa-disb>0){
                headA=headA->next;disa--;
            }
        }
        else if(disa<disb){
            while(disb-disa>0){
                headB=headB->next;disb--;
            }
        }
        //compare one by one till the end
        while(headB!=headA){
            headB=headB->next;
            headA=headA->next;
        }
        return headA;
    }

};
```

167. Two Sum II - Input array is sorted

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

};
```

168. Excel Sheet Column Title

recursion

```
class Solution {
public:
    string convertToTitle(int n) {
        string r="";
        if(n==0) return r;
        return convertToTitle((n-1)/26)+char((n-1)%26+'A');
        //char((n-1)%26+'A') //if n=26 n%26-'A'-1 too small, wrong
    }
};
```

```
class Solution {
public:
    string convertToTitle(int n) {
        string r="";
        // if(n==0) return r;
        // return convertToTitle((n-1)/26)+char((n-1)%26+'A');
        // //char((n-1)%26+'A') //if n=26 n%26-'A'-1 too small, wrong
        while(n>0){
            int re=(n-1)%26;
            r=char(re+'A')+r;
            n=(n-1)/26;

        }return r;
    }
};
```

169. Majority Element// to be reviewed

8 methods:

花花酱 LeetCode huahualeetcode 169. Majority Element

Method	Time Complexity	Space Complexity	Running time (ms)
Hash Table (unordered_map)	O(n)	O(n)	23
BST (map)	O(nlogk)	O(n)	19
Randomization	O(n)	O(1)	19 - 26
Bit vote	O(32*n)	O(1)	26 ~ 29
Boyer–Moore vote	O(n)	O(1)	19 ~ 23
Full sorting	O(nlogn)	O(1)	26
Partial sorting / partition	O(n) on average	O(1)	19
Divide and conquer	O(n) ~ O(nlogn)	O(logn)	19

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        map<int,int> count;
        const int n=nums.size();
        for(const int num: nums){
            //add to map or update value!!!!!!!!!!!!
            if(++count[num]>n/2) return num;
        }
        return -1;
        //add to map !!
//          int m=20;
//          for(int i=0;i<5;i++){
//              count[i]=m++;
//          }
//          return count[4];

    }
};
```

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int majority = nums.front();
        int count = 0;

        for (const int num : nums) {
            if (num == majority) ++count;
            else if (--count == 0) {
                count = 1;
                majority = num;
            }
        }

        return majority;
    }
};
```

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        srand(time(nullptr));
        const int n = nums.size();
        while (true) {
            const int index = rand() % n;
            const int majority = nums[index];
            int count = 0;
            for (const int num : nums)
                if (num == majority && ++count > n/2) return num;
        }
        return -1;
    }
};
```

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        std::sort(nums.begin(), nums.end());
        return nums[nums.size()/2];
    }
};
```

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        nth_element(nums.begin(), nums.begin() + nums.size() / 2, nums.end());
        return nums[nums.size() / 2];
    }
};
```

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        return majorityElement(nums, 0, nums.size() - 1);
    }
private:
    int majorityElement(const vector<int>& nums, int l, int r) {
        if (l == r) return nums[l];
        const int m = l + (r - l) / 2;
        const int ml = majorityElement(nums, l, m);
        const int mr = majorityElement(nums, m + 1, r);
        if (ml == mr) return ml;
        return count(nums.begin() + l, nums.begin() + r + 1, ml) >
               count(nums.begin() + l, nums.begin() + r + 1, mr)
               ? ml : mr;
    }
};
```

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        const int n = nums.size();
        int majority = 0;
        for (int i = 0; i < 32; ++i) {
            int mask = 1 << i;
            int count = 0;
            for (const int num : nums)
                if (num & mask) ++count;
            if (count > n/2) majority |= mask;
        }
        return majority;
    }
};
```

171. Excel Sheet Column Number

```cpp
class Solution {
public:
    int titleToNumber(string s) {
        int res=0;
        for(auto n:s){// for(int i = 0; i != s.size(); i++)
            res=res*26+n-'A'+1;
        }
        return res;
    }
};
```

172. Factorial Trailing Zeroes

```cpp
class Solution {
public:
    int trailingZeroes(int n) {
        // if (n<5) return 0;
        // if (n<10) return 1;
        // return (n/5+trailingZeroes(n/5));
        //??????????????????????
        int res=0;
        while(n>=5){
            n/=5;
            res+=n;
        }
        return res;
    }
};
```

189. Rotate Array

```cpp
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int tk=k%nums.size();
        auto l=nums.begin();
        auto r=nums.end();
        reverse(l,r);//reverse all
        reverse(l,l+tk);// reverse the left part
        reverse(l+tk,r);// reverse the right part

    }
};
```

```cpp
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n=nums.size();
        int tk=k%n;
        while(tk--){//equal to tk>0,tk--
            auto tmp=nums[n-1];
            for(int i=n-1;i>0;i--){
                nums[i]=nums[i-1];
            }
            nums[0]=tmp;
            // tk--;
        }
    }
};
```

```cpp
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n=nums.size();
        int tk=k%n;
        vector<int> t(n);
        for(int i=0;i<n;i++){
            t[i]=nums[(i+n-tk)%n] ;//when index <k, there is n added as the last round
        }
        for(int i=0;i<n;i++)
            nums[i]=t[i];

    }
};
```

190. Reverse Bits

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t res=0;
        int k=32;
        while(k--){
            res<<=1;
            res|=n&1;//get the last digit
            n>>=1;
        }
        return res;

    }
};
```

191. Number of 1 Bits

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int res=0;
        int k=32;
        while(k--){
            if(n&1&1==1) res++;
            n>>=1;
        }return res;
    }
};
```

198. House Robber

```
class Solution {
public:
    int rob(vector<int>& nums) {
        tmp=vector<int>(nums.size(),-1);
        return rob(nums,nums.size()-1);
    }
private:
    int rob(vector<int>& nums, int n){
        if(n<0) return 0;
        //if(n==1) return nums[0];
        if(tmp[n]>=0) return tmp[n];
        return tmp[n]=max(rob(nums,n-1),rob(nums,n-2)+nums[n]);
    }
    vector<int> tmp;
};
```

445. Add Two Numbers II

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        vector<int> v1,v2;

        while(l1){
            v1.push_back(l1->val);
            l1=l1->next;
        }
        while(l2){
            v2.push_back(l2->val);
            l2=l2->next;
        }
        int m=v1.size(),n=v2.size();
        int sum=0,carry=0;
        ListNode *head=nullptr,*p=nullptr;
        for(int i=m-1,j=n-1;i>=0||j>=0||carry>0;i--,j--){// do not need to compare length of two
vector
            sum=carry;
            if(i>=0)//check if still has elements
                sum+=v1[i];
            if(j>=0)//check if still has elements
                sum+=v2[j];
            carry=sum/10;

            //update head!!!!
            p=new ListNode(sum%10);
            p->next=head;
            head=p;
        }
        return head;
    }
};
```

B. Medium

2. Add Two Numbers// list

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {

        ListNode *cur=new ListNode(-1);
        ListNode *res=cur;
        int c=0,tmp=0,value=0;

        while(l1&&l2){
            tmp=l1->val+l2->val+c;
            value=tmp%10;
            c=tmp/10;

            res->next=new ListNode(value);

            l1=l1->next;
            l2=l2->next;

            res=res->next;
        }

        while(l1){
            res->next=new ListNode((l1->val+c)%10);
            c=(l1->val+c)/10;
            l1=l1->next;
            res=res->next;

        }
        while(l2){
            res->next=new ListNode((l2->val+c)%10);
            c=(l2->val+c)/10;
            l2=l2->next;
            res=res->next;

        }
        cout<<c;
        if(c==1){
            res->next=new ListNode(1);
        }
        return cur->next;
    }
};
```

Res is keeping update, so return cur.next!!!, we have to make cur as begin point!!!!
Method 1 is better

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {

        int i=0,j=0;
        ListNode *res=new ListNode(0);
        ListNode *cur=res;
        int c=0;

        while(l1->next&&l2->next){
            cout<<"here in";
            int tmp=l1->val+l2->val;
            if(c==1){
                res->val=(tmp+1)%10;
                cout<<res->val;
                c=(tmp+1)/10;
            }
            else {
                res->val=tmp%10;
                cout<<res->val;
                c=tmp/10;
            }
            l1=l1->next;
            l2=l2->next;

            res->next=new ListNode(0);
            res=res->next;
        }
        res->val=(l1->val+l2->val+c)%10;
        c=(l1->val+l2->val+c)/10;
        // cout<<c;
        while(l1->next){
            l1=l1->next;
            cout<<"here in 1";
            if(c==1){
                cout<<"here in 11";
                res->next=new ListNode((l1->val+1)%10);
                cout<<l1->val+1;
                c=(l1->val+1)/10;
            }else{
                res->next=new ListNode(l1->val);

            }res=res->next;


        }
        while(l2->next){
            l2=l2->next;
            cout<<"here in 2";
            if(c==1){
                cout<<"here in 21";
                res->next=new ListNode((l2->val+1)%10);
                c=(l2->val+1)/10;
            }else
                res->next=new ListNode(l2->val);
            res=res->next;
        }

        if(c==1) res->next=new ListNode(1);
        return cur;
    }
};
```

3. Longest Substring Without Repeating Characters

```cpp
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        vector<int> map(256,-1);
        int maxm=0, l=-1;//!!!!index add 1 to be the maxm distance,
        //as start (l) subtract minus 1

        for(int i=0;i<s.length();i++){
            if(map[s[i]]>l)//update start point(l), which is the duplicate char
                l=map[s[i]];
            map[s[i]]=i;
            maxm=max(maxm,i-l);
        }
       return maxm;

    }
};
```

JAVA

```java
/**
 * Lin Tao
 * a.ltaocs.com
 */
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int len = s.length();
        int ans = 0;
        int i = 0;
        int j = 0;
        Set<Character> set = new HashSet<>();
        while(i < len && j < len){
            if(!set.contains(s.charAt(j))){
                set.add(s.charAt(j++));
                ans = Math.max(ans, j - i);
            }else{
                set.remove(s.charAt(i++));
            }
        }
        return ans;
    }
}
```

```java
class Solution {
    public int lengthOfLongestSubstring(String s) {
        if (s == null || s.length() == 0) {
            return 0;
        }
        int[] hash = new int[256];
        int begin = 0, end = 0;
        int repeat = 0, res = 0;
        while (end < s.length()) {
            if (hash[s.charAt(end++)]++ > 0) {
                repeat++;
            }
            while (repeat > 0) {
                if (hash[s.charAt(begin++)]-- > 1) {
                    repeat--;
                }
            }
            res = Math.max(res, end - begin);
        }
        return res;
    }
}
```

```java
/*
1. hashmap/hashtable
2. two pointers for size
3. condition variable

while (end < s.length()) {
    if (hash[end] meets req) {
        modify cv
    }
    while (cv meets condition) {

        places for mim size of substring

        if (hash[begin] meets req) {
            modify cv
        }
    }

    places for max size of substring
}
*/
```

5. Longest Palindromic Substring//string, length(), charAt, substr, two dimentional vector initialization,add,

```
class Solution {
from definition: time& space 0(n^2)
public:
    string longestPalindrome(string s) {
        int l=s.length();
        string res="";
        int max=0;
        vector<vector<bool> > v(l, vector<bool>(l));
        for(int j=0;j<l;j++){
            for(int i=0;i<=j;i++){
                v[j][i]=((s[i]==s[j])&&((j-i<=2)||v[j-1][i+1]));
                //(j-i<=2) (consider situitions of 3 elements or less)should before v[j-1][i+1];
                if(v[j][i]){
                    if(j-i+1>max){
                        max=j-i+1;
                        res=s.substr(i,j-i+1);
                    }

                }
            }
        }
        return res;
    }
};
```

```
class Solution {
public:
    //time o(n^2),space o(1)
    //从中心扩散，一个中心点||两个中心点
    string res="";
    int max=0;
    string longestPalindrome(string s) {
        if(s.size()==0) return "";

        for(int i=0;i<s.length();i++){
            helper(s,i,i);//一个
            helper(s,i,i+1);//两个
        }
        return res;
    }
    string helper(string s,int i,int j){

        while(i>=0&&j<s.length()&&s[i]==s[j]){
            i--;j++;
        }
        if(j-i+1>max){
            max=j-i+1;
            res=s.substr(i+1,j-i-1);//as i--,j++, the length should shorten by 2
        }
        return res;

    }
};
```

6. ZigZag Conversion//loop

```
class Solution {
public:
    string convert(string s, int numRows) {
        string res;
        vector<string>tmp(numRows,"");
        int i=0;//track each char
        while(i<s.length()){
            for(int idx=0;idx<numRows&&i<s.length();idx++)//iterate 0-numRows满列
                tmp[idx]+=s[i++];
            for(int idx=numRows-2;idx>0&&i<s.length();idx--)//iterate numRows-2~1半列
                tmp[idx]+=s[i++];
        }
        for(string st:tmp)
            res+=st;
        return res;

    }
};
```

8. String to Integer (atoi)//test overflow/ flag of sign/long empty string

```
class Solution {
public:
    int myAtoi(string str) {
        long res=0;

        int sign=1;
        int start=0;
        while(start < str.length() && str[start] == ' ')
            start++;
        if(start==str.length()) return 0;//empty

        else if(str[start]=='-') {//get the sign
                sign=-1;
                start++;
            }
        else if(str[start]=='+') start++;

        for (int i =start ;i<str.length();i++){
            if(!isdigit(str[i])) return res*sign;
            res=res*10+(str[i]-'0');
            if(sign==1&&res>= INT_MAX) return INT_MAX;
            if(sign==-1&&res> INT_MAX) return INT_MIN;
        }


        return (int)res*sign;
    }
};
```

11. Container With Most Water//vector

```cpp
class Solution {
public:
    int maxArea(vector<int>& height) {
        int l=0,r=height.size()-1;
        int res=0;
        while(l<r){
            int h=min(height[l],height[r]);
            res=max(res,h*(r-l));
            if(height[l]<height[r]) l++;//the shorter one move to next
            else r--;
        }
        return res;
    }
};
```

12. Integer to Roman

```cpp
class Solution {
public:
    string intToRoman(int num) {
        string res;
        const string Thous[]={"","M","MM","MMM"};
        const string Hunds[]={"","C","CC","CCC","CD","D","DC","DCC","DCCC","CM"};
        const string Tens[]={"","X","XX","XXX","XL","L","LX","LXX","LXXX","XC"};
        const string Ones[]={"","I","II","III","IV","V","VI","VII","VIII","IX"};
        res+=Thous[(int)(num/1000)%10];
        res+=Hunds[(int)(num/100)%10];
        res+=Tens[(int)(num/10)%10];
        res+=Ones[(int)num%10];

        return res;
    }
};
```

```cpp
class Solution {
public:
    string intToRoman(int num) {
        string res = "";
        vector<int> val{1000,900,500,400,100,90,50,40,10,9,5,4,1};
        vector<string> str{"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};
        for (int i = 0; i < val.size(); ++i) {
            while (num >= val[i]) {
                num -= val[i];
                res += str[i];
            }
        }
        return res;
    }
};
```

15. 3Sum//2D vector/2 pointer

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

        for(int i=0;i<nums.size();i++){
            int negCur=-nums[i];
            int frontp=i+1;
            int backp=nums.size()-1;

            while(frontp<backp){
                //less than target, front point shift
                if(nums[frontp]+nums[backp]<negCur){
                    frontp++;
                }
                //larger than target, front point shift
                else if(nums[frontp]+nums[backp]>negCur){
                    backp--;
                }
                //found and push

                else{

                    int front=nums[frontp];
                    int back=nums[backp];
                    res.push_back({});
                    res.back().push_back(-negCur);
                    res.back().push_back(front);
                    res.back().push_back(back);
                    //front point duplicate then roll
                    while(frontp<backp&&nums[frontp]==front){
                        frontp++;
                    }
                   //back point duplicate then roll
                   while(frontp<backp&&nums[backp]==back){
                       backp--;
                   }

                }
            }
            // if the first is duplicate
            while(i+1<nums.size()&&nums[i+1]==-negCur)
                i++;
        }
           return res;
        }

};
```

16. 3Sum Closest//three pointers

```
class Solution {
public:
    //choose three nums which are closest to target and get their sum
    int threeSumClosest(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        int res=nums[0]+nums[1]+nums[2];
        for(int i=0;i<nums.size()-2;i++){
            int frontp=i+1;
            int backp=nums.size()-1;
            while(frontp<backp){
                int sum=nums[i]+nums[frontp]+nums[backp];
                res=abs(sum-target)<abs(res-target)?sum:res;
                if (sum<target) frontp++;
                else backp--;

            }
        }
        return res;
    }
};
```

17. Letter Combinations of a Phone Number//BFS/DFS/Backtracking/

Time complexity(4^n)

Space complexity(4^n+n)/ ($2*4^n$)

Example: backtracking to solve maze

```
bool R_solve_maze(list<int> maze[], int start, int finish, unordered_set<int>& visited, list<int>& path) {
    if (start == finish) {
        path.push_front(start);
        return true;
    }

    visited.insert(start);
    list<int>::iterator iter = maze[start].begin();
    while (iter != maze[start].end())
    {
        if (visited.count(*iter) == 0) {
            if (R_solve_maze(maze, *iter, finish, visited, path)) {
                path.push_front(start);
                return true;
            }
        }
        iter++;
    }
    return false;
}

list<int> R_solve_maze(list<int> maze[], int start, int finish) {
    unordered_set<int> visited;
    list<int> solution_path;
    R_solve_maze(maze, start, finish, visited, solution_path);
    return solution_path;
}
```

Backtracking: using DFS

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        if(digits.empty()) return{};
        vector<string> digi={" ","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};
        vector<string> ans;
        DFS(digits,ans,"",0,digi);
        return ans;
    }

    void DFS(string& digits,vector<string>& ans, string cur, int l,vector<string>& digi)
{
        if(digits.length()==l){
            ans.push_back(cur);
            return;
        }
        for(char e:digi[digits[l]-'0']){
            cur.push_back(e);//add e into cur
            DFS(digits,ans,cur,l+1,digi);
            cur.pop_back();//pop to update another e
        }

    }

};
```

BFS:

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        if(digits.empty()) return{};
        vector<string> digi={" ","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};
        vector<string> ans{""};
        for(int i=0;i<digits.length();i++){
            vector<string> tmp;
            for(string e:ans){
                for(char d:digi[digits[i]-'0'])
                    tmp.push_back(e+d);
            }
            ans.swap(tmp);
        }
        return ans;
    }

};
```

18. 4Sum

```
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        if(nums.empty()||nums.size()<4) return{};
        vector<vector<int>> res;
        sort(nums.begin(),nums.end());//first sort
        for(int i=0;i<nums.size()-3;i++){
            if(i>0&&nums[i-1]==nums[i]) continue;//check duplicate
            for(int j=i+1;j<nums.size()-2;j++){
                if(j>i+1&&nums[j-1]==nums[j]) continue;//check duplicate
                int l=j+1,h=nums.size()-1;
                while(l<h){
                    int sum=nums[i]+nums[j]+nums[l]+nums[h];
                    if(sum==target){
                        res.push_back({});
                        res.back().push_back(nums[i]);
                        res.back().push_back(nums[j]);
                        res.back().push_back(nums[l]);
                        res.back().push_back(nums[h]);
                        while(l<h&&nums[l]==nums[l+1]) l++;//check duplicate
                        while(l<h&&nums[h]==nums[h-1]) h--; //check duplicate
                        l++;
                        h--;
                    }
                    else if(sum>target)
                        h--;
                    else
                        l++;
            }
            }
        }
        return res;
    }
};
```

19. Remove Nth Node From End of List// pointer/

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode dummy(0);
        dummy.next=head;
        ListNode* slow=&dummy;
        ListNode* fast=&dummy;
        for(int i=0;i<n;i++){
            fast=fast->next;//make fast faster n steps than slow
        }
        while(fast->next){
            fast=fast->next;//stop at the end
            slow=slow->next;//stop before the one we need to delete
        }
        slow->next=slow->next->next;
        return dummy.next;//in case delete the head

    }
};
```

??

```
class Solution
{
public:
    ListNode* removeNthFromEnd(ListNode* head, int n)
    {
        ListNode** t1 = &head, *t2 = head;
        for(int i = 1; i < n; ++i)
        {
            t2 = t2->next;
        }
        while(t2->next != NULL)
        {
            t1 = &((*t1)->next);
            t2 = t2->next;
        }
        *t1 = (*t1)->next;
        return head;
    }
};
```

22. Generate Parentheses// Recursion

```
class Solution {
public:
    vector<string> generateParenthesis(int n) {
        vector<string> res;
        generateParenthesisH(res,n,"",0,0);
        return res;
    }
    void generateParenthesisH(vector<string>& res,const int& n,string cur,int l,int r){
        if(r==n){
            res.push_back(cur);
            return;
        }
        if(l<n)
            generateParenthesisH(res,n, cur+'(',l+1,r);

        if(r<l)
            generateParenthesisH(res,n, cur+')',l,r+1);

    }
};
```

```
class Solution {

public:
    vector<string> res;
    vector<string> generateParenthesis(int n) {
        help(n,1,0,"(");
        return res;
    }

    void help(int n, int fc, int mc, string s){
        if(fc==n&&mc==n){
            res.push_back(s);
            return ;
        }
        if(fc<n){
            help(n,fc+1,mc, s+"(");
        }
        if(mc<fc){
            help(n,fc,mc+1, s+")");
        }
    }
};
```

24. Swap Nodes in Pairs//dummy node/swap

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        ListNode dummy=ListNode(0);
        dummy.next=head;
        ListNode* cur=&dummy;

        while(cur->next&&cur->next->next){
            swap(cur);
            cur=cur->next->next;
        }
        return dummy.next;
    }
    void swap(ListNode* pre){
        ListNode* dummy=pre->next;
        pre->next=dummy->next;
        dummy->next=dummy->next->next;
        pre->next->next=dummy;

    }
};
```

C++

Pointer-pointer `pp` points to the pointer to the current node. So at first, `pp` points to `head`, and later it points to the `next` field of ListNodes. Additionally, for convenience and clarity, pointers `a` and `b` point to the current node and the next node.

We need to go from `*pp == a -> b -> (b->next)` to `*pp == b -> a -> (b->next)`. The first three lines inside the loop do that, setting those three pointers (from right to left). The fourth line moves `pp` to the next pair.

```
ListNode* swapPairs(ListNode* head) {
    ListNode **pp = &head, *a, *b;
    while ((a = *pp) && (b = a->next)) {
        a->next = b->next;
        b->next = a;
        *pp = b;
        pp = &(a->next);
    }
    return head;
}
```

Python

Here, `pre` is the previous node. Since the head doesn't have a previous node, I just use `self` instead. Again, `a` is the current node and `b` is the next node.

To go from `pre -> a -> b -> b.next` to `pre -> b -> a -> b.next`, we need to change those three references. Instead of thinking about in what order I change them, I just change all three at once.

```
def swapPairs(self, head):
    pre, pre.next = self, head
    while pre.next and pre.next.next:
        a = pre.next
        b = a.next
        pre.next, b.next, a.next = b, a, b.next
        pre = a
    return self.next
```

Recursion:

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        if(head == NULL)
            return NULL;
        if(head->next == NULL)
            return head;

        ListNode* next = head->next;
        head->next = swapPairs(next->next);
        next->next = head;

        return next;
    }
};
```

29. Divide Two Integers

```
class Solution {
public:
    int divide(int dividend, int divisor) {
        int ans=0;

        if(divisor==0) return INT_MAX;
        if(divisor==1) return dividend;
        if(divisor==-1) {
            if(dividend==INT_MIN) return INT_MAX;
            else return -dividend;
        }

        bool e=dividend>0;
        bool s=divisor>0;
        int sign=e^s?-1:1;

        long de=labs(dividend),dv=labs(divisor);
        while(de>=dv){
            long tmp=dv;
            int m=1;//from 1, not 0, 0 shift will always be 0
            while(tmp<<1<=de){//check here, not operate, operating is in next step
                tmp<<=1;
                m<<=1;
            }
            de-=tmp;//get the difference to do another pass
            ans+=m;
        }
        return ans*sign;
    }
};
```

31. Next Permutation

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int n=nums.size();
        int i=0;
        int j=0;
        for(i=n-2;i>=0;i--){
            if(nums[i]<nums[i+1])//find the last position i of increasing
                break;
        }
        if(i<0) return reverse(nums.begin(),nums.end());//not find
        //find the smallest digit that is bigger than nums[i]
        for(j=n-1;j>i;j--){
            if(nums[j]>nums[i])
                break;
        }
        swap(nums[i],nums[j]);
        reverse(nums.begin()+i+1,nums.end());//reverse the descending part

    }
};
```

33. Search in Rotated Sorted Array

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

            if(nums[start]<nums[mid]){//pivot on the right, first part is longer
                if(nums[mid]>target&&nums[start]<=target)
                    end=mid;
                else start=mid;
            }
            else if(nums[mid]<nums[end]){//pivot on the left, first part is shorter
                if(nums[mid]<target&&nums[end]>=target)
                    start=mid;
                else end=mid;
            }
        }
        if(nums[start]==target) return start;
        if(nums[end]==target) return end;
        return -1;

    }
};
```

34. Find First and Last Position of Element in Sorted Array//binary search/lower bound

```
class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        vector<int> res;
        int idx1=lower_bound(nums,target);
        int idx2=lower_bound(nums,target+1)-1;
        if(idx1<nums.size()&&nums[idx1]==target)
            return{idx1,idx2};
        else return{-1,-1};

    }
    //if exist, find the lower bound, if not exist find the position of it
    int lower_bound(vector<int>&nums,int target){
        int l=0, r=nums.size()-1;
        while(l<=r){
            int mid=l+(r-l)/2;
            if(nums[mid]<target){
                l=mid+1;
            }else r=mid-1;

        }
        return l;
    }
};
```

36. Valid Sudoku//map numbers to vector

```
public:
    bool isValidSudoku(vector<vector<char>>& board) {
        vector<vector<int>> row(9,vector<int>(9,0));
        vector<vector<int>> col(9,vector<int>(9,0));
        vector<vector<int>> box(9,vector<int>(9,0));
        for(int i=0;i<9;i++){
            for(int j=0;j<9;j++){
                if(board[i][j]!='.'){
                    int tmp=board[i][j]-'0'-1,k=i/3*3+j/3;
                    if(row[i][tmp]||col[j][tmp]||box[k][tmp])
                        return false;
                    row[i][tmp]=col[j][tmp]=box[k][tmp]=1;

                }

            }
        }
        return true;
    }
```

39. Combination Sum//DFS

```
P(nums, d, n, used, curr, ans):
  if d == n:
    ans.append(curr)
    return

  for i = 0 to len(nums):
    if used[i]: continue
    used[i] = True
    curr.push(nums[i])
    P(nums, d + 1, n, curr, ans)
    curr.pop()
    used[i] = False
```

Permutation P(n, d)

```
P([1, 2, 3], 0, 3, [False]*3, [], ans)
[[1,2,3],[1,3,2],
 [2,1,3],[2,3,1],
 [3,1,2],[3,2,1]]

P([1, 2, 3], 0, 2, [False]*3, [], ans)
[[1,2], [1,3], [2,1], [2,3], [3,1], [3,2]]
```

```
C(nums, d, n, s, curr, ans):
  if d == n:
    ans.append(curr)
    return

  for i = s to len(nums)
    curr.push(nums[i])
    C(nums, d + 1, n, i + 1, curr, ans)
    curr.pop()
```

Combination C(n, d)

```
C([1, 2, 3], 0, 3, [], ans)
[[1,2,3]]

C([1, 2, 3], 0, 2, [], ans)
[[1,2], [1,3], [2,3]]
```

```cpp
class Solution {
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> res;
        sort(candidates.begin(),candidates.end());
        DFS(candidates,target,res,{},0);
        return res;
    }
    void DFS(vector<int>& candidates, int target,vector<vector<int>>& res,vector<int>
cur,int idx){

        if(target==0){
            res.push_back(cur);
        }
        for(int i=idx;i<candidates.size();i++){
            if(candidates[i]>target) break;
            cur.push_back(candidates[i]);
            DFS(candidates,target-candidates[i],res,cur,i);
            cur.pop_back();
        }
    }
};
```

40. Combination Sum II

Method1： Set

```
class Solution {
public:
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        set<vector<int>> res;//避免duplicate ele 造成duplicate result
        sort(candidates.begin(),candidates.end());
        vector<int> cur;
        DFS(candidates,target,res,cur,0);
        return vector(res.begin(),res.end());
    }
    void DFS(vector<int>& candidates, int target,set<vector<int>>& res,vector<int>
cur,int idx){

        if(target==0){
            res.insert(cur);
        }
        for(int i=idx;i<candidates.size();i++){
            if(candidates[i]>target) break;
            cur.push_back(candidates[i]);
            DFS(candidates,target-candidates[i],res,cur,i+1);
            cur.pop_back();
        }
    }
};
```

Method2: skip the duplicate element

```
class Solution {
public:
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        vector<vector<int>> res;
        sort(candidates.begin(),candidates.end());
        vector<int> cur;
        DFS(candidates,target,res,cur,0);
        return res;
    }
    void DFS(vector<int>& candidates, int target,vector<vector<int>>& res,vector<int>
cur,int idx){

        if(target==0){
            res.push_back(cur);
        }
        for(int i=idx;i<candidates.size();i++){
            if(candidates[i]>target) break;
            if(i>idx&&candidates[i]==candidates[i-1]) continue;//skip the duplicate ele
            cur.push_back(candidates[i]);
            DFS(candidates,target-candidates[i],res,cur,i+1);
            cur.pop_back();
        }
    }
};
```

43. Multiply Strings//string

```
class Solution {
public:
    string multiply(string num1, string num2) {
        if(num1=="0"||num2=="0") return "0";
        string res="";
        int l=num1.size()+num2.size();
        vector<int> tmp(l,0);
        for(int i=num1.size()-1;i>=0;--i){
            for(int j=num2.size()-1;j>=0;--j){
                int a=num1[i]-'0';
                int b=num2[j]-'0';
                int mul=a*b;
                int low=i+j+1;
                int hi=i+j;
                mul+=tmp[low];
                tmp[low]=mul%10;
                tmp[hi]+=mul/10;

            }
        }
        bool s=false;
        for(int i=0;i<l;i++){
            if(tmp[i]!=0||s){
                s=true;
                res+=char(tmp[i]+'0');
            }
        }
        return res;
    }
};
```

```
string multiply(string num1, string num2) {
    string sum(num1.size() + num2.size(), '0');

    for (int i = num1.size() - 1; 0 <= i; --i) {
        int carry = 0;
        for (int j = num2.size() - 1; 0 <= j; --j) {
            int tmp = (sum[i + j + 1] - '0') + (num1[i] - '0') * (num2[j] - '0') + carry;
            sum[i + j + 1] = tmp % 10 + '0';
            carry = tmp / 10;
        }
        sum[i] += carry;
    }

    size_t startpos = sum.find_first_not_of("0");
    if (string::npos != startpos) {
        return sum.substr(startpos);
    }
    return "0";
}
```

46. Permutations//distinct elements

```
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> res;
        vector<bool> used(nums.size(),false);
        DFS(nums,res,used,{});
        return res;
    }
    void DFS(vector<int>& nums,vector<vector<int>> &res,vector<bool>& used,vector<int>
cur){
        if(cur.size()==nums.size())
            res.push_back(cur);
        for(int i=0;i<nums.size();++i){
            if(!used[i]){
                used[i]=true;
                cur.push_back(nums[i]);
                DFS(nums,res,used,cur);
                cur.pop_back();
                used[i]=false;
            }

        }
    }

};
```

47. Permutations II//duplicate elements inside//DFS

```
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        vector<vector<int>> res;
        vector<bool> used(nums.size(),false);
        DFS(nums,res,used,{});
        return res;
    }
    void DFS(vector<int>& nums,vector<vector<int>> &res,vector<bool>& used,vector<int>
cur){
        if(cur.size()==nums.size())
            res.push_back(cur);
        int prenum=nums[0]-1;//initial an unexisting int
        for(int i=0;i<nums.size();++i){
           // if(i>0&&nums[i-1]==nums[i]) continue; 这样不可行
            if(!used[i]&&nums[i]!=prenum){
                prenum=nums[i];
                used[i]=true;
                cur.push_back(nums[i]);
                DFS(nums,res,used,cur);
                cur.pop_back();
                used[i]=false;
            }

        }
    }
};
```

48. Rotate Image//2D vector

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
       if(matrix.size()==0||matrix[0].size()==0) return;
        int top=0;
        int left=0;
        int right=matrix.size()-1;//n*n colums
        int bottom=matrix.size()-1;
        int n=matrix.size();
        while(n>1){
            for(int i=0;i<n-1;i++){
                int tmp=matrix[top][left+i];
                matrix[top][left+i]=matrix[bottom-i][left];
                matrix[bottom-i][left]=matrix[bottom][right-i];
                matrix[bottom][right-i]=matrix[top+i][right];
                matrix[top+i][right]=tmp;
            }
            top++;
            bottom--;
            left++;
            right--;
            n-=2;
        }

    }
};
```

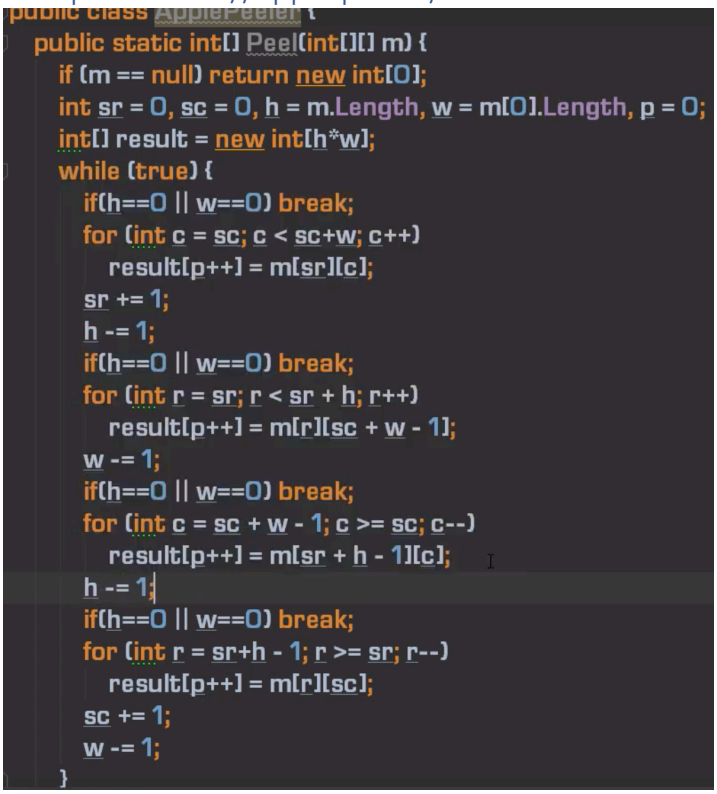
49. Group Anagrams//map

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        vector<vector<string>> res;
        unordered_map<string,vector<string>> map;//map

        for(string s:strs){
            string tmp=s;
            sort(tmp.begin(),tmp.end());//sort string
            map[tmp].push_back(s);// insert to a map value
        }
        for(auto e:map){
            res.push_back(e.second);//get map value
        }
        return res;
    }
};
```

50. Pow(x, n)//Recursion

```
class Solution {
public:
    double myPow(double x, int n) {
        if (n==0) return 1;
        if(n==1) return x;
        if(n==-1) return 1/x;
        return myPow(x*x,n/2)*(n%2==0?1:n>0?x:1/x);//双重判断*****
    }

};
```

54. Spiral Matrix//apple peeler/sr:start row



```
public class ApplePeeler {
  public static int[] Peel(int[][] m) {
    if (m == null) return new int[0];
    int sr = 0, sc = 0, h = m.Length, w = m[0].Length, p = 0;
    int[] result = new int[h*w];
    while (true) {
      if(h==0 || w==0) break;
      for (int c = sc; c < sc+w; c++)
        result[p++] = m[sr][c];
      sr += 1;
      h -= 1;
      if(h==0 || w==0) break;
      for (int r = sr; r < sr + h; r++)
        result[p++] = m[r][sc + w - 1];
      w -= 1;
      if(h==0 || w==0) break;
      for (int c = sc + w - 1; c >= sc; c--)
        result[p++] = m[sr + h - 1][c];
      h -= 1;
      if(h==0 || w==0) break;
      for (int r = sr+h - 1; r >= sr; r--)
        result[p++] = m[r][sc];
      sc += 1;
      w -= 1;
    }
```

62. Unique Paths//DP

Recursive solution here is much slow and waste memory, we will consider iterative solution.

Here's super simple recusive solution, easy to understand but poor performance. Will exceed time limit if you submit this solution

```
public int uniquePaths(int m, int n) {
        if (m==1 || n==1){
            return 1;
        }
        return uniquePaths(m-1,   n) +uniquePaths(m,   n-1);
    }
```

so we change a little bit, add memoization to improve its performance. now it beats 100% of java submissions! Please vote if you like it :)

```
public int uniquePaths(int m, int n) {
        int[][] memo = new int[m+1][n+1];
        return helper(m,n,memo);
    }
    private int helper(int m, int n,int[][]memo){
        if(memo[m][n]!=0){
            return memo[m][n];
        }
        if (m==1 || n==1){
            return 1;
        }
        memo[m][n] = helper(m-1,   n,memo) +helper(m,   n-1,memo);
        return memo[m][n];
    }
```

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

The above solution runs in `O(m * n)` time and costs `O(m * n)` space. However, you may have noticed that each time when we update `dp[i][j]`, we only need `dp[i - 1][j]` (at the previous row) and `dp[i][j - 1]` (at the current row). So we can reduce the memory usage to just two rows (`O(n)`).

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

Further inspecting the above code, `pre[j]` is just the `cur[j]` before the update. So we can further reduce the memory usage to one row.

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

`

63. Unique Paths II// padding for boundary case!!!

```
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        int m = obstacleGrid.size(), n = obstacleGrid[0].size();
        vector<vector<long> > dp(m + 1, vector<long> (n + 1, 0));//+1:the way to pad the obstacleGrid with 1 row and column and initial as below, because dp[1][1] is obstacleGrid[0][0] may be 1, so dp[1][1] has to get from the pad row or column
        //dp[0][1] = 1;or
        dp[1][0] = 1;
        for (int i = 1; i <= m; i++)
            for (int j = 1; j <= n; j++)
                if (!obstacleGrid[i - 1][j - 1])
                    dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
        return dp[m][n];
    }
};
```

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>> dp(m,vector<int>(n,0));
        dp[0][0]=grid[0][0];
        for(int i=1;i<m;i++){
            dp[i][0]=dp[i-1][0]+grid[i][0];
        }
        for(int j=1;j<n;j++){
            dp[0][j]=dp[0][j-1]+grid[0][j];
        }
        for(int i=1;i<m;i++){
            for(int j=1;j<n;j++){
                dp[i][j]=min(dp[i-1][j],dp[i][j-1])+grid[i][j];
            }
        }
        return dp[m-1][n-1];
    }
};
```

91. Decode Ways

Time complexity: O(n)

Space complexity: O(n)

```cpp
// Author: Huahua
// Runtime: 3 ms
class Solution {
public:
    int numDecodings(string s) {
        if(s.length() == 0) return 0;
        return ways(s, 0, s.length() - 1);
    }

private:
    int ways(const string& s, int l, int r) {
        if (m_ways.count(l)) return m_ways[l];
        if (s[l] == '0') return 0;
        if (l >= r) return 1; // Single digit or empty.

        int w = ways(s, l + 1, r);
        const int prefix = (s[l] - '0') * 10 + (s[l + 1] - '0');

        if (prefix <= 26)
             w += ways(s, l + 2, r);

        m_ways[l] = w;
        return w;
    }

    // Use l as key.
    unordered_map<int, int> m_ways;
};
```

Time complexity: O(n)

Space complexity: O(1)

```cpp
class Solution {
public:
    int numDecodings(string s) {
        if (s.empty() || s[0] == '0') return 0;
        if (s.length() == 1) return 1;

        const int n = s.length();
        int w1 = 1;
        int w2 = 1;
        for (int i = 1; i < n; ++i) {
            int w = 0;
            if (!isValid(s[i]) && !isValid(s[i - 1], s[i])) return 0;
            if (isValid(s[i])) w = w1;
            if (isValid(s[i - 1], s[i])) w += w2;
            w2 = w1;
            w1 = w;
        }
        return w1;
    }
private:
    bool isValid(const char c) { return c != '0'; }
    bool isValid(const char c1, const char c2) {
        const int num = (c1 - '0')*10 + (c2 - '0');
        return num >= 10 && num <= 26;
    }
};
```

71. Simplify Path

```
string simplifyPath(string path) {
    string res, tmp;
    vector<string> stk;
    stringstream ss(path);
    while(getline(ss,tmp,'/')) {
        if (tmp == "" or tmp == ".") continue;
        if (tmp == ".." and !stk.empty()) stk.pop_back();
        else if (tmp != "..") stk.push_back(tmp);
    }
    for(auto str : stk) res += "/"+str;
    return res.empty() ? "/" : res;
}
```

73. Set Matrix Zeroes

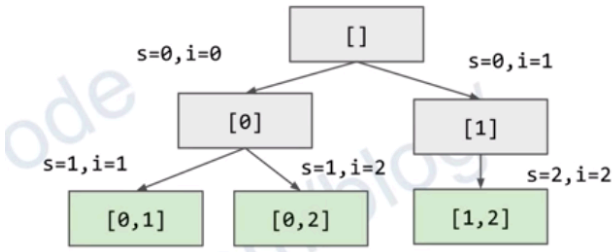
```
if (matrix == null || matrix.length == 0 || matrix[0].length == 0) return;
boolean firstRowZero = false;
boolean firstColZero = false;
for (int i = 0; i < matrix[0].length; i++) {
    if (matrix[0][i] == 0) {
        firstRowZero = true;
        break;
    }
}

for (int i = 0; i < matrix.length; i++) {
    if (matrix[i][0] == 0) {
        firstColZero = true;
        break;
    }
}
for (int x = 1; x < matrix[0].length; x++) {
    for (int y = 1; y < matrix.length; y++) {
        if (matrix[y][x] == 0) {
            matrix[y][0] = 0;
            matrix[0][x] = 0;
        }
    }
}

for (int i = 1; i < matrix[0].length; i++) {
    if (matrix[0][i] == 0) {
        for (int j = 1; j < matrix.length; j++) matrix[j][i] = 0;
    }
}

for (int i = 0; i < matrix.length; i++) {
    if (matrix[i][0] == 0) {
        for (int j = 1; j < matrix[0].length; j++) matrix[i][j] = 0;
    }
}
if (firstRowZero) for (int j = 0; j < matrix[0].length; j++) matrix[0][j] = 0;
if (firstColZero) for (int j = 0; j < matrix.length; j++) matrix[j][0] = 0;
```

78. Subsets//DFS

```cpp
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> res;
        sort(nums.begin(),nums.end());
        vector<int> cur;
        for(int i=0;i<=nums.size();i++){
            DFS(nums,0,cur,res,i);
        }

        return res;
    }
    void DFS(vector<int>& nums,int idx,vector<int> cur,vector<vector<int>>& res,int n){
        if(cur.size()==n){
            res.push_back(cur);
            return;
        }
        for(int i=idx;i<nums.size();i++) {
            cur.push_back(nums[i]);
            DFS(nums,i+1,cur,res,n);
            cur.pop_back();
        }


    }
};
```

Python, pay attention to cur.copy()

```python
class Solution:
  def subsets(self, nums):
    ans = []
    def dfs(n, s, cur):
      if n == len(cur):
        ans.append(cur.copy())
        return
      for i in range(s, len(nums)):
        cur.append(nums[i])
        dfs(n, i + 1, cur)
        cur.pop()
    for i in range(len(nums) + 1):
      dfs(i, 0, [])
    return ans
```



nums = [0, 1, 2], n = 2, C(3, 2)

```
Approach 2: Bit operation
State: n bits int / string
i-th bit 0: not selected / 1: selected

e.g. n = 3, nums = [0, 1, 2]
s = 000 => []
s = 101 => [0, 2]
s = 010 => [1]

s & (1 << i) => nums[i] selected

Time complexity: O(n*2^n), Space complexity: O(1)
```

```cpp
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        int n=nums.size();
        vector<vector<int>> res;

        for(int s=0;s<1<<n;s++){//1 shift left n times, is pow(2,n)
            vector<int> cur;
            for(int i=0;i<n;i++)
                if(s&1<<i) cur.push_back(nums[i]);//be choiced i
            res.push_back(cur);
        }
        return res;
    }
};
```

133. Clone Graph//BFS/DFS???

DFS

```
class Solution {
public:
    unordered_map<UndirectedGraphNode*, UndirectedGraphNode*> hash;
    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node) {
       if (!node) return node;
       if(hash.find(node) == hash.end()) {
           hash[node] = new UndirectedGraphNode(node -> label);
           for (auto x : node -> neighbors) {
                (hash[node] -> neighbors).push_back( cloneGraph(x) );
           }
       }
       return hash[node];
    }
};
```

BFS

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

409. Longest Palindrome

```
int longestPalindrome(string s) {
    unordered_map<char, int> dic;
    int res = 0;
    for( char c : s )
    {
        if( dic.find(c) != dic.end() )
        {
            if( ++dic[c] % 2 == 0 ) res += 2;
        }
        else
        {
            dic[c] = 1;
        }
    }

    if( s.length() > res ) ++res;

    return res;
}
```

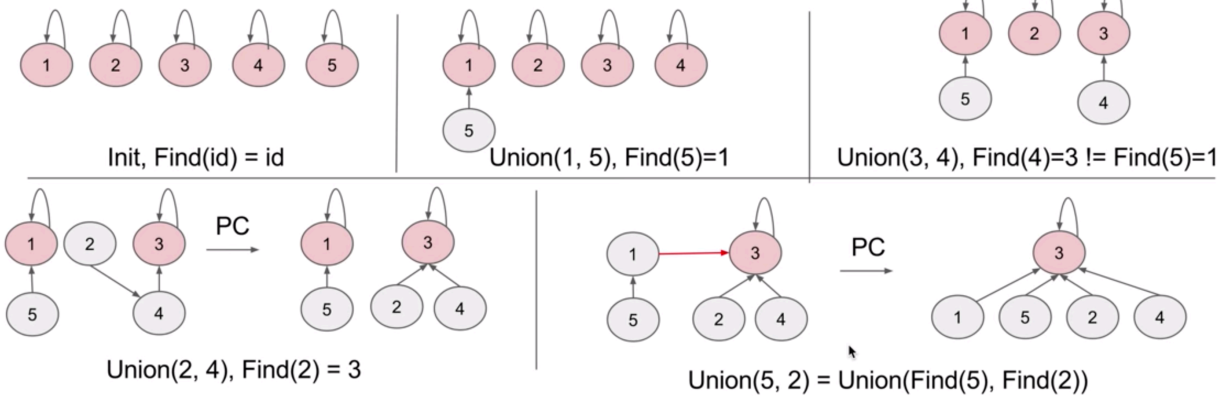
684. Redundant Connection

```
class Solution {
public:
    vector<int> findRedundantConnection(vector<vector<int>>& edges) {
        unordered_map<int, vector<int>> graph;
        for (const auto& edge : edges) {
            int u = edge[0];
            int v = edge[1];

            unordered_set<int> visited;
            if (hasPath(u, v, graph, visited))
                return edge;

            graph[u].push_back(v);
            graph[v].push_back(u);
        }
        return {};
    }
private:
    bool hasPath(int curr,
                 int goal,
                 const unordered_map<int, vector<int>>& graph,
                 unordered_set<int>& visited) {
        if (curr == goal) return true;
        visited.insert(curr);
        if (!graph.count(curr) || !graph.count(goal)) return false;
        for (int next : graph.at(curr)) {
            if (visited.count(next)) continue;
```

```
private:
    bool hasPath(int curr,
                 int goal,
                 const unordered_map<int, vector<int>>& graph,
                 unordered_set<int>& visited) {
        if (curr == goal) return true;
        visited.insert(curr);
        if (!graph.count(curr) || !graph.count(goal)) return false;
        for (int next : graph.at(curr)) {
            if (visited.count(next)) continue;
            if (hasPath(next, goal, graph, visited)) return true;
        }
        return false;
```

Union-Find

Find(x): find the root/cluster-id of x
Union(x, y): merge two clusters



Ref: https://www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf

PC = path compression

```cpp
class Solution {
public:
    vector<int> findRedundantConnection(vector<vector<int>>& edges) {

        vector<int> parents(edges.size() + 1, 0);
        vector<int> sizes(edges.size() + 1, 1);

        for(const auto& edge: edges) {
            int u = edge[0];
            int v = edge[1];
```

```
            int v = edge[1];
            if (!parents[u]) parents[u] = u;
            if (!parents[v]) parents[v] = v;
            int pu = find(u, parents);
            int pv = find(v, parents);

            // Both u and v are already in the tree
            if (pu == pv) return edge;

            // Unoin, always merge smaller set (pv) to larger set (pu)
            if (sizes[pv] > sizes[pu])
                swap(pu, pv);

            parents[pv] = pu;
            sizes[pu] += sizes[pv];
        }

        return {};
    }

private:
    int find(int node, vector<int>& parents) {
        while (parents[node] != node) {
            parents[node] = parents[parents[node]];
            node = parents[node];
        }
        return node;
```

14:16 / 16:34

1019. Next Greater Node In Linked List// use stack to store temporarily

If in decreasing trend, then they will get the same next bigger value,
If increasing, then get the next bigger value immediately
consider backward:

```
lass Solution {
ublic:
   vector<int> nextLargerNodes(ListNode* head) {
       vector<int> values;

       for (ListNode *current = head; current != nullptr; current = curre
           values.push_back(current->val);

       int n = values.size();
       vector<int> answers(n, 0);
       vector<int> stack;

       for (int i = n - 1; i >= 0; i--) {
           while (!stack.empty() && values[i] >= stack.back())
               stack.pop_back();

           if (!stack.empty())
               answers[i] = stack.back();

           stack.push_back(values[i]);
       }

       return answers;
   }
;
```

consider foreward:

```
class Solution {
public:
    vector<int> nextLargerNodes(ListNode* head) {
        vector<int> res,val,stack;
        int i=0;
        while(head){
            int num=head->val;
            val.push_back(num);
            // when came a bigger value(increase), pop idx stack utill it is not big
            while(!stack.empty()&&num>val[stack.back()]){
                res[stack.back()]=num;
                stack.pop_back();
            }
            stack.push_back(i);
            res.push_back(0);// initial with 0
            i++;
            head=head->next;
        }
        return res;
    }
};
```

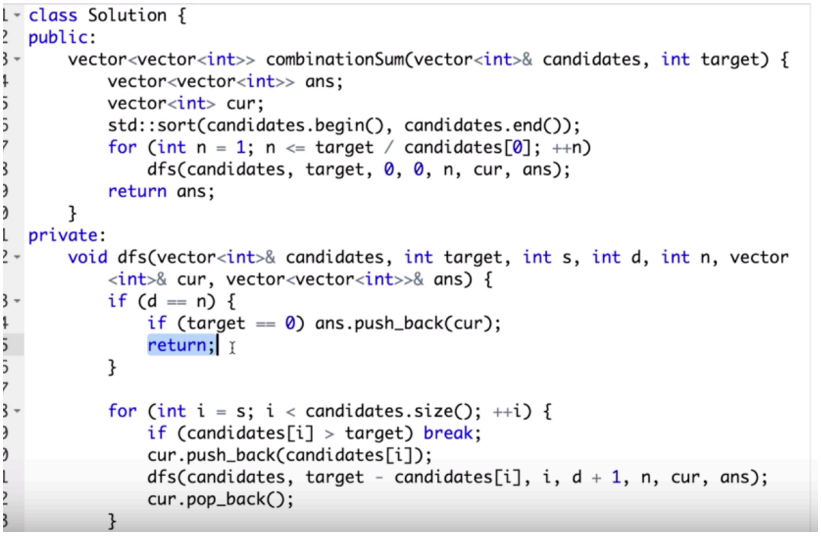
1022. Smallest Integer Divisible by K

```
class Solution {
public:
    int smallestRepunitDivByK(int K) {
        int N = 1;
        int m = 1 % K;
        while (m != 0 && N < 1000000)
        {
            m = (10 * m + 1) % K;
            N += 1;
        }
        if (N == 1000000)
            return -1;
        return N;
    }
};
```

```
class Solution {
public:
    int smallestRepunitDivByK(int K) {
        for(int r=0,N=1;N<K;++N){//initial two variables
            if(r=(r*10+1)%K==0) return N;//assign value with determin statement
        }  return -1;
    }
};
```

1021. Best Sightseeing Pair//DP

```
class Solution {
public:
    int maxScoreSightseeingPair(vector<int>& A) {
        int b=-100000;
        int a=0;

        for(int i=0;i<A.size();i++){
            a=max(a,A[i]-i+b);
            b=max(b,A[i]+i);
        }
        return a;
    }
};
```



```
class Solution {
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> ans;
        vector<int> cur;
        std::sort(candidates.begin(), candidates.end());
        for (int n = 1; n <= target / candidates[0]; ++n)
            dfs(candidates, target, 0, 0, n, cur, ans);
        return ans;
    }
private:
    void dfs(vector<int>& candidates, int target, int s, int d, int n, vector
        <int>& cur, vector<vector<int>>& ans) {
        if (d == n) {
            if (target == 0) ans.push_back(cur);
            return;
        }

        for (int i = s; i < candidates.size(); ++i) {
            if (candidates[i] > target) break;
            cur.push_back(candidates[i]);
            dfs(candidates, target - candidates[i], i, d + 1, n, cur, ans);
            cur.pop_back();
        }
```

199. Binary Tree Right Side View//recursion/loop binary tree/ recursion with the returning variable

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector <int> res;
        rightSideViewH(root,1,res);
        return res;
    }
    void rightSideViewH(TreeNode* root,int level,vector<int> &res){
        if (!root) return ;
        if(res.size()<level)//make sure every level only get one entered the vector
            res.push_back(root->val);
        rightSideViewH(root->right,level+1,res);
        rightSideViewH(root->left,level+1,res);

    }
```

200. Number of Islands

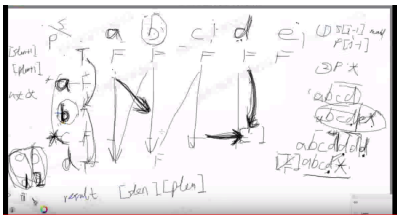
```
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {
        if (grid.empty()) return 0;
        int m=grid.size();
        int n=grid[0].size();
        int count=0;
        for(int y=0;y<m;y++){
            for(int x=0;x<n;x++){
                //when came across 1 add 1 to count
                count+=grid[y][x]-'0';
                //find the 1s that adjacent and change it to 0;
                dfs(grid,m,n,x,y);
            }
        }
        return count;


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

C. Hard

4. Median of Two Sorted Arrays

```
const int n1 = nums1.size();
const int n2 = nums2.size();
// Make sure n1 <= n2
if (n1 > n2) return findMedianSortedArrays(nums2, nums1);

const int k = (n1 + n2 + 1) / 2;

int l = 0;
int r = n1;

while (l < r) {
    const int m1 = l + (r - l) / 2;
    const int m2 = k - m1;
    if (nums1[m1] < nums2[m2 - 1])
        l = m1 + 1;
    else
        r = m1;
}

   const int m1 = l;
   const int m2 = k - l;

   const int c1 = max(m1 <= 0 ? INT_MIN : nums1[m1 - 1],
                      m2 <= 0 ? INT_MIN : nums2[m2 - 1])

   if ((n1 + n2) % 2 == 1)
       return c1;

   const int c2 = min(m1 >= n1 ? INT_MAX : nums1[m1],
                      m2 >= n2 ? INT_MAX : nums2[m2]);

   return (k + c2) * 0.5;
```

10. Regular Expression Matching



```
public class Solution {
    public boolean isMatch(String s, String p) {
        // assumption: * can't be the head of P. otherwise, index overflow.
        if (s == null || p == null) return false;
        boolean[][] match = new boolean[s.length()+1][p.length()+1];
        // no chars chosen from either of the string. should match
        match[0][0] = true;
        // i is index in matrix. to get char, we should use i-1
        for (int i = 1; i <= p.length(); i++) {
            if (p.charAt(i-1) == '*') {
                // * never be the first character
                match[0][i] = match[0][i-2];
            }
        }

        for (int si= 1; si <= s.length(); si++) {
            for (int pi = 1; pi <= p.length(); pi++) {
                if (p.charAt(pi-1) == '.' || p.charAt(pi-1) == s.charAt(si-1)) {
                    match[si][pi] = match[si-1][pi-1];
                } else if (p.charAt(pi-1) == '*') {
                    if (p.charAt(pi-2) == s.charAt(si-1) || p.charAt(pi-2) == '.') {
                        match[si][pi] = match[si][pi-2] || match[si-1][pi];
                    } else {
                        match[si][pi] = match[si][pi-2] ;
                    }
                }
            }
        }
        return match[s.length()][p.length()];
    }
}
```