LeetCode OJ (c++ version)

1. Single Number

```
int singleNumber(int A[], int n) {
    int res=0;
    for(int i=0;i<n;i++)
    {
        res=res^A[i];
    }
        return res;
    }
```

2. Maximum Depth of a Binary Tree

```
int maxDepth(TreeNode *root) {
        if(root==NULL) return 0;
        int lt=maxDepth(root->left);
        int rt=maxDepth(root->right);
        return 1+(lt>rt?lt:rt);
    }
```

3. Same Tree

```
bool isSameTree(TreeNode *p, TreeNode *q) {
        if(p==NULL&&q==NULL) return true;
        if(p==NULL||q==NULL) return false;
        if(p->val==q->val)
        return isSameTree(p->left,q->left)&&isSameTree(p->right,q->right);
        else return false;
    }
```

4. Reverse Integer

```
int reverse(int x) {
        int res=0;
        while(x!=0)
        {
            res=res*10+x%10;
            x=x/10;
        }
        return res;
```

5. Best Time to Buy and Sell Stock II

```
int maxProfit(vector<int> &prices) {

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

```
        return profit;
    }
```

6. Unique Binary Search Trees

```
int numTrees(int n) {
        // Note: The Solution object is instantiated only once and is reused by each test case.
        return numberTrees(0, n-1);
    }

    int numberTrees(int left, int right)
    {
        if(left>=right)
        return 1;
        int num=0;
        for(int i=left;i<=right;i++)
        num+=numberTrees(left,i-1)*numberTrees(i+1,right);
        return num;
    }
```

7. LinkedList Cycle

```
bool hasCycle(ListNode *head) {

        if(head==NULL) return false;
        ListNode* fast=head;
        ListNode* slow=head;
        while(true)
        {
            fast=fast->next;
            if(fast==NULL) break;
            fast=fast->next;
            if(fast==NULL) break;
            slow=slow->next;

            if(fast==slow)return true;
        }

        return false;
    }
```

8. Populating Next Right Pointers in Each Node

```
void connect(TreeLinkNode *root) {
        if(root==NULL) return;
        if(root->left!=NULL) root->left->next=root->right;
        if(root->right!=NULL)
root->right->next=(root->next==NULL?NULL:root->next->left);
        connect(root->left);
        connect(root->right);
```

```
    }
```

9. Binary Tree Preorder Traversal

```
vector<int> preorderTraversal(TreeNode *root)
    {
        if(root==NULL)
        return vector<int>();
        vector<int> res;
        preorderTraversal(root,res);
        return res;
    }

    void preorderTraversal(TreeNode* root, vector<int>& res)
    {
        if(root==NULL) return;
        stack<TreeNode*> st;
        st.push(root);
        while(!st.empty())
        {
            TreeNode*tmp;
            tmp=st.top();
            st.pop();
            res.push_back(tmp->val);
            if(tmp->right!=NULL) st.push(tmp->right);
            if(tmp->left!=NULL) st.push(tmp->left);
        }
    }
```

10. Search Insert Position

```
int searchInsert(int A[], int n, int target) {
        if(n==0) return 0;
        int start=0,end=n-1;
        int pos=findpos(A,start,end,target);
        return pos;
    }

    int findpos(int A[],int start, int end, int target)
    {
        int mid=0;
        while(start<=end)
        {
            mid=(start+end)/2;
            if(A[mid]==target) return mid;
            else if(A[mid]>target) end=mid-1;
            else start=mid+1;
```

```
            }

            if(A[mid]<target) mid=mid+1;
            return mid;
        }
```

11. Binary Tree Inorder Traversal

```
vector<int> inorderTraversal(TreeNode *root) {
            vector<int> res;
            if(root==NULL) return vector<int>();
            stack<TreeNode*> st;
            TreeNode*p=root;
            while(p!=NULL||!st.empty())
            {
                  if(p!=NULL)
                  {
                        st.push(p);
                        p=p->left;
                  }
                  else
                  {
                        p=st.top();
                        st.pop();
                        res.push_back(p->val);
                        p=p->right;
                  }
            }
            return res;
            }
```

12. Remove Dupicate From Sorted List

```
ListNode *deleteDuplicates(ListNode *head) {

            ListNode* pre=new ListNode(-1);
            if(head==NULL) return NULL;
            ListNode*runner,*cur,*res;
            res=pre;
            cur=head;
            runner=head;
            while(cur!=NULL)
            {
                  while(runner->val==cur->val)
                  {
                        runner=runner->next;
                        if(runner==NULL) break;
```

```
            }
            pre->next=cur;
            pre=cur;
            cur=runner;
        }
        pre->next=NULL;
        return res->next;
    }
```

13. Climbing Stairs

```
int climbStairs(int n) {
        int fib1=1,fib2=1;
        int res=1;
        for(int i=2;i<=n;i++)
        {
            res=fib1+fib2;
            fib1=fib2;
            fib2=res;
        }
        return res;
    }
```

14. Remove Element

```
int removeElement(int A[], int n, int elem) {
        int runner=0;
        for(int i=0;i<n;i++)
        if(A[i]!=elem) A[runner++]=A[i];
        return runner;
    }
```

15. Maximum Subarray

```
int maxSubArray(int A[], int n) {

        int sum=0,maxSb=INT_MIN;
        for(int i=0;i<n;i++)
        {
            if((sum+A[i])>A[i]) sum+=A[i];
            else sum=A[i];
            maxSb=std::max(maxSb,sum);
        }
    return maxSb;
    }
```

16. Roman to Integer

```
int cton(char c)
    {
        switch(c)
        {
```

```
            case 'I': return 1;
            case 'V': return 5;
            case 'X': return 10;
            case 'L': return 50;
            case 'C': return 100;
            case 'D': return 500;
            case 'M': return 1000;
            default: return 0;
        }
    }

    int romanToInt(string s) {


        int num=cton(s[0]);

        for(int i=0;i<s.size()-1;i++)
        {
            if(cton(s[i])<cton(s[i+1]))    num+=cton(s[i+1])-2*cton(s[i]);
            else num+=cton(s[i+1]);
        }
        return num;
    }
```

17. Merge Two Sorted Lists

```
ListNode *mergeTwoLists(ListNode *l1, ListNode *l2) {
        ListNode*pre=new ListNode(-1);
        ListNode*res=pre;
        while(l1!=NULL&&l2!=NULL)
        {
            if(l1->val>l2->val)
            {
                pre->next=l2;
                pre=pre->next;
                l2=l2->next;
            }
            else
            {
                pre->next=l1;
                pre=pre->next;
                l1=l1->next;
            }
        }
        if(l1!=NULL) pre->next=l1;
```

```
        if(l2!=NULL) pre->next=l2;

        return res->next;
    }
```

18. Integer to Roman

```
string intToRoman(int num) {
        string str;
        string symbol[]={"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};
        int value[]=    {1000,900,500,400, 100, 90,  50, 40,  10, 9,   5,  4,   1};
        for(int i=0;num!=0;++i)
        {
            while(num>=value[i])
            {
                num-=value[i];
                str+=symbol[i];
            }
        }
        return str;

    }
```

19. Remove Duplicate from Sorted Array

```
int removeDuplicates(int A[], int n) {

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

20. Symmetric Tree

```
bool isSymmetric(TreeNode*left,TreeNode*right)
    {
        if(left==NULL&&right==NULL) return true;
        else if(left==NULL||right==NULL)    return false;
        return
(left->val==right->val)&&isSymmetric(left->left,right->right)&&isSymmetric(left->right,right->l
eft);
    }

bool isSymmetric(TreeNode *root) {
```

```
        if(root==NULL) return true;
        queue<TreeNode*> q1, q2;
        if(root->left==NULL&&root->right==NULL) return true;
        if(root->left==NULL||root->right==NULL) return false;
        q1.push(root->left);
        q2.push(root->right);
        while(!q1.empty()&&!q2.empty())
        {
            TreeNode* t1,*t2;
            t1=q1.front();
            t2=q2.front();
            q1.pop();
            q2.pop();
            if((t1==NULL&&t2!=NULL)||(t1!=NULL&&t2==NULL)) return false;
            if(t1!=NULL)
              {
                  if(t1->val!=t2->val) return false;
                  q1.push(t1->left);
                  q1.push(t1->right);
                  q2.push(t2->right);
                  q2.push(t2->left);
            }
        }
         return true;
    }
```

21. Convert Sorted Array to Binary Search Tree

```
TreeNode *sortedArrayToBST(vector<int> &num) {
        if (num.size()==0) return NULL;
        return MakeTree(num, 0, num.size()-1);

    }

    TreeNode* MakeTree(vector<int> num, int start, int end)
    {
        if(start>end) return NULL;
        int mid=(start+end)/2;
        if(start==end)return new TreeNode(num[mid]);
        TreeNode* root=new TreeNode(num[mid]);
        root->left=MakeTree(num,start,mid-1);
        root->right=MakeTree(num,mid+1,end);
        return root;
    }
```

22. Pascal’s Triangle

```
vector<vector<int> > generate(int numRows) {
        vector<vector<int>> res;
        for(int i=1;i<=numRows;i++)
        {
            vector<int> ans(i,1);
            for(int k=1;k<i;k++)
            {
                ans[k]=res[i-2][k-1]+(k==i-1?0:res[i-2][k]);
            }
            res.push_back(ans);
        }
        return res;
    }
```

23. Single Number II

```
int singleNumber(int A[], int n) {

        int *bit=new int[32];
        int j=0;
        for(;j<32;j++)
        bit[j]=0;
        for(int i=0;i<n;i++)
        {
            for(j=0;j<32;j++)
            bit[j]+=(A[i] & (1 << j))==0?0:1;
        }


    int res=0;

    for(j=0;j<32;j++)
    {
        if(bit[j]%3!=0)
        res |= 1 << j;
    }

    return res;
    }
```

24. Swap Node Pairs

```
ListNode *swapPairs(ListNode *head) {
        if(head==NULL||head->next==NULL) return head;
        ListNode* first=head,*second=head->next;
        ListNode*pre;
```

```
        pre->next=head;
        while(first!=NULL&&second!=NULL)
        {
            ListNode *tmp=second->next;
            first->next=tmp;
            second->next=first;
            if(pre->next==head) head=second;
            pre->next=second;
            pre=first;
            first=tmp;
            if(first==NULL)break;
            second=tmp->next;

        }

        return head;

    }
```

25. Balanced Binary Tree

```
bool isBalanced(TreeNode *root) {
    if(root==NULL) return true;
    return
(abs(depth(root->left)-depth(root->right))<=1)&&isBalanced(root->left)&&isBalanced(root
->right);
    }

    int depth(TreeNode* root)
    {
        if(root==NULL) return 0;
        int lt=depth(root->left);
        int rt=depth(root->right);
        return 1+(lt>rt?lt:rt);
    }
```

26. Best Time to Buy and Sell Stock

```
int maxProfit(vector<int> &prices) {
        if(prices.size()==0) return 0;
        int localmin=INT_MAX;
        int profits=INT_MIN;
        for(int i=0;i<prices.size();i++)
        {
            if(prices[i]<localmin)
            localmin=prices[i];
            if(prices[i]-localmin>profits)
```

```
            profits=prices[i]-localmin;
        }

        return profits;
    }
```

27. Gray Code

```
vector<int> grayCode(int n) {

        int nsize=(1<<n);
        vector<int> res;
        for(int i=0;i<nsize;i++)
        {
            res.push_back((i>>1)^i);
        }
        return res;
    }
```

28. Binary Tree Levelorder Traversal II

```
vector<vector<int> > levelOrderBottom(TreeNode *root) {
        vector<vector<int>> res;
        vector<int>ans;
        if(root==NULL) return vector<vector<int>>();
        //stack<vector<int>> st;
        queue<TreeNode*> q1,q2;
        q1.push(root);
        while(!q1.empty())
        {
            ans.clear();
            while(!q1.empty())
            {
                TreeNode*tmp;
                tmp=q1.front();
                q1.pop();
                ans.push_back(tmp->val);
                if(tmp->left!=NULL)q2.push(tmp->left);
                if(tmp->right!=NULL) q2.push(tmp->right);
            }
            swap(q1,q2);
            res.push_back(ans);
        }
        reverse(res.begin(),res.end());
        return res;
    }
```

29. Merge Sorted Array

```
void merge(int A[], int m, int B[], int n) {
        int i=0,j=0,k=0;
        int *tmp=new int[m+n];
        while(i<m&&j<n)
        {
            if(A[i]<B[j])
            tmp[k++]=A[i++];
            else tmp[k++]=B[j++];
        }

        for(;i<m;i++)tmp[k++]=A[i];
        for(;j<n;j++)tmp[k++]=B[j];
        for(i=0;i<m+n;i++)
        A[i]=tmp[i];
        delete[] tmp;
    }
```

30. Permutations

```
vector<vector<int> > permute(vector<int> &num) {

        vector<vector<int>> res;
        vector<int> rcd;
        permute(res,rcd,num);
        return res;
    }

    void permute(vector<vector<int>> &res, vector<int> &rcd, vector<int> num)
    {
        if(num.size()<=0)
        {
            res.push_back(rcd);
            return;
        }
        for(int i=0;i<num.size();i++)
        {
            vector<int> tmp=num;
            rcd.push_back(tmp[i]);
            for(int j=i;j<num.size();j++)
                tmp[j]=tmp[j+1];
            tmp.pop_back();
            permute(res,rcd,tmp);
            rcd.pop_back();
        }
    }
```

31. Sort Colors

```
void sortColors(int A[], int n) {
        int count[3];
        memset(count,0,3*sizeof(int));
        for(int i=0;i<n;i++)
        {
            count[A[i]]++;
        }
        int k;
        for(k=0;k<count[0];k++)
        A[k]=0;
        for(k=count[0];k<count[0]+count[1];k++)
        A[k]=1;
        for(k=count[0]+count[1];k<count[0]+count[1]+count[2];k++)
        A[k]=2;
        return ;
    }
```

32. Rotate Image

```
void rotate(vector<vector<int> > &matrix) {

        if (matrix.size()==0||matrix[0].size()==0) return;
        int n=matrix.size();
        for(int i=0;i<matrix.size()/2;i++)
        for(int j=i;j<matrix[0].size()-1-i;j++)
        {
            int tmp=matrix[i][j];
            matrix[i][j]=matrix[n-1-j][i];
            matrix[n-1-j][i]=matrix[n-1-i][n-1-j];
            matrix[n-1-i][n-1-j]=matrix[j][n-1-i];
            matrix[j][n-1-i]=tmp;
        }

    }
```

33. Generate Parentheses

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

      vector<string> res;
      string ans;
      genParen(res,ans,0,n,0,0);
      return res;

    }
```

```
void genParen(vector<string>& res,string& ans,int depth,int num, int leftnum,int rightnum)
{
    if(depth==2*num)
    {
        res.push_back(ans);
        return;
    }
    if(leftnum<num)
    {
        ans.push_back('(');
        genParen(res,ans,depth+1,num,leftnum+1,rightnum);
        ans.pop_back();
    }
    if(rightnum<leftnum)
    {
        ans.push_back(')');
        genParen(res,ans,depth+1,num,leftnum,rightnum+1);
        ans.pop_back();
    }
    if(rightnum>leftnum)
    return;
}
```

34. Unique Path

```
int uniquePaths(int m, int n) {
        vector<vector<int>>count(m,vector<int>(n,0));
        for(int i=0;i<m;i++)
        count[i][0]=1;
        for(int j=0;j<n;j++)
        count[0][j]=1;

        for(int i=1;i<m;i++)
        for(int j=1;j<n;j++)
        count[i][j]=count[i-1][j]+count[i][j-1];
        return count[m-1][n-1];
    }
```

35. Minimum Sum Path

```
int minPathSum(vector<vector<int> > &grid) {
    if(grid.size()==0||grid[0].size()==0) return 0;
    int m=grid.size(),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 j=1;j<n;j++)
        sum[0][j]=sum[0][j-1]+grid[0][j];

        for(int i=1;i<m;i++)
        for(int j=1;j<n;j++)
        sum[i][j]=grid[i][j]+std::min(sum[i-1][j],sum[i][j-1]);

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

36. Linked List Cycle II

```
ListNode *detectCycle(ListNode *head) {
            if(head==NULL||head->next==NULL||head->next->next==NULL) return NULL;
            ListNode* r1(0),*r2(0);
            r1=head->next;
            r2=head->next->next;
            while(r1!=r2)
            {
                  if(r2->next!=NULL)
                  r2=r2->next;
                  if(r2->next!=NULL)
                  r2=r2->next;
                  if(r2==NULL) return NULL;
                  r1=r1->next;
            }


            if(r2->next==NULL) return NULL;
            r1=head;

            while(r1!=r2)
            {
                  r1=r1->next;
                  r2=r2->next;
            }

            return r1;
      }
```

37. Container With Most Water

```
int maxArea(vector<int> &height) {
            int area=0,maxArea=INT_MIN;
            int i=0,j=height.size()-1;
            while(i<j)
            {
                  area=min(height[i],height[j])*(j-i);
```

```
            maxArea=max(maxArea,area);
            if(height[i]>height[j]) j--;
            else i++;
        }
        return maxArea;
    }
```

38. Set Matrix Zeros

```
void setZeroes(vector<vector<int> > &matrix) {
        int m=matrix.size(),n=matrix[0].size();
        if(m==0||n==0) return;
        int indexRow=0,indexCol=0;
        for(int i=0;i<m;i++)
        if(matrix[i][0]==0)
        {indexRow=1;break;}
        for(int i=0;i<n;i++)
        if(matrix[0][i]==0)
        {indexCol=1;break;}

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

        for(int i=1;i<m;i++)
        for(int j=1;j<n;j++)
        if(matrix[i][0]==0||matrix[0][j]==0)
        matrix[i][j]=0;

        if(indexRow)
        for(int i=0;i<m;i++) matrix[i][0]=0;
         if(indexCol)
        for(int i=0;i<n;i++) matrix[0][i]=0;
        return;
    }
```

39. BinaryTree PostOrder Traversal;

```
vector<int> postorderTraversal(TreeNode *root) {

        if(root==NULL) return vector<int>();
        vector<int> res;
```

```
        stack<TreeNode*> st;
        TreeNode*p=root;
        st.push(p);
        while(!st.empty())
        {
            p=st.top();
            if(p->left==NULL&&p->right==NULL)
            {
                st.pop();
                res.push_back(p->val);
            }
            if(p->right!=NULL)
            {
                st.push(p->right);
                p->right=NULL;
            }
            if(p->left!=NULL)
            {
                st.push(p->left);
                p->left=NULL;
            }
        }
        return res;
    }
```

40. Binary Tree LevelOrder Traversal

```
vector<vector<int> > levelOrder(TreeNode *root) {

        if(root==NULL) return vector<vector<int>>();
        queue<TreeNode*> q1, q2;
        q1.push(root);
        vector<int> tmp;
        vector<vector<int>> res;
        while(!q1.empty())
        {
            tmp.clear();
            while(!q1.empty())
            {
                TreeNode *cur;
                cur=q1.front();
                q1.pop();
                tmp.push_back(cur->val);
                if(cur->left!=NULL) q2.push(cur->left);
                if(cur->right!=NULL) q2.push(cur->right);
            }
```

```
            swap(q1,q2);
            res.push_back(tmp);
        }

        return res;

    }
```

41. Search a 2D Matrix

```
bool searchMatrix(vector<vector<int> >& matrix, int target) {
        int rows = matrix.size();
        int cols = matrix[0].size();
        int low = 0;
        int high = rows * cols - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (matrix[mid/cols][mid%cols] == target)
                return true;
            else if (matrix[mid/cols][mid%cols] < target)
                low = mid + 1;
            else if (matrix[mid/cols][mid%cols] > target)
                high = mid - 1;
        }
        return false;
    }
```

42. plus one

```
vector<int> plusOne(vector<int> &digits) {
        vector<int> res;
        stack<int>st;
        int i=digits.size()-1;
        int carry=0;
        while(i>=0)
        {
            int newdigit;
            if(i==digits.size()-1)
            {
                newdigit=(digits[i]+1)%10;
                carry=(digits[i]+1)/10;
                st.push(newdigit);
            }
            else
            {
                newdigit=(digits[i]+carry)%10;
                carry=(digits[i]+carry)/10;
```

```
                st.push(newdigit);
            }
            i--;
        }
        if(carry==1)st.push(1);
        while(!st.empty())
        {
            int tmp=st.top();
            st.pop();
            res.push_back(tmp);
        }
        return res;
    }
```

43. Path Sum

```
bool hasPathSum(TreeNode *root, int sum) {
    if(root==NULL) return false;
    return findSum(root,sum);
    }

    bool findSum(TreeNode* root, int sum)
    {
        if(root==NULL) return false;
        sum=sum-root->val;
        if(sum==0&&root->left==NULL&&root->right==NULL) return true;
        else return findSum(root->left,sum)||findSum(root->right,sum);
    }
```

44. N-Queen II

```
int sum=0;
    int totalNQueens(int n) {
        if(n==0) return 0;
        int col[n],row[n];
        memset(col,0,n*sizeof(int));
        memset(row,0,n*sizeof(int));
        findSum(0,n,col,row);
        return sum;
    }

    void findSum(int index, int n, int col[], int row[])
    {
        if(index==n)
        {
            sum++;
            return;
        }
```

```
            for(int j=0;j<n;j++)
            {
                if(col[j]==1) continue;
                int i;
                for(i=0;i<index;i++)
                {
                    if(abs(i-index)==abs(row[i]-j))
                    break;
                }
                if(i==index)
                {
                    col[j]=1;
                    row[index]=j;
                    findSum(index+1,n,col,row);
                    col[j]=0;
                    row[index]=0;
                }
            }
        }
```

45. Remove nth node from the end

```
ListNode *removeNthFromEnd(ListNode *head, int n) {
            if(head==NULL) return NULL;
            ListNode* second,*first,*pre;
            pre->next=head;
            second=first=head;
            for(int i=0;i<n-1&&first!=NULL;i++)
            {
                second=second->next;
            }
            if(second==NULL) return head;
            while(second->next!=NULL)
            {
                first=first->next;
                second=second->next;
                pre=pre->next;
            }
            if(pre->next==head) return head->next;
            pre->next=first->next;
            return head;

        }
```

46. Combinations

```
vector<vector<int> > combine(int n, int k) {
        if(k==0||k>n) return vector<vector<int>>();
```

```
 vector<int> ans;
 vector<vector<int>> res;
 int index=1;
 genComb(res,ans,index,n,k);
 return res;
}

void genComb(vector<vector<int>>&res, vector<int>& ans, int index, int n, int k)
{
    if(ans.size()==k)
    {
        res.push_back(ans);
        return;
    }

    for(int i=index;i<=n;i++)
    {
        ans.push_back(i);
        genComb(res,ans,i+1,n,k);
        ans.pop_back();
    }
}
```

47. Pascal's Triangle II

```
vector<int> getRow(int rowIndex) {
    // if(rowIndex==0) return vector<int>();
     vector<int>ans(rowIndex+1,0);
     ans[0]=1;
     for(int i=2;i<=rowIndex+1;i++)
     {
         vector<int>tmp;
         tmp=ans;
         for(int j=1;j<i;j++)
         {
             ans[j]=tmp[j-1]+(j==i-1?0:tmp[j]);
         }
     }
     return ans;
}
```

48. Search in Rotated Sorted Array II

```
bool search(int A[], int n, int target) {
     int start=0,end=n-1;
     while(start<=end)
     {
         int mid=(start+end)/2;
```

```
            if(A[mid]==target) return true;
            if(A[mid]>A[start])
            {
                if(A[mid]>target&&target>=A[start])
                end=mid-1;
                else start=mid+1;

            }
            else if(A[mid]<A[start])
            {
                if(A[mid]<target&&target<=A[end])
                start=mid+1;
                else end=mid-1;
            }
            else start++;
        }
        return false;
    }
```

49. Populating Next Pointers In Each Node II

```
void connect(TreeLinkNode *root) {
        if(root==NULL) return;
        TreeLinkNode*p=root->next;
        while(p!=NULL)
        {
            if(p->left!=NULL)
            {p=p->left;break;}
            if(p->right!=NULL)
            {p=p->right;break;}
            p=p->next;
        }

        if(root->left!=NULL)root->left->next=root->right==NULL?p:root->right;
        if(root->right!=NULL) root->right->next=p;
        connect(root->right);
        connect(root->left);
    }
```

50. Palindrome Number

```
bool isPalindrome(int x) {

        int digit=0,tmp=0;
        if(x<0) return false;
        tmp=x;
        while(tmp>0)
```

```
        {
            digit++;
            tmp/=10;
        }

        return isPalin(x,digit,1);
    }

    bool isPalin(int x, int last, int first)
    {
        if(last<=first) return true;
        int t1=x,t2=x;
        for(int i=1;i<first;i++)
        t1=t1/10;
        t1=t1%10;
        for(int i=1;i<last;i++)
        t2=t2/10;
        t2=t2%10;
        if(t1==t2) return isPalin(x,last-1,first+1);
        else return false;
    }
```

51. Sum roof to leaf num

```
int sumNumbers(TreeNode *root) {
     if(root==NULL) return 0;
     int sum=0;
     return sumNum(root,sum);
    }

    int sumNum(TreeNode* root, int sum)
    {
        if(root==NULL) return 0;
        sum=sum*10+root->val;
        if(root->left==NULL&&root->right==NULL) return sum;
        return sumNum(root->left,sum)+sumNum(root->right,sum);
    }
```

52. minDepth of a Binary Tree

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

```
    }
```
53. Search element in rotated array
```
int search(int A[], int n, int target) {
        int l=0,r=n-1;
        while(l<=r)
        {
            int mid=(l+r)/2;
            if(A[mid]==target) return mid;
            if(A[mid]>=A[l])
            {
                if(target<A[mid]&&target>=A[l])
                r=mid-1;
                else l=mid+1;
            }
            else
            {
                if(target>A[mid]&&target<=A[r])
                l=mid+1;
                else r=mid-1;
            }
        }

        return -1;
    }
```
54. Trapping rain water
```
int trap(int A[], int n) {
    if(n<2) return 0;
    vector<int> leftmax(n,0),rightmax(n,0);
    leftmax[0]=A[0];
    int maxN=A[0];
    for(int i=1;i<=n-2;i++)
    {
        leftmax[i]=maxN;
        if(A[i]>maxN)
        maxN=A[i];
    }

    maxN=A[n-1];
    rightmax[n-1]=A[n];
    for(int i=n-2;i>=1;i--)
    {
        rightmax[i]=maxN;
        if(A[i]>maxN)
        maxN=A[i];
```

```
    }

    int cap=0;
    for(int i=1;i<=n-2;i++)
    {
        int tmp=std::min(leftmax[i],rightmax[i])-A[i];
        if(tmp>0) cap+=tmp;
    }

    return cap;
    }
```

55. Length of Last Word

```
int lengthOfLastWord(const char *s) {

        int tmp=0,count=0;
        int i=0;
        while(s[i]!='\0')
        {
            while(s[i]!=' '&&s[i]!='\0')
            {
                tmp++;
                i++;
            }

            count=tmp;
            if(s[i]=='\0') break;
            while(s[i]==' '&&s[i]!='\0') i++;
            tmp=0;
            if(s[i]=='\0') break;
        }

        return count;
    }
```

56. Valid Parathenses

```
bool isValid(string s) {
        stack<char> st;
        for(int i=0;i<s.size();i++)
        {
            if(s[i]=='('||s[i]=='['||s[i]=='{')
            {
                st.push(s[i]);
                continue;
            }
            if(st.empty()) return false;
```

```
            char current=st.top();
            if(s[i]==')'&&current!='(') return false;
            if(s[i]==']'&&current!='[') return false;
            if(s[i]=='}'&&current!='{') return false;
            st.pop();
        }

        if(!st.empty()) return false;
    }
```

57. Path Sum II

```
vector<vector<int> > pathSum(TreeNode *root, int sum) {
        vector<int> ans;
        vector<vector<int>> res;
        ans.clear();
        res.clear();
        if(root==NULL) return vector<vector<int>>();
        pathSum(ans,res,root,sum);
        return res;
    }

    void pathSum(vector<int> &ans, vector<vector<int>>& res, TreeNode*root, int sum)
    {
        if(sum-root->val==0&&root->left==NULL&&root->right==NULL)
        {
            ans.push_back(root->val);
            res.push_back(ans);
            return;
        }
        ans.push_back(root->val);
        if(root->left!=NULL)
        {
            pathSum(ans,res,root->left,sum-root->val);
            ans.pop_back();
        }
        if(root->right!=NULL)
        {
            pathSum(ans,res,root->right,sum-root->val);
            ans.pop_back();
        }
    }
```

58. Valid Sudoku

```
bool isValidSudoku(vector<vector<char> > &board) {

        if(board.size()<=0||board[0].size()<=0) return false;
```

```
        for(int i=0;i<board.size();i++)
        for(int j=0;j<board[0].size();j++)
        {
            if(board[i][j]=='.') continue;
            else
            {
                for(int k=0;k<board[0].size();k++)
                {
                    if(j==k) continue;
                    if(board[i][k]==board[i][j])
                    return false;
                }
                for(int k=0;k<board.size();k++)
                {
                    if(i==k) continue;
                    if(board[k][j]==board[i][j])
                    return false;
                }
                for(int p=i/3*3;p<i/3*3+3;p++)
                for(int q=j/3*3;q<j/3*3+3;q++)
                {
                    if(i==p&&j==q)continue;
                    if(board[p][q]==board[i][j])
                    return false;
                }
            }
        }

        return true;
    }
```

59. Subsets

```
vector<vector<int> > subsets(vector<int> &S) {
        vector<int>ans;
        vector<vector<int>> res;
        sort(S.begin(),S.end());
        ans.clear();
        res.clear();
        res.push_back(ans);
        subsets(ans,res,S,S.size(),0);
        return res;
    }
```

```
    void subsets(vector<int>& ans,vector<vector<int>>& res, vector<int>S, int n,int level)
    {
        if(level==n)return;

        for(int i=level;i<n;i++)
        {
            ans.push_back(S[i]);
            res.push_back(ans);
            subsets(ans,res,S,n,i+1);
            ans.pop_back();
        }
    }
```

60. Jump Game

```
bool canJump(int A[], int n) {
        if(n<0) return false;
        vector<int>jump(n,0);
        jump[0]=A[0];
        if(jump[0]>=n-1) return true;
        if(jump[0])
        for(int i=1;i<=n-2;i++)
        {
            if(jump[i-1]==0) return false;
            jump[i]=std::max(jump[i-1]-1,A[i]);
            if(jump[i]>=n-i-1) return true;
        }
        return false;
    }
```

61 Subsets II

```
vector<vector<int> > subsetsWithDup(vector<int> &S) {
        vector<int> ans;
        vector<vector<int>> res;
        ans.clear();
        res.clear();
        sort(S.begin(),S.end());
        res.push_back(ans);
        subsets(ans,res,S,S.size(),0);
        return res;
    }

    void subsets(vector<int>& ans, vector<vector<int>>& res, vector<int> S, int n, int level)
    {
        if(level==n) return;
        for(int i=level;i<n;i++)
```

```
        {
            ans.push_back(S[i]);
            res.push_back(ans);
            subsets(ans,res,S,n,i+1);
            ans.pop_back();
            while(S[i+1]==S[i]) i++;
        }
    }
```

62. Longest Common Prefix

```
string longestCommonPrefix(vector<string> &strs) {
        char curStr;
        if(strs.size()==0) return string();
        if(strs.size()==1) return strs[0];
        string res;
        for(int i=0;i<strs[0].size();i++)
        {
            curStr=strs[0][i];
            int j;
            for(j=1;j<strs.size()&&i<strs[j].size();j++)
            {
                if(strs[j][i]!=curStr) break;
            }
            if(j==strs.size()) res.append(1,curStr);
            else break;
        }

        return res;
    }
```

63. Unique Path II

```
int uniquePathsWithObstacles(vector<vector<int> > &obstacleGrid) {
        if(obstacleGrid.size()==0||obstacleGrid[0].size()==0) return 0;
        vector<vector<int>> uniP(obstacleGrid.size(),vector<int>(obstacleGrid[0].size(),0));
        int row=obstacleGrid.size(),col=obstacleGrid[0].size();
        for(int i=0;i<row;i++)
        {
            if(obstacleGrid[i][0]!=1) uniP[i][0]=1;
            else break;
        }
        for(int j=0;j<col;j++)
        {
            if(obstacleGrid[0][j]!=1) uniP[0][j]=1;
            else break;
        }
```

```
        for(int i=1;i<row;i++)
        for(int j=1;j<col;j++)
        {
            if(obstacleGrid[i][j]==1)
            {uniP[i][j]=0;continue;}
            else
            {
                uniP[i][j]=uniP[i][j-1]+uniP[i-1][j];
            }
        }

        return uniP[row-1][col-1];

    }
```

64. Longest Consecutive Sequence

```
int longestConsecutive(vector<int> &num) {
        map<int,int>hm;
        for(int i=0;i<num.size();i++)
        hm[num[i]]=i;
        vector<int> visited(num.size(),0);
        int maxLen=-1;
        int len=0;
        for(int i=0;i<num.size();i++)
        {
            if(visited[i]==1)continue;
            visited[i]=1;
            len=1;
            int index=num[i]-1;
            while(hm.find(index)!=hm.end())
            {
                if(visited[hm[index]]==1) break;
                visited[hm[index]]=1;
                len++;
                index--;
            }

            index=num[i]+1;
            while(hm.find(index)!=hm.end())
            {
                if(visited[hm[index]]==1) break;
                visited[hm[index]]=1;
                len++;
```

```
                index++;
            }
            if(maxLen<len)
            maxLen=len;
            len=0;
        }

        return maxLen;
    }
```

65. 3Sum Closest

```
int threeSumClosest(vector<int> &num, int target) {
        int curLen=0, record=INT_MAX,minV=INT_MAX;
        sort(num.begin(),num.end());
        for(int i=0;i<=num.size()-3;i++)
        {
            curLen=num[i];
            int start=i+1,end=num.size()-1;
            while(start<end)
            {
                if(std::abs(curLen+num[start]+num[end]-target)<minV)

{ minV=abs(curLen+num[start]+num[end]-target);record=curLen+num[start]+num[end];}
                if((curLen+num[start]+num[end])==target) return target;
                else if((curLen+num[start]+num[end])<target) start++;
                else end--;
            }
        }

    return record;
    }
```

66. Search for a range

```
vector<int> searchRange(int A[], int n, int target) {
        int begin=0,end=n-1,mid=-1;
        vector<int> res(2,-1);
        while(begin<=end)
        {
            mid=(begin+end)/2;
            if(A[mid]==target) break;
            if(A[mid]>target)
            end=mid-1;
            if(A[mid]<target) begin=mid+1;
        }
        if(begin>end) return res;
```

```
        res[0]=res[1]=mid;
        int index=mid-1;
        while(index>=0&&A[index]==target)
            res[0]=index--;

        index=mid+1;
        while(index<n&&A[index]==target)
            res[1]=index++;

        return res;

    }
```

67. Count and Say

```
string countAndSay(int n) {
        string prev,cur;
        prev="1";
        cur="";
        for(int i=2;i<=n;i++)
        {
            char c=prev[0];
            int count=0;
            int p=0;
            while(p<prev.size())
            {
                while(prev[p]==c&&p<prev.size())
                {
                    count++;
                    p++;
                }
                cur+=count+'0';
                cur+=c;
                count=0;
                c=prev[p];
            }
            prev=cur;
            cur="";
        }
        return prev;
    }
```

68. Flatten Binary Tree to linedList

```
void flatten(TreeNode *root) {
        if(root==NULL) return;
        stack<TreeNode*> st;
        TreeNode* prev=new TreeNode(-1);
```

```
        st.push(root);
        while(!st.empty())
        {
            TreeNode* tmp;
            tmp=st.top();
            st.pop();
            if(tmp->right!=NULL) st.push(tmp->right);
            if(tmp->left!=NULL) st.push(tmp->left);
            prev->right=tmp;
            prev->left=NULL;
            prev=prev->right;
        }
    }
```

69. Combination Sum

```
vector<vector<int> > combinationSum(vector<int> &candidates, int target) {
    if(candidates.size()==0) return vector<vector<int>>();
    sort(candidates.begin(),candidates.end());
    vector<int> ans;
    vector<vector<int>> res;
    combSum(ans,res,candidates,0,target,0);
    return res;
    }

    void combSum(vector<int>& ans,vector<vector<int>>& res, vector<int> cad, int level,int target,int sum)
    {
        if(sum==target)
        {
        res.push_back(ans);
        return;
        }
        if(sum>target)
        return ;

        for(int i=level;i<cad.size();i++)
        {
            ans.push_back(cad[i]);
            combSum(ans,res,cad,i,target,sum+cad[i]);
            ans.pop_back();
        }
    }
```

70. Triangle

```
int minimumTotal(vector<vector<int> > &triangle) {
```

```
        //if(triangle.size()<=0||triangle[0].size()<=0) return 0;
        vector<int>minRow(triangle.size(),0);
        int m=triangle.size();
        minRow[0]=triangle[0][0];
        for(int i=1;i<m;i++)
        for(int j=i;j>=0;j--)
        {
            if(j==0)
            minRow[j]+=triangle[i][0];
            else if(j==triangle[i].size()-1)
            minRow[j]=minRow[j-1]+triangle[i][j];
            else minRow[j]=triangle[i][j]+std::min(minRow[j],minRow[j-1]);
        }

        int minV=INT_MAX;
        for(int i=0;i<triangle[m-1].size();i++)
        {
            if(minRow[i]<minV) minV=minRow[i];
        }

        return minV;
    }
```

71. Partition List

```
ListNode *partition(ListNode *head, int x) {
        if(head==NULL) return NULL;
        ListNode* p=new ListNode(x-1);
        p->next=head;
        ListNode* cur,*prev,*res;
        res=p;
        while(p!=NULL&&p->val<x)
        {
            prev=p;
            p=p->next;
        }
        if(p==NULL) return head;
        if(p!=NULL)
        {
            cur=prev;
            while(p!=NULL)
            {
                if(p->val<x)
                {
                    ListNode*tmp=cur->next;
```

```
                    prev->next=p->next;
                    cur->next=p;
                    cur=p;
                    p->next=tmp;
                    p=prev;
                }
                prev=p;
                p=p->next;
            }
        }

        return res->next;
    }
```

72. Binary ZigZag level Order Traversal

```
vector<vector<int> > zigzagLevelOrder(TreeNode *root) {
    if(root==NULL) return vector<vector<int>>();
    queue<TreeNode*> q1,q2;
    vector<int> ans;
    vector<vector<int>> res;
    ans.clear();
    res.clear();
    q1.push(root);
    bool flag=1;
    while(!q1.empty())
    {
        ans.clear();
        while(!q1.empty())
        {
            TreeNode* tmp=q1.front();
            q1.pop();
            ans.push_back(tmp->val);
            if(tmp->left!=NULL) q2.push(tmp->left);
            if(tmp->right!=NULL) q2.push(tmp->right);
        }
        swap(q1,q2);
        if(flag)
        {
            res.push_back(ans);
            flag=0;
        }
        else
        {
            reverse(ans.begin(),ans.end());
```

```
                res.push_back(ans);
                flag=1;
            }
        }
        return res;
        }
```

73. Unique Binary Search Tree II

```
vector<TreeNode *> generateTrees(int n) {
            return gen(1,n);

        }

        vector<TreeNode*> gen(int start, int end)
        {
            vector<TreeNode*> subTree;
            if(start>end)
            {
                subTree.push_back(NULL);
                return subTree;
            }
            for(int i=start;i<=end;i++)
            {
                vector<TreeNode*> lefts;
                lefts=gen(start,i-1);
                vector<TreeNode*> rights;
                rights=gen(i+1,end);
                for(int j=0;j<lefts.size();j++)
                for(int k=0;k<rights.size();k++)
                {
                    TreeNode* root=new TreeNode(i);
                    root->left=lefts[j];
                    root->right=rights[k];
                    subTree.push_back(root);
                }
            }
            return subTree;
        }
```

74. Pow(x,n)

```
double pow(double x, int n) {

            if(n==0)return 1;
            if(n==1)return x;
```

```
        double temp=pow(x,abs(n/2));
        if(n>0)
        {
            if(n&1)return temp*temp*x;
            else return temp*temp;
        }
        else
        {
            if(n&1)return 1.0/(temp*temp*x);
            else return 1.0/(temp*temp);
        }
    }
```

75. Letter Combination

```
vector<string> letterCombinations(string digits) {
    string trans[]={"","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};
    vector<string> res;
    string ans;
    genComb(ans,res,0,trans,digits);
    return res;
    }

    void genComb(string& ans,vector<string>& res, int level, string trans[],string digits)
    {
        if(level==digits.size())
        {
            res.push_back(ans);
            return;
        }

        for(int i=0;i<trans[digits[level]-'0'].size();i++)
        {
            ans.push_back(trans[digits[level]-'0'][i]);
            genComb(ans,res,level+1,trans,digits);
            ans.pop_back();
        }

    }
```

76. Reverse Linkedlist II

```
ListNode *reverseBetween(ListNode *head, int m, int n) {
    if(head==NULL) return NULL;
    if(n-m<=0) return head;
    ListNode*pre=new ListNode(-1);
    pre->next=head;
```

```
        for(int i=1;i<m&&pre;i++)
        {
            pre=pre->next;
        }
        if(pre->next==NULL||pre==NULL) return head;
        ListNode* tail=pre->next;
        ListNode* runner=tail->next;
        ListNode* lat=pre->next;
        for(int i=1;i<=n-m&&runner;i++)
        {
            ListNode* tmp=runner->next;
            runner->next=lat;
            pre->next=runner;
            tail->next=tmp;
            lat=runner;
            runner=tmp;
        }
        if(m>1) return head;
        return pre->next;
        }
```

77. Valid Binary Search Tree

```
bool isValidBST(TreeNode *root) {
         if(root==NULL) return true;
         int minV=INT_MIN,maxV=INT_MAX;
         return VerifyBST(root,minV,maxV);
        }

        bool VerifyBST(TreeNode* root, int minV,int maxV)
        {
            if(root==NULL) return true;
            if(root->val>minV&&root->val<maxV)
            return
VerifyBST(root->left,minV,root->val)&&VerifyBST(root->right,root->val,maxV);
            else return false;
        }
```

78. Preorder+Inorder

```
TreeNode *buildTree(vector<int> &inorder, vector<int> &postorder) {

            TreeNode* root=NULL;
            root=buildTree(inorder,0,inorder.size()-1, postorder,0,postorder.size()-1);
            return root;

        }
```

```
    TreeNode* buildTree(vector<int> &inorder,int inbg,int inend, vector<int> &postorder,int postbg,int postend)
    {
        if(inbg>inend||postbg>postend)
        return NULL;
        int mid=-1;
        for(int i=inbg;i<=inend;i++)
        if(inorder[i]==postorder[postend]) mid=i;
        TreeNode *root=new TreeNode(inorder[mid]);
        root->left=buildTree(inorder, inbg,mid-1,postorder,postbg,postbg+mid-inbg-1);
        root->right=buildTree(inorder,mid+1,inend, postorder,postbg+mid-inbg,postend-1);
        return root;
    }
```

79. Postorder+Inorder

```
TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder) {

        TreeNode* root;
        root=buildTree(preorder,0,preorder.size()-1,inorder,0,inorder.size()-1);
        return root;
    }

    TreeNode* buildTree(vector<int>&preorder, int s0,int e0, vector<int>&inorder, int s1,int e1)
    {
        if(s0>e0||s1>e1)
        return NULL;

        int mid;
        for(int i=s1;i<=e1;i++)
        if(inorder[i]==preorder[s0])
        {
            mid=i;
            break;
        }

        TreeNode* root=new TreeNode(inorder[mid]);
        root->left=buildTree(preorder,s0+1,s0+mid-s1,inorder,s1,mid-1);
        root->right=buildTree(preorder,s0+mid-s1+1,e0,inorder,mid+1,e1);
        return root;
    }
```

80. N-Queens

```
vector<vector<string> > solveNQueens(int n) {
    if(n<=0) return vector<vector<string>>();
```

```
vector<vector<string>> res;
vector<int> row(n,0),col(n,0);
dfs(0,n,res,row,col);
return res;
}

void dfs(int r,int n,vector<vector<string>>& res,vector<int> row, vector<int> col)
{
	if(r==n)
	{
		vector<string> ans;
		for(int i=0;i<n;i++)
		{
			string tmp(n,'.');
			tmp[row[i]]='Q';
			ans.push_back(tmp);
		}
		res.push_back(ans);
		return;
	}

	for(int j=0;j<n;j++)
	{
		if(col[j]==1) continue;
		int rr;
		for(rr=0;rr<r;rr++)
		{
			if(abs(row[rr]-j)==abs(r-rr))
			break;
		}

		if(rr==r)
		{
		row[r]=j;
		col[j]=1;
		dfs(r+1,n,res,row,col);
		row[r]=0;
		col[j]=0;
		}
	}

}
```

81. Add Binary

```
string addBinary(string a, string b) {
        reverse(a.begin(),a.end());
        reverse(b.begin(),b.end());
        string res;
        res.clear();
        int len=std::max(a.size(),b.size());
        int carry=0,digit=0;
        for(int i=0;i<len;i++)
        {
                digit=((i<a.size()?a[i]-'0':0)+(i<b.size()?b[i]-'0':0)+carry)%2;
                carry=((i<a.size()?a[i]-'0':0)+(i<b.size()?b[i]-'0':0)+carry)/2;
                res.insert(res.begin(),digit+'0');
        }
        if(carry>0) res.insert(res.begin(),carry+'0');
        return res;
        }
```

82. Palindrome Partition

```
vector<vector<string>> partition(string s) {
                vector<string> ans;
                vector<vector<string>> res;
                palipar(s,0,ans,res);
                return res;
        }
        void palipar(string s, int level, vector<string>& ans,vector<vector<string>>& res)
        {
                if(level==s.size())
                {
                        res.push_back(ans);
                        return;
                }


                for(int j=level;j<s.size();j++)
                {
                                if(isPalin(s,level,j))
                                {
                                        ans.push_back(s.substr(level,j-level+1));
                                        palipar(s,j+1,ans,res);
                                        ans.pop_back();
                                }
                }
        }
```

```
bool isPalin(string s,int i, int j)
{
    while(i<=j)
    {
        if(s[i]!=s[j]) return false;
        else{i++;j--;}
    }

    return true;
}
```

83. Insertion sort list

```
ListNode *insertionSortList(ListNode *head) {
    if(head==NULL||head->next==NULL) return head;
    ListNode*cur=head;
    ListNode*runner=head;
    ListNode* insert=head->next;
    ListNode* former,*pre;
    pre->next=head;
    former=pre;
    while(insert!=NULL)
    {
        while(runner!=insert&&runner->val<insert->val)
        {
            former=former->next;
            runner=runner->next;
        }

        if(runner==insert)
        {
            cur->next=insert;
            cur=insert;
            insert=insert->next;
        }
        else
        {
            ListNode* tmp=insert->next;
            insert->next=runner;
            former->next=insert;
            cur->next=tmp;
            insert=tmp;
        }
        runner=pre->next;
        former=pre;
```

```
    }

    return pre->next;
    }
```

84. Next Permutation

```
void nextPermutation(vector<int> &num) {
    if(num.size()==0) return;
    for(int i=num.size()-2;i>=0;i--)
    {
        if(num[i+1]>num[i])
        {
            for(int j=num.size()-1;j>=i+1;j--)
            {
                if(num[j]>num[i])
                {
                    swap(num[i],num[j]);
                    break;
                }
            }
            reverse(num.begin()+i+1,num.end());
            return;
        }
    }
    reverse(num.begin(),num.end());
    return;
    }
```

85. Remove Duplicates for Sorted List II

```
ListNode *deleteDuplicates(ListNode *head) {
    if(head==NULL||head->next==NULL) return head;
    ListNode* prev=new ListNode(-1);
    ListNode* res=new ListNode(-1);
    prev->next=head;
    res=prev;
    ListNode* runner=new ListNode(-1);
    ListNode* cur=new ListNode(-1);
    cur=runner=head;
    while(runner!=NULL)
    {
        while(runner!=NULL&&runner->val==cur->val)
        runner=runner->next;
        if(cur->next==runner)
        {
            prev->next=cur;
            cur=runner;
```

```
            prev=prev->next;
        }
        else
        {
            prev->next=runner;
            cur=runner;
            //prev=prev->next;
        }
    }

    return res->next;
    }
```

86. Permutations II

```
vector<vector<int> > permuteUnique(vector<int> &num) {
    if(num.size()==0) return vector<vector<int>>();
    vector<vector<int>>res;
    vector<int>ans;
    vector<int> visited(num.size(),0);
    sort(num.begin(),num.end());
    getPermu(0,num.size(),res,ans,num,visited);
    return res;
    }

    void getPermu(int level,int n,vector<vector<int>>& res,vector<int>&
ans,vector<int>num,vector<int>visited)
    {
        if(level==n)
        {
            res.push_back(ans);
            return;
        }

        for(int i=0;i<num.size();i++)
        {
            if(visited[i]==1) continue;
            else
            {
                visited[i]=1;
                ans.push_back(num[i]);
                getPermu(level+1,n,res,ans,num,visited);
                ans.pop_back();
                visited[i]=0;
            }
```

```
            while(num[i+1]==num[i]&&i<=num.size()-2)
            i++;
        }
    }
```

87. Edit Distance

```
int minDistance(string word1, string word2) {
        int **d=new int*[word1.size()+1];
        int row=word1.size()+1;
        int col=word2.size()+1;
        for(int i=0;i<row;i++)
        {
            d[i]=new int[col];
            d[i][0]=i;
        }
        for(int j=0;j<col;j++)
        d[0][j]=j;

        for(int i=1;i<row;i++)
        {
            char ci=word1[i-1];
        for(int j=1;j<col;j++)
        {
            char cj=word2[j-1];
            if(ci==cj)
            d[i][j]=d[i-1][j-1];
            else
            {
                int replace=d[i-1][j-1]+1;
                int add=d[i][j-1]+1;
                int del=d[i-1][j]+1;
                int minL=min(replace,add);
                d[i][j]=min(minL,del);
            }
        }
        }
        return d[row-1][col-1];
```

88. Reverse Nodes in k-Group

```
ListNode *reverseKGroup(ListNode *head, int k) {

        if(head==NULL) return NULL;
        ListNode *kptr=new ListNode(-1);
        kptr->next=head;
```

```
        ListNode*pre,*runner,*start,*cur,*res;
        pre=new ListNode(-1);
        pre->next=head;
        cur=head;
        runner=head;
        start=head;
        res=pre;
        while(kptr!=NULL)
        {
            for(int i=0;i<k;i++)
            {
                kptr=kptr->next;
                if(kptr==NULL) break;
            }
            if(kptr==NULL) break;
                for(int j=0;j<k;j++)
                {
                    ListNode* tmp;
                    tmp=(runner->next==NULL?NULL:runner->next);
                    runner->next=start;
                    start=runner;
                    runner=tmp;
                }
                pre->next=kptr;
                kptr=cur;
                cur->next=runner;

                pre=cur;
                start=runner;
                cur=runner;

        }

        return res->next;
    }
```

89. Gas Stations

```
int canCompleteCircuit(vector<int> &gas, vector<int> &cost) {

    vector<int> diff(gas.size(),0);
    int left=0,sum=0,st=0;
    for(int i=0;i<gas.size();i++)
    {
        diff[i]=gas[i]-cost[i];
        left+=diff[i];
```

```
            sum+=diff[i];
            if(sum<0)
            {
                st=i+1;
                sum=0;
            }
        }
            if (left<0) return -1 ;
           return st;
        }
```

90. Distinct Subsequence

```
int numDistinct(string S, string T) {

            if(S.size()==0) return 0;
            if(T.size()==0) return 1;
            int **dp;
            dp=new int*[T.size()+1];
            for(int i=0;i<=T.size();i++)
            {
                dp[i]=new int[S.size()+1];
                dp[i][0]=0;
            }

            for(int j=1;j<=S.size();j++)
            dp[0][j]=1;

            dp[0][0]=1;

            for(int i=1;i<=T.size();i++)
            for(int j=1;j<=S.size();j++)
            {
                dp[i][j]=dp[i][j-1];
                if(T[i-1]==S[j-1])
                dp[i][j]+=dp[i-1][j-1];
            }

            return dp[T.size()][S.size()];
        }
```

91. Combinations Sum II

```
vector<vector<int> > combinationSum2(vector<int> &num, int target) {
        if(target<=0) return vector<vector<int>>();
        if(num.size()==0)return vector<vector<int>>();
        sort(num.begin(),num.end());
```

```
    vector<int> ans;
    vector<vector<int>>res;
    genComb(num,ans,res,target,0,0);
    return res;
    }

    void genComb(vector<int> &num,vector<int>& ans, vector<vector<int>>& res, int target, int cur,int level)
    {
        if(target==cur)
        {
            res.push_back(ans);
            return;
        }

        if(cur>target)
        return;
        if(level==num.size()) return;

        for(int i=level;i<num.size();i++)
        {
            ans.push_back(num[i]);
            genComb(num,ans,res,target,cur+num[i],i+1);
            ans.pop_back();
            while(num[i+1]==num[i]&&i<num.size()-1)
            i++;
        }
    }
```

92. Jump Game II

```
int jump(int A[], int n) {
    int step=1;
    if(n==1) return 0;
    int start=0,end=0;
    int maxV=INT_MIN;

    while(end<n)
    {
        maxV=INT_MIN;
    for(int i=start;i<=end;i++)
    {
        if(A[i]+i>=n-1) return step;
        else
        {
            if(A[i]+i>maxV)
```

```
                maxV=A[i]+i;
            }

        }
            start=end+1;
            end=maxV;
            step++;
        }

        return step;
        }
```

93. Merge K sorted Lists

```
ListNode *mergeKLists(vector<ListNode *> &lists) {
            if(lists.size()==0) return NULL;
            ListNode*p=lists[0];
            for(int i=1;i<lists.size();i++)
            {
                p=merge(p,lists[i]);
            }
            return p;
            }

            ListNode* merge(ListNode* l1, ListNode*l2)
            {
                ListNode*tmp=new ListNode(-1);
                ListNode* res=tmp;
                while(l1!=NULL&&l2!=NULL)
                {
                    if(l1->val<l2->val)
                    {tmp->next=l1;l1=l1->next;}
                    else
                    {tmp->next=l2;l2=l2->next;}
                    tmp=tmp->next;
                }
                if(l1!=NULL) tmp->next=l1;
                if(l2!=NULL) tmp->next=l2;
                tmp=res;
                delete tmp;
                res=res->next;
                return res;
            }
```

94. Longest Substring without repeat character

95. Zigzag Conversion

```
string convert(string s, int nRows) {
    if(nRows <= 1) return s;
        string ret;
        int zigsize = 2 * nRows - 2;
        for(int i = 0; i < nRows; ++i) {
            for(int base = i; ;base += zigsize) {
                if(base >= s.size())
                    break;
                ret.append(1,s[base]);

                if(i > 0 && i < nRows - 1) {
                    int ti = base + zigsize - 2 * i;
                    if(ti < s.size())
                        ret.append(1,s[ti]);
                }
            }
        }
        return ret;
    }
```

96. Anagram

```
int lengthOfLongestSubstring(string s) {
    vector<int>map(256,-1);
    if(s.size()==0) return 0;
    int len=0;
    int maxL=INT_MIN;
    for(int i=0;i<s.size();i++)
    {
        if(map[s[i]]==-1)
        {
            map[s[i]]=i;
            len++;
            if(len>maxL) maxL=len;
        }
        else
        {
            //len=i-map[s[i]];
            for(int j=i-len;j<map[s[i]];j++)
            map[s[j]]=-1;

            len=i-map[s[i]];
            map[s[i]]=i;
```

```
            }
        }

        if(len>maxL) maxL=len;
        return maxL;
        }
```

97. Recover Binary Tree

```
ListNode *addTwoNumbers(ListNode *l1, ListNode *l2) {
            ListNode* tmp=new ListNode(-1);
            ListNode*res=tmp;
            int carry =0;
            int digit=0;
            while(l1!=NULL||l2!=NULL)
            {
                digit=(l1!=NULL?l1->val:0)+(l2!=NULL?l2->val:0)+carry;
                ListNode*p=new ListNode(digit%10);
                tmp->next=p;
                tmp=tmp->next;
                carry=digit/10;
                if(l1!=NULL) l1=l1->next;
                if(l2!=NULL) l2=l2->next;
            }
            if(carry>0)
            tmp->next=new ListNode(1);
            return res->next;
        }
```

98. 4Sum

```
vector<vector<int> > fourSum(vector<int> &num, int target) {
        if(num.size()<4) return vector<vector<int>>();
        vector<int> ans;
        vector<vector<int>> res;
        sort(num.begin(),num.end());
        for(int i=0;i<num.size();i++)
        {
        for(int j=i+1;j<num.size();j++)
        {
            int start=j+1,end=num.size()-1;
            while(start<end)
            {
                int val=num[start]+num[end]+num[i]+num[j];
                if(val==target)
                {
                    ans.push_back(num[i]);
                    ans.push_back(num[j]);
```

```
                    ans.push_back(num[start]);
                    ans.push_back(num[end]);
                    res.push_back(ans);
                    ans.clear();
                    start++;
                    end--;
                    while(start<end&&num[start]==num[start-1]) start++;
                    while(start<end&&num[end]==num[end+1]) end--;
                    continue;
                }
                else if(val>target) {end--;while(end>start&&num[end]==num[end+1]) end--;}
                else {start++;while(start<end&&num[start]==num[start-1]) start++;}

            }
        while(num[j+1]==num[j]&&j<=num.size()-2) j++;
        }
        while(num[i+1]==num[i]&&i<=num.size()-2) i++;
        }
        return res;
        }
```

99. First Missing Positive

```
int firstMissingPositive(int A[], int n) {
        if(n==0) return 1;
        for(int i=0;i<n;i++)
        {
            while(A[i]!=i+1)
            {
                if(A[i]>n||A[i]<1||A[A[i]-1]==A[i]) break;

                    int tmp=A[i];
                    A[i]=A[tmp-1];
                    A[tmp-1]=tmp;
            }
        }

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

100. Best Time to Buy and Sell Stock III

```
int maxProfit(vector<int> &prices) {
```

```
        if(prices.size()==0) return 0;
        int *left=new int[prices.size()];
        int *right=new int[prices.size()];

        int mini=prices[0];
        left[0]=0;
        for(int i=1;i<prices.size();i++)
        {
            left[i]=max(left[i-1],prices[i]-mini);
            mini=min(mini,prices[i]);
        }

        int maxs=prices[prices.size()-1];
        right[prices.size()-1]=0;
        for(int i=prices.size()-2;i>=0;i--)
        {
            right[i]=max(right[i+1],maxs-prices[i]);
            maxs=max(maxs,prices[i]);
        }

        int res=0,maxi=INT_MIN;
        for(int i=0;i<prices.size();i++)
        {
            res=left[i]+right[i];
            if(res>maxi) maxi=res;
        }

        return maxi;

    }
```

101. Copy List with Random Point

```
map<RandomListNode*,RandomListNode*> hm;
        hm.clear();
        if(head==NULL) return NULL;
       RandomListNode* res=new RandomListNode(0);
       RandomListNode* p=head;
       RandomListNode* q=res;
       while(p)
       {
            RandomListNode*tmp=new RandomListNode(p->label);
            q->next=tmp;
            hm[p]=tmp;
```

```
            p=p->next;
            q=q->next;
        }

        p=head;
        q=res->next;
        while(p)
        {
            if(p->random==NULL) q->random=NULL;
            else
            {
                q->random=hm[p->random];
            }
                q=q->next;
                p=p->next;
        }

        return res->next;
    }
```

102. Rotate List

```
if(head==NULL) return NULL;
        ListNode* runner=head;
        ListNode* pre=new ListNode(-1);
        pre->next=head;
        int len=0;
        while(runner!=NULL)
        {
            len++;
            runner=runner->next;
            pre=pre->next;
        }
        pre->next=head;
        ListNode*ptr=head;
        pre->next=ptr;
        for(int i=0;i<len-k%len;i++)
        {
            ptr=ptr->next;
            pre=pre->next;
        }
        pre->next=NULL;
        return ptr;
```

103. sqrt(x)

```
int sqrt(int x) {
        int start=0,end=x/2<std::sqrt(INT_MAX)?x/2+1:std::sqrt(INT_MAX);
        int mid=0;
        while(start<=end)
        {
            mid=(start+end)/2;
            if(mid*mid==x) return mid;
            else if(mid*mid>x)
            end=mid-1;
            else start=mid+1;
        }

        return start/2+end/2;
    }
```

104. Valid Palindrome

```
bool isPalindrome(string s) {
        if(s.size()==0) return true;
        std::transform(s.begin(), s.end(), s.begin(), ::tolower);
        return isPalin(s);
    }

    bool isPalin(string s)
    {
        int i=0,j=s.size()-1;
        while(i<=j)
        {
            while (i<s.size()&&!isAn(s[i])) i++;
            while (j>=0&&!isAn(s[j])) j--;
            if(s[i]!=s[j]) break;
            i++;
            j--;
        }

        if(i>j) return true;
        return false;
    }

    bool isAn(char s)
    {
        if(s>='0'&& s<='9') return true;
        if(s>='a'&& s<='z') return true;
        return false;
```

```
    }
```

105. Scramble String

```
bool isScramble(string s1, string s2) {

        int A[26];
        for(int i=0;i<26;i++) A[i]=0;
        for(int i=0;i<s1.size();i++)
        A[s1[i]-'a']++;
        for(int i=0;i<s2.size();i++)
        A[s2[i]-'a']--;
        for(int i=0;i<26;i++)
        {
            if(A[i]!=0) return false;
        }
        if(s1.size()==1&&s2.size()==1) return true;
        for(int i =1; i< s1.size(); i++)
        {
                bool result= isScramble(s1.substr(0, i), s2.substr(0, i))
                    && isScramble(s1.substr(i, s1.size()-i), s2.substr(i, s1.size()-i));
                result = result || (isScramble(s1.substr(0, i), s2.substr(s2.size() - i, i))
                    && isScramble(s1.substr(i, s1.size()-i), s2.substr(0, s1.size()-i)));
                if(result) return true;
            }
        return false;

    }
```

106. Permutation Sequence

```
string getPermutation(int n, int k) {
        int total=1;
        int num[10];
        int visited[10];
        string res;
        memset(visited,0,10*sizeof(int));
        for(int i=0;i<n;i++)
        {
            num[i]=i+1;
            total*=(i+1);
        }

        k--;
        for(int i=0;i<n;i++)
        {
            total/=(n-i);
```

```
            int choose=k/total;
            int j=0;
            while(choose>=0)
            {
                if(visited[j]==0)
                {
                    choose--;
                    j++;
                }
                else j++;
            }
            res+=num[j-1]+'0';
            visited[j-1]=1;
            k=k%total;
        }
        return res;
    }
```

107. Maximum Rectangle