**1.NEXT PERMUTATION:**class Solution {

public:

    void nextPermutation(vector<int>& nums) {

        int n=nums.size();

        int i=n-2;

        while(i>=0 && nums[i]>=nums[i+1]){

            i--;

        }

        if(i>=0){

            int j=n-1;

            while(j>=0 && nums[j]<=nums[i]){

                j--;

            }

        swap(nums[i],nums[j]);

        }

        return reverse(nums.begin()+i+1,nums.end());

    }

};

**2.SPIRAL MATRIX**class Solution {

public:

    vector<int> spiralOrder(vector<vector<int>>& matrix) {

        int n=matrix.size();

        int m=matrix[0].size();

        int left=0;

        int right=m-1;

        int top=0;

        int bottom=n-1;

        vector<int>res;

        while(top<=bottom && left<=right){

            for(int i=left;i<=right;i++){

                res.push\_back(matrix[top][i]);

            }

            top++;

            for(int i=top;i<=bottom;i++){

                res.push\_back(matrix[i][right]);

            }

            right--;

            if(top<=bottom){

            for(int i=right;i>=left;i--){

                res.push\_back(matrix[bottom][i]);

            }

            bottom--;

            }

            if(left<=right){

            for(int i=bottom;i>=top;i--){

                res.push\_back(matrix[i][left]);

            }

            left++;

            }

        }

        return res;

    }

};

**3.LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS**

class Solution {

public:

    int lengthOfLongestSubstring(string s) {

        unordered\_set<char>charset;

        int n=s.length();

        int maxlen=0,i=0,j=0;

        while(i<n && j<n){

            if(charset.find(s[j])==charset.end()){

                charset.insert(s[j++]);

                maxlen=max(maxlen,j-i);

            }

            else{

                charset.erase(s[i++]);

            }

        }

        return maxlen;

    }

};

**4.** **REMOVE LINKED LIST ELEMENTS**

class Solution {

public:

    ListNode\* removeElements(ListNode\* head, int value) {

        if(head==NULL){

            return NULL;

        }

        if((head->next==NULL) && (head->val==value)){

            return NULL;

        }

        if((head->val!=value) && (head->next==NULL)){

            return head;

        }

        while((head!=NULL) && (head->val==value))

        {

            ListNode\* temp1=head;

            head=head->next;

            temp1->next=NULL;

            delete temp1;

         }

        ListNode\* temp1=head;

        while((temp1!=NULL)&&(temp1->next!=NULL))

        {

            ListNode\* front = temp1->next;

            if(temp1->next->val==value)

            {

                temp1->next=front->next;

                front->next=NULL;

                delete front;

            }

            else{

                temp1=temp1->next;

                front=front->next;

            }

        }

        return head;

    }

};

**5. PALINDROME LINKED LIST**class Solution {

public:

    bool isPalindrome(ListNode\* head) {

        if(head==NULL || head->next==NULL){

            return(head);

        }

        ListNode \*r\_head=NULL;

        ListNode \*ptr=head;

        while(ptr!=NULL){

            ListNode \*temp = new ListNode(ptr->val);

            temp->next=r\_head;

            r\_head=temp;

            ptr=ptr->next;

        }

        while(head && r\_head){

            if(head->val != r\_head->val){

                return false;

            }

            head=head->next;

            r\_head=r\_head->next;

        }

        return true;

    }

};