Dsa 6

1. next permutation
2. class Solution {
3. public:
4. void nextPermutation(vector<int>& nums) {
5. next\_permutation(nums.begin(),nums.end());
7. }

};

2) spiral matrix

class Solution {

public:

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

int n=matrix.size();

int m=matrix[0].size();

vector<int>ans;

int left=0;

int right=m-1;

int top=0;

int bottom=n-1;

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

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

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

}

top++;

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

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

}

right--;

if(top<=bottom){

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

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

}

bottom--;

}

if(left<=right){

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

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

}

left++;

}

}

return ans;

}

};

3) longest substring without repeating characters

class Solution {

public:

int lengthOfLongestSubstring(string s) {

int maxlen=0;

int left=0;

int n=s.size();

unordered\_set<char>charset;

for(int right=0;right<n;right++){

while(charset.find(s[right])!= charset.end()){

charset.erase(s[left]);

left++;

}

charset.insert(s[right]);

maxlen=max(maxlen,right-left+1);

}

return maxlen;

}

};

4) removed linked list elements

class Solution {

public:

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

ListNode\* ans = new ListNode(0, head);

ListNode\* dummy = ans;

while (dummy != nullptr) {

while (dummy->next != nullptr && dummy->next->val == val) {

dummy->next = dummy->next->next;

}

dummy = dummy->next;

}

ListNode\* result = ans->next;

delete ans;

return result;

}

};

5) palindrome linked list

class Solution {

public:

ListNode\* reversell(ListNode \* head){

ListNode \* prev=NULL;

ListNode \* temp=head;

while(temp!=NULL){

ListNode \* front=temp->next;

temp->next=prev;

prev=temp;

temp=front;

}

return prev;

}

bool isPalindrome(ListNode\* head) {

ListNode \* slow=head;

ListNode \* fast=head;

while(fast->next != NULL && fast->next->next!=NULL){

slow=slow->next;

fast=fast->next->next;

}

ListNode \* newhead=reversell(slow->next);

ListNode \* first=head;

ListNode \* second=newhead;

while(second!=NULL){

if(first->val != second->val){

reversell(newhead);

return false;

}

first=first->next;

second=second->next;

}

reversell(newhead);

return true;

}

};