***Week – 5 (26.04.2021 – 02.05.2021)***

***CODES BASED ON WEELKY TASK***

1. ***Search a 2D Matrix II:***

class Solution {

public:

bool searchMatrix(vector<vector<int>>& matrix, int target) {

int i=0, j=matrix[0].size()-1;

while(i>=0 && j>=0 && i<matrix.size() && j<matrix[0].size())

{

if(matrix[i][j] == target) return true;

else if(matrix[i][j] < target) i++;

else j--;

}

return false;

}

};

1. ***Beautiful Array:***

class Solution {

public:

int part(vector<int> &arr, int a, int b, int mask)

{

int i, j = a;

for(i=a; i<=b; i++)

{

if((arr[i] & mask) != 0)

{

swap(arr[i], arr[j]);

j++;

}

}

return j;

}

void sort(vector<int> &arr, int a, int b, int mask)

{

if(a>=b) return;

int mid = part(arr, a, b, mask);

sort(arr, a, mid-1, mask<<1);

sort(arr, mid, b, mask<<1);

}

vector<int> beautifulArray(int n) {

vector<int> res;

int i=0;

for(i=0; i<n; i++) res.push\_back(i+1);

sort(res, 0, n-1, 1);

return res;

}

};

1. ***Median of Two Sorted Arrays:***

class Solution {

public:

vector<int> msort(vector<int> nums1, vector<int> nums2)

{

vector<int> nums;

int i=0, j=0, m, n;

while(i<nums1.size() and j<nums2.size())

{

if(nums1[i]<=nums2[j]) nums.push\_back(nums1[i++]);

else nums.push\_back(nums2[j++]);

}

for(m=i; m<nums1.size(); m++) nums.push\_back(nums1[m]);

for(m=j; m<nums2.size(); m++) nums.push\_back(nums2[m]);

return nums;

}

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2)

{

vector<int> nums = msort(nums1, nums2);

if(nums.size() == 0) return 0;

if (nums.size() % 2 == 1) return double(nums[nums.size()/2]);

return (double(nums[nums.size()/2 -1]) + double(nums[nums.size()/2]))/2;

}

};

1. ***Merge k Sorted Lists:***

class Solution {

public:

ListNode\* mergeKLists(vector<ListNode\*>& lists) {

if(lists.size() == 0) return NULL;

vector<int> v;

int i;

for(i=0; i<lists.size(); i++)

{

ListNode\* temp = lists[i];

while(temp!=NULL)

{

v.push\_back(temp->val);

temp = temp->next;

}

}

if(v.size() == 0) return NULL;

sort(v.begin(), v.end());

ListNode\* start = new ListNode(v[0]);

ListNode\* temp1 = start;

for(i=1; i<v.size(); i++)

{

temp1->next = new ListNode(v[i]);

temp1 = temp1->next;

}

return start;

}

};

1. ***Maximum Subarray:***

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int i, sum = 0, m=INT\_MIN;

for(i=0; i<nums.size(); i++)

{

sum = max(nums[i], sum+nums[i]);

m = max(m, sum);

}

return m;

}

};

1. ***Majority Element:***

class Solution {

public:

int majorityElement(vector<int>& nums) {

sort(nums.begin(), nums.end());

int i, start, count;;

for(i=0; i<nums.size(); i++)

{

if(nums[i] == nums[i+(nums.size()/2)]) return nums[i];

}

return -1;

}

};

1. ***Kth Largest Element in an Array:***

class Solution {

public:

int findKthLargest(vector<int>& nums, int k) {

int max,i;

if(nums.size() == 1) return nums[0];

sort(nums.begin(),nums.end());

if(k == 1) return nums[nums.size()-1];

if(k == nums.size()) return nums[0];

for(i=nums.size()-1; i>nums.size()-1-k; i--)

{

cout<<nums[i]<<" ";

max = nums[i];

}

return max;

}

};

1. ***Longest Substring with At Least K Repeating Characters:***

class Solution {

public:

int longsubstr(string s, int start, int end, int k)

{

if (end < k) return 0;

int countMap[26] = {0};

for (int i = start; i < end; i++)

countMap[s[i] - 'a']++;

for (int mid = start; mid < end; mid++) {

if (countMap[s[mid] - 'a'] >= k) continue;

int midNext = mid + 1;

while (midNext < end && countMap[s[midNext] - 'a'] < k) midNext++;

return max(longsubstr(s, start, mid, k),

longsubstr(s, midNext, end, k));

}

return end-start;

}

int longestSubstring(string s, int k) {

return longsubstr(s, 0, s.size(), k);

}

};

1. ***Jump Game II:***

class Solution {

public:

int jump(vector<int>& nums) {

if(nums.size() <= 1) return 0;

int i, step=0, max\_reach=nums[0], reach=nums[0], jump=1;

for(i=1; i<nums.size()-1; i++)

{

step++;

max\_reach = max(max\_reach, i+nums[i]);

if(step == reach)

{

jump++;

reach = max\_reach-i;

step = 0;

}

}

return jump;

}

};

1. ***Jump Game:***

class Solution {

public:

bool canJump(vector<int>& nums) {

int n=nums.size(), reachable=0;

for(int i=0;i<n;i++)

{

if(i>reachable) return false;

reachable=max(reachable,i+nums[i]);

}

return true;

}

};

1. ***Is Subsequence:***

class Solution {

public:

bool isSubsequence(string s, string t) {

int i=0, j=0;

while(true)

{

while(j<t.size() && s[i]!=t[j])

j++;

if(j>=t.size())

break;

i++;

j++;

}

return i == s.size();

}

};

1. ***Non-overlapping Intervals:***

class Solution {

public:

int eraseOverlapIntervals(vector<vector<int>>& intervals) {

sort(intervals.begin(), intervals.end());

int l=0, min=0, le, re, rs, i;

for(i=1; i<intervals.size(); i++)

{

le = intervals[l][1];

rs = intervals[i][0];

if(le > rs)

{

re = intervals[i][1];

min++;

if(le > re) l = i;

}

else l = i;

}

return min;

}

};

1. ***Assign Cookies:***

class Solution {

public:

int findContentChildren(vector<int>& g, vector<int>& s) {

if(s.size() == 0) return 0;

sort(g.begin(), g.end());

sort(s.begin(), s.end());

int satisfy=0, i=g.size()-1, j=s.size()-1;

while(i>=0)

{

if(j>=0 && s[j] >= g[i])

{

satisfy++;

j--;

}

i--;

}

return satisfy;

}

};

1. ***Can Place Flowers:***

class Solution {

public:

bool canPlaceFlowers(vector<int>& flowerbed, int n) {

int i, count=0;

for(i=0; i<flowerbed.size(); i++)

{

if((flowerbed[i]==0 && (i==0 || flowerbed[i-1]==0)) && (i==flowerbed.size()-1 || flowerbed[i+1] == 0))

{

flowerbed[i] = 1;

count++;

}

}

if(count >= n) return true;

else return false;

}

};

1. ***Minimum Subsequence in Non-Increasing Order:***

class Solution {

public:

vector<int> minSubsequence(vector<int>& nums) {

if(nums.size() == 1) return nums;

sort(nums.begin(), nums.end());

vector<int> res;

int i=0, j=nums.size()-1, sum1 = nums[0], sum2 = nums[nums.size()-1];

res.push\_back(nums[nums.size()-1]);

while(i<j)

{

if(sum1 >= sum2)

{

j--;

sum2 = sum2 + nums[j];

res. push\_back(nums[j]);

}

if(sum1 < sum2)

{

i++;

sum1 = sum1 + nums[i];

}

}

return res;

}

};

1. ***Calculate Money in Leetcode Bank:***

class Solution {

public:

int totalMoney(int n) {

int i=1, j=0, sum=0, day=1;

while(day<=n)

{

i = 1;

while(i<=7 && day<=n)

{

sum = sum + i + j;

i++;

day++;

}

j++;

}

return sum;

}

};

1. ***Minimum Operations to Make the Array Increasing:***

class Solution {

public:

int minOperations(vector<int>& nums) {

int i, op = 0;

for(i=0; i<nums.size()-1; i++)

{

if(nums[i] > nums[i+1])

{

op = op + nums[i] - nums[i+1] + 1;

nums[i+1] = nums[i]+1;

}

else if(nums[i] == nums[i+1])

{

nums[i+1] = nums[i+1] + 1;

op++;

}

else

{

while(nums[i] >= nums[i+1])

{

nums[i+1] = nums[i+1] + 1;

op++;

}

}

}

return op;

}

};

1. ***Maximum Element After Decreasing and Rearranging:***

class Solution {

public:

int maximumElementAfterDecrementingAndRearranging(vector<int>& arr) {

int i, temp;

sort(arr.begin(), arr.end());

arr[0] = 1;

for(i=1; i<arr.size(); i++)

{

if(abs(arr[i] - arr[i-1]) <= 1) continue;

else

{

if(arr[i] < arr[i-1])

{

temp = arr[i];

arr[i] = arr[i-1];

arr[i-1] = temp;

}

arr[i] = arr[i-1] + 1;

}

}

return \*max\_element(arr.begin(), arr.end());

}

};

1. ***Candy***:

class Solution {

public:

int candy(vector<int>& ratings) {

vector<int> res(ratings.size(), 1);

int i, tcandies;

for(i=1; i<ratings.size(); i++)

if(ratings[i] > ratings[i-1]) res[i] = res[i-1] + 1;

tcandies = res[res.size()-1];

for(int i=ratings.size()-2;i>=0;i--)

{

if(ratings[i]> ratings[i+1]) res[i]= max(res[i],res[i+1] + 1);

tcandies+=res[i];

}

return tcandies;

}

};

1. ***Remove Duplicate Letters:***

class Solution {

public:

string removeDuplicateLetters(string s) {

string res="";

int last[26]={},used[26]={};

int n=s.size();

for(int i=0;i<n;i++) last[s[i]-'a']=i;

for(int i=0;i<n;i++)

{

if(used[s[i]-'a']++>0) continue;

while(!res.empty() && res.back()>s[i] && i<last[res.back()-'a'])

{

used[res.back()-'a']=0;

res.pop\_back();

}

res.push\_back(s[i]);

}

return res;

}

};

1. ***Gas Station:***

class Solution {

public:

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

int total\_cost=0,total\_fuel=0,n=cost.size();

for(int i=0;i<n;i++)

{

total\_cost+=cost[i];

total\_fuel+=gas[i];

}

if(total\_fuel<total\_cost) return -1;

int curr\_fuel=0,start=0;

for(int i=0;i<n;i++)

{

if(curr\_fuel<0)

{

start=i;

curr\_fuel=0;

}

curr\_fuel+=(gas[i]-cost[i]);

}

return start;

}

};

1. ***Check If Array Pairs Are Divisible by k:***

class Solution {

public:

bool canArrange(vector<int>& arr, int k) {

vector<int> m(k);

for(auto e: arr) ++m[(e%k+k)%k];

for(int i = 1; i <= k/2; i++)

if(m[i] != m[k-i]) return false;

return m[0]%2 ==0;

}

};

1. ***Divide Array in Sets of K Consecutive Numbers:***

class Solution {

public:

bool isPossibleDivide(vector<int>& nums, int k) {

unordered\_map<int, int> mp = {};

for (int n : nums) ++mp[n];

while (mp.size())

for (int j = 0, n = min\_element(begin(mp), end(mp))->first; ++j <= k; ++n)

if (j < k && !mp.count(n + 1)) return false;

else if (--(mp[n]) == 0) mp.erase(n);

return true;

}

};

1. ***K Closest Points to Origin:***

class Solution {

public:

vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {

int i;

vector<vector<int>>v;

vector<pair<double,int>>v1;

for(i=0;i<points.size();i++)

{

double x;

x=sqrt((points[i][0]\*points[i][0])+(points[i][1]\*points[i][1]));

v1.push\_back(make\_pair(x,i));

}

sort(v1.begin(),v1.end());

for(int i=0;i<k;i++)

{

int j=v1[i].second;

v.push\_back(points[j]);

}

return v;

}

};

1. ***Valid Permutations for DI Sequence:***

class Solution {

public:

int numPermsDISequence(string s) {

int n = s.size(), mod = 1e9 + 7, sum;

vector<int> curr(n + 1, 1), next;

for (int k = 0; k < n; k++)

{

next.assign(n + 1 - k, 0);

sum = 0;

if (s[k] == 'D')

for (int i = n-k; i; i--)

next[i-1] = (next[i-1] + (sum = (sum + curr[i]) % mod)) % mod;

else

for (int i = 0; i < n-k; i++)

next[i] = (next[i] + (sum = (sum + curr[i]) % mod)) % mod;

swap(curr, next);

curr.pop\_back();

}

return curr[0];

}

};

1. ***Burst Balloons:***

class Solution {

public:

int maxCoins(vector<int>& nums) {

int n = nums.size();

int len = n+2;

vector<int> arr = {1};

for(auto& num : nums)

arr.push\_back(num);

arr.push\_back(1);

vector< vector<int> > dp(len, vector<int>(len, 0));

for(int gap = 2; gap < len; gap++) {

for(int left = 0; left < len-gap; left++) {

int right = left+gap;

int cur = 0;

for(int mid = left+1; mid < right; mid++) {

cur = max(cur, dp[left][mid]+dp[mid][right]+arr[left]\*arr[mid]\*arr[right]);

}

dp[left][right] = cur;

}

}

return dp[0][len-1];

}

};

1. ***Count of Smaller Numbers After Self:***

class Solution {

public:

void merge\_and\_count(vector<pair<int,int>>&p,int s,int mid,int e,vector<int>&ans){

int i=s;

int j=mid+1;

while(i<=mid and j<=e)

{

if(p[i].first>p[j].first){

j++;

}else{

ans[p[i].second]+=j-(mid+1);

i++;

}

}

while(i<=mid and j>e) ans[p[i++].second]+=j-(mid+1);

sort(p.begin()+s,p.begin()+e+1);

}

void countSmaller(vector<pair<int,int>>&p,int s,int e,vector<int>&ans){

if(s>=e) return;

int mid=(s+e)/2;

countSmaller(p,s,mid,ans);

countSmaller(p,mid+1,e,ans);

merge\_and\_count(p,s,mid,e,ans);

return;

}

vector<int> countSmaller(vector<int>& nums){

vector<pair<int,int>> p;

for(int i=0;i<nums.size();i++) p.push\_back({nums[i],i});

vector<int> ans(p.size(),0);

countSmaller(p,0,p.size()-1,ans);

return ans;

}

};

1. ***Count of Range Sum:***

class Solution {

public:

long long mergeSort(vector<long long> &sums, int start, int end, long long l, long long u){

if(start == end) return 0;

if(start + 1 == end)

{

if(sums[start] >= l && sums[start] <= u)

return 1;

return 0;

}

int med = start + (end - start)/2;

long long r1 = mergeSort(sums,start,med,l,u);

long long r2 = mergeSort(sums,med,end,l,u);

int left = med, right = med;

long long res = 0;

for(int i = start; i < med; ++i)

{

while(left < end && sums[left] - sums[i] < l) left++;

while(right < end && sums[right] - sums[i] <= u) right++;

res += right - left;

}

inplace\_merge(begin(sums)+ start, begin(sums) + med, begin(sums) + end);

return res + r1 + r2;

}

int countRangeSum(vector<int>& nums, int lower, int upper) {

vector<long long> sums;

long long sum = 0;

for(int i = 0; i < nums.size(); ++i)

{

sum += nums[i];

sums.push\_back(sum);

}

return mergeSort(sums,0,sums.size(),lower,upper);

}

};

1. ***Expression Add Operators:***

class Solution {

public:

vector<string>ans;

void dfs(int cur, long long cnum, long long num, long long res, string str, string s){

cnum = cnum \* 10 + s[cur] - '0';

if (cur+1 == s.size()){

if (res + num\*cnum == 0)ans.push\_back(str + s[cur]);

return;

}

if (cnum != 0)dfs(cur + 1, cnum, num, res, str + s[cur], s);

dfs(cur + 1, 0, 1, res + num\*cnum, str + s[cur] + '+', s);

dfs(cur + 1, 0, -1, res + num\*cnum, str + s[cur] + '-', s);

dfs(cur + 1, 0, num\*cnum, res, str + s[cur] + '\*', s);

}

vector<string> addOperators(string num, int target) {

if (num.size() > 0)dfs(0, 0, 1, -(long long)target, "", num);

return ans;

}

};

1. Reverse Pairs:

class Solution {

public:

int sort\_and\_count(vector<int>::iterator begin, vector<int>::iterator end) {

if (end - begin <= 1) return 0;

auto mid = begin + (end - begin) / 2;

int count = sort\_and\_count(begin, mid) + sort\_and\_count(mid, end);

for (auto i = begin, j = mid; i != mid; ++i)

{

while (j != end and \*i > 2L \* \*j)

++j;

count += j - mid;

}

inplace\_merge(begin, mid, end);

return count;

}

int reversePairs(vector<int>& nums) {

return sort\_and\_count(nums.begin(), nums.end());

}

};