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

***CODES IN PDF***

1. ***Valid Parentheses:***

class Solution {

public:

bool isValid(string s) {

stack <char> st;

int i;

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

{

if(s[i] == '{' || s[i] == '[' || s[i] == '(') st.push(s[i]);

if(s[i] == '}' || s[i] == ']' || s[i] == ')')

{

if(st.size() == 0) return false;

else if(st.top() == '{' && s[i] == '}') st.pop();

else if(st.top() == '(' && s[i] == ')') st.pop();

else if(st.top() == '[' && s[i] == ']') st.pop();

else return false;

}

}

if(st.size() == 0)return true;

else return false;

}

};

1. ***Daily Temperatures:***

class Solution {

public:

vector<int> dailyTemperatures(vector<int>& T) {

vector<int> res;

stack<int> s;

int i;

for(i=T.size()-1; i>=0; i--)

{

while(!s.empty() && T[i] >= T[s.top()]) s.pop();

if(s.size() == 0) res.push\_back(0);

else if(T[s.top()]>T[i]) res.push\_back(s.top()-i);

s.push(i);

}

reverse(res.begin(), res.end());

return res;

}

};

1. ***Sliding Window Maximum:***

class Solution {

public:

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

vector<int> res;

deque<int> q;

int i;

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

{

if(!q.empty() && k <= i-q.front()) q.pop\_front();

while(!q.empty() && nums[q.back()] <= nums[i])

q.pop\_back();

q.push\_back(i);

if(i>=k-1) res.push\_back(nums[q.front()]);

}

return res;

}

};

1. ***Implement Stack using Queues:***

class MyStack {

public:

/\*\* Initialize your data structure here. \*/

queue<int> q;

MyStack() {

}

/\*\* Push element x onto stack. \*/

void push(int x) {

q.push(x);

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

{

q.push(q.front());

q.pop();

}

}

/\*\* Removes the element on top of the stack and returns that element. \*/

int pop() {

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

{

q.push(q.front());

q.pop();

}

int ele = q.front();

q.pop();

return ele;

}

/\*\* Get the top element. \*/

int top() {

return q.back();

}

/\*\* Returns whether the stack is empty. \*/

bool empty() {

if(q.empty()) return true;

else return false;

}

};

1. ***Implement Queue using Stacks:***

class MyQueue {

public:

/\*\* Initialize your data structure here. \*/

stack<int> s1, s2;

MyQueue() {

}

/\*\* Push element x to the back of queue. \*/

void push(int x) {

s1.push(x);

}

/\*\* Removes the element from in front of queue and returns that element. \*/

int pop() {

while(!s1.empty())

{

s2.push(s1.top());

s1.pop();

}

int ele = s2.top();

s2.pop();

while(!s2.empty())

{

s1.push(s2.top());

s2.pop();

}

return ele;

}

/\*\* Get the front element. \*/

int peek() {

while(!s1.empty())

{

s2.push(s1.top());

s1.pop();

}

int top = s2.top();

while(!s2.empty())

{

s1.push(s2.top());

s2.pop();

}

return top;

}

/\*\* Returns whether the queue is empty. \*/

bool empty() {

if(s1.empty()) return true;

else return false;

}

};

1. ***Online Stock Span:***

class StockSpanner {

public:

stack<pair<int,int>> s;

int i=0;

StockSpanner() {

}

int next(int price) {

int j;

while(!s.empty()&&s.top().first<=price) s.pop();

if(!s.empty()) j=s.top().second;

else j=-1;

s.push({price,i});

j=i-j;

i++;

return j;

}

};

1. ***LRU Cache:***

class LRUCache {

private:

list<int> dq;

int capacity;

unordered\_map<int,list<int> ::iterator> um\_ref;

unordered\_map<int,int> um\_val;

void update(int key)

{

if(um\_ref.count(key)==1)

dq.erase(um\_ref[key]);

dq.push\_front(key);

um\_ref[key]=dq.begin();

}

void remove()

{

um\_ref.erase(dq.back());

um\_val.erase(dq.back());

dq.pop\_back();

}

public:

LRUCache(int \_capacity) {

capacity=\_capacity;

}

int get(int key) {

if(um\_val.count(key)==0)

return -1;

update(key);

return um\_val[key];

}

void put(int key, int value) {

if(um\_val.size()==capacity && um\_val.count(key)==0)

remove();

um\_val[key]=value;

update(key);

}

};

1. ***Largest Rectangle in Histogram:***

class Solution {

public:

int largestRectangleArea(vector<int>& heights) {

stack<int> s;

int i, ans = 0, val;

heights.push\_back(0);

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

{

while(!s.empty() && heights[i] < heights[s.top()])

{

val = s.top();

s.pop();

if(s.empty()) ans = max(ans,heights[val]\*i);

else ans = max(ans,heights[val]\* (i-s.top()-1));

}

s.push(i);

}

return ans;

}

};

1. ***Binary Search:***

class Solution {

public:

int search(vector<int>& nums, int target)

{

int l = 0, r = nums.size()-1;

while(l<=r)

{

int m = l + (r - l) / 2;

if(nums[m] == target) return m;

if(nums[m] < target) l = m+1;

else r = m-1;

}

return -1;

}

};

1. ***Search in Rotated Sorted Array:***

class Solution {

public:

int search(vector<int>& nums, int target) {

int l = 0, r = nums.size()-1, m;

while(l<=r)

{

m = (l+r) / 2;

if(nums[m] == target) return m;

else if(nums[l] <= nums[m])

{

if(nums[l]<=target && nums[m]>=target) r = m - 1;

else l = m + 1;

}

else

{

if(nums[m]<=target && nums[r]>=target) l = m + 1;

else r = m - 1;

}

}

return -1;

}

};

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

class Solution {

public:

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

{

vector<int> merge;

int i,j;

for(i=0;i<nums1.size();i++)

{

merge.push\_back(nums1[i]);

}

for(j=0;j<nums2.size();j++)

{

merge.push\_back(nums2[j]);

}

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

int k = merge.size();

int median;

float med=0;

if(k==1)

{

return merge[0];

}

else

{

if(k%2!=0)

{

median = (k + 1)/2;

return merge[median-1];

}

else

{

int m1,m2;

m1 = k/2;

m2 = (k/2)+1;

med = (double)(merge[m1-1]+merge[m2-1])/2;

cout<<m1<<m2;

return med;

}

}

return 0;

}

};

1. ***Merge Sorted Array:***

class Solution {

public:

void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

int i,j=0;

for(i=0;i<nums1.size();i++)

{

if(j<nums2.size())

{

if(nums1[i]!=0) continue;

else

{

nums1[i]=nums2[j];

j++;

}

}

}

sort(nums1.begin(),nums1.end());

}

};

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. ***Missing Number:***

class Solution {

public:

int missingNumber(vector<int>& nums) {

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

sum = accumulate(nums.begin(), nums.end(), 0);

return ((n\*(n+1))/2 - sum);

}

};

1. ***Basic Calculator II:***

class Solution {

public:

int do\_opr(int n1, int n2, char op)

{

switch(op)

{

case '+': return n2+n1;

case '-': return n2-n1;

case '\*': return n2\*n1;

case '/': return n2/n1;

}

return 0;

}

int pred(char c)

{

if(c=='\*' || c=='/') return 2;

else if(c=='+' || c=='-') return 1;

else return 0;

}

int calculate(string s) {

stack<int> val;

stack<char> op;

int i, j=0, res, n1, n2, ans;

char opr;

s = '(' + s + ')';

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

{

j = 0;

if(isdigit(s[i]))

{

j = j\*10 + (s[i]-'0');

while(isdigit(s[i+1]))

{

i++;

j = j\*10 + (s[i]-'0');

}

val.push(j);

}

else if(s[i] == '(') op.push(s[i]);

else if(s[i] == ')')

{

while(!op.empty() && op.top() != '(')

{

n1 = val.top();

val.pop();

n2 = val.top();

val.pop();

opr = op.top();

op.pop();

res = do\_opr(n1, n2, opr);

val.push(res);

}

op.pop();

}

else if(s[i]=='+' || s[i]=='-' || s[i]=='\*' || s[i]=='/')

{

while(pred(s[i]) <= pred(op.top()))

{

n1 = val.top();

val.pop();

n2 = val.top();

val.pop();

opr = op.top();

op.pop();

res = do\_opr(n1,n2,opr);

val.push(res);

}

op.push(s[i]);

}

else continue;

}

res = val.top();

val.pop();

i = 1;

while(!val.empty())

{

res = val.top()\*pow(10,i) + res;

i++;

val.pop();

}

return res;

}

};

1. ***Min Stack:***

class MinStack {

public:

/\*\* initialize your data structure here. \*/

stack<int> s1, s2;

int min\_ele = INT\_MAX;

MinStack() {

}

void push(int val) {

s1.push(val);

min\_ele = min(min\_ele, val);

}

void pop() {

if(min\_ele < s1.top()) s1.pop();

else

{

s1.pop();

min\_ele = INT\_MAX;

s2 = s1;

while(!s2.empty())

{

min\_ele = min(min\_ele,s2.top());

s2.pop();

}

}

}

int top() {

return s1.top();

}

int getMin() {

return min\_ele;

}

};

1. ***Find Minimum in Rotated Sorted Array:***

class Solution {

public:

int findMin(vector<int>& nums) {

int l = 0, h = nums.size()-1, m;

while(l<h)

{

m = (l+h)/2;

if(nums[m] < nums[h]) h = m;

else l = m+1;

}

return nums[l];

}

};

1. ***Sort an Array:***

class Solution {

public:

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

priority\_queue<int, vector<int>, greater<int>> pq;

int i;

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

pq.push(nums[i]);

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

{

nums[i] = pq.top();

pq.pop();

}

return nums;

}

};

1. ***Longest Increasing Subsequence:***

class Solution {

public:

int lengthOfLIS(vector<int>& nums) {

vector<int> res;

int i, ans;

res.push\_back(nums[0]);

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

{

if(nums[i] > res.back()) res.push\_back(nums[i]);

else

{

ans=lower\_bound(res.begin(),res.end(),nums[i])-res.begin();

res[ans]=nums[i];

}

}

return res.size();

}

};

1. ***Longest Mountain in Array:***

class Solution {

public:

int longestMountain(vector<int>& arr) {

int i=0, j=0, ans=0;

while(j<arr.size())

{

j = i;

if(j+1<arr.size() && arr[j+1]>arr[j])

{

while(j+1<arr.size() && arr[j+1]>arr[j]) j++;

if(j+1<arr.size() && arr[j]>arr[j+1])

{

while(j+1<arr.size() && arr[j]>arr[j+1]) j++;

ans = max(ans,j-i+1);

}

}

i = max(i+1,j);

}

return ans;

}

};

1. ***Shortest Unsorted Continuous Subarray:***

class Solution {

public:

int findUnsortedSubarray(vector<int>& nums) {

if(nums.size() < 2) return 0;

vector<int> res = nums;

int i=0, j=nums.size()-1;

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

while(i<j)

{

if(nums[i]==res[i]) i++;

if(nums[j]==res[j]) j--;

if(nums[i]!=res[i] && nums[j]!=res[j]) break;

}

if(i==j) return 0;

else return j-i+1;

}

};

1. ***Merge K sorted arrays:***

typedef pair<int, pair<int, int> > ppi;

vector<int> Solution::solve(vector<vector<int> > &A) {

int c = 0, i, j;

vector<int> output;

priority\_queue<ppi, vector<ppi>, greater<ppi> > pq;

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

pq.push({ A[i][0], { i, 0 } });

while (pq.empty() == false)

{

ppi curr = pq.top();

pq.pop();

i = curr.second.first;

j = curr.second.second;

output.push\_back(curr.first);

if (j + 1 < A[i].size())

pq.push({ A[i][j + 1], { i, j + 1 } });

}

return output;

}

1. ***Find Median from Data Stream:***

class MedianFinder {

public:

/\*\* initialize your data structure here. \*/

priority\_queue<int> pmax; // max-heap

priority\_queue<int,vector<int>,greater<int>> pmin; //min-heap

MedianFinder() {

}

void addNum(int num) {

if(pmax.empty() || pmax.top()>num) pmax.push(num);

else pmin.push(num);

if(pmax.size()>pmin.size()+1)

{

pmin.push(pmax.top());

pmax.pop();

}

else

if(pmin.size()>pmax.size()+1)

{

pmax.push(pmin.top());

pmin.pop();

}

}

double findMedian() {

if(pmax.size() == pmin.size())

{

double mid=(pmax.top()+pmin.top())\*(1.0)/2;

return mid;

}

else

if(pmax.size()>pmin.size()) return pmax.top();

else return pmin.top();

}

};