11/11/24

1. Knapsack

class Solution {

public:

vector<vector<int>> t;

int solve(int w, vector<int> &val, vector<int>&wt,int n){

if(w==0 || n==0) return 0;

if(t[n][w]!=-1) return t[n][w];

if(w>=wt[n-1]){

return t[n][w]=max(val[n-1]+solve(w-wt[n-1],val,wt,n-1),solve(w,val,wt,n-1));

}

else{

return t[n][w]=solve(w,val,wt,n-1);

}

}

int knapSack(int capacity, vector<int> &val, vector<int> &wt) {

// code here

int n = val.size();

t.resize(n+1,vector<int>(capacity+1,-1));

return solve(capacity,val,wt,n);

}

};

Time Complexity: O(M\*N)

Space Complexity: O(M\*N)

2) Floor in sorted array

#include <bits/stdc++.h>

using namespace std;

class Solution {

public:

int findFloor(vector<int>& arr, int k) {

// Your code here

int ans = -1;

int left = 0;

int right = arr.size();

while(left<=right){

int mid = (left+right)/2;

if(arr[mid]==k){

return mid;

}else if(arr[mid]<k){

ans = mid;

left = mid+1;

}else{

right = mid-1;

}

}return ans;

}

};

int main() {

string ts;

getline(cin, ts);

int t = stoi(ts);

while (t--) {

vector<int> arr;

string input;

getline(cin, input);

stringstream ss(input);

int number;

while (ss >> number) {

arr.push\_back(number);

}

string ks;

getline(cin, ks);

int k = stoi(ks);

Solution ob;

int ans = ob.findFloor(arr, k);

cout << ans << endl;

}

return 0;

}Time Complexity: O(Log N)

Space Complexity: O(1)

3)Check equal arrays

#include *<bits/stdc++.h>*

**using** **namespace** **std**;

bool areEqual(vector<int>& arr1, vector<int>& arr2)

{

int N = arr1.size(), M = arr2.size();

**if** (N != M)

**return** false;

sort(arr1.begin(), arr1.end());

sort(arr2.begin(), arr2.end());

**for** (int i = 0; i < N; i++)

**if** (arr1[i] != arr2[i])

**return** false;

**return** true;

}

int main()

{

vector<int> arr1 = { 3, 5, 2, 5, 2 };

vector<int> arr2 = { 2, 3, 5, 5, 2 };

**if** (areEqual(arr1, arr2))

cout << "Yes";

**else**

cout << "No";

**return** 0;

}

Time Complexity: O(NLogN)

Space Complexity: O(1)

4) Palindromic linked list

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

bool isPalindrome(ListNode\* head) {

stack<int> st;

ListNode\* temp = head;

while(temp!=NULL){

st.push(temp->val);

temp = temp->next;

}temp = head;

while(temp!=NULL){

if(st.top()!=temp->val){

return false;

}st.pop();

temp = temp->next;

}return true;

}

};

Time Complexity: O(N)

Space Complexity: O(N)

5) Balanced tree or not

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

int solver(TreeNode\* root){

if(root==nullptr){

return 0;

}int left = solver(root->left);

if(left==-1){

return -1;

}

int right = solver(root->right);

if(right==-1){

return -1;

}if(abs(left-right)>1){

return -1;

}

// cout<<abs(left-right)<<","<<left<<","<<right<<" . ";

// return abs(left-right)<=1;

return 1+max(left,right);

}

bool isBalanced(TreeNode\* root) {

if(root==nullptr){

return 1;}

// }int ans = abs(solver(root->left)-solver(root->right));

return solver(root)!= -1 ;

}

}; Time complexity: O(N)

Space Complexity: O(1)

6) TRriplet Sum

class Solution {

public:

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

int target = 0;

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

set<vector<int>> s;

vector<vector<int>> output;

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

int j = i + 1;

int k = nums.size() - 1;

while (j < k) {

int sum = nums[i] + nums[j] + nums[k];

if (sum == target) {

s.insert({nums[i], nums[j], nums[k]});

j++;

k--;

} else if (sum < target) {

j++;

} else {

k--;

}

}

}

for(auto triplets : s)

output.push\_back(triplets);

return output;

}

};Time Complexity: O(NlogN)