Aswinnath TE

22AD011

1.0/1 Knapsack

class Solution {

public:

// Function to return max value that can be put in knapsack of capacity.

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

int n=val.size();

int tot=0;

for(int i:wt){

tot+=i;

}

vector<vector<int>> cache(n+1,vector<int>(capacity+1,0));

int ans=0;

for(int i=1;i<=n;i++){

for(int weight=0;weight<=capacity;weight++){

cache[i][weight]=cache[i-1][weight];

if(weight>=wt[i-1]){

cache[i][weight]=max(cache[i][weight],val[i-1]+cache[i-1][weight-wt[i-1]]);

}

ans=max(ans,cache[i][weight]);

}

}

return ans;

}

};

2.Floor of an array

lass Solution {

public:

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

int low=0;

int ans=-1;

int n=arr.size();

int high=n-1;

while(low<=high){

int mid=(low+high)/2;

if(arr[mid]<=k){

ans=mid;

low=mid+1;

}

else{

high=mid-1;

}

}

return ans;

}

};

3.check arrays equal

class Solution {

public:

// Function to check if two arrays are equal or not.

bool check(vector<int>& arr1, vector<int>& arr2) {

function<map<int,int>(vector<int>)> get = [](vector<int> arr){

map<int,int> mp;

for(int i:arr){

mp[i]++;

}

return mp;

};

map<int,int> mp1=get(arr1);

map<int,int> mp2=get(arr2);

for(auto i : mp1){

if(mp2.find(i.first)==mp2.end()){

return false;

}

if(mp2[i.first]!=i.second){

return false;

}

}

for(auto i : mp2){

if(mp1.find(i.first)==mp1.end()){

return false;

}

if(mp2[i.first]!=i.second){

return false;

}

}

return true;

}

};

4.triplet sum

class Solution {

public:

// Should return true if there exists a triplet in the

// array arr[] which sums to x. Otherwise false

bool find3Numbers(int arr[], int n, int x) {

sort(arr, arr + n);

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

int left=i+1;

int right=n-1;

while(left<right){

int cur=arr[left]+arr[right]+arr[i];

if(cur==x){

return true;

}

else if(cur>x){

right--;

}

else{

left++;

}

}

}

return false;

}

};

5.balanced Tree

class Solution{

public:

//Function to check whether a binary tree is balanced or not.

bool isBalanced(Node \*root)

{

function<int(Node\*)> height = [&](Node\* root){

if(root==NULL){

return 0;

}

int ans=1;

int left=height(root->left);

int right=height(root->right);

return 1+max(left,right);

};

function<bool(Node\*)> dfs = [&](Node\* root){

if(root==NULL){

return true;

}

int left=height(root->left);

int right=height(root->right);

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

return false;

}

return dfs(root->left) && dfs(root->right);

};

return dfs(root);

}

};

6.palindrome linked list

class Solution {

private:

bool check(vector<int >arr){

int n=arr.size();

int s=0;

int e=n-1;

while(s<=e){

if(arr[s]!=arr[e]){

return 0;

}

s++;

e--;

}

return 1;

}

public:

bool isPalindrome(Node \*head) {

vector<int>arr;

Node\*temp=head;

while(temp!=NULL){

arr.push\_back(temp->data);

temp=temp->next;

}

return check(arr);

}

};