WEEK 9

1 Given a graph, Design an algorithm and implement it using a program to implement Floyd Warshall all pair shortest path algorithm.

CODE :

```
#include <iostream>
#include <vector>
using namespace std;
const int INF = 999;
void floydWarshall(vector<vector<int>> &dist) {
int V = dist.size();
for (int k = 0; k < V; k++) {
for (int i = 0; i < V; i++) {
for (int j = 0; j < V; j++) {
if (dist[i][k] != INF && dist[k][j] != INF)
dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}
}
}
}
int main() {
int V;
cout << "Enter number of vertices: ";
cin >> V;
vector<vector<int>> dist(V, vector<int>(V));
cout << "Enter the adjacency matrix (use " << INF << " for no direct edge):\n";
for (int i = 0; i < V; i++) {
for (int j = 0; j < V; j++) {
cin >> dist[i][j];

if (i == j)
```

```
dist[i][j] = 0; // Distance from a vertex to itself is always 0
}
}
floydWarshall(dist);
cout << "\nShortest distances between every pair of vertices:\n";
for (int i = 0; i < V; i++) {
for (int j = 0; j < V; j++) {
if (dist[i][j] == INF)
cout << "INF ";
else
cout << dist[i][j] << " ";
}
cout << endl;
}
return 0;
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp> cd 'c:\Harshit Srikoti\cpp\Daa Lab\output'
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week9_1.exe'
Enter number of vertices: 4
Enter the adjacency matrix (use 999 for no direct edge):
0 5 999 10
999 0 3 999
999 999 0 1
999 999 999 0

Shortest distances between every pair of vertices:
0 5 8 9
INF 0 3 4
INF INF 0 1
INF INF INF 0
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

2. Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry

CODE :

```
#include <iostream>
#include<vector>
#include<algorithm>
using namespace std;
class Item{
public:
int wt;
int val;
Item(int w,int v){
wt=w;
val=v;
}
static bool compare(Item a,Item b){
double r1=(1.0*a.val)/a.wt;
double r2=(1.0*b.val)/b.wt;
return r1>r2;
} double fractional_knapsnap(vector<Item>&items,int k){
sort(items.begin(),items.end(),compare);
double profit=0.0;
for(auto it:items){
if(k>=it.wt){
profit+=it.val;
k-=it.wt;
}else{
profit+=it.val*(1.0*k)/it.wt;
```

```
k=0;


break;
} }
return profit;
}
};
int main(){
int n;
cout<<"enter number of items: ";
cin>>n;
int val[n],w[n];
cout<<"enter value of each item: ";
for(int i=0;i<n;i++){
cin>>val[i];
}
cout<<"enter weight for each item: ";
for(int i=0;i<n;i++){
cin>>w[i];
}
vector<Item> items;
for(int i=0;i<n;i++) {
items.push_back(Item(val[i],w[i]));
}
int cap;
cout<<"enter capacity: ";
cin>>cap;
Item obj(0,0);
double result = obj.fractional_knapsnap(items,cap);
cout << "Maximum profit: " << result << endl;
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week9_2.exe'
enter number of items: 6
enter value of each item: 6 10 3 5 1 3
enter weight for each item: 6 2 1 8 3 5
enter capacity: 16
Maximum profit: 22.3333
```

3. Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n, computation cost of merging them is O(m+n). (Hint: use greedy approach).

CODE :

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;
void merge(vector<int>& arr,int n) {
int cost=0;
priority_queue<int,vector<int>,greater<int>> pq;
for(int i=0;i<n;i++) {
pq.push(arr[i]); }
while(pq.size()>1) {
int p1=pq.top();
pq.pop();
int p2=pq.top();
pq.pop();
cost+=p1+p2;
pq.push(p1+p2); }
cout<<"Cost of merging: "<<cost<<endl;}
int main() {
int n;
cout<<"Enter number of files: ";
cin>>n;
vector<int>arr(n);
cout<<"Enter size of files: ";
for(int i=0;i<n;i++) {
cin>>arr[i]; }
merge(arr,n);}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp> cd 'c:\Harshit Srikoti\cpp\Daa Lab\output'
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week9_3.exe'
Enter number of files: 10
Enter size of files: 10 5 100 50 20 15 5 20 100 10
Cost of merging: 895
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

WEEK 10

1. Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.

CODE :

```
#include<bits/stdc++.h>
using namespace std;
struct Activity{
int start,end,index;
};
bool compare(Activity a,Activity b){
return a.end<b.end;
}
void maxacitivites(int start[],int end[],int n){
vector<Activity>activity;
vector<int>result;
for(int i=0;i<n;i++){
activity.push_back({start[i],end[i],i+1});
}
sort(activity.begin(),activity.end(),compare);
int last_activity=0;
for(auto it:activity){
if(it.start>last_activity){
result.push_back(it.index);
last_activity=it.end;
}}
cout<<"number of non-conflicting activities:"<<result.size();
cout<<endl;
cout<<"list of non-conflicting activities: ";
for(auto i:result){
```

```
cout<<i<<" ";


}}
int main(){
int n;
cout<<"enter number of activities: ";
cin>>n;
int start[n];
int end[n];
cout<<"enter start activities: ";
for(int i=0;i<n;i++){
cin>>start[i];}
cout<<"enter end activities: ";
for(int i=0;i<n;i++){
cin>>end[i];
}maxacitivites(start,end,n);}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> cd 'c:\Harshit Srikoti\cpp\Daa Lab\output'
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week10_1.exe'
enter number of activities: 10
enter start activities: 1 3 0 5 3 5 8 8 2 12
enter end activities: 4 5 6 7 9 9 11 12 14 16
number of non-conflicting activities:4
list of non-conflicting activities: 1 4 7 10
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

2. Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.

CODE :

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
int main() {
int n;
cout << "Enter number of tasks: ";
cin >> n;
vector<int> time(n), deadline(n);
cout << "Enter time for each task: ";
for (int i = 0; i < n; ++i) cin >> time[i];
cout << "Enter deadline for each task: ";
for (int i = 0; i < n; ++i) cin >> deadline[i];
vector<pair<int, int>> tasks;
for (int i = 0; i < n; ++i)
tasks.push_back({deadline[i], time[i]});
sort(tasks.begin(), tasks.end());
priority_queue<int> pq;
int current_time = 0;
for (auto &task : tasks) {
int d = task.first;
int t = task.second;
if (current_time + t <= d) {
current_time += t;
pq.push(t);
```

```
} else if (!pq.empty() && pq.top() > t) {
current_time -= pq.top();
pq.pop();
current_time += t;
pq.push(t);
}
}
cout << "Maximum tasks completed: " << pq.size() << endl;
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> cd 'c:\Harshit Srikoti\cpp\Daa Lab\output'
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week10_2.exe'
Enter number of tasks: 7
Enter time for each task: 2 1 3 2 2 2 1
Enter deadline for each task: 2 3 8 6 2 5 3
Maximum tasks completed: 4
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

3. Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array.

CODE :

```
#include<iostream>
#include<vector>
#include<climits>
using namespace std;
int maximum_num(vector<int>&arr){
int count=0,candidate=-1;
for(auto it:arr){
if(count==0){
candidate=it;
}else{
count+=(it==candidate)?1:-1;
}
}
count = 0;
for (int i:arr) {
if (i==candidate) count++;
}
if (count>arr.size()/2)
return candidate;
else
return -1;
}
int main(){
int n;
cout<<"enter array size: ";
cin>>n;
```

```
vector<int>arr(n);


cout<<"enter array elements: ";
for(int i=0;i<n;i++){
cin>>arr[i];
}
int res=maximum_num(arr);
if(res){
cout<<"Maximum element: "<<res;
}else{
cout<<"no maximum element";
}
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> cd 'c:\Harshit Srikoti\cpp\Daa Lab\output'
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week10_3.exe'
enter array size: 8
enter array elements: 7 4 6 7 7 7 2 7
Maximum element: 7
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

WEEK 11 :

1. Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied.

CODE :

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

using namespace std;

int mcm(vector<int> arr, int i,int j,vector<vector<int>> & dp)

{

if(i >= j)

{

return 0 ;

}

if(dp[i][j] != -1)

{

return dp[i][j];

}

int res = INT_MAX;

for(int k = i ;k<= j-1 ;k++)

{

int temp = mcm(arr,i,k,dp) + mcm(arr,k+1,j,dp)+arr[i-1]*arr[k]*arr[j];

res = min(temp,res);

}

return dp[i][j] = res ;

}

int main()

{

int n ;

cin>> n ;

vector<int> v(n);

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

```
{
cin>>v[i];
}
vector<vector<int> > dp(n,vector<int>(n,-1));
cout<<mcm(v,1,n-1,dp);
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week11_1.exe'
3
10 5
20 5
1000
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

2. Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N.

CODE :

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

using namespace std;

int cc(vector<int> arr,int i,int t, vector<vector<int>> & dp)

{

if(i == 0)

{

return (t%arr[i] == 0);

}

if(dp[i][t] != -1)

{

return dp[i][t];

}

int ntake = cc(arr,i-1,t,dp);

int take = 0 ;

if (t >= arr[i])

{

take = cc(arr,i,t-arr[i],dp);

}

return dp[i][t] = (take+ntake);

}

int main()

{

int n ;

cin>> n ;

int t ;

cin>> t ;

vector<int> v(n);
```

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


{
cin>> v[i];
}
vector<vector<int>> dp(n,vector<int> (t+1,-1));
cout<< cc(v,n-1,t,dp);
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week11_2.exe'
3
4
1 2 3
4
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

3. Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem.

CODE :

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

using namespace std;

int sbk(vector<int> arr, int i,int t,vector<vector<int>>&dp)

{

if(t == 0 )return 1;

if (i == 0)

{

return (arr[i]== t);

}

if(dp[i][t] != -1)

{

return dp[i][t];

}

int nt = sbk(arr,i-1,t,dp);

int take = 0;

if(t>= arr[i])

{

t = sbk(arr,i-1,t-arr[i],dp);

}

return dp[i][t] = (t||nt);

}

int main()

{

int n ;

cin>> n;

int t ;

int sum = 0;
```

```
vector<int> v(n);


vector<vector<int>> dp(n,vector<int>(t+1,-1));
for (int i =0 ;i<n ;i++)
{
cin>> v[i];
sum += v[i];
}
t = sum/2;
if ( sum%2 == 0 )
{
if( sbk(v,n-1,t,dp))
{
cout<< "true" ;
}
else
{
cout<< "false";
}
}
else {
cout<< false;
}
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp> cd 'c:\Harshit Srikoti\cpp\Daa Lab\output'
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week11_3.exe'
4
1 5 11 5
true
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

WEEK 12

1. Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

CODE :

```
#include<iostream>
#include<vector>
#include<string>
#include<algorithm>
using namespace std;
int lsb(string &s1,string &s2,int i,int j,vector<vector<int>>&dp){
if(i<0||j<0) return 0;
if(dp[i][j]!=-1) return dp[i][j];
if(s1[i]==s2[j]) return dp[i][j]=1+lsb(s1,s2,i-1,j-1,dp);
return dp[i][j]=0+max(lsb(s1,s2,i-1,j,dp),lsb(s1,s2,i,j-1,dp));
}
int main(){
string s1,s2;
cin>>s1;
cin>>s2;
int n=s1.size();
int m=s2.size();
vector<vector<int>>dp(n,vector<int>(m,-1));
cout<<lsb(s1,s2,n-1,m-1,dp);
return 0;
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> cd 'c:\Harshit Srikoti\cpp\Daa Lab\output'
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week12_1.exe'
AGGTAB
GXTXAAYYB
4
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

2. Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight <= w and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property)

CODE :

```
#include <bits/stdc++.h>
using namespace std;
int knapsackUtil(vector<int>& wt, vector<int>& val, int ind, int W, vector<vector<int>>&
dp) {
if (ind == 0) {
if(wt[0] <= W){
return val[0];}
else{
return 0 ;}
}
if (dp[ind][W] != -1)
return dp[ind][W];
int notTaken = 0 + knapsackUtil(wt, val, ind - 1, W, dp);
int taken = INT_MIN;
if (wt[ind] <= W)
taken = val[ind] + knapsackUtil(wt, val, ind-1, W - wt[ind], dp);
return dp[ind][W] = max(notTaken, taken);
}
int main() {
vector<int> wt = {2, 4, 5}; // Weight of items
vector<int> val = {30, 40, 60}; // Value of items
int W = 6; // Weight capacity of the knapsack
int n = wt.size(); // Number of items
vector<vector<int>>dp(n,vector<int>(W+1,-1));
cout << "The Maximum value of items the thief can steal is " << knapsackUtil(wt, val, n - 1,
W, dp) <<
```

```
endl;


return 0;
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week12_2.exe'
The Maximum value of items the thief can steal is 70
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```

3. Given a string of characters, design an algorithm and implement it using a program to print all possible permutations of the string in lexicographic order.

CODE :

```
#include<bits/stdc++.h>
using namespace std;
void func(string s,vector<string> & ans, int ind )
{
if(ind>= s.size()){
ans.push_back(s);
return ;
}
for (int j = ind ;j<s.size() ;j++){
swap(s[ind], s[j]);
func(s,ans,ind+1);
swap(s[ind],s[j]); }}
int main(){
string s ;
cin>> s;
vector<string> ans ;
func(s,ans,0);
cout<< ans.size()<<endl;
sort(ans.begin(),ans.end());
for(int i =0 ;i<ans.size();i++) {
cout<<ans[i]<< " ";
}
}
```

OUTPUT :

```
PS C:\Harshit Srikoti\cpp\Daa Lab\output> & .\'week12_3.exe'
CAB
6
ABC ACB BAC BCA CAB CBA
PS C:\Harshit Srikoti\cpp\Daa Lab\output>
```