**Week 6**

1. **Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)**

#include<iostream>

#include<vector>

using namespace std;

bool dfs(int src,vector<int>&vis,vector<vector<int>>&v,int n,int dest)

{

vis[src]=1;

if(src==dest){

return true;

}

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

if(vis[i]!=1 && v[src][i]!=0) {

return dfs(i,vis,v,n,dest);

}

}

return false;

}

int main(){

int n;

cout<<"enter the no. of nodes"<<endl;

cin>>n;

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

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

for(int j=1;j<n+1;j++)

cin>>v[i][j];

}

vector<int>vis(n+1);

int src,dest;

cout<<"Enter two vertex to find path: ";

cin>>src>>dest;

bool flag=dfs(src,vis,v,n,dest);

if(flag==true){

cout<<"Path exist";

}

else{

cout<<"Path does not exist";

}

return 0;

}

**Output:**

**enter the no. of nodes**

**2**

**34 9**

**45 65**

**Enter two vertex to find path: 2 0**

**Path does not exist**

1. **Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)**

**Program:**

#include<bits/stdc++.h >

using namespace std;

bool bipartite(int node , vector<vector<int>>&adj , vector<int>&color){

queue<int>q;

q.push(node);

color[node]=1;

while(!q.empty()){

int t=q.front();

q.pop();

for(int i=1;i<adj[t].size();i++){

if(adj[t][i]==1 && color[i]==-1){

color[i]=1-color[t];

q.push(i);

}

else if(adj[t][i]==1 && color[i]==color[t]){

return false;

}

}

}return true;

}

int main(){

int n;

cout<<"enter the no. of nodes"<<endl;

cin>>n;

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

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

{

for(int j=1;j<n+1;j++)

cin>>adj[i][j];

}

vector<int>color(n+1,-1);

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

if(color[i]==-1){

if(bipartite(i,adj,color)){

cout<<"Bipertite";

return 0; }

}

}

cout<<"Not a Bipartite!!";

return 0;

}

**Output:**

**enter the no. of nodes**

**3**

**2 5 8**

**4 6 4**

**1 6 9**

**Bipertite**

III. Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

**Program :**

#include<iostream>

#include<vector>

using namespace std;

bool cheak\_cycle(int node , vector<vector<int>>&adj , vector<int>&vis, vector<int>&dfsvis){

vis[node]=1;

dfsvis[node]=1;

for(int i=1;i<adj[node].size();i++){

if(adj[node][i]==1 && vis[i]==0) {

if(cheak\_cycle(adj[node][i],adj,vis,dfsvis))

return true;

}

else if(adj[node][i]==1 && vis[i]==1 && dfsvis[i]==1){

return true;

}

}

dfsvis[node]=0;

return false;

}

int main(){

int n;

cout<<"enter the no. of nodes"<<endl;

cin>>n;

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

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

for(int j=1;j<n+1;j++)

cin>>adj[i][j];

}

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

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

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

if(!vis[i]){

if(cheak\_cycle(i,adj,vis,dfsvis)){

cout<<"Cycle Found";

return 0;

}

}

}

cout<<"No cycle found!!";

return 0;

}

**Output:**

**enter the no. of nodes**

**3**

**3 6 9**

**4 6 0**

**1 4 2**

**No cycle found!!**

**Week 7**

**I. After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay’s house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends’ location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend’s location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra’s algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay’s results? (Hint: Print shortest path and distance from friends’ location to Akshay’s house)**

**Program :**

#include<bits/stdc++.h>

using namespace std;

int main(){

int n;

cout<<"Enter number of nodes : ";

cin>>n;

int graph[20][20];

cout<<"Enter matrix: \n";

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

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

cin>>graph[i][j];

}

}

int distance[20];

int sssp[20];

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

distance[i]=INT\_MAX;

sssp[i]=false;

}

distance[0]=0;

int c=0;

while(c<n-1){

int min=INT\_MAX;

int dest=0;

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

if(sssp[i]==false && distance[i]<min){

min=distance[i];

dest=i;

}

}

sssp[dest]=true;

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

if(graph[dest][i] && sssp[i]==false && distance[dest]+graph[dest][i]<distance[i] ){

distance[i]=distance[dest]+graph[dest][i];

}

}

c++;

}

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

cout<<i+1<<"-"<<distance[i]<<" \n";

}}

**Output:**

**Enter number of nodes : 3**

**Enter matrix:**

**4 7 9**

**1 3 2**

**0 6 4**

**2-7**

**3-9**

**II. Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.**

**Program:**

#include <bits/stdc++.h>

using namespace std;

void calulate(vector<int> &pa, int i)

{

cout << i + 1 << " ";

if (pa[i] >= 0)

calulate(pa, pa[i]);

}

void find\_path(int \*\*graph, int m, int sour)

{

vector<int> dis(m, INT\_MAX), pa(m, -1);

dis[sour] = 0;

for (int ki = 0; ki < m - 1; ki++)

{

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

{

for (int j = 0; j < m; j++)

{

if (graph[i][j] != 0)

{

if (dis[j] > dis[i] + graph[i][j])

{

dis[j] = dis[i] + graph[i][j];

pa[j] = i;

}

}

}

}

}

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

{

calulate(pa, i);

cout << ": " << dis[i] << endl;

}

}

int main()

{

int m, source, ed;

cin >> m;

int \*\*graph = (int \*\*)malloc(m \* sizeof(int \*));

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

graph[i] = (int \*)malloc(m \* sizeof(int));

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

for (int j = 0; j < m; j++) {

cin >> graph[i][j];

}

}

cin >> source;

find\_path(graph, m, source - 1);

}

**Output:**

**5**

**0 4 1 0 0**

**0 0 0 0 4**

**0 2 0 4 0**

**0 0 0 0 4**

**0 0 0 0 0**

**1**

**1 : 0**

**2 3 1 : 3**

**3 1 : 1**

**4 3 1 : 5**

**5 2 3 1 : 7**

**III. Given a directed graph with two vertices ( source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.**

**Program :**

#include <bits/stdc++.h>

using namespace std;

#define V 4

#define INF INT\_MAX

int shortestPath(int graph[][V], int u, int v, int k)

{

if (k == 0 && u == v)

return 0;

if (k == 1 && graph[u][v] != INF)

return graph[u][v];

if (k <= 0)

return INF;

int res = INF;

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

if (graph[u][i] != INF && u != i && v != i){

int rec\_res = shortestPath(graph, i, v, k-1);

if (rec\_res != INF)

res = min(res, graph[u][i] + rec\_res);

}

}

return res;

}

int main()

{

int graph[V][V] = { {0, 10, 3, 2},

{INF, 0, INF, 7},

{INF, INF, 0, 6},

{INF, INF, INF, 0}

};

int u = 0, v = 3, k = 2;

cout << "Weight of the shortest path is " <<

shortestPath(graph, u, v, k);

return 0;

}

**Output:**

**Weight of the shortest path is 9**

**Week 8**

**I. Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)**

**Program:**

#include<bits/stdc++.h>

using namespace std;

int main(){

int n;

cout<<"Enter number of nodes : ";

cin>>n;

int graph[20][20];

cout<<"Enter matrix: \n";

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

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

cin>>graph[i][j];

}

}

int parent[20];

int key[20];

bool mst[20];

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

parent[i]=-1;

key[i]=INT\_MAX;

mst[i]=false;

}

parent[0]=-1;

key[0]=0;

int c=0;

while(c<n-1){

int min=INT\_MAX;

int dest=0;

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

if(mst[i]==false && key[i]<min){

min=key[i];

dest=i;

}

}

mst[dest]=true;

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

if(graph[dest][i] && mst[i]==false && graph[dest][i]<key[i]){

parent[i]=dest;

key[i]=graph[dest][i];

}

}

c++;

}

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

cout<<parent[i]<<"-"<<i<<" \n";

}

}

**Output:**

**Enter number of nodes : 3**

**Enter matrix:**

**1 2 3**

**4 5 6**

**7 8 9**

**0-1**

**0-2**

**II. Implement the previous problem using Kruskal's algorithm.**

**Program:**

#include <bits/stdc++.h>

#define NIL -1

using namespace std;

int findParent(vector<int> parent, int u)

{

if (parent[u] < 0)

return u;

return findParent(parent, parent[u]);

}

bool UnionByWeight(vector<int> &parent, int u, int v)

{

int pu = findParent(parent, u);

int pv = findParent(parent, v);

if (pu != pv)

{

if (parent[pu] <= parent[pv])

{

parent[pu] += parent[pv];

parent[pv] = pu;

}

else

{

parent[pv] += parent[pu];

parent[pu] = pv;

}

return true;

}

return false;

}

int kruskals(int \*\*graph, int n)

{

vector<pair<int, pair<int, int>>> G;

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

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

if (graph[i][j] != 0)

G.push\_back(make\_pair(graph[i][j], make\_pair(i, j)));

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

vector<int> parent(n, NIL);

int s = 0;

for (auto i : G)

{

int u = i.second.first;

int v = i.second.second;

int w = i.first;

if (UnionByWeight(parent, u, v))

s += w;

}

return s;

}

int main()

{

int n;

cin >> n;

int \*\*graph;

graph = (int \*\*)malloc(n \* sizeof(int \*));

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

graph[i] = (int \*)malloc(n \* sizeof(int));

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

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

cin >> graph[i][j];

cout << "Minimum spanning weight : " << kruskals(graph, n) << endl;

return 0;

}

**Output:**

**4**

**0 0 7 5**

**2 4 6 3**

**0 4 3 1**

**9 0 6 5**

**Minimum spanning weight : 6**

**III. Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.**

**Program:**

#include <bits/stdc++.h>

#define NIL -1

using namespace std;

int findParent(vector<int> parent, int u)

{

if (parent[u] < 0)

return u;

return findParent(parent, parent[u]);

}

bool UnionByWeight(vector<int> &parent, int u, int v)

{

int pu = findParent(parent, u);

int pv = findParent(parent, v);

if (pu != pv)

{

if (parent[pu] <= parent[pv]) {

parent[pu] += parent[pv];

parent[pv] = pu;

}

else

{

parent[pv] += parent[pu];

parent[pu] = pv;

}

return true;

}

return false;

}

int kruskals(int \*\*graph, int n)

{

vector<pair<int, pair<int, int>>> G;

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

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

if (graph[i][j] != 0)

G.push\_back(make\_pair(graph[i][j], make\_pair(i, j)));

sort(G.begin(), G.end(), greater<pair<int, pair<int, int>>>());

vector<int> parent(n, NIL);

int s = 0;

for (auto i : G)

{

int u = i.second.first;

int v = i.second.second;

int w = i.first;

if (UnionByWeight(parent, u, v))

s += w;

}

return s;

}

int main()

{

int n;

cin >> n;

int \*\*graph;

graph = (int \*\*)malloc(n \* sizeof(int \*));

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

graph[i] = (int \*)malloc(n \* sizeof(int));

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

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

cin >> graph[i][j];

cout << "Minimum spanning weight : " << kruskals(graph, n) << endl;

return 0;

}

**Output:**

**4**

**2 5 4 3**

**1 0 9 8**

**3 6 6 0**

**1 1 2 3**

**Minimum spanning weight : 22**

**Week 9**

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

**Program:**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n, i, j, k, w;

cin >> n;

int graph[n][n];

string temp;

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

for (j = 0; j < n; j++) {

cin >> temp;

if (temp != "INF") {

graph[i][j] = stoi(temp);

} else {

graph[i][j] = 1e8; } } }

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

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

for (j = 0; j < n; j++) {

if (graph[i][k] + graph[k][j] < graph[i][j]) {

graph[i][j] = graph[i][k] + graph[k][j]; } } } }

cout << "The shortest path matrix: " << endl;

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

for (j = 0; j < n; j++) {

if(graph[i][j] >= 1e8) cout << "INF";

else cout << graph[i][j];

cout << " ";

}

cout << endl;

}

return 0;

}

**Output:**

**5**

**0 1 5 5 INF**

**INF 0 5 5 5**

**INF INF 0 INF 10**

**INF INF INF 0 20**

**INF INF INF 5 0**

**The shortest path matrix:**

**0 1 5 5 6**

**INF 0 5 5 5**

**INF INF 0 15 10**

**INF INF INF 0 20**

**INF INF INF 5 0**

**II. 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 only a fraction x; of item i, where 0 ≤x≤ 1.**

**Program:**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin >> n;

vector<double> items(n);

vector<double> val(n);

vector<vector<double>> job;

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

{

cin >> items[i];

}

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

{

cin >> val[i];

job.push\_back({val[i] / items[i], items[i], (double)(i + 1)});

}

double k;

cin >> k;

sort(job.rbegin(), job.rend());

vector<pair<double, double>> ls;

float profit = 0;

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

{

if (job[i][1] >= k)

{

profit += k \* job[i][0];

ls.push\_back(make\_pair(k, job[i][2]));

break;

}

else

{

profit += job[i][1] \* job[i][0];

}

ls.push\_back(make\_pair(job[i][1], job[i][2]));

k = k - job[i][1];

}

cout << "Maximum Value : " << profit << endl;

cout << "Item - Weight" << endl;

for (auto it : ls)

cout << it.second << " - " << it.first << endl;

return 0;

}

**Output:**

**6**

**6 10 3 5 1 3**

**6 2 1 8 3 5**

**16**

**Maximum Value : 22.3333**

**Item - Weight**

**5 - 1**

**6 - 3**

**4 - 5**

**1 - 6**

**3 - 1**

**III. 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)**

**Program:**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin >> n;

vector<int> a(n);

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

{

cin >> a[i];

}

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

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

minheap.push(a[i]);

}

int ans = 0;

while (minheap.size() >1)

{

int e1 = minheap.top();

minheap.pop();

int e2 = minheap.top();

minheap.pop();

ans += e1 + e2;

minheap.push(e1 + e2);

}

cout << ans;

return 0;

}

**Output:**

**10**

**10 5 100 50 20 15 5 20 100 10**

**895**

**Week 10**

**I. 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.**

**Program:**

#include<bits/stdc++.h>

using namespace std;

int main() {

int n;

cin>>n;

int i,s[n],f[n];

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

cin>>s[i];

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

cin>>f[i];

vector<vector<int>> a;

vector<int> act;

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

a.push\_back({f[i],s[i],i+1});

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

int e=INT\_MIN,c=0;

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

{

if(a[i][1]>=e)

{

e=a[i][0];

c++;

act.push\_back(a[i][2]);

}

}

cout<<"No. of non-conflicting activities : "<<c<<endl;

cout<<"List of selected activities : ";

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

cout<<act[i]<<",";

return 0;

}

**Output:**

**10**

**1 3 0 5 3 5 8 8 2 12**

**4 5 6 7 9 9 11 12 14 16**

**No. of non-conflicting activities : 4**

**List of selected activities : 1,4,7,10,**

**II. 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**.

**Program :**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin>>n;

int i,t[n],f[n];

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

cin>>t[i];

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

cin>>f[i];

vector<vector<int>> a;

vector<int> act;

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

a.push\_back({f[i],f[i]-t[i],i+1});

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

int e=INT\_MIN,c=0;

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

{

if(a[i][1]>=e)

{

e=a[i][0];

c++;

act.push\_back(a[i][2]);

}

}

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

cout<<"Max number of tasks : "<<c<<endl;

cout<<"Selected task Numbers : ";

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

cout<<act[i]<<",";

return 0;

}

**Output:**

**7**

**2 1 3 2 2 2 1**

**2 3 8 6 2 5 3**

**Max number of tasks : 4**

**Selected task Numbers : 1,2,3,6,**

**III. 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.**

**Program:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin>>n;

int i,a[n],c,j;

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

cin>>a[i];

bool f=0;

sort(a,a+n);

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

{

c=1;

j=i+1;

while(j<n && a[j++]==a[i])

c++;

if(c>n/2)

{

cout<<"yes\n";

f=1;

break;

}

i=j-1;

}

if(f==0)

cout<<"no\n";

if(n%2!=0)

cout<<a[n/2];

else

cout<<((float)a[n/2]+a[n/2-1])/2;

return 0;

}

**Output:**

**9**

**4 4 2 3 2 2 3 2 2**

**yes**

**2**

**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.**

**Program:**

#include<bits/stdc++.h>

using namespace std;

long matChainOrder(int \*p,int n) {

int m[n][n];

int i,j,k,l,q;

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

m[i][i]=0;

for(l=2;l<n;l++)

{

for(i=1;i<n-l+1;i++)

{

j=i+l-1;

m[i][j]=INT\_MAX;

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

{

q=m[i][k]+m[k+1][j]+p[i-1]\*p[k]\*p[j];

if(q<m[i][j])

m[i][j]=q;

}

}

}

return m[1][n-1];

}

int main()

{

int n;

cin>>n;

int p[n+1];

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

{

cin>>p[i]>>p[i+1];

}

cout<<matChainOrder(p,n+1);

return 0;

}

**Output:**

**3**

**10 30**

**30 5**

**5 60**

**4500**

**II. 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.**

**Program:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n,amt;

cin>>n;

int i,j,a[n];

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

cin>>a[i];

cin>>amt;

int ans[amt+1];

for(i=1;i<=amt;i++)

ans[i]=0;

ans[0]=1;

for(j=0;j<n;j++)

{

for(i=1;i<=amt;i++)

{

if(a[j]<=i)

ans[i]+=(ans[i-a[j]]);

}

}

cout<<ans[amt];

return 0;

}

**Output:**

**4**

**2 5 6 3**

**10**

**5**

**III. 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.**

**Program:**

#include<bits/stdc++.h>

using namespace std;

int main() {

int n;

cin>>n;

int i,j,a[n];

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

cin>>a[i];

int sum=0;

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

sum+=a[i];

if(sum%2!=0)

{

cout<<"no";

return 0;

}

sum=sum/2;

bool s[n+1][sum+1];

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

{

for(j=0;j<=sum;j++)

{

if(j==0)

s[i][j]=1;

else if(i==0)

s[i][j]=0;

else

{

if(a[i-1]>j)

s[i][j]=s[i-1][j];

else

s[i][j]=(s[i-1][j] || s[i-1][j-a[i-1]]);

}

}

}

if(s[n][sum])

cout<<"yes";

else

cout<<"no";

return 0;

}

**Output:**

**7**

**1 5 4 11 5 14 10**

**yes**