|  |  |
| --- | --- |
|  | |
| Assignment No: 05     1. A program of “Dijkstra’s Algorithm” 2. A program of “Bellman Ford’s Algorithm” | |
| Date of Performance: 04/08/2019  Date of Submission: 25/08/2019 | Student ID: 17-02-04-058  Group: B1 |

**No.1:**

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

**using namespace std;**

**#define max\_node 100**

**struct Struct**

**{**

**int s\_time;**

**int f\_time;**

**int color;**

**int parent;**

**};**

**Struct node[max\_node];**

**priority\_queue <int, vector<int>, greater<int> > Q;**

**int Time;**

**int v, e;**

**void add\_edge(list<pair<int, int> > adj\_list[], int m, int n, int wt){**

**adj\_list[m].push\_back(make\_pair(n, wt));**

**}**

**void dfs\_visit(list<pair<int, int> > adj\_list[], int s){**

**node[s].color = 1;**

**Time++;**

**node[s].s\_time = Time;**

**list<pair<int, int> >::iterator it;**

**for (it = adj\_list[s].begin(); it != adj\_list[s].end(); it++)**

**if (node[it->first].color == 0){**

**node[it->first].parent = s;**

**dfs\_visit(adj\_list, it->first);**

**}**

**node[s].color = 2;**

**Time++;**

**node[s].f\_time = Time;**

**Q.push(s);**

**}**

**void dfs(list<pair<int, int> > adj\_list[], int s){**

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

**node[i].color = 0;**

**dfs\_visit(adj\_list, s);**

**for(int i=0; i<v; i++){**

**if(node[i].color == 0){**

**dfs\_visit(adj\_list, i);**

**}**

**}**

**}**

**void dijkstra(list<pair<int, int> > adj\_list[], int s){**

**int dist[v];**

**int parent[v];**

**for(int i=0; i<v; i++){**

**dist[i] = INT\_MAX;**

**}**

**dist[s] = 0;**

**stack<int> St;**

**while(Q.empty() != true){**

**int u = Q.top();**

**Q.pop();**

**list<pair<int, int> >::iterator itr;**

**if(dist[u] != INT\_MAX){**

**for(itr = adj\_list[u].begin(); itr != adj\_list[u].end(); itr++){**

**if(dist[itr->first] > dist[u] + itr->second){**

**dist[itr->first] = dist[u] + itr->second;**

**}**

**}**

**}**

**}**

**for (int i = 0; i < v; i++){**

**if(dist[i] == INT\_MAX)**

**cout << "INF ";**

**else**

**cout << dist[i] << " ";**

**}**

**}**

**int main()**

**{**

**cout<<"Enter number of vertices/nodes: ";**

**cin>>v;**

**cout<<"Enter number of edges: ";**

**cin>>e;**

**cout<<"Enter the connected pairs and their distance(first source then destination, then weight): "<<endl;**

**list<pair<int, int> > adj\_list[v];**

**for(int i=0; i<e; i++){**

**int m,n,wt;**

**cin>>m>>n>>wt;**

**add\_edge(adj\_list, m, n, wt);**

**}**

**cout<<"Enter node to start traversing: ";**

**int s;**

**cin>>s;**

**dfs(adj\_list, s);**

**dijkstra(adj\_list, s);**

**return 0;**

**}**

**No.2:**

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

**using namespace std;**

**#define max\_edge 100**

**struct Edge**

**{**

**int source;**

**int destination;**

**int weight;**

**};**

**Edge E[max\_edge];**

**int v, e;**

**void bellman\_ford(int s){**

**int dist[v];**

**int parent[v];**

**for(int i=0; i<v; i++){**

**dist[i] = INT\_MAX;**

**parent[i] = -1;**

**}**

**dist[s] = 0;**

**for(int i=1; i<v-1; i++){**

**for(int j=0; j<e; j++){**

**int u = E[j].source;**

**int v = E[j].destination;**

**int weight = E[j].weight;**

**if(dist[u] + weight < dist[v]){**

**dist[v] = dist[u] + weight;**

**}**

**}**

**for(int j=0; j<e; j++){**

**int u = E[j].source;**

**int v = E[j].destination;**

**int weight = E[j].weight;**

**if(dist[u] + weight < dist[v]){**

**cout<<"Negative weighted cycle exists! :o ";**

**return;**

**}**

**}**

**}**

**for (int i = 0; i < v; i++){**

**if(dist[i] == INT\_MAX)**

**cout << "INF ";**

**else**

**cout << dist[i] << " ";**

**}**

**}**

**int main()**

**{**

**cout<<"Enter number of vertices/nodes: ";**

**cin>>v;**

**cout<<"Enter number of edges: ";**

**cin>>e;**

**cout<<"Enter the connected pairs and their distance(first source then destination, then weight): "<<endl;**

**for(int i=0; i<e; i++){**

**int m,n,wt;**

**cin>>m>>n>>wt;**

**E[i].source = m;**

**E[i].destination = n;**

**E[i].weight = wt;**

**}**

**cout<<"Enter node to start traversing: ";**

**int s;**

**cin>>s;**

**bellman\_ford(s);**

**return 0; }**