

**Assignment #2**

1. **Johnson's algorithm for All-pairs shortest paths**



**Date of Performance: 13-05-2024 Student ID: 20220104147**

**Date of Submission: 20-05-2024 Name: Md. Redwan Hossen**

**Group: C2**

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

**using namespace std;**

**#define INF 1e9**

**#define NIL -1**

**int node, edge;**

**int adj[100][100];**

**void createGPrime(int source)**

**{**

**for (int u = 0; u < node; u++)**

**{**

**adj[node][u] = 0;**

**}**

**}**

**bool BellmanFord(int source, int h[], int pre\_node[])**

**{**

**int dist[node];**

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

**{**

**dist[i] = INF;**

**}**

**dist[source] = 0;**

**int v = 0;**

**int u = 0;**

**int weight = adj[u][v];**

**for (int i = 0; i < node - 1; i++)**

**{**

**for ( u = 0; u < node; u++)**

**{**

**for ( v = 0; v < node; v++)**

**{**

**if (weight != 0 && dist[u] != INF && dist[v] > dist[u] + weight)**

**{**

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

**pre\_node[v] = u;**

**}**

**}**

**}**

**}**

**if (weight != 0 && dist[u] != INF && dist[v] > dist[u] + weight)**

**{**

**return false;**

**}**

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

**{**

**h[i] = dist[i];**

**}**

**return true;**

**}**

**void Dijkstra(int source, int weightPrime[], int dist[][100], int pre\_node[])**

**{**

**pair<int, int> pq[node \* node];**

**int pq\_size = 0;**

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

**{**

**dist[source][i] = INF;**

**}**

**dist[source][source] = 0;**

**pq[pq\_size++] = make\_pair(0, source);**

**while (pq\_size > 0)**

**{**

**int min\_dist = INF, min\_idx = -1;**

**for (int i = 0; i < pq\_size; i++)**

**{**

**if (pq[i].first < min\_dist)**

**{**

**min\_dist = pq[i].first;**

**min\_idx = i;**

**}**

**}**

**int u = pq[min\_idx].second;**

**swap(pq[min\_idx], pq[--pq\_size]);**

**if (min\_dist > dist[source][u]) continue;**

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

**{**

**int weight = adj[u][v] + weightPrime[u] - weightPrime[v];**

**if (weight != 0 && dist[source][v] > dist[source][u] + weight)**

**{**

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

**pq[pq\_size++] = make\_pair(dist[source][v], v);**

**pre\_node[v] = u;**

**}**

**}**

**}**

**}**

**void PrintPath(int source, int target, int pre\_node[])**

**{**

**if (source == target)**

**{**

**cout << source;**

**}**

**else if (pre\_node[target] == NIL)**

**{**

**cout << "No path from " << source << " to " << target;**

**}**

**else**

**{**

**PrintPath(source, pre\_node[target], pre\_node);**

**cout << " -> " << target;**

**}**

**}**

**void Johnson(int D[][100])**

**{**

**createGPrime(node);**

**int h[node], pre\_node[node];**

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

**{**

**h[i] = 0;**

**pre\_node[i] = NIL;**

**}**

**if (!BellmanFord(node, h, pre\_node))**

**{**

**cout << "The input graph has a negative weight cycle" << endl;**

**return;**

**}**

**for (int u = 0; u < node; u++)**

**{**

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

**{**

**int weight = adj[u][v];**

**if (weight != 0)**

**{**

**adj[u][v] = weight + h[u] - h[v];**

**}**

**}**

**}**

**cout << "Shortest paths between all pairs of nodes:" << endl;**

**for (int u = 0; u < node; u++)**

**{**

**Dijkstra(u, h, D, pre\_node);**

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

**{**

**if (D[u][v] == INF)**

**{**

**cout << u << "->" << v << ": No path" << endl;**

**}**

**else**

**{**

**cout << u << "->" << v << ": Cost = " << D[u][v] << ", Path: ";**

**PrintPath(u, v, pre\_node);**

**cout << endl;**

**}**

**}**

**cout << endl;**

**}**

**}**

**int main()**

**{**

**//freopen("johnson algo.txt", "r", stdin);**

**cout << "Enter the number of nodes & edges: ";**

**cin >> node >> edge;**

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

**{**

**for (int j = 0; j < node; j++)**

**{**

**adj[i][j] = 0;**

**}**

**}**

**cout << "Enter edges with weights :" << endl;**

**int from, to, weight;**

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

**{**

**cin >> from >> to >> weight;**

**adj[from][to] = weight;**

**}**

**int shortestPaths[100][100];**

**Johnson(shortestPaths);**

**return 0;**

**}**

**/\***

**Sample Input & Output**

**Enter the number of nodes & edges: 5 10**

**Enter edges with weights :**

**0 1 6**

**0 3 7**

**1 2 5**

**1 3 8**

**1 4 -4**

**2 1 -2**

**3 2 -3**

**3 4 9**

**4 2 7**

**4 0 2**

**Shortest paths between all pairs of nodes:**

**0->0: Cost = 0, Path: 0**

**0->1: Cost = 2, Path: 0 -> 3 -> 2 -> 1**

**0->2: Cost = 4, Path: 0 -> 3 -> 2**

**0->3: Cost = 7, Path: 0 -> 3**

**0->4: Cost = -2, Path: 0 -> 3 -> 2 -> 1 -> 4**

**1->0: Cost = -2, Path: 1 -> 4 -> 0**

**1->1: Cost = 0, Path: 1**

**1->2: Cost = 2, Path: 1 -> 4 -> 0 -> 3 -> 2**

**1->3: Cost = 5, Path: 1 -> 4 -> 0 -> 3**

**1->4: Cost = -4, Path: 1 -> 4**

**2->0: Cost = -4, Path: 2 -> 1 -> 4 -> 0**

**2->1: Cost = -2, Path: 2 -> 1**

**2->2: Cost = 0, Path: 2**

**2->3: Cost = 3, Path: 2 -> 1 -> 4 -> 0 -> 3**

**2->4: Cost = -6, Path: 2 -> 1 -> 4**

**3->0: Cost = -7, Path: 3 -> 2 -> 1 -> 4 -> 0**

**3->1: Cost = -5, Path: 3 -> 2 -> 1**

**3->2: Cost = -3, Path: 3 -> 2**

**3->3: Cost = 0, Path: 3**

**3->4: Cost = -9, Path: 3 -> 2 -> 1 -> 4**

**4->0: Cost = 2, Path: 4 -> 0**

**4->1: Cost = 4, Path: 4 -> 0 -> 3 -> 2 -> 1**

**4->2: Cost = 6, Path: 4 -> 0 -> 3 -> 2**

**4->3: Cost = 9, Path: 4 -> 0 -> 3**

**4->4: Cost = 0, Path: 4**

**\*/**