**Worksheet 2.1**

**Student Name:** Deepak Saini **UID:** 20BCS4066

**Branch:** 20BCC-1 **Section/Group:** A

**Semester:** 5th **Date of Performance:** 06-10-2022

# Subject Name: Advance Programming Lab Subject Code: 20CSP-334

**1. Aim/Overview of the practical:**

a) From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

b) Compute the transitive closure of a given directed graph using Warshall's algorithm.

**2. Task to be done:**

a) From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

b) Compute the transitive closure of a given directed graph using Warshall's algorithm.

**3. Algorithm/Flowchart (For programming-based labs):**

**a) Dijkstra’s algorithm**

* Create a set **sptSet** (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
* Assign a distance value to all vertices in the input graph. Initialize all distance values as **INFINITE**. Assign the distance value as 0 for the source vertex so that it is picked first.
* While **sptSet** doesn’t include all vertices
  + Pick a vertex **u** which is not there in **sptSet**and has a minimum distance value.
  + Include u to **sptSet**.
  + Then update distance value of all adjacent vertices of u.
    - To update the distance values, iterate through all adjacent vertices.
    - For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

**b) Floyd Warshall:**

* For the first step, the solution matrix is initialized with the input adjacent matrix of the graph. Let’s name it as reach.
* Next we need to iterate over the number of nodes from {0,1,.....n} one by one by considering them strating vertex. Similarly, another iteration is performed over the nodes {1,2,....,n} by considering ending vertex one by one.
* For the shortest path, we need to form another iteration which ranges from {1,2,...,k-1}, where vertex k has been picked up as an intermediate vertex.
* For every pair (i, j) of the starting and ending vertices respectively, there are two possible cases.
* if k is an intermediate vertex in the shortest path from i to j, then we check the condition reach[i][j] > reach[i][k] + reach[k][j] and update reach[i][j] accordingly.
* Otherwise, if k is not an intermediate vertex, we don't update anything and continue the loop.

**Transitive Closure condition:**

Only one difference of the condition to be checked when there is an intermediate vertex k exits between the starting vertex and the ending vertex. We need to check two conditions and check if any of them is true,

* Is there a direct edge between the starting vertex and the ending vertex? If yes, then update the transitive closure matrix value as 1.
* For k, any intermediate vertex, is there any edge between the (starting vertex & k) and (k & ending vertex) ? If yes,then update the transitive closure matrix value as 1.

**4. CODE:**

**a) Dijkstra**

// Dijkstra's single source shortest path using adjacency matrix representation of the graph

#include <bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 9

// function to find the vertex with minimum

// distance value, from the set of vertices not yet included

// in shortest path tree

int minDistance(int dist[], bool sptSet[])

{

// Initialize min value

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

void printSolution(int dist[])

{

cout << "Vertex \t\t Distance from Source" << endl;

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

cout << i << " \t\t\t" << dist[i] << endl;

}

void dijkstra(int graph[V][V], int src)

{

int dist[V]; // The output array. dist[i] will hold the shortest

// distance from src to i

bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest path tree

// or shortest distance from src to i is finalized

// Initialize all distances as INFINITE and stpSet[] as false

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

dist[i] = INT\_MAX, sptSet[i] = false;

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of vertices not yet processed.

//u is always equal to src in the first iteration.

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = true;

// Update dist value of the adjacent vertices of the picked vertex.

for (int v = 0; v < V; v++)

// Update dist[v] only if is not in sptSet,

// there is an edge from u to v, and total

// weight of path from src to v through u is

// smaller than current value of dist[v]

if (!sptSet[v] && graph[u][v]

&& dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

// print the constructed distance array

printSolution(dist);

}

int main()

{

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

dijkstra(graph, 0);

return 0;

}

**b) Transitive Closure using Warshall**

#include<bits/stdc++.h>

using namespace std;

#define V 4

void printSolution(int reach[][V])

{

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

{

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

{

if(i == j)

printf("1 ");

else

printf ("%d ", reach[i][j]);

}

cout<<"\n";

}

}

void transitiveClosure(int graph[][V])

{

int reach[V][V], i, j, k;

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

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

reach[i][j] = graph[i][j];

for (k = 0; k < V; k++)

{

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

{

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

{

reach[i][j] = reach[i][j] ||

(reach[i][k] && reach[k][j]);

}

}

}

cout<<"Following matrix is transitive closure of the given graph\n";

printSolution(reach);

}

int main()

{

int graph[V][V] = { {1, 1, 0, 1},

{0, 1, 1, 0},

{0, 0, 1, 1},

{0, 0, 0, 1}

};

cout<<"Given graph : \n";

printSolution(graph);

transitiveClosure(graph);

return 0;

/\*

10

(0)------->(3)

| /|\

5| |

| | 1

\|/ |

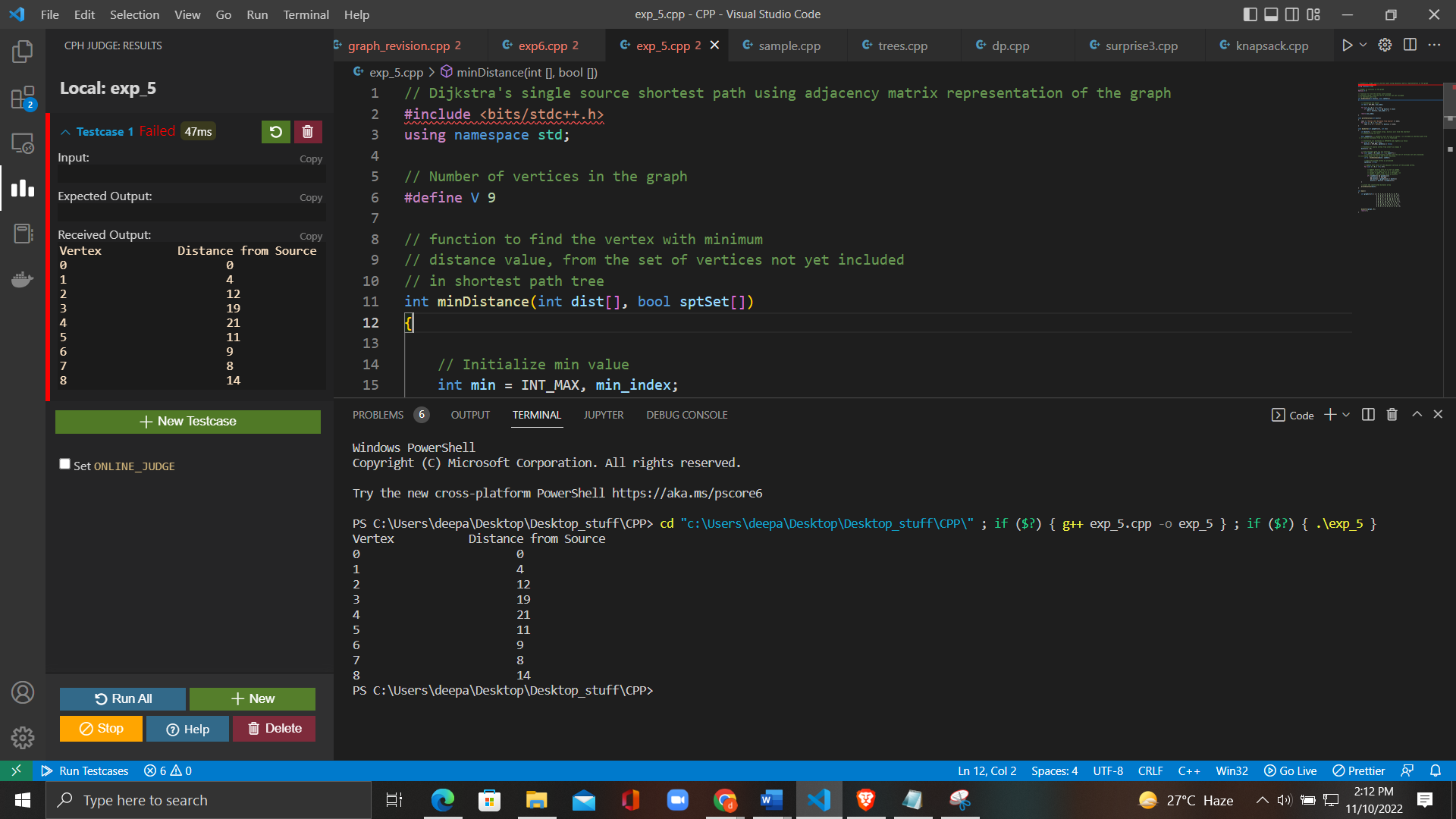
(1)-------> (2)

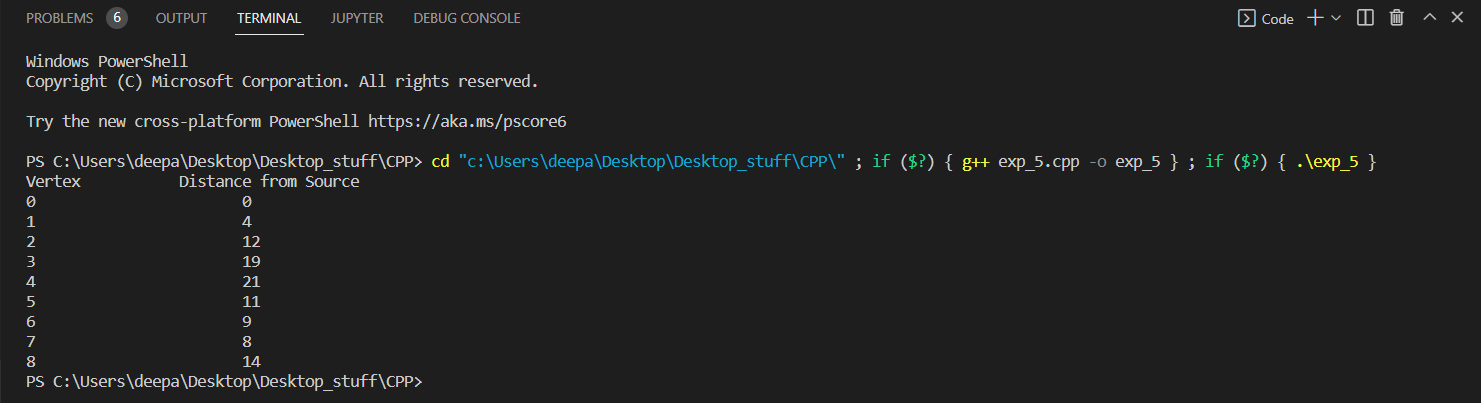
3 \*/

}

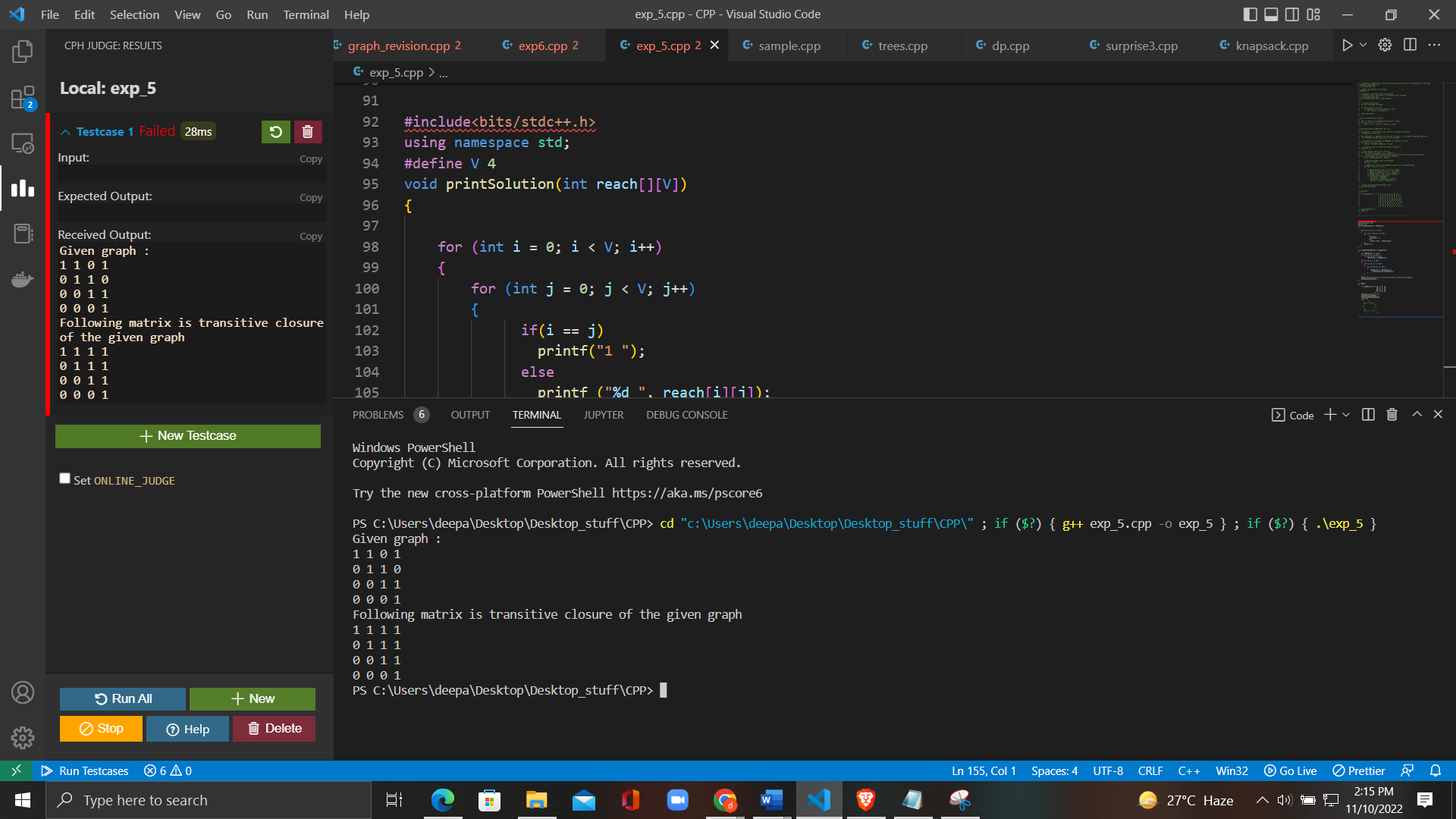
**5. Result/Output:**

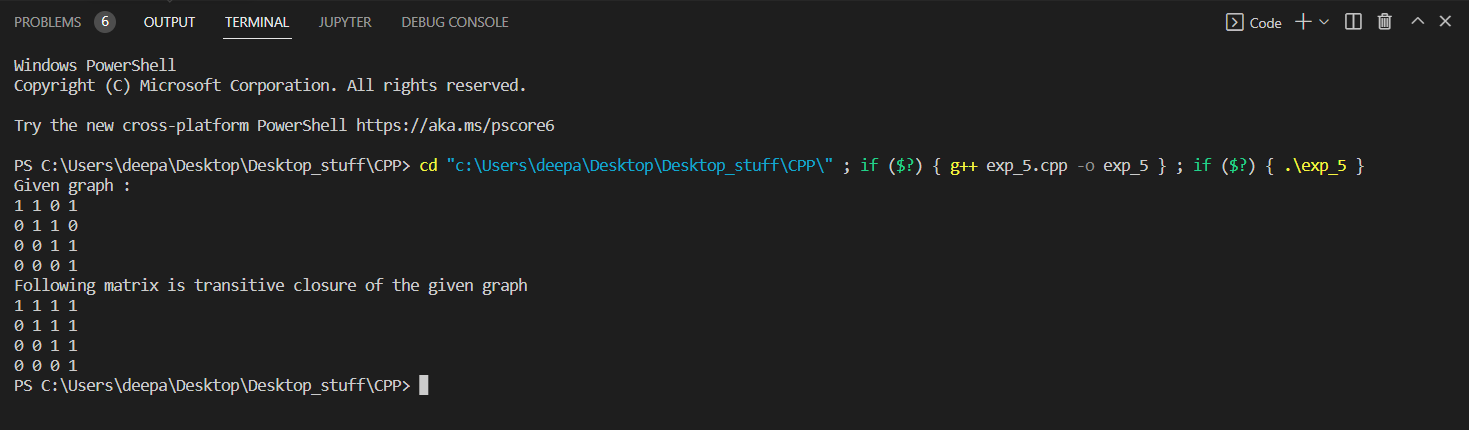
**a)** **Dijkstra**





**b) Transitive Closure using Warshall:**





**Learning Outcomes:**

* Learn shortest paths to other vertices using Dijkstra’s algorithm.
* Learn transitive closure of a given directed graph using Warshall's algorithm.

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
|  |  |  |  |