PRACTICAL – 10

AIMAN SIDDIQUA 2K18/MC/008

AIM: To write a program to find the maximum flow from source node to sink node using Ford-Fulkerson Algorithm.

CODE:

#include<bits/stdc++.h>

**using** **namespace** std;

#define V 6

**bool** **bfs**(**int** rGraph[V][V], **int** s, **int** t, **int** parent[])

{

**bool** visited[V];

memset(visited, **0**, **sizeof**(visited));

queue<**int**> q;

q.push(s);

visited[s] = true;

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

**while** (!q.empty()) {

**int** u = q.front();

q.pop();

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

**if** (visited[v] == false && rGraph[u][v] > **0**) {

**if** (v == t) {

parent[v] = u;

**return** true;

}

q.push(v);

parent[v] = u;

visited[v] = true;

}

}

}

**return** false;

}

**int** **fordFulkerson**(**int** graph[V][V], **int** s, **int** t)

{

**int** u, v;

**int** rGraph[V][V];

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

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

rGraph[u][v] = graph[u][v];

**int** parent[V];

**int** max\_flow = **0**;

**while** (bfs(rGraph, s, t, parent)) {

**int** path\_flow = INT\_MAX;

**for** (v = t; v != s; v = parent[v]) {

u = parent[v];

path\_flow = min(path\_flow, rGraph[u][v]);

}

**for** (v = t; v != s; v = parent[v]) {

u = parent[v];

rGraph[u][v] -= path\_flow;

rGraph[v][u] += path\_flow;

}

max\_flow += path\_flow;

}

**return** max\_flow;

}

**int** **main**()

{

**int** graph[V][V]

= { { **0**, **16**, **13**, **0**, **0**, **0** }, { **0**, **0**, **10**, **12**, **0**, **0** },

{ **0**, **4**, **0**, **0**, **14**, **0** }, { **0**, **0**, **9**, **0**, **0**, **20** },

{ **0**, **0**, **0**, **7**, **0**, **4** }, { **0**, **0**, **0**, **0**, **0**, **0** } };

cout << "The maximum possible flow from source to sink is "

<< fordFulkerson(graph, **0**, **5**);

**return** **0**;

}

OUTPUT:

