CIS 522: Assignment 6

Q1.) Ford Fulkerson Implementation

Ans.)

a.) **Pseudocode**:

```
Class FordFulkerson{
                    static final int V = 8;

private String[] Nodes;

public FordFulkerson(String[] Nodes){
        this.Nodes=Nodes; //Constructor to initialize the nodes for varying arguments
}

public boolean breadthFirstSearch(int residualGraph[][], int vS, int vT, int p[]) {
        boolean visited[] = new boolean[V];
        for (int i = 0; i < V; i++) {
                visited[i] = false;
        }

        // Create a queue, enqueue source vertex and mark
// source vertex as visited

        LinkedList<Integer> queue = new LinkedList<Integer>();
        queue.add(vS);
        visited[vS] = true;
        p[vS]=-1;          //mark source vertex as visited

        //Conduct BFS to check which nodes have been visited
```

```
        while (!queue.isEmpty()) {
                int u = queue.remove();
                for (int v=0; v<V; v++) {
                        if (visited[v]==false && residualGraph[u][v] > 0) {
                                queue.add(v);
                                p[v] = u;
                                visited[v] = true;
                        }
                }
        }
        return visited[vT];
}

public int maxFlow(int graph[][], int vS, int vT) {
        int maxFlow = 0; //Initially there is no flow
        int path[] = new int[V]; //Store the paths post conducting the BFS
        int x = 0;
        int y = 0;

        int residualGraph[][] = new int[V][V]; //Creates the residual graph
        for (x = 0; x < V; x++){
                for (y = 0; y < V; y++){
                        residualGraph[x][y] = graph[x][y];
                }
        }

        while (breadthFirstSearch(residualGraph, vS, vT, path)) {
                String pathString = "";    //Use this to print out the augmented path
```

```
                int pathFlow = Integer.MAX_VALUE;
                for (y=vT; y != vS; y=path[y]) {
                        x = path[y];
                        pathFlow = Math.min(pathFlow, residualGraph[x][y]);

                        pathString = " -> "+ Nodes[y] + pathString;
                }
                pathString= "S"+pathString;       //add S to show the flow out of source
                System.out.println("Residual graph.. \n"+ pathString);
                System.out.println("Flow added = " + pathFlow + "\n");

                // update residual capacities of the edges and
                // reverse edges along the path

                for (y=vT; y != vS; y=path[y]) {
                        x = path[y];
                        residualGraph[x][y] -= pathFlow;
                        residualGraph[y][x] += pathFlow;
                }

                maxFlow += pathFlow;
        }

        return maxFlow;
}
```

b.) **Time Complexity** -> Given that all the capacities are integers, then by Lemma 7.5, the running time of the above algorithm is O(mC), where 'm' is the number of edges and 'C' is the capacity of each edge.

c.) **Code Output** ->

```
Residual graph..
S -> 2 -> 4 -> T
Flow added = 7

Residual graph..
S -> 2 -> 5 -> T
Flow added = 8

Residual graph..
S -> 3 -> 5 -> T
Flow added = 2

Residual graph..
S -> 6 -> 7 -> T
Flow added = 6

Residual graph..
S -> 3 -> 5 -> 4 -> T
Flow added = 6

Residual graph..
S -> 3 -> 5 -> 7 -> T
Flow added = 3

Residual graph..
S -> 6 -> 5 -> 7 -> T
Flow added = 4

 Max Flow = 36
```

Q3.)

Ans.)

a.) This problem can be modeled as a network flow problem because we're trying to match a set of doctors with a set of holidays/vacation periods. The problem is subject to the constraint that each doctor can work at most one day in each vacation period.

In general terms, each doctor 'i' has a set S_i of days when they can work, and each doctor should be scheduled for at most 'c' days total.

In our case, we'll have four doctors(labeled D) and three holiday periods(labeled P). Plus, we'll have to factor in the constraint wherein each doctor can only work for one day during the vacation/holiday period (labeled with H). So, the final graph will have 13 vertices to accommodate the doctors, holiday periods, and the constraint alongside the source and sink nodes.

b & c.) **Problem instance and code output** ->

```
<terminated> DoctorHoliday [Java Application] C:\Users\anubh\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bin\javaw.exe  (Apr 7, 2022, 11:09:19 AM – 11:09:20 AM)
[[1, 5, 3, 1, 3, 1, 2, 5, 3, 9, 0, 0, 3], [4, 6, 1, 6, 7, 8, 8, 9, 4, 1, 0, 0, 5], [3, 5, 8, 7, 6, 0, 5, 5, 1, 2, 9, 8, 0], [2, 1, 3, 4, 4, 0, 7, 9, 1, 0, 2, 4, 2],
S -> T
Flow added = 3

Residual graph..
S -> P1 -> T
Flow added = 5

Residual graph..
S -> P3 -> T
Flow added = 1

Residual graph..
S -> D1 -> T
Flow added = 3

Residual graph..
S -> D2 -> T
Flow added = 1

Residual graph..
S -> D3 -> T
Flow added = 1

Residual graph..
S -> D4 -> T
Flow added = 4

Residual graph..
S -> H1 -> T
Flow added = 3

Residual graph..
S -> H2 -> T
Flow added = 6

Residual graph..
S -> P2 -> P3 -> T
```

```
Residual graph..
S -> P2 -> P3 -> T
Flow added = 1

Residual graph..
S -> P2 -> D1 -> T
Flow added = 2

Residual graph..
S -> D3 -> D2 -> T
Flow added = 1

Residual graph..
S -> D4 -> D2 -> T
Flow added = 1

Residual graph..
S -> H2 -> D2 -> T
Flow added = 1

Residual graph..
S -> H2 -> H1 -> T
Flow added = 1

Residual graph..
S -> H2 -> H3 -> T
Flow added = 1

 Max Flow = 35
```

Q4.)

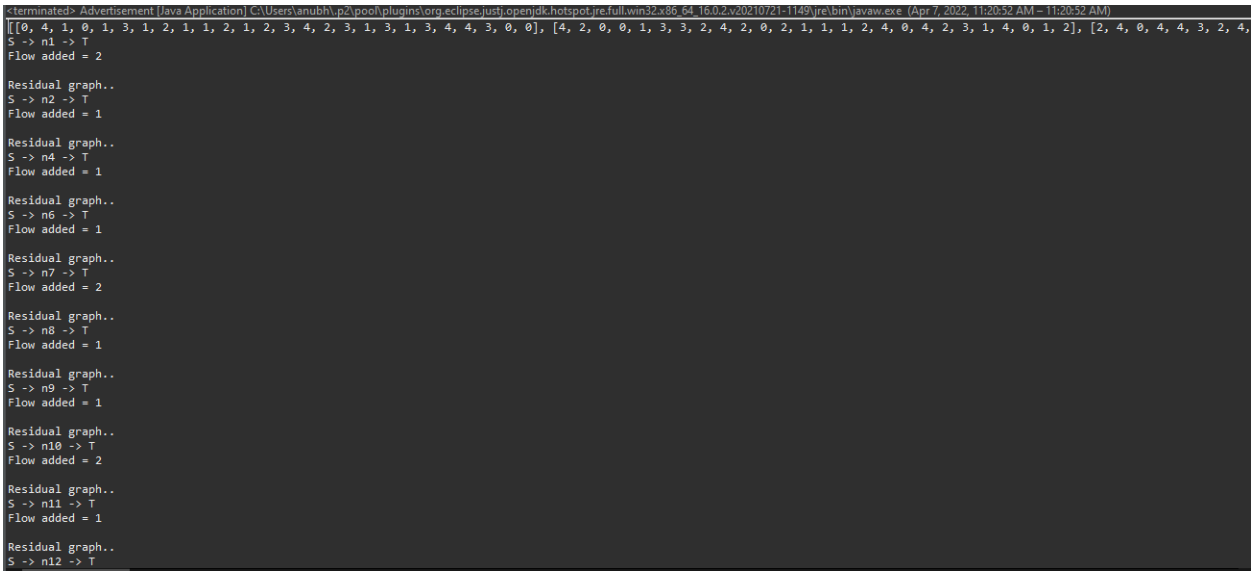
Ans.)

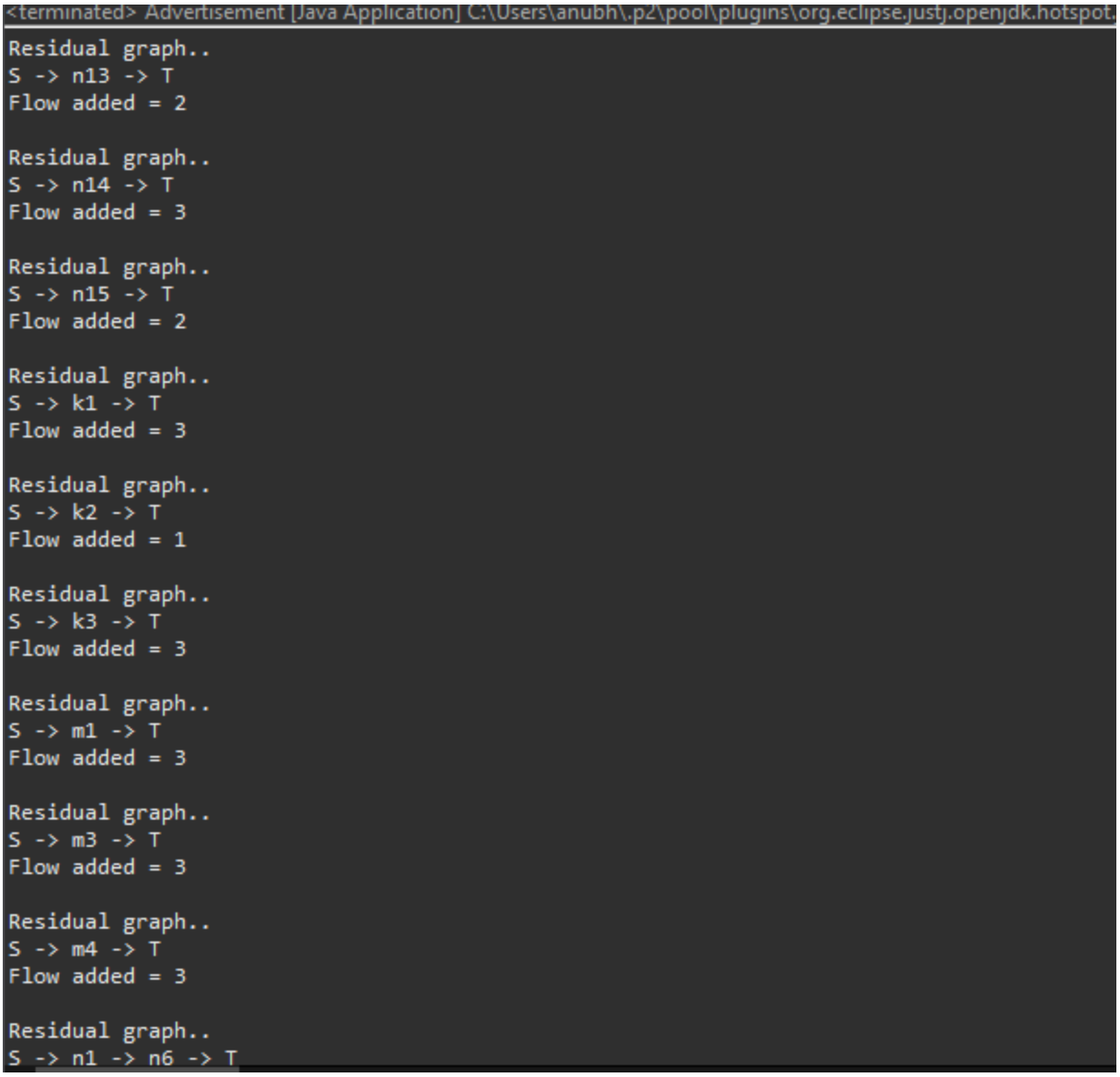
a.) This problem can be modeled as a network flow problem by examining how the contract with the 'ith' advertiser is set up. According to the question stem, the following contract is in place ->

* For a subset X~ _c [G1 ..... Gk} of the demographic groups, advertiser i wants its ads shown only to users who belong to at least one of the demographic groups in the set X~.
* For a number r~, advertiser ~ wants its ads shown to at least r~ users each minute.

In our case, we have 15 users (n) who are logging on to the site over the next five minutes. We have the registration details of 4 users (k), where k is a subset of n.

b & c.) **<u>Problem instance and code output</u>** ->

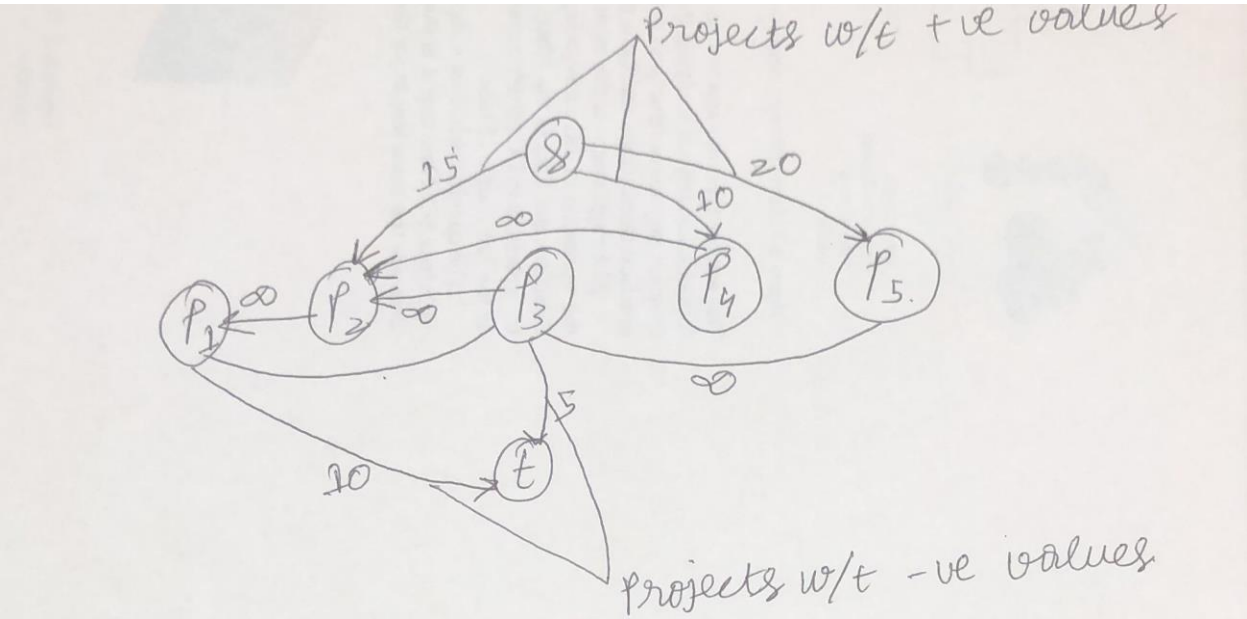



<terminated> Advertisement [Java Application] C:\Users\anubh\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.
Residual graph..
S -> n13 -> T
Flow added = 2
Residual graph..
S -> n14 -> T
Flow added = 3
Residual graph..
S -> n15 -> T
Flow added = 2
Residual graph..
S -> k1 -> T
Flow added = 3
Residual graph..
S -> k2 -> T
Flow added = 1
Residual graph..
S -> k3 -> T
Flow added = 3
Residual graph..
S -> m1 -> T
Flow added = 3
Residual graph..
S -> m3 -> T
Flow added = 3
Residual graph..
S -> m4 -> T
Flow added = 3
Residual graph..
S -> n1 -> n6 -> T

```
Residual graph..
S -> n5 -> n3 -> T
Flow added = 2

Residual graph..
S -> n5 -> n8 -> T
Flow added = 1

Residual graph..
S -> n13 -> n8 -> T
Flow added = 1

Residual graph..
S -> n14 -> n9 -> T
Flow added = 1

Residual graph..
S -> k4 -> n9 -> T
Flow added = 1

Residual graph..
S -> m2 -> n9 -> T
Flow added = 1

Residual graph..
S -> m2 -> n12 -> T
Flow added = 2

Residual graph..
S -> m2 -> n15 -> T
Flow added = 1

Residual graph..
S -> m3 -> n11 -> T
Flow added = 1


 Max Flow= 50
```

Q2.)

Ans.)



Min Cut = $p_1 + p_3 + 2p_2 = -10 - 5 + 2 * 15 = 15$

References: Following sources were consulted for the coding exercises in Q1, Q3, and Q4 ->

1.) **Brilliant** -> 1, 2
2.) **GFG**