

Theory Assignment-3

Roll No: 2021080 Section: A

Name: Pranjal Bharti Specialization: CSE

1) Answer to Question 1:

Formulation of network flow problem:

In the given problem, we must find out the least number of traffic blocks that need to be placed in order to ensure that the disease(originating in Tinkmoth) does not reach the capital city of Doweltown.

Before getting to the algorithm, we will have to convert this problem into a network flow problem. This can be accomplished using a directed graph wherein each town and village can be considered as a vertex and each railway line between a pair of towns/villages can be considered as an edge. Also, each edge will have a capacity of exactly 1 since it is known that only a single train can run on a railway line(edge) at a time.

Now, we must consider a pair of vertices as a source and sink. In this case, the vertex representing Tinkmoth will be the source while the vertex representing Doweltown will be the sink since the disease is originating from Tinkmoth while it must be prevented from reaching Doweltown, which is the destination.

This completes the first part of our question, which was, the formulation of a network flow problem using a directed graph.

Equivalence with maximum flow minimum cut value:

The next part of the question requires us to show how the application of maximum flow minimum cut theorem on this network flow problem will give us the correct answer. To prevent the disease from spreading to Doweltown, we need to find the minimum number of edges (railway tracks) that need to be removed or blocked in order to cut off the path between Tinkmoth and Doweltown. We can achieve this by finding the minimum cut in the flow network, which represents the set of edges whose removal will disconnect Tinkmoth from Doweltown. The capacity of the minimum cut is equal to the minimum number of edges that need to be blocked.

Since the capacity of each edge in the flow network is 1, only one train can pass through the track at a given point of time which is integral. Therefore, by maximizing the flow of

the network, we are trying to get the maximum number of trains that we can route through the network from the source (Tinkmoth) to the sink (Doweltown) using disjoint paths. The maximum flow value of the network is equal to the minimum number of railway tracks that need to be blocked to prevent the disease from reaching Doweltown. If we cut the edges belonging to the minimum cut(block off all the respective railway lines), we basically create divide the graph into two subparts, one containing the source and the other containing the sink. Since these two parts are entirely disconnected after blocking/cutting the edges, we ensure that the disease does not reach Doweltown and at the same time ensuring that the number of traffic blocks is the minimum.

Algorithm:

The algorithm for solving this problem is quite basic. In order to correctly solve this problem, we will use the max-flow min-cut theorem. First, we will find the maximum flow from Tinkmoth(source) to Doweltown(sink) using the Ford-Fulkerson algorithm. After this, we will find the minimum cut that separates Tinkmoth and Doweltown in the flow network. The edges that cross the minimum cut are the edges that need to be blocked.

The pseudo-code for the same is as follows:

Given a flow network (G, s, t) where G is directed graph, s is source, t is sink and a valid flow $f : E \rightarrow \mathbb{R} \geq 0$, the residual network with respect to flow f denoted as G_f is defined as follows:

- $G_f = (V, E_f)$ where $E_f = E \cup E_{rev}$
- $E_{rev} = \{(v, u) : f(u, v) > 0\}$, that is, E_{rev} contains the reverse of all edges which carry positive flow.

function fordFulkerson(G, s, t)

initialize $max_flow \leftarrow 0$

initialize $parent[]$ //to store the parents of each node in G

while s is not equal to t

// Run BFS to find augmenting path (bfs function is the same one which was already discussed in class)

bfs(G, s)

end while

// If no augmenting path is found, terminate

if no path found

```

        break
    end if
    // Find the max capacity along the augmenting path
    path_flow ← INT_MAX
    for every v in parent[v]
        int u ← parent[v]
        path_flow ← min(path_flow, G[u][v])
    end for

    // Update the residual capacities and flow along the augmenting path
    for every v in parent[v]
        u ← parent[v]
        G[u][v] ← path_flow
        G[v][u] ← G[v][u] + path_flow
    end for

    max_flow ← max_flow + path_flow

Return max_flow

```

Running time of the algorithm:

Since we are basically using the Ford-Fulkerson algorithm in order to calculate the maximum flow and minimum cut, the time complexity of the algorithm will be that of the Ford-Fulkerson algorithm itself. We are using a Breadth first search algorithm to find the augmenting path in the residual network for each iteration. This gives a time complexity of $O(EF)$ where E is the number of edges and F is the max-flow.

The algorithm will terminate when there are no longer any augmenting paths in the residual network. This gives us the final running time of the algorithm as:

$$O(|E| * \text{max-flow})$$
