Theory Assignment-3

Section:B

Roll No:2021482
Name:PARAS DHIMAN
Specialization:CSB

Roll No:2021304
Name:ADITYA DAIPURIA
Specialization:CSD

Ques1:

A. Formulation as a network flow problem:

We can represent the rail network of Sunland as a directed graph, where each station is represented as a vertex, and the railway tracks between stations are represented as directed edges with capacities. To cut down rail communication between Tinkmoth and Doweltown, we need to place traffic blocks (represented as edges with infinite capacities) between pairs of directly connected rail stations. The objective is to minimize the number of traffic blocks used.

Specifically, we can define the directed graph G = (V, E) as follows:

· V: Set of vertices representing the stations in Sunland, including Tinkmoth, Doweltown, and all other stations as V = {s1, s2, ..., sn}, where n is the total number of stations.

· E: the set of directed edges E = {(ui, vi) | ui, vi ∈ V, ui ≠ vi}, where each edge (ui, vi) represents a railway track from station ui to station vi. The capacities of the edges are defined as follows:

- If ui is Tinkmoth or Doweltown, the capacity of edge (ui, vi) is set to infinity, as there are no traffic blocks allowed at Tinkmoth or Doweltown. This ensures that trains can always travel from Tinkmoth or Doweltown to other stations, but not vice versa, preventing the spread of the contagious disease.
- If ui and vi are regular stations (not Tinkmoth or Doweltown), the capacity of edge (ui, vi) is set to 1, as only one traffic block is allowed to be placed between any pair of directly connected stations. This ensures that only one traffic block can be placed between any two stations, minimizing the expense and public notice

The objective of the problem is to find the minimum number of traffic blocks needed to completely cut down the rail communication between Tinkmoth and Doweltown. This can be formulated as a minimum cut problem in a network flow graph.

To solve this problem, we can use the Ford-Fulkerson algorithm, which is a well-known polynomial-time algorithm for finding the maximum flow in a network. The running time of the Ford-Fulkerson algorithm is O(|V||E|F), where |V| and |E| are the number of vertices and edges in the graph, respectively, and F is the maximum flow in the graph. In our case, the maximum flow is equal to the minimum number of traffic blocks needed to cut down the rail communication between Tinkmoth and Doweltown.

Once the Ford-Fulkerson algorithm finds the maximum flow, we can identify the minimum cut in the graph, which represents the traffic blocks that need to be placed to cut down the rail communication between Tinkmoth and Doweltown. The edges crossing the minimum cut represent the traffic blocks to be placed, and the capacity of the minimum cut represents the minimum number of traffic blocks needed.

B. how maximum-flow value in your network is equivalent to the solution to the original problem.

The maximum-flow value in the network G, obtained by applying the Ford-Fulkerson algorithm or any other algorithm for solving the maximum flow problem, is equivalent to the solution to the original problem of cutting down the rail communication between Tinkmoth and Doweltown.

The Ford-Fulkerson algorithm is a widely used algorithm for finding the maximum flow in a network. It iteratively finds augmenting paths in the residual graph, which is derived from the original network by subtracting the current flow from the capacities of the edges. The algorithm terminates when no more augmenting paths can be found, and the flow obtained at that point is the maximum flow in the network.

The flow obtained by the Ford-Fulkerson algorithm represents the maximum number of trains that can pass from Tinkmoth to Doweltown without encountering any traffic blocks. The capacity of the edges in the network G represents the number of traffic blocks that can be placed on each railway track. By finding the maximum flow in the network G, we ensure that the flow of trains from Tinkmoth to Doweltown is maximized, and no more trains can pass through without encountering traffic blocks.

The Max-Flow Min-Cut Theorem, as mentioned before, establishes that the maximum flow in a network is equal to the minimum capacity of a cut in the same network. In this problem, the cut with the minimum capacity in the network G represents the traffic blocks to be placed to cut down the rail communication between Tinkmoth and Doweltown. Therefore, the maximum-flow value obtained from the Ford-Fulkerson algorithm is equivalent to the solution to the original problem, which is to minimize the number of traffic blocks needed to cut down the rail communication between Tinkmoth and Doweltown, as proven by the Max-Flow Min-Cut Theorem.

ALGORITHM:

```
Sunland-Traffic-Block(G, s, t):
    Initialize the graph G with capacities and flows as
described above
    Initialize flow f to 0
    Set max_flow to 0

    while there exists an augmenting path p in G from s to t:
        Find augmenting path p using any standard graph
traversal algorithm (e.g., BFS or DFS)
        Compute bottleneck capacity c_min of path p

        Update the flow f along path p:
        for each edge (u, v) in p:
          if (u, v) is a forward edge in G:
               f[u][v] += c_min
          else if (v, u) is a backward edge in G:
               f[v][u] -= c_min

        Update the residual capacities in G:
        for each edge (u, v) in G:
          G[u][v].capacity -= c_min
          G[v][u].capacity += c_min

        Update the maximum flow:
        max_flow += c_min

    return max_flow
```

Proof:-

The Max-Flow Min-Cut Theorem states that in a flow network, the maximum value of a flow is equal to the minimum capacity of a cut.

Proof:

Let's consider a flow network G = (V, E) with capacities c(u, v) on each edge (u, v) in E, representing the capacity of the flow from vertex u to vertex v.

Let's define a cut (S, T) in the flow network G, where S is a subset of V and T is the complement of S in V (i.e., V - S).

The capacity of the cut (S, T) is defined as the sum of capacities of all edges crossing the cut, i.e., c(S, T) = Σc(u, v) for all u in S and v in T.

The flow in the network is defined as the sum of flows across all edges, i.e., f(u, v) for all (u, v) in E.

According to the flow conservation property, the flow into and out of a vertex must be conserved, except for the source and sink vertices. Mathematically, for all vertices u in V - {s, t}, the flow conservation property can be written as: Σf(u, v) - Σf(v, u) = 0, where v is a neighbor of u in G.

The capacity constraint states that the flow across an edge cannot exceed its capacity, i.e., f(u, v) <= c(u, v) for all (u, v) in E.

The goal is to find the maximum flow in the network G from the source s to the sink t.

Now, the theorem can be stated as follows: The maximum flow in the network G is equal to the minimum capacity of a cut (S, T), i.e., max-flow = min-cut.

Proof of Max-Flow Min-Cut Theorem:

a. Consider a flow network G with a maximum flow f and a cut (S, T) with capacity c(S, T).

b. By the flow conservation property, the flow into the cut (S, T) from S must equal the flow out of the cut (S, T) to T. Mathematically, Σf(u, v) for all u in S and v in T, must be equal to Σf(v, u) for all u in S and v in T.

c. The capacity constraint states that the flow across the cut (S, T) cannot exceed its capacity, i.e., Σf(u, v) for all u in S and v in T, must be less than or equal to c(S, T).

d. Therefore, the maximum flow f must be less than or equal to the capacity of the cut (S, T), i.e., f <= c(S, T).

e. Since this holds for any cut (S, T) in the network, it implies that the maximum flow in the network is less than or equal to the minimum capacity of a cut, i.e., max-flow <= min-cut.

Now, to prove the other direction, i.e., min-cut <= max-flow:

a. Assume that there exists a cut (S, T) in the network with capacity c(S, T) such that c(S, T) < f, where f is the maximum flow in the network.

b. By the flow conservation property, the flow into the cut (S, T) from S must equal the flow

<u>**C. Proper Explanation of runnig time Of Algorithm**</u>

The Ford-Fulkerson algorithm is an iterative algorithm that finds the maximum flow in a flow network. It starts with an initial flow of zero and repeatedly augments the flow along augmenting paths in the residual graph until no augmenting paths exist.

The running time of the Ford-Fulkerson algorithm can be analyzed in terms of the number of augmenting path computations, which is the dominant factor in the overall running time. Each augmenting path can be found using a depth-first search or a breadth-first search, which takes O(|E|) time, where |E| is the number of edges in the graph.

In the worst case, the algorithm may need to perform multiple augmenting path computations before reaching the maximum flow. However, it has been proven that the algorithm always terminates after a finite number of augmenting path computations, and the total number of augmenting paths is bounded by O(|V| * |E|), where |V| is the number of vertices in the graph.

The overall running time of the Ford-Fulkerson algorithm, when expressed as O(|V| * |E| * f), where f is the maximum flow value, indicates that the running time of the algorithm grows polynomially with the size of the input.

The term "f" in the time complexity expression O(|V| * |E| * f) represents the maximum flow value in the flow network. In theory, the Ford-Fulkerson algorithm could have a time complexity of O(|V| * |E| * f) if in the worst case the algorithm needs to perform O(f) augmenting path computations, and each augmenting path computation takes up to O(|E|) time. However, in practice, the maximum flow value "f" can be as large as the sum of capacities of all edges in the graph, and it is not a constant value. Therefore, the time complexity expression O(|V| * |E| * f) is not commonly used to analyze the running time of the Ford-Fulkerson algorithm in practice.

The commonly used and well-optimized implementations of the Ford-Fulkerson algorithm have a time complexity of O(|V| * |E|^2) with standard augmenting path finding techniques, and there are further optimizations, such as Dinic's algorithm, that can reduce the time complexity to O(|V|^2 * |E|) in certain cases.

In this case, as the flow is restricted to be 1 (i.e., f = 1), the running time can be further simplified to O(|V| * |E|), which means that the algorithm is efficient in most practical scenarios. The running time of O(|V| * |E|) indicates that the algorithm's efficiency is proportional to the number of vertices (|V|) and edges (|E|) in the input graph.

Reference Links:

1.) https://www.chegg.com/homework-help/questions-and-answers/towns-villages-island-sunland-connected-extensive-rail-network-doweltown-capital-sunland-d-q113191339

2.) https://people.cs.umass.edu/~sheldon/teaching/mhc/cs312/2016sp/lec/18-max-flow-handout-key.pdf

3.) https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/