Analysis and Design of Algorithms

Theory Assignment 3

Mehar Khurana (2021541) — Nalish Jain (2021543)

April 27, 2023

Question 1

The towns and villages of the Island of Sunland are connected by an extensive rail network. Doweltown is the capital of Sunland. Due to a deadly contagious disease, recently, few casualties have been reported in the village of Tinkmoth. To prevent the disease from spreading to Doweltown, the Ministry of Railway of the Sunland wants to completely cut down the rail communication between Tinkmoth and Doweltown. For this, they wanted to put traffic blocks between pairs of rail stations that are directly connected by railway track. It means if there are two stations x and y that are directly connected by railway line, then there is no station in between x and y in that particular line. If a traffic block is put in the track directly connecting x and y, then no train can move from x to y. To minimize expense (and public notice), the authority wants to put as few traffic blocks as as possible. Note that traffic blocks cannot be put in a station, it has to be put in a rail-track that directly connects two stations.

Formulate the above as a flow-network problem and design a polynomial-time algorithm to solve it. Give a precise justification of the running time of your algorithm.

Solution

1.1 Input

As input, we receive a **graph** G representing the railway network of the Island of Sunland. In this graph, the **vertices** $v \in V$ represent the different railway stations, and the **edges** $e \in E$ represent the railway tracks between these stations.

It is assumed that we are trying to block the flow of people from Tinkmoth to Doweltown only, i.e., Tinkmoth is the source (where the cases have been reported), and Doweltow is the sink. We also assume that there are no 'one-way' tracks.

1.2 Output

The **minimum number of traffic blocks** to be placed on the railway tracks in order to block the flow of people from Tinkmoth to Doweltown.

1.3 Algorithm

The algorithm we use to find the capacity of the min-cut is the **Ford-Fulkerson Algorithm**. This algorithm presents a solution to the Max-Flow Problem in a flow network. We also use the **Max-Flow Min-Cut Theorem**, which has been discussed in class, to prove that the capacity of the min-cut in a flow network is numerically equivalent to the max-flow passing from the source to the sink in that network.

Hence, a combination of the Ford-Fulkerson Algorithm and the Max-Flow Min-Cut Theorem can be used to find the solution in polynomial time.

1.4 Formulation

The problem asks us to find the minimum number of tracks to block to prevent the flow of people from Tinkmoth to Doweltown. Initially, we assign a capacity of 1 to each edge in the graph. The minimum number of blocks to be placed can be modelled by the capacity of the min-cut of this graph. Since the edges (railway tracks) given to us are undirected, we split these into two directed edges, i.e., instead of undirected edges between every connected pair of vertices (u, v) we assume a directed edge from $u \to v$ and another directed edge from $v \to u$. Both these edges have a capacity of 1. In this formulation, placing a block on a directed edge would mean that we would have to place a block on the reverse edge. These two blocks would actually be counted as a single block in the original network since it has undirected edges (and not two directed edges).

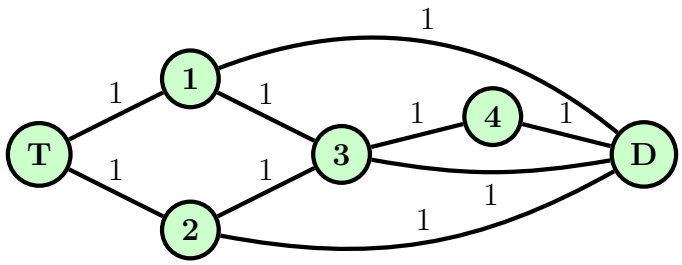

Figure 1: Initial graph: T and D represent the source and the sink respectively, and the numbers represent other stations.

Since we are trying to block the flow of people from Tinkmoth (source) to Dowelmouth (sink) only, we remove the edges moving into Tinkmoth and the edges moving out of Dowelmouth. This is done to ensure that any paths leading out of the vertex we are currently considering as the source, do not lead back into the source (flow has to move from source to sink). Similarly, edges that lead out of the sink would never be evaluated while considering the flow from the source to the sink.

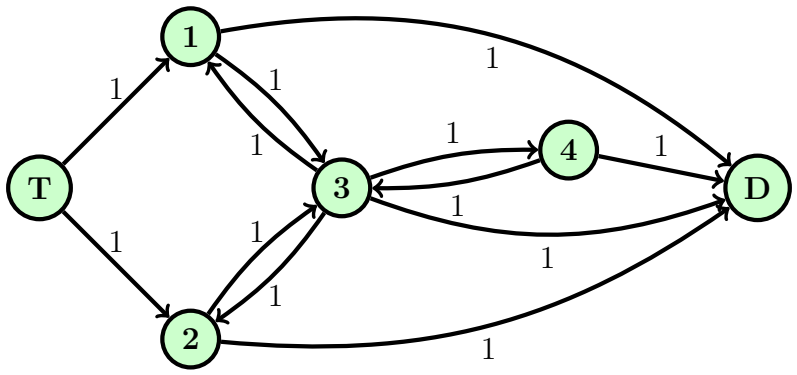

Figure 2: Graph after splitting each edge, and removing in-edges of the source and out-edges of the sink

In the above diagram, consider the cut $S_1 = \{T, 1, 2, 3\}$, $S_2 = \{4, D\}$. While calculating the capacity of this cut, we only consider the directed edges going from S_1 to S_2 (i.e., only the edges going from the set containing the source to the set containing the sink, since this is the direction of the flow we are concerned about). The capacity of this cut is, hence, 4. Applying the Ford-Fulkerson Algorithm, and subsequently, the Max-Flow Min-Cut Theorem on this graph, we find the min-cut to be $S_1 = \{T\}$, $S_2 = \{1, 2, 3, 4, D\}$, with its capacity being 2.

The capacity that we obtain by applying the Ford-Fulkerson Algorithm and the Max-Flow Min-Cut Theorem is numerically equivalent to the number of blocks we need to place. The min-cut is defined as the minimum number of edges to cut to prevent any flow from the source to the sink. So, when we block the edges going from S_1 to S_2, as has been discussed above, we also need to block any reverse edges that may exist. And hence, in the original graph, we would block the edges $T \longleftrightarrow 1$ and $T \longleftrightarrow 2$.

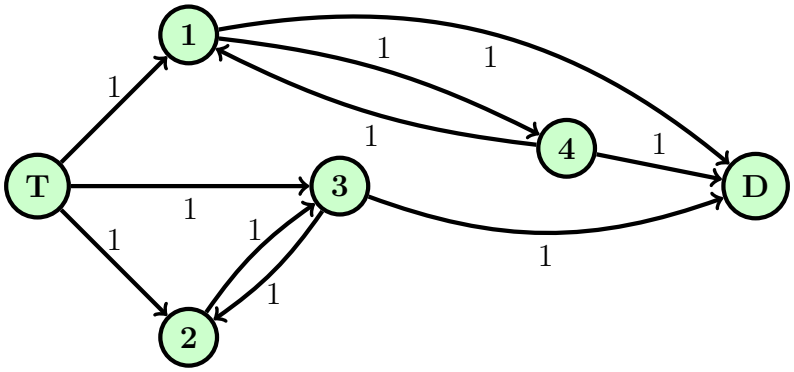

Figure 3: Another Example

Using the same approach on the above example, we find the min-cut to be $S_1 = \{T, 2, 3\}$, $S_2 = \{1, 4, D\}$, with the capacity of the min-cut being 2.

NOTE: Splitting each undirected edge into two directed edges works, because while calculating the min-cut, we only consider the edges moving from S_1 to S_2. This omits all paths that would move back from S_2 to S_1 since these would produce redundant paths. In the residual graph, if the flow through an edge is equal to the capacity of that edge, the flow through the back edge between the same two vertices is 0.

1.5 Time Complexity

We first convert all the undirected edges into a pair of forward- and backward-directed edges. This takes O($|E|$) time. Now, we remove the edges moving into the source and the edges moving out of the sink. This, in the worst case would take O($|E|$) time (if all edges have either the source or the sink at one end).

We now run the Ford-Fulkerson Algorithm on this graph. The Fulkerson-Algorithm, in each iteration, finds an augmenting path. The residual graph G_r can have a minimum of $|E|$ and a maximum of $2 \cdot |E|$ edges. For each iteration, the algorithm conducts a DFS traversal, to find an augmenting path from T to D. Hence the runtime for a single iteration is

$$O(|V| + 2 \cdot |E|) = O(|V| + |E|)$$

In each iteration, the flow in the graph is increased by at least 1, meaning that it can take a maximum of F iterations, where F is the value of the max-flow. F is upper-bounded by $|E| \cdot C$, with C being the maximum capacity of any edge, which in this case is 1. Hence, there can be a maximum of $|E|$ iterations in this problem.

The above observations help us conclude that this solution takes O($F \cdot (|V| + |E|)$) time to complete, which in this case can also be written as O($|E| \cdot (|V| + |E|)$), as explained above.