Flow Report

Amanda Sauerberg-Hansen, Kamila Garczyńska,
Nedas Šurkus, Niclas Abelsen, Tomas Babkine-Di Caprio

October 2022

1 Results

Our implementation successfully computes a flow of 163 on the input file, confirming the analysis of the inaccurate map that the pitiful Capitalist American army gathered. We have analysed the possibilities of capitalist attack on our railways to cut us off from vital supplies.

Our comrades should reinforce the following stations:

Case	Location name	Location name	Effect possible effect on flow
1	b	b	19
2	R	b	5
3	b	4	10
4	b	8	30
5	b	1	16
6	b	M	36
7	b	8	17
8	1	4	6
9	1	1	24

I repeat. You station more troops to counter enemy sabotage at:

Case	Location ID	Location ID	Effect possible effect on flow
1	28	30	19
2	29	30	5
3	31	41	10
4	38	46	30
5	39	44	16
6	39	45	36
7	39	46	17
8	40	41	6
9	40	44	24

2 Implementation details

We use a straight forward implementation of Ford-Fulkerson Algorithm as described in *Algorithm Design* by Kleinberg and Tardos.

We utilise BFS (Breadth First Search) algorithm to find an augmenting path to traverse. This algorithm runs in O($n + m$) time where n is the number of vertices and m is the number of edges. Due to this the time complexity for Ford-Fulkerson can be defined by the amount of times we search for an augmenting path in a while loop. Thus, our time complexity is O($(n + m) * F$) time, as $n + m$ is the algorithm used to find the augmenting path in the while loop and F is maximum flow. F at worst can increase by 1.

We also assume that maximum flow is minimum cut. To get the minimum cut points we use DFS (Depth First Search) algorithm to find the path trough the residual graph. This algorithm also uses same time complexity as BFS. As such it runs in O($n + m$). After we have the path we utilised a nested for loop to traverse through the graph which executes in O(n^2) time where n is the number of vertices.

$$\text{The total run time is } O(((n + m) * F) + n + m + (n^2))$$

We have implemented a 2D array that represents a graph. This arrays length is [n][n] (n representing the amount of vertices). We then filled out the whole 2D array with 0's. 0 represents that there is no path between Vertex A and Vertex B. In the input we are provided with Vertex A index, Vertex B index and capacity. To signify that there is a path between Vertex A and B we replace the 0 in [VertexA][VertexB] to the capacity and did likewise with [Vertex B][VertexA] to signify that the edge is undirected. If the capacity is -1, this edge has infinite flow. With this in mind, we assigned `float.inf` to each such edge instead of -1.

Finally, when executing Ford-Fulkerson we specified that BFS should ignore all paths with 0. This includes paths that have maximum saturation or no path. Example:

```
graph = [
        [ 0, float.inf, float.inf, 0, 0, 0 ],
        [ float.inf, 0, 4, 0, 0, 0 ],
        [ float.inf, 4, 0, 9, 14, 0 ],
        [ 0, 0, 9, 0, 0, float.inf ],
        [ 0, 0, 14, 0, 0, float.inf ],
        [ 0, 0, 0, float.inf, float.inf, 0 ]
    ];
```