Christian Møller Jensen, Martin Ejler Borup-Larsen,
Balazs David Toth, Bjørn Anton Utzon, Jonas Daniel Ancher

1 Results

Our implementation successfully computes a flow of 163 on the input file, confirming the analysis of the American enemy.

We have analysed the possibilities of decreasing the capacities near Minsk. Our analysis is summarized in the following table:

Case	4W–48	4W–49	Effect on flow
0	30	30	-
1	30	20	no change
2	20	30	no change
3	30	10	no change
4	10	30	no change
5	20	20	-10
6	10	20	-10
7	20	10	-10
8	10	10	-20
9	30	0	no change
10	0	30	no change

(a) Effects of changing flows

$$\begin{bmatrix} 29 & 30 & 5 \\ 28 & 30 & 19 \\ 31 & 41 & 10 \\ 40 & 41 & 6 \\ 40 & 44 & 24 \\ 39 & 44 & 16 \\ 39 & 45 & 36 \\ 39 & 46 & 17 \\ 38 & 46 & 30 \end{bmatrix}$$

(b) Min-cut of original data

The comrade from Minsk can remove either the train line $4W - 48$ or $4W - 49$ without loosing any overall flow.

2 Implementation details

We use the Ford-Fulkerson algorithm as introduced in chapter 7 of Algorithm design, leaning on the pseudo code from page 347. For the path finding algorithm we utilize a Depth-First-Search taking inspiration from the following link.

The total running time of our implementation is $O(Ef)$, where f is the maximum flow value. For each iteration of the while loop, we first do a depth first search, which takes time $O(E + V)$. But since we are assuming that all nodes has at least one edge, we have that $E \geq \frac{V}{2}$, which means $V \in O(E)$. Thus we can write that the run time of the depth first search is $O(E)$. Then we pass the resulting path, to the augment function. This function takes time proportional to the length of the path, which is $O(V)$, so $O(E)$. This means each iteration takes $O(E)$ time. Since all capacities are integer values, the flow value must increase by minimum 1 each iteration, upperbounding the amount of iterations by the actual max flow value.

We have implemented each undirected edge in the input graph as a bidirectional edge, represented in the Edge class. An Edge has two flows, one from a node to b node, and the reverse flow as well. Instead of having two graphs, one normal and one residual, we simply work on a single all-encompassing graph.

```
class Edge:
    def __init__(self, start: int, end: int, capacity: int, infinite: bool):
        self.infinite     = infinite
        self.start        = start
        self.end          = end
        self.ab_residual  = capacity
        self.ba_residual  = capacity

    def get_residual(self, origin:int):
```

```
        return self.ab_residual if origin == self.start else self.ba_residual

    def set_residual(self, origin:int, amount: int):
        if origin == self.start:
            self.ab_residual = amount
        else:
            self.ba_residual = amount
```