Flow Report

August, Johannes, Jonas, Martin, Michelle, Thomas
Group K

October 2020

Results

Comrades! 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 summaries in the following table:

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

In case 4, the new bottleneck becomes

10–25, 11–25, 11–24, 12–23, 18–22, 20–21, 20–23, 27–26, 28–30

The comrade from Minsk is advised to ensure that there is at least a total of capacity 40 over the railroads from 4W-48 and 4W-49, otherwise the total capacity of the network will decrease.

Implementation details

We use a straightforward implementation of Ford-Fulkerson's flow algorithm as described in Kleinberg & Tardos, *Algorithm Design*, chap. 7. We use an inverse Dijkstra's algorithm prioritizing edges with highest capacity to find an augmenting path (through the use of a heap).

The running time is $O(mC + nC)$.	
Construction of graph takes:	O(n+m)
Finding a s-t path takes:	O(m+n)
and augmenting takes:	O(n).
Therefore one-iteration of max-flow takes:	O(n+m+n) = O(n+m).
Number of iterations, where C=sum of capacity out of s:	O(C)
Total running time (construction of graph + path finding * C):	O(n+m) + O(n+m) * O(C) = O(nC+mC)

We have implemented each undirected edge in the input graph as a directed edge from u to v, and created an equal edge from v to u. Rather than have a separate residual graph, our graph updates the capacity on the directed edges as we increase/decrease flow. Our datatype for edge is this:

```
class Edge(object):
    def __init__(self, u, v, c):
        self.source = u
        self.destination = v
        self.capacity = c if c != -1 else float('inf')
        self.reverse = None
```

The edge carries information about source and destination nodes (defined as separate node objects that each have a list of outgoing edges), its capacity (we set it to infinity if it is -1 in the input) and a reference to the other directed edge in the opposite direction between v and u, which allows us to easily update the capacities of the edges when pushing flow along them. Since you can only travel along an edge in one direction (trains cannot run into each other), it doesn't conflict with the Ford-Fulkerson algorithm to use the edge in the opposite direction as the back-edge.

When printing the capacity, we divide by 2 because the outgoing edges at the min cut will have flow 0 and the reverse edge will have c(e) * 2, as it is the sum of the original edge's capacity and the reverse edge's capacity.