Flow Report

Lasse Faurby (lakl), Nikolaj Worsøe Larsen (niwl), Philip Flyvholm (phif), Simon Johann Skødt (sijs) and Thomas Rørbech (thwr).

November 15, 2023

Results

Our implementation successfully computes a flow of 163 on the input file, confirming the analysis of the American enemy. When comparing the minimum cut from `result.txt` and our implementation, we get the same minimum cut. Refer to Table 1 below.

(a) Min-cut from result.txt

v	u	c
28	30	19
29	30	5
31	41	10
38	46	30
39	44	16
39	45	36
39	46	17
40	41	6
40	44	24

(b) Min-cut from our implementations

v	u	c
28	30	19
29	30	5
31	41	10
38	46	30
39	44	16
39	45	36
39	46	17
40	41	6
40	44	24

Table 1: Comparison between result.txt and our implementation output. The tables describe the railway lines as edges from u to v with a capacity c.

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
4	10	30	no change
5	30	10	no change
6	20	20	no change
7	10	20	153
8	20	10	153
9	10	10	143

In case 7, the new bottleneck becomes

0–17, 17–20, **20–21**, 21–36a, 36–38, 38–46, 46–48, 48–54

The line from Minsk (node 20) to node 21 now becomes the new bottleneck.

The comrade from Minsk is advised to decrease both to a capacity of 20 because this does not change the max flow.

Implementation details

We use a straightforward implementation of Vanilla Ford-Fulkerson's flow algorithm described in Kleinberg's, *Algorithm Design*, Chap. 7. We use the BFS algorithm to find a path to augment that takes $O(V+E)$ running time, where V is the number of vertices and E is the number of edges. Our augment function takes $O(V)$ time as the path has at most $n-1$ edges. It takes the program $O(V+E)$ to build the graph since it needs to load all vertices and all edges. Since it is a road network, we can assume that all nodes, except start and end nodes, have at least one incident edge. This means that $E \geq V/2$, and the total running time is $O(V+E) = O(E)$.

We have implemented each undirected edge in the input graph as two directed edges that go both ways.

```
def add_edge(self, n1, n2, capacity=float('inf')):
    if n1 in self.graph and n2 in self.graph:
        self.graph[n1].append((n2, capacity))
        if n1 not in self.graph[n2]:
            self.graph[n2].append((n1, capacity))
```

Our graph and corresponding residual graph are the same, meaning that we build the graph and mutate the graph while performing augmentation.

```
def augment(path, flow, graph):
    for i in range(len(path)-1):
        n1, n2 = path[i], path[i+1]
        forward_edge_new_cap  = graph.get_path_capacity(n1, n2) - flow
        backward_edge_new_cap = graph.get_path_capacity(n2, n1) + flow
        graph.update_path_weight(n1, n2, forward_edge_new_cap)
        graph.update_path_weight(n2, n1, backward_edge_new_cap)
```

Our graph is represented by a dictionary with the nodes as keys and values are a list of tuples containing the adjacent node and capacity.