Flow Report

Alexander Berg, Thomas Hoffmann Kilbak, Kristoffer Bruun Højelse, Emil Jäpelt, Adam Negaard

October 25, 2022

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 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
[. . .]			

In both case 4 and 5, the new bottleneck becomes

52–51, 7–51, 7–50, 5–49, 46–47, 4W–49, 4W–48, R–b, b–H

Implementation details

We use a modified version of the implementation described in section 7.1 of Kleinberg and Tardos, *Algorithm Design*, Pearson 2014. The implementation has been modified in such a way that it supports edges with infinite capacities and undirected edges. The running time is $O(mnc)$.
We have implemented each undirected edge in the input graph as two directed edges as is typically done when using an adjacency list. Our datatype for edge is this:

```
class Edge
{
    public Edge other;
    public int from;
    public int to;
    public int id;
    public bool isResidualEdge;
}
```

The "other" field is a reference to the directed edge going in the opposite direction. "from" and "to" are the indices of the nodes connected by this edge. "isResidualEdge" indiates, as the name implies, whether this edge is a residual edge. The "id" field is the id of a single undirected edge. This means that two directed edges going in opposite direction have the same id. Note that the edge datatype does not contain any fields describing the flow or the capacity. To support undirected edges (meaning that the flow can go in one of two directions), the flows and capacities are stored in lists:

```
List<int> flows = new();
List<int> capacities = new();
```

The flow and capacity of an undirected edge can be looked up in these lists by using the id field in the edge datatype. This implementation allows the flow of an edge to be negative which is used to indicate the direction of flow.
We support edges with infinite capacities by allowing the capacity to be -1. We then check, when relevant, whether the capacity is infinite. We do not change the flow of edges with infinite capacities when augmenting paths.

We find the edges in the minimum cut by traversing the graph using BFS starting at the source node. We follow all edges except residual edges with absolute value of their flow equal to their capacity. The result of this is a subgraph containing the source with edges to another subgraph containing the sink. We then traverse the first subgraph. All edges going to non-visited nodes are added to a list. This list contains the edges in the minimum cut.
The program can be run using .NET 6.0 by navigating to the folder containing "Program.cs" and executing the command:

dotnet run

When the program starts it will read the file rail.txt. The program writes the solutions to standard output.