Operations Research Project

Matthias Carré

May 2025

Contents

1 Introduction

The aim of this project is to work with graphs and implement algorithms on them. For simplicity, the choice of language was Python, which makes it easy to handle various data structures.

2 Structure

2.1 Parser

The first part of the program is the parser. It reads the text file containing the graph and transforms it into Python structures as follows:

- A dictionary containing:
  - the number of nodes,
  - the number of edges,
  - the source node,
  - the sink node.
- An array containing, for each edge:
  - the start node,
  - the end node,
  - the maximum flow,
  - the cost of the edge.

2.2 Scripts

The structure created is then used by the other programs. The array is converted into an adjacency list in the form of: a Python dictionary containing, for each node, an array with:

- the destination node,
- the maximum flow,
- the cost of the edge,
- the used flow.

3 Implementations

3.1 Implemented Utility Functions

3.1.1 DFS

Two `dfs` functions are implemented:

- **dfs**: returns the list of edges forming a path from the source to the sink, stopping at the first one found.
- **dfsAccessible**: returns the list of nodes accessible from a given node.

3.1.2 Edge Functions

- **getEdge**: returns the information of the edge between two given nodes.
- **reduceFlowEdge**: decreases the flow of the edge between two given nodes by the specified value. If an edge is saturated, it is removed and added to a list.
- **increaseFlowEdge**: increases the flow of the edge between two given nodes by the specified value.

3.1.3 Residual Graph

Takes an adjacency list and a path to create the residual graph by reducing the capacities of the edges along the given path.

3.1.4 Dijkstra

Takes an adjacency list and a source and returns the distance of all nodes from the source.

3.2 Maximum Flow

To compute the maximum flow of the graph, the following steps are repeated:

- Find a path from the source to the sink using `dfs`.
- If a path is found, reduce the capacity of all edges along the path by the minimum capacity available on that path.
- Compute the residual graph and repeat the process.

When no more path is found, the graph is saturated. The maximum flow value is then obtained by summing the flows of the edges outgoing from the source.

In practice, the code works with two graphs:

- the first one where only the flow values of the edges are updated,
- and the residual graph where saturated edges are removed and where `dfs` is performed.

This makes it possible, when no more path is available, to use the first graph to get the information about the remaining capacities.

3.3 Minimum Cut

The minimum cut is closely related to the maximum flow, which is why both are computed in the same program. The same steps are followed, but in addition, the removed edges from the original graph are stored.

At the end of the algorithm, we check which removed edges have one endpoint accessible from the source and the other not. This is done using a `dfs`. From this, the list of edges forming the graph's minimum cut is deduced.

3.4 Minimum Cost Maximum Flow

To compute the minimum cost maximum flow, the following steps are repeated:

- Find the minimum cost path from the source to the sink using Dijkstra.
- If it exists, compute the residual graph by reducing the capacities along this path.
- Update the edge costs using the formula:

$$c_{i,j} \leftarrow d(i) + c_{i,j} - d(j)$$

  where d is the distance from the source and c the cost of the edge.

When no more path is found, sum the outgoing flows from the source to obtain the maximum flow. Finally, compute the total cost using:

$$\sum_{e \in G} c(e) \times f(e)$$

with $c(e)$ the cost of edge e and $f(e)$ the flow passing through e.

4 Conclusion

This project made it possible to better understand graph algorithms and how they work. Using Python was the simplest and most logical choice due to its ease in handling data structures, although it remains less efficient when working with larger datasets.