Instructors:
Michael Ritter
Hugo K. Kasuya Rosado

Exercise sheet #8
Combinatorial Optimization
SuSe 2024

Exercises with an asterisk mark "*" are optional.

1 Successive Shortest Path Algorithm

Consider the following instance of the MINIMUM COST FLOW PROBLEM in Figure 1.

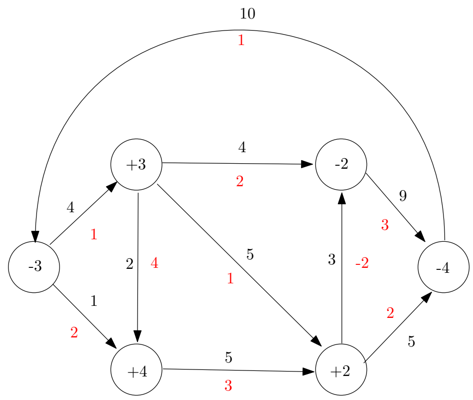

Figure 1: A network. Black numbers represent the capacity of edges and red numbers represent the costs.

a. Show that the edge costs of the instance are conservative.
b. Find an optimal solution by executing the SUCCESSIVE SHORTEST PATH ALGORITHM.

2 Irrational Capacities

Consider the following instance and partial solution of the MINIMUM COST FLOW PROBLEM with irrational capacities depicted in Figure 2. In this figure, $\phi = \frac{\sqrt{5}-1}{2}$ is the golden ratio.

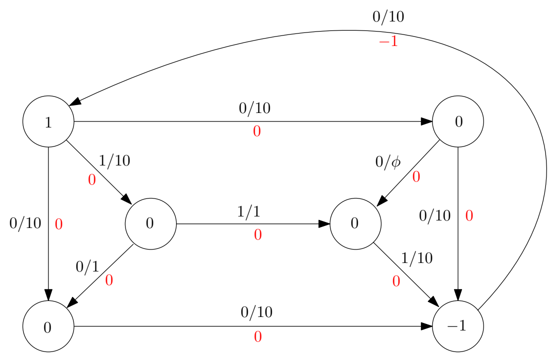

Figure 2: A network with irrational capacities and a b-flow. On each edge, the numbers in red represent its cost, and the black numbers on the left of a slash represent a flow through the edge and on the right the capacity of the edge. Numbers on the vertices represent the supply and demands.

a. What is the cost of an optimal b-flow in this network?
b. Consider an algorithm where one starts with a b-flow f and, as long as there is a negative cycle C in G_f, iteratively augments f with C. Notice that instead of a minimum mean cycle, as in the MINIMUM MEAN CYCLE-CANCELLING ALGORITHM, this algorithm just chooses some negative cycle.
Starting with the b-flow depicted in Figure 2, show that this algorithm might not terminate.
Hint: $\phi^{n+1} = \phi^n - \phi^{n+2}$, and $\sum_{i=1}^{\infty} \phi^i = 1 + \phi$.

3 Coding the Minimum Mean Cycle-Cancelling Algorithm

In this exercise, we will write a program to run the MINIMUM MEAN CYCLE-CANCELLING ALGORITHM.

a. Download from Moodle the following files and place them directory/folder: `main.py`, and `graph_module.py`.

   (a) file `main.py` is the file we will be executing in the terminal and it calls functions and methods defined in the other file;
   (b) `graph_module.py` is the file we will be working on in this exercise. It contains an incomplete implementation of the Minimum Mean Cycle Algorithm.

   With Python 3 installed, to run the code, simply open the terminal, navigate to the folder containing the files, then run the following command to run the code

```
python3 main.py
```

b. Open `graph_module.py`. This file contains the method `minimum_mean_cycle_cancelling()`, which has no arguments. This method should return the total cost of a minimum cost flow if any, otherwise, it should return `None`.

   In the file `main.py`, on line 9, one can manually write the `adjacency_matrix` which is used as the input network for the code, and on line 13 one can manually write `sup_dem` to define the supply and demands of each vertex. The variable `adjacency_matrix` is a matrix such that `adjacency_matrix[u][v][0]` is the capacity of the edge (u, v) and `adjacency_matrix[u][v][1]` is its cost, and `sup_dem[v]` denotes the supply of vertex `v` if positive, or its demand if negative. In this exercise, the input matrix should contain no anti-parallel pairs of edges (this is done to simplify the overall matrix structure). Hence, in the input `adjacency_matrix`:

   (a) **If edge (u, v) does not exist**: `adjacency_matrix[u][v][0]` $= 0$.
   (b) **If neither edges (u, v) or (v, u) exist**: then `adjacency_matrix[u][v][1]` $=$ `adjacency_matrix[v][u][1]` $= \infty$.
   (c) **There are no loops**: `adjacency_matrix[u][u][1]` $= 0$ and `adjacency_matrix[u][u][1]` $= \infty$ for all `u`.
   (d) **There are no reverse edges**: if edge (u, v) exists in the input, then its capacity `adjacency_matrix[u][v][0]` $> 0 =$ `adjacency_matrix[v][u][0]` and its cost `adjacency_matrix[u][v][1]` $= -$ `adjacency_matrix[v][u][1]`.

   Notice that if an edge (u, v) exists, then its edge capacity is greater than 0 and the edge (v, u) has capacity 0 (edge does not exist) and its cost is the negative of the cost of edge (u, v). Hence, in the residual graph, if a flow is pushed through (u, v), one needs only to update the capacity of (v, u), as its cost is already correctly defined.

   Optionally, line 30 can be uncommented to use an input network generated randomly (details are in the comments in the file).

c. Implement the remaining of `minimum_mean_cycle_cancelling()` - there is no need to optimize the time complexity of your code. Here is a list of useful variables and methods you could use for the implementation:

   (a) `self.vertex_num` returns the number of vertices and `self.vertex_list` returns a list of vertices (each vertex is an integer between 0 and `self.vertex_num-1`).
   (b) `self.original_adjacency_matrix` is the original adjacency matrix of the input network (this variable should never be modified). Here, `self.original_adjacency_matrix[u][v][0]` and `self.original_adjacency_matrix[u][v][1]` returns the capacity and the cost of edge (u, v).
   (c) `self.residual_graph` is the adjacency matrix of the residual graph that is maintained and modified by the methods defined in the file `graph_module.py`. Here, `self.residual_graph[u][v][0]` and `self.residual_graph[u][v][1]` returns the capacity and the cost of edge (u, v) in the current residual graph. Initially, `self.residual_graph` and `self.original_adjacency_matrix` are identical.
   (d) `self.initial_b_flow()` is a method that updates the current residual graph stored in `self.residual_graph` with the residual graph of some b-flow f in `self.residual_graph`. If no b-flow is found, this method returns `False`.
   (e) `self.minimum_mean_cycle()` is a method that returns a minimum mean cycle in the residual graph stored in `self.residual_graph`. If a cycle exists, this method returns a tuple `(C, mu)`, where `C` is a minimum mean cycle and `mu` is its mean cost, otherwise it returns `None`. The returned minimum mean cycle `C` is a list of lists of the form `[u,v,[capacity(u,v), cost(u,v)]]`, where (u, v) is an edge in the cycle. Use this method to iteratively find minimum mean cycles and update the residual graph `self.residual_graph` accordingly.

   The method `minimum_mean_cycle_cancelling()` should return the cost of a minimum cost b-flow. To compute the cost of a flow, compare the current residual graph `self.residual_graph` to the original `self.original_adjacency_matrix`.

d. In the file `main.py`, uncomment line 30 to generate random graphs, and modify the parameters to see how it affects the output and the running times of the algorithm.