MAC0328 GRAPH ALGORITHMS: PROGRAMMING ASSIGNMENT 4

MAX-FLOW MIN-CUT

MARCEL K. DE CARLI SILVA AND THIAGO LIMA OLIVEIRA

Due Date: 21/Dec/2022 at 11:59pm

In this programming assignment, your task is to write an implementation of the Edmonds–Karp Algorithm for the Maximum Integral Flow Problem.

This assignment will be graded out of 100 marks.

Your code **must** be written in `C++17` and use the BGL library. As before, you must use the BGL **only** for the data structures (and corresponding accessor functions) that store a digraph and the attributes for its vertices and arcs. The use of any **BGL algorithm** is forbidden.

1. Test Cases and Grading

Your program should solve each test case in $O((n+m)nm)$ time, where n is the number of vertices and m is the number of arcs of the input digraph $D = (V, A)$.

Each test case has the following format:

- The first line has two integers, n and m, the numbers of vertices and arcs, respectively, such that $1 \leq n \leq 5 \times 10^3$ and $1 \leq m \leq 5 \times 10^3$.
- The second line has two integers, s and t in the range $[1, n]$, the source/start and the target/sink of the flow.
- The next m lines have the description of the arcs. Each arc is described by three integers, u, v, and $c(uv)$, that is, the tail and head of the arc (in the range $[1, n]$, as usual), and the capacity of the arc, respectively. The capacity satisfies $1 \leq c(uv) \leq 10^3$.

The ordering of the arcs in the input will be important for the output, so it may be convenient to keep track of it. You may use the fact that a call to `boost::add_edge` returns a `std::pair<Arc,bool>`, where `Arc` is the type for an edge descriptor of the chosen graph type, containing an edge/arc descriptor and a boolean that indicates whether the edge/arc was successfully added. Hence, one may want to record the value of `a` after running the following snippet:

```
Arc a; std::tie(a, std::ignore) = boost::add_edge(u, v, digraph);
```

It is *mandatory* for the augmented digraph $\widehat{D} = (V, A \times \{\pm 1\}, \varphi)$ to have, for each original arc that has a positive amount of flow, a *new* reverse arc, *even if the digraph already has an original arc in that direction.*

For each iteration of the Edmonds–Karp algorithm, your solution should output a description of the residual capacities of all the arcs in the augmented digraph $\widehat{D} = (V, A \times \{\pm 1\}, \varphi)$ and either the flow update data or a description of a minimum cut, in the following format:

Instituto de Matemática e Estatística, Universidade de São Paulo, R. do Matão 1010, 05508-090, São Paulo, SP
E-mail address: `{mksilva,thilio}@ime.usp.br`.

- The first m lines contain a description of the residual capacities. For each arc $a \in A$, *in the same order as the input*, print a line containing two integers, $c_f((a,+1))$ and $c_f((a,-1))$, i.e., the residual capacities of the arcs $(a,+1)$ and $(a,-1)$ of $\widehat{D}$. Note that you should print both residual capacities, even if one of them is zero (so the corresponding arc lies in $\widehat{D}$ but not in the residual digraph D_f).
- If there is no augmenting path, the code must print two lines. The first line should start with three integers, 1, $\operatorname{val}(f)$, and $|S|$, where $S \subseteq V$ is such that $s \in S$, $t \notin S$, and $\operatorname{val}(f) = c(\delta^{\text{out}}(S))$. After this, the next line should have $|S|$ integers, one for each vertex of S, in the range $[1, n]$.
- If there is a *shortest* augmenting path P, the code must print two lines. The first line should start with three integers, 0, ε, and ℓ, where $\varepsilon > 0$ is the minimum residual capacity of an arc traversed by P and ℓ is the length of P. After this, the next line should have ℓ integers, one for each arc traversed by P, in order. To represent an arc (a, α) of D_f traversed by P, with $a \in A$ and $\alpha \in \{\pm 1\}$, print the index of the arc a in the input order (in the range $[1, m]$) if $\alpha = +1$ or minus that index (in the range $[-m, -1]$) if $\alpha = -1$.

Your submission should be a single file, called `NUSP.cpp`, obviously with `NUSP` replaced by your university ID number.

You can find the public test cases in the public repository for the course at https://gitlab.uspdigital.usp.br/mksilva/mac0328-2022-public/-/tree/master/assignments/asgt4/tests.