CS29003 Algorithms Laboratory

Assignment No: 10

Date: 07–April–2020

Shortest Paths in Graphs

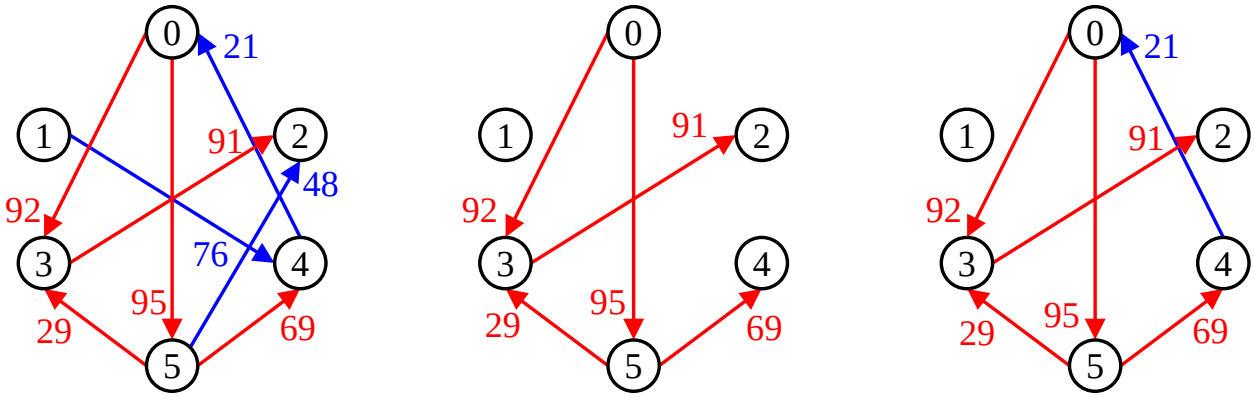
Foobarland has n cities numbered $0, 1, 2, \ldots, n-1$. The official airline for Foobarland is Air Ifoobria (AI). For any pair (i, j) of cities with $i \neq j$, there are three possibilities: (i) there are flights from i to j operated by AI, (ii) there are flights from i to j operated by private (non-AI) airlines, and (ii) there are no flights from i to j. Here, a flight means a direct (that is, non-stop) flight. For every city-to-city flight leg, the operating airline declares a ticket price. The existence of flights and the prices are not symmetric. That is, there may be a flight from City i to City j but not from City j to City i. Even when two cities are two-way connected, the airline and/or price for one direction may be different from those in the other direction.

You naturally model the flight information using a directed graph $G = (V, E)$. The vertices of G are the cities of Foobarland. If there is a flight from City i to City j, then there is a directed edge from Vertex i to Vertex j. Each edge carries two items: (i) whether the operator is AI or non-AI, and (ii) the price of the ticket.

Rules in Foobarland prohibit government employees from traveling by non-AI flights for official works. Mr. Sarkar from the Aviation Ministry is given the task of preparing charts of minimum city-to-city flight prices with multiple legs allowed. He plans to build three charts.

1. The first chart is based only on AI flights.
2. It is possible that some pairs of cities are not at all connected by AI flights. A second chart is needed where only one non-AI leg is allowed in each route not served by AI.
3. In case a pair of cities is not connected by AI flights, any schedule that can connect the pair with an arbitrary number of non-AI legs is to be stored in the third chart.

The following figure demonstrates the situation. AI flights are shown by red edges, and non-AI flights by blue edges. The prices shown in the figure are scaled-down versions of the example given in the Sample I/O section. On the left, you have the entire flight schedule for both AI and non-AI airlines. In the center, only the AI flights are shown. There is no AI-operated path from City 4 to City 2. But if you allow one non-AI leg (edge), you have two paths $4, 0, 3, 2$ and $4, 0, 5, 3, 2$ of costs $21 + 92 + 91 = 204$ and $21 + 95 + 29 + 91 = 236$ (see the right side of the figure). If you allow multiple non-AI legs, there is another path $4, 0, 5, 2$ from City 4 to City 2 having cost $21 + 95 + 48 = 164$. As another example, City 1 cannot be connected to City 0 unless multiple non-AI legs are allowed.



Part 1: The graph data type and its construction

A flight-schedule graph consists of three items: (i) the number n of cities (an integer), (ii) an $n \times n$ array of operator airlines (use a character for each entry), and (iii) an $n \times n$ array of integers storing the ticket prices. This is similar to an adjacency-matrix representation.

The (i,j)-th entry of the operator array stores a if there is an AI-operated flight from City i to City j, or n if there is a non-AI flight from City i to City j, or $-$ if there is no flight from City i to City j, or s (stay put) if $i = j$. If $i \neq j$ and there is no flight from City i to City j, store $+\infty$ as the ticket price. For $i = j$, store 0 as the ticket price.

Write a function *readgraph* that constructs G from user inputs, and returns it. The user first enters n (the number of cities). The user then supplies quadruples (i, j, c_{ij}, a_{ij}), where i is the source city, j is the destination city, c_{ij} is the ticket price for the City $i \to$ City j flight, and a_{ij} is the operator airline of that flight (a if AI, n if non-AI). The user input ends when -1 is supplied as i.

Part 2: Build the AI subgraph

Write a function *getAIgraph*(G) that produces the subgraph H of G storing only the AI flights (all vertices but only the red edges, as in the center of the figure on the previous page).

Part 3: APSP in the AI subgraph

Implement a function *APSP* to implement the Floyd–Warshall all-pairs-shortest-path algorithm. When you call this function with the AI subgraph H as input, you get the first chart C_1 of Mr. Sarkar.

Part 4: APSP with one non-AI leg allowed

The second chart C_2 of Mr. Sarkar keeps all the entries of C_1, that are less than ∞ (indicating AI connectivity). Only when $C_1(i,j) = \infty$, you need to figure out whether by allowing a single non-AI leg, you can connect City i to City j. One possibility is to add a single non-AI edge to H, and run *APSP* on this graph. This is to be done one by one for all non-AI edges. Since there are $O(n^2)$ non-AI edges, and the running time of Floyd–Warshall is $O(n^3)$, you end up with an $O(n^5)$ running time.

The problem can however be solved in $O(n^4)$ time. If $C_1(i,j) = \infty$, choose a non-AI flight (k,l). You may have $i = k$ or $l = j$ or both. Check whether $C_1(i,k)$ + the cost of the $k \to l$ flight + $C_1(l,j)$ is $< \infty$. If so, an allowed $i \to j$ route is discovered. Minimize over all non-AI flights (k,l). Write a function *APSPone*(G, C_1) to implement this idea.

Part 5: APSP with any number of non-AI legs allowed

In the third chart C_3 of Mr. Sarkar, we again have $C_3(i,j) = C_1(i,j)$ if $C_1(i,j) < \infty$ (if AI provides an $i \to j$ service, you have to take it, even if using non-AI legs can reduce the price). However, there is now no restriction on the number of non-AI legs in each minimum-price $i \to j$ path (provided that $C_1(i,j) = \infty$). Write a function *APSPany*(G, C_1) to build the third chart C_3. The design of the algorithm is left to you. Your algorithm should run in $O(n^3)$ time.

The *main*() function

- Call *readgraph* to get the flight-schedule graph G.
- Call *getAIgraph* to get the AI flight-schedule subgraph H.
- Call *APSP* on H to get C_1. Print C_1.
- Call *APSPone* to get C_2. Print C_2.
- Call *APSPany* to get C_3. Print C_3.

Note: The charts C_1, C_2, C_3 store only the cheapest flight prices. In this assignment, you do not have to prepare the cheapest allowed routes.

Submit a single C/C++ source file. Do not use global/static variables.

Sample I/O

The example given below corresponds to the figure on the first page. Better and larger samples will be provided in a separate file. A – in the charts indicates unavailability of flight routes.

```
6
0   3   9199  a
0   5   9484  a
1   4   7624  n
3   2   9055  a
4   0   2127  n
5   2   4830  n
5   3   2901  a
5   4   6877  a
-1

+++ Original graph
    0   ->   3 (9199, a)   5 (9484, a)
    1   ->   4 (7624, n)
    2   ->
    3   ->   2 (9055, a)
    4   ->   0 (2127, n)
    5   ->   2 (4830, n)   3 (2901, a)   4 (6877, a)

+++ AI subgraph
    0   ->   3 (9199, a)   5 (9484, a)
    1   ->
    2   ->
    3   ->   2 (9055, a)
    4   ->
    5   ->   3 (2901, a)   4 (6877, a)

+++ Cheapest AI prices
                0     1     2     3     4     5

    0   ->      0     - 18254  9199 16361  9484
    1   ->      -     0     -     -     -     -
    2   ->      -     -     0     -     -     -
    3   ->      -     -  9055     0     -     -
    4   ->      -     -     -     -     0     -
    5   ->      -     - 11956  2901  6877     0

+++ Cheapest prices with at most one non-AI leg
                0     1     2     3     4     5

    0   ->      0     - 18254  9199 16361  9484
    1   ->      -     0     -     -  7624     -
    2   ->      -     -     0     -     -     -
    3   ->      -     -  9055     0     -     -
    4   ->   2127     - 20381 11326     0 11611
    5   ->   9004     - 11956  2901  6877     0

+++ Cheapest prices with any number of non-AI legs
                0     1     2     3     4     5

    0   ->      0     - 18254  9199 16361  9484
    1   ->   9751     0 24065 18950  7624 19235
    2   ->      -     -     0     -     -     -
    3   ->      -     -  9055     0     -     -
    4   ->   2127     - 16441 11326     0 11611
    5   ->   9004     - 11956  2901  6877     0
```