Task 2 (Centrality): Computing Betweenness Centrality. The betweenness centrality is a number to measure the centrality of a node in a graph. It is proportional to the number of shortest paths passing through this node. Formally, it is defined as the following:

$$g(v) = \sum_{s \neq v \neq t} \frac{|\{s \overset{v}{\rightsquigarrow} t\}|}{|\{s \rightsquigarrow t\}|}$$

where $|\{s \rightsquigarrow t\}|$ denotes the number of shortest paths from s to t, $|\{s \overset{v}{\rightsquigarrow} t\}|$ denotes the number of shortest paths from s to t passes through v, and s and t and v are different. For the example directed graph specified as the following:

```
[(0,1), (0,2), (1,2), (1,3), (1,4), (2,3), (3,5), (4,5)]
```

we have the following shortest paths between different nodes with at least one middle node:

```
0 --> 1 --> 3, 0 --> 2 --> 3,
0 --> 1 --> 4,
0 --> 1 --> 4 --> 5, 0 --> 2 --> 3 --> 5, 0 --> 1 --> 3 --> 5,
1 --> 3 --> 5, 1 --> 4 --> 5,
2 --> 3 --> 5,
```

Then, the betweenness centrality for each node is:

$$\begin{aligned}
g(0) &= 0 \\
g(1) &= \frac{1}{2} + \frac{1}{1} + \frac{2}{3} = \frac{13}{6} \\
g(2) &= \frac{1}{2} + \frac{1}{3} = \frac{5}{6} \\
g(3) &= \frac{2}{3} + \frac{1}{2} + \frac{1}{1} = \frac{13}{6} \\
g(4) &= \frac{1}{3} + \frac{1}{2} = \frac{5}{6} \\
g(5) &= 0
\end{aligned}$$

This task is to compute the normalised betweenness centrality for each node in the graph, i.e.,

$$n(v) = \frac{g(v) - \min}{\max - \min}$$

Notice that if $\max = \min$ then $n(v) = 0$. The input of the algorithm is a list of pairs of natural numbers. It specifies a directed graph. The output is supposed to be a sequence of floating point numbers with 2 digits in the fraction. For the running example, we have the following output:

$$[(0, 0.00), (1, 1.00), (2, 0.38), (3, 1.0), (4, 0.38), (5, 0.00)]$$

The output is required to be sorted in the increasing order of node indices. A good algorithm is supposed to

- produce the correct sequence of normalised betweenness centrality for all nodes in a graph;
- take as little exploration time as possible.

A well-written program with good scalability will be rewarded. The participants are encouraged to explore variants of the single-source shortest paths algorithm, e.g., Dijkstra's algorithm, and the all-pairs shortest paths algorithm, e.g., Floyd-Warshall algorithm; to try data formats which are more suitable to deal with sparse matrices, e.g., compressed sparse row (CSR); and to implement parallel algorithms to cope with huge graphs with high sparsity.