Programming Assignment 4: Paths in Graphs

Revision: July 27, 2020

Introduction

Welcome to your fourth programming assignment of the Algorithms on Graphs class! In this assignments we focus on shortest paths in weighted graphs.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. compute the minimum cost of a flight from one city to another one;
2. detect anomalies in currency exchange rates;
3. compute optimal way of exchanging the given currency into all other currencies.

Passing Criteria: 2 out of 3

Passing this programming assignment requires passing at least 2 out of 3 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

Graph Representation in Programming Assignments

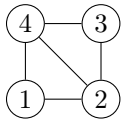
In programming assignments, graphs are given as follows. The first line contains non-negative integers n and m — the number of vertices and the number of edges respectively. The vertices are always numbered from 1 to n. Each of the following m lines defines an edge in the format `u v` where $1 \le u, v \le n$ are endpoints of the edge. If the problem deals with an undirected graph this defines an undirected edge between u and v. In case of a directed graph this defines a directed edge from u to v. If the problem deals with a weighted graph then each edge is given as `u v w` where u and v are vertices and w is a weight.

It is guaranteed that a given graph is simple. That is, it does not contain self-loops (edges going from a vertex to itself) and parallel edges.

Examples:

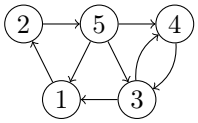
- An undirected graph with four vertices and five edges:

```
4 5
2 1
4 3
1 4
2 4
3 2
```



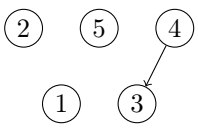
- A directed graph with five vertices and eight edges.

```
5 8
4 3
1 2
3 1
3 4
2 5
5 1
5 4
5 3
```



- A directed graph with five vertices and one edge.

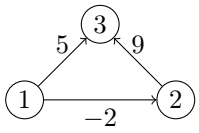
```
5 1
4 3
```



Note that the vertices 1, 2, and 5 are isolated (have no adjacent edges), but they are still present in the graph.

- A weighted directed graph with three vertices and three edges.

```
3 3
2 3 9
1 3 5
1 2 -2
```



1 Computing the Minimum Cost of a Flight

COMPLETE!

Problem Introduction

Now, you are interested in minimizing not the number of segments, but the total cost of a flight. For this you construct a weighted graph: the weight of an edge from one city to another one is the cost of the corresponding flight.

Problem Description

Task. Given an *directed* graph with positive edge weights and with n vertices and m edges as well as two vertices u and v, compute the weight of a shortest path between u and v (that is, the minimum total weight of a path from u to v).

Input Format. A graph is given in the standard format. The next line contains two vertices u and v.

Constraints. $1 \le n \le 10^4$, $0 \le m \le 10^5$, $u \ne v$, $1 \le u, v \le n$, edge weights are non-negative integers not exceeding 10^8.

Output Format. Output the minimum weight of a path from u to v, or -1 if there is no path.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	2	2	3	10	3	4	10	10	6

Memory Limit. 512MB.

Sample 1.

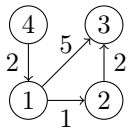
Input:

```
4 4
1 2 1
4 1 2
2 3 2
1 3 5
1 3
```

Output:

```
3
```

Explanation:



There is a unique shortest path from vertex 1 to vertex 3 in this graph ($1 \to 2 \to 3$), and it has weight 3.

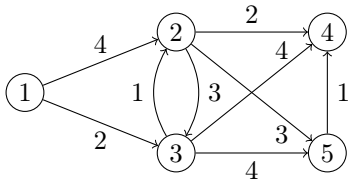
Sample 2.

Input:

```
5 9
1 2 4
1 3 2
2 3 2
3 2 1
2 4 2
3 5 4
5 4 1
2 5 3
3 4 4
1 5
```

Output:

```
6
```



There are two paths from 1 to 5 of total weight 6: $1 \to 3 \to 5$ and $1 \to 3 \to 2 \to 5$.

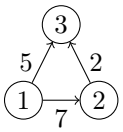
Sample 3.

Input:

```
3 3
1 2 7
1 3 5
2 3 2
3 2
```

Output:

```
-1
```

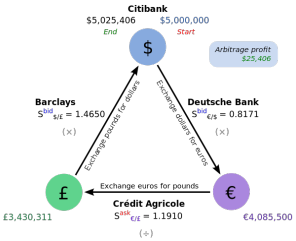


There is no path from 3 to 2.

2 Detecting Anomalies in Currency Exchange Rates GUIDED!

Problem Introduction

You are given a list of currencies $c_1, c_2, \ldots, c_n$ together with a list of exchange rates: r_{ij} is the number of units of currency c_j that one gets for one unit of c_i. You would like to check whether it is possible to start with one unit of some currency, perform a sequence of exchanges, and get more than one unit of the same currency. In other words, you would like to find currencies $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdot r_{i_{k-1},i_k}, r_{i_k,i_1} > 1$. For this, you construct the following graph: vertices are currencies $c_1, c_2, \ldots, c_n$, the weight of an edge from c_i to c_j is equal to $-\log r_{ij}$. There it suffices to check whether is a negative cycle in this graph. Indeed, assume that a cycle $c_i \to c_j \to c_k \to c_i$ has negative weight. This means that $-(\log c_{ij} + \log c_{jk} + \log c_{ki}) < 0$ and hence $\log c_{ij} + \log c_{jk} + \log c_{ki} > 0$. This, in turn, means that



$$r_{ij} r_{jk} r_{ki} = 2^{\log c_{ij}} 2^{\log c_{jk}} 2^{\log c_{ki}} = 2^{\log c_{ij} + \log c_{jk} + \log c_{ki}} > 1.$$

Problem Description

Task. Given an directed graph with possibly negative edge weights and with n vertices and m edges, check whether it contains a cycle of negative weight.

Input Format. A graph is given in the standard format.

Constraints. $1 \le n \le 10^3$, $0 \le m \le 10^4$, edge weights are integers of absolute value at most 10^3.

Output Format. Output 1 if the graph contains a cycle of negative weight and 0 otherwise.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	2	2	3	10	3	4	10	10	6

Memory Limit. 512MB.

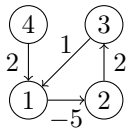
Sample 1.

Input:

```
4 4
1 2 -5
4 1 2
2 3 2
3 1 1
```

Output:

```
1
```



The weight of the cycle $1 \to 2 \to 3$ is equal to -2, that is, negative.

3 Exchanging Money Optimally

Problem Introduction

Now, you would like to compute an optimal way of exchanging the given currency c_i into all other currencies. For this, you find shortest paths from the vertex c_i to all the other vertices.

Problem Description

Task. Given an directed graph with possibly negative edge weights and with n vertices and m edges as well as its vertex s, compute the length of shortest paths from s to all other vertices of the graph.

Input Format. A graph is given in the standard format.

Constraints. $1 \le n \le 10^3$, $0 \le m \le 10^4$, $1 \le s \le n$, edge weights are integers of absolute value at most 10^9.

Output Format. For all vertices i from 1 to n output the following on a separate line:

- "*", if there is no path from s to u;
- "-", if there is a path from s to u, but there is no shortest path from s to u (that is, the distance from s to u is $-\infty$);
- the length of a shortest path otherwise.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	2	2	3	10	3	4	10	10	6

Memory Limit. 512MB.

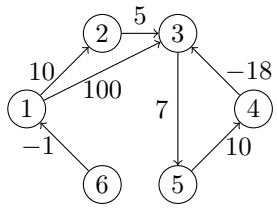
Sample 1.

Input:

```
6 7
1 2 10
2 3 5
1 3 100
3 5 7
5 4 10
4 3 -18
6 1 -1
1
```

Output:

```
0
10
-
-
-
*
```



The first line of the output states that the distance from 1 to 1 is equal to 0. The second one shows that the distance from 1 to 2 is 10 (the corresponding path is $1 \to 2$). The next three lines indicate that the distance from 1 to vertices 3, 4, and 5 is equal to $-\infty$: indeed, one first reaches the vertex 3 through edges $1 \to 2 \to 3$ and then makes the length of a path arbitrary small by making sufficiently many walks through the cycle $3 \to 5 \to 4$ of negative weight. The last line of the output shows that there is no path from 1 to 6 in this graph.

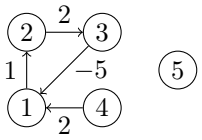
Sample 2.

Input:

```
5 4
1 2 1
4 1 2
2 3 2
3 1 -5
4
```

Output:

```
-
-
-
0
*
```



In this case, the distance from 4 to vertices 1, 2, and 3 is $-\infty$ since there is a negative cycle $1 \to 2 \to 3$ that is reachable from 4. The distance from 4 to 4 is zero. There is no path from 4 to 5.

4 Appendix

4.1 Compiler Flags

C (`gcc 7.4.0`). File extensions: `.c`. Flags:

```
gcc -pipe -O2 -std=c11 <filename> -lm
```

C++ (`g++ 7.4.0`). File extensions: `.cc`, `.cpp`. Flags:

```
g++ -pipe -O2 -std=c++14 <filename> -lm
```

If your `C/C++` compiler does not recognize `-std=c++14` flag, try replacing it with `-std=c++0x` flag or compiling without this flag at all (all starter solutions can be compiled without it). On Linux and MacOS, you most probably have the required compiler. On Windows, you may use your favorite compiler or install, e.g., `cygwin`.

C# (`mono 4.6.2`). File extensions: `.cs`. Flags:

```
mcs
```

Go (`golang 1.13.4`). File extensions: `.go`. Flags

```
go
```

Haskell (`ghc 8.0.2`). File extensions: `.hs`. Flags:

```
ghc -O2
```

Java (`OpenJDK 1.8.0_232`). File extensions: `.java`. Flags:

```
javac -encoding UTF-8
java -Xmx1024m
```

JavaScript (`NodeJS 12.14.0`). File extensions: `.js`. No flags:

```
nodejs
```

Kotlin (`Kotlin 1.3.50`). File extensions: `.kt`. Flags:

```
kotlinc
java -Xmx1024m
```

Python (`CPython 3.6.9`). File extensions: `.py`. No flags:

```
python3
```

Ruby (`Ruby 2.5.1p57`). File extensions: `.rb`.

```
ruby
```

Rust (`Rust 1.37.0`). File extensions: `.rs`.

```
rustc
```

Scala (`Scala 2.12.10`). File extensions: `.scala`.

```
scalac
```

4.2 Frequently Asked Questions

Why My Submission Is Not Graded?

You need to create a submission and upload the *source file* (rather than the executable file) of your solution. Make sure that after uploading the file with your solution you press the blue "Submit" button at the bottom. After that, the grading starts, and the submission being graded is enclosed in an orange rectangle. After the testing is finished, the rectangle disappears, and the results of the testing of all problems are shown.

What Are the Possible Grading Outcomes?

There are only two outcomes: "pass" or "no pass." To pass, your program must return a correct answer on all the test cases we prepared for you, and do so under the time and memory constraints specified in the problem statement. If your solution passes, you get the corresponding feedback "Good job!" and get a point for the problem. Your solution fails if it either crashes, returns an incorrect answer, works for too long, or uses too much memory for some test case. The feedback will contain the index of the first test case on which your solution failed and the total number of test cases in the system. The tests for the problem are numbered from 1 to the total number of test cases for the problem, and the program is always tested on all the tests in the order from the first test to the test with the largest number.

Here are the possible outcomes:

- `Good job!` Hurrah! Your solution passed, and you get a point!
- `Wrong answer.` Your solution outputs incorrect answer for some test case. Check that you consider all the cases correctly, avoid integer overflow, output the required white spaces, output the floating point numbers with the required precision, don't output anything in addition to what you are asked to output in the output specification of the problem statement.
- `Time limit exceeded.` Your solution worked longer than the allowed time limit for some test case. Check again the running time of your implementation. Test your program locally on the test of maximum size specified in the problem statement and check how long it works. Check that your program doesn't wait for some input from the user which makes it to wait forever.
- `Memory limit exceeded.` Your solution used more than the allowed memory limit for some test case. Estimate the amount of memory that your program is going to use in the worst case and check that it does not exceed the memory limit. Check that your data structures fit into the memory limit. Check that you don't create large arrays or lists or vectors consisting of empty arrays or empty strings, since those in some cases still eat up memory. Test your program locally on the tests of maximum size specified in the problem statement and look at its memory consumption in the system.
- `Cannot check answer. Perhaps the output format is wrong.` This happens when you output something different than expected. For example, when you are required to output either "Yes" or "No", but instead output 1 or 0. Or your program has empty output. Or your program outputs not only the correct answer, but also some additional information (please follow the exact output format specified in the problem statement). Maybe your program doesn't output anything, because it crashes.
- `Unknown signal 6 (or 7, or 8, or 11, or some other).` This happens when your program crashes. It can be because of a division by zero, accessing memory outside of the array bounds, using uninitialized variables, overly deep recursion that triggers a stack overflow, sorting with a contradictory comparator, removing elements from an empty data structure, trying to allocate too much memory, and many other reasons. Look at your code and think about all those possibilities. Make sure that you use the same compiler and the same compiler flags as we do.
- `Internal error: exception...` Most probably, you submitted a compiled program instead of a source code.

- `Grading failed.` Something wrong happened with the system. Report this through Coursera or edX Help Center.

May I Post My Solution at the Forum?

Please do not post any solutions at the forum or anywhere on the web, even if a solution does not pass the tests (as in this case you are still revealing parts of a correct solution). Our students follow the Honor Code: "I will not make solutions to homework, quizzes, exams, projects, and other assignments available to anyone else (except to the extent an assignment explicitly permits sharing solutions)."

Do I Learn by Trying to Fix My Solution?

My implementation always fails in the grader, though I already tested and stress tested it a lot. Would not it be better if you gave me a solution to this problem or at least the test cases that you use? I will then be able to fix my code and will learn how to avoid making mistakes. Otherwise, I do not feel that I learn anything from solving this problem. I am just stuck.

First of all, learning from your mistakes is one of the best ways to learn.

The process of trying to invent new test cases that might fail your program is difficult but is often enlightening. Thinking about properties of your program makes you understand what happens inside your program and in the general algorithm you're studying much more.

Also, it is important to be able to find a bug in your implementation without knowing a test case and without having a reference solution, just like in real life. Assume that you designed an application and an annoyed user reports that it crashed. Most probably, the user will not tell you the exact sequence of operations that led to a crash. Moreover, there will be no reference application. Hence, it is important to learn how to find a bug in your implementation yourself, without a magic oracle giving you either a test case that your program fails or a reference solution. We encourage you to use programming assignments in this class as a way of practicing this important skill.

If you have already tested your program on all corner cases you can imagine, constructed a set of manual test cases, applied stress testing, etc, but your program still fails, try to ask for help on the forum. We encourage you to do this by first explaining what kind of corner cases you have already considered (it may happen that by writing such a post you will realize that you missed some corner cases!), and only afterwards asking other learners to give you more ideas for tests cases.