INDU: Trains

Kristoffer Sahlin

Contents

Introduction

In this project, you will write a Python program to simulate trains that run back and forth on different lines—think of the tunnelbana. We will work with a simple discrete ("turn-based") model where adjacent stations are located one unit of time from each other. In each "stage" (unit of time) in the model, each train is located at a station. From one stage to another, two things can happen to a train: either it reaches the next station or stays at the same station (is delayed). The map of train stations can be visualized as a graph, where each node is a station and each edge signifies that two stations are connected by a track (see fig. 1).

This project has two tasks which build on each other. To receive a grade E-C, it is sufficient to complete Task 1. For a higher grade, you must also complete Task 2. See the general instructions in *INDU—Individual Assignment in Programming* for details on grade determination.

Model

We now describe the details of the model. Every train is associated with a line, and there are only rails between stations on the same line. When a train reaches either the northernmost or southernmost station on its line, it turns around and goes back.

At the start of the program, we will generate n trains, each randomly assigned a station and a direction (north or south). If a train is assigned to one of the final stations, its direction is predetermined: for example, a train at the northernmost station will always be heading south. In this model, we allow multiple trains at the same station at the same time (imagine an infinite number of tracks). The number of trains n is therefore independent of the number of stations, and all trains run independently of one another.

Suppose that the train Bobby is on the green line and currently located at station X in Map 1 (fig. 1), traveling southward at time t. There are then two possible states for Bobby at time $t + 1$: either Bobby arrives at station Y, or Bobby stays at station X (is delayed). Because station X has a southern neighbor (Y) that is not a final station, Bobby will maintain a southward heading in both cases.

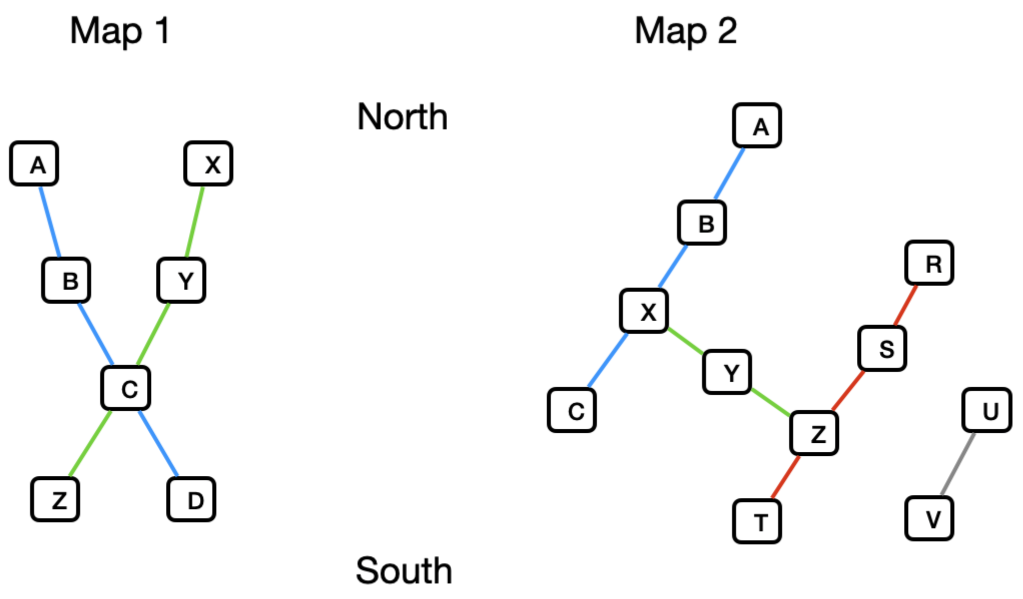

Figure 1: Two maps of possible train lines (with different lines in different colors). Each train runs back and forth on only one line. A station can connect to several lines (for example station C in Map 1 or station X in Map 2).

Now suppose instead that Bobby is on the green line at station C in Map 1 (fig. 1), again traveling southward at time t. Station C has a southern neighbor that *is* a final stop. In this case, the two possibilities for Bobby at time $t + 1$ are more distinctive: either Bobby arrives at the final stop Z and then changes direction from southward to *northward*, or Bobby remains at station C still facing southward.

Each train station is assigned a specific probability $p \in [0, 1]$ of causing a delay. The simulation advances one step at a time. In each round, the user should be able to ask where train k (where k is between 1 and n) is located and in which direction it is traveling.

Input format

The input to the simulation consists of two files. One comma-separated file contains the train stations (column 1) and their probability of delay (column 2). An example for Map 1 in fig. 1 is shown below (you should copy this to a file named `stations.txt`). We can see from this file that the stations on the green line have a greater chance of causing delays and that station C is the worst in this regard.

```
A,0.001
B,0.001
C,0.2
D,0.001
X,0.1
Y,0.1
Z,0.1
```

The other file is also a comma-separated file, this one describing the edges in the railway graph. Each line has 4 entries: a source station (column 1), target station (column 2), line (column 3), and direction (column 4). S stands for "South" and N for "North". (You should copy this over to a file named `connections.txt`).

```
A,B,blue,S
B,C,blue,S
C,D,blue,S
```

```
X,Y,green,S
Y,C,green,S
C,Z,green,S
```

The first line in the file says that:

i. there is a railway between *A* and *B*;

ii. this railway is part of the blue line;

iii. to go from *A* to *B* means traveling southward.

Obs: the program should work for any input files in this format, not only the examples we have given here. Feel free to create your own rail network and include it with your submission.

Task 1 (E-C)

At each stage of the model, the program should ask the user to make a choice, where the choices are to:

1 Simulate one unit of time forward in the model without showing any information.

2 Display the current position of a train.

q Quit.

Sample execution

Here we give an example with both delays and a change of direction. The program is expected to behave in approximately the following way:

```
Enter name of stations file: stations.txt
Enter name of connections file: connections.txt
Enter how many trains to simulate: 3

Continue simulation [1], train info [2], exit [q].
Select an option: 2
Which train [1 - 3]: 1

Train 1 on GREEN line is at station C heading in North direction

Continue simulation [1], train info [2], exit [q].
Select an option: 1

Continue simulation [1], train info [2], exit [q].
Select an option: 2
Which train [1 - 3]: 1

Train 1 on GREEN line is at station C heading in North direction (DELAY)

Continue simulation [1], train info [2], exit [q].
Select an option: 1

Continue simulation [1],train info [2], exit [q].
Select an option: 2
Which train [1 - 3]: 1

Train 1 on GREEN line is at station Y heading in North direction.
```

```
Continue simulation [1], train info [2], exit [q].
Select an option: 1

Continue simulation [1], train info [2], exit [q].
Select an option: 2
Which train [1 - 3]: 1

Train 1 on GREEN line is at station X heading in South direction.

Continue simulation [1], train info [2], exit [q].
Select an option: q
Thank you and goodbye!
```

Requirements for Task 1

- Correct implementation of the model. For higher grades, you must also satisfy the requirements described in the general instructions in *INDU—Individual Assignment in Programming*.
- **Obs:** the solution must work for maps and parameters other than those shown in the examples above.
- The task should be completed without importing nonstandard libraries (that is, any libraries that must be manually installed). Ask the instructor if you are unsure if you can use a specific library. Describe all the libraries you use in your project report.
- You do not need to anticipate *all* possible situations, for example every case of input data (there are many), but you should show that you have considered *some* in order to get points for good error handling. There are likewise many edge cases of input maps or parameters; show that you have considered some of these in order to get points for error handling and testing.

Tips

1. Use the standard library `random` both to randomly choose the positions of trains and to decide whether delays occur.
2. Consider the structure of your code before you begin programming. Should you have any specific classes? What functions will you write? Remember to separate interaction from calculation; this will greatly simplify debugging and lead to better and more easily testable code.

Task 2 (E-A)

We now extend the program from Task 1. The program should now be able to answer whether it is theoretically possible to get from station p to station q within T time units. The program should now present the following choices to the user:

1 Simulate one unit of time forward in the model without showing any information.

2 Display the current position of a train.

3 Answer if we can reach station q from station p within T time units (given our map).

q Quit.

Alternatives 1 and 2 are from Task 1, and implementing alternative 3 is Task 2. You should not include delays or the trains' current locations in the calculation for alternative 3; you should only determine whether station q lies at most T stations away in the map from station p. If there is no sequence of edges connecting the two stations, then it is impossible to travel between them regardless of T. (For example, it is impossible to get from station U or V to any other station in Map 2 in fig. 1).

Sample execution

We show here an example of usage of alternative 3.

```
Enter name of stations file: stations.txt
Enter name of connections file: connections.txt
Enter how many trains to simulate: 5

Continue simulation [1], train info [2], route info [3], exit [q].
Select an option: 3
Select a start station: A
Select an end station: X
Select timesteps: 3

Station X is not reachable from station A within 3 timesteps.

Continue simulation [1], train info [2], route info [3], exit [q].
Select an option: 3
Select a start station: A
Select an end station: Z
Select timesteps: 3

Station Z is reachable from station A within 3 timesteps.

Continue simulation [1], train info [2], route info [3], exit [q].
Select an option: q
Thank you and goodbye!
```

Requirements for Task 2

- You must have completed Task 1; all requirements for Task 1 apply also to Task 2.
- Even if you complete both tasks, you should only hand in one program. That is, you should extend the functionality of your program from Task 1 for Task 2. It is, however, a good idea to save a copy of your solution to Task 1 before you try implementing Task 2.

You should also use classes and objects in your program to a meaningful extent. You can solve the task without using object-oriented programming, but one of the grade-raising criteria **requires** that you organize your code in an object-oriented way, with the majority of your code inside of class definitions (see the general instructions).

Grading

Details can be found with the general instructions in *INDU—Individual Assignment in Programming*.