Assignment 1: Find the Best Train Connections

AI1SysProj 2021

Topic:	Search
Due on:	December 15, 2021
Version from:	November 1, 2021

1 Task Summary

Using a data set of Indian railway times, find the best connection between any two places according to different cost functions. Concretely, you have to submit solutions for the connection problems listed in `problems.csv`. The the necessary files are at `https://gitlab.rrze.fau.de/wrv/AISysProj/ws2122/search/assignment`.

2 The Data Set

The train schedule is specified in the file `schedule.csv`. We use a slightly modified variant of a data set from Kaggle[1]. It is a CSV file with 12 columns but only the following columns are relevant for us:

- **`Train No.`**: an identifier for the train.
- **`islno`**: what stop of the train is described (e.g. the fifth stop).
- **`station Code`**: an identifier for the train station.
- **`Arrival time`**: the arrival time at that stop.
- **`Departure time`**: the departure time at that stop.
- **`Distance`**: the total distance travelled until that stop (i.e. since the stop where `islno` is 1).

For example, let us take a look at the following two entries:

Train No.	islno	station Code	Arrival time	Departure time	Distance
04407	10	GD	23:30:00	23:35:00	536
04407	11	LKO	02:25:00	02:35:00	653

From this we learn that the 10th stop of train `04407` is Gonda Jn (`GD`) and that the next (11th) stop is Lucknow Nr (`LKO`). We can also see that the train travels $653 - 536 = 117$

[1] `https://www.kaggle.com/harsh16/indian-railways-time-table-for-trains-available`

kilometers from `GD` to `LKO`, which takes 2 hours and 50 minutes. Note that `02:25:00` must refer to the next day. In general we will always need to "add a day" if the arrival time at stop n is smaller than the departure time at stop $n-1$. We will do the same for the departure time at any stop if it is smaller than the arrival time, which is relevant for some of the cost functions.

3 Problems and Solutions

Aside from the schedule data, you have a file `problems.csv` that contains the connection problems you have to solve. We also provide example problems (`example-problems.csv`) and solutions (`example-solutions.csv`), which you can use for comparison. A problem file has six columns:

- **`ProblemNo`**: the number of the problem.
- **`Difficulty`**: the difficulty of the problem (you might want to start with the easier ones).
- **`From`**: where the connection should start.
- **`To`**: where the connection should end.
- **`StartTime`**: when the person wants to start travelling (relevant for the `arrivaltime` cost function).
- **`CostFunction`**: the cost function (Section 3.2).

You may assume that `From` is different from `To` and that `To` is indeed reachable from `From`. A solution file has three columns:

- **`ProblemNo`**: the number of the problem solved.
- **`Connection`**: a best connection (usually not unique). The format is described in Section 3.1.
- **`Cost`**: the cost of the solution according to the cost function.

3.1 Connection Format

The train connections have to be specified in a particular format. As an example, we will take a look at the following connection:

```
56502 :  57 -> 58 ; 57305 :  69 -> 72
```

`x : y -> z` means that we take train `x` from stop `y` (`islno`) until stop `z`. Semicolons separate trains taken consecutively. So in the example, we would first take train `56502`

Train No.	islno	station Code	Arrival time	Departure time	Distance
56502	57	SYM	14:34:00	14:35:00	588
56502	58	VLE	14:44:00	14:45:00	596
...					
57305	69	VLE	05:16:00	05:17:00	559
57305	70	SAB	05:22:00	05:23:00	562
57305	71	MUK	05:32:00	05:33:00	570
57305	72	NRT	06:00:00	06:02:00	578

Table 1: Relevant schedule data for connection `56502 : 57 -> 58 ; 57305 : 69 -> 72`.

from stop `57` to stop `58` and then continue with train `57305` on stop `69` until stop `72`. This obviously requires that stop `58` of train `56502` is the same station as stop `69` of train `57305`. Comparing with Table 1, we can see that it is a valid connection from `SYM` to `NRT`, with a change at `VLE`.

3.2 Cost Functions

In this section, discuss the different cost functions, re-using the example connection

`56502 : 57 -> 58 ; 57305 : 69 -> 72`

and the schedule data from Table 1.

`stops` The cost `stops` is the number of times we enter a station by train. In the example, we would enter the stations `VLE`, `SAB`, `MUK`, `NRT` (i.e. we don't count the station we started from). The cost is thus `4`.

`distance` The total distance travelled by train. In the example, it would be $(596 - 588) + (578 - 559) = 27$.

`duration` The total time in seconds between departure and arrival. In the example, we would leave at `14:35:00` and arrive at `06:00:00` the following day. Therefore, the cost would be `55500`. Note that we assume it takes 0 seconds to change trains, i.e. if the first train arrives at the same time as the second one leaves, we assume you don't have to wait until the next day.

`arrivaltime` The arrival time (including days). Here, the `StartTime` specified in the problem matters. If we use the example connection with the `StartTime 11:30:00`, we would arrive at `06:00:00` on the next day. The arrival time should thus be specified as `01:06:00:00`. But if we use the `StartTime 15:24:00`, we would miss the train and the arrival time would be one day later (`02:06:00:00`).

`changes` The number of train changes. It corresponds to the number of semicolons in the solution (if you put as few as possible). For our example it would be `1`.

`price` The ticket price. We assume that you have to pay `100` for each train you enter plus `1` for every kilometer travelled in that train. For our example connection, we would thus pay `108` for the first train and `119` for the second train, which means the total cost would be `227`.

`linear ...` A linear combination of the cost functions mentioned above. It has the following form

```
linearcost ::= "linear ", part, { " + ", part } ;
part       ::= integer, " * ", basiccost ;
basiccost  ::= "stops" | "distance" | "duration" |
               "arrivaltime" | "changes" | "price" ;
```

For example, the cost

```
linear 100 * changes + 1 * distance
```

would be $100 \cdot 1 + 1 \cdot 27 = 127$ for our example connection. The `arrivaltime` is here defined as the number of seconds between the `StartTime` and the time of arrival.

4 Submission

At the deadline, we will download a snapshot of your repository. It should contain

1. all your code,
2. a readme file explaining how to run your code (in particular how to use it to solve a problems file),
3. a file `solutions.csv` that lists your solutions for the problems specified in `problems.csv` in the format specified in Section 3.

5 Random Tips

- Make sure you get the handling of times right (for `duration` and `arrivaltime`), possibly by implementing tests.
- Solve the problems with lower difficulty before moving to the harder ones.
- Problems with difficulties 1–2 can be solved with a naive implementation of uniform cost search. Difficulties 3 and higher will require some optimization. In particular, you can use Dijkstra's algorithm.
- If debugging becomes too difficult, you could consider creating test cases and/or making a small test schedule.
- You will notice that efficiency matters for this problem. Nevertheless it is not necessary to use a fast programming language (e.g. everything can be solved in Python).
- If you have problems getting started, you might try to implement code that answers the following questions first, which should get you a bit more comfortable with the data set:
  1. How many different stations are there?
  2. What station is used by the fewest trains?
  3. For a station X, what stations can be reached by taking a train and travelling for exactly 1 stop?
  4. How can you build a directed graph from the information of the previous question?
  5. For any pair of stations (X, Y), is there any way to travel from X to Y?
  6. How can you add time information and distance information to the directed graph?