Homework assignment 2
Transport schedule

Main information

Deadline: June 3, 23:59
Submission: through the Canvas LMS as a **single** zip archive.

General requirements

- Use proper coding style, follow PEP-8 guidelines. Strict PEP-8 validation like the one in PyJudge won't be performed. However, if the code becomes hard to read because of multiple PEP-8 violations, your grade can be lowered
- Use constants instead of "magic numbers"
- Use functions (and optionally modules) to improve structure of your program, don't put all logic in the main program code
- Add protection against possible program crashes that come as a result of incorrect user actions, e.g. entering a non-valid number and others
- Make sure you submit your own work. All solutions will be checked for plagiarism by an automated service

Introduction

Technological development has significantly changed the way modern cities operate. Taxi ordering through a mobile app, car sharing, hotspots for internet access, live webcams, smart lighting - these being just a few examples. One of the big changes came in the area of public transport.

In this assignment, you will need to come up with a model of a simplified transport scheduling system and develop an algorithm for printing real-time departures at different locations. We will apply the following restrictions:

- Only one type of overground transport is considered (e.g. bus, tram) - you can choose any kind you want.
- There is no tracking of actual transport movement, the system only takes a fixed schedule into account (no delays, cancellations, breakdowns and similar issues)
- The time between any two stops/stations on the route is the same in both directions.
- On all routes, traffic starts and ends simultaneously on both ends. The interval between subsequent departures is constant and does not change throughout the day.

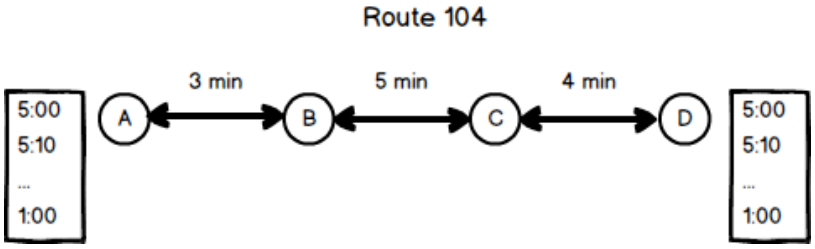
Description of tasks

You need to store the following information about the public transport schedule:

- Names of stops/stations

- Routes. A single route is formed of a sequence of stops/stations. For each route you need to store the time of first and last departures from each end stop/station (terminus) and the interval between buses/trams/trains in minutes. You also need to store the time it takes to reach each of the stations forming the route starting from the terminus.

  For example, consider an abstract bus route 104, which follows 4 stops: A,B,C,D (see picture below). Assume that the bus traffic on this particular route starts at 5:00 in the morning from both A and D and follows a 10 minute interval until 01:00 AM at night. Having interstop travel times as shown on the figure, means that station B will have buses arriving from A at 5:03, 5:13, 5:23, etc., the last bus coming at 01:03 at night. At the same time, buses will arrive at B from C at 5:09, 5:19, 5:29, etc.



  In reality, a single stop/station may have several platforms for different destinations (e.g. for bus transport, very likely there will be two stops on different sides of the same road), but for simplicity we will combine traffic for all such platforms.

  The same stop/station may be a part of several routes (will be shown in another example further down).

  Your main task in this assignment is to come up with an algorithm, which, given the current time and a station, prints a schedule of the nearest departures from it.

  The efficiency of the algorithm will greatly depend on the data structure you decide to utilize. There may even be several independent data structures.

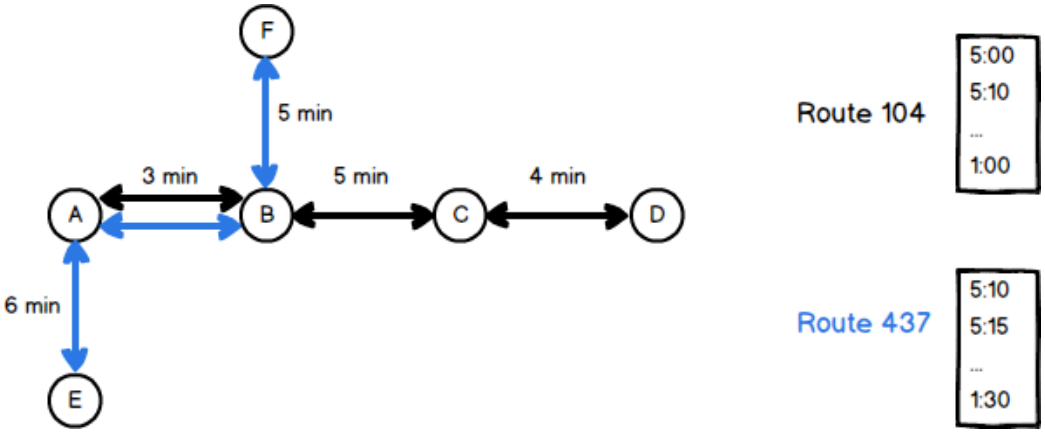
  The information needs to be stored in any text format and loaded from the file on program startup. You will work with the file in read-only mode in this assignment, the information can be initially saved using any text editor.

Completing the assignment

Create a new Python application.

- (2 points) Choose the data structure(s) to store the all the required information. You may not come with the right model straightaway. It is perfectly fine if you change it as you develop the main algorithm.
- (4 points) Implement the main algorithm, which accepts a station name and returns a list of the nearest buses/trams/trains of all routes coming through the station. To test the algorithm, you can first create a sample dataset in the program code itself (hardcode your data).
- (3 points) Create a text file that stores all the required information about the transport schedule. Your program needs to read all the information from this file on startup. After implementing this step remove all the hardcoded data.
- (1 points) Implement the user interaction workflow. It should be an infinite loop, in which the user enters a station name and the program prints information about the next departures within the next 10 minutes. The program should terminate when the user inputs an empty string.

Program flow example



Here two bus routes are given. The first one was already described above. The second one (437) is formed of stations E-A-B-F, thus stations A and B and the segment between them are shared between the two routes. Note that the second route has a different schedule (buses starting from 5:10AM with a 5 minute interval until 01:30AM).

In the program flow below, user input is shown in blue:

```
Enter station: B
Current time: 14:50
Schedule:
437, destination E, 0 min
104, destination D, 3 min
437, destination F, 4 min
104, destination A, 9 min
Enter station: A
Current time: 14:52
Schedule:
437, destination E, 1 min
437, destination F, 4 min
104, destination D, 8 min
Enter station: H
No such station!
```

Hints

- Use the **datetime** module to get the current date and time and for other operations, related to dates and times
- You may find it easier to store time as a single integer calculated as hour*60+minute, e.g. 14:53 is equal to 893, so that any time within a day falls in the range of [0..1439]