2016/7 ICTSA Programming Challenge

PERCEPTION

Version 0.107

Dr Christian Colombo, Dr Jean-Paul Ebejer,
Prof. Gordon J. Pace, Dr Chris Porter

February 13, 2017

This document outlines the programming challenge task to be solved in the 2016-2017 ICTSA Programming Challenge (colloquially referred to as *Game of Codes*). The competition opens on Friday, 17th February 2017 at 20:00 and closes on Sunday, 19th February 2017 at 20:00 ZST[1]. This document also specifies the criteria that will be used to judge the competition's entries.

1 Background

Mobile phone alarms are a thing of the past[2]. Since *Perception* released *Extraphic*, their new app, no one in Malta has set his or her alarm as they used to do before. Prior to going to bed, you make sure that the app has been set to *Sleep-As-Much-As-I-Can* mode. The app checks the time of your first meeting, checks the predicted traffic situation the following morning, and wakes you up to give just 20 minutes for a quick shower and breakfast before you leave home along the route the app selected. Other modes include the *Drive-As-Little-As-Possible* and the *Suggest-A-Route-Which-Gives-Me-An-Excuse-To-Arrive-15-Minutes-Late*. Many tried to implement this before, but the complexity of reasoning about traffic which increases and decreases at different times of the day was beyond their simple *if* statements. *Perception* have cracked the problem, though, and their app *Extraphic* even recommends places where to stop for a coffee along the way when waiting out traffic pays off better than staying on the road in a rage.

[1] Zeus Standard Time – the time as set on UNIX server zeus, based at the Department of Computer Science at the University of Malta. This time can be seen on the Programming Challenge website at `http://www.cs.um.edu.mt/svrg/56ayl4778abtds377/Game_of_Codes/`

[2] And thank god for that!

This is how you pitch your idea to the potential investors. You are the CEO of *Perception*. Actually, you are also the CTO, CIO, secretary and janitor. You are *Perception*. The lead-investor, a middle-aged balding man in a boring grey suit, and a small stain on his purple-striped yellow tie stands up to talk.

"I like the idea. But I don't like you. You don't look nerdy enough to convince me you will make this work. But I will give you a chance. If you can present a working version of the app in 48 hours, I will put my money behind your company. Call my secretary when you are done."

He stands up and leaves the room, with the rest of the entourage following him in an orderly line. The last one to leave sneers at you, switches off the light and closes the door behind him. Bright red letters reading *Perception* on the last slide of your presentation light up the room.

"Let's get to work" you tell yourself.

2 The Task

Your task is to write the core *Drive-As-Little-As-Possible* algorithm for the *Extraphic* app. Your program will be given a map, consisting of a number of named localities (or road-junctions), and a table for each road listing how long driving along that road takes at different times of the day. Some localities are marked as having a café serving a decent espresso and croissant. Your program will also be given where you will start off the journey and where you want to drive to and by what time you want to arrive there. Your program is to output a route which minimizes the journey time required to drive from the point of origin to the destination, possibly including coffee-stops on the way.

2.1 Input

Your program executable will take the following two parameters as input (in this order):

1. **`roads_file.rds`** - The path to the roads file, in `.rds` format. This format is described in Section 2.1.1.
2. **`route_file.rt`** - The path to the output route file, in `.rt` format which contains driving instructions (including coffee-stops). This format is described in Section 2.2.

You will supply a batch file which calls your program. This file will be named `journey.bat`. In this file you should make sure to set any extra parameters you may need (e.g. setting `Xmx` or `Xms` sizes for the Java virtual machine).

2.1.1 `.rds` Roads File Format

A roads (`.rds`) file is a text file with the following format:

1. **Line 1 -** A comma-separated list of localities (or road-junction) names, with each locality name made up of alphabetic characters only. In addition, names may start with the asterisk (*) symbol – localities with names which start with an asterisk are taken to be places where you can stop for a coffee (the asterisk is part of the name). You may assume that the line will contain no other symbols (*e.g.* no space between names). The localities are not listed in any particular order. Note that the names of the localities are case-sensitive and that no locality may be named `COFFEE`.

   E.g. `*Zejtun,RahalGdid,Isla,*Sliema`

2. **Line 2 -** Will contain the name of the starting locality.

   E.g. `*Zejtun`

3. **Line 3 -** Will contain the name of the destination locality (*i.e.* where you want to go).

   E.g. `RahalGdid`

4. **Line 4 -** Gives the number of time-units indicating the time by when you want to arrive at your destination. This is defined as the *Latest-Arrival-Time*.

   E.g. `20`

5. **Lines 5 (onwards) -** The time taken to drive between localities (the ones joined by a road-segment). Each line consists of comma separated values: (i) the name of the locality the road-segment starts from; (ii) the name of the locality the road-segment ends up in; (iii) a sequence of comma-separated positive integers which indicate the cost (in terms of time-units) of travelling at that specific time. The number of time-units specified will be equal to the *Latest-Arrival-Time* specified above. Note that this is a zero-based indexed list, *i.e.* time-units are denoted using $t_0, t_1, t_2, \ldots$

E.g. `*Zejtun,Isla,1,1,2,5,6,7,4,2,1,1,1,2,4,5,3,2,1,1,1,2`

This boxed example indicates that, for instance, driving along the road-segment from `*Zejtun` to `Isla` takes 2 time-units if you start at time t_2, thus arriving at `Isla` at time t_4, but 7 time-units if you start at time t_5, therefore arriving at time t_{12}. Note that there are enough time-units to cover the *Latest-Arrival-Time*.

If no `.rds` file line exists starting with a particular pair of localities, it is to be taken that there is no road-segment between the two localities, and one cannot drive directly from one to the other. Note that the relationship between the two edges is directional, *e.g.* `(*Zejtun,Isla)` $\neq$ `(Isla,*Zejtun)`.

2.2 Output

The output of the program is a route text file (with extension `.rt`) where each line is either the name of a locality (as found in the first line of an `.rds` file), or a wait instruction of the form: `COFFEE n`, where n is a positive integer indicating how long you will stay at the current location sipping a coffee.

The constraints on the route file are as follows:

- Each line in the route file which specifies a locality, may be interpreted as *drive-to that locality*.
- The output may start with a `COFFEE` instruction, and must end at the destination (specified in the roads file). Note that the starting locality is not required in the route file. It is assumed that you are at the starting locality defined in the second line of the roads file.
- Subsequent localities must have a line in the input file indicating a road-segment between the two localities.
- `COFFEE` instructions may only be used when at a coffee-break locality (one whose name starts with an asterisk) or at the beginning of the journey (you can always have a coffee before you leave home).
- A `COFFEE` instruction may **not** be followed by other `COFFEE` instructions.
- For `COFFEE` instructions, $n \in \mathbb{Z}^+$.
- The total arrival time of the solution must **not** exceed the *Latest-Arrival-Time* (defined in the fourth line of the input `.rds` file).

An example of a route file is shown in Listing 2.

2.3 Calculating your Journey Time

Time-units are counted from the time you leave the starting locality (home) to the time you arrive at your destination. Note that initial coffees at the starting/home locality do not count towards your travelling time. On the other hand, coffees at intermediate destinations are counted as part of the journey. The aim of your program is to minimize the *journey time*. Also, you may not exceed the *Latest-Arrival-Time* specified in the fourth line of the input roads (.rds) file.

2.4 Sample Problem

This section describes an example of the problem definition and of the program output you are required to generate. Listing 1 presents a typical roads text file (.rds). This will serve as input to your program.

Listing 1: problem.rds roads file.

```
*Zebbug,SanGwann,Paceville,*Qormi
*Zebbug,Paceville
10
*Zebbug,*Qormi,4,1,2,8,7,3,5,6,15,17
*Qormi,Paceville,1,8,6,6,3,1,3,4,5,7
*Zebbug,SanGwann,4,4,3,4,1,6,8,9,3,10
SanGwann,Paceville,1,1,2,4,5,5,6,7,7,1
```

The problem defined in Listing 1 gives rise to the road network graph described in Figure 1.

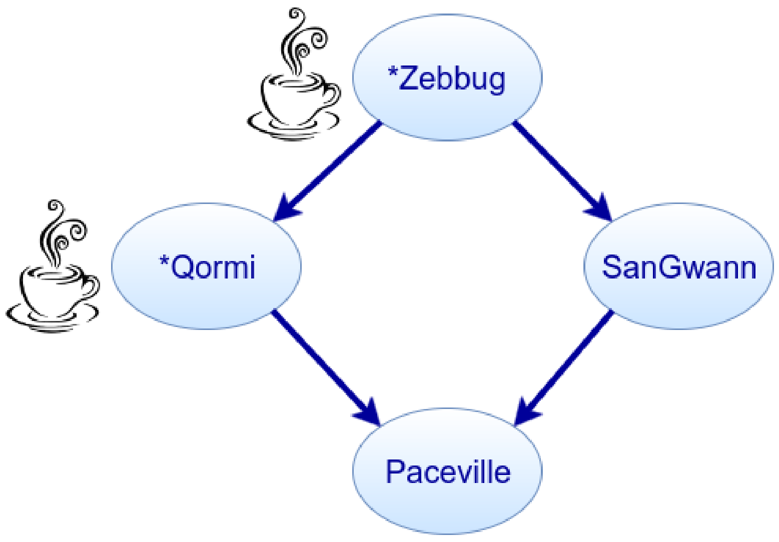

Figure 1: A network with four cities as described in problem.rds.

A possible program output file is shown in Listing 2.

Listing 2: `output1.rt` route file.

```
COFFEE 2
*Qormi
COFFEE 1
Paceville
```

The score for the route defined in Listing 2 is **5** (time-units) and is computed as shown in Table 1.

Table 1: `*Zebbug`, `*Qormi`, and `Paceville` route. Brown cells denote coffee-stops, green cells denote leaving time and grey cells denote travelling time. The number in each cell indicates how many time-units it takes to reach the destination if we left at that time-point. This route gives a score of 5.

Start	Destination	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
`*Zebbug`	`*Qormi`	4	1	2	8	7	3	5	6	15	17
`*Qormi`	`Paceville`	1	8	6	6	3	1	3	4	5	7

You have two coffees in `*Zebbug` at t_0 and t_1. My, you are going to get some caffeine jitters! Note that these time-units do not count for your final journey time (since you are sipping coffee at home in your hello-kitty pyjamas). The starting locality (`*Zebbug`) is implicit and is not defined in the output route file. Leaving at t_2 from `*Zebbug` will take 2 time-units – t_3 and t_4 – to arrive to `*Qormi`, therefore getting there at t_4. We start counting the journey time specified at a particular time-unit always from the next one. If you leave `*Qormi` immediately (at t_4) it would take 3 time-units to get to `Paceville`. So you decide to stop for 1 time-unit for a coffee (at t_4, which now counts towards your journey time) and leave at t_5 when it only takes 1 time-unit to get to `Paceville`. You arrive at the club (at your destination, `Paceville`) at t_6, thus your journey takes 5 time-units, from t_2 to t_6 (both inclusive). This is within the defined *Latest-Arrival-Time* of 10, specified in Listing 1. This is a valid solution, since with 5 time-units of journey time you have not exceeded the *Latest-Arrival-Time*, making it to work on time. Note that this route is not necessarily the optimal solution.

Another possible (worse performing) route file is shown in Listing 3. The score for the route defined in Listing 3 is **10** (time-units) and is computed as shown in Table 2.

Listing 3: `output2.rt` route file.

```
SanGwann
Paceville
```

Leaving immediately (t_0) from `*Zebbug` takes 4 time-units to drive to `SanGwann`. You get to `SanGwann` at t_4, and decide to leave immediately (drive-through) for `Paceville`. This last segment of the journey takes 5 time-units, arriving at the destination at time t_9. The total time of the journey is of 10 time-units (from t_0 to t_9, inclusive), and is therefore worse than the previous route example shown in Listing 2. This is also a valid route, since you have not exceeded the *Latest-Arrival-Time* of 10.

Table 2: `*Zebbug`, `SanGwann`, and `Paceville` route. Green cells denote leaving time and grey cells denote travelling time. This route gives a score of 10.

Start	Destination	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
`*Zebbug`	`SanGwann`	4	4	3	4	1	6	8	9	3	10
`SanGwann`	`Paceville`	1	1	2	4	5	5	6	7	7	1

3 Judging Criteria

The judging panel will run all program submissions on a number of problem `.rds` files. Each program will be given up to 1 minute to generate an output route file (`.rt`). Programs not terminating on a particular problem within 1 minute, or producing wrong output (*e.g.* unknown localities or journey times exceeding *Latest-Arrival-Time*) will get no points for that problem. For each problem instance, the programs will be ranked by journey time (the shorter, the better) defined by the output route file. Points are then given to each team according to their ranking: best 50 points, second 30 points, third 20 points, fourth 10 points and fifth 5 points. In the case of ties, the least number of localities will be used as a tie-breaker (*e.g.* less wear and tear on the car). In effect, this is equal to the number of non-`COFFEE` driving instructions in your route file. In case of further ties, the journey with most time-units spent drinking (Irish) coffees will be used as a tie-breaker. We would rather spend time in a coffee-shop reading, than behind a steering wheel uttering expletives while stuck in traffic. If this is insufficient to break a tie, the teams involved in the tie get the same number of points.

4 Submission Rules

All submissions have to be in the form of a batch file (i.e. `journey.bat`) which calls your program executable. The batch file has to accept two parameters as described in Section 2.1.

No new windows or graphical user interfaces are to be opened by your program. The program will be executed in a virtual machine on an Intel-based PC and having 4GB of RAM. The supported 64-bit operating system will be Microsoft Windows 10. This will have the latest versions of the .NET Framework and Java installed. Specific language runtime libraries which are required have to be free and bundled with your solution. It is in your interest to ensure your solution is straightforward to execute – include documentation where necessary. Any submissions that fail to execute on the aforementioned environment or any submission that takes longer than 1 minute to output a result **and terminate** will **not** be considered by the judging panel.

Each registered team is allowed to submit as many solutions as it wishes. The last submission on Sunday, 19th February 2017 at 20:00 ZST will be considered for the awards. After this deadline no corrections or resubmissions will be allowed. Submissions can only be done through the programming challenge website – note that the program must finish uploading before the aforementioned time.