COMP9024 19T3

Assignment
Sydney Transport Planner

Data Structures and Algorithms

Change Log

We may make minor changes to the spec to address/clarify some outstanding issues. These may require minimal changes in your design/code, if at all. Students are strongly encouraged to check the change log regularly.

Version 1: Released on 30 October 2019

Objectives

The assignment aims to give you more independent, self-directed practice with

- advanced data structures, especially graphs
- graph algorithms
- asymptotic runtime analysis

Admin

Marks	3 marks for stage 1 (correctness) 5 marks for stage 2 (correctness) 2 marks for stage 3 (correctness) 1 marks for complexity analysis 1 mark for style ——————— Total: 12 marks
Due	11:00:00am on **Tuesday** 19 November (week 10)
Late	2 marks (16.67%) off the ceiling per day late (e.g. if you are 25 hours late, your maximum possible mark is 8)

Aim

Your task is to write a program `myTrain.c` for finding an optimal train connection that takes into account a given arrival time and user preferences.

Input

Station names

Your program should start by prompting the user to input a positive number *n* followed by *n* lines, each containing the name of a train station. An example is:

```
prompt$ ./myTrain
Enter the number of stations: 3
Central
Parramatta
Chatswood
```

You may assume that:

- The input is syntactically correct.
- Station names require no more than 31 characters.

- No station name will be input more than once.

Hint:

To read a single line with a station name you may use:

```
scanf("%s", station);   // station is a character array (= string variable)
```

Trains

Next, your program should ask the user for the number *m* of trains running during a day, followed by *m* schedules. Each schedule requires the number of stops, *k*≥2, followed by *k*2* lines of the form:

```
hhmm
station-name
```

meaning that passengers can get on or off the train at that time (`hh` – hour, `mm` – minute) at that station. An example is:

```
Enter the number of trains: 2
Enter the number of stops: 3
0935
Parramatta
1002
Central
1030
Chatswood
Enter the number of stops: 2
1010
Central
1025
Chatswood
```

You may assume that:

- The input is syntactically correct: 4 digits followed by a single space followed by the name of a station.
- All times are valid and range from 0000 to 2359.
- There are no overnight trains: Each train will reach its final stop before midnight.
- Only valid station names that have been input before will be used.

Queries

After reading the network, your program should prompt the user to search for a connection:

```
From: Parramatta
To: Chatswood
Arrive by: 1230
```

Again you may assume that the input is correct: Only stations that have been entered before are used and a valid time is given.

If the user inputs "`done`," then your program should terminate:

```
From: done
Thank you for using myTrain.
prompt$
```

Stage 1 (3 marks)

For stage 1, you should demonstrate that you can read the input and generate a suitable data structure.

For this stage, all test cases will only use queries (`From`, `To`, `ArriveBy`) for which

- there is only one train between `From` and `To` ; and
- this train is guaranteed to arrive on, or before, the requested time, `ArriveBy` .

Hence, all you need to do for this stage is find and output this train connection, including all stops along the way and the arrival/departure times. Here is an example to show the desired behaviour of your program for a stage 1 test:

```
prompt$ ./myTrain
Enter the number of stations: 7
Burwood
Central
Chatswood
Hornsby
Parramatta
Redfern
Wynyard
Enter the number of trains: 2
Enter the number of stops: 5
0935
Parramatta
0950
Burwood
1005
Redfern
1012
Central
1017
Wynyard
Enter the number of stops: 2
1459
Chatswood
1530
Hornsby

From: Chatswood
To: Hornsby
Arrive by: 1700

1459 Chatswood
1530 Hornsby

From: Burwood
To: Central
Arrive by: 1015

0950 Burwood
1005 Redfern
1012 Central

From: done
Thank you for using myTrain.
prompt$
```

Stage 2 (5 marks)

For stage 2, you should extend your program for stage 1 such that it always finds, and outputs, a connection between `From` and `To` that

- arrives on, or before, the requested time `ArriveBy` ; **and**
- departs as late as possible.

You should assume that:

- Changing trains takes no time: Passengers arriving at a station can get onto any other train that leaves that station at the same time or later.
- In all test scenarios there will be at most one connection that satisfies all requirements.

If there is no connection, the output should be:

```
No connection found.
```

Here is an example to show the desired behaviour and output of your program for a stage 2 test:

```
prompt$ ./myTrain
Enter the number of stations: 5
Parramatta
Burwood
Ashfield
Central
Chatswood
Enter the number of trains: 2
Enter the number of stops: 4
0935
Parramatta
0945
Burwood
1002
Central
1030
Chatswood
Enter the number of stops: 3
0940
Parramatta
0950
Ashfield
1000
Central

From: Parramatta
To: Chatswood
Arrive by: 1030

0940 Parramatta
0950 Ashfield
1000 Central
Change at Central
1002 Central
1030 Chatswood

From: Parramatta
To: Central
Arrive by: 0959

No connection found.

From: done
Thank you for using myTrain.
```

Stage 3 (2 marks)

For stage 2, you should extend your program for stage 2 such that:

> *If there are two or more connections with the same latest departure time, choose the one with the **earliest** arrival time.*

Here is an example to show the desired behaviour and output of your program for a stage 3 test:

```
prompt$ ./myTrain
Enter the number of stations: 3
Central
Parramatta
Chatswood
Enter the number of trains: 2
Enter the number of stops: 3
0935
Parramatta
1002
Central
1030
Chatswood
Enter the number of stops: 2
1010
Central
1025
Chatswood

From: Parramatta
To: Chatswood
Arrive by: 1030

0935 Parramatta
1002 Central
Change at Central
1010 Central
1025 Chatswood

From: done
Thank you for using myTrain.
```

Complexity Analysis (1 mark)

Your program should include a time complexity analysis for the worst-case asymptotic running time of your program, in Big-Oh notation, depending on the size of the input:

1. the number of stations, *n*
2. the number of trains, *m*
3. the maximum number *k* of stops on a train line.

Hints

If you find any of the following ADTs from the lectures useful, then you can, and indeed are encouraged to, use them with your program:

- linked list ADT : `list.h`, `list.c`
- stack ADT : `stack.h`, `stack.c`
- queue ADT : `queue.h`, `queue.c`
- graph ADT : `Graph.h`, `Graph.c`
- weighted graph ADT : `WGraph.h`, `WGraph.c`

You are free to modify any of the five ADTs for the purpose of the assignment (*but without changing the file names*). If your program is using one or more of these ADTs, you should submit both the header and implementation file, even if you have not changed them.

Your main program file `myTrain.c` should start with a comment: `/* ... */` that contains the time complexity of your program in Big-Oh notation, together with a short explanation.

Testing

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this assignment. It expects to find, in the current directory, the program* `myTrain.c` *and any of the admissible ADTs (*`Graph,WGraph,stack,queue,list`*) that your program is using, even if you use them unchanged. You can use dryrun as follows:*

```
prompt$ 9024 dryrun myTrain
```

Please note: Passing dryrun does not guarantee that your program is correct. You should thoroughly test your program with your own test cases.

Submit

For this project you will need to submit a file named `myTrain.c` and, optionally, any of the ADTs named `Graph,WGraph,stack,queue,list` that your program is using, even if you have not changed them. You can either submit through WebCMS3 or use a command line. For example, if your program uses the `Graph` ADT and the `queue` ADT, then you should submit:

```
prompt$ give cs9024 assn myTrain.c Graph.h Graph.c queue.h queue.c
```

Do not forget to add the time complexity to your main source code file `myTrain.c`.

You can submit as many times as you like — later submissions will overwrite earlier ones. You can check that your submission has been received on WebCMS3 or by using the following command:

```
prompt$ 9024 classrun -check assn
```

Marking

This project will be marked on functionality in the first instance, so it is very important that the output of your program be ***exactly*** correct as shown in the examples above. Submissions which score very low on the automarking will be looked at by a human and may receive a few marks, provided the code is well-structured and commented.

Programs that generate compilation errors will receive a very low mark, no matter what other virtues they may have. In general, a program that attempts a substantial part of the job and does that part correctly will receive more marks than one attempting to do the entire job but with many errors.

Style considerations include:

- Readability
- Structured programming
- Good commenting

Plagiarism

Group submissions will not be allowed. Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar projects in previous years, if applicable) and serious penalties will be applied, particularly in the case of repeat offences.

- ***Do not copy ideas or code from others***
- ***Do not use a publicly accessible repository or allow anyone to see your code, not even after the deadline***

Please refer to the on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Plagiarism and Academic Integrity
- UNSW Plagiarism Policy Statement
- UNSW Plagiarism Procedure

Help

See FAQ for some additional hints.

Finally …

Have fun! Michael