COMP9024 23T3
Dr. Aditya Joshi

# Assignment
TripView

## 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 20 October 2023**

## 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)<br>5 marks for stage 2 (correctness)<br>2 marks for stage 3 (correctness)<br>1 mark for complexity analysis<br>1 mark for style<br>---<br>Total: 12 marks |
| **Due** | 5:00:00pm on **Monday** 13 November (week 10) |
| **Late** | 5% penalty per day late<br>(e.g. if you are 25 hours late, your mark will be reduced by 10%) |

## Aim

The objective is to write a program `tripView.c` that generates an optimal trip on (a part of) Sydney's railway network based on user preferences.

## Input

### Railway stations

The first input to your program consists of an integer *n* > 0, indicating the number of railway stations on the network, followed by *n**2 lines of the form:

```
railway-station
transfer-time
```

where the first line is the name of a station and the second line denotes the time – in minutes – it takes to transfer to a different train at that station.

Here is an example:

```
prompt$ ./tripView
Size of network: 3
HarrisPark
1
TownHall
3
NorthSydney
2
```

You may assume that:

- The input is syntactically correct.
- The maximum length (`strlen()`) of the name of a railway station is 16 and will not use any spaces.
- The transfer time will be a positive integer.
- No name will be input more than once.

*Hint:*

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

```
scanf("%s", name);
```

where `name` is a string, i.e. an array of `chars`.

### Timetables

The next input to your program is an integer *m* > 0, indicating the number of trains on any day, followed by *m* timetables. Each timetable starts with the number *s* > 1 of stops followed by *s*2* lines of the form:

```
station
hhmm
```

meaning that you can get on or off the train at that station at the given time (`hh` – hour, `mm` – minute).

Here is an example:

```
Number of timetables: 2
Number of stops: 3
HarrisPark
0945
TownHall
1020
NorthSydney
1035
Number of stops: 2
TownHall
1024
NorthSydney
1033
```

You may assume that:

- The input is syntactically correct.
- All times are given as 4 digits and are valid, ranging from 0000 to 2359.
- Only train stations that have been input earlier as part of the network will be used.
- The stops are input in the correct temporal order.
- Each stop will be visited at most once in a single timetable.
- All trains reach their final stop before midnight.

**Trip View**

The final input to your program are user queries:

```
From: HarrisPark
To: NorthSydney
Arrive at or before: 1200
```

As before, you may assume that the input is correct: Two different valid railway stations followed by a valid time in the form of 4 digits.

Your program should terminate when the user enters `"done"` when prompted with `From:`

```
From: done
Bye
prompt$
```

## Stage 1 (3 marks)

Stage 1 requires you to generate a suitable data structure from the input.

Test cases for this stage will only use queries `FromStation`, `ToStation`, `ArrivalTime` such that:

- there exists one, and only one, train that travels from `FromStation` to `ToStation` ;
- this train arrives on, or before, the given `ArrivalTime` ; and
- this train is the desired output for the query.

Therefore, at this stage all you need to do is find and output the connection between the two train stations, including all the stops along the way and the arrival/departure times.

Here is an example to demonstrate the expected behaviour of your program for a stage 1 test:

```
prompt$ ./tripView
Size of network: 7
Ashfield
5
Central
8
HarrisPark
1
MilsonsPoint
2
NorthSydney
2
Redfern
5
TownHall
3
Number of timetables: 2
```

```
Number of stops: 5
HarrisPark
0945
Ashfield
0955
Redfern
1006
TownHall
1020
NorthSydney
1035
Number of stops: 4
Redfern
1359
Central
1406
TownHall
1410
MilsonsPoint
1430

From: Central
To: MilsonsPoint
Arrive at or before: 1600

1406 Central
1410 TownHall
1430 MilsonsPoint

From: Ashfield
To: NorthSydney
Arrive at or before: 1040

0955 Ashfield
1006 Redfern
1020 TownHall
1035 NorthSydney

From: done
Bye
prompt$
```

## Stage 2 (5 marks)

For the next stage, your program should find and output a connection from `FromStation` to `ToStation` that:

- may involve one or more train changes;
- arrives at `ToStation` no later than `ArrivalTime` ; and
- leaves as late as possible.

Note that you can get onto a different train at any station, but it is necessary to take into account the time it takes to change trains at that station.

In all test scenarios for this stage there will be **at most one** connection that satisfies all requirements.

Here is an example to demonstrate the expected behaviour of your program for stage 2:

```
prompt$ ./tripView
Size of network: 6
Ashfield
5
Central
8
HarrisPark
1
NorthSydney
2
Redfern
5
TownHall
3
Number of timetables: 2
Number of stops: 5
HarrisPark
0945
Ashfield
0955
Redfern
1006
TownHall
1020
NorthSydney
1035
```

```
Number of stops: 3
HarrisPark
0950
Central
1010
TownHall
1017

From: HarrisPark
To: NorthSydney
Arrive at or before: 1040

0950 HarrisPark
1010 Central
1017 TownHall
Change at TownHall
1020 TownHall
1035 NorthSydney

From: done
Bye
prompt$
```

If there is no connection that satisfies the requirements, then the output should be: `No connection.`

```
From: HarrisPark
To: TownHall
Arrive at or before: 1015

No connection.
```

## Stage 3 (2 marks)

For the final stage, if there are multiple possible connections with the same latest departure time, your program should take into account the additional user preference that:

- among all the connections with the latest possible departure time, choose the one with the **shortest overall travel time.**

You may assume that there will never be more than one connection with the latest possible departure time **and** the shortest overall travel time. Note also that travel time includes the time it takes to change trains and the waiting time if applicable.

Here is an example to demonstrate the expected behaviour of your program for stage 3:

```
prompt$ ./tripView
Size of network: 3
HarrisPark
1
NorthSydney
2
TownHall
3
Number of timetables: 2
Number of stops: 3
HarrisPark
0945
TownHall
1020
NorthSydney
1035
Number of stops: 2
TownHall
1024
NorthSydney
1033

From: HarrisPark
To: NorthSydney
Arrive at or before: 1040

0945 HarrisPark
1020 TownHall
Change at TownHall
1024 TownHall
1033 NorthSydney

From: done
Bye
prompt$
```

## Complexity Analysis (1 mark)

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

1. the size of the network, *n*
2. the number of timetables, *m*
3. the maximum number of stops on any one timetable, *s*.

## 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`
- priority queue ADT : `PQueue.h`, `PQueue.c`
- graph ADT : `Graph.h`, `Graph.c`
- weighted graph ADT : `WGraph.h`, `WGraph.c`

**You are free to modify any of the six 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 `tripView.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* `tripView.c` *and any of the admissible ADTs (`Graph,WGraph,stack,queue,PQueue,list`) that your program is using, even if you use them unchanged. You can use dryrun as follows:*

```
prompt$ 9024 dryrun tripView
```

*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 `tripView.c` and, optionally, any of the ADTs named `Graph,WGraph,stack,queue,PQueue,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 tripView.c Graph.h Graph.c queue.h queue.c
```

Do not forget to add the time complexity to your main source code file `tripView.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 programs must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar assessments in previous years, if applicable) and serious penalties will be applied, including an entry on UNSW's plagiarism register.

You are not permitted to use code generated with the help of automatic tools such as GitHub Pilot, ChatGPT, Google Bard.

- ***Do not copy ideas or code from others***
- ***Do not use a publicly accessible repository or allow anyone to see your code***
- ***Code generated by GitHub Pilot, ChatGPT, Google Bard and similar tools will be treated as plagiarism.***

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
- UNSW Plagiarism Management Procedure

## Help

See FAQ for some additional hints.

## Finally …

Have fun!