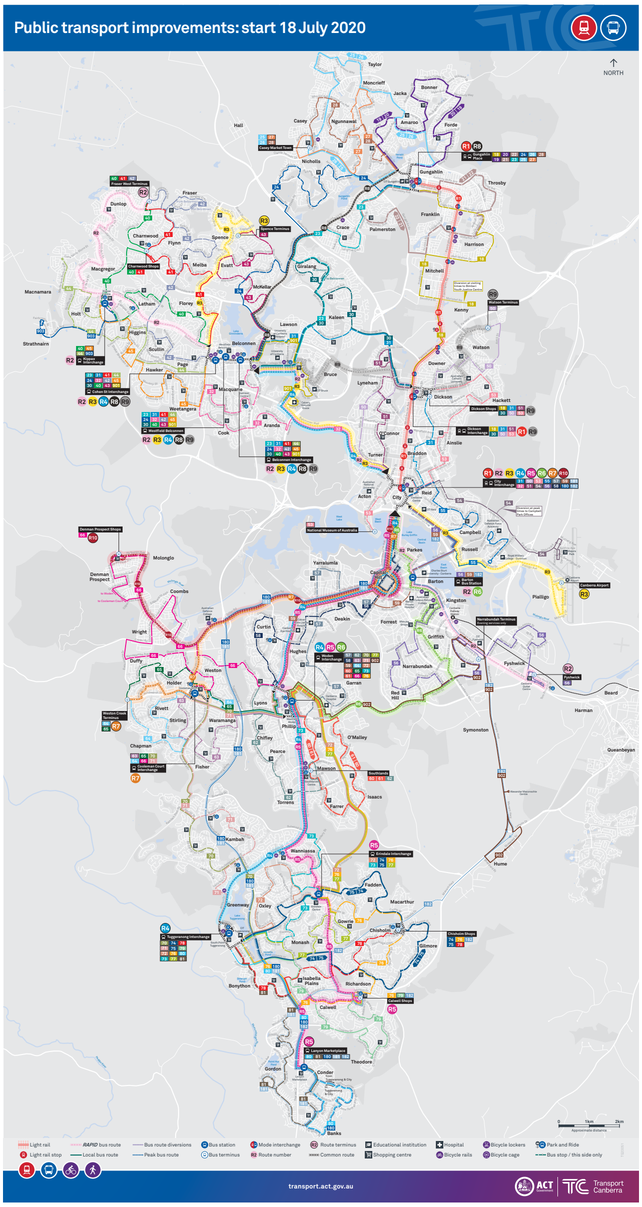
II.2315 – Project : Advanced algorithm and programming

Australia - Canberra



Sarah HEOUAINE - David LAMY-VERDIN - Elia TSO

2020 - 2021

Table of contents

1 Introduction

1.1 City of choice

We selected the city of Canberra, capital of Australia, for our project. This city has a total of 2433 stations and 2759 links.

1.2 Goals of the project

This project had for first goal to create a graph representing the transport map of Canberra out of mere data. After this graph has been created, we could use it to reach other goals described in the below list:

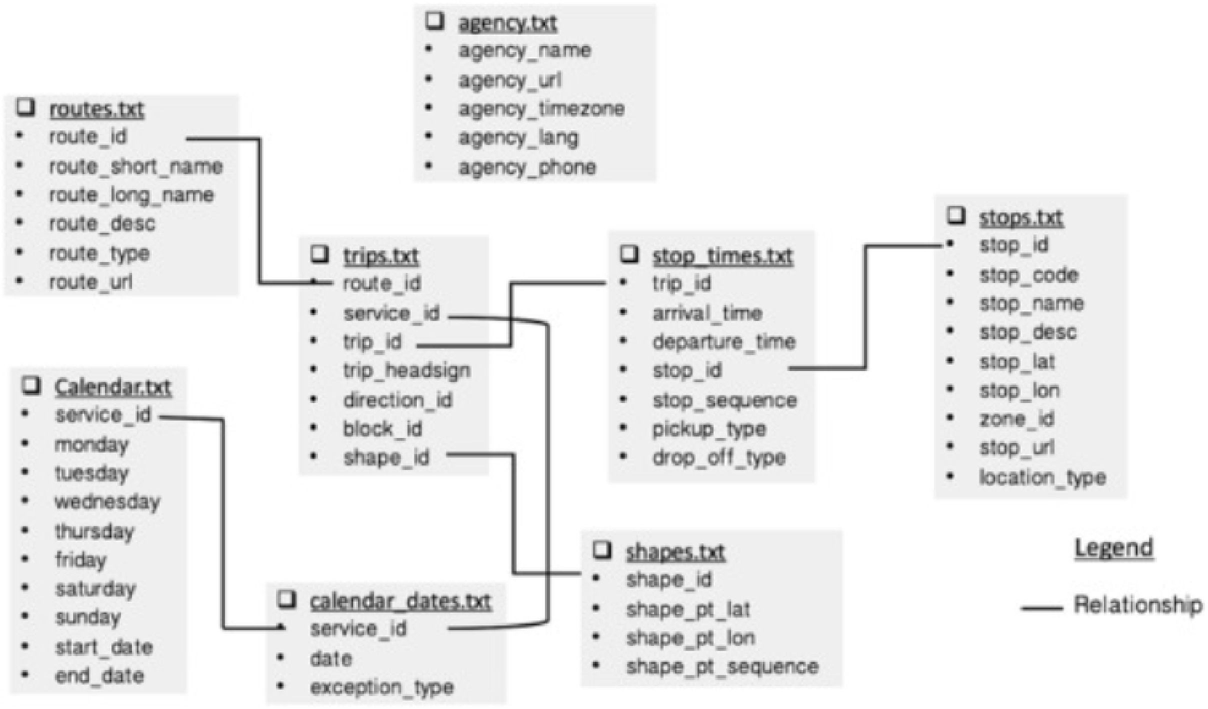
- Search algorithms
  Implementation of the Bread-First Search Algorithm
  Implementation of the Dijkstra Algorithm

- Applications of those algorithms
  Searching for shortest paths
  Splitting the map into clusters

Through this project, optimization also had to be done as searching for shortest paths and splitting into clusters a graph as large as Canberra Transport Map was particularly demanding on resources and took a considerable amount of time.

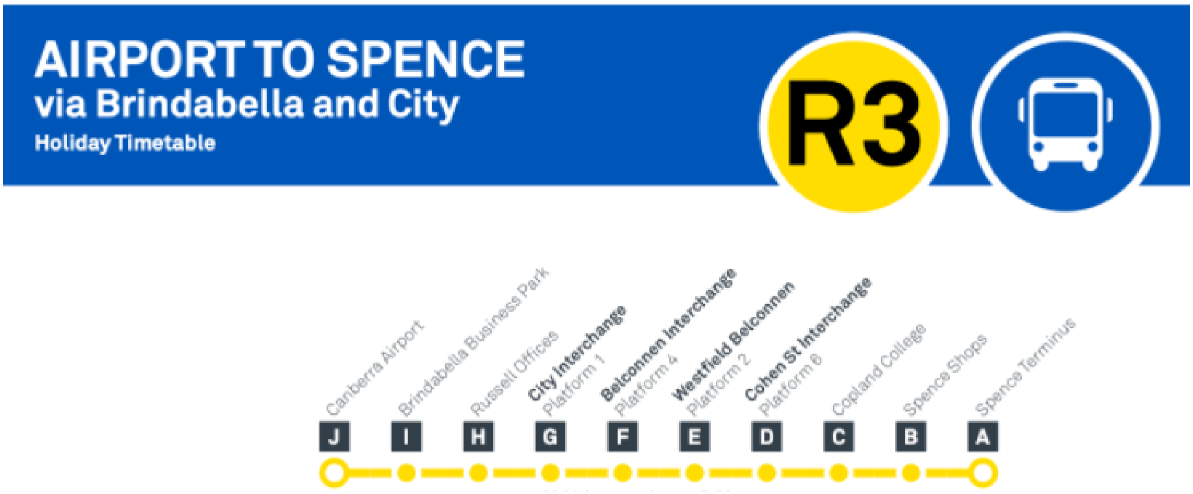
1.3 Collection of data : GTFS files

To build our graph, we used data retrieved from Australia Government official website. This data come as multiple *.txt* files. After studying them, it has been figured out that only *stops.txt* and *stop_times.txt* were actually useful.



Using only those, we could create the stations which are represented in our graph as nodes and we could also link each of them and thus creating our edges.

The *stops.txt* file provide us with *stop_id*, *stop_lat*, *stop_lon* which are all the information we needed to create our nodes while the *stop_times.txt* give us *trip_id* which allow us to create our edges. A trip informs about which stations are linked together and in which order as it defines a bus or subway line such as the R3 line of Canberra for example:



2 Creation of the graph

2.1 Graph

`Graph.java` is the class used for building graphs. The class contains the three following attributes:

- `private Map<Integer, List<DirectedEdge>> map = new TreeMap<Integer,List<DirectedEdge>>()` which are the adjacency lists of every node of our graphs. Each `Key` of the `Map` designates a node of our graph and the corresponding `Value` of the `Map` is its adjacency list.
- `private boolean weighted` indicates if the graph is weighted or not.
- `private boolean directed` indicates if the graph is directed or not.

Besides these attributes, `Graph.java` also has several methods whose most important ones are the following:

- `private void addNodesFromTxt(File stopsFile)`
- `private void addEdgesFromTxt(File stopTimesFile)`
- `private void addWeightsFromTxt(File stopsFile)`

All of these methods allow us to parse *stops.txt* and *stop_times.txt*, retrieve the data and create the content of our graph which are nodes and edges.