Usage and implementation of the graph theory algorithms for calculating the best routes for public transport

Subject: Mathematics

Aim: implement a computer program similar to 9292, providing the “best” route for travelling between one station to another station, on a limited model.

Research question: **What algorithms of graph theory are the least resource-consuming for calculating optimal commuting routes (in terms of spent time, price, or the number of transfers)?**

Exam session: May 2024

Word count: 3991

Contents

Introduction and Personal Engagement

Public transport is a complex system. For commuters to navigate the system comfortably, online applications such as 9292 were developed. The user can indicate a starting point, a destination point, time of arrival or departure and receive multiple journey options. These applications undoubtedly use graph theory algorithms, which will be described further.

Graph theory is a popular topic in competitive programming, even though it is a mathematical concept. I enjoy competitive programming, and I became a finalist in the NIO (Dutch Informatics Olympiad) in 2023. This work’s purpose is to bridge the gap between competitive programming, applied programming, and discrete mathematics, which includes graph theory.

Aim: implement a computer program similar to 9292, providing the “best” route for travelling between one station to another on a limited model.

Research question: **What algorithms of graph theory are the least resource-consuming for calculating optimal (in terms of spent time, price, or the number of transfers) commuting routes?**

Chapter 1. Definition of graph. Formalising public transport system to a graph

Section 1.1. Definition of graph

A graph is a structure containing a set of vertices (or nodes, notated as V) and edges between nodes (edges are also called links, notated as E).

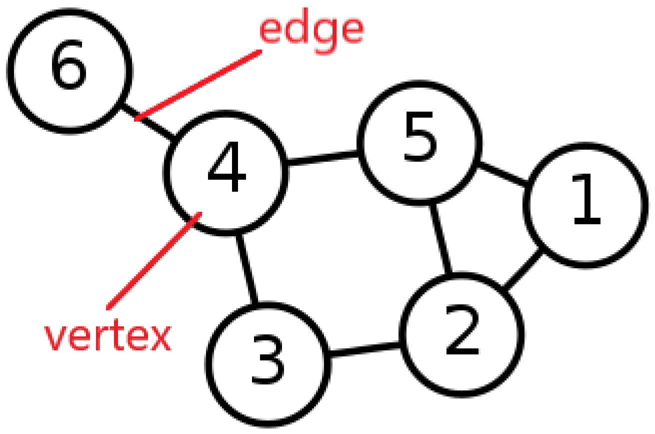


Figure 1. Azatoth. 6n-graf. Graph, created in Neato. March 7, 2007. Retrieved on March 16, from Wikimedia Commons: https://commons.wikimedia.org/wiki/File:6n-graf.svg

In this graph (see Fig. 1), vertices are numbered. If, in a graph, edges are only traversable in one direction, the graph is *directed*, else it is *undirected*. Moreover, in Figure 1, vertices 4, 5, 2, and 3 form a *cycle*.

Section 1.2. How can a transport system be generalized to a graph system?

Let us discover the imaginary Roman Empire road scheme.

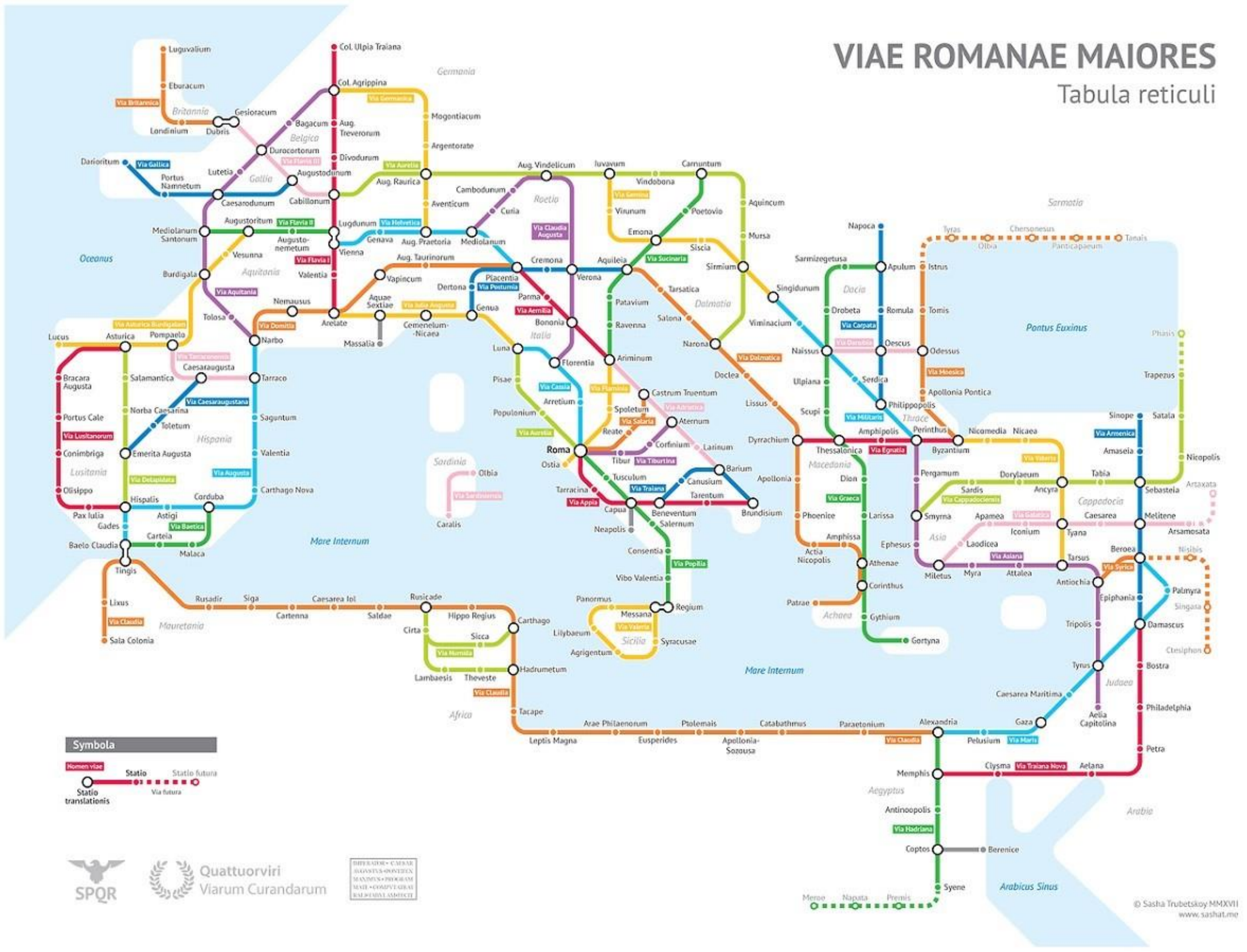


Figure 2. Trubetskoy, Sasha. Roman Roads. June 3, 2017. Retrieved on March 16, 2023, from Visual Capitalist: https://www.visualcapitalist.com/roman-empires-roads-map/

In this transport model (see Fig. 2), cities are considered as vertices and roads between them as edges. This model has cycles and is undirected. This model is similar to the model that will be presented further. Instead of cities, there will be stations and stops, instead of roads, there will be public transport routes.

One always pays a fee for commuting. In terms of the graph model, there is a cost for traversing an edge. It also takes time to traverse the edge. In conclusion, edges have their characteristics. In this case, it is said that the graph is *weighted*.

Then, some edges (or parts of the route) are served by multiple bus routes at the same time. This means that sometimes 2 nodes are connected by two or more edges. These edges might have different characteristics, like price or time.

Plus, it has been observed that traversing the edge in forward and reverse direction might take different time and cost differently. This means the graph model has to be directed. It is also necessary to create edges for the forward and reverse direction of traversal, if necessary, with different characteristics.

Lastly, consider going from stop A to stop B to stop C, all on the same bus. However, the first time you go from stop A to stop B, get off, get on the same bus at stop B, and go for stop C. We have path $A \to B \to C$. The other case is going from stop A directly to stop C on the same bus. The path is $A \to C$. The price for path $A \to B \to C$ is more expensive than one for path $A \to C$. This is true for all bus and train routes, though the price difference might vary depending on the type of transportation. This brings us to the concept of boarding cost, which will be described in detail.

Section 1.3. What is the "best" route on public transport?

Some criteria for the "best" route must be defined.

One of the most straightforward criteria is the cost of the path. People would like to minimise the cost of their travelling.

Another useful criterion is the time needed for the journey. It might be even more important than the price: in most cases, one could spend an extra euro to save an hour.

On a longer route, transfers become nearly unavoidable. For some, transfers are a stressful experience. They might not excel at orienteering and therefore might miss a train or a bus. Therefore, minimising the number of transfers might be useful.

It is expected that optimising each of these criteria individually is possible with an algorithm from graph theory. Before analysing the algorithms that might help us with the optimisations, discussing the time complexity of algorithms is essential to make sure the most efficient algorithms are used.

Chapter 2. Time Complexity

Time complexity is an asymptotic estimate for a function of the time for the algorithm to complete with the size of input as an argument. *O-notation* is used to show the upper bound of such an estimate. Let's take binary search as an example. One has to guess a number from 1 to n, by asking questions like: "Is the number larger or equal to x?". Turns out that it is optimal to act in the following way. If you have a search segment $[l, r]$, you should ask for $x = \frac{(l+r)}{2}$, so that, whatever the answer is, your search segment becomes 2 times smaller. Repeat until the search segment is down to 1. It is easy to calculate that it will take you $\log_2 n$ questions to guess the number. So, the time complexity of the algorithm is $O(\log_2 n) \approx O(\log n)$. Since, in most algorithms, the base of the logarithm is 2, it is omitted. (A. Bhargavae, 2016, p.25) The same goes for the coefficients of polynomials, and all the power of the polynomial other than the biggest one. This means $O(7n^3 + 5n^2 + 6n)$ is simplified to $O(n^3)$.

Chapter 3. Description of graph theory algorithms

Section 3.1. Introduction

There are three points of optimisation: minimum number of transfers, minimum total cost, and minimum total time for a journey. The first optimisation is the problem of the shortest path on an unweighted graph. The latter two require solving the same problem on a weighted graph. The breadth-first search is the fastest and most reliable algorithm for solving the first problem, while the Dijkstra algorithm is the fastest for the second one.

Section 3.2. Breadth-first search

Breadth-first search (normally referred to as *BFS*) is used for traversing a graph. A queue is created for vertices that are to be processed. Starting from the source, all the vertices reachable directly from the current vertex are pushed to the queue, then the current vertex is removed from the queue, and the next vertex is processed. The procedure is repeated until there are no unvisited vertices. The graph may contain cycles, so an array of *Boolean* $used$ is implemented to make sure a vertex is not visited twice. Boolean is a data type that descends from logic and Boolean algebra. It has two possible values: 0 (false) and 1 (true). The time complexity is $O(n + m)$, where n is the number of vertices and m is the number of edges ("Breadth-first search").

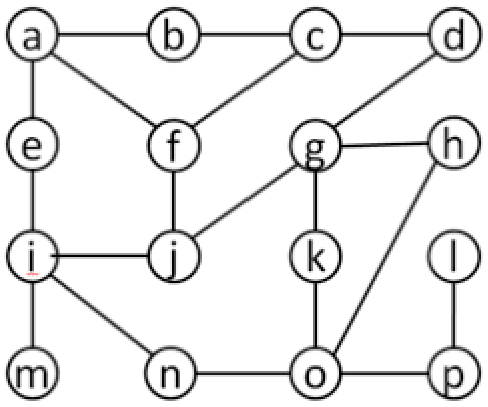


Figure 3. Sample graph. Retrieved on March 23, 2023, from StackExchange: https://softwareengineering.stackexchange.com/questions/288702/how-definite-of-an-order-is-there-to-a-depth-first-search-of-a-graph

Starting from a (see Figure 3), the order of traversal is as follows:

Layer 1: a

Layer 2: b, e, f

Layer 3: c, i, j

Layer 4: d, g, m, n

Layer 5: k, h, o

Layer 6: p

Layer 7: l

For a vertex, we define the first adjacent vertex present in the previous layer of BFS traversal as its *parent*. In the traversal shown above, the vertex b's parent is a. Vertex a is the source; therefore, it has no parents.

Similarly, if vertex a is the parent of vertex b, then b is a son of vertex a. ("Breadth-first search")

Section 3.3. Dijkstra algorithm

Dijkstra algorithm is used for calculating the shortest path in a weighted graph with n vertices and m edges. Just like BFS, it starts with a chosen point – the source. Let us denote it as s. The algorithm proceeds as follows.

Create an array d, where $d[u]$ is the shortest distance between u and s. Initially, $d[s] = 0$, the path length for other vertices is chosen to be positive infinity (or, in practice, an extremely large number). Create a *Boolean array* $used$, where $used[u]$ means whether the node has been visited or not. ("Dijkstra algorithm")

At each of n iterations,

- Pick an unvisited vertex u with the least value of $d[u]$. Set $used[u] = 1$.

- Perform *relaxations* from vertex u as follows. For each neighbouring vertex to, suppose the weight of edge (u, to) is len. Perform the following operation: $d[to] = \min(d[to], d[u] + len)$.

When n vertices are visited, the algorithm terminates.

However, it is not enough to get the lowest possible price. The travel advice includes the path to the destination. It is possible to modify the Dijkstra algorithm to restore the shortest path from vertex s to any vertex. Create an array of parents p and set $p[s] = -1$. When doing relaxations, if $d[to] > d[u] + len$, set $d[to] = d[u] + len$ and $p[to] = u$. The path from s to u is $M = (s, \dots, p\left[p[p[u]]\right], p[p[u]], p[u], u)$. ("Dijkstra algorithm")

"The running time of the algorithm consists of n searches for a vertex with the smallest value $d[v]$ among $O(n)$ unmarked vertices, plus m relaxation attempts". ("Dijkstra algorithm") Thus, the estimated complexity is $O(n^2 + m)$.

Using certain non-linear data structures, the algorithm can be optimised to $O(m \log n)$. It only makes sense for sparse graphs (graphs for which $m \approx n$ holds), because for full or nearly full graphs $m \approx n^2$ holds, therefore the estimated complexity is $O(n^2 \log n)$, which is worse than the "classic algorithm" ("Dijkstra on sparse graphs"). Moreover, the mentioned non-linear data structures are notoriously complex.

It is important to mention that the Dijkstra algorithm only works correctly if all the weights are non-negative. ("Dijkstra algorithm"). This is the case for the model presented in Chapter 4.

Dijkstra algorithm is considered to be "*greedy*". Greedy algorithms are algorithms that make a local optimal decision at every iteration and get a globally optimal solution.

Chapter 4. Applying the algorithms to a model

Section 4.1. Introduction to the model

How are these algorithms used in a real situation? To answer this question, I built a small model, mostly revolving around Leiden.

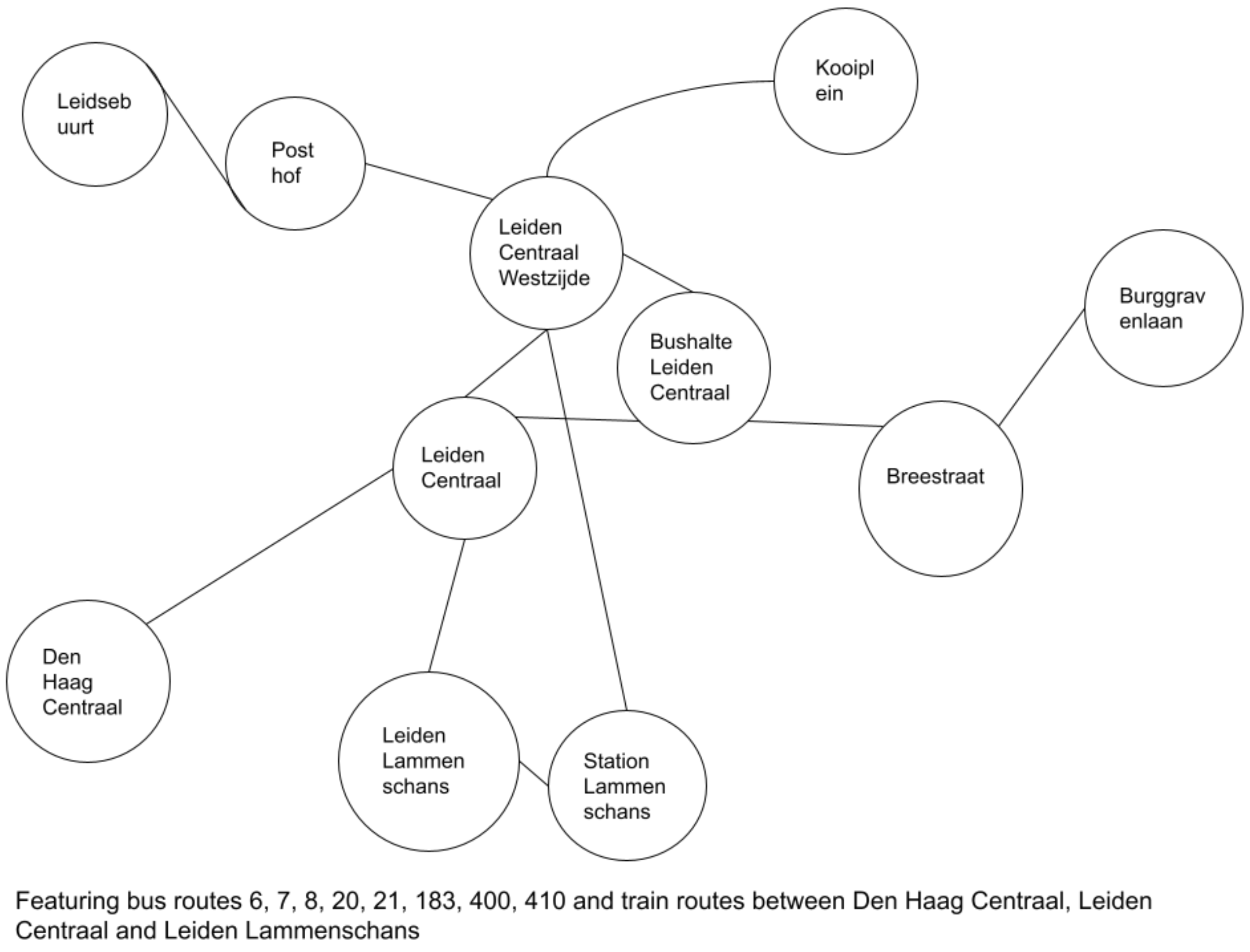

Leidseb uurt
Post hof
Kooipl ein
Leiden Centraal Westzijde
Bushalte Leiden Centraal
Burggrav enlaan
Leiden Centraal
Breestraat
Den Haag Centraal
Leiden Lammen schans
Station Lammen schans
Featuring bus routes 6, 7, 8, 20, 21, 183, 400, 410 and train routes between Den Haag Centraal, Leiden Centraal and Leiden Lammenschans

Figure 4. Author, *Graph of the model.* Graph

In Figure 4, edges between two "neighbouring" vertices are present in the mini-model. In other words, if there is a possible path $A -> B -> C$, there will be two edges: $A \to B$ and $B \to C$. This will have to change when we start showing how BFS and Dijkstra algorithm work on this model.

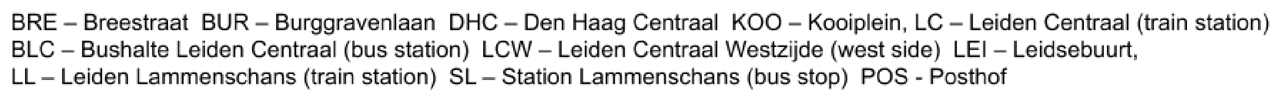

BRE – Breestraat BUR – Burggravenlaan DHC – Den Haag Centraal KOO – Kooiplein, LC – Leiden Centraal (train station)
BLC – Bushalte Leiden Centraal (bus station) LCW – Leiden Centraal Westzijde (west side) LEI – Leidsebuurt,
LL – Leiden Lammenschans (train station) SL – Station Lammenschans (bus stop) POS - Posthof

Figure 5. Author, *Codes for stations (chosen arbitrarily)*. Scheme

Some codes are chosen for stations (see Figure 5). They will be used further for brevity.

Section 4.2. Modification of the model for BFS

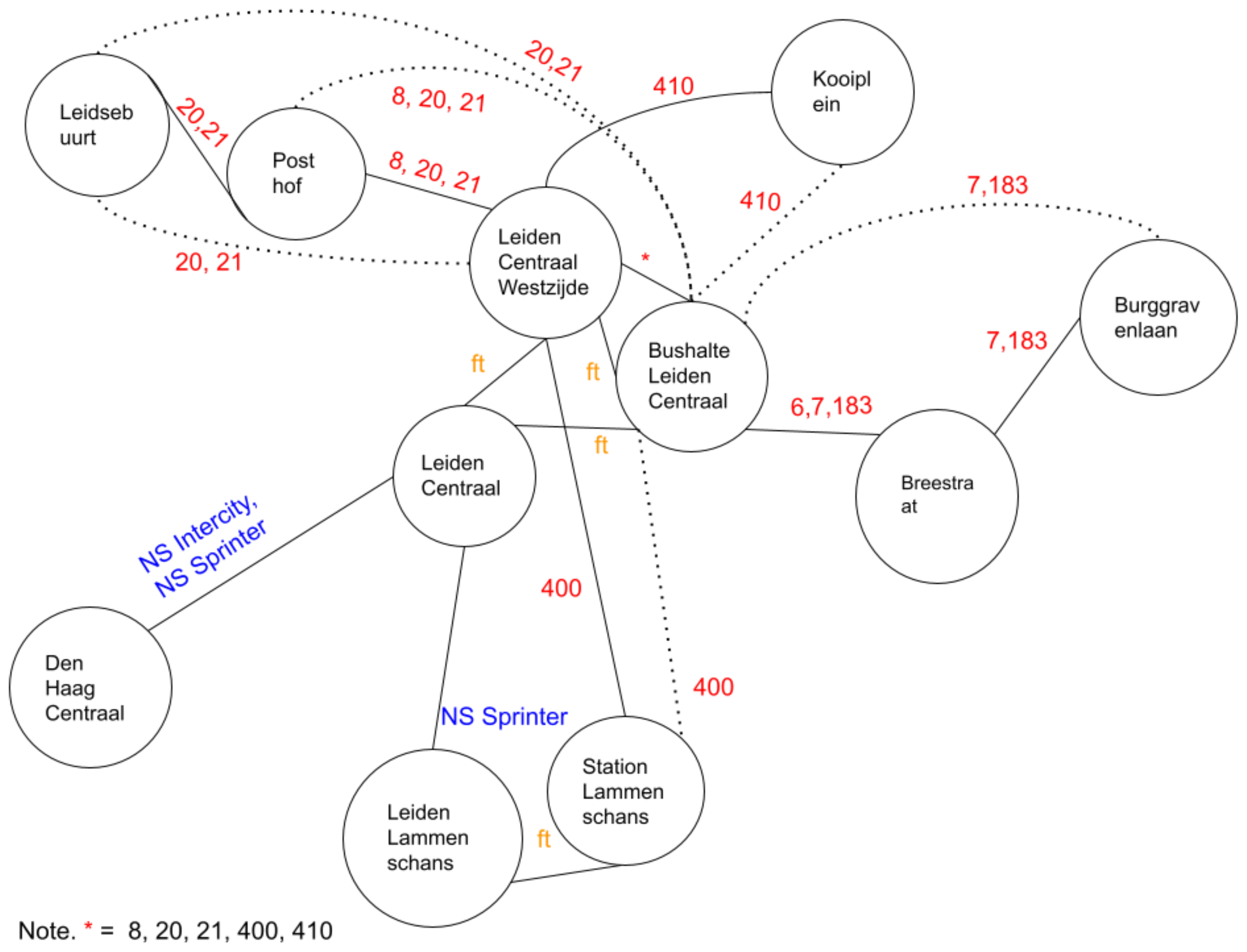


Figure 6. Author, *Graph for BFS*. Graph

In Figure 6, there are numbers or words near every edge. They represent the method of traversing the edge. Red numbers stand for bus routes. If there are multiple notes near the edge, it means there are multiple edges between adjacent vertices. For instance, Posthof and Leiden Centraal Westzijde are connected by bus routes 8, 9, 20, 21, and more. *"ft"* means "on foot". Two vertices connected by *"ft"* edge are reachable on foot. Train routes are blue.

To make BFS solve the task of calculating the minimum amount of transfer on a route, an adjustment to the initial model is to be made. If there is a path $A \to B \to C \to D \to \cdots$ traversed by a single bus route or train, we have to add edges so that every vertex in the said path is directly connected to every vertex that follows later in the same path. In other words, besides edges $\{A, B\}$, $\{B, C\}$, $\{C, D\}$, edges $\{A, C\}$, $\{B, D\}$, $\{A, D\}$ should be present in the graph to make BFS valid. In the picture, they are indicated by dotted lines.

Let us take a journey from Leidsebuurt to Burggravenlaan. Before we start, we need to create two arrays: p for registering the parent of a current vertex, and d for registering the number of different routes of public transportation needed to get from the starting point. We define $p[LEI] = -1$ and $d[LEI] = 0$ because LEI is the source. Then, we complete BFS traversal as explained earlier, with a few changes.

When dealing with vertex v and its son to, assign $p[to] = v$ and $d[to] = \min(d[v] + k,$ $d[to])$. k is 0 when the type of edge $\{v, to\}$ is "ft" and 1 otherwise.

The number of transfers is $d[BUR] - 1$. If we use k types of transport (walking on foot does not count as public transport), then we change routes $k - 1$ times. In the example, $d[BUR] = 2$, therefore the minimal number of transfers is 1.

Section 4.3. Modification of the model for the Dijkstra algorithm for calculating the cheapest path

It is too difficult to draw a graph that would help explain the application of the algorithm. Therefore, a table representation is used (see Table 1). The intersection of row i and column j is the information about the edge (i, j): possible prices for traversing the edge, as well as routes that operate on this edge. The latter are given in brackets.

from\to	BRE	BUR	DHC	KOO	LC	BLC	LCW	LEI	LL	SL	POS
BRE	0	1.31€ (7, 183)				1.30€ (6, 7, 183)					
BUR	1.27€ (7, 183)	0				1.49€ (7, 183)				-	
DHC			0		4.10€ (Sprinter, Intercity)	-	-				
KOO				0		1.54€ (410)	1.48€ (410)				
LC		-	4.10€ (Sprinter, Intercity)	-	0	0 (walk)	0 (walk)		2.60€ (Sprinter)		
BLC	1.27€ (6, 7, 183)	1.50€ (7, 183)		1.58€ (410)	0 (walk)	0	0 (walk) or 1.16€ (8,20, 21, 400, 410)	1.47€ (20, 21)	-	1.82€ (400)	1.24€ (20, 21, 8)

from\to	BRE	BUR	DHC	KOO	LC	BLC	LCW	LEI	LL	SL	POS
LCW				1.48€ (410)	0 (walk)	0 (walk) or 1.13€ (20, 21, 8) or 1.14€ (400, 410)	0	1.39€ (20, 21)	-	1.74€ (400)	1.16€ (20, 21, 8)
LEI						1.45€ (20,21)	1.41€ (20,21)	0			1.33€ (20, 21)
LL	-	-		-	2.60€ (Sprinter)	-	-	-	0	0 (walk)	-
SL						1.82€ (400)	1.76€ (400)	-	0 (walk)	0	
POS						1.20€ (20, 21, 8)	1.16€ (20, 21, 8)	1.31€ (20,21)	-	-	0

Table 1. Author. Representation of the model graph with prices to traverse every edge. Sources of prices: 9292, NS

A journey from Leidsebuurt to Burggravenlaan would serve as a good example. We proceed with the Dijkstra algorithm. For example, starting from Leidsebuurt, we do 3 relaxations: $d[POS] = €1.33$, $d[LCW] = €1.41$, and $d[BLC] = €1.49$ (see table 1). Proceed until all the vertices are visited. As a result of the algorithm execution, we receive $d[BUR] = €2.95$ as the answer. Check the answer on the 9292 website. The proposed price is €1.87. To understand the mistake, one should return to the concept of boarding costs.

Section 4.4. Note about boarding cost

Observations can be made about the price of the buses when using the same bus route. 9292 is the source for all the prices in the example. Route 20 might act as an example. It costs €1.47 to travel from Bushalte Leiden Centraal to Leidsebuurt bus stop, €1.24 from BLC to Posthof and €1.31 from Posthof to Leidsebuurt. So it is more expensive by €1.08 to travel in part than directly. Let us look from a different angle. The distance traversed in parts and directly is the same. What is different is the number of boardings: two when travelling in parts and one when travelling directly. It is possible to define the cost difference as *boarding cost*.

Boarding cost for different bus routes is nearly always the same. The largest deviation is found between stations Posthof, Leiden Centraal Westzijde, and Bushalte Leiden Centraal. The difference between going from Posthof to BLC directly and with a stop at LCW is €1.09 (see table 1). This change will be considered as a discrepancy.

Most importantly, boarding cost is only charged once if transferring between the same type of transport. This is the root of the mistake made previously. The difference between the calculated and actual prices is exactly €1.08.

There are only two train paths in the presented model - Leiden Centraal to Den Haag Centraal and Leiden Lammenschans to Leiden Centraal, and they are not part of one train route. However, the price difference is still observed and equal to $€2.60 + €4.10 - €4.30 = €2.40$. This is the boarding cost for trains. It is only charged once as well.

Note: the boarding cost is charged every time a commuter transfers from bus to train and vice versa.

Boarding costs is one of the reasons to group edges into bus routes. Bus routes will feature the boarding cost as well as the cost to move between two neighbouring stations without boarding cost.

Section 4.5. Modification of the model for the Dijkstra algorithm for calculating the quickest path

The graph used for applying this modification is too complex to display, even with a table. It is, in a way, a combination of the graph used for applying the two former algorithms. If a transport route features some two stations u and v, there will be an edge between two nodes representing these two stations. The edge will contain the following information: the route number and the array of pairs: the time of departure from u and the time of arrival to v.

The execution of the algorithm is similar to the one looking for the cheapest route, however, some changes are made. Suppose somebody is at point v at time t. It is known that certain types of transports, including bus routes, visit the stop. Then, the timetable of every bus route visiting the stop should be loaded. Then, the soonest possible time when the bus visits the stop f so that $f > t$ is found. Afterwards, it is possible to iterate through the stops of the bus before the final stop, relaxing in the same way as in the classic Dijkstra algorithm. Suppose that the bus visits the stop c times, which is reflected in the timetable. Then, it takes $O(c)$ operations to find f. Thus the complexity of the given modification of the Dijkstra algorithm is increased to $O(n^2 + mc)$, where n is the number of nodes and m is the number of edges.

Remember that the goal is to build travel advice for a journey rather than simply get the answer. The travel advice must include timings for every action like getting on and off the bus. It is necessary to implement the aforementioned changes for BFS and the modification of the Dijkstra algorithm for calculating the minimum price, so the program outputs travel advice with the earliest departure. Therefore, the complexity of BFS is increased to $O(n + mc)$ and the complexity of the Dijkstra algorithm for price calculation to $O(n^2 + mc)$.

Section 4.6. The choice

Notice that all described algorithms only provide the optimal route in a single aspect. *BFS* looks for the path with the least number of transfers. The *Dijkstra algorithm* looks for either the cheapest or the least time-consuming path. In other words, a single algorithm will determine the best route in a single criterion.

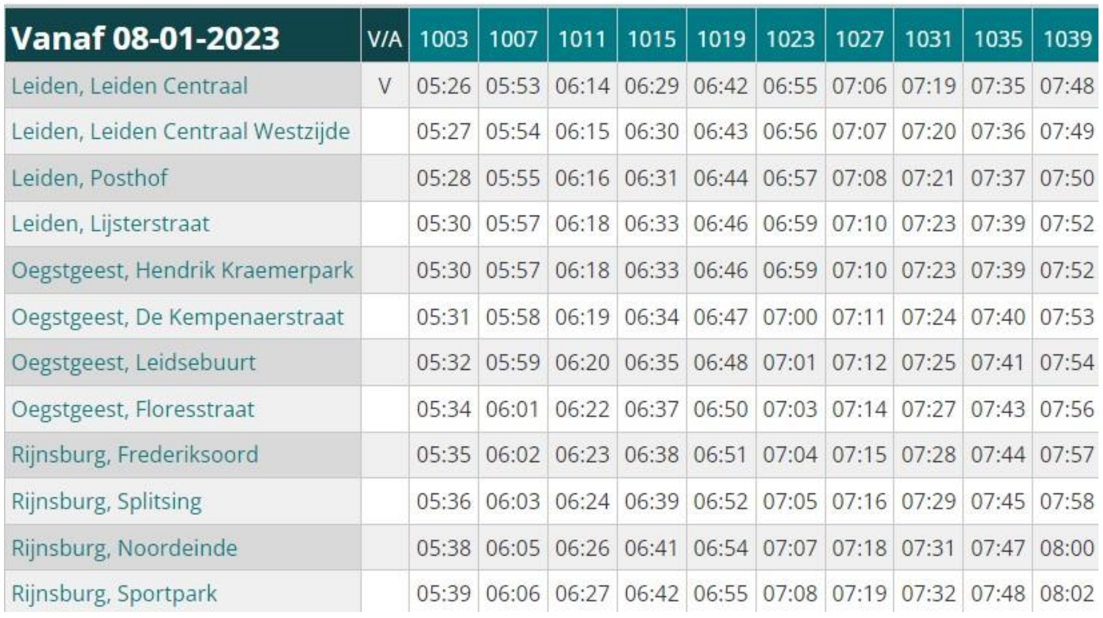
Therefore, one could get three optimal routes according to three different criteria, and choose from the three optimal routes.

Chapter 5. On the way to computer implementation

The aforementioned observations bring us to a possible implementation. The proposed implementation is as follows. There are two main types of objects in the model – stops/stations (nodes) and routes (single edge or collection of edges).

Stations contain information about what routes go through this station, and, for every route, the index of the station in the route, starting from 0 and counting in the direction of the route.

In the presented model, there are three types of routes – going on foot, bus route, and train route. The first type is a special case. It features a starting point, ending point, zero cost, and the time needed to walk from the starting point to the endpoint. Bus and train routes are different. Route object contains the boarding cost, the stations visited by the route and the timetable in order of visiting, and the *true* cost of the edge. The true cost of the edge is the actual cost needed to get from point A to point B, which you can see in 9292, minus the boarding cost. The timetable looks similar to this.



Vanaf 08-01-2023	V/A	1003	1007	1011	1015	1019	1023	1027	1031	1035	1039
Leiden, Leiden Centraal	V	05:26	05:53	06:14	06:29	06:42	06:55	07:06	07:19	07:35	07:48
Leiden, Leiden Centraal Westzijde		05:27	05:54	06:15	06:30	06:43	06:56	07:07	07:20	07:36	07:49
Leiden, Posthof		05:28	05:55	06:16	06:31	06:44	06:57	07:08	07:21	07:37	07:50
Leiden, Lijsterstraat		05:30	05:57	06:18	06:33	06:46	06:59	07:10	07:23	07:39	07:52
Oegstgeest, Hendrik Kraemerpark		05:30	05:57	06:18	06:33	06:46	06:59	07:10	07:23	07:39	07:52
Oegstgeest, De Kempenaerstraat		05:31	05:58	06:19	06:34	06:47	07:00	07:11	07:24	07:40	07:53
Oegstgeest, Leidsebuurt		05:32	05:59	06:20	06:35	06:48	07:01	07:12	07:25	07:41	07:54
Oegstgeest, Floresstraat		05:34	06:01	06:22	06:37	06:50	07:03	07:14	07:27	07:43	07:56
Rijnsburg, Frederiksoord		05:35	06:02	06:23	06:38	06:51	07:04	07:15	07:28	07:44	07:57
Rijnsburg, Splitsing		05:36	06:03	06:24	06:39	06:52	07:05	07:16	07:29	07:45	07:58
Rijnsburg, Noordeinde		05:38	06:05	06:26	06:41	06:54	07:07	07:18	07:31	07:47	08:00
Rijnsburg, Sportpark		05:39	06:06	06:27	06:42	06:55	07:08	07:19	07:32	07:48	08:02

Figure 7. Part of the timetable of bus route 20 (departing from Leiden Centraal and arriving in Noordwijk). Screenshot. Source: Timetables for Arriva bus routes. Retrieved on June 15, 2023, from *Arriva*: https://arriva-reisinfo.fis.nl/

A column regards the journey made by a single bus, a row regards all the instances of a bus route visiting a set station.

Timetables for the forward route and reverse route going under the same number are different, so forward and reverse routes should be considered different routes, even if they go by the same number.

Plus, only stations present in the model presented in Chapter 4 will be included in the timetable.

Also, it is easier to manage time in computer implementation if it is expressed in minutes passed since midnight.

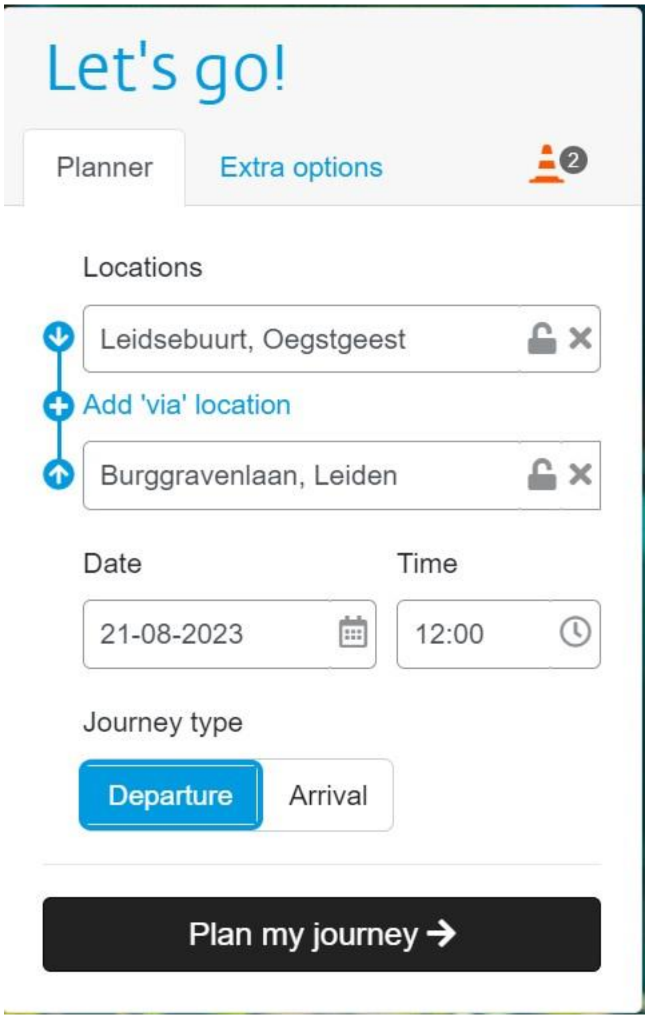
Chapter 6. The program

The whole repository is available on GitHub, a platform for publication of code ("Extended Essay Application"). It includes the code of the application written in the C++ programming language, the files needed to successfully execute the program, and the code of the auxiliary program which was used to pre-process timetables, also written in C++. The application was programmed entirely by myself, except for *initialisationRoutes* and *initialisationStations* functions. The code used in these functions is modified from *C++_Stories*. The auxiliary program was written entirely by myself.

All the timetables were provided by *Arriva* and *NS*, and processed either by the auxiliary program or manually.

Chapter 7. Testing

Test 1. Leidsebuurt to Burggravenlaan, departing at 12:00 on Monday, quickest trip preferred



Locations

Leidsebuurt, Oegstgeest

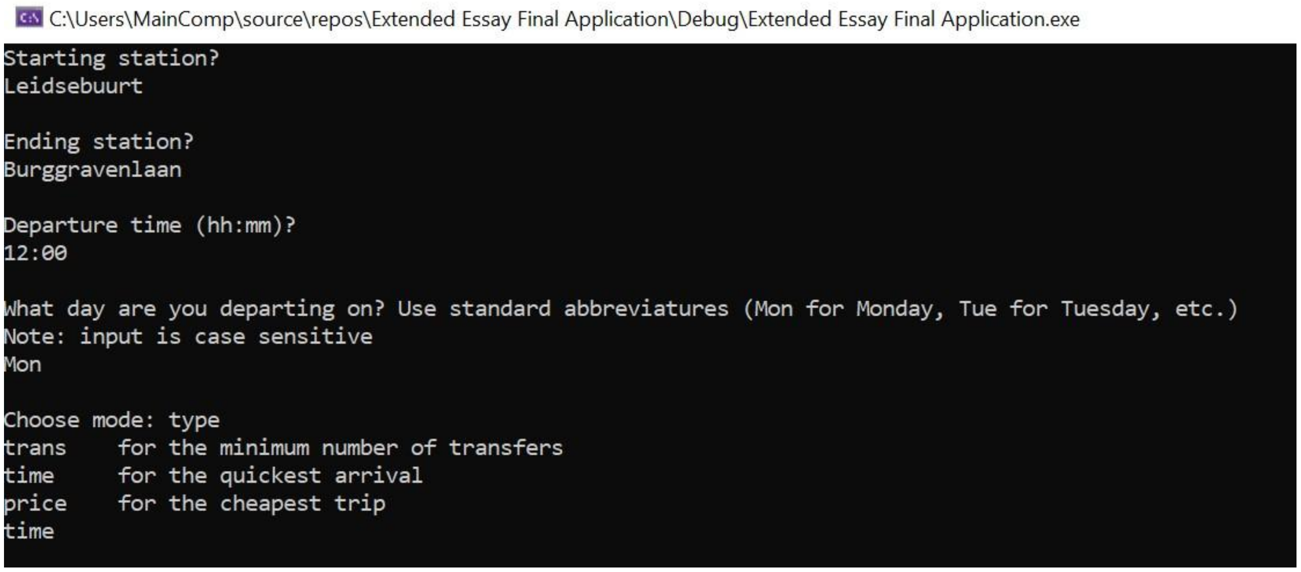
Burggravenlaan, Leiden

Date: 21-08-2023

Time: 12:00

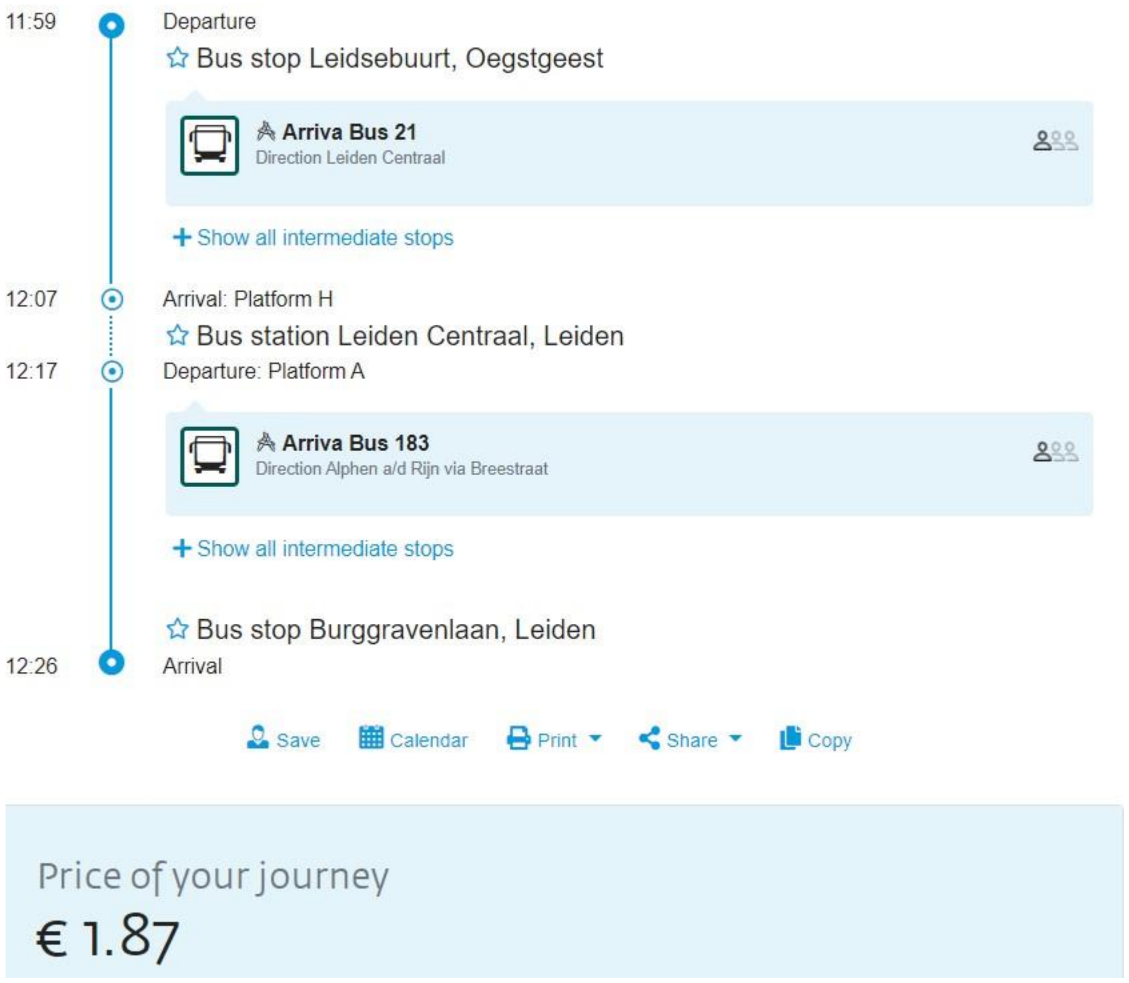
Journey type: [✓ Departure] Arrival

Figure 8. *Setup for Test 1 on 9292*. Screenshot


C:\Users\MainComp\source\repos\Extended Essay Final Application\Debug\Extended Essay Final Application.exe

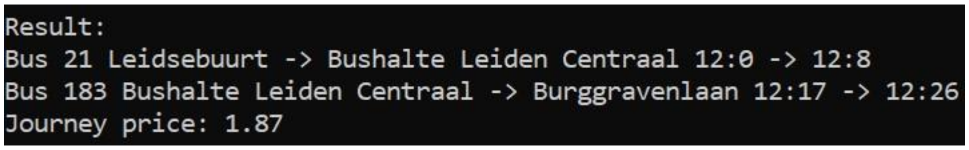
```
Starting station?
Leidsebuurt

Ending station?
Burggravenlaan

Departure time (hh:mm)?
12:00

What day are you departing on? Use standard abbreviatures (Mon for Monday, Tue for Tuesday, etc.)
Note: input is case sensitive
Mon

Choose mode: type
trans    for the minimum number of transfers
time     for the quickest arrival
price    for the cheapest trip
time
```

Figure 9. *Setup for Test 1 in my program*. Screenshot


11:59 Departure
Bus stop Leidsebuurt, Oegstgeest
Arriva Bus 21
Direction Leiden Centraal
+ Show all intermediate stops
12:07 Arrival: Platform H
Bus station Leiden Centraal, Leiden
12:17 Departure: Platform A
Arriva Bus 183
Direction Alphen a/d Rijn via Breestraat
+ Show all intermediate stops
Bus stop Burggravenlaan, Leiden
12:26 Arrival

Price of your journey
€ 1.87

Figure 10. *Results of Test 1 on 9292*. Screenshot

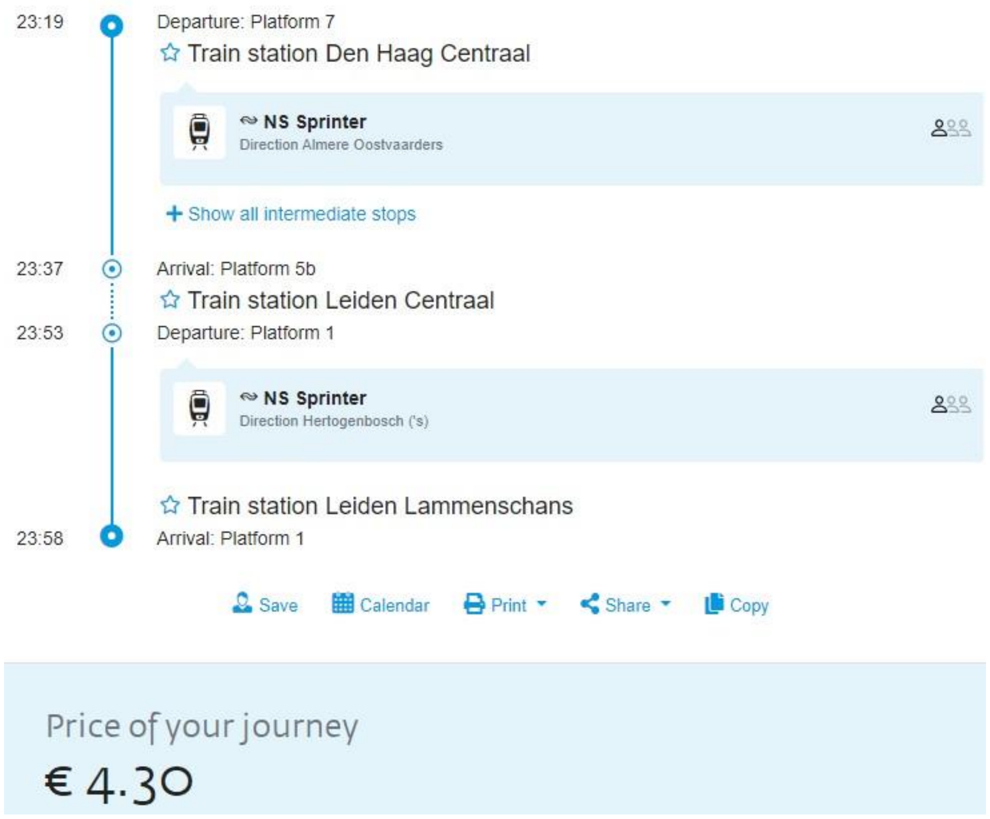


```
Result:
Bus 21 Leidsebuurt -> Bushalte Leiden Centraal 12:0 -> 12:8
Bus 183 Bushalte Leiden Centraal -> Burggravenlaan 12:17 -> 12:26
Journey price: 1.87
```

Figure 11. *Results of Test 1 in my program.* Screenshot

The results are nearly identical (see Figures 10, 11). The only difference is, bus 21 departs 1 minute earlier compared to my program.

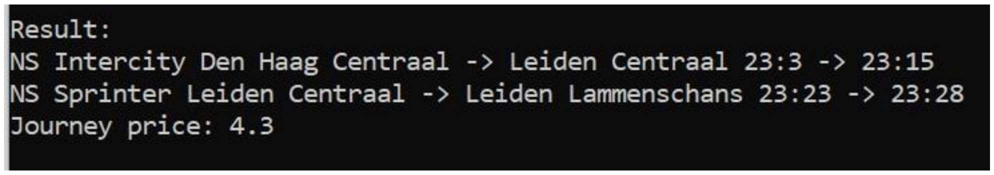
Test 2. Den Haag Centraal to Leiden Lammenschans, departing at 23:00 on Friday, lowest price preferred



23:19 Departure: Platform 7
Train station Den Haag Centraal
NS Sprinter
Direction Almere Oostvaarders
+ Show all intermediate stops
23:37 Arrival: Platform 5b
Train station Leiden Centraal
23:53 Departure: Platform 1
NS Sprinter
Direction Hertogenbosch ('s)
Train station Leiden Lammenschans
23:58 Arrival: Platform 1

Price of your journey
€ 4.30

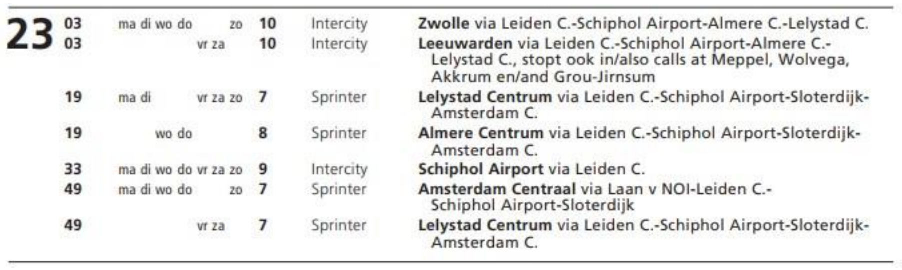
Figure 12. *Results of Test 2 on 9292.* Screenshot



```
Result:
NS Intercity Den Haag Centraal -> Leiden Centraal 23:3 -> 23:15
NS Sprinter Leiden Centraal -> Leiden Lammenschans 23:23 -> 23:28
Journey price: 4.3
```

Figure 13. *Results of Test 2 in my program.* Screenshot

This time, the results are different (see Figures 12, 13). In order to determine the correct travel advice, a timetable can be used, this time provided by NS.

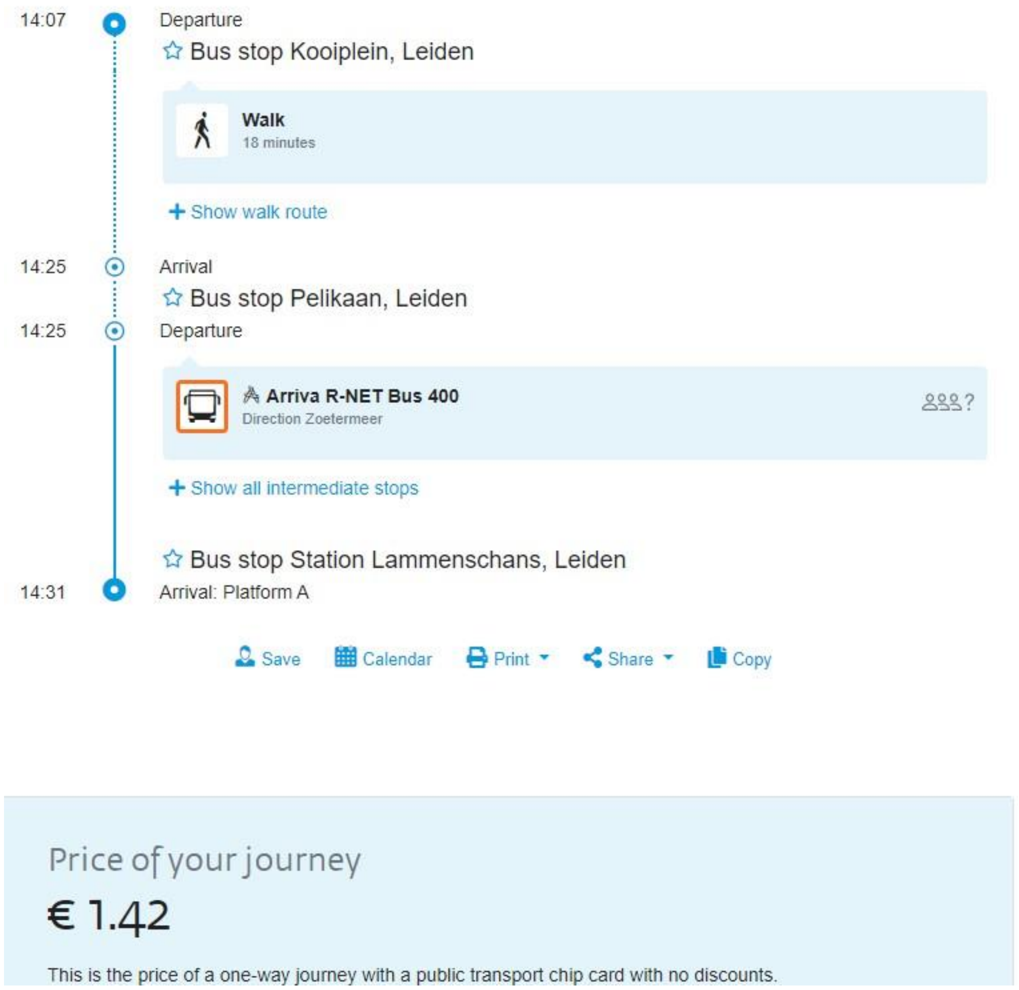


23	03	ma di wo do zo	10	Intercity	**Zwolle** via Leiden C.-Schiphol Airport-Almere C.-Lelystad C.
	03	vr za	10	Intercity	**Leeuwarden** via Leiden C.-Schiphol Airport-Almere C.-Lelystad C., stopt ook in/also calls at Meppel, Wolvega, Akkrum en/and Grou-Jirnsum
	19	ma di vr za zo	7	Sprinter	**Lelystad Centrum** via Leiden C.-Schiphol Airport-Sloterdijk-Amsterdam C.
	19	wo do	8	Sprinter	**Almere Centrum** via Leiden C.-Schiphol Airport-Sloterdijk-Amsterdam C.
	33	ma di wo do vr za zo	9	Intercity	**Schiphol Airport** via Leiden C.
	49	ma di wo do zo	7	Sprinter	**Amsterdam Centraal** via Laan v NOI-Leiden C.-Schiphol Airport-Sloterdijk
	49	vr za	7	Sprinter	**Lelystad Centrum** via Leiden C.-Schiphol Airport-Sloterdijk-Amsterdam C.

Figure 14. Extract of the *timetable of trains departing from Den Haag Centraal to Leiden Centraal and beyond*. Retrieved on August 12, 2023, from NS: https://assets.travelsupport-p.cla.ns.nl/stations/vertrekstaten/gvc2.pdf

It is seen that a train indeed departs to Leiden at 23:03 on Friday (denoted as *vr*) (see Fig. 14). Then, my program provides better travelling advice on this test. 9292 might not have the newest train timetables.

Test 3. Kooiplein to Station Lammenschans, departing at 14:00 on Saturday, minimum number of transfers preferred



14:07 Departure
☆ Bus stop Kooiplein, Leiden

Walk
18 minutes

+ Show walk route

14:25 Arrival
☆ Bus stop Pelikaan, Leiden
14:25 Departure

Arriva R-NET Bus 400
Direction Zoetermeer

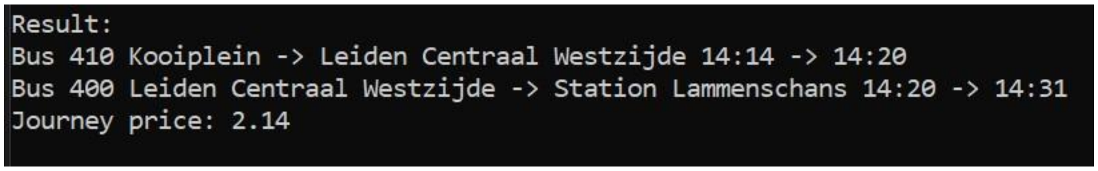
+ Show all intermediate stops

☆ Bus stop Station Lammenschans, Leiden
14:31 Arrival: Platform A

Save Calendar Print Share Copy

Price of your journey
€ 1.42
This is the price of a one-way journey with a public transport chip card with no discounts.

Figure 15. *Results of Test 3 on 9292.* Screenshot

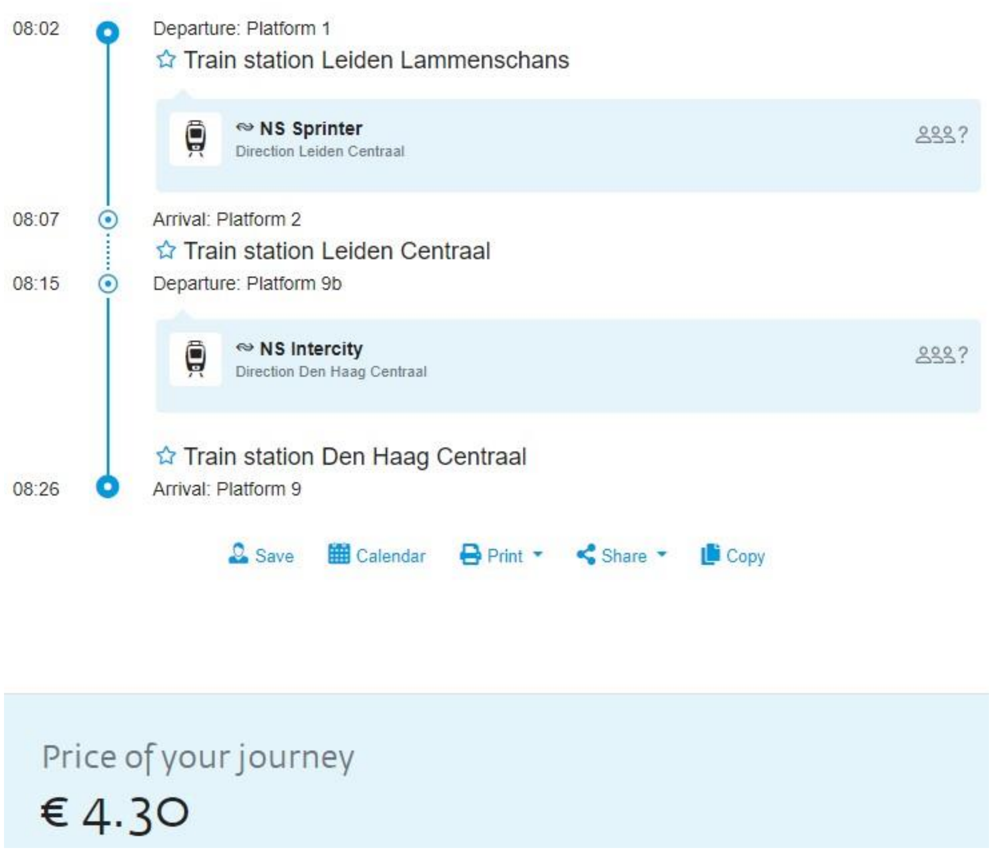


```
Result:
Bus 410 Kooiplein -> Leiden Centraal Westzijde 14:14 -> 14:20
Bus 400 Leiden Centraal Westzijde -> Station Lammenschans 14:20 -> 14:31
Journey price: 2.14
```

Figure 16. *Results of Test 3 in my program.* Screenshot

Here, 9292 proposes a different approach than my program (see Figures 15, 16). It advises walking 18 minutes to a stop (not present in my model) and taking 400 bus to Station Lammenschans. The resulting prices differ; the arrival times do not. Also, the path advised by 9292 has 0 transfers against 1 by my program. The reason is that the Pelikaan bus stop is present in my model, and neither are long walking edges.

Test 4. Leiden Lammenschans to Den Haag Centraal, departing at 8:00 on Sunday, lowest price preferred


08:02 Departure: Platform 1
☆ Train station Leiden Lammenschans
NS Sprinter
Direction Leiden Centraal
08:07 Arrival: Platform 2
☆ Train station Leiden Centraal
08:15 Departure: Platform 9b
NS Intercity
Direction Den Haag Centraal
☆ Train station Den Haag Centraal
08:26 Arrival: Platform 9

Price of your journey
€ 4.30

Figure 17. *Results of Test 4 on 9292.* Screenshot

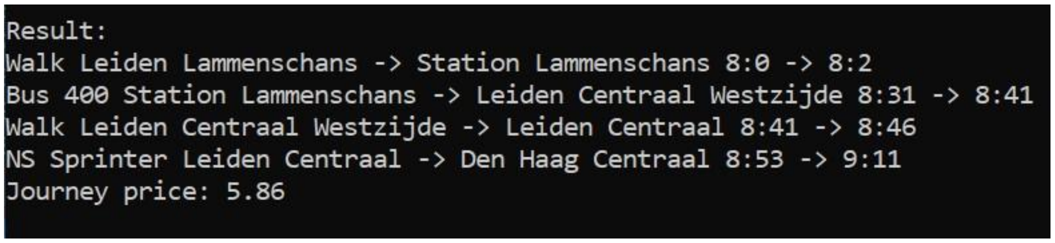
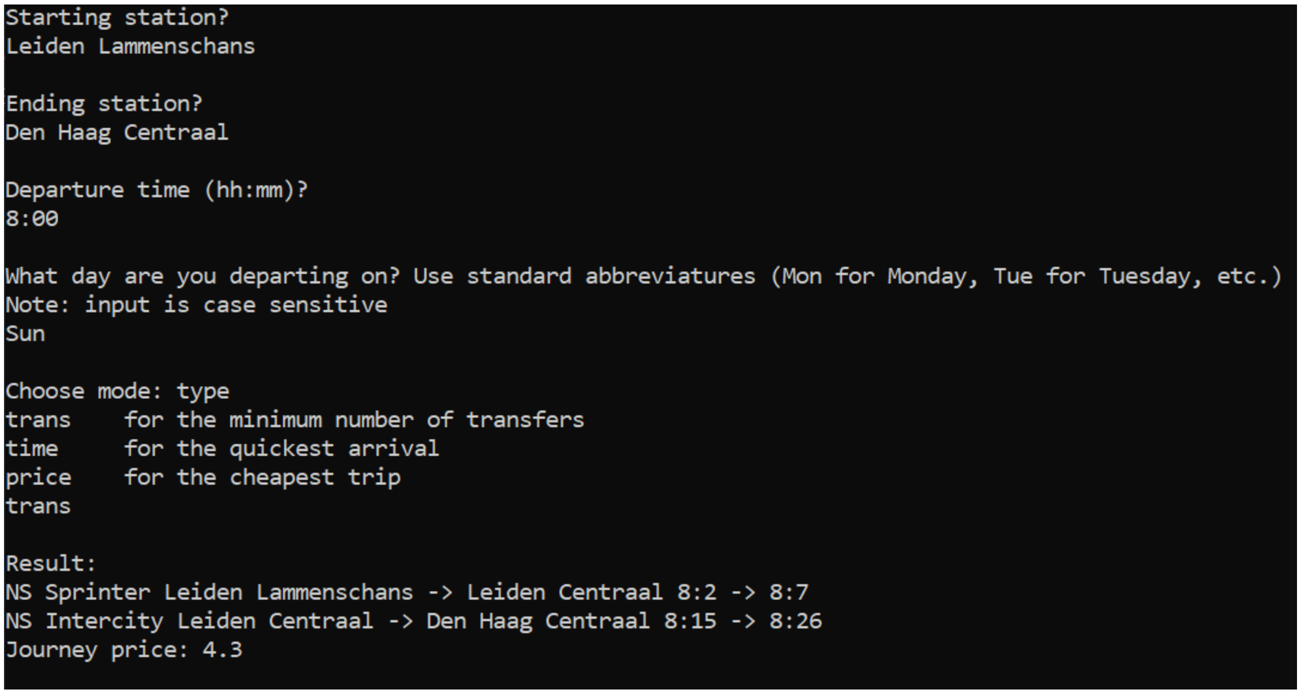

```
Result:
Walk Leiden Lammenschans -> Station Lammenschans 8:0 -> 8:2
Bus 400 Station Lammenschans -> Leiden Centraal Westzijde 8:31 -> 8:41
Walk Leiden Centraal Westzijde -> Leiden Centraal 8:41 -> 8:46
NS Sprinter Leiden Centraal -> Den Haag Centraal 8:53 -> 9:11
Journey price: 5.86
```

Figure 18. *Results of Test 4 in my program.* Screenshot

The results of my program are vastly inferior to the advice 9292 proposes (see Figures 17, 18). Interestingly enough, if one tries transfer mode in my program, the result would be identical to the one of 9292 (see Figure 19).



```
Starting station?
Leiden Lammenschans

Ending station?
Den Haag Centraal

Departure time (hh:mm)?
8:00

What day are you departing on? Use standard abbreviatures (Mon for Monday, Tue for Tuesday, etc.)
Note: input is case sensitive
Sun

Choose mode: type
trans    for the minimum number of transfers
time     for the quickest arrival
price    for the cheapest trip
trans

Result:
NS Sprinter Leiden Lammenschans -> Leiden Centraal 8:2 -> 8:7
NS Intercity Leiden Centraal -> Den Haag Centraal 8:15 -> 8:26
Journey price: 4.3
```

Figure 19. *New setup and results of Test 4 in my program.* Screenshot

Why did my program make a mistake in price mode? To grasp this issue, it is necessary to return to the concept of boarding cost and revisit the principle of the Dijkstra algorithm.

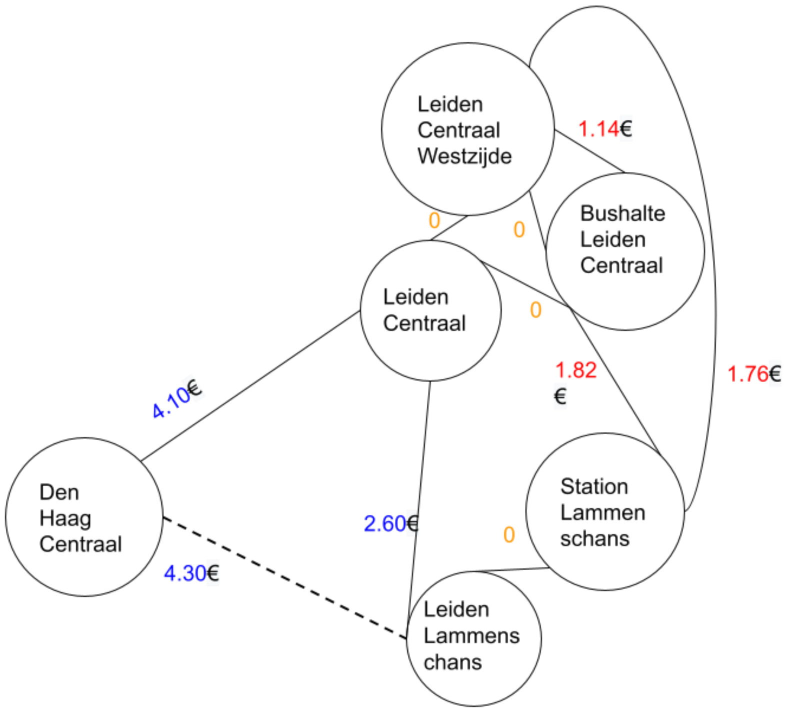


Figure 20. Author. *A small graph to explain the mistake.* Scheme

A small model is provided to explain the mistake (see Fig. 20). One starts from Leiden Lammenschans. According to the Dijkstra algorithm, the minimum price to reach Leiden Centraal is 1.76€. And it is true! However, once one transfers to the train, they must pay a boarding fee of 2.40€. On the other side, if one reaches Leiden Centraal by train, they don't have to pay the boarding fee, therefore, reaching Den Haag Centraal at a lower price of 4.30€. It is shown by the dashed edge between Leiden Lammenschans and Den Haag Centraal. Making such an edge in a computer problem is hardly possible, as it is a combination of two different edges, with no common points in time.

The conclusion follows that, for calculating the path with the lowest price, an optimal local solution does not always lead to an optimal global solution, meaning that a greedy algorithm does not lead to a correct solution on any test. It has been mentioned that the classic Dijkstra algorithm is greedy. Therefore, in order to get the correct result on any test, a different algorithm is needed, or Dijkstra needs to be modified in such a way that it is not greedy and considers more cases. This will inevitably lead to an increase in complexity.

For Test 4, it is not critical, as one can choose different criteria and determine the best route using correct travelling advice focusing on these criteria.

Other limitations of the program

As the custom program is compared against 9292, it is reasonable to mention some other shortcomings of the custom program.

- The application is completely offline, meaning there is no response to delays and strikes.
- The model is limited to a few stops in and around Leiden and a few public transport routes.
- In programming, frontend is the part of the program that interacts with users (design, buttons, etc.) and backend is the part of the program that makes the program operate and is not accessible to users. The frontend part barely exists in my program.
- It is possible to pick an address as a destination or departure point in 9292. In my program, one can only pick stations.

However, trying to resolve these limitations is simply unfeasible in the scope of the Extended Essay.

Conclusion

The public transfer system model can be formalised into a graph with nodes being stations and edges being routes that operate between these nodes. The provided model is a weighted and directed graph. It also contains multiple edges between some nodes. It is reasonable to group edges into routes to simplify the calculation of the arrival times. The prices are influenced by the so-called boarding cost, which is only charged when boarding transport for the first time or when transferring between different types of transport.

The research question was: **what algorithms of graph theory are the least resource-consuming for calculating optimal commuting routes (in terms of spent time, price, or the number of transfers)?**

The breadth-first search can be used to find the minimum number of transfers, since one can get from station A to station B by traversing a single edge, provided that one can get from station A to station B using a single bus or train route.

Dijkstra algorithm is applicable to the problem of calculating the quickest trip between two stations. To make it possible, timetables should be loaded and processed during the execution of the algorithm.

I attempted to use the Dijkstra algorithm to calculate the cheapest path between two stations. Evidently, because of the nature of boarding costs, a locally optimal solution does not necessarily lead to a globally optimal solution. Therefore, a different algorithm or a modification of the Dijkstra algorithm is needed.

The program has been implemented to demonstrate the work of the algorithms. Tests showed that, on a limited model, the program produces generally feasible travel advice in most cases, especially when minimising transfers or prioritising the quickest journey.

Works Cited

- *9292.* https://www.9292.nl

- “Breadth-first search”. Retrieved on February 28, 2023 from *Algorithms for Competitive Programming*: https://cp-algorithms.com/graph/breadth-first-search.html

- “Dijkstra algorithm”. Retrieved on February 28, 2023, from *Algorithms for Competitive Programming*: https://cp-algorithms.com/graph/dijkstra.html

- “Dijkstra on sparse graphs”. Retrieved on February 28, 2023, from *Algorithms for Competitive Programming*: https://cp-algorithms.com/graph/dijkstra_sparse.html

- Author (2023, August 16). “Extended Essay Application”. *GitHub*, https://github.com/jVfiodar/Extended-Essay-Application

- Hauskrecht, M. “Graphs”. Retrieved on February 28, 2023 from *University of Pittsburgh*, https://people.cs.pitt.edu/~milos/courses/cs441/lectures/Class25.pdf

- Bhargavae, A. (2016). *Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People*. Simon and Schuster.

- “How to Iterate Through Directories in C++”. Retrieved on July 30, 2023 from *C++ Stories*: https://www.cppstories.com/2019/04/dir-iterate/

- *NS*. https://www.ns.nl

- Trubetskoy, Sasha. *Roman Roads*. June 3, 2017. Retrieved on March 16, 2023, from *Visual Capitalist*: https://www.visualcapitalist.com/roman-empires-roads-map/

- Sample graph. Retrieved on March 23, 2023, from *StackExchange*: https://softwareengineering.stackexchange.com/questions/288702/how-definite-of-an-order-is-there-to-a-depth-first-search-of-a-graph

- Timetable of trains departing from Den Haag Centraal to Leiden, Haarlem/Schiphol Airport, Amsterdam, Hoorn, Almere, Zwolle, Leeuwarden/Groningen. Retrieved on August 12, 2023, from *NS*: https://assets.travelsupport-p.cla.ns.nl/stations/vertrekstaten/gvc2.pdf

- Timetable of trains departing from Leiden Centraal to Den Haag Centraal, Rotterdam, Dordrecht, Roosendaal, Vlissingen. Retrieved on August 12, 2023, from *NS*: https://assets.travelsupport-p.cla.ns.nl/stations/vertrekstaten/ledn2.pdf

- Timetable of trains departing from Leiden Centraal to Schiphol Airport, Zaandam, Hoorn/Amsterdam/Venlo, Lelystad, Zwolle, Leeuwarden/Groningen, Alphen aan den Rijn, Woerden, Utrecht, s'Hertogenbosch. Retrieved on August 12, 2023, from *NS*: https://assets.travelsupport-p.cla.ns.nl/stations/vertrekstaten/ledn3.pdf

- Timetable of trains departing from Leiden Lammenschans to Leiden Centraal. Retrieved on August 12, 2023, from *NS*: https://assets.travelsupport-p.cla.ns.nl/stations/vertrekstaten/ldl0.pdf

- Timetables for Arriva bus routes. Retrieved on June 15, 2023, from *Arriva*: https://arriva-reisinfo.fis.nl/