# COMP261 Assig 2 Report

## What my code does and doesn't do:

My program:

- Allows the user to specify the start and end nodes
- Highlights the generated route
- Prints out the trip without road duplicates
- Calculates the shortest route for distance or time
- Can consider cycling, walking or driving
- Takes turn restrictions into account
- Can avoid traffic lights
- Doesn't go down one-way roads in the wrong direction

## Pseudocode: (Not up to date)

```
A* pseudocode:

let g = cost from start node to current node
let h = heuristic cost from current node to end node
let f = h + g

Add new fringe element with
    node = start node,
    g = 0,
    f = g + h = h,
    previous = null,
    segment = null

use priority queue with priority given to smallest f value

while fringe is not empty:
    let currentElement = pop node with smallest f value off fringe.

    if currentElement.node is end node, solution found, return currentElement to
be reversed for the path

    if currentElement.node is visited, continue/skip.
    mark currentElement.node as visited

    for each node connectedNode connected to currentNode via segment s
        let newRealCost = currentNode.g + cost to connectedNode
        add new fringe element with
            node = connectedNode,
            previous = currentElement,
            g = newRealCost,
            f = newRealCost + connectedNode.h,
            segment = s
    end for
end while
```

## Description of path and heuristic estimate:

For the distance mode, the program uses the direct distance between the current node and the goal node as the heuristic.

For the time mode, the program uses time = distance/velocity calculation and adds extra cost for things like lower road class or traffic lights. The heuristic assumes that the road class is the best and the speed limit is the highest value from the loaded dataset.

## Testing:

I tested the program by using the small data set and trying a few paths by hand and comparing them to what the algorithm outputted. I also had a look at mapping/route planning services online and compared the results.

## Page 6 answers:

### A*

Going from node D to node H.

Step 0:

Fringe Elements {<D, null, 0, 25>}

Element to visit next: <D, null, 0, 25>

Step 1:

Fringe Elements {

<E, D, 10, 26>, <- new

<F, D, 8, 27>, <- new

<C, D, 14, 51>, <- new

<A, D, 15, 53> <- new

}

Element to visit next: <E, D, 10, 26>

Step 2:

Fringe Elements {

<F, D, 8, 27>,

<H, E, 31, 31>, <- new

<C, D, 14, 51>,

<A, D, 15, 53>

}

Element to visit next: <F, D, 8, 27>

Step 3:

Fringe Elements {

<G, F, 18, 19>, <- new

<H, E, 31, 31>,

<I, F, 23, 39>,

<C, D, 14, 51>,

<A, D, 15, 53>

}

Element to visit next: <G, F, 18, 19>

Step 4:

Fringe Elements {

<H, G, 28, 28>, <- new

<H, E, 31, 31>,

<I, F, 23, 39>,

<I, G, 32, 48>, <- new

<C, D, 14, 51>,

<A, D, 15, 53>

}

Element to visit next: <H, G, 28, 28>

Step 5:

Goal node reached: <H, G, 28, 28>

Reverse path: H -> G -> F -> D

Path: D -> F -> G -> H

## Dijkstra's:

Going from node D to node H.

Step 0:

Fringe Elements {<D, null, 0>}

Element to visit next <D, null, 0>

Step 1:

Fringe Elements {

<F, D, 8>, <- new

<E, D, 10>, <- new

<C, D, 14>, <- new

<A, D, 15> <- new

}

Element to visit next: <F, D, 8>

Step 2:

Fringe Elements {

<E, D, 10>,

<C, D, 14>,

<A, D, 15>

<G, F, 18>, <- new

<I, F, 23> <- new

}

Element to visit next: <E, D, 10>

Step 3:

Fringe Elements {

<C, D, 14>,

<A, D, 15>

<G, F, 18>,

<I, F, 23>,

<H, E, 31> <- new

}

Element to visit next: <C, D, 14>

Step 4:

Fringe Elements {

<A, D, 15>,

<G, F, 18>,

<B, C, 22>, <- new

<I, F, 23>,

<H, E, 31>

}

Element to visit next: <A, D, 15>

Step 5:

Fringe Elements {

<G, F, 18>,

<B, A, 22>, <- new

<B, C, 22>,

<I, F, 23>,

<H, E, 31>

}

Element to visit next: <G, F, 18>

Step 6:

Fringe Elements {

<B, A, 22>,

<B, C, 22>,

<I, F, 23>,

<H, G, 28>, <- new

<H, E, 31>,

<I, G, 32> <- new

}

Element to visit next: <B, A, 22>

Step 6:

Fringe Elements {

<B, C, 22>,

<I, F, 23>,

<H, G, 28>,

<H, E, 31>,

<I, G, 32>

}

C already visited so not added.

Element to visit next: <B, C, 22>

Step 7:

Fringe Elements {

<I, F, 23>,

<H, G, 28>,

<H, E, 31>,

<I, G, 32>

}

B now already visited so skipped.

Element to visit next: <I, F, 23>

Step 8:

Fringe Elements {

<H, G, 28>,

<H, E, 31>,

<I, G, 32>,

<G, I, 37> <- new

}

Element to visit next: <H, G, 28>

Step 9:

Goal node reached: <H, G, 28>

Reverse Path: H -> G -> F -> D

Path: D -> F -> G -> H

The reason that A* takes fewer steps than 1-1 Dijkstra's is because A* has some sense of getting closer to the goal, or the direction that it is going in.

This means that it tends not to visit nodes that are not in the right direction unless all other options have been exhausted. Dijkstra's algorithm is more of a brute force approach.