**Discussion Disclosure**

Sections of the assignment were discussed with:

* Munro Forgan

**Sudocode:**

A\* Search Sudocode:

Input: A weighted graph representing the road network of Auckland, A start node representing the starting intersection, A goal node representing the destination intersection and a heuristic function for each node

Output: the shortest path from the starting intersection/node to the destination intersection/node

Initially while all of the nodes in the graph are unvisited and the fringe has a single element (the starting intersection)

While(the fringe is not empty and the route has not been found){

Set current node to the item in the fringe with minimal f\*

Add current node to the visited;

If( current node is the destination){

Set endNodeFound to true;

}

For(every road starting or ending in node\*){

If( the road is oneway and node\* ends there){

Check next road

}

Finds the node at the other end of the road and sets node\* to that;

Calculates the distance travelled to get to node\* and sets g\*

Calculates the remaining distance to the end node from node\* and sets H\*

If(node\* has already been visited or the new distance to the end node is greater than prev\* try the next neighbour );

If(the fringe does not contain node\* and the new distance to the endnode is smaller than the previous){

Add <node\*,current node,g\*,f\*> to the fringe

}

}

}

Generate set to store articulation points (AP)

**findAllArticulationPoints{**

Set all nodes depth to infinity;

Set number of subtrees of all nodes to 0;

Initializing a array containing all nodes in the graph as a list of unvisited nodes;

While(there is nodes left that are not visited){

Chooses a random node and sets as root;

Removes root from list of unvisited nodes;

For(all neighbouring roads ){

Find node at end of road and set is as neighbour;

Checks if neighbour starts or terminates at root;

Removes neighbour from unvisited;

If(the neighbours depth has not been set){

Call recursive findAllArticulationPoints(neighbour, 1, root)

Increase number of the neighbours subtrees by 1

}

If(the number of the neighbours subtrees is greater than 1){

Add neighbour to AP

}

}

}

}

**RecursiveFindArticulationPoints(currentNode, depth, parentNode ){**

Set depth of currentNode to depth;

Set reachBack to depth;

For(all neighbours of currentNode){

Determines if the neighbour starts or ended at currentNode;

If(the neighbour is the parent){

skip this neighbour;

}

Remove neighbour from unvisited nodes;

If(this neighbour’s depth has been set){

change reachback to the smallest of the two;

}

// case 2: indirect alternative path: neighbour is an unvisited child in the same sub-tree

// calculate alternative paths of the child, which can also be reached by itself

childReach = recursiveFindArticulationPoints(neighbour, depth + 1, node);

set reachback to the smallest of childreach and reachback

if(childReach is smaller than or the same size as depth){

add currentNode to articulation points

}

}

**}**