The Auckland Road System

In this assignment, you will continue with the Road System for Auckland, by adding a functionality for critical intersection identification with the *Articulation Points* algorithm, and another functionality for finding the *minimum spanning tree*.

For this assignment, you may reuse the core part of your program or the template code from assignment 2. The datasets and template code are reused here, and provided here again.

Resources

The assignment webpage also contains:

- An archive of the template code and a small example.
- An archive of the road data files.
- The marking guide.

To submit

You should submit four things:

- All the source code (.java files) for your program, including the template code if you use it. **Please make sure you do this**, without it we cannot give you any marks. **Again: in addition to the runnable `.jar` file, submit all your `.java` files**.
- Any other files your program needs to run that *aren't* the data files provided.
- A report on your program to help the marker understand the code. The report should:
  - describe what your code does and doesn't do (e.g., which stages did you do).
  - give a detailed pseudocode for the algorithms (articulation points and minimum spanning tree).
  - outline how you tested that your program worked.
  - Your answers to the questions from page 5.

  The report should be clear, but it does not have to be fancy – very plain formatting is all that is needed. It must be either a `txt` or a `pdf` file. It does not need to be long, but you need to help the marker see what you did.

Note that for marking, you will need to sign up for a 15 minute slot with the markers.

Critical intersections

The first feature is an analysis tool that might be used by emergency services planners who want to identify every intersection that would have bad consequences for emergency services if it were blocked or disabled in some way. An intersection that is the only entrance way into some part of the map is an critical intersection ('articulation point'). Your program should identify all such intersections and colour highlight them. Note that emergency services don't care about one way roads - in an emergency they can go either way, if necessary. They also don't care about 'no right turn' restrictions.

The articulation points algorithm assumes that the graph is undirected, doesn't care about the lengths of edges, and doesn't care whether there are multiple edges between two nodes. In fact, all it needs is a collection of nodes and the set of neighbouring nodes of each node. This means that you cannot use exactly the same data structures as you used for the route finding algorithm. For the route finding, each node (intersection) needed a set of the edges (road segments) coming out of the node, where the segments included the length and the node at the other end; for the articulation points, each node needs a set of neighbouring nodes (i.e. the nodes at the other end of segments both in and out of the node). You should extend your program to build this structure as it reads the data.

Hint: The graph may be disconnected and contain multiple isolated connected components. Your program should be able to find all the articulation points for all the connected components.

Hint: There are **240** articulation points in the small graph, and **10853** articulation points in the large graph.

Minimum spanning tree

The second feature is to find the minimum spanning tree of the graph. This could be used for network design, such as building a power supply network between the traffic lights of the intersections and a power station with the minimum cost. As the articulation points, the minimum spanning does not consider one way roads either. However, unlike articulation points, it cares about the edge weights to minimise the total weights of the spanning tree. So you need to modify your data structure for finding the minimum spanning tree as follows: *add the edge weight to each neighbouring node (this can be done by creating a new* `MSTNode` *class, which includes the fields of the neighbouring node and its corresponding edge weight). If there are multiple edges between two nodes, regardless of their direction, the minimum weight is stored.*

Hint: The graph may be disconnected and contain multiple isolated connected components. Your program should be able to find the minimum spanning trees for all the connected components.

Your program

You are suggested to solve this problem in stages as below, the provided marking guide gives a detailed breakdown of core/completion/challenge.

Minimum – Articulation points.

- Ensure that, from a given `Node`, you can quickly find all the adjacent `Nodes`.
- Implement the articulation points algorithm to find *all* the nodes that are articulation points, then highlight them. This can be done with either the recursive or iterative version of the algorithm, but the iterative version is worth more marks (see the completion part of the marking guide).
- The large graph is disconnected. So your program should be able to find all the articulation points for all the connected components.

  Hint: The report you write must include a detailed pseudocode specification of the algorithm you used. Write this detailed pseudocode algorithm before you try to code it up! And then update the pseudocode if you have to change the code later. It is seldom a good idea to launch into coding of this kind of program without working through the detailed design first.

Core – Minimum spanning tree

- Ensure that, from a given `Node`, you can quickly find all the adjacent `Nodes` with the edge weight to it. If there are multiple edges to the node, only store the minimum weight.
- Implement the algorithm to find the minimum spanning tree, and highlight it. This can be done by either Prim's algorithm or Kruskal's algorithm. You will get extra marks for using *Kruskal's algorithm with disjoint set data structure* (see the completion part of the marking guide).
- The large graph is disconnected. So your program should be able to find all the MSTs for all the connected components.

  Hint: again, write up a detailed pseudocode algorithm.

Completion – Questions.

- Answer the question about articulation points.
- Answer the question about minimum spanning tree.
- Answer the question about Fast Fourier Transform.

Challenge – Advanced articulation points and MST.

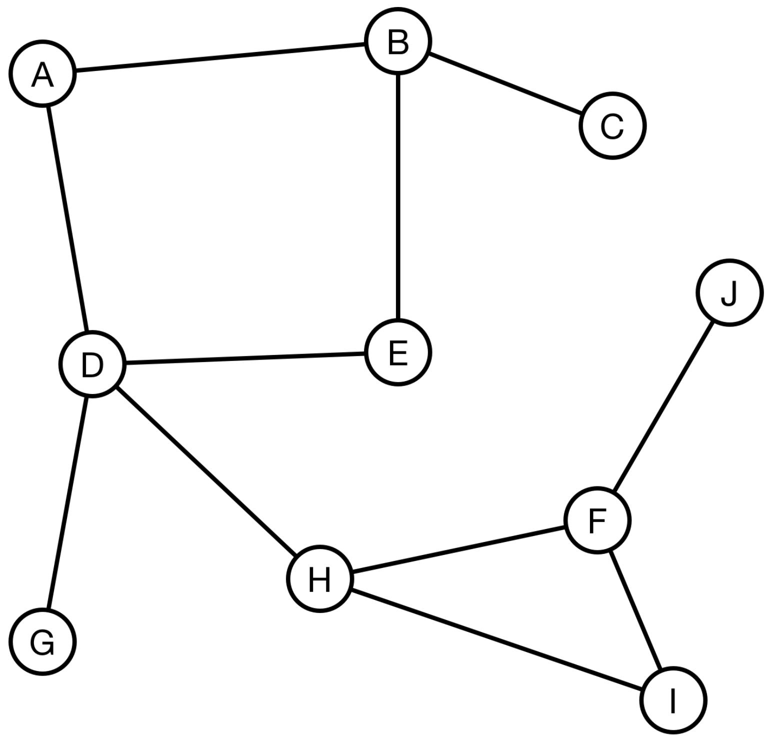
- Use the iterative version of articulation points algorithm. This is harder to understand and write than the recursive version of the algorithm.

- Use the disjoint set data structure to implement Kruskal's algorithm. Note that the Kruskal's algorithm without the disjoint set data structure cannot get the extra marks.

Question

Question 1. (Articulation points)

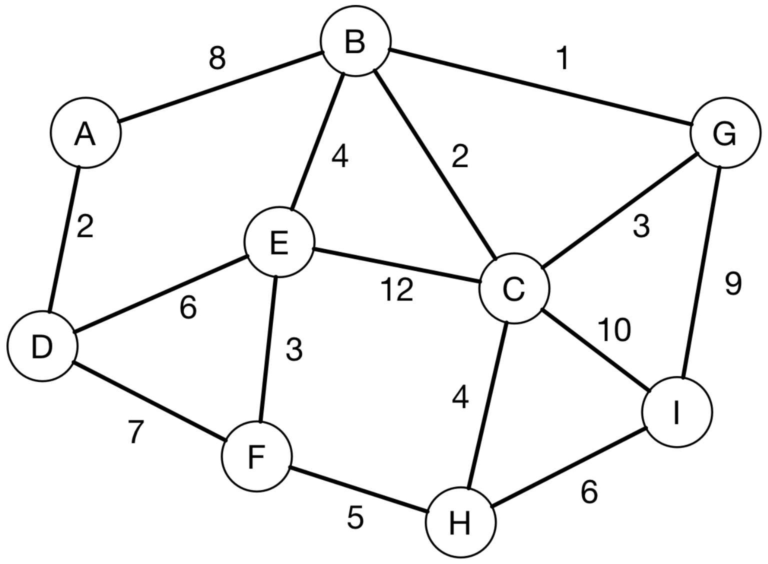
We are trying to find all the articulation points for the graph below by the articulation points algorithm. Assume that we start from node A, and the depth of A is set to 0. When visiting neighbouring nodes, we follow the alphabetic order, i.e. if we have neighbours {B, D}, we will visit B first.



- (7 marks) Write the depth and reachBack of all the nodes.
- (3 marks) List all the articulation points. For each articulation point, explain which criteria the articulation point algorithm used to identify it.

Question 2. (Minimum spanning tree)

Consider the following undirected and weighted graph, we are trying to find the MST for the graph.



- (5 marks) If we use Prim's algorithm, starting from node A, list the edges in the order that they are added to the MST.

  Each edge is represented as the two nodes that they connect and the edge weight: e.g. "EF 3" for the edge between E and F with weight 3.

  To make the answer unique and easy to mark, list the nodes in each edge in *alphabetical* order (e.g. DE not ED).

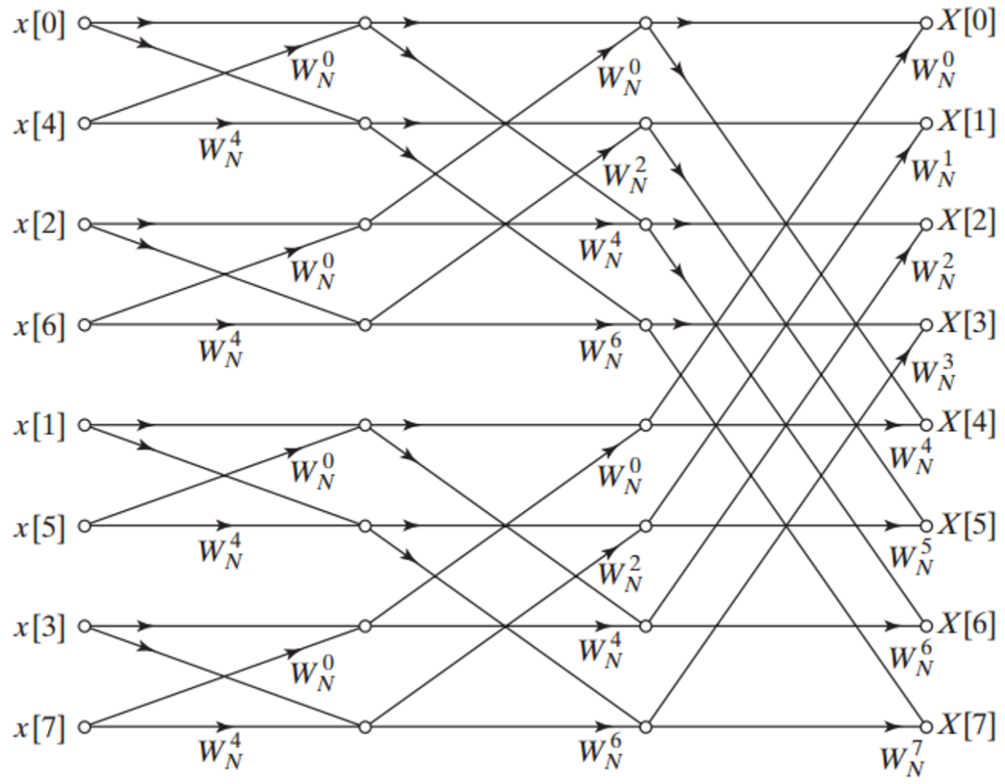
- (5 marks) If we use Kruskal's algorithm with disjoint set data structure, draw the final tree *in the disjoint set* (**NOT the final MST!**).

  Note: The disjoint set data structure has three methods: MakeSet(x), Find(x) and Union(x,y). Union(x,y) never merges deeper trees into shorter trees.

  Note: When calling Union(x,y), x and y always follow *alphabetical* order. For example, we always call Union(D,E), but never call Union(E,D).

Question 3. (Fast Fourier Transform)

The divid-and-conquer process of the Fast Fourier Transform (FFT) process of a 8-point time series (elements could be complex) is shown as follows, where $W_N^k = e^{-i\frac{2k\pi}{N}}$.



- (8 marks) Calculate the total number of complex multiplications and additions of the above FFT process (the lecture slides already showed that the normal Fourier Transform process requires 64 complex multiplications and 56 complex additions). Show your working.
- (7 marks) In the above example, the inputs are reordered as $(x[0], x[4], x[2], x[6], x[1], x[5], x[3], x[7])$. For a 16-point time series $(x[0], x[1], \ldots, x[15])$, write the reordered list of the inputs.