Algorithms & Analysis: Assignment 2 Report

Task A & B:

For these tasks, we chose a Coordinate representation(*), where "edge weights" are represented by the terrain cost. For all inbound edges, the weight is the coordinate terrain cost. Each coordinate has 1 value for upto 4 edges as opposed to multiple inbound edges to represent the same weight.

Coordinates directly adjacent in NORTH, SOUTH, EAST, WEST directions are called neighbours. VALID neighbours are IMPASSABLE and UNVISITED.

Using buildLinks() every Coordinate is linked to a neighbour with the least overall path cost (called minPathCost) until the goal coordinate is reached. We wait until all relevant coordinates have a link BEFORE generating the path.

Explored coordinates:

- visited: a List of visited coordinates, with the last visited Coordinate's neighbours being added to toVisit.
- toVisit: a "waiting list" of coordinates sorted by their minPathCost. The coordinate with the lowest minPathCost is moved into visited.

When generating the path, all links are connected from the goal -> start node.

Task C:

Starts or goals (nodes) were abstracted into a graph representation where "edges" were the least cost path between any 2 nodes. The edge weights was the least cost of the path. These abstracted edges are called NodeEdges.

Hence, we were able to implement this by checking all (start/node) tuple possibility, and comparing edge weights to find the cheapest NodeEdge path.

In buildLinks(), we chose to adopt a clean-slate approach. Explored coordinates (visited + toVisit) were reset before finding another path, avoiding the build process from being cut short from an already visited goal. Setting any 2 nodes as the start and goal, will recreate an "edge" between. This way, all edges between each node is created.
As a result, each node can simultaneously form an edge to every other node, AND a path won't overlap itself.

Task D(**):

As an extension of the travelling salesman problem (transform-and-conquer), our choice was to brute force the solution with O((n-2)!) time.

In addition to starts and goals, waypoints were considered as nodes. Node paths from WAYPOINT permutations to ensure each edge is only calculated ONCE. Rather than going through ALL NODE permutations and choosing orders where the start and goal are the first/last nodes in the order.

This reduces the cost from:	O(n!) complexity	(+ overhead to remove invalid orders)
To:	O((n-2)!) complexity	(avoid including start and goal node in permutations)

To facilitate the logic from previous, we created an Order class to facilitate the logic from c). As an abstraction of NodeEdge, we can retrieve the optimal path by traversing the best order of nodes, whose edges yield the least cost path. Now we can turn DijkstraPathFinder into an abstraction of the Order class, comparing all orders to find the order, and hence the path with the least cost.

*Node: Task D, expands this into a graph representation
**Node: No other links to code/articles referenced since being mainly developed by ourselves, after working on problems from https://projecteuler.net

Report by Reuben Abraham (s3717497) Jeremy Quintana (s3719476)