Project: Heuristic algorithm development

For your project this year, unfortunately social distancing has required us to move away from using the micro:bit devices. (The planned activity involved network communication over the device radios, which in turn would have required physical proximity.) In lieu of this, we have a different, and timely, network-based problem for you to address:

OPERATION: VACCINATION

At long last, after many months of New Zealand's isolation from the world, a vaccine for the novel coronavirus has been developed! A first batch will soon be arriving at Auckland Airport from the overseas facility that produced it, and you have been tasked with distributing it to the most vulnerable population in Auckland: the elderly living in residential care settings. You have a team of four couriers available to do this distribution, and your job is to get the vaccine delivered from the airport to all 143 rest homes in the city, as quickly as possible.

There is a catch, however: due to the severe economic impact of this period of isolation, private travel for your team will not be possible. There's just no money in the budget for petrol, unfortunately. Instead, you will need to figure out how to deliver the vaccines using the public transportation system.

Figure 1: Your chariot awaits! (Image by Akld guy, from 1.)

The assignment details

You have been provided with this document (README.pdf), a simplified version of the Auckland public transport network (network.graphml), a list of rest

homes (rest_homes.txt), some helper files (project_utils.py, akl_zoom.png), and a document explaining how to work with these data and helper files (GETTING_STARTED.pdf). You need to develop an algorithm that will return four ordered lists of network node names, representing the paths taken by your four couriers. Each path must start at Auckland Airport, and can visit any sub-set of the rest homes so long as each rest home is visited by one of your four paths. Your algorithm must be *non-trivial*—your code should attempt to solve the problem of finding shortest paths, and not just return an arbitrary list or the starting list. It should also not solve the problem by using code you found on the Internet; please only use the recommended Python libraries, including any you have been using already as part of the course.

In addition, we would like you to prepare a short design brief for your chosen algorithm. In it, you should include the following:

1. Describe how your algorithm works, in simple terms.
2. Explain why you chose this algorithm.
3. State the total computation time required to find a solution.
4. Give the distance travelled by each of your four couriers.

This design brief should be no more than 1/2 page (250 words), written in paragraph form.

Because this project is an open-ended problem, we will introduce a competitive element for extra credit. Your algorithm will be judged based on minimising the longest distance any of your couriers travels (the length of the longest path among your four paths).

However, one consequence of this assignment being open-ended is that you may find yourself investing more time than it's worth to try to squeeze out extra performance. Don't fall into the trap of diminishing returns! We expect that you should spend no more than about 10 hours on this project. If you are having trouble, please get in touch with the course staff to get help. If you are really keen and want to spend many hours optimizing your solution, take a step back and remember that you do have other courses to work on...

Strategy hints

This problem is a generalization of the classic travelling salesman problem, in which the shortest path that visits a set of locations exactly once is sought. There is no quick way to exactly solve this problem; the fastest known algorithms are $O(n^2 2^n)$, and rapidly become impractical as the number of destinations grows.

Instead, *heuristic* algorithms are used, which find approximate solutions quickly. A common, and relatively simple, type of heuristic that can be used is a *greedy* algorithm: one that proceeds step-by-step, choosing the best option for each step and hoping this is similar to the best option overall. For example, you might always choose to visit the closest remaining destination next (the nearest

Figure 2: A travelling salesman, in an age before algorithm. (Image by Jan Saudek, from 2.)

neighbour algorithm). Other, more complex options are available, which can yield somewhat better results.

For this problem, you also need to decide how to partition the rest homes between your four couriers. You might choose to group the rest homes by geography, or by direction, for instance.

For the ambitious, it is common to approach a problem like this with *meta-heuristics*: you try one approach, and then use a different algorithm to tweak that approach and find improvements. Examples include genetic algorithms and simulated annealing. You will study these more in later courses.

Where should I begin?

0. Try the examples given in GETTING_STARTED.pdf.
1. Create a shortened list of rest homes, perhaps only 3 or 4 of them, that you can use for initial code development.
2. Write a function that, given the network, a starting node, and a list of possible destination nodes, finds the node from the list that is closest to the starting node in the network (i.e. the shortest length of the shortest path).
3. Use this function to implement the nearest-neighbor algorithm:
    a. Find the nearest node to the starting node.
    b. Remove this node from the destination list, and add it to a path list.
    c. Add the distance to this node to a running total of path length.
    d. Set the new node as the starting node.
    e. If the destination list is not empty, go to step a.
    f. Return the list of nodes visited and the total path length.
4. Once you are convinced this works as expected, try it on larger subsets of the full list of rest homes. Ensure that your program saves its output to a text file. (It will take a while (at least 15 minutes) to do the full list itself!)
5. Write a function that splits the rest home list into one list east of a given longitude and one list west of that longitude.

6. Use your nearest-neighbor algorithm to find routes in each of these two sub-lists, using the longitude of the airport. How much shorter are these two routes?
7. Write a similar function that splits north/south based on latitude, and use it to split your East and West lists and obtain four sub-lists, one for each courier.
8. Check that you have a valid solution to the problem, and then start experimenting!

What if I'm lost?

If you can't get the program outlined above to work, please ask the course staff for help! If you're having trouble improving on the method outlined above, note that the competition is optional. You can receive full credit even if your solution to the path-planning problem is very sub-optimal; any non-trivial solution that visits all of the rest homes will meet the basic requirements.

Submission instructions

For the project, please submit the following 10 files to Canvas. **DO NOT** put your submission in a zip archive, or submit any other files. Please **do not** submit a bitbucket repository for this assignment.

- project_code.py (your code)
- path_1.txt, path_2.txt, path_3.txt, path_4.txt (your four paths)
- path_1.png, path_2.png, path_3.png, path_4.png (images of your four paths)
- design_brief.pdf (your design brief)

The path images can be generated using the helper files, as explained in GETTING_STARTED.pdf.

Rubric

Criteria	Mark	Condition
Function	0	Not attempted, nothing working.
	6	Code runs and generates the same output as was submitted.
Style	0	No comments or docstrings.
	1	Unclear or insufficient comments.
	2	Cannot be faulted.
Design	0	Not provided.
Brief	2	Unclear and/or incomplete
	4	Clearly explained, with all elements present.
Contest	2	See below.

Please note that the 6 marks for code function are an all-or-nothing mark; we will not attempt to de-bug your code, you must submit code that runs and generates output that matches what you submit to receive credit.

Tournament Breakdown

Marks will be allocated based on a ranking of distances across the class, with the shortest max distance in first place:

Ranking	Mark
1 (Champion)	2.0
2	1.75
3	1.5
4-10	1.0
11-20	0.5
21+	0

Marking Notes

You may notice that this project can generate more than 12 marks. The additional marks of this project will be honoured as plussage for the lab section of this course.