Exercise 12 – Dijkstra and A* algorithms

Overview

In this exercise, we are going to look at how the Dijkstra's and the A* algorithms can be applied to solve shortest-distance problems on graphs. In the last part of the exercise, you will additionally be asked to implement these algorithms in Python.

Q1 Dijkstra's algorithm

It has been only few months since you started your internship at the European Space Agency, but you are making very good progress. As a reward, you have been assigned to a team that is working on the design of a secret Rover that can autonomously cover very large distances on the planet Mars. Your current task for today is to help the robot find the shortest path to navigate between different points of interest on the planet. Since you took the Autonomous Mobile Robots course, you are confident that you can find an initial easy solution to this problem just by using the following map, which highlights the points of interest and a distance metric between them:

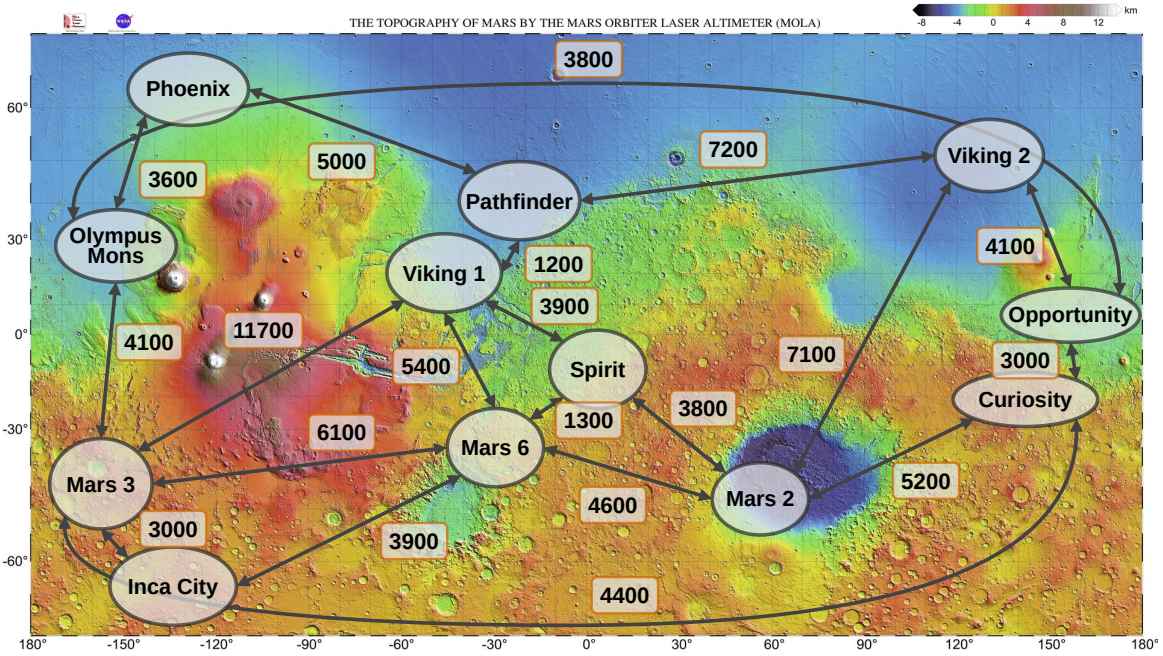

Figure 1: Map of the surface of Mars, including the locations of past and current Mars missions with fictitious costs to move between them. The costs are estimated based on true distance values in kilometers (source: https://nssdc.gsfc.nasa.gov/planetary/mars_mileage_guide.html), increased to account for detours, terrain, weather, etc. Map source: https://attic.gsfc.nasa.gov/mola/images.html.

1. What properties does a graph need to have, so that Dijkstra's algorithm can be applied and is guaranteed to terminate with the optimal path between two nodes on the graph?

2. Assuming the graph given in Figure 1, write down the adjacency list of each node, including the edge weight for each element. You do not need to sort the elements by weights.

3. Execute Dijkstra's algorithm manually to obtain the shortest path from "Mars 3" to "Pathfinder".

Answer:

1. The edge weights of the graph need to be positive.

2. See Table 1.

3. See Table 2. The shortest path is: Mars 3 → Mars 6 → Spirit → Viking 1 → Pathfinder. In the accepted list, entries that occurred in three consecutive steps were omitted for brevity. In the exam, you would always need to spell out the whole accepted list (no omissions permitted). Don't worry, the problem in the exam question will be shorter than this one here. Furthermore, to save space, entries in the "frontier" list in Table 2 are updated in place. This update step is sub-optimal, because it requires searching through the "frontier" list, which has a worst-case complexity linear in the number of open nodes N. If we were to implement the algorithm, a faster alternative would be to simply add new entries for all the nodes encountered in the current iteration and use a priority queue as underlying data structure. This would have a complexity for the insertion of a new entry logarithmic in N, and a constant complexity for the retrieval of the current best entry.

Node	Adjacent nodes - Format: (adjacent node, edge weight)
Viking 1	(Pathfinder, 1200), (Spirit, 3900), (Mars 3, 11700), (Mars 6, 5400)
Viking 2	(Opportunity, 4100), (Mars 2, 7100), (Pathfinder, 7200)
Phoenix	(Pathfinder, 5000), (Olympus Mons, 3600)
Olympus Mons	(Phoenix, 3600), (Opportunity, 3800), (Mars 3, 4100)
Mars 2	(Mars 6, 4600), (Spirit, 3800), (Viking 2, 7100), (Curiosity, 5200)
Mars 3	(Olympus Mons, 4100), (Viking 1, 11700), (Mars 6, 6100), (Inca City, 3000), (Curiosity, 4400)
Mars 6	(Inca City, 3900), (Mars 3, 6100), (Viking 1, 5400), (Spirit, 1300), (Mars 2, 4600)
Inca City	(Mars 3, 3000), (Mars 6, 3900)
Spirit	(Viking 1, 3900), (Mars 2, 3800), (Mars 6, 1300)
Pathfinder	(Phoenix, 5000), (Viking 1, 1200), (Viking 2, 7200)
Opportunity	(Viking 2, 4100), (Curiosity, 3000), (Olympus Mons, 3800)
Curiosity	(Opportunity, 3000), (Mars 2, 5200), (Mars 3, 4400)

Table 1: Solution to Q1.2: Adjacency list.

Q2 A* algorithm

You have found out the above map only contains a subset of the points of interest that you should take into account, and you would like to find a faster way to compute the distance between two given points. Fortunately, you remember another method that you learned during your AMR class, namely the A* algorithm.

1. Explain how you could improve on the speed of Dijkstra's algorithm and clarify how the A* algorithm helps to address this limitation.

2. What condition does a heuristic need to fulfill so that the A* algorithm is guaranteed to return an optimal solution?

3. What would the heuristic need to be for optimal efficiency (i.e. only accepting nodes on the true best path)?

Step	**Accepted**			**Frontier**		
	Node	Arrival cost	Best parent	Node	Lowest arrival cost	Best parent
1	Mars 3	0	-	Inca City	3000	Mars 3
				OM	4100	Mars 3
				Curiosity	4400	Mars 3
				Mars 6	6100	Mars 3
				Viking 1	11700	Mars 3
2	Mars 3	0	-	OM	4100	Mars 3
	Inca City	3000	Mars 3	Curiosity	4400	Mars 3
				Mars 6	6100	Mars 3
				Viking 1	11700	Mars 3
3	Mars 3	0	-	Curiosity	4400	Mars 3
	Inca City	3000	Mars 3	Mars 6	6100	Mars 3
	OM	4100	Mars 3	Phoenix	7700	OM
				Opportunity	7900	OM
				Viking 1	11700	Mars 3
4	...			Mars 6	6100	Mars 3
	Inca City	3000	Mars 3	Opportunity	7400	Curiosity
	OM	4100	Mars 3	Phoenix	7700	OM
	Curiosity	4400	Mars 3	Mars 2	9600	Curiosity
				Viking 1	11700	Mars 3
5	...			Opportunity	7400	Curiosity
	OM	4100	Mars 3	Spirit	7400	Mars 6
	Curiosity	4400	Mars 3	Phoenix	7700	OM
	Mars 6	6100	Mars 3	Mars 2	9600	Curiosity
				Viking 1	11500	Mars 6
6	...			Spirit	7400	Mars 6
	Curiosity	4400	Mars 3	Phoenix	7700	OM
	Mars 6	6100	Mars 3	Mars 2	9600	Curiosity
	Opportunity	7400	Curiosity	Viking 1	11500	Mars 6
				Viking 2	11500	Opportunity
7	...			Phoenix	7700	OM
	Mars 6	6100	Mars 3	Mars 2	9600	Curiosity
	Opportunity	7400	Curiosity	Viking 1	11300	Spirit
	Spirit	7400	Mars 6	Viking 2	11500	Opportunity
8	...			Mars 2	9600	Curiosity
	Opportunity	7400	Curiosity	Viking 1	11300	Spirit
	Spirit	7400	Mars 6	Viking 2	11500	Opportunity
	Phoenix	7700	OM	Pathfinder	12700	Phoenix
9	...			Viking 1	11300	Spirit
	Spirit	7400	Mars 6	Viking 2	11500	Opportunity
	Phoenix	7700	OM	Pathfinder	12700	Phoenix
	Mars 2	9600	Curiosity			
10	...			Viking 2	11500	Opportunity
	Phoenix	7700	OM	Pathfinder	12500	Viking 1
	Mars 2	9600	Curiosity			
	Viking 1	11300	Spirit			
11	...			Pathfinder	12500	Viking 1
	Mars 2	9600	Curiosity			
	Viking 1	11300	Spirit			
	Viking 2	11500	Opportunity			
12	...					
	Viking 1	11300	Spirit			
	Viking 2	11500	Opportunity			
	Pathfinder	**12500**	Viking 1			

Table 2: Solution to Q1.3. “Olympus Mons” is abbreviated with “OM”.

4. Table 3 gives the values of a heuristic for the map from Figure 1, still assuming "Pathfinder" as the goal, and based on the beeline distances between locations[1]. Explain why this heuristic fulfills the conditions you stated in 2. and 3.

5. Using the heuristic from Table 3, execute the A* algorithm manually (similar to Q1.3) to obtain the shortest path from "Mars 3" to "Pathfinder".

Node x	Heuristic value $h(x)$
Viking 1	835
Viking 2	6610
Phoenix	4315
Olympus Mons	5475
Mars 2	4585
Mars 3	5815
Mars 6	2610
Inca City	6030
Spirit	2585
Pathfinder	0
Opportunity	8680
Curiosity	9635

Table 3: Heuristic values for the map in Figure 1, consisting of optimistic estimates (beeline) of the cost to move from the respective location to the goal, "Pathfinder".

Answer:

1. When choosing a node to expand next, Dijkstra's algorithm only considers the cost from the start node to the open nodes. Additionally considering an estimate of the cost from the respective open node to the goal can help focusing the search on more promising paths, leading to fewer nodes being expanded and a solution returned more quickly. A* makes use of such an estimate in the form of a heuristic.

2. The heuristic needs to be *admissible*, i.e. it should never overestimate the true cost of reaching the goal from the respective node. This can be expressed as $h(x) \leq d(x, \text{goal})$, where $h(x)$ is the value of the heuristic for node x, and $d(x, \text{goal})$ is the true minimum cost on the graph for reaching the goal starting at node x. If the heuristic wasn't admissible, the algorithm could terminate assuming that the path to the goal via any of the nodes remaining in the queue is worse than the shortest path found so far, although a shorter path exists, which is however not discovered since the heuristic overestimates the true cost. For the standard form of A* as seen in the lecture, the heuristic must also be *consistent*, that is $h(x) \leq d(x, y) + h(y)$, i.e. the heuristic from node x must not be greater than the cost on the graph from x to a linked node y plus the heuristic from y to the goal, $h(y)$. All consistent heuristics (with $h(x_g) = 0$) are admissible, but not all admissible heuristics are consistent (relatively rare case). In the second case (admissible but not consistent), the algorithm will still run and find the optimal path, but a node would potentially need to be explored multiple times, leading to longer runtimes.

3. The optimal heuristic (minimum run time) is when the heuristic is the true cost on the graph, i.e. $h(x)$ is the actual minimum cost on the graph to reach the goal. Although this would be ideal, calculating this cost is essentially the original problem, so we rarely haves access to an optimal heuristic unless we have already solved the problem.

4. The cost values given in Figure 1 are based on the beeline distances, increased by some amount to account for various factors (see figure caption). Therefore, the beeline distance is an admissible and consistent heuristic. You can verify this by manually checking that $h(x)$ is always less than or equal to the true cost from the node to the goal on the graph.

5. See Table 4. Note how the A* algorithm takes fewer steps until the solution is found. Furthermore, note that similarly to the solution of Dijkstra's, also in this solution the update steps were performed in place. See Answer 3 in Q1 for considerations on how the complexity of this operation could be improved.

[1]The concept of a shortest path on a continuous 3D surface (manifold) is called a *geodesic*. This would be a straight line on a plane. Planets are generally roughly spheres, and geodesics on spheres are so-called 'great circle' paths. For this exercise assume that finding the geodesic distances part is done, and the beeline distances are the shortest possible traversal.

Step	**Accepted**			**Frontier**				
	Node	Arrival cost	Best parent	Node	Lowest arrival cost g	Heuristic cost to goal h	Total $g+h$	Best parent
1	Mars 3	0	-	Mars 6	6100	2610	8710	Mars 3
				Inca City	3000	6030	9030	Mars 3
				OM	4100	5475	9575	Mars 3
				Viking 1	11700	835	12535	Mars 3
				Curiosity	4400	9635	14035	Mars 3
2	Mars 3	0	-	Inca City	3000	6030	9030	Mars 3
	Mars 6	6100	Mars 3	OM	4100	5475	9575	Mars 3
				Spirit	7400	2585	9985	Mars 6
				Viking 1	11500	835	12335	Mars 6
				Curiosity	4400	9635	14035	Mars 3
				Mars 2	10700	4585	15285	Mars 6
3	Mars 3	0	-	OM	4100	5475	9575	Mars 3
	Mars 6	6100	Mars 3	Spirit	7400	2585	9985	Mars 6
	Inca City	3000	Mars 3	Viking 1	11500	835	12335	Mars 6
				Curiosity	4400	9635	14035	Mars 3
				Mars 2	10700	4585	15285	Mars 6
4	...			Spirit	7400	2585	9985	Mars 6
	Mars 6	6100	Mars 3	Phoenix	7700	4315	12015	OM
	Inca City	3000	Mars 3	Viking 1	11500	835	12335	Mars 6
	OM	4100	Mars 3	Curiosity	4400	9635	14035	Mars 3
				Mars 2	10700	4585	15285	Mars 6
				Opportunity	7900	8680	16580	OM
5	...			Phoenix	7700	4315	12015	OM
	Inca City	3000	Mars 3	Viking 1	11300	835	12135	Spirit
	OM	4100	Mars 3	Curiosity	4400	9635	14035	Mars 3
	Spirit	7400	Mars 6	Mars 2	10700	4585	15285	Mars 6
				Opportunity	7900	8680	16580	OM
6	...			Viking 1	11300	835	12135	Spirit
	OM	4100	Mars 3	Pathfinder	12700	0	12700	Phoenix
	Spirit	7400	Mars 6	Curiosity	4400	9635	14035	Mars 3
	Phoenix	7700	OM	Mars 2	10700	4585	15285	Mars 6
				Opportunity	7900	8680	16580	OM
7	...			Pathfinder	12500	0	12500	Viking 1
	Spirit	7400	Mars 6	Curiosity	4400	9635	14035	Mars 3
	Phoenix	7700	OM	Mars 2	10700	4585	15285	Mars 6
	Viking 1	11300	Spirit	Opportunity	7900	8680	16580	OM
8	...			Curiosity	4400	9635	14035	Mars 3
	Phoenix	7700	OM	Mars 2	10700	4585	15285	Mars 6
	Viking 1	11300	Spirit	Opportunity	7900	8680	16580	OM
	Pathfinder	**12500**	Viking 1					

Table 4: Solution to Q2.5. “Olympus Mons” is abbreviated with “OM”.

Q3 Implement the algorithm in Python

Your supervisor is happy that you managed to solve your problem on paper. However, you are now asked to implement your algorithms in Python, providing functions that could efficiently compute shortest distances between many more points on the planet. Specifically, your input data consists of a set of graphs with N nodes and M edges and is provided to you as a text file for each graph (e.g., `example_graph.txt` in the exercise material). The first line of the file contains the two integers N and M. The second line contains two integers representing the index of the start node and of the target node. The remaining M lines each encode an edge in the graph, represented as three integers: the index of the first node in the edge, the index of the second node in the edge, and the weight of the edge. You may assume that the graph is undirected, and that each edge is only encoded once in the text file (i.e., an edge between two nodes i and j will be represented only once, either as `i j distance_between_node_i_and_node_j` *or* as `j i distance_between_node_i_and_node_j`). As an example, assuming that the map from names to node indices is $\{'\text{Viking } 1' \to 0, '\text{Viking } 2' \to 1, '\text{Phoenix}' \to 2, '\text{Olympus Mons}' \to 3, '\text{Mars } 2' \to 4, '\text{Mars } 3' \to 5, '\text{Mars } 6' \to 6, '\text{Inca City}' \to 7, '\text{Spirit}' \to 8, '\text{Pathfinder}' \to 9, '\text{Opportunity}' \to 10, '\text{Curiosity}' \to 11\}$, the graph in Fig. 1 can be represented as:

```
12 20
5 9
0 9 1200
0 8 3900
0 5 11700
0 6 5400
1 10 4100
1 4 7100
1 9 7200
2 9 5000
2 3 3600
3 10 3800
3 5 4100
4 6 4600
4 8 3800
4 11 5200
5 6 6100
5 7 3000
5 11 4400
6 7 3900
6 8 1300
10 11 3000
```

1. Implement the function `dijkstra` in the file `shortest_path.py` attached to this exercise, that runs Dijkstra's algorithm on a given graph and returns the shortest distance and the shortest path between a given start node and a given target node. The function prototype is provided, as well as utility functions to read the file content and convert it to a graph. Once implemented, verify the prediction of your algorithm by running:

   ```
   python find_shortest_path.py --shortest_path_type dijkstra
   ```

   To implement the priority queue, you may use the `heapq` module from the Python Standard Library.

2. Assume now that you are provided also with the 2-D coordinates of each location in the map. Specifically, for each input graph a corresponding coordinate file is also provided (e.g., `example_coordinates.txt` for the `example_graph.txt` graph). The coordinate file contains N lines, with the i-th line containing two floats representing the 2-D coordinates of the i-th location on the map (i.e., the i-th node in the graph). Here we will just consider a 2D planar world (no wrap-around). Implement the A* algorithm, assuming as the heuristic function for each node its Euclidean distance from the target node. In particular, based on the function `dijkstra` from the previous step, implement the function `_find_shortest_path` in the file `shortest_path.py`, according to the explanations provided in the code. This function should support both Dijkstra's and A* algorithm, running A* when the input coordinates are provided and Dijkstra otherwise. Once implemented, verify the prediction of your algorithm by running:

```
python find_shortest_path.py --shortest_path_type a_star
```

3. Assume now that you want to compute the shortest distance and the shortest path from a given start node to *all* the other nodes in the graph. Implement the function `dijkstra_multi_target` by adapting the functions from the previous steps according to the explanations provided in the code. Once implemented, verify the prediction of your algorithm by running:

   ```
   python find_shortest_path.py --shortest_path_type dijkstra_multitarget
   ```

Answer: See the separate file `shortest_path_solution.py`.