NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ3005 Artificial Intelligence

Assignment 1

Lab group: FS3

Group Name: All Stars

<u>Team members:</u>

Members	Matriculation Number
Anil Ankitha	U1923151C
Song Yiming	U1722427F
Tony Ho Man Tung	U1922328C

Task 1

Approach

For Task 1, we have implemented the Dijkstra’s algorithm which is optimal for finding the shortest path between one node to the next.
According to Dijkstra's algorithm, it advances away from the beginning node by visiting the next node with the lowest weight, and this process is continued until the next node with the lowest weight is the end node. Dijkstra’s is an uninformed search algorithm which is optimal compared to BFS and UCS as it does not rely on any heuristic function. Additionally, BFS fails in weighted graphs and all nodes in Dijkstra's are initially inserted into a set, whereas nodes in UCS are inserted lazily.
As per project guidelines using the dictionary from “Dist.json”, we create a data structure of type Graph(), where the nodes are points and edge weights are the distance obtained from the file. In our implementation, we record the possible next destinations and the total weight to visit that destination for each node that we visit. If we've seen a destination before and the weight to visit is lower than it was before, this new weight will take its place [1]. We keep reviewing until the destination node weight has the lowest overall weight of all available possibilities. Finally, we obtain the shortest path between two nodes as we move backwards and find all the nodes along this path.

Output

```
Input Start Point: 1
Input End Point: 50
Loading task...

Shortest path:  1 -> 1363 -> 1358 -> 1357 -> 1356 -> 1276 -> 1273 -> 1277 -> 1269 -> 1267 -> 1268 -> 1284 ->
1283 -> 1282 -> 1255 -> 1253 -> 1260 -> 1259 -> 1249 -> 1246 -> 963 -> 964 -> 962 -> 1002 -> 952 -> 1000 ->
998 -> 994 -> 995 -> 996 -> 987 -> 988 -> 979 -> 980 -> 969 -> 977 -> 989 -> 990 -> 991 -> 2369 -> 2366 -> 2340
-> 2338 -> 2339 -> 2333 -> 2334 -> 2329 -> 2029 -> 2027 -> 2019 -> 2022 -> 2000 -> 1996 -> 1997 -> 1993 ->
1992 -> 1989 -> 1984 -> 2001 -> 1900 -> 1875 -> 1874 -> 1965 -> 1963 -> 1964 -> 1923 -> 1944 -> 1945 -> 1938
-> 1937 -> 1939 -> 1935 -> 1931 -> 1934 -> 1673 -> 1675 -> 1674 -> 1837 -> 1671 -> 1828 -> 1825 -> 1817 ->
1815 -> 1634 -> 1814 -> 1813 -> 1632 -> 1631 -> 1742 -> 1741 -> 1740 -> 1739 -> 1591 -> 1689 -> 1585 -> 1584
-> 1688 -> 1579 -> 1679 -> 1677 -> 104 -> 5680 -> 5418 -> 5431 -> 5425 -> 5424 -> 5422 -> 5413 -> 5412 ->
5411 -> 66 -> 5392 -> 5391 -> 5388 -> 5291 -> 5278 -> 5289 -> 5290 -> 5283 -> 5284 -> 5280 -> 50

Shortest Distance:  148648.63722140013

Completed task.
```

Task 2

Approach

The approach and implementation of Task 2 is similar to Task 1 with the exception of having the edge values of the graph is a list of the form [Distance, Cost] between two nodes from the files “Dist.json” and “Cost.json”. Additionally, the energy budget is checked along with the distance between the nodes before checking the shortest distance between them. If the total cost weight has exceeded the budget then the loop is called to continue or else the checking of distance weight can continue.

For this example, where start node is 1 and end node is 50, the nearest shortest distance is noticeably greater than that of Task 1 provided that the energy constraint has been met.

Output

```
Input Start Point: 1
Input End Point: 50
Input Budget of Path: 287932
Loading task...

Shortest path:  1 -> 1363 -> 1358 -> 1357 -> 1356 -> 1276 -> 1273 -> 1277 -> 1269 -> 1267 -> 1268 -> 1284 ->
1283 -> 1282 -> 1255 -> 1253 -> 1260 -> 1259 -> 1249 -> 1246 -> 963 -> 964 -> 962 -> 1002 -> 952 -> 1000 ->
998 -> 994 -> 995 -> 996 -> 987 -> 988 -> 979 -> 980 -> 969 -> 977 -> 989 -> 990 -> 991 -> 2369 -> 2366 -> 2340
-> 2338 -> 2339 -> 2333 -> 2334 -> 2329 -> 2029 -> 2027 -> 2019 -> 2022 -> 2000 -> 1996 -> 1997 -> 1993 ->
1992 -> 1989 -> 1984 -> 2001 -> 1900 -> 1875 -> 1874 -> 1965 -> 1963 -> 1964 -> 1923 -> 1944 -> 1945 -> 1938
-> 1937 -> 1939 -> 1935 -> 1931 -> 1934 -> 1673 -> 1675 -> 1674 -> 1837 -> 1671 -> 1828 -> 1825 -> 1817 ->
1815 -> 1634 -> 1814 -> 1813 -> 1632 -> 1631 -> 1742 -> 1741 -> 1740 -> 1739 -> 1591 -> 1689 -> 1585 -> 1584
-> 1688 -> 1579 -> 1679 -> 1677 -> 104 -> 5680 -> 5418 -> 5431 -> 5425 -> 5429 -> 5426 -> 5428 -> 5434 ->
5435 -> 5433 -> 5436 -> 5398 -> 5404 -> 5402 -> 5396 -> 5395 -> 5292 -> 5282 -> 5283 -> 5284 -> 5280 -> 50

Shortest Distance:  150784.607221936

Total Energy:  287931
Loop Count (for comparison):  24748
Completed task.
```

Task 3

Approach

The A* Search is similar to Dijkstra's algorithm with an additional consideration for the heuristic cost(H Cost), which is the estimated cost to reach the goal. We do so by calculating the Euclidean distance from each node to the end node from the coordinates of the nodes given in "Coord.json". Instead of looking for the node with lowest weight to visit next, we look for the node with the lowest combined weight, which is the weight plus the H Cost of that node. In this case, the code prioritizes nodes that help to move closer to the goal [2]. While A* Search will return the same path as Dijkstra's with or without the energy constraint, it will go through fewer loops to do so (25948 loops for Dijkstra's without energy constraint, 6360 for A*, 24748 loops for Dijkstra's with energy constraint, 8118 for A*)

Output

```
Input Start Point: 1
Input End Point: 50
Input Budget of Path: 287932
Loading task...

Shortest path:  1 -> 1363 -> 1358 -> 1357 -> 1356 -> 1276 -> 1273 -> 1277 -> 1269 -> 1267 -> 1268 -> 1284 ->
1283 -> 1282 -> 1255 -> 1253 -> 1260 -> 1259 -> 1249 -> 1246 -> 963 -> 964 -> 962 -> 1002 -> 952 -> 1000 ->
998 -> 994 -> 995 -> 996 -> 987 -> 988 -> 979 -> 980 -> 969 -> 977 -> 989 -> 990 -> 991 -> 2369 -> 2366 -> 2340
-> 2338 -> 2339 -> 2333 -> 2334 -> 2329 -> 2029 -> 2027 -> 2019 -> 2022 -> 2000 -> 1996 -> 1997 -> 1993 ->
1992 -> 1989 -> 1984 -> 2001 -> 1900 -> 1875 -> 1874 -> 1965 -> 1963 -> 1964 -> 1923 -> 1944 -> 1945 -> 1938
-> 1937 -> 1939 -> 1935 -> 1931 -> 1934 -> 1673 -> 1675 -> 1674 -> 1837 -> 1671 -> 1828 -> 1825 -> 1817 ->
1815 -> 1634 -> 1814 -> 1813 -> 1632 -> 1631 -> 1742 -> 1741 -> 1740 -> 1739 -> 1591 -> 1689 -> 1585 -> 1584
-> 1688 -> 1579 -> 1679 -> 1677 -> 104 -> 5680 -> 5418 -> 5431 -> 5425 -> 5429 -> 5426 -> 5428 -> 5434 ->
5435 -> 5433 -> 5436 -> 5398 -> 5404 -> 5402 -> 5396 -> 5395 -> 5292 -> 5282 -> 5283 -> 5284 -> 5280 -> 50

Shortest Energy:  287931
Total Distance:  150784.607221936

Loop Count (for comparison) :  8118
Completed task.
```

Contribution

Members	Contribution
Anil Ankitha	Project Report, Task 1
Song Yiming	Project Report, Task 3
Tony Ho Man Tung	Task 2, Task 3

Conclusion

To conclude, Dijkstra's implementation for Task 1 and 2 considering the distance weight is optimal compared to other uniformed shortest path algorithms.
Furthermore, the A* algorithm will always produce an optimal and complete solution in terms of limitations.

Reference

[1] Implementation of Dijsktra's algorithm, 11th Jan 2017, Retrieved from -
https://benalexkeen.com/implementing-djikstras-shortest-path-algorithm-with-python/
[2] A* (A Star) Search Algorithm – Computerphile, 16 Feb 2017, Retrieved from-
https://www.youtube.com/watch?v=ySN5Wnu88nE&ab_channel=Computerphile