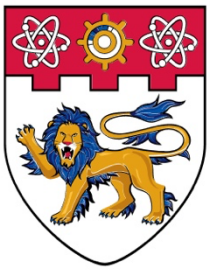

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3005: Lab 1, Finding a shortest path with An Energy Budget.

Lab Report

DSAI 1

Name: Karanam Akshit (U2020311E)

1. Program Architecture

The Programming Language used for this assignment is Java. Some object-oriented principles such as Classes, Polymorphism and Encapsulation has been used. These are the classes that were defined:

- Node:
  - Contains it coordinates, number and a list of neighbours
- Edge:
  - Contains the 2 connecting Node objects, the distance and the cost
- EdgeKey:
  - A Helper object that contains the Integer values of both the Node objects in an Edge (used in the Graph object, for easy retrieval of Edge object)
- Graph:
  - contains a HashMap<NodeNumber, Node>
  - contains a HashMap<EdgeKey, Edge>
- Path:
  - Contains a List of Node objects in the path
  - The current distance, energy and cost (for heuristic function)

There are a total of 3 tasks performed:

- Task 1: ucsUnrestricted search
- Task 2: ucsRestricted search
- Task 3: a* search

The details of the approaches and its output are elaborated in section 2 of this report.

Run the Main.java file to check for the outputs.

2. Tasks

2.1 Task 1

Approach:

I have applied the Uniform-Cost Search (UCS) algorithm to solve this task. Priority queue with the help of binary heaps was used to achieve a better efficiency.

We want to find the shortest path in terms of distance, which is an optimal problem. In the lecture, we have learnt various approaches to solving this searching task, such as Depth First

Search (DFS), Breadth First Search (BFS) and Iterative Deepening Search. DFS is not ideal in this case as it cannot guarantee convergence and optimality.

BFS and Iterative Deepening Search are usually used for the smallest number of edges from the source to destination. However, since the graph in this Assignment is a weighted graph, it does not seem to be the best choice to use these search algorithms. Hence, it was decided that UCS was the best approach for this assignment.

Steps in the algorithm:

1. *Create a new priority queue on Path objects (Comparator set to **distance**, i.e. the path with the least distance would be first dequeued)*
2. *Initialise a visited list. Add the source node into a new Path object, and it the Path object to the priority queue*

While the priority queue is not empty:

a. *Dequeue from priority queue*
b. *If reached goal node -> return this path*
c. *For each neighbour v*
   i. *if v is not visited*
   ii. *add this node to visited*
   iii. *add node to the current Path.*
   iv. *Add the new Path to the priority queue*

By running this algorithm, the calculated shortest path is 148648.63722140007 and the total energy cost is 294853.0

This is the output from the code:

Task 1:
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: 148648.63722140007
Total energy cost: 294853.0

2.1 Task 2

Approach:

For this task, a subvariant of the UCS is used, named “ucsRestricted” in the code. In the priority queue we still keep the path object and still dequeue using the “distance” of the path.

However, there is a small change in the visited list (HashMap<Node,List<Path>>). Instead of restricting the node to be visited only once, we change it such that each time a particular node (v) is visited, it is only added to the Path iff the new distance is less than that of all existing paths that are stored in ‘visited’. The rationale behind this step is such that if a new path taken to reach a particular node is larger than previously explored path, it means that the new path is not the shortest distance, and hence we ignore it.

Also, it is also ensured that the energy does not exceed the energy budget.

Steps in the algorithm:

1. *Create a new priority queue on Path objects (Comparator set to **distance**, i.e. the path with the least distance would be first dequeued)*
2. *Initialise a visited list (as described above). Add the source node into a new Path object, and it the Path object to the priority queue*

While the priority queue is not empty:

   a. *Dequeue from priority queue*
   b. *If reached goal node -> return this path*
   c. *For each neighbour v*
      i. *Check if new energy cost <= Energy_Budget*
      ii. *Check if the new Path v is shorter than any other previously explored path*
      iii. *if(ii) then (iv) else ignore*
      iv. *add this node to visited*
      v. *add node to the current Path.*
      vi. *Add the new Path to the priority queue*

This is the output from the code:

Task 2:
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 ->
2465 -> 2466 -> 2384 -> 2382 -> 2385 -> 2379 -> 2380 -> 2445 -> 2444 -> 2405 -> 2406 -> 2398 -> 2395 -> 2397 -> 2142 -> 2141 -> 2125 -> 2126 -> 2082 -> 2080 ->

2071 -> 1979 -> 1975 -> 1967 -> 1966 -> 1974 -> 1973 -> 1971 -> 1970 -> 1948 -> 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: 150335.55441905273
Total energy cost: 259087.0

2.2 Task 3

For the last task, an a* search algorithm is implemented. The coordinates of the nodes were used to calculate the 'Euclidean Distance' between 2 nodes. This is the heuristic function used for the a* search algorithm. A similar strategy is used as Task 2, but instead of using 'distance' as a notion for comparison, the 'score' is used. The score is derived from the heuristic function.

For example: (Reach from A→ D)
Path: A→ B (Distance = 10, Score = 10 + heuristic function(B,D))
When C is reached: Path A→B→C (Distance = 20, Score = 20 + heuristic function(C,D))

Steps in the algorithm:

1. *Create a new priority queue on Path objects (Comparator set to **score**, i.e. the path with the least score would be first dequeued)*
2. *Initialise a visited list (as described in task 2). Add the source node into a new Path object, and it the Path object to the priority queue*

While the priority queue is not empty:

a. *Dequeue from priority queue*
b. *If reached goal node -> return this path*
c. *For each neighbour v*
   i. *Check if new energy cost <= Energy_Budget*
   ii. *Check if the new Path v is shorter than any other previously explored path*
   iii. *if(ii) then (iv) else ignore*
   iv. *add this node to visited*
   v. *add node to the current Path.*
   vi. *Add the new Path to the priority queue*

This is the output from the code:

Task 3:
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 ->
2465 -> 2466 -> 2384 -> 2382 -> 2385 -> 2379 -> 2380 -> 2445 -> 2444 -> 2405 -> 2406 -> 2398 -> 2395 -> 2397 -> 2142 -> 2141 -> 2125 -> 2126 -> 2082 -> 2080 ->
2071 -> 1979 -> 1975 -> 1967 -> 1966 -> 1974 -> 1973 -> 1971 -> 1970 -> 1948 -> 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: 150335.55441905273
Total energy cost: 259087.0

3. Conclusion

This assignment and project have been helpful in making me have a better and deeper understanding of the various search methods introduced in lecture. It was also interesting to see how to design the program architecture to suit the needs of the problem. I also learned on how to modify exisiting algorithms (ucs in this case), to suit new needs and problems.