CS5011 A1 Report

Introduction

i.This assignment is CS5011:A1-Search-Coastguard Recue Simulations. My student ID is 190004531, and the date of submission is Feb 17th.

ii.For the implemented three parts:
Basic Agent: Completed all functions and passed 27 tests.
Intermediate Agent: Completed all functions and passed 27 tests.
Advanced Agent: On the basis of BestF(Best First Search) algorithm, adding DFS(Deep First Search) Tie-breaking strategy, which reduces the number of exploration nodes. Other functions not attempted.

iii.Compiling and running instructions with examples:

```
//How to Compile
javac A1main.java
//Run BFS and Example
java A1main <BFS> <ConfID>
java A1main BFS CONF22
//Run DFS and Example
java A1main <DFS> <ConfID>
java A1main DFS CONF22
//Run BestF and Example
java A1main <BestF> <ConfID> <null/Alphabet/AlphabetDeepFirst>
java A1main BestF CONF22
java A1main BestF CONF22 Alphabet
java A1main BestF CONF22 AlphabetDeepFirst
//Run A* and Example
java A1main <AStar> <ConfID>
java A1main AStar CONF22
```

Also, we can use Junit4 Test Class 'A1mainTest' which is in the test package.

Design, Implementation and Evaluation

i. Design

This route search system can use 6 different algorithms to explore routes according to start node and goal node.In order to find an excellent path, defining the problem as: On a two-dimensional map, plan a path between two points for the robot, as well as avoiding static obstacles on the map.

The PEAS(Performance, Environment, Actuators, Sensors) in this problem is:
Performance measure : The cost of the robot to get the target location, and the number of node the robot has explored.
Environment : A map with known location information.

Actuators : Robot moves on the map
Sensors : No

ii. Implementation

Based on the implementation, all the following algorithms are based on the general search algorithm, which differs only in the implementation of expand and Frontier.poll() strategies.

BFS(Breadth-first Search) algorithm: Before entering the node of the next depth level, first explore all neighbor nodes of the current depth. Therefore, the first-in-first-out principle of Queue is used when expanding nodes to complete breadth-first traversal.

DFS(Deep-first Search) algorithm: The algorithm starts from the root node and explores along the current branch as much as possible before backtracking. Therefore, Stack is adopted, and Stack's last-in-first-out principle is adopted when expanding nodes to complete depth-first traversal.

BestF(Best First Search), BestFA(Best First Search with Alphabetical order) and BestFDF(Best First Search with Deep First) algorithms: These three algorithms select the node with the smallest cost (the Manhattan distance from the node to the goal) in Frontier to explore. Therefore, the Frontier of algorithms is implemented by a priority queue. The priority queue sorts the nodes, and selects the node with the smallest cost as the next node to be explored.
Firstly, considering that there is a situation in Frontier, there are multiple nodes with the same minimum cost. Then, we define a set A={a,b,c,...}, which contains the nodes with the same smallest cost in the current Frontier. Therefore, we need to decide which node in set A to choose. So this system uses three different Tie-breaking strategies to choose better nodes.

BestF uses a random selection for a node in set A to explore.
BestFA uses alphabetical order to select nodes in set A for exploration (SE>E>NE>NW>W>SW).
However, there is a situation in set A that certain two nodes have the same direction and cost. Therefore, the BestFDF algorithm selects deeper nodes for exploration based on the idea of the DFS algorithm.

A algorithm:* The algorithm is similar to BestF in implementation, which both use the priority queue data structure. But the cost of sorting consists of two parts, which are g(n) and h(n). n is the next node on the path, g(n) is the cost of the path from the starting node to n, and h(n) is a heuristic function used to estimate the cost of the cheapest path from n to the goal. The difference is that when updating the frontier, the A* algorithm updates the cost value of the visited node. This also provides the possibility to find the optimal path from the starting point to the current point, and it also provides the possibility to find the global optimal solution.

iii. Evaluation

For these 6 algorithms, I ran the 27 tests provided and got the following table(Figure 1) and some interesting findings.

For an example, BFS(C,E) means the cost and explored nodes of BFS algorithm. First of all, we can see that BFS algorithm always can find the optimal solution, so the sum of BFS cost is 191. But, the number of nodes explored by BFS is almost twice that of DFS. For the BestF algorithms, their total number of exploration nodes is less than DFS, as well as having a smaller cost. However, BestF algorithms cannot always find the optimal solution.

	BFS(C,E)	DFS(C,E)	BestF(C,E)	BestFA(C,E)	BestFADF(C,E)	AStar(C,E)	
TCONF00	5,22	5,6	5,6	5,6	5,6	5,10	
TCONF01	3,10	3,4	3,4	3,4	3,4	3,4	
CONF0	5,26	5,6	5,6	5,6	5,6	5,6	
CONF1	5,31	9,11	5,6	5,6	5,6	5,6	
CONF2	10,36	14,25	10,11	10,11	10,11	10,20	
CONF3	6,32	7,29	6,7	6,7	6,7	6,14	
CONF4	4,27	7,10	4,5	4,5	4,5	4,9	
CONF5	6,23	6,7	6,7	6,7	6,7	6,14	
CONF6	6,23	7,9	6,7	6,7	6,7	6,13	
CONF7	4,18	6,7	4,5	4,5	4,5	4,5	
CONF8	5,20	6,7	5,7	6,7	6,7	5,10	Flag
CONF9	4,19	8,19	5,6	5,6	5,6	4,9	
CONF10	fail	fail	fail	fail	fail	fail	
CONF11	fail	fail	fail	fail	fail	fail	
CONF12	5,13	5,6	5,6	5,6	5,6	5,6	
CONF13	8,27	8,13	8,9	8,9	8,9	8,13	
CONF14	7,23	8,10	8,9	8,9	8,9	7,16	
CONF15	12,27	13,15	13,19	13,18	13,17	12,26	
CONF16	14,29	18,21	14,15	14,15	14,15	14,18	
CONF17	10,28	10,23	10,25	10,21	10,22	10,27	
CONF18	14,29	17,19	17,24	17,25	17,22	14,28	
CONF19	9,23	9,20	9,14	9,14	9,14	9,20	
CONF20	7,34	10,12	10,12	10,12	10,12	7,16	
CONF21	16,40	17,19	17,24	17,23	17,22	17,40	Flag
CONF22	7,33	18,21	9,14	9,15	9,12	7,20	
CONF23	9,35	10,12	10,11	10,11	10,11	9,23	
CONF24	10,36	21,29	12,23	12,25	12,21	10,28	
SUM	191,664	247,360	206,282	207,280	207,269	192,401	

Figure 1 : Test result of all algorithms

Firstly, from this table, we can find that the AStar algorithm can always find the optimal solution, except for CONF21. To be more specific, when I use this Expand() function below(Figure 2), which is the PPT of L5_w3_Search-pp.41, in Test CONF21. I got the result of BFS is (Cost = 16, explored = 40). The result of AStar is (Cost = 17, explored = 40). I have checked that both algorithms explored same nodes so that the cost of AStar should have been 16, not 17. Because when these two algorithms visit the same number of nodes, they should obtain the same information entropy.

About Test CONF21, there are 40 movable nodes in the map. The cost will be the optimal if all the nodes are explored (like the BFS).
Then I found there is something wrong with the AStar in lecture(Figure 2).

A* Search Algorithm

function EXPAND(*node, problem, frontier, explored, goal*) **returns** a set of nodes

next_states ← SUCCESSOR-FN(STATE[*node*],*problem*)

successors ← empty set

  for each *state* **in** *next_states*

  nd ← MAKE-NODE(*node*,*state*,*goal*)

  if state of *nd* is not contained in a node of *explored* or *frontier* then

    add *nd* to *successors*

  else if *state* is in a node in *frontier* but with higher PATH-COST then***

    replace old node with *nd*

  end for

return *successors*

end

*** If we find a shorter path

Figure 2 : AStar algorithm pseudo code

For this red sentence, we need to add one more sentence.

else if (state is in a node in *frontier* but with higher PATH-COST then replace old node with nd)

else if (state is in a node in *explored*, not in *frontier*, but with higher PATH-COST then **re-add** this node in frontier which use the lower cost. And delete this node from *explored*)

Because we need to consider the nodes in explored. If we do not consider the explored, we may poll a certain key node of the optimal path in advance, so that the end route will not be contain polled nodes. On the other hand, we should not give up the judgment of nodes in explored of AStar Algorithm.

After change the AStar algorithm. The result of AStar algorithm in Test CONF21 is (C=16,E=39)(Figure 3).

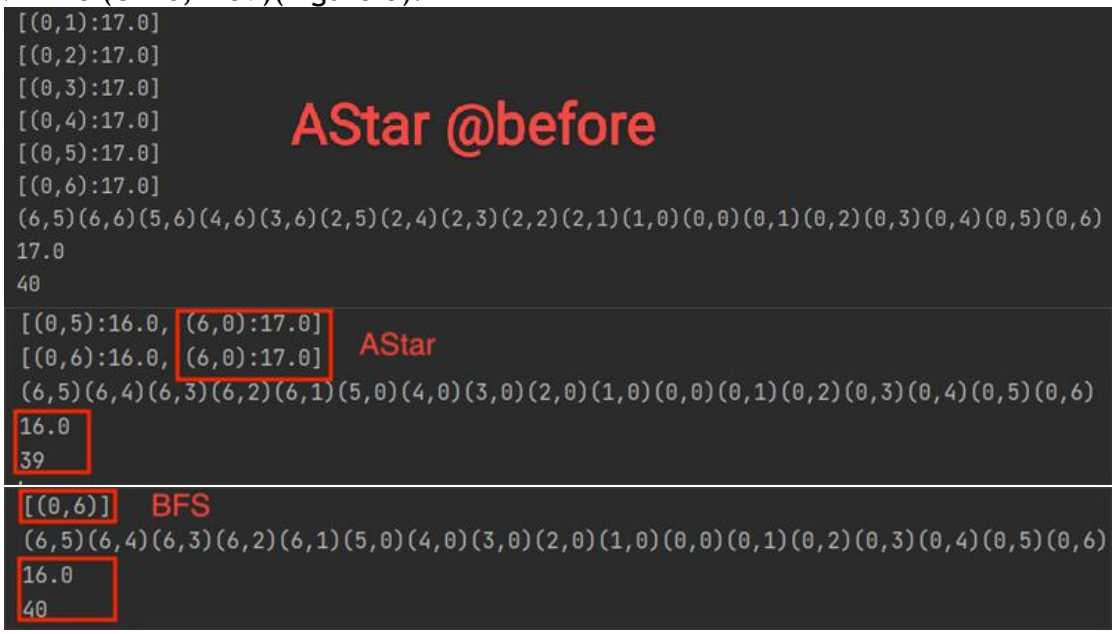

Figure 3 : Using AStar and BFS in CONF21

Then, let us observe the three BestF algorithms in the table. The effect of merely transforming Tie-breaking strategies in algorithms is not obvious. So, the results of Tie-breaking strategy are very random and not easy to evaluate. However, after adding a Deep First strategy to BestF, the number of nodes explored has dropped. The previous test example is not obvious in the results because the map is not complex enough. In the last 10 Tests (CONF15-CONF24), the number of exploration nodes has dropped significantly.

In conclusion, we can find that AStar algorithm has excellent performance in finding the optimal solution. Meanwhile, the DFS algorithm can significantly reduce the number of nodes explored. In the future, we can combine these two algorithms to study algorithms with better performance.

Test Summary

Because there used to be some problems in testing CONF21. I will show the result of six algorithms of Test CONF21(Figure 4).

Test	Algorithm	Cost	Explored
CONF21	BFS	16	40
	DFS	17	19
	BestFR	17	24
	BestFA	17	23
	BestFADF	17	22
	AStar	16	39

Figure 4: CONF21 Test Table

The outputs of CONF21 are:
BFS:

```
[(0,4)]
[(0,5)]
[(0,6)]
(6,5)(6,4)(6,3)(6,2)(6,1)(5,0)(4,0)(3,0)(2,0)(1,0)(0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)
16.0
40
```

DFS:

```
[(6,4), (5,5), (3,3), (3,2), (3,1), (2,0), (0,5)]
[(6,4), (5,5), (3,3), (3,2), (3,1), (2,0), (0,6)]
(6,5)(6,6)(5,6)(4,6)(3,6)(2,5)(2,4)(2,3)(2,2)(2,1)(1,0)(0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)
17.0
19
```

BestF:

```
[(0,5):1.0, (4,3):7.0, (2,0):8.0, (3,1):8.0, (6,4):8.0, (5,3):8.0, (4,2):8.0]
[(0,6):0.0, (4,3):7.0, (2,0):8.0, (3,1):8.0, (6,4):8.0, (5,3):8.0, (4,2):8.0]
(6,5)(6,6)(5,6)(4,6)(3,6)(2,5)(2,4)(2,3)(2,2)(2,1)(1,0)(0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)
17.0
24
```

BestFA:

```
[(0,5):1.0, (3,2):7.0, (4,3):7.0, (5,3):8.0, (6,4):8.0, (2,0):8.0, (3,1):8.0]
[(0,6):0.0, (4,3):7.0, (3,2):7.0, (5,3):8.0, (6,4):8.0, (2,0):8.0, (3,1):8.0]
(6,5)(6,6)(5,6)(4,6)(3,6)(2,5)(2,4)(2,3)(2,2)(2,1)(1,0)(0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)
17.0
23
```

BestFAD:

```
[(0,5):1.0, (4,3):7.0, (3,2):7.0, (5,4):7.0, (2,0):8.0, (3,1):8.0, (6,4):8.0]
[(0,6):0.0, (3,2):7.0, (4,3):7.0, (5,4):7.0, (2,0):8.0, (3,1):8.0, (6,4):8.0]
(6,5)(6,6)(5,6)(4,6)(3,6)(2,5)(2,4)(2,3)(2,2)(2,1)(1,0)(0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)
17.0
22
```

AStar:

```
[(0,5):16.0, (6,0):17.0]
[(0,6):16.0, (6,0):17.0]
(6,5)(6,4)(6,3)(6,2)(6,1)(5,0)(4,0)(3,0)(2,0)(1,0)(0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)
16.0
39
```

Also,all the other output are correct.

Bibliography

docs.oracle.com. (n.d.). *Java API Reference*. [online] Available at: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/PriorityQueue.html.

docs.oracle.com. (n.d.). *Java API Reference*. [online] Available at: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Queue.html [Accessed 17 Feb. 2021].

junit.org. (n.d.). *JUnit API*. [online] Available at: https://junit.org/junit4/javadoc/latest/ [Accessed 17 Feb. 2021].

Skeet, J. (2011). *How to avoid java.util.ConcurrentModificationException when iterating through and removing elements from an ArrayList*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/8104692/how-to-avoid-java-util-concurrentmodificationexception-when-iterating-through-an [Accessed 17 Feb. 2021].