CS5011 A1 Report

matriculation number: 160021429

1. Introduction

The main aim of the assignment 1 is to implement the Java program that provides several search algorithms that searches the path for the aircraft. According to the specification, implementing the BFS and DFS is a bagic agent, and implementing the Best-First Search and A* Search is an intermediate agent. I implemented all of them. Furthermore, I have done 2 extensions: 1) Implementing the Bidirectional Search, 2) Improve the DFS to check the Weather Forecast Coordinates that are obstacle points.

1 - 1. Check Table

Things that I implemented:

Part	**Functionality**
Basic Agent	BFS
Basic Agent	DFS
Intermediate Agent	Best-First Search
Intermediate Agent	A* Search
Extension 1	Bidirectional Search
Extension 2	Weather Forecast Obstacles (add to DFS)

1 - 2. How to run it

Basically, the program requires at least 4 command-line arguments. The first argument is the name of the search algorithm. The second argument should be a positive integer N, which is the number of parallels of Oedipus. The third and forth arguments should be the coordinate of the starting point and goal.

To compile the source code, you just need to type “javac *.java”.

Instructions to run the program:

- BFS : java A1main BFS N <d_s, angle_s> <d_g, angle_g>
- DFS : java A1main DFS N <d_s, angle_s> <d_g, angle_g>
- DFS with weather coordinates :
  java A1main DFS N <d_s, angle_s> <d_g, angle_g> [obstacle_coordinates]

- Best-First : java A1main BestF N <d_s, angle_s> <d_g, angle_g>
- A* Search : java A1main AStar N <d_s, angle_s> <d_g, angle_g>
- Bidirectional Search : java A1main BiDir N <d_s, angle_s> <d_g, angle_g>

i.e.

1) java A1main BFS 6 1,0 5,90 -> BFS
2) java A1main DFS 6 1,0 3,135 1,90 1,135 -> DFS with weather obstacles

2. Design, Implementation and Evaluation

2 - 1. Design and Implementation

Basically, all classes of search algorithms are child classes of the class called “Search”. I implemented some essential methods such as checkIfVisited(), which checks if the search algorithm visited a given coordinate before. By doing this, I could reuse some essential methods that are used by more than 2 search algorithms.

Also, each class for search algorithm has a method called search(), which performs the corresponding search. For instance, if you generate a BFS type instance and call the search method, then it will perform the BFS to find the path from starting point to the goal.

Furthermore, I made the instances for starting coordinate and goal coordinate as public static attributes. By using the Singleton pattern, all search algorithms that I implemented could easily access to the coordinate whenever they needed.

Moreover, as mentioned in the specification, all of the search algorithms that I implemented print out the essential information for each step, so that the user could check if the search algorithms work properly.

2 - 1- 1. BFS

The BFS.java contains the class BFS, which performs the breadth first search. If you see the method search, you could find a while loop that iterates the queue. Basically, what I did is dequeue a node from the queue, and check if the current node is a goal. If not, expand the child nodes, and enqueue those child nodes to the queue.

When it expands the child nodes, it will validate the child nodes' coordinates, and abandon the nodes with invalid coordinates. By doing this, the search() method will run the while loop until it finds the goal. When it finds the goal, it will then print out the result path.

Moreover, if the BFS fails to find the goal for some reason, for example, the goal is (0,0) which is a point that the aircraft cannot reach, the search() method will print out the suitable error message.

My BFS can find the path with the smallest number of moves. Thus, it is possible to say that the BFS that I implemented is complete and optimal.

2 - 1 - 2. DFS

To implement the DFS, I implemented the search method to call itself recursively. So, the search() method in the class DFS will check the given node. If the current node is not the goal, it will expand the child nodes, and use the for loop to iterate the list of child nodes. In that for loop, the search() method will call itself recursively.

The DFS that I implemented will call itself recursively until it finds the goal. Also, it checks the visited node to avoid the endless loop. To implement this, I used boolean 2D-array. So, if the search algorithm will check and update the boolean value in that 2D array, which will help the search algorithm to avoid endless loop.

Moreover, the DFS also prints out the suitable error message if the search algorithm failed to find the goal. This might happen if and only if the goal is a coordinate that the aircraft cannot reach.

My DFS can always find the path if the coordinates of both starting point and goal are valid and reachable, but it cannot guarantee that the found path is the shortest path. Thus, the DFS that I implemented is complete, but not optimal.

2 - 1 - 3. Best First Search

The implementation of my Best First Search is almost same with the DFS. The only difference between the DFS and Best First Search is that the Best First Search uses the heuristic method when it expands the node. So, when the search() method of the Best First Search expands the child nodes, it will use the heuristic method that calculates the remaining distance from target node to goal, and compare the future costs of the expanded nodes. Then, it will use the node with the smallest future cost as a next node, and call the search() method recursively.

When it calculates the future cost, it will calculate the remaining distance by calculating the Euclidean Distance ($\sqrt{d1^2 + d2^2 + 2 * d1 * d2 * cos(a_{diff})}$, where d1 = the distance of current node, d2 = the distance of goal, and a_{diff} = difference between angle of current node and goal). As the Best First Search uses the heuristic method, it could find the shortest path. Furthermore, as I mentioned above, the structure of the search() method of the Best First Search is similar to the search() method of the DFS, which means that it could always find the goal except some unexpected situations (i.e. goal is (0,0), which is the point that the aircraft cannot reach). Thus, it is possible to say that the Best First Search is optimal and complete for this problem.

2 - 1 - 4. A* Search

The structure of the search() method of my A* Search is similar to the BFS. Basically, my A* Search also uses the queue. It also enqueues the expanded nodes to the queue, and iterating the queue by using the while loop. However, the A* Search uses the heuristic method to calculate the future cost and past cost. In my implementation, the past cost is the remaining distance from parent node to goal, and the future cost is the remaining distance from child node to the goal. So, it compares the future cost of the parent node and child node, and only appends the nodes, whose future cost is smaller than the parent node's future cost, to the queue.

By using this heuristic method (euclidean distance), the A* search could abandon the inefficient paths, and could easily find the shortest path. Thus, the A* Search that I implemented is complete and optimal.

2 - 1 - 5. Bidirectional Search

The Bidirectional Search also uses the queue to store the expanded nodes. However, the bidirectional search should expand nodes from starting point and goal simultaneously, thus, I used 2 queues to implement it.

My bidirectional search uses 2 boolean 2D arrays, so that the program could check the intermediate point. As we expand child nodes from 2 different directions, we could find the shortest path within shorter time than the BFS. However, the bidirectional search does not use the heuristic method, thus, it still takes more steps than both Best First Search and A* Search.

As the bidirectional search uses similar algorithm with BFS, my bidirectional search is optimal and complete.

2 - 1 - 6. Weather Forecast Obstacles

In the real world, we might face with some unexpected cases in which the weather forecasts show that some points cannot be crossed, where storms for example form obstacles at specific grid coordinate.

To implement this, I made a new method called generateForecastList(), which generates a list of coordinates that the aircraft cannot cross due to the weather issue. To use this, the user should add more command line arguments. For example, “java DFS 5 1,0 4,90 1,90 3,90” will generate a list of obstacles, where (1, 90) and (3, 90) are obstacles in this case. Then, the DFS will check the list of obstacles when it expands the nodes.

2 - 2. Evaluation

2 - 2 - 1. BFS

My BFS uses the queue to store the expanded nodes. As I mentioned above, it is complete and optimal. Thus, ideally, the BFS should be able to find the shortest path. In fact, however, the path that the BFS found is not actually optimal in this problem.

It is clear that the length of arc is generally greater than the length of line. Thus, if the aircraft moves either H90 or H270, it’s path would be longer than H180 and H360. As the BFS does not have heuristic method, the result of the BFS cannot be better than A* Search. Henceforth, it would be better to use A* Search than BFS in this problem.

2 - 2 - 2. DFS

As the DFS is not optimal, the path that it found is usually inefficient. Furthermore, as I used the recursive way to implement the DFS, StackOverflowError might occur if the size of the Oedipus is huge enough and the starting point is far away from the goal. Every single time it calls the search function recursively, the JVM will add data to stack, and eventually the Stack Overflow might occur. Henceforth, DFS would be the worst choice in this problem.

2 - 2 - 3. Best First Search

As the Best First Search uses the heuristic method, it will perform much better than both BFS and DFS. Also, it founds the path within the shortest time. Also, it uses the Euclidean Distance as a heuristic method, the Best First Search that I implemented finds the “actual optimal path”.

2 - 2 - 4. A* Search

As the A* Search also uses the heuristic method with Euclidean Distance, the A* Search could find the “actual optimal path”. As the A* Search expands the nodes and store the expanded nodes in the queue, usually takes more execution time than Best First Search. However, the A* Search uses less memory than Best First Search, since the Best First Search add data to stack everytime it calls the search() method recursively, thus still A* Search is as good as Best First Search for this problem.

2 - 2 - 5. Bidirectional Search

The bidirectional search takes less steps than BFS, since it generally expands less nodes than BFS. Thus it is generally better than BFS. However, the bidirectional search also not uses the heuristic method, it is possible to say the Best First and A* Search would be better choices.

3. Testing

3 - 1. DFS

java A1main DFS 5 2,90 4,225

```
Current coordinate: 4, 180
Number of moves : 20
Frontier : (2,90) (2,135) (2,180) (2,225) (2,270) (2,315) (2,0) (2,45) (3,45) (3,90) (3,135) (3,180) (3,225) (3,270) (3,315) (3,0) (4,0)
(4,45) (4,90) (4,135) (4,180)
Expanded nodes:
        Node1 : Coordinate = 4, 225 & Path = H90
        Node2 : Coordinate = 4, 135 & Path = H270
        Node3 : Coordinate = 3, 180 & Path = H360

Current coordinate: 4, 225
Found the goal!
Number of moves : 21
Frontier : (2,90) (2,135) (2,180) (2,225) (2,270) (2,315) (2,0) (2,45) (3,45) (3,90) (3,135) (3,180) (3,225) (3,270) (3,315) (3,0) (4,0)
(4,45) (4,90) (4,135) (4,180) (4,225)

Result Path : H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90
Total moves = 21
```

java A1main DFS 180 108,90 135,225

```
Result Path : H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180,
H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90,
 H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90
, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H
90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90,
H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90,
 H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H9
0, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90, H90, H180, H90, H90, H90, H
90, H90, H90, H90, H180, H90, H90, H90, H90, H90, H90
Total moves = 222
```

java A1main DFS 15 0,0 2,90

```
Starts DFS

The aircraft cannot reach or fly over the pole, such as (0,0)
DFS failed to find the path!
```

java A1main DFS 5 4,0 0,0

```
Current coordinate: 3, 45
Already visited this coordinate...
Number of moves : 1
Frontier : (4,0) (4,45)

Current coordinate: 4, 315
Already visited this coordinate...
Number of moves : 0
Frontier : (4,0)

Current coordinate: 3, 0
Already visited this coordinate...
Number of moves : 0
Frontier : (4,0)

DFS failed to find the path!
```

As you could see above, my DFS works properly, and could handle errors if the starting point or goal is (0,0).

3 - 2. BFS

java A1main BFS 5 4,0 0,0

```
Current coordinate: 1, 225
Queue : (3, 225), (1, 315), (1, 225), (2, 270), (2, 225), (2, 135), (1, 180), (3, 180), (1, 180), (1, 90), (2, 135)
Expanded nodes:
       Node1 : Coordinate = 1, 270 & Path = H90
       Node2 : Coordinate = 1, 180 & Path = H270
       Node3 : Coordinate = 2, 225 & Path = H180

Current coordinate: 1, 180
Queue : (3, 180), (1, 180), (1, 90), (2, 135), (1, 270), (1, 180), (2, 225)
Expanded nodes:
       Node1 : Coordinate = 1, 225 & Path = H90
       Node2 : Coordinate = 1, 135 & Path = H270
       Node3 : Coordinate = 2, 180 & Path = H180

BFS failed to reach the goal
Please check if the coordinate of the goal is valid
```

java A1main BFS 5 0,0 4,0

```
Starts BFS

The aircraft cannot reach or fly over the pole, such as (0,0)
BFS failed to reach the goal
Please check if the coordinate of the goal is valid
```

java A1main BFS 5 3,0 4,135

```
Current coordinate: 2, 135
Queue : (4, 135), (2, 135), (2, 45), (1, 90), (3, 90), (4, 135), (4, 45), (3, 90), (1, 90), (1, 0), (2, 45), (3, 270), (3, 180), (2, 225)
, (4, 225), (2, 315), (2, 225), (1, 270), (3, 270), (4, 315), (4, 225), (3, 270), (1, 0), (1, 270), (2, 315), (3, 225), (3, 135), (2, 180
), (4, 180)
Expanded nodes:
        Node1 : Coordinate = 2, 180 & Path = H90
        Node2 : Coordinate = 2, 90 & Path = H270
        Node3 : Coordinate = 1, 135 & Path = H360
        Node4 : Coordinate = 3, 135 & Path = H180

Current coordinate: 4, 135
Found the goal!

Result Path: H90 H90 H90 H180
The number of total moves = 22
```

java A1main BFS 180 108,90 135,225

```
        Node1 : Coordinate = 134, 315 & Path = H90
        Node2 : Coordinate = 134, 225 & Path = H270
        Node3 : Coordinate = 133, 270 & Path = H360
        Node4 : Coordinate = 135, 270 & Path = H180

Current coordinate: 81, 225
Queue : (83, 225), (134, 270), (134, 180), (133, 225), (135, 225), (81, 225), (81, 135), (80, 180), (82, 180), (135, 225), (135, 135), (1
34, 180), (136, 180), (80, 180), (80, 90), (79, 135), (81, 135), (136, 180), (136, 90), (135, 135), (137, 135), (82, 0), (82, 270), (81,
315), (83, 315), (134, 0), (134, 270), (133, 315), (135, 315), (81, 45), (81, 315), (80, 0), (82, 0), (135, 45), (135, 315), (134, 0), (1
36, 0), (80, 90), (80, 0), (79, 45), (81, 45), (136, 90), (136, 0), (135, 45), (137, 45), (79, 135), (79, 45), (78, 90), (80, 90), (137,
135), (137, 45), (136, 90), (138, 90), (82, 315), (82, 225), (81, 270), (83, 270), (134, 315), (134, 225), (133, 270), (135, 270)
Expanded nodes:
        Node1 : Coordinate = 81, 270 & Path = H90
        Node2 : Coordinate = 81, 180 & Path = H270
        Node3 : Coordinate = 80, 225 & Path = H360
        Node4 : Coordinate = 82, 225 & Path = H180

Current coordinate: 135, 225
Found the goal!

Result Path: H90 H90 H90 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H1
80 H180 H180 H180 H180
The number of total moves = 443
```

As you could see above, my BFS works properly, and handles error with suitable error messages.

3 - 3. A* Search

java A1main AStar 5 3,0 4,135

```
Current coordinate: 2, 90
Queue : (2, 90), (1, 45), (1, 315), (1, 45), (1, 315), (4, 135),
Expanded nodes:
        Node1 : Coordinate = 2, 135 & Path = H90

Current coordinate: 1, 45
Queue : (1, 315), (1, 45), (1, 315), (4, 135), (2, 135),
Expanded nodes:
        Node1 : Coordinate = 1, 90 & Path = H90

Current coordinate: 1, 315
Queue : (1, 45), (1, 315), (4, 135), (2, 135), (1, 90),
Expanded nodes:

Current coordinate: 4, 135
Found the goal!

Result Path: H90 H90 H90 H180
```

java A1main AStar 5 3,0 0,0

```
Current coordinate: 1, 0
Queue :
Expanded nodes:

A* Search failed to reach the goal
Please check if the coordinate of the goal is valid
```

java A1main AStar 5 0,0 3,0

```
Starts A* Search

The aircraft cannot reach or fly over the pole, such as (0,0)
A* Search failed to reach the goal
Please check if the coordinate of the goal is valid
```

java A1main AStar 180 108,90 135,225

```
Current coordinate: 79, 90
Queue : (135, 225), (83, 225), (81, 225), (80, 180), (80, 180), (79, 135), (80, 0), (80, 0), (79, 45),
Expanded nodes:
        Node1 : Coordinate = 79, 135 & Path = H90
        Node2 : Coordinate = 79, 45 & Path = H270
        Node3 : Coordinate = 78, 90 & Path = H360

Current coordinate: 135, 225
Found the goal!

Result Path: H90 H90 H90 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H180 H1
80 H180 H180 H180 H180
```

The A* Search also works perfectly, and can handle errors with suitable error messages.

3 - 4. Best First Search

java A1main BestF 180 108,90 135,225

```
Current coordinate: 134, 225
Number of moves : 29
Frontier : (108,90) (108,135) (108,180) (108,225) (109,225) (110,225) (111,225) (112,225) (113,225) (114,225) (115,225) (116,225) (117,22
5) (118,225) (119,225) (120,225) (121,225) (122,225) (123,225) (124,225) (125,225) (126,225) (127,225) (128,225) (129,225) (130,225) (131
,225) (132,225) (133,225) (134,225)
Expanded nodes:
        Node1 : Coordinate = 135, 225 & Path = H180
        Node2 : Coordinate = 133, 225 & Path = H360
        Node3 : Coordinate = 134, 270 & Path = H90
        Node4 : Coordinate = 134, 180 & Path = H270

Current coordinate: 135, 225
Found the goal!
Number of moves : 30
Frontier : (108,90) (108,135) (108,180) (108,225) (109,225) (110,225) (111,225) (112,225) (113,225) (114,225) (115,225) (116,225) (117,22
5) (118,225) (119,225) (120,225) (121,225) (122,225) (123,225) (124,225) (125,225) (126,225) (127,225) (128,225) (129,225) (130,225) (131
,225) (132,225) (133,225) (134,225) (135,225)

Result Path : H90, H90, H90, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180, H180,
H180, H180, H180, H180, H180, H180, H180, H180, H180
Total moves = 30
```

java A1main BestF 5 0,0 3,0

```
Starts Best-First Search

The aircraft cannot reach or fly over the pole, such as (0,0)
Best-First Search failed to find the path!
```

java A1main BestF 5 3,0 0,0

```
Frontier : (3,0) (2,0)

Current coordinate: 3, 0
Already visited this coordinate...
Number of moves : 1
Frontier : (3,0) (2,0)

Current coordinate: 3, 45
Already visited this coordinate...
Number of moves : 0
Frontier : (3,0)

Current coordinate: 3, 315
Already visited this coordinate...
Number of moves : 0
Frontier : (3,0)

Current coordinate: 4, 0
Already visited this coordinate...
Number of moves : 0
Frontier : (3,0)

Best-First Search failed to find the path!
```

java A1main BestF 5 3,0 4,90

```
Frontier : (3,0) (3,45)
Expanded nodes:
        Node1 : Coordinate = 3, 90 & Path = H90
        Node2 : Coordinate = 2, 45 & Path = H360
        Node3 : Coordinate = 4, 45 & Path = H180
        Node4 : Coordinate = 3, 0 & Path = H270

Current coordinate: 3, 90
Number of moves : 2
Frontier : (3,0) (3,45) (3,90)
Expanded nodes:
        Node1 : Coordinate = 4, 90 & Path = H180
        Node2 : Coordinate = 2, 90 & Path = H360
        Node3 : Coordinate = 3, 135 & Path = H90
        Node4 : Coordinate = 3, 45 & Path = H270

Current coordinate: 4, 90
Found the goal!
Number of moves : 3
Frontier : (3,0) (3,45) (3,90) (4,90)

Result Path : H90, H90, H180
Total moves = 3
```

The Best First Search also works fine. Also, as you could see, the total moves of the Best First Search is much less than other search algorithms. It is because that my Best First Search uses recursive way with heuristic methods. My best first search could find best path within shortest time. Thus, it would be possible to say that the Best First Search would be the best search algorithm for the given problem.

3 - 5. Bidirectional Search

java A1main BiDir 5 3,0 4,90

```
        Node2 : Coordinate = 3, 45 & Path = H270
        Node3 : Coordinate = 2, 90 & Path = H360
        Node4 : Coordinate = 4, 90 & Path = H180

Current coordinate = 4, 0
Queue1 : (3, 90), (3, 0), (2, 45), (4, 45), (3, 0), (3, 270), (2, 315), (4, 315), (2, 45), (2, 315), (1, 0), (3, 0)
Expanded nodes:
        Node1 : Coordinate = 4, 45 & Path = H90
        Node2 : Coordinate = 4, 315 & Path = H270
        Node3 : Coordinate = 3, 0 & Path = H360

Current coordinate = 4, 180
Queue2 : (4, 90), (3, 135), (4, 90), (4, 0), (3, 45), (3, 135), (3, 45), (2, 90), (4, 90)
Expanded nodes:
        Node1 : Coordinate = 4, 225 & Path = H90
        Node2 : Coordinate = 4, 135 & Path = H270
        Node3 : Coordinate = 3, 180 & Path = H360

Current coordinate = 3, 90
Queue1 : (3, 0), (2, 45), (4, 45), (3, 0), (3, 270), (2, 315), (4, 315), (2, 45), (2, 315), (1, 0), (3, 0), (4, 45), (4, 315), (3, 0)

Result path: H90 H90 H180
The total number of moves = 11
```

java A1main BiDir 6 3,90 4,225

```
Current coordinate = 5, 270
Queue2 : (4, 225), (4, 135), (3, 180), (5, 180), (3, 270), (3, 180), (2, 225), (4, 225), (5, 270), (5, 180), (4, 225), (4, 0), (4, 270),
(3, 315), (5, 315), (3, 315), (3, 225), (2, 270), (4, 270)
Expanded nodes:
        Node1 : Coordinate = 5, 315 & Path = H90
        Node2 : Coordinate = 5, 225 & Path = H270
        Node3 : Coordinate = 4, 270 & Path = H360

Current coordinate = 3, 0
Queue1 : (2, 45), (4, 45), (2, 135), (2, 45), (1, 90), (3, 90), (4, 135), (4, 45), (3, 90), (5, 90), (3, 225), (3, 135), (2, 180), (4, 18
0), (2, 180), (2, 90), (1, 135), (3, 135), (4, 180), (4, 90), (3, 135), (5, 135)
Expanded nodes:
        Node1 : Coordinate = 3, 45 & Path = H90
        Node2 : Coordinate = 3, 315 & Path = H270
        Node3 : Coordinate = 2, 0 & Path = H360
        Node4 : Coordinate = 4, 0 & Path = H180

Current coordinate = 4, 135
Queue2 : (3, 180), (5, 180), (3, 270), (3, 180), (2, 225), (4, 225), (5, 270), (5, 180), (4, 225), (4, 0), (4, 270), (3, 315), (5, 315),
(3, 315), (3, 225), (2, 270), (4, 270), (5, 315), (5, 225), (4, 270)

Result path: H90 H180 H90 H90
The total number of moves = 22
```

As you could see above, the bidirectional search works properly. It could find the optimal path by expanding the child nodes from 2 ways.

Queue1 contians the expanded nodes from starting point, and Queue2 contains the expanded nodes from goal. By enqueuing and dequeuing the nodes, the bidirectional search could easily find the best path.

As you could see below, the bidirectional search also handles the (0,0) coordinate issue.

java A1main BiDir 5 0,0 3,135

```
Starts Bi-directional Search

The aircraft cannot reach or fly over the pole, such as (0,0)
```

java A1main BiDir 5 3,0 0,135

```
Starts Bi-directional Search

Current coordinate = 3, 0
Queue1 :
Expanded nodes:
        Node1 : Coordinate = 3, 45 & Path = H90
        Node2 : Coordinate = 3, 315 & Path = H270
        Node3 : Coordinate = 2, 0 & Path = H360
        Node4 : Coordinate = 4, 0 & Path = H180

Please check the command line arguments
Aircraft cannot reach or fly over the pole such as (0,0)!
```

3 - 6. Weather Obstacles with DFS

java A1main DFS 5 2,90 4,225 3,90 2,135

```
Frontier : (2,90) (2,45) (2,0) (2,315) (2,270) (2,225) (2,180) (3,180) (3,225) (3,270) (3,315) (3,0) (3,45) (4,45) (4,90) (4,135)
Expanded nodes:
       Node1 : Coordinate = 4, 180 & Path = H90
       Node2 : Coordinate = 4, 90 & Path = H270
       Node3 : Coordinate = 3, 135 & Path = H360

Current coordinate: 4, 180
Number of moves : 16
Frontier : (2,90) (2,45) (2,0) (2,315) (2,270) (2,225) (2,180) (3,180) (3,225) (3,270) (3,315) (3,0) (3,45) (4,45) (4,90) (4,135) (4,180)

Expanded nodes:
       Node1 : Coordinate = 4, 225 & Path = H90
       Node2 : Coordinate = 4, 135 & Path = H270
       Node3 : Coordinate = 3, 180 & Path = H360

Current coordinate: 4, 225
Found the goal!
Number of moves : 17
Frontier : (2,90) (2,45) (2,0) (2,315) (2,270) (2,225) (2,180) (3,180) (3,225) (3,270) (3,315) (3,0) (3,45) (4,45) (4,90) (4,135) (4,180)
 (4,225)

Result Path : H270, H270, H270, H270, H270, H270, H180, H90, H90, H90, H90, H90, H180, H90, H90, H90, H90
Total moves = 17
```

As you could see above, you need to add more command line arguments to use the weather obstacle mode DFS. The example above runs the DFS which the starting point (2, 90) and goal (4, 225), and 2 obstacles (3, 90) and (2,135). So, when the DFS expands the child nodes, it will check the weather obstacles' coordinates. If one of the expanded nodes is an obstacle, then the program will remove that node from the search tree, so that the DFS could avoid the weather obstacle.

If the starting point is a weather obstacle, then my program will print out suitable error message.

java A1main DFS 5 2,90 4,225 2,90

```
Starts DFS with the forecast mode

Current coordinate: 2, 90
Aircraft cannot fly over this coordinate due to the weather issue!

DFS failed to find the path!
ys60@lyrane:~/Documents/cs5011/CS5011P1/A1src $
```

Similarly, if the goal is a weather obstacle, the DFS will fail to find the path, since the aircraft cannot reach the goal due to the weather issue. If so, the program will also print out the error message.

java A1main DFS 5 2,90 4,225 4,225

```
Current coordinate: 2, 45
Already visited this coordinate...
Number of moves : 0
Frontier : (2,90)

Current coordinate: 1, 90
Already visited this coordinate...
Number of moves : 0
Frontier : (2,90)

Current coordinate: 3, 90
Already visited this coordinate...
Number of moves : 0
Frontier : (2,90)

DFS failed to find the path!
```

As you could see above, the forecast mode DFS also works properly just like normal DFS.

4. Bibliography

Euclidean Distance : <https://studres.cs.st-andrews.ac.uk/CS5011/Lectures/L5_w3.pdf>