Assignment 1

Saurav Jha

09 Oct, 2019

1 Program Overview

The submission intends to implement the following algorithms for the given problem:

1. Breadth First Search (BFS),
2. Depth First Search (DFS),
3. Best First Search (BestF),
4. A* Search (AStar), and
5. Bidirectional Search (Bidirec).

The program can be run as:

java A1main <DFS|BFS|AStar|BestF|Bidirec> <N> <d_s, angle_s> <d_g, angle_g> [BFS|BestF]*

where * denotes an optional argument. The first four arguments are same as mentioned in the instructions. The fifth argument applies only for Bidirectional search and implies whether to use the BFS or BestF search technique.

1.1 States and transitions

Each discrete state[1] is represented by the pair *<d, angle>* where d refers to the n^{th} circle while *angle* is the elevation on this circle with following properties:

1. An instance of a problem with N worlds has the set of possible worlds as $[0, 1, 2, ..., N-1]$ while the set of valid worlds as $[1, 2, ..., N-1]$ considering prohibited flights over the pole or beyond the outermost world.
2. Each circle has eight valid angles with an exception of the 0^{th} world: $[0, 45, 90, ..., 315]$.

The set of possible next states is governed by algorithm 1:

[1]The terms "state" and "node" are used interchangeably hereafter.

Algorithm 1 State transition

```
1: procedure GET NEXT VALID STATES
2:     currentState ← <d, angle>
3:     N ← no. of worlds
4:     possibleNextStates ← list of possible transitions
5:     nextGreaterAngle = (angle + 45) mod 360
6:     nextSmallerAngle = (angle − 45) if angle > 0 else 315
7:     possibleNextStates.append(<d, nextGreaterAngle>)
8:     if d > 1 then
9:         possibleNextStates.append(<d-1, angle>)
10:    if d < N then
11:        possibleNextStates.append(<d+1, angle>)
```

1.2 Identifying directions

The four possible directions are: 90, 180, 270, 360. Within the same world, an upward transition (i.e., angle + 90) corresponds to "H90" while moving down corresponds to "H270". For inter-world transitions, moving towards the center (i.e., d - 1) is denoted by "H360" while moving away from the center is denoted by "H180" (See figure 1).

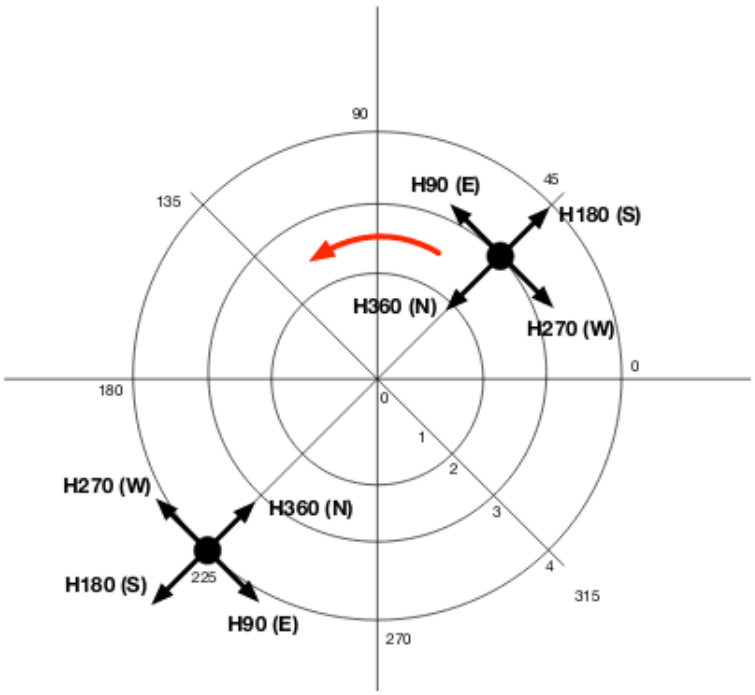

Figure 1: Figure showing the directions corresponding to moves.

1.3 Representation of State

The class "State" contains attributes to build the graph of Oedipus. The private attributes: "circle" refers to the n^{th} world, "meridian" refers to the angle while "noOfCircles" contains the value for N. All three parameters are set using the constructor. A public attribute "parent" keeps track of the parent state which led to a child being visited, and is useful in tracing out the route.

Additionally, for A* search, the private parameters "fsccore", "gscore", and "hscore" are accessible by their respective getter and setter methods. Unlike other techniques, Bidirectional search leverages two parents, viz. "source_parent" and "goal_parent" to keep track of parents of states visited from the source and goal states.

2 Design

This section discusses the strategy underlying each of the search technique.

1. BFS
   BFS works by starting at the source node and exploring all of its neighbour nodes prior to moving to the nodes at the next level. Since Oedipus consists of circular worlds with inter-world routes, the search must avoid falling into cycles. To take care of this, the solution keeps track of all the nodes which have already been explored once.

2. DFS
   Opposite to BFS, DFS is designed to start at the source node and keep exploring one of its neighbour branches as far as possible. If the goal is not reached, the algorithm backtracks and proceeds with other possible branches along the way. Similar to BFS, we keep the track of nodes which have already been visited in order to avoid loops.

3. BestF
   Falling into the informed category of searches, the design for BestF leverages a heuristic, i.e., Euclidean polar distance to predict how close the next node is to the goal. The node with the least distances is considered as the most promising node to be chosen next for exploration. Mathemetically, if (r1, α_1) and (r2, α_2) be two points in a polar Euclidean space, then the distance 'd' between them is given by:

   $$d = \sqrt{r_1^2 + r_2^2 - 2r_1r_2cos(\alpha_1 - \alpha_2} \tag{1}$$

   In particular, if a node N has neighbours N_1, N_2, N_3 with corresponding distances d_1, d_2, d_3 in the order $d_2 < d_3 < d_1$ then the algorithm selects N_2 as the first node to be visited next. BestF avoids loop by keeping track of visited nodes.

4. AStar

   Unlike Best-first search relying solely on the distance of each adjacent node from the goal to decide on the best node, A* decides the best node to traverse using a f-value which is the sum of two values:

   (a) g = the cost to move from the source node to a given node on Oedipus,

   (b) h = the estimated cost to move from that node to the goal node.

   Similar to Best First search, we leverage the polar distance (See eqn 1) as the heuristic. Algorithm 2 depicts the update rule for frontier.

Algorithm 2 Algorithm for selecting node in A*

```
1: procedure FRONTIER UPDATE
2:     state ← current node
3: For child in state.adjacent_nodes:
4:     temp_gscore ← d(child, state)
5:     child_hscore ← d(child, goal)
6:     temp_fscore ← temp_gscore + child_hscore
7:     if child.IsVisited & temp_fscore >= child_fscore then
8:         continue
9:     if !child.IsVisited || temp_fscore < child_fscore then child.parent =
   state child_gscore ← temp_gscore child_fscore ← temp_fscore
10:        if Frontier.contains(child) then Frontier.remove(child)
           Frontier.push(child)
```

5. Bidirectional Search

   The design for bidirectional search incorporates the idea of running two simultaneous searches: one forward from the source node and one backward from the goal node. As depicted in algorithm 3, the procedure terminates as soon as we find a node that is common to the explored path in both the searches. The current program intends to leverage BFS and BestF techniques for the forward and backward search.

Algorithm 3 Bidirectional Search

```
1: procedure BIDIREC
2:     source, goal ← the source and the goal nodes for search
3:     sourceExploredStates ← list of states explored from source node
4:     goalExploredStates ← list of states explored from goal node
5:     sourceExploredStates.add(source)
6:     goalExploredStates.add(goal)
7: while (!sourceExploredStates.isEmpty() & !goalExploredStates.isEmpty()):
8:     sourceExploredStates ← BFS(source)
9:     goalExploredStates ← BFS(goal)
10:    if sourcExploredStates ∩ goalExploredStates then
11:        break
```

3 Implementation details

All the search techniques leverage a list of states to keep track of visited nodes.

1. BFS
   BFS uses a queue data structure for frontier. The first-in-first-out structure of a queue guarantees that the nodes explored earlier are visited first. If a neighbouring node is not there already in the list of visited nodes, then it is added to the frontier and list of visited nodes while its parent is assigned to be the previous node.

2. DFS
   In contrast to BFS, DFS leverages a stack data structure for frontier. The last-in-first-out property of stack ensures that the most recently added node is the one to be visited first. The search thus progresses along a branch until stepping on a node whose all neighbours have been visited and backtracks from there.

3. BestF
   Since best first search needs to keep track of the polar distance of each adjacent node to the goal and prioritize them accordingly, a priority queue is used as the frontier. At each step of search, we consider all adjacent nodes from the current node and identify the one with the least polar distance from the goal node. A manual comparator function prioritizes the adjacent nodes on the basis of increasing polar distance from the goal. We proceed with the general criteria of Breadth First Search to explore each of the neighbours of the next node except that the priority queue sorts the nodes on the basis of increasing order of distance.

4. AStar
   Similar to best first search, A* also uses a priority queue to arrange the visited nodes per the heuristic. However, instead of tracking the distance of each adjacent node from the goal, the frontier operates on the basis of the F-value (See algorithm 2).

5. Bidirectional Search
   Since the underlying algorithm for bidirectional search are BFS and BestF, all its data structures resemble with these. However, instead of keeping just one list of visited nodes and a frontier, we maintain two of these for the simultaneous searches from the source and the goal node.

4 Evaluation

An example of source = (2,0), goal = (2,135) and N = 5 is used to evaluate the working of all the methods by demonstrating few steps for a concise illustration. For each route traced out, the total cost is calculated by adding over the Euclidean polar distances between successive nodes.

1. BFS
   In Step 1 (See Table 3), the search explores the next possible transitions for source, ends up with $\{(2, 45), (2, 315), (1, 0), (3, 0)\}$, and places (2,45) at the front of the queue - to be visited on Step 2. The algorithm progresses in a similar fashion till Step 12 when the next node in the frontier is (2,135), which is the goal. The search thus terminates at Step 13 with the following solution:

   Route : (2,0) => (2,45) => (2,90) => (2,135)
   Directions: [H90, H90, H90]
   Total cost of path: 5.846

Table 1: First and last three steps for BFS

Step	Current Node	Expanded Nodes	Frontier
1	(2,0)	{}	{(2,45), (2,315), (1,0), (3,0)}
2	(2,45)	{(2,0)}	{(2,315), (1,0), (3,0), (2,90), (1,45), (3,45)}
3	(2,315)	{(2,0), (2,45)}	{(1,0), (3,0), (2,90), (1,45), (3,45), (2,270), (1,315), (3,315) }
11	(3,315)	{(2,0), (2,45), (2,315), (1,0), (3,0), (2,90), (1,45), (3,45), (2,270), (1,315)}	{(4,0), (2,135), (1,90), (3,90), (4,45), (2,225), (1,270), (3,270), (4,315)}
12	(3,45)	{(2,0), (2,45), (2,315), (1,0), (3,0), (2,90), (1,45), (3,45), (2,270), (1,315), (3,315)}	{(2,135), (1,90), (3,90), (4,45), (2,225), (1,270), (3,270), (4,315)}
13	(2,135)	Terminate	-

2. DFS
   Table 2 shows the first three and the last two out of 22 steps of DFS. The last node in the frontier is the one to be visited at each subsequent step. The final solution details are listed below:

   Route: (2,0) => (2,315) => (2,270) => (2,225) => (2,180) => (2,135)
   Directions: [H270, H270, H270, H270, H270]
   Total cost of path: 9.427

3. BestF
   At Step 1 (Table 3), the algorithm discovers that (1,0) with the least distance of 2.99, is the best possible adjacent neighbour. Step 2, therefore, considers (1,0) as the current node and computes the distance of its adjacent nodes followed by an update of the frontier. The solution details are as follows:

Table 2: Steps for DFS

Step	Current Node	Expanded Nodes	Frontier
1	(2,0)	{}	{(2,45), (2,315), (1,0), (3,0)}
2	(3,0)	{(2,0)}	{(2,45), (2,315), (1,0), (3,45), (3,315), (4,0)}
3	(4,0)	{(2,0), (3,0)}	{(2,45), (2,315), (1,0), (3,45), (3,315), (4,45), (4,315)}
21	(3,135)	{(2,0), (3,0), (4,0), (4,315), (3,315), (2,315), (1,315), (1,270), (2,270), (3,270), (4,270), (4,225), (3,225), (2,225), (1,225), (1,180), (2,180), (3,180), (4,180), (4,135)}	{(2,45), (2,315), (1,0), (3,45), (3,315), (4,45), (4,270), (3,270), (2,270), (1,0), (1,225), (2,225), (3,225), (4,180), (3,180), (2,180), (1,135), (2,135), (3,135), (4,90), (3,90), (2,135) }
22	(2,135)	Terminate	-

Route: (2,0) => (1,0) => (1,45) => (1,90) => (1,135) => (2,135)
Directions: [H360, H90, H90, H90, H180]
Total cost of path: 4.923

Table 3: Best First Search for source = (2,0), goal = (2,135), N = 5

Step	Current Node	Expanded Nodes	Frontier with polar distance
1	(2,0)	{}	{(1,0): 2.99, (2,315): 3.58, (2,45): 3.40, (3,0): 4.99}
2	(1,0)	{(2,0)}	{(1,45): 2.61, (1,315): 2.72, (3,0): 4.99, (2,315): 3.58 , (2,45): 3.40}
5	(1,135)	{(2,0), (1,0), (1,45), (1,90)}	{(2,135): 0.0, (1,315): 2.71, (1,180): 1.70, (2,315): 3.57, (2,45): 3.40, (3,0): 4.99, (2,90): 1.94}
6	(2,135)	Terminate	-

4. AStar
   (2,0) in Step 1 (Table 4) has the same F-score as that of (1,0). We, therefore, **break the tie** by visiting (1,0) in step 2 while skipping (2,0)

by adding it to the explored nodes. While transiting from Step 3 to 4, the search finds the node (1,90) with a F-score of 4.65 which is lower than that of (2,315). (1,90) is subsequently expanded in Step 5. The final solution details are the same as the best first search:

Route: (2,0) => (1,0) => (1,45) => (1,90) => (1,135) => (2,135)
Directions: [H360, H90, H90, H90, H180]
Total cost of path: 4.923

Table 4: A-star search process with source = (2,0), goal = (2, 135) and N = 5

Step	Current Node	Expanded Nodes	Frontier with F-scores
1	(2,0)	{}	{(2,0): 3.99, (1,0): 3.99, (2,315): 5.20, (2,45): 5.35, (3,0): 5.99}
2	(1,0)	{(2,0)}	{(1,315): 4.53, (1,45): 4.58, (3,0): 5.99, (2,45): 5.35, (2,315): 5.20 }
5	(1,90)	{(2,0), (1,0), (1,315), (1,45)}	{(1,135): 4.92, (2,315): 5.20, (2,90): 5.89, (1,270): 5.78, (2,45): 5.35, (3,0): 5.99}
6	(1,135)	{(2,0), (1,0), (1,315), (1,45), (1,90)}	{(2,135): 4.92, (2,45): 5.35, (2,315): 5.20, (1,270): 5.78, (3,0): 5.99, (1,180): 6.59, (2,90): 5.89}
7	(2,135)	Terminate	-

5. Bidirectional Search
   The simultaneous search processes from the source and goal nodes greatly reduce the number of search steps for the bidirectional search. Table 5 depicts the search which in contrast to BFS (table 3), finds the route in just two steps. The final solution details are listed below:

   Route: (2,0) => (2,45) => (2,90) => (2,135)
   Directions: [H90, H90, H90]
   Total cost of path: 5.846

4.1 Cost analyses

In terms of the path cost, A* search consistently performs better or as well as other algorithms. As depicted in Table 6, A* performs on par with Bidirectional (with BFS) and BFS in three out of four cases and outperforms both in one instance. DFS, on other hand, lags behind other methods.

Table 5: Bidirectional search with source = (2,0), goal = (2, 135) and N = 5

Step	Current Node		Expanded Nodes		Frontier with F-scores	
	From source	From goal	From source	From goal	From source	From goal
1	(2,0)	(2,135)	{}	{}	{(2,45), (2,315), (1,0), (3,0)}	{(2,180), (2,90), (1,135), (3,135)}
2	(2,45)	(2, 180)	{(2,0)}	{(2,135)}	{(2,315), (1,0), (3,0), (2,90), (1,45), (3,45)}	{(2,90), (1,135), (3,135), (2,225), (1,180), (3,180)}

In terms of the number of search states visited, Bidirectional search with BestF outperforms all others in average followed by general best first search. In contrast, uninformed techniques such as BFS and DFS with broader search area lag behind with bigger search paths.

Table 6: Path costs and number of search states for different problems: note that no. of search states visited in Bidirec is one each from source and goal node.

Problem	Algorithm	No. of search states visited	Path cost
source = (1,315), goal = (4,45), N = 5	BFS	25	4.791
	DFS	27	26.384
	BestF	7	6.58
	AStar	9	4.791
	Bidirec (BFS)	4*2	4.791
	Bidirec (BestF)	4*2	4.791
source = (4,0), goal = (7,90), N = 8	BFS	43	10.794
	DFS	45	42.816
	BestF	5	10.794
	AStar	13	10.794
	Bidirec (BFS)	6*2	10.794
	Bidirec (BestF)	4*2	16.640
source = (2,45), goal = (9,225), N = 10	BFS	71	14.795
	DFS	43	40.653
	BestF	18	13.714
	AStar	18	12.897
	Bidirec (BFS)	28*2	21.298
	Bidirec (BestF)	6*2	30.568
source = (7,270), goal = (7,90), N = 10	BFS	24	26.175
	DFS	36	27.282
	BestF	16	21.585
	AStar	24	15.897
	Bidirec (BFS)	3*2	26.175
	Bidirec (BestF)	2*2	26.175