CS5011: Assignment 1 - Search - Robot-Assisted Evacuation

(Word Count :3000, Estimated time taken to complete the essay: 36 Hours) Student ID: 180029539

Implemented Parts of Search Techniques:

- I have implemented the two uninformed search techniques and tested all of the maps and showed some of the map's results in the report below.
- I have also implemented both the informed search techniques and tested all of the maps and showed some of the maps's results in the report below.
- I have then implemented the A* search by taking manhattan distance as the heuristic and compared with the results of euclidean distance

Working Functionalities of the Search Techniques:

- All the functionalities of the uninformed search algorithms are working in the code. They all give the output as they are supposed to. I have also validated them by printing the output of each iteration in the output.
- All the functionalities of the informed search algorithms are working in the code. They all give the output as they are supposed to. I have also validated them by printing the output of each iteration in the output.

Real-time Applications of search algorithms:

There are a lot of real-time applications for search algorithms. They can be seen in very familiar places like post offices, libraries, hospitals. Search algorithms are used in courier services to plan the routes. They make use of the algorithms that give the shortest path and are complete like the informed search techniques. They are used in games like chess and checkers to find the optimal solution to win against a human player. They are used in library systems to find the resources easily. It is also used in major applications like google maps to give the direction between two places really quickly. It is also used in pages like wikipedia to search for the right content searched by the user. Mostly informed search algorithms are used in real time applications due to the reduced time complexities. But, there is always a chance to overestimate a heuristic value.

Part 1 - Uninformed Search Techniques

Design and Testing:

1) Introduction:

State Space:

- All the possible states that can be reached by the robot.
- {A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P} for Adjacency matrix of order 16
- {A,B,C,D,E,F,G} for Adjacency matrix of order 7

Initial State:

- The initial state is defined as the state from which the robot starts exploration.
- It can be derived by getting either the row or column in the adjacency matrix with the value of 2

Goal State:

- The Goal state is defined as the exit to be reached by the robot.
- It can be derived by getting either the row or column in the adjacency matrix with the value of 8.

Actions:

- Travelling from one location to the next feasible location by the robot.

Path Cost:

- The path cost is assumed to be 1 as per the requirements for the part 1.
- For part 2, the path cost I have considered is the euclidean distance between the present state and the state from which it travelled from.

2) Implementation of BFS and DFS:

- We start by creating a common class to be used for uninformed search techniques from where the execution starts(the main method included).
- Next, I have created a generic node class with several functions like makeNode() method which takes properties like state and parent node and returns a node.I have also created other functions like goalTest() to test if the given node is goal node or not, and expandBFS() or expandDFS() to create child nodes of a given node by taking inputs like input map, respective frontier, explored nodes and present node.
- Next, I also created createRootNode() and createGoalNode() methods to create the initial node and the goal node. I have achieved this by traversing the diagonal of the adjacency matrix and returning row numbers of entries ‘2’ and ‘8’ and created nodes by using their respective states.
- Next, I have created an Agent class through which we will traverse the problem state space.
- Through the Agent class, we will perform the search techniques of BFS and DFS through bfsSearch() and dfsSearch() which take root node, goal node and the map to search and give you a possible route it exists.
- I have also created a mapChoice() method for the agent for taking user input to select the map.
- I have understood from my research that using a stack for DFS and queue for BFS would be optimal.

- Then, I created checkQueue(), checkStack() methods for checking elements in the frontier and displayQueue(), displayStack() for displaying elements in the frontier.
- Later, created the array of Character to hold the explored nodes, and overridden the contains method to search for Character objects in the array.
- Also, created the searchExecution() method to allow the user to select between DFS and BFS implementations.
- Till here the code is the same for both the DFS and BFS implementation.

Breadth First Search:

- For BFS, I have created the method breadthFirstSearch() which takes root node, goal node and the selected ‘map’ as parameters.
- First, I initialised a queue that is used as the frontier for BFS and a character array for storing the explored nodes.
- I have also initialised an arraylist of nodes to hold the expanded child nodes in each iteration.
- I have also created a node current node to store the node that is to be tested and expanded in each iteration, and ‘currentiteration’ to get the number of levels we are traversing(depth).
- Then, I have pushed the root node into the frontier.
- Then, the pseudocode for the BFS algorithm implemented is :
    - Display the current status of the current node, frontier and the explored nodes so far.
    - Check if the current node is goal or not.
    - Expand the children of the current node (by traversing the same row as current node in the adjacency matrix and search for value ‘5’ and also checking if they are not already part of the frontier or explored states array).
    - This occurs inside expandBFS() method as discussed before which return an arraylist with the children nodes.
    - Display the expanded nodes.
    - Remove the current node.
    - Insert the child nodes into the Queue.
    - Check if frontier is empty
        - If Yes, that means Queue is empty and no possible path exists from initial node to goal node. Exit the code displaying the status.
        - If No, update the current node with the head of the queue and loop back to the beginning of the loop.

Depth First search:

- For DFS, I have created the method depthFirstSearch() which takes root node, goal node and the selected ‘map’ as parameters.
- First, I initialised a stack that is used as the frontier for DFS and a character array for storing the explored nodes.
- I have also initialised an arraylist of nodes to hold the expanded child nodes in each iteration.
- I have also created a node ‘currentnode’ to store the node that is to be tested and expanded in each iteration, and ‘currentiteration’ to get the number of levels we are traversing(depth).
- Then, I have pushed the root node into the frontier.
- Then, the pseudocode for the DFS algorithm implemented is :
    - Display the current status of the current node, frontier and the explored nodes so far.
    - Check if the current node is goal or not.
    - Expand the children of the current node (by traversing the same row as current node in the adjacency matrix and search for value ‘5’ and also checking if they are not already part of the frontier or explored states array).
    - This occurs inside expandDFS() method as discussed before which returns an arraylist with the children nodes.
    - Display the expanded nodes
    - Pop the current node
    - Push the child nodes into the Stack
    - Check if frontier is empty
        - If Yes, that means Stack is empty and no possible path exists from initial node to goal node. Exit the code displaying the status.
        - If No, update the current node with the head of the queue and loop back to the beginning of the loop
- As we are using a stack as our frontier unless the max depth is reached it will not traverse the other children at the same level.

3) Choice of Order to add new states(Tie-breaking strategy):

- I have used alphabetical order for the uninformed search techniques as my tie-breaking strategy.
- I have used the same alphabetical order for my informed search techniques when two nodes have the same heuristic cost.

4) Testing the implementation of BFS and DFS:

- I have taken some maps and manually figured out the path traversed for the uninformed search algorithms and compared it with my output of the code.
- I have also tried giving other inputs then specified in the code and saw if they are being handled or not by my code.
- Then, I have printed the current node, expanded nodes, explored nodes, and the contents of the frontier at each level and checked if contents were valid or not.

5) Evaluating DFS and BFS implementations on given maps:

- For all the given maps the DFS completed the search by exploring the least number of nodes. But, the path cost was very high for DFS.
- But, I think this is because all these maps might have their goal states deep inside the tree. Hence, we were able to reach the goal state by traversing less number of nodes using DFS, But the path cost was higher.
- The DFS had less explore nodes but higher path cost. The BFS had higher explored nodes but lower path cost.

6) Example Runs of BFS and DFS:

BFS vs DFS of map6:

{2, 5, 10, 10, 5, 10, 10},
{5, 0, 5, 10, 10, 10, 10},
{10, 5, 0, 5, 10, 10, 10},
{10, 10, 5, 8, 10, 10, 5},
{5, 10, 10, 10, 0, 5, 10},
{10, 10, 10, 10, 5, 0, 5},
{10, 10, 10, 5, 10, 5, 0}

Welcome to Maps of Destruction: **Pick a map from 1 to 9 :6** **Select Search Algorithm: 1)BFS 2)DFS 3)Compare Both 4)Exit: 1** **Iteration 0:** **Current Node: A** **The Frontier Elements are:A** **The Explored States are:** **The Expanded nodes in this iteration are:B,E** **Iteration 1:** **Current Node: B** **The Frontier Elements are:B,E** **The Explored States are: A** **The Expanded nodes in this iteration are:C** **Iteration 2:** **Current Node: E** **The Frontier Elements are:E,C**	**Welcome to Maps of Destruction:** **Pick a map from 1 to 9 :6** **Select Search Algorithm: 1)BFS 2)DFS 3)Compare Both 4)Exit: 2** **Iteration 0:** **Current Node: A** **The Frontier Elements are:A** **The Explored States are:** **The Expanded nodes in this iteration are:E,B** **Iteration 1:** **Current Node: B** **The Frontier Elements are:E,B** **The Explored States are: A** **The Expanded nodes in this iteration are:C** **Iteration 2:** **Current Node: C** **The Frontier Elements are:E,C**

The Explored States are: A,B **The Expanded nodes in this iteration are:F** **Iteration 3:** **Current Node: C** **The Frontier Elements are:C,F** **The Explored States are: A,B,E** **The Expanded nodes in this iteration are:D** **Iteration 4:** **Current Node: F** **The Frontier Elements are:F,D** **The Explored States are: A,B,E,C** **The Expanded nodes in this iteration are:G** **Iteration 5:** **Current Node: D** **The Frontier Elements are:D,G** **The Explored States are: A,B,E,C,F** **The Path traversed to reach goal state is : A->B->C->D** **The Path Cost is:3**	**The Explored States are: A,B** **The Expanded nodes in this iteration are:D** **Iteration 3:** **Current Node: D** **The Frontier Elements are:E,D** **The Explored States are: A,B,C** **The Path traversed to reach goal state is : A->B->C->D** **The Path Cost is:3**

BFS vs DFS of map7:

(an example of no possible path between initial and goal state)

{0, 5, 10, 10, 10, 10, 10},
{5, 0, 5, 10, 10, 10, 10},
{10, 5, 2, 5, 10, 10, 10},
{10, 10, 5, 0, 10, 10, 5},
{10, 10, 10, 10, 8, 10, 10},
{10, 10, 10, 10, 10, 0, 5},
{10, 10, 10, 5, 10, 5, 0}

Welcome to Maps of Destruction: **Pick a map from 1 to 9 :7** **Select Search Algorithm: 1)BFS 2)DFS 3)Compare Both 4)Exit: 1** **Iteration 0:** **Current Node: C** **The Frontier Elements are:C** **The Explored States are:** **The Expanded nodes in this iteration are:B,D** **Iteration 1:** **Current Node: B** **The Frontier Elements are:B,D** **The Explored States are: C** **The Expanded nodes in this iteration are:A** **Iteration 2:** **Current Node: D** **The Frontier Elements are:D,A** **The Explored States are: C,B**	**Welcome to Maps of Destruction:** **Pick a map from 1 to 9 :7** **Select Search Algorithm: 1)BFS 2)DFS 3)Compare Both 4)Exit: 2** **Iteration 0:** **Current Node: C** **The Frontier Elements are:C** **The Explored States are:** **The Expanded nodes in this iteration are:D,B** **Iteration 1:** **Current Node: B** **The Frontier Elements are:D,B** **The Explored States are: C** **The Expanded nodes in this iteration are:A** **Iteration 2:** **Current Node: A** **The Frontier Elements are:D,A** **The Explored States are: C,B**

The Expanded nodes in this iteration are:G **Iteration 3:** **Current Node: A** **The Frontier Elements are:A,G** **The Explored States are: C,B,D** **Iteration 4:** **Current Node: G** **The Frontier Elements are:G** **The Explored States are: C,B,D,A** **The Expanded nodes in this iteration are:F** **Iteration 5:** **Current Node: F** **The Frontier Elements are:F** **The Explored States are: C,B,D,A,G** **No Path exists from Initial State to Goal State thru BFS.**	**Iteration 3:** **Current Node: D** **The Frontier Elements are:D** **The Explored States are: C,B,A** **The Expanded nodes in this iteration are:G** **Iteration 4:** **Current Node: G** **The Frontier Elements are:G** **The Explored States are: C,B,A,D** **The Expanded nodes in this iteration are:F** **Iteration 5:** **Current Node: F** **The Frontier Elements are:F** **The Explored States are: C,B,A,D,G** **No Path exists from Initial State to Goal State thru DFS.**

Part 2 - Informed Search Techniques

1) Heuristics used for the Informed search techniques:

Heuristic for Best First Search:

- The best first search is also called the greedy method.
- In best first search, we assign a heuristic value based on the node properties and expand the nodes based on the heuristic value.
- For this, I thought priority queue would be best for this implementation, as priority queues hold not only the data of the object but are also sorted based on the assigned priority.
- In our case, the priority is the heuristic value. As we have two cartesian coordinates for the states given, I thought it would be best to use the euclidean distance as my heuristic value. The robot is not restricted to 4 directions. It can travel at any angle which makes the use of euclidean distance optimal.
- The heuristic value is calculated for each node when its created. It is equal to the euclidean distance of the present node to the goal node.
- When the node selection happens, the lowest heuristic value is given higher priority.

Heuristic for A* Search:

- The A* search is just a better implementation of the best first search
- In the best first, the heuristic generally calculates the cost value based only on the future. But, A* also considers the cost value of the past.
- We use priority queue similar to the best first search as the implementation is similar. The only change is calculation of the heuristic value.

- In our case, I have taken the euclidean distance to the goal node and added the path cost till that node to calculate the heuristic value of each node.
- These will have a slight advantage as these not only look forward to the goal but also calculates the cost incurred to reach the node.

2) Implementation of Best First search and A* search:

- As the implementation of my informed and uninformed algorithms are similar, I will describe only the differences between the two implementations.
- First, I have added a locChoice() method in the Agent class to retrieve the cartesian coordinates of the selected map.
- In the Node class, I have added extra required variables like heuristic cost, X & Y coordinates. Then, I modified makeNodeAFS() and makeNodeBFS() methods and added functionalities like calculating the heuristic value from the coordinates of present node and goal node.
- The class Node implements the Comparable interface to be used by the Priority Queue
- The rest of the execution is exactly same as uninformed search algorithms.

3) Testing the implementation of Best First search and A* search:

- For testing the implementation of informed search techniques, I have printed the current node, frontier with state and heuristic value at each node, list of explored nodes and the path traversed to go from root node to goal node.
- Then validated by manually traversing the tree and calculating the heuristic values and comparing them at each step.

4) Evaluating Best First search and A* search implementations on given maps:

- I ran the maps given against my formulated algorithms and found out that the number of nodes explored is almost the same in both the algorithms. But, in some of the maps, A* traversed the map by exploring fewer nodes when compared to the best first search algorithm.
- The path cost was almost similar in both the scenarios. As, the path traversed from root node to goal node was almost same in majority of the situations.

Running the Implementations:

Uninformed Search Techniques:

- Run the code ’ java -jar Search1.jar ‘ and follow the instructions. No parameters required.

Informed Search Techniques:

- Run the code ’ java -jar Search2.jar ‘ and follow the instructions. No parameters required. Same for the part 3. Just run ’ java -jar Search3.jar ‘. No parameters.