STUDENT ID- 103528424

STUDENT NAME- SHIFAT BIN RAHMAN

N-PUZZLE IMPLEMENTATION FOR UNINFORMED AND INFORMED SEARCH ALGORITHMS

**INSTRUCTIONS**

1. Goto the zip file and start the cmd in the directory of the folder.
2. Then type out the commands for the search algorithms initialization nPuzzler.bat <filename> <method>
3. Repeat step 2 for different search method algorithms

**FEATURES**

1. BFS with repeated state checks done for both NxN and NxM puzzles.
2. DFS with repeated state checks done for both NxN and NxM puzzles.
3. GBFS with repeated state checks done with heuristic sorting for both NxN and NxM puzzles.
4. A\* Search with repeated state checks done with heuristic sortings for both NxN and NxM puzzles.
5. Implemented Depth limit to DFS so that it doesn’t stack overflow when searching way too many nodes without finding any solution.
6. Invalid arguments and wrong readme file will now prompt the user with the issue
7. Implemented NxM puzzle format for any algorithm
8. Fixed the puzzle size larger than 80 puzzle, so now double digits in the puzzle are working as intended

**KNOWN BUGS**

StackOverflow Error for the DFS solution even if it the depth is set to low value

**LIMITATIONS**

1. Did not manage to fix much of the slowness of the program.
2. Did not manage to implement the node expansion direction order of the program.
3. A\* search algorithm for NxM puzzle bigger than 3x3 gets quite slow. (For 4x3 it is above 30 minutes and more).

**ACKNOWLEDGEMENTS/RESOURCES**

1. <https://www.geeksforgeeks.org/best-first-search-informed-search/>

For understanding how BFS works in pseudocode

2. https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/

For understanding how DFS works in pseudocode

1. <https://www.cs.princeton.edu/courses/archive/spr18/cos226/assignments/8puzzle/index.html>

For explaining how the 8-puzzle works in Discussion

1. <https://ai.stackexchange.com/questions/8902/what-are-the-differences-between-a-and-greedy-best-first-search>

For explaining the difference between the two algorithms in Discussion

DISCUSSION

*Problem*- 8-Puzzle is a 3x3 grid style puzzle that is labeled from 1 to 8 and a blank tile, where the purpose of the puzzle is to go from the starting problem state to the end goal state by sliding a blank tile by the lowest moves possible. A state is a ‘screenshot’ of the entire puzzle grid. When a tile gets slided in a direction, the opposite number tile to the blank tile gets exchanged with the blank tile. The permissible directions for sliding the tile are up, down, left, and right.

*Solution*- The solution to finding an optimal goal state is by using Tree Search Algorithms. There are 2 types of searches implemented here to find the solution. Uninformed and informed search. Uninformed search consists of BFS and DFS whilst informed search is GBFS and A\* algorithms for this program. Uninformed search does not take advantage of any state-specific knowledge to make the search more efficient. Informed search however does.

*Terminologies*-

Searching: It is a step-by-step process for resolving a search problem in a certain search space. There are three basic variables that can contribute to a search problem:

*Search Space*: A search space is a collection of probable solution states that a system could have.

*Start State*: This is the state in which the agent starts the search.

*Goal test*: This is a function that looks at the current state and returns whether or not the goal state has been achieved.

Search tree: A Search tree is a tree representation of a search problem. The root of the search tree is the node that corresponds to the initial state.

Actions: It describes all of the actions available to the agent.

Transition model: A transition model is a description of what each action does.

Path Cost: It is a function that assigns a real cost to each path.

Solution: It is an action sequence that connects the start node to the goal node. If it is optimal then it is called an optimal solution.

*Properties-*

There are 4 types of properties that we can associate with each search algorithm.

Completeness: A search algorithm is said to be complete if it guarantees to return a solution for any random input if at least one solution exists.

Optimality: A solution found for an algorithm is said to be optimal if it is guaranteed to be the best solution (lowest path cost) among all other solutions.

Time complexity: It is a measure of how long it takes an algorithm to complete its task.

Space Complexity: The maximum storage space required at any point during the search, as determined by the problem's complexity.

Time and space complexity are measured by:

b - maximum branching factor of the search tree

d - depth of the least-cost solution

m - maximum depth of the state space (may be infinite)

*Overall Code Structure and Functionality-*

The main method of the program lies in the nPuzzler class. Here the main class takes in the arguments first for the program. Then after the correct arguments are read in, it initializes the correct search method from the argument and reads the text file and parses and formats it into a nPuzzle class which contains the start and goal states of the puzzle. After that the initialized method was an object of SearchMethod which is later inherited by the appropriate search strategy method classes. Each search algorithm classes make use of LinkedLists of the PuzzleState class objects (nodes), to store and retrieve their Frontier queues for LIFO or FIFO. After that it starts the expanding of nodes and compares each child node to see if the node is the goal state or node here. If not, then continue checking others and when done adds it to the frontier and the Searched list so that the node isn’t repeatedly checked elsewhere in the tree. In informed searches, however, there is an evaluation function check before each node is added to frontier, and then the frontier is sorted by their evaluation function. Then it expands the lowest node with the heuristic and so on until it finds the solution.

*Uninformed Searches-*

1.Breadth-First Search-

A search algorithm is deemed complete if it is always guaranteed to find a goal state if one exists. BFS is definitely complete even without any repeated state check. It traverses through the tree by searching and expanding all the neighboring nodes before going deeper. This ensures that it finds the shortest path. The time complexity of such search algorithm is *O(bd+1)*. The space complexity, however, is *O(bd+1).* So, space is the biggest problem here since it is keeping lots of nodes in memory. But it is optimal since each step costs only 1.

The tree search is implemented by using a FIFO queue for the frontier. Which means First In First Out. The first element inserted into the queue is always one first one that gets removed later. Then while the frontier is not empty, it loops through each item off the list and checked to see if its equal to the goal node, if not then its expanded and all the child nodes are added to the frontier to be searched again.

2.Depth-First Search-

DFS is ***not***complete without any repeated state checks in place. This is because it gets into an infinite loop when in cycles or fails in infinite depth. That’s why I implemented repeated ***and*** depth check so that it doesn’t get into an infinite loop or stack overflow. It traverses through the tree by searching and expanding as many nodes in depth as possible before backtracking. The time complexity of such search algorithm is *O(bm)* which is linear space. But the space complexity however is *O(bm),* which is worse if m is larger than d.

The tree search is implemented by using a LIFO queue for the frontier. Last In First Out. The last element added in the frontier is taken out first. So, implementing this in the code is basically the same as BFS just that we remove the Frontier elements from the last always.

*Informed Searches-*

3. Greedy Best-First Search-

GBFS is an informed search algorithm. So, it uses a heuristic evaluation to sort the Frontier queue in ascending order and then it always expands the nodes with the lowest heuristic values. Its completeness and optimality though are the same as DFS while time and space complexity are both *O*(*bm* ).

While the frontier is not empty, it loops through each item off the queue and checks to see if its equal to the goal node, if not then it expands the node and sorts the heuristic for the evaluation function. The heuristic function is calculated by looping and checking whether or not the number of child nodes match the goal state. If it doesn’t then the heuristic value gets incremented.

4. A-Star Search-

A\* is also an informed search algorithm. It is almost like GBFS except for the fact that it also considers the cost required to get to the current node. In the evaluation function, it calculates by adding the heuristic and the cost up then sorts it through the frontier to get the best optimal path. This is what makes A\* optimal every time.

Unless there are infinite nodes, it is always complete. The space complexity here is proportional to the number of nodes in the Frontier and length of the found path. The time complexity is *O(bm),* however, it needs to keep all the nodes in memory besides the Frontier.

*Program Overview Diagram (UML) –*

Timeline

Description automatically generated

*Search Tree Example-*

Implementing a 2x3 puzzle in GBFS. To show the states in each stage I have made a method in PuzzleState class that takes in the 2D array of the Puzzle to print out the states for each expansion:

Text

Description automatically generated

The directions to solution:

A screenshot of a computer

Description automatically generated with medium confidence

Note: The print method for the puzzle is rotated 90 degrees to right because that’s how the 2D array is being stored. Also the last move doesn’t show up.

**NM-Puzzle-test.txt**

2x3

1 2 3 4 5 0

1 2 5 3 4 0

The root node starts at the first node. Then it expands nodes in both directions left and up. And since it’s a GBFS, it saw the up-direction node had a lower heuristic value than the left one. So, it chose to expand that further and heuristic values lower. Finally in the 5th node it has found the goal state.

Diagram, engineering drawing

Description automatically generated

Right



|  |  |
| --- | --- |
| 1 | 2 |
| 5 | 3 |
| 4 | 0 |