|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command Run | Problem | Solution Cost | Number of nodes  visited | Avg CPU Time  (seconds) |
| BreadthFirstEightPuzzle | 0 1 2 4 5 3 7 8 6 | 4 | 16 | 0.077 |
| DepthFirstEightPuzzle | 0 1 2 4 5 3 7 8 6 | 1062 | 1069 | 0.428 |
| ManhattanEightPuzzle | 0 1 2 4 5 3 7 8 6 | 4 | 5 | 0.077 |
| NumDisplacedEightPuzzle | 0 1 2 4 5 3 7 8 6 | 4 | 5 | 0.084 |
| BreadthFirstEightPuzzle | 1 2 3 4 0 8 7 6 5 | 6 | 97 | 0.084 |
| DepthFirstEightPuzzle | 1 2 3 4 0 8 7 6 5 | 972 | 979 | 0.425 |
| ManhattanEightPuzzle | 1 2 3 4 0 8 7 6 5 | 6 | 12 | 0.077 |
| NumDisplacedEightPuzzle | 1 2 3 4 0 8 7 6 5 | 6 | 15 | 0.088 |
| BreadthFirstEightPuzzle | 3 1 2 4 5 6 0 7 8 | 14 | 4946 | 0.133 |
| DepthFirstEightPuzzle | 3 1 2 4 5 6 0 7 8 | 76412 | 79266 | 385.218 |
| ManhattanEightPuzzle | 3 1 2 4 5 6 0 7 8 | 14 | 108 | 0.107 |
| NumDisplacedEightPuzzle | 3 1 2 4 5 6 0 7 8 | 14 | 240 | 0.120 |
| BreadthFirstEightPuzzle | 8 6 7 2 5 4 3 0 1 | 31 | 58589872 | 8286.130 |
| DepthFirstEightPuzzle | 8 6 7 2 5 4 3 0 1 | 89237 | 95339 | 482.747 |
| ManhattanEightPuzzle | 8 6 7 2 5 4 3 0 1 | 31 | 7736 | 1.425 |
| NumDisplacedEightPuzzle | 8 6 7 2 5 4 3 0 1 | 31 | 186686 | 1999.487 |
| BreadthFirstEightPuzzle | 6 4 7 8 5 0 3 2 1 | 31 | 56846268 | 7620.279 |
| DepthFirstEightPuzzle | 6 4 7 8 5 0 3 2 1 | 58761 | 59792 | 207.686 |
| ManhattanEightPuzzle | 6 4 7 8 5 0 3 2 1 | 31 | 7774 | 1.405 |
| NumDisplacedEightPuzzle | 6 4 7 8 5 0 3 2 1 | 31 | 188943 | 1886.840 |

This data was collected over the course of 10 runs of each algorithm with each problem set. The version of the program used Program Milestone 2. I did not modify the code based on the suggestions until after the testing was done. The specs of the machine used were as follows:

Asus Rampage IV Extreme Motherboard

Intel i7-3970X 6 Core/12 Thread 3.5GHz (4.0GHz Turbo) Processor

32GB of Corsair Dominator Platinum RAM

EVGA GTX 690

Samsung 840Evo 1TB SSD

Linux Mint 16

All tests were run giving the JVM 64GB of maximum memory and 20GB of initial stack space. There was a 64GB swap partition used on the SSD.

The first 3 puzzles provided were various tests that I knew could be completed in a reasonable time on the school's computers. The last two represent worst-case scenarios for the 8-puzzle problem. The worst-cases require 31 moves to solve optimally.

As far as the quality of the solution yielded, it is clear that depth-first search is a very bad way to go. All of the other algorithms were able to solve in the optimal number of moves. Meanwhile, depth-first required hundreds if not thousands of moves for the solution it found.

This is because it picked a path down one branch of the tree and only turned back if it found that it was backtracking. This can be seen by noting that the number of moves in the solution found is only slightly less (relatively speaking) than the number of nodes tested. This indicates relatively little backtracking occurred.

As far as the number of nodes visited, the outcome depends on the puzzle tested. For non-worst cases, depth first was the worst because it went very far down a branch to find the solution. In the worst case scenarios, breadth-first performed very poorly, having visited millions of nodes before finding the solution. This is because in the worst-cases, the solution was at a depth of 31 nodes. This meant the breadth-first search had to expand all of the nodes in the tree for depth < 31. When it got the the 31st level, it was able to find a solution. This illustrates the danger of using a breadth first search for potentially difficult problems like this. If this algorithm had been tried with this problem on a school computer, no solution would have been reached because there would not have been enough memory.

Interestingly, the worst-case scenarios produced very different results for the two different heuristic methods. The Manhattan Distance algorithm was able to find a solution in either case by visiting approximately 7700 nodes while the Number-of-nodes-displaced algorithm took even more resources than depth first search to find the solution. In both cases, an optimal solution was found, but this clearly shows that the Manhattan Distance algorithm is superior to the Number-of-nodes-displaced algorithm.

The rest of the statistics from these tests illustrate the resources consumed by each algorithm. In worst cases, using anything other than the Manhattan distance algorithm on a weak computer will result in no solution being found. My SSD still hasn't forgiven me for all of the paging faults.