## Item 1: DFS Performance

Length of solution path found: 40 edges

Number of nodes expanded: 40

MAX_OPEN_LENGTH: 7

Length of solution path found is the number of operations performed to reach the goal state using the certain algorithm from the initial state. This is important when comparing alternative search algorithms because it shows the number of steps to take to reach the goal state (shortest path or not). Number of nodes expanded is the number of times to take a state out of open list and put into close list before reaching the goal state. This is important when comparing alternative search algorithms because it shows how efficient the algorithm is or how long it takes to find the solution. MAX_OPEN_LENGTH is the maximum number of states in the open list at a time. This is important when comparing alternative search algorithms because it shows the amount of memory the algorithm uses.

## Item 4: Alternative Search Methods for the Towers of Hanoi

| Algorithm name | Length of solution path | Number of nodes expanded | MAX_OPEN_LENGTH |
|---|---|---|---|
| Iterative depth-first search | 40 | 40 | 7 |
| breadth-first search | 15 | 70 | 16 |
| iterative-deepening depth-first search | 15 | 441 | 7 |

## Item 5: Blind Search on My A2 Problem Formulations

Farmer, Fox, etc.

| Algorithm name | Length of solution path | Number of nodes expanded | MAX_OPEN_LENGTH |
|---|---|---|---|
| Iterative depth-first search | 7 | 7 | 3 |
| breadth-first search | 7 | 9 | 2 |
| iterative-deepening depth-first search | 7 | 42 | 3 |

Find the Number (Default: Guess Number = 2, Max Number = 10)

| Algorithm name | Length of solution path | Number of nodes expanded | MAX_OPEN_LENGTH |
|---|---|---|---|
| Iterative depth-first search | 8 | 12 | 14 |
| breadth-first search | 4 | 91 | 93 |
| iterative-deepening depth-first search | 4 | 125 | 10 |

## Item 8: Heuristics for the Eight Puzzle

h_euclidean

| puzzle instance name | puzzle instance permutation | success (yes/no) | count of expanded nodes | aborted (yes/no) |
|---|---|---|---|---|
| puzzle10a | [4, 5, 0, 1, 2, 3, 6, 7, 8] | yes | 16 | no |
| puzzle12a | [3, 1, 2, 6, 8, 7, 5, 4, 0] | yes | 38 | no |
| puzzle14a | [4, 5, 0, 1, 2, 8, 3, 7, 6] | yes | 54 | no |
| puzzle16a | [0, 8, 2, 1, 7, 4, 3, 6, 5] | yes | 165 | no |

h_hamming

| puzzle instance name | puzzle instance permutation | success (yes/no) | count of expanded nodes | aborted (yes/no) |
|---|---|---|---|---|
| puzzle10a | [4, 5, 0, 1, 2, 3, 6, 7, 8] | yes | 30 | no |
| puzzle12a | [3, 1, 2, 6, 8, 7, 5, 4, 0] | yes | 59 | no |
| puzzle14a | [4, 5, 0, 1, 2, 8, 3, 7, 6] | yes | 141 | no |
| puzzle16a | [0, 8, 2, 1, 7, 4, 3, 6, 5] | yes | 402 | no |

h_manhattan

| puzzle instance name | puzzle instance permutation | success (yes/no) | count of expanded nodes | aborted (yes/no) |
|---|---|---|---|---|
| puzzle10a | [4, 5, 0, 1, 2, 3, 6, 7, 8] | yes | 12 | no |
| puzzle12a | [3, 1, 2, 6, 8, 7, 5, 4, 0] | yes | 16 | no |
| puzzle14a | [4, 5, 0, 1, 2, 8, 3, 7, 6] | yes | 37 | no |
| puzzle16a | [0, 8, 2, 1, 7, 4, 3, 6, 5] | yes | 79 | no |

h_custom

| puzzle instance name | puzzle instance permutation | success (yes/no) | count of expanded nodes | aborted (yes/no) |
|---|---|---|---|---|
| puzzle10a | [4, 5, 0, 1, 2, 3, 6, 7, 8] | yes | 13 | no |
| puzzle12a | [3, 1, 2, 6, 8, 7, 5, 4, 0] | yes | 34 | no |
| puzzle14a | [4, 5, 0, 1, 2, 8, 3, 7, 6] | yes | 52 | no |
| puzzle16a | [0, 8, 2, 1, 7, 4, 3, 6, 5] | yes | 143 | no |

## Item 9: Evaluating my Custom Heuristic

(a) My heuristic looks for the average of h_manhattan and h_euclidean, which have the better performance in this game. However, when the successors all have the same estimate cost, we need a second estimator to make the decision closer to goal state. Therefore, I calculate the average of h_manhattan and h_euclidean cost estimators.

(b) It outperforms h_euclidean and h_hamming in solving all the puzzles. However it performs worse than h_manhattan because in this ‘8puzzle’ game, manhattan distance is the best estimator due to the movement property of tiles.

(c) It costs negligibly more than others because all of their runtimes are O(1) for each state to calculate its heuristic value.