**Team: Marvin Biscocho (ONLINE)**

**Class: CS 440 Artificial Intelligence**

**Homework 1**

**Statement of Individual Contribution:**

Marvin: architected the Unified Top Level Search for both parts, focused on the pocket cube/rubik’s algorithm and results. Developed the algorithms for 1.1

**Part 1 (for everybody):**

1. For every algorithm in 1.1 (DFS, BFS, Greedy, A\*) and every one of the three mazes (medium, big, open): give the maze with the computed path, the solution cost, and the number of expanded nodes (12 cases total).
2. For 1.2, for each of the three mazes (tiny, small, medium): give the solution path, solution cost, and number of expanded nodes for your A\* algorithm. Discuss your heuristic, including its admissibility.

**Part 2 (for four-credit students):**

1. Discuss your implementation and heuristic. For each of the three inputs (1.1-1.3): give solution paths, number of expanded nodes, running time.

Execution:

Execute the Rubiks.exe <full path to file> <full path to goal state file>

Screen Prompts:

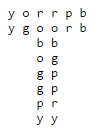
<add screenshots>

* Select Non Rotational Invariance
* Select number for depth (recommendation, depths > 6 take longer time to run as an increased number of nodes may be added to the frontier)
* Select a value to inflate/deflate the heuristic.

Implementation:

We map each state as a List<List<char>> in C#. Similar to the pacman maze in the first problem.

IE: cube1\_1.txt

Node class is developed to store the current state map, goal state map, cost variables (f, g, h). It also has a method to return a list of 11-12 states (1 state for each possible 90 degree rotation (T, T’, Ba, Ba’, Bo, Bo’, F, F’, R, R’, L, L’). Routines can also compare current state vs goal state and return the number of matching tiles. The node is left in the 2D state, we know each (x,y) coordinate that represents a tile on a face and properly rotate colors based on the given rotation.

The list of nodes represent the frontier that needs to be explored next, based on cost. With each turn representing a cost of 1, the ‘g’ cost is always 1. The ‘h’ cost is the number of displaced cubes. Each level of the iterative deepening represents, represents the number of sequential moves made. See the screen shot below.

EX: Tree

Start state is at the root level, (d=0, 0 moves made), the level is below, d=1, is 1-sequential move.

Algorithm: Iterative deepening A\*. Reference: [Link](https://heuristicswiki.wikispaces.com/Rubik%27s+cube). As we choose the node which resulted in the least cost, which is influenced by the heuristic chosen. As imagined, but not shown in the Tree, each 1-sequential move can be followed up with another move, which results in potentially 12\*12=144 potential new states. We try to minimize the search space by not allowing sequential moves that result in a previous state (IE: T T’) to be added onto the frontier. That’s why at 2-sequential moves can only add 11 more states.

Heuristic:

The assignment asks students to consider the number of misplaced cubes as the heuristic. Following that request, we don’t believe it is admissible. Consider the scenario below:

|  |  |
| --- | --- |
| **Goal State:** | **Current State:** |
| y y r r b b  y y r r b b  o o  o o  g g  g g  p p  p p | y o r r p b  y o r r p b  b b  o o  g g  g g  p p  y y |

The number of misplaced cubes is 8. Every rotation impacts at least 8 faces. At most 12 faces. But in this scenario 8 faces are impacted.

The heuristic of misplaced cubes will indicate 8 moves are required to our goal state. This is clearly wrong. It overestimates as we can clearly see 1 90 degree rotation of the top will reach our goal state. Now, if we divide by 8, we would get 1 in this case. We believe by dividing by a certain factor, this heuristic can be made admissible.

The homework doesn’t request for optimal solution for Part 2. So, when executing an “inflated” heuristic is used, optimality may be impacted, but enables the search to potentially return a solution faster.

|  |  |
| --- | --- |
| Input 1.1 |  |
| <add screenshots> | Sequence of moves: <add output>  Number of nodes expanded by A\*: <add output>  Search Time: <add output> |
| Input 1.2 |  |
| <add screenshots> | Sequence of moves: <add output>  Number of nodes expanded by A\*:<add output>  Search Time: <add output> |
| Input 1.3 |  |
| <add screenshots> | Sequence of moves: <add output>  Number of nodes expanded by A\*:<add output>  Search Time: <add output> |

1. Discuss modifications to your implementation and heuristic. For each of the three inputs (2.1-2.3): give solution paths, number of expanded nodes, running time. For input 2.1 (=1.1), discuss the differences from the previous part.

Execution:

Execute the Rubiks.exe <full path to file> <full path to goal state file>

<add screenshots>

* Select Rotational Invariance
* Select No for created DB pattern (explained later)
* Select number for depth (recommendation, depths > 6 take longer time to run as an increased number of nodes may be added to the frontier)
* Select a value to inflate/deflate the heuristic.

Implementation:

The implementation is similar, but involving rotational invariance resulted in 24 for potential goal states. Code orients each face as the front face. For each face, we rotate the cube counter clockwise resulting in 24 different goal states. Code arbitrarily picks one tile on the start state cube and figure out 1 corner (see below).

|  |  |  |
| --- | --- | --- |
| **Current State:** | **Potential Goal State:** | **Ideal Goal State:** |
| y o r r p b  y o r r p b  b b  o o  g g  g g  p p  y y | \* o r \* \* \*  \* \* \* \* \* \*  \* \*  \* \*  \* \*  \* \*  \* \*  y \* | o o r r \* \*  o o r r \* \*  \* \*  \* \*  \* \*  \* \*  y y  y y |

After determining the ideal goal state (based on the selected corner, perhaps this can be optimized to select a better corner), the ideal goal state is compared to each of the possible 24 goal states. The one that matches the ideal goal state best is our target goal state. With this, goal state detection doesn’t really change in our implantation. Only our goal state node changes. We tried enabling the goal state to change based on the move. IE: As we made a rotation, we actually could have gotten closer to a different goal state. But as we did more moves, we feel like we started chasing a moving target with the goal state constantly changing.

The repeated state detection changed. Instead of using a List of Nodes, we implemented a Dictionary<int, List<Nodes>>. The int is the number of matches the current node has to goal state. This resulted in having potentially 24 buckets, each containing a list of nodes that has the same number of matches. This will reduce the search time because there are very many permutations with the rubiks cube solver.

Extra:

As a bonus, we tried to create a pattern database which holds each possible state and the number of matching faces. This could be used as a lookup table instead of computing the matches again later. In simulating all the possible states by rotating the cube at different depths (1 move, 2 sequential moves, 3 sequential moves, etc). The expectation is that first state that results in 24 face matches would indicate the max number of moves required to solve the cube in the given start and goal state. Observations, problems with 6+ sequential moves resulted in a long time to create the pattern database, which occurs before the search begins. If a solution is found, the search depth is also known. This is used as a cutoff into the search routine to prevent more nodes to be added onto the frontier.

We also implanted a depth cutoff. Based on the long running searches, we allow the user to set a cutoff. We should play around with these and state some testing we did. Depths > 7+ took a long time, etc.

Heuristic: This didn’t change