Atanas Golev

A* on the Megaminx

How to Run It

g++ -O3 megaminx.cpp
./a.out

Cube Data Structure

I wrote my code in C++. I used a struct that contains a two-dimensional char array to store the cube. These dimensions represent the 12 faces of the megaminx, and the 11 stickers on each cube.

Rotation

I've chosen to randomize only clockwise and solve only counterclockwise.

Randomizer

The functions *rotateClock* and *rotateCounterClock* each have 12 possible different faces to spin. The random number generator `rand()%12` can be used to select a face to rotate at random. To randomize properly, I give *rand()* a new random seed by using *srand(time(NULL))* at the start of *main*.

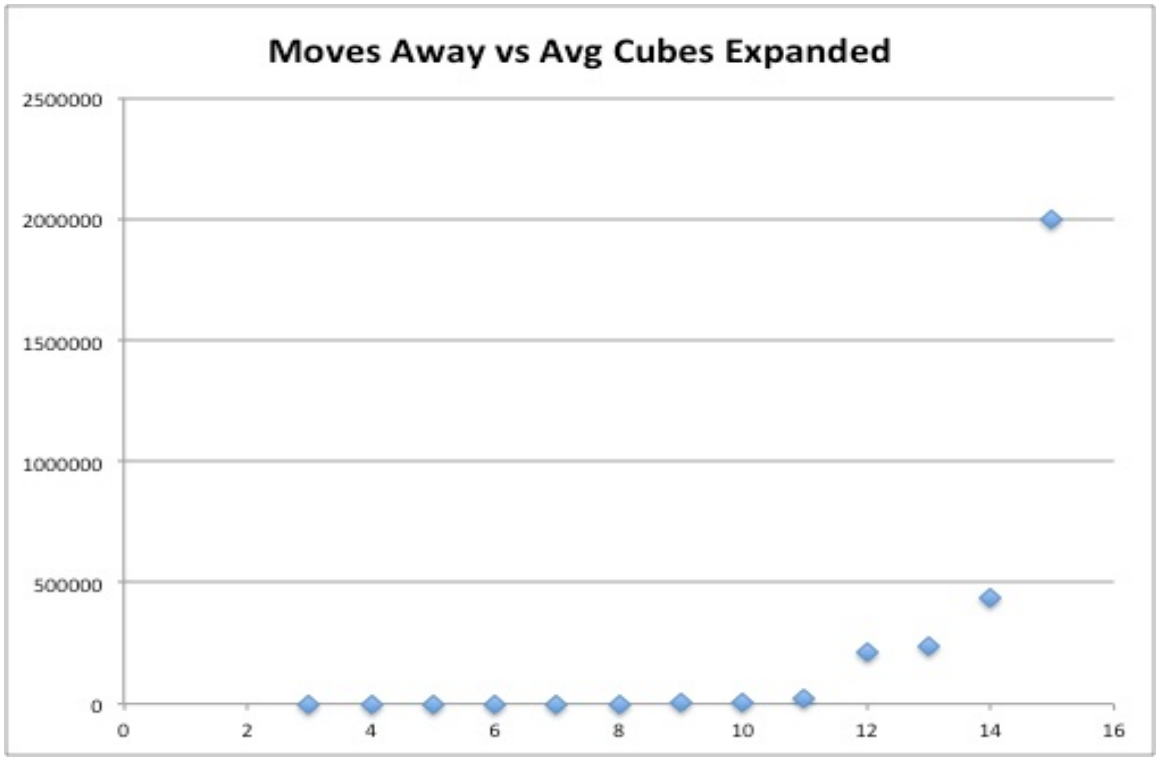
Heuristic

My heuristic was *ceiling((Sum of Manhattan distances of each sticker from original position)/15).* I defined the Manhattan distance as 1 if the sticker was on an adjacent face from the original, 7 if the sticker was on the direct opposite face from the original, and 2 otherwise. The 7 is because that's the minimum number of clockwise turns it would take a sticker to move from one face to its direct opposite face. The 2 is a catchall because I was too lazy to hardcode all the other possibilities: they'd be between 2-6, based upon which face and sticker our sticker was located on. That'd mean having to hardcode a table of 110 different possibilities for each face to get a precise Manhattan distance.

A* Data Structures

My A* data struct contained three items: the unsolved megaminx, the heuristic value *h*, and the distance from the original randomized megaminx *f*. I also overloaded the struct's greater than sign in order to make it priority queue friendly. I chose not to include the parent megaminx into the data structure in order to make it easier on memory. I also hardcoded a solved megaminx and two parallel 2-dimensional int arrays containing the rotation tables as global variables for quicker processing.

Plot of Average Nodes



I hardcoded a limit of a maximum of 2,000,000 nodes expanded into my program to prevent my laptop from exploding. My program worked perfectly for all instances entering an input of less than 15 rotations, but all four times I entered an input of 15 rotations, my laptop crashed before I could even hit this limit. It'd be interesting to analyze why exactly 15, but this seems to be the limit of what my laptop can handle. I have a 2.4 GHz Intel Core i5 Processor with 8 GB of RAM.

Distance	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Avg
3	5	5	5	6	8	5.8
4	9	9	9	16	9	10.4
5	15	24	31	26	12	21.6
6	56	71	52	53	27	51.8
7	100	175	67	302	53	139.4
8	623	192	202	813	273	420.6
9	2901	6100	757	344	496	2119.6
10	12399	672	1503	1973	4204	4150.2
11	6886	6362	15064	766	96274	25070.4
12	826890	67910	30456	95346	23871	208894.6
13	783604	25718	11947	209050	178388	241741.4
14	238609	167494	192128	1191114	392671	436403.2

What I've Learned

I took a top-down approach for the first part of this project, which is different from what I usually do, and it was pretty cool. The modeling part was great. Making A* functional was a little tedious, mostly the figuring out how to efficiently juggle all the data without running out of memory, but in the end, I was satisfied with my working program.

I learned that having a working knowledge of the entire algorithm before you start design is a necessity when taking on a large project. I naively decided to go with a two-dimensional character array that was not stored in a struct for the first part of my project. When I realized that this wouldn't work together with some of the necessary functionality of A*, I had to change around a lot of my previously written code to deal with that. This could've been avoided if I had had a solid understanding of how A* works before I had started on the design.