I consulted the following sources during the creation of this project:

* “8 puzzle problem using A\* search algorithm in English\ Artificial intelligence tutorial English.” - <https://www.youtube.com/watch?v=wJu3IZq1NFs>
* “Solving 8 puzzle with A\* search” - <https://www.youtube.com/watch?v=GuCzYxHa7iA>”
* “8 puzzle problem in ai” - <https://www.youtube.com/watch?v=uJA0i90uCGE&t=378s>
* “Uniform Cost Search” - <https://www.youtube.com/watch?v=dRMvK76xQJI&t=311s>
* “A\* Search” - <https://www.youtube.com/watch?v=6TsL96NAZCo&t=284s>
* “Inheritance in C++” – <https://www.geeksforgeeks.org/inheritance-in-c/>
* “char to int” - <http://www.cplusplus.com/forum/beginner/68260/>
* “How to pass a pointer by reference? C++” - <http://www.cplusplus.com/forum/general/85183/>
* “A\* heuristic calculation with Euclidean Distance “ - <https://stackoverflow.com/questions/16869920/a-heuristic-calculation-with-euclidean-distance>
* “Binary Search Function Template” - <http://www.cplusplus.com/reference/algorithm/binary_search/>

CS170: Assignment 1 Write up

Jorge Marin, SID: 862009642

**Introduction**

This is Project #1 for Professor Niloofar Montazeri for CS 170. The goal of this project was to solve an 8-puzzle problem with the specified algorithm selected by the user. The three algorithms allowed were a Uniform Cost Search, A\* with Misplaced Tile Heuristic, and A\* with Euclidean Distance Heuristic. The attached code was written in C++ and utilized an object-oriented structure to not have any unnecessary repeated code. It is composed of a tree class populated with node pointers. Tree traversal would be the same method regardless of what algorithm the user inputs, the only thing that differs is the calculated h(n) value for each node.

The following are all unique methods that I used to optimize this program that I felt was necessary to share here:

* Created a Node abstract base class that was inherited by 3 derived classes (UniformCostSearch, MisplacedTile, and EuclideanDistance). The only pure virtual function inside the base class was the calcH() function since the only difference between the 3 derived classes is the method in which the h(n) value was calculated. All other shared functions and variables were able to be referenced directly from the base class.
* To easily compare if a puzzle board state has been created, my calcNodeVal() function would create the entire array of numbers (i.e. the puzzle board) into a single numerical integer and would compare these unique values with each other. So, for example, puzzle board [1, 2, 3, 4. 5, 6, 7, 8, 0] would be converted to an int containing 123456780. This would make comparisons faster as I would no longer have to iterate through each item in both arrays and check each individual number.
* The 3 insert functions for the vectors, makes sure the container remains sorted from least to greatest. In regard to the Frontier, the node with the smallest f(n) value is at the first index of the vector (index 0). The main reason for sorting the vectors from least to greatest is because I would then be able to perform a binary search on the vector to make searching for a value much faster and efficient. Binary search’s runtime of O(log n) becomes very essential when the program has to compare and find a value within thousands of nodes.

**Uniform Cost Search**

The first heuristic for tree traversal was Uniform Cost Search. In this case, since the tree was not already constructed with a known path cost, the h(n) was hardcoded to 0. During traversal, the next node to be expanded was determined by the smallest f(n) value which was essentially the g(n) value of the node (how deep it is in the tree). Because of this, Uniform Cost Search is the worst performing algorithm since the heuristic is not that complex. This algorithm would essentially expand all the nodes in every level of the tree until it comes across a successful path.

**Misplaced Tile Heuristic**

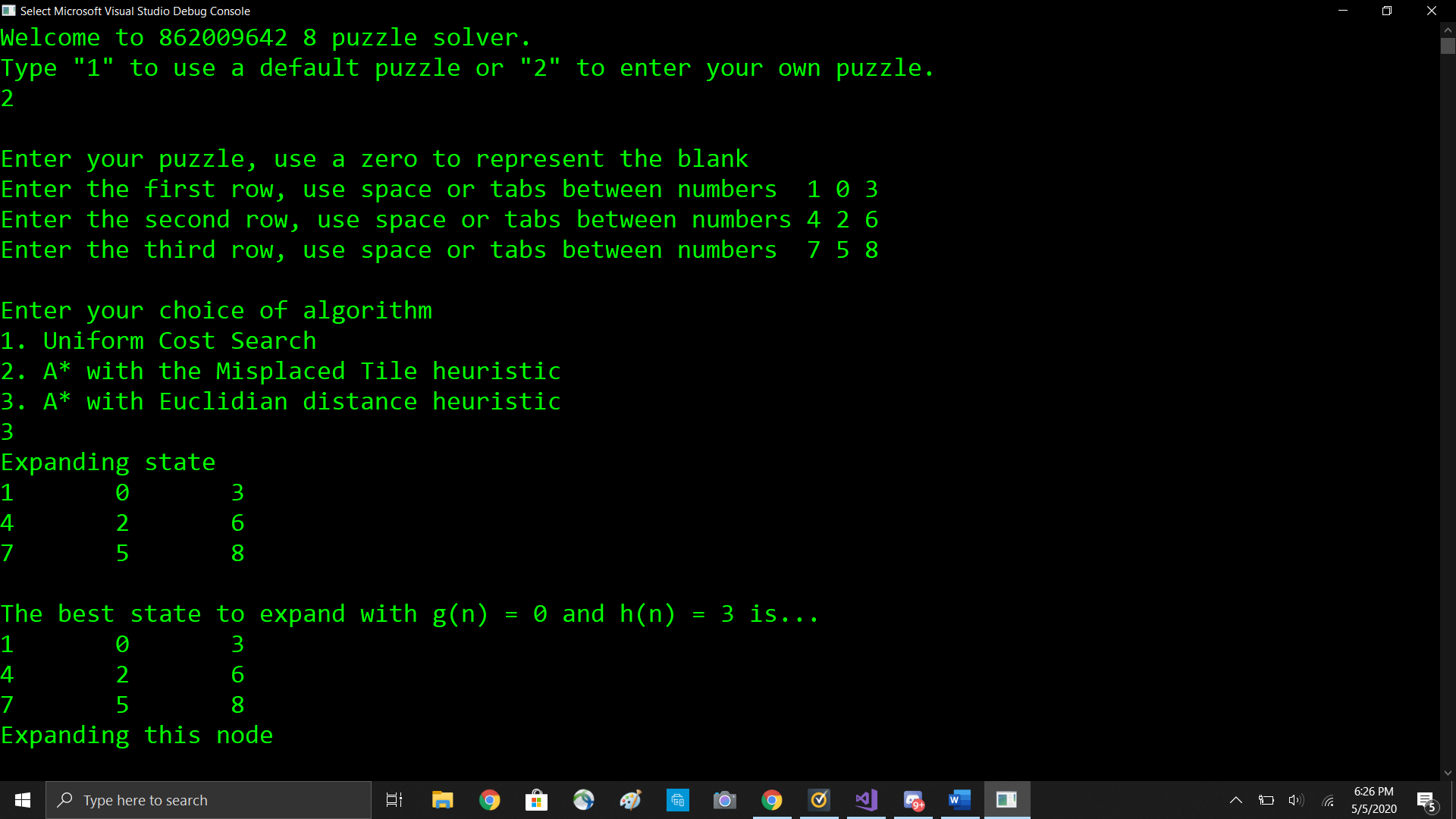
The Misplaced Tile Heuristic is an algorithm that counts the number of moves necessary to move all misplaced tiles into their correct placement (without counting the blank space). This algorithm performed the best in my implementation when I compared the total nodes expanded with the highest frontier size of the other algorithms. This calculation would in turn be an underestimate than the actual moves needed to get each tile into the correct location. Therefore, the calculated f(n) would be less in the Misplaced Tile Heuristic then in the Uniform Cost Search algorithm.

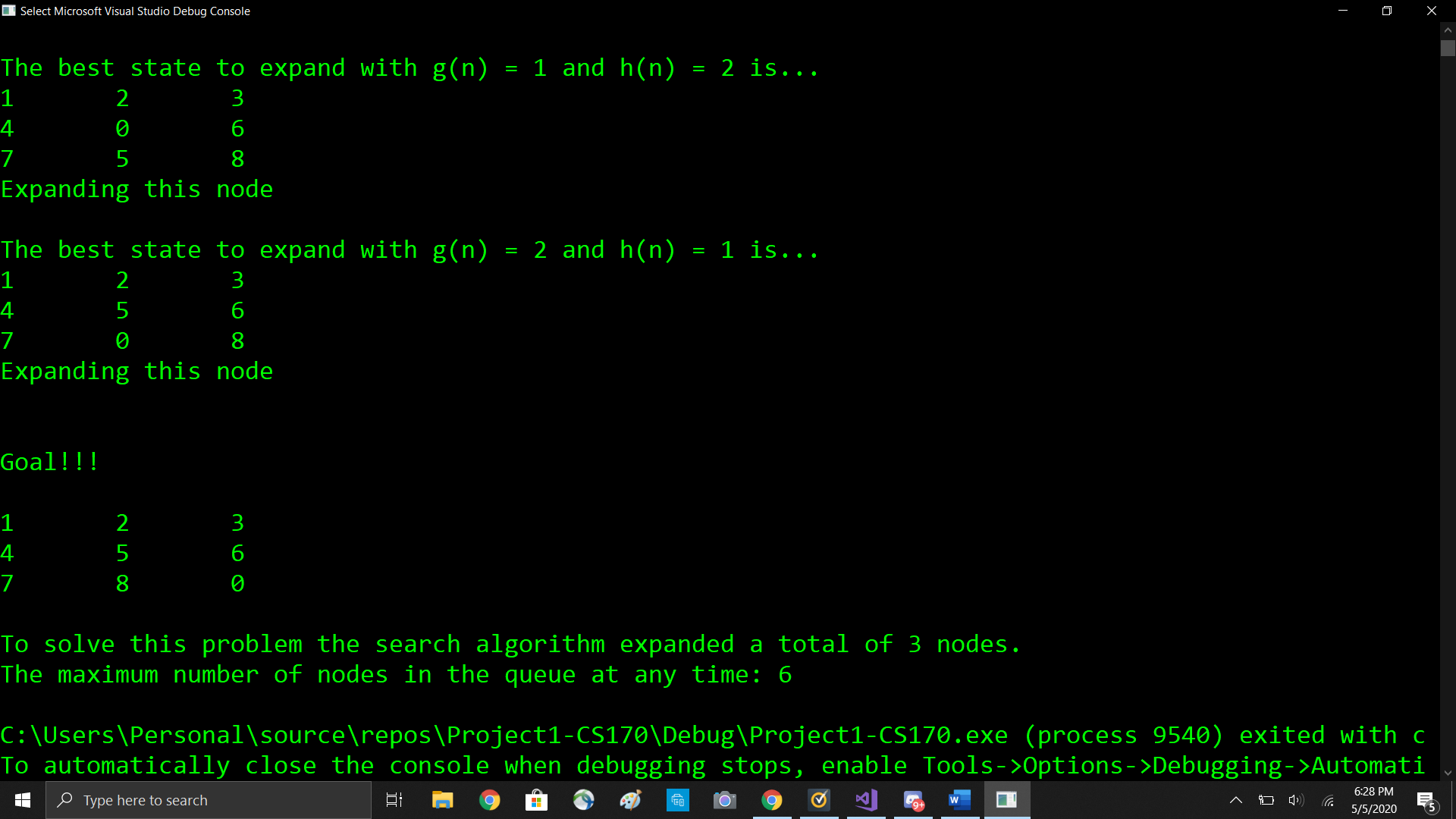
**Euclidean Distance Heuristic**

This h(n) value is calculated by adding up the distance between any misplaced tile and the correct location the number is supposed to be in (again, not counting the blank). This algorithm performed better than the Uniform Cost Search algorithm but not as good as the Misplaced Tile Heuristic. I believe this Euclidean Distance Heuristic is not as optimal because the size of the puzzle board is relatively small. If the puzzle board would have been of size 15 or more, the possible distance between the misplace tile and the actual tile location increases, which in tail would lead to a more accurate Euclidean Distance size. This would then give a more accurate f(n) value.

**TEST CASES:**

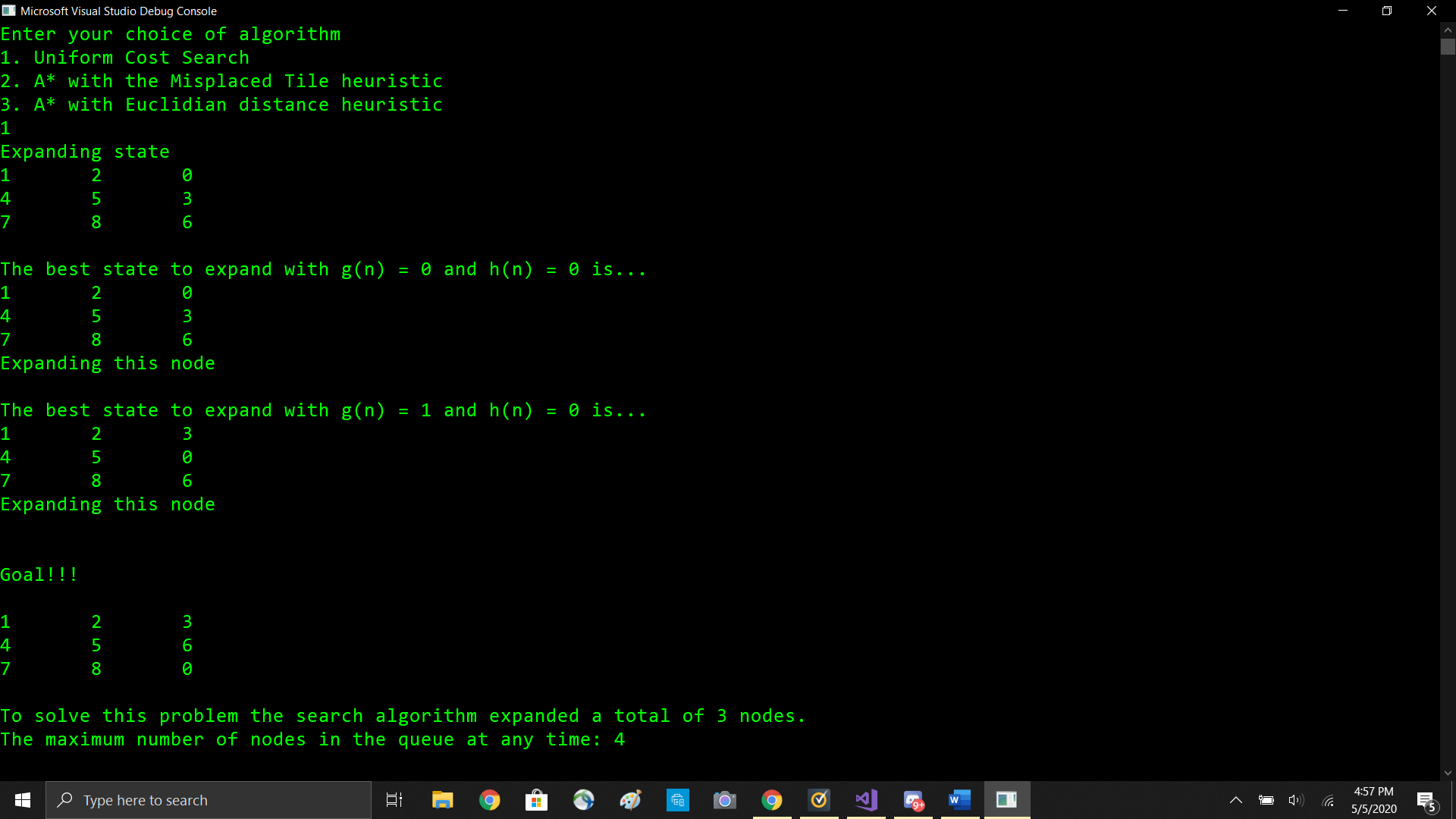
**Trace of Sample Problem with A\*, Euclidean Distance Heuristic**



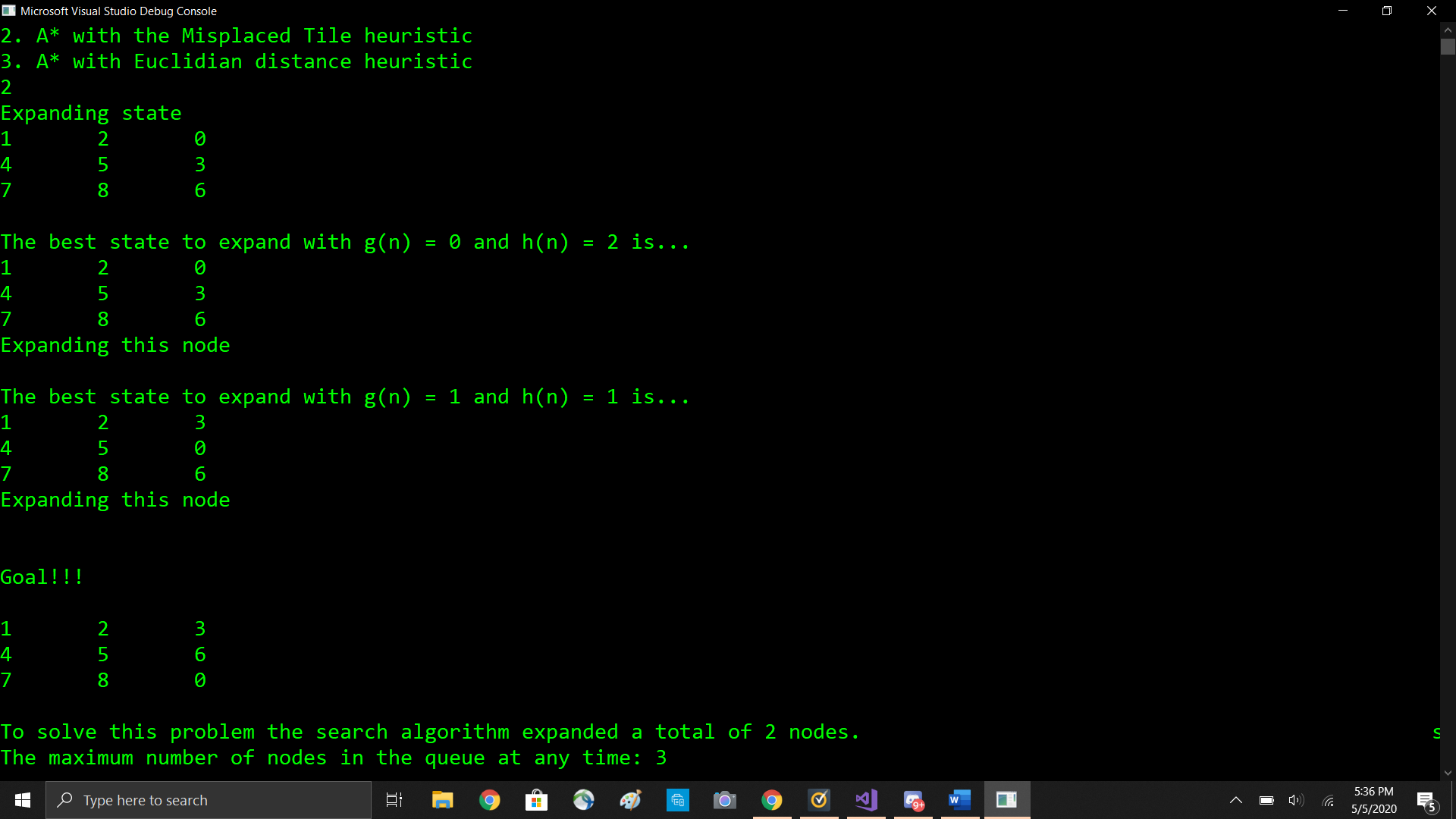


**Easy**

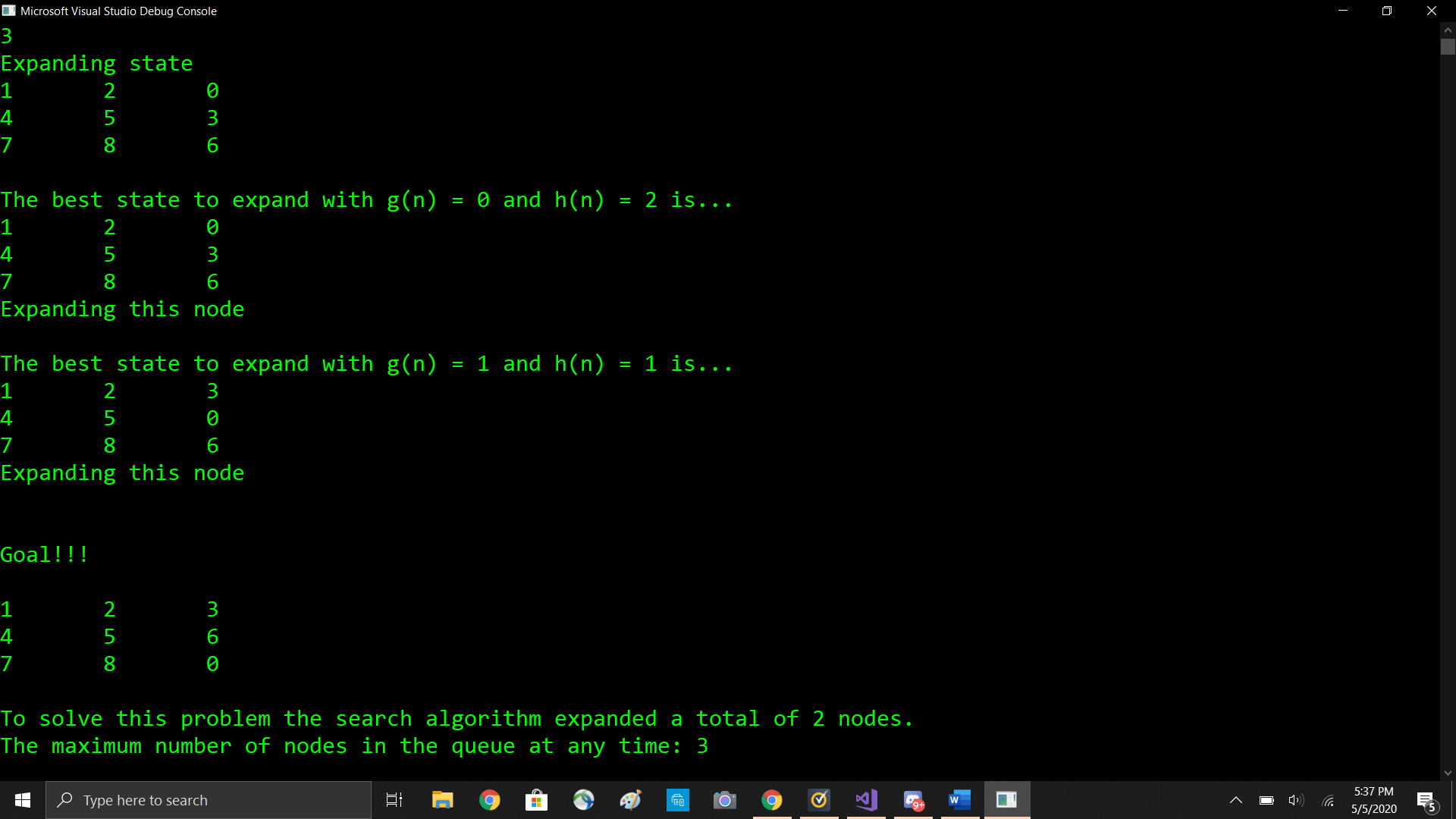
Uniform Cost Search



A\* With the Misplaced Tile Heuristic

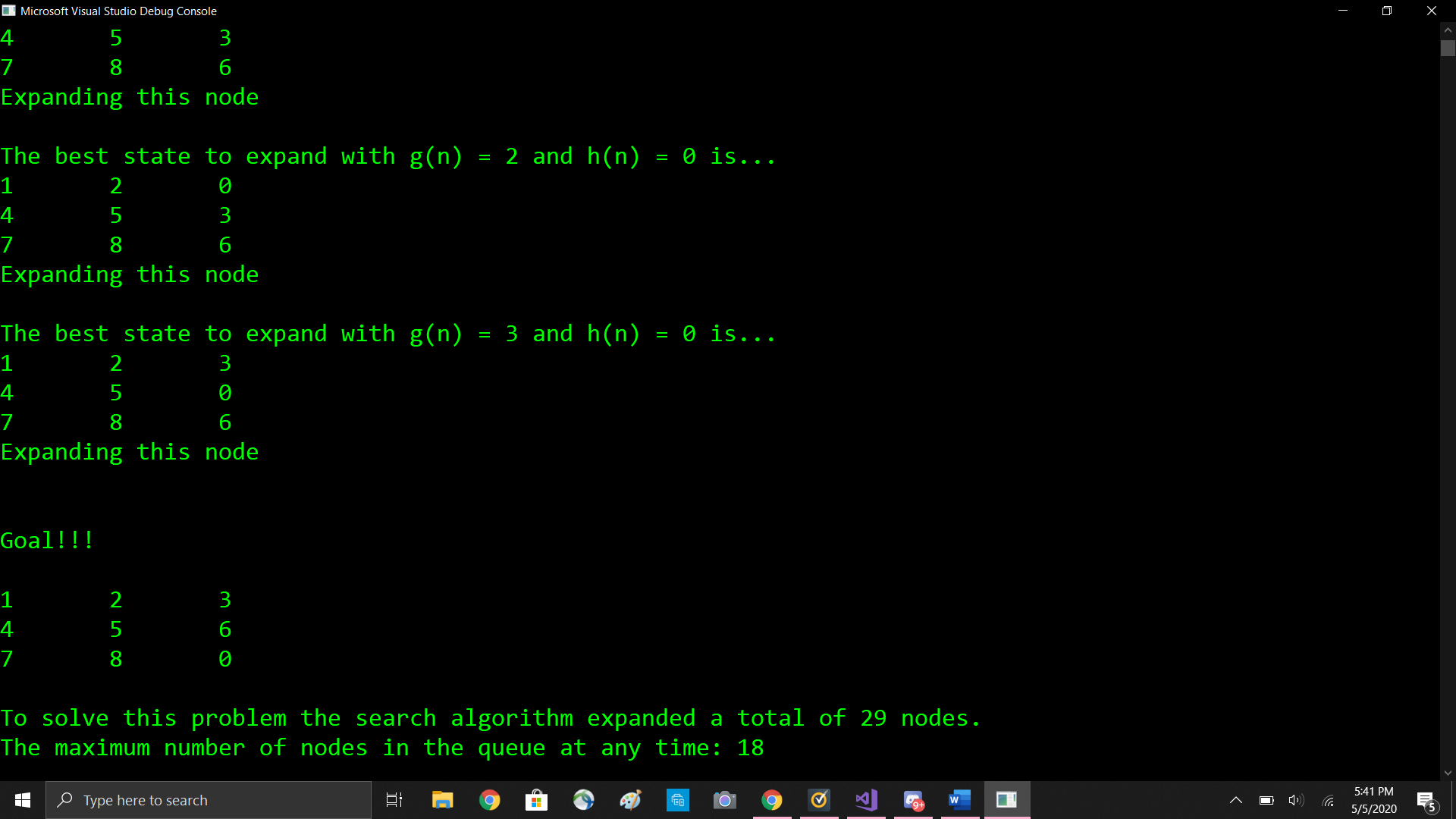


A\* with the Euclidean Distance Heuristic

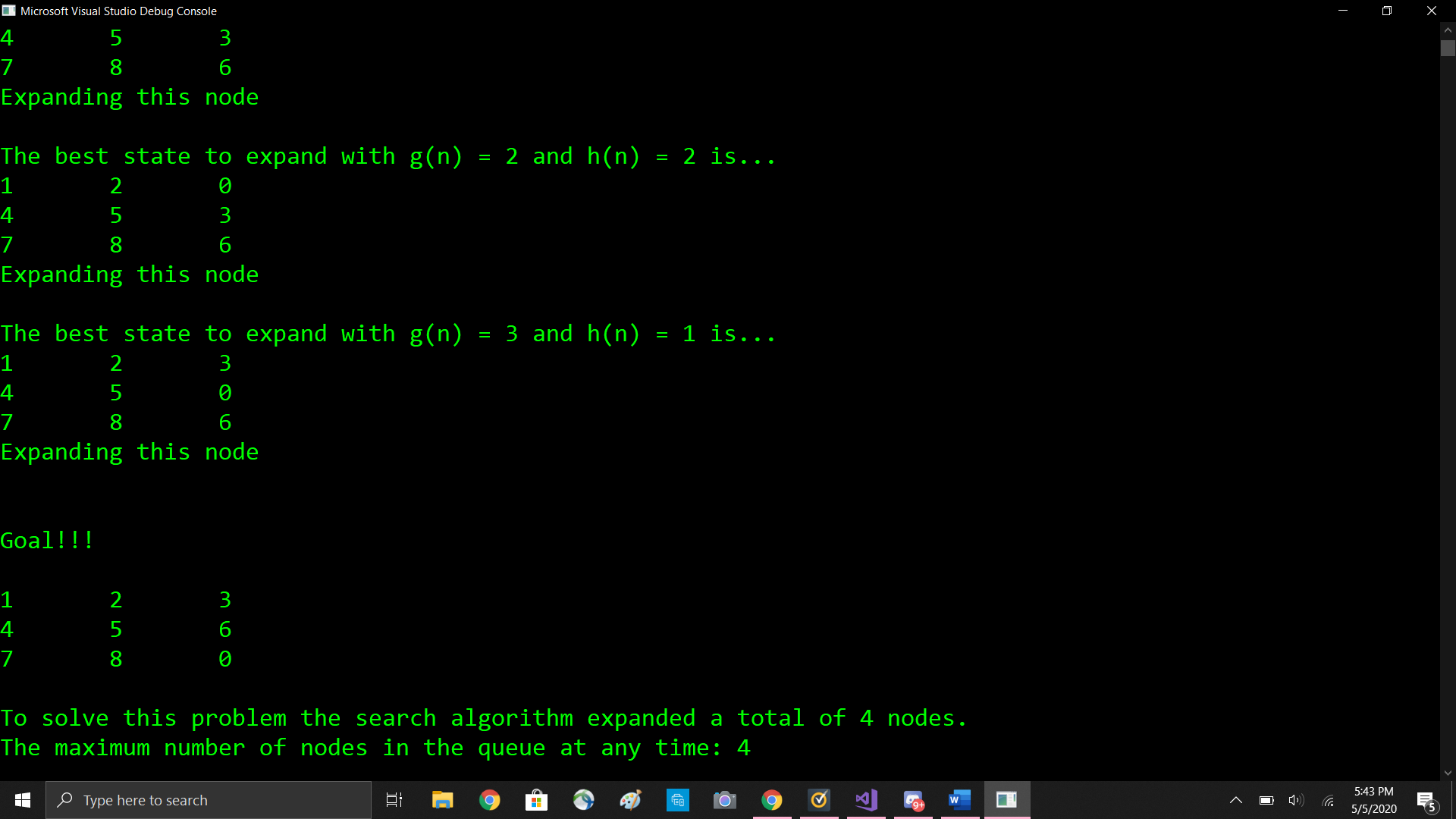


**Doable**

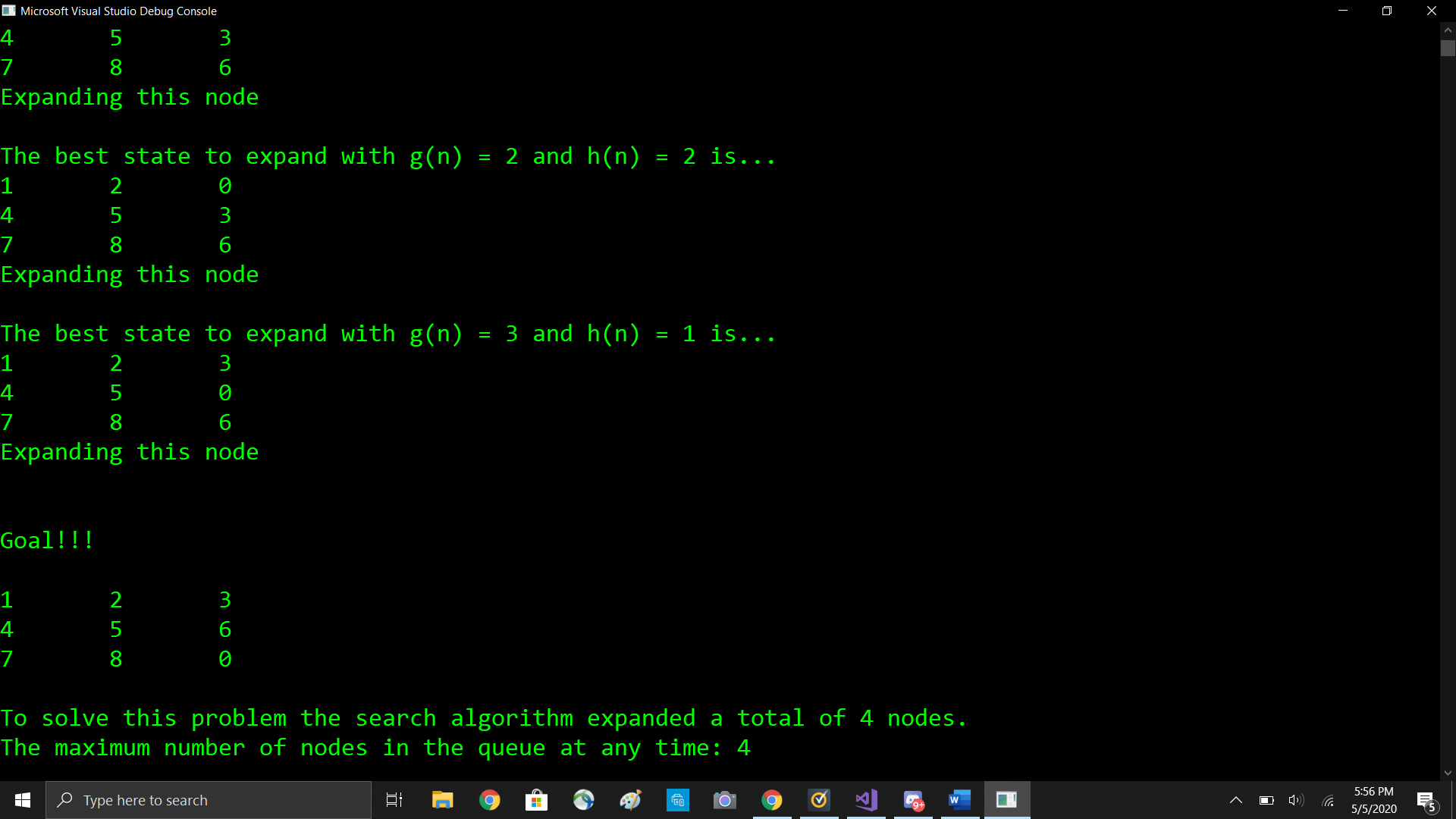
Uniform Cost Search



A\* With the Misplaced Tile Heuristic

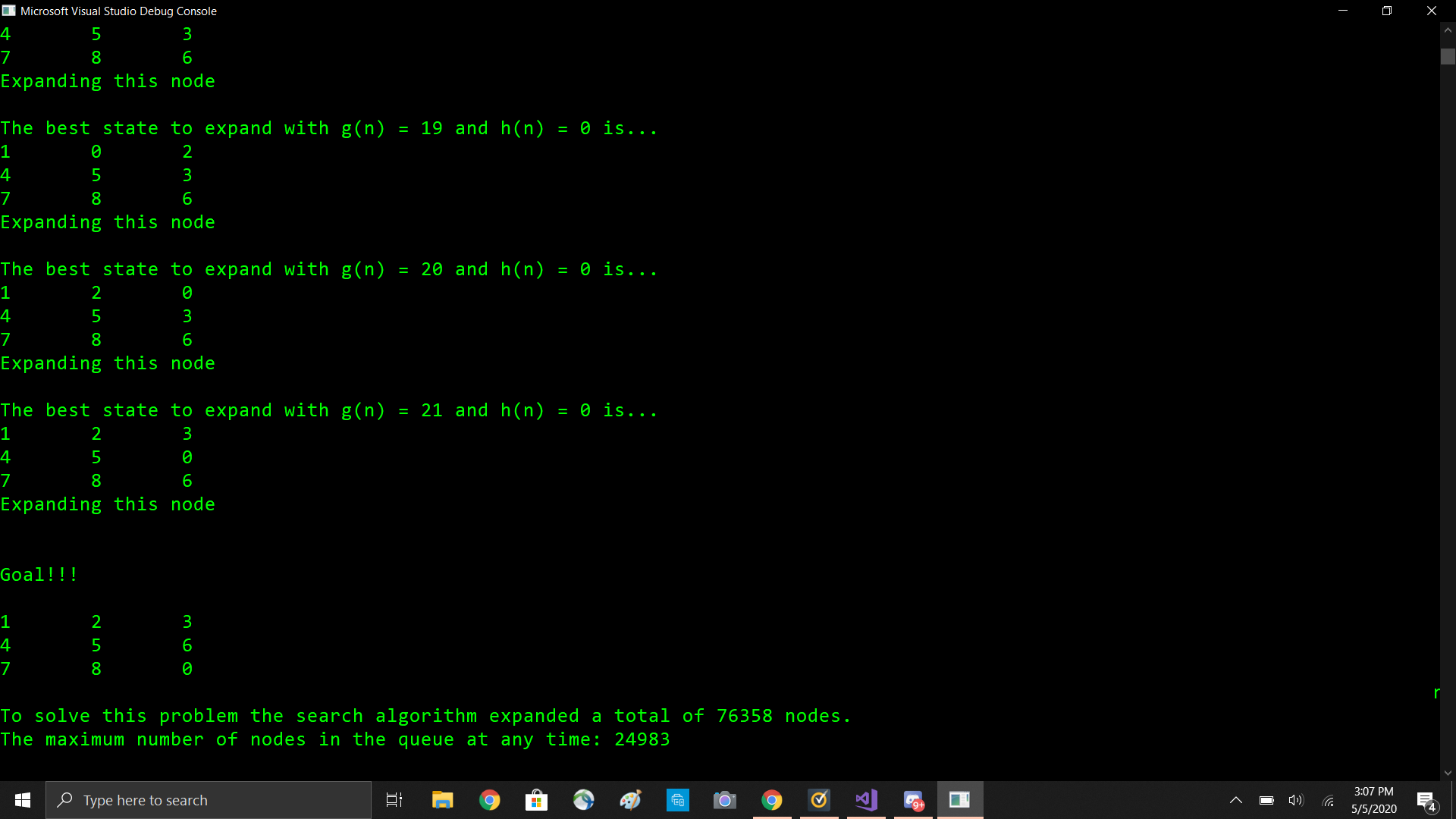


A\* with the Euclidean Distance Heuristic

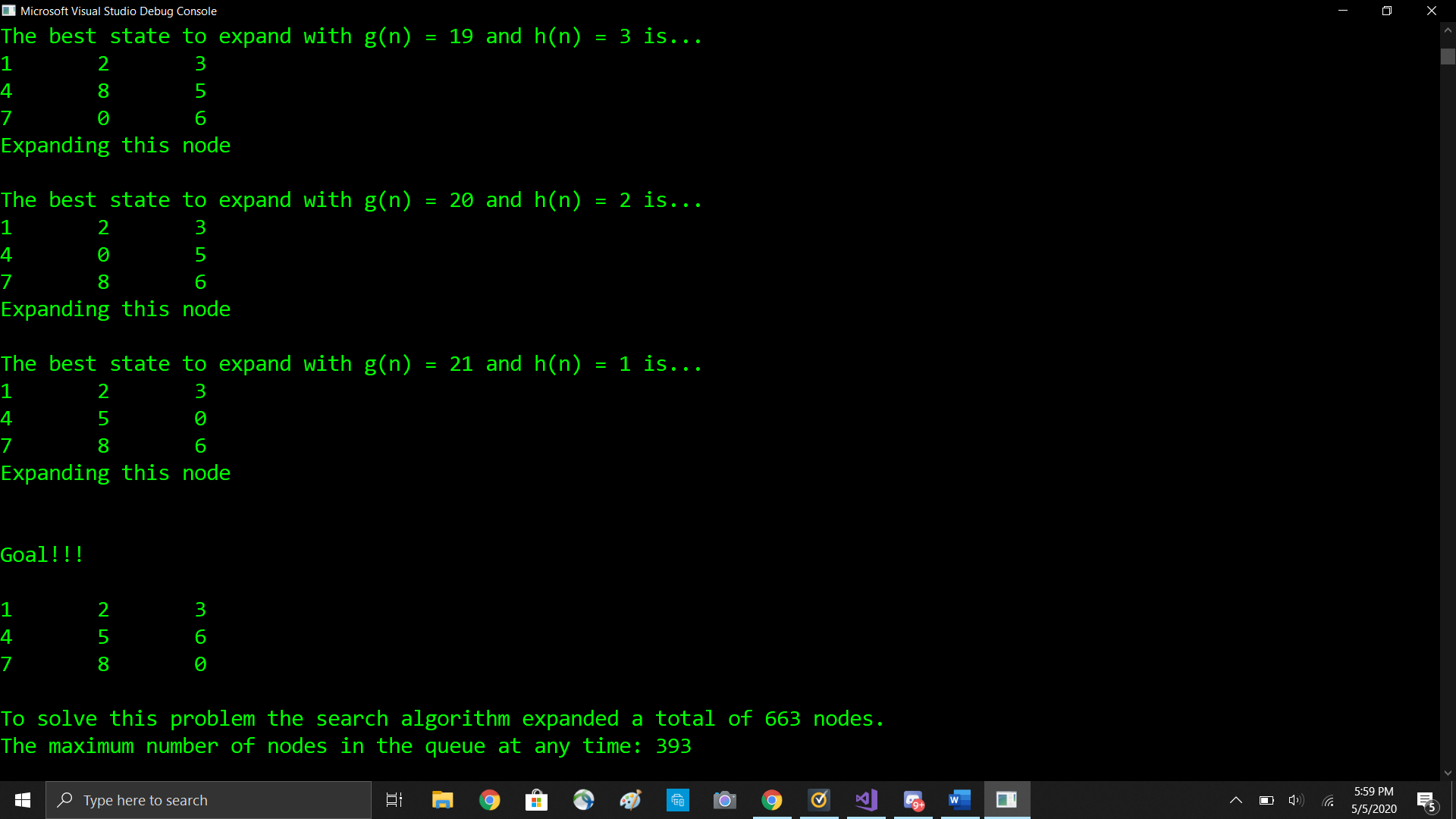


**Oh Boy**

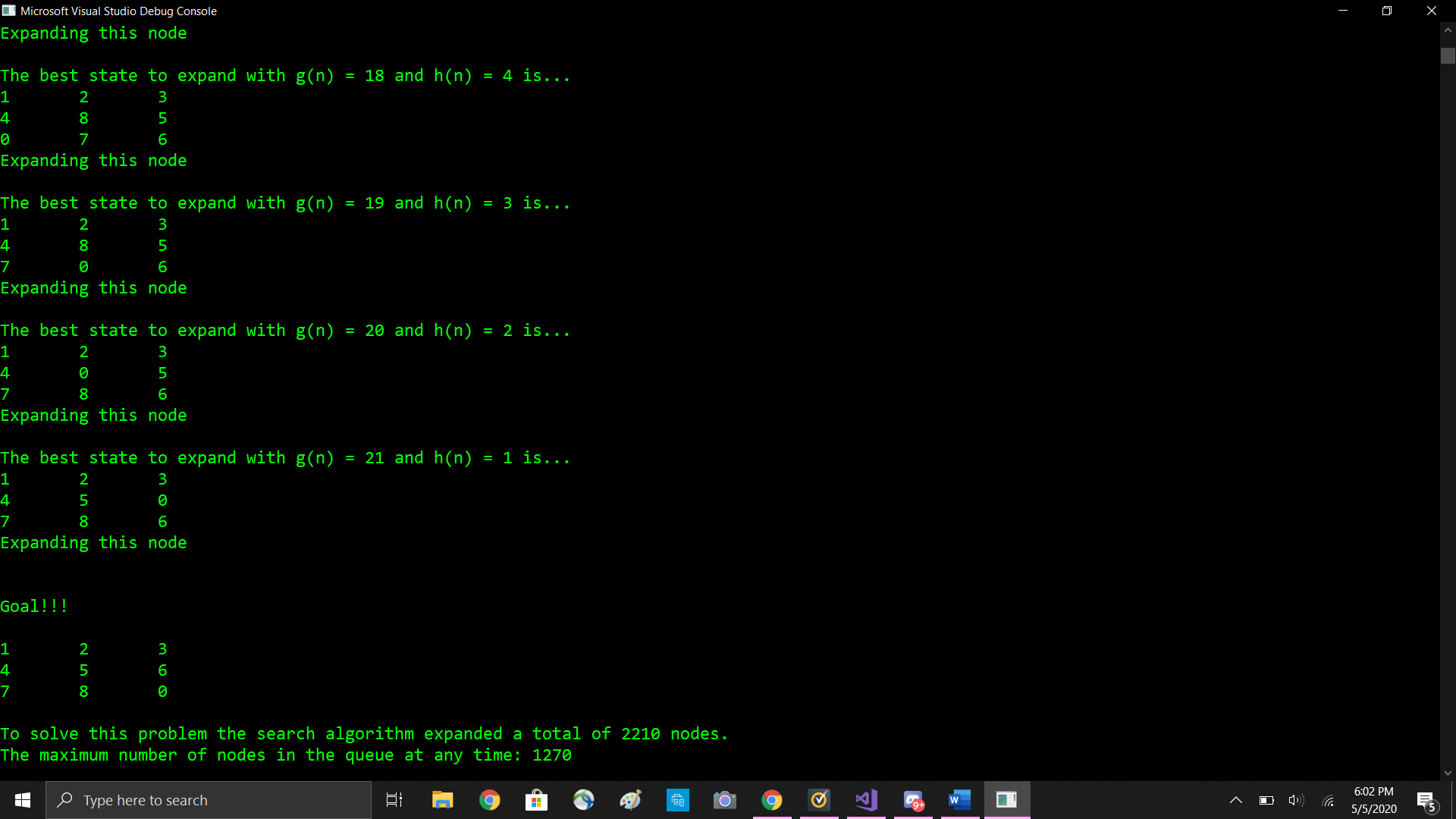
Uniform Cost Search



A\* With the Misplaced Tile Heuristic



A\* with the Euclidean Distance Heuristic



**Impossible**

I ran the impossible puzzle board using my fastest algorithm and allowed the program to run for an hour and 45 minutes with no results. I abandoned this search as it was taking too long to create all the possibilities. When I quit the program, the number of created nodes was up to 137,000 nodes.