Diego Fernandez (df2855)

Normal Magic Squares

Introduction

A Magic Square of order n is defined as an n by n grid of positive integers such that the sum of each column, row, and diagonal all add up to the same number. A Normal Magic Square is defined as a Magic Square that uses each integer from 1 to n^2 to fill the square.

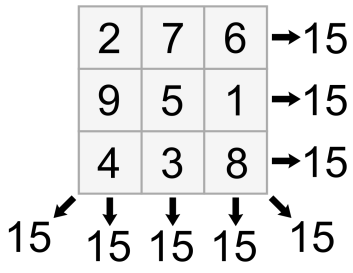

Image 1: A 3 by 3 normal magic square.

The sum of all columns, rows, and diagonals is known as the magic constant. For example, in the case of the image above, the magic constant is 15.

Project Overview

For this project, I will use a brute force method, alongside various constraints, to find the total number of Normal Magic Squares of order n. I will then compare the speed at which this number is calculated with and without parallelization. The total number of Magic Squares is known for orders up to 6. These values are shown below.

Order	Total (including rotations and reflections)	Unique (excluding rotations and reflections)
1	0	0
2	0	0
3	8	1
4	7040	880
5	2,202,441,792	275,305,224
6	17,753,889,197,660,635,632	

Needless to say, the total number increases extremely rapidly (the permutations of possible squares would be the factorial of *n* squared). As such, for this project, I will only focus on squares of orders **3** and **4**.

In addition, I will only be focusing on the total number of squares, counting ones that are rotations and reflections of one another.

Algorithm

I will be using a brute force algorithm. For each index, we will assign it one value from the remaining numbers. If it breaks any magic square rule, discard it. Otherwise, recursively check the next index.

The various algorithms and settings we use will be compared against each other based on the time it takes for the program to finish running.

Efficiency Improvements

We can add several constraints to the problem, which will greatly increase efficiency. The first and most major is that the magic constant is known for all normal magic squares, as it follows a simple formula. The formula is shown below.

$$M = n \cdot \frac{n^2+1}{2}$$

For a square of order 3, this number is 15. For a square of order 4, this number is 34. Thus, for each row/column/diagonal, we only need to check $n - 1$ squares since we will know the last square's value has to be M minus the sum of the rest.

I will also experiment with different structures to represent the problem and see which is most efficient.

Parallelization

I will explore various parallelization techniques to see how we can improve efficiency. The main method I will try is having a maximum recursion depth. I will see which depth is the most efficient. I anticipate that simple parallelization with `par` will be enough, but I will experiment with various other parallelization techniques to see which works best.

Input Data

The only input that will be needed will be the order *n* of the square. This can be provided as an argument and will not be a bottleneck for the algorithm.