Name: Thuan Tran
Date: March 13, 2017
Program 4 Report

Table of Contents

Important to know

To compile the program, type g++ -std=c++14 *.cpp
To run the program, type ./a.out {size of the population} {number of iteration}
For example: ./a.out 50 100 will create a population of 50 puzzles and will have 100 iterations
Enter the text file name with extension: “test.txt” for example (There will be 2 sample text files provided with the program)
There are two text file attached with the program. A “test.txt” text file is a normal sudoku puzzle. A “easy.txt” text file is a very simple sudoku puzzle

What is the problem?
Given a Sudoku puzzle, we need to solve it using genetic algorithm

Purpose of the program
This program will use genetic algorithm to solve a 9x9 Sudoku puzzle in a non-deterministic time depending on the number of iteration and the size of the population.
This program utilized design patterns, Polymorphism, abstract classes to solve the problem

Overview of the program
This program composed from many different classes, each serves its own functionality:

- Puzzle and Sudoku classes: The Puzzle class will be an abstract class for the Sudoku Puzzle. These classes will be used to act as a Puzzle
- Reproduction and SudokuOffspring classes: Reproduction is used to generate new puzzle with “mutation”. SudokuOffspring is a subclass that is used to “mutate” Puzzle
- Fitness and SudokuFitness: Given a Puzzle, we need to determine how good is the solution of that Puzzle. SudokuFitness will calculate the “fitness” of a Sudoku. Lower fitness means better solution
- PuzzleFactory and SudokuFactory: These classes servers as a factory that generate new Puzzle
- Population and SudokuPopulation: These classes serves as a container for the Puzzle, generating new puzzles, etc.

Puzzle and Sudoku Classes
Anatomy of Puzzle class
Puzzle and SudokuPuzzle serves as the classes the hold the representation of a Puzzle (display). These classes are able to take in a Puzzle from a text file and print out its content
The Puzzle base class have two virtual method that let the base class implements because given different Puzzle, there are different mechanism to read a Puzzle and print out

Anatomy of Sudoku class
The Sudoku class holds the Sudoku representation in a 9x9 array using a struct. This struct is composed of two values. The first is the actual value of the puzzle. And the second is a Boolean value to indicate if the value is “fixed” or not, meaning if the value at the location is a predetermined value or not.
The Sudoku class takes in a user input and open a text file in that directory. It will then process the text file and create the Puzzle along the way.
The same thing is with the output of the puzzle, it will traverse a 9x9 array and print out the puzzle in a user friendly way
The Sudoku class also have other three methods that are used to insert a value into the array , check whether the value at a particular location is “fixed” or not and return a copy of the 9x9 array of the Puzzle. These three methods will be used when we create a new “mutated” Puzzle in SudokuOffspring

Reproduction and SudokuOffspring Classes

These two classes are used to reproduce a new Puzzle based on existing ones with some probability that the value will be changed (mutated)

Anatomy of Reproduction Class

This class is used to create a new Puzzle. In this assignment, this is the base class of SudokuOffspring

Anatomy of SudokuOffspring Class

The makeOffSpring method in Reproduction class and SudokuOffspring class takes in a pointer to a puzzle and copy over the element in that Puzzle with some probability that some of the value will change.
There is a probability of 10% for it to mutate. And if the program were to mutate the element at the location, there is a random chance of number from 1-9 will be picked.
This cannot mutate “fixed” value, so if the program were to encounter that, the program will skip over.

Fitness and SudokuFitness Classes

These classes are used to asset if the solution to the puzzle is good enough. The lower fitness means the better. Fitness is equal to 0 means a solution has been found.

Anatomy of Fitness Class

This class is used to asset the fitness of a Puzzle. In this assignment, this is the base class of SudokuFitness

Anatomy of SudokuFitness Class

This class is used to asset the fitness of a Sudoku by traversing its entire 9x9 array. It will travseres each row, each column and each 3x3 block. During the traversal at each row, column or block, it will keep a set to maintain if it have encountered that element before. If it is, then it will increase the fitness.

PuzzleFactory and SudokuFactory

These classed are used to create new puzzle based on existing one. It will utilize a Reproduction object to create newly mutated puzzle

Population and SudokuPopulation

These classes serve as a container to hold Puzzle and can use multiple methods to change the population or queries the information

Anatomy of Population Class

Serve as the base class for SudokuPopulation . The data member of the classes are put as protected so that subclass can used it.

Anatomy of SudokuPopulation Class

There are 4 main methods of the SudokuPopulation class:

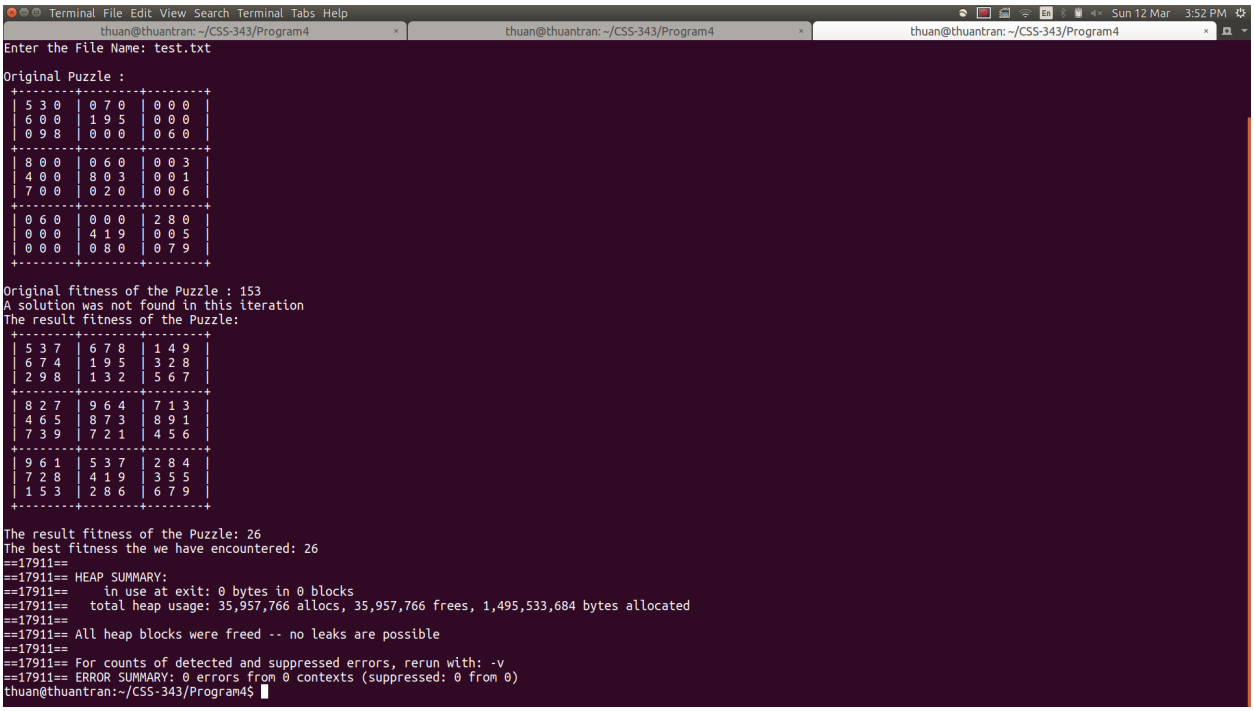
- bestFitness(): This method is used to find the lowest fitness of the member of the population at the current time. It will do this by traversing the population and save the best fitness possible
- bestIndividual(): Same as above, find the Puzzle that has the best fitness at the current population
- newgeneration(): Populate so that the population will reach a certain size. This method will use the SudokuFactory to create all the new Puzzles until it reach the maximum size.
- naturalSelection(): Eliminate 90% least fit member of the population. This method first find the “mark” of the fitness, meaning the index of the fitness so that all the fitness above it will be eliminated. After that, it sorted a container and eliminate any element below that index.

Driver

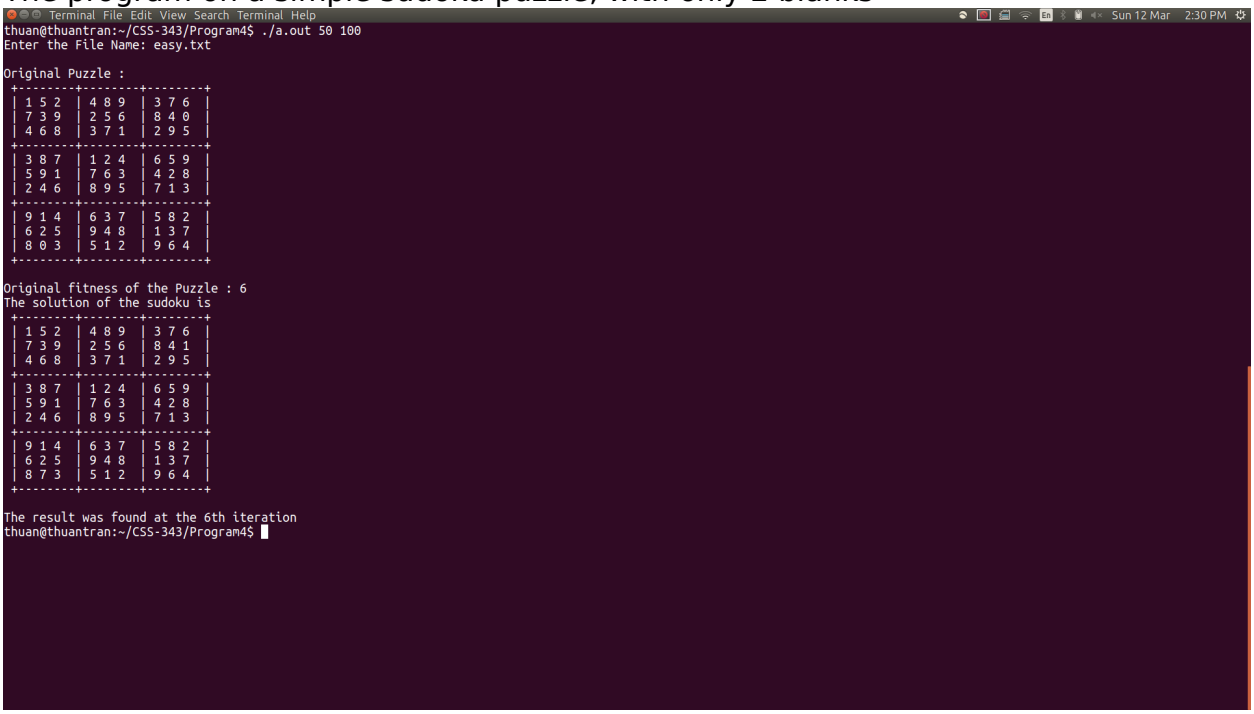
The Driver will be used to demonstrate the program. The driver will create an initial puzzle, a population and traverse through an iteration loop until a solution has been found or it reached the maximum iteration

Result

No memory leak or errors



The program on a simple sudoku puzzle, with only 2 blanks



Assumption

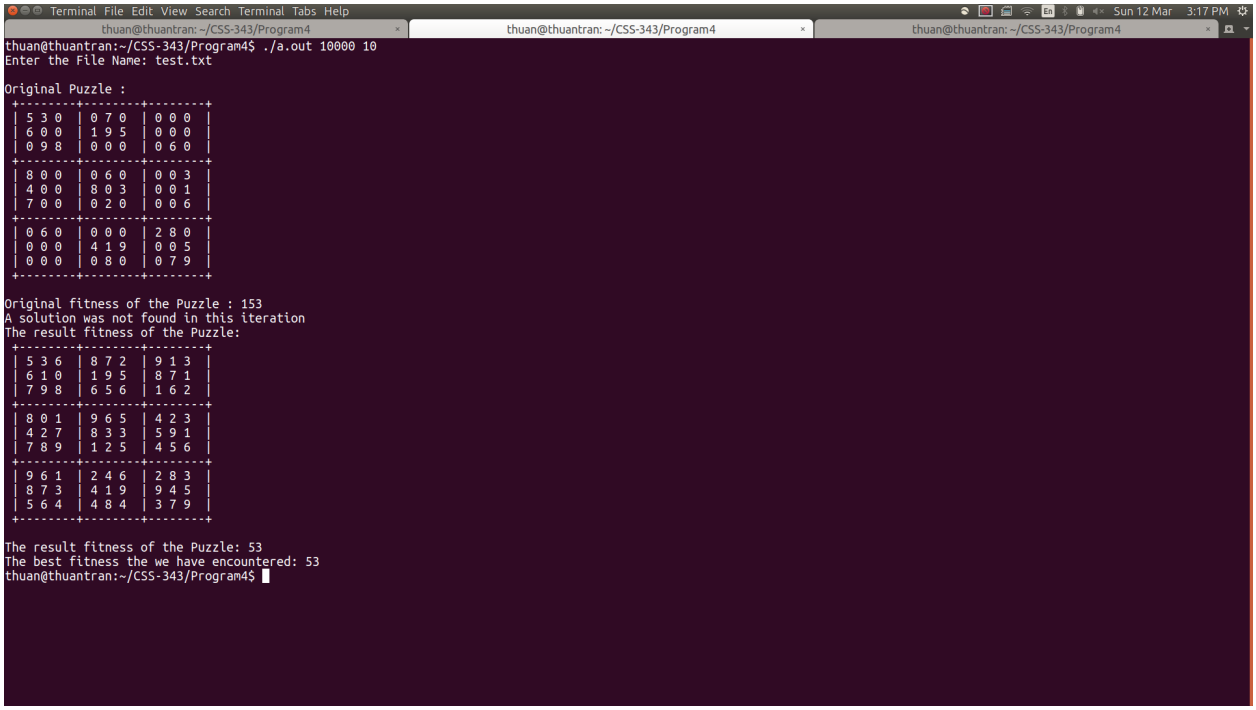
- The program will be compiled using C++14
- The text file is in correct format: 9 rows and 9 elements in each row
- The text file is in the same directory as the program

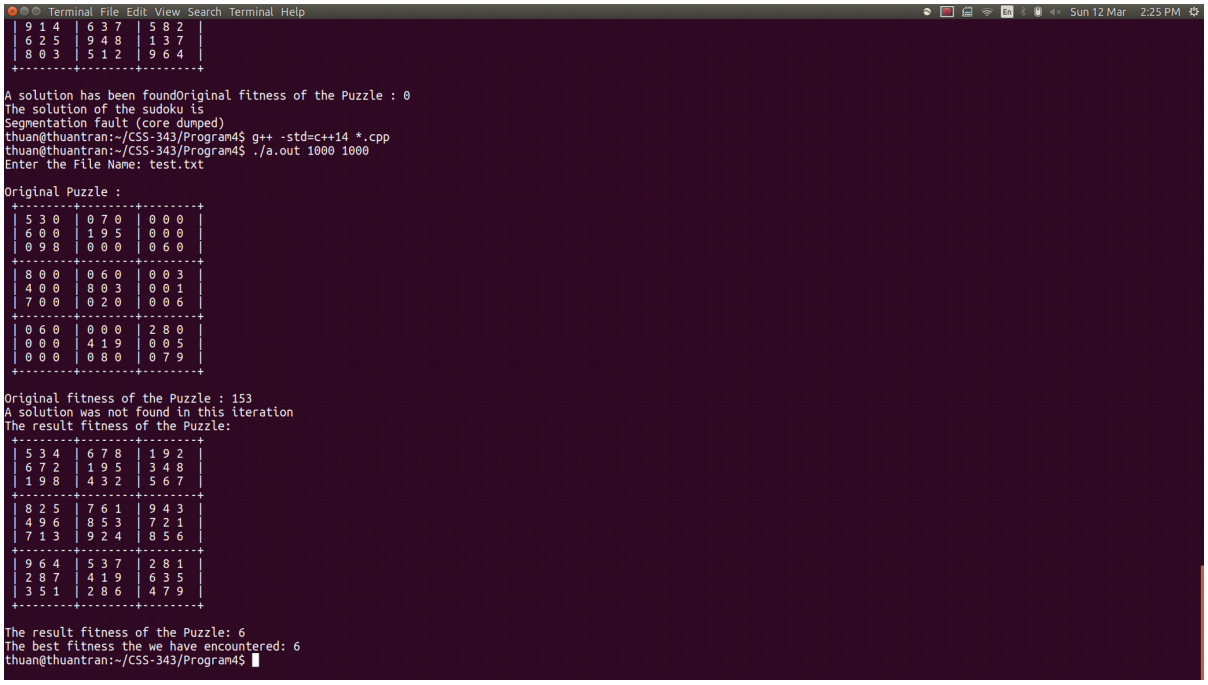
Analysis

Given this problem as a non-deterministic problem, it is difficult to generate an exact estimate how long the program will take. However, after several trials and errors, here are my hypothesis:

- The program depends on the Puzzle. If the puzzle is simple, then the program is able to have a solution
- The program also depends on the size of the population and the number of iteration as well. The more size and iteration, the better. Below is the picture when the population is 10000 but there is only 10 iteration. We can see the the result fitness is 53. However, the other below picture have a population of

1000 and 1000 iteration has the result fitness of 6. So it is better to increase both the population and iteration at the same time rather than to increase them seperately.





- Adding new genetic operations might be helpful. Since in this program, a block might be good enough but in the next iteration, it was mutated to have

a new value. If the program were to combine that good part to another good part of another solution, then it will be quicker to find the solution

Conclusion

The program worked as expected, even though depending on the puzzle, it might take time to compute the result