**NAME**: SUHAS PALANI

**CWID**: **A20548277**

**CS 480 Fall 2023 Programming Assignment #02**

Due: **Monday, November 27, 2023, 11:59 PM CST**

Points: **100**

**Instructions:**

1. Place **all your deliverables (as described below) into a single ZIP** file named:

LastName\_FirstName\_CS480\_Programming02.zip

1. Submit it to Blackboard Assignments section before the due date. **No late submissions will be accepted**.

**Objectives:**

1. (100 points) Implement and evaluate a constraint satisfaction problem algorithm.

**Problem description:**

Sudoku is a combinatorial, logic-based, number-placement puzzle. In classic Sudoku, the objective is to fill a 9 × 9 grid with digits so that each column, each row, and each of the nine 3 × 3 sub-grids that compose the grid contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution (see Figure 1 below). [source: [Sudoku - Wikipedia](https://en.wikipedia.org/wiki/Sudoku)].

|  |  |
| --- | --- |
| 1. unsolved Sudoku puzzle | 1. solved Sudoku puzzle |
| A typical Sudoku puzzle, with nine rows and nine columns that intersect at square spaces. Some of the cells are filled with a number; others are blank cells to be solved. | The previous puzzle, showing its solution. |

*Figure 1: Sudoku puzzle: (a) unsolved, (b) solved* [source: [Sudoku - Wikipedia](https://en.wikipedia.org/wiki/Sudoku)]*.*

Your task is to implement in Python the following constraint satisfaction problem algorithms **(refer to lecture slides and/or your textbook for details and pseudocode)**:

* Brute force (exhaustive) search algorithm,
* Constraint Satisfaction Problem (CSP) back-tracking search,
* CSP with forward-checking and MRV heuristics,

and apply them to solve the puzzle (provided in a CSV file).

Your program should:

* Accept two (2) command line arguments, so your code could be executed with

python cs480\_P02\_AXXXXXXXX.py MODE FILENAME

where:

* + cs480\_P02\_AXXXXXXXX.py is your python code file name,
  + MODE is mode in which your program should operate
    - 1 – brute force search,
    - 2 – Constraint Satisfaction Problem back-tracking search,
    - 3 – CSP with forward-checking and MRV heuristics,
    - 4 – test if the completed puzzle is correct.
  + FILENAME is the input CSV file name (unsolved or solved sudoku puzzle),

Example:

python cs480\_P02\_A11111111.py 2 testcase4.csv

If the number of arguments provided is NOT two (none, one, or more than two) or arguments are invalid (incorrect file, incorrect mode) your program should display the following error message:

ERROR: Not enough/too many/illegal input arguments.

and exit.

* Load and process input data file specified by the FILENAME argument (assume that input data file is ALWAYS in the same folder as your code - this is REQUIRED!).
* Run an algorithm specified by the MODE argument to solve the puzzle (or test if the solution is valid – MODE 4),
* Report results on screen in the following format:

Last Name, First Name, AXXXXXXXX solution:

Input file: FILENAME.CSV

Algorithm: ALGO\_NAME

Input puzzle:

X,6,X,2,X,4,X,5,X

4,7,X,X,6,X,X,8,3

X,X,5,X,7,X,1,X,X

9,X,X,1,X,3,X,X,2

X,1,2,X,X,X,3,4,X

6,X,X,7,X,9,X,X,8

X,X,6,X,8,X,7,X,X

1,4,X,X,9,X,X,2,5

X,8,X,3,X,5,X,9,X

Number of search tree nodes generated: AAAA

Search time: T1 seconds

Solved puzzle:

8,6,1,2,3,4,9,5,7

4,7,9,5,6,1,2,8,3

3,2,5,9,7,8,1,6,4

9,5,8,1,4,3,6,7,2

7,1,2,8,5,6,3,4,9

6,3,4,7,2,9,5,1,8

5,9,6,4,8,2,7,3,1

1,4,3,6,9,7,8,2,5

2,8,7,3,1,5,4,9,6

where:

* + AXXXXXXXX is your IIT A number,
  + FILENAME.CSV input file name,
  + ALGO\_NAME is the algorithm name (TEST for mode 4),
  + AAAA is the number of search tree nodes generated (0 for mode 4),
  + T1 is measured search time in seconds (0 for mode 4),
* Save the solved puzzle to INPUTFILENAME\_SOLUTION.csv file.
* In MODE 4 (test) your program should display the input puzzle along with a message

This is a valid, solved, Sudoku puzzle.

if the solution is correct and

ERROR: This is NOT a solved Sudoku puzzle.

if it is not correct.

**Input data file:**

Your input data file is a single CSV (comma separated values) file containing the Sudoku puzzle grid (see Programming Assignment #02 folder in Blackboard for sample files). The file structure is as follows:

X,6,X,2,X,4,X,5,X

4,7,X,X,6,X,X,8,3

X,X,5,X,7,X,1,X,X

9,X,X,1,X,3,X,X,2

X,1,2,X,X,X,3,4,X

6,X,X,7,X,9,X,X,8

X,X,6,X,8,X,7,X,X

1,4,X,X,9,X,X,2,5

X,8,X,3,X,5,X,9,X

You **CANNOT** modify nor rename input data files. Rows and columns in those files represent individual rows and columns of the puzzle grid as shown on Figure 1. You can assume that file structure is correct without checking it.

CSV file data is either:

* a character X corresponding unassigned (empty) grid cell,
* a positive integer (from the {1, 2, 3, 4, 5, 6, 7, 8, 9} set) corresponding to an assigned grid cell value.

**Deliverables:**

Your submission should include:

* Python code file(s). Your .py file should be named:

cs480\_P02\_AXXXXXXXX.py

where AXXXXXXXX is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.

* this document with your results and conclusions. You should rename it to:

LastName\_FirstName\_CS480\_Programming02.doc or pdf

Use testcase6.csv input data file and run all three algorithms to solve the puzzle. Repeat this search ten (10) times for each algorithm and calculate corresponding averages. Report your findings in the Table A below.

|  |
| --- |
| **Table A** |
| Algorithm | | Number of generated nodes | Average search time in seconds |
| Brute force search | | 241273 | 1.6345 |
| CSP back-tracking | | 6 | 0.0005 |
| CSP with forward-checking and MRV heuristics | | 6 | 0.0002 |

What are your conclusions? Which algorithm performed better? What is the time complexity of each algorithm. Write a summary below

|  |
| --- |
| **Conclusions** |
| 1. **Number of Generated Nodes:**  * **Brute Force Search:** 241,273 nodes * **CSP Backtracking:** 6 nodes * **CSP with Forward-Checking and MRV Heuristics:** 6 nodes  1. **Average Search Time:**  * **Brute Force Search:** 1.6345 seconds * **CSP Backtracking:** 0.0005 seconds * **CSP with Forward-Checking and MRV Heuristics:** 0.0002 seconds  1. CSP Backtracking and CSP with Forward-Checking and MRV Heuristics shown superior efficiency compared to Brute Force Search in terms of the quantity of created nodes. 2. The Brute Force Search method had the greatest average search time, suggesting a greater temporal complexity in comparison to the CSP-based algorithms. 3. The use of MRV (Minimum Remaining Values) and forward-checking techniques in CSP (Constraint Satisfaction Problem) algorithms resulted in a significant decrease in the search space and enhanced efficiency.   **Time Complexity:**   * The brute force search algorithm has an exponential time complexity, namely O(9^(n^2)), where n is the size of the grid. * CSP Backtracking is often more effective than bruteforce, however its efficiency might vary depending on the difficulty of the task. * CSP using Forward-Checking and MRV Heuristics is the most efficient, compared to brute force and backtracking, varies depending on the complexity of the puzzle, although it is frequently superior.   **Final Note:**  Finally, the optimum algorithm is determined by the individual needs and limitations. CSP with heuristics (such as MRV and forward-checking) is a suitable choice for general-purpose Sudoku solution, giving a nice combination of efficiency and generality. |