**SSN College of Engineering**
**Department of Computer Science and Engineering**
**CS1504—Artificial Intelligence**
**Assignment 2: Sudoku Problem**

---

Name: Aviansh Gupta
Reg No: 185001028
Class & Section: CSE - A

---

## Sudoku Rules

1. All the numbers should be in the range 1 to 9.
2. All the numbers in a row must be unique.
3. All the numbers in a column must be unique.
4. All the numbers in the corresponding 3x3 must be unique.

## State Space Formulation

1. **Initial State:** A partially filled 2d matrix representing a Sudoku puzzle(0 means empty cell).
2. **Actions:** Store all possible non conflicting values(according to the rules) in a list and return that. Eg. if for a cell if values 1, 3, 4, 6, 9 are already there in the respective row, column or 3x3 the list of possible values will be [2, 5, 7, 8].
3. **Goal Test:** If all the 81 cells are filled then return True else return False.
   Note: No need to check if the solution is correct or not as all the values are filled only after checking for conflicts.
4. **Final State:** A fully filled 2d matrix representing a solved sudoku.
5. **Path Cost:** None (Our intention is not to find the path but to find the goal state).

## File Description

1. **sudoku.py:** It contains the implementation of the Problem abstract class and the Analysis class for the empirical analysis of the search strategies used.
2. **helper.py:** It contains the following:
   - Problem abstract class
   - Node class for defining states.
   - InstrumentedProblem class for analysis.
   - Breadth First Search Implementation
   - Depth First Search Implementation
   - Depth Limit Search Implementation
   - Iterative Deepening Search Implementation
3. **instances.txt:** It contains a dictionary with various instances of sudoku separated by keys(easy, medium and hard).

## Search Strategies Used

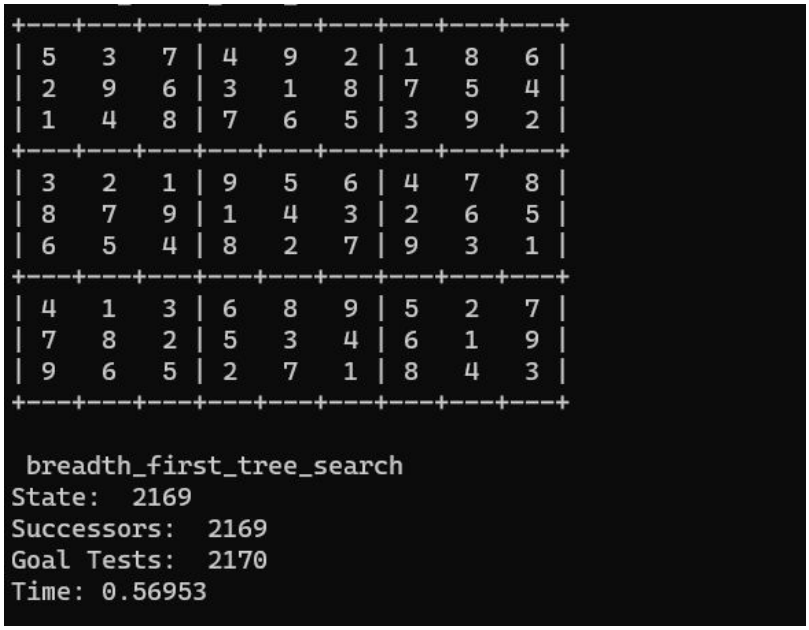
1. Breadth First Search (BFS)
2. Depth First Search (DFS)
3. Depth Limit Search (DLS)
4. Iterative Deepening Search (IDS)

## Analysis

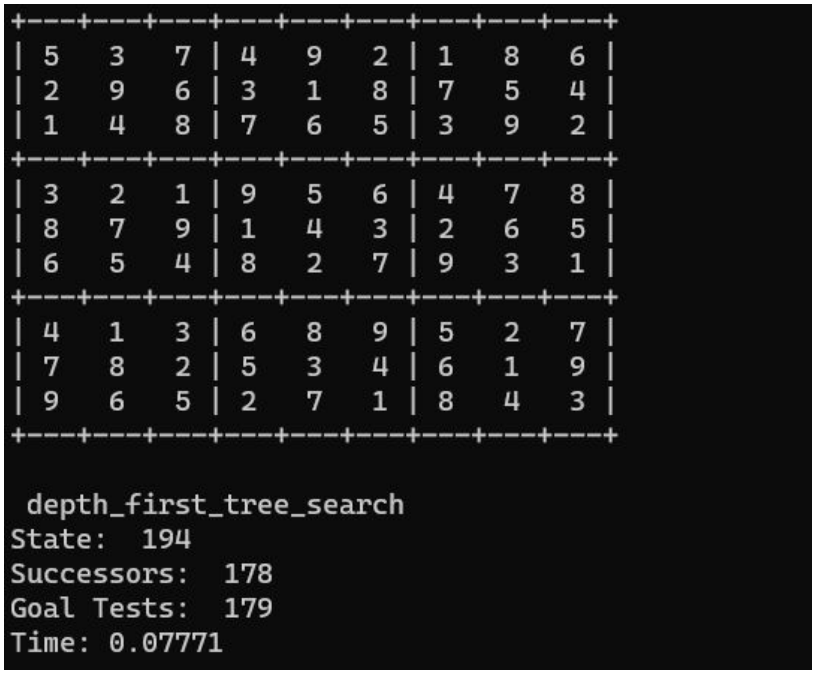
**For single Instance:**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | | | | | | | 8 | |
| | | 6 | 3 | 1 | | 7 | 5 | |
| 1 | | | 7 | | | | 9 | |
| | | | | | 6 | | 7 | |
| | 7 | 9 | | | | 2 | 6 | |
| | 5 | | 8 | | | | | |
| | 1 | | | | 9 | | | 7 |
| | 8 | 2 | | 3 | 4 | 6 | | |
| | 6 | | | | | | | 3 |

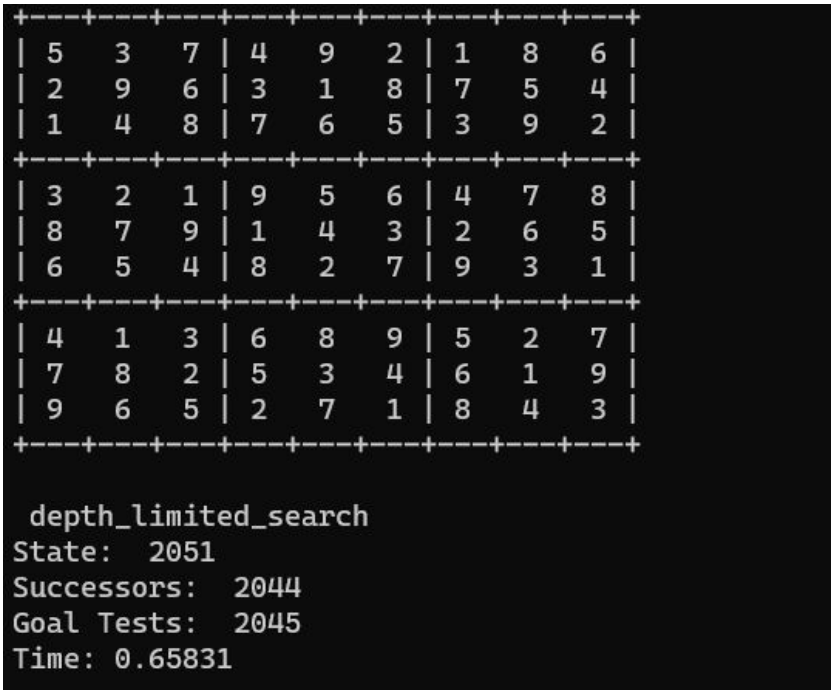
1. **Breadth First Search(bfs)**



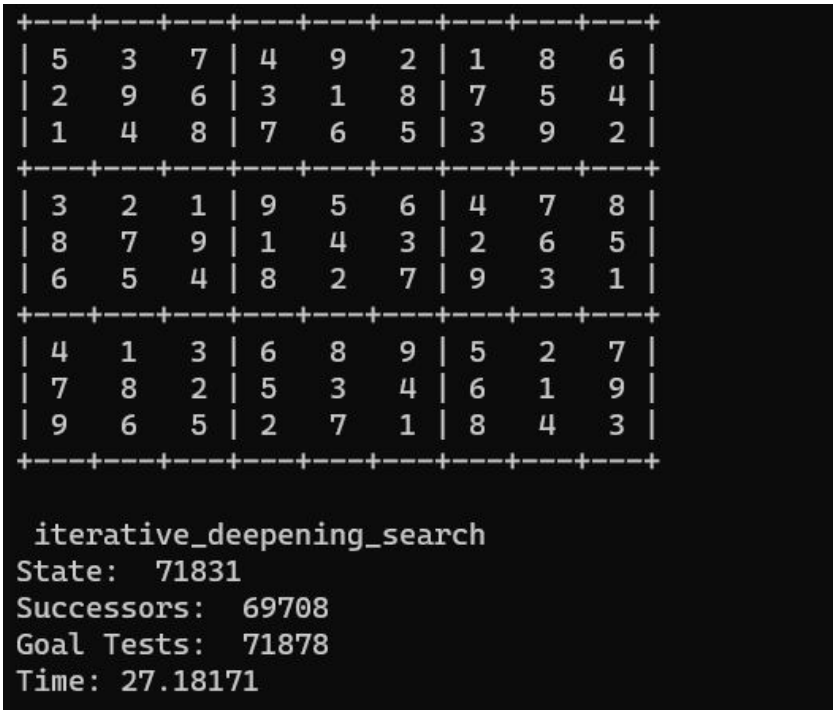

2. **Depth First Search(dfs)**



3. **Depth Limit Search(dls)**




4. **Iterative Deepening Search**



**For Easy Instances**

| Searcher | Avg. Total States | Avg. Successors | Avg. Goal Tests | Avg. Time |
|---|---|---|---|---|
| **BFS** | 563 | 563 | 564 | 0.32195 |
| **DFS** | 321 | 314 | 315 | 0.17002 |
| **DLS (Limit: 81)** | 305 | 297 | 298 | 0.17026 |
| **IDS** | 18880 | 18357 | 18922 | 7.66987 |

**For Medium Instances**

| Searcher | Avg. Total States | Avg. Successors | Avg. Goal Tests | Avg. Time |
|---|---|---|---|---|
| **BFS** | 115486 | 115486 | 115490 | 16.57369 |
| **DFS** | 71859 | 71845 | 71845 | 9.34528 |
| **DLS (Limit: 81)** | 43715 | 43698 | 43701 | 6.45239 |
| **IDS** | 3719482 | 3604031 | 3604020 | 619.726835 |

**For Hard Instances**

| Searcher | Avg. Total States | Avg. Successors | Avg. Goal Tests | Avg. Time |
|---|---|---|---|---|
| **BFS** | 325546 | 325546 | 325546 | 85.44823 |
| **DFS** | 198646 | 198632 | 198632 | 46.374615 |
| **DLS (Limit: 81)** | 126987 | 126970 | 126970 | 30.342595 |
| **IDS** | - | - | - | - |

**Note:** The hard and medium instances were taken from a different dataset than that of the easy ones. IDS was taking very long to complete for the hard puzzles that’s why it's not included.

## Conclusions

- ➔ From the above data it is clear that Iterative deepening search (IDS) takes the longest time as it generates and tests too many states which is because after each iteration it starts the search from the first node.
- ➔ On the other hand Depth first search (DFS) and Depth limited Search (DLS) are almost similar in performance and take the lowest time as the nodes generated are less in both since the implementation of both the algorithms is almost the same with the inclusion of a depth limit in DLS.
- ➔ Breadth First Search takes time in between DFS and IDS. It generates more nodes as it performs level wise search.

## Q) Is it possible to set a reasonable limit a priori for Depth Limit Search?

**Ans:** We can set the limit of DLS to the total number of empty cells in a Sudoku problem as the maximum depth of the state tree will be that only. But for that every time we call the function we need to change the limit.
So, we can set the limit to an upper limit of 81(9x9 sudoku) as the depth never exceeds that and according to the above data it works fine as the above test was done with the limit set to 81.

## Q) What are your ideas to go beyond the basic search strategies to efficiently solve a SuDoKu puzzle?

**Ans:** Since in the actions we are generating the number of possible values that can be placed in a cell so we can prioritize the filling of cells based on the minimum possible values i.e. Instead of choosing the first empty cell we can choose the cell having the minimum number of possible values that can be added to the cell. By choosing the cell with 2 possible values instead of 3 or 4 we can reduce the branching factor and hence solve the problem more efficiently.
This idea can be implemented using a priority queue based on the number of possible values for a cell.