Algorithm and Analysis Assignment 2 Report

(Apoorv Bhardwaj (s3757978))

Killer Sudoku Solver

NAIVE backtracking Killer Sudoku Solver:

The backtracking killer sudoku solver is very much like the backtracking approach that I applied in backtracking standard sudoku solver except for one thing. In the killer solver, I created a new constraint with respect to the 'sum' value of respective cages. Hence, the algorithm checks for cage 'sum' value, unique row, unique column and unique subGrid (box) simultaneously. Though, it solves the given killer sudoku accurately, the execution time is a bit on the higher side.

ADVANCED Killer Sudoku Solver:

To improve the performance of Naive backtracking solver I transformed the killer sudoku cage constraints into a subset sum problem. To obtain subsets with given cardinality and fixed sum equal to the given 'sum' value of a cage, I used dynamic programming.

However, this does not end here. Once, we have the subsets or possible solutions from our dynamic programming approach, we need to further consider the fact that the new subsets formed by applying permutation to these subsets (dynamic programming output) should also be considered possible solutions.

Finally, for each cage I construct **a 'List' of 'Stack'**, where each 'Stack' represents a possible solution.

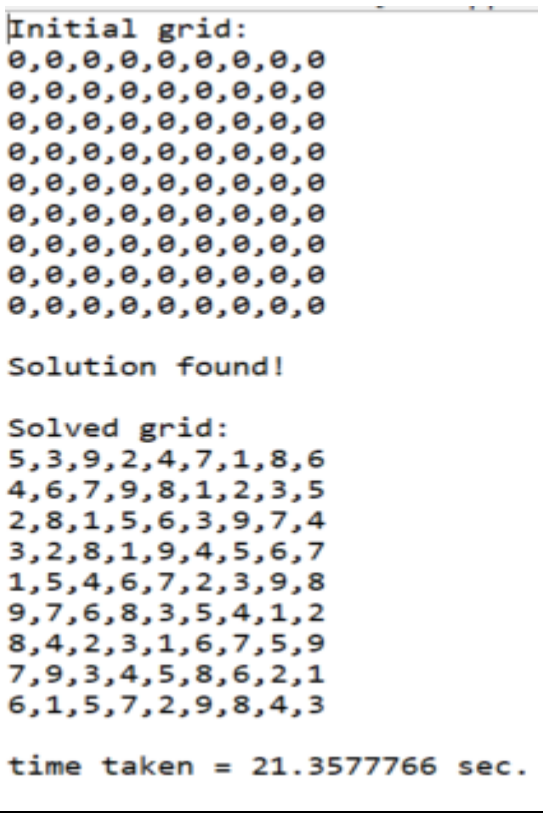
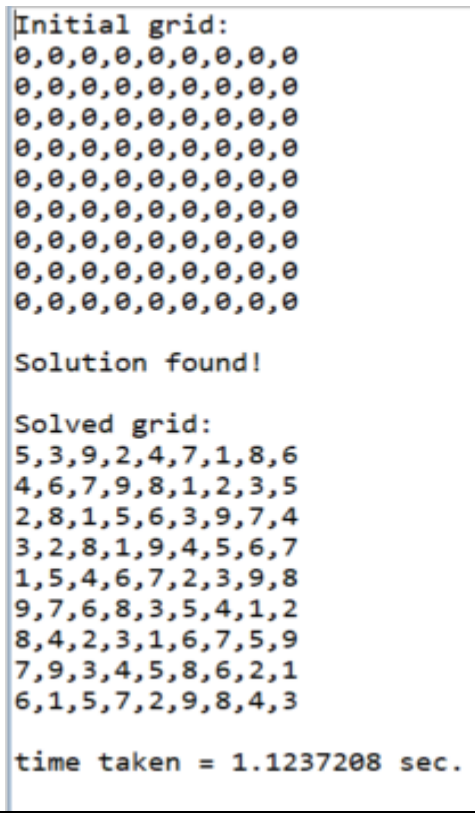
We recursively apply these 'Stack' values until we obtain a valid solution.

For example, for a cage with fixed ''sum'' value of **10,** possible values **(1,9)** and given cardinality **3**. Dynamic programming approach will give us **[5, 3, 2**] and **[6, 1, 4**]. There will be more combinations like [7, 2 ,1], but for the sake of simplicity in this example we will assume that there exists only two set of possible values. After considering the permutation within each set, our final 'List' will look like

```
List => [
        [5 , 3 , 2] , [3 , 2 , 5] , [2 , 3 , 5] , ……  , [6 , 1 , 4] , [1 , 4 , 6] , [4 , 1 , 6] , [4 , 6 , 1] , …
]
```

DIFFERENCES:

	Backtracking Solver	Advance Solver
1.	Operates like a depth first search (DFS), visiting (assigning value to) **one cell** at a	Values are assigned to all the cells belonging to a **cage** simultaneously

	time and backtracking when constraints are violated.	
2.	Possible solution set if formed at an individual **cell level**	Transforms cage constraints into subset sum problem. Uses dynamic programming to identify possible solutions at **cage level**.
3.	Possible solution set is the **same** for every cell.	Possible solution set is **different** for every cage, depending upon the cage size and 'sum' value
4.	Execution time is higher 	Much lower execution times 

PERFORMANCE:

KILLER SUDOKU		
TEST CASE	BACKTRACKING	ADVANCED
easy-killer-44-01.in	0.015 sec.	0.015 sec
easy-killer-99-01.in	22.93 sec	1.12 se (**5**%)
easy-killer-99-02.in	0.15 sec	0.113 sec
easy-killer-99-03.in	23.38 sec	4.23 sec (**18** %)
hard-killer-99-01.in	478.3 sec	56 sec (**11**%)

OBSERVATIONS:

- Execution time data shows that Advanced solver is much efficient in solving killer sudoku puzzle compared to naïve backtracking solver
- Killer solver is about 90% faster than the naïve backtracking solver.