

DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068



**MINI PROJECT REPORT
ON
RAT IN A MAZE USING BACKTRACKING ALGORITHM**

Design and Analysis of Algorithm

**Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING
(Data Science)**

Submitted by

R D Lohith - ENG20DS0032

Rudra Narayan Chetty - ENG20DS0036

Chandan Poonacha - ENG20DS0013

Under the supervision of

Prof. Dr. Shivamma D

DAYANANDA SAGAR UNIVERSITY
SCHOOL OF ENGINEERING
Kudlu Gate, Bangalore - 560068



CERTIFICATE

This is to certify that **Mr. R D Lohith (ENG20DS0032), Mr. Rudra Narayan Chetty (ENG20DS0036) and Mr. Chandan Poonacha (ENG20DS0013)** has satisfactorily completed their Mini Project as prescribed by the University for fifth semester B.tech Programme in Computer Science and Engineering during the year 2021-22 at the School of Engineering, Dayananda Sagar University, Bangalore.

Date:

Signature of faculty in-charge

Max Marks	Marks Obtained

Signature of Chairman

Department of Computer Science and Engineering (Data Science)

DECLARATION

We, **R D Lohith (ENG20DS0032), Rudra Narayan Chetty (ENG20DS0036) and Chandan Poonacha (ENG20DS0013)**, students of the fifth semester B.Tech in Computer Science and Engineering, at School of Engineering, **Dayananda Sagar University**, hereby declare that the mini project titled “**RAT IN A MAZE USING BACKTRACKING ALGORITHM**” has been carried out and submitted in partial fulfillment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2022-2023**.

Student

Signature

R D Lohith
ENG20DS0032

Rudra Narayan Chetty
ENG20DS0036

Chandan Poonacha
ENG20DS0013

Place : Bangalore

Date : 8.12.22

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue our Bachelor's degree in this institution.

We would like to thank **Dr. Uday Kumar Dean, School of Engineering & Technology, Dayananda Sagar University** for his constant encouragement and expert advice.

We would like to thank our guide **Dr. Shivamma, Professor, Dept. of Computer Science and Technology (Data Science), Dayananda Sagar University**, for sparing her valuable time to extend help in every step of our project work, which paved the way for smooth progress and the fruitful culmination of the project.

We are also grateful to our family and friends who provided us with every requirement throughout the course. We would like to thank one and all who directly or indirectly helped us in the Project work.

INDEX

	PAGE NO
1 Introduction	1
1.1 About the problem	1
1.2 About the DAA technique	1
2 Problem Statement	2
3 S/W and H/W Requirements	3
4 Design	4
4.1 Flowchart	5
5 Code (Implementation)	6-8
6 Testing	9
7 Output	10
8 Conclusion	11
9 References	12

ABSTRACT

The Rat in a Maze program determines the way from the start to the end of the maze. A rat starts from source and has to reach the destination. The rat can move only in two directions: forward and down. In the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination.

INTRODUCTION

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time (by time, here, is referred to the time elapsed till reaching any level of the search tree). Backtracking can also be said as an improvement to the brute force approach. So basically, the idea behind the backtracking technique is that it searches for a solution to a problem among all the available options. Initially, we start the backtracking from one possible option and if the problem is solved with that selected option then we return the solution else we backtrack and select another option from the remaining available options. There also might be a case where none of the options will give you the solution and hence we understand that backtracking won't give any solution to that particular problem. We can also say that backtracking is a form of recursion. We can conclude that backtracking at every step eliminates those choices that cannot give us the solution and proceeds to those choices that have the potential of taking us to the solution.

1.1 About the Problem

There are three types of problems in backtracking –

- 1) Decision Problem – In this, we search for a feasible solution.
- 2) Optimization Problem – In this, we search for the best solution.
- 3) Enumeration Problem – In this, we find all feasible solutions.

1.1 About the Technique

Generally, every constraint satisfaction problem which has clear and well-defined constraints on any objective solution, that incrementally builds candidate to the solution and abandons a candidate (“backtracks”) as soon as it determines that the candidate cannot possibly be completed to a valid solution, can be solved by Backtracking. However, most of the problems that are discussed, can be solved using other known algorithms like Dynamic Programming or Greedy Algorithms in logarithmic, linear, linear-logarithmic time complexity in order of input size, and therefore, outshine the backtracking algorithm in every respect.

PROBLEM STATEMENT

Given a $N \times N$ board with the Knight placed on the first block of an empty board. Moving according to the rules of chess knight must visit each square exactly once. Print the order of each cell in which they are visited.

SOLUTION

By using the Naive Algorithm to generate all tours one by one and check if the generated tour satisfies the constraints. Backtracking works in an incremental way to attack problems. Typically, we start from an empty solution vector and one by one add items (Meaning of item varies from problem to problem. In the context of Knight's tour problem, an item is a Knight's move). When we add an item, we check if adding the current item violates the problem constraint, if it does then we remove the item and try other alternatives. If none of the alternatives works out then we go to the previous stage and remove the item added in the previous stage. If we reach the initial stage back then we say that no solution exists. If adding an item doesn't violate constraints then we recursively add items one by one. If the solution vector becomes complete then we print the solution.

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS

PROCESSOR: INTEL CORE I3

RAM: 4 GB

ROM: 256 GB

SOFTWARE REQUIREMENTS

FRONT END : JAVA

BACK END : JAVA

ADDITIONAL TOOLS : BLUE J

DESIGN

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally. Solving one piece at a time, and removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree) is the process of backtracking.

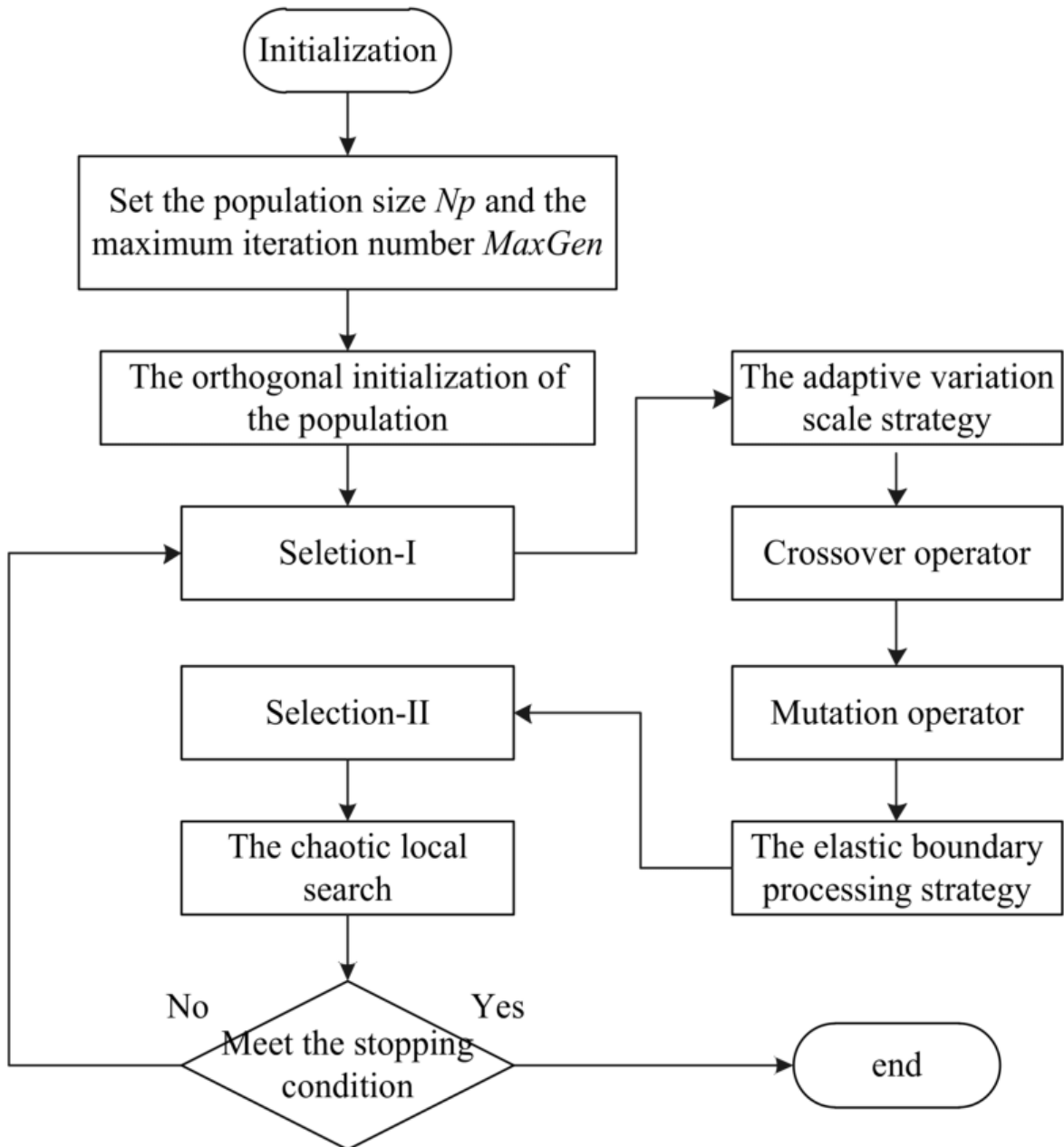
Approach:

Form a recursive function, which will follow a path and check if the path reaches the destination or not. If the path does not reach the destination then backtrack and try other paths.

Algorithm:

- 1) Create a solution matrix, initially filled with 0's.
- 2) Create a recursive function, which takes initial matrix, output matrix and position of rat (i, j).
- 3) if the position is out of the matrix or the position is not valid then return.
- 4) Mark the position `output[i][j]` as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.
- 5) Recursively call for position (i+1, j) and (i, j+1).
- 6) Unmark position (i, j), i.e `output[i][j] = 0`.

Flow Chart



IMPLIMENTATION

```
public class RatMazeProblem
{
    static int R;
    static int C;
    void displaySolution(int result[][])
    {
        System.out.println("The resultant matrix is: ");
        for (int r = 0; r < R; r++)
        {
            for (int c = 0; c < C; c++)
            {
                System.out.print(" " + result[r][c] + " ");
            }
            System.out.println();
        }
    }
    boolean isValid(int maze[], int r, int c)
    {
        return (r >= 0 && r < R && c >= 0 && c < C && maze[r][c] == 0);
    }
    boolean findPathMaze(int maze[][]) {
        int maz[][] = new int[R][C];
        for(int r = 0; r < R; r++)
        {
            for(int c = 0; c < C; c++)
            {
                maz[r][c] = 1;
            }
        }
        if (solveMazeUtility(maze, 0, 0, maz) == false)
```

```

    {
        System.out.print("Path from source to destination doesn't exist");
        return false;
    }

    displaySolution(maz);
    return true;
}

boolean solveMazeUtility(int maze[][], int r, int c, int res[][])
{
    if (r == R - 1 && c == C - 1 && maze[r][c] == 0)
    {
        res[r][c] = 0;
        return true;
    }
    if (isValid(maze, r, c) == true)
    {
        if (res[r][c] == 0)
        {
            return false;
        }
        res[r][c] = 0;
        if (solveMazeUtility(maze, r + 1, c, res))
        {
            return true;
        }
        if (solveMazeUtility(maze, r, c + 1, res))
        {
            return true;
        }
        if (solveMazeUtility(maze, r - 1, c, res))
        {
            return true;
        }
    }
}

```

```

        if (solveMazeUtility(maze, r, c - 1, res))
        {
            return true;
        }
        res[r][c] = 1;
        return false;
    }
    return false;
}

public static void main(String argsv[])
{
    RatMazeProblem r = new RatMazeProblem();
    int maze[][] = { { 0, 1, 1, 1 },
                     { 0, 0, 0, 1 },
                     { 0, 0, 0, 1 },
                     { 0, 0, 0, 1 } };

    R = maze.length;
    C = maze[0].length;
    r.findPathMaze(maze);
}
}

```

TESTING

TRIAL 1:

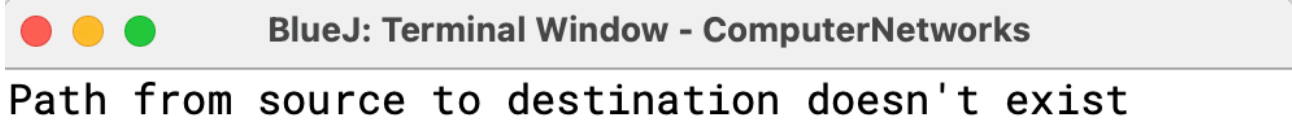
```
public static void main(String argsv[])
{
    RatMazeProblem r = new RatMazeProblem();
    int maze[][] = { { 0, 1, 1, 1 },
                     { 0, 0, 0, 1 },
                     { 0, 0, 0, 1 },
                     { 0, 0, 0, 1 } };
    R = maze.length;
    C = maze[0].length;
    r.findPathMaze(maze);
}
```

TRIAL 2:

```
public static void main(String argsv[])
{
    RatMazeProblem r = new RatMazeProblem();
    int maze[][] = { { 0, 1, 1, 1 },
                     { 0, 0, 0, 1 },
                     { 0, 0, 0, 1 },
                     { 1, 1, 0, 0 } };
    R = maze.length;
    C = maze[0].length;
    r.findPathMaze(maze);
}
```

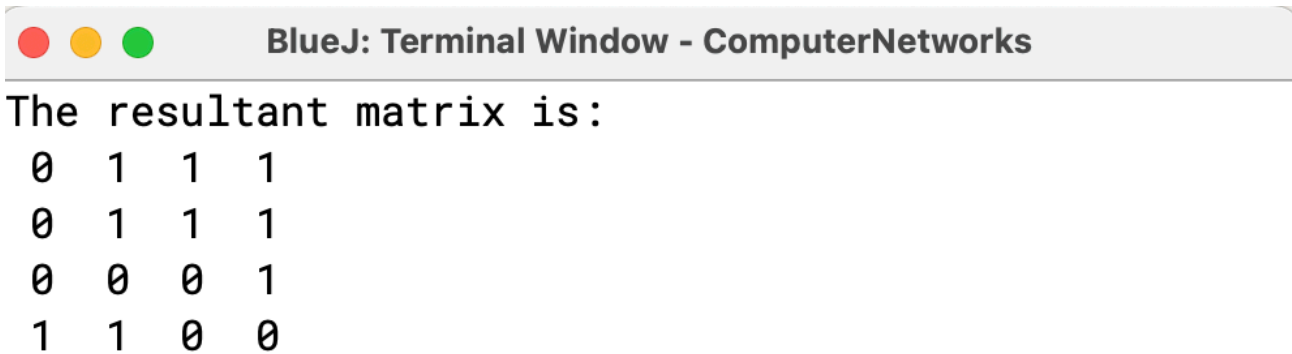
OUTPUT

OUTPUT 1 :



Path from source to destination doesn't exist

OUTPUT 2 : 0 is the path the rat has to take.



The resultant matrix is:

0	1	1	1
0	1	1	1
0	0	0	1
1	1	0	0

CONCLUSION

In the maze matrix, 1 means the block is a dead end and 0 means the block can be used in the path from source to destination. Note that this is a simple version of the typical Maze problem. For example, a more complex version can be that the rat can move in 4 directions and a more complex version can be with a limited number of moves. The path for the rat to take shown by the '0' in the maze.

Time Complexity: $O(2^{(n^2)})$.

The recursion can run upper-bound $2^{(n^2)}$ times.

Space Complexity: $O(n^2)$.

Output matrix is required so an extra space of size $n*n$ is needed.

REFERENCES

- 1) Gurari, Eitan (1999). "CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms". Archived from [the original](#) on 17 March 2007.
- 2) Biere, A.; Heule, M.; van Maaren, H. (29 January 2009). *Handbook of Satisfiability*. IOS Press. ISBN 978-1-60750-376-7.
- 3) Watson, Des (22 March 2017). [A Practical Approach to Compiler Construction](#). Springer. ISBN 978-3-319-52789-5
- 4) Rossi, Francesca; van Beek, Peter; Walsh, Toby (August 2006). "Constraint Satisfaction: An Emerging Paradigm". *Handbook of Constraint Programming*. Amsterdam: Elsevier. p. 14. ISBN 978-0-444-52726-4. Retrieved 30 December 2008.