

Data Structures and Algorithm
Mini Project



DATA STRUCTURES AND ALGORITHMS

Mini Project Report

on

“Non Linear Data Structures”

Minimum steps to reach target by a Knight

Submitted by:

Demira Ramnani	C030	70322000161
Reneeka Nadkarni	C034	70322000186
Aneri Patel	C056	70322000099
Semester/Year: VII/IV		

B. Tech Integrated Program
Department of Computer Science Engineering

SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)
Computer Engineering Department (B.Tech Integrated Sem VII)

Data Structures and Algorithm
Mini Project

PART A

A.1 AIM:

Identify a problem statement and design a solution using Non-Linear Data Structure

A.2 Pre requisite:

Basic Knowledge of Data Structures

A.3 Outcome:

After successful completion of this experiment students will be able to:

Develop a solution for a problem using non-linear data structure

A.4 Theory:

A mini project is desirable to be completed by a group of three or four students

A.5 Procedure/Task:

1. Identify a real life problem and design a solution using non-linear data structure
2. Prepare the document. Save and close the file and name it as RollNo._MiniProject.

SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)
Computer Engineering Department (B.Tech Integrated Sem VII)

Data Structures and Algorithm
Mini Project

PART B

Roll No. : C030, C034, C056	Name: Demira Ramnani, Reneeka Nadkarni, Aneri Patel
Class : B	Batch : B1-B2
Date of Experiment: 30-10-23	Date/Time of Submission :1-11-23
Grade :	

B.1 Project Details

1) Motivation to take the proposed problem statement

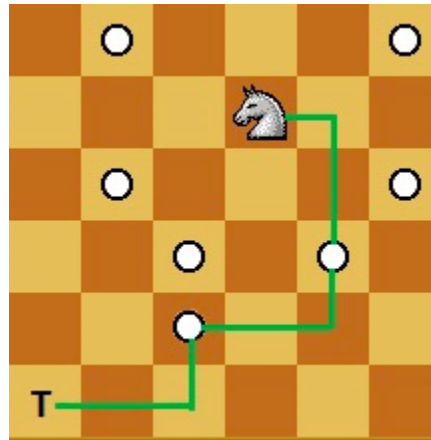
We were motivated to take this as our problem statement topic for our project since we felt it would be interesting to explore applications related to using tree search algorithms like BFS & DFS that have been taught to us in class.

The tree data structure allows us to see the possible states of the problem, and the tree search algorithm allows us to visit each node, thus evaluating it according to a cost function.

In this problem, we use tree search algorithm by representing each possible position of the knight as a node in the tree. The children of the node represent the possible moves that the knight can make from that position. The cost function can be the number of steps required to reach the target position from the current position. We see the possible moves towards the target position and choose the most efficient path.

We wished to develop a project that would develop our skills in search algorithms, problem solving, and programming and rewarding to complete. We are excited to learn more about this and curate an efficient and accurate solution.

Data Structures and Algorithm
Mini Project



2) Solution description

Problem Statement: Given a chessboard and the initial position of a knight (x, y) and a target position (tx, ty) on the same chessboard, find the minimum number of steps required for the knight to reach the target position.

Input:

n: Size of the chessboard (8x8 in this code).

x, y: Initial position of the knight.

tx, ty: Target position on the chessboard.

Solution:

1. Initialize a 2D array `dp` with dimensions 8x8 to store minimum step counts for various positions on the chessboard.
2. Create a recursive function `getsteps(x, y, tx, ty)` to calculate the minimum steps required for the knight to reach the target position (tx, ty) from its current position (x, y).
3. In the `getsteps` function:

SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)
Computer Engineering Department (B.Tech Integrated Sem VII)

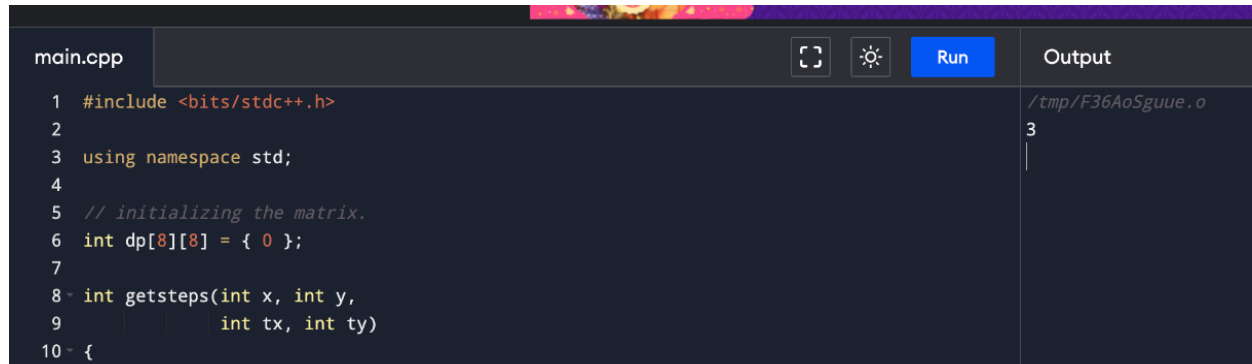
Data Structures and Algorithm

Mini Project

- If the knight is already on the target position ($x == tx$ and $y == ty$), return 0.
 - Check if the minimum steps for the given positions (x, y) and (tx, ty) have already been calculated. If yes, return the stored value from `dp`.
 - Divide the chessboard into four blocks (N-E, E-S, S-W, W-N) based on the knight's current position and the target position.
 - Calculate two possible positions ($x1, y1$) and ($x2, y2$) where the knight can move based on the block division. These positions minimize the distance to the target.
 - Recursively calculate the minimum steps required to reach the target from both ($x1, y1$) and ($x2, y2$).
 - Update `dp` with the minimum of these two values + 1 (accounting for the current move).
 - Since the problem is symmetric, exchange the coordinates x and y and update `dp` accordingly.
 - Return the minimum steps from `dp`.
4. In the main function:
- Initialize the chessboard size (n) and the initial knight position (x, y) and the target position (tx, ty).
 - Handle special cases where the minimum steps are known to be 4 due to the specific positions of the knight and target.
 - Set up predefined values in `dp` for specific differences between the knight's position and the target's position.
 - Call the `getsteps` function with the initial positions to find the minimum steps required to reach the target.
 - Print result/ans.

Data Structures and Algorithm
Mini Project

B.2 Output

A screenshot of a C++ IDE. The editor shows a file named 'main.cpp' with the following code:

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 // initializing the matrix.
6 int dp[8][8] = { 0 };
7
8 int getsteps(int x, int y,
9             int tx, int ty)
10 {
```

The IDE has a 'Run' button and a 'Output' pane on the right. The output pane shows the number '3'.

The output is 3, i.e the minimum number of steps taken by the knight from (4,5) to reach the target at (1,1) according to our input.

B.3 Project Code:

```
#include <bits/stdc++.h>

using namespace std;

int dp[8][8] = { 0 }; // initialise matrix; 8x8 board

int getsteps(int x, int y, int tx, int ty)
{
    // if knight position = target position return 0.
    if (x == tx && y == ty)
        return dp[0][0];

    else {
        if (dp[abs(x - tx)][abs(y - ty)] != 0) //if absolute diff between position is not 0,
            return calculated value
```

SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)
Computer Engineering Department (B.Tech Integrated Sem VII)

Data Structures and Algorithm

Mini Project

```
return dp[abs(x - tx)][abs(y - ty)];

else {

    int x1, y1, x2, y2; // (x1, y1) and (x2, y2) are possible positions.

    // From position of knight, the chess board can be divided into four blocks

    if (x <= tx) {

        if (y <= ty) {

            x1 = x + 2;

            y1 = y + 1;

            x2 = x + 1;

            y2 = y + 2;

        } else {

            x1 = x + 2;

            y1 = y - 1;

            x2 = x + 1;

            y2 = y - 2;

        }

    } else {

        if (y <= ty) {

            x1 = x - 2;

            y1 = y + 1;
```

SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)
Computer Engineering Department (B.Tech Integrated Sem VII)

Data Structures and Algorithm

Mini Project

```
x2 = x - 1;

y2 = y + 2;

} else {

    x1 = x - 2;

    y1 = y - 1;

    x2 = x - 1;

    y2 = y - 2;

}

}

// ans=1 + minimum of steps required from (x1, y1) and (x2, y2).

dp[abs(x - tx)][abs(y - ty)] = min(getsteps(x1, y1, tx, ty), getsteps(x2, y2,
tx, ty)) + 1;

// switching coordinates x with y of both knight & target shld gv same ans
dp[abs(y - ty)][abs(x - tx)] = dp[abs(x - tx)][abs(y - ty)];

return dp[abs(x - tx)][abs(y - ty)];

}

}

}

int main()
```


SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)
Computer Engineering Department (B.Tech Integrated Sem VII)

Data Structures and Algorithm
Mini Project

```
{  
  
    int i, n, x, y, tx, ty, ans;  
  
    n = 100; // size of chess board n*n  
  
    x = 4;  
  
    y = 5;  
  
    tx = 1;  
  
    ty = 1;  
  
    // Exception- four corner points for which the minimum steps is 4.  
  
    if ((x == 1 && y == 1 && tx == 2 && ty == 2) || (x == 2 && y == 2 && tx == 1 && ty == 1))  
        ans = 4;  
  
    else if ((x == 1 && y == n && tx == 2 && ty == n - 1) || (x == 2 && y == n - 1 && tx == 1 &&  
        ty == n))  
        ans = 4;  
  
    else if ((x == n && y == 1 && tx == n - 1 && ty == 2) || (x == n - 1 && y == 2 && tx == n &&  
        ty == 1))  
        ans = 4;  
  
    else if ((x == n && y == n && tx == n - 1 && ty == n - 1) || (x == n - 1 && y == n - 1 && tx ==  
        n && ty == n))  
        ans = 4;  
  
    else {  
  
        // dp[a][b]- a, b is the difference of x & tx and y & ty respectively.
```

Data Structures and Algorithm

Mini Project

```
dp[1][0] = 3;
dp[0][1] = 3;
dp[1][1] = 2;
dp[2][0] = 2;
dp[0][2] = 2;
dp[2][1] = 1;
dp[1][2] = 1;
ans = getsteps(x, y, tx, ty);
}
cout << ans << endl;
return 0;
}
```

B.4 Observations and Learning:

This problem is solved by recursively traversing the tree and updating the minimum number of steps required to reach each node.

The algorithm works by building up a table of the minimum number of steps required to reach each possible position on the chessboard, starting from the knight's current position. The table is initialized to infinity, and then the algorithm recursively updates the table entries for each position, using the following formula:

$$dp[i, j] = \min(dp[i + 1, j + 2], dp[i + 2, j + 1], dp[i - 1, j + 2], dp[i - 2, j + 1], dp[i + 1, j - 2], dp[i + 2, j - 1], dp[i - 1, j - 2], dp[i - 2, j - 1]) + 1$$

Data Structures and Algorithm
Mini Project

where $dp[i, j]$ represents the minimum number of steps required to reach position (i, j) from the knight's current position. The algorithm terminates when it has updated the table entry for the target position.

Understood how to efficiently navigate a grid or board that is crucial in many computer science applications. We demonstrated the use of dynamic programming and memoization to optimize this process of calculation. Done by storing and reusing previously computed values, i.e, avoiding redundant calculations and significantly improving efficiency.

B.5 Conclusion:

Successfully implemented application using BFS, i.e, to find minimum number of steps from knight to target position in C++. This problem requires a good understanding of tree data structures and recursive algorithms and provides an opportunity to learn about the dynamic programming algorithm, which is a powerful technique for solving optimization problems, by avoiding redundant calculations.

The code is well-written and efficient, and is able to solve the problem for large chessboards in a reasonable amount of time.