KNIGHT'S TOUR

A MINI PROJECT REPORT

18CSC204J -Design and Analysis of Algorithms Laboratory

Submitted by

Aniruddha Ghosh [RA2011030010038]

Rohan Kumar [RA2011030010050]

Under the guidance of

Dr. B Yamini

Assistant Professor, Department of Networking and Communication

In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in Cybersecurity

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

July 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this mini project report titled “KNIGHT’S TOUR” is the bonafide work done by Aniruddha Ghosh (RA2011030010038) and Rohan Kumar (RA2011030010050) who carried out the mini project work and Laboratory exercises under my supervision for **18CSC204J - Design and Analysis of Algorithms Laboratory**. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Dr. B Yamini
ASSISTANT PROFESSOR
18CSC204J -Design and Analysis of Algorithms
Course Faculty
Department of Networking and Communications

Signature of the Internal Examiner-I **Signature of the Internal Examiner-II**

ABSTRACT

The Knight's Tour is a classic chess problem which was studied (and probably solved) over 1000 years ago. The famous mathematician, Euler, published the first rigorous mathematical analysis of the problem in 1759. A knight can move in an “L shape” moving two squares in one direction and one square in a perpendicular direction. In this position, the black knight can move to any of the squares marked with green dots. While constructing a knight’s tour, a knight visits a space that has no move choices left and there are still spaces on the board that have not been visited, the constructor has run into a dead end.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATION

SYMBOLS/ ABBREVIATION	MEANING / EXPANSION
findSolutions	Find Optimal Solution
NLU	Natural Language understanding

CHAPTER-1

PROBLEM DEFINITION

A knight's tour is a series of moves that knight makes while visiting every cell of n x n of the chessboard exactly once. Consider the n x n chessboard as a knight's graph G = (V,E) where vertices € V represent the cell of the chessboard and every edge € E represent a knight's move from one cell to another. It is a special case of Hamiltonian-Path problem. The knight tour can either be open or close. A knight's tour is closed tour if a knight starts from one cell and visits all other cells of n x n chessboard exactly once and start position is reachable by one knight's move from last visited cell. Otherwise, it is an open path.

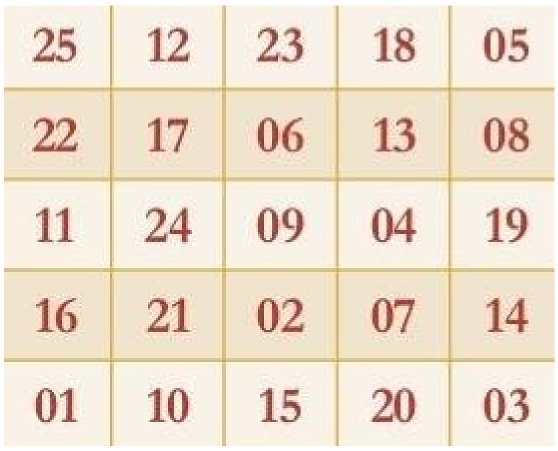
CHAPTER-2
PROBLEM EXPLANATION

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. This is because the process of finding the solution from the various option available is repeated recursively until we don’t find the solution or e reach the final state. So 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.

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 (since backtracking algorithms are generally exponential in both and space). However, a few problems still remain, that only have backtracking algorithms to solve them until now. Consider a situation that you have three boxes in front of you and only one of them has a gold coin in it but you do not know which one. So, in order to get the coin, you will have to open all of the boxes one by one. You will first check the first box, if it does not contain the coin, you will have to close it and check the second box and so on until you find the coin. This is what backtracking is, that is solving all sub-problems one by one in order to reach the best possible solution.

CHAPTER-3
DESIGN TECHNIQUES

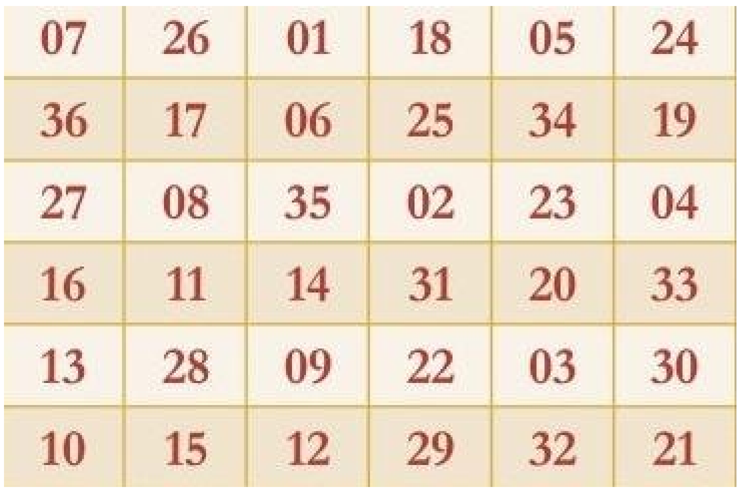
For chessboard with n = 5 and starting point as (4, 0). This is an open path.



25	12	23	18	05
22	17	06	13	08
11	24	09	04	19
16	21	02	07	14
01	10	15	20	03

For chessboard with n = 6 and starting cell as (0, 0).

This is a close cycle.



07	26	01	18	05	24
36	17	06	25	34	19
27	08	35	02	23	04
16	11	14	31	20	33
13	28	09	22	03	30
10	15	12	29	32	21

Keeping in mind Knight's Tour isn't a game, it's a real time problem solver which brings the final output. The user can't and doesn't have the option to simultaneously update the tour.

CHAPTER-4

ALGORITHM FOR THE PROBLEM

Pseudo Code for Backtracking:

Recursive backtracking solution.

```
void findSolutions(n, other params) :
  if (found a solution) :
     solutionsFound = solutionsFound + 1;
    displaySolution();
    if (solutionsFound >= solutionTarget) :
          System.exit(0);
    return
for (val = first to last) :
  if (isValid(val, n)) :
    applyValue(val, n);
    findSolutions(n+1, other params);
    removeValue(val, n);
```

CHAPTER-5

EXPLANATION OF ALGORITHM

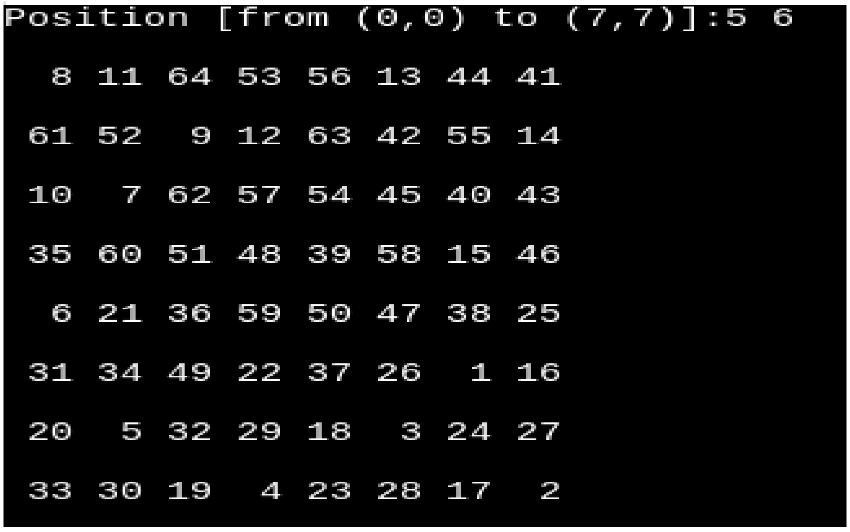
Pseudo Code for Knight's Tour

```
KNIGHT-TOUR(sol, i, j, step_count, x_move, y_move)
    if step_count == N*N
        return TRUE
    for k in 1 to 8
        next_i = i+x_move[k]
        next_j = j+y_move[k]
        if IS-VALID(next_i, next_j, sol)
            sol[next_i][next_j] = step_count
            if KNIGHT-TOUR(sol, next_i, next_j, step_count+1, x_move, y_move)
                return TRUE
            sol[next_i, next_j] = -1
    return FALSE
START-KNIGHT-TOUR()
    sol =[][]
    for i in 1 to N
        for j in 1 to N
            sol[i][j] = -1
    x_move = [2, 1, -1, -2, -2, -1, 1, 2]
    y_move = [1, 2, 2, 1, -1, -2, -2, -1]
    sol[1, 1] = 0
    if KNIGHT-TOUR(sol, 1, 1, 1, x_move, y_move)
        return TRUE
    return FALSE
```

Input

5 6

Output



```
Position [from (0,0) to (7,7)]:5 6

  8 11 64 53 56 13 44 41

 61 52  9 12 63 42 55 14

 10  7 62 57 54 45 40 43

 35 60 51 48 39 58 15 46

  6 21 36 59 50 47 38 25

 31 34 49 22 37 26  1 16

 20  5 32 29 18  3 24 27

 33 30 19  4 23 28 17  2
```

CHAPTER-6

COMPLEXITY ANALYSIS

There are N2 Cells and for each, we have a maximum of 8 possible moves to choose from, so the worst running time is O(8N^2).

Auxiliary Space: O(N2)

Important Note:

No order of the x Move, y Move is wrong, but they will affect the running time of the algorithm drastically. For example, think of the case where the 8th choice of the move is the correct one, and before that our code ran 7 different wrong paths. It's always a good idea a have a heuristic than to try backtracking randomly. Like, in this case, we know the next step would probably be in the south or east direction, then checking the paths which lead their first is a better strategy.

CHAPTER-7

CONCLUSION

Finally, we would like to conclude that this project of "Making a Knight's Tour using C++" can be best implemented by using backtracking approach as it is having the best time complexity and is more efficient compared to naive and brute force approach. Also, we would like to tell that the sudoku solver has many real lives uses as well, there are many people in the community who are serious hustlers of the game and for these percentage of people "Knight's Tour" can be a very useful and quick tool for cross verification.

REFERENCES

[1] https://www.javatpoint.com/mini-max-algorithm-in-ai
[2] https://en.wikipedia.org/wiki/Minimax
[3] https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_max_min_problem.htm
[4] https://codecrucks.com/max-min-problem/
[5] https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
[6] https://www.ques10.com/p/9057/write-an-algorithm-to-find-minimum-andmaximum-v-1/
[7] https://thedeveloperblog.com/daa/daa-max-min-problem
[8] https://www.upgrad.com/blog/min-max-algorithm-in-ai/
[9] https://www.techiedelight.com/find-minimum-maximum-element-arrayminimum-comparisons/
[10] https://towardsdatascience.com/understanding-the-minimax-algorithm726582e4f2c6
[11] https://en.wikipedia.org/wiki/Backtracking
[12] https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-1/
[13] https://www.geeksforgeeks.org/backtracking-introduction/
[14] Introduction To Algorithms – By Thomas H. Cornwell

APPENDIX

CODE

Program for Knight's Tour

```
#include <iostream>
#include <iomanip>
#include <conio.h>
using namespace std;
void possible();
void exits(int);
const int ver[]={-1,-2,-2,-1,1,2,2,1},
hor[]={2,1,-1,-2,-2,-1,1,2};
int row,col,npos;
int board[8][8],nextij[8][8][8],accessible[8][8];
int main()
{
int count=1,k,j;
cout <<"Position [from (0,0) to (7,7)]:";
cin>>row>>col;
cout<<endl;
board[row][col]=count;
while(count!=64)
{
count++;
possible();
exits(count);
}
for(j=0;j<=7;j++)
{
for(k=0;k<=7;k++)
cout<<setw(3)<<board[j][k];
cout <<"\n\n";
}
getch();
return 0;
```

```
}
void possible()
{
int npos;
for(int r=0;r<=7;r++)
{
for(int c=0;c<=7;c++)
{
npos = 0;
for(int i=0;i<=7;i++)
{
if(((r+ver[i] >=0) && (r+ver[i] <=7))&&((c+hor[i] >=0) &&
(c+hor[i] <=7))&&(board[r+ver[i]][c+hor[i]] == 0))
{
nextij[r][c][npos] = i;
npos++;
}
}
accessible[r][c] = npos;
}
}
}
void exits(int l)
{
int min =
accessible[row+ver[nextij[row][col][0]]][col+hor[nextij[row][col][0]]];
int r = row+ver[nextij[row][col][0]],c=col+hor[nextij[row][col][0]];
for(int i=1;i < accessible[row][col];i++)
if(min >=
accessible[row+ver[nextij[row][col][i]]][col+hor[nextij[row][col][i]]])
{
min
=accessible[row+ver[nextij[row][col][i]]][col+hor[nextij[row][col][i]]];
r = row + ver[nextij[row][col][i]];
c = col + hor[nextij[row][col][i]];
}
```

```
board[r][c]=l;
row = r;
col = c;
}
```