Knight's tour Problem

Kalpana Khutela
IIB2019019

Devang Bharti
IIB2019020

Hitka Rajesh Kumar
IIB2019021

Abstract—**In this report we used modified backtracking approach to solve the knight's tour problem**

Keyword : backtracking, time complexity, space complexity.

I. Introduction

In this problem, we have to propose a algorithm to solve knight's tour problem

we have given a brief idea of our algorithm **part II**.

II. Algorithm Approach

1) A BLOCK OF THE CHESS 'B' CAN BE TAKEN INTO FURTHER MOVE OF THE KNIGHT FROM BLOCK 'A' IF:

   'B' IS UNVISITED. 'B' CAN BE REACHED BY A SINGLE MOVE OF THE KNIGHT.

2) ASSUME 'A' TO BE THE RANDOM INITIAL POSITION FROM WHERE THE KNIGHT STARTS IT TRAVEL.

3) SINCE 'A' IS VISTED IT IS NEEDED TO MARK IT WITH "1".

4) TO MOVE TO ANY POSITION 'B' FROM 'A'

5) LET A SET 'S' HAVE ALL THE POSSIBLE POSITIONS WHICH CAN BE REACHED FROM 'A'.
6) FROM THIS SET 'S' CHOOSE THE PATH WHICH IS HAVING MINIMUM ONWARD MOVES.

7) AFTER BEING CHOSEN MARK IT '2'

8) THIS IS THE MAIN LOGIC BEHIND THIS PROBLEM AND THIS GOES ON FURTHER .

   For Example -
   Let input be
   for the position (0,0)

   There are 8 points (1,2) (2,1) (2,-1) (1, -2) (-1,-2) (-2,-1) (-2,1) and (-1,2).
   In 8 points (2,-1) (1, -2) (-1,-2) (-2,-1) (-2,1) (-1,2) points are invalid
   So we have to go either (1,2) or (2,1).
   We will choose any of them and then from that point will see the next 8 points and consider only the valid moves.
   If at any position no possible move will not possible then back to the previous move and from that go to the other path and stop if the step cont will be 64.

III. Pseudo Code I

```
Function isValid
Pass :int x, int y

  if $((x >= 0 and y >= 0) and (x < N and y < N))$ then
    return true
  end if
  retun False
```

IV. Pseudo Code II

```
Function isEmpty
Pass :int result[], int x, int y

  return (isValid(x, y))  (result[y*N+x] ¡ 0);
```

V. Pseudo Code III

```
Function int degree
Pass :nit result[], int x, int y

  $int count = 0$
  $int i = 0$ to $N$
  if $IsEmpty(result, x + nextx[i], y + nexty[i])$ then
    $cunt = count + 1$
    return count
```

VI. Pseudo Code IV

```
  Function FindNMove
  Pass : int result[], int x, int y
    $int min_deg_index = -1, c, min_deg = (N + 1), nx, ny$
    $int start \leftarrow rand()$ mod $N$
    $count \leftarrow 0$ to $N$
    $int i = (start + count)$ mod $N$
    $nx = *x + next_x[i]$
    $ny = *y + next_y[i]$
    if $isEmpty(result, nx, ny) and c = degree(result, nx, ny) < mindeg$ then
      $minimumindex = i$
      $mindegree = c$
    end if
  end for
  if $mindegindex == -1$ then
    return FALSE$nx = *x + nextx[mindegindex]ny = *x + nexty[mindegindex]result[ny * N + nx] = result[(*y)*N+(*x)]+1*x = nx*y = ny$return TRUE
```

VII. Pseudo Code V

```
  Function Print result
  Pass : int result[]
    $i = 0$ to$Nj = 0$ to$NPrintresult[j * N + 1]$
  end for
end for
```

VIII. Pseudo Code VI

```
Function bool checkNeighbour
Pass :int x, int y, int xx, int yy

    $(inti = 0)$ to $N$ if $(((x + nextx[i]) == xx)((y + nexty[i]) == yy))$ then return truereturn false
```

IX. Pseudo Code VII

```
  Function bool knightTour
    $intresult[N * N]$ $inti = 0$; to $N * N$
    $result[i] = -1$
    $intsx, sy$
    $cin >> sx >> sy$
    $intx = sx, y = sy$
    $result[y * N + x] = 1$
    $inti = 0$ to $N * N - 1$
    if $(findNmove(result, x, y) == 0)$ then
        return falseif (!checkNeighbour(x,y,sx,sy)) then return false
        $printResult(result)$
        return true
```

X. Time Complexity

the time needed for this algorithm grows roughly linearaly with the number of squares of the chessboard that is pow(n,2).

XI. Auxiliary Space Complexity

No extra space is used in this algorithm, so auxiliary space is of order pow(n,2).Only the input array is of size n*n. Space Complexity = Input Space+Auxiliary Space, which is equal to pow(n,2).

XII. Conclusion

The above proposed algorithm efficiently gives whether it can cover the board or not. The algorithm proposed is very efficient both space and time wise

XIII. References

1) optimization of knight and tour algorithm
2) Knight and tour algorithm

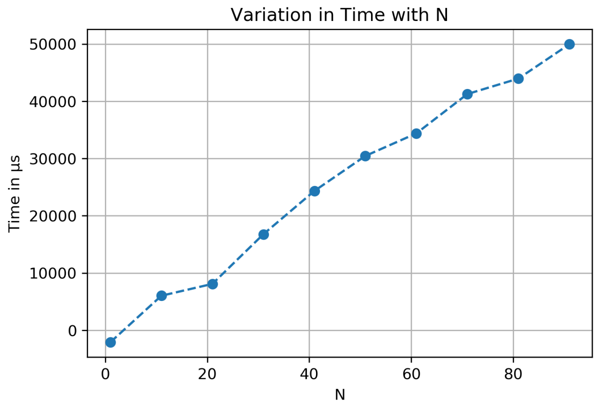

Fig. 1. Time complexity curve

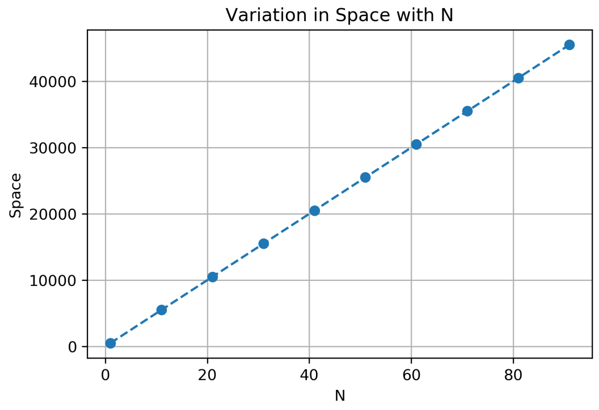

Fig. 2. Auxiliary space complexity curve