CSE 310 Honors Contract - Knights Tour Algorithm Time Complexity Analysis

Alexander Wood and Taman Truong

April 29th, 2022

Runtime Analysis - Knights Tour Constructor

The run time for the Knights Tour Constructor is $O(mn)$, since we are creating the $m \times n$ board using a nested for-loop containing two for-loops. The line-by-line analysis of each line in the Knights Tour Constructor is shown below, commented with "// $O(.)$" for the time complexity of each line and "// total $O(.)$" for the time complexity of the entire function.

```
#include <iostream>
#include <iostream>
#include <string>

class KnightsTour
{
private:
  int m, n, totalMoves;
  int ** board;
public:
  KnightsTour(int, int);
  bool isSafe(int, int);
  void printBoard();
  bool solveTourRecursive(int, int, int);
  void solveTour();

};

 // define possible x and y moves
 int xMoves[] = {1, 2, -2, -1, -1, 1, -2, 2}; // O(1)
 int yMoves[] = {-2, -1, -1, 2, -2, 2, 1, 1}; // O(1)

 // constructor
 KnightsTour::KnightsTour(int l, int w) // total O(mn)
 {
   // defines m, n, and the total number of moves
   this->m = l; // O(1)
   this->n = w; // O(1)
   this->totalMoves = l * w; // O(1)

   // create mxn board
   board = new int*[this->m]; // O(1)
   for(int i = 0; i < this->m; i++) // O(m)
   {
       board[i] = new int[this->n]; // O(1)
   }

   // set every spot to negative infinity // total O(mn)
```

```
    for(int i = 0; i < this->m; i++) // O(m)
    {
        for(int j = 0; j < this->n; j++) // O(n)
        {
            board[i][j] = INT_MIN; // O(1)
        }
    }
}
```

Runtime Analysis - Is-Safe Helper Function

The run time for the Is-Safe Helper Function is $O(1)$, since we are checking whether or not a particular knight move is legal (it will stay within the bounds of the board and not move to some point outside of the board). The line-by-line analysis of each line in the Is-Safe Helper Function is shown below, commented with "// $O(.)$" for the time complexity of each line and "// total $O(.)$" for the time complexity of the entire function.

```
// checks whether a given piece is good or not
bool KnightsTour::isSafe(int x, int y) // O(1)
{
    // checks if the move is in bounds or not
    if(((x >=0) && (x < this->m)) && ((y >=0) && (y < this->m))) // O(1)
    {
        // in bounds
        return true; // O(1)
    }
    // not in bounds
    return false; // O(1)
}
```

Runtime Analysis - Print-Board Helper Function

The run time for the Knights Tour Constructor is $O(mn)$, since we are traversing through the $m \times n$ board using a nested for-loop containing two for-loops. The line-by-line analysis of each line in the Print-Board Helper Function is shown below, commented with "// $O(.)$" for the time complexity of each line and "// total $O(.)$" for the time complexity of the entire function.

```
// print board
void KnightsTour::printBoard() // total O(mn)
{
   for(int i = 0; i < this->m; i++) // O(m)
   {
       for(int j = 0; j < this->n; j++) // O(n)
       {
           std::cout << board[i][j] << " "; // O(1)
       }
       std::cout << std::endl; // O(1)
   }
}
```

Runtime Analysis - Knight's Tour Recursive Solver Function

The run time for the Knight's Tour Recursive Solver Function is $O(8^{mn})$, since we are checking whether or not a particular knight move is legal (it will stay within the bounds of the board and not move to some point outside of the board), place it on the board array, and do this for all $m \times n$ squares on the board for all 8 possible knight square moves. However, not all squares - especially the 2 squares near the edges and vertices of the $m \times n$ board -

do not have 8 possible knight square moves, so the actual runtime for this recursive solver is slightly faster - $O(k^{mn})$ for $k < 8$. The line-by-line analysis of each line in the Knight's Tour Recursive Solver Function is shown below, commented with "// $O(.)$" for the time complexity of each line and "// total $O(.)$" for the time complexity of the entire function.

```
// the recursive call made to solve the tour
bool KnightsTour::solveTourRecursive(int moveCount, int x, int y) // O(8^(mn))
{
  // will hold the moves being tested
  int nextX, nextY; // O(1)

  // base case
  if(moveCount == this->totalMoves) // O(1)
  {
       // solution found
       return true; // O(1)
  }

   // else, solve recursively by iterating through moves
   for(int i = 0; i < 8; i++) // O(8^(mn))
   {
       // set next moves
       nextX = x + xMoves[i]; // O(1)
       nextY = y + yMoves[i]; // O(1)

       // check if move is in bounds AND is negative infinity
       if((isSafe(nextX, nextY) == true) && (board[nextX][nextY] == INT_MIN)) // O(1)
       {
           // move is valid, so make that edge
           board[nextX][nextY] = moveCount; // O(1)
           moveCount++; // O(1)

           // recursive step - solve next step
           if(solveTourRecursive(moveCount, nextX, nextY)) // O(mn)
           {
               return true; // O(1)
           }

           // else, if move is not valid for the future, back track
           board[nextX][nextY] = INT_MIN; // O(1)
           moveCount--; // O(1)
       }
   }
   // no solution
   return 0; // O(1)
}
```

Runtime Analysis - Solve Knight's Tour Function

The run time for the Solve Knight's Tour Function is $O(8^{mn})$, since we are solving the entire Knight's Tour with a $m \times n$ board using the recursive method defined previously. The line-by-line analysis of each line in the Solve Knight's Tour Function is shown below, commented with "// $O(.)$" for the time complexity of each line and "// total $O(.)$" for the time complexity of the entire function.

```
// solve tour - called from main
void KnightsTour::solveTour() // total O(8^(mn))
{
   // start at (0, 0)
   board[0][0] = 0; // O(1)

   // call recursive solution
   if(solveTourRecursive(1, 0, 0) == true) // O(8^(mn))
   {
        // if solution found, prints the board with move numbers
        printBoard(); // O(mn)
    }
    else
    {
        std::cout << "Something went wrong here..." << std::endl; // O(1)
    }
}
```

Runtime Analysis - Main Method

The run time for the Main Method is $O(8^{mn})$, since we are solving the entire Knight's Tour with a $m \times n$ board using the Solve Knight's Tour Function defined previously. The line-by-line analysis of each line in the Main Method is shown below, commented with "// $O(.)$" for the time complexity of each line and "// total $O(.)$" for the time complexity of the entire function.

```
#include <iostream>
#include <iostream>
#include <string>

#include "KnightsTour.h"

int main(int argc, const char * argv[]) // total O(8^(mn))
{
    // defining integers m,n for the size of the board
    int m, n; // O(1)

    std::cout << "Welcome to the Knights Tour solution by Alexander Wood and Taman Truong\n"
    << "This program will produce an open Knights Tour Solution for a mxn size board"
    << std::endl; // O(1)

    // grab dimensions
    std::cout << "Please enter size for m: " << std::endl; // O(1)
    std::cin >> m; // O(1)
    std::cin.ignore(); // O(1)

    std::cout << "Please enter size for n: " << std::endl; // O(1)
    std::cin >> n; // O(1)
    std::cin.ignore(); // O(1)

    // define board size
    KnightsTour * board1 = new KnightsTour(m, n); // O(1)

    // solve tour
    board1->solveTour(); // O(8^(mn))
}
```

Conclusion

This question is an application of the Graph Theory concept of a Hamiltonian graph. For a graph G to be Hamiltonian, there exists a subgraph $H \subseteq G$ that is a *spanning* subgraph of G if $V(H) = V(G)$. A spanning cycle (path) of G is called a *Hamiltonian* cycle (path). What makes Hamiltonian cycles applicable to computer science is that there is no efficient method for determining if a graph does or does not have a Hamiltonian cycle (path); thus, making this problem NP-complete. There are two general theorems in Graph Theory about deciding if a graph suffices at being Hamiltonian:

Ore's Theorem

If G is a graph with $|G| \geq 3$ and $d(x)+d(y) \geq |G|$ for all distinct nonadjacent vertices x and y, then G is Hamiltonian.

Dirac's Theorem

If $\delta(G) \geq \frac{1}{2}|G| > 1$, then G is Hamiltonian.

In relation to the Knights Tour problem, there is a theorem that lists a set of conditions on an $m \times n$ board such that a closed Knights Tour exists:

Schwenk's Theorem

Let $m, n \in \mathbb{Z}^+$ such that $m \geq n$. A (closed) Knight's Tour in the $m \times n$ board exists unless

1. m, n are both odd
2. $m = 1, 2, 4$
3. $m = 3$ and $n = 4, 6, 8$

Finding an efficient way to check whether or not a graph contains a Hamiltonian cycle (path) will ultimately same the question if $NP \subseteq P$ and many other questions that would spark a revolutionary change in mathematics, computer science, and the world around it.