# HW07: Knight's Move

**CSE2050 Fall 2017**
**Instructors: Jeffrey Meunier & Wei Wei**
**TAs: Jenny Blessing, Param Bidja, Yamuna Rajan & Zigeng Wang**

## 1. Introduction

In this assignment, we use graph traversal to solve an interesting chess problem. Specifically, we will use implement dfs and use it to solve the problem.

## 2. Objectives

The purpose of this assignment is to give you experience in:

- Understanding the dfs graph traversal
- How to use dfs to solve some real world problems.
- Understanding when to use dfs instead of bfs.

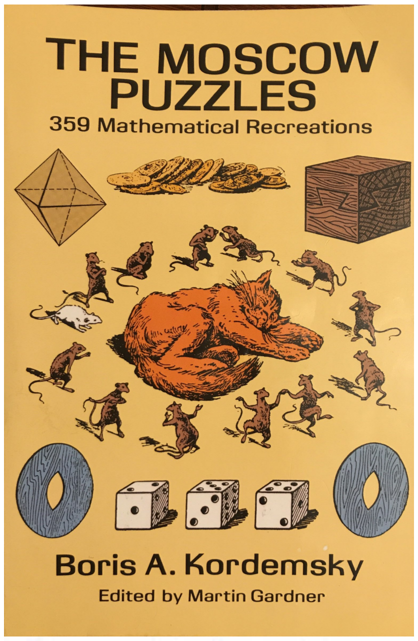
***Note**: Before you start, if you are not familiar with graph and graph traversal algorithms, it is recommended you review the corresponding class notes.*

## 3. Background

### 3.1. Knight's Move

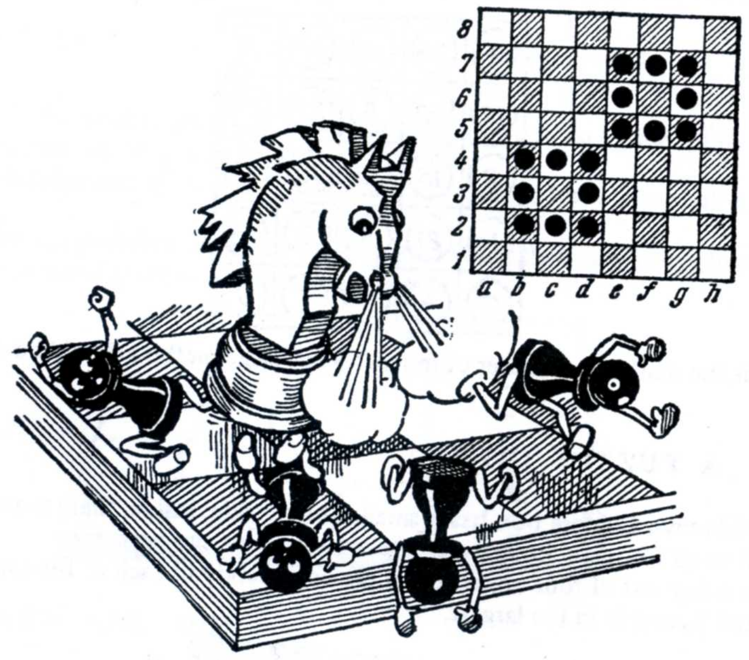
This problem is from book "The Moscow puzzles". The author of this book is Boris A. Kordemsky, and this book is edited by Martin Gardner. This problem appears in the chapter II Difficult Problems.

The following page displays the cover of the book, and the illustration and description of this problem from the book.



## 110. KNIGHT'S MOVE

To solve this problem you need not be a chess player. You need only know the way a knight moves on the chessboard: two squares in one direction and one square at right angles to the first direction. The diagram shows 16 black pawns on a board.



Can a knight capture all 16 pawns in 16 moves?

We will solve this problem using graph traversal. We consider the knight's position on the chessboard as a vertex in a graph. At the beginning, the knight can capture one of the 16 pawns. Therefore, we can start from any of the 16 starting points. At the end, all the pawns will be gone. Hence, the end point is at one of the 16 vertices.

The key to solving this problem is to construct a graph that has 16 vertices, and define edges according to the rule of how knights move in chess.

Once we have the graph, we can traverse the graph and the goal is to traverse every single vertex without repeating.

There are multiple solutions to this problem. We will try to find a solution using DFS. Since it is not guaranteed that we can find a solution with one DFS traversal, we will do DFS multiple times. To make sure we traversal different paths when we do multiple DFS traversals, we change the original DFS algorithm a little bit to make it behave randomly. That is, when we push all the neighbors of a vertex on to the stack that is used for DFS, we randomly shuffle the order of the neighbors before we push them onto the stack. This ensures that for each DFS, the path traversed is random.

## 4. Assignment

In the skeleton zip file, you will find a skeleton for your .py file named *chess.py*. All you need to do is to complete the skeleton code based on the instructions and submit it to the Mimir system.

There are two places you need to fill in code.

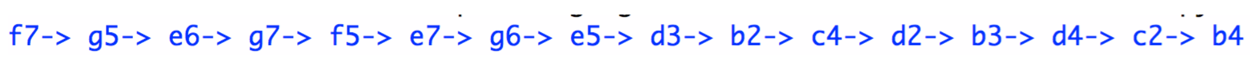
### 4.1. Method randomdfs

We first copy code from dfs method and make necessary changes. The goal is to make sure that when we add neighbors to a stack, we add them in a random order. This way, every time we do this random dfs, the result is random.

### 4.2. Inverse DFS tree and display result

The randomdfs(v) method will return a depth-first search tree. This tree is implemented as a dictionary. For each key (vertex), the value is its parent vertex. Now we need to check whether this tree is actually linear. If this tree is linear (a straight line) then we have a solution. If this is true, we need to make a new dictionary that is the inverse of this dictionary tree. That is, the key and value is switched for each item. Using this inversed dictionary, we print out all the moves the knight takes, and return this inversed dictionary as the return value of the knightsmove function.

Below is a screenshot of my results.



```
f7-> g5-> e6-> g7-> f5-> e7-> g6-> e5-> d3-> b2-> c4-> d2-> b3-> d4-> c2-> b4
```

## 5. Submit your work to Mimir

Submit your code to Mimir after you complete your code. Mimir will automatically check the output of your code and grade your submission. You can submit your code to Mimir up to **30 times** to refresh your existing score before the submission deadline.

## 6. Due date

This lab assignment is worth **4 points** in the final grade. It will be due by **11:59pm on December 8th, 2017**. A penalty of **10% per day** will be deducted from your grade, starting at 12:00am.

## 7. Getting help

Start your project early, because you will probably not be able to get timely help in the last few hours before the assignment is due.

- Go to the office hours of instructors and TAs.
    - Prof. Wei Wei: Mon. 2:30 - 3:15pm, Wed. 2:30 - 3:15pm, Thurs. 2:30 - 3:15pm, Fri. 2:30 - 3:15pm @ITE258
    - Jenny Blessing: Fri. 12pm - 2pm @ITE140
    - Param Bidja: Tues. 2pm - 3pm @ITE140
    - Yamuna Rajan: Tues. 11am - 12pm, Wed. 9:30am – 10:30am @ITE140
    - Zigeng Wang: Mon. 3pm - 4pm @ITE140
- Post your questions on Piazza. TAs and many of your classmates may answer your questions.
- Search the answer of your unknown questions on the Internet. Many questions asked by you might have been asked and answered many times online already.