

Programming Languages

Professor: Benjamín Valdés Aguirre

**Implementing Knight’s Path Algorithm in Prolog**

Bernardo Estrada Fuentes

A01704320

Contents

1. Introduction
2. Algorithm
3. Understanding backtracking
4. Implementation
5. Installation
6. Conclusions
7. Bibliography

**1 Introduction**

In this project, I will be working on the implementation of a solution for the knight’s path algorithm. Also, I will add a visualizer to be able to better understand the path taken. I will be using Prolog as a programming language due to its ability to easily backtrack and implement this type of algorithms.

Note the full scope of the project is to run the algorithm and visualization as efficiently and user friendly as possible, in further versions the visualizer and the compiled program are expected to be joined to create a seamless preview.

**2 Algorithm**

The algorithm I will be using is based on Warnsdroff rule, which consists of taking the path with the least possible continuations to find the best possible answer.

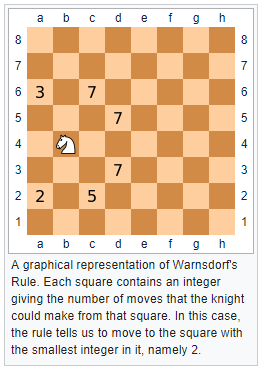


Figure 1: Warnsdroff Rule Visualization

As seen on Figure 1, the knight can move to 6 spaces. To choose where to go, each of this space is checked for next moves and the one with the least, in this case a2 with 2 possible moves, is chosen. This process is repeated until there are no more moves available.

**3 Understanding Backtracking**

To understand backtracking and how it helps us solve this problem we can think about many stereotypical chess players. How they are portrayed to think all the possible moves their opponent can take. This is very similar to backtracking, in the way that backtracking tries a solution and checks the following steps, if results of these are not favorable, the algorithm goes back and tries another solution.

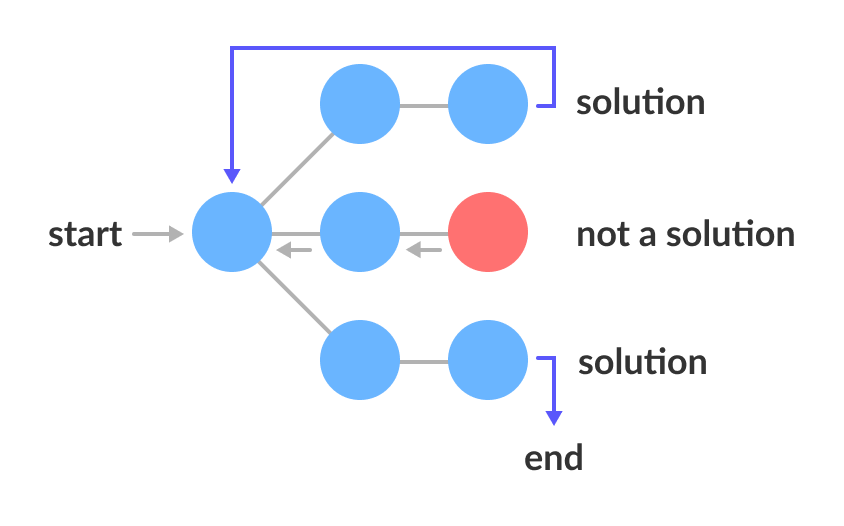


Figure 2: Backtracking Visualization

**4 Implementation**

The algorithm has been implemented to run in both the SWI-Prolog engine and the GPLC-C compilation engine. The algorithm is the same, just some things have been changed to work in each environment.

A benchmarking script has ben created, it runs each program with the same input values one hundred times and print the average and total time it took.

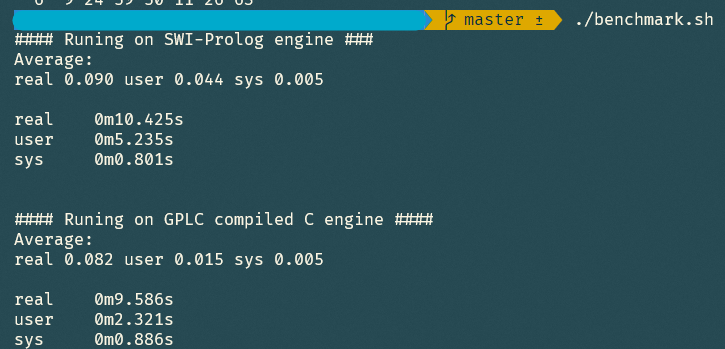


Figure 3: Benchmarks

As seen on Figure 3, the difference between the SWI and the compiled engines are significantly different. The compiled version has a 225% improvement over the Interpreted version, both running with a 10x10 board and an initial position of 2, 4.

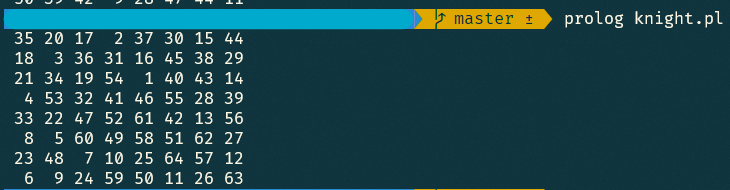


Figure 4: Output

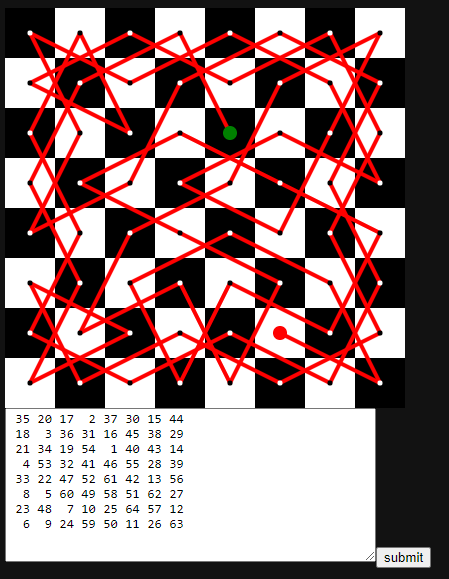
Taking this Output, we can copy it and paste in the visualizer (index.html) to see the path in an easier way to understand.

Figure 5: Visualization

**5 Installation**

(Ubuntu)

To run this program, its only necessary to clone the repo and install the following dependencies.

sudo apt-get install swi-prolog

sudo apt-get install gprolog

To run the SWI program run:

prolog knight.pl

To compile and run the GPLC program run:

gplc knightGPLC.pl -o o

./o

Backtracking Algorithm. (2021). Retrieved 25 May 2021, from <https://www.programiz.com/dsa/backtracking-algorithm>