TEC | Tecnológico de Costa Rica

Knight's Tour

Project

Sebastián M. Chen Cerdas
Valerie M. Hernández Fernández
Oscar M. Soto Varela

Costa Rica Institute of Technology
Cartago Central Technology Campus
Computer Engineering Academic Area

CE 3104 Languages, Compilers, and Interpreters
Eng. Marco Rivera Meneses, MGP
Class 01

I Semester

May 19, 2023

Contents

1. Knight's Tour

1.1. Source code

Check out the project source code at

1.2. Objective

Knight's Tour (PDC) is the first project/programmed homework from the course CE3104 Compiler and Interpreter Languages. The main objectives of this homework were to reinforce the topics studied such as functional paradigm and racket programming and work on the engineering knowledge of the students. In groups would work together to design, work, and resolve for a common goal, developing logical and graphical software with its respective documentation, while developing the organization a planification skills to end up the homework on time.

2. Description

For this project the students received the task to resolve the famous and old mathematical problem Knight's Tour, in summary the goal is to find a sequence of moves of a knight on a chessboard such that the knight visits every square exactly once. The path doesn't necessarily have to be closed, so the knight is not forced to end the path in the same square that it started**.** The solution program should also be able to find solutions in chessboards with size $N \times N$. With a minimum size of 5×5 and a maximum of 18×18. It should be able to find at least 5 different solutions for the problem and graphically draw one of them.

There are some other considerations requested for the Knight's Tour. The code for the solution should be programmed in the Racket language, using functional paradigm. That implies a complete restriction of the use of iterative functions such as let, for, when, etc. At least for the logical solution, the Graphical User Interface (GUI) is an exception for such restriction.

3. Requirements

Essentials for the best user experience with the Knight's Tour app are listed below:

- IDE : DrRacket
- Operative System. : Windows 10
- Programming language: Racket v. 8.8
- RAM : 128 Mb
- Storage : 1.5 Mb

4. Algorithms

4.1. Sorting

To achieve the solution through Warnsdorff's algorithm it was needed to develop a sorting algorithm to take the best paths for the knight. It was selected Quicksort as the sorting algorithm for this task, with the modification to be able to sort degrees instead of natural numbers as does Quicksort.

For a better understanding it's important to clarify that a degree is a list containing a list with the knight position and the number of movements that a knight can do from its position. An example of a degree can be seen in **Figure 1.**

Figure 1. 5 × 5 board central position degree as sample of a degree's list element.

So, the Quicksort algorithm modified for degrees, has the purpose of facilitating find the movement with a smaller number of future possible moves. And the complete code can be found in the GitHub repository, this Quicksort algorithm can be found with the name of sort-degrees in the repository.

4.2. Heuristic

The algorithm used to create the solutions of the knight's tour problem it's based on the Warnsdorff's rule, it uses a heuristic to select the movement where the knight would have the least amount of onwards moves, when counting the onwards moves we don't count the revisiting squares. To adapt to the creation of several solutions a randomized selection is made from time to time to create a different solution from the same start.

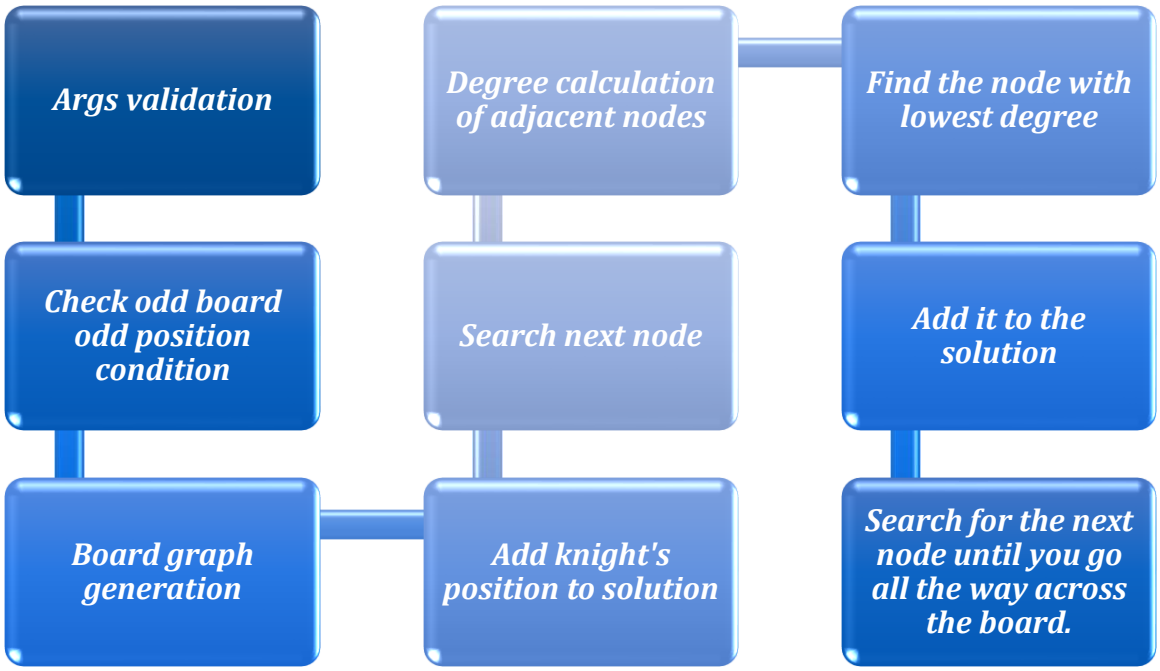

Figure 2. Summary of solution computation algorithm.

It is important to emphasize that the project is developed using the functional programming paradigm. Therefore, the use of variables or any function of the Racket language that does not respect the paradigm is not part of the implementation. The process will be repeated until the number of solutions desired by the user is generated.

5. Data Structures

Matrix [list of lists]: For this project, the use of a list that contains multiple sub lists was implemented in several ways. The first one is a $N \times 2$ matrix made to represent the movements of the knight. The list contains pairs of numbers inside several sub lists, which represent the knight's positions in X and Y coordinates. The size of the list depends on the number of movements made by the knight for each solution. An example of this kind of matrix can be seen in **Figure 3**.

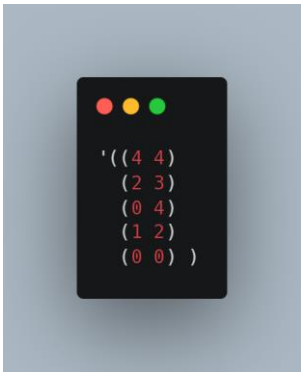

Figure 3. $N \times 2$ matrix sample of the solution's simple form.

The second implementation of a list of lists is $N \times N$ matrix made to represent the chessboard in which the knight moves. It is a list that contains the rows of the board as sub lists of numbers. Each number represents a knight's movement, and it is placed in the respective row, a 0 in a row means that the knight hasn't stepped on that square. Below you can see an example of a 5×5 solution matrix:

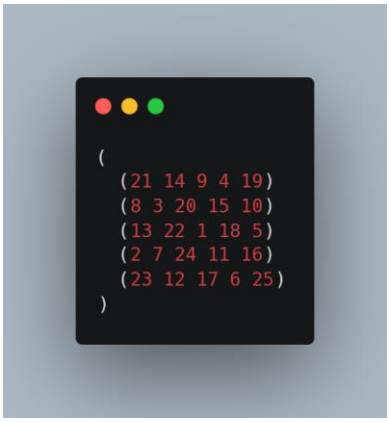

Figure 4. Solution board matrix 5×5 sample .

As can be seen in **Figure 4**, the horse started in row number 2 and column number 2, as it is indicated with the number 1 because it is the first position of the horse. For a better comprehension of the structure, you can see **Figure 5**, where the same matrix is drawn on a GUI.

Figure 5. GUI view of a solution matrix 5 × 5 form.

The third implementation is a list of lists to represent the game board as a graph, to implement the solution. It's made of lists that contain the board position as a node and the paths available to that node as pairs of numbers inside a list as edges. This structure was implemented because the possible routes are constant during the execution of the algorithm since the board does not vary over time. Then, to improve the speed of the algorithm, this structure is used as a road map for the solution.

```
'(
  ((0 0) ((1 2) (2 1)))
  ((0 1) ((1 3) (2 0) (2 2)))
  ((0 2) ((1 0) (1 4) (2 1) (2 3)))
  ((0 3) ((1 1) (2 2) (2 4)))
  ((0 4) ((1 2) (2 3)))
  ((1 0) ((0 2) (2 2) (3 1)))
  ((1 1) ((0 3) (2 3) (3 0) (3 2)))
  ((1 2) ((0 0) (0 4) (2 0) (2 4) (3 1) (3 3)))
  ((1 3) ((0 1) (2 1) (3 2) (3 4)))
  ((1 4) ((0 2) (2 2) (3 3)))
  ((2 0) ((0 1) (1 2) (3 2) (4 1)))
  ((2 1) ((0 0) (0 2) (1 3) (3 3) (4 0) (4 2)))
  ((2 2) ((0 1) (0 3) (1 0) (1 4) (3 0) (3 4) (4 1) (4 3)))
  ((2 3) ((0 2) (0 4) (1 1) (3 1) (4 2) (4 4)))
  ((2 4) ((0 3) (1 2) (3 2) (4 3)))
  ((3 0) ((1 1) (2 2) (4 2)))
  ((3 1) ((1 0) (1 2) (2 3) (4 3)))
  ((3 2) ((1 1) (1 3) (2 0) (2 4) (4 0) (4 4)))
  ((3 3) ((1 2) (1 4) (2 1) (4 1)))
  ((3 4) ((1 3) (2 2) (4 2)))
  ((4 0) ((2 1) (3 2)))
  ((4 1) ((2 0) (2 2) (3 3)))
  ((4 2) ((2 1) (2 3) (3 0) (3 4)))
  ((4 3) ((2 2) (2 4) (3 1)))
  ((4 4) ((2 3) (3 2)))
)
```

Figure 6. Graph matrix form of a 5 × 5 board sample.

6. Main functions

♘ **create-graph**: this function receives the board size to create a square matrix as the graph structure where the first element is the node, and the second element is a list with the edges. To create the graph, it calls an auxiliar function to generate and get the edges, the nodes are the squares on the board which the knight can reach, and the edges represent the L movement of the knight to said nodes.

```
(define (create-graph board-size (node '(0 0)) (graph '()))
  (cond
    ((= (first node) (second node) (- board-size 1))
      (append graph (list (cons node (list (get-edges node board-size)))))
    )
    (else
      (cond
        ((< (second node) (- board-size 1))
          (create-graph
            board-size
            (list (first node) (+ (second node) 1))
            (append graph (list (cons node (list (get-edges node board-size)))))
          )
        )
        (else
          (create-graph
            board-size
            (list (+ (first node) 1) 0)
            (append graph (list (cons node (list (get-edges node board-size)))))
          ))))))

(define (get-edges position board-size)
  (cond
    ((or (null? position) (null? board-size)) (error "kt-get-edges arguments must be non-null"))
    ((not (valid-size? board-size)) (raise-argument-error 'kt-get-edges "board-size doesn't meet the
requirements" board-size))
    ((not (valid-position? position board-size)) (raise-argument-error 'kt-get-edges "position doesn't
meet the requirements" position))
    (else (generate-edges (first position) (second position) board-size))
))
```

♘ **solution**: this function receives the size of the board and the initial position of the knight to generate a solution for the knight's tour problem. To create a solution this function calls an auxiliar function to generate the solution, this auxiliar function starts building the solution from the graph using the main algorithm which decides the best next movement for the knight.

```
(define (solution (board-size 8) (knight-position '(0 0)))
  (cond
    ((or (null? board-size) (null? knight-position)) (error "kt-solution arguments must be non-null"))
    ((not (valid-size? board-size)) (raise-argument-error 'kt-solution "board-size doesn't meet the
requirements" board-size))
    ((not (valid-position? knight-position board-size )) (raise-argument-error 'kt-solution "position
doesn't meet the requirements" knight-position))
    ((not (tour? knight-position board-size)) (displayln (string-append "A complete Knight's Tour don't
exist from odd position '" (~a knight-position) " in odd square board size '(" (~a board-size) ") 🐴
\n")) '())
    (else (create-solution board-size knight-position (create-graph board-size)))
  ))

(define (generate-solution board-size knight-position graph (solution '()))
  (cond
    ((null? knight-position)
      (cond
        ((equal? (length solution) (expt board-size 2)) solution)
        (else '())
      ))
    (else
      (generate-solution
        board-size
        (next-node
          graph
          (available-edges (edges knight-position graph) solution)
          (append solution (list knight-position))
        )
        graph
        (append solution (list knight-position))
      ))))
```

♘ **solutions**: this function receives the size of the board, the initial position of the knight and the number of solutions wanted to create different solutions for the knight's tour and then print them on the console. It uses the main algorithm to generate the solutions.

```
(define (solutions (n 5) (board-size 8) (knight-position '(0 0)))
  (cond
    ((or (null? board-size) (null? knight-position)) (error "kt-solutions arguments must be non-null"))
    ((or (not (exact-positive-integer? n)) (> n (* board-size 2))) (raise-argument-error 'kt-solutions
"n doesn't meet the requirements" n))
    ((not (valid-size? board-size)) (raise-argument-error 'kt-solutions "board-size doesn't meet the
requirements" board-size))
    ((not (valid-position? knight-position board-size )) (raise-argument-error 'kt-solutions "position
doesn't meet the requirements" knight-position))
    ((not (tour? knight-position board-size)) (displayln (string-append "A complete Knight's Tour don't
exist from odd position '" (~a knight-position) " in odd square board size '(" (~a board-size) ") 🐴
\n")) '())
    (else (displayln "(")(print-solutions (create-solutions n board-size knight-position (create-graph
board-size)))(displayln ")"))
  ))

(define (create-solutions n board-size knight-position graph (solutions '()) (solution (create-solution
board-size knight-position graph)))
  (cond
    ((equal? (length solutions) n) solutions)
    (else
      (cond
        ((available? solution solutions) (create-solutions n board-size knight-position graph (cons
solution solutions)))
        (else (create-solutions n board-size knight-position graph solutions))))))


(define (print-solutions solutions)
  (cond
    ((null? solutions) (display ""))
    (else
      (display "  ")
      (displayln (car solutions))
      (print-solutions (cdr solutions)))))
```

♘ **test**: this function receives the size of the board and the solution of the knight's tour and prints the solution on the console as a matrix with the movements of the knight.

```
(define (test (board-size 8) (solution (solution board-size '(0 0))))
  (cond
    ((or (null? board-size) (null? solution)) (error "kt-test arguments must be non-null"))
    ((not (valid-size? board-size)) (raise-argument-error 'kt-test "board-size doesn't meet the
requirements" board-size))
    ((not (valid-solution? board-size solution)) (raise-argument-error 'kt-test "solution doesn't meet
the requirements" solution))
    (else (print-board (generate-board board-size solution)))
  )
)

(define (print-board board)
  (cond
    ((null? board) board)
    (else
      (displayln "(")
      (print-rows (+ (exact-floor (log (expt (length board) 2) 10)) 1) board)
      (displayln ")")
    )
  )
)
```

♘ **paint**: this function receives the size of the board and the solution of the knight's tour for that same board to call the graphic interface with these values loaded.

```
(define (paint (board-size 8) (solution (solution board-size '(0 0))))
  (cond
    ((or (null? board-size) (null? solution)) (error "kt-paint arguments must be non-null"))
    ((or (not (valid-size? board-size)) (> board-size 18)) (raise-argument-error 'kt-paint "board-size
doesn't meet the requirements" board-size))
    ((not (valid-solution? board-size solution)) (raise-argument-error 'kt-paint "solution doesn't meet
the requirements" solution))
    (else (visualizer board-size solution (generate-board board-size solution)))
  )
)
```

7. Issues

As in any kind of software development, there were multiple issues found along the development process, some of them were successfully solved and a still requires a solution. For both circumstances, those active and solved issues are listed below:

7.1. Active Issues

By the end of the due date almost everything were functioning correctly, but there was still found a minimal issue with no enough time to be solved. You can read more information about this issue below.

Table 1. Knight's Tour project active issues.

I	Description	Tries	Considerations	Status
1	There is a bug involving the GUI, in which some computers with certain screen resolution can't display the program window properly, and cropping part of the window. This bug seems to be unusual, and laptops appear to be more susceptible to this happening.	Several changes with the window size parameters were tried but any of them worked	It seems that racket GUI maybe is not much portable to different environments with too different screens	Unsolved

7.2. Fixed Issues

As in any programming project, throughout the development process there were found and solved a several issues. But fortunately, all of them were solved successfully by the end of the due date. So, below you can find more information about some of the most relevant solved issues in table:

Table 2. Knight's Tour project fixed issues.

I	Description	Tries	Solution	Considerations	Results	References
1	Before adding a slider, the program used to show the matrix through a button that showed an animation of the knight's movement. The issue with this implementation was that it used a thread that couldn't animate big matrixes without visual bugs.	It was intended to fix that through a timing adjustment, but it didn't work.	The best solution was to replace the button with a slider, and it worked perfectly.	As a recommendation, consider to use a slider to show progressive solutions in a practical way.	Racket buttons can be impractical, but it is concluded that there are a different variety of alternatives that can replace them.	*docs.racket-lang.org/gui/slider*
2	It was unable to create a backwards animation of the knight's movements with slider	Ignore the backwards movement and only show forward move with the slider.	We adapted the function up to be able to go backward and move properly in the solution through the slider	It is important to consider that the sliders are difficult to position, and require more space than buttons. So, they aren't the best tool if you lack of space.	Sliders can be a bit tricky depending of what you plan to do. But they also are really functional an useful even tough they are a bit rough to position.	*github/goober99/lisp-gui-examples*
3	There were generated several sliders on a same windows, if you called the draw function more than one time	It was tried to solve limiting the maximum times you could call the function to only one.	It was implemented a kind of singleton for the windows generation and components	The Racket GUI can be a bit different than usual libraries and because of the functional paradigm, implementing a singleton can be tricky to understand	At the end the multiple calls to the draw function were handled correctly and didn't triggered more errors or bugs.	*docs.racket-lang.org/gui/slider*

8. Project management

As a software project, there was carried out some project management techniques to handle and track the development process, especially because the limited time of only two weeks disposed to develop the Knight's Tour solution.

8.1. Plan

The planification consisted mostly of three activity tables with deadlines for each task, with specified description, deadline, student responsible.

Table 3. Weekly goals.

Activity	May	
	1st Week (08 - 12)	2nd Week (15 - 19)
Operational and administrative aspects assessment.		
Possible solutions research and analysis.		
Algorithm development.		
GUI development.		
Code debugging.		
Technical documentation.		

In the next table you can see the development of the activities in the project with a brief description, date and the respective responsible. But you can see an extended description of the tasks in the table .

Table 4. Activities implemented by contributors in deadlines calendar.

Activity			May									
Description	**Area**	**Contributors**	**08**	**09**	**10**	**11**	**12**	**15**	**16**	**17**	**18**	**19**
Feature assignment	ADM	Team										
Repository structure	ADM	Valerie										
Log control form	DOC	Valerie										
Args error handling	LOG	Valerie										
Board processing	LOG	Valerie										
Node processing	LOG	Valerie										
Degree sorting algorithm	LOG	Valerie										
Edge processing	LOG	Valerie										
Graph processing	LOG	Valerie										
Heuristic algorithm	LOG	Valerie										
Debugging	LOG	Team										
Display set-up	GUI	Sebastián										
Board processing	GUI	Óscar										
Position sorting algorithm	GUI	Óscar										
Manual animation	GUI	Óscar										
Automatic animation	GUI	Sebastián										
Components set-up	GUI	Valerie										
Debugging	GUI	Team										
Log's documentation review	ADM	Valerie										
Technical documentation	DOC	Team										
Demo	DOC	Valerie										
User manual	DOC	Valerie										
Deadline	ADM	Team										

8.2. Log

In this section you can find one log table for each member, where you can track the progression and work of each student individually along with the commits in the GitHub repository.

Table 5. Sebastian's Log.

Date	Duration (hr.)	Description	Contributors
08/05/2023	3	There were assign different roles and tasks for each group member	Valerie Sebastián Óscar
10/05/2023	2	Today we got to catch up on the code and the commits that Valerie did, so we are on the same page. We also got to point out questions about the problem and took notes to ask the professor about them.	Valerie Sebastián
12/05/2023	2	Added small comments for the simple functions on the logic file.	Sebastián
18/05/2023	3	Adding data structure and functions to the documentation	Sebastián

Table 6. Óscar's Log.

Date	Duration (hr.)	Description	Contributors
08/05/2023	3	There were assign different roles and tasks for each group member	Valerie Sebastián Óscar
11/05/2023 - 15/05/2023	12.33	In this period is started the development of the GUI till be able to draw a matrix and a solution with a button.	Óscar
16/05/2023	10	The visualization of the solution is improved with the addition of slider	Óscar Valerie
17/05/2023	15.49	Solved errors between the slider and the horse animation.	Óscar
17/05/2023	1	Errors with the automatic view button solved. Ended up erasing the complete button because of lack of utility compared with the slider	Óscar
18/05/2023	3	The chessboard were resized to properly see it on others computers and resolutions	Óscar
18/05/2023	10	Worked in the Knight's Tour formal presentation of documentation.	Óscar

Table 7. Valerie's Log.

Date	Duration (hr.)	Description	Contributors
06/05/2023	3	The project requirements were read and the problem was evaluated from a mathematical point of view. In addition, the project folder structure was established in the repository and a form for the log control was created.	Valerie
08/05/2023	2.3	Validation functions were implemented to validate arguments received by the parameters of the main functions of the algorithm. The name of the project was also updated from "Knight" to "Knight's Tour" in all the instances related to the project.	Valerie
09/05/2023	2.6	Some board parsing functions were implemented. Most of them are related to the basics op: create a board of specific size, check the size of the board, get the value at the required position, check if the position is available using 0 as availability value.	Valerie
10/05/2023	2	Today we got to catch up on the code and the commits that Valerie did, so we are on the same page. We also got to point out questions about the problem and took notes to ask the professor about them.	Valerie Sebastián
10/05/2023	2	Functions related to the graph's node edges were implemented.	Valerie
11/05/2023	5	The algorithm of the main functions "Paint" and "Test" has been implemented. Research is carried out on the mathematical evaluation of the problem since there are boards with dimensions where a complete tour isn't possible.	Valerie
12/05/2023	1	Review and correction of the latest version of code documentation.	Valerie
12/05/2023	1	An error was found and corrected in the edge filter function because it did not consider the case of going out of the upper range of the matrix. In addition, the function that generates a graph with all the positions of the board and their respective movement options is implemented.	Valerie
16/05/2023	10	Improved visualization of the solution and addition of slider	Valerie Oscar

9. Conclusions

It was successfully developed a complete functional program capable of resolve the Knight's Tour problem with an open solution in a chessboard of any size and initializing in any given position.

A simple and user-friendly Graphical User Interface was successfully developed to help to show the solutions in a way more visual and easier to understand.

The logical code solution was entirely development with Racket and complete functional paradigm oriented. Basing the solution on the use of lists and recursion to solve the problem.

10. Recommendations

As a recommendation for another similar project is the implementation of a closed solution to add a another and more challenging feature to the program.

It is recommended to develop a program able to work entirely from the Graphical User Interface to create a more user-friendly program. It could be challenging because of the Racket GUI libraries complexity but surely would improve the final program.

If it is possible, is recommended to implement iteration in some parts of the code to improve the efficiency and time of execution. This is because there is a notorious time to solve big sized chessboards with a entirely recursion oriented solution.

11. References

Felleisen, M., Findler, R. B., Flatt, M., Krishnamurthi, S., Barzilay, E., McCarthy, J., & Tobin-Hochstadt, S. (2015). The racket manifesto. In 1st Summit on Advances in Programming Languages (SNAPL 2015). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. Felleisen, M., Findler, R. B., Flatt, M., Krishnamurthi, S., Barzilay, E., McCarthy, J., & Tobin-Hochstadt, S. (2015). The racket manifesto. In 1st Summit on Advances in Programming Languages (SNAPL 2015). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Findler, R. B. (2014). DrRacket: The Racket Programming Environment. Racket Language Documentation.

Torino, G. C., Rivera, D. P., Capodilupo, C. M., Nadal, K. L., & Sue, D. W. (Eds.). (2019). Microaggression theory: Influence and implications. John Wiley & Sons. https://doi.org/10.1002/9781119466642