

Documentación

# Tarea 1 - Paradigma Funcional

Curso: Lenguajes de programación

---

Estefanny Villalta Segura

Sebastián Hidalgo Vargas

Valeria Morales Alvarado

19 de mayo de 2023


## Descripción de los algoritmos de solución desarrollados

### **PDC-sol**

Para la función PDC- sol se utilizó el algoritmo de Warnsdorff. Luego de realizar la investigación respectiva del funcionamiento y condiciones del problema del caballo se encontraron dos posibles algoritmos que resultaron llamativos para la implementación de la presente tarea, Backtracking y el algoritmo de Warnsdorff. De los estos se seleccionó el segundo, ya que el backtracking podía resultar poco eficiente en casos de tableros grandes.

La idea principal del algoritmo de Warnsdorff es visitar primero las casillas con menos movimientos posibles, esto con el propósito de evitar casillas insaladas, lo que a su vez, evita caminos sin salida. Para la implementación de este algoritmo se siguieron los siguientes pasos:

1. Iniciar la ruta con la casilla inicial seleccionada por el usuario
2. Mientras no se hayan visitado todas las casillas
  - a. Encontrar posibles movimientos desde la posición actual, estos deben satisfacer la condición de movimiento del caballo, deben estar dentro del tablero y no pueden haber sido visitadas antes.
  - b. Para cada uno de los posibles movimientos, encontrar el número de casillas adyacentes o casillas accesibles
  - c. Ordenar posibles movimientos según este número, de manera ascendente
  - d. Se selecciona el movimiento con menor número como siguiente movimiento
  - e. En caso de que no haya ningún movimiento posible: retroceder al movimiento anterior y seleccionar la siguiente casilla con menor número. Si se retrocede hasta el movimiento inicial, no hay solución.
3. Una vez se completó el tablero, se encontró la solución.


	8		1	
7				2
				
6				3
	5		4	

### PDC-test

Para realizar la verificación de una matriz solución se optó por realizar dos verificaciones principales las cuales son primordiales para determinar si es una solución o no. La primera es que el tablero o matriz solución debe ser del tamaño declarado por el usuario, de lo contrario no se recorrieron todas las casillas por lo que no sería una solución. La segunda es que para cada movimiento la casilla siguiente debe estar dentro de los posibles movimientos de la misma, por lo que debe seguir el movimiento del caballo, debe estar dentro de los rangos del tablero y no puede haber aparecido previamente en la solución. Si se recorre todo el tablero cumpliéndose esta condición significa que es una solución del problema.

### PDC-Todas

Esta función se encarga de calcular 5 posibles recorridos distintos para el problema del caballo. Esta implementa la técnica algorítmica de backtracking para obtener dichas soluciones ya que verifica todos los caminos posibles dada una coordenada inicial específica. La función principal recurre a una función auxiliar que busca validar que el primer elemento de la lista de movimientos posibles tenga más futuros movimientos, y se invoca recursivamente con los posibles movimientos generados a partir de esta coordenada. En cuanto la primera coordenada no tenga más oportunidades de avanzar, se ejecuta la recursión con el objetivo de encontrar los posibles movimientos de los demás elementos de la lista. Al llamar una nueva posición, esta se añade a la lista de movimientos ya recorridos, y para terminar el ciclo recursivo de cada solución, se comprueba que el largo de la lista solución sea igual al tamaño N del tablero, lo que implica que ya se recorrió todas las posiciones de este. Adicionalmente, debido a la



complejidad en tiempo de ejecución del backtracking, se definió una restricción adicional de un límite tiempo para realizar un *timeout* al transcurrir el límite y así prevenir un *overflow* o un tiempo de ejecución excesivo ya que sólo se necesita el cálculo de 5 soluciones.

### **PDC-Pintar**

Esta función contiene dos parámetros de los cuales uno es utilizado para formar el tablero y otro para llevar la solución del usuario. También está encargada de generar el canvas donde se verán la soluciones al problema. Para generar el tablero se usa la entrada del usuario y así generar la matriz sobre la cual podrá ser observada la solución, para generar la solución se utiliza el resultado proveniente de PDC-Test, el cual es un lista con pares ordenados los cuales forman la ruta del caballo.

## Descripción de las funciones implementadas

### **PDC-sol**

quicksort: Ordenar matriz de posibles movimientos dependiendo del número de vecinos (celdas accesibles) de cada uno de manera ascendente.

neighbour-count: Determina el número de casillas accesibles de un posible movimiento


last-element: retorna el último elemento de una lista, en ese caso sería correspondiente al último movimiento realizado

possibleMoves: genera una lista con los posibles movimientos desde una posición en el tablero

validate: verifica los movimientos válidos en términos de posición y elimina aquellos cuya posición adquiere valores fuera del tablero

visited: elimina movimientos que ya han sido visitados dentro de los posibles movimientos desde una posición, de manera que no se puede pasa por la misma casilla dos veces

miembro: determina si un elemento se encuentra dentro de una lista



getSize: retorna el tamaño de una lista o el número total de elementos en una lista

solutionBoard: toma la matriz solución por posiciones generada de los algoritmos anteriores y genera la matriz solución representativa del tablero.

printMatrix: hace el display en consola de una matriz

### **PDC-test**

getMov: a partir de una matriz solución se obtiene la matriz de movimientos secuenciales posición por posición

getPos: para un elemento en una matriz, retorna la posición en el que se ubica dicho elemento. Las posiciones adquieren valores de fila y columna partiendo de 1.

searchRow: busca un elemento en una fila, si lo encuentra, retorna la columna en la que se encuentra el elemento, de lo contrario retorna cero.

### **PDC-Todas**

printTodas: realiza un display de las 5 soluciones obtenidas

getFive: obtiene los primeros 5 elementos de una lista

inBoard?: verifica si una posición está dentro de los límites tablero

euclidean: calcula la distancia euclidiana entre dos puntos

contains: verifica si un elemento está dentro de una lista

neighbour: verifica si un movimiento es permitido

solution?: verifica si una solución es válida

validInput?: verifica si los parámetros de la función son válidos

getSteps: utiliza backtracking para resolver el problema de obtener 5 soluciones diferentes

deletePaths: elimina caminos ya explorados

moveScope: crea una lista con los movimientos que cumplen las condiciones correctas

## PDC-Pintar

**adjustSol:** esta función se encarga de multiplicar el valor de los números provenientes de **getMov** de modo que se acomode en un posición en específico del canvas.

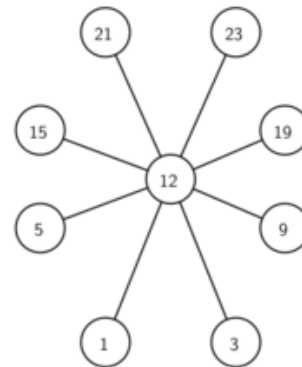
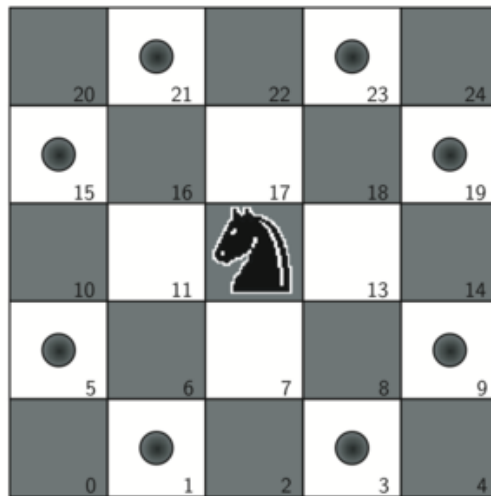
**animation:** esta función se encarga de dibujar la solución sobre el tablero de ajedrez por medio de una línea la cual tiene sus valores definidos en los ejes XY respectivamente para cada punta de la línea.

## Estructuras de datos utilizadas

**Listas:** Las listas se utilizaron principalmente en esta función para representar pares ordenados representativos de una posición en el tablero de ajedrez.

**Matrices:** Se utilizaron como listas de listas para almacenar una secuencias de posiciones. Las matrices que se destacan son: la matriz de movimientos realizados, que contiene el camino realizado para llegar a la solución; la matriz que almacena los posibles movimientos partiendo de una posición, la cual sirve para seleccionar el próximo movimiento y la matriz solución, la cual contiene una representación del tablero donde cada elemento tiene un número que corresponde a su posición dentro del camino que sigue la solución.

**Grafos (implícito):** La combinación de algunas de las estructuras de datos mencionadas anteriormente resultan en la aplicación implícita del grafo. Esto de manera que la matriz de posibles movimientos sería una serie de nodos accesibles por un nodo inicial que sería la posición actual. Una vez se selecciona uno de los posibles movimientos este se agrega a la matriz de movimientos realizados, que a su vez podría interpretarse como el camino que siguen las aristas del grafo, tal que conecta los nodos según se van seleccionando los movimientos.



## Problemas sin solución

En relación con la interfaz gráfica a pesar de tener una función recursiva para las líneas con un delay para poder ver el modelo de solución por medio de trazas no se logró el objetivo de hacer la solución animada, solo aparece la solución completa.

## Problemas encontrados

Uno de los problemas encontrados se relaciona con la función PDC-todas. La idea inicial para encontrar 5 diferentes soluciones era utilizar como base la función PDC-sol, de manera que se encontrara una manera de modificarla para que el algoritmo tomará diferentes caminos. Debido al algoritmo diseñado para obtener la solución del problema del caballo, al llamar PDC-sol el resultado siempre sería el mismo. Por lo que este era el problema, se requería encontrar un algoritmo auxiliar que modificara el camino que tomaba la solución.

Se intentó variar el criterio de selección del algoritmo para que no eligiera exclusivamente la siguiente casilla con el menor número de vecinos (posibles movimientos válidos) sino que eligiera una aleatoria. Sin embargo este enfoque ocasionó recorridos inválidos.

Al modificar el algoritmo base de PDC-Todas e implementar el backtracking se calculaban todas las soluciones para el recorrido sin embargo esta técnica algorítmica consumía altos recursos computacionales y su tiempo de ejecución no era el óptimo por

lo que se decidió limitar el tiempo de cálculo para que, al exceder el límite, terminara la recursión y se quedará con las soluciones que encontró hasta el momento ya que solo nos interesaba las primeras cinco.

## Plan de actividades

### Valeria Morales Alvarado

8/05/2023: Tuvo lugar la primera reunión relacionada al desarrollo de la tarea, durante está se distribuyeron los quehaceres principales. El área de trabajo asignada fue lógica de la tarea.

9-12/05/2023: Se asignaron estos días para investigar el problema del caballo y diferentes formas de solucionarlo, así como algoritmos relacionados. Se planteó explorar las opciones más viables iniciando la implementación de las mismas en Racket.

12/05/2023: Se asignó la fecha para trabajar en la función PDC-sol. Se busca investigar posibles implementaciones y algoritmos para solucionar el problema del caballo. A su vez se busca al menos iniciar o realizar pruebas de los algoritmos encontrados para determinar cuál es el más conveniente para obtener la solución.

Luego de realizar la investigación correspondiente se seleccionó de primeras el algoritmo de Warnsdorff para iniciar la implementación de la función PDC-sol. Para realizar esta implementación se realizó el siguiente pseudocódigo:

1. Crear tablero de tamaño size x size con todas las casillas sin visitar
2. Marcar la casilla (x, y) como visitada y agregarla al camino -> 1
3. Mientras no se hayan visitado todas las casillas:
  - a. La lista de movimientos está vacía? -> No hay solución, fin del tour
  - b. Crear lista de casillas adyacentes a la casilla actual (x, y) que no han sido visitadas
  - c. Si la lista está vacía -> retroceder al movimiento anterior:
    - i. Eliminar último movimiento de lista de movimientos y de posibles movimientos
    - ii. Seleccionar nuevo elemento (pos 0)
    - iii. Repetir paso 3



- d. Hay solo un posible movimiento -> incluir como siguiente movimiento, repetir el paso 3
- e. Calcular el número de jugadas posibles para cada movimiento en la lista
- f. Ordenar lista de posibles movimientos según el número
- g. Seleccionar elemento con menor número -> repetir paso 3
- h. Hay varias con el mismo número -> seleccionar una al azar (?)

Finalmente se logró implementar con éxito la función, de manera que se verificó que retorna correctamente una secuencia de movimientos que resuelve el problema del caballo.

13/5/23: Se propuso una reunión para revisión de la parte lógica. Según la división de tareas realizadas la próxima función a implementar es PDC-todas.

15/5/23: Se asignó el día para el desarrollo de la función PDC-todas, para esto se buscaron diferentes maneras de llegar a soluciones diferentes, sin embargo no hubo éxito.


16/5/23: Se asignó el día para buscar maneras de implementar PDC-todas, sin embargo no se tuvo éxito. En lugar de esto, mediante la comunicación por Whatsapp se llegó al acuerdo de intercambiar tareas para adquirir nuevas perspectivas en la solución de las funciones faltantes. La nueva tarea asignada para el día era la solución de la función PDC-test. Para dicha función se optó por realizar una serie de verificaciones esenciales utilizando funciones implementadas en el desarrollo de PDC-sol. Finalmente, se pudo crear de forma preliminar la función PDC-test.

17/5/23: Se asignó trabajar en la documentación de la tarea, el manual de usuario y la bitácora.

18-19/05/2023: Se asignaron estos días para resolver los problemas relacionados a la función PDC-todas y asimismo avanzar con la documentación.

### **Sebastián Hidalgo Vargas**

6/05/23: Se comenzó por estudiar el álgebra de las trayectorias posibles que el caballo podía desplazarse, y se encontró la restricción de que si no se toman en cuenta los bordes del tablero, podrían generarse movimientos inválidos. De igual manera, se intentaron sentar las bases teóricas detrás del problema con el fin de contar con el suficiente entendimiento para encontrar la solución.



7/05/23: Retomando del día anterior, se halló que existen dos tipos de tour, el abierto y el cerrado. Se analizaron las consideraciones algebraicas del tablero de Ajedrez, al mismo tiempo que se tomó en cuenta las propiedades del Grafo del Caballo y se estudiaron algunos enfoques de solución.

14/05/23: Se desarrolló en pseudocódigo algunas aplicaciones de solución de previo al desarrollo del código.

15/05/23: Se intentó utilizar el código de PDC-Sol como base para poder resolver el PDC-Todas, sin embargo se encontró el problema de que al variar los criterios de selección del algoritmo inducía a soluciones erróneas o bien no se resolvían del todo por lo que se cambió el enfoque.

18/05/23: Se decidió utilizar backtracking para la generación de las cinco soluciones. Se encontraron problemas de eficiencia debido al alto consumo de recursos computacionales.


19/05/23: Se resolvió el problema del tiempo de ejecución delimitando un intervalo de tiempo para que al excederse este, se pare la recursión y se mantengan solo las soluciones que el algoritmo calculó hasta ese preciso momento ya que solo se requieren cinco soluciones.

## **Estefanny Villalta Segura**

7/05/23: se dió inicio a la investigación del problema del caballo para tener una visión más amplia sobre como se observaba esta, además se inició con la investigación sobre la librería de la interfaz de racket.

8/05/23: primera reunión grupal donde se conversó sobre el problema del caballo a partir de la investigación previa además se repartieron las tareas y funciones dentro del equipo de trabajo.

11/06/23: se buscó la forma de armar una interfaz gráfica con la librería gráfica de racket para tener una visión más amplia sobre la estructura de la misma.



13/06/23: se dio inicio a la formación del tablero de ajedrez en la interfaz gráfica, se necesito hacer ajustes en las coordenadas del canvas así como buscar una forma la cual permita formar tableros pares e impares pero con la misma estructura. Este día también tuvimos una reunión grupal para ver el trabajo de cada uno.

16/06/23: este día se convocó a otra reunión grupal donde se habló principalmente de los avances de cada integrante así como conversar sobre el trabajo escrito que debíamos entregar una vez terminado el tablero, se dio inicio a la línea que debe trazar para generar la solución, esta línea necesitaba un par de pares ordenado para formar la ruta de solución deseada sobre el tablero.

18/06/23: última reunión con la integrante del grupo Valeria para poder ver el trabajo completo funcionando.

## Conclusiones

El problema del caballo es bien conocido por ser un problema complejo y difícil de resolver debido a la cantidad de posibles movimientos y recorridos que pueden surgir por la naturaleza del caballo de ajedrez, dificultando la obtención de una solución óptima. Sin embargo, gracias a los algoritmos y enfoques presentados, fue posible encontrar soluciones apropiadas para el acertijo.

De la misma manera, la metodología propuesta para encontrar la resolución del problema implementa la heurística de Warnsdorff, y para seleccionar la mejor casilla candidata usa las propiedades implícitas de los grafos y algoritmos de ordenamiento y la aplicación de métodos algorítmicos como lo es el backtracking y así asegurar un comportamiento determinístico a la hora de seleccionar las casillas del recorrido.

Con esto se concluye que se cumplió el objetivo de la tarea, ya que se exploraron diferentes conceptos clave del paradigma de programación funcional, así como su utilización en la manipulación de aplicación de estructuras de datos a este. Además, estos conceptos se utilizaron para implementar de manera efectiva el programa propuesto para la solución del problema del caballo en el ajedrez.

## Recomendaciones

Se recomienda a cualquier persona interesada en resolver este milenario problema, estudiar detenidamente las propiedades algebraicas del tablero de ajedrez, la teoría de grafos involucrada en los posibles movimientos del caballo al ocupar una posición cualquiera, así como también heurísticas planteadas como la regla de Warnsdorff y la viabilidad de algoritmos recursivos como el backtracking y sus implicaciones en la complejidad en tiempo de ejecución, o alternativas de enfoque como la son las redes neuronales para atacar y resolver exitosamente este rompecabezas.

## Bibliografía

Miller, B., Ranum, D. (2006). Python for Everybody. Luther College. Capítulo 7.12: Construcción del grafo de la gira del caballo. Recuperado de <https://runestone.academy/ns/books/published/pythoned/Graphs/ConstruccionDeIGrafoDeLaGiraDelCaballo.html>

Alvarez, D. Romero, R. (2012). ALGORITMO DETERMINÍSTICO PARA RESOLVER EL PROBLEMA DEL TOUR ABIERTO DEL CABALLO SOBRE TABLEROS DE AJEDREZ . Recuperado de <http://www.din.uem.br/sbpo/sbpo2012/pdf/arq0509.pdf>

GeeksforGeeks. (2021). The Knight's Tour Problem. Recuperado de <https://www.geeksforgeeks.org/the-knights-tour-problem/>

GeeksforGeeks.. (2021). Warnsdorff's Algorithm for Knight's Tour Problem. Recuperado de <https://www.geeksforgeeks.org/warnsdorffs-algorithm-knights-tour-problem/>

Racket. (n.d.). GUI [Online documentation]. Retrieved May 19, 2023, from <https://docs.racket-lang.org/gui/>