TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN
ÁREA ACADÉMICA DE INGENIERÍA EN COMPUTADORES
LENGUAJES, COMPILADORES E INTÉRPRETES (CE-3104)

Tarea 1
PDC

Realizado por:
OLMAN CASTRO Hernández - 2016003915
PABLO MORA Rivas - 2017114091
EDUARDO MOYA BELLO - 2015096664

Profesor:
Ing. Marco Rivera

Cartago, 14 de mayo de 2019

Índice

Manual de usuario

Introducción

El programa PDC se encarga de resolver el problema del caballo, el cual es un antiguo problema matemático. Teniendo una cuadrícula de n x n casillas y un caballo de ajedrez colocado en una posición cualquiera (x, y) el caballo debe pasar por todas las casillas y una sola vez (Hernández, 2019).

Pasos previos antes de la ejecución

Para poder ejecutar el programa en su computador, primeramente debe de cerciorarse de que Drracket se encuentre instalado en la misma, de no ser así, se debe proceder a descargar desde la página oficial (`https://racket-lang.org/download/`) y seguir los pasos de instalación propuestos por la misma.

Seguidamente, se debe descargar el repositorio de Github (`https://github.com/papmora/Tarea1`) donde se encuentran almacenados los diferentes archivos ejecutables.

Luego de tener los archivos en su computador, debe de abrirlos en Drracket; ya se desde el menú interno (File/Open...) o con un doble click sobre el ejecutable.

Seguidamente, debe correr el programa (ejecutarlo) desde el botón verde que aparece en la esquina superior derecha. Si, se desea correr el archivo PCD-Paint, debe seguir los mismos pasos, pero al finalizar debe detener el ciclo de ejecución en el botón stop.

Finalmente, debe el nombre de la función y parámetros necesarios para su ejecución (explicados más adelante) en la consola (casilla inferior) y presionar la tecla Enter para obtener los resultados del programa.

Cómo ejecutar el programa

El programa está realizado en el lenguaje de programación Scheme y cuenta con cuatro funciones principales:

```
(PDC-Sol n pos)
(PDC-Todas n pos)
(PDC-Test n sol)
```

```
(PDC-Paint n sol)
```

PDC-Sol es una función que recibe un entero que dice el tamaño de la matriz (n) y una posición y un par ordenado (pos) que indica dónde se encuentra inicialmente el caballo. Por ejemplo:

```
> (PDC-Sol 5 '(4 4))
(list (list 4 21 16 11 6) (list 15 10 5 22 17) (list 20 3 24 7 12)
(list 9 14 1 18 23) (list 2 19 8 13 0))
```

PDC-Todas es igual que PDC-Sol, pero devuelve todas las soluciones posibles para el n definido.

PDC-Test es una función que recibe dos argumentos, un entero que indica el tamaño de la matriz y una estructura solución. Indica si es correcta o no mediante un booleano. Por ejemplo:

```
> (PDC-Test 5 '((4 21 16 11 6) (15 10 5 22 17) (20 3 24 7 12) (9
14 1 18 23) (2 19 8 13 0)))
#true
> (PDC-Test 5 '((0 21 16 11 6) (15 10 5 22 17) (20 3 24 7 12) (9
14 1 18 23) (2 19 8 13 4)))
#false
```

PDC-Paint recibe un entero que indica el tamaño de la matriz y una estructura solución, la idea es probar la solución encontrada pero utilizando la librería Racket Graphical Interface Toolkit y representarla de una forma gráfica/animada.

```
> (PDC-paint 5 '((4 21 16 11 6) (15 10 5 22 17) (20 3 24 7 12) (9
14 1 18 23) (2 19 8 13 0)))
> (PDC-paint 8 '((0 59 38 33 30 17 8 63) (37 34 31 60 9 62 29 16)
(58 1 36 39 32 27 18 7) (35 48 41 26 61 10 15 28) (42 57 2 49 40
23 6 19) (47 50 45 54 25 20 11 14) (56 43 52 3 22 13 24 5) (51 46
55 44 53 4 21 12)))
```

Funciones utilizadas

PDC-Sol

Esta función recibe dos argumentos un entero que indica el tamaño de la matriz

y un par ordenado que indica donde se encuentra inicialmente el caballo. Para un máximo de matriz 10x10 comprobado con pruebas, de ejecución rápida. No se continuó la comprobación para matrices más grandes; y como un mínio una matriz 5x5.

Como resultado debe retornar una estructura (a definir por el estudiante) con una solución de la secuencia de casillas que el caballo va visitando.

```
>(PDC-Sol 8 '(1 1))
(solución)
```

Para realizar esta función, se ocupan de varias funciones auxiliares, principalmente de (`PDC-Sol-aux`), pero también ocupa varias para el manejo de matrices, entre ellas podemos mencionar:

- (`imprimir`): Función para imprimir matriz, esta función básicamente recibe la matriz que se desea imprimir en consola y mientras no sea nula la devuelve.
- (`crear-matriz`): Función para crear una matriz con todos los valores que se encuentran en su interior iguales a 0. Esta función recibe una matriz y el tamaño (n*n) que desea el usuario que tenga la matriz y mientras n sea diferente de 1, se llama a (`crear-matriz-aux`)
- (`crear-matriz-aux`): Esta función básicamente llena la matriz de 0's recorriendo toda la matriz y usando la función (`lista-ceros`), hasta que el contador que es igual a 'n' sea 0.
- (`lista-ceros`): Al igual que la función anterior con ayuda de un contador llena una lista con 0's.
- (`obt-ele-matriz`): Esta función es de vital importancia, pues nos permite obtener un elemento específico de la matriz a partir de la fila y la columna asignada. Esta función recorre las listas y con ayuda de las funcione (`obt-ele-lista`) obtiene el elemento en la matriz.
- (`obt-ele-lista`): Esta función nos permite obtener un elemento específico de una lista, recorriendola con un contador hasta que el contador sea 0.
- (`cambiar-ele-lista`): Esta función recorre la lista hasta encontrar el elemento solicitado y cambia el valor sin modificar los otros elementos dentro de la lista.

- (`cambiar-ele-matriz`): Esta función recorre la matriz por listas y con ayuda de la función (`cambiar-ele-lista`) encuentra el elemento solicitado y cambia el valor sin modificar los otros elementos dentro de la matriz.

Todas estas funciones en conjunto me permite utilizar correctamente la función (`PDC-Sol-aux`), que verifica si es el primer valor para colocarlo, si el flag es 1 para imprimir la matriz, si los movimientos son válidos y en caso que se cumlpla todo llama recursivamente a la funcion sobreescribiendo los valores de la matriz para dar la resultante, sino cumple el movimiento válido este aumenta un contador para verificar si otro movimiento le permite lograrlo.

PDC-Todas

Al igual que la función anterior recibe dos parámetros, un entero que indica el tamaño de la matriz y un par ordenado que indica donde se encuentra inicialmente el caballo y retorna todas las soluciones posibles. La solución por defecto a este algoritmo es una matriz 5x5, sin embargo puede resolver una matriz aún mayor por ejemplo 6x6, pero su ejecución es bastante lenta, ya que alrededor de unas 5 minutos esta función no a terminado de encontrar todas la soluciones.

```
>(PDC-Todas 8 '(1 1))
((solución 1)
(solución 2)
.....
.....
(solución N))
```

Esta función realiza lo mismo que la anterior, solo que utiliza un backtracking para encontrar más de una solución y por lo tanto encontrar todas las posibles.

PDC-Test

Validaciones

Se deben realizar varias validaciones para tener certeza de que la solución es correcta. La primera es que sea una matriz cuadrada NxN, por lo que la función `is-nxn` se encarga de verificar que el tamaño de la matriz (número de filas) sea igual al número de columnas. La función recibe una matriz y un n a verificar, y retorna un booleano. Para esto se requieren dos funciones auxiliares:

- `len`: recibe una lista y retorna la cantidad de elementos en ella (sumando 1 cada vez que se realiza un `cdr` mientras la lista no esté vacía).
- `rows-n`: es una función que recibe una matriz y verifica que todas sus filas sean del mismo tamaño (n).

Además, se debe validar que los elementos de la matriz no se repitan, por lo que se creó la función `not-repeated`, que llama a su función auxiliar `not-repeated-aux` la cual busca que haya un único número x en un intervalo de 0 a $n^2 - 1$.

Otra validación esencial es que el movimiento realizado sea aceptado como movimiento de caballo, es decir, que forme una letra L en la cuadrícula. De esto se encarga la función `l-move` la cual recibe dos pares ordenados y verifica que tengan una forma de L.

Funciones adicionales

Una función muy importante en `PDC-Test` es `pos` la cual indica la posición de un elemento de una matriz. La función recibe una matriz y un elemento y retorna el par ordenado de donde se encuentra dicho valor. Se utilizan tres funciones auxiliares:

- `pos-in-list`: indica la posición (índice) de un elemento en una lista.
- `pos-in-list-aux`: función auxiliar que permite la recursividad recibiendo la posición inicial de la lista (0 por defecto). Se va recorriendo la lista mediante el uso del `cdr` a medida que se aumenta la posición. Cuando ambos elementos son iguales, se retorna la posición. Si la lista se vacía, significa que no está el elemento por lo que se decidió retornar un -1.
- `pos-aux`: función auxiliar de `pos`. Verifica que el elemento se encuentre en las filas de la matriz mediante la función integrada `member`, y obtiene su

posición mediante `pos-in-list`. Si la matriz está vacía o no se encuentra el elemento, retorna (-1, -1).

Función final

La función final de `PDC-Test` cuenta con la función base y una auxiliar. La función base se encarga de validar que sea una matriz NxN (`is-nxn`) y que no se repitan elementos (`not-repeated`: cabe reiterar que también valida que haya un único elemento para cada posición de la matriz). La función auxiliar (`pdc-test-aux`) verifica que x y $x+1$ formen una L (o un movimiento de caballo válido.

PDC-Paint

Esta función es la encargada de dibujar el proceso del caballo en un tablero de ajedrez con el fin de poder tener una representación gráfica de lo que ocurre a fondo en el sistema permitiendo ejecutar como mínimo una matriz 5x5 y máximo una matriz 10x10, no por el hecho que el algoritmo no funcione con matriz más grandes, sino por relaciones de aspecto gráfico y calidad de visualización para el usuario .

La primer función ejecutada es `(pdc-test largo matriz)` explicada en el punta anterior, encargada de verificar que la matriz sea del tamaño especificado y que la matriz sea una solución correcta del caballo, de no ser así, el sistema automáticamente retorna falso y no muestra ningún proceso gráfico.

Seguidamente `(pintar 0 0 0 largo ventana mapa-pixeles)` mediante la segunda función ejecutada llamada pintar, el sistema se encargada de dibujar un tablero de ajedrez en donde se realizarán los movimientos del caballo, y además, dimensiona la ventana presentada al tamaño necesario del tablero una mejor visualización del proceso.

Una vez verificada que la matriz es solución, el sistema ejecuta la función `(juego matriz ventana mapa-pixeles 0 largo)`, en donde se ingresan los parámetros ventana (ventana gráfica donde se debe ir dibujando), mapa-pixeles (mapa de ubicación de los pixeles en la ventana), “0” (posición de inicio a comprobar en la matriz) y largo (tamaño de la matriz).

Esta función es el ciclo principal del juego, encargada mediante las funciones `(buscar-row elemento matriz '() 0)` y `(buscar-colum elemento matriz`

`'() -1)`, que revisen: elemento (movimento de caballo a buscar), una lista vacía para guardar los elementos y la posición en donde se debe iniciar la búsqueda.

Las funciones, son encargadas de ubicar en qué coordenadas de la matriz se encuentra el siguiente movimiento del caballo.

Finalmente, `(caballo (buscar-row elemento matriz '() 0) (buscar-colum elemento matriz '() -1) largo ventana mapa-pixeles)`la función caballo, se pinta en el tablero los movimientos de del caballo en el tablero, ingresando las coordenadas, el largo de la matriz, la ventana y el mapa de pixeles; realizando las comprobaciones necesarias para utilizar la función `(draw-solid-rectangle mapa-pixeles) (make-posn a b) 71 71 "yellow")` para dibujar el caballo.

Luego de este proceso, la función continua en un ciclo hasta resolver el caballo y seguidamente cerrar la ventana.

Estructuras de datos utilizadas

Listas

Las listas son un conjunto de elementos ordenados. En Scheme, se escriben de la siguiente manera:

```
'(1 2 3 4 5)
```

Matrices

Una matriz no es más que una lista de listas. En el caso particular del problema del caballo, las matrices son de tamaño NxN (es decir, son cuadradas) por lo que la cantidad de elementos de la lista debe ser igual que la cantidad de elementos de las sublistas. Por ejemplo:

```
'((1 2) (3 4)) ;Matriz 2x2
'((1 2 3) (4 5 6) (7 8 9)) ;Matriz 3x3
'((1 2 3 4) (5 6 7 8) (9 10 11 12) (13 14 15 16)) ;Matriz 4x4
```

Algoritmos detallados

Como el paradigma que se trabaja en este caso es el funcional, los algoritmos detallados se encuentran explicados anteriormente en la sección de funciones utilizadas, ya que todos los algoritmos son funciones.

Problemas conocidos

Función index-of

Se investigó la existencia de una función integrada de Scheme llamada *index-of* para encontrar el índice de un elemento en una lista, pero en DrRacket se encontró el error de que la función no estaba definida.

Asimismo encontramos un problema con la realización de las primeras dos funciones, pues la manera en la que se pensó realizar las funciones tenían unas limitantes con el backtraking, por lo tanto para correr estas dos funciones se tiene que tener racket como lenguaje y no Advanced Student, esto según información que encontramos en internet.

Actividades realizadas por estudiante

Días trabajados	Eduardo Moya	Pablo Mora	Olman Castro
3 de mayo 2019	0h	4h	0h
6 de mayo 2019	0h	5h	0h
8 de mayo 2019	1h	4h	0h
10 de mayo 2019	0h	0h	4h
11 de mayo 2019	9h	0h	Días
13 de mayo 2019	1.5h	3.5h	3.5h

14 de mayo 2019	0h	3h	9h
15 de mayo 2019	0h	3h	3h
Total de horas	11.5h	22.5h	14.5h

Tabla 1. Horas laboradas por estudiante

En la tabla anterior, se muestran las horas y días en que los estudiantes trabajaron en el proyecto; al final de la misma se puede ver el total de horas trabajadas.

Problemas encontrados

Mala lectura

Aunque no sea un error directamente, uno de los errores encontrados fue una mala comprensión del problema (mala lectura) por lo que dificulto el trabajo en el área de la interfaz gráfica, retrasando la conclusión del trabajo en el tiempo establecido.

Organización

Otro de los problemas, fue una mala distribución del tiempo y, por otra parte, no establecer fechas anticipadas para generar avances del proyecto.

Conclusiones y Recomendaciones

Conclusiones

Se considera que este problema es más fácil tratarlo con algún otro lenguaje, ya que según el integrante Mora, "el manejo de matrices tuvo sus momentos de confusión y era más fácil el recorrer una matriz con dos bucles *for*".

Los problemas encontrados relacionados con la distribución de tareas y el manejo del tiempo demuestran que el grupo de trabajo debe reforzar habilidades blandas tales como organización y buena comunicación, ya que en el ámbito profesional ambos aspectos son indispensables.

Recomendaciones

Cómo se mencionó anteriormente se recomienda tratar este problema con otros lenguajes. Sin embargo si desea resolver este problema con algún lenguaje funcional como DrRacket, se recomendaría con el lenguaje Racket y no Advanced Student si desea resolverlo sin problemas para realizar más de una acción en un condicional. En caso contrario puede separar el problema en funciones más pequeñas, pues fue lo que se consideró cerca de la entrega.

Referencias bibliográficas

Racket. Recuperado de https://docs.racket-lang.org/reference/pairs.html

Geeksforgeeks (2013, 2019). Sector-136, Noida, Uttar Padres-201305. Recuperado de https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-1/

Geeksforgeeks (2013, 2019). Sector-136, Noida, Uttar Padres-201305. Recuperado de https://www.geeksforgeeks.org/warnsdorffs-algorithm-knights-tour-problem/

Anexos

Anexo 1: Bitácora digital

Bitácora 1

Fecha: sábado 11 de mayo de 2019

Participante(s): Eduardo Moya

Asunto: Elaboración de PDC-Test

Descripción detallada: Se comenzó a la 1:08 p.m. con el desarrollo de la función PDC-Test. Debido a conversaciones informales que se tuvieron con los compañeros del equipo después del horario de clases, se tiene una idea de cómo proceder a encontrar una solución. Se empieza a crear una función, pero después de varios minutos se llega a la conclusión de que es más efectivo dividir el problema en varias funciones. Se inició con las validaciones.

Se procede a realizar una función que verifica que una matriz sea NxN (is-nxn). Se encuentra el problema de necesitar el tamaño de las filas y columnas por lo que se

crea una función que lo soluciona. Se observa que la función is-nxn tiene una parte recursiva (ir verificando todas las filas de la matriz), por lo que se crea una función auxiliar. Asimismo, se crea una función que indica la posición de un elemento en la matriz. Se investigó la existencia de una función integrada de Scheme llamada *index-of* pero no funcionó en DrRacket. Para saber la posición de un elemento en una matriz también se necesita saber la posición de un elemento en una lista por lo que se crea una función para tal fin.

Continuando con las validaciones, se analiza alguna manera para validar que la solución propuesta sea la correcta. Además de que sea una matriz sea NxN, los elementos no se pueden repetir y tienen que ser todos diferentes, por lo que se crea una función que solucione este problema (not-repeated). También, todos los movimientos tienen que ser válidos, por lo que deben formar una letra L, entonces se crea una función que lo valide. Para realizar esta función, fue necesario tener una ayuda visual, ya que se torna muy complejo mediante el uso de *car* y *cdr*:

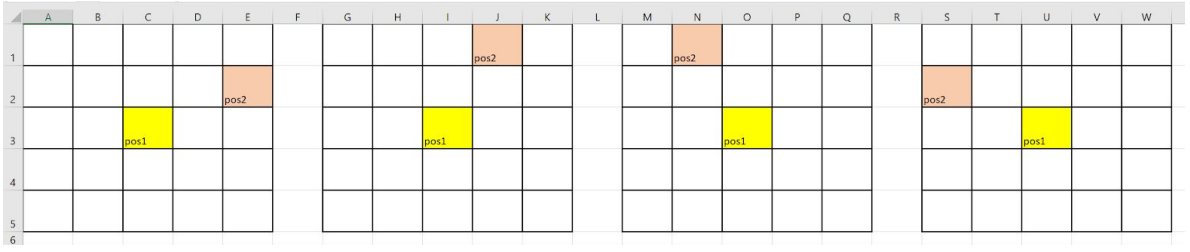

Figura 1: Ayuda visual de los posibles movimientos correctos. Realizado en Excel. Se muestran 4 de 8 soluciones

Posteriormente se unen todas las validaciones y se realiza la función final. De esta sesión, la mayoría de problemas fueron sintácticos en el manejo de paréntesis. Fue de gran ayuda el uso de paréntesis cuadrados. Se finalizó la sesión a 9:48 p.m. No hubo pausas durante el tiempo de trabajo.

Bitácora 2

Fecha: viernes 10 de mayo de 2019

Participante(s): Olman Castro

Asunto: Investigación de la interfaz gráfica

Descripción detallada: Se comenzó a las 9:00 pm la investigación de cómo realizar una interfaz gráfica en Dr Racket buscando en el repositorio oficial de Racket, se logra obtener como manejar eventos, crear ventanas, botones, dibujar en pantalla y

dibujar un tablero de ajedrez con los eventos y funciones de la interfaz; se finaliza a la 1:00 am del siguiente día.

Bitácora 3

Fecha: lunes 13 de mayo de 2019

Participante(s): Olman Castro

Asunto: Creación de interfaz gráfica

Descripción detallada: Se inició a trabajar a las 10:00 pm, creando la interfaz gráfica y sus eventos, se logra pintar en pantalla el caballo y que reaccione a la hora de tocar una casilla para que pueda verse el progreso del pintado, se crean los métodos para poder incorporar el tamaño de la matriz, la posición de inicio del caballo y la matriz a utilizar. Se finalizó a la 1.30 am del siguiente día.

Bitácora 4

Fecha: martes 14 de mayo de 2019

Participante(s): Olman Castro

Asunto: Funcionamiento de la interfaz

Descripción detallada: Se inició a trabajar a las 7:00 am, se corrigen errores en la interfaz, se crea una ventana con condiciones específicas dependiendo del tamaño de la matriz, se implementan los algoritmos para el movimiento del caballo y se crea una ventana para la recibir los datos ingresados por el usuario, se descubre error a la hora de la lectura del problema que debe ser entonces corregido, el cual, demanda la mayoría del tiempo, ya que se debe cambiar la lógica completa del programa. Se inician los cambios en el la lógica, eliminando la ventana de recepción de datos e intentando unir el PDC-paint con el PDC-test. Se finaliza a las 4:00 pm de ese día

Bitácora 5

Fecha: miércoles 15 de mayo de 2019

Participante(s): Olman Castro

Asunto: Conclusión del programa

Descripción detallada: Se inició a trabajar a las 10 pm, se une la función PDC-test a el PDC-paint, se pinta el caballo con la forma lógica de la matriz, seguidamente se

optimiza el código para un mejor funcionamiento, y se documenta por completo el código. Se finaliza a la 1:00am de la siguente día.

Bitácora 6

Fecha: viernes 3 de mayo de 2019

Participante(s): Pablo Mora

Asunto: investigación sobre funciones necesarias para las primeras 2 funciones

Descripción detallada: Se inició a trabajar a las 1 pm, con el fin de visualizar en papel, de qué manera podría resolver este problemas en DrRacket, luego de algunas investigaciones en internet consideré que la manera más sencilla era seguir un proceso similar al de GeeksForGeeks que resolvía el mismo problema usando backtraking en JAVA. Ese mismo día concluí que tenía que desarrollar funciones básicas para manejar matrices en DrRacket. Término a las 5 pm con esta tarea y decido realizar esas funciones básicas para matrices otro día.

Bitácora 7

Fecha: Lunes 6 de mayo de 2019

Participante(s): Pablo Mora

Asunto: Realización de funciones básicas para manejar matrices

Descripción detallada: Se inició a trabajar a las 11 am, realizando funciones para manejar matrices, entre ellas crear una matriz llena de 0's que asimismo ocupaba una función que crea una lista llena de 0's. También buscar un elemento en una lista para luego aplicarlo en una función que busca un elemento en una matriz. Siguiendo el mismo ejemplo se crea una función que elimina un elemento en una lista para posteriormente se usado por una función que elimina un elemento en una matriz. Para finalizar mi trabajo este día, desarrolló una función para retornar una matriz. (5 horas)

Bitácora 8

Fecha: miércoles 8 de mayo de 2019

Participante(s): Pablo Mora, Eduardo Moya

Asunto: Función Imprimir Todas

Descripción detallada: Al hablar con mi compañero de grupo Eduardo llegamos a la conclusión que era más simple crear la función que imprima todas las soluciones y que para imprimir una sola solución solo tendríamos que poner un break luego de imprimir la primera matriz. Por lo tanto procedo a desarrollar la función, tomando como ejemplo el código de GeeksForGeeks en JAVA, por lo tanto sea crea una función auxiliar a la primera que valide las diferentes necesidades de la función y luego llame recursivamente a la misma función para ir encontrando una solución, se documenta la parte de backtraking, pues primero se quiere encontrar una solución y no se considera necesario por el momento. Al encontrar un error pues la matriz resultante me tira varios campos con 0's, decido seguir otro día. (4 horas)

Bitácora 9

Fecha: Domingo 13 de mayo de 2019

Participante(s): Pablo Mora, Eduardo Moya

Asunto: Conclusión del programa

Descripción detallada: Como no encontraba error en mi lógica, decido hacer una llamada por Skype con mi compañero de grupo Eduardo, entre los dos encontramos un error en una condición pues se alternaba el eje x y el eje y en una validación. Al cambiarlo, esperábamos que funcionara pues ya no veíamos otro error, sin embargo el programa nos decía que todas las condiciones eran falsas en algún punto. Procedemos a usar un debugger y nos dimos cuenta que el algoritmo funcionaba pero este no hacía backtraking y por lo tanto cuando no encontraba camino no podía devolverse y buscar otro. Por esta razón procedo a sacar el backtraking que se suponía sería solo para todas las soluciones y a intentar ponerlo en el programa para que cumpla su función. Al no encontrar cómo hacerlo y por otras obligaciones a cumplir decido seguir buscando solución otro día. (3 horas y media)

Bitácora 10

Fecha: martes 14 de mayo de 2019

Participante(s): Pablo Mora

Asunto: Finalizando las funciones

Descripción detallada: Luego de varios intentos por colocar el backtraking con condiciones este no funcionaba como esperaba pues en el código de GeeksForGeeks éste se aplicaba por pasos, es decir se llamaba una función después de la otra sin ninguna condición. Al pedir ayuda a otros grupos sobre cómo implementaron el backtraking para solucionar el problema, me doy cuenta que ellos usaban otros métodos seguramente más propios de un lenguaje funcional y que el mío quería emular mucho implementaciones como las que vi en JAVA. Por lo tanto sentí que lo único que podía hacer era ver si podía implementarse en Racket mi función, pues ya era tarde para desarrollar una nueva desde cero. (3 horas)

Bitácora 11

Fecha: miércoles 15 de mayo de 2019

Participante(s): Pablo Mora

Asunto: Encuentro solución al problema

Descripción detallada: Se inició a trabajar en la noche del martes 14 y madrugada del miércoles 15 y se logra encontrar que usando lenguaje Racket y no Advance Student podía aplicar mi función como yo deseaba si cambios drásticos, solo agregando dos líneas pequeñas, prosigo a hacerlo y las funciones funcionan a la perfección. (3 horas)