Instituto Tecnológico de Costa Rica

Área académica de Ingeniería en Computadores

II Semestre 2019

Tarea 1. Paradigma Funcional. Nombre código: Wazitico

wazi
tico

Equipo de trabajo

- **Esteban Alvarado Vargas** [2018109336] - *Desarrollador* - estalvgs1999
- **Alejandro Ibarra Campos** [2018089951] - *Desarrollador* - AlejandroIbarraC
- **Jesús Sandoval Morales** [2018203706] - *Desarrollador* - shuzz22260427

Responsable Técnico:

- **Marco Rivera Meneses** - *Profesor del Curso*

Curso:

Lenguajes, Compiladores e Intérpretes - CE 3104.

Lenguaje utilizado:

Scheme/Racket

Nota: Este documento es una transcripción con formato de la Wiki de este proyecto en GitHub para visualización fuera de línea. Se insta a visitar el enlace web del repositorio. https://github.com/AlejandroIbarraC/Wazitico

a. Descripción del Problema

Wazitico - Introducción

Wazitico corresponde a la primer tarea del curso de Lenguajes, Compiladores e Intérpretes [CE 3104] en el cual se deberá implementar una aplicación basada en la popular aplicación móvil **Waze**, cuyo fin será poder obtener direcciones y rutas desde diversos puntos en un mapa, dichos puntos serán desplegados en una interfaz gráfica amigable para facilidad visual del usuario.

La aplicación ha sido desarrollada en su totalidad, en el lenguaje de **paradigma funcional** *"Racket"* el cual se trabajará en el ambiente de programación *DrRacket*. El mapa se modelará por medio de grafos, los cuales a su vez se modelarán dentro del lenguaje por medio de listas, las cuales contendrán los diferentes puntos que actuarán como "Destinos" y "Partidas". **Wazitico** provee las siguientes funcionalidades:

- **Creación de Mapas:**

  La aplicación permite al usuario diseñar sus propios mapas, mediante el modo **Custom City**. En este, el usuario podrá ingresar locaciones y conectarlas, decidiendo la distancia y el sentido de cada carretera, es decir si es de vía única o doble.

- **Mapa de la ciudad:**

  Arcadia Bay es la ciudad que **Wazitico** trae cargada para navegar. La aplicación brinda un mapa con todas las localidades importantes, carreteras y permite encontrar rutas entre ellos.

- **Búsqueda de Rutas:**

  Wazitico tiene como principal función brindar al usuario la mejor ruta hacia su destino; dándole al usuario como *primera opción la ruta más corta*, sin embargo el sistema le proporciona todas las rutas hacia su destino, de modo que el usuario pueda elegir la que le sea conveniente.

Objetivos

Objetivo General

Desarrollar una aplicación que permita reafirmar el conocimiento del paradigma de programación funcional.

Objetivos Específicos

- Crear una aplicación que resuelva el problema utilizando DrRacket.
- Aplicar los conceptos de programación funcional.
- Crear y manipular listas como estructuras de datos.

b. Estructura de datos desarrolladas

Estructuración de los datos utilizados en el programa.

Para el planteamiento del problema se consideró útil modelar los datos y estructurarlos de forma que pudieran ser manejables de forma lógica, debido a esto se decidió expresar los grafos por medio de **listas**, que a su vez contenía otras sub-listas y puntos almacenados, los cuales representaban los lugares y sus conexiones en el mapa.

Esta estructura de datos se supuso como la más optima debido al paradigma funcional en el cual se desarrolló la susodicha aplicación, ya que en los lenguajes que lo aplican, se modelan estructuras y cúmulos de datos complejos únicamente por medio de **listas**, con el fin de facilitar su manejo lógico

Dicha estructura se puede visualizar de la siguiente manera:

- **Destinos**

  Se manejan como los nodos asociados a sus respectivas listas, pueden ser añadidos o eliminados tanto en el grafo estático como en el dinámico.
  Ejemplo: Un nodo se representa como: (nombre ((vec1 peso1) (vec2 peso2) ... (vecN pesoN)))

- **Conexiones**

  Las conexiones se visualizan de forma estructurada como una sub-lista la cual contiene un punto y otra lista con los puntos conectados a este primer punto.

Ejemplo: El nodo 'a se conecta con el nodo 'b de forma bidireccional, y entre ambos hay una distancia de valor 2, se asigna la conexión entre ambos nodos de manera opuesta:
((a ((b 2))) (b ((a 2))))

- **Grafo o mapa**

  La lista principal que contiene todas las otras sub-listas con sus respectivos nodos. Se modela en la interfaz por medio de un mapa.
  Ejemplo: Un mapa con cuatro ciudades podría verse así:
  (("NYC" (("WA" 2) ("NO" 3))) ("WA" (("NYC" 3) ("SF" 5))) ("SF" (("WA" 5) ("NO" 1))) ("NO" (("NYC" 3) ("SF" 6))))

c. Descripción de funciones implementadas

Funciones implementadas.

1. Grafo estático

- **Creación del grafo**

  El grafo se implemento una *lista de listas* conteniendo puntos la cual se predefinirá al iniciar la aplicación y se ejecutarán el resto de funciones sobre dicha lista, para su creación se asigno una variable tipo lista.

  El grafo se encuentra en el archivo arcadiaBay-graph.rkt y está definido como una variable a la que se puede acceder desde otros módulos.

- **Conexiones entre puntos**

  Las conexiones entre los puntos del grafo se verá inalteradas una vez se definan, esto con el fin de posteriormente calcular las distancias por medio del peso entre nodos. Cabe aclarar la posibilidad de conexiones tanto unilaterales como bilaterales. Dichas conexiones se definirán por medio de sublistas, simulando las estructura de un árbol y sus nodos.

---

2. Grafo dinámico

Las funciones desarrolladas para el grafo dinámico son:

- **Graph Maker** (graph-maker node)

  Esta función se encarga de agregar nuevos nodos al grafo. Inicialmente dichos nodos serán puntos en el espacio o mapa sin ningún tipo de conexión entre sí, representados como (nodo ()). Después de recibir el argumento la función aplicará la primitiva "set" a la variable mutable que contiene el grafo, esto activará dos casos: En caso de que el grafo este vacío creará una lista con el argumento dado como primer elemento, de otra forma añadirá el nodo a los elementos ya existentes.

```
> (define (graph-maker node)
    (cond ((null? custom-graph)
           (set! custom-graph (list (list node '()))))
          (else
           (set! custom-graph (list* (list node '()) custom-graph)))))
```

- **Connect** (connect node1 node2 weight bidirectional)

  Esta función realiza las conexiones entre puntos, esta toma como argumentos dos nodos, Nodo 1 y Nodo 2, el peso entre ellos, y un booleano que determina si la conexión será en una o dos direcciones. Esta función maneja dos casos y accederá a uno u otro dependiendo del valor de verdad que se le otorgue al booleano que describe la relación de bidireccionalidad. Al igual que en las demás funciones, “conexiones¨ altera la variable que contiene al grafo e inserta los puntos asociados en una lista de listas, esto con la finalidad de representar sus conexiones a modo de árbol.

- **Add neighbor** (add-neighbor node1 node2 weight graph)

  Esta función se encarga de agregar el nuevo nodo a los vecinos de un nodo dado. Lo hace llamando la actual lista de vecinos de ese nodo y aplicando un cons del nuevo nodo con la lista antigua, retornando la lista actualizada.

```
> (define (add-neighbor node1 node2 weight graph)
      (cons (list node2 weight) (node? node1 graph)))
```

- **Remove node** (remove-node node graph)

  Esta función se encarga de eliminar un nodo de un grafo dado, buscando en la cabeza de la lista una coincidencia con el nodo a borrar. Si no lo hace, llama recursivamente a la función, pasando el resto de la lista. Así hasta que haya una coincidencia y retorne el grafo sin dicho nodo. Esta función es necesaria cada vez que se agregan conexiones.

```
> (define (remove-node node graph)
    (cond ((null? graph)
           '())
          ((equal? node (caar graph))
           (cdr graph))
          (else (cons (car graph)
                      (remove-node node (cdr graph))))))
```

3. Funciones Básicas de Grafos

Las funciones desarrolladas para el manejo de grafos son:

- **solution?** (solution? end path)

  Indica si una ruta dada es solución de un grafo.

- **neighbors?** (neighbors? node1 node2 graph)

  Indica si dos nodos son vecinos.

- **get-neighbors** (get-neighbors node1 graph)

  Retorna una lista con todos los vecinos de un nodo dado.

- **extend** (extend path graph)

  Extiende un ruta dada a nuevas rutas, estas serán todas las posibilidades del último nodo (Se extiende a sus vecinos).

- **weight-btw-nodes** (weight-btw-nodes node1 node2 graph)

  Retorna el peso que existe entre dos nodos. Si no están conectados retorna 0.

- **path-weight** (path-weight path graph)

  Recorre la ruta tomando pares de nodos (siempre los dos primeros de lista) averiguando el peso que hay entre ellos y lo suma a lo que retorna la llamada recursiva sin el primer nodo de la ruta, de modo que al final la suma de cada par de como resultado el peso total de la ruta.

- **node?** (node? node graph)

  Retorna la *lista de vecinos* de un nodo dado.

---

4. Rutas

Las funciones desarrolladas para la búsqueda de rutas en un grafo son:

- **get-path** (get-path start end graph)

  Esta función encapsula la función de búsqueda por DFS. Retorna la lista de rutas ordenada de menor a mayor según su peso.

- **DFS-ALL** (DFS-ALL start end graph)

  Esta función se encarga de encontrar todas las rutas entre dos nodos dados, mediante el algoritmo de búsqueda por profundidad (Detallado en la sección de algoritmos desarrollados). Retorna una lista con todas las rutas posibles.

- **BFS-ALL** (BFS-ALL start end graph)

  Esta función se encarga de encontrar todas las rutas entre dos nodos dados, mediante el algoritmo de búsqueda por anchura (Detallado en la sección de algoritmos desarrollados). Retorna una lista con todas las rutas posibles.

d. Descripción de los algoritmos utilizados

Algoritmos Desarrollados

Para el proyecto se desarrollaron varios algoritmos los cuales tuvieron como finalidad la funcionalidad del proyecto como un todo, la mayoría de ellos fueron algoritmos de lógica, tanto para la creación como para el manejo del grafo.Los algoritmos usados fueron los siguientes:

1. Algoritmo usado para la creación del grafo dinámico

Este algoritmo es una simple adición que muta la variable global o dinámica cada vez que el usuario desee añadir o eliminar un elemento nodo o destino.

2. Algoritmo DFS o búsqueda a profundidad

Este algoritmo es altamente usado para el recorrido de arboles y a su vez de grafos, ya que en el proyecto se usan listas para modelar ambos. Por esta razón, se consideró óptima su implementación dentro de la lógica del búsqueda de rutas

El principio funcional de DFS, es realizar una búsqueda ordenada sobre todos los nodos, sin embargo dicha búsqueda no es uniforme, esto se debe a que en este algoritmo se plantea la búsqueda desde la raíz del árbol y va pasando por los hijos de cada uno de sus hijos hasta llegar al final(esto quiere decir realiza búsquedas de manera vertical y en caso de no encontrar nada sube y sigue así hasta terminar todo el árbol).

3. Algoritmo BFS o búsqueda por anchura

Este algoritmo al igual que el DFS se usa para recorrer estructuras modeladas como árboles.

El principio funcional de la búsqueda por anchura sería la contra-parte del algoritmo visto previamente, en este caso se modela la búsqueda nodo a nodo, pero de manera horizontal, es decir, toma la búsqueda por capas desde el nodo más a la izquierda o derecha y en caso de no encontrar nada baja al siguiente nivel.

e. Problemas Conocidos

En esta sección se detallan los ***problemas que se presentan en el desarrollo y no han sido solucionados.*** Sin embargo, actualmente y hasta el instante de la redacción de este documento, no ha existido permanencia de ninguno de los problemas encontrados durante el transcurso del mismo, esto; sin embargo, no implica que no se hayan encontrado problemas de manera temporal, los cuales serán discutidos en el apartado "Problemas encontrados".

---

f. Problemas encontrados

---

1. Problemas a la hora de modificar la variable dinámica

Durante el desarrollo de la parte dinámica del mapa se encontró un inconveniente a la hora de manejar la variable que contendría al grafo, esto sucedió debido a que a la hora de llamarla en cualquiera de las funciones implementadas la variable no presentaba cambios posteriores.

- **Motivos**

  Posteriormente se descubrió que la razón por lo que sucedía esto era debido a que la función ya recibía un argumento, el cual se creía que era la lista que se estaba usando, pero esto era solo un argumento que se modificaba de manera local.

- **Solución**

  Como se mencionó anteriormente el problema era causado debido a darle un argumento a la función que se suponía debía modificar la variable dinámica, así que la manera de solucionarlo fue eliminar dicho argumento e interactuar de manera directa con el cuerpo de la función.

Referencia

racket, D. (2019). Defining a global variable inside a function in racket. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/33132824/defining-a-global-variable-inside-a-function-in-racket [Accessed 20 Aug. 2019].

2. Problemas con interfaz gráfica

Durante el desarrollo de la interfaz de interacción con el usuario de encontraron diversos problemas, los cuales será generalizados en este apartado por motivos de agilizar su lectura y comprensión. La gran mayoría fueron problemas con tamaños de ventanas, botones y demás, entre otros inconvenientes menores.

- **Motivos**

  La mayoría de los incordios se suscitaban debido a la poca familiaridad por parte del equipo de trabajo con el lenguaje "Racket", así como sus herramientas para desarrollo de interfaces gráficas.

- **Solución**

  El método que se usó para solucionar todos los problemas encontrados con la GUI fue el de investigar e implementar con prueba y error lo aprendido paso a paso hasta finalmente obtener mayor dominio del lenguaje.

Referencias

- Docs.racket-lang.org. (2019). The Racket Graphical Interface Toolkit. [online] Available at: https://docs.racket-lang.org/gui/ [Accessed 20 Aug. 2019].
- Groups.google.com. (2019). Google Groups. [online] Available at: https://groups.google.com/forum/#!topic/racket-users/pJzgbKRBN8U [Accessed 20 Aug. 2019].

3. Problemas con desarrollo de algoritmos lógicos

El mayor problema de este apartado fue el intento de implementar el algoritmo de "Dijkstra" como el principal método para el calculo de rutas por medio del peso entre sus nodos. Sin embargo, se decidió tomó la decisión por parte de todo el grupo de desistir de esta idea y darle otro enfoque al implementar otros algoritmos.

- **Motivos**

  Después de mucho análisis se llegó a la conclusión de que el algoritmo de "Dijkstra" tiene un fuerte enfoque a lenguajes orientados a objetos, esto debido que se basa principalmente en el manejo de variables y una abstracción con el fin de modelarlo en un lenguaje con paradigma funcional como "Racket" se consideró poco eficiente.

- **Solución**

  Como principal solución a dicho problema se decidió cambiar el algoritmo principal e implementar DFS y BFS, esto debido a que se consideró optimo y ,a su vez, la diferencia de eficiencia a la hora de modelarlo en el proyecto era prácticamente nula.

Referencias

- CodinGame. (2019). Coding Games and Programming Challenges to Code Better. [online] Available at: https://www.codingame.com/playgrounds/7656/los-caminos-mas-cortos-con-el-algoritmo-de-dijkstra/el-algoritmo-de-dijkstra [Accessed 20 Aug. 2019].
- GeeksforGeeks. (2019). BFS vs DFS for Binary Tree - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/bfs-vs-dfs-binary-tree/ [Accessed 20 Aug. 2019].

g. Organización del Proyecto

Plan de actividades

El proyecto se ha desarrollado bajo el planeamiento mostrado en el siguiente enlace web:

https://docs.google.com/spreadsheets/d/1tXv7tf6-taOxsFZYWTudcCyqJww2UNQo3-9L5kPX77I/edit#gid=2060662469.

h. Conclusiones y recomendaciones

Conclusiones

A lo largo de todo el proyecto se obtuvieron múltiples conocimientos los cuales serán, sin duda alguna, provechosos en futuros proyectos, en este apartado se pretende resumirlos de manera concisa y directa.

1. La importancia de estructurar cúmulos de datos complejos como listas para facilitar su manejo lógico a la hora de usar un paradigma de tipo funcional, en este caso en Racket.
2. Se concienció sobre el buen uso y manejo de un lenguaje que inferencia tipos, y así mismo se pudo visualizar de primera mano su potencia.

3. Como grupo se pudo concluir y contrastar la diferencia entre paradigmas, remarcando esto la importancia de mantener buenas practicas e implementar de manera correcta las directrices de cada paradigma a la hora de programar.
4. La importancia de crear un ambiente de interacción usuario-código amigable e intuitivo y junto con esto poner en perspectiva práctica la importancia de desarrollar dicho ambiente con cambios dinámicos en tiempo real por parte del usuario.
5. El paradigma funcional fomenta un pensamiento distinto que obliga a dividir los grandes problemas en problemas más simples que solucionar, dando como resultado un conjunto de soluciones pequeñas que al unirse solucionan el problema mayor.

Recomendaciones

1. Tratar de comprender de forma natural la estructura de un paradigma funcional antes de empezar a intentar programar, esto con el fin de evitar errores básico o problemas de frustración al no entender ciertos conceptos
2. Evitar al máximo el uso de variables; el paradigma como tal de un lenguaje funcional trata de restringir la declaración de variables a la hora de implementar métodos. Se recomienda practicar el uso de recursividad para complementar el no usar variables
3. Entender plenamente el modelamiento de estructuras en Racket por medio de listas, esto con el fin de poder facilitar la implementación y desarrollo de métodos de carácter lógico

i. Bibliografía

Bibliografía

- Guzman, J. E. (2006). Introducción a la programación con Scheme. Cartago: Editorial Tecnológica de Costa Rica.
- racket, D. (2019). Defining a global variable inside a function in racket. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/33132824/defining-a-global-variable-inside-a-function-in-racket [Accessed 20 Aug. 2019].
- Docs.racket-lang.org. (2019). The Racket Graphical Interface Toolkit. [online] Available at: https://docs.racket-lang.org/gui/ [Accessed 20 Aug. 2019].
- Groups.google.com. (2019). Google Groups. [online] Available at: https://groups.google.com/forum/#!topic/racket-users/pJzgbKRBN8U [Accessed 20 Aug. 2019].
- CodinGame. (2019). Coding Games and Programming Challenges to Code Better. [online] Available at: https://www.codingame.com/playgrounds/7656/los-caminos-mas-cortos-con-el-algoritmo-de-dijkstra/el-algoritmo-de-dijkstra [Accessed 20 Aug. 2019].
- GeeksforGeeks. (2019). BFS vs DFS for Binary Tree - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/bfs-vs-dfs-binary-tree/ [Accessed 20 Aug. 2019].

j. Bitácora

9 De agosto de 2019.

El día de hoy desde plena mañana planeamos una reunión para empezar la planificación del proyecto en cuestión. La reunión estaba programada para las 5 p.m del día presente; sin embargo, por motivos de lluvia y choque de horarios en cursos y actividades lectivas decidimos reunirnos de manera no presencial por medio de una llamada en la plataforma "Discord" a las 8:12 p.m. Durante la reunión se hizo la división principal de tareas por medio de un Excel en línea, en dicha hoja de cálculo se acordaron las tareas a realizar por cada miembro así como las horas que se deberán, teóricamente, invertir en cada una. La reunión termino con el acuerdo por parte de todos nosotros, los miembros del grupo, acordando empezar al día siguiente con las investigaciones iniciales. Se termina la llamada a las 10:54 p.m del presente día

10 de agosto del 2019

El día de hoy se crea el grupo de WhatsApp "Por la teja en lenguajes" el cual tendrá como fin la comunicación por parte de todos los miembros de manera ágil y rápida. En principio cada miembro del grupo empezó con su tarea inicial asignada

Jesús: Inicia investigación sobre variables mutables y su modificación, investiga sobre algoritmos de búsqueda de rutas y las funciones que serán necesarias

Alejandro:Inicia investigación sobre el manejo de la GUI en Racket y su interfaz en general

Esteban:Apoya a Alejandro con la GUI, además empieza a crear grafos de prueba para futuras pruebas

13 de agosto del 2019

El día de hoy se realizó el primer commit por parte de Esteban, el cuál contenía el primer prototipo de gráfico estático y el cuál se utilizará para pruebas posteriores y la prueba de todos los métodos en general. El hecho más importante ocurrido hoy fue el descubrimiento, debido a una consulta con el profesor Marcos Rivera, acerca de que se debía implementar una versión dinámica del grafo, el cuál deberá ser capaz de actualizarse en tiempo real. Los avances de hoy por parte de cada miembro fueron:

Alejandro:Empieza a montar la primera ventana únicamente con dos botones, le pide ayuda a Esteban con el scaling de una imagen, discute con el grupo el dibujo de las rutas

Esteban: Pide hacer parte de la documentación general del código, ayuda a Alejandro después de terminar un quiz virtual de física, hace el primer commit con el grafo de prueba, investiga sobre parte lógica para apoyar a Jesús en su trabajo

Jesús: Sigue la investigación de "Dijsktra", implementa internamente la función de vecinos y la función que verifica la existencia de una ruta

14 de agosto de 2019

El día de hoy Esteban amanece enfermo por lo cuál no asiste al TEC, lo cual imposibilita una reunión presencial, debido a esto la mayoría de ideas se discuten por medio del grupo "Por la teja en lenguajes". Se empieza a implementar la variable global que contendría al grafo capaz de ser mutable por medio de los métodos. Los avances de hoy por parte de cada miembro fueron:

Alejandro: Discute por medio del grupo posibles diseños de mapas de fondo para implementarlos, avanza fructuosamente en gran parte de la interfaz después de programar 5 horas continuas y revisar la página de Gui de Racket(La cual esta

referenciada en la documentación respectiva). Indicia a Jesús la necesidad de tres métodos específicos para ajustar el grafo dinámico a la interfaz. Dichos métodos son "Crear grafo", el cual es capaz de añadir nodos y eliminarlos para representar el grafo, "Conexiones", el cual es capaz de conectar dos nodos y marcar el peso entre ellos, así como la posibilidad de de decidir la direccionalidad de las rutas, y "Rutas", el cual busca la ruta mas corta y la devuelve, así como las rutas alternas ordenadas de mayor a menor según el peso de sus nodos

Esteban: Implementa la función lógica para obtener los pesos entre nodos con el fin de ayudar a Jesús con la lógica de rutas, implementa una función que retorna el peso de una ruta completa, aconseja a Jesús sobre el uso de BFS Y DFS como algoritmos principales para el cálculo de rutas, esto después de muchas pruebas por su parte

Jesús: Empieza a implementar el "set!" como primitiva para ajustar la variable mutable y ajusta la lógica previa para ser capaz de implementarse sobre el nuevo grafo dinámico, empieza la implementación de las funciones indicadas por Alejandro. En este punto empieza a dudar sobre la idea de implementar Dijkstra e investiga sobre otros algoritmos de búsqueda para las rutas, los cuales concluye con Esteban que serían BFS Y DFS

15 de agosto de 2019

Este fue el día con mayor cantidad de aportes por parte todos los miembros, además de la mayor reunión colaborativa ya que los 3 miembros nos ayudamos mutuamente en nuestras diferentes áreas con el fin de mantener una división sana del trabajo Los avances de hoy por parte de cada miembro fueron:

Esteban: Ayuda a Jesús con la lógica y Alejandro con partes de la interfaz, agrega los algoritmos de BFS y DFS que implementó

Alejandro: Termina la penúltima versión de la interfaz, ayuda a Esteban con la implementación de la lógica de rutas

Jesús: Añade los algoritmos solicitados por Alejandro con el fin de complementarlo con los hechos para rutas por parte de Esteban, ayuda a Alejandro con un algoritmo para dibujar flechas bonitas en la interfaz

16 de agosto de 2019

Para este punto el proyecto estaba en su punto final, debido a que todas las tareas estaban listas y funcionando de la manera esperada, debido a esto se decide empezar por la la creación de la documentación pertinente. Además de esto la interfaz se completó de forma definitiva y entre todos los miembros se procedió a unir la parte lógica con la de interfaz. Los avances de hoy por parte de cada miembro fueron:

Jesús: Empieza la documentación en la wiki de GitHub usando formato MarkDown

Esteban: Empieza el manual de usuario

Alejandro: Sube la versión final de la interfaz, le termina retoques estéticos a todos los mapas usando Adobe Photoshop

17 de agosto de 2019

El día de hoy se encontraron errores después de conectar la lógica con la interfaz, por lo cual se procedió a corregirlos. El día de hoy el proyecto estaba listo y funcional en su epítome Los avances de hoy por parte de cada miembro fueron:

Esteban: Arregla el algoritmo para la conexión de rutas y lo mejora considerablemente, se soluciona el error, el cuál consistía en un problema al llamar a la variable global

Jesús: Corrige un problema en la actualización de la variable global al insertarse en los métodos, el problema era causado por un mal uso del "set!", el error se soluciona de forma exitosa

Alejandro: Reconecta las funciones corregidas por Jesús y Esteban con la interfaz de forma exitosa

20 de agosto de 2019

Se añade el manual de usuario por parte de usuario, hasta este punto todos los miembros estuvieron trabajando en la documentación del proyecto. Se termina la documentación y el manual de usuario

Alejandro: Termina de comentar las especificaciones sobre la interfaz

Esteban: Le da los toques finales al manual de usuario y pide la opinión de los otros miembros del grupo

Jesús: Termina de explicar los algoritmos y las estructuras de datos usadas en el proyecto

Todos: Redactan sobre los errores, recomendaciones y conclusiones

21 de agosto de 2019

Se da la revisión final al proyecto y se formatea toda la documentación pertinente, incluyendo la versión final de manual de usuario, con sus descripciones apropiadas y actualizadas. Se termina la bitácora de forma colectiva.