WAZITICO

OUTSMARTING TRAFFIC, TOGETHER

DOCUMENTACIÓN TÉCNICA.

VERSIÓN: 1.0

CARTAGO, COSTA RICA.

AGOSTO, 2019.

Tabla de Contenidos

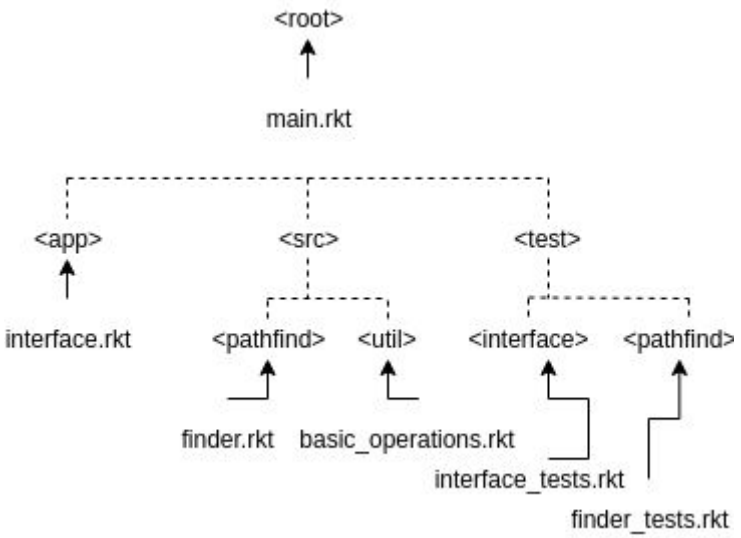
Documentación Técnica Wazitico

Introducción

Wazitico, la aplicación de ahora en adelante, corresponde al Proyecto I para el curso de Lenguajes, Compiladores e Intérpretes. (CE3104), Módulo Lenguajes. Este curso introduce los cuatro paradigmas de programación principales más representativos. Además introduce los principios y métodos utilizados en el diseño e implementación de compiladores e intérpretes para dichos lenguajes.

El proyecto consiste en la implementación de una aplicación que permita reafirmar el conocimiento del paradigma de programación funcional utilizando DrRacket. La programación funcional, como su nombre lo indica, está centrada en funciones, las cuales son tratadas como ciudadanos de primera clase. La aplicación tiene como objetivo el desarrollo de un grafo mixto que simule la famosa aplicación Waze, pero con la habilidad del usuario de crear su propio mapa. Waze es una aplicación social de tránsito automotor en tiempo real y navegación asistida por GPS desarrollada por Waze Mobile. El 11 de junio 2013, Google completó la adquisición de Waze en $966 millones de dólares. Los usuarios de Waze son denominados Wazers, y, a diferencia de los softwares de navegación asistida por GPS tradicionales, este es mantenido por los usuarios y aprende de las rutas recorridas por sus usuarios para proveer información de enrutamiento y actualizaciones de tráfico en tiempo real.

Estructura del Proyecto



Descripción de las funciones implementadas

interface.rkt

- addNodeToGraph: función que agrega un nodo al grafo
- addPosToHash: agrega la posición del nodo al hash
- joinNodes: función que une los nodos
- drawWeight:
- getSelectedPath: obtiene la ruta desde la list-box

finder.rkt

- addToGraph: recibe dos nodos y los agrega al grafo como una conexión con el formato (primero (segundo)); si el primero o ambos nodos están en el grafo, esta función solo agrega lo necesario.
- addByIndex: recibe el índice del nodo que hay que reemplazar y lo reemplaza con el nodo más la nueva conexión.
- axisExists: revisa si una arista ya existe
- findPaths: encuentra todas las rutas entre el origen y el destino en el grafo (utiliza búsqueda por anchura)

basic_operations.rkt

- extractElement: extrae el n elemento de una lista
- removeElement: elimina el n elemento de una lista
- lastElement: extrae el último elemento de una lista
- reverseList: invierte una lista
- reverseSublists: invierte sublistas
- solution?: revisa si hay una ruta
- getNeighbors: obtiene los vecinos en un grafo
- extend: crea nuevos caminos

Descripción de las estructuras de datos desarrolladas

Grafos

- Grafo de lugares: el grafo se utiliza para crear los nodos (puntos de origen y destino) y elegir la mejor ruta hacia el destino. Se maneja con el siguiente formato de aristas: ((origen1, (destino1, destino2)), (origen2 (destino1, destino2, destino3)))

Listas

- Lista de rutas: esta lista contiene todas las rutas posibles del nodo origen al nodo destino. Esta contiene una lista para cada ruta. La lista se utiliza con el formato ((nodo1, nodo2, nodoFinal), (nodo1, nodoFinal)), siendo los nodoN los nodos visitados en el camino y el nodo final el destino.

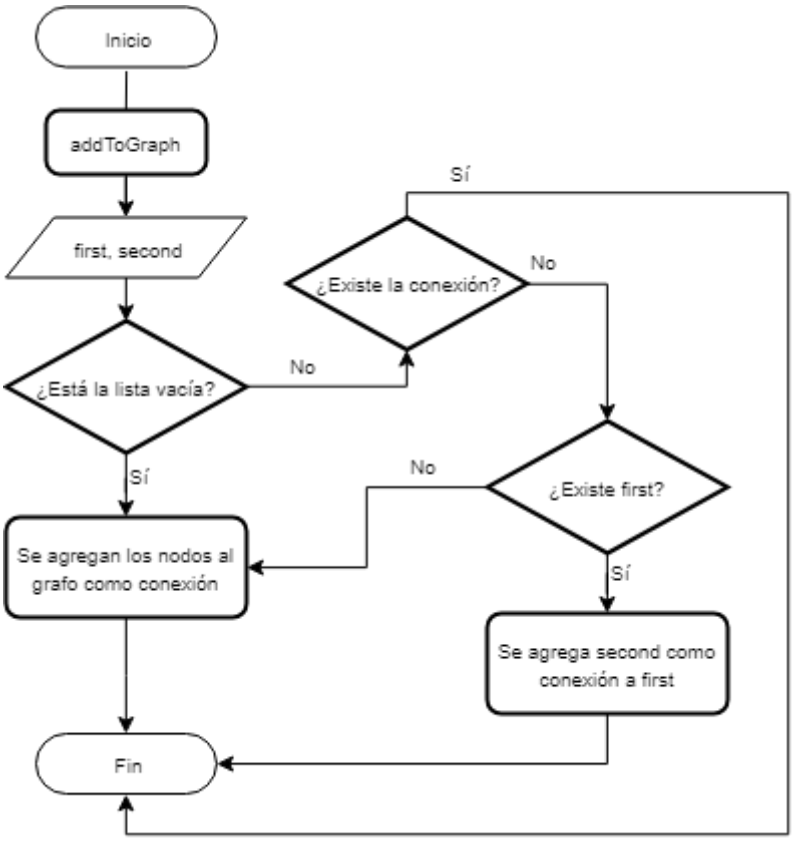
Hash Tables

- Hash de nombres: contiene la traducción del nombre elegido para el nodo en el grafo a la id que se maneja dentro del código.
- Hash de coordenadas: contiene la traducción de la id de uno de los nodos a las coordenadas de este.

Descripción detallada de los algoritmos desarrollados

Algoritmo para agregar elementos al grafo

Este algoritmo recibe un nodo origen y un nodo destino y crea una ruta de una vía entre ellos. Si el nodo origen ya se encuentra en el grafo, el nodo destino solo se agrega a la lista de nodos que se conectan con el nodo origen utilizando la función *addByIndex*. Si ambos el nodo origen y el nodo destino ya se encuentran en la lista como una ruta, no se agrega ninguno.



Algoritmo para posicionar nodos y sus conexiones en la interfaz

1. Posicionar Nodos: El algoritmo para posicionar nodos en la interfaz se divide en dos funciones.
    a. drawNodes: Recibe como parámetros: color, x, y, name, color. Verifca si el color es vacío y toma como color de nodo el negro, sino toma el color ingresado por el usuario. Además, dibuja una ellipse en la posición x, y ingresado por parámetro.
    b. addNodetoGraph: Añade posiciones x y al HashTable, relaciona el nombre del nodo con estas posiciones.

2. Posicionar Conexiones entre dos nodos: Recibe como parámetros color, x1, y1, x2, y2. Hace uso de la función addToGraph que se encuentra en *finder.rkt,* añade las posiciones a la lista de líneas y dibuja una flecha con las posiciones ingresadas por el usuario.

Algoritmo para encontrar todas las rutas

Se utiliza el proceso de búsqueda de encontrar las rutas mediante anchura primero, extiende todas las rutas de forma simétrica. Recibe como parámetros src, dest, graph. Src representa el nodo de inicio, dest el nodo de llegada y graph, el grafo en el cual se quieren encontrar las rutas.

Dado un grafo G y un vértice inicial, en nuestro caso src, una búsqueda en anchura procede explorando las aristas en el grafo para encontrar todos los vértices en G para los cuales hay una ruta a partir de src. Lo notable de este tipo de búsquedas es que encuentra todos los vértices que estén a una distancia k de src antes de encontrar cualesquiera vértices que estén a una distancia k+1. Una buena manera de visualizar lo que hace el algoritmo es imaginar que está construyendo un árbol, un nivel de árbol a la vez. Una primera búsqueda en anchura agrega todos los hijos del vértice inicial antes de que comience a descubrir a alguno de los nietos.

Problemas conocidos

- Cuando se cambia de tamaño la ventana, desaparecen los puntos.
- El código no se puede ejecutar desde el archivo main.rkt

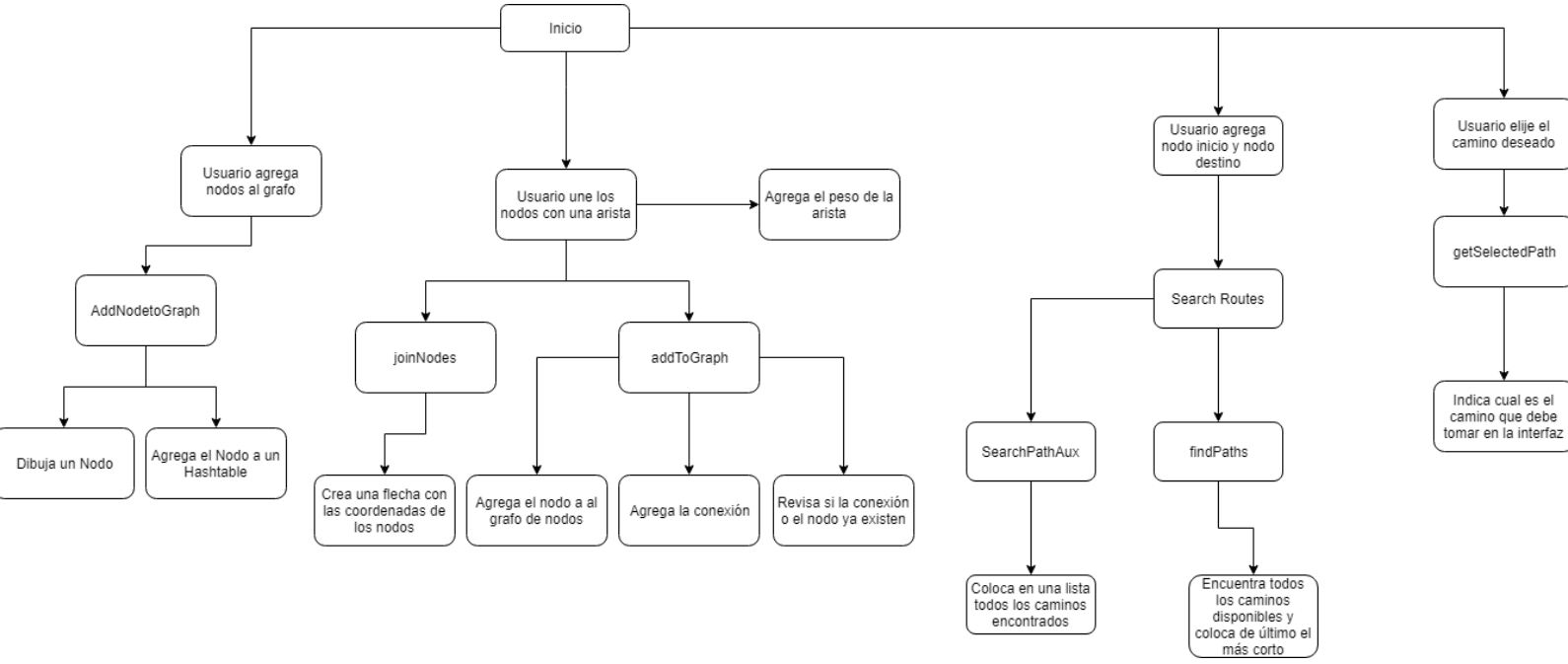
Problemas encontrados

- Búsqueda de caminos: se intentó implementar el algoritmo Dijkstra, pero gracias a la falta de variables en el lenguaje no terminó funcionando; luego se implementó una especie de algoritmo BackTrack, pero este terminó sin funcionar igual que el anterior; por último, se soluciona el problema implementando una búsqueda por anchura.

Plan de Actividades

1. Creación del grafo mixto
2. Creación de las aristas con un peso
3. Definir punto de inicio y destino
4. Encontrar rutas y la ruta más corta
5. Interfaz
    a. Representación del grafo en la interfaz
    b. Representación de las aristas
    c. Representación del lugar de salida y su destino
    d. Representación de todos los caminos y la ruta más corta

Flujo del Proyecto



Conclusiones

1. Para hacer la búsqueda de todas las rutas se utilizó búsqueda por anchura para mejorar rendimiento.
2. Una ventaja de utilizar scheme es que las variables son dinámicamente tipadas
3. Racket no depende de variables globales, todo se pasa a través de parámetros.
4. Racket, al ser un lenguaje de alto nivel de abstracción y complejidad baja hace que sea fácil para un programador comenzar a aprender este lenguaje.
5. Como el programa no depende de variables globales, se vuelve menos volátil.

Recomendaciones

1. Utilizar búsqueda por anchura en vez de búsqueda por profundidad, esto debido a que los nodos no tienden a tener muchos hijos (conexiones), entonces por anchura leería menos nodos para terminar cada recorrido.
2. El uso de un HashTable para guardar los nombres lo hace eficiente
3. Utilizar recursividad para dividir los problemas en problemas más pequeños hasta llegar a un caso que sea más sencillo de resolver y luego juntar los resultados de los subproblemas hasta llegar al resultado del problema grande
4. Utilizar diversos parámetros en vez de variables globales, esto para que se procesen los parámetros sin modificar los argumentos.
5. Se pueden usar HashTables para relacionar el nombre que utiliza el usuario para representar el nodo con una forma más fácil de representar los nodos, como integers.

Bitácora

Actividad	Fecha de ejecución	Persona(s)
Creación del Grafo mixto	11 de Agosto	Jonathan Iván
Encontrar Rutas	Del 16 al 17 de Agosto	Angelo
Creación de aristas con un peso	17 de Agosto	Angelo Iván
Representación del grafo y aristas en la interfaz	Del 18 y 19 de Agosto	Angelo Jonathan
Representación de todos los caminos	18 y 19 de Agosto	Angelo Jonathan
Mostrar distancia del origen al final	21 de agosto	Angelo Iván

Bibliografía

- MANUAL DE USUARIO DE LA APLICACIÓN INFORMÁTICA “PROGRAMA DE EJERCICIOS”. (n.d.). [ebook] Disponible en: http://www.sermef-ejercicios.org/webprescriptor/ayuda/manualUsuario.pdf [Accesado 19 Ago. 2019].
- https://www.areatecnologia.com/diagramas-de-flujo.htm#Como_Hacer_un_Diagrama_de_Flujo
- Flatt Matthew, Bruce Findler Robert, Clements John. The Racket Graphical Interface Toolkit. Racket. https://docs.racket-lang.org/gui/