Instituto Tecnológico de Costa Rica

Lenguajes Compiladores e Intérpretes

I Semestre 2021

Grupo 1:

Freddy Armando Fallas Garro.
Harold Espinoza Matarrita

Profesor:
Marco Rivera Meneses

Tarea 1

Índice

Manual de usuario

1. Desde el entorno DrRacket abra el archivo “interfazG.rkt”
2. Ejecute el código
3. Desde el entorno ingrese el comando “(waztico nodoInicial nodoDestino, grafo con arcos)” como se muestra en el siguiente ejemplo:
   (wazitico 'A 'C '((A (B 5) (C 7) (J 6)) (B (C 5)) (J (B 3) (D 7) (C 9))(C)(D)))
4. Al ejecutar el código se abrirá una ventana, haga doble clic en el recuadro gris de “calcular resultado” para calcular las rutas de viaje

Descripción de los algoritmos desarrollados

Algoritmo de búsqueda: Inicialmente comprueba que el origen exista dentro del grafo y que tenga aristas. Está constituido por 4 listas: vecinos, aquí se almacenan los grafos por los cuales se a pasado durante la ruta hacia el destino, lista ruta, en esta se almacena todas las aristas encontradas por cada grafo dentro de la lista de vecinos, lista visitados, aquí se almacenan todos los grafos que fueron recorridos globalmente en el algoritmo para evitar pasar más de 2 0 3 veces por el mismo grafo y evitar que entre en un ciclo indefinido, lista costos, almacena los costos de cada arista recorrida durante el algoritmo las cuales se suman al encontrar el destino. Cuando el algoritmo encuentra el destino, suma el costo de la ruta y la agrega a la lista de vecinos la cual sería la ruta encontrada, el algoritmo al encontrar una solución hará backtracking hasta llegar a un grafo que no a sido visitado, esto se realiza comparando las listas de vecinos y ruta hasta que el primer elemento de ambas listas sean distintos, el resultado va a ser retornado hasta que la lista de rutas sea nula lo cual va a significar que todas las rutas ya fueron exploradas.
El algoritmo siempre revisa si un grafo ya fue revisado con el objetivo de obtener resultados directamente de resultados anteriores para así evitar ejecutar nuevamente los procedimientos del algoritmo y también para evitar que entre en un ciclo lo cual hace que el algoritmo ahorre recursos.

Algoritmo quicksort: se implementa el quicksort para ordenar los elementos de las rutas encontradas, para ello se parte desde el primer elemento de la lista que funciona como pivote, con base en este se sacan todos los elementos menores que a su vez van siendo ordenados de manera recursiva, luego con base al mismo pivote se calculan los elementos mayores de la lista de la misma manera y una vez que se calcularon se procede a realizar el concatenado de los menores con el pivote y los mayores de la lista.

Funciones implementadas

Función de concatenar dos listas: Une dos listas en una sola, recorriendo una de ella y agregando cada elemento a la otra.

Función primero hijo: Recibe el grafo del cual se quiere obtener su primer arista.

Función booleana bool-hijos: Devuelve #t si un grafo tiene aristas y un #f si el grafo no tiene aristas.

Función hijos-origen: Busca un grafo en el cual retorna cada uno de sus aristas pero sin los costos.

Función bool-origen: Revisa si realmente existe el origen.

Función sumar-costos: Suma cada elemento de una lista y retorna el resultado obtenido.

Función menores: construye una lista con los elementos menores a un número dado

Función mayores: construye una lista con los elementos mayores a un número dado

Función arc: Desde un grafo dado compara con los círculos dibujados para saber si se debe dibujar un arco desde ese punto

Función arcAux1: función auxiliar de “arc” que dibuja los arcos desde el punto inicial dado hacia los puntos posibles.

Problemas encontrados

1. El algoritmo dejaba rutas sin explorar, esto era causado debido a que si un grafo era agregado a la lista de visitados y otro grafo tenía una arista hacia este mismo grafo, el algoritmo no permite seguir adelante porque lo interpreta como ya visitado, este problema fue encontrado mediante depuración, debido a este problema surge la idea de revisar entre los resultados ya encontrados, ya que si el grafo está dentro de visitados significa que ya todas sus opciones fueron recorridas, por lo que se recorre los resultados hasta encontrar el grafo, si se encuentra significa que los grafos siguientes son una ruta posible por lo que se retorna los siguientes grafos y sus costos, por lo que, este problema fue solucionado.

2. La interfaz en el conteo de nodos no tenía en cuenta los terminales por lo que se dibujaba una menor cantidad de nodos. Se soluciona pasando los terminales sin ningún arco

Problemas sin solución

1. Al cerrar la ventana desde la “X” el programa no detiene su ejecución, simplemente cierra la ventana. Una solución es detener la ejecución desde el el entorno de programación.

Plan de actividades

- Desarrollo del algoritmo de búsqueda, búsqueda de información y elaboración, responsables: Armando Fallas, fecha límite para realizar la tarea 22/05/21
- Realizar interfaz gráfica para agregar nodos y aristas. Responsable Harold Espinoza, fecha límite de elaboración 20/05/21
- Realizar interfaz gráfica del dibujo de los nodos. Responsable Harold Espinoza fecha límite de elaboración 25/05/21
- Diagrama de flujo del algoritmo. Responsable Armando Fallas, fecha límite de elaboración 26/05/21
- Desarrollar la documentación de la tarea, responsables Armando Fallas y Harold Espinoza, fecha límite de elaboración 26/05/21.

Estructuras de datos desarrolladas

Se utiliza un grafo que se determina de la siguiente manera:

((a (b 7) (c 8)) (c (g 5)))

Este se compone de una lista principal que contiene los nodos y sus respectivos arcos, así cada elemento de la lista principal es una sublista que contiene como primer elemento el nodo desde donde se parte, como elementos siguientes aparecen sublistas las cuales contienen como primer elemento el nodo hacia el que se puede viajar y como segundo elemento el costo de viajar a ese nodo. A continuación se presentan los elementos partiendo del ejemplo anterior.

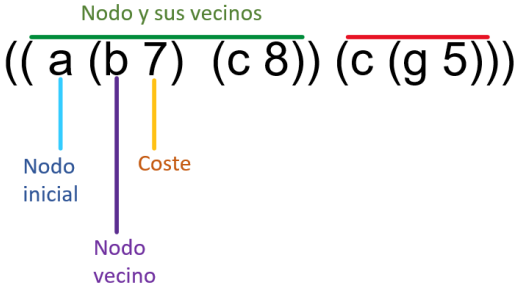
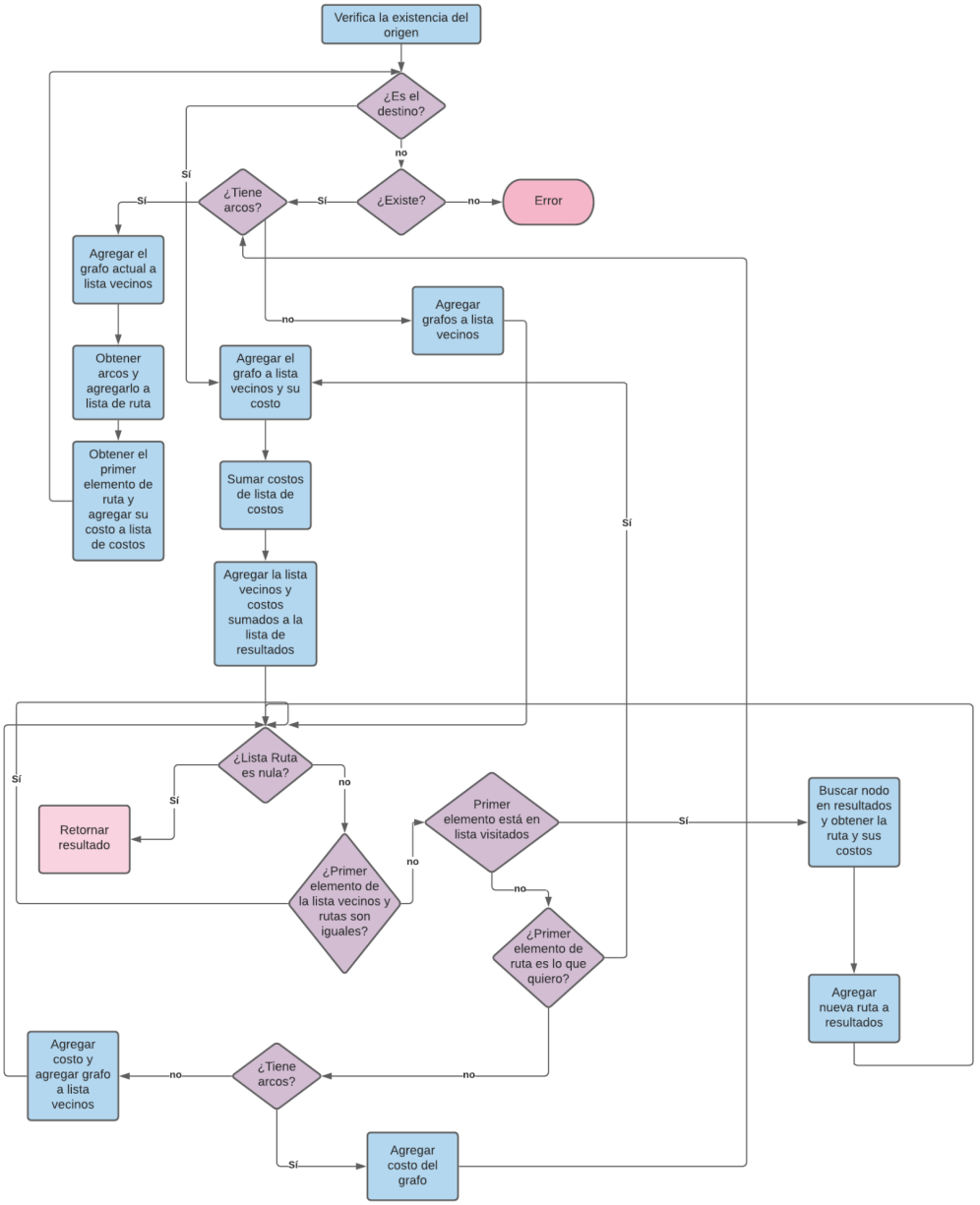

Nodo y sus vecinos
((a (b 7) (c 8)) (c (g 5)))
Nodo inicial
Coste
Nodo vecino

Diagrama del algoritmo de búsqueda



Conclusiones

Programar en racket es un entorno desconocido para muchos en la actualidad lo cual dificulta la programación debido a los nuevos lenguajes que se aprenden inicialmente, pero con la elaboración de esta tarea programada se logró una comprensión tanto del lenguaje y el paradigma funcional.

La elaboración de una interfaz gráfica mediante las librerías existentes está muy limitado en cuanto a sus funcionalidades, graphics unas de las 2 librerías usadas en la tarea

programada solo permite realizar dibujos, en cambio toolkit es una libreria mas enfocada a formularios lo cual se debió usar ambas lo que provocaba errores de compatibilidad entre ellas, lo cual hizo necesario usarlas en ventanas distintas.

La elaboración de un algoritmo complejo como el desarrollado en la tarea fue desafiante debido a ser realizado mediante únicamente el uso del paradigma funcional, el cual agrega un grado de dificultad en el desarrollo, lo que hizo muy importante una muy buena planificación inicial y el desarrollo de un diagrama de flujo con los pasos deseados.

Crear un plan de actividades y fechas límites para su elaboración facilita llevar el control del avance de la tarea y asegurarse de que todo se esté haciendo de forma correcta, lo cual logra poder tener la tarea prácticamente realizada en su fecha de entrega.

Recomendaciones

No agregar más de 5 nodos al grafo debido a que esa es su capacidad máxima, esto limitante es principalmente en la parte gráfica, el algoritmo de búsqueda soporta cualquier tamaño de grafo, pero si se desea incrementar la cantidad, la solución es implementar el método de dibujo programado a partir de los 5 ya implementados.

Para ingresar el grafo, el nodo de origen y destino es recomendable agregarlos desde la consola de programación ya que no se pudo implementar este método dentro de la interfaz.

Luego de cerrar la ventana desde la “X” detenga la ejecución del programa desde el entorno de programación.

Bitácora

Primera reunión, realizada el 17/05/21 a las 5 pm, se creó el plan de actividades a realizar y se repartió las tareas a realizar, duración de la reunión 1 hora.

19/05/21 Se hace una reunión para organizar el trabajo y definir métodos para el avance del código. También se investiga acerca de la librería “The Racket Graphical Interface Toolkit” para crear la interfaz de la aplicación.

19/05/21 Se planteó un diagrama de flujo para poder desarrollar el algoritmo de búsqueda, este mismo día se inició con la programación del algoritmo.

20/05/21 Se crea una ventana base para la interfaz gráfica con botones y entradas básicas.

22/05/21 Se finalizó el algoritmo de búsqueda, el cual solo presentaba un error al elaborar las pruebas debido a que dejaba rutas sin explorar.

25/07/21 Se arreglaron problemas presentes en el algoritmo de búsqueda, se realizaron las pruebas necesarias, se continuó con la elaboración de la documentación y el diagrama de flujo del algoritmo de búsqueda.

Segunda reunión, 24/05/21 a las 6 pm, se presentó los avances de las tareas y se propuso como continuar con lo que quedaba del proyecto, se discutió errores presentes en el código y de posibles soluciones.

24/05/21 Se investiga acerca de la librería “graphics” de racket para utilizarla en conjunto, también se añade la visualización de la ventana y el display de las rutas obtenidas.

25/05/21 Se descarta la librería “The Racket Graphical Interface Toolkit” por cuestiones de compatibilidad, además se implementa el dibujado del grafo y las rutas obtenidas, esto para un máximo de 5 nodos.

Referencias

Rodríguez Lozano, F. J. (2013). *Representación Gráfica en Scheme* [Diapositivas]. uco.es. http://www.uco.es/users/ma1fegan/Comunes/asignaturas/pd/Scheme-comandos-graficos.pdf

Graphics: Legacy Library. (s. f.). Racket documentación. Recuperado 26 de mayo de 2021, de https://docs.racket-lang.org/graphics/index.html