ALGORITMO QUE DÉ UNA SOLUCIÓN ÓPTIMA PARA EL RUTEO DE VEHÍCULOS ELÉCTRICOS

Kevin Arley Parra Henao
Universidad EAFIT
Colombia
kaparrah@eafit.edu.co

Daniel Alejandro Mesa Arango
Universidad EAFIT
Colombia
damesaa@eafit.ecu.co

Mauricio Toro
Universidad EAFIT
Colombia
mtorobe@eafit.edu.co

RESUMEN
Diseñar un algoritmo para encontrar las rutas óptimas para que un conjunto de camiones eléctricos visite un conjunto de clientes. [1] A corto y largo plazo, está tecnología promete cambiar la forma en qué nos transportarnos, sin embargo, su consumo energético y la necesidad de recargar las baterías empleadas por vehículos eléctricos, hace necesario considerar las rutas que se toman. Algunos problemas relacionados son: El problema del camino más corto, grafos bipartitos, ciclos impares, coloración, componentes fuertemente conexas y árbol de expansión mínima. Como una solución aproximada es el algoritmo de TSP que puede ser implementado de varias formas, tal que se haga un recorrido completo del grafo o mapa, se tiene entonces como resultado un tiempo aceptable en el que se puede visitar cada uno de los nodos buscando siempre la eficacia y rendimiento de los camiones eléctricos, como tal el problema no tiene una solución defectiva debido a la cantidad de variables que posee el problema y no contar con un algoritmo capaz de hacer el recorrido o buscar la combinación mas optima en un tiempo considerablemente aceptable, por esto hoy día solo se han planteado teorías cercanas a una solución definitiva.

Palabras clave

Algoritmos → Diseño → Transporte → Inteligencia artificial → Optimización → Eficiencia → Ruteo → Vehículos → Vehículos eléctricos → Grafos → Mapa → Ruta corta → Ruta más eficiente → Mejor ruta → Tiempo → Estructura de datos → Diseño de algoritmos

Palabras clave de la clasificación de la ACM

Computabilidad → Abstracción → Diseño y análisis de algoritmos → Análisis de algoritmos para grafos → Análisis de algoritmos de aproximación → Algoritmos de aproximación numérica → Diseño y análisis de estructura de datos → Comprensión de datos → Técnicas de diseño de algoritmos → Divide y conquistaras

1. INTRODUCCIÓN

El problema por resolver es un Problema de Ruteo de Vehículos (VRP, por sus siglas en inglés, Vehicle Routing Problem) con la modificación de que los vehículos a considerar son eléctricos, por lo que el tiempo de carga y el consumo energético serán factores importantes a tener en cuenta. El primer problema planteado tipo VPR fue el del agente viajero o TSP (Travelling Salesman Problem) introducido por Flood en 1956. [2]

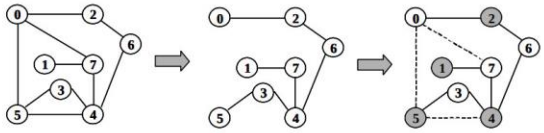
2. PROBLEMA

La pregunta por resolver es: Dado una lista de clientes ubicados en un mapa vial bidimensional, un depósito de inicio y fin, y unas restricciones de tiempo y energía ¿cuáles son las rutas para un número libre de camiones eléctricos, para visitar todos los clientes, minimizando el tiempo total, que es la suma del tiempo del recorrido más el tiempo que toman las recargas de batería? [1]

3. TRABAJOS RELACIONADOS

3.1 GRAFOS BIPARTITOS, CICLOS IMPARES Y COLORACIÓN [3]

El problema de verificar si un grafo es bipartito es equivalente al problema de colorabilidad de grafos para dos colores, simplemente son el mismo problema, pero con diferente nomenclatura. Con respecto al problema de detección de la existencia de ciclos impares, cualquier grafo con un ciclo de longitud impar es claramente no colorable para dos colores. En vista de que los tres problemas son equivalentes es posible elaborar un solo algoritmo que resuelva los tres problemas. Para colorear un grafo con dos colores, dado un color asignado a un vértice v, debe colorearse el resto del grafo asignando el otro color a cada vértice adyacente a v. Este proceso es equivalente a alternar los dos colores para los nodos en cada nivel de un árbol DFS, y durante el retorno de las llamadas recursivas verificar si no hay arista que conecte dos nodos del mismo color, ya que tal arista es evidencia de un ciclo de longitud impar. A continuación, ilustraremos esta idea, usando el árbol DFS de la sección de recorrido de grafos. La primera figura muestra el grafo original, la segunda el árbol DFS y la tercera es el grafo coloreado a blanco y gris, usando la heurística descrita anteriormente. Una vez construido el árbol DFS y coloreado los nodos, es sencillo comprobar que las aristas 5-4 y 0-7 conectan dos nodos del mismo color, lo cual es evidencia de la existencia de un ciclo de longitud impar. De hecho, el ciclo 0-5-3-4-7-0 es un ciclo de longitud impar. Si eliminamos 5-4 y 0-7 el grafo sería colorable y no habría ciclos de longitud impar.



Grafica 1: árbol DFS

3.2 COMPONENTES FUERTEMENTE CONEXAS

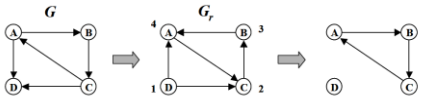
Una componente fuertemente conexa de un dígrafo es un conjunto máximo de nodos en el cual existe un camino que va desde cualquier nodo del conjunto hasta cualquier otro nodo del conjunto. Si el dígrafo conforma una sola componente fuertemente conexa, se dice que el dígrafo es fuertemente conexo. [3] Una solución para esto sería aplicar

el método Kosaraju, que a su vez usa DFS para recorrer el grafo y hallar la solución.

-Se ejecuta un DFS y se construye un dígrafo nuevo Gr, invirtiendo las direcciones de todos los arcos de G. Esto es, se permutan los nodos del grafo en el orden definido por el recorrido en postorden del mismo (para un dígrafo acíclico este proceso es equivalente a un ordenamiento topológico). Se deben enumerar los nodos en el orden de terminación de las llamadas recursivas del DFS.

- Luego se ejecuta un DFS en Gr, partiendo del nodo con numeración más alta de acuerdo con la numeración asignada en el paso anterior. Si el DFS no llega a todos los nodos, se inicia la siguiente búsqueda a partir del nodo restante con numeración más alta.

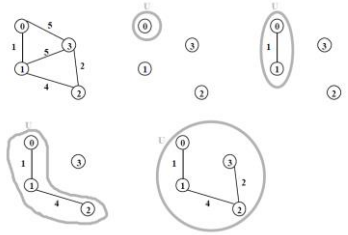
- Cada árbol del conjunto de árboles DFS resultante es una componente fuertemente conexa de G.



Grafica 2: ilustración método de Kosaraju

3.3 ÁRBOL DE EXPANSIÓN MÍNIMA [3]

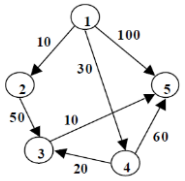
Encontrar la manera menos costosa de conectar todos los puntos, entonces nos enfrentamos al problema de encontrar un árbol de expansión mínima (MST – Minimun Spanning Trae). Un árbol de expansión de un grafo conexo es un subgrafo que contiene todos los nodos del grafo y no tiene ciclos. El árbol de expansión mínima de un grafo pesado no dirigido es el árbol de expansión cuyo peso (la suma de los pesos de todas sus aristas) no es mayor al de ningún otro árbol de expansión. El algoritmo de Prim es tal vez el algoritmo de MST mas sencillo de implementar y el mejor método para grafos densos. Este algoritmo puede encontrar el MST de cualquier grafo conexo pesado. El siguiente ejemplo ilustra el funcionamiento del algoritmo. La secuencia de ilustraciones va de izquierda a derecha y de arriba hacia abajo. La primera imagen muestra el grafo pesado y las siguientes muestran el funcionamiento del algoritmo de Prim y como va cambiando el conjunto U durante la ejecución.



Grafica 3: *ilustración funcionamiento algoritmo de Prim*

3.4 EL CAMINO MAS CORTO

En la Teoría de Grafos, uno de los problemas más conocido es el del camino más corto. El problema consiste en encontrar un camino entre dos vértices (o nodos) de tal manera que la suma de los pesos de las aristas que lo constituyen es mínima. [4] Una solución sería: El algoritmo de dijkstra determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean pesos.



Grafica 4: *ilustración de* dijkstra

4. AGENTE VIAJERO Y DIVISIÓN DEL PROBLEMA COMO SOLUCIÓN AL RUTEO DE VEHÍCULOS ELÉCTRICOS

Utilizamos una estructura en c++ para almacenar los datos de los nodos, a saber: identificación en la variable id de tipo entero, nombre en la variable name de tipo string, coordenada x, y en las variables de tipo float x, y respectivamente, el tipo de estación (de carga, depósito o cliente) en la variable type de tipo char y el tipo de estación (para estaciones de carga) en la variable station_type de tipo entero. Organizamos todos los nodos en un vector de nodos, y también tenemos un vector para los nodos de carga, ya que planeamos para la solución que implementaremos hacer una división de los nodos por regiones, finalmente planteamos la utilización del agente viajero como una solución óptima que bien podría cambiar en un futuro.

4.3 Criterios de diseño de la estructura de datos

La implementación que se tienen en el momento nos parece que es eficiente ya que nos permite dividir el problema por camiones para diferentes sectores y así abastecer los clientes y tener a disposición siempre la estación de carga más cercana para no tardar en ir y cargar la complejidad seria entonces relativamente baja, aunque aún no sabemos si la podemos reducir más.

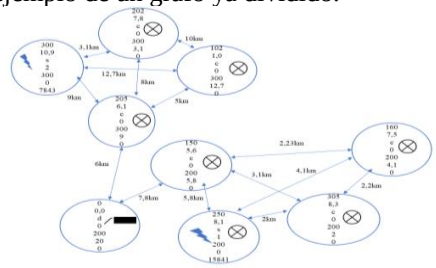
5. AGENTE VIAJERO Y EL RUTEO DE VEHÍCULOS ELÉCTRICOS

La estructura de datos diseñada esta implementada en el lenguaje de programacion c++, esta estructura de datos calcula el tiempo para abastecer el total de clientes de una empresa con vehículos eléctricos, busca en si la forma aproximada mas optima para realizar dicha tarea teniendo las consideraciones necesarias como la duración de la batería tiempo de estadía en cada nodo (que representa los clientes), tiempo total que un camión puede ser utilizado y el tiempo de carga de la batería, el recorrido a través del grafo se hace por medio del algoritmo del agente viajero que calculo cual es el nodo más cercano y se traslada hasta este repitiendo el proceso cada vez además el grafo esta dividido en varias regiones teniendo en cuenta cual es la estación de carga mas cercana, en caso tal de que un camión no alcance a realizar la ruta entera de la región en 10 horas este termina la ruta y un camión nuevo empieza el recorrido

de lo faltante se tiene en cuenta por supuesto del tiempo que demora en regresar al depósito.

5.1 Estructura de datos

los datos entrantes se dividen por subregiones de acuerdo con la estación de carga mas cercana sin importar el tipo de estación y desde allí se conecta el depósito a cada nodo mas cercano de cada subregión, en la grafica 5 se muestra un ejemplo de un grafo ya dividido.



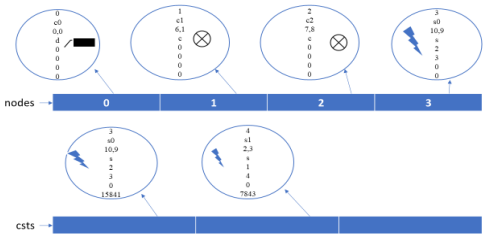
***Grafica 5*:** Grafo dividido en subregiones de acuerdo con la estación de carga más cercana.

5.2 Operaciones de la estructura de datos

Creación: el sistema lee un archivo con las características de las siguientes imágenes, los primeros datos se guardan en variables globales para luego poder ser utilizados sin ningún problema, luego de esto vienen las coordenadas, para esto se crea un vector de nodos, la clase nodo contiene cada uno de los datos propios de estos, como su tipo, su identificación entre otros datos, también se identifican las estaciones y se guardan en un vector diferente, luego se lee tanto las horas de carga como la batería que suministra cada tipo de estación y se genera para cada estación la pendiente de acuerdo a su tipo de carga como esta en la gráfica 6. Por último, se divide cada nodo por estación mas cercana y se generan grafos con respectivas distancias por cada región, en un map de vectores y un map de matrices respectivamente.

```
n = 3
m = 1
u = 2
breaks = 4
r = 125.0
speed = 40.0
Tmax = 10.0
Smax = 2.04
```

```
st_customer = 0.5
Q = 16000.0
Coordinates
0 depot 45.35 5.63 d 0
1 c0 6.01 9.52 c 0
2 c0 10.01 9.52 c 0
3 s0 82.04 37.54 s 1
4 s1 18.29 102.2 s 0
i
0 0.31 0.39 0.51
0 0.62 0.77 1.01
0 1.26 1.54 2.04
g
0 13600 15200 16000
0 13600 15200 16000
0 13600 15200 16000
```



***Grafica 6*:** Imagen de vectores de creación al momento de leer el archivo fuente.

Luego de esto se divide por regiones como se muestra en la grafica 5 para esto se crea un map de vectores que contienen los nodos “rs”, y un map de grafos o de matrices “graphs” que contiene solo coordenadas de acuerdo con el id de los nodos y en estas posiciones están las distancias entre si calculadas con el método “dist”, la identificación en los map se hace mediante el id de las estaciones de carga así siempre se lleva control de las regiones.

5.3 Criterios de diseño de la estructura de datos

la eficiencia de los vectores en c++ por ser variables los tamaños ahorran tiempo de ejecución y memoria ya que acceder a ellos y agregar es O(1) también los mapas en caso promedio es O(1) sus operaciones y en el peor de los casos es O(n) lo que nos da mucha ventaja en tiempo de ejecución en comparación con otras estructuras de datos además los mapas nos permiten una identificación mas exacta para lo que se necesitaba en la estructura.

5.4 Análisis de Complejidad

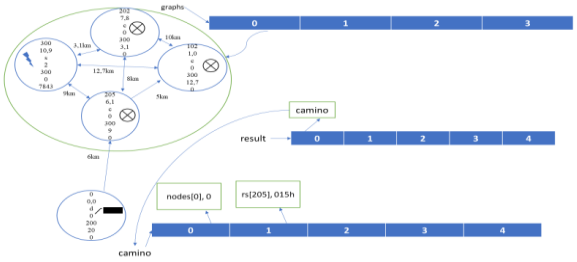
Complejidad de las operaciones de la estructura diseñada.

Método	Complejidad
Lectura	**O(n) (n el total de nodos)**
División en regiones	**O(n*m)(n el total de nodos y m de estaciones de carga)**
Creación de grafos	**O(n*(m^2))(n el número de regiones y m el número de nodos en la región)**
Guardar un camino	**O(1)**
Tsp, vecino más cerca	**O(n^2)(n el número de nodos en la región)**
Imprimir ruta	**O(n*m)(n es el número de caminos y m el número nodos en la ruta o camino)**

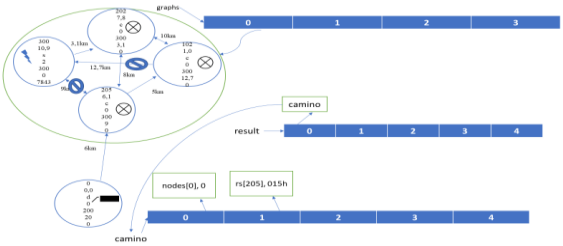
Tabla 1: Tabla para reportar la complejidad de las operaciones

5.5 Algoritmo

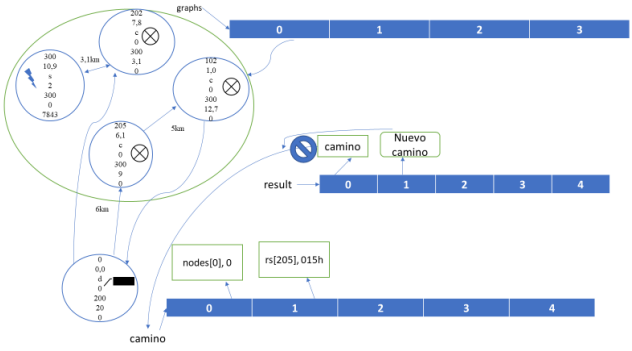
Recorrido: como se muestra en la gráfica 7 las flechas de un solo sentido muestran el recorrido del camión (algoritmo) siempre se escoge el vecino más cercano de cada nodo y además no toma en cuenta si esta es de tipo “s” o estación, siempre se verifica el nivel de batería para ir al siguiente nodo, si este no es suficiente se dirige a la estación de carga asignada de la región, también se hace verificación del tiempo que lleva el camión a este tiempo se le hace un tiempo de gracia de 1.5 horas para estar seguros que volverá en 10 horas al depósito en caso tal de que este alcance el tiempo limite la ruta que estaba siendo guardada en un vector se guarda en otro vector llamado “result” y arranca otro camión desde el nodo 0 o depósito y se limpia el vector auxiliar “camino”, cabe aclarar que cada nodo visitado se marca con una valor booleano de “true” en un vector de tipo bool.



Grafica 7: recorrido de la estructura para una sola region.



Grafica 8: operación de recarga, se vuelve a calcular desde la estacion el nodo mas cercano sin importar donde iba antes de llegar



Grafica 9: muestra el retorno de un camion por cumplir sus horas de servicio, inicia otro desde el deposito calculando el mas cercano, se agrega un nuevo vector de “camino” a “result”.

Carga de bateria: la carga de la bateria es una funcion lineal en partes anteriores se menciono que para cada estacion se calcula la pendiente con el maximo de bateria no teneiendo en cuenta los otros niveles ya que no lo vimos necesario, con esto y otra variable involucrada en la ecuacion, sea nivel de bateria o distancia (en kilometros) se puede saber en cualquier momento la tercera de estas es asi como sabemos cuanto necesitamos en determinado momento sea en dsitancia o bateria.

5.6 Cálculo de la complejidad del algoritmo

La complejidad del algoritmo resulta ser la cantidad de subregiones por la cantidad de nodos al cuadrado de cada subregión que resulta siendo la cantidad de nodos totales al cuadrado puesto que cada subregión contiene los nodos iniciales, la complejidad en si del algoritmo es la mencionada anteriormente, imprimir los datos si resulta un poco mas costosa ya que se recorre un vector de vectores en donde estos últimos contienen la ruta.

Sub problema	**Complejidad**
Creación de la estructura	O(r*(m^2))
Tsp para cada subregión	O(r*(m ^2))
Imprimir las rutas	O(t*(k))
Complejidad Total	O(2(r*m^2) + t*k)

Tabla 2: donde r representa la cantidad de regiones, m es la cantidad de nodos es tal región, t es el numero de rutas creadas o encontradas y k la cantidad de nodos en cada rut encontrada.

5.7 Criterios de diseño del algoritmo

Consideramos que al asignarle una ruta a varios camiones podríamos reducir el tiempo además se tenia que tener en cuenta que los camiones solo pueden operar cierto número de horas, por esto decidimos que para reducir lo tedioso de recargar la batería a cada nodo le asignáramos una estación de carga dividir tal grafo en regiones en donde el centro de cada región fuese una estación de carga sin importar el tipo, ya que si solo escogíamos cierto tipo de estación podría un nodo o varios estar muy lejos de una estación rápida, por esto quisimos asignarlas sin importar su tipo, ahora bien para cada subregión lanzamos tsp con un costo de O(n^2) siendo n la cantidad de nodos en una subregión que en si son los nodos totales ya que los nodos de cada subregión suman el total, con esta estructura ahorramos tiempo en el viaje a recargar batería, en el recorrido de nodo a nodo y además revolvemos el problema del tiempo máximo de cada camión ya que este antes de hacer el traslado a otro nodo verifica si se pasas de un tiempo máximo si lo hace verifica su cantidad de batería en caso tal de que no alcance

para llegar al deposito recarga y luego se dirige al deposito y de nuevo se calcula el nodo mas cercano para que empiece otro camión a recorrer y termine dicha región, tanto el tiempo de ejecución como el tiempo en horas de la respuesta resulta satisfactorio ya que desde una primera solución logramos reducir el tiempo así como también satisface cada una de las necesidades del problema planteado.

5.8 Tiempos de Ejecución

	24f0.txt	24t0.txt	38f0.txt	38t0.txt
Mejor caso	19.6ms	20.8ms	19.2ms	21.3ms
Caso promedio	20.5ms	21.53ms	22.2ms	22.6ms
Peor caso	21.4ms	22.9ms	24.7ms	25.2ms

Tabla 3: Tiempos de ejecución del algoritmo con diferentes conjuntos de datos

5.9 Memoria

	24f0.txt	24t0.txt	38f0.txt	38t0.txt
Consumo de memoria	2 MB	3 MB	3 MB	3 Mb

Tabla 4: Consumo de memoria del algoritmo con diferentes conjuntos de datos.

En la tabla anterior se muestra el uso de la memoria la ejecución es en milisegundos y el programa para, por esto es poca la memoria que usa en tiempo de ejecución ya que es muy rápido, además podríamos considerar que usa memoria solo para guardar vectores y matrices.

5.10 Análisis de los resultados

Algoritmo para ruteo de vehículos eléctricos	Tsp(vecino mas cercano)	Vector de grafos
Espacio en le Heap	**1MB**	**1 MB**
Ruteo para 24f0.txt	**2.2ms**	**10.3ms**
Ruteo para 24t0.txt	**3.3ms**	**9.8ms**
Ruteo para 38f0.txt	**2.4ms**	**10.8ms**
Ruteo para 38t0.txt	**2.1ms**	**10.8ms**
Ruteo para 38f1.txt	**1.9ms**	**11ms**

Tabla 5: Análisis de los resultados obtenidos con la implementación del algoritmo

Como se puede observar en la tabla anterior hacer le recorrido de las subregiones tiene un muy buen tiempo de ejecución y su consumo de memoria no es elevado, el consumo de memoria en si va ligado a el almacenamiento de los datos en los diferentes vectores y matrices al igual que el tiempo de lectura y creación de los datos es mayor ya que se lee, se divide y luego se generan los grafos teniendo en si un tiempo de ejecución mas elevado que el de solo hacer tsp para una cantidad de estaciones o regiones.

6. CONCLUSIONES

1. Los resultados de la estructura y algoritmo diseñados arrojan resultados satisfactorios, con un tiempo de ejecución corto y poca memoria usada, creemos que es una buena técnica dividir el problema, puesto que se reduce el tiempo de carga y por consiguiente el del recorrido.

2. En la solución final los resultados no son los exactos puesto que la solución más eficiente aun no es posible de encontrarla, pero creemos que para un problema tan complejo generar un tiempo de 17 días mas o menos para cada archivo de datos es una buena solución considerando la cantidad de nodos o clientes que contienen cada uno y las diferentes variables que afectan el problema

3. En nuestra primera solución la mayoría de los archivos pasaban de 600 y 700 horas por lo cual no considerábamos que fuera un buen tiempo de solución mientras que al llegar a dividir cada subregión por estación mas cercana el tiempo se redujo considerablemente no pasando de 500 horas ningún archivo y generando para cada uno de los archivos un numero aceptable de camiones ente 50 y 70 que es la mitad o menos de la mitad de los nodos en cuanto a lo mencionado en las primeras paginas de este documento, hemos implementado el agente viajero y por su puesto con las respectivas modificaciones necesarias para el problema planteado los demás algoritmos mencionados nos parece que no satisfacen en su totalidad las cualidades necesarias planteadas así que hemos escogido el que nos aparece es el algoritmo más acertado

4. La implementación puede ser extendida a otros algoritmos que bien pueden aumentar el tiempo de ejecución, pero disminuyen el tiempo de respuesta o de solución, en unos primeros momentos los inconvenientes se causaron en el momento de calcular los vecinos y costos de ir a ellos puesto que algunos no tenían vecinos por lo cual las variables no cambiaban su valor y los resultados eran erróneos.

6.1 Trabajos futuros

Aun puede ser mejorado con la implementación del mismo algoritmo pero utilizando otra estructura de datos, podríamos en un futuro hacer esta mejora y ver que pasa con el tiempo de ejecución aun así la respuesta es probablemente la misma, para mejorar la respuesta nos gustaría hacer la implementación de otro algoritmo que pueda aprovechar mejor los 30 sí que nos dan como meta para hallar una buena solución, si bien el tiempo de

ejecución aumentaría aun seria aceptable y posiblemente la solución sería aún más acertada.

AGRADECIMIENTOS

Esta investigación fue soportada parcialmente por el programa de becas con aportes de empleados EAFIT.

Esta investigación fue soportada parcialmente por el programa de créditos condonables, Alianza MINTIC – MEN – ICETEX (ACCES)

Nosotros agradecemos por la ayuda con la metodología a Agustín Nieto, estudiante de EAFIT, por los comentarios que nos hizo para mejorar el código de la lectura del archivo.

Nosotros queremos agradecer Alejandro Montoya profesor de Ingeniería de Producción por su charla e información sobre el problema tratado en este informe.

REFERENCIAS

[1] Restrepo, A., & Toro, M. 2018. Algoritmo para ruteo de vehículos eléctricos (2nd ed.). https://github.com/mauriciotoro/ST0247-Eafit/tree/master/proyecto, March 4, 2018, Medellín.

[2] ROCHA MEDINA, Linda Bibiana; GONZÁLEZ LA ROTTA, Elsa Cristina; ORJUELA CASTRO, Javier Arturo. Una Revisión al Estado del Arte del Problema de Ruteo de Vehículos: Evolución Histórica Y Métodos De Solución. **Ingeniería**, [S.l.], v. 16, n. 2, p. 35-55, dec. 2011. ISSN 2344-8393. Disponible en: <https://revistas.udistrital.edu.co/ojs/index.php/reving/article/view/3832>. Date accessed: 03 mar. 2018. doi:https://doi.org/10.14483/23448393.3832.

[3] Coto, E. 2003. Algoritmos Básicos de Grafos. http://ccg.ciens.ucv.ve/~ernesto/nds/CotoND200302.pdf, March 4, 2018, Caracas.

[4] 3.1. El camino más corto. n.d. E-ducativa.catedu.es. http://educativa.catedu.es/44700165/aula/archivos/repositorio//4500/4724/html/31_el_camino_ms_corto.html, March 4, 2018.

[5] Arias Figueroa, J. 2013. CAMINO MAS CORTO: ALGORITMO DE DIJKSTRA. Algorithms and More. https://jariasf.wordpress.com/2012/03/19/camino-mascorto-algoritmo-de-dijkstra/, March 4, 2018.

Adobe Illustrator, utilización el 14 de mayo de 2018. Descarga hecha de: http://www.adobe.com/Illustrator.

Grafica 1: Coto, E. 2003. Algoritmos Básicos de Grafos. http://ccg.ciens.ucv.ve/~ernesto/nds/CotoND200302.pdf, March 4, 2018, Caracas.

Grafica 2: Coto, E. 2003. Algoritmos Básicos de Grafos. http://ccg.ciens.ucv.ve/~ernesto/nds/CotoND200302.pdf, March 4, 2018, Caracas.

Grafica 3: Coto, E. 2003. Algoritmos Básicos de Grafos. http://ccg.ciens.ucv.ve/~ernesto/nds/CotoND200302.pdf, March 4, 2018, Caracas.

Grafica 4: Coto, E. 2003. Algoritmos Básicos de Grafos. http://ccg.ciens.ucv.ve/~ernesto/nds/CotoND200302.pdf, March 4, 2018, Caracas.