MEJORÍA EN LA MOVILIDAD DEL TRÁFICO EN MEDELLÍN: ALGORITMO AIGNADOR DE RUTAS A EMPLEADOS DE UNA EMPRESA PARA LA OPTIMIZACIÓN DE LAS VÍAS.

Juan Sebastián Sanín V.
Universidad EAFIT
Colombia
jssaninv@eafit.edu.co

Juan Pablo Peña F.
Universidad EAFIT
Colombia
jppenaf@eafit.edu.co

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

RESUMEN

El problema nos propone un déficit en la movilidad en Medellín y en su medio ambiente el cual ya afecta a sus habitantes, para este problema implementaremos un algoritmo el cual va a hacer un orden en los vehículos que llegan únicamente a el conductor a su lugar de trabajo, para que así recoja a sus compañeros en el trayecto sin verse afectado. El problema es muy importante para la mejora del manejo de las estructuras de datos, ya que necesitamos generar respuesta en el menor tiempo posible (4 segundos) y esto nos generará una mejor actitud al enfrentarnos a estos problemas ya que son situaciones que vemos diariamente y necesitan una respuesta urgente. Hay una gran cantidad de problemas con respecto a esta temática, como lo es, repartir periódicos en el recorrido de menor distancia, llegar a un lugar en específico con varias posibilidades de rutas diferentes, etc.

Palabras clave

- Matriz
- Voraz
- Tiempo
- Memoria
- Datos
- Ruta

Palabras clave de la clasificación de la ACM

Solo las que están en esta lista de ACM, en http://bit.ly/2oVE5 2i
No pueden ser inventadas por ustedes.

Ejemplo: Theory of computation → Design and analysis of algorithms → Graph algorithms analysis → Shortest paths

1. INTRODUCCIÓN

Abordando el problema desde la ciudad de Medellín, se ha notado el aumento notable de la contaminación del aire, con un gran aporte debido al crecimiento de la venta de vehículos en la ciudad, lo cual también empeora el tráfico de esta, puesto que es uno de los medios de transporte más útiles para las personas y con el cual pueden llegar a su trabajo o estudio en su propio vehículo. Estos factores perjudican enormemente la calidad de vida, no solo de los habitantes de Medellín, sino del área metropolitana, debido a que afecta la salud de las personas que respiramos el aire tan contaminado y además al alterar el tráfico las personas tendrían que madrugar más a cumplir sus obligaciones, lo cual genera más caos y estrés.

2. PROBLEMA

La problemática a resolver consiste en: En Antioquia, más exactamente en Medellín, se presenta una alta congestión de automóviles y otros transportes, donde podemos visualizar que muchas personas utilizar autos ocupando únicamente un espacio, lo que se quiere mejorar con este proyecto es directamente a dicho problema de movilidad, en el cual las empresas propongan a sus empleados que a medida que vayan en el trayecto al trabajo, recojan a sus compañeros para así evitar el uso de más autos sin ocupar completamente todos los espacios y puedan hacerlo sin tener una alta variación en el tiempo normal en sus trayectos a la empresa.

3. TRABAJOS RELACIONADOS

3.1 El problema de Santiago

Algoritmo para encontrar el camino más corto de un punto de inicio a un punto final implementando backtracking.

El problema consiste en donde dos personas quieren hacer el recorrido de Sevilla hasta el santuario de Santiago el apóstol en España, una de las dos personas marca el camino más corto recorriendo cada ciudad como él lo presenta, mientras que la otra persona quiere hacer un recorrido más alterno. Para este problema se hace una solución mediante el recorrido de grafos (cada nodo sería cada ciudad) usando backtracking, haciendo una comparativa de cada recorrido, de una ciudad a otra, pero a medida que encuentre el recorrido con menor distancia no siga haciendo una comparativa innecesaria, si no que nos informe del camino ya registrado más corto.

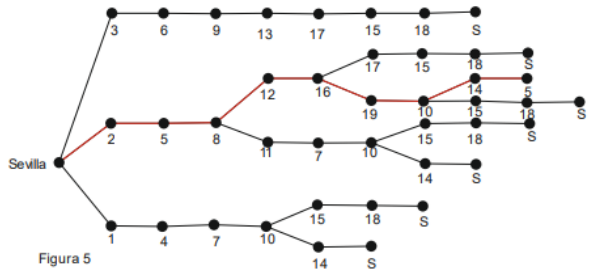


Figura 1. Representación del problema uno con sus respectivos valores de distancia entre cada ciudad(nodo) hasta el santuario (nodo final).

3.2 Problema de la gira del caballo

Algoritmo para que un caballo en un tablero de ajedrez vaya a todos los cuadros.

Este problema consiste en un caballo del ajedrez, el cual tiene que visitar todos los cuadros del tablero una única vez. Sabemos que en un tablero de 8 x 8 el número de giras posibles es de 1.305x10^35. La solución para este problema sería representar cada cuadro con los movimientos posibles del caballo en un grafo y usar un algoritmo de recorrido, el cual sería apto para este problema si se plantea un tablero de pequeñas dimensiones, la fuerza bruta, para así ir guardando cada gira del caballo.

3.3 El recorrido del cartero

Algoritmo para que una persona recorra un barrio pasando por todas las calles y minimizando el número de calles que debe recorrer más de una vez.

Este problema consta de un cartero que tiene que salir de la oficina de correos y tiene que recorrer todas las calles entregando correos y volver a la oficina. Se tiene en cuenta que una calle va de una esquina a la otra y que la oficina de correos está ubicada en una esquina. Se busca que el recorrido minimice el número de calles que el cartero está obligado a recorrer más de una vez.

3.4 El problema de la mesa redonda

Algoritmo para para encontrar el número de permutaciones que se deben hacer para que todos los elementos de un conjunto de un tamaño de 7 puedan ser combinados todos entre sí.

Un grupo de 7 personas acuerdan cenar juntas en diferentes ocasiones. En cada ocasión se sientan alrededor de una mesa redonda de modo que cada persona tiene a sus dos lados comensales distintos en cenas diferentes. Se busca saber cuántos días deben citarse para cenar si todos quieren sentarse junto a todos los demás.

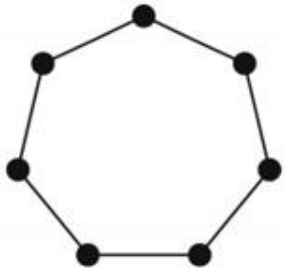


Figura 2.

Figura 3.

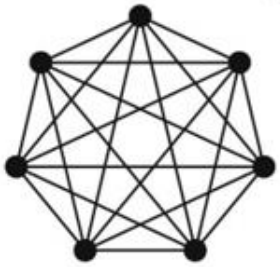


Figura 4.

4. TÍTULO DE LA PRIMERA SOLUCIÖN DISEÑADA

A continuación, explicamos la estructura de datos y el algoritmo.

4.1 Estructura de datos

La estructura de datos que se va a utilizar es la **Matriz**, ya que como todos los nodos están conectados entre sí la mejor forma de guardar el tiempo de recorrido de un nodo con otro es con esta estructura.

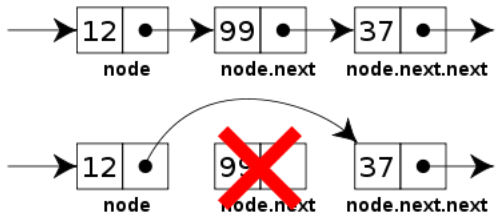
	1	2	3	EAFIT
1	0 min	4 min	8 min	12 min
2	4 min	0 min	6 min	20 min
3	8 min	6 min	0 min	5 min
EAFIT	12 min	20 min	5 min	0 min

Figura 5. Matriz. La primera fila y la primera columna representan los nodos, el resto la distancia en tiempo entre estos.

El tiempo que tarda un nodo en llegar a el mismo es 0 (cero), por tanto, la diagonal de la matriz siempre va a ser este número.

4.2 Operaciones de la estructura de datos

Diseñen las operaciones de la estructura de datos para solucionar el problema eficientemente. Incluyan una imagen explicando cada operación



Gráfica 2: Imagen de una operación de borrado de una lista encadenada

4.3 Criterios de diseño de la estructura de datos

Inicialmente, pensamos en diseñar la estructura de datos en forma de matriz debido al espacio en memoria que utiliza, ya que, en comparación con otras estructuras, en este caso la matriz es la que menos ocupa.

Además, usamos esta estructura porque se nos hace sencillo el manejo de los datos que en este caso es la distancia en tiempo de un nodo a otro, y gracias a esto se puede lograr reducir el código para un mejor entendimiento.

4.4 Análisis de Complejidad

Método	Complejidad
Creación de la Matriz	O(n^2)

Tabla 1. Complejidad de la operaciones para el peor de los casos.

4.5 Algoritmo

```
public static void main(String[] args) {
        int h;
        int count =0;
        try{
        int i=0;
        File f = new File("data.txt");
        Scanner s = new Scanner(f);
        while(s.hasNextLine()){
        String line = s.nextLine();
        String cortar [] = line.split(" ");
        String vx=cortar[i];
        String vy=cortar[i+1];
        String vz=cortar[i+2];
           System.out.println(vx);
//   int Vx= Integer.parseInt(cortar[i]);
//            int Vy= Integer.parseInt(cortar[i+1]);
//            int Vz= Integer.parseInt(cortar[i+2]);
          // Orden[vx][vy]=vz;
           matriz(vx,vy,vz);
          }
         // }
      // }
       }
       catch(Exception e){
      }
  }
 public static  void matriz (String vx, String vy, String vz){
    int Vx= Integer.parseInt(vx);
    int Vy= Integer.parseInt(vy);
   Orden[Vx][Vy]=vz;
 }
```

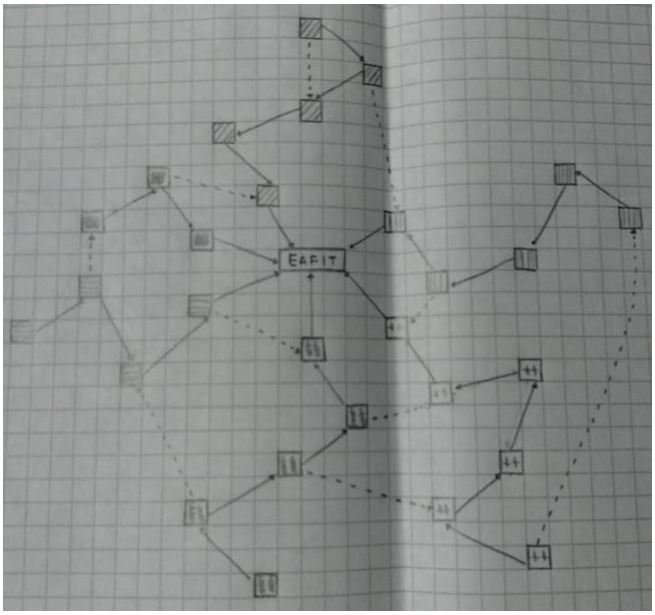


Figura 6. Gráfica del algoritmo.

4.6 Cálculo de la complejidad del algoritmo

Calculen la complejidad del algoritmo para el peor de los casos, el mejor de los casos y el caso promedio

Sub problema	Complejidad

Crear el grafo de *Bruijn* con las secuencias	O(S)
Actualizar el grafo de *Bruijn* con las secuencias	$O(A.V^2)$
Encontrar los genes	O(S)
Complejidad Total	$O(A.S^2 + V)$

Tabla 2: complejidad de cada uno de los sub problemas que componen el algoritmo. Sea A la longitud de una secuencia de ADN, S el número de secuencias de ADN, y V el número de K-meros diferentes que se obtienen de las secuencias de ADN.

NOTA: Sin complejidad total, el análisis no sirve de nada

4.7 Criterios de diseño del algoritmo

Expliquen por qué diseñaron así el algoritmo. Usen criterios objetivos. Criterios objetivos son, por ejemplo, la eficiencia en tiempo y memoria. Criterios no objetivos y que rebajan la nota son: “me enfermé”, “fue la primera que encontré”, “la hice el último día”, etc. Recuerden: este es el numeral que más vale en la evaluación con 40%.

4.8 Tiempos de Ejecución

Calculen, (I) el tiempo de ejecución y (II) la memoria usada del algoritmo, para el Conjunto de Datos que está en el ZIP:

Tomen 100 veces el tiempo de ejecución y memoria de ejecución, para cada conjunto de datos

	Conjunto de Datos 1	***Conjunto de Datos 2***	***...Conjunto de Datos n***
Mejor caso	10 sg	20 sg	5 sg
Caso promedio	12 sg	10 sg	35 sg
Peor caso	15 sg	21 sg	35 sg

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

4.9 Memoria

Mencionar la memoria que consume el programa para varios ejemplos

	Conjunto de Datos 1	***Conjunto de Datos 2***	***...Conjunto de Datos n***
Consumo de memoria	10 MB	20 MB	5 MB

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

Para medir la memoria que consume un programa, se utilizan generadores de perfiles (en Inglés, profilers). Uno muy bueno para Java es VisualVM, desarrollado por Oracle,http://docs.oracle.com/javase/7/docs/technotes/guides/visualvm/profiler.html No dejen de usarlo en sus proyectos y en la vida. Para usarlo hay que generar un .jar que es como un ejecutable de Java. En Netbeans "martillo con escoba" y en BlueJ "archivo, generar .jar".

4.10 Análisis de los resultados

Expliquen los resultados obtenidos. Hagan una gráfica con los datos obtenidos, como por ejemplo:

Tabla de valores durante la ejecución

Estructuras de autocompletado	LinkedList	Arrays	HashMap
Espacio en el Heap	60MB	175MB	384MB
Tiempo creación	1.16 - 1.34 s	0.82 - 1.1 s	2.23 - 2.6 s
Tiempo búsqueda ("a")	0.31 - 0.39 s	0.37 - 0.7 s	0.22 - 0.28 s
Tiempo búsqueda ("zyzzyvas")	0.088 ms	0.038 ms	0.06 ms
Búsqueda ("aerobacteriologically")	0.077 ms	0.041 ms	0.058 ms
Tiempo búsqueda todas las palabras	6.1 - 8.02 s	4.07 - 5.19 s	4.79 - 5.8 s

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

REFERENCIAS

1. Núñez S., R., Núñez V., J., Paluzo H., E. y Salguero Q., E. (2016). *Jugueteando con Grafos*. [online] Fisem.org. Disponible en: http://www.fisem.org/www/union/revistas/2016/46/10_21-401-1-ED.pdf. 4-6.

2. Miller, B., Ranum, D. and College, L. (2019). *Solución de problemas con algoritmos y estructuras de datos*. [online] Interactivepython.org. Disponible en: http://interactivepython.org/runestone/static/pythoned/Graphs/ElProblemaDeLaGiraDelCaballo.html.

3. Mate.dm.uba.ar. (2007). [online] Disponible en: http://mate.dm.uba.ar/%7Espuddu/teo_de_grafos/grafo1.doc.

4. Núñez S., R., Núñez V., J., Paluzo H., E. y Salguero Q., E. (2016). *Jugueteando con Grafos*. [online] Fisem.org. Disponible en:

http://www.fisem.org/www/union/revistas/2016/46/10_21-401-1-ED.pdf. 10-11.