Algoritmos Avanzados
Laboratorio I
Fuerza Bruta

FRANCO LEAL GUARDA

Profesora: Mónica Villanueva

Ayudante: Gerardo Zúñiga

Fecha: 04 de Mayo 2018

Compiled May 4, 2018

En este informe, se presenta el problema y solución correspondientes al primer laboratorio de algoritmos Avanzados, el cuál consiste en el uso de un algoritmo de fuerza bruta para encontrar el camino más corto entre dos nodos de un grafo. La estructura dele informe consiste en un análisis del enunciado, identificación de objetivos, marco teórico. Luego, se procede a describir características principales del programa y se da respuesta a las 5 preguntas típicas para un algoritmo.

1. INTRODUCCIÓN

El problema presentado en el enunciado consiste en buscar, mediante la técnica de fuerza bruta, el camino más corto entre un par de nodos. Por lo tanto, se deben generar todas las combinaciones posibles dadas por el grafo entre el nodo de origen y el nodo de destino, para luego proceder a encontrar el de menor peso. Se procede a definir las estructuras de datos utilizadas para el desarrollo del programa, además de analizar la complejidad de la solución y concluir respecto al método de fuerza bruta.

Objetivo Principal:

1. Desarrollar un algoritmo que resuelva el problema planteado (Encontrar el camino más corto entre un par de nodos). El algoritmo debe ser implementado mediante fuerza bruta y el manejo de datos se debe realizar mediante el uso de archivos de texto de entrada y salida (Entrada.in, Salida.out).

Objetivos Secundarios:

1. Revisar el algoritmo respondiendo las 5 preguntas típicas para realizar una evaluación más detallada del mismo.

2. DESCRIPCIÓN DEL PROBLEMA

Don Luis quiere encontrar el camino más corto que le permita llegar desde su casa hasta el colegio pasando por la casa de todos los niños a los que debe retirar. Además, se le debe informar si debe salir más temprano o no en caso de que la planificación óptima exceda una hora y 45 minutos de viaje. Para lograr este objetivo, se debe realizar un programa escrito en el lenguaje C utilizando un algoritmo de fuerza bruta. Los datos se reciben a través de un archivo de texto de nombre "entrada.in", el cual contiene en la primera línea la cantidad de niños a recoger, seguido de las distancias entre los diversos nodos presentes en nuestro grafo. La solución se debe entregar en un archivo llamado "salida.out", cuyo formato consiste en el recorrido óptimo que debe realizar Don Luis, posiblemente seguido de un mensaje de alerta en caso de que requiera más tiempo para el viaje, finalizando con el tiempo en minutos que demorará en realizar el recorrido completo.

3. MARCO TEÓRICO

En esta sección se describen conceptos clave relacionados al análisis de la solución:

A. Algoritmo

Secuencia finita y ordenada de pasos que permite solucionar un problema, debe tener un estado inicial y entradas.

B. Fuerza bruta

Método de resolución de problemas que consiste en generar todas las soluciones posibles dadas por el problema, para luego

buscar entre éstas las que son válidas.

C. Tiempo de ejecución

Tiempo total que demora un algoritmo en ser ejecutado. Las instrucciones básicas poseen un tiempo de ejecución constante

D. Lenguaje C

Lenguaje de programación perteneciente al paradigma imperativo procedural. Permite controlar el uso de memoria (asignar, liberar, reasignar), además de permitir la creación de estructuras de datos.

4. DESCRIPCIÓN DE LA SOLUCIÓN

El problema se abordó primero leyendo el archivo de entrada y generando la matriz de adyacencia del grafo. Luego, se inicializa el nodo 0 que corresponde a la raíz del árbol. Después, se comienza a llenar el árbol con todos los caminos posibles dada la matriz de adyacencia. Seguido de esto, se inicializa el camino en el que se guardará el camino final de menor peso y luego se realiza este análisis recorriendo el árbol. Una vez terminado este recorrido, en el camino inicializado anteriormente se encuentra la solución de menor tiempo para el recorrido. El programa se escribe en el lenguaje de programación C, haciendo uso de bibliotecas (stdio.h y stdlib.h) y de estructuras de datos definidas por el programador.

Estructuras de datos: Se definen diversas estructuras de datos entre las cuales tenemos:

- Node: Representa un nodo del árbol. Contiene el valor del nodo que representa, un puntero a su hijo izquierdo, un puntero a su hermano derecho y un puntero a su nodo padre.
- Adjacents: Estructura que contiene un puntero a entero en el cual se guardan todos los nodos adyacentes a otro nodo y la cantidad de éstos.
- Path: Estructura que contiene un puntero a entero con los nodos que conforman el camino, la cantidad de nodos y el peso del camino.
- GraphMatrix: Contiene un puntero doble a entero, en el cual se representa la matriz de adyacencia del grafo. También contiene un entero que representa la cantidad de nodos del grafo.

A. Funciones y procedimientos principales

Se utilizaron funciones principales y secundarias para la resolución del enunciado, las cuales se describen a continuación:

- addSon(): Este procedimiento almacena un nodo como hijo izquierdo del nodo padre o como hermano derecho del hijo izquierdo del nodo padre.
- wasVisited(): Función que revisa iterativamente si un nodo fue visitado en la línea de los padres del nodo del árbol que se entrega como parámetro.
- completeAdjacents(): Procedimiento que agrega en el árbol todos los nodos adyacentes del nodo que se entrega como parámetro.
- completeTree(): Procedimiento que agrega los adyacentes a todo los nodos que posee.
- getPath(): Función que devuelve el camino desde una hoja a la raíz.
- lowerDist(): Función que compara dos caminos y devuelve el que posea un peso menor.
- ǵetLower(): Función principal, recursiva. Se encarga de recorrer el árbol y cuando encuentra una hoja, obtiene el camino y actualiza en la variable global finalPath el camino de menor peso encontrado.

Algoritmo fuerza bruta():
Entradas: Matriz de adyacencia, nodo. **Salida:** Tipo void, no posee retorno. Se encarga de almacenar en finalPath el camino menor encontrado.

1. Inicializar variables locales.
2. **Si** hijo izquierdo distinto de nulo, **hacer**:
3. fuerza bruta (matriz,hijo izquierdo):
4. **Si no**
5. camino1 = obtener camino mediante padres
6. **Si** hermano derecho distinto de nulo, **hacer**:
7. fuerza bruta (matriz,hermano derecho):
8. **Si no**
9. camino2 = obtener camino mediante padres
10. camino3 = menorDistancia(camino1,camino)
11. caminoFinal = menor(camino3,caminoFinal)

Algoritmo obtenerCamino():
Entradas: Matriz de adyacencia, nodo, cantidad de nodos en grafo **Salida:** Camino obtenido a partir del árbol

1. Inicializar camino vacio
2. Mientras nodo distinto de nulo, hacer:
3. agregar nodo al camino
4. nodo = padre de nodo
5. n = n + 1
6. Si largo de camino menor a cantidad de nodos en grafo
7. devolver nulo
8. devolver el camino

5. ANÁLISIS DE LA SOLUCIÓN Y RESULTADOS

- ¿El algoritmo se detiene?
Si, se detiene cuando escribe los resultados finales en el archivo de salida. - ¿Resuelve el problema?
Si, determina el camino mínimo entre la casa del chofer y el colegio, agregando el tiempo que demora y si debe o no levantarse más temprano. - ¿Es eficiente?
A continuación, se procede a calcular el orden de complejidad - ¿Se puede mejorar?
Si, eliminando el uso de la variable global. - Otro método
Implementar la solución con backtracking, goloso u otro método de resolución de problemas.

A. Complejidad de la solución

La complejidad se puede calcular a partir de la cantidad de datos que posee la entrada del problema, utilizando la notación O()
- Complejidad getPath():
En este método, la sección que se presenta que acota superiormente la complejidad es el ciclo while, el cuál se ejecuta n veces, siendo n la cantidad de nodos (en el peor caso, todos).Por esto, la complejidad es

$$O(n)$$

- Complejidad getLower():
La complejidad de este algoritmo se puede determinar utilizando la fórmula de la recursión. Utilizando esta técnica, al ver que cada vez que se realiza un llamado recursivo se elimina un nodo del grafo, la función del tiempo quedaría como t(n)=2t(n-1)+o(1)

$$O(2^n)$$

- Complejidad leerArchivo():
En esta función se lee el archivo mediante un ciclo while, el cual se ejecuta en el peor de los casos n*(n-1). Por lo tanto la complejidad de este procedimiento es

$$O(n^2)$$

6. TRAZA DE LA SOLUCIÓN

La traza de la solución se analizará con el siguiente archivo de entrada:

3
0 1 2
0 2 3
1 2 4
1 3 1
1 21 12
2 1 4
2 3 8
3 1 12
3 21 10

Con este archivo, se genera el siguiente árbol

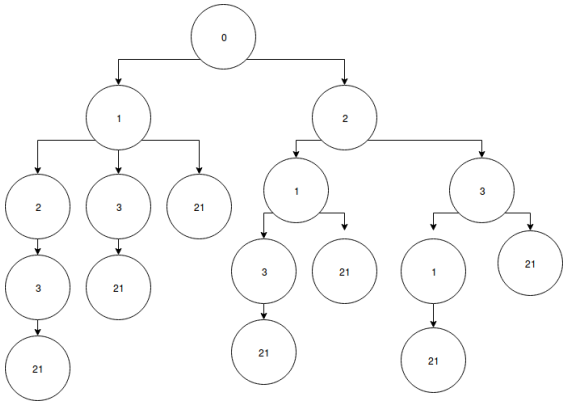

Fig. 1. Árbol archivo entrada ejemplo.

Luego, se comienza a trabajar con este árbol. Cuanto encuentra una hoja, obtiene el camino devolviéndose hasta la raíz y calculando el peso (en este caso, el camino sería 0-1-2-3-21, con peso 24). Luego de obtener este camino, luego recorre por 0-1-3-21, calcula su peso y no actualiza el valor ya que el camino no cumple con la condición de tener todos los nodos. Así, continua sucesivamente hasta encontrar que el camino de peso menor está dado por el orden: 0 - 2 - 1 - 3 - 21, con peso 18.

7. CONCLUSIONES

El uso de la fuerza bruta nos permite generar todos los estados posibles para un problema. Además, nos permite encontrar la mejor solución posible. Sin embargo, este método es ineficiente, ya que se generan estados que no son factibles, lo cual retrasa el procesamiento de la información, generando mayor tiempo de ejecución. El uso de árboles para solucionar el problema ayudó a la simplificación dela generación de los estados mediante el uso de la recursión. Existen diversos métodos para resolver este problema. Por ejemplo, al compartir información con otros estudiantes que se enfrentaron a este problema, se obtuvo que se podía realizar mediante colas o listas enlazadas.