Estructuras de datos y algoritmos I

TP final: “Problema del viajante”

Alumno: Gurvich, Agustín
Fecha de entrega: 24/07/2019

Motivación

La idea de este trabajo es poder resolver el problema del viajante (TSP por sus siglas en inglés). Para ello, hacemos uso de una estructura de datos especial y un algoritmo diseñado para resolver el problema acortando caminos. La implementación consiste en generar primero un camino completo, sin importar su costo, y luego buscar nuevos caminos cuyo costo sea menor al costo del mejor camino. En caso de encontrarse, el camino actual se vuelve el mejor y continúa la ejecución hasta haber tratado de realizar todos los viajes posibles.

Estructura

La estructura planteada en el trabajo es la siguiente:

```
typedef struct _AlmacenarCamino{
  int** listaDeVisitados;
  int cantidadVisitados;
  int costoTotal;
} AlmacenarCamino;
```

Cada parte de la estructura **AlmacenarCamino**:

- **listaDeVisitados**: Es una matriz bidimensional de enteros de tamaño nx3 (donde n es la cantidad de ciudades del diagrama), que cumple la función de almacenar un viaje realizado. La primer casilla de la submatriz es el índice de la ciudad de origen, la segunda es el índice de la ciudad de destino y la tercera es el costo de realizar dicho viaje.
- **cantidadVisitados:** Es la cantidad de viajes que tiene almacenada la estructura.
- **int costoTotal**: Es el costo de realizar todos los viajes almacenados en listaDeVisitados.

Durante la ejecución se crean dos instancias de esta estructura:

- **caminoParcial**, que se encarga de almacenar el camino actual que se recorre
- **caminoFinal**, que se encarga de guardar el mejor camino hasta el momento.

Consideraciones previas a la ejecución del programa:

Para poder obtener los datos necesarios para resolver el problema, el archivo de entrada debe tener una forma especial. La primer línea debe decir simplemente “Ciudades”, la segunda estar formada por los nombres de las ciudades, strings sin espacios en blanco separados por comas y espacios, la tercera decir simplemente “Costos” y de allí en mas escribir línea por línea el nombre de la ciudad de origen, una coma, el nombre de la ciudad de destino, una coma, y el costo de realizar dicho viaje (se adjunta un archivo a modo de ejemplo).

Compilación y ejecución:

Para compilar el programa debe ejecutarse la siguiente línea:

gcc main.c funcionesAuxiliares.c -pedantic -Wall -std=c99

Luego para la ejecución debemos ejecutar:

./a.out archivoEntrada archivoSalida

Donde **archivoEntrada** es el nombre del archivo del cual vamos a leer los datos, y **archivoSalida** es el nombre del archivo donde escribiremos la solución al problema.

Resolución del problema:

Dividimos el problema en dos partes: la lectura del archivo y la solución del problema.
El programa primero leerá el archivo de entrada correctamente formateado, encontrará la cantidad de ciudades y, si solamente se encontró una ciudad, no se ejecutará completamente pues no hay nada que resolver. De no ser así, se almacenarán, los nombres de las ciudades en una matriz bidimensional de caracteres. Por defecto la cantidad máxima de ciudades es 30, pues el algoritmo toma demasiado tiempo para resolver diagramas tan complejos. Dicho límite puede cambiarse modificando el valor de la constante **MAX_CIUDADES**.
De haberse encontrado una cantidad válida de ciudades, se procederá a crear la matriz de adyacencia correspondiente al diagrama. La fila enésima de dicha matriz corresponde con la enésima ciudad en la matriz que almacena las ciudades. Las columnas siguen la misma regla. Si se puede viajar entre dos ciudades, el valor de la matriz en esa posición es estrictamente mayor a 0. Caso contrario, es 0.
Luego se procede a resolver el problema haciendo uso de la función **resuelve_tsp.** Dicha función toma dos caminos: **caminoParcial**, que almacena el camino parcial que estoy realizando actualmente, **caminoFinal**, que almacena el mejor camino posible (si no encontré ningún camino todavía, el costo total es -1). Además toma la matriz de adyacencia, un array con las ciudades visitadas(cada ciudad se corresponde con su índice en la matriz de adyacencia, si la casilla vale 0 es porque no se visitó todavía, vale 1 si se visitó), la cantidad total de ciudades del diagrama y el índice la ciudad en la que me encuentro actualmente. La función resuelve el problema de manera recursiva usando la siguiente idea:

- Si ya viajamos por n-1 ciudades, significa que ya realicé una vuelta entera. Si además soy vecino de la ciudad de origen (la ciudad con el índice 0) y puedo viajar a ella, significa que encontré un camino válido.
- Sino, todavía estoy realizando viajes. Voy a tratar de visitar todas las casillas vecinas posibles. Si puedo viajar y además el costo de moverme a esa ciudad sumado al costo de mi

viaje hasta ahora es menor que el costo del mejor camino, o si todavía no encontré ningún camino, y además no visité dicha ciudad, entonces voy a realizar el viaje. Luego voy a seguir tratando con distintos caminos posibles para ver si encuentro uno mejor que el actual mejor camino.

Al finalizar la ejecución de la función **caminoFinal** tendrá un registro del mejor camino para visitar las ciudades, por lo cual las imprimirá en el archivo de salida.