

Estructuras de Datos y Algoritmos I

Trabajo practico final – Wirzt, Santiago – 27/6/2019

El propósito del trabajo es proporcionar un programa en C, capaz de resolver problemas del tipo TSP (Travelling salesman problem).

Estructuras:

La estructura usada en mi programa es de la forma:

```
typedef struct _Ciudades {  
    int cantidad;  
    char** nombres;  
    int* matrizCostos;  
} * Ciudades;
```

Ciudades es siempre un puntero porque nunca hago uso de la estructura localmente, sino que siempre prefiero utilizarla en memoria.

Los miembros de la estructura son:

cantidad: Es un int que almacena la cantidad de ciudades que tiene esa estructura.

nombres: Es un array de **char*** que almacena los nombres de las ciudades. El índice donde se almacena cada nombre representa el número que se le asigna a cada una en el momento de la lectura.

matrizCostos: Es un array de ints que representa una matriz cuadrada (nxn) simétrica, donde la posición (x,y) representa el costo de viaje entre las ciudades x e y. Por defecto, todas sus posiciones tienen valor 0. Debido a su forma de array unidimensional para acceder a una posición se debe calcular el índice de la siguiente forma:

(Fila * n) + Columna o **(Columna * n) + Fila**, donde n representa el tamaño de la matriz nxn.

Durante la resolución del ejercicio se hace uso de otra estructura definida de la forma:

```
typedef struct _Solucion {  
    int costo;  
    int* movimientos;  
} * Solucion;
```

Cuyos miembros son:

costo: Que representa el costo que almacena la solución actualmente, por defecto es -1.

movimientos: Array de ints, que almacena el orden de proceder de la solución, donde cada número representa una ciudad y debe estar conectada con las ciudades adyacentes.

La estructura Solucion no almacena el tamaño de su array, pero su uso siempre esta acompañado de la estructura Ciudades, cuyo miembro **cantidad** es el tamaño de **movimientos**.

Resolución:

La resolución del problema se hace a partir de una función recursiva, que a base de fuerza bruta, alcanza el camino con el menor costo total. La función recibe argumentos:

void brute_force(Ciudades c, Solucion mejor, Solucion actual, int nivel) {

La funcion precede a probar todos los caminos posibles, pero siempre arrancando de la ciudad 0. Esto es debido a que en este problema en particular, la solución es circular, lo que implica que, no importa donde arranque, la solucion es siempre análoga a arrancar de 0, por lo que, para evitar probar los mismos caminos en distinto orden, se toma a 0 como inicio siempre.

Durante la resolucion tambien se hace uso de la mejor Solucion, ya que en caso de haber encontrado una, el programa pregunta si la solucion que se esta intentando actualmente sobrepasa el **mejor precio actual**, lo que implicaria que ya no es necesario seguir avanzando en ese camino.

Una vez finalizada la recursión, se tiene por hecho que la solución encontrada es la mejor porque todas las opciones fueron comprobadas.

En casos donde la cantidad de Ciudades es mayor a 15, decidí utilizar un metodo de heurística (nearest neighbour) donde busco un camino posible avanzando de ciudad en ciudad por el camino más barato. Esto no devuelve la mejor solución pero permite poner una cota superior rapidamente para eliminar caminos incorrectos rapidamente en la función **brute_force**.

Condiciones de ejecución:

El programa espera que el archivo de datos ingresados este correctamente formateado, para evitar alentar el proceso de entrada revisando cada situación de error.

Debido a esto, junto con el programa **main**, se incluye un **check**, que recibe el archivo de entrada e informa al usuario los errores que contiene.

Se espera que el usuario ejecute **check** antes que **main** debido que una entrada erronea puede provocar que este último funcione de forma indefinida.

Formas de ejecución:

Junto con los archivos, tambien se provee un makefile, que simplifica los pasos a la hora de compilar.

- Para compilar los programas se debe tener instalado **make**.

Si esta instalado, solo se debe escribir **make** con la consola ubicada en la carpeta del trabajo.

En caso contrario, debe instalarse o ejecutarse las siguientes líneas de forma manual:

<code>gcc -c colasenlazadas.c -pedantic -Wall -std=c99</code>
<code>gcc -o main main.c colasenlazadas.o -pedantic -Wall -std=c99</code>
<code>gcc -c slist.c -pedantic -Wall -std=c99</code>
<code>gcc -o check revisaentrada.c slist.o -pedantic -Wall -std=c99</code>

Una vez compilado, se debe ejecutar el programa de control de entrada:

`./check <archivoEntrada>.txt`

Este imprimirá en consola todos los problemas encontrados, una vez solucionados se recomienda volver a ejecutar para comprobar que no se olvidó nada.

Cuando el programa anterior no imprima nada se puede proceder a ejecutar el main.

`./main <archivoEntrada>.txt <archivoSalida.txt>`

Si el programa encuentra algun percance durante su ejecución, este se imprimirá en consola, de lo contrario terminara sin decir nada.

La solucion puede ser encontrada en **`<archivoSalida>.txt`**