Guion de prácticas
El problema del viajante de comercio
Parte 2

Metodología de la Programación
Grado en Ingeniería Informática
Curso: 2016-2017

1. El Problema del Viajante de Comercio

El objetivo de esta práctica es continuar con el diseño e implementación de clases con memoria dinámica para resolver el problema del viajante de comercio (TSP, por travelling salesmen problem).

2. Parte 2

En la Parte 1, una solución se representaba simplemente como un vector de enteros. En esta parte, lo haremos con una clase. Además haremos más eficiente la clase problema y la completaremos.

Clase Solución

Esta clase representa una solución del problema, es decir, almacena una permutación de ciudades (valores enteros entre 0 y $n-1$, siendo n la cantidad de ciudades del problema). Por tanto, la clase debe almacenar el recorrido, mediante un vector dinámico de índices de ciudades de un Problema, y el número de ciudades (tamaño del Problema). Entre los métodos provistos, tienen que aparecer al menos, los siguientes:

```
Solucion(int v[], int n) //construye una solución a partir del vector v
Solucion(int n) // construye una solución de manera aleatoria
Solucion(const Solucion &otra) // construye una solución como copia de otra
~Solucion() // Destructor
bool esCorrecta() // verifica si los valores almacenados representan una permutación
int ciudad(int pos) // devuelve la ciudad que ocupa la posición pos
int posicion(int c) // devuelve la posición de la ciudad c en el recorrido
string to_string() // devuelve el recorrido como un string separando los datos por comas
```

Clase Problema

Los algoritmos que resuelven el problema del viajante de comercio requieren calcular la distancia entre dos ciudades un elevado número de veces. Para ello, es deseable tener precalculadas dichas distancias en una matriz. Se propone modificar la clase `Problema` añadiendo como miembro una matriz dinámica de enteros, `distancias`, que contenga en su posición (i, j) la distancia entre la ciudad en la posición i y la ciudad en la posición j de la secuencia de ciudades. Note que esta matriz se debe rellenar al construir un objeto de la clase `Problema`, que la distancia

de la ciudad i a la ciudad j es igual a la distancia de la ciudad j a la ciudad i y que la distancia de una ciudad a sí misma es cero. Note también que, de esta forma, la llamada al método `distancia(i,j)` sólo tiene que devolver `distancias[i][j]`.

Posteriormente, se deben agregar los siguiente métodos a la clase `Problema`:

```
int longitudRecorrido(Solucion s)
Problema(const Problema &otro)
~Problema()
```

El método `longitudRecorrido` calcula y devuelve la longitud del recorrido almacenado en la Solucion s. Los dos últimos métodos se encargan de construir un problema como copia de otro y destruir un problema, respectivamente.

Para facilitar la implementación de la clase y evitar la repetición de código se sugiere crear un método que reserve memoria, otro que libere memoria y otro que copie los datos de los objetos. Note que, al menos, los métodos de reserva y liberación deben ser privados.

Programa Principal

Implemente un programa `tspParte2.cpp` que permita crear un problema a partir de un nombre de fichero indicado como parámetro.

Posteriormente, cree tres soluciones, dos aleatorias y otra el recorrido $\{0, 1, ..., n\}$, que deben ser evaluadas.

El programa tiene que mostrar las tres soluciones y la longitud de los correspondientes recorridos.

Recuerde reservar la memoria estrictamente necesaria.

3. Recomendaciones

Antes de empezar a implementar se debe pensar y analizar detenidamente el problema para poder identificar los métodos y datos miembro de cada clase. Para cada método identifique: qué debe hacer, qué datos necesita para trabajar (argumentos), qué devuelve (tipo de salida), cómo debe llamarse para que sea legible el código y, finalmente, cómo debe hacerlo. Esta última parte debe plasmarse en un pequeño esquema en papel, que debe realizarse antes de escribir línea alguna de código. Si fuera necesario, puede definir métodos auxiliares (considere si deben ser públicos o privados).

Utilice los conceptos vistos en teoría y en las sesiones de prácticas: interfaz pública y privada de cada módulo, compilación separada, fichero makefile, etc, organizando todo lo necesario en un directorio que se llamará `tspParte2` con subdirectorios `src, include, obj, bin y doc`.

4. Material a Entregar

Cuando esté todo listo y probado, deberá empaquetar la estructura de directorios en un archivo con el nombre `tspParte2.zip` y lo entregará en la plataforma decsai en el plazo indicado. No deben entregarse archivos objeto (.o) ni ejecutables (conviene ejecutar make clean antes de proceder al empaquetado).

La pareja de estudiantes deben escribir un informe donde consten los nombres y DNI de los integrantes del grupo, los problemas que hayan podido surgir durante el desarrollo de la práctica, capturas de pantalla, etc. Este informe, en formato pdf, se guardará en la carpeta doc. El alumno debe asegurarse de que ejecutando las siguientes órdenes se compila y ejecuta correctamente su proyecto:

```
unzip tspParte2.zip
make
bin/tspParte2 prueba.tsp
```

La ejecución de `valgrind --leak-check=full --track-origins=yes bin/tspParte2 prueba.tsp` debe finalizar con 0 errores.