Práctica 3: Ciudades en Ruta

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

DECSAI

Estructuras de Datos

Grado en Ingeniería Informática. Grupo C

Índice de contenido

1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

1. Practicar con T.D.A implementados en la STL
2. Resolver un problema eligiendo la mejor estructura de datos para las operaciones que se solicitan
3. Seguir asimilando los conceptos de documentación de un tipo de dato abstracto (T.D.A)

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos
2. Haber estudiado el Tema 2: Abstracción de datos. Templates.
3. Haber estudiado el Tema 3: T.D.A. Lineales.
4. Haber estudiado el Tema 4: STL e Iteradores.
5. Aunque no es un requisito indispensable, haber realizado la práctica 2 ayudará a entender mejor el problema.

2. STL

EL objetivo en esta práctica es hacer uso de la librería STL en C++ http://www.cplusplus.com/reference/stl/. Con tal objetivo vamos a centrarnos en resolver un problema con dicha librería. Previo al uso de esta librería, como en la anterior práctica se requiere que el alumno diseñe nuevos T.D.A eficientes para resolver el problema.

3. Ejercicio

Esta práctica tendrá como objetivo descifrar una ruta dentro de un conjunto de rutas. Una ruta se define como un código (que la identifica) y una secuencia de posiciones geográficas, dadas por pares (latitud, longitud), asociadas a ciudades. Por lo tanto nuestro objetivo es dada una ruta obtener la información de las ciudades con dichas posiciones geográficas además que dirección (Este,North-Este, Norte, North-Oeste, Oeste, Sur, Sur-Este) se debe seguir para llegar a la siguiente ciudad dentro de la ruta.

Asi por ejemplo dada la siguiente ruta, con código R15:

R15;37.165;-3.58501;37.7704;-3.79999;40.4;-3.68335;41.65;-0.889982;42.4704;-2.42999;42.85;-2.66998;

Deberemos obtener la siguiente descripción**:**

Ruta R15
Granada;Spain;37.165;-3.58501;313269
_____Direccion N
Jaen;Spain;37.7704;-3.79999;92909
_____Direccion N
Madrid;Spain;40.4;-3.68335;2808718
_____Direccion E
Zaragoza;Spain;41.65;-0.889982;548955
_____Direccion NW
Logrono;Spain;42.4704;-2.42999;123918
_____Direccion NW
Vitoria;Spain;42.85;-2.66998;199109

Pero antes de abordar este problema vamos a retomar los TDA definidos en la práctica 2 y los vamos a redefinir. Así las tareas que debemos abordar son las siguientes:

1. Modificar el TDA Ciudades
2. Probar que funciona con test_ciudades (ver test_ciudades.cpp situado en el directorio src del material). Este fichero no debe modificarse.
3. Crear el TDA Ruta
4. Crear el TDA Rutas
5. Probar que funciona con test_rutas (ver test_rutas.cpp situado en el directorio src del material). Este fichero no debe modificarse.
6. Crear el programa descifra_ruta

3.1. TDA Ciudades

Un objeto del TDA Ciudades contiene un conjunto de ciudades. Como se vio en la práctica 2 cada ciudad se describe por el nombre de ciudad, nombre de país, una posición geográfica (dada por latitud y longitud) y la población.

Vamos a modificar TDA Ciudades para que la representación ahora use un vector de la STL. Modificar la implementación de acuerdo a esta nueva representación. Así la clase Ciudades tendrá como mínimo:

```
#include "Ciudad.h"
#include <vector>
using namespace std;
Class Ciudades{
private:
  vector<Ciudad> datos;
  vector<int> indice_ciudad;
  ...
```

Además vamos a añadir a la clase Ciudades dos iteradores uno no constante y otro constante que itere por las ciudades (ordenadas por país y a igualdad de país por ciudad).

De forma que nuestra clase será algo parecido a :

```
#include "Ciudad.h"
#include <vector>
using namespace std;
class Ciudades{
private:
  vector<Ciudad> datos;
  vector<int> indice_ciudad;
  ...
public:
…

   class iterator{
       private:
          vector<Ciudad>::iterator it;
    …
    };

    class const_iterator{
        private:
           vector<Ciudad>::const_iterator it;
     …
     }:
     iterator begin();
     iterator end();
     const_iterator begin()const;
     const_iterator end()const;

  };//end Ciudades
```

Por otro lado para acceder a toda la información de una ciudad dada su localización

geográfica de forma rápida y eficiente en la representación de ciudades nos haría falta añadir algún tipo de estructura que dada una posición geográfica nos de toda la información de la ciudad. Por ejemplo pensad en un map.

3.2. Programa test_ciudades

El TDA Ciudades, Ciudad y Posición tiene que funcionar con el el fichero test_ciudades. Para ello debe generar el programa test_ciudades. Un ejemplo de llamada podría ser:

```
prompt>bin/test_ciudades datos/ciudades.csv
```

Este programa recibe un fichero con las ciudades. Con estos datos testea los TDA Ciudades, Ciudad y Posición, además de los iteradores previamente comentados. Si visualizamos el fichero test_ciudades.cpp el alumno/a verá cinco partes diferenciadas:

- Sección 1: En esta sección se comprueba la declaración, lectura y escritura de los TDA desarrollados. La lectura hará uso de los métodos de inserción. Hay que tener en cuenta que los ficheros de entrada no están ordenados. Y por lo tanto cada vez que se inserta una ciudad se debe insertar de forma ordenada. Recuerde que deberemos ordenar por (país, ciudad) y además sólo por ciudad.
- Sección 2. Consulta de información y opera con los datos leidos. Así pide al usuario que inserte el nombre de dos ciudades y se obtendrá la distancia en kms entre ellas. Para ello se ha dado implementada la función ***DistanciaKm*** que a partir de dos posiciones obtiene la distancia en kms entre las dos ciudades.
- Sección 3. Continua con operaciones para consultar información. En este caso se obtiene todas las ciudades de un país concreto.
- Sección 4.- Finalmente en la sección 4 se comprueba el operador para borrar una ciudad del conjunto.
- Sección 5.- Se testea los iteradores. Con tal fin se imprime un objeto de tipo ciudades de orden de menor a mayor país (alfabéticamente) y en el sentido contrario.

3.3. TDA Ruta

El alumno/a creará el TDA Ruta que se especifica como un código (único) y una secuencia de posiciones geográficas. Un ejemplo de entrada para ruta sería :

R15;37.165;-3.58501;37.7704;-3.79999;40.4;-3.68335;41.65;-0.889982;42.4704;-2.42999;42.85;-2.66998;

Siendo R15 el código, y a continuación una secuencia de posición geográfica (dada por pares latitud y longitud).

La representación que vamos a dar para el TDA **Ruta** es:

```
#include “Posicion.h”
#include <string>
using namespace std;
class Ruta{
private:
  list<Posicion> datos;
  string codigo;
  ...
public:
…

   class iterator{
       private:
          list<Posicion>::iterator it;
          …
          };
          class const_iterator{
              private:
                 list<Posicion::const_iterator it;
          …
          }:
          iterator begin();
          iterator end();
          const_iterator begin()const;
          const_iterator end()const;

};//end Ruta
```

Como se puede observar hemos definido para Ruta también dos iteradores uno no constante y otro constante, que itera por las posiciones geográficas que son almacenadas en el campo

datos.

3.4. T.D.A Rutas

Un objeto del TDA Rutas representa un conjunto no repetido de objetos de tipo Ruta. Vamos a elegir una representación que sea lo más eficiente en tiempo a la hora de hacer consultas de una ruta concreta, insertar una nueva ruta y borrar una ruta. Para ellos usaremos la siguiente representación:

```
#include “Ruta.h”
#include <string>
using namespace std;
class Rutas{
private:
  map<string,Ruta> datos
  ...
public:
…

   class iterator{
       private:
          map<string,Ruta>::iterator it;
          …
          };
         class const_iterator{
             private:
                map<string,Ruta>::const_iterator it;
          …
          }:
          iterator begin();
          iterator end();
          const_iterator begin()const;
          const_iterator end()const;

        };//end Rutas
```

Se usará un mapa en el que el código de ruta será la clave y la Ruta será la información asociada. Pensad en el conjunto de operaciones necesarias para poder interactuar con objetos de tipo Rutas. Además implementaremos dos iteradores uno constante y otro no constante que permitirá iterar sobre los objetos de tipo Ruta almacenados en el objeto Rutas.

3.5. Programa test_rutas

Nuestros TDA Ruta y Rutas debe funcionar con el código en test_rutas.cpp. Para ello generaremos el programa test_rutas. Una posible ejecución desde la línea de ordenes podría ser la siguiente:

```
prompt>bin/test_rutas datos/Rutas.txt
```

Este programa recibe un fichero con un conjunto de Rutas (ver sección 3.8).

Con estos datos testeamos los TDA Ruta y Rutas, además de los iteradores previamente comentados. Si visualizamos el fichero test_rutas.cpp el alumno/a verá cuatro partes diferenciadas:

- Sección 1: Se comprueba la declaración, lectura y escritura de objeto de tipo Rutas (*misrutas*). Además en el proceso de lectura se comprobará que la lectura de Ruta es correcta además del método de inserción de Rutas.
- Sección 2. Consulta de la información de una ruta concreta. Y comprobación del operador de salida de Ruta.
- Sección 3. Borrado de una ruta y comprobación de que el borrado es correcto.
- Sección 4.- Comprobación de los iteradores de Ruta y Rutas que son correctos. Para ello se imprime las rutas en orden alfabético (de forma creciente) por código. Y también se imprime en sentido inverso.

3.6. Programa descifra_ruta

El alumno/a deberá crear un programa que permita descifrar las ciudades implicadas en una ruta. Para ello creará el programa descifra_ruta. Una posible llamada desde la línea de órdenes es la siguiente:

```
prompt>bin/descifra_ruta datos/ciudades.csv datos/Rutas.txt R15
```

Los parámetros dados son:

1. El fichero con las ciudades
2. El fichero con las rutas
3. La ruta a descifrar

Tras la ejecución en el terminal debe aparecer la información que se muestra en el cuadro a la derecha.

Ruta R15
Granada;Spain;37.165;-3.58501;313269
_____Direccion N
Jaen;Spain;37.7704;-3.79999;92909
_____Direccion N
Madrid;Spain;40.4;-3.68335;2808718
_____Direccion E
Zaragoza;Spain;41.65;-0.889982;548955
_____Direccion NW
Logrono;Spain;42.4704;-2.42999;123918
_____Direccion NW
Vitoria;Spain;42.85;-2.66998;199109

En la ruta con código R15 se parte de Granada y se termina en Vitoria. La siguiente ciudad desde Granada es Jaén que está al norte, a continuación Madrid que también esta al norte con respecto a Jaén. Por ejemplo Logroño está al North-Oeste con respecto a Zaragoza.

Para hacer consultas rápidas de dada una posición geográfica nos devuelva la ciudad, sería conveniente añadir en Ciudades un mapa que tenga como clave una posición y como valor asociado un índice a la ciudad.

3.6.1. Como determinar la dirección.

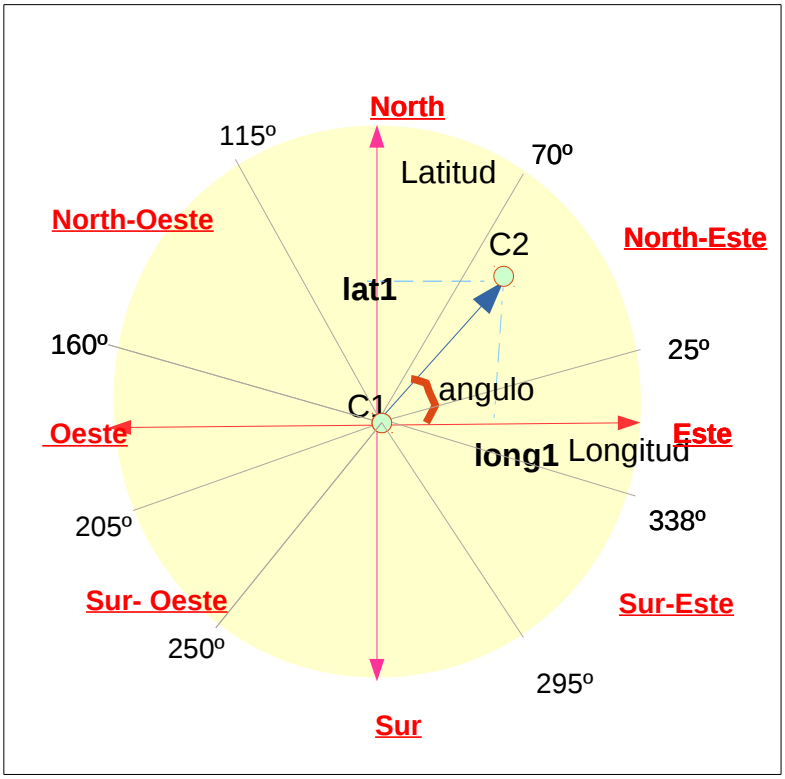
Para definir al dirección de una ciudad con respecto a otra seguiremos los siguientes pasos:

1. Sean C1 y C2 las posiciones geográficas de dos ciudades en una ruta.
2. Obtener lat1 como C1.latitud-C2.latitud
3. Obtener lon1 como C1.longitud-C2.longitud.

   Calcular el ángulo como:

```
float angulo = atan2(lat1,lon1);
if (angulo>0)
   angulo= angulo*360/(2*M_PI);
else angulo=(angulo+2*M_PI)*360/(2*M_PI);
```

4. Dependiendo del rango donde se encuentra el valor de ángulo se determinará la dirección. Así por ejemplo si ángulo es 340º la dirección será Este. Los rangos para las distintas direcciones podéis consultarlas en la imagen a la derecha.



3.7. Fichero con las ciudades

El fichero con las ciudades se compone de una serie de líneas. Cada línea se corresponde con la información de una ciudad

```
#city;country;lat;lng;pop
El Maiten;Argentina;-42.05;-71.1666;4269
Makinsk;Kazakhstan;52.6404;70.41;20365
Magdeburg;Germany;52.1304;11.62;227378
Allahabad;India;25.455;81.84;1137219
Brazzaville;Congo (Brazzaville);-4.25919;15.2847;1259445
Haikou;China;20.05;110.32;1606808
Abra Pampa;Argentina;-22.7167;-65.7;4480
Ussuriysk;Russia;43.8;132.02;140673
Moradabad;India;28.8418;78.7568;754069
Londonderry;United Kingdom;55.0004;-7.33328;82635
Koszalin;Poland;54.2;16.1833;107450
Phnom Penh;Cambodia;11.55;104.917;1466000
Phatthalung;Thailand;7.615;100.081;43522
….
```

El fichero contiene:

1. En la primera línea es un comentario.
2. A continuación en cada línea viene la información de cada ciudad, separada por “;”:
    - Nombre de la ciudad
    - Nombre del país
    - Latitud expresada en grados
    - Longitud expresada en grados.
    - Población.

3.8. Fichero con las Rutas

El fichero con las rutas contiene un linea por ruta. Cada ruta se describe por:

1. Un código de Rutas. A continuación un carácter ';'
2. Un conjunto de posiciones geográficas separadas por ';'. A su vez cada posición se define por latitud y longitud separando los valores por ';'.

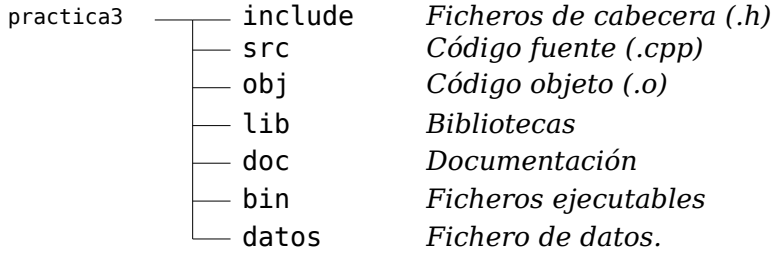
Un ejemplo de fichero de rutas es el siguiente:

```
R13;-15.4996;-69.1667;-17.1996;-
46.87;47.9204;123.51;29.5671;103.733;33.6304;46.43;53.1947;44.0444;54.5653;100.565;48.71;44.5;-
33.5995;26.88;38.2087;128.591;19.6337;30.4166;-5.77961;34.9;46.7833;-92.1064;-31.2;-55.75;
R14;-10.9;-37.12;-36.7167;-73.1167;-
3.36999;29.14;56.1572;10.2107;6.9;37.75;23.6833;86.9833;42.8299;75.2846;-18.1818;49.405;21.2704;-
98.78;11.7104;11.08;9.23333;29.8333;39.9403;-82.0133;41.56;60.64;-17.9296;25.84;
R15;37.165;-3.58501;37.7704;-3.79999;40.4;-3.68335;41.65;-0.889982;42.4704;-2.42999;42.85;-2.66998;
```

4. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "practica3.*tgz*" y entregarlo antes de la fecha que se publicará en la página web de la asignatura (en PRADO). Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable que haga una "limpieza" para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

El alumno debe incluir el archivo *Makefile* para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios*:*



Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta "*practica3*") para ejecutar:

```
prompt% tar zcv practica3.tgz practica3
```

tras lo cual, dispondrá de un nuevo archivo practica3.tgz que contiene la carpeta practica3 así como todas las carpetas y archivos que cuelgan de ella.