
# Práctica 6. Mallas regulares

## Sesiones de prácticas: 2

### Objetivos

Utilizar mallas regulares para minimizar el tiempo de búsqueda del Aeropuerto más cercano.

### Descripción de la EEDD

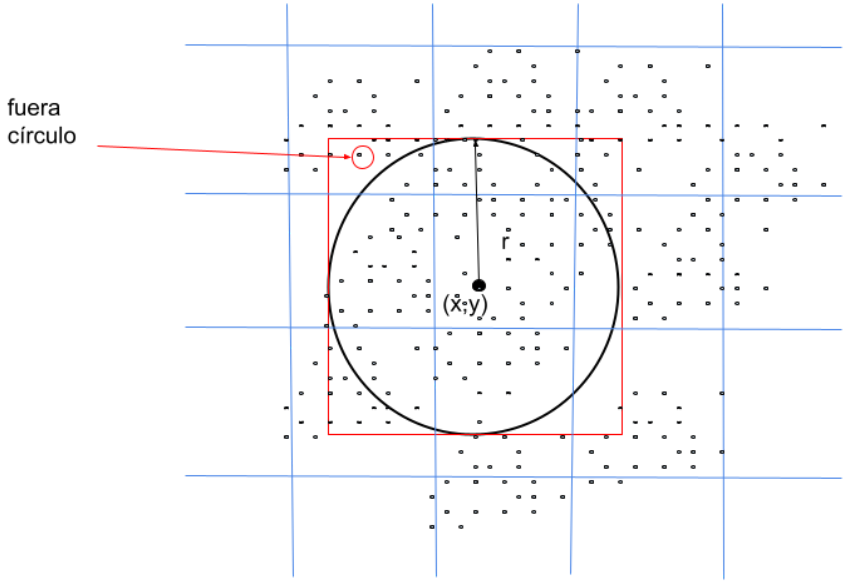
La aplicación desarrollada hasta ahora ha usado las estructuras de datos más adecuadas en cada proceso, sin embargo, no se ha tenido en cuenta el uso de las coordenadas UTM. En esta práctica vamos a utilizar una malla regular para almacenar los aeropuertos.

```
template <Typename T>
class MallaRegular {
            …
public:
      MallaRegular(float aXMin, float aYMin, float aXMax, float aYMax, int
nDiv);
….
      vector<T> buscarRadio(float xcentro, float ycentro, float radio);
      unsigned maxElementosPorCelda();
      float promedioElementosPorCelda();
}
```

Tener casillas vacias en cuenta en promediosporCelda

Esta clase tiene la funcionalidad de la Malla Regular dada en la Lección 17, pero añadiendo las funciones definidas arriba. Las funciones *maxElementosPorCelda()* y *promedioElementosPorCelda()* sirven de métrica para conocer la carga y distribución de puntos en la estructura y decidir así el tamaño de ésta. La función buscarRadio() devuelve todos los aeropuertos dentro de un *radio* centrado en un punto con coordenadas *(xcentro,ycentro)*. Podemos asumir que todo tipo T con el que se instancie la clase tiene los métodos getX() y getY() implementados, que en nuestro caso son respectivamente la longitud y la latitud en coordenadas UTM del aeropuerto. Para realizar esta operación, se considera el cuadrado de búsqueda que engloba al círculo y luego se calcula la distancia al centro. Se

deben considerar sólo los puntos dentro del círculo, como se indica en el dibujo. Las coordenadas de las cuatro esquinas del cuadrado que contiene al círculo son: (x+r,y+r), (x+r,y-r), (x-r,y-r) y (x-r,y+r). Para conocer la equivalencia entre kms y coordenadas terrestres basta con saber que un grado de Longitud o Latitud equivale a 111.1 kms.




Por otro lado, para calcular en kms la distancia entre dos puntos con coordenadas terrestres, se debe considerar la esfericidad de la tierra. Para ello se puede utilizar el algoritmo de *Haversine* (mirar Harversine.pdf adjunto a la práctica) que devuelve los kilómetros entre dos coordenadas terrestres. El algoritmo de conversión se describe a continuación:

```
Input: (lat1, lon1) (lat2, lon2)  (dos puntos)
Output: d  (distancia entre los dos puntos)
begin
      R = radio medio de la Tierra (6.378kms)
      IncrLat = (lat2 - lat1)*(PI/180)
      IncrLon = (lon2 - lon1)*(PI/180)
      a = sin² (IncrLat/2) + cos (lat1*(PI/180))*
      cos(lat2*(PI/180))* sin²(IncrLon/2)
      c = 2 * atan2 (sqrt (a), sqrt (1-a))
      d = R * c
end
```

Para más información visitar:
https://www.genbeta.com/desarrollo/como-calcular-la-distancia-entre-dos-puntos-geograficos-en-c-formula-de-haversine

**Descripción de la práctica:**

La relación entre *VuelaFlight* y *Aeropuerto* ahora es doble. En concreto *VuelaFlight::airportsID* es una composición en la que sustituiremos la tabla de dispersión implementada en la práctica anterior mediante un std::unordered_map[1] usando el código IATA del aeropuerto como clave. La nueva relación *VuelaFlight::airportsUTM* es una asociación implementada mediante una malla regular que clasifica los aeropuertos a partir de su posición.

La malla se crea a partir de las coordenadas antes citadas instanciadas a objetos tipo UTM. Se consideran *(aXmin, aYmin)* las coordenadas X e Y más pequeñas localizadas en el fichero de aeropuertos y *(aXmax, aYmax)* las coordenadas máximas en X e Y respectivamente. Esta área define todo el plano de búsqueda y se pueden obtener conforme se van leyendo los aeropuertos y se insertan en el std::unordered_map *VuelaFlight::airportsID*. El número de casillas en la malla vendrá determinado por el promedio de aeropuertos por casilla, debiendo estar entre 10-15. Por lo tanto, se debe probar con diferentes números de casillas de la malla hasta obtener un promedio de aeropuertos por casilla entre 10-15. Además, se debe calcular cuántos aeropuertos tiene la casilla más poblada.

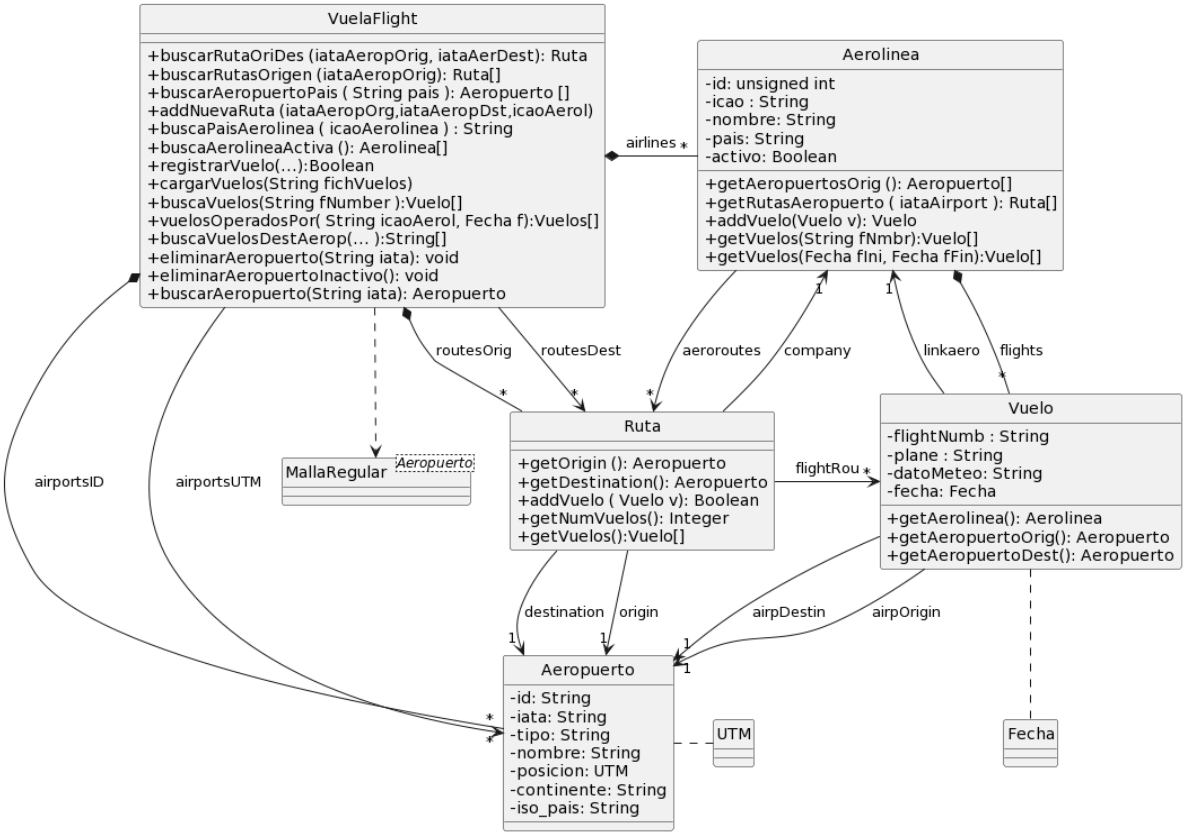
En el siguiente esquema UML se representa la nueva funcionalidad. De nuevo el constructor de la clase *VuelaFlight* se encarga de generar todo el sistema, incluyendo la malla regular. Para alimentar a la malla primero se han debido cargar en el unordered_map todas los aeropuertos y luego recorrerlo secuencialmente insertando en la malla cada una de los aeropuertos. La malla debe instanciarse como *MallaRegular<Aeropuerto*>* para albergar las direcciones de memoria de los *Aeropuertos* del mapa.

**VuelaFlight:**

- *VuelaFlight::rellenaMalla()*, Método privado para insertar los aeropuertos leídos en la malla. Se utiliza al final del constructor de VuelaFlight tras haber cargado todos los ficheros de datos.
- *VuelaFlight::buscarAeropuertosRadio(UTM pos, unsigned float radioKm) : Aeropuerto[]*. Dada una posición devuelve un vector de aeropuertos que se encuentran en un determinado radio.
- *VuelaFlight::aeropuertosMasSalidas(UTM pos, unsigned float radioKm): Aeropuerto[]*, que devuelve los 5 aeropuertos en el área indicada con rutas de salida que tiene más vuelos registrados. Nota: una vez obtenidos los vuelos salientes de los aeropuertos en cuestión, instanciar de forma adecuada una estructura

[1] La estructura *std::unordered_map* se implementa como una tabla de dispersión cerrada con cubetas

*std::priority_queue* que permita localizar eficientemente cuáles son los aeropuertos a devolver.




**Programa de prueba:**

Crear un programa de prueba con las siguientes indicaciones:

- Implementar la misma funcionalidad de la Práctica 5, exceptuando que ahora no usamos la tabla hash definida en la práctica anterior en la relación *VuelaFlight::airportsID* sino un std::unordered_map. Adaptar los métodos de VuelaFlight existentes para trabajar con las nuevas estructuras.
- Implementar la malla regular como un template según la Lección 17 añadiendo a VuelaFlight los métodos *buscarAeropuertosRadio()* y *aeropuertosMasSalidas()* descritos anteriormente.
- Buscar los aeropuertos en un radio de 300kms de Jaén capital (Longitud: -3.7902800, Latitud: 37.7692200).
- ¿Cual es el aeropuerto más cercano a Jaén capital?
- Mostrar los nombres de los 5 aeropuertos con más vuelos salientes en un radio de 800KM de Madrid capital

- ¿Qué ciudad europea, Londres o Venecia, concentra más aeropuertos en un radio de 400Kms? Se pueden obtener sus coordenadas utilizando esta web:https://www.ign.es/iberpix/visor/

**Para los que trabajan por parejas:**

La Unión Europea ha proporcionado una subvención al ayuntamiento de Jaén para construir un aeropuerto en Jaén capital, con iata JEN, en el que operará inicialmente una única aerolínea, IBERIA. Esta aerolínea realizará las siguientes rutas, Jaén-Madrid, Jaén Barcelona y sus respectivas vueltas. Añade este nuevo aeropuerto en el std::unordered_map, con las coordenadas indicadas anteriormente, y añade a la aerolínea las nuevas rutas. Obtener todos los aeropuertos en un radio de 300Km de Antequera (37.0193800, -4.5612300) y visualizar sus datos para comprobar que aparece el nuevo aeropuerto creado.

**VOLUNTARIO: Visualización 2D con la clase Img**

Para poder visualizar el resultado de forma gráfica, se adjunta a esta práctica la clase **Img**. Esta clase es en realidad una matriz de píxeles, creada tomando como parámetro el tamaño de un recuadro en número de píxeles en (tamaFilas, tamaColumnas). Esta clase es capaz de dibujarse como imagen de modo que cada consulta generará un fichero imagen resultado con *Img::guardar()*. Ejecutar la función *main()* que aparece junto al código para comprobar el funcionamiento y visualizar la imagen resultante. En la imagen adjunta a la práctica se puede ver la disposición espacial de todos los aeropuertos.

Para que esta clase sea útil en nuestro caso, hacemos coincidir las esquinas de la imagen con las del cuadro de trabajo donde se incluyen todas las coordenadas de las imágenes. Consideramos, pues, que la esquina inferior izquierda de nuestros datos es: (longitud, latitud) = (17.7342,-176.637) y la esquina superior derecha es: (longitud, latitud) = (71.2995, -64.7347). En el ejemplo de prueba se pinta un recuadro y se pinta también un pixel azul.

Utilizar esta clase para dibujar con distinto color las imágenes de la zona, en una imagen de 600*600.

**Estilo y requerimientos del código:**

1. El código debe ser claro, tener un estilo definido y estar perfectamente indentado, para ello se pueden seguir algunos de los estilos preestablecidos para el lenguaje C++ (http://geosoft.no/development/cppstyle.html).

2. Deben comprobarse todas los posibles errores y situaciones de riesgo que puedan ocurrir (desbordamientos de memoria, parámetros con valores no válidos, etc.) y

lanzar las excepciones correspondientes, siempre que tenga sentido. Leer el tutorial de excepciones disponible en el repositorio de la asignatura en docencia virtual.

3. Se valorará positivamente la calidad general del código: claridad, estilo, ausencia de redundancias, etc.