Práctica 6. Mallas regulares

Sesiones de prácticas: 2

Objetivos

Utilizar mallas regulares para realizar búsquedas eficientes por rangos.

Descripción de la EEDD

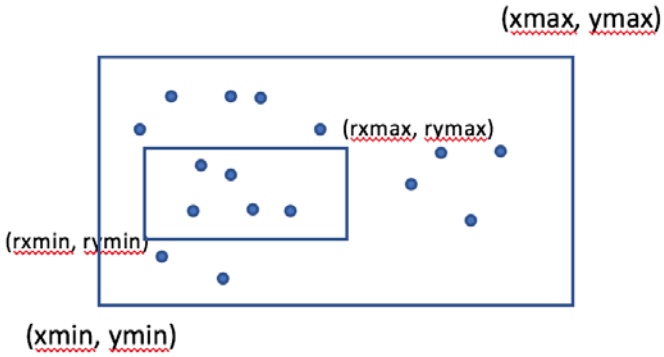
La nueva versión de la red social ahora permite geo-localizar las imágenes usando el GPS del móvil. Para representar las coordenadas GPS donde se tomó la imagen, se proporciona la clase UTM (adjunta a la práctica), una clase que guarda las coordenadas X e Y.

Para manejar de forma eficiente esta nueva característica, se va a emplear una malla regular instanciada a los punteros de las imágenes, ya previamente almacenadas en el mapa de imágenes de la clase ImageBook:

```
template <Typename T>
class MallaRegular {
                ...
public:
MallaRegular(float aXMin, float aYMin, float aXMax, float aYMax, int nDiv);

vector<T> buscarRango(float rxmin, float rymin, float rxmax, float rymax);
unsigned int maxElementosPorCelda();
float promedioElementosPorCelda();
```

Esta clase tiene la funcionalidad de la Malla Regular explicada en la lección 17-18, pero añadiendo las funciones definidas arriba. Las funciones maxElemetosPorCelda() y promedioElementosPorCelda() sirven de métrica para conocer la carga y distribución de puntos en la estructura y decidir así el tamaño de ésta. El método buscarRango() indica un rango válido dentro del área global de todos los datos y devuelve todos los valores incluidos entre (rxmin, rymin)-(rxmax, rymax). Podemos asumir que todo tipo T con el que se instancie la clase tiene los métodos getX() (similar a getLongitud()) y getY() (similar a getLatitud()) implementados. Resumiendo, la función buscarRango() devuelve los objetos localizados dentro de un área concreta definida a través de los parámetros que se le pasan a la función:



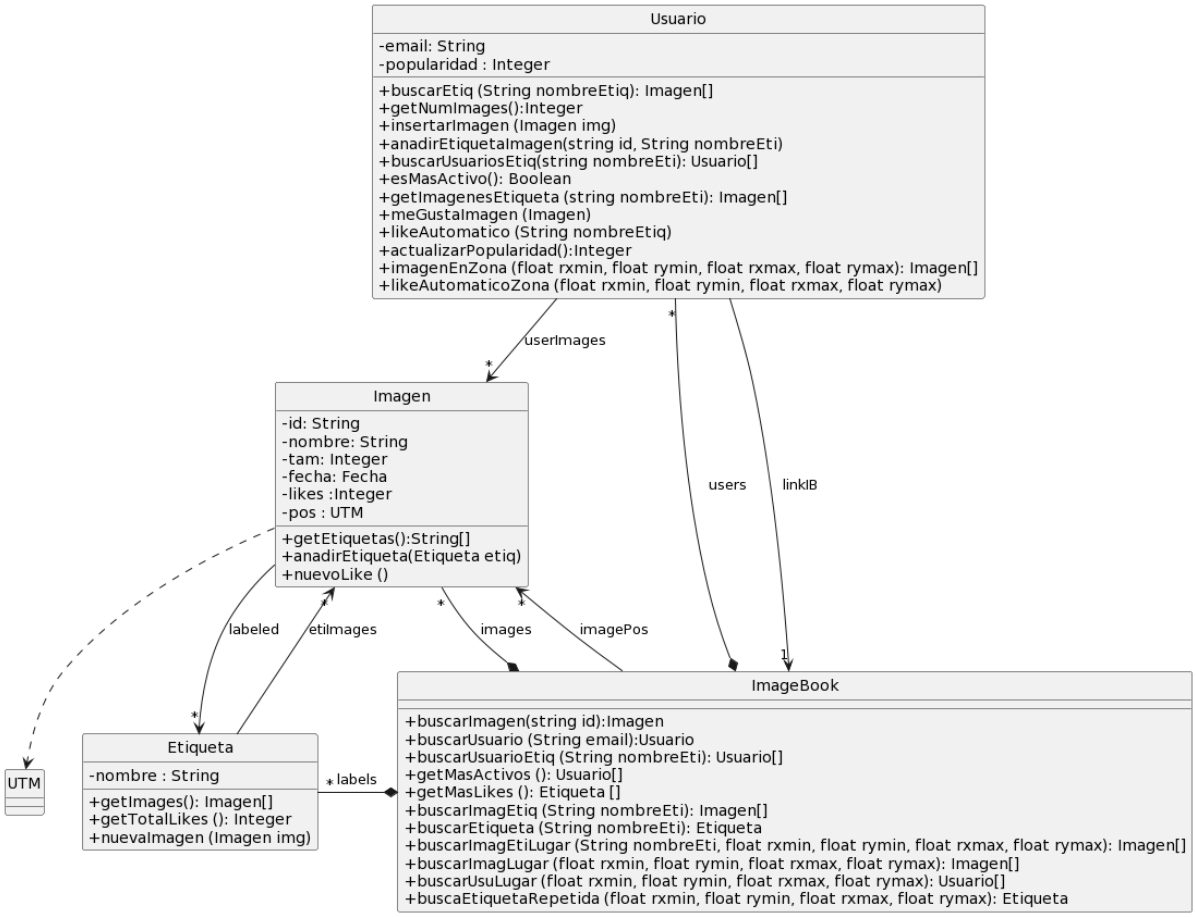
Descripción de la práctica:

La relación entre ImageBook e Imagen ahora es doble. En concreto ImageBook::images es una composición que se implementará mediante un std::map usando el id de la imagen como clave. La nueva relación *ImageBook::imagePos* es una asociación implementada mediante una malla regular con la posición de cada imagen. Para asociar a cada imagen una ubicación, se utilizará un nuevo fichero de imágenes (*imagenes_v2.csv*) adjunto a esta práctica en vez del utilizado en las anteriores prácticas. **La primera de las coordenadas se considerará la longitud y la segunda la latitud.**

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 imágenes 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 las imágenes del fichero y se insertan en el mapa *ImageBook::images.* El número de casillas en la malla vendrá determinado por el promedio de imágenes por casilla, debiendo estar entre 15-20. Por lo tanto, se debe probar con diferentes números de casillas de la malla hasta obtener un promedio de imágenes por casilla entre 15-20. Además, se debe calcular cuántas imágenes tiene la casilla más poblada.

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

Se podrán mantener las funciones de prácticas anteriores, modificándolas convenientemente cuando afecten a la relación ImageBook::images. Además, se implementará esta nueva funcionalidad:



ImageBook:

- *buscarImagLugar(float rxmin, float rymin, float rxmax, float rymax):* que devuelve las imágenes dentro del recuadro de búsqueda.
- *buscarImagEtiLugar(string nombre, float rxmin, float rymin, float rxmax, float rymax):* que devuelve las imágenes con una determinada etiqueta dentro del recuadro de búsqueda.
- *buscarUsuarLugar(float rxmin, float rymin, float rxmax, float rymax)*: que devuelve todos los usuarios que han hecho al menos una foto en esa zona.
- *buscaEtiquetaRepetida(float rxmin, float rymin, float rxmax, float rymax)*: que devuelve la etiqueta más repetida en una zona dada. En caso de empate cualquiera es válida.

Usuario:

- *imagenEnZona(float rxmin, float rymin, float rxmax, float rymax):* que devuelve las imágenes que el usuario ha realizado en la zona dada.
- *likeAutomaticoZona(float rxmin, float rymin, float rxmax, float rymax)*: que da un nuevo “me gusta” a todas las fotos de los demás usuarios que han hecho una foto en esa zona.

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 una tabla hash en la relación ImageBook::images sino un std::map.
- Implementar la malla regular como un template según la Lección 17-18 añadiendo la función *buscarRango()* descrita anteriormente.
- Mostrar los identificadores de las imágenes que se encuentran en el rango (rxmin=34.04, rymin=-81.06) y (rxmax=55.04, rymax=-65.06) y que comparten alguna de las etiquetas de la imagen más reciente del usuario "kenny_ohara73@yahoo.com".
- Mostrar el email de todos los usuarios que han tomado una foto en el rango (rxmin=36.388698, rymin=-121.72439) y (rxmax=39.388698, rymax=-89.72439).
- Mostrar el nombre de la etiqueta que más se repite en aquellas imágenes localizadas en el rango (rxmin=30.0201, rymin=-98.2340) y (rxmax=60.0039, rymax=-80.99).
- Buscar las imágenes del usuario "beau1@hotmail.com" que se encuentran localizadas en el rango (rxmin=30.8304, rymin=-94.8684) y (rxmax=47.3304, rymax=-65.3684) y obtener de ellas la imagen con más likes (en caso de empate coger una de ellas). A continuación, dar *like* a aquellas imágenes que están localizadas dentro del rango de la imagen con más likes (rxmin=longitudImagenMásLikes-0.1, rymin=latitudImagenMásLikes-0.1) y (rxmax=longitudImagenMásLikes+0.1, rymax=latitudImagenMásLikes+0.1). Para comprobar que los likes se han asignado correctamente, mostrar los likes de las imágenes antes y después del cambio.

Para parejas:

En los próximos días se va a inaugurar un nuevo centro comercial en la ciudad de Jaén llamado *JaénPlaza* (longitud=30, latitud=-75). Este centro comercial ha organizado un sorteo de una Playstation 5 para todos aquellos usuarios que hayan asistido el día de la inauguración. Para certificar que una persona ha asistido al día de la inauguración, deberán de tomar una foto en el centro comercial, subirla a nuestra red social y etiquetarla con la etiqueta (#JaenPlazaPlay5). Los asistentes al evento serán todos los usuarios cuyo email comience por la letra *'e'*.

Una vez todos los asistentes hayan realizado el proceso de hacer la fotografía y subirla a la plataforma con el hashtag mencionado, se procederá a hacer el sorteo. Para ello, se comprobará que todas las imágenes etiquetadas se encuentran en el rango de la localización del centro comercial y, de entre todos los que cumplan la condición, se escogerá un usuario al azar mediante un random.

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.

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.