Práctica 6. Mallas regulares

Sesiones de prácticas: 2 (entrega única 12 enero 2021)

Objetivos

Utilizar mallas regulares para realizar búsquedas eficientes por rangos.

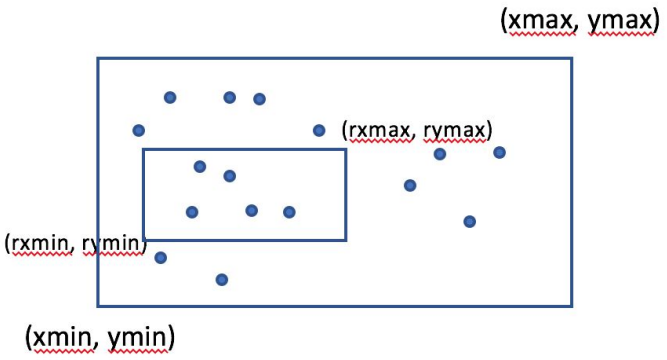
Descripción de la EEDD

Ziri es una aplicación de inteligencia artificial que aprende de los mensajes que escriben los usuarios de una red social. Un número aleatorio de esas frases y sus usuarios es enviada a Ziri para estudiar a nivel sociológico de qué se habla en todo momento y dónde, porque se utiliza la localización del usuario. Como cada usuario está localizado a través del GPS de su móvil, la posición de dicho usuario será determinante para hacer estudios sociológicos.

Para hacer tal análisis se ha decidido utilizar el Gestor de textos ya implementado para chequear alguna de las frases que han publicado sus usuarios. En realidad el proceso de analizar una frase cualquiera podría compararse con el de comprobar un documento, por lo que la podemos considerar de forma similar y podremos añadir las palabras que no están en el diccionario y contabilizarlas. La malla regular va a permitir localizar a usuarios por zonas, y saber de qué se habla en una determinada zona. La funcionalidad que se pide es la que se indica a continuación:

```
  template <class T>
   class MallaRegular {
            …
   public:
      MallaRegular(float aXMin, float aYMin, float aXMax, float aYMax, int
nDivX int nDivY);
      vector<T> buscarRango(float rxmin, float rymin, float rxmax,
                                                          float rymax);
      unsigned maxElementosPorCelda();
      float promedioElementosPorCelda();
   }
```

Esta clase tiene la funcionalidad de la Malla Regular dada en la lección 17-18, pero añadiendo las funciones definidas arriba. La de buscarRango() indica un rango válido dentro del recuadro global de 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() y getY() implementados, que en nuestro caso son respectivamente la latitud y la longitud.

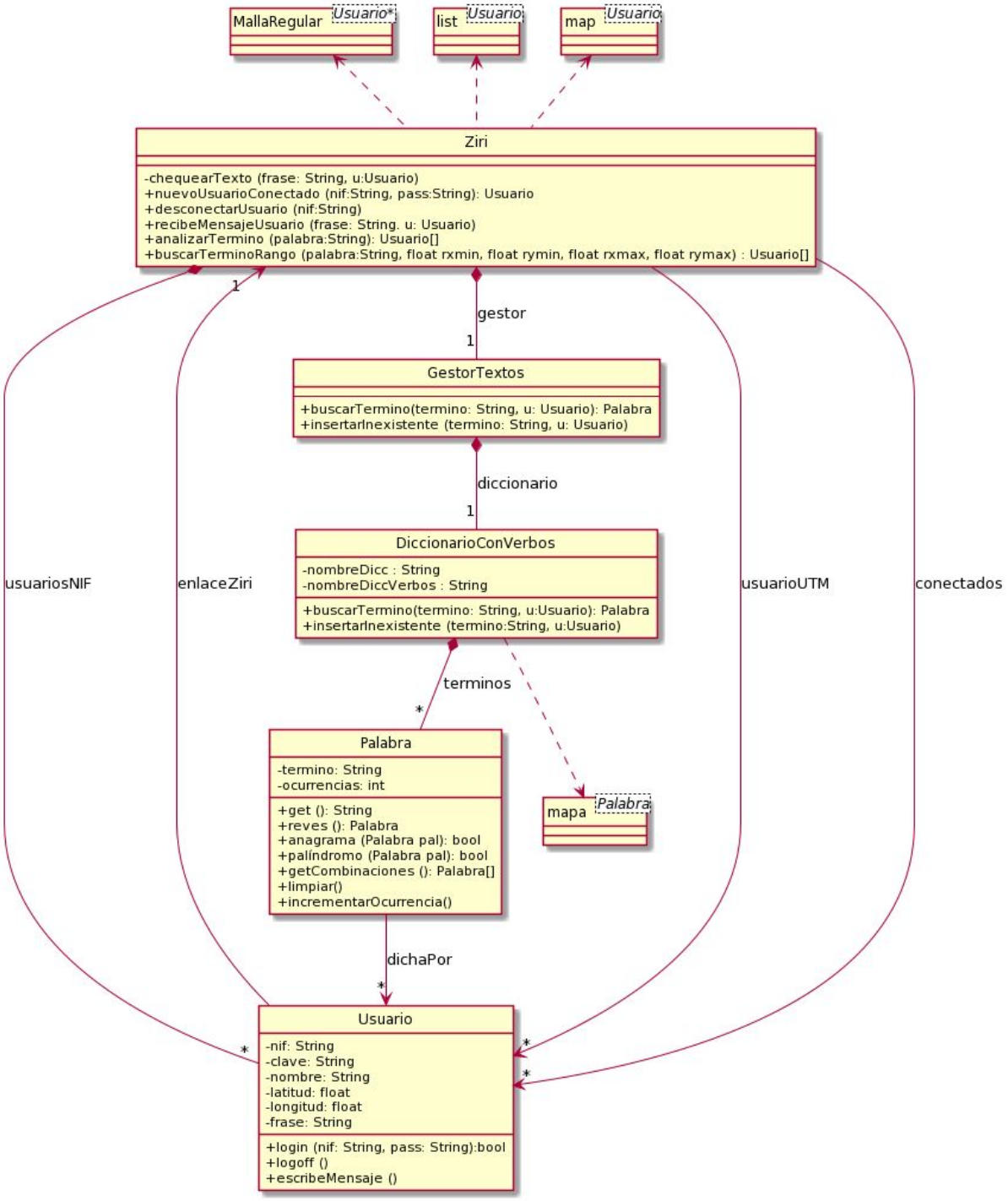


Diseño del modelo:

La malla se crea a partir del rango de valores del fichero de entrada, siendo (aXmin,aYmin) las latitudes y longitudes más pequeñas del fichero y (aXmax, aYmax) las coordenadas máximas del recuadro, que se pueden obtener mientras se va rellenando el conjunto de usuarios. Por tanto, la malla regular se construye a partir de los datos del fichero *usuarios_frases*, que también añade una frase por usuario. Dicha frase se considera un documento, al que tiene acceso el usuario, tal y como aparece en el diagrama UML.

El número de casillas en la malla vendrá determinado por el promedio de usuarios por casilla, debiendo estar entre 2 y 3 usuarios. Calcular también el número máximo de usuarios en una casilla.

Todos los usuarios se guardarán en un mapa a través de la composición *usuariosNIF* y **también todos** se añadirán a través de la relación *usuarioUTM*. Se usará otra EEDD, una lista para introducir los usuarios conectados en un momento dado a Ziri. Por tanto, se usarán tres EEDD diferentes para los usuarios, un mapa, una malla regular y una lista.



Las operaciones a realizar en Ziri son:

- login(): el usuario mete la clave a través de su teléfono móvil. Para agilizar el proceso se logearán de forma automática muchos usuarios como se indica más adelante.

- nuevoUsuarioConectado: se le pasa el NIF y el password como parámetro, se busca en el sistema Ziri y se introduce en la lista de usuarios conectados. También devuelve el usuario que se ha conectado.
- desconectarUsuario: se elimina el usuario de la lista de usuarios conectados.
- recibeMensajeUsuario: una vez que el usuario se ha conectado a Ziri, envía su mensaje a Ziri y éste lo procesa mediante el método privado chequearTexto().
- chequearTexto: método privado que comprueba si cada palabra se encuentra en el diccionario con verbos mediante el método buscarTermino():Palabra, y en caso de no estar, se inserta con el método insertarInexistente(), ambos métodos del *GestorTextos*, ya que esta clase hace de intermediaria con la clase *DiccionarioConVerbos*.
- analizarTermino: devuelve todos los usuarios que han dicho el término dado y
- buscarTerminoRango: devuelve todos los usuarios que han escrito la palabra o término dados en el rango indicado por la cuadrícula (xmin,ymin) y (xmax,ymax).

La clase *GestorTextos* hace de intermediaria entre Ziri y *DiccionarioConVerbos*.

La clase *DiccionarioConVerbos*, contiene el diccionario de palabras en español y el fichero con los verbos conjugados que se insertarán en un mapa. De nuevo hay que incrementar las ocurrencias en caso de que una palabra procedente de un usuario ya existiese en el DiccionarioConVerbos.

Programa de prueba:

1. Carga los datos de todos los usuarios del fichero adjunto en Ziri y muestra la cantidad de usuarios que hay en la aplicación por consola.
2. Conectar a todos los usuarios excepto los 50 últimos, que se quedarán sin conectarse directamente al sistema.
3. Intenta conectar al usuario que si existe con NIF 34923452L y clave pU7Pqq.
4. Hacer que todos los usuarios conectados escriban un mensaje (el almacenado en Usuario)
5. Muestra por consola los datos de todos los usuarios que han escrito la palabra “casa”.
6. Muestra por consola los datos de todos los usuarios que han escrito un mensaje en el rango de Jaén (latitud, longitud): (37,3)-(38,4) alguna de las siguientes palabras: “ganas”, “extranjero” y “es”. Muestra también el número de ocurrencias de cada una de estas tres palabras.

Práctica obligatoria para parejas y voluntaria para el resto:

Cambiar map por set

Se puede cambiar la funcionalidad de un map por un set haciendo que la clave sea el dato y sobrecargando el **operator<** sobre dicho dato. Haced eso entre la relación de *DiccionarioConVerbos* y *Palabra*.

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 los usuarios. Consideramos pues que la esquina inferior izquierda de nuestros datos es: (longitud, latitud) = (-9,99443, 35,86688) y la esquina superior derecha es: (longitud, latitud) = (3,98926, 43,272616). En el ejemplo de prueba se pinta un recuadro y se pinta también un pixel azul.

Pintar de azul todos los puntos del mapa y de color rojo los que estén dentro del recuadro de Jaén.

Utilizar esta clase para dibujar el resultado de: *analizarTermino()* y de la *búscarTerminoRango()*.

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.