Examen de Prácticas de Introducción al Software

Septiembre 2013

Se desea implementar un algoritmo llamado k-means para calcular la partición que minimiza el error cuadrático para unos puntos en el plano. Se dispone de una clase ya realizada llamada ParticionOptima que es un programa que implementa el algoritmo. A su vez, ParticionOptima usa clases del paquete fundamentos utilizado en las prácticas de la asignatura.

Por otro lado se dispone de la clase llamada Punto, que sirve para guardar las coordenadas de los puntos en el plano (abscisa y ordenada).

Punto
- double x - double y
+ Punto(double x, double y) + double getX() + double getY()

Los métodos hacen lo siguiente:

- constructor: copia los parámetros en los atributos
- getX, getY: métodos observadores que retornan el atributo x e y respectivamente.

Se pide realizar una tercera clase, llamada ListaPuntos, que guarda una lista de puntos y tiene distintos métodos útiles para el algoritmo k-means.

ListaPuntos
- ArrayList<Punto> listaP
+ ListaPuntos() + int getNum() + Punto getFirst() + Punto getLast() + void anadePunto(double x, double y) + double distancia(Punto p1, Punto p2) + int[] buscaParticion(Punto c1, Punto c2) + Punto calculaCentroide(int[] indexLista, int n)

Los métodos hacen lo siguiente:

- constructor: que crea la lista vacía
- getNum: retorna el tamaño de la lista
- getFirst(): retorna el primer punto de la lista de puntos listaP; si la lista está vacía, retorna el punto de coordenadas (0,0)
- getLast(): retorna el último punto de la lista de puntos listaP; si la lista está vacía, retorna el punto de coordenadas (0,0)

- anadePunto: añade al final de la lista de puntos listaP un nuevo punto con las coordenadas que se pasan como parámetro
- distancia: calcula y retorna la distancia euclídea entre dos puntos que se pasan como parámetros, utilizando la siguiente formula:

  $$distancia((x_1,y_1),(x_2,y_2)) = sqrt((x_1-x_2)^2+(y_1-y_2)^2)$$
- buscaParticion: este método recibe como parámetros dos puntos c_1 y c_2 en el plano y retorna una tabla de índices que es un array que tiene el mismo número de elementos que la lista de puntos listaP. Los elementos de esa tabla tienen dos valores posibles: 1 o 2. El elemento número i tiene valor 1 si y sólo si el punto i de la lista listaP está más cerca (en distancia euclidea) de c_1 que de c_2. En el otro caso, su valor es 2. Para obtener los índices se hace un recorrido de todos los elementos de lista y se comprueba para cada uno la condición de cercanía indicada.
- calculaCentroide: calcula y retorna el centro de masa para una partición dada. La partición se indica a través de una tabla de índices que se pasa como parámetro. Esta tabla tiene el mismo número de elementos que la lista de puntos listaP, y cada uno de sus elementos es un 1 (para indicar que se trata de un punto de la primera partición) o un 2 (para indicar que se trata de un punto de la segunda partición). El segundo parámetro (n) indica si se trata de la primera partición o de la segunda. El centro de masa de un conjunto de puntos es un punto cuya coordenada X es la media de las coordenadas X del conjunto y cuya coordenada Y es la media de las coordenadas Y del conjunto.

Para implementar calculaCentroide puede usarse el siguiente pseudocódigo:

```
real x=0,y=0,cont=0;
para i desde 0 hasta el tamaño de indexLista-1 hacer
   si indexLista[i] = n entonces
      Punto p=elemento i de listaP
      x = x + coordenada X de p
      y = y + coordenada Y de p
      cont++
   fin si
fin para
x=x/cont;
y=y/cont;
retorna nuevo Punto de coordenadas {x,y}
```

Se da el esqueleto de la clase ListaPuntos y se pide completarla.

Valoración:

- Métodos constructor, getNum, getFirst, getLast: 0.5 puntos cada uno.
- Método anadePunto y distancia: 1 punto cada uno
- Métodos buscaParticion y calculaCentroide: 3 puntos cada uno
- Compilación/Ejecución del programa ParticionOptima: 1 punto