Resolucion de la recuperacion del Segundo Parcial.pdf

Estructuras de datos y algoritmos (Universitat Politecnica de Valencia)

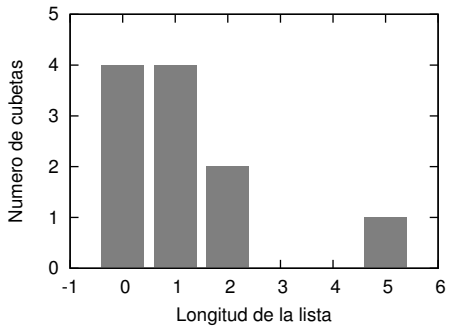
Resolución de la recuperación del segundo parcial de EDA
Escola Tècnica Superior d'Enginyeria Informàtica
18 de junio de 2014 – Duración 2 horas

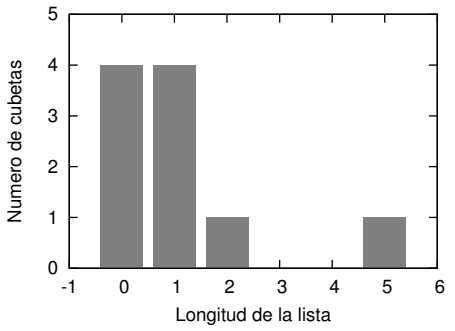
Pregunta 1 (2.5 puntos) Se dispone de una tabla de dispersión con resolución de colisiones por encadenamiento separado y $N = 10$ cubetas. La tabla guarda números enteros. La función de dispersión es $f(x) = x \mod N$. Se pide:

a) **(1.0 puntos)** Dibuja el histograma de ocupación de la tabla tras realizar las siguientes inserciones sobre una tabla de dispersión inicialmente vacía: 31, 34, 33, 53, 14, 13, 20, 56, 71, 15, 23, 3, 62.

b) **(0.75 puntos)** ¿Es posible realizar inserciones en la tabla del apartado a) para que su histograma de ocupación sea el del histograma que aparece a la izquierda? En caso afirmativo, da valores concretos de elementos tales que, al insertarlos, se obtiene una tabla con este histograma. En caso negativo, razona la respuesta.

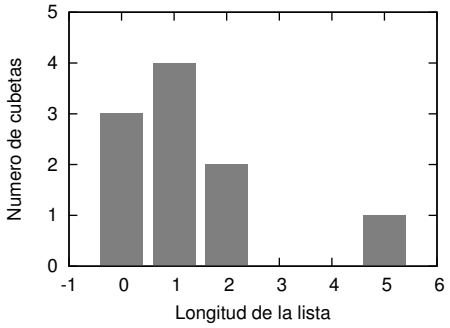
c) **(0.75 puntos)** ¿Es posible realizar borrados en la tabla del apartado a) para que su histograma de ocupación sea el del histograma que aparece a la derecha? En caso afirmativo, da valores concretos de elementos tales que, al borrarlos, se obtiene una tabla con este histograma. En caso negativo, razona la respuesta.





Solución

a)



b) No es posible puesto que al añadir elementos el número de cubetas vacías no se puede incrementar.

c) Si es posible, eliminando cualquiera de estos valores: 20, 62, 15 y 56; y cualquiera de estos: 31, 71, 34 y 14 o también eliminando el 31 y el 71 o el 34 y el 14..

Pregunta 2 (2.5 puntos) Un sistema guarda para cada usuario el momento de su última entrada expresada en milisegundos desde el dia 1 de enero de 1070 (lo que devuelve System.currentTimeMillis()). Esta información la tenemos guardada en una tabla hash *TablaHash <String,Long>tabla* (atributo de la clase *Aplicacion*). Queremos borrar aquellos usuarios que no hayan entrado en el sistema desde una fecha determinada. Para ello queremos añadir una nueva funcionalidad a la clase *Aplicacion* con el siguiente perfil *public void purgar(Long fecha)* que reciba una fecha expresada en milisegundos y elimine de la tabla todas aquellas entradas anteriores.

Solución

```
1  public class Aplicacion {
2      //atributos
3      TablaHash<String,Long> tabla;
4
5      //metodos
6      ...
7      ...
8      public void purgar(Long fecha){
9          ListaConPI<String> claves = tabla.claves();
10         for (claves.inicio(); !claves.esFin(); claves.siguiente()) {
11             String e = claves.recuperar();
12             Long t = tabla.recuperar(e);
13             if (t < fecha) tabla.eliminar(e);
14         }
15     }
16 }
```

Pregunta 3 (2.5 puntos) Un proceso industrial que se inicia diariamente genera millones de valores al dia (N valores) y en cualquier momento se desea consultar los K mayores valores generados hasta ese instante. Dada la magnitud de N, no es posible mantener todos los N valores en una estructura de datos y ordenarlos en cada consulta. Se pide implementar en la clase `MonticuloBinario`, un método eficiente `insertarHastaK` que reciba como parámetro un valor y lo inserte en el montículo manteniendo fija su talla en K. Es importante destacar que puesto que el método se invoca diariamente, el montículo inicialmente no tiene ningún elemento. Puedes suponer que la clase `MonticuloBinario` está definida como sigue:

```
public class MonticuloBinario<E extends Comparable<E>> implements ColaPrioridad<E> {
  protected E elArray[];
  protected static final int CAPACIDAD_POR_DEFECTO = 40;
  protected int talla;

  public MonticuloBinario(){talla=0; elArray=(E[]) new Comparable[CAPACIDAD_POR_DEFECTO];}
  public MonticuloBinario(int K) {talla=0; elArray=(E[]) new Comparable[K+1];}
  public boolean esVacia(){ ... }
  public E recuperarMin(){ ... }
  public void insertar(E x){ ... }
  public E eliminarMin{ ... }
  protected void hundir(int hueco){ ... }
  public void arreglarMonticulo{ ... }
}
```

Solución

```
public void insertarHastaK(E e) {
  if (talla < elArray.length) {
    elArray[++talla] = e;
    if (talla == elArray.length-1) arreglarMonticulo();
  }
  else {
    if (elArray[1].compareTo(e) < 0) {
      elArray[1] = e;
      hundir(1);
    }
  }
}
```

Pregunta 4 (2.5 puntos) Se desea determinar si un conjunto de valores reales se ajusta *aproximadamente* a una función lineal creciente. Para ello un ingeniero informático ha pensado un algoritmo que consiste en comprobar si la diferencia entre todo par de valores consecutivos de menor a mayor está acotada por un epsilon. Escribe un método eficiente estático que recibe una cola de prioridad no vacia de valores `double` y un valor `double` epsilon positivo que implemente dicho algoritmo. El método debe devolver `true` si el conjunto de valores se ajusta aproximadamente a una función lineal creciente y `false` en caso contrario. Por ejemplo, dado un el valor epsilon 0.05 y el conjunto $C_1 = \{0.3, 0.29, 0.34\}$ el resultado del método debe ser `true`, mientras que si el conjunto es $C_1 = \{0.3, 0.29, 0.36\}$ el resultado deber ser `false`.

Solución

```
public static boolean casiLineal(ColaPrioridad<Double> q, Double epsilon) {
  double a = q.eliminarMin();
  while (!q.esVacia()) {
    double b = q.eliminarMin();
    // a<=b
    if ( b-a > epsilon ) return false;
    a = b;
  }
  return true;
}
```