|  |
| --- |
| **Semana 3 - Ejercicios de grupo**  Diseño de algoritmos  Facultad de Informática - UCM |

|  |  |  |
| --- | --- | --- |
|  | **Nombres y apellidos de los componentes del grupo que participan** | **ID Juez** |
| **1** | Gabriel Emilio Lugo Estevez | DA17 |
| **2** | Andrés Ramiro Ramiro | DA27 |
| **3** | Cintia María Herrera Arenas | DA11 |
| **4** | Mateo Gonzalez De Miguel | DA10 |

Instrucciones:

1. Para editar este documento, es necesario hacer una copia de él. Para ello:
   * Alguien del grupo inicia sesión con la cuenta de correo de la UCM (si no la ha iniciado ya) y accede a este documento.
   * Mediante la opción *Archivo* → *Hacer una copia*, hace una copia del documento en su propia unidad de *Google* *Drive*.
   * Abre esta copia y, mediante el botón *Compartir* (esquina superior derecha), introduce los correos de los demás miembros del grupo para que puedan participar en la edición de la copia.
2. La entrega se realiza a través del Campus Virtual. Para ello:
   * Alguien del grupo convierte este documento a PDF (dándole como nombre el número del grupo, 1.pdf, 2.pdf, etc…). Desde *Google Docs*, puede hacerse mediante la opción *Archivo* → *Descargar* → *Documento PDF*.
   * Esta misma persona sube el fichero PDF a la tarea correspondiente del *Campus Virtual*. Solo es necesario que uno de los componentes del grupo entregue el PDF.

|  |
| --- |
| **3.1 Cola de medianas** |

El objetivo de hoy es resolver el **problema 8 ¿El mediano, por favor?**, del [juez automático](http://tais.fdi.ucm.es). Ahí podéis encontrar el enunciado completo del problema. Lo que necesitamos para resolver el problema es una cola de prioridad un tanto especial, en la que el elemento más prioritario no es el menor (como en las colas de mínimos) o el mayor (como en las colas de máximos), sino que es la *mediana*, el elemento que ocuparía la posición central si los elementos se ordenaran. Más precisamente, en este contexto[[1]](#footnote-1) la mediana de una serie ordenada de *N* elementos, *x*1, *x*2, …, *xN*, es el valor *x*(*N* + 1) / 2 (división entera).

Implementad una clase ColaMedianas que ofrezca la funcionalidad necesaria para resolver el problema:

class ColaMedianas {

public:

void insertar(int x);

bool vacia() const;

int mediana() const;

void quitarMediana();

};

*Pista*: Esta vez no puede hacerse utilizando un comparador de valores de un tipo apropiado, ya que cualquier comparador recibe dos valores, y la mediana depende de todos los valores de la cola, pero sí puede hacerse utilizando objetos de la clase std::priority\_queue.

**Solución**: (Escribid aquí las explicaciones necesarias para contar de manera comprensible vuestra solución. Incluid el código de la clase ColaMedianas, y el coste justificado de sus métodos. Extended el espacio al que haga falta.)

(Tomando n como el tamaño de la cola)

La clase de ColaMedianas tiene dos colas, una para guardar los elementos de menor tamaño (colaMaximos), y otra para guardar los elementos mayores que la mediana(colaMinimos).

Cuando se inserta un elemento, se decide a cual cola tiene que ir el nuevo elemento a insertar. Si las colas no están vacías, se compara el nuevo elemento con el elemento top() de cada cola. Dicho elemento se introducirá en la cola correspondiente. Si ambas colas están vacías, guarda por defecto el elemento en la colaMaximos. (Coste O(log n), ya que consultar si la cola es vacia y el elemento top() es constante, pero hacer push() requiere coste log n)

La funcion vacia() devuelve true si ambas colas son vacias, y false en caso contrario. (coste O(1), ya que consultar el tamaño de las colas es constante)

La función mediana calcula cuál seria la posición de la mediana a devolver. Se suman los tamaños de ambas colas +1, y se divide entre dos. Después se compara con el tamaño de cada cola para saber en que cola estaría la mediana. El coste de la funcion seria O(1), ya que consultar el elemento top() y size() es constante

Al reequilibrar, lo que hacemos es mirar los tamaños de las colas, si la resta de alguno de los dos (la cual conseguimos con la función “isBalanced()”) es mayor que uno, se inserta el valor “top” de la otra cola en ésta, y se llama a “pop” en la primera. El coste en el caso peor es de O(logN).

La funcion isBalanced tiene coste O(1), ya que consultar el tamaño es constante

La función void quitarMediana() calcula la posición del elemento mediana, si el tamaño de la colaMayores es mayor que el valor de la posición de la mediana se elimina el top de esta cola, en caso contrario se elimina el top de la colaMinimos. El coste de esta función por los métodos pop() es orden de log N. siendo N el número de elementos.

#ifndef COLAMEDIANAS\_H\_

#define COLAMEDIANAS\_H\_

#include <queue>

#include <vector>

class ColaMedianas {

public:

ColaMedianas(){}

void insertar(int x){

if (!colaMinimos.empty() && colaMinimos.top() < x) {

colaMinimos.push(x);

}

else if(!colaMaximos.empty() && colaMaximos.top() > x){

colaMaximos.push(x);

}

else {

colaMaximos.push(x);

}

reequilibrar();

}

void reequilibrar() {

//se comparan los tamaños de ambas colas. Si es igual, o la diferencia de tamaños es mayor que 1, no se hace nada. Si es mayor o menos que 1, se reajustan los tamaños

if(isBalanced() > 1) {

colaMinimos.push(colaMaximos.top());

colaMaximos.pop();

}

else if(isBalanced() < -1) {

colaMaximos.push(colaMinimos.top());

colaMinimos.pop();

}

}

int isBalanced() {

return colaMaximos.size() - colaMinimos.size();

}

bool vacia() const{

return colaMaximos.empty() && colaMinimos.empty();

}

int mediana() const{

int mediana;

int medianaPos = (colaMinimos.size() + colaMaximos.size() + 1) /2;

//si el tamaño de la colaMayores es mayor que la mediana, se devuelve el elemento top de esa cola

if(medianaPos <= colaMaximos.size()){

mediana=colaMaximos.top();

}

//si no, se devuelve el de la cola de minimos

else{

mediana=colaMinimos.top();

}

return mediana;

}

void quitarMediana(){

int medianaPos = (colaMinimos.size() + colaMaximos.size() + 1) /2;

//si el tamaño de la colaMayores es mayor que la mediana, se elimina el elemento que este en top en esa cola

if(medianaPos <= colaMaximos.size()){

colaMaximos.pop();

}

//si no, se elimina el de la cola de minimos

else{

colaMinimos.pop();

}

reequilibrar();

}

private:

std::priority\_queue<int, std::vector<int>, std::less<int>> colaMaximos; //guarda los elementos menores

std::priority\_queue<int, std::vector<int>> colaMinimos; //guarda los elementos mayores

};

#endif /\* INDEXPQ\_H\_ \*/

Utilizando la clase ColaMedianas resolved el problema completo del juez, que ahora debe ser ya muy sencillo.

**Número de envío:**

**Solución**: (Escribid aquí el código de la función resuelveCaso(), con los comentarios sobre la solución que consideréis oportunos, y su coste justificado. Extended el espacio al que haga falta.)

El coste de la función resuelveCaso() corresponde al coste de la función Insertar (Log N) ( y el coste adicional de las funciones requilibrar y eliminar), iterando N veces, siendo N el número de eventos. El coste total de la función en el caso peor es N Log N.

bool resuelveCaso() {

// leer los datos de la entrada

int n, elem;

ColaMedianas c;

std::cin >> n;

if (!std::cin) // fin de la entrada

return false;

for(int i = 0; i < n; i++){

std::cin >> elem;

if(elem != 0){

c.insertar(elem);

} else{

if(c.vacia()){

std::cout << "ECSA ";

} else{

std::cout << c.mediana() << " ";

c.quitarMediana();

}

}

}

std::cout << std::endl;

// escribir sol

return true;

}

1. En estadística, la mediana de una muestra ordenada se define de forma distinta distinguiendo casos según el número de valores de la muestra sea par o impar. Si es impar, la mediana es el elemento central, *x*(*N* + 1) / 2. Si es par, la mediana es la media de los dos valores centrales, (*xN* / 2 + *xN* / 2 + 1) / 2. [https://es.wikipedia.org/wiki/Mediana\_(estadística)](https://es.wikipedia.org/wiki/Mediana_(estad%C3%ADstica)) [↑](#footnote-ref-1)