Práctica 1

Escobar Rosales Luis Mario

17 de octubre , 2021

La práctica consiste en lo siguiente:

- Almacenar un arreglo de n dimensiones en uno de una sola dimensión.

Desarrollo :

- Se creó la clase arreglo que implenta los métodos de la interfaz IArreglo
- Los atributos de la clase que se elijieron fueron: int [] arr que será el arreglo en unidimensinal que representará las multiples dimensiones de la estructura , int [] deltas arreglo de las dimensiones del arreglo , int len longitud de arr.
- Primero se implementó el método obtenerIndice(int [] indices) con la optimización del polinomio de redireccionamiento.Este método fue el primero en implementarse porque los métodos restantes dependen de él.
- Se ocupó este método en almacenarElemento() obtenerElemento().

Preguntas

1.-Explica la estructura de tu código, explica en más detalle tu implementación del método obtenerIndice.

Se utilizaron 3 atributos: arr, delta, len
El método constructor inicia obteniendo la longitud del arreglo unidimensional arr . Para esto se mutiplican los elementos del arreglo de entrada. inicializamos nuestro atributo arr con la longitud obtenida.
Despues inicializamos nuestro arreglo de dimensiones, delta, con la longitud del arreglo de entrada,luego copiamos sus elementos.

El método obtener indice es la implementación de la fórmula del polinomio de redireccionamiento, (optimizado).

```java
    public int obtenerIndice(int [] indices){
      int index  = indices[0];
      for(int i= 1;  i <  indices.length; i++){
        if(indices [i] < 0 || indices [i] > this.delta[i] ) throw new IndexOutOfBoundsException("Indice no valido");
          index = indices[i] + this.delta[i] * index  ;
      }
      return index;
    }

}
```

Declaramos una variable int index.index va a ser una variable acumuladora para nuestro resultado.Su valor inicial será indices[0], es decir, i_0.
En cada iteración del ciclo, se sigue el siguiente patrón : se multiplica la variable acumuladora por la dimensión delta[i] y se le suma el indice siguiente(hasta llegar a (i_{n-1})).

Los siguentes métodos son triviales:

```java
    /**
     * Devuelve el elemento que se encuentra en la posicion <code>th</code> en el
     * arreglo multidimensional.
     * @param indices arreglo con los indices del elemento a recuperar.
     * @return el elemento almacenado en la posicion <code>i</code>.
     */

    public int obtenerElemento(int [] indices){
      if(indices.length <= 0 ) throw new IllegalArgumentException("Arreglo de indices no valido");
      return this.arr[obtenerIndice(indices)];
    }

    /**
    * Asigna un elemento en la posicion <code>th</code> del arreglo multidimensional.
    * @param indices arreglo con los indices donde se almacenara el elemento.
    * @param elem elemento a almacenar.
    */
    public void almacenarElemento(int [] indices, int elem){
      if(indices.length <= 0) throw new IllegalArgumentException("Arreglo de indices no valido");
      this.arr[obtenerIndice(indices)] = elem;
    }
```

Básicamente invocamos a obtenerIndice() para acceder a la posición donde queremos obtener o asignar un elemento.

2. ¿Cuál es el orden de complejidad de cada método?
Veamos que el método obtenerIndice(), hace un recorrido desde i = 1 hasta (indices.length)-1, es decir, un recorrido lineal. Como no hay ninguna otra iteración anidad o invocación a otro método, el orden de este método es lineal, $O(n)$.
Veamos que los otros métodos solo requieren ese metodo para el acceso a esa posición, y como el acceso es $O(1)$, entonces igualemnte son $O(n)$. El método constructor hacer recorridos lineales, los ciclos no están anidados, ni se invoca a métodos por lo que también es $O(n)$.

$\therefore$ la complejidad del programa es $O(n)$