Grado en Ingeniería Informática

Estructura de Datos y Algoritmos, Grupo 84M, 2015/2016

17 de Marzo de 2016

Nombre y Apellidos:

…….

**PROBLEMA 1 (1 punto)**

Dadas las clases:

```
public class SNodeInteger {

    public Integer elem;
    public SNodeInteger next;

    public SNodeInteger(Integer e) {
        elem = e;
    }

}
```

```
public class SStack implements IStack {
       SNode peak = null;
....
}
```

donde **IStack** es la especificación del TAD Pila de Enteros.

Se pide:

a) Añade un método en la clase **SStack** que reciba un número entero **x** como parámetro. El objetivo del método es borrar de la pila todos los elementos cuyo valor es múltiplo de **x**. Algunos ejemplos son:

| Pila inicio (cima elemento izquierda) | Operación | Pila final (cima elemento izquierda) |
|---|---|---|
| 5 4 3 10 15 2 5 | pila.removeMultiples(5) | 4 3 2 |
| 3 1 2 8 | pila.removeMultiples(5) | 3 1 2 8 |
| 3 1 3 5 8 | pila.removeMultiples(1) | 3 1 3 5 8 |
| 3 1 3 5 8 | pila.removeMultiples(2) | 3 1 3 5 |

b) Razona sobre el caso peor y promedio del algoritmo. ¿Cuál es la complejidad del removeMultiples?.

c) Si el TAD fuera una lista simplemente enlazada, ¿cuál sería la complejidad del método removeMultiples? ¿Y si fuera una lista doblemente enlazada?

**Nota:** La solución requiere que uséis los métodos de la clase IStack. Sin embargo, **no es necesario añadir su implementación en la solución**.

**Solución:**

```
public void removeMultiples(int element)
{
	//pila auxiliar para apilar elementos validos de la pila original.
	SStack pilaAux = new SStack();
	while (!isEmpty()){
		//si cima es igual al elemento a eliminar

		if (top()%element==0){
			//lo quitamos
			pop();
		}
		else
		{
		//si hay que conservarlo lo llevamos a la pila aux.
		pilaAux.push(pop());
		}

	}
		//una vez terminada la pila original se recorre la pila aux y se apilan
		while (!pilaAux.isEmpty()){
			push(pilaAux.pop());
		}
}
```

b) No hay diferencia entre el peor caso y el caso promedio ya que siempre es necesario recorrer la pila completa. La complejidad del método es lineal (O(n)).

c) La complejidad del método seguiría siendo lineal (O(n)). En realidad, la implementación usando listas doblemente enlazada no proporciona ninguna ventaja ya que el método requiere que la lista se recorra de extremo a extremo.

**PROBLEMA 2 (1 punto)**

Desarrolla un **método RECURSIVO** para la ordenación de un array de números enteros basado en el algoritmo Quicksort. Para esta implementación se toma como pivote **el elemento más a la derecha**.

**Nota:** Para facilitar el ejercicio se incluyen comentarios de ayuda.

```
public static void quicksort_Right(int vector[], int left, int right) {

                int pivot=vector[right]; //last element is the pivot
                int i=left; // i performs the search from left to right
                int j=right; // j performs the search from right to left
                int aux;

                // while searches were not crossed
                while(i<j){

                     // the greater element than the pivot is searched
                       while(vector[i]<pivot ) i++;


                       // the lower element than the pivot is searched
                    while(vector[j]>=pivot && i<j) j--;

                // if no crossed
                    if (i<j) {
                        aux= vector[j];     // are swapped

                        vector[j]=vector[i];
                        vector[i]=aux;


                    }
                 }

                vector[right]=vector[i]; // the pivot is put in the corresponding place
                vector[i]=pivot; // lower elements to the left and higher to the right

                 if(left<i-1)
                    quicksort(vector,left,i-1); // left subarray is sorted
                 if(i+1 <right)
                    quicksort(vector,i+1,right); // right subarray is sorted
            }
```