ESTRUCTURA DE DATOS Y ALGORITMOS

SEGUNDO EXAMEN PARCIAL

Leganés, 5 abril 2013

Nombre y Apellidos:

Algunas recomendaciones:

- Antes de comenzar el examen, escribe tu nombre.
- Lee atentamente el enunciado de cada ejercicio ANTES de contestar.
- Marcar la opción correcta en las preguntas de tipo test o completar código cuando se pida

1. ¿Qué salida se muestra por consola el siguiente código que prueba las **pilas** implementadas mediante una **lista simple**? (suponiendo que el método toString() muestra los elementos de la pila comenzando por la cima) **(1 pto.)**

Solución:

a) Pila: [D C B A]
b) Pila: [D C A]
c) Pila: [A C D]
d) Pila: [A B C D]

```
public static void main(String[] args) {
    // PROBANDO SSTACK
    SStack<String> s = new SStack<String>();
    s.push("A");
    s.push("B");
    s.pop();
    s.push("C");
    s.push("D");
    System.out.println("Pila: "+s.toString());

}
```

2. ¿Qué salida se muestra por consola el siguiente código que prueba las **pilas** implementadas mediante un **array**? (suponiendo que en el constructor se indica el tamaño del array y que el método toString() muestra los elementos de la pila comenzando por la cima) **(1 pto.)**

Solución:

a) [1 2 3]
b) [3 2 1]
c) ¡¡la pila está llena y no puedo insertar el elemento 3!! Pila: [1 2]
d) ¡¡la pila está llena y no puedo insertar el elemento 3!! Pila: [2 1]

```
public static void main(String[] args) {
    // PROBANDO ASTACK
    AStack<Integer> s = new AStack<Integer>(2);
    s.push(1);
    s.push(2);
    s.push(3);
    System.out.println(s.toString());

}
```

3. ¿Qué salida muestra por consola el siguiente código que prueba las **colas doblemente enlazadas**? (suponiendo que el método toString() imprime por pantalla los elementos de la cola en orden comenzando por el primero) **(1 pto.)**

Solución:

a) cola: [4 3 2]
b) cola: [3 4 2]
c) cola: [2 3 4]
d) cola: [2 4 3]

```
public static void main(String[] args) {
    //PROBANDO DQUEUE
    DQueue<Integer> dq = new DQueue<Integer>();
    dq.addFirst(4);
    dq.addFirst(3);
    dq.addLast(2);
    System.out.println("cola: "+dq.toString());

}
```

4. ¿Qué salida se muestra por consola el siguiente código que prueba las colas implementadas con un array? (suponiendo que el método toString() imprime por pantalla los elementos de la cola en orden comenzando por el primero) **(1 pto.)**

```
public static void main(String[] args) {
    // PROBANDO AQUEUE
    AQueue<Integer> q = new AQueue<Integer>(2);
    q.enqueue(1);
    q.enqueue(2);
    q.dequeue();
    q.enqueue(3);
    System.out.println("Cola: "+q.toString());
```

Solución:

a) cola: [2 3]
b) cola: [1 2 3]
c) cola: [3 2]
d) ¡¡la cola está llena y no puedo insertar el elemento 3!! Cola: [1 2]

5. ¿Cuál es la complejidad del siguiente algoritmo? **(2 puntos)**

```
/** Busca un elemento en una matriz
y muestra por pantalla las posiciones donde se encuentra */
    public static void buscaLinealMatriz(int elem, int[][] matriz){
        System.out.println("*** BÚSQUEDA LINEAL en MATRIZ ***");
        for(int i=0;i<matriz.length; i++){
            for(int j=0; j<matriz[i].length; j++){
                if (matriz[i][j]== elem){
                    System.out.print("("+i+","+j+")");
                }
            }
            System.out.println("");
        }
    }
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(\log_2 n)$

6. Estima el tiempo de ejecución del método *toString()* de la siguiente pregunta en número de acciones, T(n), y calcula su orden, O(¿?). Justifica tu respuesta **(2 ptos.)**

Solucion:

T(n) = 1 + 1 + (n+1) + n + n(2) + 2 = 4n + 4

También válido 7n + 5 si se tienen en cuenta las operaciones de concat + getElem + toString

O(n)

```
@Override
public String toString() {
    String result = null;  // 1 op
    for (INode<E> nodeIt = frontNode; nodeIt!=null; nodeIt = nodeIt.nextNode) { // 1 + (n+1) + n
        if (result == null) {  //1 op
            result = "(" + nodeIt.getElement().toString() + "]"; // 1 op o 4 op
        } else {
            result += "," + nodeIt.getElement().toString();      // 1 op o 4 op
        }//total if: 2 op (siempre)
    } //total for: 2n op
    return result == null?"empty":result; // 1 op o 2 op
}
```

7. Teniendo en cuenta el código de implementación del método ***toString()*** de ***SQueue*** siguiente, implementa el método que calcule el tamaño de la Cola, ***size()***, implementada con una lista simple (NOTA: no existe un atributo *size* en la clase y la cola no se puede ver modificada tras ejecutar el método.
(3 ptos.)

```
public String toString() {
    String result = null;
    for (SNode<E> nodeIt = frontNode; nodeIt!=null; nodeIt = nodeIt.nextNode) {
        if (result == null) {
            result = "[" + nodeIt.getElement().toString() + "]";
        } else {
            result += "," + nodeIt.getElement().toString();
        }
    }
    return result == null?"empty":result;
}
```

Solución:

```
public int size() {
    int result = 0;
    for (SNode<E> nodeIt = frontNode; nodeIt!=null; nodeIt = nodeIt.nextNode) {
        result++;
    }
    return result;
}
```

8. Completa el código del método *pushLast* que inserta un elemento al final de una pila implementada con una lista simple *(SStack)* **(3 ptos.)**

```
/** método que inserta un elemento al final de una pila
 *  utiliza una pila temporal para almacenar los elementos */
public void pushLast (E elem){
    SStack<E> sTemp = new SStack<E>();
    while (!this.isEmpty()) {
        sTemp.push(this.pop());
    }



}
```

Solución:

```
/** método que inserta un elemento al final de una pila
 *  utiliza una pila temporal para almacenar los elementos */
public void pushLast (E elem){
    SStack<E> sTemp = new SStack<E>();
    while (!this.isEmpty()) {
        sTemp.push(this.pop());
    }
    this.push(elem);
    while (!sTemp.isEmpty()) {
        this.push(sTemp.pop());
    }
}
```

9. Escribe un método que reciba 2 pilas ordenadas en orden creciente y devuelva una pila con los elementos de ambas ORDENADA en orden decreciente (**3 puntos**)
PISTA: 1) comparar cimas; 2) desapilar la adecuada; 3) apilar en la pila resultado

Solución:

```
public static SStack<Integer> mezclarPilas(SStack<Integer> s1, SStack<Integer> s2) {
    SStack<Integer> sTemp=new SStack<Integer>();

    while (!s1.isEmpty() && !s2.isEmpty()){
        if (s1.top()<s2.top()) sTemp.push(s1.pop());
        else sTemp.push(s2.pop());
    }
    //puede que en la pila s1 queden elementos
    while (!s1.isEmpty()){
        sTemp.push(s1.pop());
    }

    //puede que en la s2 todavia queden elementos
    while (!s2.isEmpty()){
        sTemp.push(s2.pop());
    }
    return sTemp;
}
```

10. Implementa de forma recursiva el algoritmo **getAt(index)** en una Cola de enteros (*Integer*) implementada con una lista simple (*SQueue*). Este método devuelve el elemento en la posición *index*. NOTA importante: no es necesario conservar la cola (**3 puntos**)

Solución:

```
/** implementación recursiva del algoritmo getAt en una Cola
 * devuelve el elemento que se encuentre en la Cola en el índice indicado */
public static Integer getAtRec(int index, SQueue<Integer> q){
    if (index > q.getSize() || index <=0) return -1; // error, index mayor que tamaño de cola
    else if (index==1) return q.dequeue();
    else{
        q.dequeue();
        return getAtRec(index-1,q);
    }
}
```