Estructuras de Datos 1 - ST0245
Segundo Parcial

Nombre .....
Departamento de Informática y Sistemas
Universidad EAFIT

Octubre 23 de 2018

Para propósitos de este parcial, se considerará esta implementación de un árbol binario y sus estos recorridos.

```
1  //Arbol binario
2  class BNode{
3    BNode izq;
4    BNode der;
5    int val;
6  }
7
8  //Recorridos
9  void preorden(BNode nodo){
10   if(nodo != null){
11     System.out.println(nodo.val);
12     preorden(nodo.izq);
13     preorden(nodo.der);
14   }
15 }
16 void posorden(BNode nodo){
17   if(nodo != null){
18     posorden(nodo.izq);
19     posorden(nodo.der);
20     System.out.println(nodo.val);
21   }
22 }
23 void inorden(BNode nodo){
24   if(nodo != null){
25     inorden(nodo.izq);
26     System.out.println(nodo.val);
27     inorden(nodo.der);
28   }
29 }
```

1 Pilas y Colas 30%

Polka desea implementar una cola de una manera muy especial: Él ya tiene implementada una pila, pero no quiere implementar una cola desde el principio, entonces él ha decido usar la pila que ya tiene implementada para implementar la cola. Él quedará conforme si implementa las dos funciones principales de la cola: `offer(e)` (añade el elemento e al inicio de la cola), `poll()` (extrae y elimina el primer elemento que entró a la cola). Él ya implementó `offer(e)`, pero no ha podido con `poll()`, entonces él quiere que tú lo implementes. Las variables $s1, s2$ son dos pilas que se inicializan en el constructor de la clase `Cola`.

Nota: En una pila, la función `pop()` retorna y extrae el elemento en el tope de la pila, y la funcion `push(x)` añade el elemento x al tope de la pila.

Nota 2: En la vida real no se implementa una cola con pilas porque no es eficiente.

```
1  class Cola{
2    Stack<Integer> s1, s2;
3    Cola(){
4      s1 = new LinkedList<>();
5      s2 = new LinkedList<>();
6    }
7    void offer(int e){
8      s1.push(e);
9    }
10   int poll(){
11     //Completa por favor
12     if(s2.isEmpty()){
13       while(_______________)
14           s2.push(___________);
15       }
16     }
17     return ___________;
18   }
19 }
```

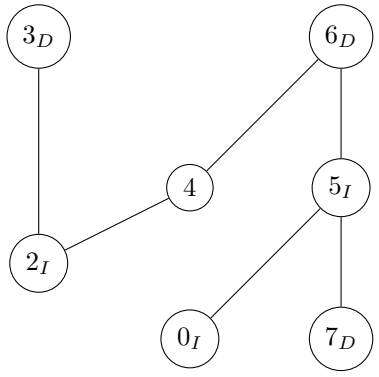
(i) (10%) Completa la línea 13 .....

(ii) (10%) Completa la línea 14 .....

(iii) (10%) Completa la línea 17 .....

2 Árboles 30%

Nota: El sub-índice D indica que el nodo es un hijo derecho con respecto a su padre. El sub-índice I indica que es un hijo izquierdo con respecto a su padre.



a (10%) ¿Cuál es la salida del recorrido en *in-orden* del árbol anterior, tomando el nodo 4 como el nodo raíz?

(i) 3, 2, 0, 7, 5, 6, 4

(ii) 2, 3, 4, 0, 5, 7, 6

(iii) 4, 2, 3, 6, 0, 5, 7

(iv) 4, 7, 3, 6, 2, 0, 5

b (10%) Asuma que el nodo 4 es la raíz del árbol binario anterior. Asuma que la salida del recorrido *pre-orden* es $\{a_1, a_2, a_3, ..., a_n\}$ y la salida del recorrido *in-orden* es $\{b_1, b_2, b_3, ..., b_n\}$. ¿Cuál es el primer valor de i para el cual $a_i = b_i$? Note que la i empieza en 1.

(i) 5

(ii) 4

(iii) 3

(iv) 2

Por ejemplo, asuma que las salidas son $a = \{4, 6, 5, 0, 7, 2, 3\}$ y $b = \{6, 4, 7, 0, 5, 2, 3\}$. La respuesta sería 4 porque $a_4 = b_4 = 0$ e $i = 4$ es el primer valor para el que $a_i = b_i$.

c (10%) ¿Es un ***árbol binario de búsqueda*** el árbol anterior?

(i) Sí

(ii) No

3 Listas 30%

El método `add(x)` agrega el elemento x al fin de la lista. El método `get(i)` retorna el valor en la posición i de la lista. El metodo `remove(i)` elimina el elemento en la posición i en la lista. Considere el siguiente método y asuma que el método se llama con una lista vacía l. El operador % es el residuo de la división entera de dos enteros.

```
1  int metodo
2  (LinkedList<Integer> l,int a,int b){
3    int i=0;
4    for(int j=a;j<=b;j++){
5      int s=j;
6      while(s > 0){
7        l.add(s%10);
8        i=(i+1)%l.size();
9        s=s/10; //division entera
10     }
11   }
12   return l.get(i);
13 }
```

a (10%) ¿Cuál es la complejidad asintótica, **en el peor de los casos**, del algoritmo anterior? Asuma que $a - b < 0$.

(i) $O(|b-a| \times \log_{10} b)$

(ii) $O(|b-a|)$

(iii) $O(|b-a| \times \log_2 |b-a|)$

(iv) $O(|b-a|^2)$

b (10%) ¿Qué imprime el algoritmo anterior cuando $a = 2, b = 7$? Como un ejemplo, para $a = 2, b = 5$ la respuesta es 5.

(i) 3

(ii) 7

(iii) 4

(iv) 5

c (10%) ¿Cuál es la complejidad asintótica, **en el peor de los casos**, de `remove(i)` en una lista enlazada?

(i) $O(\log n)$

(ii) $O(n)$

(iii) $O(1)$

(iv) $O(n^2)$

4 Grafos 10%

Un grafo con matrices de adyacencia se puede definir como `int [][] grafoAM`. El método para obtener los sucesores de cierto vértice (`vertex`) tiene que recorrer la matriz buscando los vértices a los que se puede llegar a partir de `vertex`. Las filas representan el origen y las columnas el destino. En la matriz se guarda el peso que hay para llegar de un origen a un destino. Si no hay conexión entre un origen y un destino, se coloca el valor 0 en esa casilla. Juanito hizo el siguiente código, ayúdalo a completarlo.

```
1  class Graph {
2    private int [][] grafoAM;
3    public ArrayList<Integer>
         getSuccessors(int vertex) {
4      ArrayList<Integer> sucesores = new
           ArrayList<Integer>();
5      for(int i = 0;i < size;i++){
6        if(.....){
7              sucesores.add(i);
8        }
9      }
10     return sucesores;
11   }
12 }
```

a (10%) Complete la línea 6 .....