**Práctico 1.**

**INF310-SX Estructuras de Datos II. Gestión 2-2019**

|  |  |
| --- | --- |
| 1. Una **circular single-list** es una lista enlazada simple y común de enteros, con la diferencia de que su último nodo (cuando la lista no es vacía) tiene su Link apuntando al primer nodo.  Usando una circular single-list y **un solo** puntero raíz L que apunte al último nodo, implemente un ADT Bicola de enteros.  Bicola() //Construye una bicola vacía.  void add(int x) //inserta x al final de la bicola.  void addToFront(int x) //inserta x al principio de la bicola.  int pop() //Remueve el 1er elemento de la bicola,  //devolviendo (return) éste elemento.  int popLast() //Remueve el último elemento de la bicola,  //devolviendo (return) éste elemento.  int lenght() //Return la cantidad de elementos.  *Para implementar esta class, cree un nuevo package (paquete), en el código fuente.*  2. Usando una (sola) Single-List de enteros, implemente un ADT ColaX, cuyas operaciones son:  **public class ColaX {**    **public ColaX(){**  //Construye una cola vacía  **}**    **public void add(int x){**  **/\***  Si x NO existe, x se inserta al final de la cola (como en una cola normal).  Si x existe, pero x está en el frente de la cola, “no pasa nada”.  Si x existe y x no está en el frente de la cola, x se desplaza una posición  hacia el frente de la cola.  **\*/**  **}**    **public int pop(){**  //Extrae el 1er elemento de la cola (como en una cola normal)  **}**  **}**  **Por ejemplo:**  **ColaX p = new ColaX();** //p=[ ].  **p.add(9);** //p=[ **9** ]. Primera inserción.  **p.add(8);** //p=[9, **8**]. Como el 8 no existe, se deposita al  // final de la cola. | **p.add(4);** //p=[9, 8, **4**]. Como el 4 no existe, se deposita al final  //de la cola.  **p.add(4);** //p=[9, **4**, 8]. Como el 4 existe y no está en el frente,  //el 4 se mueve una posición hacia el frente.  **p.add(9);** //p==[9, 4, 8]. El 9 existe, pero está en el frente de  //la cola. Entonces, “no pasa nada”.  **p.add(6);** //p=[9, 4, 8, **6**]. Como el 6 no existe, se deposita al  //final de la cola.  **p.add(8);** //p=[9, **8**, 4, 6]. Como el 8 existe y no está en el frente,  //el 8 se mueve una posición hacia el frente.  **int x = p.pop();** //x=9, p=[8, 4, 6]. El primer elemento (el 9) es  //sacado de la cola.  *Para implementar esta class, cree un nuevo package (paquete), en el código fuente.*  3. En la class Lista (ver código fuente), implementar el procedimiento  **public void del(int k)**  //Elimina el elemento de la **posición** k de la lista (las posiciones se enumeran  //desde el 0). Si la posición k no existe, “no pasa nada”.  **Ejemplo de Uso:**  Lista P=new Lista(); //P=[ ]  P.add(5); //P=[ 5]  P.add(8); //P=[ 5,8]  P.add(3); //P=[ 3, 5, 8]  P.**del**(1); //P=[ 3, 8] (el 5 estaba en la pos. 1)  P.**del**(6); //P=[ 3, 8] (La pos. 6 no existe)  4. En la class Lista (ver código fuente), implementar **RECURSIVAMENTE** la función  **public boolean exist(int x)**  //Devuelve true si x está en la lista. Si x no está en la lista, devuelve false.  // Este método obviamente debe usar una función máscara recursiva.  **Ejemplo de Uso (siguiendo el de la pregunta 3):**  boolean b=P.exist(15); //b=false. (15 no existe).  b=P.exist(8); //b=true. (8 está en la lista P). |

5. En un BST de enteros, decimos que un nodo N con data=d es un ladder (escalera) si tiene **exactamente dos hijos**, su hijo izquierdo tiene un data=d‒1 y su hijo derecho tiene data=d+1

N

En la class Arbol, escriba la función

**public int cantLadders()**

la cual devuelve (return) la cantidad de nodos ladder que tiene el árbol. Por ejemplo;

(En los gráficos no se dibujan los punteros null)

Árbol B

Árbol A

int x=A.cantLadders(); //x=3

x=B.cantLadders(); //x=0

|  |  |
| --- | --- |
| 6. En la class Arbol, implementar la función  **public int cantGajos()**  //Devuelve la cantidad de gajos que tiene el árbol.  //Un gajo es simplemente un hijo no-nulo de un nodo.  Por ejemplo:  cantGajos()= 4  //En total, el árbol tiene 4 hijos no-nulos.  7. En la class Arbol (Bst) escriba el método  **public void descendientes(int x)**  el cual muestre en consola (print) todos los nodos descendientes de x. Si x no existe “no pasa nada”.  Los descendientes de **x** son todos los nodos del árbol izquierdo y del árbol derecho de x (no se incluye a x).  *Note que una hoja no tiene descendientes.* | 8. En la class Arbol, implementar la función  **public boolean isPadre(int p, int h)**  //Devuelve true si p es padre de h.  //Si p o h no existen, o p no es padre de h, devuelve false.  Por ejemplo:  padre(30,40)=true  padre(40,30)=false  padre(20,50)=false  padre(80,40)=false  padre(20,25)=true  9. En la class Arbol (ver código fuente), implementar la función  **public int isHoja(int x)**  //Devuelve true si x es una hoja.  //Si x no es una hoja o x no existe, devuelve false.  Por ejemplo:  isHoja(30)= false  isHoja(10)= true  isHoja(50)= true  isHoja(80)= false |

10. En la class árbol binario de enteros, escribir el procedimiento:

**public void delHoja(int x)**

la cual quite del árbol aquella hoja cuyo Data es x. **Si x no existe o x no es hoja**, este procedimiento **no realiza ninguna acción**.

Por ejemplo:

(En los gráficos no se dibujan los punteros null)

Árbol A

**delHoja(5);**

*Tome en cuenta que la Raíz también puede ser una hoja.*

**NOTA.** *Todos los problemas de este práctico, deben ser implementados en computadora.*