Estructuras de Datos - Primer Parcial

Licenciatura en Ciencias de la Computación – Ingeniería en Computación – Ingeniería en Sistemas de Software
Universidad Nacional del Sur – 25/4/2023

(B)

Observaciones generales:

- **REALICE LOS EJERCICIOS EN HOJAS SEPARADAS** ← ← ← ✥
- Lea todo el ejercicio antes de comenzar a desarrollarlo.
- Numere y ponga su nombre a todas las hojas. Indique cuántas hojas entrega. ← ← ← ✥
- Recuerde que se evalúan corrección, eficiencia y legibilidad de sus soluciones.

(17)

Ejercicio 1:

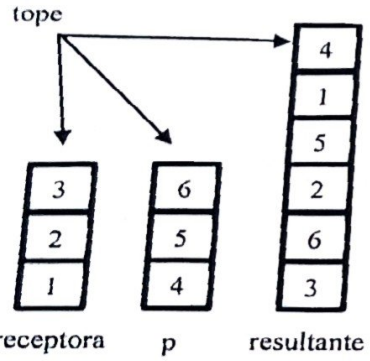
a) En el lenguaje Java, defina la interfaz *Stack<E>* necesaria para implementar una pila genérica de tipo E. Defina solo encabezado y las operaciones top, push y pop.

b) En el lenguaje Java programe la clase *Nodo<E>* necesaria para implementar una pila con enlaces. Implemente solo el encabezado, atributos de instancia y el o los constructores.

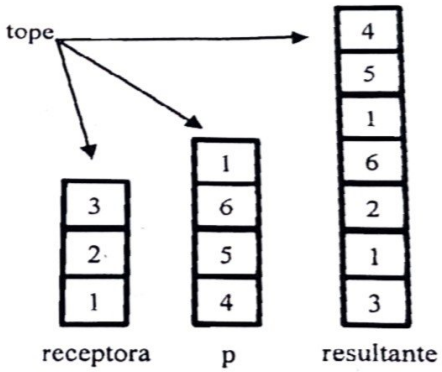
c) En el lenguaje Java programe la clase *PilaEnlazada<E>* que implemente una pila con enlaces tal como la que se vio en clase. Implemente solo el encabezado, atributos de instancia, el o los constructores y el método push.

d) Agregue a la clase *PilaEnlazada<E>* definida en el inciso anterior un método tal que reciba por parámetro una pila genérica *p* y retorne una nueva pila producto de intercalar el contenido de la pila receptora del mensaje y la pila *p*. Tenga en cuenta que luego de ejecutarse este método ambas pilas quedarán vacías. Implemente todos los métodos auxiliares implementados. No debe implementar los métodos correspondientes a los mensajes que se le envíen a la pila *p*. Las pilas pueden tener distintos tamaños.

Ejemplo 1:



Ejemplo 2:



e) Calcule el orden del tiempo de ejecución de la solución planteada en el inciso **(d)**. Justifique adecuadamente.

Ejercicio 2:

a) Defina el encabezado de la interfaz *PositionList<E>* y las signaturas de las operaciones *first()*, *next(p)* y *addBefore(p,e)*. La interfaz *PositionList<E>* corresponde a la dada en clase.

b) Defina la interfaz *Position<E>* completa.

c) Escriba en Java un método tal que dada una lista L genérica modifique el estado interno de L tal que se concatene al final el contenido inicial de L pero en orden inverso. Implemente todos los métodos auxiliares utilizados. Asuma que cuenta con el TDALista totalmente implementado. Para implementar este método no debe utilizar ninguna ED auxiliar. No debe usar iteradores (Iterator) para resolver este ejercicio.

Ejemplo:
L = ["hola", "que", "tal"] → L=["hola", "que", "tal", "tal", "que", "hola"]

d) Escriba en Java un método que reciba una lista de colas de caracteres y retorna una lista de caracteres que respeten el orden original. En este caso las colas no se deben destruir. Puede asumir que el TDA cola cuenta con un método clone. Implemente todos los métodos auxiliares utilizados. Asuma que cuenta con el TDALista y TDACola totalmente implementados. No debe usar iteradores (Iterator) para resolver este ejercicio.

Ejemplo: (las colas internas se representan rodeadas por | y |)
L = [|a,b,z,q|,|d,m|,|z|,|f,g,y|,|a,k|]

LRes = [a,b,z,q,d,m,z,f,g,y,a,k]

e) Justificando adecuadamente, calcule el tiempo de ejecución del procedimiento solución al inciso (d), asumiendo que las operaciones first(), last(), next(p), addAfter(p,x), addFirst(x) son de tiempo constante mientras que addBefore(p,x), prev(p), remove(p) son de tiempo lineal. Detalle la estructura subyacente del TDA Cola para justificar el tiempo de ejecución de sus operaciones.

PositionList<E>	
size( )	addAfter(p,e)
isEmpty()	addBefore(p,e)
first()	remove(p)
last()	set (p, e)
next(p)	positions()
prev(p)	iterator()
addFirst(e)	
addLast(e)	