SOLUCIÓN CC3001 Control 1 Otoño 2023

Profs. Nelson Baloian, Benjamín Bustos, Sebastián Ferrada y Patricio Poblete

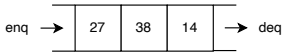
6 de octubre de 2023

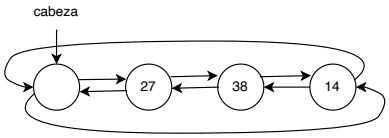
Instrucciones:

- Tiempo: 2 horas
- El control es INDIVIDUAL
- Entregue cada pregunta en una hoja separada
- No olvide poner su nombre en cada hoja
- Está permitido usar apuntes, ya sea en papel o en formato digital
- Si usa un dispositivo para revisar su apunte en formato digital, está completamente prohibido utilizarlo para cualquier otra finalidad. En particular, no se puede usar chat, ni mail ni ninguna otra forma de comunicación con otra persona o inteligencia artificial. Tampoco puede usarlo para fotografiar el enunciado.

P1. Nombre: _____

En el apunte aparecen dos implementaciones para el TDA Cola: usando un arreglo y usando una lista de enlace simple. En este problema se le pide que usted implemente una cola usando una **lista de doble enlace**. Usted debe escribir el constructor de la clase (que inicializa una cola vacía), y los métodos `is_empty`, `enq` y `deq`. Acá hay un ejemplo de una cola y de su representación como lista de doble enlace:






```
class NodoLista:
    def __init__(self,prev,info,sgte):
        self.info=info
        self.prev=prev
        self.sgte=sgte
class Cola:
    def __init__(self):
        self.cabeza=NodoLista(None,0,None)
        self.cabeza.prev=self.cabeza
        self.cabeza.sgte=self.cabeza
    def enq(self,x):
        p=NodoLista(self.cabeza,x,self.cabeza.sgte)
        self.cabeza.sgte.prev=p
        self.cabeza.sgte=p
    def deq(self):
        assert self.cabeza.sgte is not self.cabeza
        x=self.cabeza.prev.info
        q=self.cabeza.prev.prev
        self.cabeza.prev=q
        q.sgte=self.cabeza
        return x
    def is_empty(self):
        return self.cabeza.sgte is self.cabeza
```

P2. Nombre: _____

1. [3 puntos] Resuelva la siguiente ecuación de recurrencia mediante el método del polinomio característico:

$$a_n = a_{n-1} + 12a_{n-2}$$

$$a_0 = 0,\ a_1 = 7$$

Solución:

$$\lambda^n = \lambda^{n-1} + 12\lambda^{n-2}$$
$$\lambda^2 - \lambda - 12 = 0$$
$$\lambda = -3, 4$$
$$a_n = A4^n + B(-3)^n$$
$$a_0 = 0 \Rightarrow B = -A$$
$$a_1 = 7 \Longrightarrow A = 1$$
$$\Longrightarrow a_n = 4^n - (-3)^n$$

2. [3 puntos] Resuelva la ecuación

$$f(n) = 2f(\sqrt{n}) + 1$$

Sugerencia: Haga el cambio de variable $n = 2^k$ y aplique el Teorema Maestro
Solución:

$$f(2^k) = 2f(2^{k/2}) + 1$$

Definiendo $T(k) = f(2^k)$ tenemos

$$T(k) = 2T(\frac{k}{2}) + 1$$

Esto corresponde al caso $r = 0$, $p = 2$, $q = 2$ del Teorema Maestro, por lo tanto

$$T(k) = \Theta(k^{\log_2 2}) = \Theta(k)$$

y deshaciendo el cambio de variable y de función,

$$f(n) = \Theta(\log_2 n)$$

P3. Nombre: _____

Dado un arreglo $A[0], \ldots, A[n-1]$, se define un elemento mayoritario como aquel elemento que aparece en más de la mitad de los casilleros ($> n/2$ apariciones). Suponga que no existe relación de orden entre los elementos, y sólo se permite preguntar si $A[i] == A[j]$, instrucción que se realiza en tiempo constante. Diseñe un algoritmo basado en dividir-para-reinar que determine si el arreglo A tiene un elemento mayoritario en tiempo $O(n \log n)$. Describa su algoritmo en palabras, no necesita programarlo. Para simplificar, puede suponer que n es una potencia de 2. Demuestre que su algoritmo cumple con la complejidad temporal exigida. (Sugerencia: divida el arreglo por la mitad.)

Solución:

Caso base de la recursión:

- Arreglo de tamaño uno, en cuyo caso el valor almacenado es un elemento mayoritario [1 pto.].

Llamado recursivo:

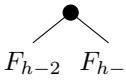
- Se divide el arreglo en dos. Se calcula recursivamente si cada mitad tiene elemento mayoritario [0.5 ptos.].
- Si ninguna de las dos mitades tiene elemento mayoritario, el arreglo original tampoco tiene (si el arreglo original tiene un elemento mayoritario, al menos una de las mitades tiene un elemento mayoritario) [1 pto.].
- Si una de las mitades tiene elemento mayoritario, se busca dicho valor en la otra mitad, contando cuántas veces se repite en el arreglo [1 pto.].
- Si en total el valor se repite $> n/2$ veces, el arreglo tiene un elemento mayoritario [0.5 ptos.].
- Si ambas mitades tienen un elemento mayoritario distinto, hay que realizar este proceso dos veces, quedarse con el valor que se repita más veces, y verificar si se repite $> n/2$ veces [0.5 ptos.].

Complejidad:

- Se realizan dos llamados recursivos de tamaño $n/2$. Buscar y contar un valor en una de las mitades del arreglo toma $n/2$ comparaciones, por lo tanto toma tiempo $O(n)$ [0.5 ptos.].
- Esto a lo más se realiza dos veces, por lo que el costo extra total es $O(n)$ [0.5 ptos.]
- Planteando la ecuación de recurrencia se obtiene $T(n) = 2T(n/2) + O(n)$, que por el Teorema Maestro tiene solución $O(n \log n)$ [0.5 ptos].

P4. Nombre:

Un árbol de Fibonacci de altura h se define de la siguiente manera:

$F_0 =$ □ $\quad F_1 =$  $\quad F_h =$ 

Escriba una función recursiva que chequee si un árbol dado es un árbol de Fibonacci o no. En caso de que lo sea, debe retornar su altura, y en caso contrario debe retornar un -1.

Su solución puede ser de una de las dos formas siguientes. Escoja una de las dos y diga cuál eligió:

- Suponiendo que en el árbol las hojas son punteros nulos, la solución debe ser una función `es_Fibonacci(p)`, donde `p` es el puntero a la raíz del árbol, o bien
- Suponiendo que el árbol tiene nodos de tipo `Nodoi` y `Nodoe`, la solución debe agregar un método `es_Fibonacci()` a cada una de esas clases.

Solución:

```
# Versión con punteros nulos
def es_Fibonacci(p):
    if p is None:
        return 0
    if p.izq is None and p.der is None:
        return 1
    h_i=es-Fibonacci(p.izq)
    h_d=es_Fibonacci(p.der)
    if h_i>=0 and h_d>=0 and h_d==h_i+1:
        return h_d+1
    return -1
```

```
# Versión con Nodoe, Nodoi
class Nodoe:
    ...
    def es_Fibonacci(self):
        return 0
    ...
class Nodoi:
    ...
    def es_Fibonacci(self):
        if self.izq is Nodoe and self.der is Nodoe:
            return 1
        h_i=self.izq.es_Fibonacci()
        h_d=self.der.es_Fibonacci()
        if h_i>=0 and h_d>=0 and h_d==h_i+1:
            return h_d+1
        return -1
    ...
```