Guía Pre Control 1

P1. P1 Control Otoño 2023

sgte_dato(): entrega el sgte dato de una secuencia de entrada

Salida: 1) Todos los números negativos, en orden inverso al orden que aparecieron
2) Todos los números positivos, en el orden en que aparecieron

Recordemos

Pila: lista LIFO (Last-In-First-Out) → los elementos de la pila salen en el orden inverso en que ingresaron

Cola: lista FIFO (First-In-First-Out) → los elementos van saliendo en orden de llegada.

Solución

```
cola = Cola()          # Creamos las dos estructuras a usar y
pila = Pila()          # una variable para guardar el dato actual
valor = sgte_dato()
while valor != 0:  # Mientras no lleguemos al último valor.
  if valor < 0:
      pila.push(valor)     # Guardamos los valores en sus pilas y
  else:                    # colas correspondientes.
      cola.enq(valor)
  valor = sgte_dato()   # Actualizamos el valor

# Ahora queda imprimir los valores
# Primero la Pila con los valores negativos
while not pila.is_empty():
    print(pila.pop())
# Ahora la cola con los valores positivos
while not cola.is_empty():
    print(cola.deq())
```

P2. Ecuaciones de recurrencia.

$a_n = 3a_{n-1} + 4a_{n-2}$
$a_1 = 2$
$a_0 = 3$

2.1 Parte 1

Asumimos que $a_n = \lambda^n$, entonces la recurrencia es

$$\lambda^n = 3\lambda^{n-1} + 4\lambda^{n-2}$$

Dividimos toda la recurrencia por λ^{n-2}, nos queda

$$\lambda^2 - 3\lambda - 4 = 0$$

Las raíces de esta ecuación son $\phi = 4$ y $\hat{\phi} = -1$.

Asumimos que la solución es de la forma $a_n = A\phi^n + B\hat{\phi}^n$. Usando las condiciones iniciales, tenemos que $a_0 = 2 = A + B$ y $a_1 = 3 = 4A - B$. Resolviendo estas dos ecuaciones tenemos que $A = 1$ y $B = 1$. De esa forma, la ecuación queda:

$$a_n = 4^n + (-1)^n$$

Noten que el término $(-1)^n$ es constante sin importar el valor de n. Por lo tanto la recurrencia se resuelve como

$$a_n = \Theta(4^n)$$

2.2 Parte 2

Tenemos la recurrencia:

$$f(n) = 2f(\sqrt{n}) + \log_2 n$$

Asumimos que $n = 2^k$, por lo tanto $k = \log_2 n$ y $\sqrt{n} = 2^{k/2}$. La recurrencia queda

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

Ahora haremos un cambio de función $s(k) = f(2^k)$, por lo tanto

$$s(k) = 2s(k/2) + k$$

La ecuación para la función s se puede resolver con el teorema maestro, en donde $p = 2$, $q = 2$, $C = 1$ y $r = 1$. Del teorema maestro cumple el segundo caso en donde $p = q^r$. Por lo tanto la solución de s es:

$$s(k) = \Theta(k \log k)$$

Reemplazando $k = \log_2 n$, tenemos

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

c) $$a_n = 2a_{n-1} + a_{n-2}$$
$$a_0 = 0$$
$$a_1 = 4$$

Cambio de variable $a_n = \lambda^n$

$$\lambda^n = 2\lambda^{n-1} + \lambda^{n-2} \;/ : \lambda^{n-2}$$

$$\lambda^2 = 2\lambda + 1 \Rightarrow \lambda^2 - 2\lambda - 1 = 0$$

$$\lambda_{1,2} = \frac{2 \pm \sqrt{4 - 4\cdot(-1)}}{2} = \frac{2 \pm \sqrt{8}}{2} = \frac{2 \pm 2\sqrt{2}}{2} = 1 \pm \sqrt{2}$$

$$\lambda_1 = 1 + \sqrt{2} \qquad \lambda_2 = 1 - \sqrt{2}$$

Eso da una solucion del tipo

$$\lambda^n = A\lambda_1^n + B\lambda_2^n$$

Caso $a_0 = 0 \rightarrow 0 = A + B$

Caso $a_1 = 4 \rightarrow 4 = A(1+\sqrt{2}) - A(1-\sqrt{2})$

$$4 = A(\cancel{1} + \sqrt{2} - \cancel{1} + \sqrt{2})$$

$$\frac{4}{2\sqrt{2}} = A \rightarrow A = \frac{2}{\sqrt{2}} = \frac{2\sqrt{2}}{2} = \sqrt{2}$$

$$\lambda^n = \sqrt{2}\,(1+\sqrt{2})^n - \sqrt{2}\,(1-\sqrt{2})^n$$

$|1-\sqrt{2}| < 1 \rightsquigarrow$ va a tender a 0

$$O\left((1+\sqrt{2})^n\right)$$

P3. P2.2 Control Otoño 2023

$$T(n) = 8T(n-1) - 15T(n-2)$$
$$T(0) = 1, \quad T(1) = 1$$

a)
```
def T(n):
    if n==0:
        return 1
    elif n==1:
        return 1
    return 8·T(n-1) - 15·T(n-2)
```

Tiempo: Exponencial utilitando una intuición similar a lo que pasa con las sumas de fibonacci

b)
```
def T(n):
    values = np.zeros(n+1)
    values[0] = 1
    values[1] = 1
    for k in ranges(2, n+1):
        values[k] = 8 values[k-1] - 15 values[k-2]
    return values[n]
```

$O(n)$ → hay un ciclo for que hace n pasos.

c) $T(n) = \lambda^n$

$$\lambda^n = 8\lambda^{n-1} - 15\lambda^{n-2} \quad / : \lambda^{n-2}$$

$$\lambda^2 = 8\lambda - 15 \quad \rightarrow \quad \lambda^2 - 8\lambda + 15 = 0$$

$$\lambda_1 = 5 \qquad \lambda_2 = 3$$

$$\lambda^n = A\lambda_1^n + B\lambda_2^n$$

$T(0) = 1$ $\qquad 1 = A + B \qquad A = 1 - B$

$$\lambda^n = (1-B)\lambda_1^n + B\lambda_2^n$$

$T(1) = 1$

$$1 = (1-B)\cdot 5 + B\cdot 3$$

$$1 = 5 - 5B + 3B$$

$$1 - 5 = -5B + 3B$$

$$-4 = -2B \longrightarrow B = 2$$

$$A = 1 - 2 = -1$$

$$\lambda^n = -1(5)^n + 2(3)^n$$

P4. Lista Rotada

lista original [13, 20, 34, 41, 55, 62, 75, 84, 93]
minimo

lista rotada [75, 84, 93, 13, 20, 34, 41, 55, 62]
minimo

↳ es creciente hasta que aparece el minimo y luego vuelve a ser creciente.
↳ los números en verde son mayores a los números en morado

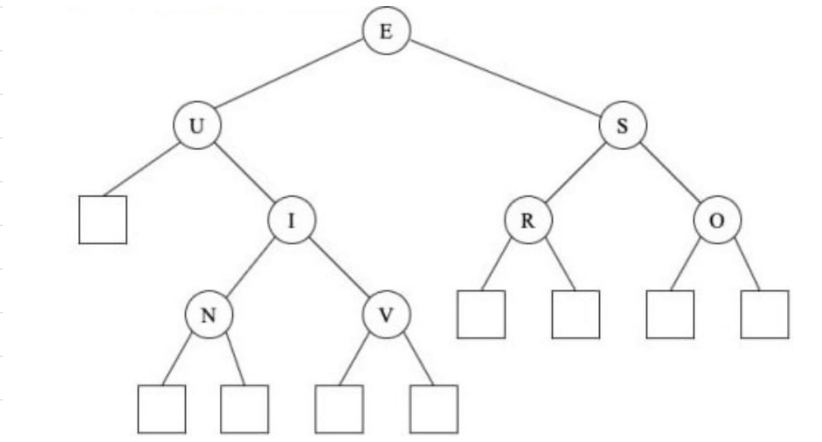
```python
# Comentaremos con una función auxiliar
def minimo(a, ini, fin): # buscar el minimo entre dos rangos.
    if ini == fin:
        return ini
    mid = (ini+fin)//2
    if mid < fin and a[mid+1] < a[mid]:
        return mid+1
    if mid > ini and a[mid] < a[mid-1]
        return mid

    if a[fin] > a[mid]:
        return minimo(a, ini, mid-1)
    else:
        return minimo(a, mid+1, fin)


def encuentra_minimo(a):
    return minimo(a, 0, len(a)-1)
```

La complejidad de esta función es básicamente la misma que del algoritmo de búsqueda binaria. Una vez que se analiza el elemento del medio, se decide analizar solo la primera o segunda mitad de la lista. Por eso es $O(\log n)$

P5. Árbol de letras



a) **Altura**

```
def Altura(arbol):
    def obtenerAltura(nodo):
        # Caso base
        if nodo is None:
            return 0
        else:
            return 1 + max(obtenerAltura(nodo.izq), obtenerAltura(nodo.der))
    return obtenerAltura(arbol.raiz)
```

b) **Preorden:** Visitar la raiz, recorrer el subárbol izquierdo y recorrer el subárbol derecho

"E - U - I - N - V - S - R - O"

Inorden: Recorrer el subárbol izquierdo, visitar la raiz, recorrer el subárbol derecho

"U - N - I - V - E - R - S - O"

Postorden: Recorrer el subárbol izquierdo, recorrer el subarbol derecho y visitar la raiz

"U - N - V - I - R - O - S - E"

P6. Buscar elementos en arreglos ordenados

```
def buscar(A, x):
    if A[1] == x:
        return 1
    k = 2
    while A[k] != ∞ :
        if A[k] > x:
            return busquedaBinaria(A, x, i//2, i)

    return búsquedaBinaria(A, x, i//2, i)
```

Peor Caso: x se encuentra casi al final del arreglo

Tiempo: tiempo para llegar a $k > n$
+ tiempo de búsqueda binaria entre k y $k/2$ → $k/2$ n°

Sea k, sabemos que $k > n$ y $k/2 < n$
diremos que $k = 2n - 2$ $\rightsquigarrow \log(2n-2) + \log(n-1)$

Como vamos avanzando de 2 por 2 ↳ $\log(2n^2 - \cancel{2n} \cancel{+2n} + 2)$

$2^i = k \rightarrow i = \log_2(k)$

$\log(2n^2 + 2)$

$O(\log(n^2))$

P7. Matriz ascendente

$$\begin{pmatrix} 12 & 28 & 35 & 56 & 72 \\ 25 & 33 & 40 & 61 & 80 \\ 37 & 44 & 52 & 65 & 84 \\ 50 & 60 & 71 & 86 & 90 \end{pmatrix}$$

a) Búsqueda secuencial : Pasar por todos los números (Peor Caso)
$O(m \times n)$

b) Búsqueda Binaria en cada fila: Pasa por cada fila (Peor Caso)
$O(m \times \log(n))$
↳ Búsqueda Binaria
↳ Cantidad de filas.

c) Si $x < a_{m,1}$ se descartan n elementos correspondiente a la fila m
$x > a_{m,1}$ se descartan elementos correspondientes a la columna 1

En cada iteración se descarta toda una fila o toda una columna por lo que el peor caso es tener que descartar todas las filas y columnas $O(m+n)$

P8. Filtrando números positivos

→ (1) → (3) → (4) → (-6) → (7) → (-10) → (8)   Caso 1

→ (-1) → (3) → (4) → (-6) → (8)   Caso 2

→ (1) → (3) → (4) → (-6) → (-8)   Caso 3

```
def filtrarpositivos(self):   # dentro de la clase lista enlazada
    def filtrar(nodo):
        if nodo.sgte != None:
            if nodo.sgte.info ≤ 0:  # Hay que sacarlo
                nodo.sgte = nodo.sgte.sgte
                filtrar(nodo)              } Se vuelve a revisar en caso
            else:                            de que hayan dos o más
                filtrar(nodo.sgte)           negativos seguidos.
        return
    cabecera = self.cabecera
    while cabecera is not None and cabecera.info ≤ 0:   } Buscar el
        cabecera = cabecera.sgte                          primer positivo
    filtrar(cabecera)
    self.cabecera = cabecera.
```