PEP 2 _ TDA's

Rodrigo Pereira Yañez
16.610.470-K

14/12/24

PROBLEMA 1 (10 puntos):

Sea P una pila (tope el primero de la izquierda) y C una cola (frente el primero de la izquierda), inicializadas como: P = [2, 3, 4, 2, 2, 3, 4, 2] y C = [2, 1, 2, 3] ¿cuál es el estado de la lista L después de aplicar el algoritmo de la figura nro. 1?

```
algoritmo(pila *P, cola *C): lista*
    lista *L ← crea_lista()
    num valor
    while (not(es_cola_vacia(C))) do
        valor ← frente(C)→dato
        decolar(C)
        num s ← 0
        num i ← 0
        while not(es_pila_vacia(P)) and (i < valor) do
            s ← s + tope(P)→dato
            desapilar(P)
            i ← i + 1
        inserta_inicio(L, s)
    return L
```

Figura nro. 1: algoritmo pregunta nro. 1.

Sol: P = [2, 3, 4, 2, 2, 3, 4, 2]
C = [2, 1, 2, 3]

0+4=4
s+ tope(P)→dato

while	valor	decolar(C)	s	i	while	s	desapilar(P)	i	inserta_inicio(L,s)
—	—	C=[2,1,2,3]	-	-	———	-	P=[2,3,4,2,2,3,4,2]	-	———
!F=t	2	C=[1,2,3]	0	0	!F=t ∧ t=t	2	P=[3,4,2,2,3,4,2]	1	
					!F=t ∧ t=t	5	P=[4,2,2,3,4,2]	2	
					!F=t ∧ F=F				
									L=[5]
!F=t	1	C=[2,3]	0	0	!F=t ∧ t=t	4	P=[2,2,3,4,2]	1	
					!F=t ∧ F=F				

↓ Continua...

7 + 2 = 9

S ← tope(P) → dato

while	valor	descolar (C)	S	i	while	S	desapilar (P)	i	inserta. inicio (L, S)
									L = [4, 5]
!F = t	2	C = [3]	0	0	!F = t ∨ t = t	2	P = [2, 3, 4, 2]	1	
					!F = t ∨ t = t	4	P = [3, 4, 2]	2	
					!F = t ∨ F = F				
									L = [4, 4, 5]
!F = t	3	C = [ ]	0	0	!F = t ∨ t = t	3	P = [4, 2]	1	
					!F = t ∨ t = t	7	P = [2]	2	
					!F = t ∨ t = t	9	P = [ ]	3	
					!t = F ∨ F = F				
									L = [9, 4, 4, 5]
!t = F									
—	—	———	-	-	———	-	———	-	———

return L ⇒ L = [9, 4, 4, 5]

∴ El algoritmo retorna una lista L = [9, 4, 4, 5].

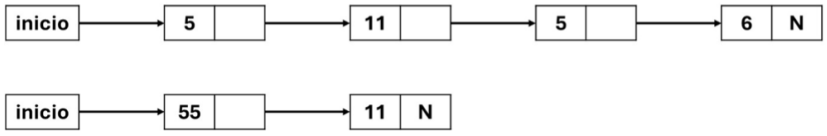
PROBLEMA 2 (20 puntos):

Escriba un algoritmo que reciba dos listas enlazadas simples y que retorne una tercera lista que contenga la unión de ambas listas de entrada, es decir, una lista con los elementos que se encuentran en cualquiera de las dos listas, sin repetición.

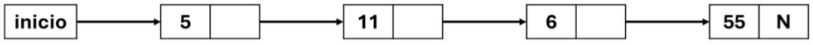
Para su solución considere:

- Las listas de entrada pueden tener elemento repetidos.
- Pude utiliza sin definir las siguientes operaciones del TDA lista:
  - crea_lista(): lista*
  - inserta_final(lista *l, num valor)

Ejemplo de entrada:



Ejemplo de salida:



Sol:

```
REGISTRO nodo:
  NUM dato
  nodo *siguiente

REGISTRO lista:
  nodo *inicio

busca_dato (lista *l, NUM d): NUM
  nodo *ptr ← l->inicio
  WHILE ptr <> NULL DO
    IF ptr->dato = d THEN
      RETURN 1
    ptr ← ptr->siguiente
  RETURN 0
```

```
union_lista(lista *L1, lista* L2): Lista*
    lista *L3 ← crea_lista()
    nodo *ptr ← L1 -> inicio
    WHILE ptr <> NULL DO
        IF !not(busca_dato(L3, ptr -> dato))) THEN
            inserta_final(L3, ptr -> dato)
        ptr ← ptr -> siguiente
    ptr ← L2 -> inicio
    WHILE ptr <> NULL DO
        IF (not(busca_dato(L3, ptr -> dato))) THEN
            inserta_final(L3, ptr -> dato)
        ptr ← ptr -> siguiente

    RETURN L3
```

PROBLEMA 3 (20 puntos):

La operación semisuma de una pila de números enteros retorna una nueva pila donde el primer elemento (tope) es la suma del primer con el segundo elemento de la pila original. El segundo elemento de la nueva pila corresponde a la suma del tercer con el cuarto elemento de la pila original, y así sucesivamente. En caso de que la pila original tenga un número impar de elementos, el último elemento de la nueva pila será el doble del valor del último elemento de la pila original.

Describa en pseudocódigo un algoritmo para la operación semisuma de una pila. Al finalizar la ejecución del algoritmo, la pila de entrada debe contener los mismos elementos y en el mismo orden que al comienzo del algoritmo. En la figura nro. 2 se muestra un ejemplo de la operación semisuma.

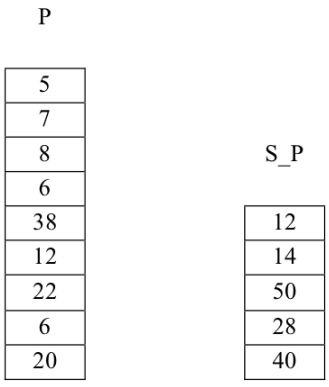


Figura nro. 2: pila P y su pila semisuma S_P

Sol:

```
REGISTRO nodo:
  NUM dato
  nodo * siguiente

REGISTRO pila:
  NUM size
  nodo * tope

nueva_pila ( ) : pila*
  pila *p ← pila_vacia()
  p -> tope ← NULL
  RETURN p

es_pila_vacia (pila* p) : NUM
  IF ( p->tope = NULL ) THEN
    RETURN 1
  RETURN 0
```

```
apilar (pila *p, NUM valor):
    nodo *n ← nodo_vacio()
    n -> dato ← valor
    n -> siguiente ← p -> tope
    p -> tope ← n
    p -> size ← p -> size + 1

desapilar ( pila * p )
    IF (es_pila_vacia (p) = 0) THEN
        nodo * ptr_aux ← p -> tope
        p -> tope ← p -> tope -> siguiente
        p -> size ← p -> size - 1
        liberar (ptr_aux)

tope (pila *p) : nodo*
    IF (es_pila_vacia(p) = 0) THEN
        RETURN p -> tope
    RETURN NULL

semisuma (pila *p): pila*
    pila *pila_resultado ← nueva_pila()
    pila *pila_aux ← nueva_pila()
    nodo * ptr ← p -> tope
    WHILE ptr <> NULL DO
        NUM suma ← 0
        IF (ptr -> siguiente <> NULL) THEN
            suma ← ptr -> dato + ptr -> siguiente -> dato
            ptr ← ptr -> siguiente -> siguiente
        ELSE
            suma ← ptr -> dato * 2
            ptr ← NULL
        apilar (pila_aux, suma)
    WHILE (not(es_pila_vacia(pila_aux)) DO
        apilar (pila_resultado, tope(pila_aux) -> dato)
        desapilar (pila_aux)
    Liberar (pila_aux)
    RETURN pila_resultado
```

PROBLEMA 4 (20 puntos):

En un grafo simple, dos aristas son adyacentes si comparten el mismo vértice. En el grafo de la figura nro. 3 las aristas (2, 5) y (2, 3) son adyacentes.

Escriba en pseudocódigo un algoritmo que reciba como entrada un grafo y una arista, y retorne una lista con las aristas adyacentes a la arista de entrada. Por ejemplo, para el grafo de la figura nro. 3 las aristas adyacentes de (3, 4) son (2, 3) y (4, 5).

Para su solución considere:

- Indique explícitamente la representación utilizada para el grafo. En caso de no indicarla, la pregunta será evaluada con 0 punto.
- Se dispone de las operaciones:
  - crea_lista_adyacentes(): lista*
    - crea una lista de aristas adyacentes vacía.
  - inserta_arista(lista *L, num u, num v)
    - inserta la arista (u, v) en la lista L.

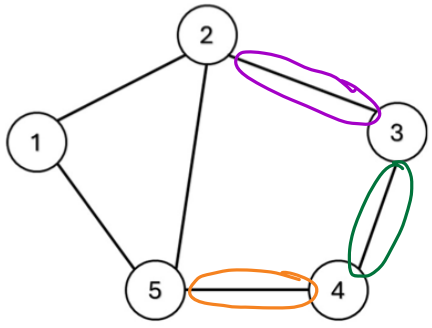


Figura nro. 3: grafo simple.

Sol:

Usando una representación de Matriz de Adyacencia

```
REGISTRO Grafo
   NUM  c_vertices
   NUM  A[c_vertices][c_vertices]

obtiene_aristas_adyacentes (Grafo * G, NUM n1, NUM n2) : lista*
   Lista *L1 ← crea_lista_adyacentes()
   FOR i ← 0 to G -> c_vertices STEP +1
      IF ((G -> A[n1-1][i] = 1) AND (i+1 <> n2)) THEN
         inserta_arista (L1, n1, i+1)
      IF ((G -> A[n2-1][i] = 1) AND (i+1 <> n1)) THEN
         inserta_arista (L1, n2, i+1)
   RETURN L1
```

Representando el grafo en matriz de adyacencia

	1	2	3	4	5
1		1			1
2	1		1		1
→ 3		1		1	
→ 4			1		1
5	1	1		1	

Cumple Condiciones

Arista (3,4)

obtiene_aristas_adyacentes(G, 3, 4)

L1 = [3, 2, 4, 5]