Universidad Interamericana de Panamá

Facultad de Ingeniería, arquitectura y diseño

Escuela de Ingeniería y Sistemas.

Materia: Estructura de Datos II — Cuatrimestre: IIQ_2019
Código: 301-00040 — Facilitador: Leonardo Esqueda
Nombre completo: Yanelis Ayala — ID: 4-754-1723
Carrera: Ingenieria Industrial y de Sistemas

Parcial #1

Instrucciones: La prueba parcial está enfocada en dos (2) pates: Teórico y Práctico; constando con 13 problemas. Lea detenidamente antes de responder.

A) Teoría:

1. **¿Qué son las Estructuras de Datos? 5ptos**
   R. Las estructuras de datos en programación son diferentes formas de organizar información para manipular, buscar e insertar estos datos de manera eficiente.

2. **¿De dónde vienen las estructuras de datos? 5ptos**
   R. Estructura de datos vienen de la ciencias de la computación, para que puedan ser utilizados de manera eficiente. Las estructuras de datos se basan generalmente en la capacidad de un ordenador para recuperar y almacenar datos en cualquier lugar de su memoria.

3. **¿Mencione las dos (2) ramas de la estructura de datos interna? 5ptos**
   R. Las ramas de la estructura de datos interna con Estàticas y Dinámicas

4. ¿**Cómo interactúan las estructuras de datos internas y externas? 5ptos**
   R. Datos internos : son los que residen en la memoria principal del ordenador.
   Datos externos : residen en un soporte de almacenamiento externo.

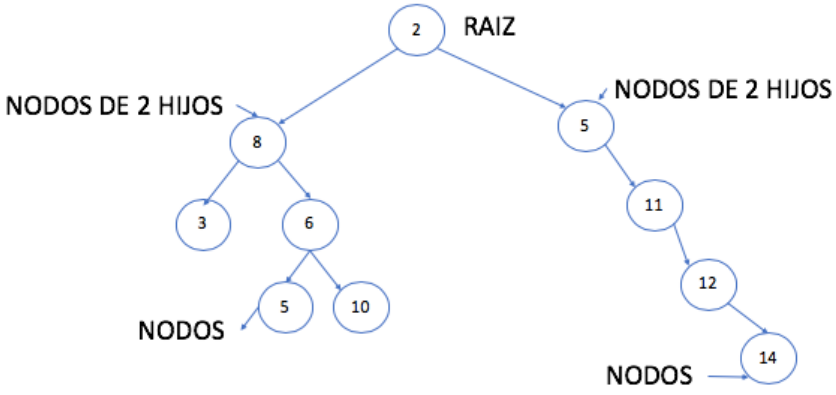
5. ¿**Cómo funcionan los árboles Binarios? 5ptos**
   R. Un árbol binario es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre "binario"). Si algún hijo tiene como referencia a

null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. Los nombres dados para un estilo particular de recorrido vienen de la posición del elemento de raíz con respecto a los nodos izquierdo y derecho. Imagine que los nodos izquierdo y derecho son constantes en espacio, entonces el nodo raíz pudiera colocarse a la izquierda del nodo izquierdo (pre-orden), entre el nodo izquierdo y derecho (in-orden), o a la derecha del nodo derecho (post-orden).

6. **Describa un caso en donde usted podría realizar y utilizar un árbol binario. 5ptos**
R. Sirven para encriptar archivos, o para crear compiladores.
Los árboles binarios se utilizan para almacenar expresiones aritméticas en memoria, esencialmente en compiladores de lenguajes de programación. Una expresión es una secuencia de tokens (componentes de léxicos que siguen unas reglas establecidas). Un token puede ser un operando o bien un operador.

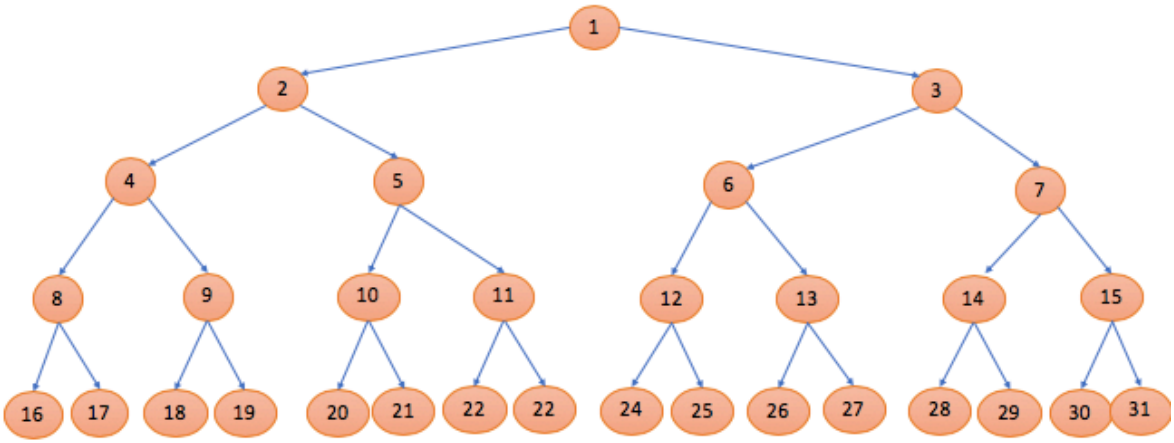
7**. Mencione y explique 5 propiedades de los métodos de ordenamiento de los arboles binarios. 5ptos**
R. 1. Tienen un nodo Raiz.
3. Todos los nodos excepto la raiz , tienen una sola entrada.
4. Tiene una ruta unica de nodo raiz.
5. Apunta como maximo a 2 elementos

8. **Distribuya los nodos -5,2,-11,4,-13,5,3,-14,1,6,10,-12,8 en un árbol binario y determine ¿Cuáles son los nodos hoja? ¿Cuáles son nodos de 2 hijos? 5ptos**
   R.



9. **Realice el recorrido preorden, inorden y postorden de los árboles siguientes: 5ptos**

A.



Recorrido de Preorden

1,2,4,8,16,17,9,18,19,5,10,20,21,11,22,23,3,6,12,24,25,13,26,27,7,14,28,29,15,30,31.

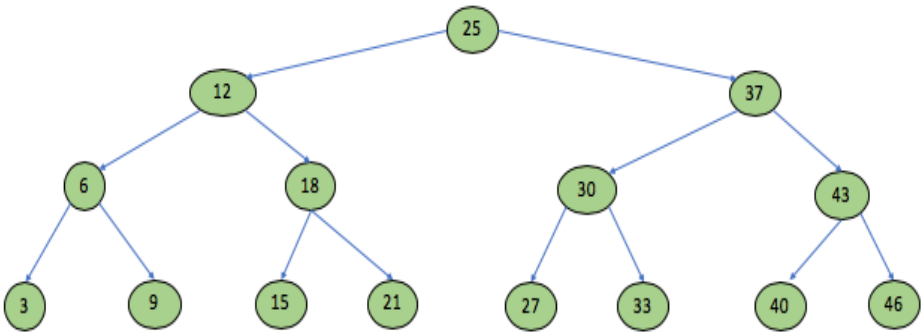
Recorrido postorden:

16,17,8,18,19,9,4,20,21,10,22,23,11,5,2,24,25,12,26,27,13,6,28,29,14,30,31,15,7,3,1.

Recorrido inorden:

16,8,17,4,18,9,19,2,20,10,21,5,22,11,23,1,24,12,25,6,26,13,27,3,28,14,29,7,30,15,31

B.



Recorrido preorden
25,12,6,3,9,18,15,21,37,30,27,33,43,40,46

Recorrido postorden
3,9,6,15,21,18,12,27,33,30,40,46,43,37,25

Recorrido inorden
3,6,9,12,15,18,21,25,27,30,33,37,40,43,46

B) Práctica:

1. Realice un algoritmo lógico el cual explique y demuestre el proceso de inserción de los elementos del problema teórico número dos (2) en un árbol binario. 15ptos

```
class Node(object):

    def __init__(self, value):

      self.value = value

      self.left = None

      self.right = None


class BinaryTree(object):

  def __init__(self, root):

    self.root = Node(root)


  def insert(self, data):

    print("metodo para insertar un valor en un arbol binario: ", data)

    print("verificando si el arbol tiene un root...")

    if self.root is None:

      print("arbol no tiene root, valor de root es: ", data)

      self.root = Node(data)

    else:

      print("arbol si tiene root")

      self._insert(data, self.root)


  def _insert(self, data, current_node):
```

```
    if data < current_node.value:
      print(f"verificando nodo izquierdo ya que {data} < {current_node.value}")
      if current_node.left is None:
        print("se ha insertado en el lado izquierdo del arbol el valor: ", data)
        current_node.left = Node(data)
      else:
        self._insert(data, current_node.left)
    elif data > current_node.value:
      print(f"verificando nodo derecho ya que {data} > {current_node.value}")
      if current_node.right is None:
        print("se ha insertado en el lado derecho del arbol el valor: ", data)
        current_node.right = Node(data)
      else:
        self._insert(data, current_node.right)
    else:
      print("el valor ya se encuentra en el arbol")



t = BinaryTree(1)
t.insert(2)
t.insert(10)
t.insert(8)
t.insert(5)
```

2. Desarrolle un programa en el lenguaje C++ o python en donde cargue por defecto los elementos del árbol: 9 – A y realice los recorridos: preorden, inorden y postorden 15ptos

```
class Node(object):
    def __init__(self, value):
      self.value = value
      self.left = None
      self.right = None


class BinaryTree(object):
  def __init__(self, root):
    self.root = Node(root)


  def print_tree(self, traversal_type):
    if traversal_type == "preorder":
      return self.preorder_print(tree.root, "")
    if traversal_type == "inorder":
      return self.inorder_print(tree.root, "")
    if traversal_type == "postorder":
      return self.postorder_print(tree.root, "")
    else:
      print("No soportado")


  def preorder_print(self, start, traversal):
    if start:
```

```
    traversal += (str(start.value) + "-")
    traversal = self.preorder_print(start.left, traversal)
    traversal = self.preorder_print(start.right, traversal)
  return traversal

def inorder_print(self, start, traversal):
  if start:
    traversal  = self.inorder_print(start.left, traversal)
    traversal += (str(start.value) + "-")
    traversal = self.inorder_print(start.right, traversal)
  return traversal

def postorder_print(self, start, traversal):
  if start:
    traversal  = self.inorder_print(start.left, traversal)
    traversal = self.inorder_print(start.right, traversal)
    traversal += (str(start.value) + "-")
  return traversal

def populate_tree(self, value, node):
  if self.root is None:
    self.root = Node(value)
  else:
```

```
      left_value = value + value;
      right_value = left_value + 1
      # print(left_value)
      if left_value > 0 and left_value < 31:
        print("insertando valor: ", left_value)
        print("insertando valor: ", right_value)
        node.left = Node(left_value)
        node.right = Node(right_value)
        self.populate_tree(left_value, node.left)
        self.populate_tree(right_value, node.right)
      # else:
      #   # print('finish populating tree'


tree = BinaryTree(1)
tree.populate_tree(1, tree.root)


# populate_tree()
print("impresion en preorder")
print(tree.print_tree("preorder"))
print("impresion en inorder")
print(tree.print_tree("inorder"))
print("impresion en postorder")
print(tree.print_tree("postorder"))
```

3. Las Torres de Hanoi es un juego matemático que consiste en tres varillas verticales y un número indeterminado de discos que determinarán la complejidad de la solución. No hay dos discos iguales, están colocados de mayor a menor en una varilla ascendentemente, y no se puede colocar ningún disco mayor sobre uno menor a él en ningún momento. El juego consiste en pasar todos los discos a otra varilla colocados de mayor a menor ascendentemente.

Leyenda: Dios al crear el mundo, colocó tres varillas de diamante con 64 discos en la primera. También creó un monasterio con monjes, los cuales tienen la tarea de resolver esta Torre de Hanoi divina. El día que estos monjes consigan terminar el juego, el mundo acabará. El mínimo número de movimientos que se necesita para resolver este problema es de 264-1. Si los monjes hicieran un movimiento por segundo, los 64 discos estarían en la tercera varilla en poco menos de 585 mil millones de años. Como comparación para ver la magnitud de esta cifra, la Tierra tiene como 5 mil millones de años, y el Universo entre 15 y 20 mil millones de años de antigüedad, sólo una pequeña fracción de esa cifra. Resolución: el problema de las Torres de Hanoi es curioso porque su solución es muy rápida de calcular, pero el número de pasos para resolverlo crece exponencialmente conforme aumenta el número de discos. Para obtener la solución más corta, es necesario mover el disco más pequeño en todos los pasos impares, mientras que en los pasos pares sólo existe un movimiento posible que no lo incluye. El problema se reduce a decidir en cada paso impar a cuál de las dos pilas posibles se desplazará el disco pequeño:

El algoritmo en cuestión depende del número de discos del problema.
Si inicialmente se tiene un número impar de discos, el primer movimiento debe ser colocar el disco más pequeño en la pila destino, y en cada paso impar se le mueve a la siguiente pila a su izquierda (o a la pila destino, si está en la pila origen).
La secuencia será DESTINO, AUXILIAR, ORIGEN, DESTINO, AUXILIAR, ORIGEN, etc.
Si se tiene inicialmente un número par de discos, el primer movimiento debe ser colocar el disco más pequeño en la pila auxiliar, y en cada paso impar se le mueve a la siguiente pila a su derecha (o a la pila origen, si está en la pila destino).
La secuencia será AUXILIAR, DESTINO, ORIGEN, AUXILIAR, DESTINO, ORIGEN, etc. 15 ptos

```
# torre de hanoi

def imprime_movimiento(fr, to):
  print("mueve desde " + str(fr) + " a " + str(to))

# fr: front, to, spare
def torres(n, fr, to, spare):
  if n == 1:
    imprime_movimiento(fr, to)
  else:
    torres(n-1, fr, spare, to)
    torres(1, fr, to, spare)
    torres(n-1, spare, to, fr)


torres(5, "f", "t", "s");
```