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: Carlos Rivera — ID: 8-759-2457
Carrera: Ing. Sistema computacional

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 es una fortma particular de estructura los datos en una computadora para que puedan ser utilizados de manera eficiente, son adecuados para diferente tipo de aplicaciones y para tareas especifica.

2. ¿De dónde vienen las estructuras de datos? 5ptos

R: las estructura de datos vienen de la ciencia de la computacion.

3. ¿Mencione las dos (2) ramas de la estructura de datos interna? 5ptos

R: las dos ramas de la estructura datos interna son:
1.estaticas
2.dinamicas.

4. ¿Cómo interactúan las estructuras de datos internas y externas? 5ptos

R: las estructura de datos interna interactua las diferente tipo lineales no lineales, arreglos matrices y los recorrido de los arboles binaries, y las externa interactua en la informacion de la base datos y los diferente tipos de archivo de una estructura datos.

5. ¿Cómo funcionan los árboles Binarios? 5ptos

R: los arboles binarios son estructura que se componen de una nueva clase de nodo funcionan con un o mas hijo los cuales se dividen en subarbol derecho y subarbol izquierdo cada uno con su caracteristica y dividen en tres recorrido preorden, postorden y inorden.

6. Describa un caso en donde usted podría realizar y utilizar un árbol binario. 5ptos

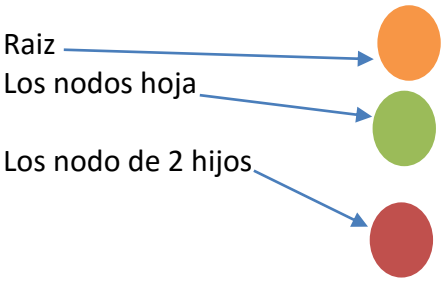
R: podria utilizer un arbol binario en como llegar a un pueblo con diferentes nombre pero con recorrido de postorden de forma (izquierda, derecha y raiz)

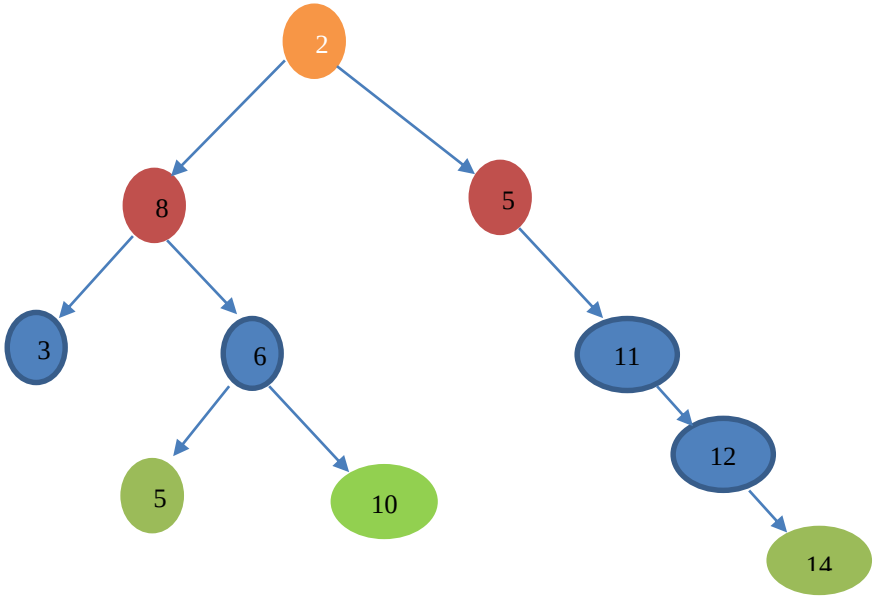
7. Mencione y explique 5 propiedades de los métodos de ordenamiento de los arboles binarios. 5ptos

R: 1.Tiene un nodo al que se le llama raiz del arbol.
2. Todos los nodos, except la raiz, tienen una sola linea de entrada(el nodo raiz no tiene ninguna).
3. Existe una ruta única del nodo raíz a todos los demás nodos del árbol.
4.si hay una ruta, entonces a-b se le denomina hijo de a y es el nodo raíz de un subárbol.
5.Apunta como máximo a otros 2 elementos IMAGEN DE UN ARBOL BINARIA

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:

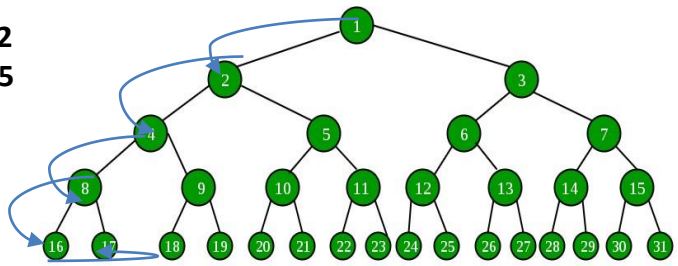




9. Realice el recorrido preorden, inorden y postorden de los árboles siguientes: 5ptos

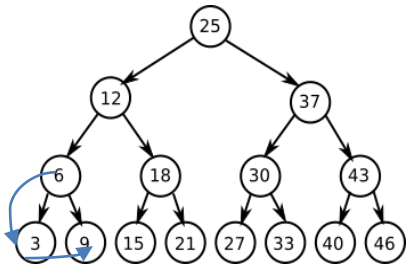
Recorrido de preorden

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

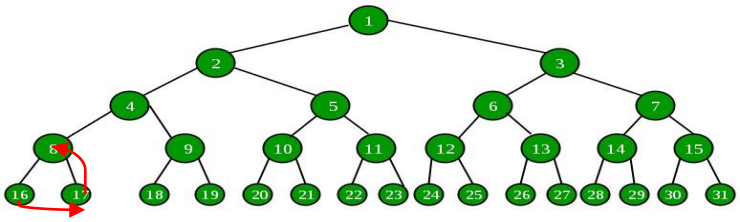


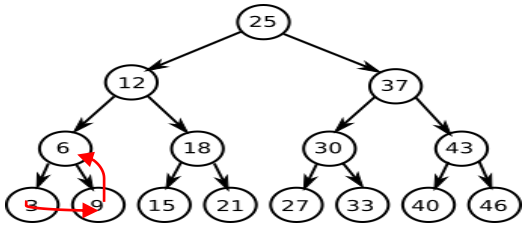
a) Recorrido preorden

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



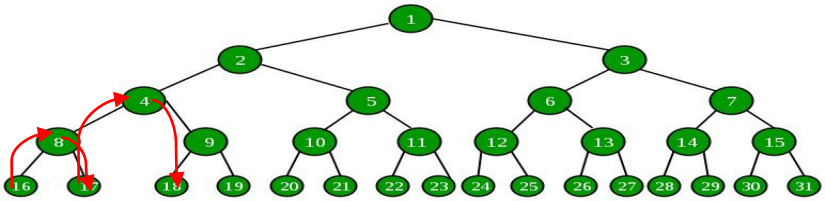
Recorrido postorden:
16, 17, 8, 18, 19, 9, 4, 20, 21, 10, 22, 23, 11, 5, 2, 24, 25, 12, 26
, 27, 13, 6, 28, 29, 14, 30, 31, 15, 7, 3, 1

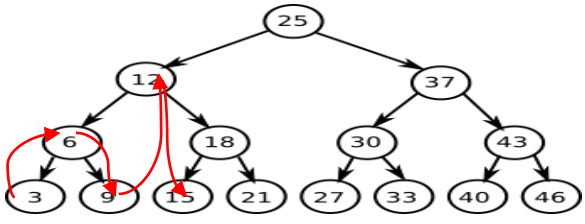




Recorrido postorden
3, 9, 6, 15, 21, 18, 12, 27, 33, 30, 40, 46, 43, 37, 25

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





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

10. Describa un problema del mundo real y que usted considere que pueda ser solucionado utilizando las estructuras de datos. Mencione qué modelo estructura de datos utilizaría, porqué y desarrolle la solución 10ptos

R:

El modelo de estructura datos que utilizaria para solucionar seria los **registro** formado por la unión de varios elementos bajo una misma estructura, en cuanto a la solucion del problema agregar en las maquina recargar saldo para Sistema del metro o metrobus recarga en moneda (martinelli), en los registro del sistema de la maquina seleccionar como recargar efectivo o moneda, este servicio de recarga mejoraria la calidad del usuaria en cuanto las recarga y solucionaria el problema de el dolar escaso en el mercado.

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

R:

```
class Nodo:
   def __init__(self, nombre=None, apellido=None, telefono=None, direccion=None, izq=None, der=None):
      self.nombre = nombre
      self.apellido = apellido
      self.telefono = telefono
      self.direccion = direccion
      self.izq = izq
      self.der = der
   def __str__(self):
      return "%s %s" %(self.nombre, apellido, telefono, direccion)

class aBinarios:
   def __init__(self):
      self.raiz = None

   def agregar(self, elemento):
      if self.raiz == None:
         self.raiz = elemento
      else:
         aux = self.raiz
         padre = none
      while aux !=None:
         padre = aux
         if int(elemen.apellido)>= int(aux.apellido):
            aux =aux.der
         else:
            aux = auz.izq
      if int(elemento.apellido)>= int (padre.apellido):
          padre.der = elemento
      else:
            padre.izq = elemento
def nombre(self, elemento):
            if elemento!= None:
               print(elemento)
               self.nombre(elemento.izq)
               self.nombre(elemento.der)
def apellido(self, elemento):
    if elemento != None:
               print(elemento)
               self.apellido(elemento.izq)
               self.apellido(elemento.der)
               print(elemento)

    def telefono(self, elemento):
              print(elemento)
              self.telefono(elemento.izq)
              print(elemento)
              self.telefono(elemento.der)

    def direccion(self, elemento):
       print(elemento)
       self.direccion(elemento.izq)
       print(elemento)
       self.direccion(elemento.der)

    def getRaiz(self):
            return self.raiz
```

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
   R:

Lenguaje en c++ 111

```
//Descripcion:
Confeccione un programa que haga uso de apuntadores, estructuras dinÃ¡micas, estructuras
selectivas y repetitivas para desarrollar el siguiente MenÃ°.

1. Apuntadores: usado en la logica del arbol.
2. Estructuras dinamicas: struct node para crear el node de arbol .
3. Estruccturas selectivas: if, if-else, switch.
4. Estructuras repetitivas: for, while.

Las estructuras selectivas y repetitivas se les conocen como estructura de control de
flujo (Control Flow).
*/

// librerias a importar
#include <iostream> // para el input
#include <string> // para el uso de texto
#include <limits> // validaciones
#include <stack> // para el uso de pilas
#include <queue>// para el uso de colas
#include <stdio.h.>//para escribir datos en la pantalla
#include <locale.h>//para usa acento en las palabras
#include <stdlib.h>//Sirve para ejecutar subprocesos o comandos del sistema operativo
using namespace std;



void limpiarPantalla(); // limpia la pantalla.
void pilas(int n); // opcion 1, Pilas
void colas(int n); // opcion 2, Colas
void arboles(int n); // opcion 3, Arboles, la Logica de Arboles (NO IMPLEMEMENTADO AUN)
int menuPilasYColas(char title); // MENU Y LOGICA DE PILAS Y COLAS.
void menuInicial(); // funcion recursiva (que se llama a si misma hasta que se presione
"4" para salir de la aplicacion en el menu principal)


// function principal
int main()

{
    setlocale(LC_ALL, "spanish");
    // ir al menu inicial
    menuInicial();
    return 0;
}

// declaracion de la funcion.
void menuInicial()
{
  cout << "\n"
  "------------------------------\n"
  "        Menu Principal\n"
```

```
  "------------------------------\n"
  ;
  cout << "1. Pilas\n";
  cout << "2. Colas\n";
  cout << "3. Árboles Binarios\n";
  cout << "4. Salida\n";
  cout << "\n" << "Escoje una opción: ";
  int menuOption = 0;
  cin >> menuOption;
  while( menuOption > 5 || menuOption < 1){
    cin.clear(); //clear bad input flag
    cin.ignore(numeric_limits<streamsize>::max(), '\n'); //descartar input debe ser int
    cout << "Ingrese una opción valida\n";
    cin >> menuOption;
  }

  bool volverAEjecutar = true;
  switch(menuOption)
  {
        case(1):
        {
            system("CLS");
          pilas(menuOption);
          break;
        }
        case(2):
        {
            system("CLS");
          colas(menuOption);
          break;
        }
        case(3):
        {
            system("CLS");
          arboles(menuOption);
          break;
        }
        case(4):
        {
          // si presiona 4 en el menu, volverAEjecutar es falso
          volverAEjecutar = false;
        }
        default: cout<<"Ha ocurrido un error";
  }
  limpiarPantalla();
  // si volver a ejecutarse es true, ejecutar menuInicial (proceso recursivo)
  if(volverAEjecutar)
  {
    menuInicial();
  }
  cout << "\nHaz elegido salir, hasta luego :)\n";
};



// function para cada estructura de datos;
void limpiarPantalla()
{
  printf("\e[2J\e[H");
};


int menuPilasYColas(string title)
{
  cout << "\n"
  "------------------------------\n"
```

```
"          "<<title<< "\n"
"------------------------------\n"
;
cout << "1. Insertar\n";
cout << "2. Extraer\n";
cout << "3. Visualizar\n";
cout << "4. Salida\n";
cout << "\n" << "Escoje una opción: ";
int menuOption = 0;
cin >> menuOption;

// cola
queue<int> myqueue;


// pila
stack<int> mystack;

// estructura selectiva: while;
// el usuario debe escojer una opcion validad del menu (1 al 4)
// si elige otro valor le pedira que ingrese uno valido.
while( menuOption != 4 ){
  cin.clear(); //clear bad input flag
  cin.ignore(numeric_limits<streamsize>::max(), '\n'); //descartar input, debe ser int
  if(menuOption > 4 || menuOption < 1){
    cout << "Ingrese una opción valida: ";
  }else{
    switch(menuOption)
    {
      case(1):
      {
        cout << "INSERTAR EN " << title << ": ";
        int value;
        cin >> value;
        if(title == "COLA")
        {
          myqueue.push(value);
        }
        else // PILAS
        {
          mystack.push(value);
        };
        break;
      }
      case(2):
      {
        cout << "\nEXTRAER EN " << title << "\n";
        if(title == "COLA")
        {
          myqueue.pop();
        }
        else // PILAS
        {
          mystack.pop();
        };
        break;
      }
      case(3):
      {
        cout << "\nVISUALIZAR EN " << title << "\n\n";
        if(title == "COLA")
        {
          queue<int> tmp_q = myqueue; //copy the original queue to the temporary queue
          while (!tmp_q.empty())
          {
```

```
              cout << "| " << tmp_q.front() << " ";
              tmp_q.pop();
            };
          }
          else // PILAS
          {
            stack<int> tmp_s = mystack;
            while (!tmp_s.empty())
            {
              cout << "| " << tmp_s.top() << " ";
              tmp_s.pop();
            };
          };
          break;
        }
        case(4):
        {  // limpiar pantalla;
          cout << "\n\nHaz elegido salir, hasta luego\n";
          return 0;
        }
        default: cout<<"Ha ocurrido un error\n";
      };
      cout << "\n\nRealize una acción: ";
    }
    cin >> menuOption;
  }

  return menuOption;
};


// opción pila
void pilas(int n)
{
  limpiarPantalla();
  menuPilasYColas("PILA");
};

// opciÃ³n cola
void colas(int n)
{
  limpiarPantalla();
  menuPilasYColas("COLA");
};



/* estructura dinamica para el Arbol */
struct node
{
  int data;
  node* izquierda;
  node* derecha;
};

// cantidad de nodos en un arbol
int countNodes( node *n )
{
  // si esta vacio retorna 0, significa que no tiene nodos.
  if ( !n )
    return 0;
  else {
    int count = 1;   // Empezar contador para saber la cantidad de nodos.
    count += countNodes(n->izquierda);  // Agrega el numero de nodos recursivamente
    count += countNodes(n->derecha);  // Agrega el numero de nodos recursivamente
    return count;  // retorna el total de nodos
```

```
  }
}


// constructor de arbol
node* newNode(int idata)
{
  node* node = new struct node;
  node->data = idata;
  node->izquierda = NULL;
  node->derecha = NULL;
  return node;
}

// recorrido preorder
void iterate_pre(node *n){
    if(!n)
    {
      return;
    };
    // recorrido de forma recursiva
    cout << n->data << " ";
    iterate_pre(n->izquierda);
    iterate_pre(n->derecha);
}

// recorrido postorder
void iterate_post(node *n){
    if(!n)
    {
      // si no hay no imprima nada
      return;
    };

    // recorrido de forma recursiva
    iterate_post(n->izquierda);
    iterate_post(n->derecha);
    cout << n->data << " ";
}

// recorrido inorder
void iterate(node *n){
    if(!n)
    {
      return;
    }
    // recorrido de forma recursiva
    iterate(n->izquierda);
    cout << n->data << " ";
    iterate(n->derecha);
}


void arboles(int n)
{
  limpiarPantalla();
  cout << "\n"
  "----------------------------\n"
  "         ÁRBOLES            \n"
  "----------------------------\n"
  ;
  cout << "1. Crear Árbol\n";
  cout << "2. Recorrido Preorden\n";
  cout << "3. Recorrido Post Orden\n";
  cout << "4. Recorrido en Orden\n";
  cout << "5. Salida\n";
```

```
cout << "\n" << "Escoje una opción: ";
int menuOption = 0;
cin >> menuOption;
while( menuOption != 5 ){
  cin.clear(); //clear bad input flag
  cin.ignore(numeric_limits<streamsize>::max(), '\n'); //descartar input debe ser int
  if(menuOption > 6 || menuOption < 1){
    cout << "Ingrese una opción válida: ";
  }else{
    node *root;
    switch(menuOption)
    {
      case(1):
      {
        int nodeCount = countNodes(root);
        if(nodeCount > 0)
        {
          cout << "Árbol ya inicializado \n";
          break;
        };
        cout << "Inizializando Árbol \n";
        root = newNode(10);
        root->izquierda = newNode(9);
        root->derecha = newNode(8);
        cout << "Listo... \n";
        break;
      }
      case(2):
      {
        int nodeCount = countNodes(root);
        if(nodeCount == 0)
        {
          cout << "Árbol Vacio \n";
          break;
        };
        iterate_pre(root);
        break;
      }
      case(3):
      {
        int nodeCount = countNodes(root);
        if(nodeCount == 0)
        {
          cout << "Árbol Vacio \n";
          break;
        };
        iterate_post(root);
        break;
      }
      case(4):
      {
        int nodeCount = countNodes(root);
        if(nodeCount == 0)
        {
          cout << "Árbol Vacio \n";
          break;
        };
        iterate(root);
        break;
      }
      case(5):
      {
        return;
      }
      default:
```

```
        {
          cout << "Listo... \n";
          return;
        }
      }
      cout << "\n\nRealize una acción: ";
    };
    cin >> menuOption;
  };
};
```

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

R:

```
#origen, auxiliar, destino

def imprime_movimiento(origen, destino):
  print("mueve desde " + str(origen) + " a " + str(destino))

# origen: front, destino, auxiliar
def torres(n, origen, destino, auxiliar):
  if n == 1:
    imprime_movimiento(origen, destino)
  else:
    torres(n-1, origen, auxiliar, destino)
    torres(1, origen, destino, auxiliar)
    torres(n-1, auxiliar, destino, origen)


torres(5, "o", "d", "a");
```