UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES

ESTRUCTURA DE DATOS II

CONTENIDO:

LAB-7. ARBOLES BINARIOS DE BÚSQUEDA. ELIMINAR . . .

PORCENTAJE TERMINADO: 100%

GRUPO: 14

INTEGRANTES	DT	HG	HI	EVAL
Flores Veizaga Eudenia Gandira	1	0	1	100
Garcia Taboada Brayan Albaro	1	0	1	100

//DT, días trabajados

//HG, horas grupo

//HI, horas individual

// Eval

Fecha de presentación : Martes,04 de junio de 2024

Fecha Presentada : Martes, 04 de junio de 2024

Días de Atraso : 0

ACTIVIDAD EN PAREJAS DE MISMO GRUPO.

Eliminar elementos de un Árbol Binario de Búsqueda.

1. **A1.eliminarSup(x) :** Método que elimina el elemento x, del árbol A1. Si el elemento a eliminar es un nodo raíz, buscar el elemento inmediato Superior, para eliminar.
2. **A1.eliminarInf(x):** Método que elimina el elemento x, del árbol A1. Si el elemento a eliminar es un nodo raíz, buscar el elemento inmediato Inferior, para eliminar
3. **A1.eliminarHojas() :** Método que elimina los nodos hoja de árbol A1.
4. **A1.eliminarPares() :** Método que elimina los elementos pares del árbol A1.
5. **A1.eliminar(L1) :** Método que elimina los elementos de la lista L1 que se encuentran en el árbol A1.
6. **A1.eliminarMenor():** Método que elimina el elemento menor del árbol A1.
7. **A1.eliminarMayor():** Método que elimina el elemento mayor del árbol A1.

```
public void eliminar(int x) {
   raiz = eliminar(x, raiz);
}

private Nodo eliminar(int x, Nodo p) {
   if (p == null) {
      return null;
   }
   if (x == p.elem) {
      return eliminarNodo(p);
   }
   if (x < p.elem) {
      p.izq = eliminar(x, p.izq);
   } else {
      p.der = eliminar(x, p.der);
   }
   return p;
}

public Nodo eliminarNodo(Nodo p) {
   if (p.izq == null && p.der == null) {
      return null;
   }
   if (p.izq != null && p.der == null) {
      return p.izq;
   }
   if (p.izq == null && p.der != null) {
      return p.der;
   }
   Nodo q = p.izq;
   while (q.der != null) {
      q = q.der;
   }
```

```
      int y = q.elem;
      eliminar(y);
      p.elem = y;
      return p;
   }
```

//A1.eliminarSup(x) : Método que elimina el elemento x, del árbol A1.Si el elemento a eliminar es un nodo raíz, buscar el elemento inmediato Superior, para eliminar.

```
   public void eliminarSup(int x) {
      raiz = eliminarSup(x, raiz);
   }

   private Nodo eliminarSup(int x, Nodo p) {
      if (p == null) {
         return null;
      }
      if (x == p.elem) {
         return eliminarNodoSup(p);
      }
      if (x < p.elem) {
         p.izq = eliminarSup(x, p.izq);
      } else {
         p.der = eliminarSup(x, p.der);
      }
      return p;
   }

   public Nodo eliminarNodoSup(Nodo p) {
      if (p.izq == null && p.der == null) {
         return null;
      }
      if (p.izq != null && p.der == null) {
         return p.izq;
      }
      if (p.izq == null && p.der != null) {
         return p.der;
      }
      Nodo q = p.izq;
      while (q.der != null) {
         q = q.der;
      }
      int y = q.elem;
      eliminarSup(y);
      p.elem = y;
      return p;
   }
```

//A1.eliminarInf(x): Método que elimina el elemento x, del árbol A1. Si el elemento a eliminar es un nodo raíz, buscar el elemento inmediato Inferior, para eliminar

```
   public void eliminarInf(int x) {
      raiz = eliminarInf(x, raiz);
   }
```

```
    private Nodo eliminarInf(int x, Nodo p) {
        if (p == null) {
            return null;
        }
        if (x == p.elem) {
            return eliminarNodoInf(p);
        }
        if (x < p.elem) {
            p.izq = eliminarInf(x, p.izq);
        } else {
            p.der = eliminarInf(x, p.der);
        }
        return p;
    }

    public Nodo eliminarNodoInf(Nodo p) {
        if (p.izq == null && p.der == null) {
            return null;
        }
        if (p.izq != null && p.der == null) {
            return p.izq;
        }
        if (p.izq == null && p.der != null) {
            return p.der;
        }
        Nodo q = p.izq;
        while (q.der != null) {
            q = q.der;
        }
        int y = q.elem;
        eliminarInf(y);
        p.elem = y;
        return p;
    }
  //A1.eliminarHojas() : Método que elimina los nodos hoja de árbol A1.

    public void eliminarHojas() {
        eliminarHojas(raiz);
    }

    private void eliminarHojas(Nodo p) {

    }
  //A1.eliminarPares() : Método que elimina los elementos pares del árbol A1.

    public void eliminarPares() {
        eliminarPares(raiz);
    }

    private void eliminarPares(Nodo p) {
        if (p == null) {
```

```
            return;
        }
        if (p.elem / 2 == 0) {
            eliminar(p.elem);
        }
        eliminarPares(p.izq);
        eliminarPares(p.der);
    }
//A1.eliminar(L1) : Método que elimina los elementos de la lista L1 que se encuentran en el
árbol A1.
  public void eliminar(LinkedList<Integer> L1) {
      for (Integer ele : L1) {
          raiz=eliminar(ele,raiz);
      }
    }
//A1.eliminarMenor(): Método que elimina el elemento menor del árbol A1.
    public void eliminarMenor() {
        Nodo p=raiz;
        while(p.izq!=null) p=p.izq;
        eliminar(p.elem);
    }

//A1.eliminarMayor(): Método que elimina el elemento mayor del árbol A1.
    public void eliminarMayor() {
        Nodo p=raiz;
        while(p.der!=null) p=p.der;
        eliminar(p.elem);
    }}

class Nodo {

    public Nodo izq;
    public Nodo der;
    public int elem;

    public Nodo(int elem) {
        this.elem = elem;
        izq = der = null;
    }
```