UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES

ESTRUCTURA DE DATOS II

CONTENIDO:

LAB-3. ABB CON LIBRERÍA DE LISTAS

PORCENTAJE TERMINADO: 100%

Grupo 14

Garcia Taboada Brayan Albaro

Fecha de presentación : Jueves , 09 de mayo de 2024

Fecha Presentada : Jueves , 09 de mayo de 2024

Días de Atraso : 0

ÁRBOLES CON LIBRERÍA DE LISTAS.

TRABAJO INDIVIDUAL.

Sea A1, un Árbol Binario de Búsqueda. Implementar los siguientes métodos:

1. **A1.generarElem(n, a, b) :** Método que genera n elementos aleatorios enteros diferentes entre a y b inclusive.

2. **A1.insertar(x) :** Método que inserta el elemento x, en el árbol A1 en su lugar correspondiente.

3. **A1.preOrden() :** Método que muestra los elementos del árbol A1 en preOrden.

4. **A1.inOrden() :** Método que muestra los elementos del árbol A1 en inOrden.

5. **A1.postOrden() :** Método que muestra los elementos del árbol A1 en postOrden.

6. **A1.niveles():** Método que muestra los elementos del árbol A1, por niveles.

7. **A1.desc():** Método que muestra los elementos del árbol A1 de mayor a menor.

8. **A1.seEncuentra(x) :** Métodos lógico que devuelve True, si el elemento x, se encuentra en el árbol A1.

9. **A1.cantidad() :** Método que devuelve la cantidad de nodos del árbol A1.

10. **A1.suma() :** Método que devuelve la suma de los elementos del árbol A1.

11. **A1.menor() :** Método que devuelve el elemento menor del árbol A1.

12. **A1.mayor() :** Método que devuelve el elemento mayor del árbol A1.

13. **A1.preoOrden(L1) :** Método que encuentra en la lista L1, el recorrido de preOrden de los elementos del árbol A1.

14. **A1.inOrden(L1) :** Método que encuentra en la lista L1, el recorrido de inOrden de los elementos del árbol A1.

15. **A1.postOrden(L1) :** Método que encuentra en la lista L1, el recorrido de postOrden de los elementos del árbol A1.

16. A1.niveles(L1) : Método que encuentra en la lista L1, el recorrido por niveles de los elementos del árbol A1.

17. A1.mostrarNivel(): Método que muestra los elementos del árbol y el nivel en el que se encuentran. (Recorrer el árbol en cualquier orden)

18. A1.sumarNivel(L1) : Método que encuentra en la Lista de acumuladores por nivel L1, la suma de los elementos de cada nivel.

COMENTARIOS

La verdad esta Estructura de datos es muy interesante y a la vez la veo muy útil en cierto sentido para algunos casos y los códigos son entendible y no tan complicados de leer.

```
public class Arbol {


   public Nodo raiz;


   public Arbol() {
      this.raiz = null;
   }


   //1.  A1.generarElem(n, a, b) : Método que genera n elementos aleatorios enteros diferentes entre a y b inclusive.
   public void generarElem(int n, int a, int b) {
      for (int i = 0; i < n; i++) {
         insertar((int) Math.floor(a + Math.random() * (b - a)));
      }
   }


   //2.  A1.insertar(x) : Método que inserta el elemento x, en el árbol A1 en su lugar correspondiente.
   public void insertar(int x) {
      raiz = insertar(x, raiz);
   }


   private Nodo insertar(int x, Nodo p) {
```

```
    if (p == null) {
        return new Nodo(x);
    }
    if (x < p.elem) {
        p.izq = insertar(x, p.izq);
    } else {
        p.der = insertar(x, p.der);
    }
    return p;
}
//3.  A1.preOrden() : Método que muestra los elementos del árbol A1 en preOrden.

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

private void preOrden(Nodo p) {
    if (p == null) {
        return;
    }
    System.out.println(p.elem);
    preOrden(p.izq);
    preOrden(p.der);
}

//4.  A1.inOrden() : Método que muestra los elementos del árbol A1 en inOrden.
public void inOrden() {
    inOrden(raiz);

}
```

```
private void inOrden(Nodo p) {
   if (p == null) {
      return;
   }
   inOrden(p.izq);
   System.out.println(p.elem);
   inOrden(p.der);

}
//5.  A1.postOrden() : Método que muestra los elementos del árbol A1 en postOrden.

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

private void postOrden(Nodo p) {
   if (p == null) {
      return;
   }
   postOrden(p.izq);
   postOrden(p.der);
   System.out.println(p.elem);
}

//6.      A1.niveles(): Método que muestra los elementos del árbol A1, por niveles.
public void elementoNivel() {
   elementoNivel(raiz, 1);
}

private void elementoNivel(Nodo p, int nivel) {
   if (p == null) {
```

```
            return;
        }
        elementoNivel(p.izq, nivel + 1);
        System.out.println(p.elem + "\t" + nivel);
        elementoNivel(p.der, nivel + 1);
    }

    //7.  A1.desc(): Método que muestra los elementos del árbol A1 de mayor a menor.
    public void desc() {
        desc(raiz);

    }

    private void desc(Nodo p) {
        if (p == null) {
            return;
        }
        desc(p.der);
        System.out.println(p.elem);
        desc(p.izq);

    }

    //8.  A1.seEncuentra(x) : Métodos lógico que devuelve True, si el elemento x, se encuentra en el árbol A1.
    public boolean seEncuentra(int x) {
        return seEncuentra(x, raiz);
    }

    private boolean seEncuentra(int x, Nodo p) {
        if (p == null) {
```

```
        return false;
    }
    if (x == p.elem) {
        return true;
    }
    if (x < p.elem) {
        return seEncuentra(x, p.izq);
    } else {
        return seEncuentra(x, p.der);
    }

}

//9.  A1.cantidad() : Método que devuelve la cantidad de nodos del árbol A1.
public int cantidad() {
    return cantidad(raiz);
}

private int cantidad(Nodo p) {
    return p == null ? 0 : cantidad(p.izq) + cantidad(p.der) + 1;
}

//10.  A1.suma() : Método que devuelve la suma de los elementos del árbol A1.
public int suma() {
    return suma(raiz);
}

private int suma(Nodo p) {
    if (p == null) {
        return 0;
    } else {
```

```
        return suma(p.izq) + suma(p.der) + p.elem;
    }
}

//11.  A1.menor() : Método que devuelve el elemento menor del árbol A1.
public int menor() {
    return menor(raiz);
}

private int menor(Nodo p) {
    if (p.izq == null) {
        return p.elem;
    } else {
        return menor(p.izq);
    }
}

//12.   A1.mayor() : Método que devuelve el elemento mayor del árbol A1.
public int mayor() {
    return mayor(raiz);
}

private int mayor(Nodo p) {
    if (p.der == null) {
        return p.elem;
    } else {
        return menor(p.der);
    }
}
```

//13. A1.preoOrden(L1) : Método que encuentra en la lista L1, el recorrido de preOrden de los elementos del árbol A1.

```
public void preOrden(LinkedList<Integer> L1) {
    preOrden(raiz, L1);
}

private void preOrden(Nodo p, LinkedList<Integer> L1) {
    if (p == null) {
        return;
    }
    L1.add(p.elem);
    preOrden(p.izq, L1);
    preOrden(p.der, L1);
}
```

//14. A1.inOrden(L1) : Método que encuentra en la lista L1, el recorrido de inOrden de los elementos del árbol A1.

```
public void inOrden(LinkedList<Integer> L1) {
    inOrden(raiz, L1);

}

private void inOrden(Nodo p, LinkedList<Integer> L1) {
    if (p == null) {
        return;
    }
    inOrden(p.izq, L1);
    L1.add(p.elem);
    inOrden(p.der, L1);

}
```

//15. A1.postOrden(L1) : Método que encuentra en la lista L1, el recorrido de postOrden de los elementos del árbol A1.

```
public void postOrden(LinkedList<Integer> L1) {
    postOrden(raiz, L1);
}

private void postOrden(Nodo p, LinkedList<Integer> L1) {
    if (p == null) {
        return;
    }
    postOrden(p.izq, L1);
    postOrden(p.der, L1);
    L1.add(p.elem);
}
```

//16. A1.niveles(L1) : Método que encuentra en la lista L1, el recorrido por niveles de los elementos del árbol A1.

```
public void niveles(LinkedList<Integer> L) {
    LinkedList<Nodo> L1 = new LinkedList();
    if (raiz == null) {
        return;
    }
    L1.add(raiz);
    while (!L1.isEmpty()) {
        Nodo p = L1.getFirst();
        L.add(p.elem);
        if (p.izq != null) {
            L1.add(p.izq);
        }
        if (p.der != null) {
            L1.add(p.der);
```

```
        }
        L1.removeFirst();

    }
}
```

//17. A1.mostrarNivel(): Método que muestra los elementos del árbol y el nivel en el que se encuentran. (Recorrer el árbol en cualquier orden)

```
public void mostrarNivel() {
    elementoNivel(raiz, 1);
}

private void mostrarNivel(Nodo p, int nivel) {
    if (p == null) {
        return;
    }
    elementoNivel(p.izq, nivel + 1);
    System.out.println(p.elem + "\t" + nivel);
    elementoNivel(p.der, nivel + 1);
}
```

//18. A1.sumarNivel(L1) : Método que encuentra en la Lista de acumuladores por nivel L1, la suma de los elementos de cada nivel.

```
public void sumarNivel() {
    int max = cantidad();
    ArrayList<Integer> L1 = new ArrayList(max);
    for (int i = 0; i < max; i++) {
        L1.add(0);
    }
    sumarNivel(raiz, 0, L1);
    int i = 0;
    while (L1.get(i) != 0) {
```

```
        System.out.println(i + 1 + "\t" + L1.get(i));
        i++;
    }
}

public void sumarNivel(Nodo p, int nivel, ArrayList<Integer> L1) {
    if (p == null) {
        return;
    }
    L1.set(nivel, L1.get(nivel) + p.elem);
    sumarNivel(p.izq, nivel + 1, L1);
    sumarNivel(p.der, nivel + 1, L1);
}
```