**PRIMER PARCIAL**

**INF310 SX**‒ **Estructuras de Datos II. Gestión 1-2019.**

**Subgrupo: S-Z**

Árbol Binario

1. En la class Arbol (desordenado pero sin duplicados), escriba el procedimiento

**public void delHojaCercana(int x)**

el cual haga lo siguiente:

* Si x no existe, éste método no realiza ninguna acción (“no pasa nada”).
* Si x es una hoja, x es eliminada.
* Si x no es una hoja, aquella hoja cuyo valor sea el más cercano a x, será eliminada del árbol.

(Si existen 2 o más hojas con la misma cercanía, escoja al azar una de ellas)

**Por ejemplo**: (En el gráfico, no se dibujan los punteros null)

**Árbol A**

|  |  |
| --- | --- |
| **A.delHojaCercana(100);** | Como el 100 **no** existe en A, el árbol queda igual (“no pasa nada”). |
| **A.delHojaCercana(20);**  Como el 20 es una hoja, ésta es eliminada del Árbol. | **Árbol A** |
| **A.delHojaCercana(60);**  Las hojas son 35, 80, 5, 50 y 25. Escogemos la *más cercana* a **60**, haciendo la diferencia en valor absoluto:  Math.abs(**60**-35) =25, Math.abs(**60**-80) =20  Math.abs(**60**-5) = 55, Math.abs(**60**-**50**) =**10**  Math.abs(**60**-25) =35  La diferencia más pequeña es **10** y le corresponde a la hoja 50. Entonces, **50** será eliminada | **Árbol A** |

|  |  |
| --- | --- |
| **A.delHojaCercana(30);**  Las hojas son 35, 80, 5, y 25. Escogemos la *más cercana* a **30**, haciendo la diferencia en valor absoluto:  Math.abs(**30**-**35**) = **5**, Math.abs(**30**-80) =50  Math.abs(**30**-5) = 25 Math.abs(**30**-**25**) = **5**  La diferencia más pequeña es **5**, pero hay dos hojas con esa diferencia: Escogemos al azar una de ellas, para ser eliminada (Aquí escogimos al **35**) | **Árbol A** |
| **A.delHojaCercana(70);**  Como el 70 es una hoja, ésta es eliminada del Árbol. | **Árbol A** |
| **A.delHojaCercana(40);**  Las hojas son 80, 5 y 25. Escogemos la *más cercana* a **40**, haciendo la diferencia en valor absoluto:  Math.abs(**40**-80) =40 Math.abs(**40**-5) = 35,  Math.abs(**40**-**25**) =**15**  La diferencia más pequeña es **15** y le corresponde a la hoja 25. Entonces, 2**5** será eliminada | **Árbol A** |

**Listas**

2. Implementar una class Lista, la cual almacena pares (Data, Peso). La Lista no admite duplicados y se mantiene ordenada, según el siguiente criterio:

* Si Data1 < Data2, entonces, (Data1, Peso1) < (Data2, Peso2)
* Si Data1=Data2, entonces (Data1, Peso1) < (Data2, Peso2), si Peso1 < Peso2

**public Lista()**

//**Constructor**. La Lista está vacía.

**public void add(int data, int peso)**

//Adiciona el par (data, peso) a la Lista, manteniéndola ordenada. Si el par (data, peso), ya existe en la Lista, “no pasa nada”.

**Por ejemplo (en el main):**

Lista P = new Lista(); //P=(vacía)

P.add(20, 5); // P=[**(20,5)**] El par (20, 5) se inserta a la Lista.

P.add(10, 8); // P=[**(10, 8)**, (20,5)] Como el 10 < 20, el par (10,8) se inserta antes del (20,5).

P.add(30, 1); // P=[(10, 8), (20,5), **(30, 1)**] Como el 20 < 30, el par (30, 1) se inserta después del (20,5).

P.add(20, 1); // P=[(10, 8), **(20, 1)**, (20,5), (30, 1)] Puesto que el data 20 ya existe, usamos los pesos para saber cuál es menor:

// Como el peso **1** < peso 5, entonces (20, 1) < (20, 5).

P.add(10, 8); // P=[(10, 8), (20, 1), (20,5), (30, 1)] La Lista P se mantiene igual, porque el par (10,8) ya está en la Lista.

P.add(8, 10); // P=[**(8, 10)**, (10, 8), (20, 1), (20,5), (30, 1)] Como el 8 < 10, el par (8, 10) se inserta antes del (10, 8).

Tome en cuenta que el Nodo de su Lista (ver código fuente) tiene los siguientes campos:

|  |  |  |
| --- | --- | --- |
| **Data** | **Peso** | **Link** |