Tarea 3

Trabajo a entregar por GRUPO vía EDUCA. Indique claramente los integrantes de su grupo. El código debe poder probarse sin usar ningún IDE desde la linea de comandos. No incluya ninguna clase en package.

Un **Árbol Binario de Búsqueda o BST** (Binary Search Tree): *Es un árbol binario donde cada nodo tiene una clave comparable (Comparable) y datos asociados, que satisface la restricción que la clave en cualquier nodo es mayor que las claves de todos los nodos que se encuentran en el subárbol izquierdo y es menor que todas las claves que se encuentran en el subárbol derecho.*

Dado un nodo x en el BST. Si y es un nodo en el súbarbol izq. de x, entonces clave(y) <= clave(x). Si y es un nodo en el subarbol derecho de x, entonces clave(x) <= clave(y).

En esta clase veremos de forma práctica una posible implementación de un BST como una estructura de datos sencilla que nos permite buscar eficientemente una información en memoria.
Analizaremos su rendimiento en cuanto a las operaciones tradicionales (buscar, insertar, borrar entre otras) y los condicionamientos de su utilización.

Cuando se tiene que agregar elementos cuya llegada a la estructura de datos es aleatoria, puede considerarse al BST como una estructura ideal por su sencillez y eficiencia. El tiempo para la búsqueda, inserción o borrado se encuentra en relación a la altura del árbol. Una altura en el orden logarítmico del número de elementos nos asegura un buen rendimiento.

Por otra parte, si tenemos una entrada de datos ordenada o semiordenada (no randómico), esta estructura puede degenerar en una simple lista enlazada y así en un tiempo lineal afectando el rendimiento de acceso a la estructura.

Considere el archivo `BST.java` que es una implementación sencilla de BST (El mismo se encuentra en Educa).

Parte I - Ejercicios prácticos(56p)

Ejercicio 1 (10p)

Resuelva los siguientes ejercicios (en un solo fuente)

a) Convierta su código a Generic, de forma que ahora pueda asegurar que los datos guardados en el BST sean de un solo tipo que implementa la interface Comparable. Entonces el construir un BST debe lucir de la siguiente forma (por ejemplo utilizando String, aquí se muestra el main de BST del código entregado)

```
public static void main ( String [] args ) {
    BST<String> t = new BST<String>();
    String [] A = { "hola", "amor", "roma", "barco", "mesa", "musica", "zapato", "amigo"
};
    for ( int k=0; k < A.length; k++ )
        t.agregar( A[k]);
    t.imprimir();
    String k = t.buscar("mesa");
    if ( k != null )
System.out.println("Si existe!!" + k);
}
El ejemplo produciría la siguiente salida:
amigo amor barco hola mesa musica roma zapato
Si existe!!mesa
```

b) Realice los ajustes de forma que ahora pueda utilizarse el *for-each* o *for extendido* de java (implementación de un iterador) Por ejemplo:

```
BST<String> bsts = new BST<String>();
...
for ( String s : bsts)
     System.out.printf("%s",s);
```

c) Agregue un método que retorne en un Iterable los datos que se encuentran entre dos claves min y may. Por ejemplo debería de funcionar así:

```
BST<Integer> bi = new BST<Integer>();
...
Iterable<Integer> it  = bi.claves(min, may) ;      /* Retorna las claves entre min y may */
for ( int i : it )                                 /* Utilizamos el iterador para imprimirlos */
      System.out.printf("%d ", i);
```

Si las claves no son válidas (claves negativas, min>may, u otros), o bien no se encuentran claves en el rango, el método debe lanzar una excepción.

Rúbrica	Puntaje
Realización de modificaciones para uso de generics en a)	2
Construcción correcta de bst al menos con datos de prueba de tipo Float, Double, Integer, String, proveídos por el alumno en a)	2
Comentarios Aclaratorios en a)	1
Pruebas correctas al menos con datos de prueba de tipo Float, Double, Integer, String, proveídos por el alumno en b)	2
Lanza la excepción solicitada en b)	1
Comentarios aclaratorios en b)	1

Ejercicio 2 (5p)

Implemente un método que reciba un BST y un entero positivo *k*, y retorne el *k*-ésimo elemento más pequeño en el BST. Por ejemplo, en la Figura a de abajo, el tercer elemento más pequeño es 7, y el sexto elemento más pequeño es 15. Si *k* es mayor al número de nodos en el BST, entonces retornar un mensaje de error.

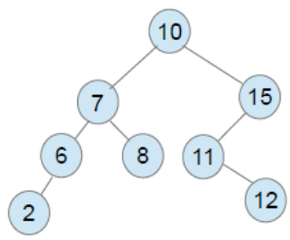


Figura a

Rúbrica	Puntaje
Implementación correcta para al menos dos casos de prueba con al menos 10 nodos proveídos por el alumno	2
Implementación correcta para tipos de datos Integer, Float, String, con datos de prueba proveídos por el alumno	2
Comentarios Aclaratorios	1

Ejercicio 3 (6p)

Dado un BST T y un vértice x de T, definimos una función ACC-GREATER(x) como

$$\text{ACC-GREATER}(x) = x.key + \sum_{y,key>x.key} y.key.$$

Esto es, ACC-GREATER(x) es la suma acumulada de todas las claves que son mayores a x más la propia clave de x.

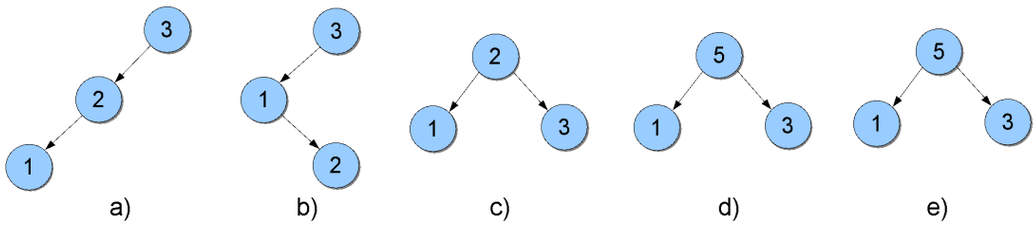
Escribe un programa que tome un BST T como entrada y retorne otro BST T' donde todas las claves de T' está actualizadas utilizando la función ACC-GREATER aplicada a los nodos de T.

Rúbrica	Puntaje
Implementación correcta de un método de recorrido de un BST.	2
Implementación correcta de la función ACC-GREATER para cada nodo de un BST.	2
Implementación correcta del árbol de salida *T'*.	1
Al menos 3 ejemplos de prueba proveídos por el grupo.	1

Ejercicio 4 (10p)

Implemente en BST un **método estático** que dado dos BST retorne un enumerado que indique:

1) Si son iguales en forma y en valores (son idénticos). Asegurar que no sean la misma referencia.
2) Si son diferentes solo en forma (los elementos son iguales pero la ubicación es diferente) pero iguales en valores.
3) Si son diferentes en valores.



Ejemplos: *comparar(a,b)* es el caso 2, *comparar(b,c)* es el caso 2, *comparar(c,d)* es el caso 3, *comparar(c,c)* es un error, *comparar(d,e)* es el caso 1.

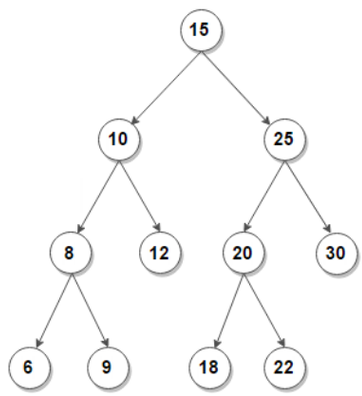
Indique el costo asintótico en términos de *O* de su implementación. En este caso se tiene dos tamaños a considerar, el tamaño del primer BST y el tamaño del segundo BST. Por tanto, la respuesta del tiempo debe estar basada en estos dos parámetros.

Rúbrica	**Puntaje**
Implementación correcta del método solicitado en 1)	3
Implementación correcta del método solicitado en 2)	3
Implementación correcta del método solicitado en 3)	2
Se indica coste asintótico	1
provisión de datos de prueba para los ítem 1) 2) 3)	1

Ejercicio 5 (5p)

Implemente una subrutina que toma como entrada un BST *T* y un rango entero *[a,b]* y que retorna el número de subárboles de T (diferentes a *T*) cuyos nodos tienen claves en el rango *[a,b]*.

Por ejemplo, el árbol de abajo tiene 6 subárboles en el rango [5,20]. ¿Puedes ver cuáles son esos subárboles?

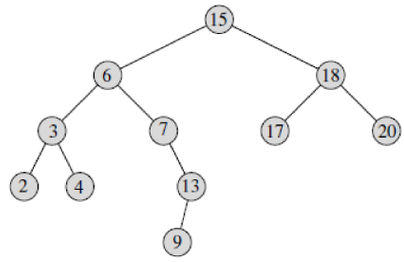


Rúbrica	**Puntaje**
Implementación correcta del método de recorrido del BST.	2
La subrutina implementada retorna el conteo correcto de subárboles.	2
Al menos 3 ejemplos son proveídos por el grupo.	1

Ejercicio 6 (15p)

Amplíe la clase BST entregada por el profesor agregándole los siguientes métodos (asuma que las claves no se repiten). Haga

que el BST sea genérico (uso de *Generics*)



a) **agregar** agregar un nodo del árbol (NO recursivo)
b) **eliminar** elimina un nodo del árbol (NO recursivo)
c) **esLleno** retorna *True* si es un el árbol lleno
d) **esCompleto** retorna *True* si el árbol está completo
e) **cntHojas** retorna la cantidad de nodos "hojas" el árbol
   En el ejemplo del BST de la derecha, se retornaría 5
f) **Sucesor** retorna el menor valor siguiente a la clave analizada
   En el ejemplo del BST de la derecha, el sucesor de 3 es 4.
   El sucesor de 13 es 15.
g) **Predecesor** retorna el mayor valor previo a la clave analizada
   En el ejemplo del BST de la derecha, el predecesor de 13 es 9, el predecesor de 17 es 15.

Por cada método indique su costo temporal en términos de "O grande". Indique también el *costo espacial* para cada método. Proporcione casos de prueba para verificar las implementaciones.

Rúbrica	**Puntaje**
Implementación correcta de a)	2
Implementación correcta de b)	2
Implementación correcta de c)	2
Implementación correcta de d)	2
Implementación correcta de e)	2
Implementación correcta de f)	2
Implementación correcta de g)	2
Indica costo temporal y espacial para cada método	1

Ejercicio 7 (5p)

Nosotros podemos ordenar un arreglo de *n* elementos construyendo un BST conteniendo estos números (utilizando una sucesión de inserciones uno a uno en el árbol) y luego haciendo un recorrido simétrico (in-orden) sobre el mismo.

¿Cuál es el costo asintótico en el mejor y peor caso de este algoritmo de ordenación? Fundamente su respuesta.

¿Cómo podría evitar o minimizar el peor caso?
Implemente esta idea en Java y compruebe empíricamente el rendimiento de su algoritmo ordenando arreglos numéricos de tamaños definidos por la línea de comandos. Para ello incluya la tabla de rendimiento similar a la solicitada en el ejercicio II.1 de la Tarea 2-U2.

Por ejemplo: `$ java Ordenar_con_BST 1000 2000 3000 4000 5000 60000`

Rúbrica	**Puntaje**
Respuesta correcta para el costo asintotico	1
Respuesta correcta para el peor caso	1
Implementación correcta para la prueba empirica	2
Comentarios aclaratorios	1