

Segundo Parcial de Estructura de Datos y Algoritmos

Ejer 1	Ejer 2	Nota
/6	/4	/10

Duración: 2 horas 10 minutos

Condición Mínima de Aprobación: Sumar no menos de cuatro puntos

Muy Importante

Al terminar el examen deberían subir los siguientes 2 archivos, según lo explicado en los ejercicios:

- 1) Solo las siguientes clases Java: **FiboTree.java** y **SimpleOrDefault.java** según lo pedido. **Cualquier código auxiliar debe estar dentro de esos 2 archivos.**
- 2) Todos los códigos que les subimos para este examen pertenecen al **package core, no cambiar eso. Colocar solo sus clases implementadas (FiboTree y SimpleOrDefault) con el nombre del usuario de Uds. en campus antes de subirlo.**

Ej:

package lgomez;

Además en la clase GraphFactory, modifique la línea import según corresponda:

Ej:

import dario.SimpleOrDefault;

cambiarlo a:

import lgomez.SimpleOrDefault;

Ejercicio 1

Bajar de Campus el proyecto maven **Ejer1.zip**. Utilizar ese código

La clase **FiboTree** es un árbol que se construye con el objetivo de generar un binary tree que sea “**en su forma**” como el peor caso de AVL, o sea, **Fibonacci Tree**.

Sin embargo, porque tiene **una forma tan especial** no se acepta que el usuario inserte o borre datos.

Solo se puede crear un Fibonacci Tree solicitando el “orden” deseado, tal cual fue visto en clase. Por tal motivo, no tiene sentido guardar en sus nodos la altura almacenada ni el factor de balance, pues **nunca tiene que hacer rotaciones**.

Se pide:

1.1) Implementar el constructor de un solo parámetro:

```
public FiboTree(int N)
```

que genera un árbol de Fibonacci de orden pedido. Si N no es positivo, lanzar RuntimeException (no podríamos graficarlo si N es 0 porque el graficador precisa que el root no sea null).

NO CAMBIAR NADA DEL CODIGO YA ESCRITO.

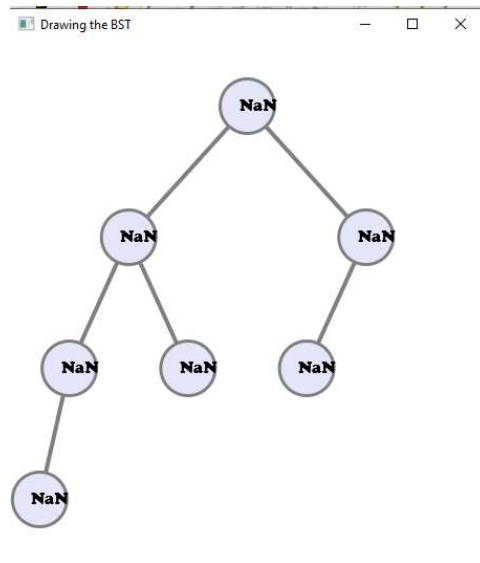
En el código que bajaron está lo que se precisa para graficarlo. Como en esta parte solo se está interesado en que solo tenga “la forma de un Fibonacci Tree” con el orden solicitado, en sus datos se deberá almacenar **Float.NaN**. (o sea, en esta parte no estamos interesados en considerar su información).

Caso de Uso A

Si en TestGUI, en el método createModel invocamos:

```
private FiboTree createModel() {  
    FiboTree myTree = new FiboTree(4);  
    return myTree;  
}
```

Debe obtenerse

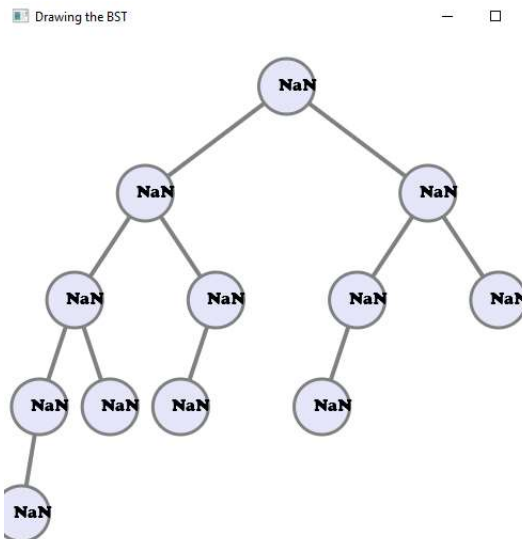


Caso de Uso B

Si en TestGUI, en el método createModel invocamos:

```
private FiboTree createModel() {  
    FiboTree myTree = new FiboTree(5);  
    return myTree;  
}
```

Debe obtenerse



Caso de Uso C

Si en TestGUI, en el método createModel invocamos:

```
private FiboTree createModel() {  
    FiboTree myTree = new FiboTree(0);  
    return myTree;  
}
```

Debe obtenerse un RuntimeException. Por ejemplo:

Caused by: [java.lang.RuntimeException](#): We only accept positive N

1.2) Implementar el constructor de 2 parámetros según se explica a continuación

Se quiere finalmente implementar un verdadero Fibonacci Tree no solo en "su forma" (como el ejercicio 1.1) sino asegurando que está ordenado (debe ser un BST).

Para ello, este constructor recibe un parámetro adicional: "lower" que es el mínimo valor numérico que debe estar presente en el mismo, y todos los demás valores deben ser mayores que él. Los valores que deben estar presente en el árbol deben formar una secuencia entera incremental con step 1: lower, lower+1, lower+2, lower+3, etc.

Obviamente este árbol es un verdadero Fibonacci Tree y por lo tanto esos números no pueden estar en cualquier lugar.

Leer detalladamente lo que se solicita:

- Terminar de implementar el constructor de dos parámetros, donde el primer parámetro es el orden del árbol de Fibonacci deseado y el segundo parámetro es el mínimo valor que debe estar presente en este árbol (los demás valores son lower+1, lower+2, lower+3, etc.).
- Para lograr esto se debe recorrer el **Fibonacci Tree ya construido** (ver el uso del this(N) en la primera línea) y a partir de ese árbol obtenido con la forma deseada, se debe **"cambiar" la property "data" de cada nodo recorriendo el árbol una sola vez**, para que quede con los valores correctos. No debe quedar ningún "data" con valor Float.NaN.

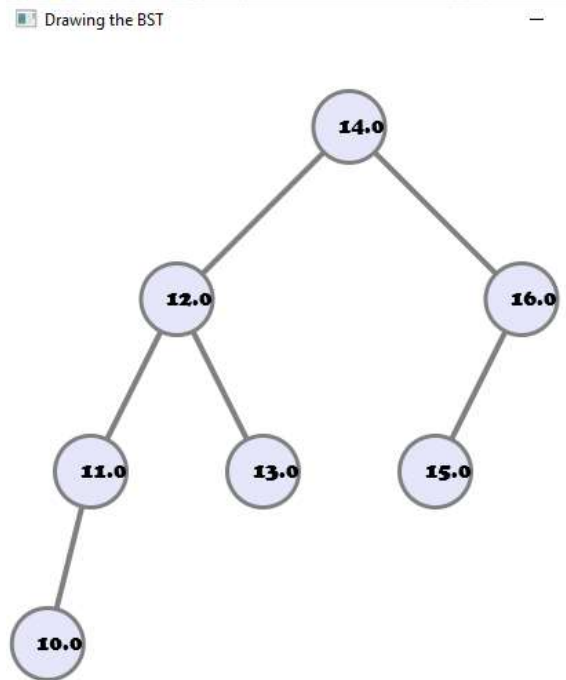
```
public FiboTree(int N, int lower){  
    this(N); // el constructor del 1.1  
    // COMPLETAR CON el cambio de Float.NaN segun lo explicado  
}
```

Caso de Uso A

Si en TestGUI, en el método createModel invocamos:

```
private FiboTree createModel() {  
    FiboTree myTree = new FiboTree(4, 10);  
    return myTree;  
}
```

Debe obtenerse




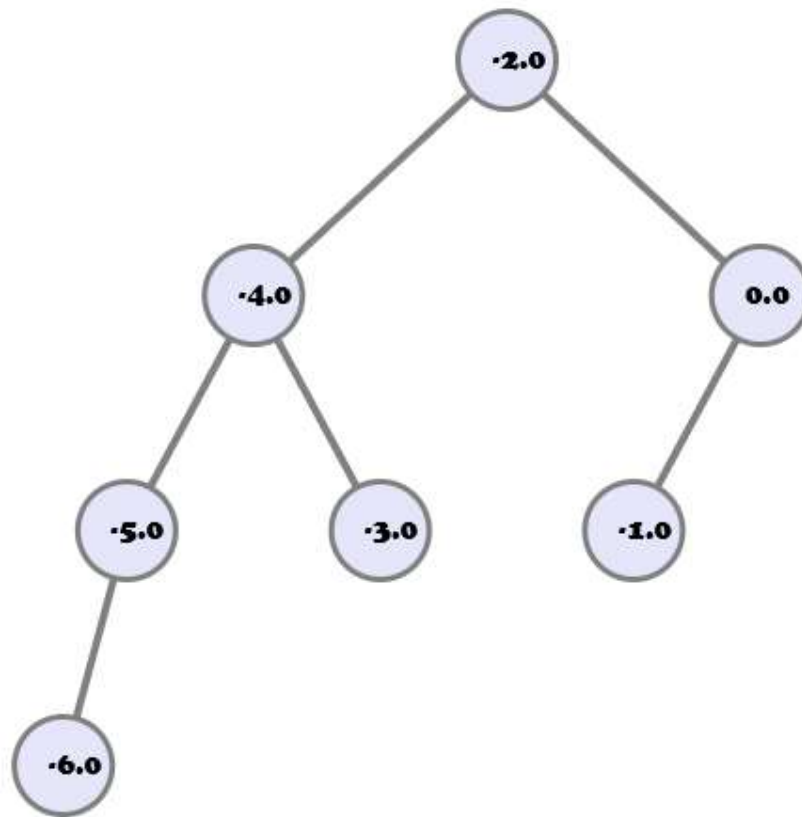
Caso de Uso B

Si en TestGUI, en el método createModel invocamos:

```
private FiboTree createModel() {  
    FiboTree myTree = new FiboTree(4, -6);  
    return myTree;  
}
```

Debe obtenerse

 Drawing the BST



Caso de Uso C

Si en TestGUI, en el método createModel invocamos:

```
private FiboTree createModel() {  
    FiboTree myTree = new FiboTree(-1, 2);  
    return myTree;  
}
```

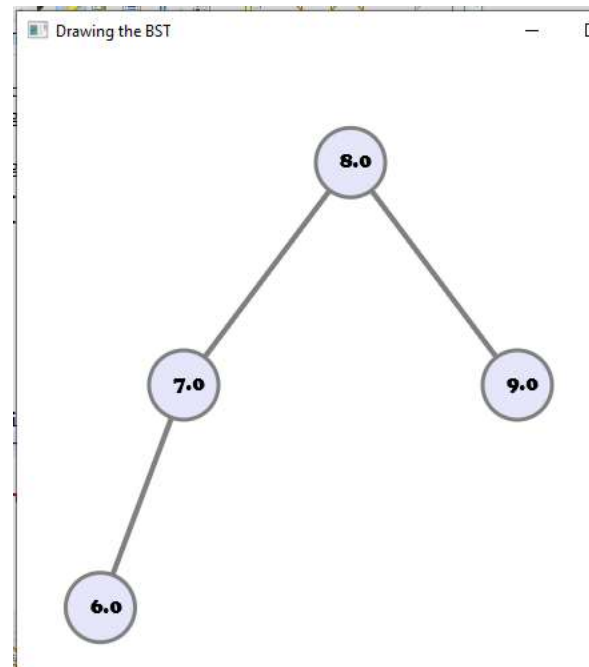
Debe obtenerse RuntimeException tipo

Caused by: [java.lang.RuntimeException](#): We only accept positive N

Caso de Uso D

Si en TestGUI, en el método createModel invocamos:

```
private FiboTree createModel() {  
    FiboTree myTree = new FiboTree(3, 6);  
    return myTree;  
}
```



Ejercicio 2

Bajar de Campus el proyecto maven **Ejer2.zip**.

Utilizar ese código que es un SUBCONJUNTO del grafo con Lista de Adyacencia implementado en clase.

NO cambiar nada de lo ya implementado.

En redes sociales existe una métrica muy importante para determinar qué tan agrupado están los vecinos de un nodo.

Definición: Coeficiente Local de Agrupamiento de un Vértice V en un grafo simple sin lazos no dirigido.

Es la proporción entre **cantidad de pares de vecinos de V que están conectados entre ellos por un eje** respecto a la **cantidad de ejes que “potencialmente” se podría tener si los vecinos de V formaran un grafo completo**.

Un nodo tiene 0 o 1 vecino se considera que su Coeficiente es -1 porque no puede dividirse por 0.

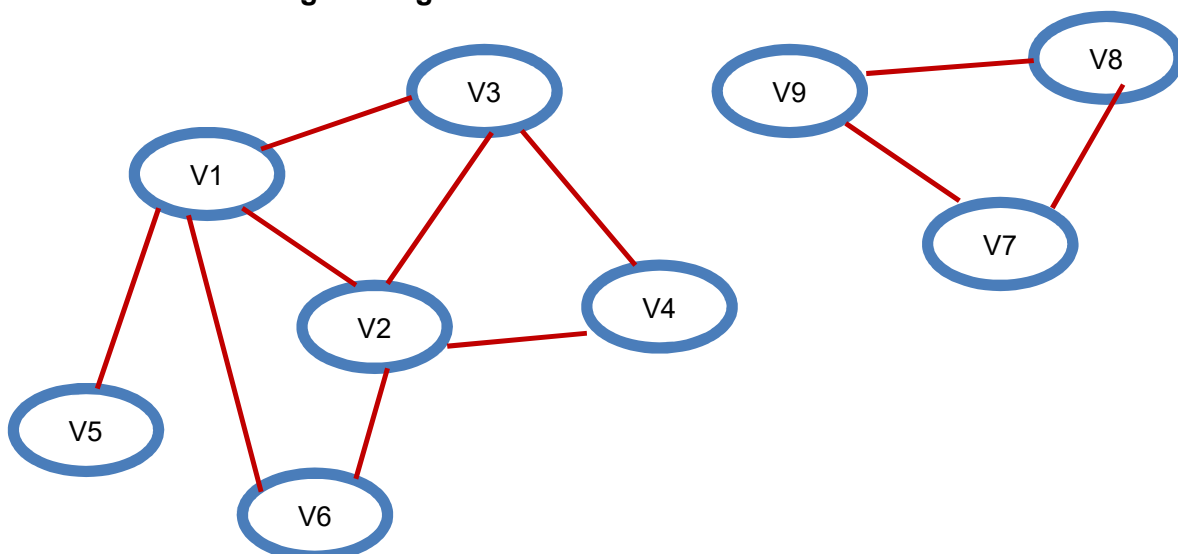
Como indica la definición, para cierto vértice V se debe calcular un cociente entre 2 valores.

El numerador calcula los triángulos dónde V es uno de los vértices.

El denominador calcula la cantidad de ejes que tendrían los vecinos de V si formaran un grafo completo (fácil de calcular porque es

$$0.5 * \text{degree}(V) * (\text{degree}(V) - 1)$$

Caso A. Dado el siguiente grafo

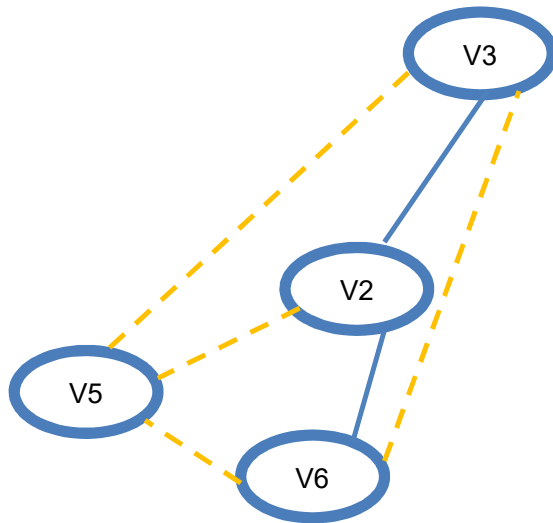


- Si el nodo de análisis es **V1**.

Numerador a considerar: Los vecinos de v1 son v2, v3, v5 y v6. Los pares de esos nodos que tienen un eje entre ellos son solo 2: (v2, v3), (v2, v6). Entonces, **son 2** los triángulos donde v1 es uno de sus vértices..

Notar que v2 v3 y v4 forman triángulo, pero v1 no es parte del mismo, por eso no se lo considera. Análogamente v9, v7 y v8 forman triángulo, pero v1 no es parte del mismo y no se lo tiene en cuenta.

Denominador a considerar: sus vecinos son v2, v3, v5 y v6 y si formaran un grafo completo **tendríamos 6 ejes**. Eso es potencial, no necesariamente existen esas conexiones en el grafo. Lo podemos calcular con la fórmula $0.5 * \text{degree}(v1) * (\text{degree}(v1) - 1)$



Así, el **Coeficiente Local de Agrupamiento de v1 es** $2/6$ o sea **0.333**

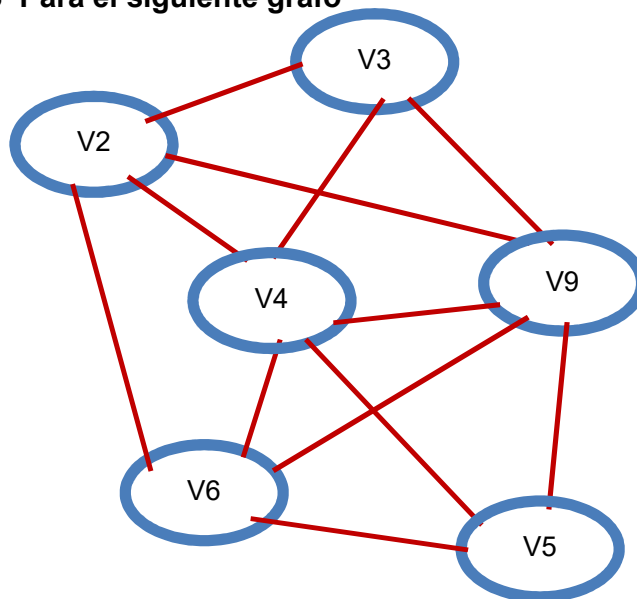
- Si el nodo de análisis es v5, el **Coeficiente Local de Agrupamiento de v5 es** **-1.0**

- El **Coeficiente Local de Agrupamiento** para **v2** es **0.5**. Sus vecinos son: v1, v3, v4 y v6. Los pares de esos nodos que tienen un eje entre ellos son **tres** (v1, v3), (v3, v4) y (v1, v6). Otra vez tiene 4 nodos vecinos y por lo tanto el **denominador es 6**. Su Coeficiente Local de Agrupamiento es $3/6$ o sea **0.5**

- El **Coeficiente Local de Agrupamiento** para **v8** es **1** (hay un único par de nodos que forma triángulo con él). Sus vecinos son 2 y la cantidad de ejes del grafo completo que formarían esos 2 nodos es **1**. Su Coeficiente Local de Agrupamiento es entonces $1/1$ o sea, **1.0**

Y así podríamos analizar los coeficientes para otros nodos.

Caso B Para el siguiente grafo



- El **Coeficiente Local de Agrupamiento** para **v2** es **0.8333**. Sus vecinos son: v3, v4, v6 y v9. Los pares de esos nodos que tienen un eje entre ellos son **cinco** (v3, v4), (v3, v9), (v4, v6) y (v4, v9) y (v6, v9). Además, los vecinos de v2 son 4 y formarían un grafo completo con **6 ejes**. Su Coeficiente Local de Agrupamiento es $5/6$ o sea **0.8333**

- El **Coeficiente Local de Agrupamiento** para **v9** es **0.7**. Sus vecinos son: v2, v3, v4, v5 y v6. Los pares de esos nodos que tienen un eje entre ellos son **siete** (v2, v3), (v2, v4), (v2, v6), (v3, v4), (v4, v5), (v4, v6) y (v5, v6). Además, los vecinos de v9 son 5 y formarían un grafo completo con **10 ejes**. Su Coeficiente Local de Agrupamiento es $7/10$ o sea **0.7**

Se pide

Completar el método **solo en la clase SimpleOrDefault**

```
public double getLocalClusteringCoeff(V aVertex);
```

según lo explicado sin visitar nodos que se sabe no contribuyen al resultado. Ya se hicieron algunas validaciones.