Lun 16 de diciembre de 2019

SEGUNDO PARCIAL

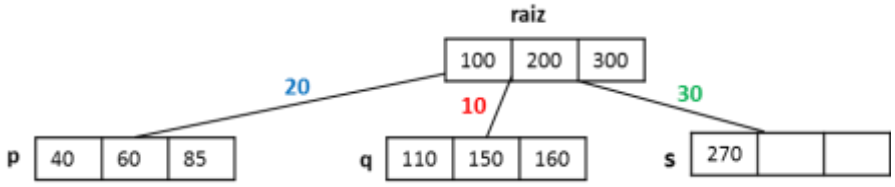
INF310 SX– Estructuras de Datos II. Gestión 2-2019.

Subgrupo: A-L

Árbol M-Vías pesado

1. Para que un árbol M-Vías se convierta en *pesado*, basta asignar un peso > 0 a cada uno de los punteros no-nulos. El peso del puntero que sostiene a un Nodo, se almacena en el mismo Nodo (La Raíz excepcionalmente tiene su peso=0, porque nadie la sostiene).

Por ejemplo




```
raíz.getPeso()=0      //La raíz siempre tiene costo=0
p.getPeso()=20
q.getPeso()=10
s.getPeso()=30
```

//En la consola un nodo se muestra con: el peso, un guión y luego los datos. Por ejemplo, el nodo p se verá **`20-[40|60|85].`**

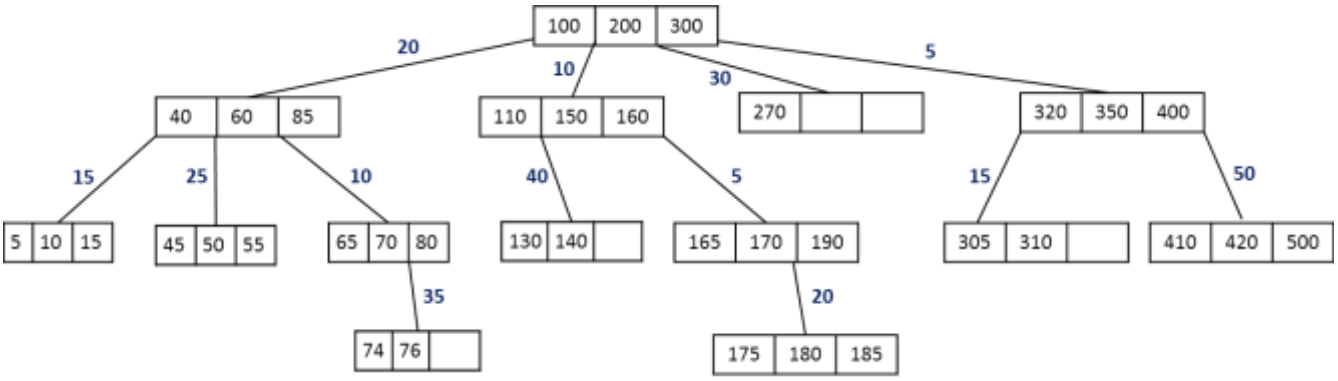
Se define el **costo** de un data x, como la suma de los pesos de los punteros que se deben recorrer desde la raíz hasta el nodo que contiene a x. Sabiendo esto, escriba la función

```
public int getFirstData(int cost)
```

la cual, **usando** una función **máscara RECURSIVA**, busque el nodo cuyo costo es `cost`. Si tal nodo existe, devolver el **primer data** de ése nodo; caso contrario, devolver –1.

Por comodidad, asuma que todos los datas son mayores que 0.
Si dos o más nodos tienen el mismo costo, escoja el primero que encuentre su algoritmo.

Por ejemplo: Dado el árbol A




```
A.getFirstData(40)=-1  //Porque no existe un nodo cuyo costo sea 40.

A.getFirstData(0)=100    //Devolver el primer data de la Raíz, porque la raíz tiene costo 0.

A.getFirstData(65)=74   //Porque partiendo desde la Raíz hasta llegar al nodo [74|76| ], recorremos los punteros cuyos pesos
                        //son: 20+10+35=65. Entonces, devolver el primer data de ése nodo (o sea el 74).
```

`A.getFirstData(35)=5` //Partiendo desde la Raíz hasta llegar al nodo [5|10|15], recorremos los punteros cuyos pesos son: //20+15=35. Note que también sería correcto devolver 175, pues el nodo [175|10|15] también tiene //costo 35.

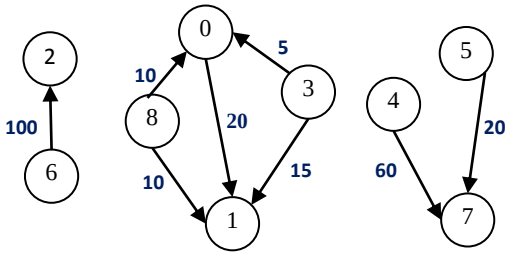
Grafos con Peso

2. Informalmente, llamamos **costo de una isla** (Componente Conexo), a la suma de los pesos de sus aristas. En la class Grafo, escriba la función

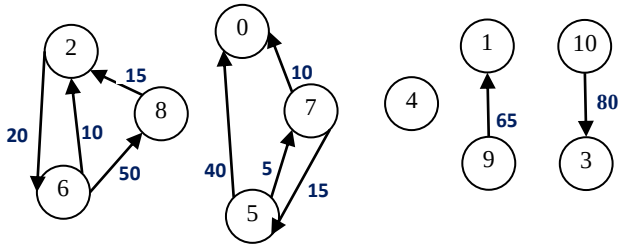
```
public int costoIsla(int u)
```

la cual devuelva el costo de las isla que **contiene** al vértice u.

Por ejemplo:



Grafo A



Grafo B

- El Grafo A tiene tres islas:
  La isla (2,6) cuyo costo es 100
  La isla (0, 8, 3, 1) cuyo costo es 10+5+10+20+15 = 60 (solo se suman los pesos de todas las aristas que hay en la isla).
  La isla (5, 4, 7) cuyo costo es 60+20 = 80

  Así,
  `A.costoIsla(2)=100` //Porque el vértice 2 está en la primera isla (2,6) y su costo es 100.
  `A.costoIsla(6)=100` //Porque el vértice 6 está en la primera isla (2,6) y su costo es 100.
  `A.costoIsla(8)=60` //Porque el vértice 8 está en la segunda isla (0, 8, 3, 1) y su costo es 60.
  `A.costoIsla(1)=60` //Porque el vértice 1 está en la segunda isla (0, 8, 3, 1) y su costo es 60.
  `A.costoIsla(7)=80` //Porque el vértice 7 está en la tercera isla (5, 4, 7) y su costo es 80.
  `A.costoIsla(5)=80` //Porque el vértice 5 está en la tercera isla (5, 4, 7) y su costo es 80.

- El Grafo B tiene cinco islas:

  La isla (2,6, 8) cuyo costo es 20+10+15+50 = 95
  La isla (0, 5, 7) cuyo costo es 40+5+10+15 = 70
  La isla (4) cuyo costo es 0 (porque en esta isla no hay aristas)
  La isla (1,9) cuyo costo es 65
  La isla (10,3) cuyo costo es 80

  Así,
  `B.costoIsla(2)=95` //Porque el vértice 2 está en la primera isla (2,6, 8) y su costo es 95.
  `B.costoIsla(8)=95` //Porque el vértice 8 está en la primera isla (2,6, 8) y su costo es 95.
  `B.costoIsla(5)=70` //Porque el vértice 5 está en la segunda isla (0, 5, 7) y su costo es 70.
  `B.costoIsla(4)=0` //Porque el vértice 4 está en la tercera isla (4) y su costo es 0.