Pregunta 1

A la tercera pràctica, vau implementar una **cua amb prioritats** utilitzant un ArrayList en forma de **heap de tripletes** (prioritat, timestamp, valor).

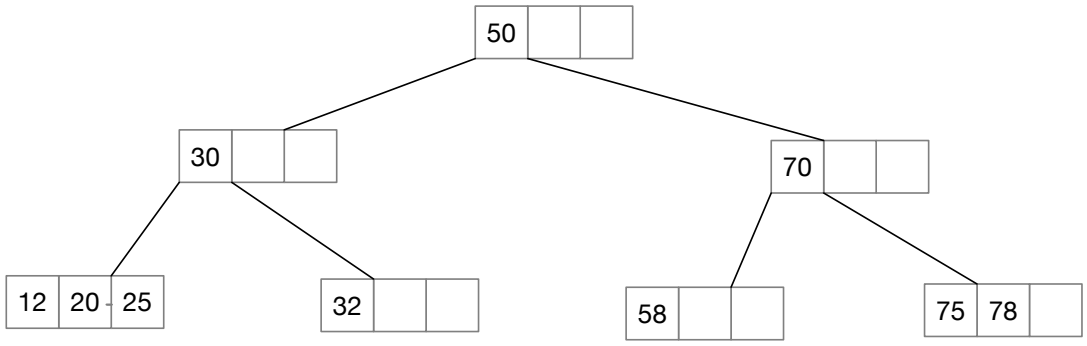
Donada la seqüència d'operacions:

```
var pq = new HeapQueue<Integer, String>();
pq.add(10, "a");
pq.add(20, "b");
pq.add(15, "c");
pq.add(20, "d");
pq.add(15, "e");
var first = pq.remove();
var second = pq.remove();
```

El que es demana és que **expliqueu, a cada pas, com s'executen aquestes operacions i perquè**. Mostreu el **heap** en forma **d'arbre de tripletes**, tot indicant també la **posició de cada node a l'ArrayList subjacent**.

Pregunta 2

Partint sempre de l'arbre B que es mostra a la figura:



Realitzeu les següents operacions, tot indicant **què feu en cada cas i perquè**:

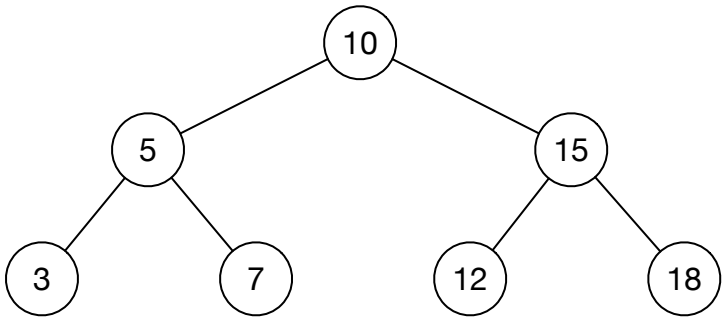
- **Eliminar** la clau **50**
- **Inserir** la clau **28**

Pregunta 3

Donat un **arbre binari de cerca** que, per simplificar, suposarem d'enters, i un valor **min** i un **max**, implementeu la funció **count** que retorni el **nombre d'elements de l'arbre que tenen valors en el rang tancat [min, max]**.

```
public static int count(BinaryTree<Integer> bst, int min, int max) {}
```

Per exemple, a l'arbre:



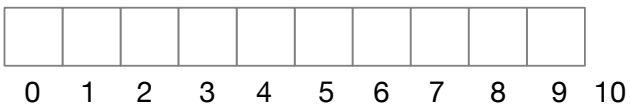
Si cridem a la funció amb min=8 i max=12, el resultat és 2.

Com sempre, **raoneu els casos simples i recursius**, especialment aplicant la propietat de que **l'arbre és un arbre binari de cerca**.

NOTA: **No s'acceptaran solucions que no facin servir la propietat de que l'arbre és un arbre binari de cerca.**

Pregunta 4

Donada la següent **taula de hash de 10 posicions**, on els elements a inserir tenen una **clau** que és un **enter**.



L'estratègia que seguirem serà la de **dispersió tancada** en la que la **funció de hash** serà:

$$h_i(k) = (k + i^2) \; mod \; 10$$

Les operacions a executar, **explicant què succeeix en cada cas, com es modifica l'array i perquè**:

- t.put(12, "a")
- t.put(22, "b")
- t.put(33, "c")
- t.put(44, "d")
- t.put(52, "e")
- t.remove(22)
- t.put(62, "f")

Annex:

```
interface BinaryTree<E>{
    E root();
    BinaryTree<E> left();
    BinaryTree<E> right();
    boolean isEmpty();
    int size();
}
```

```
interface Map<K, V> {
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    int size();
}
```