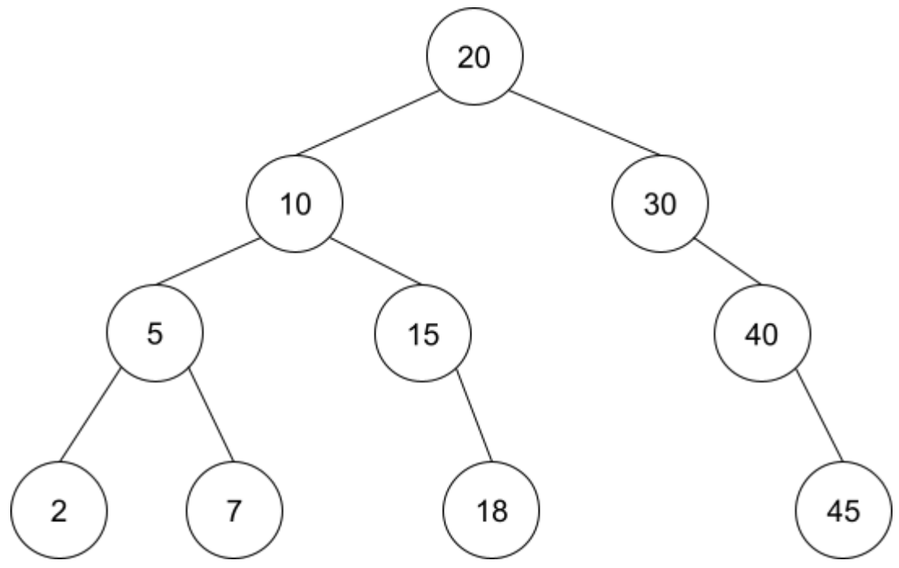
Ejercicio 1

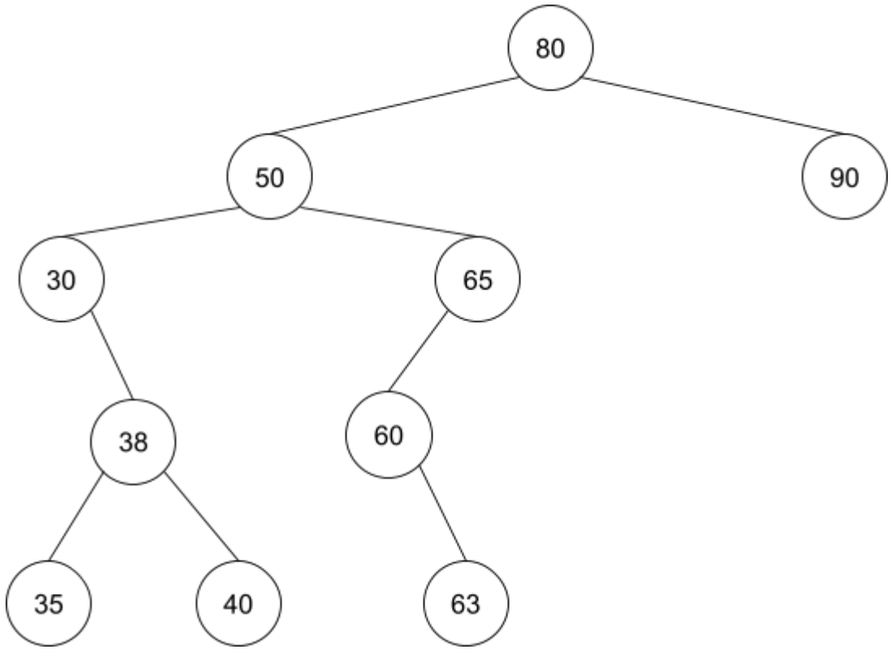
a) Responda qué valor retornará el método **calcular** al ser invocado como **calcular(raiz, 2)** sobre el árbol de la derecha donde **raíz** referencia al nodo raíz de dicho árbol binario.

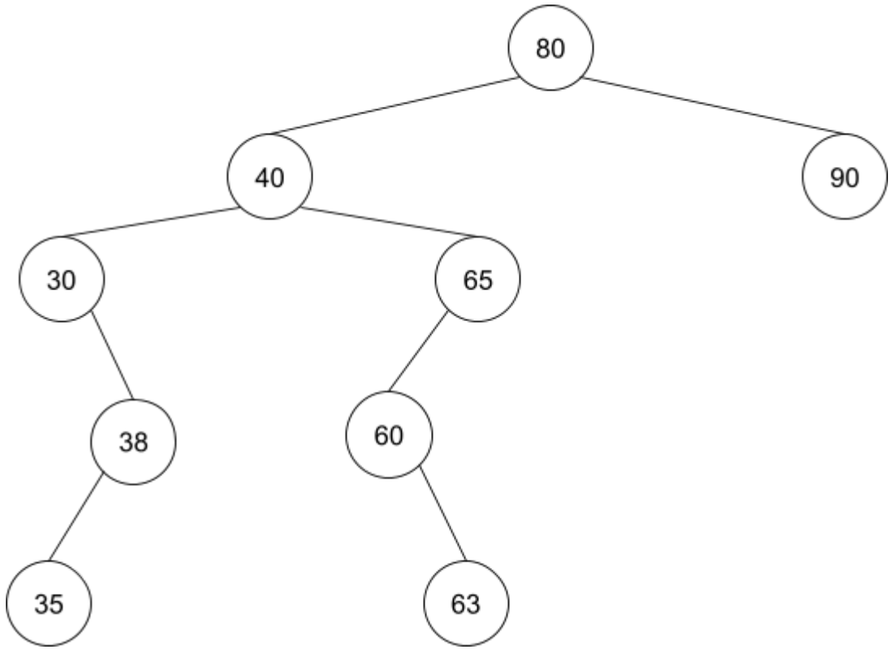
```
private int calcular(TNode root, int level) {
        int result = 0;
        if (root != null) {
                if (level == 0) {
                        result = 2 * root.getElem();
                } else {
                        result = (2 * root.getElem()) + calcular(root.getLeft(), level - 1) + calcular(root.getRight(), level - 1);
                }
        }
        return result;
}
```



result = 2 * 20 + calcular(root.getLeft(), 1) + calcular(root.getRight(), 1);
result = 2 * 20 + 2 * 10 + calcular(root.getLeft().getLeft(), 0) + calcular(root.getLeft()getRight(), 0) + 2 * 30 + calcular(root.getRight().getLeft(), 0) + calcular(root.getRight()getRight(), 0);
result = 2 * 20 + 2 * 10 + 2 * 5 + 2 * 15 + 2 * 30 + 0 + 2 * 40;
result = 240;

b) Dado el árbol de búsqueda binario de la derecha, dibuje como podría quedar luego de **borrar el nodo 50**.

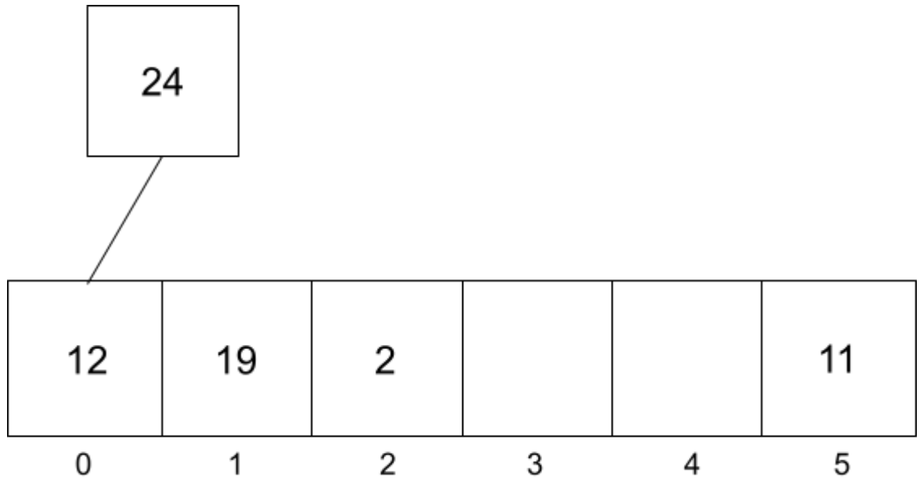


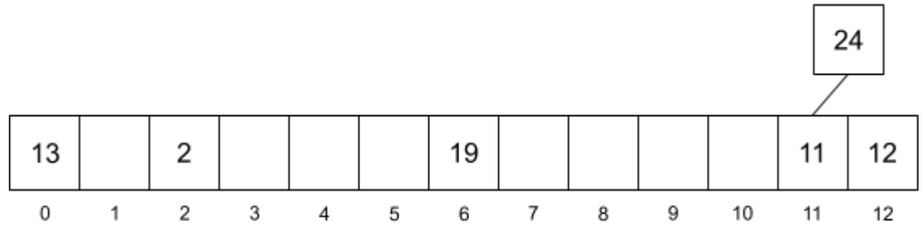


→ Para que siga siendo árbol binario: Busco el más mayor del subárbol izquierdo, lo reemplazo y lo elimino de hoja (o el menor del subárbol derecho).

Ejercicio 5

a) Dada la tabla de hashing de la derecha que se comporta como el HashTable de JAVA, y se encuentra cargada de la manera que se muestra. Si M = 6 y ρd = 1. **Indique cómo quedará si se inserta la clave 13**.





- *1ro: Calcular el threshold*

  L = M * Rp * ρd

  L = 6 * 1 * 1 = 6 → Será necesario hacer crecimiento ya que hay 5 elementos y habrán 6 después de insertar el 13.

  La cantidad de elementos debe ser menor al límite.

- *2do: Aumentar!* Aplico la fórmula M = M * 2 1

  M = 6 * 2 + 1 = 13 → L = 13 * 1 * 1 = 13 OK!

- *3ro: Reacomodar elementos!* Aplico la fórmula Elem mod M = nuevoBalde

  12 mod 13 = 12
  19 mod 13 = 6
  2 mod 13 = 2
  11 mod 13 = 11
  24 mod 13 = 11
  13 mod 13 = 0

b) Qué **pros y contras** tendrá un factor de carga de diseño chico (por ej. 0,2) comparado con otro muy grande (por ej. mayor a 3) en las estructuras de hashing separado con crecimiento.

Factor de Carga ρ (o densidad de almacenamiento): Es la proporción de espacio utilizado respecto del espacio total de almacenamiento asignado.

ρ (rho) = cantidad de datos / capacidad total de los baldes

El valor que se elige para ρd mantiene un equilibrio entre longitud de listas de rebalse vs espacio que se desperdicia.

Un valor grande provocará que la estructura se llene demasiado y probablemente haya más colisiones y listas de rebalse más largas.

Un valor muy bajo mantendrá baja la relación entre elementos en la estructura respecto al tamaño de la misma, o sea un mayor desperdicio de espacio.

Entonces…

Pros → Un factor de carga de diseño chico hará menos posible que haya listas de rebalse.
Contras → Un factor de carda de diseño chico mantendrá mayor desperdicio de espacio.

c) **Responda verdadero o falso y justifique**: Si hay un único thread de ejecución en el proceso no habrá problemas de sincronización.

Verdadero! Ya que se utiliza sincronización cuando hay varios Threads accediendo al mismo recurso, al haber un único Thread no generará problema.