6. Colas de Prioridades

Carlos Zerón Martínez

Universidad Nacional Autónoma de México

zeron@ciencias.unam.mx

Martes 8 de Diciembre de 2020

Introducción

El objetivo es tener la forma de atender al siguiente objeto más importante de una colección de personas o tareas

Ejemplos

- Los médicos en un hospital atienden por lo general al siguiente paciente más crítico antes de alguno que llegó primero
- Al seleccionar el programa a ejecutar dentro de un sistema computacional, un criterio del sistema operativo puede ser tomar al que tenga mayor prioridad (hay tareas internas más prioritarias que las aplicaciones de los usuarios)
- En un centro de control de tráfico aéreo, decidir a cuál vuelo permitir el aterrizaje de varios que se acercan al aeropuerto. La elección puede ser influenciada por ejemplo por factores como la distancia a la pista correspondiente o por la cantidad de combustible que les queda a los aviones.

Generalidades

- La comparación de objetos (prioridades) mediante claves, que son valores asignados a un elemento que representan las prioridades de los objetos (no necesariamente son atributos que forman parte del objeto, pueden ser objetos externos)
- La clave o prioridad no necesariamente es única y es posible incluso que la clave de un objeto pueda cambiar a través del tiempo.
- La clave es de tipo comparable (el tipo de la clave define un orden total entre cualesquiera dos objetos de ese tipo)

Operaciones de una cola de prioridades

Una cola de prioridades es una estructura de datos que permite almacenar una colección de elementos con prioridades y cuenta con operaciones para:

- insertar elementos
- eliminar y conocer el elemento con mayor prioridad al momento
- determinar si es vacía
- determinar el número de elementos que la ocupan

TDA para cola de prioridades

En esta versión que veremos, se tiene acceso a aquel objeto que tiene valor mínimo en la clave dentro de la cola.

$\Rightarrow$

Mientras más bajo es el valor de la clave que tenga un objeto, tiene mayor prioridad (prioridad de objetos inversamente proporcional al valor de las claves)

Datos: Pares de Objetos (Elemento, Clave).

- El elemento es un objeto de tipo genérico
- La clave debe ser un objeto comparable

El objeto es un par al cual denominamos **entrada comparable**

TDA para cola de prioridades

Operaciones:

- isEmpty() : $\{V, F\}$. Devuelve un valor lógico que indica si la cola de prioridades tiene elementos o no.
- size() : $\mathbb{N} \cup \{0\}$. Devuelve el número de entradas (*Elemento*, *Clave*) existentes en la cola de prioridades.
- insert(e) : $\varnothing$. Inserta una entrada comparable e en la cola de prioridades.
- deleteMin() : (*Elemento*, *Clave*). Devuelve y elimina la entrada comparable que contiene la clave mínima. Nombre alternativo: removeMin().
- min() : (*Elemento*, *Clave*). Devuelve la entrada comparable que contiene la clave mínima.

Representación de una entrada comparable

Con una interfaz *ComparableEntry* se representa de forma abstracta un par de objetos (Elemento, Clave). Para formar una entrada comparable necesitamos objetos de tipo genérico y objetos comparables:

```
public interface ComparableEntry {

    /**
     * Devuelve la clave de esta entrada
     */
    public Comparable getKey();

    /**
     * Devuelve el elemento almacenado en
     * esta entrada
     */
    public Object getElement();

}
```

Interfaz para el TDA Cola de Prioridades

```
public interface ColaPrioridades {

    /**
     * Indica si la cola de prioridades
     * está vacía.
     */
    boolean isEmpty();

    /**
     * Devuelve el numero de pares (Elemento, Clave)
     * existentes en la cola de prioridades.
     */
    int size();

    /**
     * Inserta una entrada comparable en la cola de prioridades.
     */
    void insert(ComparableEntry e);
```

Interfaz para el TDA Cola de Prioridades

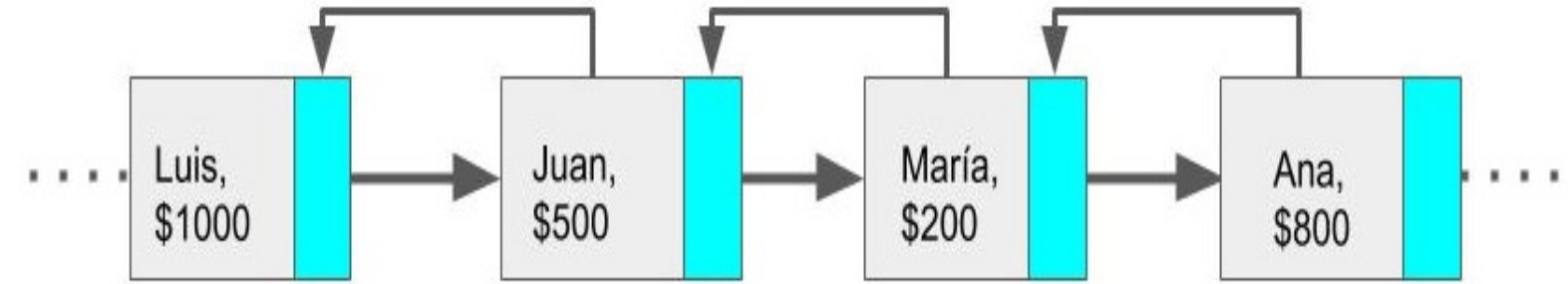
```
    /**
     * Devuelve la entrada comparable que contiene la clave minima
     * en la cola de prioridades.
     */
    ComparableEntry min();

    /**
     * Devuelve y elimina de la cola de prioridades la entrada comparable
     * que contiene la clave minima.
     */
    ComparableEntry removeMin();
}
```

Implementación con listas no ordenadas

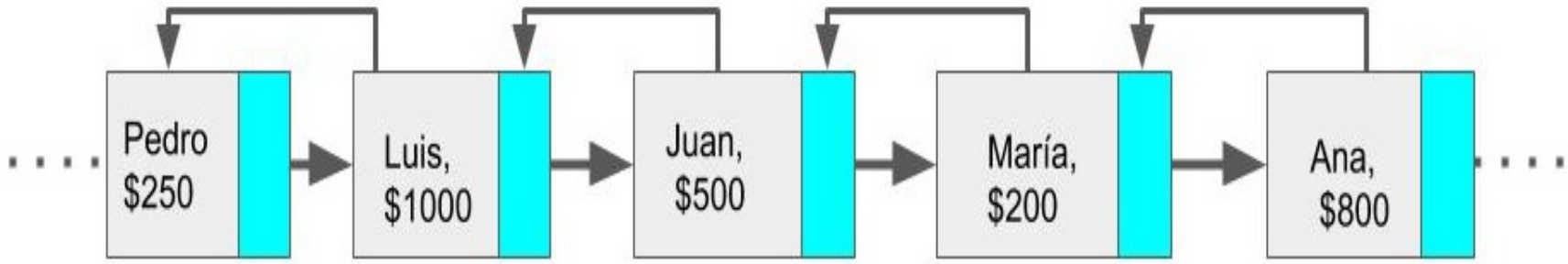
Se utiliza una lista doblemente ligada en general desordenada con respecto a los valores de las claves comparables que representan las prioridades. Si la lista tiene n elementos:

- La operación *insert*(e), al no hacer ningún tipo de comparación, se lleva a cabo en tiempo $O(1)$.
- El acceso a la entrada con clave mínima para *deleteMin*() o *min*() cuesta tiempo $O(n)$ sobre el tamaño de la lista, ya que se determina a pie la entrada con clave mínima.
- Aprovechando la implementación de listas doblemente ligadas, las operaciones *isEmpty*() y *size*() se ejecutan en tiempo $O(1)$
- Eliminaciones y accesos a la entrada con clave mínima son lentos pero las inserciones son rápidas.

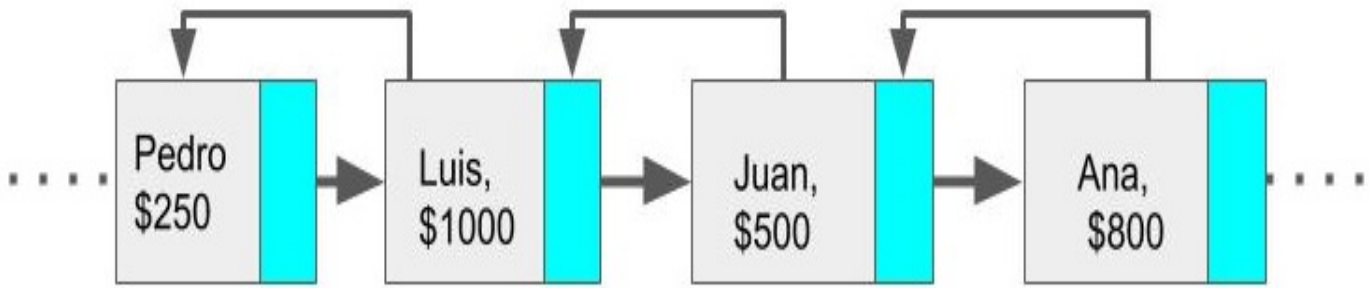
Implementación con listas no ordenadas



insert(Pedro, $250)



deleteMin()

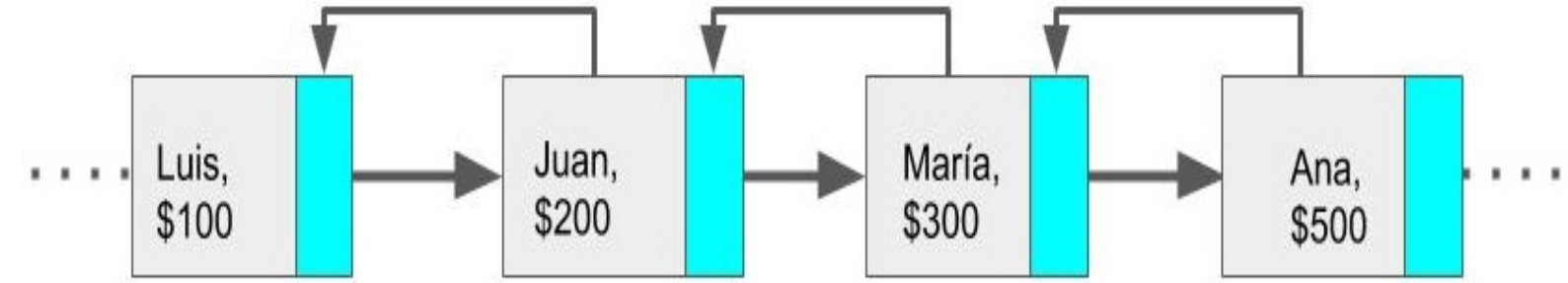


Implementación con listas ordenadas

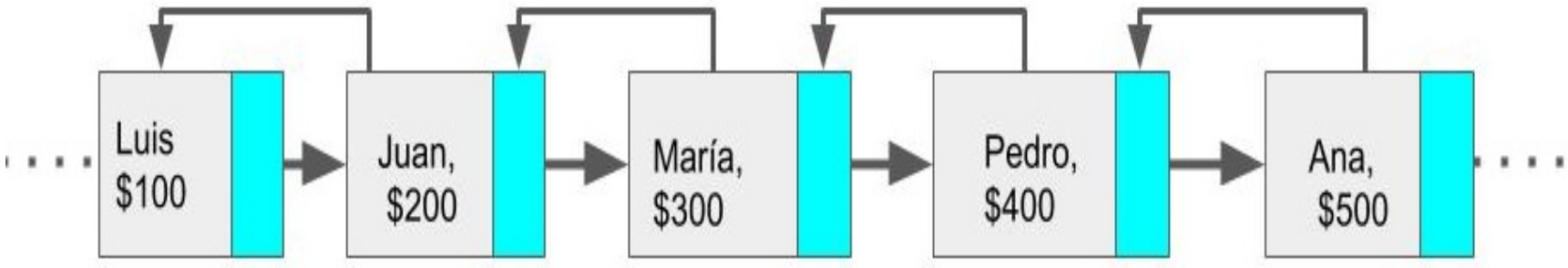
Se utiliza una lista doblemente ligada en la cual se mantiene el orden de los valores de las claves comparables que representan las prioridades. Si la lista tiene n elementos:

- La operación *insert*(e), al hacer comparaciones hasta encontrar la posición que le corresponde a la nueva entrada, se lleva a cabo en tiempo $O(n)$.
- El acceso a la entrada con clave mínima para *deleteMin*() o *min*() cuesta tiempo $O(1)$ sobre el tamaño de la lista, pues el hecho de que la lista esté ordenada implica que la clave mínima se encuentra en la primera posición.
- Aprovechando la implementación de listas doblemente ligadas, las operaciones *isEmpty*() y *size*() se ejecutan en tiempo $O(1)$
- Eliminaciones y accesos a la entrada con clave mínima son rápidos pero las inserciones son lentas.

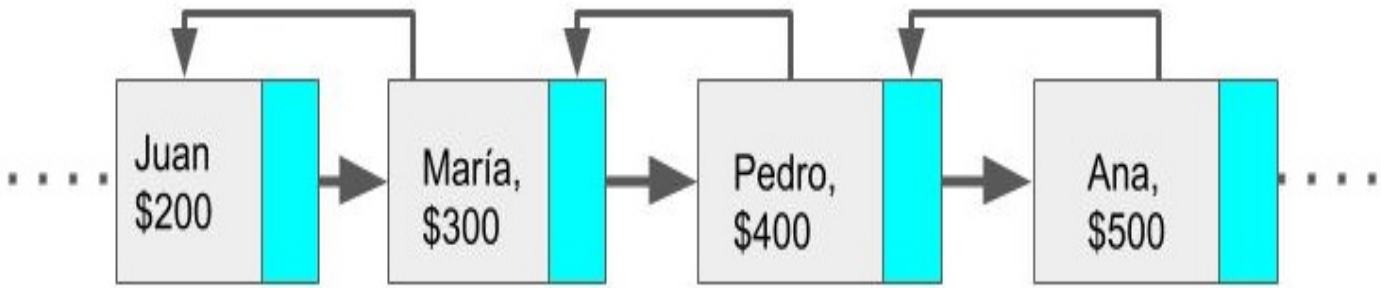
Implementación con listas ordenadas



insert(Pedro, $400)



deleteMin()



Implementación con heaps

Propiedad local de orden 1 para heap mínimo

Para cada nodo z y su padre u, donde z es distinto a la raíz, la clave de la entrada de z es mayor o igual a la clave de la entrada de u.

Inserción

Se le asigna el nodo z con la máxima posición posible a la entrada.

El árbol es binario completo pero podría no cumplir la propiedad local de orden.

Si la cola sólo tiene un nodo después de la inserción, teminamos, de otro modo, se efectúa el proceso de Burbujeo hacia arriba (*Up heap*)

Implementación con heaps

Burbujeo hacia arriba:

Se compara la clave de la entrada insertada en z con la clave de la entrada en el padre de z. Sea u tal nodo:

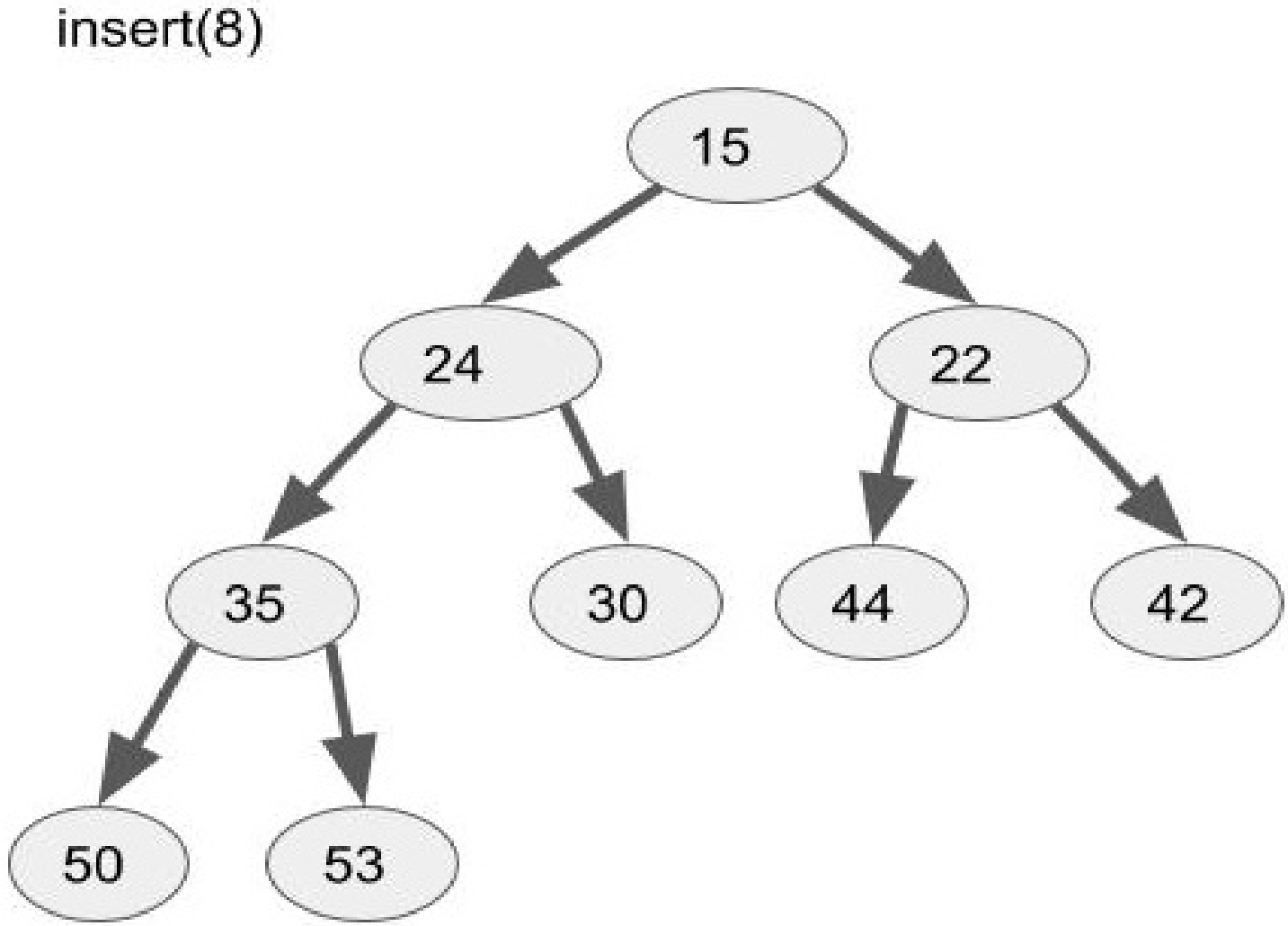
- Si la clave de z es mayor o igual a la de u, terminamos
- En caso contrario, se intercambian las entradas en z y u, se sigue el proceso con $z = u$,

El proceso sigue hasta que se cumpla la propiedad local de orden 1 o se alcance la raíz.

Como se hace un número de intercambios proporcional a la altura del heap, que es $\lfloor log_2 n \rfloor$, donde n es el número de nodos, el peor caso es $O(\log n)$.

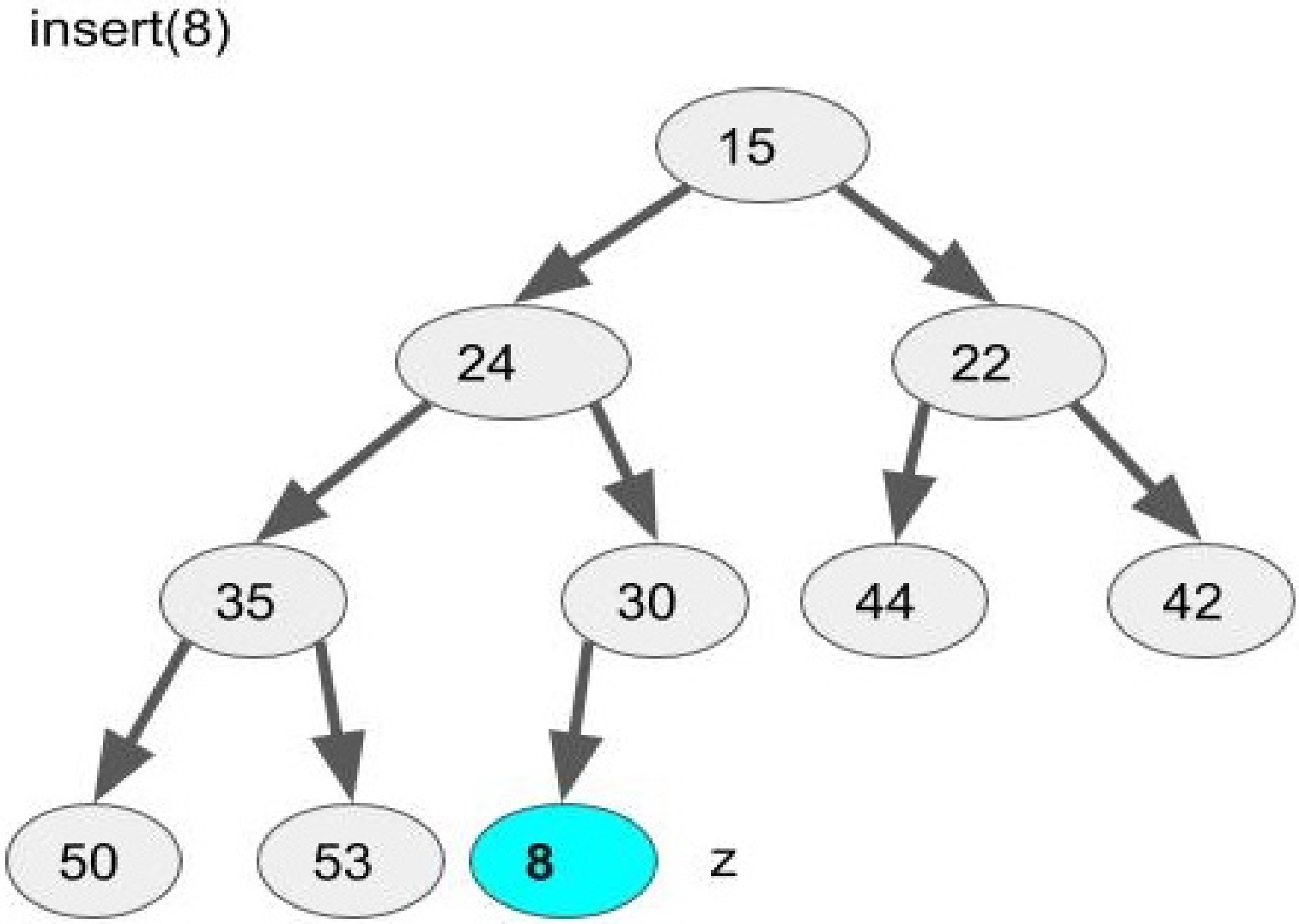
Implementación con heaps

Ejemplo de inserción



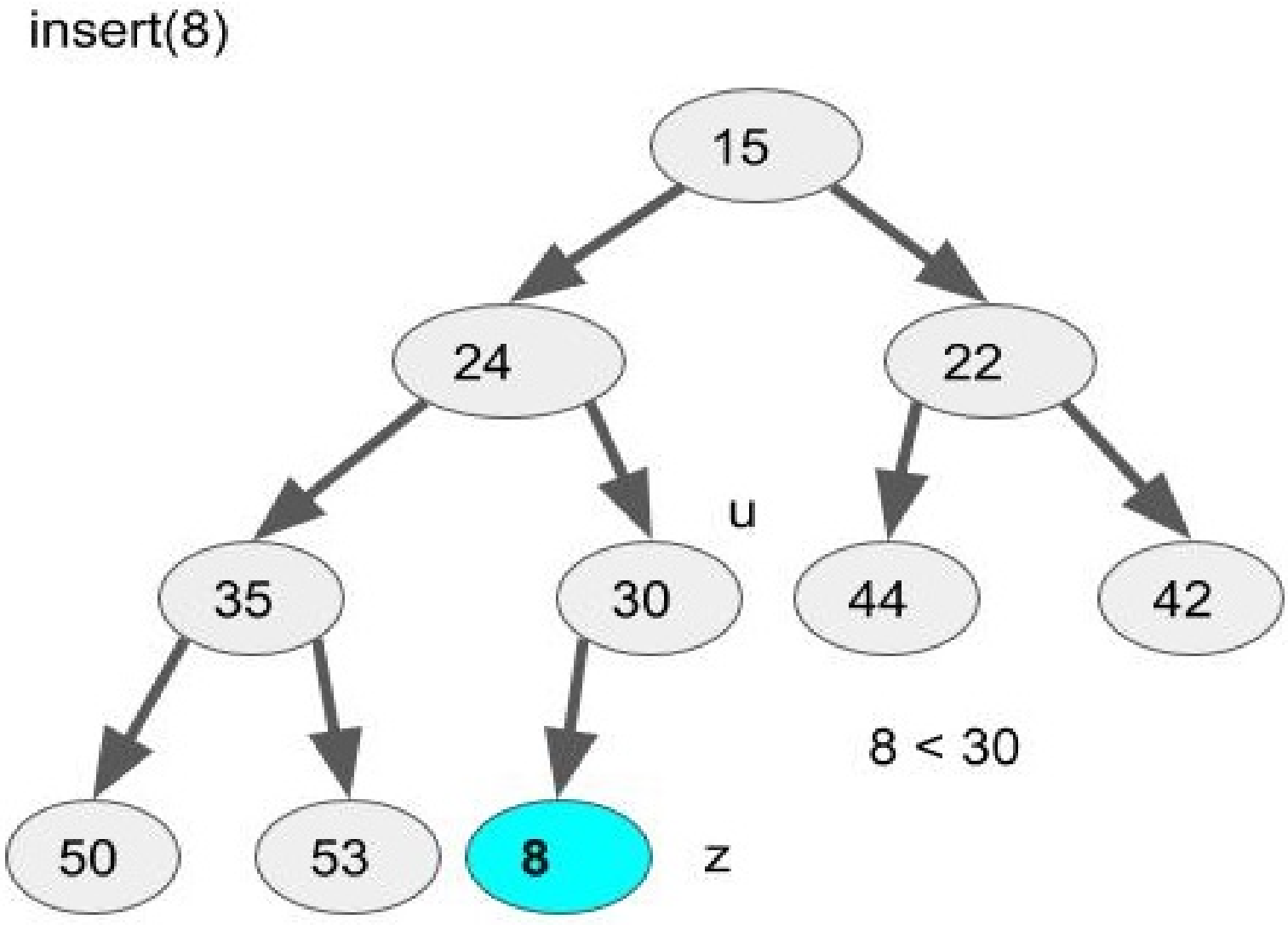
Implementación con heaps

Ejemplo de inserción



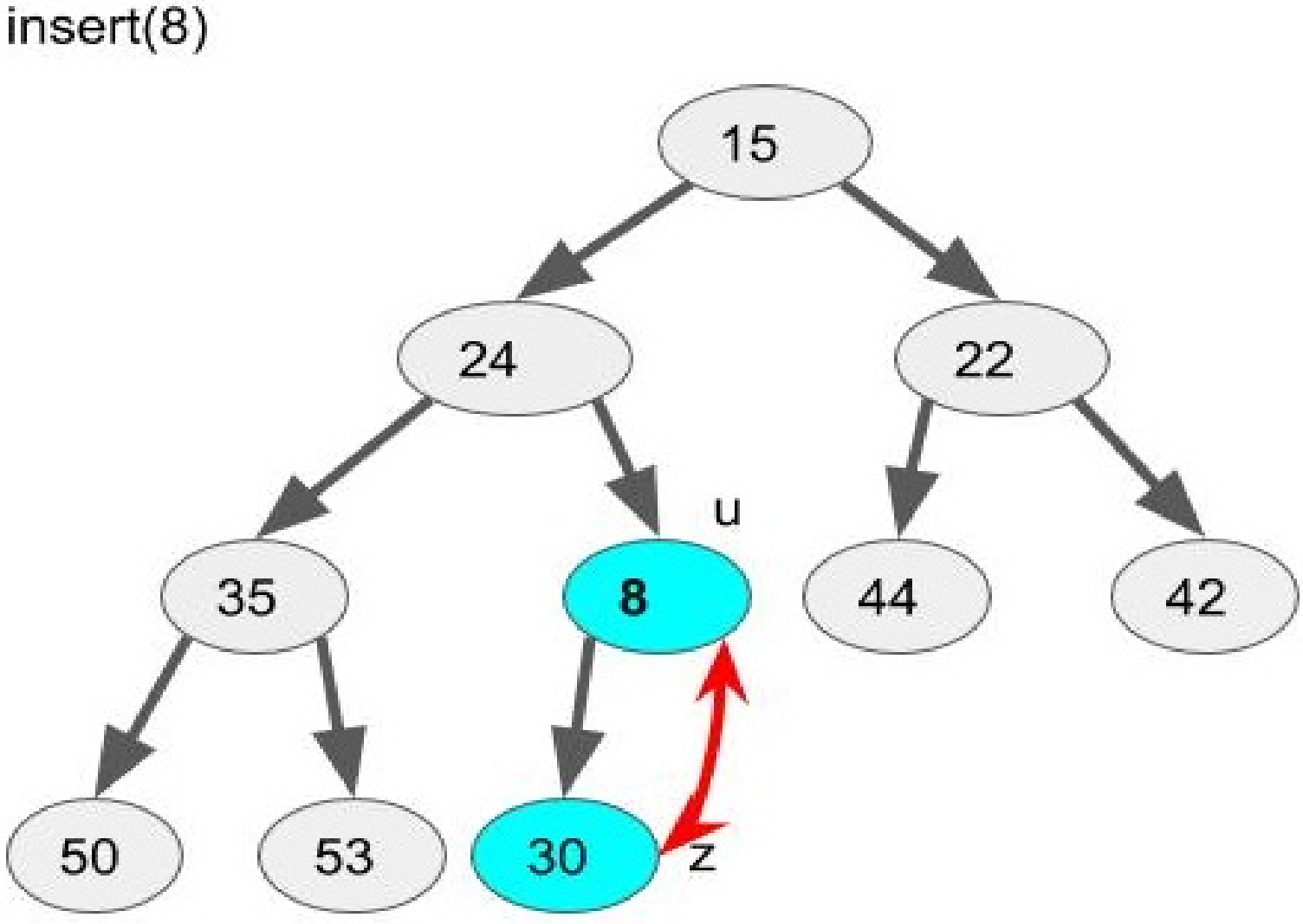
Implementación con heaps

Ejemplo de inserción



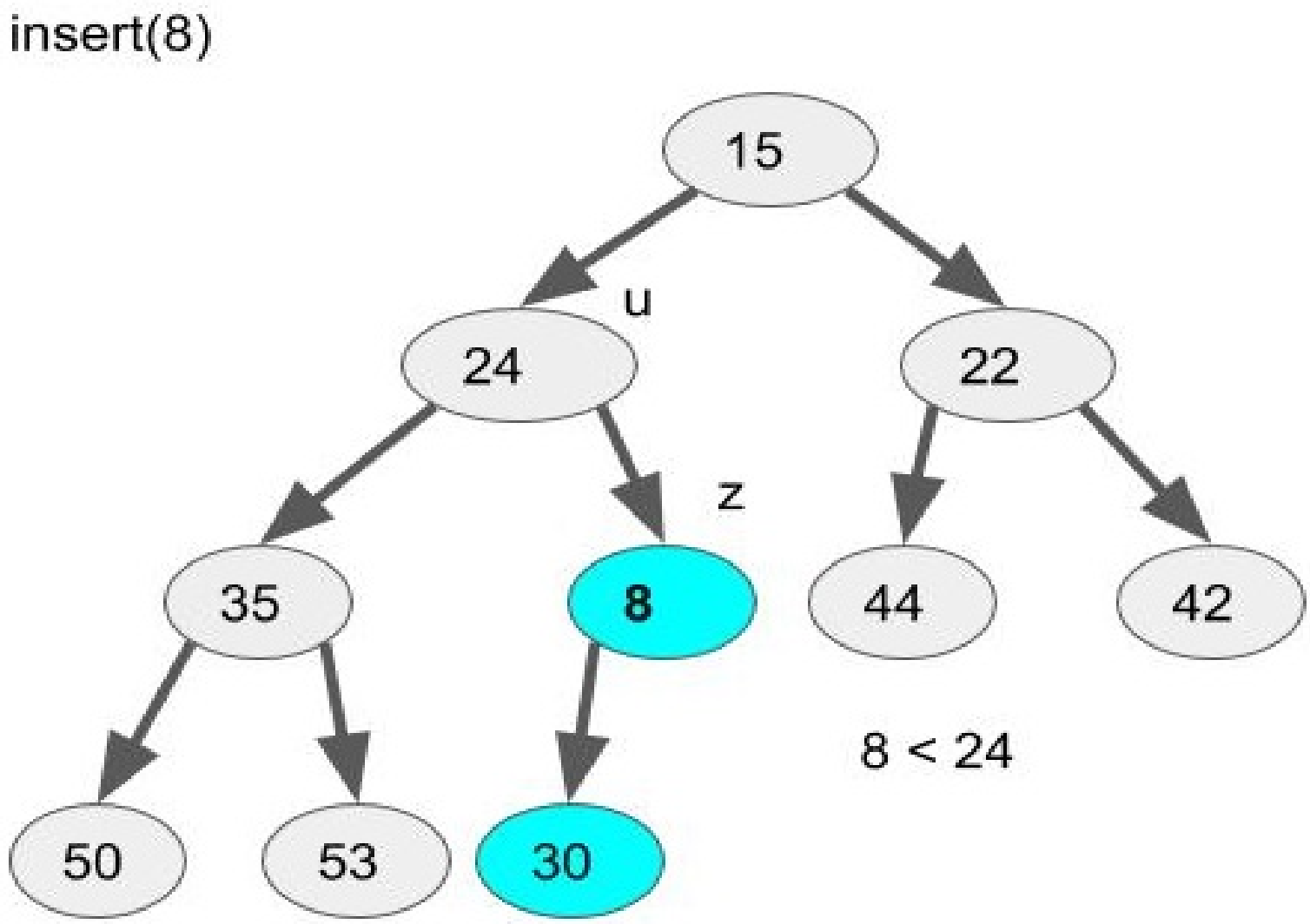
Implementación con heaps

Ejemplo de inserción



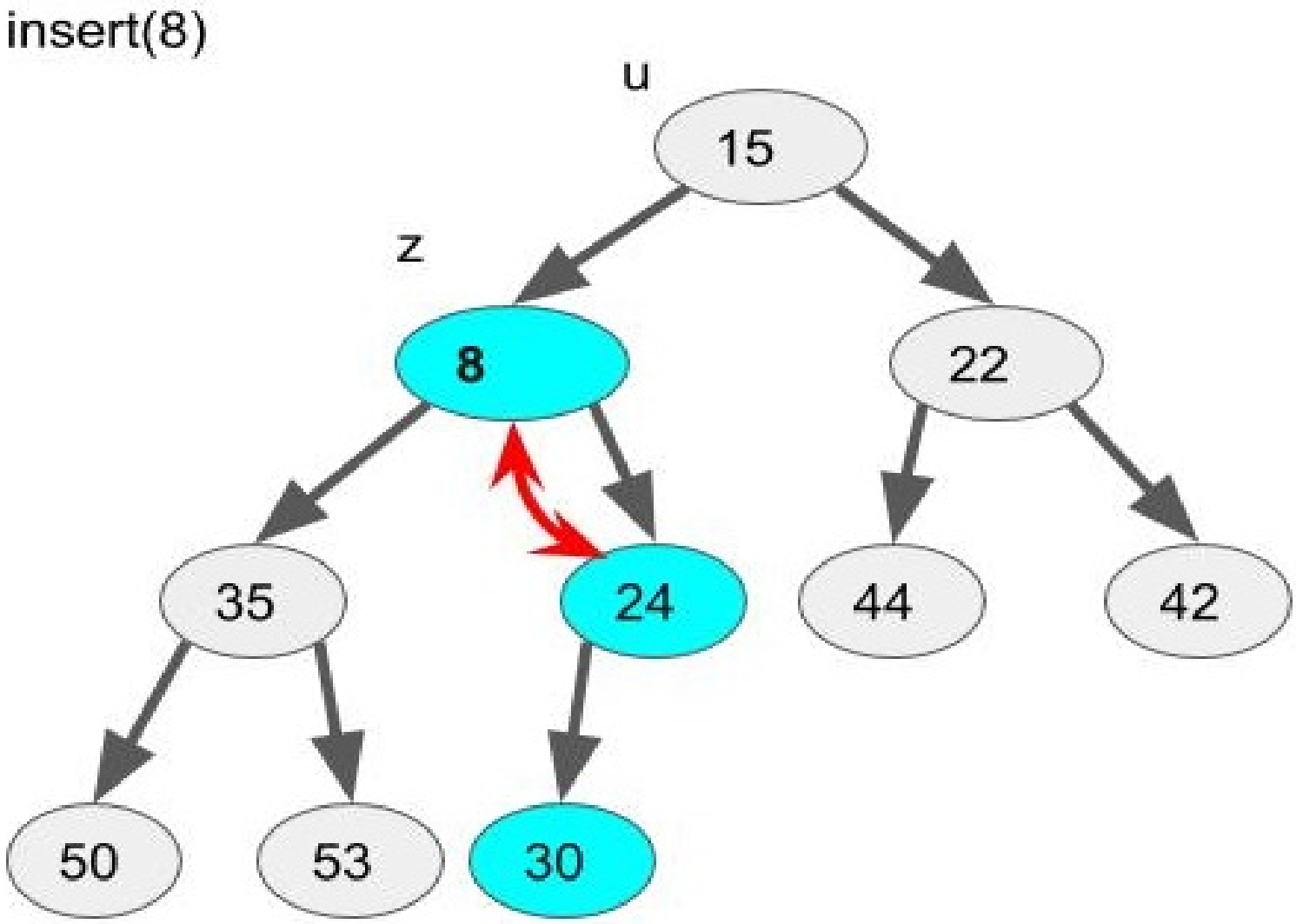
Implementación con heaps

Ejemplo de inserción



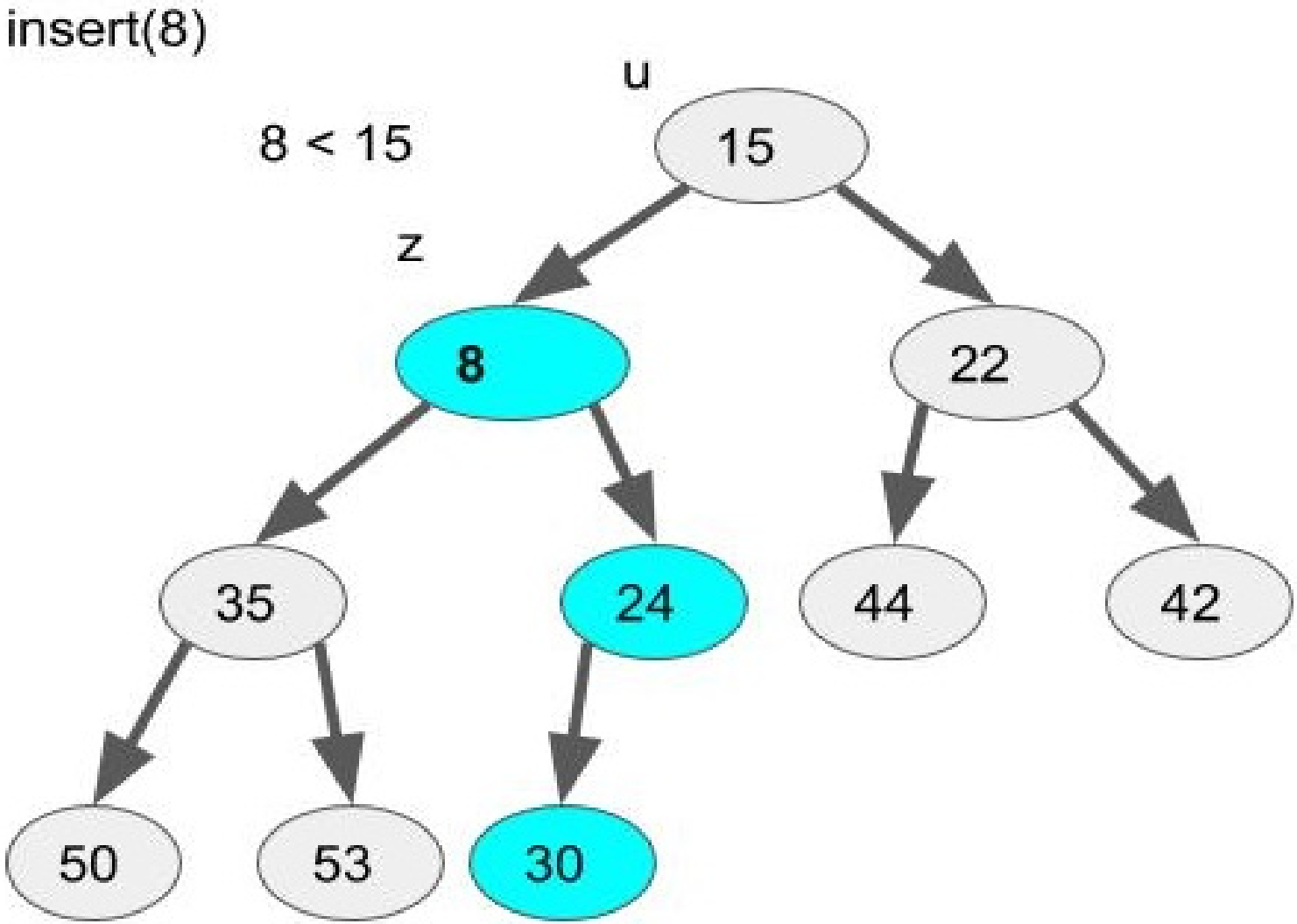
Implementación con heaps

Ejemplo de inserción



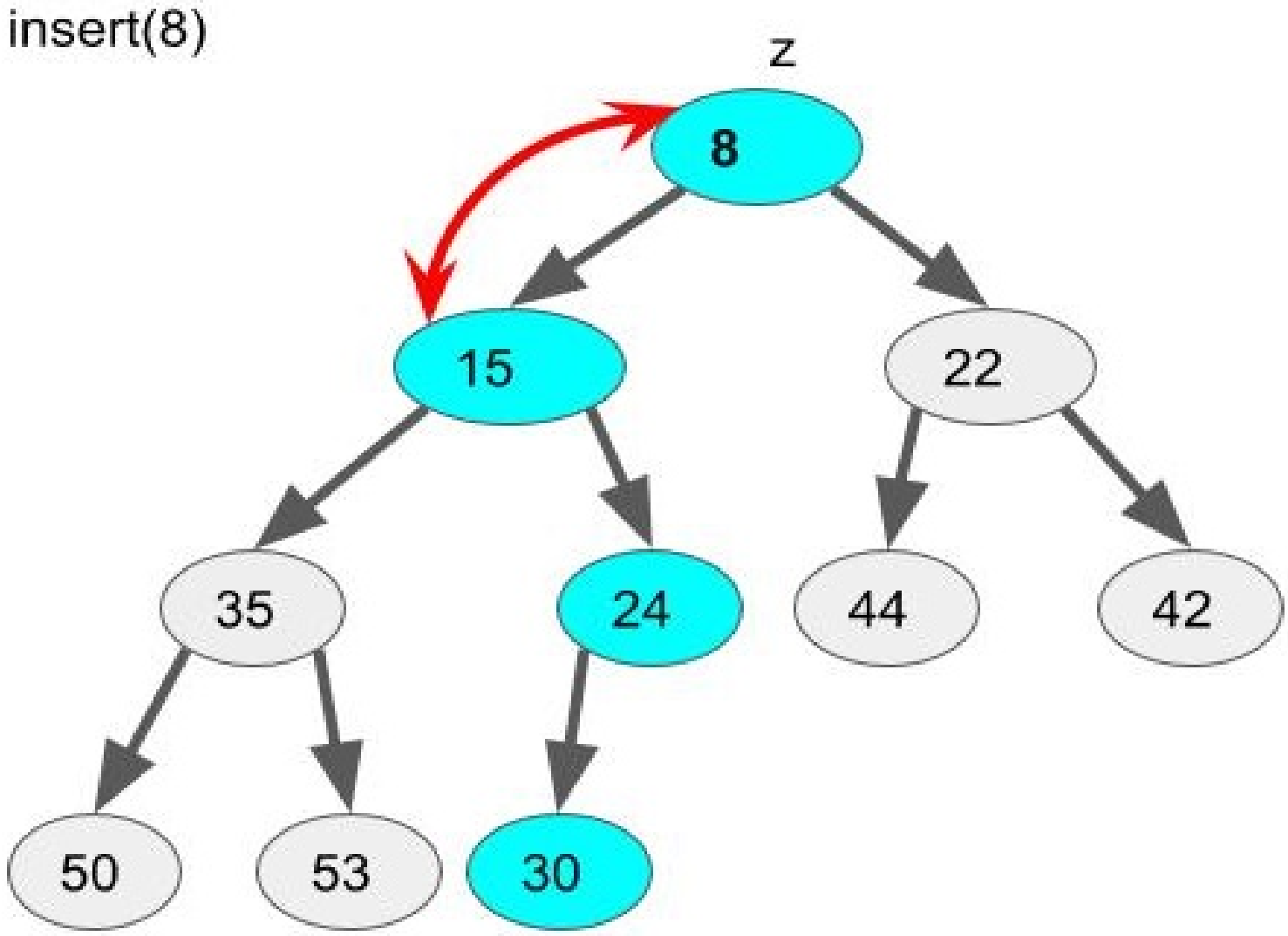
Implementación con heaps

Ejemplo de inserción



Implementación con heaps

Ejemplo de inserción



Implementación con heaps

Propiedad local de orden 2 para heap mínimo

Para cada nodo z, la clave de la entrada de z es menor o igual a las claves de todos sus hijos.

Eliminación

Se respalda la llave de la raíz, se copia el contenido del nodo en la máxima posición y se elimina éste último.

El árbol es binario completo pero podría no cumplir la propiedad local de orden.

Si la cola tiene a lo más un nodo después de la eliminación, teminamos, de otro modo, se efectúa el proceso de Burbujeo hacia abajo (*Down heap*)

Burbujeo hacia abajo

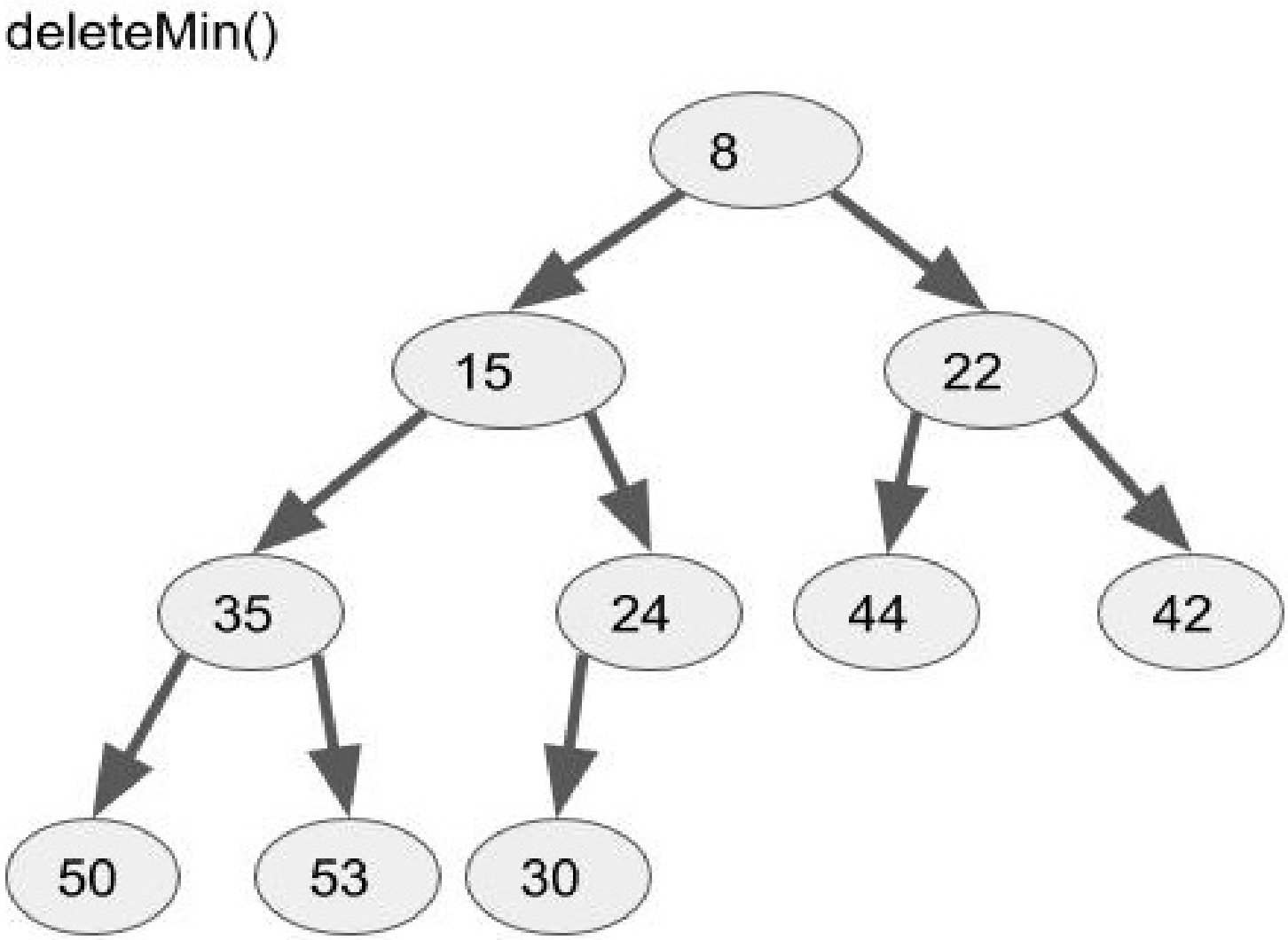
- Supongamos que r es la raiz y distinguimos dos casos:
  - Si r no tiene hijo derecho, sea s el hijo izquierdo de r.
  - De otro modo, r tiene ambos hijos y tomamos como s al hijo de r con la entrada de clave mínima.
- Comparamos las claves de r y s
  - Si la clave de r es menor o igual a la de s, terminamos
  - De lo contrario, intercambiamos las entradas en los nodos r y s y hacemos $r = s$

El proceso continua hasta que se cumpla la propiedad de orden local 2 o se alcance un nodo hoja.

Como se hace un número de intercambios proporcional a la altura del heap, que es $\lfloor log_2 n \rfloor$, donde n es el número de nodos, el peor caso es $O(\log n)$.

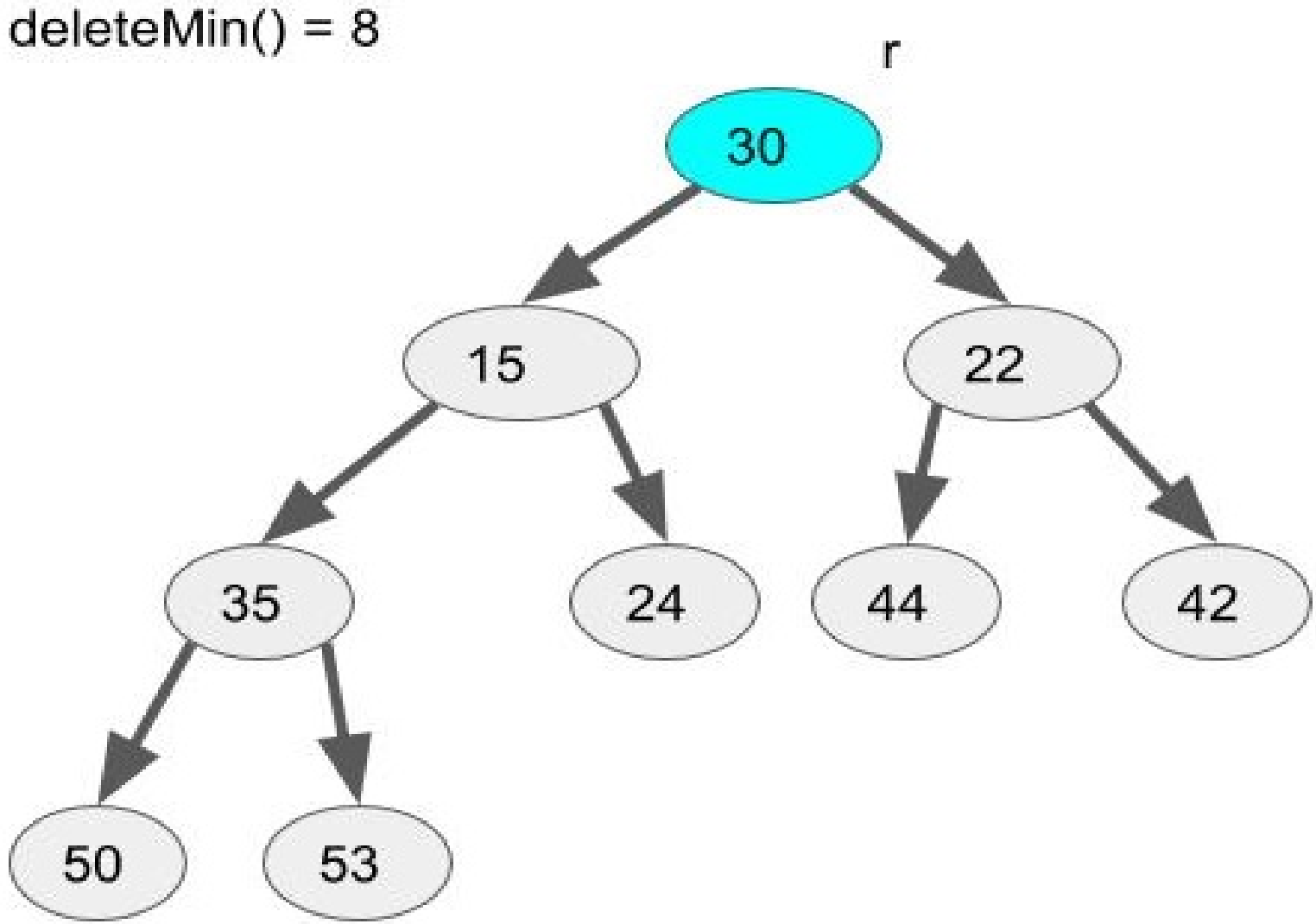
Implementación con heaps

Ejemplo de eliminación la entrada con clave mínima



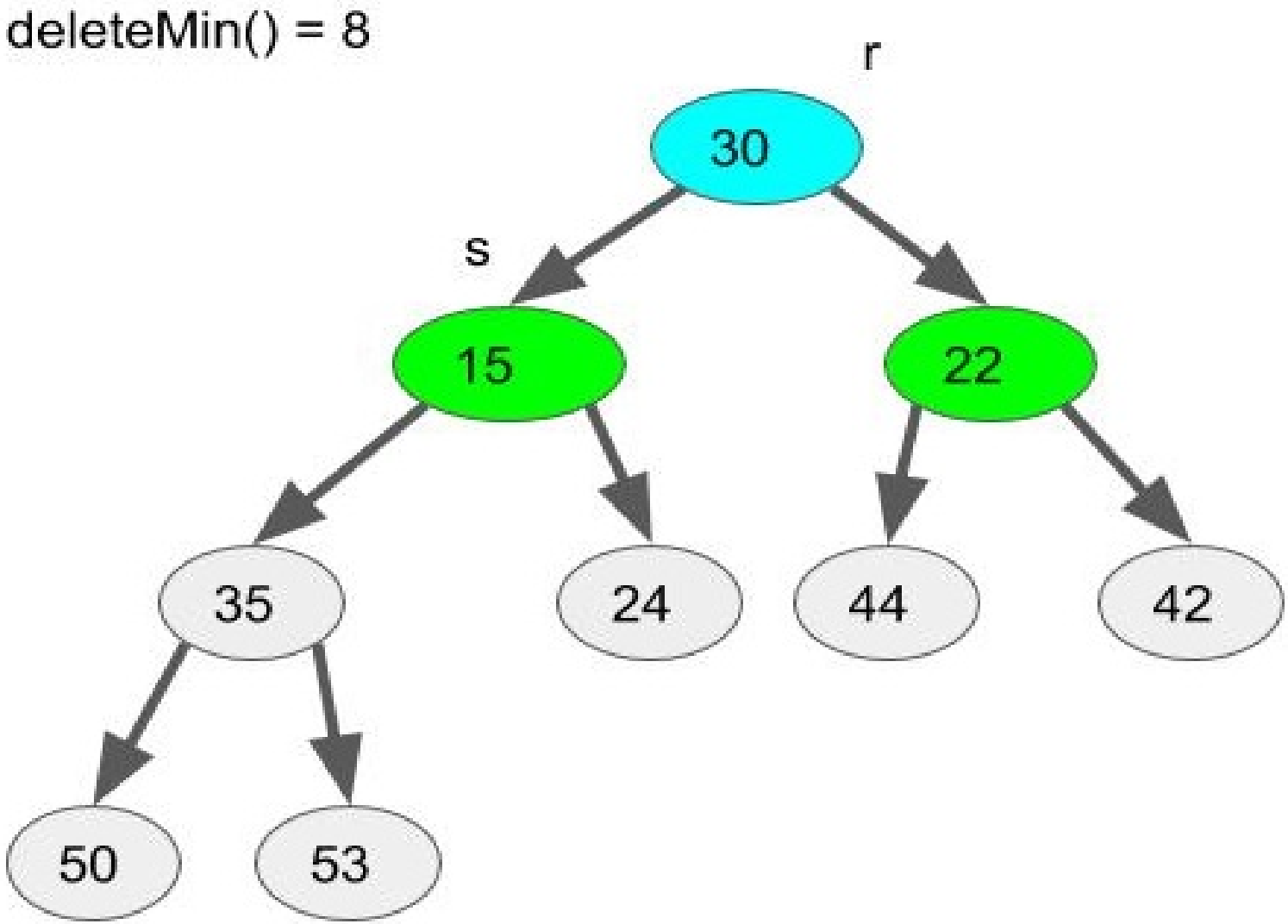
Implementación con heaps

Ejemplo de eliminación la entrada con clave mínima



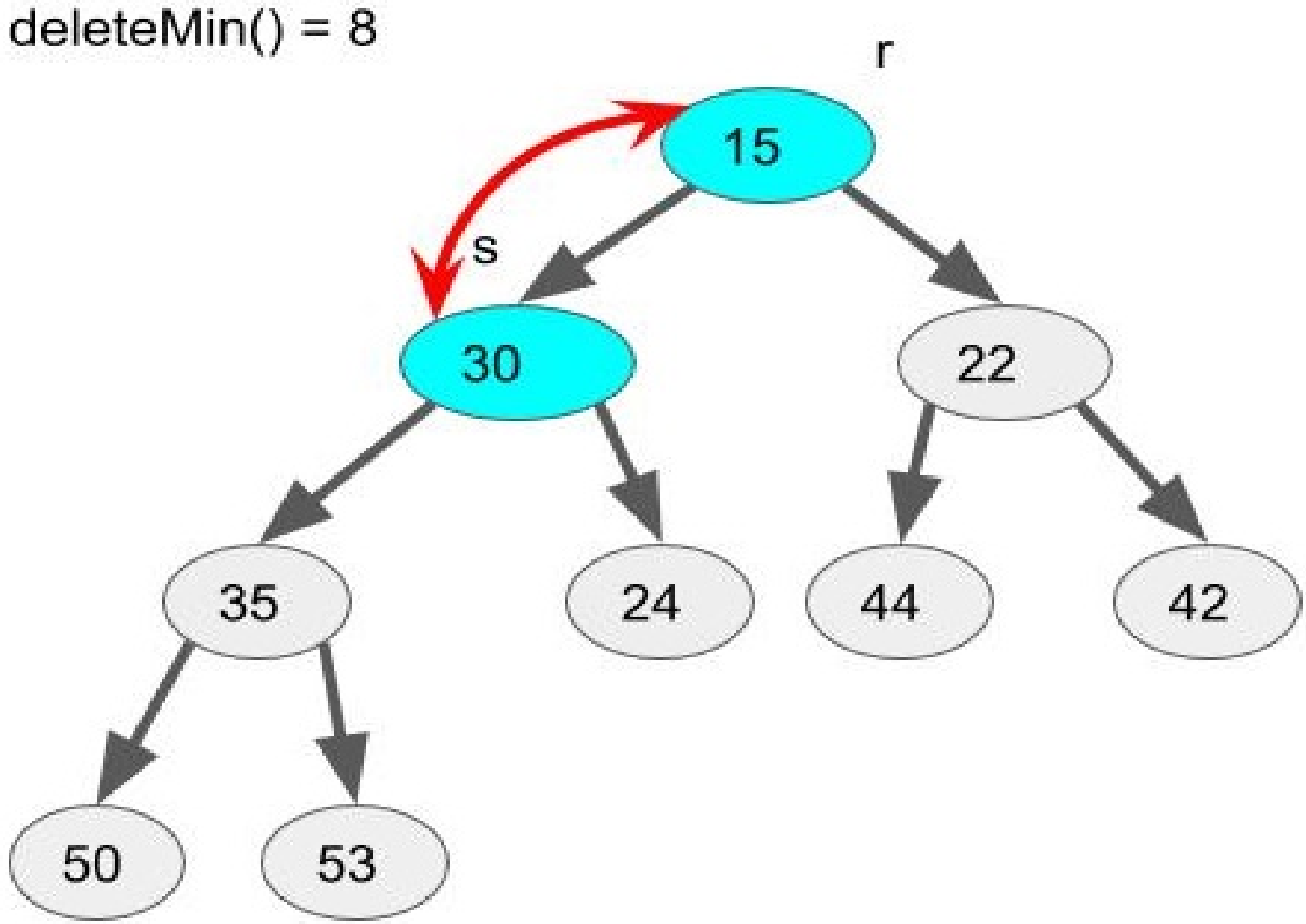
Implementación con heaps

Ejemplo de eliminación la entrada con clave mínima



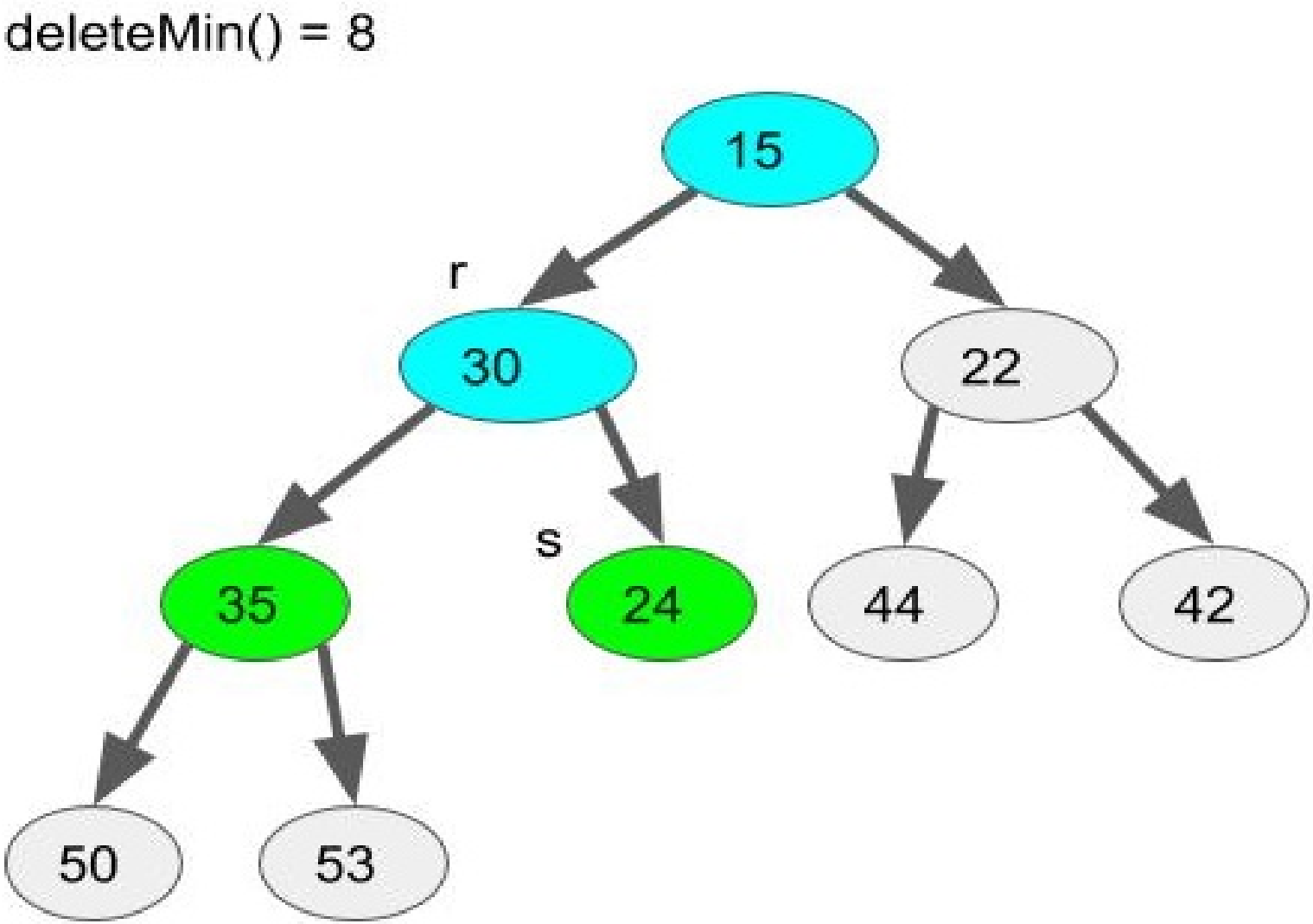
Implementación con heaps

Ejemplo de eliminación la entrada con clave mínima



Implementación con heaps

Ejemplo de eliminación la entrada con clave mínima



Implementación con heaps

Ejemplo de eliminación la entrada con clave mínima