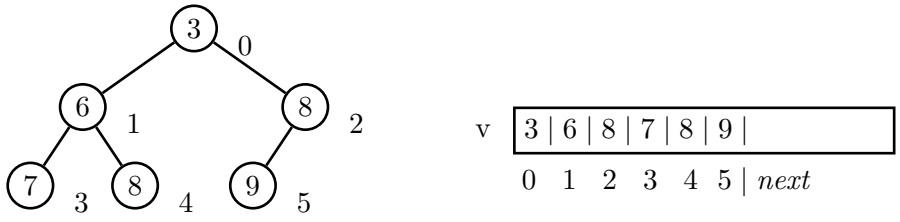

Figura 3.1: Ejemplo de montículo de Williams.

representación. El constructor anónimo crea un montículo vacío, con capacidad para un tamaño `size` que recibe como argumento.

En base a esta implementación, la creación de un montículo vacío y la operación *min*, que simplemente devolvería $v[0]$, tienen coste constante. La operación *insert* sigue la siguiente secuencia:

- Si $next < size$, añade el nuevo elemento en la posición *next* del vector, posición que corresponde a la hoja más a la derecha posible del último nivel del árbol.
- Incrementa en uno el valor de *next*. De este modo, el árbol conserva la propiedad de ser casi completo.
- Para preservar también la propiedad de montículo, se hace *flotar* el elemento, intercambiándolo repetidamente con su padre hasta conseguir que, o bien el elemento sea mayor o igual que su padre, o bien el elemento se convierta en la raíz.

Si n es el número de elementos del montículo, este proceso puede llevarse a cabo en un tiempo en $\Theta(\log n)$, ya que el acceso al padre se realiza en tiempo constante y, en el peor caso, ha de recorrerse el camino completo desde la hoja a la raíz. La implementación en Dafny de ambas operaciones `insert` y `float` se muestra en la figura 3.3.

El invariante del bucle **while** de `float` expresa que el predicado `isHeap` se satisface para todos los elementos del vector, excepto quizás para la relación entre $v[j]$ y su padre, propiedad que es un debilitamiento de la postcondición. Al terminar el bucle, o bien la relación entre $v[j]$ y su padre es la correcta, o bien $v[j]$ no tiene padre.

La operación *deleteMin* ha de suprimir la raíz del montículo. Como resultado, se obtienen dos montículos. Para unirlos de nuevo, se realiza la siguiente secuencia de operaciones: