Instituto Politécnico Nacional

Escuela Superior de Cómputo

Estructuras de Datos

Tema 13: Montículo (Heap)

M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom edgardoadrianfrancom

Contenido

- *Introducción*
- *Heap o Montículo*
  - *Altura*
  - *Prioridades*
  - *Inserción*
  - *Borrar*
  - *Actualizar*
- *Ordenamiento por montículo*
- *Implementación de vector como montículo*
- *Unión de dos montículos*
- *Construcción de montículo ascendente*

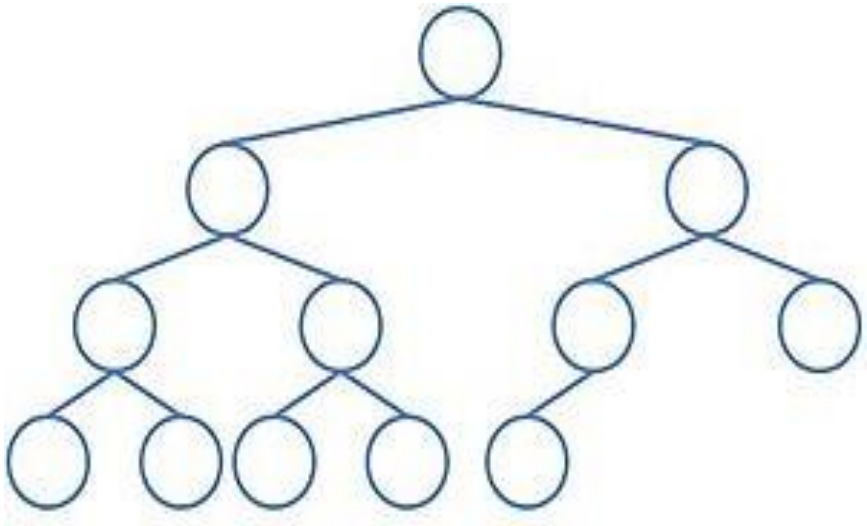
Introducción

- El montículo es una estructura de datos que soporta sólamente 3 operaciones: **conocer la cima** del montículo y **eliminar el elemento de la cima** y **agregar un nuevo elemento**.
- Puede sonar limitante el hecho de que no se puede saber si un elemento está o no está dentro de los datos guardados y muchas veces sí lo es, pero hay algunos problemas donde lo único que se requiere es saber cuál es el elemento mayor, o en otras ocaciones se requiere conocer el menor; por tanto los montículos a veces funcionan mejor que los árboles binarios de búsqueda pero no constituyen un reemplazo.

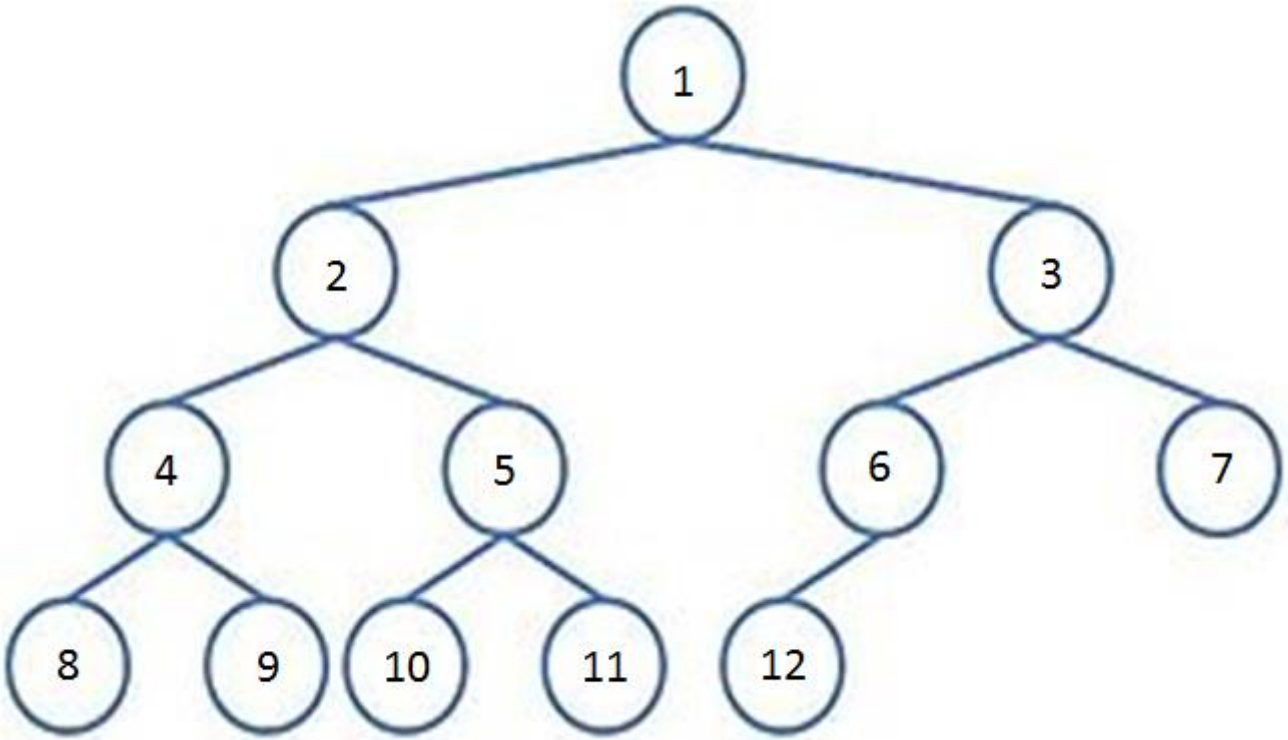
Heap o Montículo

- Un **heap** o **montículo** es un árbol binario completo, y además parcialmente ordenado.

- **Completo:** que tiene todos sus niveles completos a excepción del último. Y el último nivel contiene los nodos agrupados de izquierda a derecha

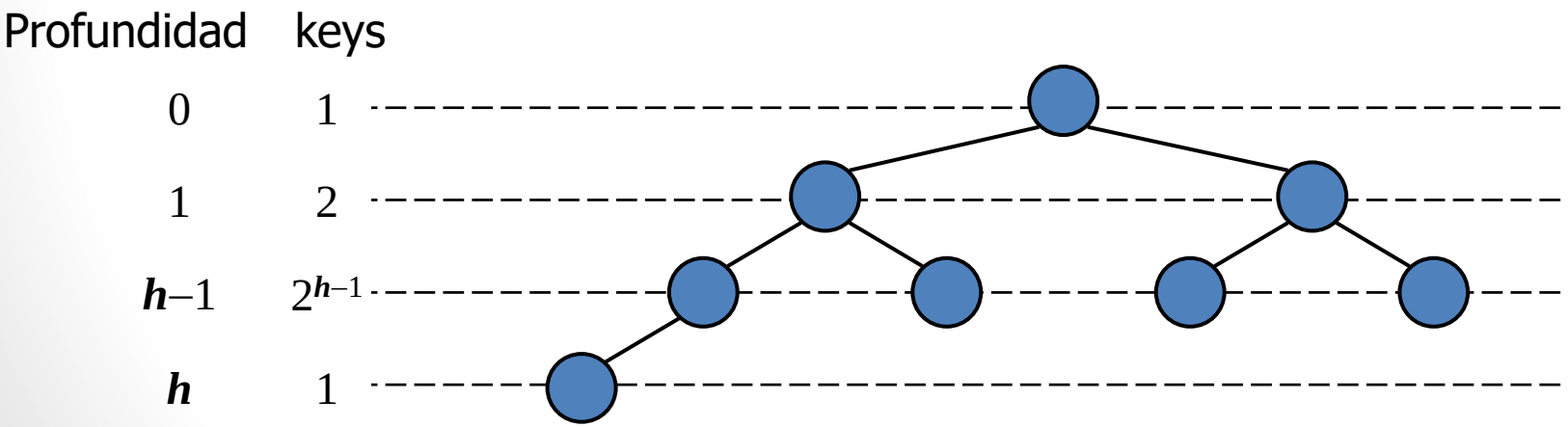


- **Parcialmente ordenado:** tiene todas y cada una de sus ramas, consideradas como listas, totalmente ordenadas, ya sea de forma creciente o decreciente



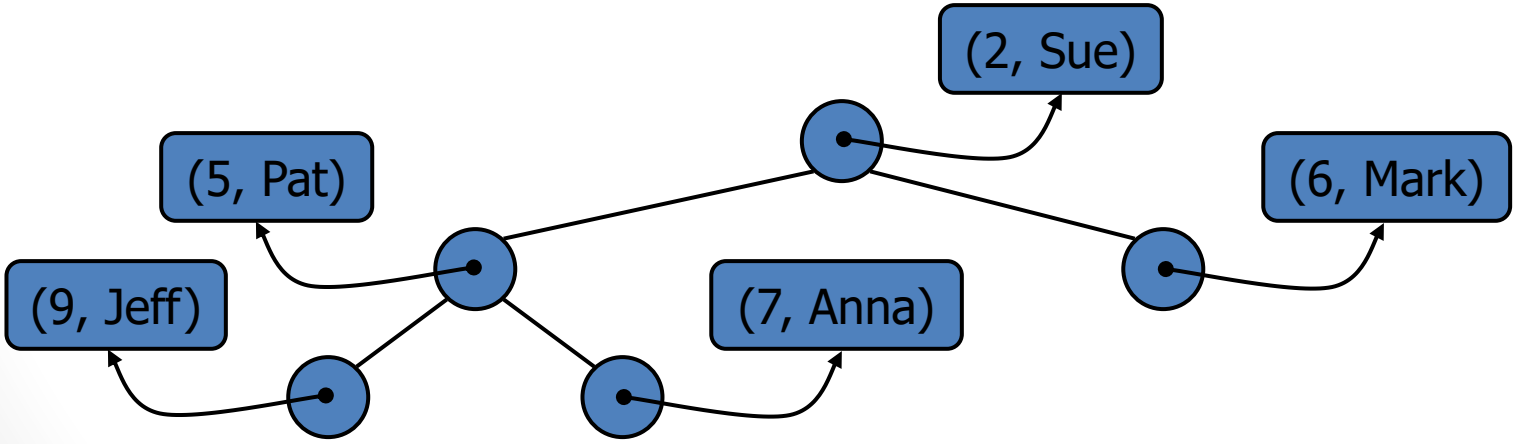
Altura del Montículo

- Un montículo que almacena n **keys** tiene una altura O (log n).
- h será la altura de un montículo que almacena n **keys**.
  - Dado que hay 2 keys de profundidad i = 0, ..., h - 1 y al menos una **key** a profundidad h, tenemos $\boldsymbol{n} \geq 1 + 2 + 4 + \ldots + 2^{\boldsymbol{h}-1} + 1$
- Por lo tanto, $\boldsymbol{n} \geq 2^{\boldsymbol{h}}$, i.e., $\boldsymbol{h} \leq \log \boldsymbol{n}$



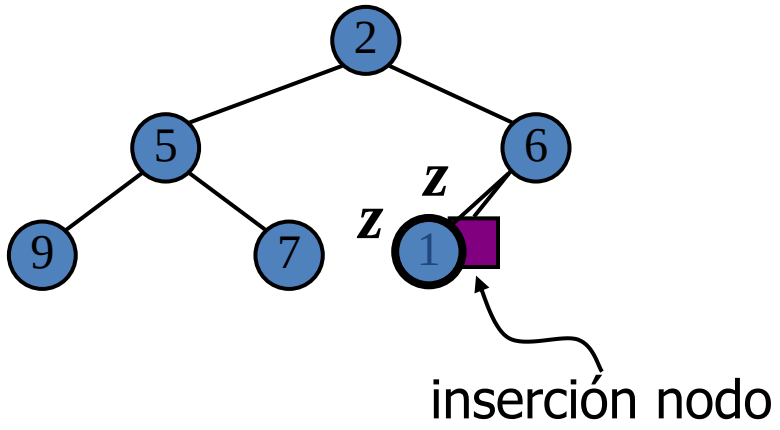
Montículos y colas con prioridad

- Podemos usar un montículo para implementar una cola de prioridad.
- Almacenamos un elemento (clave, elemento) en cada nodo interno.
- Hacemos un seguimiento de la posición del último nodo
- Para simplificar, mostramos solo las keys en las imágenes



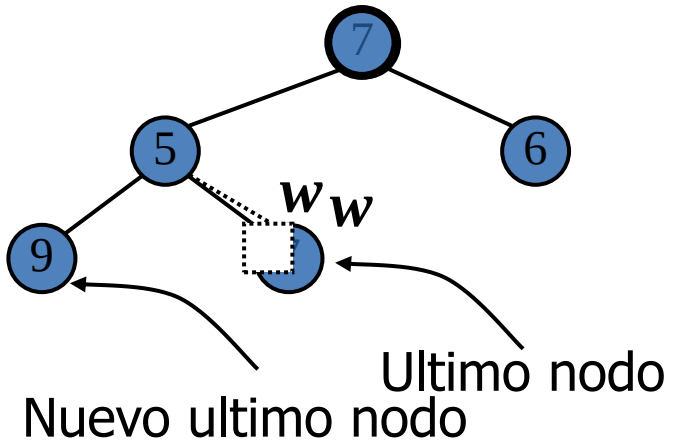
Inserción

- Método insertItem de la cola de prioridad ADT corresponde a la inserción de una clave k en el montículo.
- El algoritmo de inserción consta de tres pasos:
  - Encuentre el nodo de inserción z (en el último nodo)
  - Almacene k en z
  - Restaurar la propiedad de orden acumular.



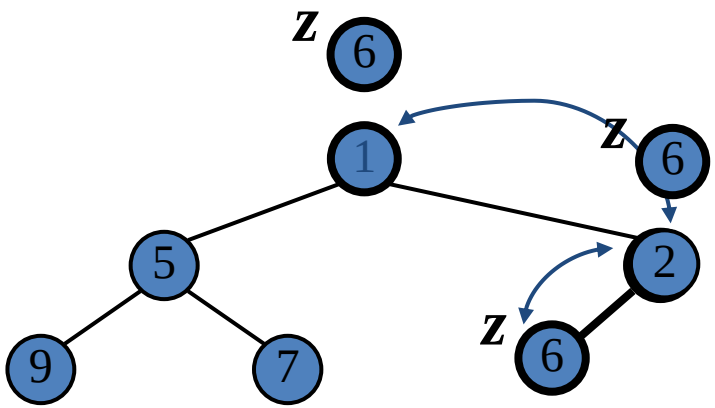
Remover

- Método removeMin de la cola de prioridad ADT corresponde a la eliminación de la clave raíz del montículo.
- El algoritmo de eliminación consta de tres paso:
  - Reemplace la key raíz con la key del último nodo w.
  - Eliminar w.
  - Restaurar la propiedad de orden acumular.



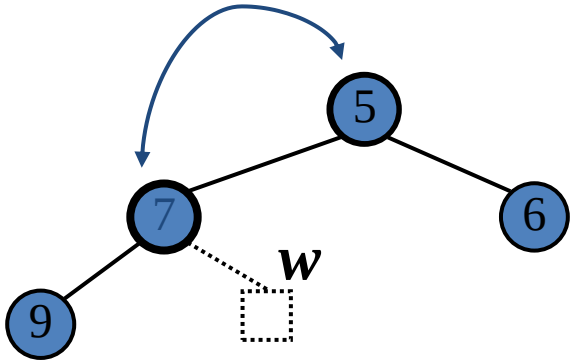
Upheap

- Después de la inserción de una nueva key k, la propiedad de orden acumular puede ser violada
- El algoritmo upheap restaura la propiedad de orden de pila al intercambiar k a lo largo de una ruta ascendente desde el nodo de inserción
- Upheap termina cuando la key k alcanza la raíz o un nodo cuyo padre tiene una clave menor o igual a k
- Como un montículo tiene una altura O (log n), upheap se ejecuta en tiempo O (log n)



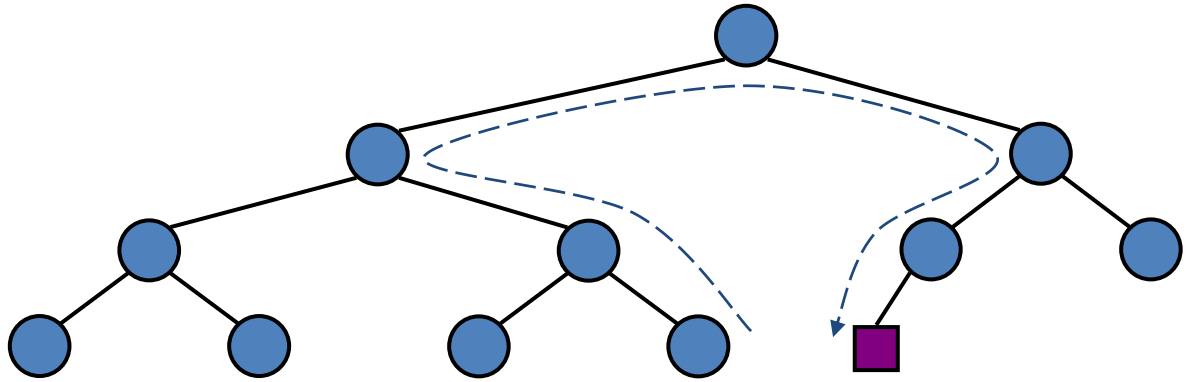
Downheap

- Después de reemplazar la key raíz con la key k del último nodo, la propiedad de orden de pila puede ser violada.
- El algoritmo downheap restaura la propiedad de orden de pila al intercambiar la key k a lo largo de una ruta descendente desde la raíz
- Upheap termina cuando la key k alcanza una hoja o un nodo cuyos hijos tienen claves mayores o iguales que k
- Como un montículo tiene una altura O (log n), downheap se ejecuta en O (log n)



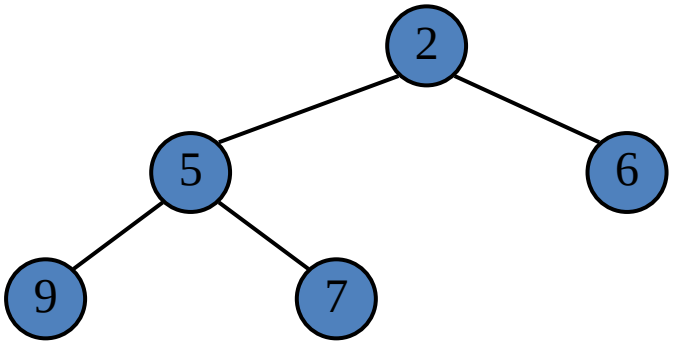
Actualizar ultimo nodo

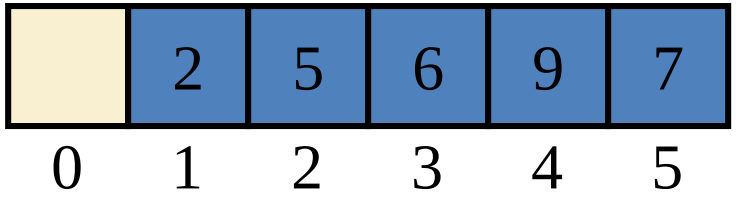
- El nodo de inserción se puede encontrar atravesando una ruta de nodos O (log n)
  - Subir hasta que se alcanza un hijo izquierdo o la raíz
  - Si se llega a un hijo de la izquierda, vaya al hijo de la derecha
  - Bajar a la izquierda hasta que se alcanza una hoja
- Algoritmo similar para actualizar el último nodo después de una eliminación



Implementación de un Vector en un montículo

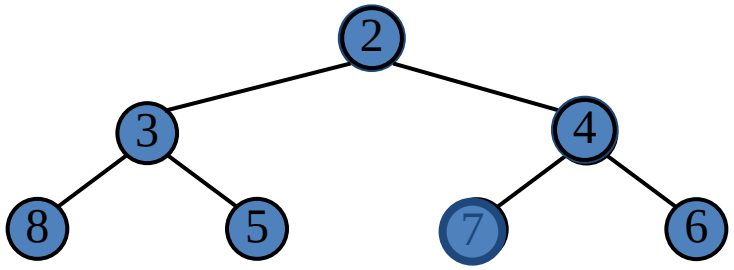
- Podemos representar un montículo con n keys mediante un vector de longitud n + 1
- Para el nodo en el rango i:
  - el hijo izquierdo está en el rango 2i
  - el hijo correcto está en el rango 2i + 1
- Los enlaces entre nodos no se almacenan explícitamente
- La celda del rango 0 no se usa
- La operación insertar corresponde a insertar en el rango n + 1
- La operación remover corresponde a eliminar en el rango n





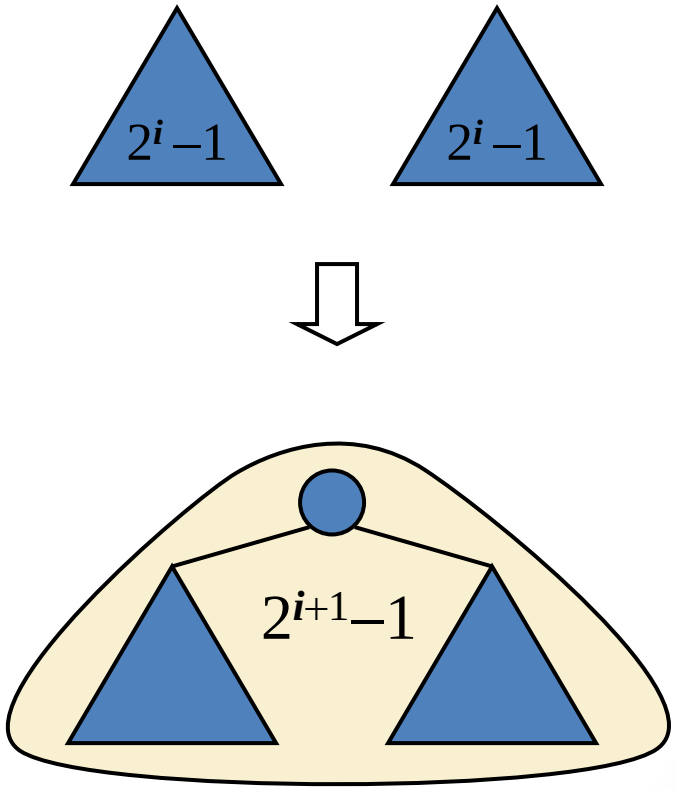
Combinar dos montículos

- Nos dan dos montones y una key k
- Creamos un nuevo montón con el nodo raíz que almacena k y con los dos montones como subárboles
- Realizamos un downheap para restaurar la propiedad de orden de montículo

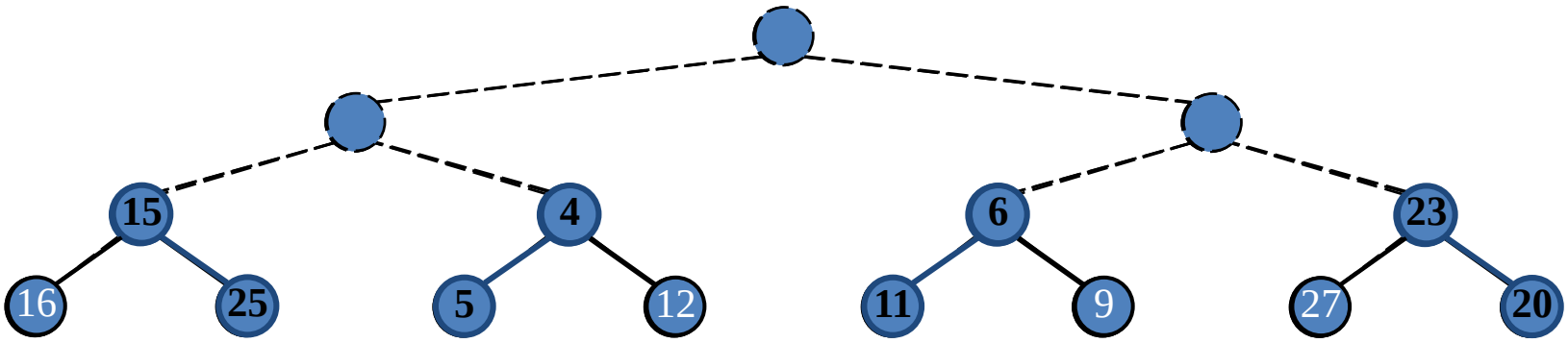


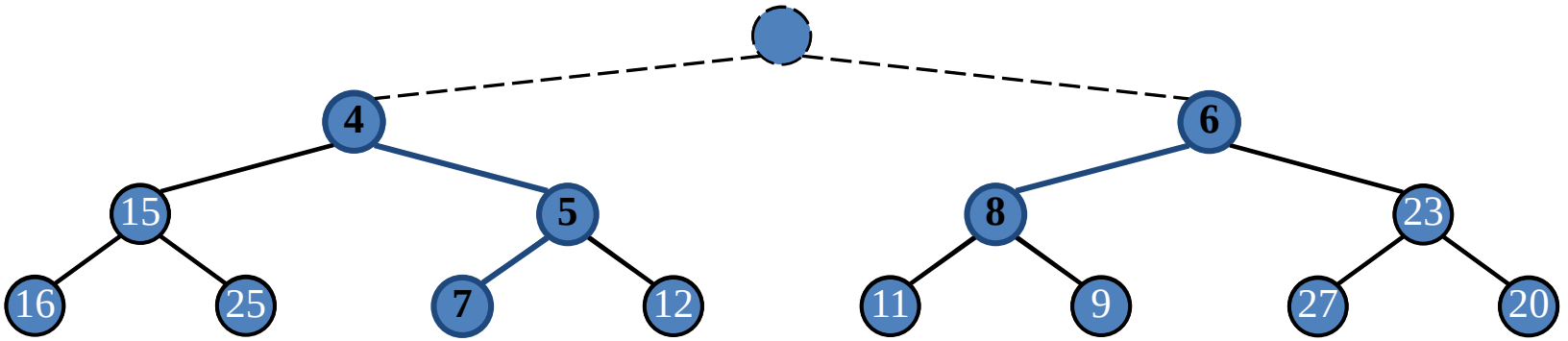
Construcción de montículo Bottom-up

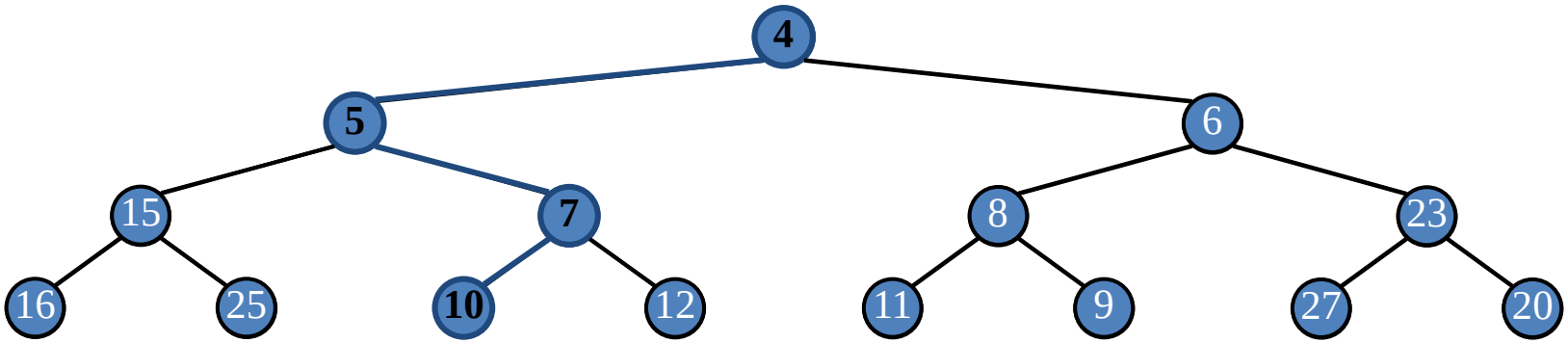
- Podemos construir un montículo y almacenar n keys dadas en el uso de una construcción de abajo hacia arriba con log n fases
- En la fase ***i***, los pares de montones con $2^i - 1$ keys se combinan en montones con $2^{i+1} - 1$ keys



Ejemplo




4
6
15
5
8
23
16
25
7
12
11
9
27
20


4
5
6
15
7
8
23
16
25
10
12
11
9
27
20

Análisis

- Visualizamos el peor de los casos de un downheap con una ruta que va primero a la derecha y luego va repetidamente a la izquierda hasta la parte inferior del montículo (esta ruta puede diferir de la ruta real downheap)
- Como cada nodo está atravesado como máximo por dos rutas, el número total de nodos de las rutas es **O (n)**
- Por lo tanto, la construcción del montículo de abajo hacia arriba se ejecuta en el tiempo **O (n)**
- La construcción del montículo ascendente es más rápida que n inserciones sucesivas y acelera la primera fase del ordenamiento del montículo.