Heap Binomial

Prof. Rhadamés Carmona

Última actualización: 09/05/0291

Agenda

- Motivación
- Concepto
- Estructura de datos
- Operaciones
- Comparación con binary heap

Motivación

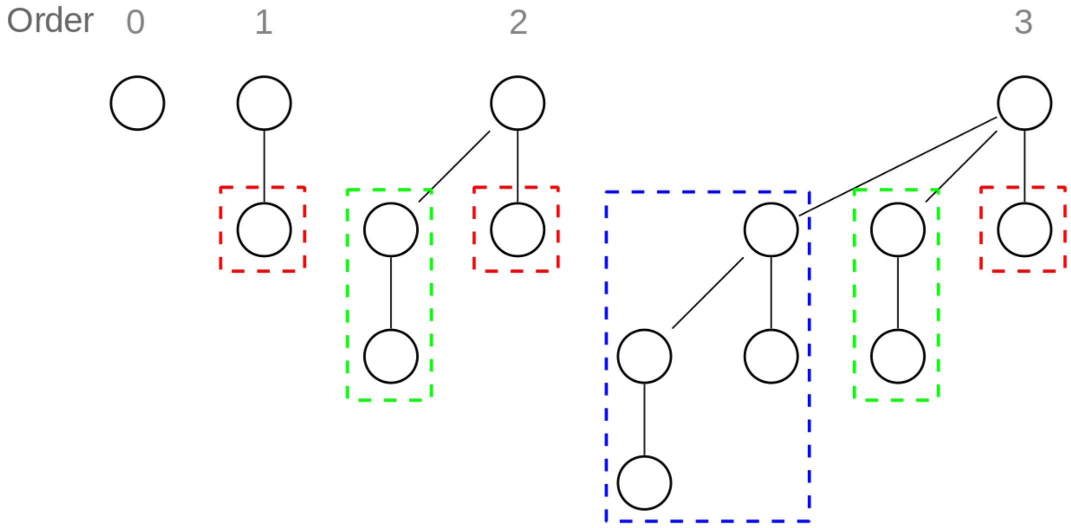
- Las operaciones del heap son eficientes: consultar el menor O(1), extraer el menor O(log*n*), insertar O(log*n*). Y dado el apuntador a un nodo, reducir clave O(log*n*), y eliminar O(log*n*).
- Pero, la unión de dos heaps es O(*n*), y **puede ser reducido a O(log*n*)** si usamos un heap binomial.

Concepto

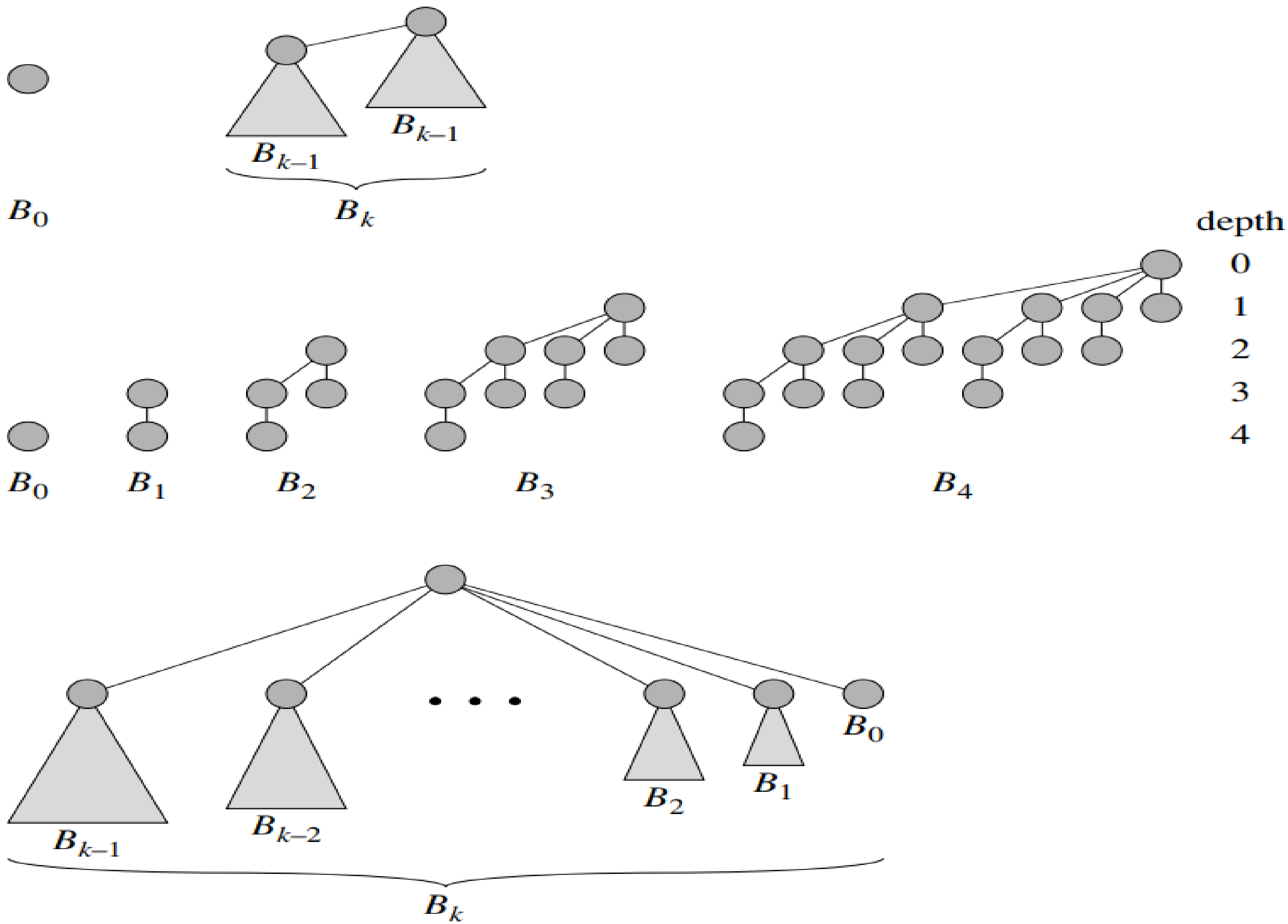
- El heap bonimial es una estructura de datos utilizada para implementar una cola de prioridad. Es una extensión del heap, que es más rápido para operaciones de unión.
- Un Heap Binomial es una colección de Árboles Binomiales.

Concepto

- Un árbol binomial de orden 0 tiene 1 nodo
- De orden *k* puede construirse con dos árboles binomiales de orden *k*-1, y haciendo que uno de ellos sea el hijo más izquierdo del otro.



Concepto



Concepto

- Un ábol binomial tiene las siguientes propiedades:
  - Tiene 2^k nodos
  - Altura k
  - Hay C(k,i) nodos a profundidad i=0..k (ver triángulo de pascal y #nodos por nivel)
  - La raíz tiene grado k, y es el nodo de mayor grado
  - El grado de dicho nodo de log(*n*)

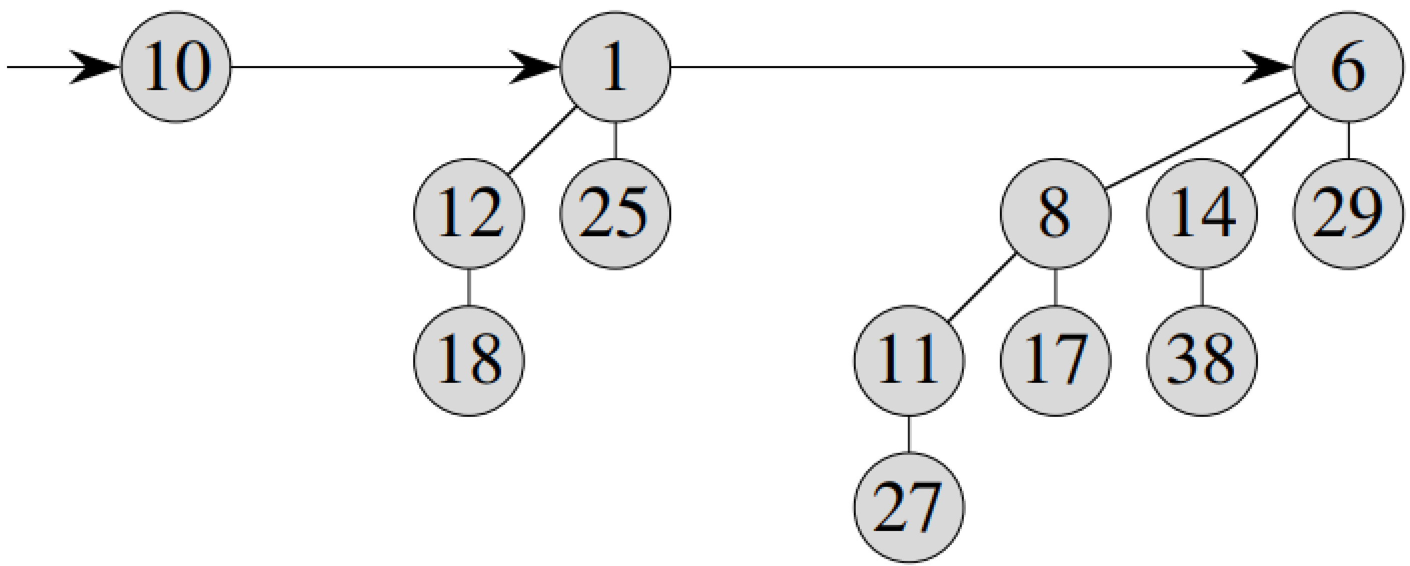
Concepto

- Un heap binomial H es un conjunto de árboles binomiales que satisfacen las siguientes propiedades
  - Cada árbol binomial en H obedece la propiedad de min-heap: la clave de un nodo es >= a la de su padre.
  - Para n claves, hay a lo sumo $\lfloor \log n \rfloor + 1$ árboles binomiales en H, todos de distintos grados.

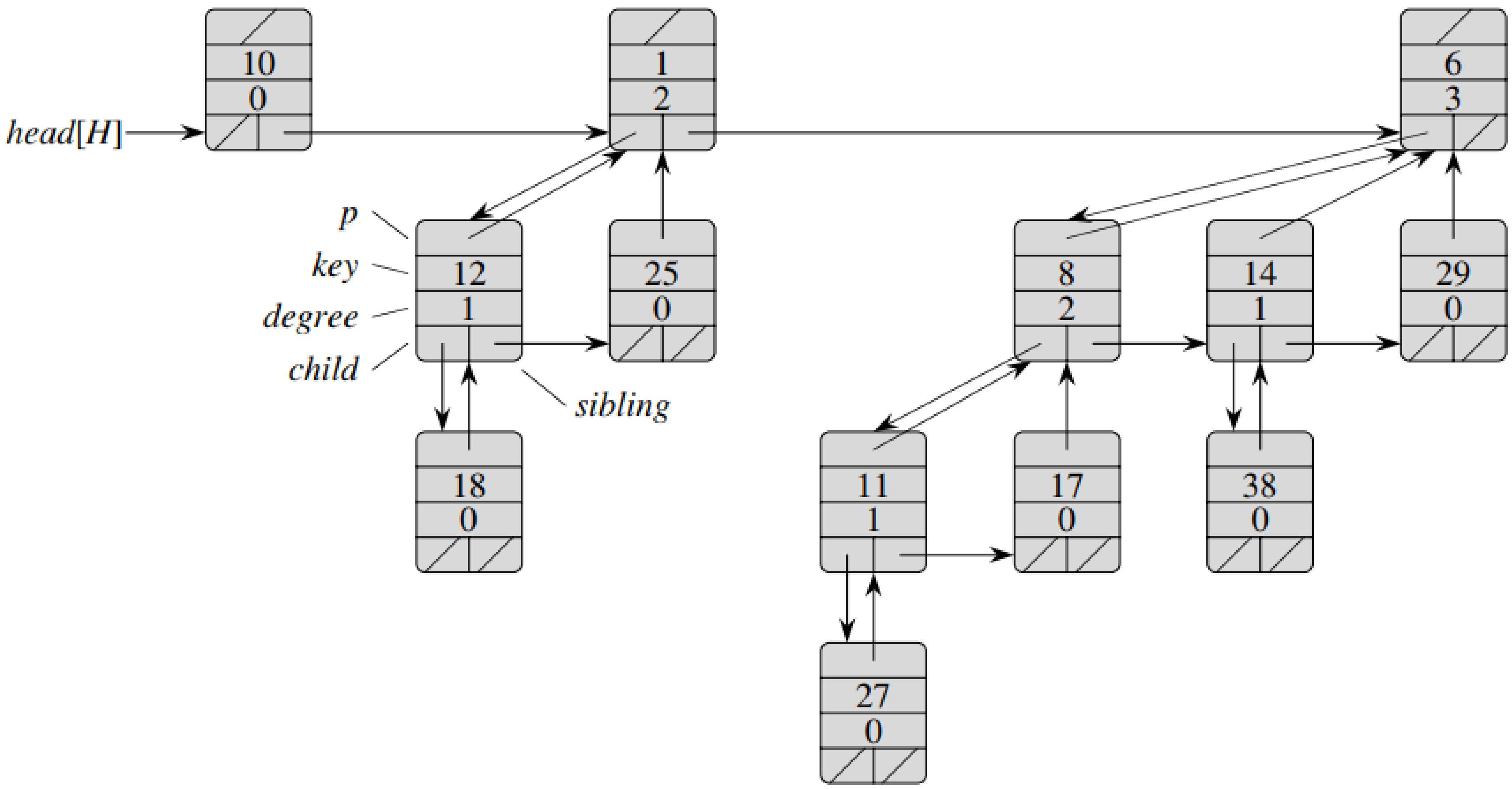
Concepto

- Ejemplo con *n*=13

árboles ordenados según el grado (izquierda a derecha)



Estructura de datos

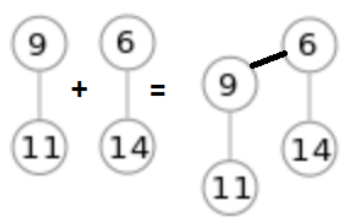


Estructura de datos

- Dibujar en la pizarra un heap binomial con n nodos, variando *n*=1...16. Now!.
- Note que la lo sumo hay $\lfloor \log n \rfloor + 1$ árboles binomiales.

Operaciones (crear, mínimo, unión)

- Crear binomial heap: simplemente colocar head = NULL, O(1).
- Encontrar mínimo: recorremos la lista de raíces, buscando el mínimo. Como hay a lo sumo $\lfloor \log n \rfloor + 1$ raíces, esto es O(log$n$).
- Unión: une pares de árboles binomiales con el mismo grado.



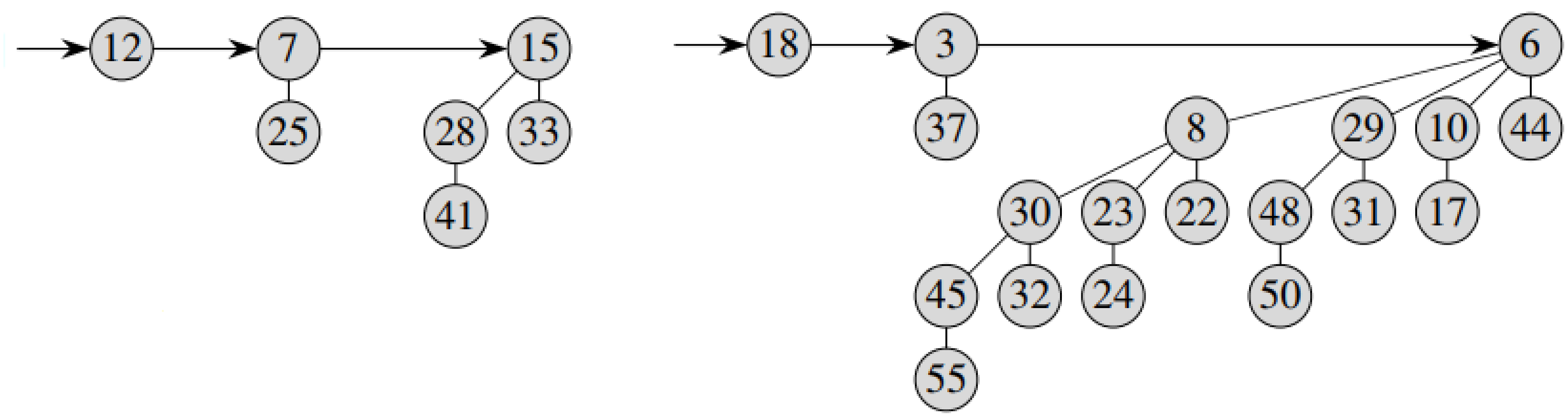

```
function mergeTree(p, q)
    if p.root.key <= q.root.key
        return p.addSubTree(q)
    else
        return q.addSubTree(p)
```

Operaciones (unión)

- (1) Las 2 listas de heaps son recorridas secuencialmente, de manera similar a la mezcla ordenada de 2 listas de O(log*n*) entradas.
- (2) Si solo hay un árbol de grado *k*, este pasa intacto al head binomial resultante (a menos que ya haya uno de grado *k* en el resultante formado por 2 de grado *k*-1, en cuyo caso se mezclarían para formar uno de grado *k*+1).
- (3) Si hay dos árboles de grado k, se fusionan en uno de grado *k*+1 en O(1) usando mergeTree.
- (4) En algún paso, es posible encontrar solo un árbol de grado *k*+1 que deberá fusionarse con el obtenido en el paso (3) para formar uno de grado *k*+2. Este fenómeno podría repetirse en posteriores pasos de mezcla.
- Por eficiencia, dejar la unión de A U B sobre la estructura de A.

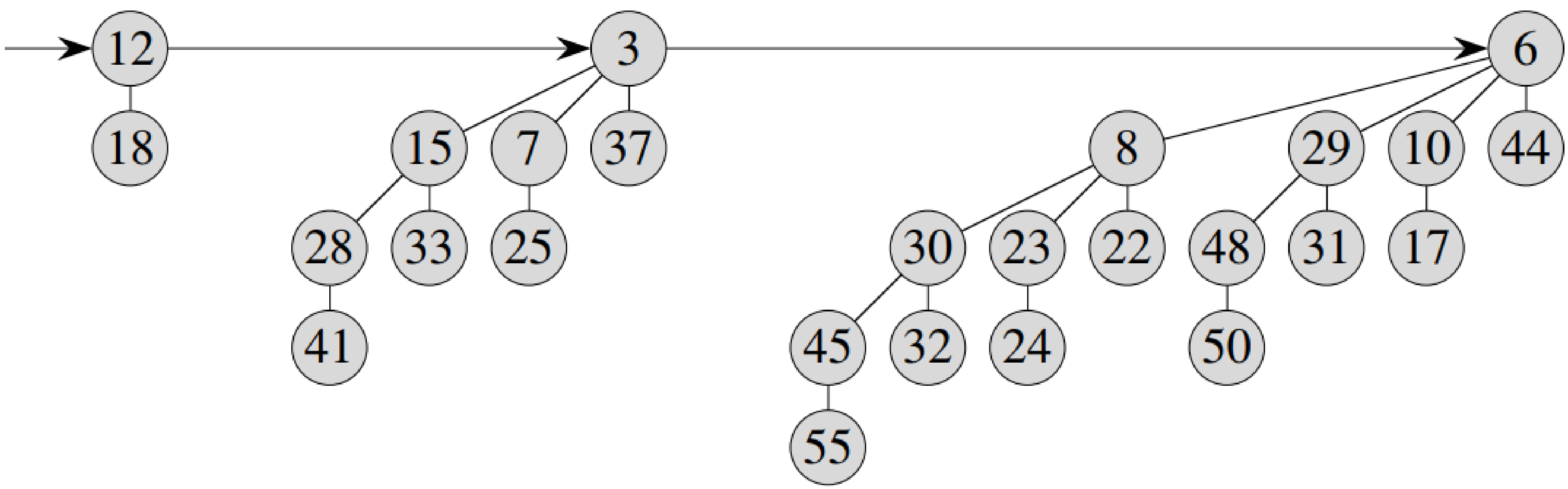
Operaciones (unión)

- Ejercicio: Realizar la unión de estos 2 heaps binomiales



Operaciones (unión)

- Resultado:



Operaciones (insertar)

- La solución trivial (dado que contamos con unión) es crear un heap binomial en O(1) con un árbol binomial que contenga solo el nodo a insertar, e invocar a unión, resultando O(log*n*).
- El mejor caso es que no haya un árbol binomial de grado 0.
- El peor caso es que sí lo haya, pero que también hayan árboles binomiales de todas las potencias de 2. De allí el O(log*n*).
- Con *n* inserciones, es O*(1) amortizado; ese peor caso solo puede pasar 1 vez en *n*.

Operaciones (decrementar key)

- Suponiendo que estamos en el nodo x, decrementamos su clave (x->key--) y aplicamos la operación “flotar” de ser necesario, similar a como se haría en un minheap. En el peor de los casos, el nodo x es una hoja, y debe flotar hasta la raíz, lo cual es $O(\log n)$.

Operaciones (eliminar mínimo)

- Primero buscamos el mínimo entre las raíces de los árboles binomiales, en O(log*n*).
- Creamos un nuevo heap binomial, moviendo los hijos del nodo removido, en O(log*n*).
- Llamamos a unión de los dos heaps binomiales, que también es O(log*n*).

Comparación con Heap

Operación	Binary Heap	Binomial Heap
Insertar	**O(log)**	**O(log), O*(1)**
Mínimo	**O(1)**	**O(log)**
Eliminar mínimo	O(log)	O(log)
Eliminar nodo	O(log)	O(log)
Union	**O(n)**	**O(log)**
Reducir clave	O(log)	O(log)
Crear	O(1)	O(1)

Ideas finales

- Hay otro tipo de heap llamado Heap de Fibonacci, donde la mayoría de las operaciones son O*(1) amortizado.
- Solo borrar el mínimo sería O(log).