Heaps

Algoritmos y Estructuras de Datos II

1^{er} cuatrimestre de 2019

Motivación

Implementar una cola de prioridad. Operaciones que nos interesan:

Máximo	Determinar el elemento más prioritario.
Agregar	Agregar un elemento.
Sacar máximo	Sacar el elemento más prioritario.
Conj→cola	Convertir un conjunto en una cola de prioridad.

La prioridad puede ser cualquier **relación de orden total**.
(¿Qué era una relación de orden total?).

Hablamos siempre del **máximo**.

Posibles implementaciones (sin usar heap)

	Máximo	**Agregar**	**Sacar máximo**	**Conj→cola**
Lista	?	?	?	?
Lista + máximo	?	?	?	?
Lista ordenada	?	?	?	?
AVL + máximo	?	?	?	?

Posibles implementaciones (sin usar heap)

	Máximo	**Agregar**	**Sacar máximo**	**Conj→cola**
Lista	$O(n)$	$O(1)$	$O(n)$	$O(n)$
Lista + máximo	$O(1)$	$O(1)$	$O(n)$	$O(n)$
Lista ordenada	$O(1)$	$O(n)$	$O(1)$	(sorting)
AVL + máximo	$O(1)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$

Spoiler

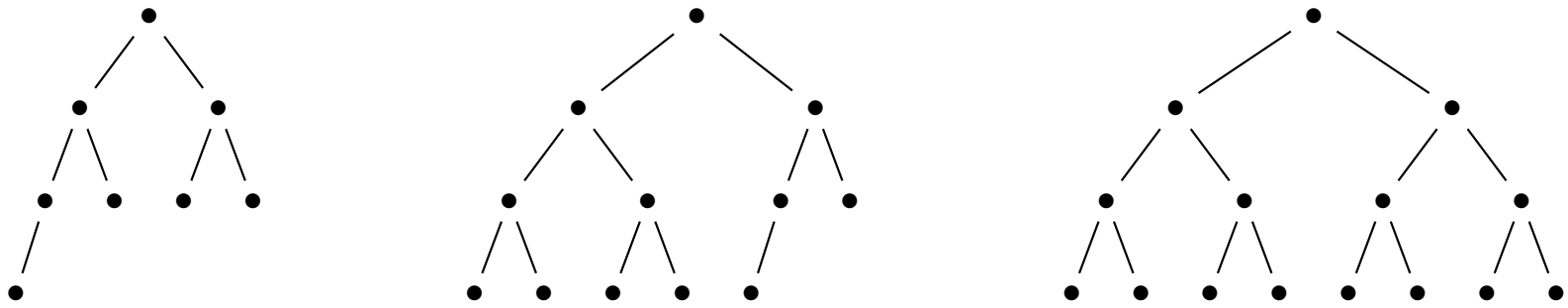
¿Para qué queremos un heap si podemos usar un AVL?

- Más sencillo de implementar.
- Mejores constantes.
- Se puede hacer sin punteros.
- La operación **Conj**$\rightarrow$**cola** es estrictamente mejor.

Heap: invariante de representación

Un heap es un árbol binario con un invariante:

- La raíz es el máximo.
- Completo, salvo por el último nivel.
- Izquierdista.



- El invariante se cumple recursivamente para los hijos.
- (yapa) Todos los caminos de la raíz a una hoja son secuencias ordenadas.

Heap: técnicas de implementación

Técnica de implementación con arreglos

Los elementos se pueden guardar en un arreglo de tamaño N.

Las siguientes funciones sirven para navegar el árbol:

$$\begin{aligned} \texttt{HIJO\_IZQ}(i) &= 2 * i + 1 \\ \texttt{HIJO\_DER}(i) &= 2 * i + 2 \\ \texttt{PADRE}(i) &= \lfloor \frac{i-1}{2} \rfloor \end{aligned}$$

Usando índices $0 \leq i < N$.

En nuestra estructura interna vamos a tener:

- **v** arreglo de tipo paramétrico T de tamaño N (T debe tener una relación de orden total)
- nat **N**

Heap: algoritmos

Máximo $O(1)$

- Está en la raíz del árbol (v[0])

Agregar $O(\log n)$

- Ubicar el elemento respetando la forma del heap.
- Mientras sea mayor que su padre, intercambiarlo con el padre. (*Sift up*).

siftUp (arreglo v, nat i) $O(\log n)$

- Si $i \neq 0$ y $v[i] > v[padre(i)]$
  - swap(v[i],v[padre(i)])
  - siftUp(v,padre(i))

Heap: algoritmos

Sacar máximo $O(\log n)$

- Reemplazar la raíz del árbol por el "último" elemento, respetando la forma del heap.
- Borrar el "último" elemento.
- Empezando desde la raíz, mientras sea menor que uno de sus hijos, intercambiarlo con el mayor de sus hijos. (*Sift down*).

siftDown (arreglo v, nat i) $O(\log n)$

- Si $v[i]$ no es hoja
  - m = max(v[hijo_izq(i)], v[hijo_der(i)])
  - Si $m > v[i]$
    - swap(v[i],m)
    - siftDown(v,índice(m))

OJO! qué pasa si no tiene hijo derecho?

previamente guardo el valor de índice(m)

Heap: algoritmos

Ejemplo: insertar en secuencia $6, 4, 2, 9, 3, 8, 5$ y sacar el máximo.

Heap: algoritmos

Conj→cola (*algoritmo simple*) $O(nlog(n))$

- Para $i = 0, \ldots, N$
  - agregar(v[i])

Conj→cola (*heapify - Algoritmo de Floyd*) $O(n)$

- Armar un árbol con los elementos respetando la forma.
- Para $i = \frac{N}{2}, \ldots, 0$
  - siftDown(v, i)

Heap: algoritmos

Ejemplo: *heapificar* la secuencia $6, 4, 2, 9, 3, 8, 5$.

Heap: otras técnicas de implementación

Técnica de implementación con punteros

Si el heap tiene n elementos, la posición del último se puede encontrar a partir de la representación en binario de n, ignorando el dígito 1 más significativo.

$n = 14 = (1110)_2 \quad \rightsquigarrow \quad$ [derecha, derecha, izquierda]