La sala de urgencias

- Al llegar a una sala de urgencia, la persona es evaluada
- Se le asigna un número del 1 al 5 según la urgencia
- Cuando se desocupa un box, se hace pasar al siguiente
- La persona que pasa es la más urgente que aún está en espera
- En caso de empate, pasa la persona que llegó primero

¿Qué tan caro es decidir la próxima persona que pasará?

Nivel	Tipo de urgencia	Color	Tiempo de espera
1	RESUCITACIÓN	ROJO	Atención de forma inmediata
2	EMERGENCIA	NARANJA	10 – 15 minutos
3	URGENCIA	AMARILLO	60 minutos
4	URGENCIA MENOR	VERDE	2 horas
5	SIN URGENCIA	AZUL	4 horas

La cola de prioridades

Una estructura de datos con las siguientes operaciones:

- Insertar un dato con cierta prioridad a la cola
- Extraer el dato con más prioridad de la cola

E idealmente:

- Cambiar la prioridad de un dato ya insertado

La cola de prioridades

Claramente hay que mantener cierto orden de los datos

¿Cuál es el costo de mantener **ordenados** los datos?

¿Y al llegar n datos nuevos?

La cola de prioridades

Solo necesitamos la información del más prioritario

Quizás podemos no tener un orden total de los datos

¡Necesitamos algún tipo de estructura interna!

La cola de prioridades

¿Que debe conocer la estructura en todo momento?

¿Será posible hacer una estructura recursiva?

El Heap Binario

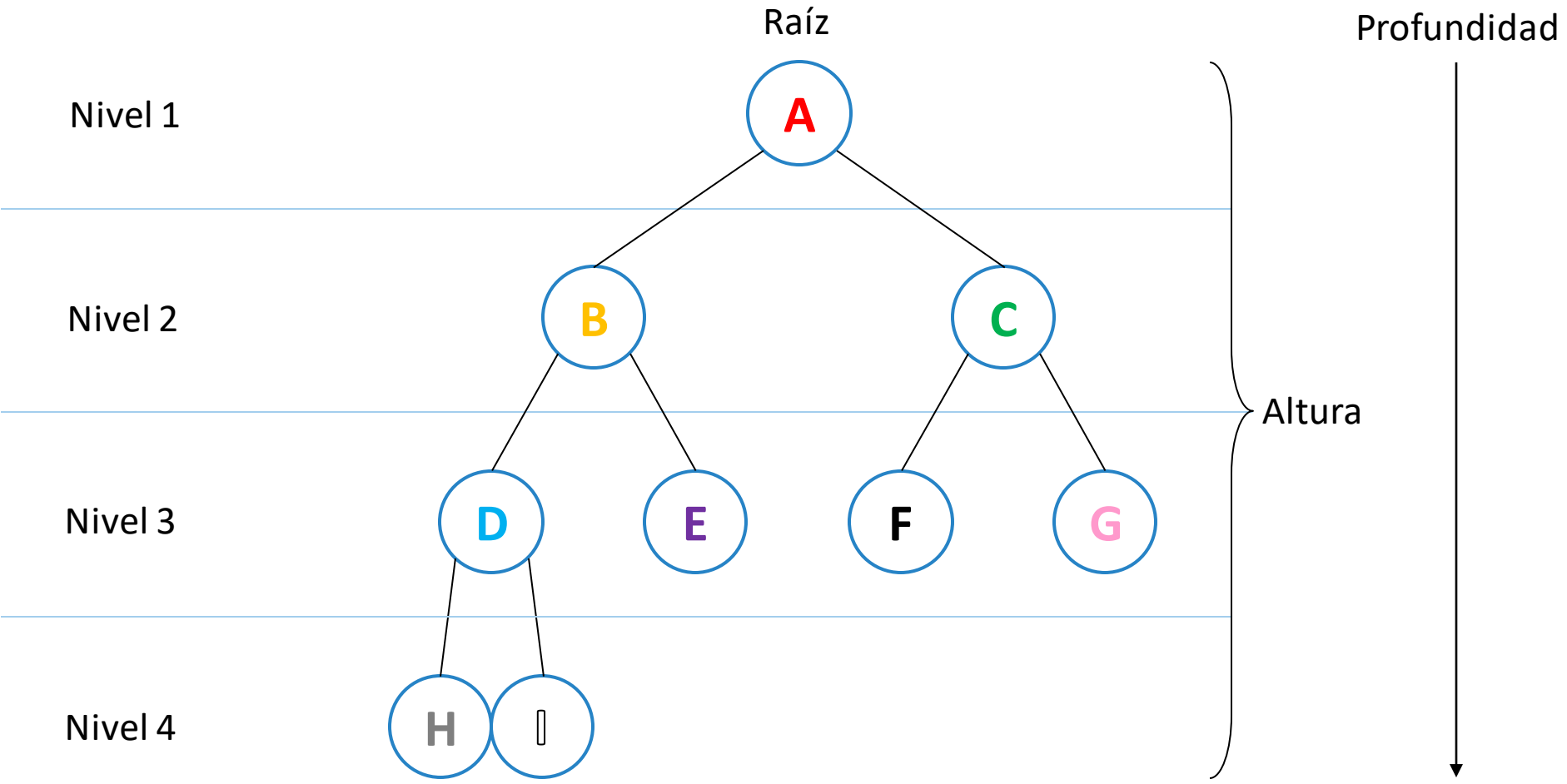
Es un **árbol binario**, con el elemento más prioritario como raíz

Los demás datos están divididos en dos grupos

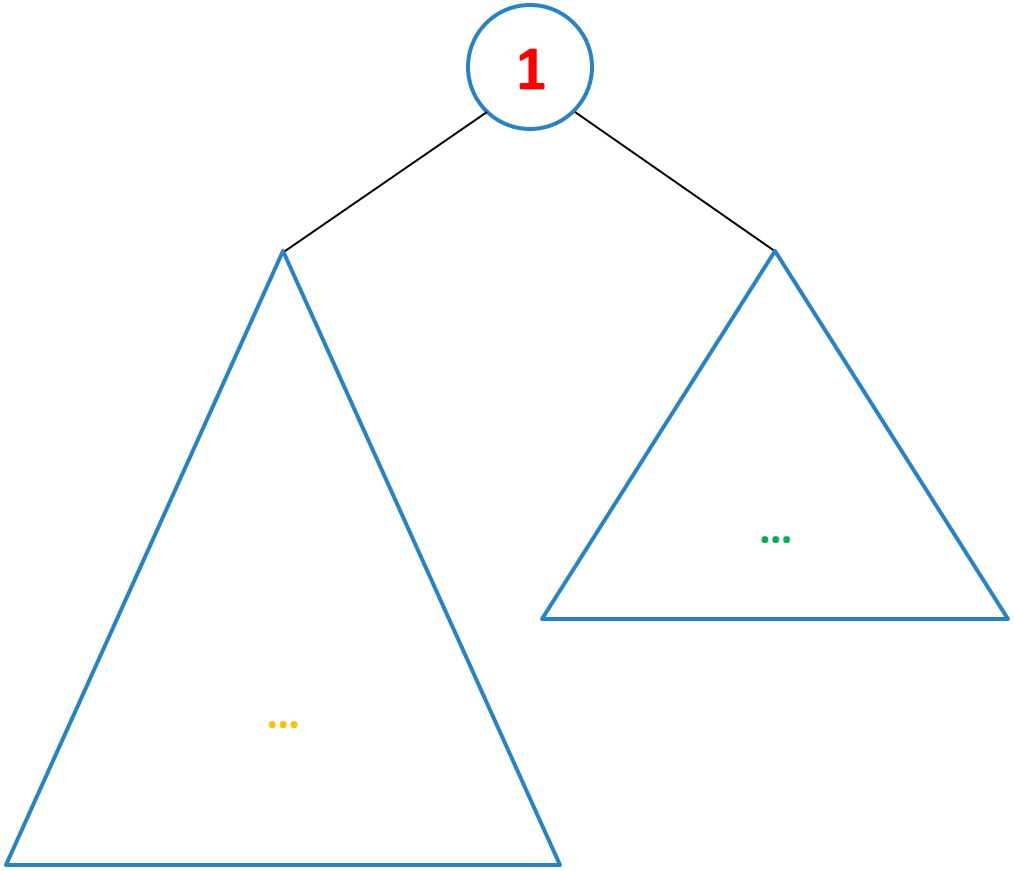
Cada grupo está a su vez organizados como un heap binario

Estos heap binarios cuelgan de la raíz como sus hijos

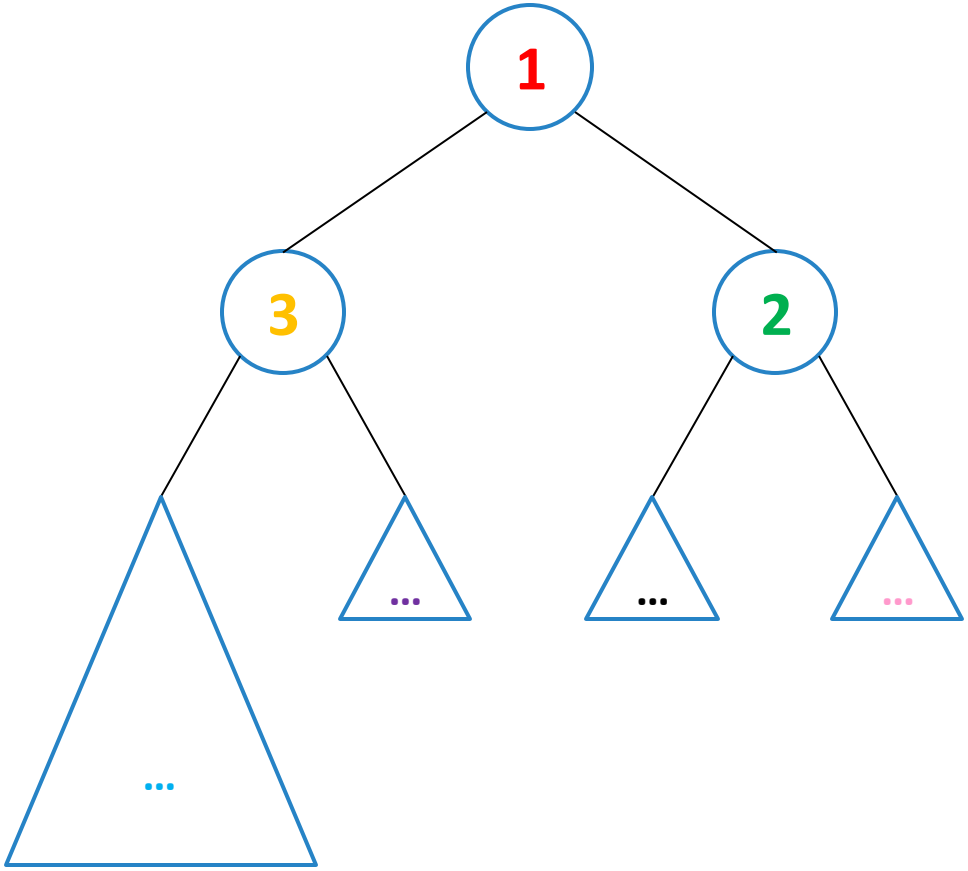
Anatomía de un Árbol Binario



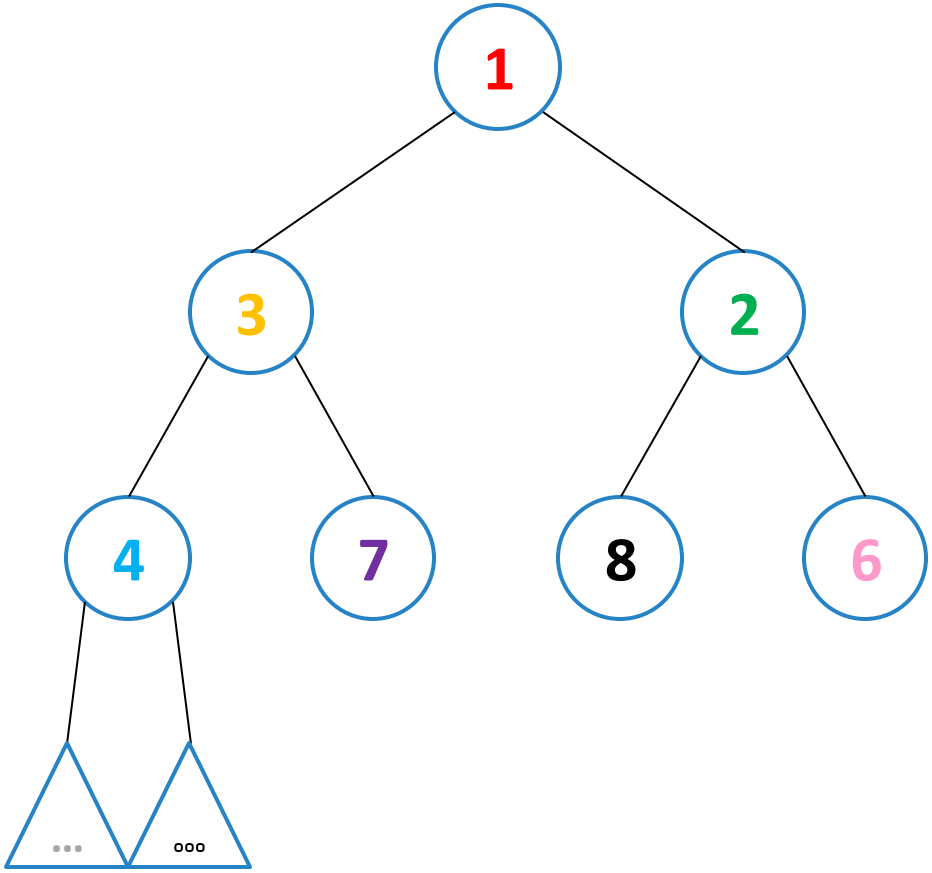
El Heap Binario



El Heap Binario



El Heap Binario



Operaciones del Heap

Al insertar y extraer elementos, el Heap debe reestructurarse para conservar sus propiedades

¿Cómo se definen estas operaciones de manera recursiva?

Idealmente queremos que el heap llene los niveles en orden

$extract(H)$:

$best \leftarrow H.root,$ $next \leftarrow \emptyset$

if H *tiene un hijo,* H':

$next \leftarrow extract(H')$

else if H *tiene dos hijos*:

$H' \leftarrow$ *el hijo de* H *de mayor prioridad*

$next \leftarrow extract(H')$

$H.root \leftarrow next$

return $best$

Finitud

La cantidad de niveles de un heap es finita

En cada llamada a **extract** bajamos un nivel

Por lo tanto la extracción termina en tiempo finito

Correctitud

PD: La extracción en un heap de altura h preserva las propiedades del heap

Por **inducción** sobre la altura

Caso Base: La extracción en un heap de altura 1 deja un heap vacío, el cual es un heap válido.

HI: La extracción de un heap de altura $\leq i$ preserva las propiedades del heap.

Al extraer en un heap H de altura $i + 1$, se identifica cual de ambos hijos tiene la raíz más prioritaria x, y se extrae de ahí. Este hijo tiene altura $\leq i$, por lo que luego de extraer su raíz sigue siendo un heap. Luego H deja x como su propia raíz, la cual es mayor a las raíces de ambos hijos. Por lo tanto H sigue siendo un heap.

Es decir, para cualquier altura h, la extracción preserva las propiedades del heap

$insert(H, e)$:

    if H $tiene$ 0 o 1 $hijos$:

        $H' \leftarrow un\ nuevo\ hijo\ para\ H$

        $H'.root \leftarrow e$

    $else$:

        $H' \leftarrow el\ hijo\ de\ H\ de\ menor\ altura$

        $insert(H',\ e)$

    if $H'.root > H.root$:

        $H'.root \rightleftarrows H.root$

Finitud

La cantidad de niveles de un heap es finita

En cada llamada a **insert** bajamos un nivel

Por lo tanto la extracción termina en tiempo finito

Correctitud

PD: La inserción en un heap de altura h preserva las propiedades del heap

Por **inducción** sobre la altura

Caso Base: La inserción de e en un heap H de altura 1 crea un nuevo hijo para H, con e como raíz. Si la raíz de H tiene menos prioridad que e, entonces se intercambian. Así, H queda como un heap, ya que su raíz es la de mayor prioridad y su hijo es un heap.

HI: La inserción en un heap H de altura $\leq i$ preserva las propiedades del heap.

Al insertar e en un heap H de altura $i+1$ se inserta e en H', uno de sus hijos. Ya que estos son de altura $\leq i$, quedan como heap luego de la inserción. Luego intercambia las raíces entre H y H' si estas violan la propiedad del heap. Tanto H' como su hermano siguen siendo un heap, y ahora H tiene como cabeza al elemento más prioritario, por lo que sigue siendo un heap.

Es decir, para cualquier altura h, la inserción preserva las propiedades del heap.

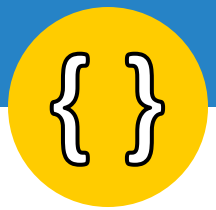
El Heap Binario

¿Cómo se puede acotar la altura de un heap con n datos?

Considerando eso,

¿Cuál es la complejidad de sus operaciones?

El Heap como arreglo

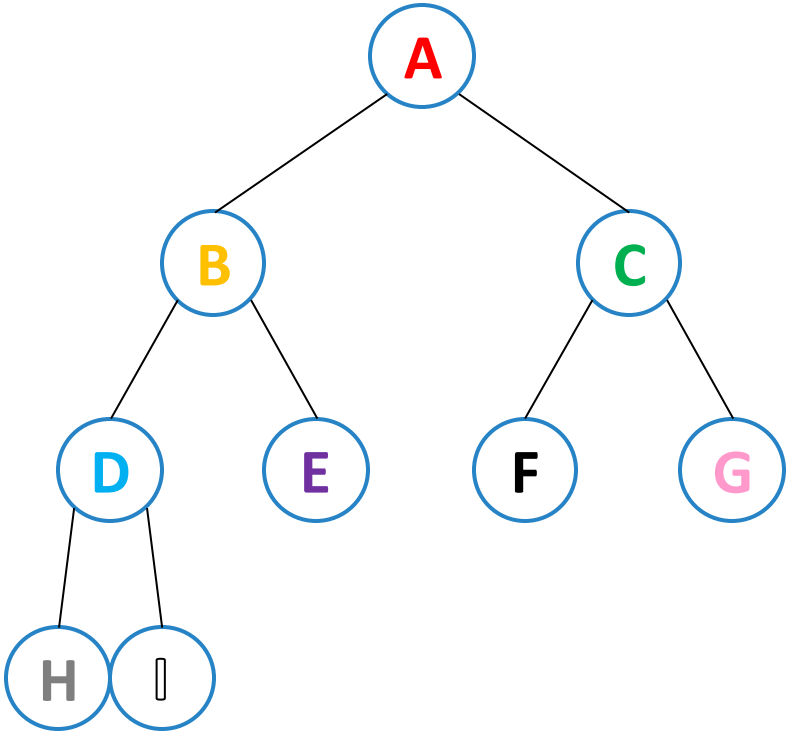


Usualmente, la cantidad de datos a insertar es conocida

Y como la inserción puede hacerse en cualquier hoja

Es posible implementar el heap de forma compacta en un **arreglo**

El Heap como arreglo



A	B	C	D	E	F	G	H	
1	2	3	4	5	6	7	8	9

Cambio de prioridad

Tenemos acceso directo a cualquier elemento del heap

¿Cómo aprovecharlo para cambiar la prioridad de un dato?

$$sift\ up(H, i):$$

$$if\ i\ tiene\ padre:$$

$$i' \leftarrow el\ padre\ de\ i$$

$$if\ H[i'] > H[i]:$$

$$H[i'] \rightleftarrows H[i]$$

$$sift\ up(H, i')$$

$sift\ down(H, i)$:

    if i *tiene hijos*:

        $i' \leftarrow$ *el hijo de* i *de mayor prioridad*

        if $H[i'] > H[i]$:

            $H[i'] \rightleftarrows H[i]$

            $sift\ down(H, i')$

Heaps & Selection Sort

La clase pasada vimos Selection Sort

¿Será posible usar un heap para mejorar su rendimiento?

Heapsort

Para la secuencia inicial de datos, A.

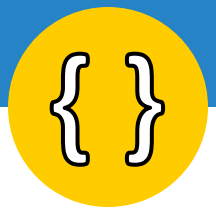
1. Convertir A en un min-heap con los datos
2. Definir una secuencia ordenada, B, inicialmente vacía
3. Extraer el menor dato x de A e insertarlo al final de B
4. Si quedan elementos en A, volver a 2.

Complejidad

¿Cómo se demuestra que Heapsort es correcto?

¿Cuál es su complejidad?

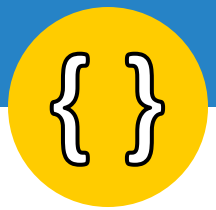
Heapsort



En la práctica, se usa un mismo arreglo para A y B

Eso significa que **Heapsort** no requiere memoria adicional

El Heap como arreglo



Se puede definir la inserción y la extracción desde $sift\ up$ y $down$.

Esto facilita **mucho** la implementación

$\boldsymbol{insert(H, e)}$:

$\boldsymbol{i \leftarrow la\ primera\ celda\ en\ blanco\ de\ H}$

$\boldsymbol{H[i] \leftarrow e}$

$\boldsymbol{sift\ up(H, i)}$

$extract(H)$:

$i \leftarrow$ *la ultima celda no vacia de* H

$best \leftarrow H[0]$

$H[i] \rightleftarrows H[0]$

$H[i] \leftarrow \emptyset$

$sift\ down(H, 0)$

return $best$