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 a la siguiente persona
- La persona que pasa es la con mayor urgencia que aún está en espera
- En caso de empate, pasa la persona que llegó primero

¿Cuánto cuesta decidir cuál es la próxima persona que hay que hacer pasar?

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 determinada prioridad en la cola
- Extraer el dato con más prioridad de la cola

E idealmente:

- Cambiar la prioridad de un dato ya insertado

Propiedad de orden de la cola

Claramente hay que mantener cierto orden de los datos

¿Cuál es el costo —en términos del número de operaciones o pasos básicos— de mantener los datos **ordenados**?

¿Y al llegar n datos nuevos?

¿Es necesario un orden total?

Solo necesitamos cual es el dato más prioritario

Quizás podamos darnos el lujo de no tener un orden total de los datos

¡Necesitamos algún tipo de estructura interna!

Cómo aprovechar la propiedad de orden parcial

¿Qué información contenida en la estructura debe estar fácilmente disponible en todo momento?

¿Será posible hacer una estructura recursiva?

¿Por qué querríamos tener estructuras recursivas?

- los algoritmos para recorrerlas o buscar información en ellas son también recursivos y, por lo tanto, más simples
- la implementación de la estructura se simplifica

El *heap* binario: Una estructura recursiva

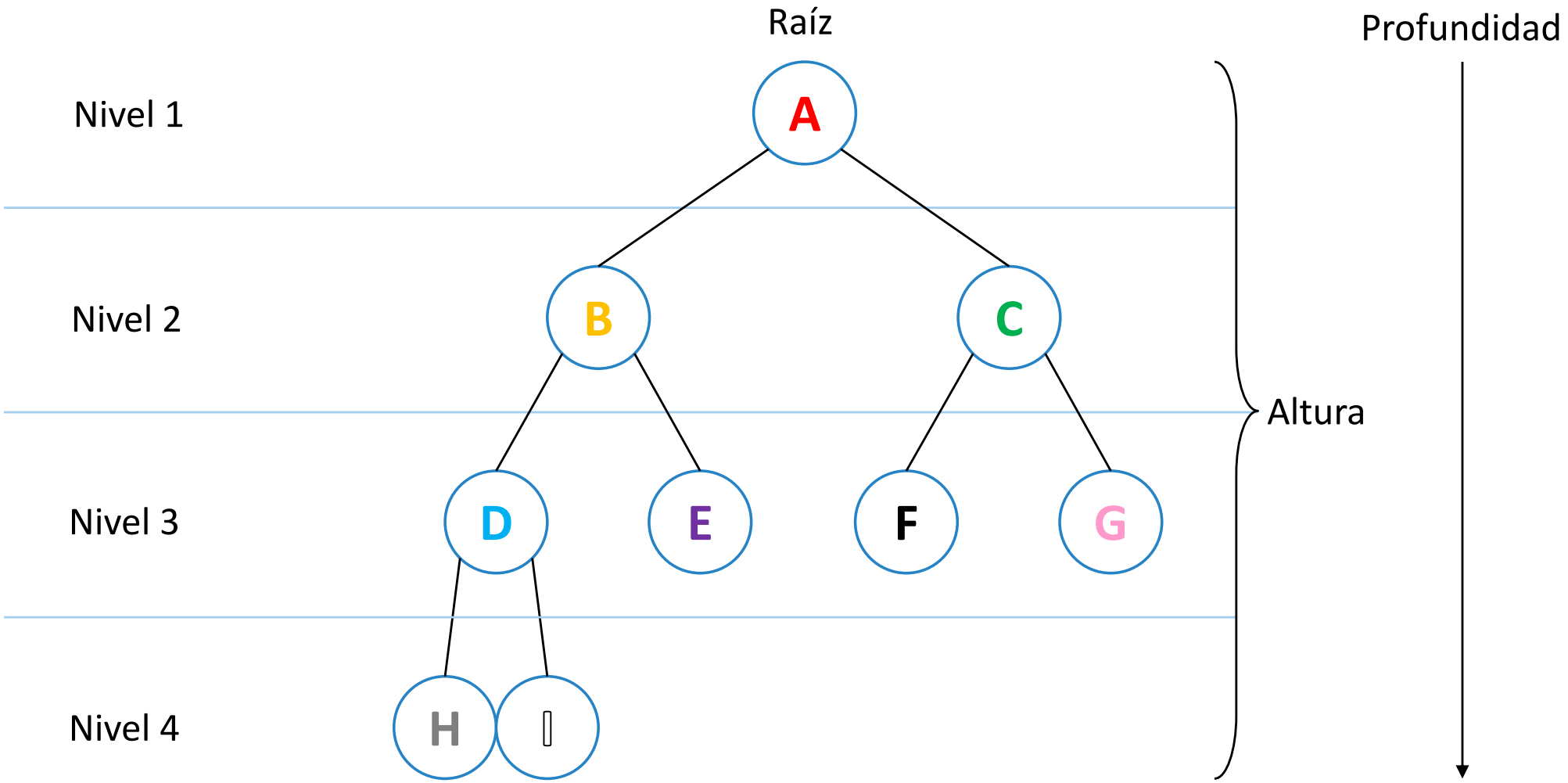
Es un **árbol binario**, con el elemento más prioritario como raíz:

- (ver próx. diapositiva)

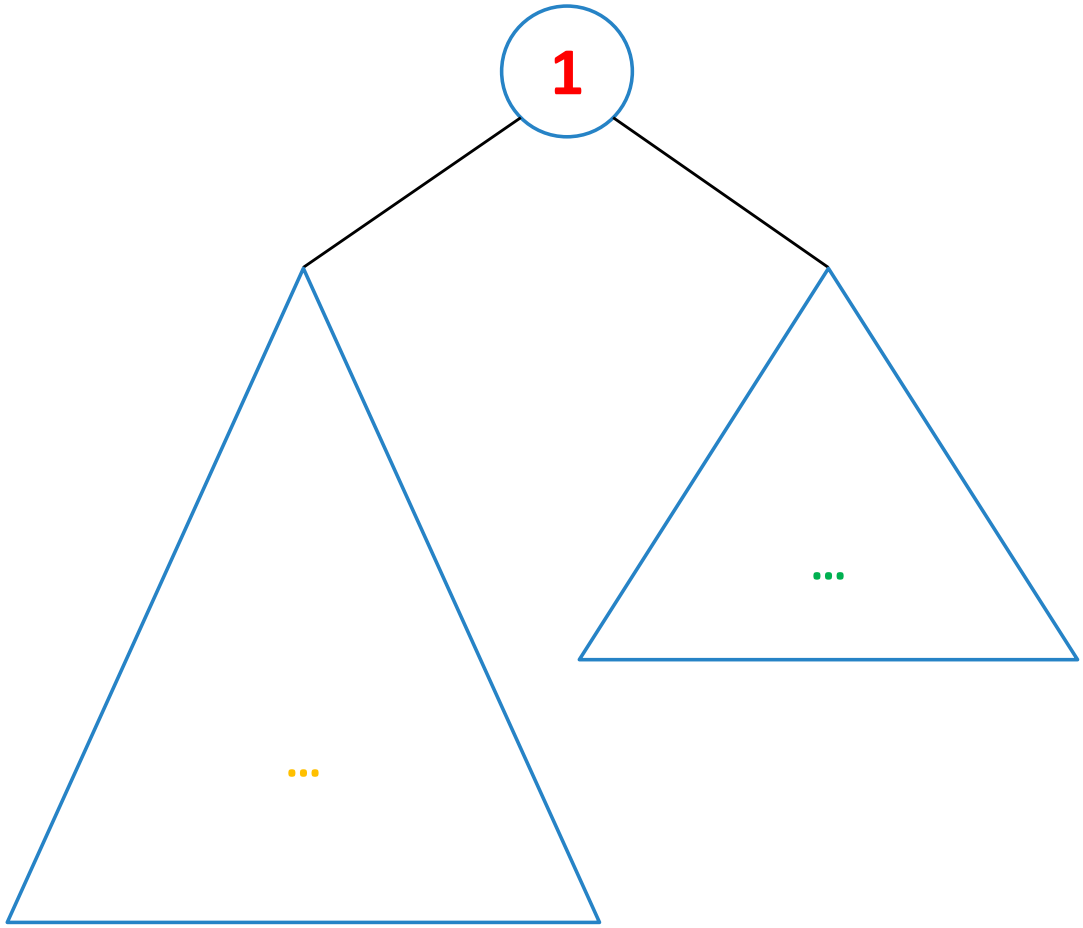
Los demás datos están divididos en dos grupos:

- (ver diapositivas subsiguientes)
- cada grupo está organizado a su vez —recursivamente— como un heap binario
- estos dos heaps binarios cuelgan de la raíz como sus hijos

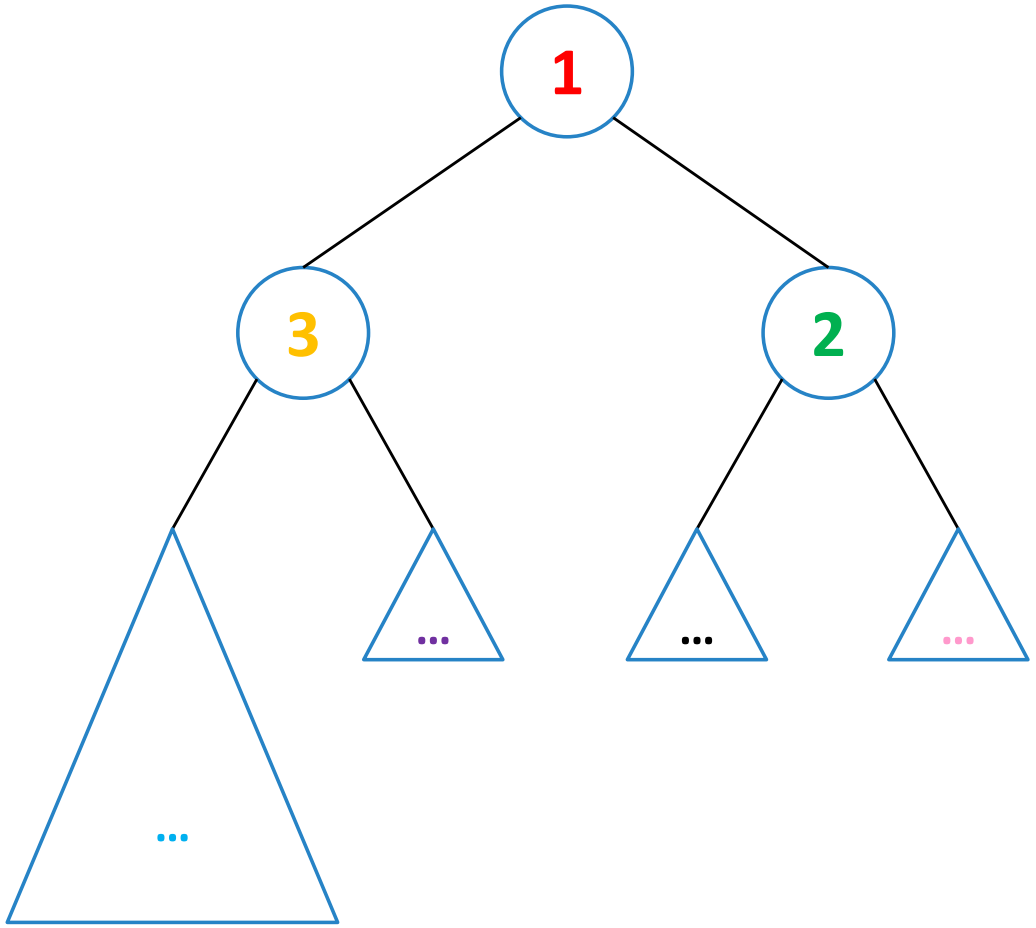
Anatomía de un árbol binario



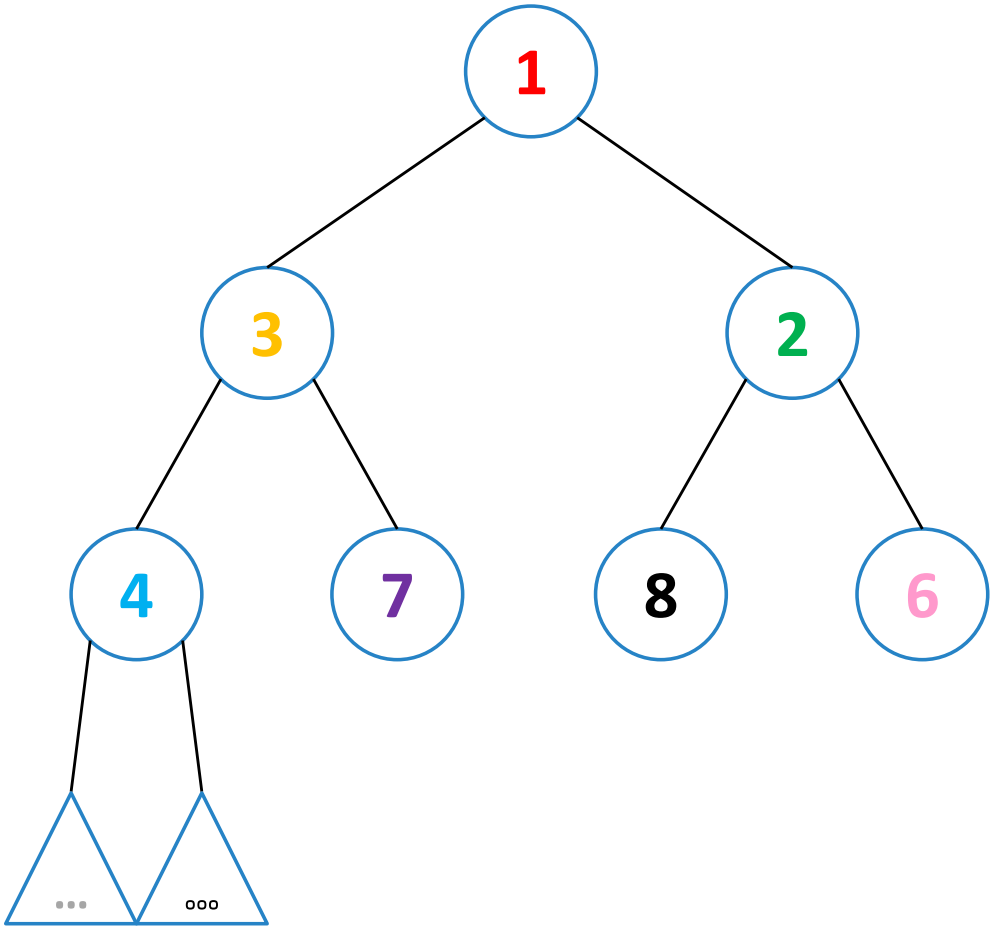
Un heap binario: La raíz y sus dos hijos



Un heap binario:
Cada hijo de la raíz es un heap binario



Un heap binario: Recursivo a todo nivel



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 $best$

$insert(H, e)$:

    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 $H'.root > H.root$:

        $H'.root \rightleftarrows H.root$

Altura de un heap binario

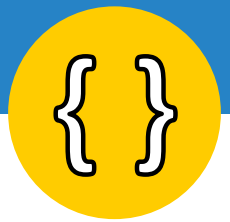
Dado que el número de pasos de los algoritmos depende de la altura del heap,

… ¿cuál es la altura de un heap con n datos?

Considerando eso,

¿Cuál es la complejidad de sus operaciones?

Una implementación simple de un heap binario

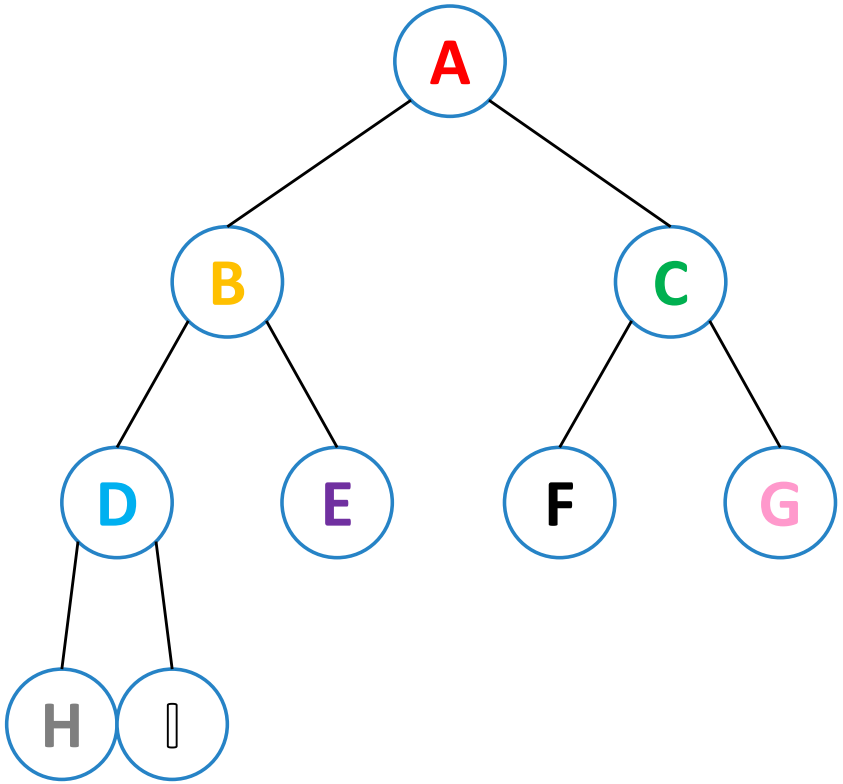


Si podemos suponer una cantidad máxima de datos que pueden estar en el heap simultáneamente

… y como la inserción puede hacerse en cualquier hoja

Entonces, es posible implementar el heap de forma compacta en un **arreglo**

Un heap binario como un arreglo



A	B	C	D	E	F	G	H	▯
1	2	3	4	5	6	7	8	9