Diseño: Elección de estructuras II

Algoritmos y Estructuras de Datos

2 de noviembre de 2023

Plan del día

- Repaso de la práctica pasada
- Repaso de las teóricas
- Ejercicios
- Devolución de parciales a las 20hs

TAD vs. Estructura vs. Módulo

- Tres cosas que usan nombres parecidos pero es importante distinguir

TAD vs. Estructura vs. Módulo

- Tres cosas que usan nombres parecidos pero es importante distinguir
  - **TAD** – Describe un comportamiento. Ej.: Conjunto, diccionario, cola, matriz.

TAD vs. Estructura vs. Módulo

- Tres cosas que usan nombres parecidos pero es importante distinguir
  - **TAD** – Describe un comportamiento. Ej.: Conjunto, diccionario, cola, matriz.
  - **Estructura** en memoria – Cómo se organiza físicamente la información en memoria. Ej.: Arreglo, lista enlazada, árboles.

TAD vs. Estructura vs. Módulo

- Tres cosas que usan nombres parecidos pero es importante distinguir
  - **TAD** – Describe un comportamiento. Ej.: Conjunto, diccionario, cola, matriz.
  - **Estructura** en memoria – Cómo se organiza físicamente la información en memoria. Ej.: Arreglo, lista enlazada, árboles.
    - Cada estructura tiene ventajas y desventajas en cuanto a la complejidad del acceso a esa información

TAD vs. Estructura vs. Módulo

- Tres cosas que usan nombres parecidos pero es importante distinguir
  - **TAD** – Describe un comportamiento. Ej.: Conjunto, diccionario, cola, matriz.
  - **Estructura** en memoria – Cómo se organiza físicamente la información en memoria. Ej.: Arreglo, lista enlazada, árboles.
    - Cada estructura tiene ventajas y desventajas en cuanto a la complejidad del acceso a esa información
  - **Módulo** – Un TAD implementado sobre una estructura.

TAD vs. Estructura vs. Módulo

- Tres cosas que usan nombres parecidos pero es importante distinguir
  - **TAD** – Describe un comportamiento. Ej.: Conjunto, diccionario, cola, matriz.
  - **Estructura** en memoria – Cómo se organiza físicamente la información en memoria. Ej.: Arreglo, lista enlazada, árboles.
    - Cada estructura tiene ventajas y desventajas en cuanto a la complejidad del acceso a esa información
  - **Módulo** – Un TAD implementado sobre una estructura.
    - Puede nombrarse según el nombre de ambas (ConjuntoSobreArray, ConjuntoSobreABB, etc.)

TAD vs. Estructura vs. Módulo

- Tres cosas que usan nombres parecidos pero es importante distinguir
  - **TAD** – Describe un comportamiento. Ej.: Conjunto, diccionario, cola, matriz.
  - **Estructura** en memoria – Cómo se organiza físicamente la información en memoria. Ej.: Arreglo, lista enlazada, árboles.
    - Cada estructura tiene ventajas y desventajas en cuanto a la complejidad del acceso a esa información
  - **Módulo** – Un TAD implementado sobre una estructura.
    - Puede nombrarse según el nombre de ambas (ConjuntoSobreArray, ConjuntoSobreABB, etc.)
    - O por la complejidad de una operación común (ej.: ConjuntoLineal, ConjuntoLogarítmico, etc.)

Práctica 7, ejercicio 1

Confeccione una tabla comparativa de las complejidades de peor caso de las operaciones de pertenencia, inserción, borrado, búsqueda del mínimo y borrado del mínimo para conjuntos de naturales sobre las siguientes estructuras:

- Lista enlazada
- Lista enlazada ordenada
- Árbol binarios de búsqueda
- Árbol AVL
- *heap*
- *trie*

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada					
Lista Ordenada					
ABB					
AVL					

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$				
Lista Ordenada					
ABB					
AVL					

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$			
Lista Ordenada					
ABB					
AVL					

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$		
Lista Ordenada					
ABB					
AVL					

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	
Lista Ordenada					
ABB					
AVL					

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada					
ABB					
AVL					

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB					
AVL					

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL					

Práctica 7, ejercicio 1

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$

Heaps - Definición

- Árbol binario balanceado

Heaps - Definición

- Árbol binario balanceado
- No está ordenado ni es “de búsqueda”

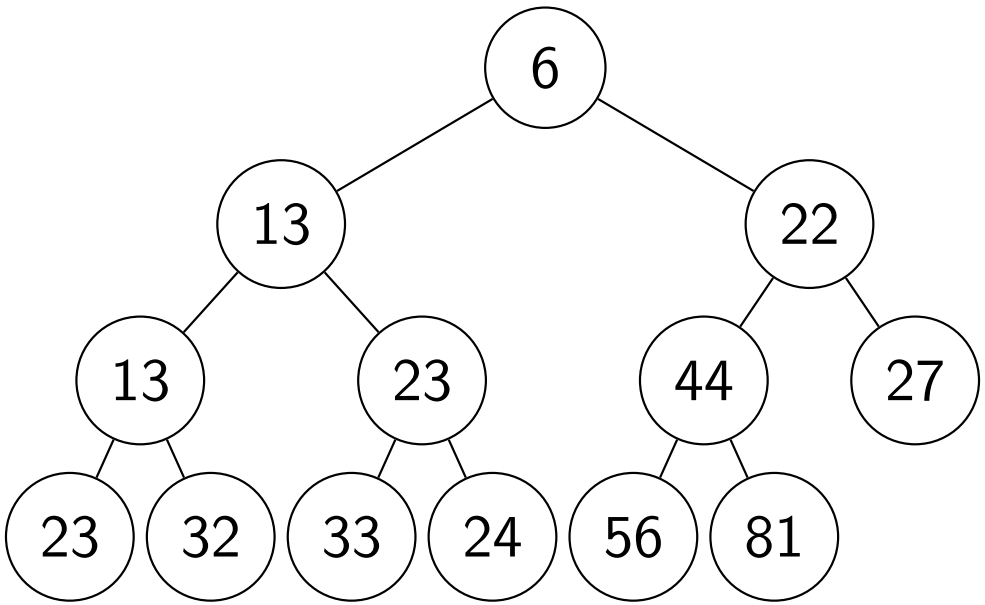
Heaps - Definición

- Árbol binario balanceado
- No está ordenado ni es "de búsqueda"
- Clave o valor es $\geq$ a los hijos

Heaps - Definición

- Árbol binario balanceado
- No está ordenado ni es "de búsqueda"
- Clave o valor es $\geq$ a los hijos
- Todo subárbol es un *heap*

Heaps - Ejemplo



Heaps - Implementaciones

- Nodos simple y doblemente enlazados

Heaps - Implementaciones

- Nodos simple y doblemente enlazados
- Arreglo (particularmente eficiente)

Heaps - Implementaciones

- Nodos simple y doblemente enlazados
- Arreglo (particularmente eficiente)
  - Cada nodo `n` se almacena en la posición `i` del arreglo

Heaps - Implementaciones

- Nodos simple y doblemente enlazados
- Arreglo (particularmente eficiente)
  - Cada nodo `n` se almacena en la posición `i` del arreglo
  - Tenemos una función `p(n)`:

Heaps - Implementaciones

- Nodos simple y doblemente enlazados
- Arreglo (particularmente eficiente)
  - Cada nodo `n` se almacena en la posición `i` del arreglo
  - Tenemos una función `p(n)`:
    - `p(n) = 0` si `n` es la raíz del heap

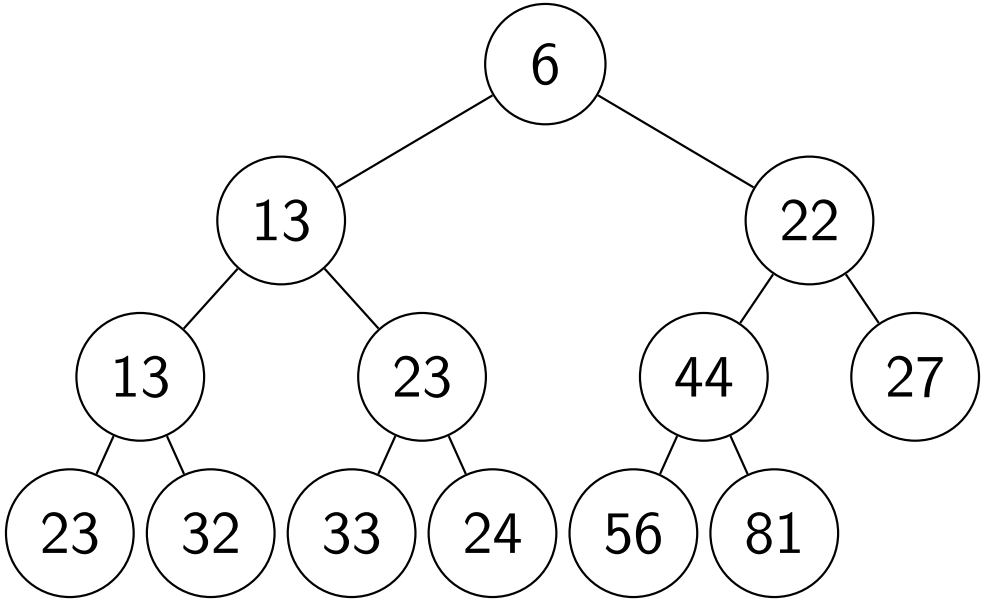
Heaps - Implementaciones

- Nodos simple y doblemente enlazados
- Arreglo (particularmente eficiente)
  - Cada nodo `n` se almacena en la posición `i` del arreglo
  - Tenemos una función `p(n)`:
    - `p(n) = 0` si `n` es la raíz del heap
    - `p(n) = 2×p(m) + 1` si `n` es el hijo izquierdo de `m`

Heaps - Implementaciones

- Nodos simple y doblemente enlazados
- Arreglo (particularmente eficiente)
  - Cada nodo `n` se almacena en la posición `i` del arreglo
  - Tenemos una función `p(n)`:
    - `p(n) = 0` si `n` es la raíz del heap
    - `p(n) = 2×p(m) + 1` si `n` es el hijo izquierdo de `m`
    - `p(n) = 2×p(m) + 2` si `n` es el hijo derecho de `m`

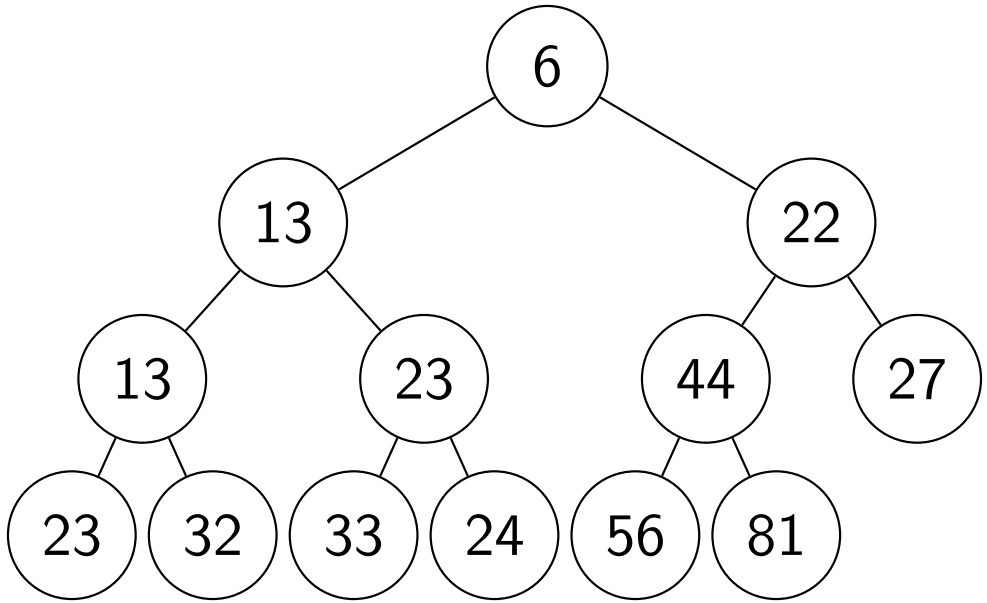
Heaps - Ejemplo de implementación



6	13	22	13	23	44	27	23	32	33	24	56	81
0	1	2	3	4	5	6	7	8	9	10	11	12

- `p(6) = 0`

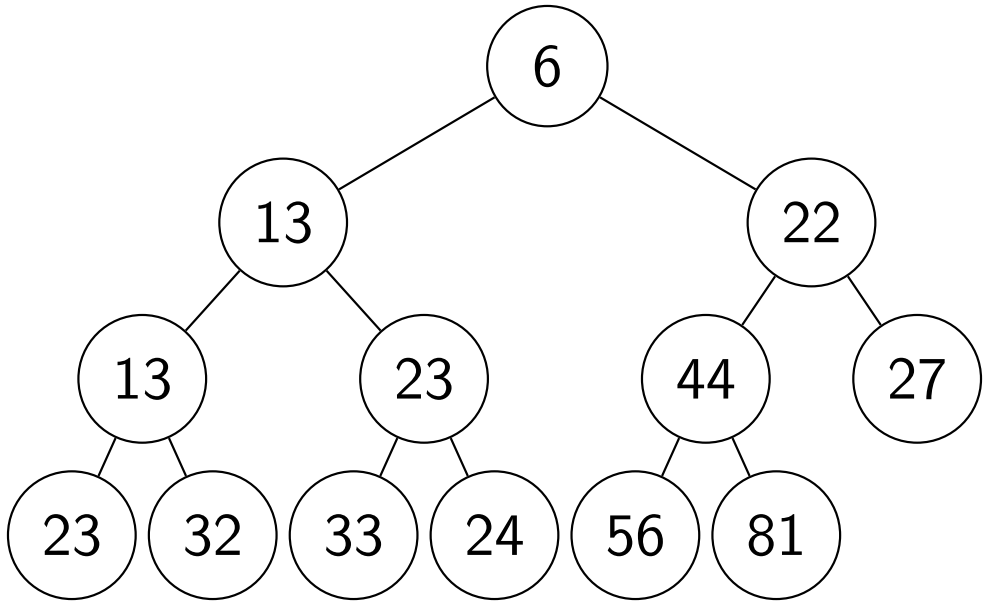
Heaps - Ejemplo de implementación



6	13	22	13	23	44	27	23	32	33	24	56	81
0	1	2	3	4	5	6	7	8	9	10	11	12

- p(6) = 0
- p(22) = 2×p(6) + 2 = 2×0 + 2 = 2

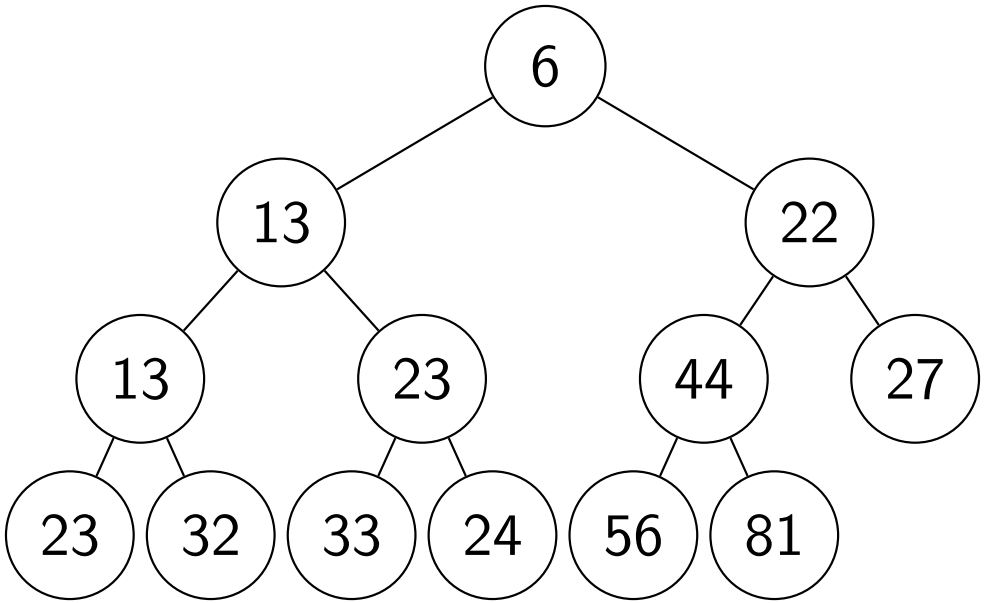
Heaps - Ejemplo de implementación



6	13	22	13	23	44	27	23	32	33	24	56	81
0	1	2	3	4	5	6	7	8	9	10	11	12

- `p(6) = 0`
- `p(22) = 2×p(6) + 2 = 2×0 + 2 = 2`
- `p(44) = 2×p(22) + 1 = 2×(2×p(6) + 2) + 1 = 2×2+1 = 5`

Heaps - Ejemplo de implementación



6	13	22	13	23	44	27	23	32	33	24	56	81
0	1	2	3	4	5	6	7	8	9	10	11	12

- p(6) = 0
- p(22) = 2×p(6) + 2 = 2×0 + 2 = 2
- p(44) = 2×p(22) + 1 = 2×(2×p(6) + 2) + 1 = 2×2+1 = 5
- p(81) = 2×p(44) + 2 = 2×5 + 2 = 12

Práctica 7, ejercicio 1 (cont.)

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$
Heap					

Práctica 7, ejercicio 1 (cont.)

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$
Heap	$O(n)$				

Práctica 7, ejercicio 1 (cont.)

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$
Heap	$O(n)$	$O(log\ n)$			

Práctica 7, ejercicio 1 (cont.)

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$
Heap	$O(n)$	$O(log\ n)$	$O(n)$		

Práctica 7, ejercicio 1 (cont.)

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$
Heap	$O(n)$	$O(log\ n)$	$O(n)$	$O(1)$	

Práctica 7, ejercicio 1 (cont.)

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$
Heap	$O(n)$	$O(log\ n)$	$O(n)$	$O(1)$	$O(log\ n)$

- Es un TAD con las operaciones
  - Vacia(out c: cola)
  - Encolar(inout c: cola, in e: α)
  - Desencolar(inout c: cola, out e: α)
  - Próximo(in c: cola, out e: α)

Heaps - Cola de prioridad

- Es un TAD con las operaciones
  - Vacia(out c: cola)
  - Encolar(inout c: cola, in e: α)
  - Desencolar(inout c: cola, out e: α)
  - Próximo(in c: cola, out e: α)
- Implementación sobre *heap* (detalles en la teórica)
  - Próximo(): El elemento de prioridad máxima está en la posición 0

Heaps - Cola de prioridad

- Es un TAD con las operaciones
  - Vacia(out c: cola)
  - Encolar(inout c: cola, in e: α)
  - Desencolar(inout c: cola, out e: α)
  - Próximo(in c: cola, out e: α)
- Implementación sobre *heap* (detalles en la teórica)
  - Próximo(): El elemento de prioridad máxima está en la posición 0 $\Rightarrow O(1)$

Heaps - Cola de prioridad

- Es un TAD con las operaciones
  - Vacia(out c: cola)
  - Encolar(inout c: cola, in e: α)
  - Desencolar(inout c: cola, out e: α)
  - Próximo(in c: cola, out e: α)
- Implementación sobre *heap* (detalles en la teórica)
  - Próximo(): El elemento de prioridad máxima está en la posición 0 $\Rightarrow O(1)$
  - Encolar(): Consiste en insertar el elemento al final del heap y reacomodar

Heaps - Cola de prioridad

- Es un TAD con las operaciones
  - Vacia(out c: cola)
  - Encolar(inout c: cola, in e: α)
  - Desencolar(inout c: cola, out e: α)
  - Próximo(in c: cola, out e: α)
- Implementación sobre *heap* (detalles en la teórica)
  - Próximo(): El elemento de prioridad máxima está en la posición 0 $\Rightarrow O(1)$
  - Encolar(): Consiste en insertar el elemento al final del heap y reacomodar $\Rightarrow O(log\ n)$

Heaps - Cola de prioridad

- Es un TAD con las operaciones
  - Vacia(out c: cola)
  - Encolar(inout c: cola, in e: α)
  - Desencolar(inout c: cola, out e: α)
  - Próximo(in c: cola, out e: α)
- Implementación sobre *heap* (detalles en la teórica)
  - Próximo(): El elemento de prioridad máxima está en la posición 0 $\Rightarrow O(1)$
  - Encolar(): Consiste en insertar el elemento al final del heap y reacomodar $\Rightarrow O(log\ n)$
  - Desencolar(): Consiste en reemplazar la raíz con la última hoja, eliminar ésta y reacomodar

Heaps - Cola de prioridad

- Es un TAD con las operaciones
  - Vacia(out c: cola)
  - Encolar(inout c: cola, in e: α)
  - Desencolar(inout c: cola, out e: α)
  - Próximo(in c: cola, out e: α)
- Implementación sobre *heap* (detalles en la teórica)
  - Próximo(): El elemento de prioridad máxima está en la posición 0 $\Rightarrow O(1)$
  - Encolar(): Consiste en insertar el elemento al final del heap y reacomodar $\Rightarrow O(log\ n)$
  - Desencolar(): Consiste en reemplazar la raíz con la última hoja, eliminar ésta y reacomodar $\Rightarrow O(log\ n)$

Heaps - Algoritmo de Floyd

- Convierte un array cualquiera en un heap
- "Heapifica" subárboles de abajo hacia arriba
- Complejidad: $O(n)$ (mejor que la alternativa de encolar de a un elemento que es $O(n \log n)$)
- Una vez más: detalles en la teórica

Bueno, basta de charla

Bueno, basta de charla

Práctica 7, Ejercicio 12