TEMA 4
El tipo conjunto

PROGRAMACIÓN Y ESTRUCTURAS DE DATOS

Tipo conjunto

- 1. Definiciones generales
- 2. Diccionario
  - 2.1. Tabla de dispersión
- 3. Cola de prioridad
  - 3.1. Montículo
  - 3.2. Cola de prioridad doble
    - 3.2.1. Montículo doble

1. Tipo Conjunto
DEFINICIONES

- Un conjunto es una colección de elementos, cada uno de los cuales puede ser un conjunto, o un elemento primitivo que recibe el nombre de átomo
- Todos los miembros del conjunto son distintos
- El orden de los elementos no es importante (distinto de las listas)

Notación de Conjuntos

- Se representa encerrando sus miembros entre llaves {1,2,5}
- Relación fundamental, la de pertenencia: $\in$ $\{x / x \in \text{Naturales}\}$, $\{x / x < 8\}$
- Existe un conjunto especial sin elementos: $\varnothing$
- $A \subset B$ si todo elemento de A también lo es de B
- Conjunto Universal: formado por todos los posibles elementos que puede contener

1. Tipo Conjunto
SINTAXIS

MODULO GENERICO ModuloConjunto
MODULO Conjunto **USA** Boolean, Natural
SINTAXIS

Crear ( ) → Conjunto
Insertar(Conjunto, Ítem) → Conjunto
Eliminar(Conjunto, Ítem) → Conjunto
Pertenece(Conjunto, Ítem) → Boolean
EsVacíoConjunto(Conjunto) → Boolean
Cardinalidad(Conjunto) → Natural
Unión(Conjunto, Conjunto) → Conjunto
Intersección(Conjunto, Conjunto) → Conjunto
Diferencia(Conjunto, Conjunto) → Conjunto

VAR

C, D: Conjunto; x, y: Ítem;

1. Tipo Conjunto

SEMÁNTICA (I)

EsVacíoConjunto(Crear) ←→ Cierto
EsVacíoConjunto(Insertar(C, x)) ←→ Falso
Insertar(Insertar(C, x), y) ←→
    si (x == y) **entonces** Insertar(C, x) //no se permiten elementos repetidos
    sino Insertar(Insertar(C, y), x) //da igual el orden de inserción de los elem.
Eliminar(Crear, x) ←→ Crear
Eliminar(Insertar(C, x), y) ←→
    si (x == y) **entonces** C // ¿y si permitieran elementos repetidos?
    sino Insertar(Eliminar(C, y), x)
Pertenece(Crear, x) ←→ Falso
Pertenece(Insertar(C, x), y) ←→
    si (x == y) **entonces** Cierto
    sino Pertenece(C, y)
Cardinalidad(Crear) ←→ 0
Cardinalidad(Insertar(C,x)) ←→ 1+Cardinalidad(C)
Unión(Crear, C) ←→ C
Unión(Insertar(C,x), D) ←→
    si (Pertenece(D, x)) **entonces** Unión(C, D)
    sino Insertar(Unión(C, D), x)

1. Tipo Conjunto

SEMÁNTICA (II)

Diferencia(Crear, C) ←→ Crear
Diferencia(Insertar(C,x), D) ←→
    si (Pertenece(D, x))
    entonces Diferencia(C, D)
    sino Insertar(Diferencia(C, D), x)
Intersección(Crear, D) ←→ Crear
Intersección(Insertar(C,x), D) ←→
    si (Pertenece(D, x))
    entonces Insertar(Intersección(C, D), x)
    sino Intersección(C, D)

1. Tipo Conjunto

IMPLEMENTACIÓN

- **Mediante un vector**
  - **-Vector de bits/enteros (cada componente corresponde a un elemento del conjunto universal)**

1	0	0	0	1	0
0	1	2	3	4	5

  - **-Vector de elementos**

1	9	0	4	2	

- **Almacenar los elementos conforme se inserten (mediante listas, árboles, ...):**

Espacio proporcional al conjunto representado

1. Tipo Conjunto

EJERCICIO

- **Rellenar la siguiente tabla de complejidades (peor caso):**

m=elem. conjto. n=elem. conjto. Univ.	Vector de Bits	Lista ordenada	Lista desordenada
Búsqueda			
Inserción			
Unión			

2. DICCIONARIO

DEFINICIÓN

- Subtipo del CONJUNTO, con las operaciones:
  - CREAR
  - INSERTAR
  - BORRAR
  - BÚSQUEDA

2. DICCIONARIO

IMPLEMENTACIÓN

Implementaciones sencillas:

- **Mediante listas o vectores**

  Búsqueda, Inserción y Borrado:

Listas:	**O (n)**
Vector Bits:	**O (1)**
Vector Elementos:	**O (n)**

- **Mediante TAD Tabla de Dispersión (HASHING)**

2.1. TABLA DE DISPERSIÓN (HASHING)

DEFINICIÓN

- **HASHING: Utilizaremos la información del elemento a almacenar para buscar su posición dentro de la estructura**
- **Operaciones:**
  - **Búsqueda.** O(1)
  - **Inserción.** O(1)
  - **Borrado.** O(1)

2.1. TABLA DE DISPERSIÓN (HASHING)

MÉTODO

- Dividir el conjunto en un número finito “B” de clases
- Se usa función de dispersión H, tal que H(x) será un valor entre 0 y B-1

Formas de dispersión:

Abierta: No impone tamaño límite al conjunto

Cerrada: usa un tamaño fijo de almacenamiento (limita el tamaño)

2.1. Tabla Hash. Dispersión Cerrada

DEFINICIÓN

- **Los elementos se almacenan en tabla de tamaño fijo (TABLA DE DISPERSIÓN)**
- **La tabla se divide en B clases, y cada una podrá almacenar S elementos**
- **La Función de dispersión se implementa mediante una función aritmética**

H(x)= x MOD B

2.1. Tabla Hash. Dispersión Cerrada

INSERCIÓN

Caso COLISIÓN: x1, x2 (SINÓNIMOS/ H(x1) = H(x2))

ESTRATEGIA DE REDISPERSION:

- **Elegir sucesión de localidades alternas dentro de la tabla, hasta encontrar una vacía**

H(x), h1(x), h2(x), h3(x), ...

- **Si ninguna está vacía: no es posible insertar**

2.1. Tabla Hash. Dispersión Cerrada

INSERCIÓN. EJEMPLO

Ejemplo. Insertar en una tabla de dispersión cerrada de tamaño *B=7*, con función de dispersión *H(x)= x MOD B*, y con estrategia de redispersión *la siguiente posición de la tabla*, los siguientes elementos: *23, 14, 9, 6, 30, 12, 18, 25*

0	14	14	14	14	14	14	14
1						18	18
2	23	23	23	23	23	23	23
3		9	9	9	9	9	9
4				30	30	30	30
5					12	12	12
6			6	6	6	6	6
	23, 14 un sólo intento	9 dos intentos	6 un sólo intento	30 tres intentos	12 un sólo intento	18 cinco intentos	25 tabla llena

Nº TOTAL DE INTENTOS HASTA LA CLAVE 18:

14

2.1. Tabla Hash. Dispersión Cerrada

BÚSQUEDA. BORRADO

BÚSQUEDA DE ELEMENTOS

Buscar en sucesión de localidades alternas dentro de la tabla, hasta encontrar una vacía:

H(x), h1(x), h2(x), h3(x), ...

BORRADO DE ELEMENTOS

Hay que distinguir durante la <u>búsqueda</u>:

- Casillas vacías
- Casillas suprimidas

<u>Durante la inserción</u> las casillas suprimidas se tratarán como espacio disponible.

2.1. Tabla Hash. Dispersión Cerrada

ANÁLISIS (I)

♯ *ESTRATEGIA DE REDISPERSIÓN LINEAL ("siguiente posición")*:

- No eficiente. Larga secuencia de intentos

$h_i(x) = (H(x) + 1 \cdot i) \text{ MOD } B/ \quad c=1 \qquad \boldsymbol{h_i(x) = (h_{i-1}(x) + 1) \text{ MOD } B}$

♯ *ESTRATEGIA DE REDISPERSIÓN ALEATORIA*:

$h_i(x) = (H(x) + c \cdot i) \text{ MOD } B \ / \quad c>1 \qquad \boldsymbol{h_i(x) = (h_{i-1}(x) + c) \text{ MOD } B}$

Sigue produciendo AMONTONAMIENTO: siguiente intento sólo en función del anterior

c y B no deben tener factores primos comunes mayores que 1

♯ *E.R. CON 2ª FUNCION DE HASH*:

$k(x) = (x \text{ MOD } (B-1)) + 1$

$h_i(x) = (H(x) + k(x) \cdot i) \text{ MOD } B \qquad \boldsymbol{h_i(x) = (h_{i-1}(x) + k(x)) \text{ MOD } B}$

B debe ser primo

2.1. Tabla Hash. Dispersión Cerrada

ANÁLISIS (II)

♯ *LA MEJOR FUNCIÓN DE DISPERSIÓN*:

- Que sea fácil de calcular
- Que minimice el nº de colisiones
- Que distribuya los elementos de forma azarosa
- Debe hacer uso de toda la información asociada a las etiquetas

2.1. Tabla Hash. Dispersión Cerrada

ANÁLISIS (III)

- Estrategia de redispersión aleatoria
  - c y B no deben tener factores primos comunes mayores que 1 para que busque en todas las posiciones de la tabla
    - Ejemplo→ c=4; B=6

      $h_i(x) = (H(x) + c \bullet i) \text{ MOD } B = (h_{i-1}(x) + c) \text{ MOD } B$

      H(10)=10 MOD 6=4

      $h_1(10)=(4+4 \bullet 1) \text{ MOD } 6=(4+4) \text{ MOD } 6=2$

      $h_2(10)=(4+4 \bullet 2) \text{ MOD } 6=(2+4) \text{ MOD } 6=0$

      $h_3(10)=(0+4) \text{ MOD } 6=4$; $h_4(10)=(4+4) \text{ MOD } 6=2$

x		x		x	
0	**1**	**2**	**3**	**4**	**5**

    - Ejemplo→ ¿c=6; B=9? ¿c=2; B=9?

- **Estrategia de redispersión con 2ª función hash**
  - B debe ser primo para que busque en todas las posiciones de la tabla
    - c=k(x) → 1…B-1 // k (x) = (x MOD (B-1)) + 1

      $h_i(x) = (H(x) + k(x) \bullet i) \text{ MOD } B = (h_{i-1}(x) + k(x)) \text{ MOD } B$

      B=7; k(x)=1…6

      B=11;k(x)=1…10

2.1. Tabla Hash. Dispersión Cerrada

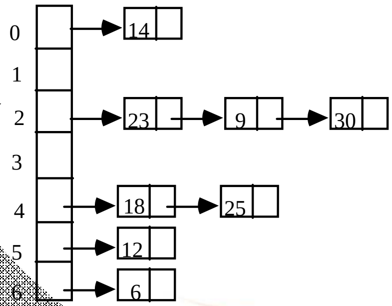
EJERCICIOS

1) Insertar en una tabla de dispersión cerrada de tamaño *B=7*, con función de dispersión *H(x)= x MOD B*, y con estrategia de redispersión *segunda función hash*, los siguientes elementos: *23, 14, 9, 6, 30, 12, 18*

2.1. Tabla Hash. Dispersión Abierta

DEFINICIÓN

- Elimina el problema del CLUSTERING SECUNDARIO (colisiones entre claves no sinónimas)
- Las colisiones se resuelven utilizando una lista enlazada



2.1. Tabla Hash

FACTOR DE CARGA (I)

$$\alpha = \frac{n}{|B|}$$

n = nº elem. de la tabla. B = tamaño de la tabla

HASH CERRADO: $0 \leq \alpha \leq 1$

HASH ABIERTO: $\alpha \geq 0$ (No hay límite en el nº de elementos en cada casilla).

2.1. Tabla Hash

FACTOR DE CARGA (II)

E: Nº Esperado de Intentos.
c.éx: con éxito. **s.éx**: sin éxito.

	H.C.L.		**H.C.Aleat.**		**H.Abierto**	
α	**E c.éx**	**E s.éx.**	**E c.éx**	**E s.éx.**	**E c.éx**	**E s.éx.**
0.1	1.06					
0.25	1.17					
0.5	1.50					
0.75	2.50	8.5	1.9	4.0	1.8	2.0
0.9	5.50	50.5	2.6	10.0	1.9	2.0
0.95	10.50					

2.1. Tabla Hash

COMPARACIÓN HASH ABIERTO Y CERRADO

- H.A. es más eficiente y con menor degradación (cuanto más lleno funciona mejor que el H.C.)
- H.A. requiere espacio para los elementos de la lista, por lo que H.C. es más eficiente espacialmente
- Reestructuración de las tablas de dispersión:
  $n \geq 0{,}9\ B$ (H.C.)
  $n \geq 2\ B$ (H.A.)

→ Nueva tabla con el doble de posiciones

3. COLA DE PRIORIDAD

DEFINICION (I)

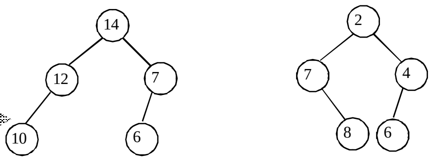
- Conjunto de elementos ordenados con las operaciones:
  Crear ( ) –> ColaPrioridad
  EsVacio () –> Boolean
  Insertar (ColaPrioridad, Item) –> ColaPrioridad
  BorrarMínimo (ColaPrioridad) –> ColaPrioridad
  BorrarMáximo (ColaPrioridad) –> ColaPrioridad
  Búsqueda (ColaPrioridad, Item) –> Boolean
  Cardinalidad (ColaPrioridad) –> Natural
  Copiar (ColaPrioridad) –> ColaPrioridad
  Mínimo (ColaPrioridad) –> Item
  Máximo (ColaPrioridad) –> Item

3. COLA DE PRIORIDAD

DEFINICION (II)

- *Árbol Mínimo (Máximo):*

Árbol en el que la etiqueta de cada nodo es menor (mayor) que la de los hijos.

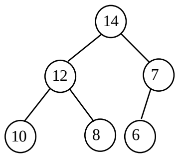


3. COLA DE PRIORIDAD

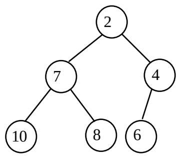
DEFINICION (III)

- *Heap Mínimo (Máximo):*

Árbol binario completo en que además es ARBOL MINIMO o MAXIMO.



HEAP MAXIMO



HEAP MINIMO

3. COLA DE PRIORIDAD

DEFINICION (IV)

- *Implementación Cola Prioridad:*
  - LISTA DESORDENADA:
    - INSERCION: O(1)
    - BORRADO: O(n)
  - LISTA ORDENADA: ascendente o descendentemente.
    - INSERCION: O(n)
    - BORRADO: O(1)
  - ARBOL BINARIO DE BUSQUEDA:
    - INSERCION: O(n)
    - BORRADO: O(n)

3. COLA DE PRIORIDAD

DEFINICION (V)

- *Implementacion Cola Prioridad:*
  - **HEAP o MONTICULO:**

INSERCION:	O(log n)
BORRADO:	O(log n)

3.1. HEAP MAXIMO (MINIMO)

INSERCION

- *METODO:*

1.- Insertar en la posición correspondiente para que siga siendo un árbol completo.

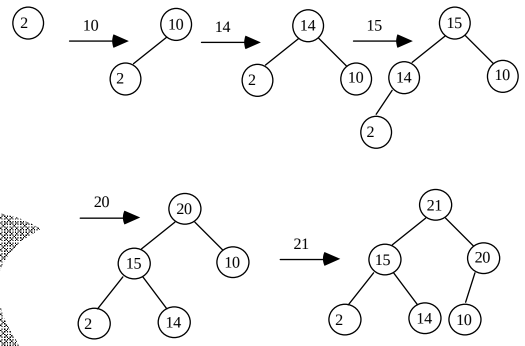
2.- Reorganizar para que cumpla las condiciones del HEAP:

- Comparar con el nodo padre: si no cumple las condiciones del árbol mínimo/máximo, entonces intercambiar ambos.

3.1. HEAP MAXIMO (MINIMO)

INSERCION. EJEMPLO

- *Insertar: 2, 10, 14, 15, 20 y 21 en un heap máximo inicialmente vacío*



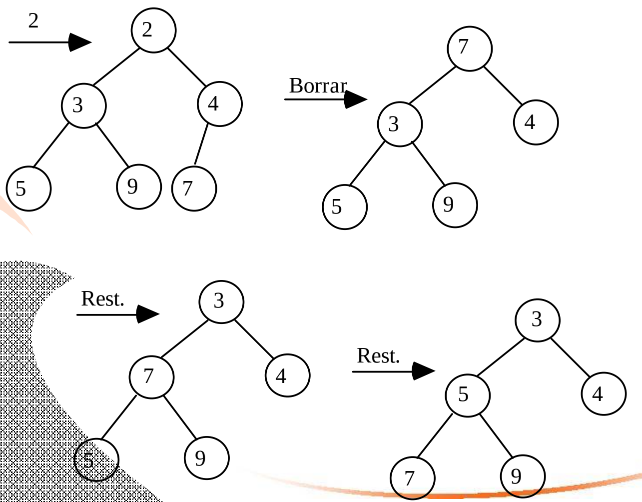
3.1. HEAP MAXIMO (MINIMO)

BORRADO.

- *METODO:*
  - Se sustituye la raíz con el elemento más a la derecha en el nivel de las hojas
  - Mientras no sea un HEAP se hunde ese elemento sustituyéndolo con el más pequeño (montículo mínimo) o el mayor (montículo máximo) de sus hijos

3.1. HEAP MAXIMO (MINIMO)

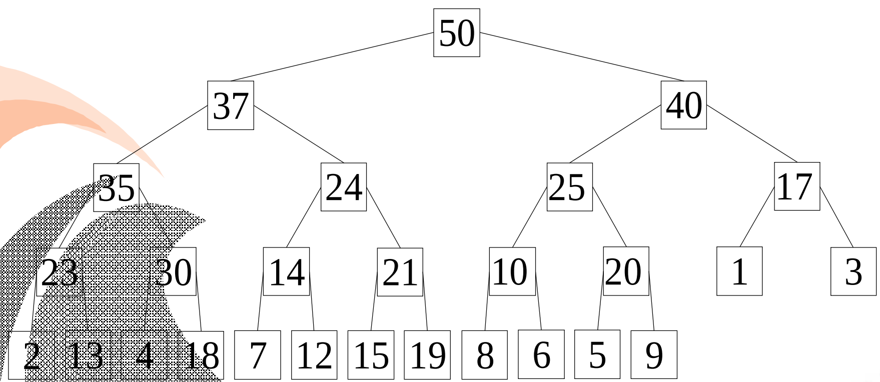
BORRADO. EJEMPLO



3.1. HEAP MAXIMO (MINIMO)

BORRADO. EJEMPLO II

- Realiza dos borrados sobre el siguiente montículo máximo.



3.1. HEAP MAXIMO (MINIMO)

INSERCIÓN. EJEMPLO II

- Sobre el resultado anterior, realiza las inserciones: 60, 36

3.1. HEAP MAXIMO (MINIMO)

REPRESENTACIÓN

- ENLAZADA: problema en inserción al necesitar realizar recorridos ascendentes.

- SECUENCIAL (en un vector):
  Hijos de p[i] son p[2•i] y p[2•i+1]. Padre de p[i] es p[i DIV 2] con DIV la división entera

3.1. HEAP MAXIMO (MINIMO)

APLICACIÓN HEAPSORT

- Algoritmo de ordenación de un vector de elementos

- METODO:
  1) Insertar los elementos en un HEAP
  2) Realizar borrados de la raíz del HEAP

- IMPLEMENTACIÓN (UN SÓLO VECTOR):
  1) Dejar parte izquierda del vector para el HEAP, y parte derecha para los elementos todavía no insertados.
  2) Borrar la raíz del HEAP llevándola a la parte derecha del vector.

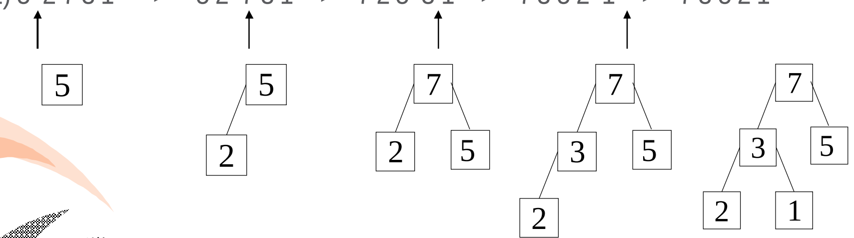
- COMPLEJIDAD:
  O (n log n)

3.1. HEAP MAXIMO (MINIMO)

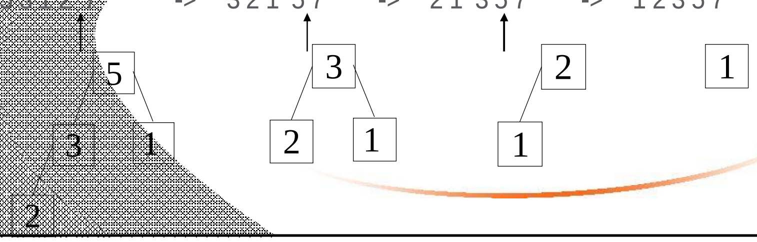
HEAPSORT. EJERCICIO

- Ordenar el vector 5 2 7 3 1 usando un heap máximo

1) 5 2 7 3 1 -> 5 2 7 3 1 -> 7 2 5 3 1 -> 7 3 5 2 1 -> 7 3 5 2 1



2) 5 3 1 2 7 -> 3 2 1 5 7 -> 2 1 3 5 7 -> 1 2 3 5 7



3.1. HEAP MAXIMO (MINIMO)

HEAPSORT. EJERCICIO

- Ordenar el vector 9 5 7 4 8 6 2 1 usando un heap mínimo

3.2. COLA DE PRIORIDAD DOBLE (DEAP)

DEFINICIÓN (I)

- Cola de Prioridad doble: cola de prioridad en la que se soporta la operación de borrado de la clave máxima y mínima.
- DEAP: Es un Heap que soporta las operaciones de cola de prioridad doble.

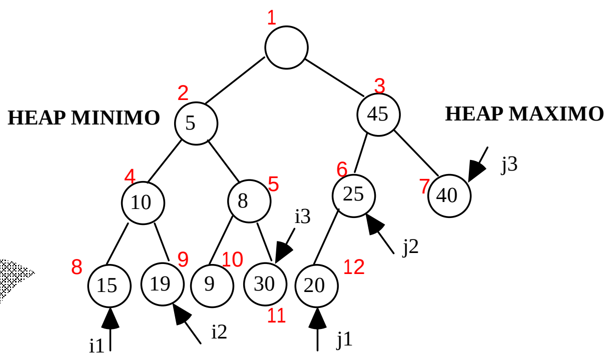
DEFINICION: es un árbol binario completo el cual o es vacío o satisface las siguientes propiedades:

1) La raíz no contiene elementos
2) El subárbol izquierdo es un HEAP mínimo
3) El subárbol derecho es un HEAP máximo
4) Si el subárbol derecho no es vacío:
   - Sea "i" cualquier nodo del subárbol izquierdo.
   - Sea "j" el nodo correspondiente en el subárbol derecho. Si "j" no existe, entonces sea el nodo del subárbol derecho que corresponde al padre de "i".

Entonces: clave (i) < clave (j)

3.2. COLA DE PRIORIDAD DOBLE (DEAP)

DEFINICIÓN (II)



3.2. COLA DE PRIORIDAD DOBLE (DEAP)

IMPLEMENTACIÓN

Igual que en un HEAP, sólo que la primera posición no se utilizará.

$$j = i + 2^{(\log_2 i) - 1}$$ ( ) parte entera

Si j > n **Entonces** j = j **DIV** 2

Ejemplo:
simétrico de i1 (i=8). $j = 8 + 2^{(\log_2 8) - 1} = 8 + 2^2 =$ **<u>12</u>**
simétrico de i2 (i=9). $j = 9 + 2^{(\log_2 9) - 1} = 9 + 2^2 = 13$. como j>n --> j = 13 DIV 2 = **<u>6</u>**

3.2. COLA DE PRIORIDAD DOBLE (DEAP)

INSERCIÓN

1) Se inserta el elemento en el siguiente índice del árbol completo

2) Se compara el nodo insertado con el nodo simétrico correspondiente, realizando el intercambio en caso que no se cumpla la condición 4 de la definición del DEAP:

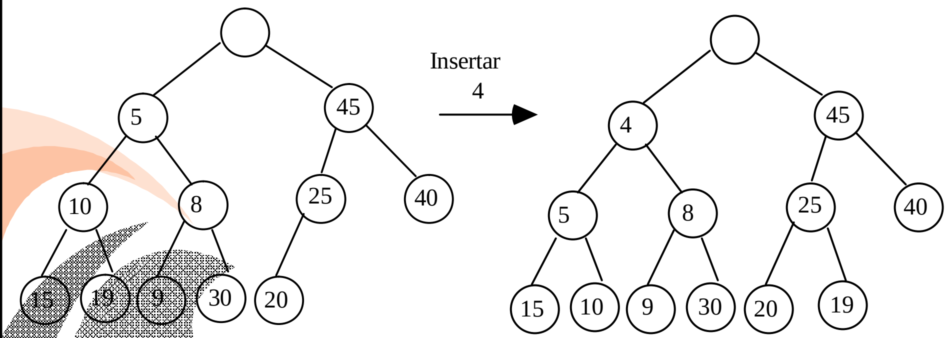
$$i = n - 2^{(\log 2\ n) - 1}$$ ( ) parte entera

3) Actualizar el HEAP mediante el proceso de "ascensión" del elemento insertado.

Ejemplo:
simétrico de j1 (j=12). i = 12 - 2 $^{(\log_2 12) - 1}$ = 12 - 2^2 = **8**

3.2. COLA DE PRIORIDAD DOBLE (DEAP)

INSERCIÓN



3.2. COLA DE PRIORIDAD DOBLE (DEAP)

INSERCIÓN

- Sobre el DEAP anterior insertar: 35, 7, 50, 17, 12, 27, 55

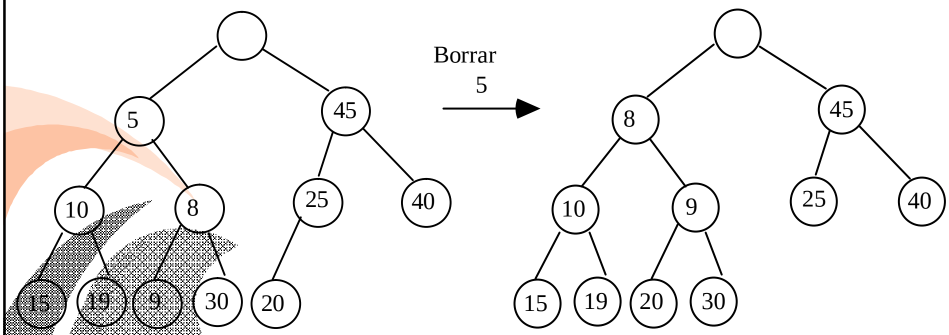
3.2. COLA DE PRIORIDAD DOBLE (DEAP)

BORRADO

1) Intercambiar la raíz a borrar del HEAP con el elemento más a la derecha del último nivel del árbol, y borrar éste.
2) Actualizar el HEAP, "hundiendo" la clave intercambiada.
3) Comprobar que la clave intercambiada no incumpla la condición del DEAP con su correspondiente nodo simétrico
4) Actualizar el montículo en el que quede la clave intercambiada

3.2. COLA DE PRIORIDAD DOBLE (DEAP)

BORRADO



3.2. COLA DE PRIORIDAD DOBLE (DEAP)
BORRADO

MONTÍCULO DOBLE: Borrar los elementos mínimo y máximo de forma sucesiva.