Tema 2. MONTÍCULOS BINARIOS

Estructuras de Datos y Algoritmos II
Grado en Ingeniería Informática

M José Polo Martín
mjpolo@usal.es

Universidad de Salamanca

curso 2024-2025

DEPARTAMENTO DE INFORMÁTICA Y AUTOMÁTICA

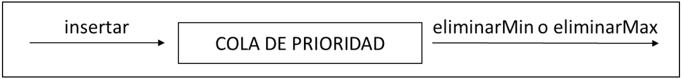
Contenido

1 Introducción

Cola de PRIORIDAD

Tipo abstracto de datos que almacena una colección de elementos y que permite al menos las dos operaciones siguientes:

- Insertar un elemento
- Buscar, devolver y eliminar el elemento con valor mínimo|máximo en alguno de sus campos de información(clave)



Aplicaciones y posibles implementaciones

- Aplicaciones:
  - Colas de impresión con prioridades
  - Planificador sistema operativo en entorno multiusuario
  - Nuevo Algoritmo de Ordenación
  - Implantación eficiente de algunos algoritmos de grafos (tema 4)
- Posibles implementaciones:
  - Listas enlazadas
  - Listas enlazadas clasificadas
  - Árbol binario de búsqueda
  - **Montículo binario, montículo o *heap***

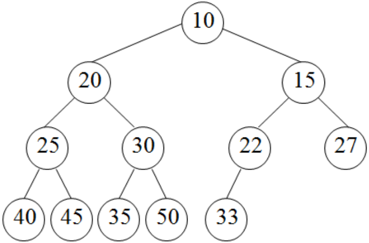
2 Nivel abstracto o de definición

Montículo Binario

Árbol binario casi-completo (completo, con la posible excepción del nivel más bajo, el cual se llena de izquierda a derecha) en el cual, para todo nodo n se cumple la siguiente condición:
“la clave en el padre de n es menor (o igual) que la clave de n, con la excepción obvia de la raíz (que no tiene nodo padre)”

- Observaciones:
  - Regularidad de un a.b. casi-completo $\Rightarrow$ se puede representar mediante una matriz sin necesidad de recurrir a punteros
  - El elemento con valor mínimo siempre se encuentra en la raíz
  - Puede redefinirse cambiando la condición de prioridad de mínimo a máximo

Ejemplo



?	**10**	**20**	**15**	**25**	**30**	**22**	**27**	**40**	**45**	**35**	**50**	**33**			
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**			

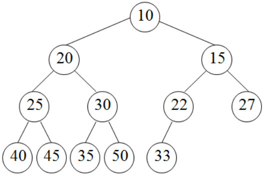
Propiedades

Propiedad de orden

El elemento con valor mínimo siempre se encuentra en la raíz

Propiedad de la estructura

Para cualquier elemento en la posición $\mathbf{i}$ de la matriz, el hijo izquierdo está en la posición $\mathbf{2i}$, el hijo derecho está en la posición siguiente al hijo izquierdo $\mathbf{(2i+1)}$ y el padre está en la posición $\mathbf{[i/2]}$



?	10	20	15	25	30	22	27	40	45	35	50	33			
0	1	2	3	4	5	6	7	8	9	10	11	12			

Especificación de operaciones

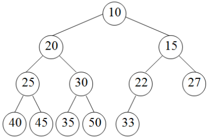
- **inserta(x,m):** Inserta un nuevo elemento **x** en el montículo **m**
- **eliminaMin(m):** Elimina y devuelve el elemento del montículo **m** con valor mínimo en su campo clave
- **decrementaClave(p,x,m):** Reduce el valor de la clave del elemento de la posición **p** del montículo **m** una cantidad positiva **x**
- **incrementaClave(p,x,m):** Aumenta el valor de la clave del elemento de la posición **p** del montículo **m** una cantidad positiva **x**
- **construirMonticulo(m):** Construye un montículo binario a partir de una colección de elementos

3 Nivel de representacion o implementación

- Un montículo binario puede representarse mediante una matriz y un entero que indique el tamaño actual del montículo

Algorithm Declaraciones básicas

constantes
  MAX = 100
tipos
tipoElemento = **registro**
  clave : *tipoClave*
  información : *tipoInformación*
fin registro
tipoMontículo = **registro**
  elementos : *matriz*[0..*MAX*] de tipoElemento
  tamaño : *entero*
fin registro



?	10	20	15	25	30	22	27	40	45	35	50	33			
0	1	2	3	4	5	6	7	8	9	10	11	12			

Inserción de un elemento en el montículo

- Añadir un elemento x al montículo manteniendo las propiedades de estructura y orden que los definen
- Proceso:
  1. Añadir un nuevo nodo (hueco) en la siguiente posición disponible del a.b. casi-completo.
  2. Distinguir los siguientes casos:
     - Si x se puede asignar al hueco manteniendo la propiedad de orden del montículo, se asigna y finaliza el proceso
     - En otro caso se desliza el elemento del nodo padre al hueco, subiendo el hueco hacia la raíz, hasta poder asignar x al hueco

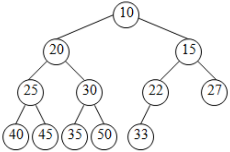
Estrategia de Filtrado Ascendente

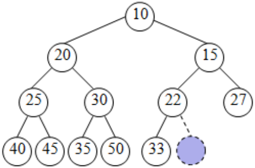
El nuevo elemento se filtra en el montículo hasta encontrar su posición correcta

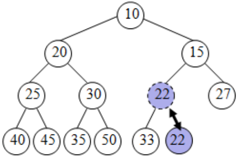
Ejemplo filtrado ascendente

insertar un nuevo elemento con clave 5 en el montículo

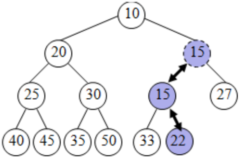
añadir hueco (m.tamaño++) ⟹⟹ 5 < 22 (intercambio) ⟹⟹ 5 < 15 (intercambio)

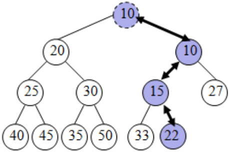


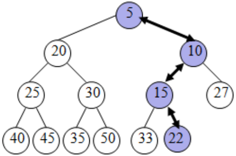




⟹ 5 < 10 (intercambio) ⟹⟹ **nuevo mínimo (¡¡¡ No siempre!!!)** ⟹







Algoritmo de INSERCIÓN

Algorithm inserta(x:**tipoElemento**, **referencia** m:**tipoMontículo**)

Entrada: x elemento a insertar
Salida: el montículo m con el nuevo elemento x

$hueco : entero$
si $m.tamaño \geq MAX$ **entonces**
    implementar según especificación y diseño ->"montículo lleno"
si no
    $m.tamaño \leftarrow m.tamaño + 1$
    $hueco \leftarrow m.tamaño$
    mientras $m.elementos[hueco \div 2].clave > x.clave$ **hacer**
        $m.elementos[hueco] \leftarrow m.elementos[hueco \div 2]$
        $hueco \leftarrow hueco \div 2$
    fin mientras
    $m.elementos[hueco] \leftarrow x$
fin si

Observaciones

- Si el elemento a insertar es el nuevo mínimo, el hueco debe subir hasta la raíz. En el momento que i tome el valor 1 habrá que romper el ciclo mientras
- Dos soluciones:
  1. Comprobación explícita en la condición del bucle ($i \neq 1$ AND ...)
  2. Asignar un valor (centinela) en la posición 0 del array (menor o igual que cualquier elemento del montículo) que asegure que el bucle termina
- La operación insertar es $O(\log n)$ ¿Puedes justificar esta afirmación?

Eliminación del elemento mínimo del montículo

- Eliminar del montículo el elemento que contiene la clave mínima manteniendo las propiedades de estructura y orden que los definen.
- Proceso:
  1. Se elimina el valor del elemento con clave mínima creando un hueco en la raíz
  2. Se reduce el tamaño del montículo en una unidad.
  3. El que era ultimo elemento debe moverse a otro lugar dentro del montículo conservando la propiedad de orden.
  4. Distinguir los siguientes casos:
     - Si puede asignarse al hueco manteniendo la propiedad de orden del montículo, se asigna y finaliza el proceso
     - En otro caso se intercambia el hueco con el menor de sus dos hijos, empujando el hueco hacia abajo un nivel, hasta poder asignar el último elemento al hueco

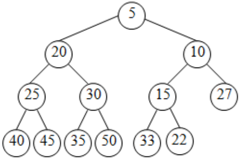
Estrategia de Filtrado Descendente

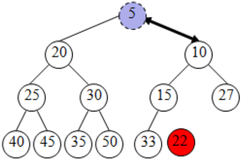
Se comparan la clave de un elemento con la del menor de sus dos hijos...

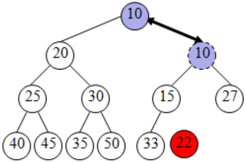
Ejemplo filtrado descendente

eliminar el elemento mínimo del montículo

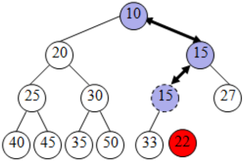
hueco en la raíz (m.tamaño–) $\Longrightarrow\Longrightarrow$ recolocar último elemento $\Longrightarrow\Longrightarrow$ $22 > 10$ (intercambio)

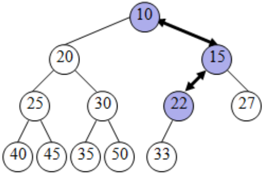






$\Longrightarrow$ $22 > 15$ (intercambio) $\Longrightarrow\Longrightarrow$ $22 < 33$ **(¡¡¡ Fin filtrado!!!)**





Algoritmo de ELIMINACIÓN

Utiliza estrategia de filtrado descendente

Algorithm eliminarMin(**referencia** m:**tipoMontículo**)

Entrada: el monticulo m
Salida: el elemento mínimo y el montículo m actualizado

```
i, hijo : entero
mínimo : tipoElemento
si vacio(m) entonces
    implementar según especificación y diseño                ->"montículo vacio"
si no
    mínimo ← m.elementos[1]
    último ← m.elementos[m.tamaño]
    m.tamaño ← m.tamaño − 1
    hueco ← 1
    finFiltrado ← FALSO
    mientras (2 ∗ hueco ≤ m.tamaño Y NO finFiltrado) hacer
        hijo ← 2 ∗ hueco
        si hijo ≠ m.tamaño entonces
            si m.elementos[hijo + 1].clave < m.elementos[hijo].clave entonces
                hijo ← hijo + 1
            fin si
        fin si
        si m.elementos[hijo].clave < último.clave entonces
            m.elementos[hueco] ← m.elemento[hijo]
            hueco ← hijo
        si no
            finFiltrado ← VERDADERO
        fin si
    fin mientras
    m.elementos[hueco] ← último
    devolver  mínimo
fin si
```

Observaciones

- En el filtrado descendente se compara la clave del último elemento con la del menor de los hijos del hueco. Debe tenerse en cuenta que el último nodo interno puede tener un único hijo
- ¿De qué orden es la operación **eliminarMin**?
- Las operaciones **decrementarClave** e **incrementarClave** necesitarán las estrategias de filtrado ascendente y descendente, respectivamente, para mantener la propiedad de orden si esta se pierde
- Dos soluciones para la operación **construirMontículo** que toma como entrada n claves y las coloca en un montículo vacío

  Solución 1: Realizar n operaciones insertar sucesivas $\Rightarrow O(n \log n)$

  Solución 2: Existe otra solución de $O(n)$ (Ver ejemplo)

Algoritmo Construir Montículo (solución 2)

- Colocar las n claves dentro del árbol en cualquier orden, manteniendo la propiedad de la estructura
- Realizar un filtrado descendente de todos los nodos internos para conseguir la propiedad de orden y obtener un montículo

---

Algorithm construirMontículo(**referencia** m:**tipoMontículo**)

Entrada: montículo *m* con los n elementos asignados respectando la propiedad de la estructura
Salida: montículo *m* respetando tanto la propiedad de la estructura como la de orden

1: i, n : *entero*
2: $n \leftarrow m.tamaño$
3: **para** $i \leftarrow n \div 2$ **bajando hasta** 1 **hacer**
4:     *filtradoDescendente*(m, i)
5: **fin para**

En la línea 4 se invoca a una función **genérica** que realiza el **filtrado descendente** del elemento de la posición ***i*** en el montículo ***m***

Análisis del Algoritmo Construir Montículo

Solución 2

- Se realiza el filtrado descendente de todos los nodos internos del montículo binario
- En el peor de los casos, cada uno de estos nodos debe descender desde su nivel a un nivel hoja, es decir, como máximo su altura
- Un árbol binario **completo** de altura ***h*** tiene

nodos	*altura*	*patrón*
1	h	$n_0 = 1 = 2^{h-h}$
2	$h-1$	$n_1 = 2 = 2^{h-(h-1)}$
2^2	$h-2$	$n_2 = 4 = 2^{h-(h-2)}$
...	...	
2^{h-1}	1	$n_{(i-1)} = 2^{h-(1)}$
2^h	0	$n_i = 2^{h-(0)}$

- La suma de las alturas de todos los nodos internos de un árbol binario completo(peor de los casos) es

$$1 \cdot h + 2 \cdot (h-1) + 4 \cdot (h-2) + ... + 2^{h-1} \cdot 1$$

- Esta suma indica el número de veces que se ejecuta la instrucción barómetro y, por tanto, el orden del algoritmo

Análisis del Algoritmo Construir Montículo

- Calculamos la suma multiplicando por dos la ecuación anterior y restando (algunos términos se cancelan)

$$\begin{array}{rcccccccccccc}
2S & = & & & 2h & + & 4(h\text{-}1) & + & 8(h\text{-}2) & +...+ & 2^h \\
S & = & h & + & 2(h\text{-}1) & + & 4(h\text{-}2) & + & 8(h\text{-}3) & +...+ & \\
\hline
S & = & -h & + & 2 & + & 4 & + & 8 & +...+ & 2^h
\end{array}$$

$$t(n) = -h + 2 + 4 + 8 + ... + 2^h = -h + 2^{h+1} - 2$$

- Teniendo en cuenta que un árbol casi-completo tiene entre 2^h y 2^{h+1} nodos (transparencia 11, tema 1), la suma que obtenemos es de orden $O(n)$ siendo n el número de nodos

$$t(n) = 2^{h+1} - 2 - h = n - (2 + h) \in \mathbf{O(n)}$$

4 Ordenación por montículos

- Los montículos se pueden utilizar como método de ordenación interna con un comportamiento $O(nlogn)$
- El algoritmo basado en esta idea se denomina ordenación por montículo (*heapsort*) y se basa en:
  - **Construir** un montículo binario de n elementos
  - Efectuar **n** operaciones **eliminarMin**: los elementos más pequeños dejan primero el montículo
  - Cada operación eliminarMin contrae el montículo en uno. Si se aprovecha la que era última celda del montículo para almacenar el elemento eliminado, al final de la operación las n celdas del array contienen los elementos en orden decreciente

Análisis de algoritmo de ordenación por montículos

- Crear montículo requiere un tiempo lineal, $O(n)$, y genera un montículo de altura $[logn]$
- La operación eliminarMin se ejecutará $(n-1)$ veces y filtra la raíz a lo largo de un camino de longitud máxima $logn$, por tanto, en el peor caso requiere un tiempo que es $O(logn)$
- Habrá también $n-1$ operaciones de asignación que requieren un tiempo constante, $O(1)$
- Por tanto el tiempo, $t(n)$, necesario para ordenar los n elementos verifica

$$t(n) \in O(n) + (n-1)log(n) + (n-1)O(1) \in \mathbf{O(nlogn)}$$

Observaciones

- Para obtener los elementos en orden creciente, basta con cambiar la propiedad de orden de forma que el padre tenga una clave mayor que los hijos y considerar la operación eliminarMax

Definición alternativa de Montículo Binario

Árbol binario casi-completo en el cual, para todo nodo n se cumple que, la clave en el padre de n es **mayor** (o igual) que la clave de n, con la excepción obvia de la raíz"

- Cambia la propiedad de orden del montículo $\Rightarrow$ el elemento con valor **máximo** siempre se encuentra en la raíz

Ejercicios

Ejercicio 1: Analizar qué ocurre en el montículo de la figura 1 en los siguientes casos:

1. Cuando se inserta un nuevo nodo con valor 9 para su campo clave
2. Cuando se elimina el elemento mínimo
3. Cuando se incrementa en 2 la clave del elemento situado en la posición 10
4. Cuando se decrementa en 40 la clave del elemento situado en la posición 23

Ejercicio 2: Dado el montículo binario de la figura 2 dibujar el montículo, explicando brevemente el proceso, después de

1. Realizar una operación que decremente en 620 el valor del elemento de la posición 18.
2. Realizar una operación que incremente en 400 el valor del elemento de la posición 17.

Figura 1

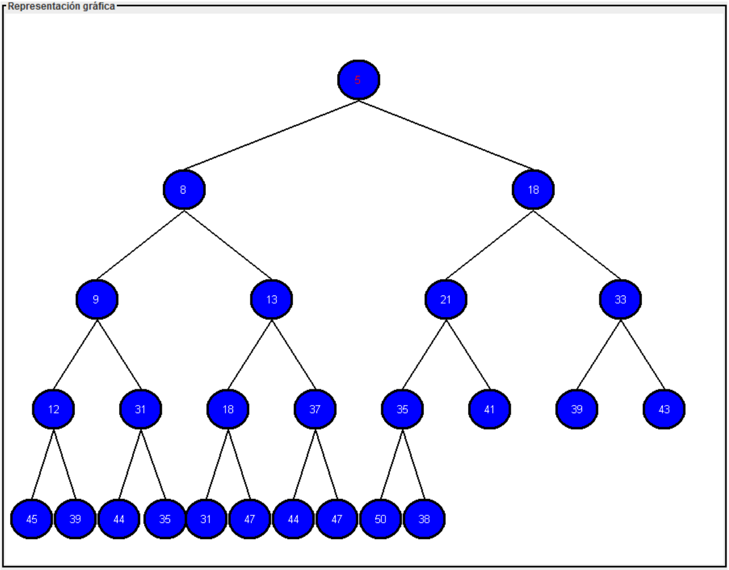
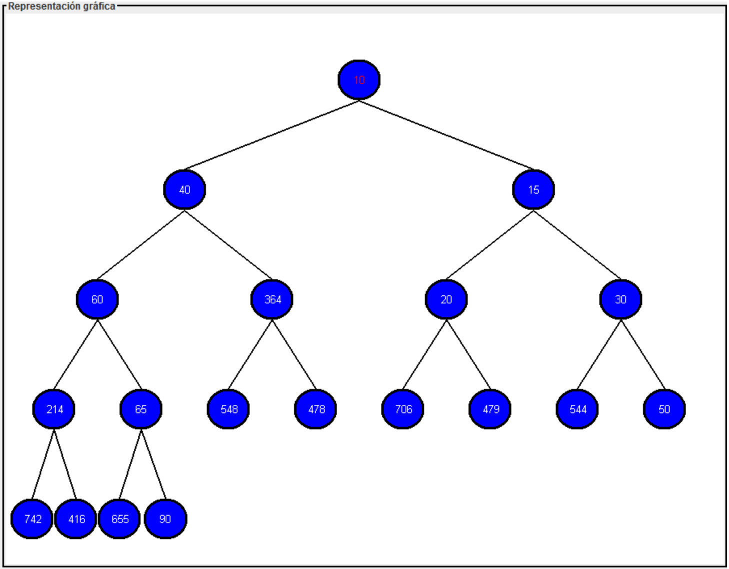

Representación gráfica
5
8
18
9
13
21
33
12
31
18
37
35
41
39
43
45
39
44
35
31
47
44
47
50
38

Figura 2


Representación gráfica
10
40
15
60
364
20
30
214
65
548
478
706
479
544
50
742
416
655
90

Ejercicio 3

Considerando una montículo binario en el que se han introducido aleatoriamente 10 claves manteniendo la propiedad de la estructura.

1. ¿Cuantas veces es necesario aplicar el algoritmo de filtrado descendente para conseguir la propiedad de orden?
2. Suponiendo que inicialmente el contenido del array que representa el montículo es el que muestra la figura, aplicar el algoritmo de filtrado descendente las veces necesarias para completar la tabla de forma que muestre el contenido del array después de cada filtrado.

Contenido inicial→	?	99	35	127	191	198	304	54	107	37	115

Ejercicio 4

Utilizando la aplicación RAED estudiar y analizar los procesos de filtrado ascendente y descendente aplicando las operaciones típicas de inserción y eliminación a diferentes ejemplos de montículos binarios.

Nota: En el apartado dedicado a la aplicación RAED se proporcionan dos ficheros con los montículos correspondientes a los ejercicios 1 y 2.