

Estructuras de Datos y Algoritmos

TDA Cola de Prioridad

- Cada elemento tiene asignada una prioridad
- Los elementos de mayor (menor) prioridad son procesados primero
- Los elementos de igual prioridad son procesados por orden de llegada
- Consideraciones:
 - ♦ *cola*: entidades esperando por un servicio
 - ♦ *prioridad*: las entidades NO son atendidas por orden de llegada

1er. cuatrimestre 2002

EDyA

2

TDA Cola de Prioridad

CP: ColaPrioridad

CP = (a₁₁ a₁₂ a₁₃ a₂₁ a₃₁ a₃₂ ...)

Prioridad (a_{ij}) = Prioridad (a_{i,j+1})

Prioridad (a_{ij}) > Prioridad (a_{i+1,k})

Prioridad (a₁₁) es maxima/ minima

1er. cuatrimestre 2002

EDyA

3

TDA Cola de Prioridad

- ♦ CrearColaP : \longrightarrow ColaPrioridad
- ♦ Insertar: Item x Prioridad x ColaPrioridad \longrightarrow ColaPrioridad
- ♦ EliminarMin: ColaPrioridad \longrightarrow ColaPrioridad
Pre: not Vacía(CP)
- ♦ RecuperarMin: ColaPrioridad \longrightarrow Item
Pre: not Vacía(CP)
- ♦ Vacía: ColaPrioridad \longrightarrow Booleano

1er. cuatrimestre 2002

EDyA

4

TDA Cola de Prioridad

- Podemos considerar la operación EliminarMin definida como sigue

EliminarMin:

ColaPrioridad \longrightarrow Item x ColaPrioridad

Pre: not Vacía(CP)

- La prioridad puede ser un valor asignado directamente o calculado por una función aplicada sobre algunos campos del registro de datos.

1er. cuatrimestre 2002

EDyA

5

TDA Cola de Prioridad *Posibles implementaciones*

- Lista ordenada por prioridad y orden de llegada - los elementos con la misma prioridad se operan en orden -
- Lista de colas - cada elemento de la lista tiene una prioridad y una cola de los elementos con esa prioridad
- Cualquier otra estructura que facilite la operación para encontrar el mínimo elemento (heap)

1er. cuatrimestre 2002

EDyA

6

TDA Conjunto

- Un conjunto es una colección de miembros.
- Cada miembro puede ser un conjunto ó
- Cada miembro puede ser un átomo (elto. primitivo).
- Todos los miembros son distintos. Ningún conjunto puede contener 2 copias del mismo elemento.

1er. cuatrimestre 2002

EDyA

7

TDA Conjunto: Operaciones



- Definir: \longrightarrow Conjunto
- Union: Conjunto x Conjunto \longrightarrow Conjunto
- Interseccion: Conjunto x Conjunto \longrightarrow Conjunto
- Diferencia: Conjunto x Conjunto \longrightarrow Conjunto
- Igual: Conjunto x Conjunto \longrightarrow Boolean
- Asignar: Conjunto \longrightarrow Conjunto
- Vaciar: Conjunto \longrightarrow Conjunto
- Miembro: Conjunto x elemento \longrightarrow Boolean
- Vacía: Conjunto \longrightarrow Boolean

1er. cuatrimestre 2002

EDyA

8

TDA Conjunto: Implementación

- Vector de bits
 - ☞ conjunto universal pequeño con elementos de tipo ordinal (enteros, caracteres, enumerados)
 - ☞ el i-ésimo bit es verdadero 
i es un elemento del conjunto
 - ☞ referencia directa al bit apropiado 
mejor tiempo de la operaciones

1er. cuatrimestre 2002

EDyA

9

TDA Conjunto: Implementación

- Listas
 - ☞ mas general
 - ☞ conjunto universal sin restricciones
 - ☞ puede utilizarse:
 - listas
 - listas ordenadas
 - punteros
- Arboles (??)

1er. cuatrimestre 2002

EDyA

10

TDA Mapeo

- Una Correspondencia ó Mapeo es una función de elementos de un tipo_dominio a elementos de un tipo_rango
- $M(d) = r$ expresa que el mapeo M asocia el elemento r del rango con el elemento d del dominio

1er. cuatrimestre 2002

EDyA

11

TDA Mapeo: Operaciones

- Definir: \longrightarrow Mapeo
- Asignar: Mapeo \longrightarrow Mapeo
- Vaciar: Mapeo \longrightarrow Mapeo
- Valor: Mapeo x Dominio \longrightarrow Rango x Boolean
Devuelve el valor de rango asociado al valor de dominio dado. Si el dominio es indefinido devuelve *false*, sino *true*.
- Asociar: Mapeo x Dominio x Rango \longrightarrow Mapeo
Asocia el valor dado por el rango al valor del dominio. Si ya existe una asociación para ese dominio, la reemplaza
- Dominio: Mapeo \longrightarrow Conjunto
- Rango: Mapeo \longrightarrow Conjunto

1er. cuatrimestre 2002

EDyA

12

TDA Mapeo: *Implementación*

- *Arreglos*
 - dominio finito y ordinal
 - se requiere de un valor del tipo_rango que signifique “indefinido”

```
type
  mapeo = array[tipo_dominio] of tipo_rango
```

1er. cuatrimestre 2002

EDyA

13

TDA Mapeo: *Implementación*

- *Listas*
 - $(d_1, r_1), (d_2, r_2), \dots, (d_k, r_k)$
 - d_1, d_2, \dots, d_k son los miembros del dominio
 - r_i valor asociado a d_i $i=1, 2, \dots, k$.

```
tipo elemento de la lista
  tipo_elemento = record
    dominio : tipo_dominio;
    rango : tipo_rango
  end;
```

1er. cuatrimestre 2002

EDyA

14

Análisis de algoritmos

- Se analiza el algoritmo en un marco formal
- Se necesita un modelo de computación:
 - ☞ las instrucciones se ejecutan de modo secuencial
 - ☞ cada instrucción sencilla tarda exactamente una unidad de tiempo (asignación, comparación, adición)
 - ☞ no se indica la unidad utilizada
 - ☞ suponemos una memoria infinita
- Se analiza el tiempo de ejecución

1er. cuatrimestre 2002

EDyA

15

INSERCIÓN DIRECTA

```
...
(1) for j:= 2 to n do begin                (c1; n)
(2)   clave := A[j];                       (c2; n-1)
(3)   i := j-1;                             (c3; n-1)
(4)   while (i>0) and (A[i] > clave)      (c4; ∑j:2..n tj)
       do begin
(5)     A[i+1] := A[i];                     (c5; ∑j:2..n (tj-1))
(6)     i := i-1                            (c6; ∑j:2..n (tj-1))
       end;
(7)   A[i+1] := clave;                       (c7; n-1)
end ...
```

1er. cuatrimestre 2002

EDyA

16

INSERCIÓN DIRECTA

$t_j =$ veces que se ejecuta el while para cada j

- caso 1: los elementos están ordenados
 - $t_j=1 \quad \forall j:2..n$ (4) $c_4 \cdot \sum_{j:2..n} 1$
 - $T(n) = a n + b$
- caso 2: los elementos están inversamente ordenados
 - $t_j=j \quad \forall j:2..n$ (4) $c_4 \cdot \sum_{j:2..n} j$
 - $T(n) = a n^2 + b n + c$

1er. cuatrimestre 2002

EDyA

17

Algoritmos: *Tiempo de Ejecución*

- se define como una función de la entrada de datos \longrightarrow su tamaño n
- en general se considera $T(n)$ como el tiempo de ejecución del peor caso \curvearrowright el máximo valor para entradas de tamaño n
- se puede calcular $T_{\text{prom}}(n)$ (mucho más difícil de calcular)

1er. cuatrimestre 2002

EDyA

18

Algoritmos: *Tiempo de Ejecución*

- INSERCIÓN DIRECTA

$$\frac{a n^2}{\uparrow} + b n + c$$

- para tamaños de entrada grandes
 - términos de menor orden insignificantes
 - coeficientes constantes poco significativos

velocidad de crecimiento



eficiencia
asintótica del
algoritmo

1er. cuatrimestre 2002

EDyA

19

Algoritmos: *Notación Asintótica O*

- $T(n)$ es de $O(n^2)$ $\Rightarrow n^2$ es cota superior

$$\exists c \text{ y } n_0 \text{ tales que para } n \geq n_0, T(n) \leq cn^2$$

- $T(n)$ es de $O(f(n))$ $\Rightarrow f(n)$ es cota superior

$$\exists c \text{ y } n_0 \text{ tales que para } n \geq n_0, T(n) \leq cf(n)$$

velocidad de crecimiento

1er. cuatrimestre 2002

EDyA

20

Algoritmos: *Tiempo de ejecución*

- $T(n) = 3n^3 + 2n^2 \Rightarrow O(n^3)$

tomando $n_0 = 0$ y $c = 5$

- $T(n) = 3^n$ no es $O(2^n)$

se prueba por el absurdo

1er. cuatrimestre 2002

EDyA

21

Tiempo de ejecución: *reglas generales*

- Regla 1: *regla de la suma*

se toma el máximo de los ordenes de ejecución.

Sean p, q dos fragmentos de programa que se ejecutan en forma consecutiva,

$$T_p(n) \Rightarrow O(f(n))$$

$$T_q(n) \Rightarrow O(g(n))$$

$$T_p(n) + T_q(n) \Rightarrow O(\max(f(n), g(n)))$$

1er. cuatrimestre 2002

EDyA

22

Tiempo de ejecución: *reglas generales*

- Regla 2: ciclo *FOR*

el tiempo de ejecución de las instrucciones internas, incluyendo condiciones, por el número de iteraciones.

```
For j:= n downto 1 do
  a[j] := j+3;
```

la asignación tarda tiempo 1 $\Rightarrow O(1)$

son n iteraciones $\Rightarrow O(n)$

1er. cuatrimestre 2002

EDyA

23

Tiempo de ejecución: *reglas generales*

- Regla 3: ciclos *FOR anidados*

Analizarlos de adentro hacia fuera. El tiempo de ejecución total resulta de sumar, en cada iteración los tiempos empleados en ejecutar el cuerpo del ciclo.

```
For i:=1 to n-1 do
```

```
  for j:=1 to n do
```

```
    O(1)
```

$$T(n) = \sum_{i=1, n-1} n = n \cdot (n-1) = n^2 - n \Rightarrow O(n^2)$$

1er. cuatrimestre 2002

EDyA

24

Tiempo de ejecución: reglas generales

- For i:= 1 to n do
 $a[i] := 0;$ } $O(n)$
- for i:= 1 to n do
 for j:= 1 to n do
 $a[i] := a[i] + a[j] + i + j;$ } $O(n^2)$

$$O(\max(n, n^2)) = O(n^2)$$

1er. cuatrimestre 2002

EDyA

25

Tiempo de ejecución: reglas generales

- Regla 4: Sentencia IF

```
if cond
then S1
else S2
```

el tiempo de ejecución nunca es más grande que el tiempo empleado por la condición más el mayor de los tiempos de S1 y S2.

1er. cuatrimestre 2002

EDyA

26

Tiempo de ejecución: subprogramas

- Subprogramas no-recursivos
 se calcula el tiempo de cada procedimiento / función de adentro hacia afuera.
- Subprogramas recursivos
 - se asocia a cada procedimiento una función de tiempo $T(n)$ desconocida
 - se obtiene una recurrencia para $T(n)$ (una ecuación para $T(n)$ en función de $T(k)$)

1er. cuatrimestre 2002

EDyA

27

Tiempo de ejecución: ejemplo

```
Function suma (n: integer) : integer;
var i, suma_parcial: integer;
begin
(1) suma_parcial := 0;
(2) for i:= 1 to n do
(3) suma_parcial := suma_parcial + i*i*i;
(4) suma := suma_parcial
end
```

$$T(n) = 4n + 2 \longrightarrow O(n)$$

1er. cuatrimestre 2002

EDyA

28

Tiempo de ejecución: ejemplo

```
Function factorial (n: integer): integer;
begin
if n <= 1      O(1)
then factorial := 1      O(1)
else factorial := n * factorial (n-1)  ??
End
```

$$T(n) = \begin{cases} c + T(n-1) & \text{si } n >= 1 \\ d & \text{si } n <= 1 \end{cases} \longrightarrow O(n)$$

1er. cuatrimestre 2002

EDyA

29

Tiempo de ejecución: logaritmos

- Un algoritmo es $O(\log n)$ si usa un tiempo constante en dividir el problema en partes (más o menos iguales) \longrightarrow Búsqueda binaria
- los algoritmos que utilizan la estrategia “divide y vencerás” utilizan tiempo $O(n \log n)$
- divide y vencerás:
 se parte el problema en 2 subproblemas más o menos iguales.
 Se resuelven los subproblemas en forma recursiva.
 Se unen las 2 soluciones para llegar a la solución global

1er. cuatrimestre 2002

EDyA

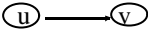
30

Grafos : *modelo para representar relaciones arbitrarias entre objetos de datos*

- $G = (V, E)$
 - V conjunto de *vértices* / E conjunto de *arcos* $\langle u, v \rangle$
- Grafo no dirigido (*grafo*) :
 - $\langle u, v \rangle$ par no ordenado ($\langle u, v \rangle = \langle v, u \rangle$)



- Grafo dirigido (*digrafo*) :
 - $\langle u, v \rangle$ par ordenado ($\langle u, v \rangle \neq \langle v, u \rangle$)



1er. cuatrimestre 2002

EDyA

31

Grafos: *Definiciones*

- máximo nro. de arcos en
 - grafo: $mv = n(n-1)/2$
 - digrafo: $mv = n(n-1)$
 - (n vértices)
- nota: no puede haber múltiples ocurrencias del mismo arco (multigrafo)
- un grafo/digrafo con n vértices y mv arcos se dice *completo*

1er. cuatrimestre 2002

EDyA

32

Grafos : *Definiciones*

- Si $\langle u, v \rangle$ pertenece al grafo G
 - u y v son vértices *adyacentes*
 - el arco $\langle u, v \rangle$ es *incidente* en u y en v
- Si $\langle u, v \rangle$ pertenece al digrafo G
 - u es el origen
 - v es el destino
 - v es *adyacente* a u
 - el arco $\langle u, v \rangle$ es *incidente* en u y v
- El *grado* de un vértice v es el nro. de arcos incidentes en él.

1er. cuatrimestre 2002

EDyA

33

Grafos: *Definiciones*

- En un digrafo :
 - grado de entrada de v
 - grado de salida de v
- En un grafo/digrafo G con e arcos, n vértices y $d_i =$ grado del vértice i

$$e = (\sum d_i) / 2, \text{ para } i:1..n$$

1er. cuatrimestre 2002

EDyA

34

Grafos : *Definiciones*

- Un subgrafo G' de G es tal que
 - $\Rightarrow V(G') \subseteq V(G)$
 - $\Rightarrow E(G') \subseteq E(G)$
- Un camino del vértice u al vértice v es la secuencia de vértices u, i_1, i_2, \dots, i_k, v , donde: $\langle u, i_1 \rangle, \langle i_1, i_2 \rangle, \dots, \langle i_k, v \rangle$ son arcos
- Un camino de longitud n del vértice u al vértice v es un camino en el que intervienen $n+1$ vértices y n arcos

1er. cuatrimestre 2002

EDyA

35

Grafos: *Definiciones*

- Un camino simple es aquel en el que todos los vértices que intervienen son distintos (excepto posiblemente u y v)
- Un *ciclo* es un camino en el cual el primer vértice y el último son el mismo
- Si existe un camino del vértice u al vértice v, decimos que v es alcanzable desde u, (los vértices u y v están conectados).
- Un grafo es *conexo* si hay un camino de u a v, $\forall u, v \in V(G)$

1er. cuatrimestre 2002

EDyA

36

Grafos : *Definiciones*

- Un digrafo con la propiedad anterior es fuertemente conexo
- Si el digrafo G no es FC, pero el grafo G' subyacente es conexo, \implies **G es débilmente conexo**
- Un grafo G es completo si hay un arco entre cualquier par de vértices
- Un grafo G se dice etiquetado si sus arcos tienen un *peso* asociado (tiempo, costo, distancia)
- El peso de un camino en un grafo etiq. es la suma de los pesos de los arcos que conforman el camino.

TDA Grafo/Digrafo

- Se podría definir un TDA Grafo, y un TDA Digrafo.
- Operaciones generadoras:
 - crear el grafo/digrafo,
 - insertar un vértice,
 - insertar un arco,
 - eliminar un arco
- Operaciones observadoras:
 - verificar la existencia de un arco,
 - buscar los vértices adyacentes,
 - recorridos del grafo,
 - buscar caminos, . . .

Grafo/digrafo: *Recorridos*

- Dado un grafo/digrafo $G = (V, E)$ y un vértice v visitar todos los vértices en G que son alcanzables desde v



búsqueda en profundidad
búsqueda en anchura

Grafos: *Búsqueda en profundidad*

Es un proceso recursivo

- Visitar el vértice inicial v
- Seleccionar un vértice w , adyacente a v que aún no haya sido visitado
- Iniciar la búsqueda en profundidad en el vértice w
- Cuando en la búsqueda el vértice visitado u , no tiene más adyacentes sin visitar, volver atrás en el recorrido hasta un vértice visitado que tenga al menos un vértice k adyacente sin visitar, y comenzar la búsqueda en profundidad desde k

Grafos: *Búsqueda en profundidad*

- El proceso termina cuando no quedan vértices sin visitar que sean alcanzables desde alguno de los visitados
- Se necesita una estructura auxiliar para mantener la información de los vértices ya 'visitados'
- orden de ejecución $\mathcal{O}(V+E)$?

Grafos: *Búsqueda en profundidad*

```

for cada vértice  $u \in V[G]$ 
  marca[u]  $\leftarrow$  no-visitado
for cada vértice  $u \in V[G]$ 
  if no-visitado(u)
    busq_prof (u)

```

```

marca[u] visitado
for cada vértice  $v \in \text{Ady}(u)$ 
  if no-visitado(v)
    busq_prof (v)

```

\leftarrow bprof(G)

\leftarrow busq_prof(u)

Grafos: *Búsqueda en anchura*

- Visitar el vértice inicial v
- Visitar todos los vértices adyacentes a v , que no hayan sido visitados (w_i)
- Visitar, para cada w_i , todos los vértices adyacentes que no hayan sido visitados aún
- . . .

Es necesario mantener una estructura auxiliar para mantener la información referente a los vértices ya visitados .

Es necesario utilizar algún tipo de estructura auxiliar para no perder la información de los últimos vértices tratados para poder continuar.

Grafos: *Recorridos-Consideraciones*

- Si en el grafo G hay vértices que no son alcanzables desde v , todo el subgrafo que nace en v queda sin recorrer



- el proceso de recorrido debe ejecutarse hasta que no queden vértices en el grafo sin visitar



- se utiliza la misma estructura que mantiene la información de los visitados.

Grafos: *Representación*

- Matriz de Adyacencia / Lista de Adyacencia
- La **matriz de adyacencia** de un grafo G con m vértices es una matriz de $m \times m$ de valores booleanos ó 0's y 1's, definida de la siguiente forma :

$$a_{ij} \begin{cases} \text{true / 1} & \text{si existe un arco del vértice } i \text{ al vértice } j \text{ (} j \text{ es adyacente a } i \text{)} \\ \text{false / 0} & \text{en caso contrario} \end{cases}$$

- **Matriz de adyacencia etiquetada:** cada a_{ij} es el peso del arco (i, j) , si no existe tal arco contiene un valor ∞ o nulo (de acuerdo al tipo).

Grafos: *Representación*

- La **lista de adyacencia** de un grafo G es una lista de vértices, en la que para cada vértice se mantiene una lista de los vértices adyacentes al vértice en estudio.

- **Registros de la lista de vértices:**

vértice	sig-ver.	ver-ady.	(*)
---------	----------	----------	-----

(*) información adicional del vértice

- **Registros de la lista de vértices adyacentes:**

ver-destino	sig-ady.	(*)
-------------	----------	-----

(*) información adicional del arco

Grafos: *Lista de adyacencia: Variantes*

- 1 los vértices se pueden mantener en un arreglo en el que cada elemento tiene una lista de vértices adyacentes
- 2 en la lista de vértices adyacentes se puede mantener una referencia al vértice en la lista de vértices, en lugar de mantener el vértice en si mismo.

Multidigrafos

- entre el vértice u y el vértice v puede haber más de un arco
- puede ser etiquetado
- entre dos vértices, arcos distintos tienen etiquetas distintas
- puede implementarse con
 - matriz de adyacencia (lista de etiquetas ó cantidad de arcos)
 - lista de adyacencia (lista de etiquetas, ó se repite el vértice destino con cada etiqueta)

Grafos: *Matriz de Caminos*

- Grafo G no etiquetado
- Matriz de adyacencia M_G
- M_G^2 matriz de caminos de longitud 2
- ...
- M_G^n matriz de caminos de longitud n

$m_n(i,j)$ = nro de caminos de longitud n que conectan v_i con v_j

- $B_n = M_G + M_G^2 + \dots + M_G^n$
(matriz de caminos de longitud $\leq n$)

1er. cuatrimestre 2002

EDyA

49

Grafos: *Redes de actividades:*

- Digrafo acíclico
- Los vértices representan tareas o actividades
- Los arcos representan relaciones de precedencia
- El arco $\langle i, j \rangle$ indica que la actividad i debe completarse antes de comenzar la actividad j (es un prerequisite)
- Algunas actividades son independientes de otras

1er. cuatrimestre 2002

EDyA

50

Grafos: *Ordenación Topológica*

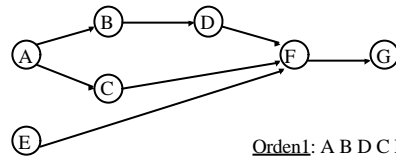
- Es un orden lineal de una red de actividades, tal que dados dos vértices i, j , si i precede a j en la red, también lo precede en el orden lineal.
- Algoritmo
O (V+E)
– aplicar un recorrido en profundidad para cada vértice
– al finalizar con cada vértice guardarlo en una estructura apropiada

1er. cuatrimestre 2002

EDyA

51

Grafos: *Red de Actividades y Ordenación Topológica*



Orden1: A B D C E F G

Orden2: A B C D E F G

Orden3: A E B D C F G

...

1er. cuatrimestre 2002

EDyA

52

Arbol Binario

Un árbol binario es un conjunto finito de nodos que tienen las siguientes características:

- está vacío ó contiene un nodo especial que provee una entrada a la estructura

- los restantes nodos están divididos en 2 subconjuntos que son en sí mismos árboles binarios:

subárbol izquierdo
subárbol derecho

1er. cuatrimestre 2002

EDyA

53

Arbol Binario: *Características*

- cada nodo del árbol contiene un elemento
- cada nodo tiene un grado (número de subárboles asociados)
- cada nodo con grado 0 \rightarrow hoja o terminal
- cada nodo que no es hoja ni raíz \rightarrow interior
- sea N un nodo de un árbol T, con un sucesor izquierdo S1 y un sucesor derecho S2, entonces:
 - N es el padre de S1 y S2
 - S1 es el hijo izquierdo de N
 - S2 es el hijo derecho de N

1er. cuatrimestre 2002

EDyA

54

Arbol Binario: Características

- cada nodo de un árbol T tiene un único padre, su predecesor
- un camino ó paso del nodo N al nodo L es el conjunto de nodos por los que hay que pasar para llegar de N a L
- un nodo L es descendiente de un nodo N si existe un camino desde N hasta L
- si el nodo L es descendiente del nodo N entonces N es antecesor de L
- cada nodo tiene 0, 1 ó 2 hijos

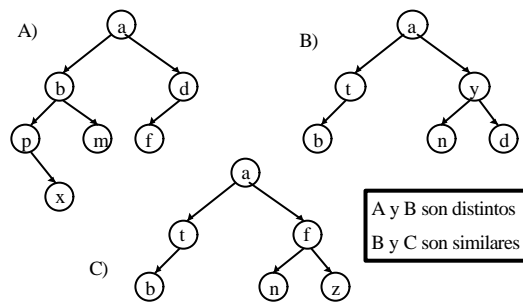
Arbol Binario: Características

- cada nodo del árbol tiene asignado un número de nivel. La raíz está en nivel 0 y cualquier otro nodo está en un nivel mayor en una unidad que el nivel de su padre
- la longitud de un camino es el número de nodos por los que pasa menos 1
- la altura del árbol es la longitud del camino más largo de la raíz a una hoja
- la altura de un nodo es la longitud del camino más largo desde el nodo a una hoja
- la profundidad de un nodo es la longitud del camino que va desde la raíz del árbol hasta el nodo

Arbol Binario: Definiciones

- **Arboles Binarios Distintos:**
cuando tienen estructuras diferentes
- **Arboles Binarios Similares:**
cuando sus estructuras son idénticas, pero la información contenido en sus nodos difiere entre sí
- **Arboles Binarios Equivalentes:**
cuando son similares y además coinciden en la información contenida en sus nodos

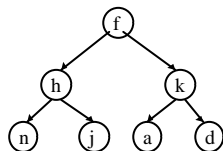
Arbol Binario: Ejemplos



Arbol Binario: Definiciones

- **Arbol Binario Completo:**
un árbol en el que todos los nodos, excepto los del último nivel (las hojas) tienen 2 hijos.

Número de nodos
(arb. bin. comp.) =
 $2^{h+1} - 1$
(h: altura del árbol)



Arbol Binario: Recorridos

Recorrer significa visitar los nodos del árbol en forma sistemática, de forma que, todos los nodos del mismo sean visitados una sola vez

- Recorrido en preorden
- Recorrido en Inorden
- Recorrido en PosOrden
- Recorrido por Niveles

Arbol Binario: *Recorridos*

- Recorrido en Preorden
 - visitar la raíz
 - recorrer el subárbol izquierdo en preorden
 - recorrer el subárbol derecho en preorden
- Recorrido en Inorden
 - recorrer el subárbol izquierdo en inorden
 - visitar la raíz
 - recorrer el subárbol derecho en inorden
- Recorrido en Posorden ???
- Recorrido por Niveles ???

1er. cuatrimestre 2002

EDyA

61

Arbol Binario: *Operaciones*

- Determinar si un árbol está vacío
- Determinar si un ítem está presente en el árbol
- Vaciar un árbol
- Determinar el nivel de un ítem en el árbol
- Determinar el padre de un ítem en el árbol
- Determinar los hijos izq. y der. de un ítem en el árbol
- Recorridos (preorden, inorden, posorden, por niveles)
- Crear un árbol binario dados dos árboles binarios A1 y A2 que serán sus hijos, y un ítem que será el ítem correspondiente a la raíz del nuevo árbol.
- ...

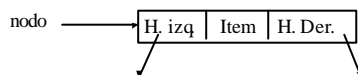
1er. cuatrimestre 2002

EDyA

62

Arbol Binario: *Implementaciones*

- En forma estática (utilizando arreglos)
en cada subíndice del arreglo se indica el índice del nodo padre.
- En forma dinámica (utilizando punteros)

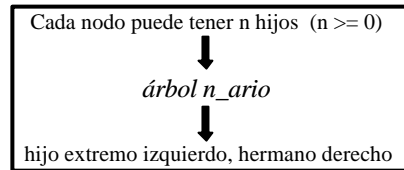


1er. cuatrimestre 2002

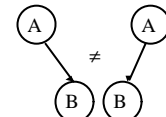
EDyA

63

Arbol genérico: *Características*



En un árbol binario, siempre se distingue entre hijo izquierdo e hijo derecho



1er. cuatrimestre 2002

EDyA

64

Arbol genérico

- Operaciones equivalentes a las de árbol binario adaptadas a la nueva situación.
- Recorridos equivalentes
- Recorrido en Preorden
 - visitar la raíz
 - recorrer el subárbol izquierdo en preorden
 - por cada hermano derecho del HEI
 - recorrer el subárbol en preorden

1er. cuatrimestre 2002

EDyA

65

Arbol Genérico: *Implementación*

- Arreglos
 - nodos del árbol numerados de 1 a n
 - si T es un árbol n-ario mantenemos sus nodos en un arreglo lineal A en el cual:

$$\begin{cases} A[i] = 0 & \text{si es el nodo raíz} \\ A[i] = j & \text{si } j \text{ es el nodo padre de } i \end{cases}$$
 - debe mantenerse un arreglo paralelo con los elementos en sí mismos o decidirse por trabajar con un arreglo de registros.

1er. cuatrimestre 2002

EDyA

66

Arbol Genérico: *Implementación*

- Lista de hijos
 - se forma para cada nodo una lista de sus hijos
 - se mantiene un arreglo de listas, indexado según el número de los nodos del árbol
 - los elementos de cada lista son los números de los nodos
 - se debe mantener en el arreglo además los elementos que están guardados en el árbol (también podrían mantenerse en un arreglo paralelo)

1er. cuatrimestre 2002

EDyA

67

Arbol genérico: *lista de hijos*

Type	
indice= 1..maxnodos;	nodos = array [indice]
nodo= record	of nodo;
rotulo:TipoElem;	
hijos: Lista	ArbolG= record
end;	arbol : nodos;
	raiz : indice
	end
Lista tiene elementos de tipo indice	

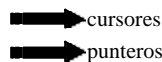
1er. cuatrimestre 2002

EDyA

68

Arbol Genérico: *Implementación*

- HEI - HD (hijo extremo izquierdo - hermano derecho)
 - se puede trabajar con:



En cualquier caso cada nodo consiste de: el elemento, un cursor/puntero a su hijo más izquierdo, y un cursor/puntero a su hermano derecho. El único nodo que siempre tendrá el hermano derecho con valor nulo es la raíz

1er. cuatrimestre 2002

EDyA

69

Arbol genérico: *HEI-HD*

- Type
 - enlace = ^nodo;
 - nodo = record
 - rotulo: TipoElem;
 - heizq,
 - herder : enlace
 - end;
 - ArbolG : enlace;

1er. cuatrimestre 2002

EDyA

70

Arbol Genérico: *Posorden*

```

Procedure posorden (A: arbol);
  Procedure posordenaux (n: nodo);
    var h: nodo;
    begin
      h := hijoIzquierdo(n);
      while noNulo(h) do
        begin
          posordenaux (h);
          h := hermanoDerecho (h)
        end;
      visitar (n)
    end; {de posordenaux}
  begin {de posorden}
    if noNulo(A)
      then posordenaux( raiz(A))
    end;
  end;
  
```

1er. cuatrimestre 2002

EDyA

71

Bosque Abarcador de un digrafo

- Conjunto de árboles generados a partir del recorrido de un digrafo
- Bosque abarcador en profundidad
- Bosque abarcador en anchura
- Cada árbol se forma con vértices del digrafo y aquellos arcos que en el recorrido llevan a vértices sin visitar.

1er. cuatrimestre 2002

EDyA

72

Bosque abarcador en *profundidad*


- Distintos tipos de arcos
 - ☞ *arcos de árbol*: son los que forman el árbol
 - ☞ *arcos de avance*: van de un vértice v a un vértice w que es un descendiente propio de v en el árbol abarcador
 - ☞ *arcos de retroceso*: van de un vértice v a un vértice w que es un antecesor de v en el árbol abarcador
 - ☞ *arcos cruzados*: van de un vértice v a otro vértice w que no es ancestro ni descendiente

1er. cuatrimestre 2002

EDyA

73

Bosque abarcador en *profundidad*

- Se trabaja sobre el algoritmo de búsqueda en profundidad
- Se agrega un arreglo *nrobp* para guardar el orden en profundidad en que fueron visitados los vértices  numeración en profundidad

w es un descendiente de v
 $nrobp[v] \leq nrobp[w] \leq nrobp[v] + nroDesc.$

1er. cuatrimestre 2002

EDyA

74

Bosque abarcador en *profundidad*

- Se utiliza el arreglo para identificar los distintos tipos de arcos y armar el árbol abarcador
- *arcos de avance*
 - de baja numeración a alta numeración
- *arcos de retroceso*
 - de alta numeración a baja numeración
- *arcos cruzados*
 - de alta numeración a baja numeración

1er. cuatrimestre 2002

EDyA

75

Bosque abarcador en un grafo

- Cada árbol del bosque es un componente conexo del grafo.
- Bosque abarcador en profundidad y en anchura
- No hay aristas cruzadas
- Las aristas de retroceso y avance son las mismas
 - aristas de árbol
 - aristas de retroceso

1er. cuatrimestre 2002

EDyA

76

Puntos de articulación de un grafo

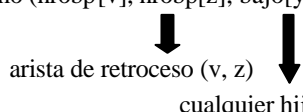
- *Punto de articulación*: un vértice que al ser removido junto con sus arcos incidentes divide al grafo en 2 o mas componentes.
- Grafo biconexo: grafo sin puntos de articulación
- Para encontrar los puntos de articulación se trabaja con el algoritmo de búsqueda en profundidad y el árbol abarcador en profundidad

1er. cuatrimestre 2002

EDyA

77

Puntos de articulación de un grafo

- calcular el $nrobp[v]$ para todo vértice v , utilizando el recorrido en profundidad.
- para cada vértice v , obtener $bajo[v]$
 $\text{mínimo}(nrobp[v], nrobp[z], bajo[y])$

se calcula en postorden

1er. cuatrimestre 2002

EDyA

78

Puntos de articulación de un grafo

- Se encuentran los puntos de articulación
- la raíz es punto de articulación sssi tiene 2 o más hijos.
- un vértice v que no es raíz es punto de articulación sssi hay un hijo w de v tal que $\text{bajo}[w] \geq \text{nrobp}[v]$
(v desconecta w y sus descendientes del resto del grafo)

1er. cuatrimestre 2002

EDyA

79

Arbol Binario: Arbol de Expresión

Un árbol binario puede utilizarse para representar expresiones algebraicas

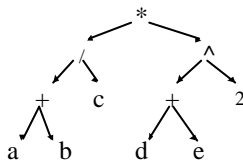
- cada nodo interior del árbol representa un operador
- cada hoja contiene un operando
- los distintos recorridos del árbol (preorden, inorden, posorden) permiten obtener las notaciones prefija, infija y posfija de la expresión respectivamente

1er. cuatrimestre 2002

EDyA

80

Arbol Binario: Arbol de Expresión



- Expresión (infija) $(a + b) / c * (d + e) ^ 2$
- prefija (preorden): $* / + a b c ^ + d e 2$
- posfija (posorden): $a b + c / d e + 2 ^ *$

1er. cuatrimestre 2002

EDyA

81

Arbol Binario: Arbol de expresión

- Type
 - enlace = ^nodo;
 - nodo = record
 - case rotulo:(hoja, interior) of
 - hoja: (operando: real);
 - interior: (operador: char;
 - hijoizq, hjoder: enlace)
 - end

end

1er. cuatrimestre 2002

EDyA

82

GDA: expresión aritmética

- un grafo dirigido acíclico (GDA) es útil para representar expresiones aritméticas con subexpresiones comunes.
- ejemplo:
 $((a + b) * c + ((a + b) + e) * (e + f)) * ((a + b) * c)$

Ejercicio: escriba un subprograma para convertir un árbol de expresión con operadores + y * en un GDA.

Ejercicio: escriba un subprograma para construir un GDA a partir de una expresión.

1er. cuatrimestre 2002

EDyA

83

Búsqueda por clave

- método de dirección (1) / método de árbol (2)
 - calcula la dirección de un elemento en base a su clave
(hashing)
 - serie jerarquica de comparaciones
(árbol binario de búsqueda, árbol B, etc.)
- método interno (1) / método externo (2)
 - mantiene los elementos en la memoria RAM
 - utiliza memoria secundaria

1er. cuatrimestre 2002

EDyA

84

Búsqueda por clave

- método digital (1) / método no-digital (2)
 - (1) claves como una secuencia de dígitos ó caracteres alfabéticos. (Trie)
 - (2) comparan claves completas
- método unidimens.(1) / método multidimens. (2)
 - (1) utiliza claves de un atributo
 - (2) utiliza claves de múltiples atributos



TDA *Tabla de Búsqueda/Diccionario* (inserción, eliminación, es miembro, mínimo, búsqueda)

1er. cuatrimestre 2002

EDyA

85

Método por *transformación de claves*

- aumenta la velocidad de búsqueda
- el tiempo de búsqueda es independiente del número de elementos
- se trabaja sobre la base de una función de transformación o función hash (h)
- se asume que los elementos están almacenados en un arreglo \rightarrow hash cerrado/interno

Función hash: función que aplicada a la clave del elemento da como resultado un índice

1er. cuatrimestre 2002

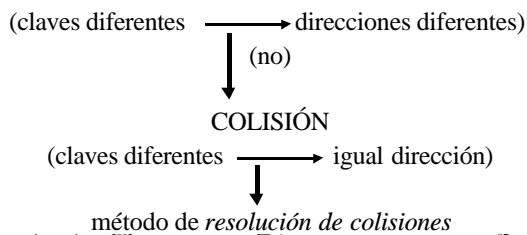
EDyA

86

Transformación de claves - *hashing*

La función hash debe ser:

- simple de calcular
- uniforme



1er. cuatrimestre 2002

EDyA

87

Funciones hash más utilizadas

- Función módulo
- Función cuadrado: eleva al cuadrado la clave y toma los dígitos centrales
- Función plegamiento: divide la clave en partes de igual número de dígitos y opera con ella
- Función truncamiento: toma algunos dígitos de la clave y forma con ellos una dirección
- Funciones aplicadas a claves de tipo cadena

1er. cuatrimestre 2002

EDyA

88

Hash - *Resolución de colisiones*

- reasignación (rehash)
función de rehash (rh) acepta un índice y genera un nuevo índice.
 - si la posición $h(\text{clavex})$ ya está ocupada se calcula $rh(h(\text{clavex}))$.
 - si también está ocupada se calcula $rh(rh(h(\text{clavex})))$
 - el proceso continua hasta que se encuentra una posición vacía ó el elemento buscado

1er. cuatrimestre 2002

EDyA

89

Hash: *Colisiones - Reasignación*

- resolución lineal de colisiones:
 rh da el índice siguiente al indicado
fuerte agrupamiento posible alrededor de ciertas claves.
- resolución de colisiones con doble dirección hash
 - h_1 : función hash primaria
 - h_2 : función hash utilizada por rh para el rehash
 - $rh(i) = (i + h_2(\text{clavex})) \bmod \text{máximo}$,
 - $i_0 = h_1(\text{clavex})$

1er. cuatrimestre 2002

EDyA

90

Hash: Reasignación - problemas

- asume tabla de tamaño fijo m
- insertar el elemento $m+1$

utilizar una tabla más grande

nueva función hash

recalcular todos los valores de la tabla original

1er. cuatrimestre 2002

EDyA

91

Hash: Reasignación - problemas

- eliminar un registro r_1 con valor hash h_1

la posición queda vacía

- buscar el registro r_2 con valor hash h_1

debe estar en otra posición h_2

posición h_2 vacía $\rightarrow r_2$ no está en la tabla

ERROR

puede estar más adelante

marcar al registro como *eliminado*

1er. cuatrimestre 2002

EDyA

92

Hash: Encadenamiento (Hash abierto)

- un arreglo de buckets
- cada subíndice es un posible valor hash
- cada bucket es una lista ordenada o no
- los elementos de igual valor hash h_j están en la lista del bucket h_j correspondiente
- si busco r_j y $h(r_j) = k$
 - acceder a $B[k]$
 - recorrer la lista

1er. cuatrimestre 2002

EDyA

93

Arbol Binario de Búsqueda

- buena implementación para tabla de búsqueda.
- operación de búsqueda de $O(\log n)$
- árbol binario
- cada nodo contiene información estructurada con un campo clave
- existe una relación de orden asociada al campo clave

base para el ordenamiento de la información

1er. cuatrimestre 2002

EDyA

94

Arbol Binario de Búsqueda

Definición:

cada nodo N del árbol tiene la siguiente propiedad:

- *el valor clave de N es mayor que cualquier valor clave de su subárbol izquierdo
- *el valor clave de N es menor que cualquier valor clave de su subárbol derecho

el recorrido en inorden del árbol produce un listado

1er. cuatrimestre 2002

EDyA

95

ABB: búsqueda

- comparar la clave buscada con la raíz del árbol
- si coinciden la búsqueda culmina con éxito
- si es menor la búsqueda continua por el subárbol izquierdo
- si es mayor la búsqueda continua por el subárbol derecho
- si se llega a un subárbol nulo la búsqueda culmina sin éxito

1er. cuatrimestre 2002

EDyA

96

ABB: inserción

- comparar la clave a insertar con la raíz del árbol
 - si es mayor \longrightarrow avanzar hacia el subárbol der.
 - si es menor \longrightarrow avanzar hacia el subárbol izq.
- repetir hasta que se cumpla:
 - el subárbol es vacío \longrightarrow agregar el elemento
 - la clave coincide con la raíz del subárbol \longrightarrow el elemento ya está presente en el árbol, no se efectúa la operación

1er. cuatrimestre 2002

EDyA

97

ABB: eliminación

- *Objetivo*: eliminar un nodo del árbol sin violar los principios de ABB.
- *Casos*:
 - el elemento a eliminar es terminal (hoja)
 - se suprime el nodo
 - el elemento a eliminar tiene un solo descendiente
 - se sustituye el nodo por su descendiente
 - el elemento a eliminar tiene los 2 descendientes
 - se sustituye el nodo por:
 - el más izquierdo de sus descendientes derechos
 - el más derecho de sus descendientes izquierdos

1er. cuatrimestre 2002

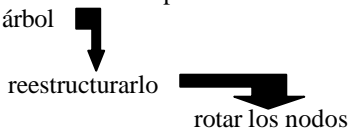
EDyA

98

Arboles Balanceados por altura: avl

- es un árbol binario de búsqueda
- cada nodo tiene un balance: ASI - ASD
- el balance de cada nodo $\in [-1, 0, 1]$

Al insertar/eliminar un nodo se puede desbalancear el árbol



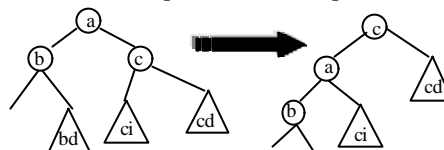
1er. cuatrimestre 2002

EDyA

99

Avl: rotaciones

- Rotación a izquierda (nodo pivote a)



q \longleftarrow hijo_der(a); temp \longleftarrow hijo_izq(q);
hijo_izq(q) \longleftarrow a; hijo_der(a) \longleftarrow temp

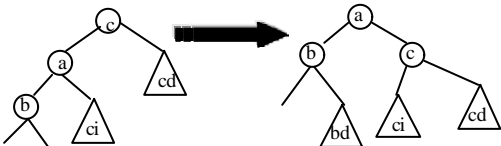
1er. cuatrimestre 2002

EDyA

100

Avl: rotaciones

- Rotación a derecha (nodo pivote c)



q \longleftarrow hijo_izq(c); temp \longleftarrow hijo_der(q)
hijo_der(q) \longleftarrow c; hijo_izq(c) \longleftarrow temp

1er. cuatrimestre 2002

EDyA

101

avl: rotaciones

rotación simple

- nodo con balance 2 ó -2
- un hijo con balance 1 ó -1



rotación con pivote en el nodo con balance 2 ó -2

rotación doble

- nodo con balance 2 ó -2
- un hijo con balance -1 ó 1



– rotación con pivote en el nodo hijo
– rotación con pivote en el nodo padre

1er. cuatrimestre 2002

EDyA

102

avl: *inserción y eliminación*

- un nodo insertado/eliminado puede
 - causar que el balance de un subárbol pase de -1 a 0 ó de 1 a 0
No rebalancear
 - causar que el balance de un subárbol pase de 0 a 1 ó de 0 a -1
No rebalancear
 - causar que el balance de un subárbol pase de -1 a -2 ó de 1 a 2
Rebalancear

1er. cuatrimestre 2002

EDyA

103

Arbol de Recuperación: *TRIE* (Búsqueda Digital)

- Se utiliza para representar conjuntos de cadenas de caracteres
- ¿Es una estructura apropiada para implementar el TDA Tabla de Búsqueda ?
- Es apropiada cuando muchas palabras comparten un prefijo
- Cada camino de la raíz a una hoja corresponde a una palabra

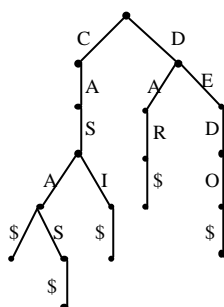
1er. cuatrimestre 2002

EDyA

104

Arbol TRIE

- Los nodos del trie corresponden a los prefijos de las palabras del conjunto que se representa
- Se considera un símbolo especial \$ para marcar el final de las palabras.



1er. cuatrimestre 2002

EDyA

105

Arbol TRIE: *Implementación*

- Un nodo de un trie puede considerarse como un mapeo con dominio en {A, B, ...Z, \$}. El conjunto de valores del rango son apuntadores a nodos del TRIE
- La profundidad del árbol depende de la longitud de las palabras almacenadas en él.
- Si lo utilizamos para implementar Tabla de Búsqueda, cada palabra representa una clave.

1er. cuatrimestre 2002

EDyA

106

Arbol TRIE: *Implementación*

- Nodo del TRIE implementado con arreglos (arreglo de punteros a nodos del TRIE)

```
type
    nodo_ptro = ^nodo_trie;
    letras = 'A'..'Z';
    nodo_trie = array [letras] of nodo_ptro;
```

1er. cuatrimestre 2002

EDyA

107

Arbol TRIE: *Implementación*

- Nodo del TRIE implementado con Listas

```
type
    letras = 'A'..'Z';
    nodo_ptro = ^nodo_trie;
    nodo_trie = record
        letra : letras;
        nodo_hijo,
        nodo_sgte : nodo_ptro
    end;
```

1er. cuatrimestre 2002

EDyA

108

Arbol 2-3: Características

- Cada **nodo interior** tiene 2 ó 3 hijos
- Todos los caminos que van de la raíz a una hoja tienen idéntica longitud
- Un árbol con 1 ó 0 nodos es un caso especial
- Los elementos están colocados en las hojas respetando un orden lineal.

1er. cuatrimestre 2002

EDyA

109

Arbol 2-3: Características

- Se supone que el ordenamiento de los elementos está basado en un **campo clave**
- En cada nodo **interior** se coloca:
 - la **clave** del elemento más pequeño que sea descendiente del 2do. hijo
 - en caso de existir 3er. hijo, se coloca la **clave** más pequeña que sea descendiente del 3er. hijo
- Un árbol 2-3 de k niveles tiene entre 2_{k-1} y 3_{k-1} hojas.

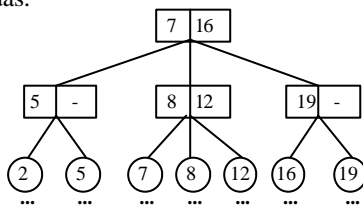
1er. cuatrimestre 2002

EDyA

110

Arbol 2-3: Características

- Para realizar una búsqueda de una clave ó elemento en un árbol 2-3, de manera eficiente, deben tenerse en cuenta las características dadas.



1er. cuatrimestre 2002

EDyA

111

Arbol 2-3: Inserción

- Se recorre el árbol hasta el nivel inferior de nodos internos, buscando la posición del nuevo elemento X con clave k_x
- Si el nodo tiene sólo 2 hijos, se hace que X sea el 3er. hijo, colocándolo en el orden adecuado, ajustando lo que sea necesario.

1er. cuatrimestre 2002

EDyA

112

Arbol 2-3: Inserción

- Si el nodo tiene ya 3 hijos, (es decir X será el cuarto), partir el nodo en 2 nodo y nodo'. Insertar a nodo' como hijo de p (padre de nodo)
 - si p tiene 2 hijos, nodo' será el 3er. hijo en el orden adecuado,
 - si p tiene 3 hijos, p se divide en p y p' y se repite el proceso recursivamente.
- si se llega al caso de tener que dividir la raíz, se crea una nueva raíz y se incrementa en 1 el número de niveles del árbol

1er. cuatrimestre 2002

EDyA

113

Arbol 2-3: Eliminación

- Siempre se elimina una hoja.
- Al eliminar puede suceder que disminuya en 1 el número de niveles del árbol.
- Se recorre hasta las hojas buscando el elemento X con clave k_x
- Si el nodo padre de X, p tiene 3 hijos al suprimir X el árbol queda en condiciones correctas

1er. cuatrimestre 2002

EDyA

114

Arbol 2-3: *Eliminación*

- Si el nodo padre p tiene 2 hijos, al suprimir X quedará con 1 hijo
 - si p es la raíz del árbol, se suprime p y el nodo hijo queda como raíz
 - si p no es la raíz
 - si padre(p) tiene otro hijo, adyacente a p (por la derecha ó la izquierda), que tiene 3 hijos, se transfiere un hijo a p y el árbol queda en condiciones (reacomodando las claves)
 - si los hermanos de p tienen sólo 2 hijos se transfiere el único hijo de p a un hermano adyacente y se elimina p
 - se repite el proceso recursivamente

1er. cuatrimestre 2002

EDyA

115

Arbol 2-3: *Implementación*

```

type
  tipo_clave = ...; enlace_h = ^nodo2_3;
  tipo_elemento = record
    clave : tipo_clave;
    ...
  end;
  tipo_nodo = (hoja, interior);
  nodo2_3 = record
    case clase: tipo_nodo of
      hoja: (elemento: tipo_elemento);
      interior: (primer_h, segundo_h, tercer_h: enlace_h;
                menor_seg, menor_ter: tipo_clave)
    end;
end;

```

1er. cuatrimestre 2002

EDyA

116

```

Procedure INSERTAR (x: tipo_elemento; var A: arbol2_3)
Var auxiliar, temp: enlace_h; menor: tipo_clave;
begin
  {si está vacío . . .};
  insertInt (A, x, temp, menor);
  if temp <> nil
  then begin
    {crea la nueva raíz a partir de auxiliar, sus hijos están
    apuntados por A y temp}
    nuevonodo (auxiliar, interior);
    auxiliar^.primer_h := A;
    auxiliar^.segundo_h := temp;
    auxiliar^.tercer_h := nil;
    auxiliar^.menor_seg := menor
    A:= auxiliar
  end
end;

```

1er. cuatrimestre 2002

EDyA

117

```

Procedure InsertInt ( ptro2_3: enlace_h; x: tipo_elemento;
  var nuevo: enlace_h; {puntero al nodo creado}
  var valormenor: tipo_clave {menor del subárbol 'nuevo'})
Var temp: enlace_h; menor: tipo_clave;
begin
  nuevo:= nil;
  if ptro2_3^.clase = hoja
  then if ptro2_3^.elemento <> clave(x)
  then begin
    nuevonodo(nuevo, hoja);
    if ptro2_3^.elemento < clave(x)
    then begin
      asignar (nuevo^.elemento, x); asignar(valormenor, clave(x))
    end
  else begin
    asignar(nuevo^.elemento, ptro2_3^.elemento);
    asignar(ptro2_3^.elemento, x);
    asignar(menorvalor, clave(ptro2_3^.elemento));
  end
end . . .

```

1er. cuatrimestre 2002

EDyA

118

Arbol B: *Características*

- Se construye hacia arriba, a partir de la base
- Llamaremos **página** a cada nodo de un árbol B
 - secuencia ordenada de k claves
 - secuencia de k+1 descendientes
 - por cada clave se mantiene una asociación a la información adicional
- Llamaremos **orden** de un árbol B al número máximo de descendientes que puede tener una página

1er. cuatrimestre 2002

EDyA

119

Arbol B: *Características*

- Un árbol B de orden m:
 - cada página tiene un máximo de m descendientes
 - cada página, excepto la raíz y las hojas tiene por lo menos $\lceil m/2 \rceil$ descendientes
 - la raíz tiene por lo menos 2 descendientes (a menos que sea una hoja)
 - todas las hojas aparecen en el mismo nivel
 - cada página que no es hoja con k descendientes contiene k-1 claves
 - una página hoja contiene $\lceil m/2 \rceil - 1$ claves como mínimo y m-1 claves como máximo

1er. cuatrimestre 2002

EDyA

120

Arbol B: Definición de tipos

- Type
 - paginaB = ^pagina
 - pagina = record
 - cant_claves : integer;
 - claves: array[1..maxclaves] of tipoclave;
 - hijos: array[1..maxclaves+1] of paginaB;
 - datos: array[1..maxclaves] of . . .
 - end;
 - ArbolB = paginaB

1er. cuatrimestre 2002

EDyA

121

Arbol B: Búsqueda

- Se comienza por la raíz
- Se ubica la clave en la página
 - ¿está? → termina el proceso
 - ¿no está? → se decide por que hijo bajar
(el arreglo de claves está ordenado)
- Se repite el proceso con la página hijo elegida
- El proceso termina cuando se encuentra el elemento ó se llega a un descendiente nulo

1er. cuatrimestre 2002

EDyA

122

Arbol B: Inserción

- Siempre se inserta una hoja
- Se busca recursivamente hasta el nivel de las hojas, dónde se debe insertar
- se inserta:
 - en una página en la que hay lugar (reacomodar claves)
 - en una página en la que no hay lugar **división y promoción** (a la vuelta de la recursión)

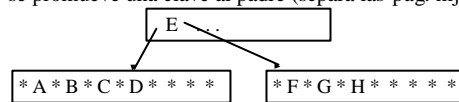
1er. cuatrimestre 2002

EDyA

123

Arbol B: Inserción

- División y promoción
 - *A*B*C*D*E*F*H* (* indica descendiente nulo)
- insertar G: la página está llena (orden 8) se divide en 2 páginas distribuyendo las claves se promueve una clave al padre (separa las pag. hijas)




1er. cuatrimestre 2002

EDyA

124

Arbol B: Eliminación


- Siempre se elimina de una hoja
 - caso 1: eliminar una clave de una página hoja que no provoca deficiencia de claves. Se reacomodan las claves dentro de la página
 - caso 2: eliminar una clave de una página hoja J que provoca deficiencia de claves y tiene una página vecina I que supera el mínimo de claves  **redistribuir**
- intervienen la página J, la página vecina I y la página padre P

1er. cuatrimestre 2002

EDyA

125

Arbol B: Eliminación

- caso 3: eliminar una clave de una página hoja J que provoca deficiencia de claves y no tiene una página vecina que supere el mínimo de claves.
 - No se puede redistribuir  **concatenar**
 - En una única página las claves de la página J, con las claves de la página vecina I, y su clave padre (que las separaba)
 - Cuando la insuficiencia se repite en los niveles superiores, se repite el proceso de concatenación

1er. cuatrimestre 2002

EDyA

126

Arbol B: Eliminación

- Cuando en el proceso de concatenación se llega al nivel de la raíz y la raíz es absorbida en la concatenación decreta la cantidad de niveles del árbol.
- caso 4: eliminar una clave que no está en una página hoja. Se intercambia con su sucesor inmediato que este en una hoja y luego se elimina de la hoja siguiendo alguno de los casos anteriores.

1er. cuatrimestre 2002

EDyA

127

Arbol k-d: Características

- Registros de n campos
- k campos forman la clave ($1 < k \leq n$)
- una consulta especifica condiciones sobre algunos o todos los campos que forman la clave
- arbol binario de búsqueda: en cada nivel del árbol la distribución de los elementos se realiza según un campo clave

1er. cuatrimestre 2002

EDyA

128

Arbol k-d: características

- cada nodo P tiene asociado un discriminante
 - número de campo clave con respecto al que se analiza para distribuir, $0 \leq \text{disc} \leq k-1$
 - disc se almacena en cada nodo o se calcula
 - $\text{disc}(P)$: discriminante del nodo P
- todos los nodos en el mismo nivel tienen igual valor de discriminante
- k campos clave para el nodo P: $k_0(P), k_1(P), \dots, k_{k-1}(P)$

1er. cuatrimestre 2002

EDyA

129

Arbol k-d: características

- Sean $HI(P)$ el subárbol izquierdo y $HD(P)$ el subárbol derecho del nodo P
- Para cualquier nodo P en un árbol k-d

$$\text{sea } j = \text{disc}(P) \begin{cases} \text{para cualquier nodo } Q \in HI(P) \\ k_j(Q) < k_j(P) \\ \text{para cualquier nodo } R \in HD(P) \\ k_j(R) > k_j(P) \end{cases}$$

1er. cuatrimestre 2002

EDyA

130

Arbol k-d: características

- $\text{Disc}(\text{raiz}) = 0 \rightarrow k_0(P)$
- $\text{disc}(HI(\text{raiz})) = \text{disc}(HD(\text{raiz})) = 1$
- nivel k-1: $\text{disc}(P) = k-1 \quad \forall P \in \text{al nivel}$
- nivel k: $\text{disc}(P) = 0 \quad \forall P \in \text{al nivel}$
- el ciclo se repite

1er. cuatrimestre 2002

EDyA

131

Arbol k-d: características

- ¿claves iguales? \rightarrow ¿qué hijo visitar?
(P nodo del árbol, Q buscado)
 \downarrow
Sucesor (P, Q)
- sea $j = \text{disc}(P)$
 - si $k_j(P) < k_j(Q)$, Sucesor es $HI(P)$ ó $HD(P)$
 - si $k_j(P) = k_j(Q)$, depende de los campos restantes
 - si $k_{j+1}(P) < k_{j+1}(Q)$ $HD(P)$
 - si $k_{j+1}(P) > k_{j+1}(Q)$ $HI(P)$
 - si $k_{j+1}(P) = k_{j+1}(Q)$ sigue en $j+2$

1er. cuatrimestre 2002

EDyA

132

Arbol k-d: *características*

- consulta ó búsqueda:
 - exacta (un valor para cada campo de la clave)
 - parcial (un valor para algunos campos de la clave)
 - por rango (un subrango de valores para algunos campos de la clave)

1er. cuatrimestre 2002

EDyA

133

Arbol k-d: *consulta exacta*

- Sea Q la clave buscada
- Sea P el elemento guardado en el nodo actual del arbol k-d
- Sea $\text{disc}(P)$ el número de campo clave según el cual se analizó para distribuir al momento de la inserción:
 - comparar $k_{\text{disc}(P)}(Q), k_{\text{disc}(P)}(P)$
 - si menor \rightarrow buscar por el subárbol izquierdo
 - si mayor \rightarrow buscar por el subárbol derecho
 - si igual \rightarrow elegir por que subárbol buscar

1er. cuatrimestre 2002

EDyA

134

Arbol k-d: *consulta parcial*

- Sean $k_i(Q)$ y $k_j(Q)$ los valores de los campos de la clave Q por los que se desea buscar
- Sea P el elemento del nodo actual
- Si $\text{disc}(P) = i$ ó $\text{disc}(P) = j$
 - comparar $k_{\text{disc}(P)}(Q), k_{\text{disc}(P)}(P)$
 - si menor buscar por el subárbol izquierdo
 - si mayor buscar por el subárbol derecho
 - si igual elegir por que subárbol buscar
- Si $\text{disc}(P) \diamond i$ y $\text{disc}(P) \diamond j$
 - buscar por los 2 subárboles

1er. cuatrimestre 2002

EDyA

135

Arbol k-d: *consulta por rango*

- Sean $k_{i1}..k_{i2}, k_{j1}..k_{j2}$ rangos de valores de los campos i, j de la clave por los que se desea buscar
 - Sea P el elemento del nodo actual
 - Si $\text{disc}(P) = i$
 - verificar $k_{i1} \leq k_{\text{disc}(P)}(P) \leq k_{i2}$
 - Si $\text{disc}(P) = j$
 - verificar $k_{j1} \leq k_{\text{disc}(P)}(P) \leq k_{j2}$
 - Si $\text{disc}(P) \diamond i$ y $\text{disc}(P) \diamond j$
 - buscar por los 2 subárboles
- } Buscar según corresponda

1er. cuatrimestre 2002

EDyA

136

Arbol k-d: *inserción*

- Recibe la raíz del árbol y un elemento
- siempre se inserta una hoja
- ¿en el árbol hay un elemento con valores idénticos en todos los campos de la clave?
 - si \rightarrow retorna el nodo y no inserta
 - no \rightarrow inserta el nodo en el árbol
 - en cada nodo P calcula el SUCESOR(P, Q) para buscar el lugar en que debe insertar
 - Q es nulo \rightarrow llegó a una hoja e inserta
 - Q no es nulo \rightarrow repite el proceso con el SUCESOR(P, Q)

1er. cuatrimestre 2002

EDyA

137

Arbol k-d: *eliminación*

- Eliminar la raíz de un subárbol (nodo P)
- P es hoja :
 - se elimina quedando como subárbol vacío
- P no es hoja:
 - debe ser reemplazado por uno de sus descendientes: Q
 - Q debe mantener el orden impuesto por P
 - El elemento almacenado en Q se almacena en P
 - Se elimina el viejo Q

1er. cuatrimestre 2002

EDyA

138

Arbol k-d: *eliminación*

- Sea $\text{disc}(P) = j$
 - Q es el máximo elemento en j para el hijo izquierdo de P ó
 - q es el mínimo elemento en j para el hijo derecho de P
- La búsqueda del mínimo ó máximo valor en j es equivalente a una búsqueda parcial sobre k_j para el máximo ó mínimo valor

1er. cuatrimestre 2002

EDyA

139

Heap: *Arbol parcialmente ordenado*

- árbol binario
- todos los niveles del árbol están completos excepto el último
- el último nivel se completa de izquierda a derecha
- existe una relación de orden entre el valor de un nodo y el valor de sus hijos (menor ó mayor)

1er. cuatrimestre 2002

EDyA

140

Heap: *Características*

- Heap_Mínimo : la relación entre padre e hijos es '<'
- Heap_Máximo : la relación entre padre e hijos es '>'
- En un heap_mínimo/ heap_máximo el valor del nodo raíz es el valor mínimo/ máximo del conjunto de elementos

Estructura apropiada para implementar el tda Cola de Prioridad

1er. cuatrimestre 2002

EDyA

141

Heap: *Características*

- Las operaciones de inserción y borrado de elementos deben realizarse teniendo en cuenta que después de ellas el árbol debe mantener su condición de *heap*.
- En general interesa eliminar el nodo raíz, es decir el mínimo ó máximo
- Hay variantes de Heaps: max-min heaps, k-heaps, etc.

1er. cuatrimestre 2002

EDyA

142

Heap: *Operaciones*

Eliminar_Min (elimina la raíz del heap)

- eliminar el nodo raíz
- hacer nodo raíz a la hoja más derecha del último nivel (temporariamente)
- empujar el elemento hacia abajo, intercambiándolo con uno de sus hijos (el de menor valor)
- repetir el último paso hasta que el elemento:
 - quede en una hoja, ó
 - tenga menor valor que sus hijos

1er. cuatrimestre 2002

EDyA

143

Heap: *Operaciones*

Inserción de un elemento

- se inserta como hoja, en el último nivel y tan a la izquierda como sea posible (¿Qué pasa cuando el último nivel está completo?)
- si el nuevo elemento tiene menor valor que su padre lo hacemos subir, intercambiándolo
- repetir el paso anterior hasta que el nuevo elemento:
 - llegue a una posición en que sea mayor que su padre, ó
 - llegue a la raíz

1er. cuatrimestre 2002

EDyA

144

Heap: *Implementación*

- Se puede implementar con punteros (como un árbol binario)
- Se puede implementar con un arreglo en el que los elementos guardados en los nodos del árbol se mantienen respetando lo siguiente:
 - el nodo q está en A[i]
 - el hijo izquierdo de q está en A[2i]
 - el hijo derecho de q está en A[2i + 1]

1er. cuatrimestre 2002

EDyA

145

Heap: *Implementación*

- Mantenemos un arreglo con los elementos
- mantenemos la posición del último elemento para simplificar las operaciones
- Heap = Record
 - nodos : array[1..max] of tipoelem;
 - ultimo: 1..max
 - end;

1er. cuatrimestre 2002

EDyA

146