



Diseño y Análisis de Algoritmos

ESTRUCTURAS DE DATOS

Realizado por:

Gabriela Márquez

C.I. 15.622.883

Mayo 2005

Estructuras de Datos

- Matrices
- Pilas y Colas
- Tablas Asociativas
- Estructuras de Conjuntos Disjuntos (Partición)

Matrices:

Una matriz es una estructura de datos que consta de un número fijo de ítems del mismo tipo.

- Matrices monodimensionales → Arrays

Para declarar una matriz:

vec : matriz [1..50] de enteros

Donde:

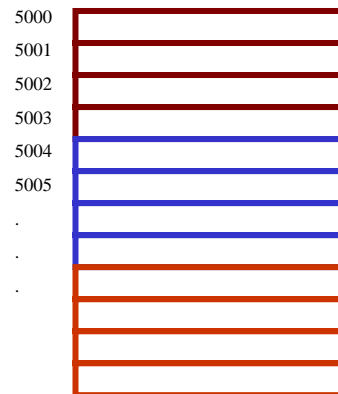
vec[1] = primer elemento

vec[50] = último elemento

Propiedad esencial de una matriz:

La propiedad esencial de una matriz es que podemos calcular la dirección de cualquier elemento dado en un tiempo constante.

Ejemplo: Si la matriz vec comienza en la dirección 5000, y sus variables enteras ocupan 4 bytes de almacenamiento cada una, entonces la dirección del elemento cuyo índice es k está dada claramente por: $4996+4k$



Acceder a un elemento:
 $4996+4k$

PILAS.

- Son implementadas eficientemente con matrices monodimensionales.
- Estructura con política **LIFO** (*last in, first out*), el último elemento insertado es el primer elemento en ser extraído de la pila.

Operaciones principales:

- Pila vacía
- Sacar un elemento (POP).
- Meter un elemento (PUSH).

Implementación:

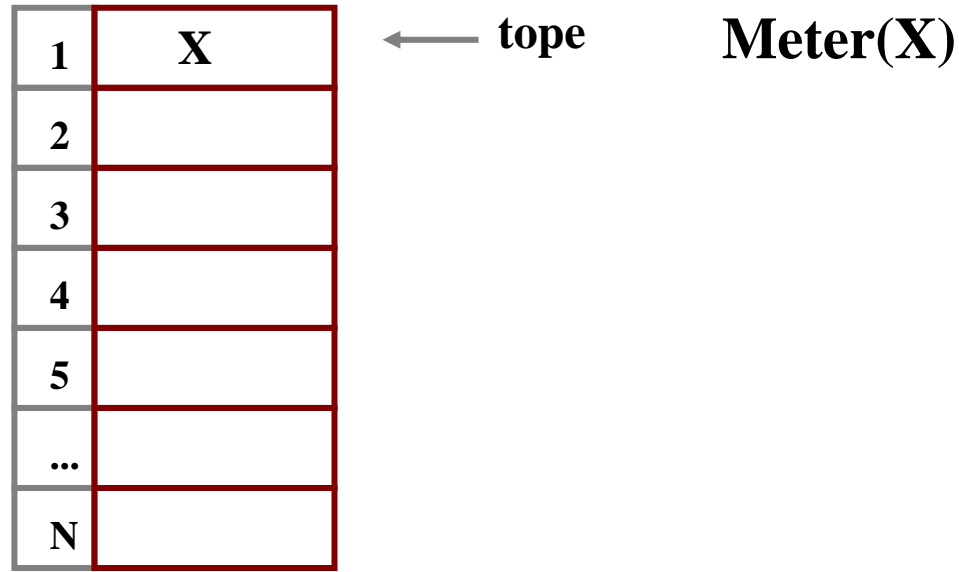
pila: arreglo [1..N] de *tipo_elemento*

tope = 0

tope = 0 → La pila está vacía.

1	
2	
3	
4	
5	
...	
N	

Array → PILA



Array → PILA

Al insertar un elemento en la pila, se incrementa el contador *tope*, que señalará el índice del vector donde se encuentra el último elemento insertado

1	X
2	X
3	
4	
5	
...	
N	

← tope

Meter(X)

Array → PILA

1	X
2	X
3	X
4	X
5	
...	
N	

← **tope**

A medida que se introducen elementos en la *pila*, se incrementa el *tope*

Array → PILA

20/05/05

Pila.Vacia () : Lógico

{ pos: regresa verdadero si la cadena esta vacía }

1	Si $\text{tope}=0$ entonces retornar Verdadero sino retornar Falso	tope : indica el indice correspondiente al tope de la pila
1	$\text{tope} = 0 \Rightarrow \text{VERDADERO}$	Regresa verdadero pues la pila está vacía.
2	$\text{tope} \neq 0 \Rightarrow \text{FALSO}$	Regresa falso pues la pila no está vacía.

20/05/05

Meter (Elemento: Tipo_Elemento)

{pre: Elemento < > Vacio}

{pos: pila=pila+elemento}

1	tope = tope + 1	tope: indica el índice correspondiente al tope de la pila
2	pila [tope] = elemento	
1	Elemento = X => pila[tope]=X	Inserta X (del tipo Tipo_Elemento) en pila[tope]

20/05/05

Sacar ()

{pre: pila < > Pila.Vacia}

{pos: pila = pila - elemento}

1	Si Pila.Vacia() entonces desplegar “Error underflow” sino tope = tope – 1 retornar pila[tope + 1]	tope: indica el índice correspondiente al tope de la pila
1	pila[tope]=X, sacar() => X, tope= tope-1	Devuelve el elemento en el tope de la pila, y decrementa el contador “tope”.

COLAS.

- Se implementan eficientemente con matrices monodimensionales.
- Estructuras con política **FIFO** (*first in, first out*), el primer elemento que se inserta es el primer elemento en ser sacado de la cola.

Operaciones principales:

- Poner un elemento: *Enqueue*
- Quitar un elemento: *Dequeue*

Implementación:

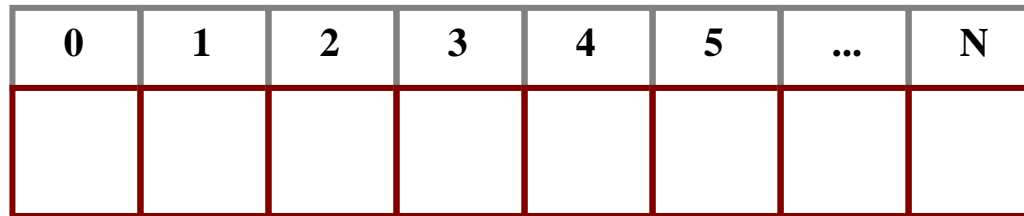
cola: matriz $[0.. N-1]$ de *tipo_elemento*

head = 0

tail = 0

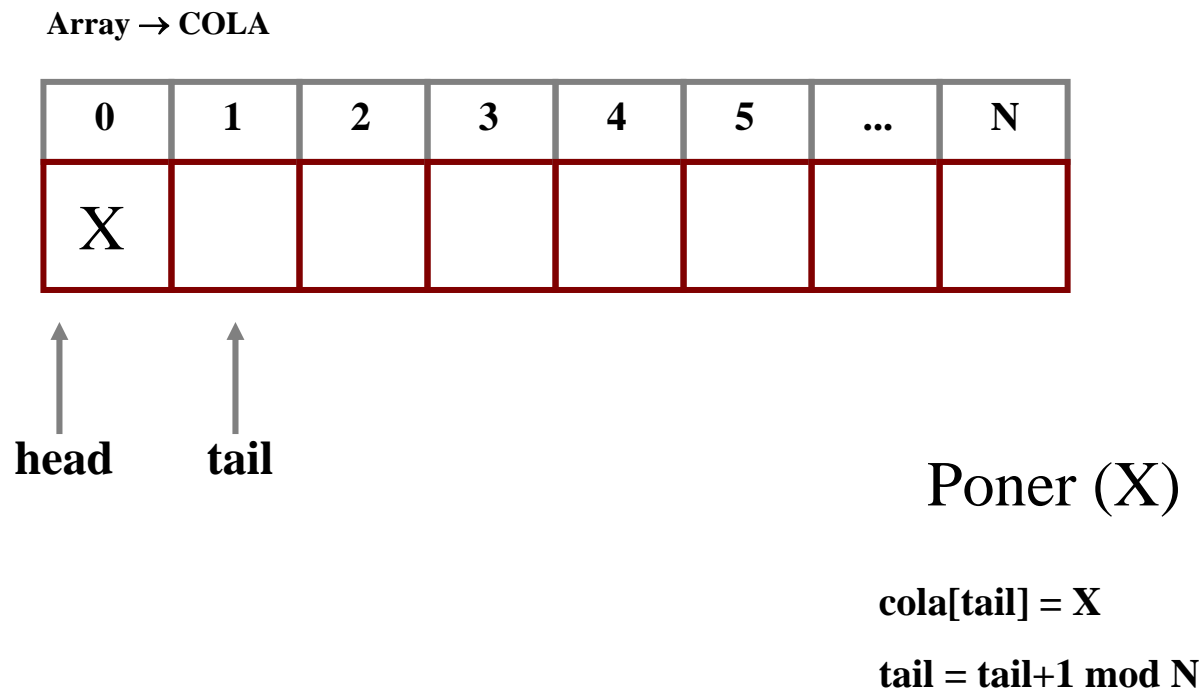
Cola vacía: $\text{head} = \text{tail}$

Array \rightarrow COLA



↑ ↑
head tail

Al insertar un elemento en la cola, se incrementa el contador *tail*, que señala el índice del vector donde se podrá insertar un nuevo elemento. *head* señala la cabeza de la *cola*.



A medida que se insertan elementos en la cola, se incrementa el contador *tail*.

Array → COLA

0	1	2	3	4	5	...	N
X	X	X	X	X			

↑
head

↑
tail

Poner (X)

$\text{cola}[\text{tail}] = X$

$\text{tail} = \text{tail} + 1 \bmod N$

Al eliminar elementos en la *cola*, se incrementa el valor del contador *head*, que señala el próximo elemento en salir de la *cola*.

Array → COLA

0	1	2	3	4	5	...	N
X	X	X	X	X			

↑
head

↑
tail

Quitar ()

$x = \text{cola}[\text{head}]$

$\text{head} = \text{head} + 1 \bmod N$

A medida que se eliminan elementos en la *cola*, se incrementa el contador *head*.

Array → COLA

0	1	2	3	4	5	...	N
X	X	X	X	X			

↑
head

↑
tail

Quitar ()

$x = \text{cola}[\text{head}]$

$\text{head} = \text{head} + 1 \bmod N$

20/05/05

Poner (Elemento: Tipo_Elemento)

{pre: Elemento < > Vacio}

{pos: cola = cola+elemento}

1	cola [tail]= elemento	N: número de elementos máximos que se pueden almacenar en la cola. Declarada globalmente
2	tail = tail+1 mod N	
1	Elemento = X => cola[tail]=X	Inserta X (del tipo Tipo_Elemento) en cola[tail]

20/05/05

Quitar ()

{pre: Elemento < > Vacio}

{pos: cola = cola+elemento}

1	X = cola [head]= elemento	N: número de elementos máximos que se pueden almacenar en la cola.
2	Head = head+1 mod N	
3	retornar X	
1	cola[head]=X, Quitar() => X, head se incrementa	Devuelve el elemento que se encuentra de primero en la cola.

Tiempos de Ejecución:

Matrices unidimensionales:

Calcular una dirección: constante

Leer un elemento o cambiar su valor: $O(1)$

Recorrer el arreglo: $\Theta(N)$

Matrices n-dimensionales:

Calcular una dirección: constante

Leer un elemento o cambiar su valor: $O(1)$

Recorrer la matriz, por ejemplo si fuera una matriz $n \times n$: $\Theta(n^2)$

TABLAS ASOCIATIVAS

- Sus índices no están restringidos entre dos cotas:

$T[1]$ $T[10^6]$

1	10^6

- Se pueden utilizar cadenas en vez de números como índices:

$T["juan"]$ $T["pedro"]$

juan	pedro

La manera más sencilla de implementación: LISTAS ENLAZADAS

tipo *lista_tabla* = \uparrow *nodo_tabla*

tipo *nodo_tabla* = **registro**

índice: *tipo_índice*

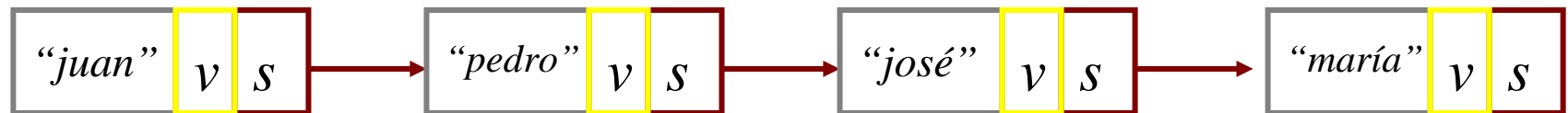
valor: *información*

siguiente: \uparrow *nodo_tabla*



A diferencia de las matrices, las tablas no se pueden implementar de tal manera que se garantice que todas sus búsquedas requieran un tiempo constante.

T[“maria”]:



Con ésta implementación, el acceso a T[“maria”], se consigue recorriendo la lista hasta que se encuentre “maría” en el campo *índice* de un nodo, o hasta que se alcance el final de la lista. Esta implementación no es eficiente.

Codificación Dispersiva (*hash coding*)

o dispersión (*hashing*)

Sea U el universo de índices potenciales, y sea $N \ll |U|$, donde N son los posibles valores del resultado de la siguiente función:

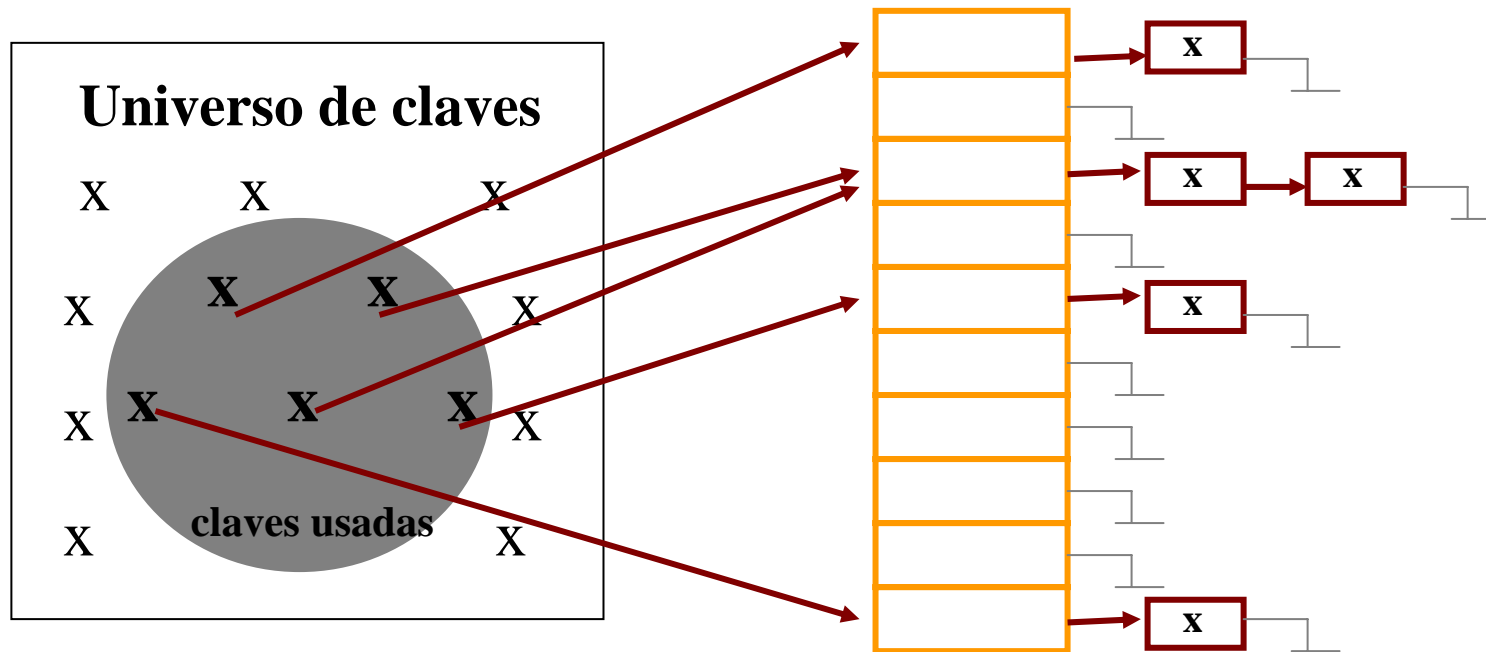
Función de Dispersión:

$$h : U \rightarrow \{0, 1, 2, \dots, N-1\}$$

h proporcionará el índice del vector

$h(x) \neq h(y)$, para la mayoría de los pares $x \neq y$

Si $x \neq y$ y $h(x) = h(y)$: **tabla de dispersión abierta o con encadenamiento**



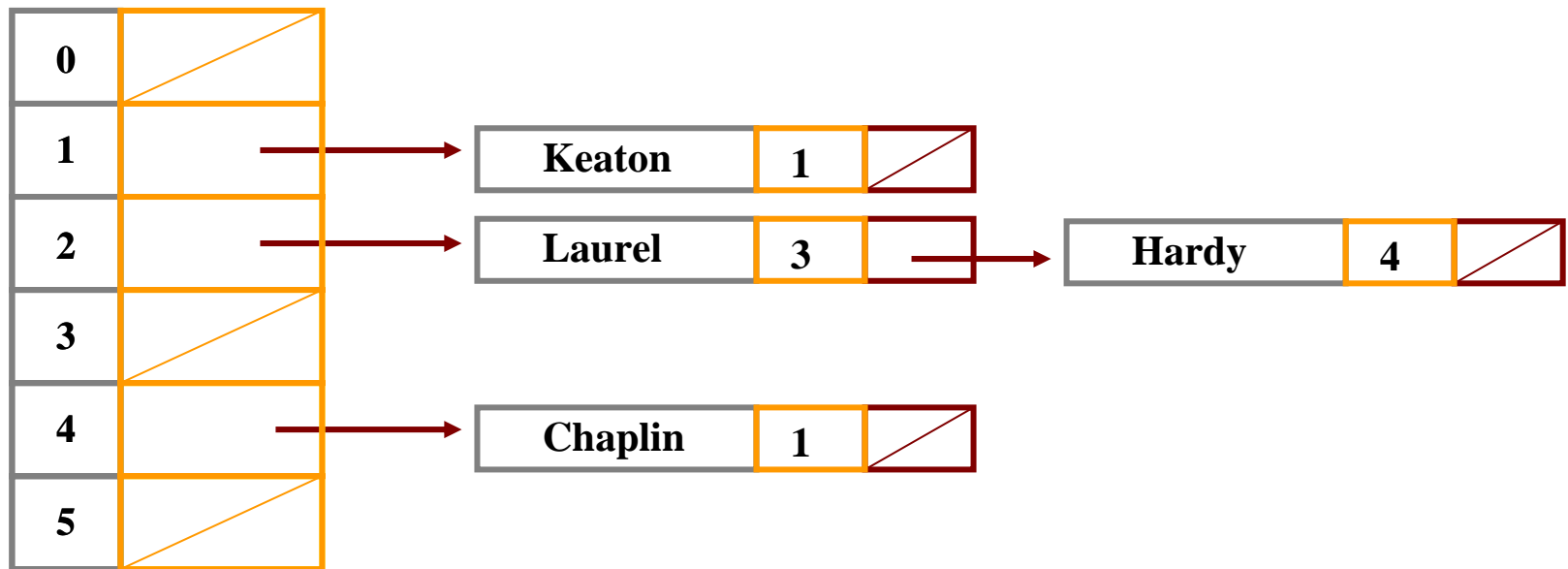
Función Hash

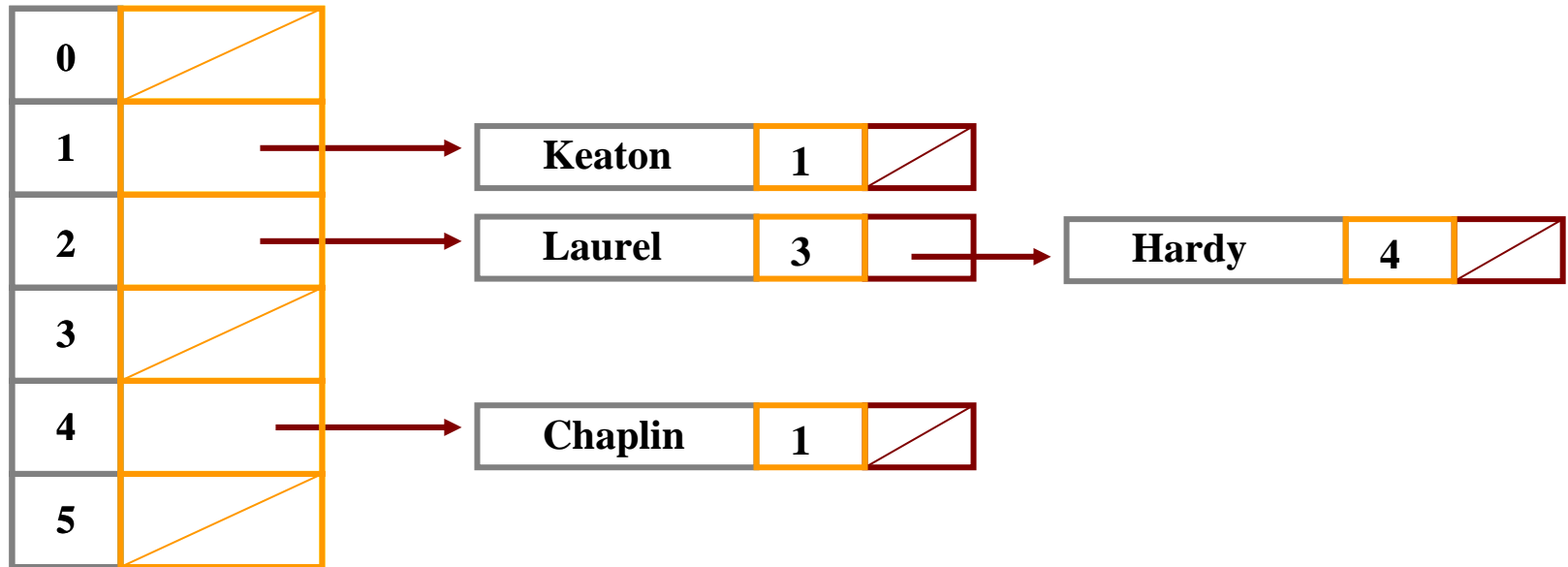
Listas enlazadas

Supongase una tabla asociativa T . Sea $h(x)$ una determinada función de dispersión, y x , las posibles claves que se desean almacenar en la tabla asociativa.

$$T[\text{“Laurel”}] = 3 \quad T[\text{“Chaplin”}] = 1 \quad T[\text{“Hardy”}] = 4 \quad T[\text{“Keaton”}] = 1$$

Ejemplo: $N=6$, $h(\text{“Laurel”}) = h(\text{Hardly}) = 2$, $h(\text{Chaplin}) = 4$, $h(\text{Keaton}) = 1$.





Factor de carga:

El factor de carga de una tabla es m/N , donde m es el número de índices distintos que se han almacenado en la tabla, y N es el tamaño de la matriz que se emplea para implementarla.

Para el ejemplo, el factor de carga es $4/6 = 0,666\dots$

ESTRUCTURAS DE CONJUNTOS DISJUNTOS

$$S = \{ S1, S2, S3, \dots, SN \}$$

- Rótulo del conjunto.
- Operaciones de buscar.
- Operaciones de fusionar.

- Rótulo del conjunto:

$$S = \{1, 2, 3, 4\} \quad \text{rótulo : 2}$$

$$S = \{11, 13, 15, 17\} \quad \text{rótulo : 15}$$

Ejemplo: El rótulo es el menor elemento del conjunto

$$S = \{1, 3, 5, 7, 9\} \quad \text{rótulo : 1}$$

$$S = \{2, 4, 6, 8, 10\} \quad \text{rótulo : 2}$$

$$S = \{20\} \quad \text{rótulo : 20}$$

Implementación con vectores:

$$S = \{1, 3, 5, 7, 9\}$$

rótulo : 1

$$S = \{2, 4, 6, 8, 10\}$$

rótulo : 2

conjunto: matriz [1 .. N] de tipo entero

1	2	3	4	5	6	7	8	9	10
1	2	1	2	1	2	1	2	1	2

20/05/05

Buscar (Elemento: Entero) : Entero

{pre: vector<>Vacío}

{pos:retorna un entero}

1	retornar conjunto[elemento]	
1	Buscar(5) => <i>conjunto[5] = 1</i>	Regresa el valor de <i>conjunto[elemento]</i>
2	Buscar(2) => <i>conjunto[2] = 2</i>	
3	Buscar(7) => <i>conjunto[7] = 1</i>	
4	Buscar(10) => <i>conjunto[10] = 2</i>	

1	2	3	4	5	6	7	8	9	10
1	2	1	2	1	2	1	2	1	2

20/05/05

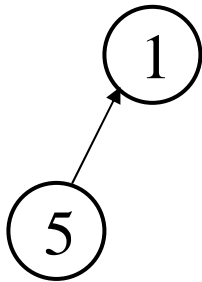
Unir (rótulo1: Entero, rótulo2: Entero)

{pre: vector<>Vacío}

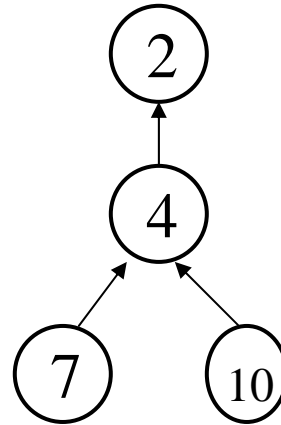
1	$i = \min(\text{rótulo1}, \text{rótulo2})$	min(): devuelve el mínimo de dos valores max(): devuelve el máximo de dos valores
2	$j = \max(\text{rótulo1}, \text{rótulo2})$	
3	[Si $\text{conjunto}[k]=j$ entonces $\text{conjunto}[k]=i$] $k=1,N$	
1	Unir(1,2) $\text{conjunto}[2]=1$ $\text{conjunto}[4]=1$ $\text{conjunto}[6]=1$ $\text{conjunto}[8]=1$ $\text{conjunto}[10]=1$	

1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1

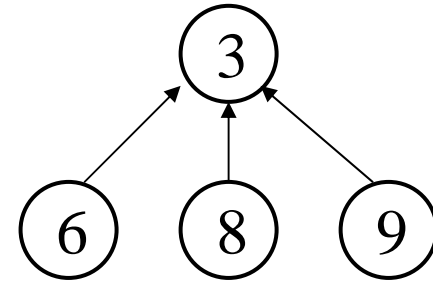
Para mejorar la implementación: representar cada conjunto como un árbol con raíz.



$S1 = \{1,5\}$



$S2 = \{2,4,7,10\}$



$S3 = \{3,6,8,9\}$

1	2	3	4	5	6	7	8	9	10
1	2	3	2	1	3	4	3	3	4

20/05/05

Buscar1 (Elemento: Entero) : Entero

{pre: vector<>Vacío}

{pos:retorna un entero}

1	$r = \text{elemento}$	
2	$(\text{conjunto}[r] \langle \rangle r) [r = \text{conjunto}[r]]$	
3	retornar r	
1	<p>Buscar1(10):</p> <p>$r = 10; \text{conjunto}[10] \langle \rangle 10$</p> <p>$r = 4; \text{conjunto}[4] \langle \rangle 4$</p> <p>$r = 2; \text{conjunto}[2] == 2$</p> <p>retorna $r = 2.$</p>	Regresa el valor de $\text{conjunto}[\text{elemento}]$

1	2	3	4	5	6	7	8	9	10
1	2	3	2	1	3	4	3	3	4

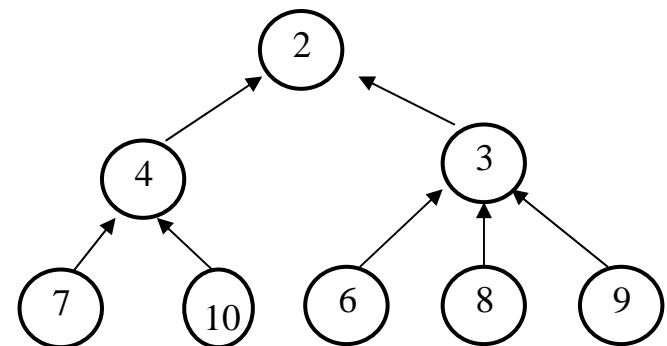
20/05/05

Unir1 (rótulo1: Entero, rótulo2: Entero)

{pre: vector<>Vacío}

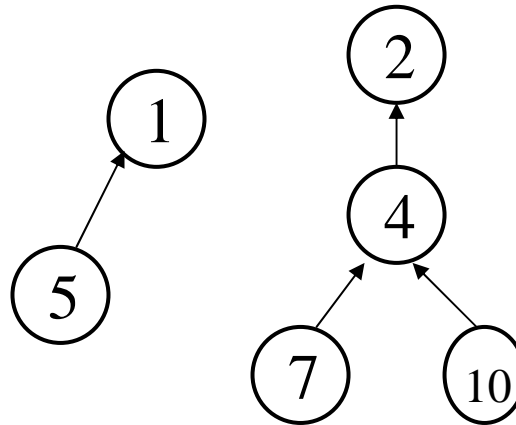
1	Si rótulo1 < rótulo2 entonces <i>conjunto[rótulo2] = rótulo1</i> sino <i>conjunto[rótulo1] = rótulo2</i>	
1	Unir(2,3). 2 < 3, por lo tanto <i>conjunto[3]=2</i>	

1	2	3	4	5	6	7	8	9	10
1	2	2	2	1	3	4	3	3	4

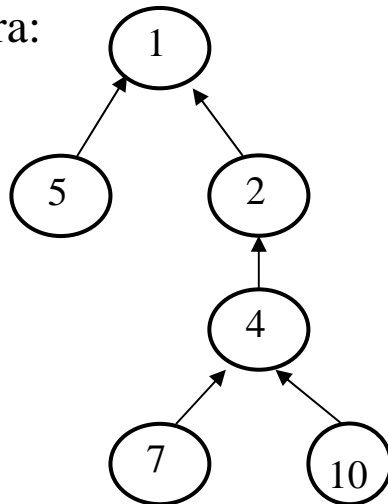


Mejorando la implementación: tomar en cuenta la altura del árbol para asignar el rótulo.

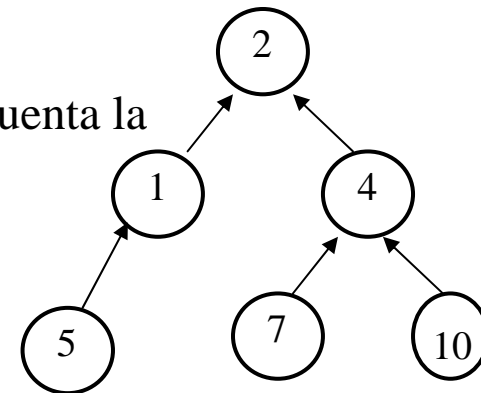
Dados dos árboles



Sin tomar en cuenta la altura:



Tomando en cuenta la altura:



20/05/05

Unir2 (rótulo1: Entero, rótulo2: Entero)

{pre: vector<>Vacío}

1	<p>Si $altura[rótulo1] = altura[rótulo2]$ entonces $altura[rótulo1] = altura[rótulo1] + 1$ $conjunto[rótulo2] = rótulo1$</p> <p>sino</p> <p>si $altura[rótulo1] > altura[rótulo2]$ entonces $conjunto[rótulo2] = rótulo1$</p> <p>sino</p> <p>$conjunto[rótulo1] = rótulo2$</p>	
1	<p>Unir2(1,2). $altura[1] < altura[2]$, por lo tanto $conjunto[1] = 2$</p>	

Como resultado del algoritmo anterior, se obtendría:

1	2	3	4	5	6	7	8	9	10
2	2	3	2	1	3	4	3	3	4