Estructuras de datos
Clase teórica 4

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN

Contenido

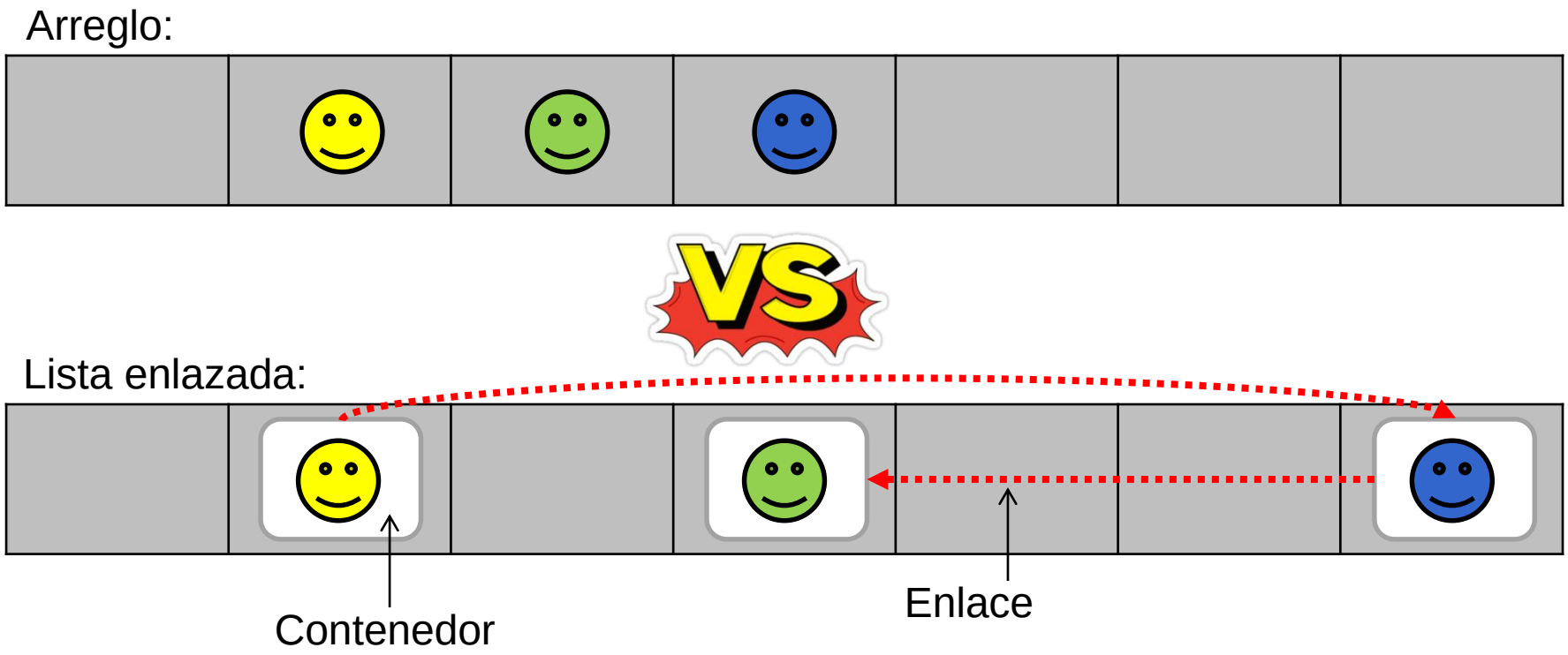
- Listas enlazadas

Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Lista enlazada

Una lista enlazada (o ligada) es una estructura de datos que permite almacenar elementos pero, a diferencia de los arreglos, no necesariamente en posiciones consecutivas de memoria (ni siquiera deben estar “en orden”).



Lista enlazada

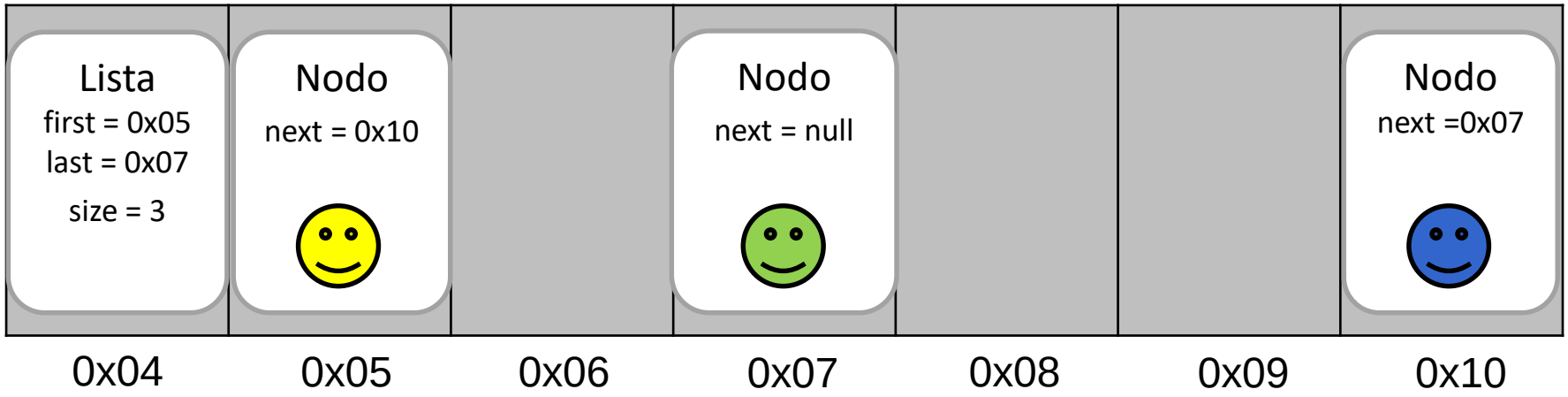
En una lista, cada elemento se encuentra “contenido” dentro de un nodo el cual generalmente tiene, además del elemento, un apuntador o referencia a otro nodo (o a otros dos).

Siendo así, la lista como tal no contiene mayor cosa, generalmente solo la referencia a ciertos nodos de la lista (normalmente primero y último) y el tamaño de la misma.

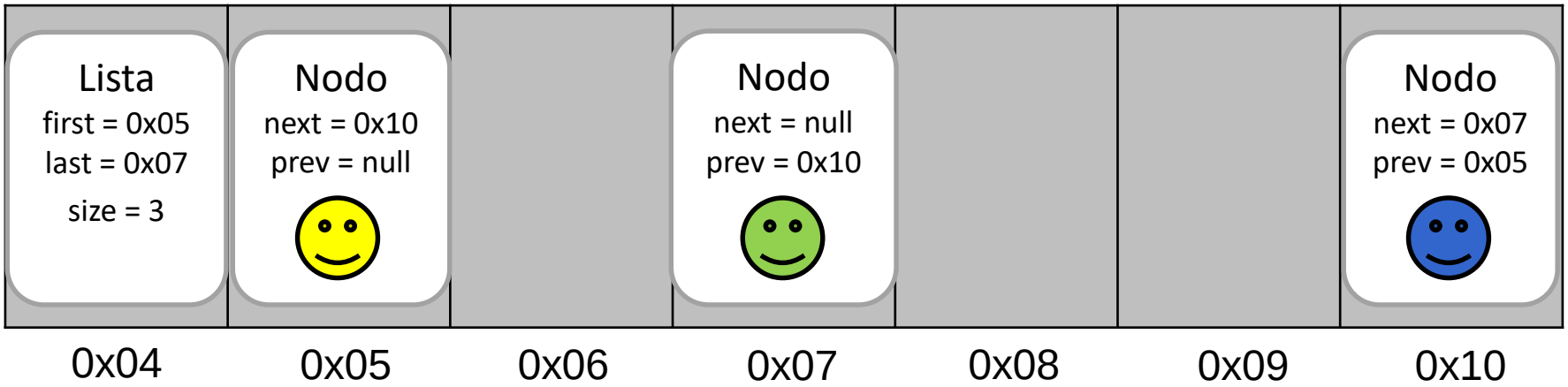
Dependiendo de si cada nodo apunta solamente al siguiente en la lista diremos que esta es simplemente enlazada, o si apunta tanto al siguiente como al anterior diremos que es doblemente enlazada. La diferencia entre una y otra son las operaciones que permiten (en una doblemente enlazada por ejemplo se puede borrar el último elemento).

Lista enlazada

Lista simplemente enlazada:



Lista doblemente enlazada:



Indexación en una lista enlazada

Ya que conocemos la “arquitectura” de una lista enlazada, pasemos a analizar cada una de las operaciones básicas: indexación, inserción, borrado, y búsqueda. Comencemos con la indexación y consideremos una lista doblemente enlazada, sabiendo que en una simplemente enlazada los procesos son similares.

Como la lista en realidad solo “sabe” donde están el primer y último elemento, acceder a ellos dos implica *O(1)*, mientras que para el resto de índices “llegar a ellos”, saltando uno por uno.

Atención, el siguiente código no es necesariamente lo que hace Java, es mas bien un pseudocódigo para entender la eficiencia.

Indexación en una lista enlazada

```
j // índice del elemento a ser accesado
if (j<0 || j>=lista.size)
   return ERROR
else{
   i = 0
   if (j < lista.size/2){
      p = lista.first // el primer nodo de la lista
      for (i=0; i<j; i++)
         p = p.next
   }
   else{
      p = lista.last // el ultimo nodo de la lista
      for (i=0; i<lista.size-j; i++)
         p = p.prev
   }
   return p
}
```

Como puede verse, en el peor de los casos hay que recorrer la mitad de los elementos, por tanto este proceso es *O(n)*

Inserción en una lista enlazada

En la inserción hay que considerar tres casos:

- Cuando se realiza al final de la lista
- Cuando se realiza al inicio de la lista
- Cuando se realiza en un índice específico

El primer caso es sumamente simple, basta con solicitar nuevo espacio de memoria para un nodo que contendrá el elemento, luego enlazar dicho nodo con el último nodo de la lista (a menos que la lista esté vacía, caso en el cual se omite este paso y en cambio se actualiza la referencia del primero), y finalmente actualizar la referencia del último.

Inserción en una lista enlazada

En síntesis, es un procedimiento más o menos así:

```
// e es el elemento a ingresar
p = new nodo
p.elm = e
if (lista.size > 0){
   (lista.last).next = p
    p.prev = lista.last
}
else
   lista.first = p
lista.last = p
lista.size++
```

Como puede verse, incluso cuando hay muchos elementos en la lista, la cantidad de operaciones a realizar es constante y por tanto este proceso es *O(1)*

Inserción en una lista enlazada

Insertar al inicio de la lista es igualmente simple:

```
// e es el elemento a ingresar
p = new nodo
p.elm = e
if (lista.size > 0){
   (lista.first).prev = p
   p.next = lista.first
}
else
   lista.last = p
lista.first = p
lista.size++
```

Igual que en la inserción al final, incluso cuando hay muchos elementos en la lista, la cantidad de operaciones a realizar es constante y por tanto este proceso es *O(1)*

Inserción en una lista enlazada

En el tercer caso, cuando la inserción se desea realizar en un índice específico *k* de la lista se debe llegar primero al nodo que está ocupando esa posición (verificando obviamente que ($0 \leq k \leq size-1$).

Una vez allí se solicita espacio de memoria para el nodo que contendrá el nuevo elemento y se actualizan los enlaces correspondientes.

Como quedará claro a continuación, dado que llegar a una posición implica $O(n)$ y actualizar los enlaces $O(1)$, podemos decir que en total esta operación implica $O(n)+O(1) = O(n)$

Inserción en una lista enlazada

En síntesis, es un procedimiento más o menos así:

```
if (k < 0 || k >= lista.size)
   return ERROR
else{
   q = lista.first
   for (i=0; i<k; i++)
      q = q.next
   p = new nodo
   p.elm = e
   p.prev = q.prev
   (q.prev).next = p
   q.prev = p
   p.next = q
   lista.size++
}
```

Borrado en una lista enlazada

En el caso del borrado, igual que con arreglos, hay que analizar dos casos:

- Si se conoce el índice del elemento
- Si no se conoce

En el primer caso el análisis es muy similar al de la inserción en el sentido que:

1. Si se va a borrar el último elemento hay que: 1) actualizar el enlace *next* del penúltimo elemento, 2) actualizar la referencia del último, y 3) actualizar el tamaño de la lista. Todo lo cual implica *O(1)*
2. Si se va a borrar el primer elemento hay que: 1) actualizar el enlace *prev* del segundo elemento, 2) actualizar la referencia del primero, y 3) actualizar el tamaño de la lista. Todo lo cual implica *O(1)*

Borrado en una lista enlazada

3. Si se va a borrar un elemento diferente al primero o al último hay que: 1) llegar hasta ese elemento 2) actualizar los enlaces de su anterior y de su siguiente, y 3) actualizar el tamaño de la lista. Todo lo cual implica *O(n)*

Ahora, cuando no se conoce el índice del elemento que se va a borrar básicamente lo que hay que hacer es una búsqueda del elemento, lo cual ya vimos implica *O(n)*. En caso de encontrarlo hay que: 1) actualizar los enlaces de su anterior y de su siguiente, y 2) actualizar el tamaño de la lista. Todo esto implica *O(n).* Adicionalmente, si el elemento a borrar resulta ser o el primero o el último hay que actualizar las referencias correspondientes.

Resumiendo, este proceso implica *O(n)+O(1) = O(n).*

Búsqueda en una lista enlazada

¿Será que al igual que en los arreglos, en caso que los elementos de una lista estén ordenados, es posible usar búsqueda binaria?

```
//e es el elemento a ser buscado
int ini = 0, fin = L.size, pos;
while (ini <= fin){
     pos = (ini+fin)/2;
     if (L.get(pos) == e)
          return pos;
     else if (e < L.get(pos))
          fin = pos-1;
     else
          ini = pos+1;
 }
 return -1;
```

¿Cuál es la eficiencia de este algoritmo?

$f(n) = 3+4*\log_2(n)*n+1$, por tanto $O(n*\log_2(n))$

Y si en vez de usar el indexado, aprovechamos la arquitectura de la lista moverse hacia adelante y hacia atrás partiendo de un determinado nodo ¿Cómo se optimizaría el algoritmo?

```
int ini = 0, fin = L.size(), pos, p = L.first;
pos = (ini+fin)/2;
for(i=0; i<pos; i++)
   p = p.next
while (p.elm != e && ini <= fin){
   if (e < p.elm){
      fin = pos - 1;
      for(i=0; i<pos=(ini+fin)/2; i++)
         p = p.prev;
   }
   else{
      ini = pos + 1;
      for(i=0; i<pos=(ini+fin)/2; i++)
         p = p.next;
   }
}
if (p.elm == e)
   return pos;
else
   return -1;
```

¿Cuál es la eficiencia de este algoritmo?

*f (n)= 5+6*n+2,* por tanto *O(n)*

Búsqueda en una lista enlazada

Si lo mejor que podemos hacer usando búsqueda binaria es *O(n)*, ¿por qué no usar simplemente búsqueda lineal?

```
p = lista.first;
i = 0;
while (i < lista.size && p.elm != e){
   p = p.next;
   i++;
}
if (p.elm == e)
   return i;
else
   return -1;
```

¿Cuál es la eficiencia de este algoritmo?

*f (n)= 2+3*n+2,* por tanto *O(n)*

Igual al de la búsqueda binaria pero con constantes menores y siendo bastante más simple

Tabla resumen

Recapitulando la clase de hoy tenemos que:

Estructura	Inserción	Indexación	Búsqueda	Borrado
Lista enlazada	*O(n)* Excepto que sea al inicio o al final de la lista, en cuyo caso es *O(1)*	*O(n)* Excepto que sea el primero o el último, en cuyo caso es *O(1)*	*O(n)*	*O(n)* Excepto que sea el primero o el último, en cuyo caso es *O(1)*

Con los elementos de juicio que hemos adquirido hasta el momento podemos discutir las siguientes cuestiones:

¿Cuándo es conveniente usar un arreglo estático en vez de uno dinámico?

¿Cuándo es conveniente usar una lista enlazada en vez de un arreglo dinámico?