Estructuras de datos
Clase teórica 3

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MEDELLÍN

Contenido

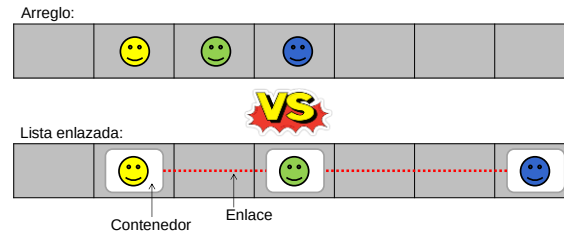
- Listas simple y doblemente enlazadas

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").



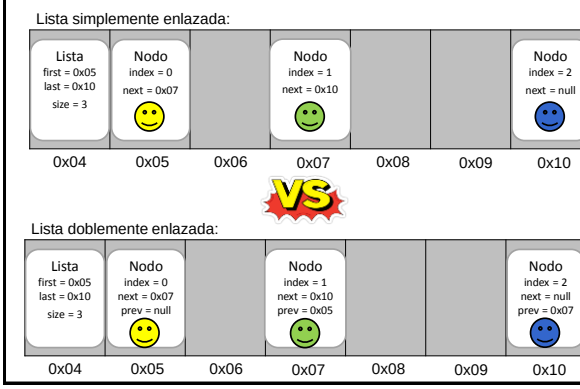
Lista enlazada

En una lista, cada elemento se encuentra "contenido" dentro de un nodo el cual generalmente contiene, además del elemento, un apuntador o referencia a otro nodo (o a otros dos) y el índice de dicho elemento dentro de la lista.

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 ir de atrás hacia adelante, mientras que en la simple no)

Lista enlazada



Búsqueda en una lista enlazada

Ya que conocemos la "arquitectura" de una lista enlazada, pasemos a analizar cada una de las operaciones básicas: inserción, indexación, búsqueda, y borrado.

Comencemos con la búsqueda y consideremos una lista doblemente enlazada, sabiendo que en una simplemente enlazada los procesos son similares pero más sencillos.

Lo primero que hay que analizar es que, incluso si los elementos en la lista estuvieran en orden, no es posible realizar búsqueda binaria pues, como vimos bien en la clase anterior, dicho proceso solo funciona con arreglos donde los elementos están en posiciones contiguas de memoria.

Búsqueda en una lista enlazada

Siendo así, la búsqueda debe ser lineal, pero con dos diferencias fundamentales respecto a como lo hacemos con arreglos:

1. El punto de partida no es el inicio del arreglo sino el primer nodo de la lista (del cual se tiene la referencia).
2. Para pasar de un elemento al siguiente no nos movemos un "espacio" hacia adelante en memoria (pues en la lista no están necesariamente contiguos ni en orden), si no que utilizamos la referencia que tiene cada nodo del siguiente.

Búsqueda en una lista enlazada

En síntesis, es un procedimiento más o menos así (atención, esto no es Java, es mas bien un pseudocódigo):

```
// e es el elemento buscado
p = lista.first // el primer nodo de la lista
i = 0
while (i < lista.size && p.elm != e){
   p = p.next
   i++
}
if (p.elm = e)
   return i
else
   return -1
```

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

Indexación en una lista enlazada

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 es necesario hacer un recorrido lineal:

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

Como puede verse, en el peor de los casos hay que recorrer todos 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
p.index = lista.size
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 relativamente simple. Se realiza un procedimiento similar a cuando es al final pero con un paso adicional: actualizar los índices del resto de elementos:

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

Como puede verse, la actualización de los enlaces implica *O(1)* pero la de los índices *O(n)*, por tanto el proceso en general es *O(1)+O(n) = O(n)*

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 ≤ k ≤ size-1*).

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

Finalmente es necesario recorrer los elementos restantes de la lista (de *k* hacia adelante) actualizando sus índices.

Como quedará claro a continuación, dado que llegar a una posición implica *O(n)*, actualizar los enlaces *O(1)*, y actualizar los índices siguientes *O(n),* podemos decir que en total esta operación implica *O(n)+O(1)+O(n) = 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.index = k
   p.prev = q.prev
   (q.prev).next = p
   q.prev = p
   p.next = q
   for (i=0; i<lista.size-k-1; i++){
      q.index++
      q = q.next
   }
   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) disminuir el índice de todos los elementos en una unidad, 3) actualizar la referencia del primero, y 4) actualizar el tamaño de la lista. Todo lo cual implica *O(n)*

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, 3) actualizar los índices de todos los elementos a partir de su siguiente en adelante, y 4) 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, 2) actualizar los índices de todos los elementos a partir de su siguiente en adelante, y 3) 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(n) = O(n).*

Tabla resumen

Recapitulando la clase de hoy tenemos que:

Estructura	Inserción	Indexación	Búsqueda	Borrado
Lista enlazada	*O(n)* Excepto que sea al final de la lista (lo normal), 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 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?