Algoritmos y estructuras de datos

Listas enlazadas

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

21 de agosto de 2024

Citas

John F. Kennedy

Si no podemos eliminar ahora nuestras diferencias, al menos podemos hacer al mundo seguro para la diversidad. En el análisis final, nuestro enlace común más elemental es que todos habitamos este pequeño planeta. Todos respiramos el mismo aire…Y todos somos mortales.

Doctor Who

¡El enlace está roto! ¡De regreso a la Guerra Temporal, Rassilon! ¡De regreso al infierno!

Ella Wheeler Wilcox

Provee el enlace, y así la tierra y el cielo se unirán en una cadena continua de vida sin fin.

Ventajas y desventajas de usar arreglos

Si queremos un arreglo con entradas `a[0]..a[n-1]`:

1. Se puede declarar fácilmente, ya sea de forma estática si `n` es una constante, o de forma dinámica en caso contrario.
2. La entrada `a[i]` se puede leer o escribir en una unidad de tiempo.
3. Como las entradas son consecutivas en memoria, se puede avanzar a la siguiente (o a la anterior) en una unidad de tiempo.

Por otro lado:

1. Un arreglo estático es de tamaño fijo (a veces muy grande o muy pequeño).
2. Un arreglo dinámico puede cambiar de tamaño, pero tal vez se requieren `n` unidades de tiempo para mover el arreglo a otro lugar.
3. Insertar o eliminar un elemento en la posición `i` requiere hasta `n-i` unidades de tiempo para mover `a[i]..a[n-1]`.

Una estructura de datos distinta

Nos gustaría una estructura de datos que conservara esta ventaja:

1. Se puede avanzar al siguiente elemento en una unidad de tiempo.

Al mismo tiempo que no tenga las desventajas de los arreglos:

2. Puede aumentar su tamaño en uno en una unidad de tiempo.
3. Se puede insertar o eliminar un elemento en una unidad de tiempo.

A cambio de eso estamos dispuestos a que:

4. Cada elemento ocupe más espacio del estrictamente necesario.
5. Encontrar el elemento `i` tome `i` unidades de tiempo.

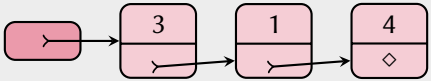
Observaciones

Esto sería útil si se van a procesar los elementos uno tras otro del primero al último.
También si sólo se va a insertar o eliminar el primer elemento.

Lista enlazada

Una lista enlazada puede estar vacía o consistir de una secuencia de nodos, donde cada nodo contiene un dato y sabe dónde está el siguiente nodo o, alternativamente, que no hay siguiente nodo.

Ejemplo



Aplicaciones de listas enlazadas

Secuencias

Una secuencia s consiste de $n \geq 0$ elementos $(s_1, \ldots, s_n)$ en las posiciones $1, \ldots, n$.
Algunas operaciones con una secuencia son:

1. Crear una secuencia s vacía con $n = 0$.
2. Regresar el elemento s_i que está en la posición $1 \leq i \leq n$.
3. Regresar el sucesor s_{i+1} del elemento en la posición $1 \leq i < n$.
4. Insertar un elemento x en la posición $1 \leq i \leq n + 1$, incrementando la posición de los elementos $s_i, \ldots, s_n$ e incrementando n.
5. Extraer un elemento x de la posición $1 \leq i \leq n$, decrementando la posición de los elementos $s_{i+1}, \ldots, s_n$ y decrementando n.

Con listas enlazadas estas operaciones toman 1, i, i, i, i pasos, respectivamente.

Aplicaciones de listas enlazadas

Enteros largos

Un entero largo s consiste de $n \geq 1$ dígitos $(s_0, \ldots, s_{n-1})$ en las posiciones $0, \ldots, n-1$. El entero largo representado (en base b) es

$$b^0 s_0 + b^1 s_1 + \cdots + b^{n-1} s_{n-1}.$$

Algunas operaciones con enteros largos son:

1. La asignación de un entero largo de n dígitos.
2. La comparación de dos enteros largos de n y m dígitos.
3. La suma de dos enteros largos de n y m dígitos.
4. La multiplicación de dos enteros largos de n y m dígitos.

Con listas enlazadas estas operaciones toman n, $n+m$, $n+m$, nm pasos, respectivamente.

Aplicaciones de listas enlazadas

Polinomios

Un polinomio s consiste de $n \geq 0$ monomios $((a_1, b_1), \ldots, (a_n, b_n))$ donde $a_i \neq 0$ y $0 \leq b_1 < \cdots < b_n$. El polinomio representado (en la variable x) es

$$a_1 x^{b_1} + \cdots + a_n x^{b_n}.$$

Algunas operaciones con polinomios son:

1. La asignación de un polinomio de n monomios.
2. La comparación de dos polinomios de n y m monomios.
3. La suma de dos polinomios de n y m monomios.
4. La multiplicación de dos polinomios de n y m monomios.

Con listas enlazadas estas operaciones toman n, $n + m$, $n + m$, nm pasos, respectivamente.

Lista enlazada

Definición de tipos asociados a una lista

Definiremos un tipo estructurado `nodo` para representar un nodo y también un tipo `lista` para representar una lista enlazada. El tipo `nodo` consiste de un dato `a` y un apuntador `sig` al siguiente nodo (que valdrá `NULL` si no hay siguiente nodo).

```
typedef struct nodo {
  int a;              // dato almacenado
  struct nodo *sig;   // enlace al siguiente
} nodo;
```

Por otro lado, el tipo `lista` es un apuntador a `nodo`.

```
typedef nodo *lista;
```

Note que los tipos `nodo *`, `struct nodo *` y `lista` son equivalentes.

Lista enlazada

Creación de un nodo

Esta función pide la memoria para un nodo y llena sus campos.

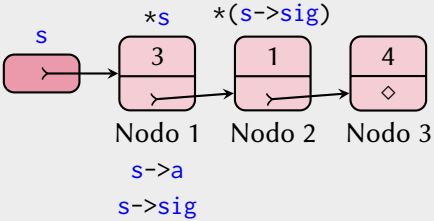
```
nodo *creaNodo(int x, nodo *p) {
  nodo *t = (nodo *) malloc(sizeof(nodo));
  t->a = x;
  t->sig = p;
  return t;
}
```

Lista enlazada

De regreso al ejemplo

Declaramos una lista enlazada vacía `s` con `lista s = NULL;`

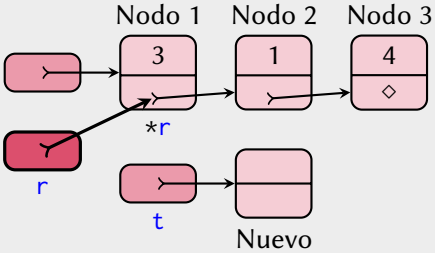
La siguiente es una lista enlazada que ya tiene tres nodos:



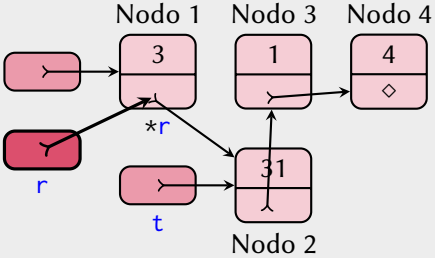
Lista enlazada

Ejemplo de inserción de un nodo

Justo después de pedir el nodo



Justo antes de terminar



Lista enlazada

Inserción de un nodo

Como esto implica modificar una lista enlazada `s`, requerimos de una referencia `r` al enlace en donde queremos insertar el nodo. Por ejemplo, si queremos insertar al inicio, esa referencia sería `r = &s`.

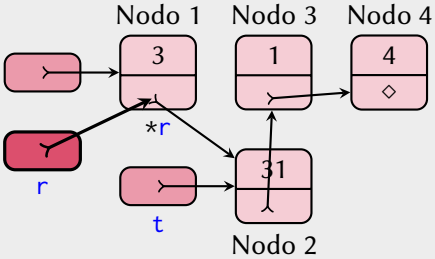
```
void insertaNodo(nodo **r, int x) {
  nodo *t = creaNodo(x, *r); // creamos un nodo
  *r = t;                    // actualizamos el enlace
}
```

Se puede hacer en una línea: `*r = creaNodo(x, *r);`

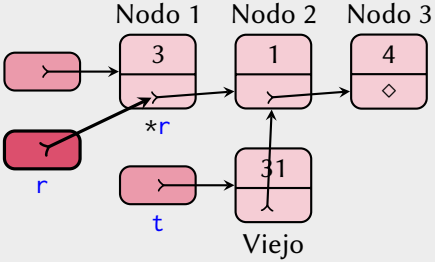
Lista enlazada

Ejemplo de eliminación de un nodo

Justo después de copiar el enlace



Justo antes de liberar el nodo



Lista enlazada

Eliminación de un nodo

Como esto implica modificar una lista enlazada s, requerimos de una referencia r al enlace que apunta al nodo que queremos eliminar. Por ejemplo, si queremos eliminar al inicio, esa referencia sería r = &s.

```
int eliminaNodo(nodo **r, int *x) {
  nodo *t = *r;   // copiamos el enlace
  if (t == NULL) // si no hay nodo
    return 0;     // no se pudo
  *x = t->a;      // copiamos el dato
  *r = t->sig;    // actualizamos el enlace
  free(t);        // liberamos el nodo
  return 1;       // si se pudo
}
```

Lista enlazada

Recorrido por valor

En una lista enlazada s no podemos procesar un nodo arbitrario a menos que tengamos una referencia al mismo. Una posibilidad que tenemos es comenzar en s y si no es nulo, comenzamos en el primer nodo y avanzamos sucesivamente hasta encontrar el nodo buscado o llegar al último nodo. Esta operación es común y se escribe de forma sencilla:

```
for (nodo *p = s; p != NULL; p = p->sig)
  procesaNodo(p); // lo que se quiera hacer
```

Observe que en cada iteración de este ciclo p apunta sucesivamente a los nodos de la lista s. Si s está vacía, el ciclo se detiene inmediatamente.

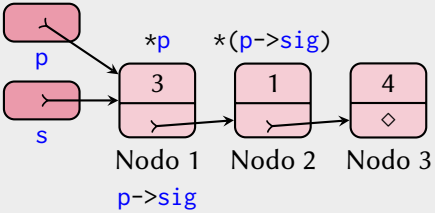
Lista enlazada

Ejemplo de recorrido de una lista enlazada por valor

Paso 1

p apunta al nodo 1

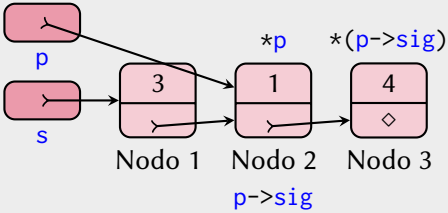
*p es el nodo 1



Paso 2

p apunta al nodo 2

*p es el nodo 2



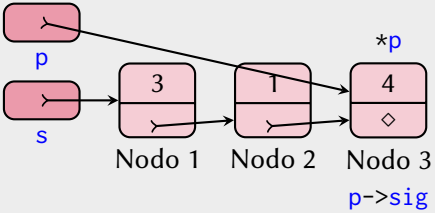
Lista enlazada

Ejemplo de recorrido de una lista enlazada por valor

Paso 3

p apunta al nodo 3

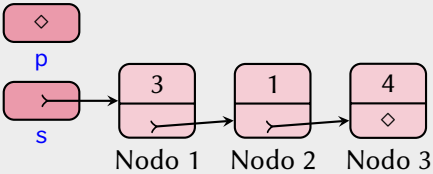
*p es el nodo 3



Paso 4

p es nulo

*p es ilegal



Lista enlazada

Longitud de una lista enlazada

Observe que el parámetro s se pasó por valor (en otras palabras: es una copia). Por eso, si lo modificamos nada le pasa a la lista original.

```
int longitudLista(lista s) {
  for (int i = 0; s != NULL; s = s->sig) // s es copia
    i++;                                 // incrementa la longitud
  return i;                              // i final es la longitud
}
```

Lista enlazada

Búsqueda lineal de un dato en una lista enlazada

Regresando un apuntador al nodo si el dato está:

```
nodo *buscaLista(lista s, int x) {
  while (s != NULL && s->a != x) // hay nodo sin x
    s = s->sig;                  // avanza
  return s;
}
```

Lista enlazada

Búsqueda lineal de un dato en una lista enlazada

Regresando la posición en la lista enlazada si el dato está:

```
int posicionLista(lista s, int x) {
  for (int i = 1; s != NULL; s = s->sig, i++)
    if (s->a == x)          // si contiene el dato
      return i;             // termina
  return 0;                 // el dato no estuvo
}
```

Lista enlazada

Recorrido por referencia

En una lista enlazada `s` que se pasó por valor no podemos modificar qué nodo es el primero (aunque sí se pueden modificar los demás). Si queremos poder modificar el primer nodo (por ejemplo, para agregar el primer nodo) entonces debemos pasar la lista enlazada por referencia. Esta operación es muy común y se escribe de esta forma:

```
for (lista *r = &s; *r != NULL; r = &((*r)->sig))
  procesaNodo(r); // lo que se quiera hacer
```

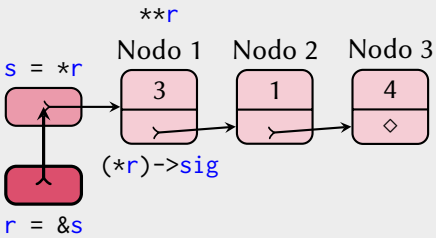
En cada iteración de este ciclo `r` apunta sucesivamente a los apuntadores a los nodos de la lista `s`. Si `s` está vacía, el ciclo se detiene inmediatamente.

Lista enlazada

Ejemplo de recorrido de una lista enlazada por referencia

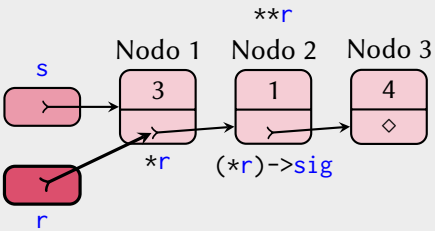
Paso 1

`r` apunta al apuntador al nodo 1
`*r` apunta al nodo 1
`**r` es el nodo 1



Paso 2

`r` apunta al apuntador al nodo 2
`*r` apunta al nodo 2
`**r` es el nodo 2

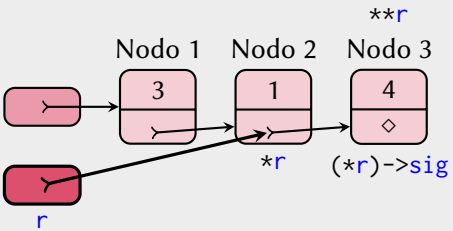


Lista enlazada

Ejemplo de recorrido de una lista enlazada por referencia

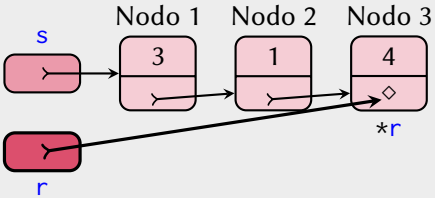
Paso 3

r apunta al apuntador al nodo 3
*r apunta al nodo 3
**r es el nodo 3



Paso 4

r apunta a un apuntador nulo
*r es nulo
**r es ilegal



Lista enlazada

Destruir una lista enlazada

Para destruir una lista enlazada debemos eliminar todos sus nodos.

```
void destruyeLista(lista *r) {
  int x;
  while (eliminaNodo(r, &x));
}
```

Recuerda que `elimina` regresa verdadero si eliminó un nodo y falso si no había nodo a eliminar. Al terminar `*r` vale `NULL`.

Lista enlazada

Copiar una lista enlazada

Para copiar una lista enlazada debemos copiar todos sus nodos.

```
void copiaLista(lista s, lista *r) {
  for (; s != NULL; s = s->sig) { // para cada nodo de s
    insertaNodo(r, s->a);         // copia el dato a r
    r = &((*r)->sig);             // avanza en lista r
  }
}
```

Lista enlazada

Ejercicios

1. Escribe una función que reciba una lista enlazada `s` y un dato `a` y regrese cuántas veces aparece `a` en `s`.
2. Escribe funciones que reciban una lista enlazada `s` y una posición `i` y digan qué dato está allí, inserten un nuevo nodo allí o eliminen el nodo que está allí. Tus funciones deben reportar si lo pedido se pudo hacer.
3. Escribe una nueva función de inserción que cuando inserte un dato mantenga los datos de los nodos ordenados crecientemente.
4. Escribe una función que parta una lista enlazada `s` en mitades delantera y trasera (la delantera puede tener un nodo más).
5. Escribe una función que mezcle dos listas enlazadas ordenadas crecientemente en una sola lista enlazada ordenada crecientemente.