Estructuras de Datos

Listas

Como ya hemos visto, un array, de 100 elementos por ejemplo, requiere un bloque de memoria contigua equivalente, aproximadamente, a:

```
100 * sizeof(dato)
```

Donde dato corresponde al tipo de dato que se desee almacenar en el arreglo.

Suponiendo que usamos malloc para solicitar nuestro bloque de memoria, el tamaño del mismo sólo se puede modificar (agrandar):

- usando realloc
- pidiendo otro bloque de memoria más grande (con malloc), copiando la información posición por posición y liberando el bloque anterior (con free).

Todo esto, es sumamente costoso. Una alternativa es usar estructuras de datos dinámicas, que nos den flexibilidad para modificar su tamaño. La más sencilla de ellas son las Listas. Comencemos con ella.

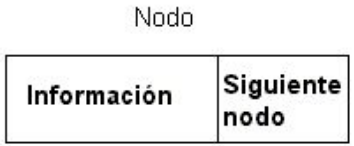
Listas

Una lista enlazada es una estructura dinámica formada por una serie de nodos, conectados entre sí a través de una referencia (puntero), en donde se almacena la información de los elementos de la lista.

Por lo tanto, los nodos de una lista enlazada se componen de dos partes principales:

- la información que queremos guardar.
- un puntero al siguiente elemento.

Se lo suele representar como:



Listas

Una definición de lista enlazada suele ser similar a esta:

```
typedef struct _SNodo {
  int dato;
  struct _SNodo *sig;
} SNodo;
```

Como podemos ver, la información a almacenar es un entero (dato) y, el puntero al siguiente elemento (sig).

Suele resultar confusa este tipo de definición así que analicemos por parte la estructura.

Vemos que la definición comienza con la palabra clave typedef. La misma permite definir un alias para un tipo de dato.
Un ejemplo sencillo de su uso sería:

```
typedef int* punteroInt;
punteroInt a = malloc(sizeof(int));
*a = 4;
```

Donde definimos a punteroInt como un puntero a un entero. Luego de esto, lo puedo usar como un tipo de dato ya que, cuando se compile, se reemplaza por el tipo de dato original.

Ahora que sabemos en qué consiste typedef, lo que estamos teniendo es que toda esta estructura termina siendo SNodo.

```
typedef struct _SNodo {
  int dato;
  struct _SNodo *sig;
} SNodo;
```

Otra opción de hacer esto sería:

```
struct _SNodo {
  int dato;
  struct _SNodo *sig;
}

typedef struct _SNodo SNodo;
```

Listas

Ya entendimos una parte de la definición, vayamos ahora por analizar el contenido de la estructura.

```
typedef struct _SNodo {
  int dato;
  struct _SNodo *sig;
} SNodo;
```

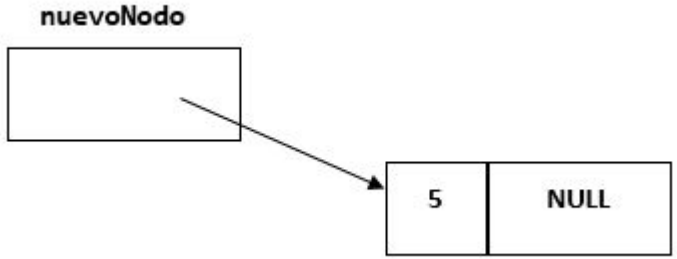
Como podemos ver, la información a almacenar es un entero (dato) y, el puntero al siguiente elemento (sig).
Suele resultar confuso que el tipo de sig es un puntero a la estructura (struct _SNodo) que estoy definiendo pero, sabemos que un puntero es solamente una variable que guarda una dirección de memoria y, que el tipo asociado es usado para que luego se sepa, en la ejecución, qué se supone que hay ahí.

Un ejemplo de su uso es:

```
SNodo* nuevoNodo = malloc(sizeof(SNodo));
nuevoNodo->dato = 5;
nuevoNodo->sig = NULL;
```

Acá estamos pidiendo memoria para un SNodo y, luego almacenando un 5 en el entero y, NULL en el puntero.

Es decir, tendríamos:



Hasta acá tenemos un nodo inicializado pero, ¿cómo hacemos para construir la lista? Tenemos dos opciones:

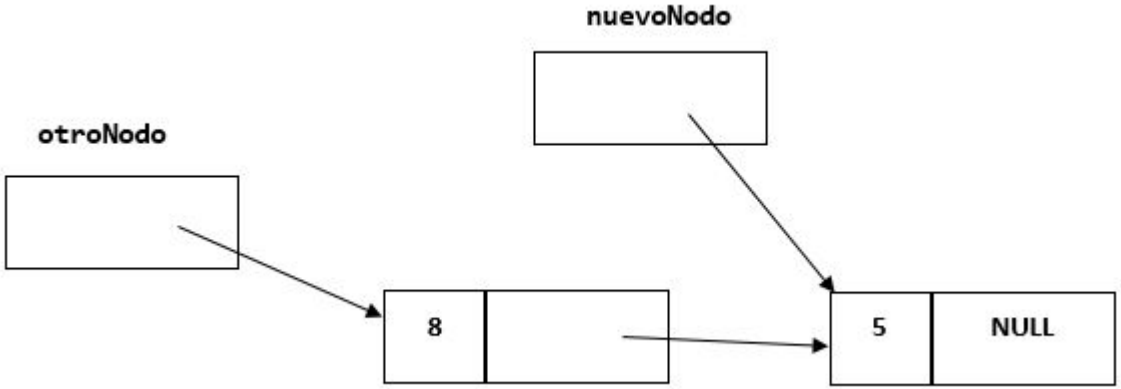
- crear un nuevo nodo inicial y, hacer que su siguiente (sig) apunte al nodo anterior
- ir guardando en el puntero sig del último nodo de la lista, la dirección del siguiente nodo

¿Qué significa crear un nuevo nodo inicial y, hacer que su siguiente apunte al nodo anterior? Vamos a verlo en código:

```
SNodo* nuevoNodo = malloc(sizeof(SNodo));
nuevoNodo->dato = 5;
nuevoNodo->sig = NULL;
SNodo* otroNodo = malloc(sizeof(SNodo));
otroNodo->dato = 8;
otroNodo->sig = nuevoNodo;
```

Es decir, creamos un nuevo nodo, y su sig apunta al anterior inicio de la lista. O sea, estamos agregando al inicio.

En un diagrama, lo que vimos en código, se vería reflejado como:

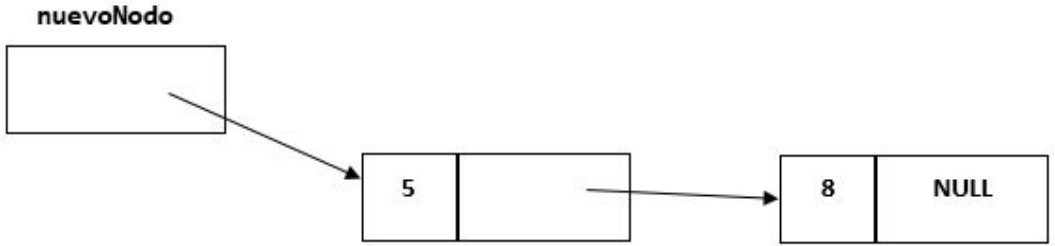


Ahora bien, tenemos que tener cuidado porque se cambia el inicio, es decir, si tenemos una variable que guarda el inicio de la lista, dicha variable debería ser actualizada luego de agregar el nuevo nodo (otroNodo).

¿Qué significa ir guardando en el puntero sig del último nodo de la lista la dirección del siguiente nodo? Vamos a verlo con un ejemplo:

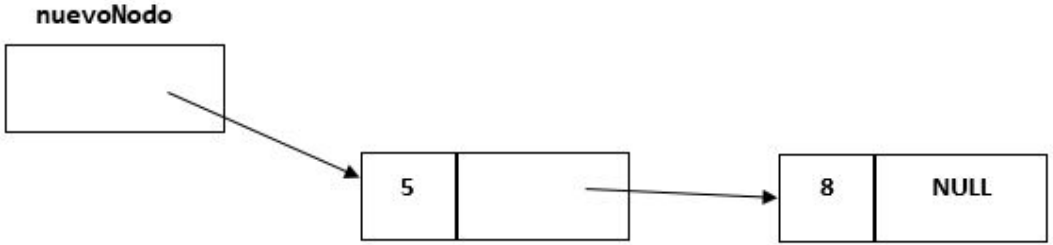
```
SNodo* nuevoNodo = malloc(sizeof(SNodo));
nuevoNodo->dato = 5;
nuevoNodo->sig = malloc(sizeof(SNodo));
nuevoNodo->sig->dato = 8;
nuevoNodo->sig->sig = NULL;
```

Es decir, en memoria tenemos:



Ahora bien, para eso tenemos que ir hasta el último nodo. Eso nos genera una pregunta, ¿cómo hacemos para determinar si un nodo es el último?
Como podemos suponer, la respuesta resulta simple, es aquel Nodo cuyo atributo sig vale NULL.

Es decir, tenemos que iterar preguntando si el valor de sig es NULL.



Analicemos el siguiente bloque de código:

```
SNodo* nuevoNodo = malloc(sizeof(SNodo));
nuevoNodo->dato = dato;
nuevoNodo->sig = NULL;

if (inicio == NULL) inicio = nuevoNodo;
else {
  SNodo* temp = inicio;
  for(;temp->sig!=NULL; temp=temp->sig);
  temp->sig =  nuevoNodo;
}
```

La primer parte crea un Nodo (nuevoNodo) y, lo inicializa con un valor y con siguiente valiendo NULL.

Este nodo va a ser el último nodo de la lista.

```
SNodo* nuevoNodo = malloc(sizeof(SNodo));
nuevoNodo->dato = dato;
nuevoNodo->sig = NULL;
```

Lo que nos resta es posicionarnos en el último actual y ponerle como siguiente a nuevoNodo.

Lo que puede pasar es que la lista esté vacía, para esto me fijo si el puntero que apunta al inicio de la lista es NULL (inicio). En este caso, simplemente el nuevo nodo creado (nuevoNodo) es el inicio de la lista (inicio).

```
if (inicio == NULL) inicio = nuevoNodo;
```

Si la lista no está vacía, usamos un puntero auxiliar para recorrer la lista (temp).

Nos vamos desplazando nodo por nodo hasta llegar al último nodo. Notemos que el for escrito no tiene sentencias, sino que sólo ejecuta el bloque de incremento hasta que la condición sea falsa.

Al salir del for, estamos posicionados en el último nodo actual de la lista, por lo que a sig le asignamos el nuevoNodo, agregándolo al final de la lista.

```
else {
  SNodo* temp = inicio;
  for(;temp->sig!=NULL; temp=temp->sig);
  temp->sig =  nuevoNodo;
}
```