Estructuras de Datos

Listas

Retomando las ideas de la presentación anterior, teníamos estas estructuras definidas:

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

typedef struct SList {
  SNodo *primero;
  SNodo *ultimo;
} SList;
```

Para continuar, vamos a crear nuevas estructuras que nos permitan trabajar con las listas pero, agregándole la posibilidad de recorrerlas en ambos sentidos.

Consideremos las siguientes definiciones:

```
typedef struct _DNodo {
  int dato;
  struct _DNodo* sig;
  struct _DNodo* ant;
} DNodo;

typedef struct {
  DNodo* primero;
  DNodo* ultimo;
} DList;
```

Podemos notar que la única diferencia sustancial (además de los nombres), es que la estructura DNodo contiene otro puntero además de sig: ant.

El objetivo de este puntero es que cada nodo apunte a su inmediato anterior para que, de esta forma, cada nodo conozca a su anterior y a su siguiente en la lista.
Usándolo vamos a poder recorrer la lista desde el último al primero.

```
typedef struct _DNodo {
  int dato;
  struct _DNodo* sig;
  struct _DNodo* ant;
} DNodo;

typedef struct {
  DNodo* primero;
  DNodo* ultimo;
} DList;
```

Pero, antes de hacer eso, tenemos que pensar: ¿cómo adaptamos la función agregar y eliminar que teníamos a esta nueva estructura?

Comencemos con la función agregar y, su versión al inicio. Esto era lo que teníamos.

```
SNodo* slist_agregar_inicio(SNodo* lista, int dato) {
  SNodo* nuevoNodo = malloc(sizeof(SNodo));
  nuevoNodo->dato = dato;
  nuevoNodo->sig = lista;
  return nuevoNodo;
}
```

Lo que nos faltaría en este caso es:

- inicializar el puntero ant a NULL. ¿Por qué NULL? Porque el primer nodo de la lista no tiene un anterior.
- en caso de que ya hubiera un nodo en la lista, el anterior del primer nodo debe ser nuevoNodo.

Con esos cambios, la función quedaría así:

```
DNodo* dlist_agregar_inicio(DNodo* lista, int dato) {
  DNodo* nuevoNodo = malloc(sizeof(DNodo));
  nuevoNodo->dato = dato;
  nuevoNodo->sig = lista;
  nuevoNodo->ant = NULL;
  if (lista!=NULL)
    lista->ant = nuevoNodo;
  return nuevoNodo;
}
```

Ahora, escribamos una versión usando la estructura DList.

Con la estructura, si agrego un nodo al inicio, debo modificar el puntero a primero y, actualizar el puntero al último si este es NULL(eso indicaría que el nodo que acabo de agregar es el primer y último nodo de la lista).

```
DList* dlist_agregar_inicio(DList* lista, int dato) {
  DNodo* nuevoNodo = malloc(sizeof(DNodo));
  nuevoNodo->dato = dato;
  nuevoNodo->sig = lista->primero;
  nuevoNodo->ant = NULL;
  if (lista->primero != NULL)
    lista->primero->ant = nuevoNodo;
  if (lista->ultimo == NULL)
    lista->ultimo = nuevoNodo;
  lista->primero = nuevoNodo;
  return lista;
}
```

Quedan pendientes las funciones para agregar al final usando y, sin usar, la estructura DList.

En el primer caso, voy a tener que iterar o hacer una recursión mientras que en el segundo es muy similar a la función que acabamos de escribir.

Vamos a escribir la versión de DNodo iterando hasta posicionarnos en el último nodo.

```
DNodo* dlist_agregar_final(DNodo* lista, int dato) {

  DNodo* nuevoNodo = malloc(sizeof(DNodo));
  nuevoNodo->dato = dato;
  nuevoNodo->sig = NULL;
  nuevoNodo->ant = NULL;

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

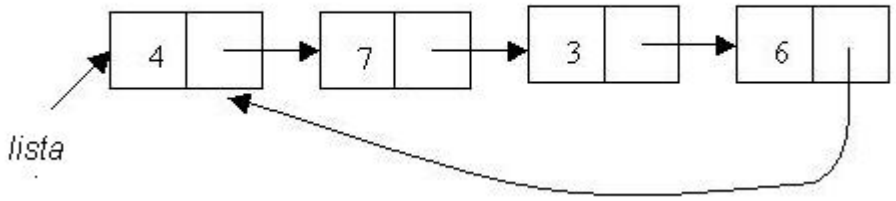
Comparen este código con el de la función slist_agregar_final.

Finalmente, vamos a agregar al final usando la estructura DList:

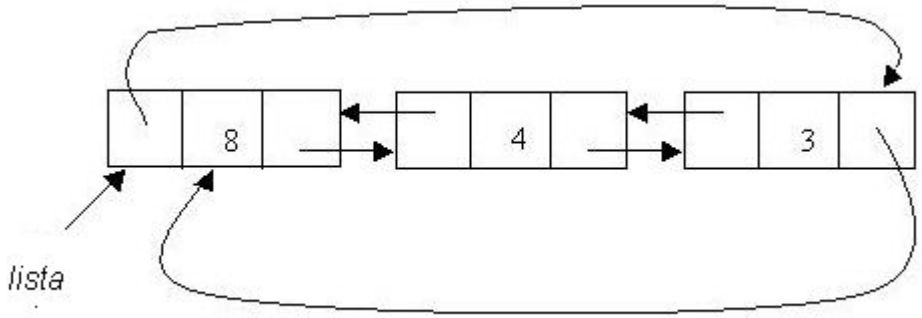
```
DList* dlist_agregar_final(DList* lista, int dato) {
  DNodo* nuevoNodo = malloc(sizeof(DNodo));
  nuevoNodo->dato = dato;
  nuevoNodo->sig = NULL;
  nuevoNodo->ant = lista->ultimo;
  if (lista->ultimo != NULL)
    lista->ultimo->sig = nuevoNodo;
  if (lista->primero == NULL)
    lista->primero = nuevoNodo;
  lista->ultimo = nuevoNodo;
  return lista;
}
```

La última variante que vale la pena mencionar es cuando el siguiente del último elemento de la lista apunta al primero y, en caso de que sea doblemente enlazada, el anterior del primero es el último de la lista; la lista con esta característica se denomina Lista Circular.

Acá podemos ver una representación gráfica de una lista circular simplemente enlazada:



Una doblemente enlazada sería:



Podemos pensar que, en este tipo de lista, no necesitamos otra estructura, además de la definición propia de Nodo, para acceder, sin iterar o usar recursividad, al último nodo de la lista.