INGENIERÍA UNIVERSIDAD DE LOS ANDES MÉRIDA VENEZUELA | DEPARTAMENTO DE COMPUTACIÓN ESCUELA DE SISTEMAS

Laboratorio de Sistemas Operativos
Semestre A-2018
Práctica de Nivelación
Poslaboratorio

Prof. Rodolfo Sumoza
Prep. Alvaro Araujo
Prep. Luis Sanchez

1 Lista Doblemente Enlazada

Es una estructura dinámica de datos que consiste en un conjunto de nodos enlazados secuencialmente, cada nodo contiene el dato y dos punteros, uno de los cuales apunta al siguiente elemento y otro que apunta al elemento anterior. Se denomina doblemente enlazada porque hay un enlace total de los nodos, además existen enlaces en dos direcciones; es decir, se puede acceder al elemento siguiente o al elemento anterior.

2 Implementación de la Lista Doblemente Enlazada.

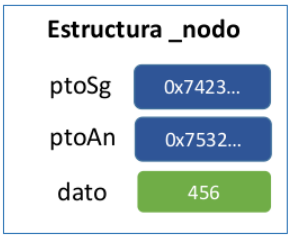
Para implementar una lista doblemente enlazada se puede definir el nodo de la siguiente manera:

```
1 typedef struct _node
2 {
3   int dato;
4   struct _node * ptoSg;
5   struct _node * ptoAn;
6 }nodo;
```

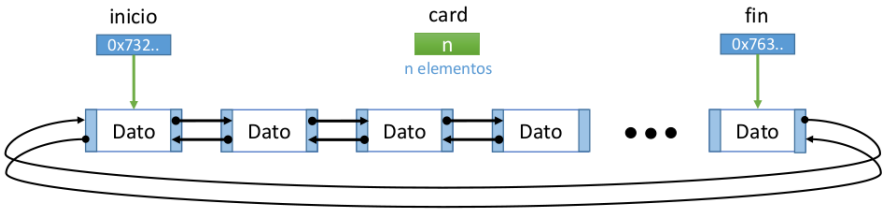
Se puede definir también la estructura de la lista de la siguiente manera:

```
typedef struct _list
{
  nodo *inicio;
  nodo *fin;
  int card;
}lista;
```

La definición de la estructura "_nodo" se puede representar graficamente de la siguiente manera:



La definición de la estructura "_lista" se puede representar graficamente de la siguiente manera:



3 Funciones a Implementar

3.1 int insertar(lista *ptolista, void * dato)

Esta función recibe como parámetros un puntero a una lista y la dirección de memoria de un dato que se desea insertar a dicha lista. Se debe tener en cuenta que esta función de inserción debe ingresar el nodo como ultimo elemento, mas específicamente se debe insertar dicho elemento como nodo siguiente al nodo apuntado por el puntero "inicio". También se debe tomar en consideración que los elementos de la lista de este ejercicio poseen claves únicas; es decir, no pueden haber dos nodos con la misma clave, por lo tanto, se deben tener en

cuenta las siguientes situaciones: si no existe suficiente memoria para insertar un nuevo nodo se debe retornar el valor -1, si ya hay un nodo que posea el dato que se desea insertar se debe retornar -2 y si la inserción se llevo a cabo de manera exitosa se debe retornar 0, modificar el puntero inicio e incrementar la cardinalidad de la lista.

3.2 int eliminar(lista *ptolista, void * dato)

Esta función recibe como parámetros un puntero a una lista y la dirección de memoria de un dato que se desea eliminar de dicha lista. Hay que tener consideración de dos situaciones, la primera es que el elemento que se quiere eliminar no existe en la lista en este caso, se debe retornar el valor -1 y la segunda es que se elimine correctamente el elemento en dicho caso, se debe retornar el valor 1.

3.3 void listar (void * ptolista)

El objetivo de la presente función es imprimir todos los elementos de la lista, el parámetro que recibe es la dirección de memoria de la lista, para la salida se debe indicar el inicio de la lista con el carácter '*' y en cada linea el dato correspondiente a un nodo, al terminar la lista se debe terminar con el carácter '*'. La salida debe tener la siguiente forma:

```
*
1
3
4
6
7
*
```

3.4 nodo * posN(int pos, lista * ptolista)

En esta función se debe prestar atención al dato que se retorna, se tiene que buscar el nodo que se encuentre en la posición "pos" de la lista y retornar un puntero a ese nodo, vale destacar que dicha posición es relativa al nodo apuntado por el puntero "inicio". Se debe tener en cuenta que desde el nodo inicio se puede ir en dos direcciones, se puede utilizar esta cualidad de las listas doblemente enlazadas para alcanzar un poco mas rápido la posición requerida.

3.5 void quicksort(void * ptolista)

Esta función debe ordenar la lista de menor a mayor por el método de ordenamiento QuickSort. El parámetro que recibe es la dirección de memoria de la lista a ordenar.

3.6 void destructor(lista * ptolista)

Esta función debe liberar toda la memoria utilizada por la lista, se tiene que liberar nodo por nodo.

4 Pruebas y Ejecución

Junto con esta practica se proporcionan 6 archivos, a continuación se describen:

1. "listadoble.h", en este archivo se deben implementar cada una de las funciones aquí tratadas, ya trae consigo sus definiciones.
2. "numerosEnteros.c", este archivo contiene un código fuente para probar las funciones a implementar.
3. "numeros100.txt", este archivo contiene 100 números aleatorios para realizar una prueba al programa, la salida que se debería obtener para estos 100 números se encuentra en el archivo "r_100.txt".
4. "numeros10000.txt", este archivo contiene 10000 números aleatorios para realizar una prueba al programa, la salida que se debería obtener para estos 10000 números se encuentra en el archivo "r_10000.txt".

Para ejecutar el programa de prueba se debe enviar como parámetro la cantidad de números de entrada, y enviar como flujo de entrada alguno de los dos archivos de prueba, para probar con el archivo de 100 números, se debe ejecutar de la siguiente manera:

```
usuario@pc:/$ ./numerosEnteros.out 100 < numeros100.txt
```