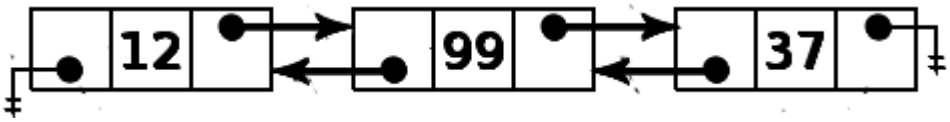
Asignatura: Algoritmia y Estructura de Datos
Profesor: D. Sc. Gerardo García Gil
2021-B
Alumno: José Rafael Ruiz Gudiño
Ingeniería en Desarrollo de Software
Centro de Enseñanza Técnica Industrial (CETI)

-Presentación

En términos muy generales, el área de estudio de estructuras de datos como disciplina de las ciencias computacionales (CS) es el almacenamiento y la manipulación de la información. Las estructuras de datos, siendo una disciplina dentro de la categoría de la computación teórica, estudian de manera más abstracta la relación entre una colección de datos y las operaciones que pueden ser aplicadas a dicha colección. A continuación, se hablará acerca de una de las estructuras de datos en el manejo de colecciones: las listas doblemente enlazadas.

-Introducción

En ciencias de la computación, una lista doblemente enlazada es una estructura de datos que consiste en un conjunto de nodos enlazados secuencialmente. Cada nodo contiene tres campos, dos para los llamados enlaces, que son referencias al nodo siguiente y al anterior en la secuencia de nodos, y otro más para el almacenamiento de la información (en este caso un entero). El enlace al nodo anterior del primer nodo y el enlace al nodo siguiente del último nodo, apuntan a un tipo de nodo que marca el final de la lista, normalmente un nodo centinela o puntero null, para facilitar el recorrido de la lista. Si existe un único nodo centinela, entonces la lista es circular a través del nodo centinela.



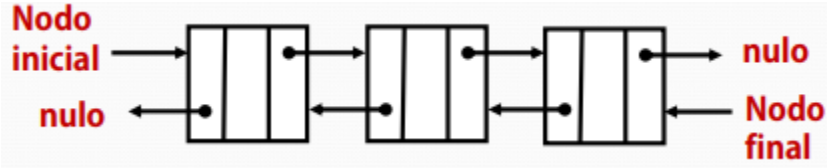
-Desarrollo

El doble enlace de los nodos permite recorrer la lista en cualquier dirección. Mientras que agregar o eliminar un nodo en una lista doblemente enlazada requiere cambiar más enlaces que en estas mismas operaciones en una lista enlazada simple, las operaciones son más simples porque no hay necesidad de mantener guardado el nodo anterior durante el recorrido, ni necesidad de recorrer la lista para hallar el nodo anterior, la referencia al nodo que se quiere eliminar o insertar es lo único necesario.

Representación gráfica

La lista doble tiene un apuntador inicial y un apuntador final

El primero y el último nodo de la lista apuntan a nulo



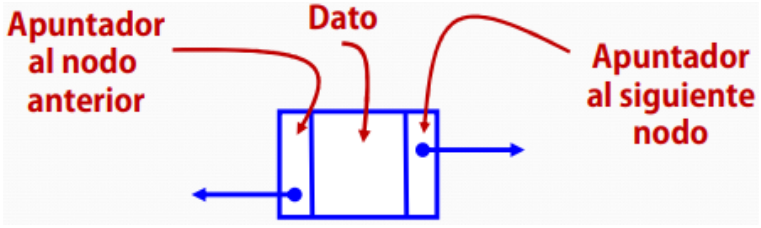
Almacenamiento de los datos en una lista doble

- Ordenados:
  - Se recorren lógicamente los nodos de la lista doble.
  - Se ubica la posición definitiva de un nuevo nodo.
  - Se mantiene el orden lógico de los datos.
- Desordenados:
  - El nuevo dato se agrega al final de la lista doble.

Arquitectura de un nodo

Cada nodo tiene 3 secciones:

1. Dato (puede ser simple o compuesto).
2. Apuntador o referencia que enlaza al siguiente nodo en secuencia lógica.
3. Apuntador que enlaza al nodo anterior.



-Implementación

Una lista doblemente enlazada es una lista lineal en la que cada nodo tiene dos enlaces, uno al nodo siguiente, y otro al anterior.

Las listas doblemente enlazadas no necesitan un nodo especial para acceder a ellas, pueden recorrerse en ambos sentidos a partir de cualquier nodo, esto es porque a partir de cualquier nodo, siempre es posible alcanzar cualquier nodo de la lista, hasta que se llega a uno de los extremos.

El nodo típico es el mismo que para construir las listas que hemos visto, salvo que tienen otro puntero al nodo anterior:

```
struct nodo {
   int dato;
   struct nodo *siguiente;
   struct nodo *anterior;
};
```

Operaciones básicas con listas doblemente enlazadas

De nuevo tenemos el mismo repertorio de operaciones sobre este tipo listas:

- Añadir o insertar elementos.
- Buscar o localizar elementos.
- Borrar elementos.
- Moverse a través de la lista, siguiente y anterior.

Código de lista doblemente enlazada en C

```
#include <stdio.h>
#define ASCENDENTE 1
#define DESCENDENTE 0
typedef struct _nodo {
   int valor;
   struct _nodo *siguiente;
   struct _nodo *anterior;
} tipoNodo;
typedef tipoNodo *pNodo;
typedef tipoNodo *Lista;
/* Funciones con listas: */
void Insertar(Lista *l, int v);
void Borrar(Lista *l, int v);
void Buscar(Lista l, int v);
void BorrarLista(Lista *);
void MostrarLista(Lista l, int orden);
int main() {
   Lista lista = NULL;
   pNodo p;
   Insertar(&lista, 7);
   Insertar(&lista, 4);
   Insertar(&lista, 8);
   Insertar(&lista, 1);
   Insertar(&lista, 9);
   Insertar(&lista, 6);
   MostrarLista(lista, ASCENDENTE);
   MostrarLista(lista, DESCENDENTE);
   Borrar(&lista, 8);
   Borrar(&lista, 1);
   Borrar(&lista, 3);
```

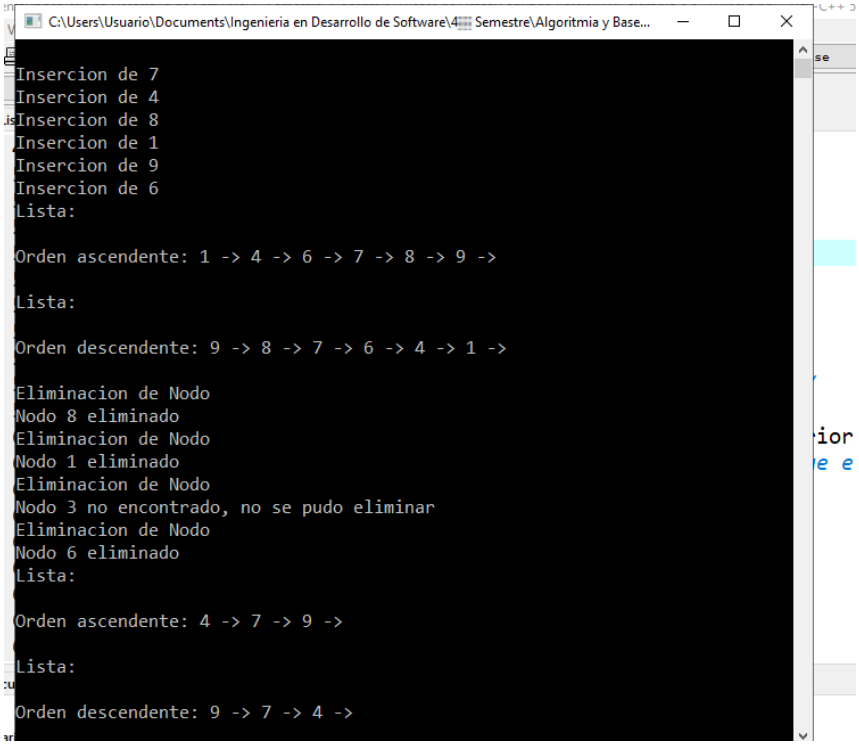
```
  Borrar(&lista, 6);
  MostrarLista(lista, ASCENDENTE);
  MostrarLista(lista, DESCENDENTE);
  Buscar(lista,7);
  Buscar(lista,8);
  BorrarLista(&lista);
  MostrarLista(lista,ASCENDENTE);
  return 0;
}
void Insertar(Lista *lista, int v) {
  pNodo nuevo, actual;
  printf("\nInsercion de %d",v);
  /* Crear un nodo nuevo */
  nuevo = (pNodo)malloc(sizeof(tipoNodo));
  nuevo->valor = v;
  /* Colocamos actual en la primera posición de la lista */
  actual = *lista;
  if(actual) while(actual->anterior) actual = actual->anterior;
  /* Si la lista está vacía o el primer miembro es mayor que el nuevo */
  if(!actual || actual->valor > v) {
    /* Añadimos la lista a continuación del nuevo nodo */
    nuevo->siguiente = actual;
    nuevo->anterior = NULL;
    if(actual) actual->anterior = nuevo;
    if(!*lista) *lista = nuevo;
  }
  else {
    /* Avanzamos hasta el último elemento o hasta que el siguiente tenga
       un valor mayor que v */
    while(actual->siguiente &&actual->siguiente->valor <= v)
      actual = actual->siguiente;
    /* Insertamos el nuevo nodo después del nodo anterior */
    nuevo->siguiente = actual->siguiente;
    actual->siguiente = nuevo;
    nuevo->anterior = actual;
    if(nuevo->siguiente) nuevo->siguiente->anterior = nuevo;
  }
}
void Buscar(Lista lista,int v){
          printf("\nBusqueda");
          pNodo nodo = lista;
  while(nodo && nodo->valor < v) nodo = nodo->siguiente;
  while(nodo && nodo->valor > v) nodo = nodo->anterior;

  /* El valor v no está en la lista */
  if(!nodo || nodo->valor != v){
   printf("\nNodo %d no encontrado, no se pudo eliminar",v);
  }else printf("\nNodo %d encontrado",v);
}
void Borrar(Lista *lista, int v) {
  pNodo nodo;
  printf("\nEliminacion de Nodo");
  /* Buscar el nodo de valor v */
  nodo = *lista;
  if(nodo){
```

```
  while(nodo && nodo->valor < v) nodo = nodo->siguiente;
  while(nodo && nodo->valor > v) nodo = nodo->anterior;
  /* El valor v no está en la lista */
  if(!nodo || nodo->valor != v){
   printf("\nNodo %d no encontrado, no se pudo eliminar",v);
         return;
  }
  /* Borrar el nodo */
  /* Si lista apunta al nodo que queremos borrar, apuntar a otro */
  if(nodo == *lista)
    if(nodo->anterior) *lista = nodo->anterior;
    else *lista = nodo->siguiente; // si nodo es igual al primer elemento de la lista
  if(nodo->anterior) /* no es el primer elemento */
     nodo->anterior->siguiente = nodo->siguiente;
  if(nodo->siguiente) /* no es el último nodo */
     nodo->siguiente->anterior = nodo->anterior;
  free(nodo);
  printf("\nNodo %d eliminado",v);
  } else printf("\nLista vacia ");
}
void BorrarLista(Lista *lista) {
  pNodo nodo, actual;
  printf("\nEliminacion de Lista");
  actual = *lista;
  while(actual->anterior) actual = actual->anterior;
  while(actual) {
     nodo = actual;
     actual = actual->siguiente;
     free(nodo);
  }
  *lista = NULL;
  printf("\nLista borrada\n");
}
void MostrarLista(Lista lista, int orden) {
  pNodo nodo = lista;
   printf("\nLista:\n");
  if(!lista) printf("\nLista vacia");
  nodo = lista;
  if(orden == ASCENDENTE) {
     while(nodo->anterior) nodo = nodo->anterior;
     printf("\nOrden ascendente: ");
     while(nodo) {
        printf("%d -> ", nodo->valor);
        nodo = nodo->siguiente;
     }
  }
  else {
     while(nodo->siguiente) nodo = nodo->siguiente;
     printf("\nOrden descendente: ");
     while(nodo) {
        printf("%d -> ", nodo->valor);
        nodo = nodo->anterior;
     }
  }
  printf("\n");
```

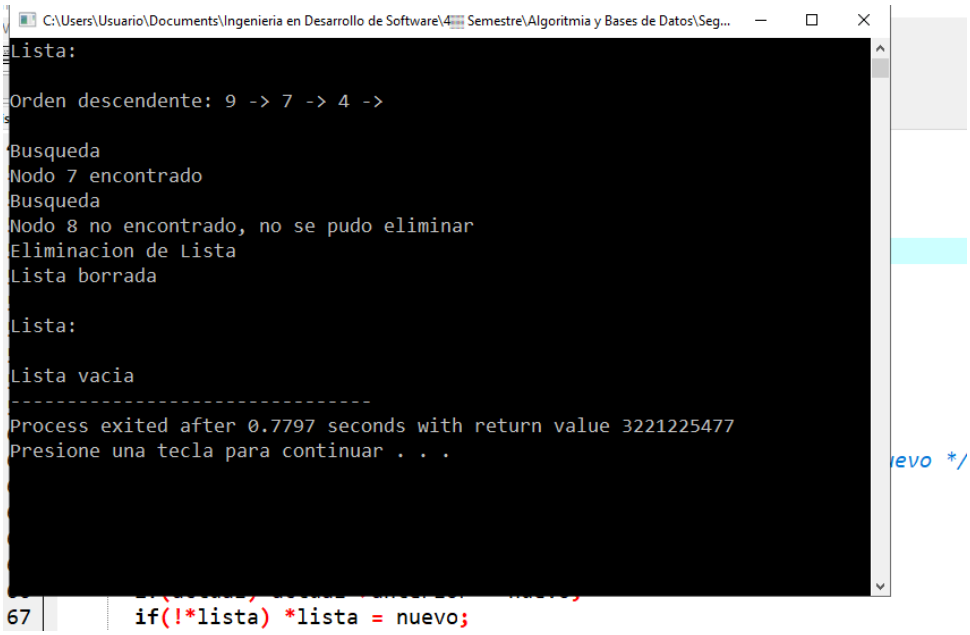
}

-Resultados

Aquí se realiza la insercion, se despliegan los datos de forma ascendente y descendente, ademas de que se eliminan 3 nodos.



Aquí se realiza un busqueda y se elimina la lista por completo



-Conclusión

En esta practica las cosas se complican más con respecto a las anteriores, esto debido a que tenemos otro campo de tipo puntero en nuestra estructura nodo que es nodo anterior, esto porque

en este tipo de lista cada nodo apunta hacia el nodo anterior y siguiente, ademas de que cada extremo de la lista apunta a null. Aquí los datos se pueden representar de dos formas dependiendo desde que punto de la lista lo vamos a desplegar. Fue algo complicado pero al estarlo haciendo lo pude comprender.

-Referencias

B, L. T. (s. f.). *Listas Dobles Enlazadas*. Instituto Tecnológico Nuevo Laredo. Recuperado 25 de octubre de 2021, de http://www.itnuevolaredo.edu.mx/takeyas/apuntes/estructura%20de%20datos/Apuntes/03-ListasDobles.pdf

Con Clase. (s. f.). *C Con Clase | Estructuras de datos (cap5)*. Recuperado 24 de octubre de 2021, de http://conclase.net/c/edd/cap5

Universidad de Granada. (s. f.). *LISTAS DOBLEMENTE-ENLAZADAS*. Recuperado 24 de octubre de 2021, de https://ccia.ugr.es/%7Ejfv/ed1/tedi/cdrom/docs/ldoble.html