



Asignatura: Teoría de Autómatas

Profesor: D. Sc. Gerardo García Gil

2021-B

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 fundamentales en el manejo de colecciones: las listas abiertas.

-Introducción

En ciencias de la computación, una lista enlazada es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.

Una lista enlazada es un tipo de dato autorreferenciado porque contienen un puntero o enlace (en inglés link, del mismo significado) a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

Las listas enlazadas pueden ser implementadas en muchos lenguajes. Lenguajes tales como Lisp, Scheme y Haskell tienen estructuras de datos ya construidas, junto con operaciones para acceder a las listas enlazadas. Lenguajes imperativos u orientados a objetos tales como C o C++ y Java, respectivamente, disponen de referencias para crear listas enlazadas.

-Desarrollo

Una lista es una estructura dinámica de datos que contiene una colección de elementos homogéneos (del mismo tipo) de manera que se establece entre ellos un orden. Es decir, cada elemento, menos el primero, tiene un predecesor, y cada elemento, menos el último, tiene un sucesor.

Tanto las estructuras vistas en la sección anterior (pilas) como las que veremos en la siguiente (colas) son tipos de listas.

Podemos distinguir, atendiendo a la organización de los nodos, entre:

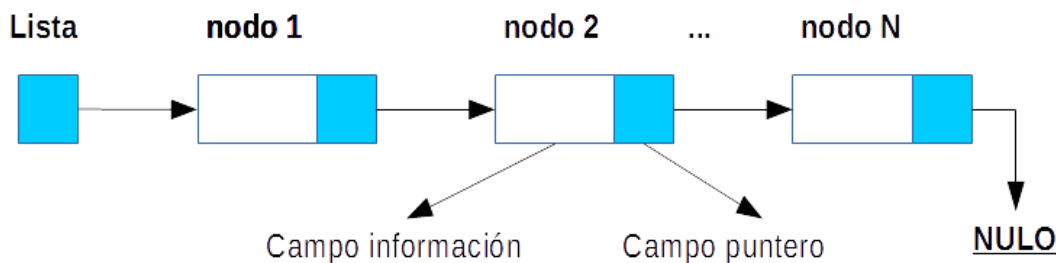
- Listas simplemente enlazadas: cada nodo tiene un campo que apunta al siguiente nodo.
- Listas doblemente enlazadas: cada nodo dispone de un puntero que apunta al siguiente nodo, y otro que apunta al nodo anterior.

Otra distinción puede ser:

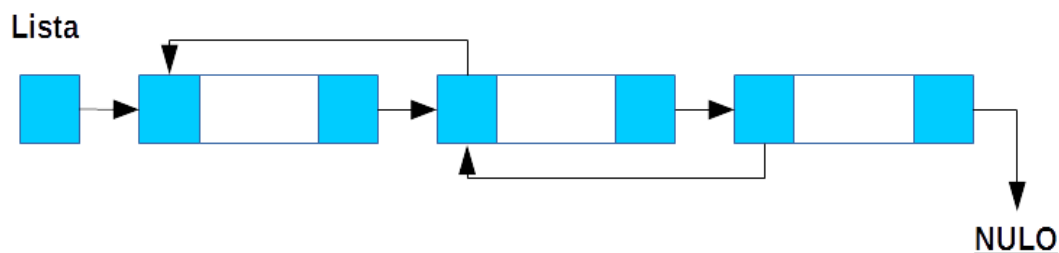
- Listas lineales: son listas que tienen un comienzo y un final.
- Listas circulares: en estas listas el último elemento apunta al primero, por lo tanto podríamos estar recorriéndolas siempre, ya que no tienen final.

Representación gráfica

Lista simplemente enlazada

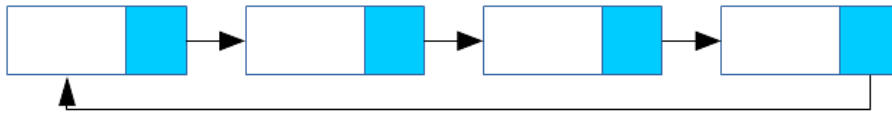


Lista doblemente enlazada



Lista circular

Cabecera



Operaciones básicas sobre una lista

Las operaciones básicas que podemos realizar en cualquier lista, independiente del tipo que sea, son la siguientes:

- **Crear:** con esta operación se genera todo lo necesario para trabajar con una lista.
- **Insertar:** permite añadir un elemento a la lista. En este caso debemos indicar al programa si vamos a añadir el elemento nuevo al comienzo de la lista o al final de la misma.
- **Eliminar:** se usará para borrar un elemento de la lista. También podremos indicar si queremos borrar el primero o el último.
- **Vacía:** devolverá cierto si la lista está vacía.

Se podrán realizar otras operaciones como destruir la lista completamente, contar el número de elementos de la misma, añadir un elemento en una determinada posición, borrar un elemento de una determinada posición, comprobar si hay un dato en la lista, etc.

-Implementación

Para implementar una lista simple enlazada, se define una clase Nodo, la cual indica el tipo de los objetos guardados en los nodos de la lista. Para este caso se asume que los elementos son String (puede ser cualquier tipo). También es posible definir nodos que puedan guardar tipos arbitrarios de elementos.

Dada la clase Nodo, se puede definir una clase, ListaSimple, definiendo la lista enlazada actual. Esta clase guarda una referencia al nodo cabeza y una variable va contando el número total de nodos.

Ejemplo de lista abierta con clases en C++

```
#include<iostream>
#include<string>
using namespace std;
class Nodo {
public: //Creacion de la clase Nodo
    int dato;
    Nodo *siguiente;
};
class Lista {
```

```

        private:
            Nodo *lista; // declaracion de la lista de tipo Nodo
    public:
        Lista();//Constructor
        ~Lista();//Destructor
        void InsertarNodo(int n){
            Nodo *nuevo_nodo = new Nodo();
            nuevo_nodo->dato = n;
            Nodo *aux1 = lista;
            Nodo *aux2;
            while(aux1!=NULL&&aux1->dato<n){
                aux2 = aux1;
                aux1 = aux1->siguiente;
            }
            if(lista==aux1)
                lista = nuevo_nodo;
            else
                aux2->siguiente = nuevo_nodo;

            nuevo_nodo->siguiente = aux1;
        }
        void BorrarNodo(int);
        void BorrarLista();
        void MostrarLista(){
            Nodo *actual = new Nodo;
            actual = lista;
            while(actual!=NULL){
                cout<<actual->dato<<"->";
                actual = actual->siguiente;
            } if(actual==NULL) cout<<"NULL";
            cout<<endl;
        }
};

Lista::Lista(){

    lista = NULL;
}

Lista::~Lista(){

};

void Lista::BorrarNodo(int n){
    if(lista!=NULL){
        Nodo *aux_borrar;
        Nodo *anterior=NULL;

        aux_borrar = lista;
        while(aux_borrar!=NULL&&aux_borrar->dato!=n){
            anterior = aux_borrar;
            aux_borrar = aux_borrar->siguiente;
        }
        if(aux_borrar==NULL) cout<<"Elemento NO encontrado"<<endl;
        else if(anterior==NULL){
            lista = lista->siguiente;
            delete aux_borrar;
        }
    }
}

```

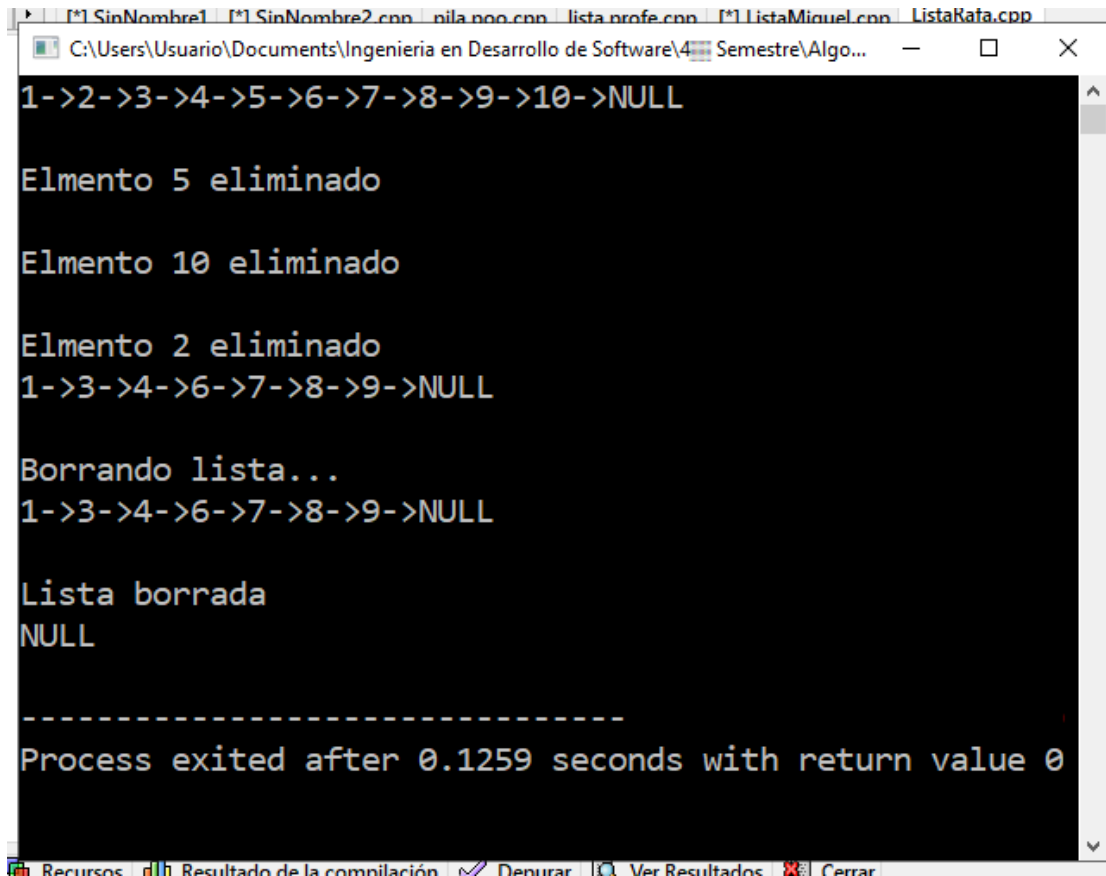
```

        cout<<"\nElmento "<<n<<" eliminado"<<endl;
    }
    else{
        anterior->siguiente = aux_borrar->siguiente;
        delete aux_borrar;
        cout<<"\nElmento "<<n<<" eliminado"<<endl;
    }

}
}
void Lista::BorrarLista(){
    cout<<"\nBorrando lista..."<<endl;
    int n;
    Nodo *aux = new Nodo();
    while(lista!=NULL){
        aux = lista;
        n = aux->dato;
        lista = lista->siguiente;
        cout<<n<<"->";
        delete aux;
    }cout<<"NULL"<<endl;
    cout<<"\nLista borrada"<<endl;
}
int main(){
    Lista lis;
    lis.InsertarNodo(1);
    lis.InsertarNodo(5);
    lis.InsertarNodo(8);
    lis.InsertarNodo(2);
    lis.InsertarNodo(7);
    lis.InsertarNodo(4);
    lis.InsertarNodo(10);
    lis.InsertarNodo(3);
    lis.InsertarNodo(6);
    lis.InsertarNodo(9);
    lis.MostrarLista();
    lis.BorrarNodo(5);
    lis.BorrarNodo(10);
    lis.BorrarNodo(2);
    lis.MostrarLista();
    lis.BorrarLista();
    lis.MostrarLista();
    return 0;
}

```

-Resultados

A screenshot of a C++ program execution window. The window title bar shows several files: f*1 SinNombre1, f*1 SinNombre2.cpp, nila.ppp.cpp, lista.profe.cpp, f*1 ListaMiguel.cpp, and ListaRafa.cpp. The file path is C:\Users\Usuario\Documents\Ingeniería en Desarrollo de Software\4 Semestre\Algo... The program output is as follows:

```
1->2->3->4->5->6->7->8->9->10->NULL  
  
Elmento 5 eliminado  
  
Elmento 10 eliminado  
  
Elmento 2 eliminado  
1->3->4->6->7->8->9->NULL  
  
Borrando lista...  
1->3->4->6->7->8->9->NULL  
  
Lista borrada  
NULL  
  
-----  
Process exited after 0.1259 seconds with return value 0
```

The bottom status bar shows icons for Recursos, Resultado de la compilación, Depurar, Ver Resultados, and Cerrar.

Como podemos observar primero se ingresa 10 elemento de tipo entero a la lista y con el mismo método de insertar se ordenan los elementos de la lista de manera ascendente, después se mandan a eliminar 3 elementos de la lista, primero se checa si se encuentran dentro de la lista y después los elimina satisfactoriamente. Al final se manda a llamar el método de borrar lista por lo que se puede observar que al llamar el método mostrar lista sólo muestra null debido a que eliminamos todos los elementos de la lista.

-Conclusión

Este tema de lista me parecio ya un poco mas facil de comprender debido a que ya estoy mas familiarizado con el tema de TDAs y estrucutras de datos de este tipo por el conocimiento previo que he desarrollado previamente con las practicas anteriores. Pero aún así, fue un poco desafiante ya que lo tuve que implementar en lo que seria programacion orientada a objetos. Me gustó la práctica y al mismo tiempo pude aprender mas sobre las listas, aunque no todo ya que hay varios tipos y este es apenas el comienzo.

-Referencias

Gómez, F. J. (s. f.). 3.3. Listas / Programación avanzada: Estructuras de datos y funciones. IEDA. Recuperado 11 de octubre de 2021, de http://agrega.juntadeandalucia.es/repositorio/02122016/a5/es-an_2016120212_9131705/33_listas.html#:~:text=Una%20lista%20es%20una%20estructura,el%20%C3%BAltimo%20tiene%20un%20sucesor.

Núñez Madrid, O. A. (s. f.). *1. implementacion - UBO - Estructuras de Datos*. UBO - Estructuras de Datos. Recuperado 11 de octubre de 2021, de <https://sites.google.com/site/edatosubo/4-estructura-de-datos-dinamicas/1-listas/1-lista-enlazada-simple-o-lista-simplemente-enlazada/1-implementacin>