Algoritmos y Estructura de datos

Clase 08

Listas doblemente enlazadas y listas circulares

Profesor: Carlos Diaz

Contenido

- Insertar al final de la lista doblemente enlazada
- Codificación de insertarFinLista
- Implementado insertarAntes()
- Implementado modificar()
- La clase NodoCircular para una lista circular simplemente enlazada
- La clase ListaCircular
- La función crearLista()
- La función insertarAcceso()
- Método visualizar()
- Ejercicios

Insertar al final de la lista

- Para facilitar, podemos añadir a la clase ListaDoble el nodo cola que marca el final de la lista: NodoDoble* cola;
- El constructor inicializa cabeza=cola=NULL, indicando que al principio la lista está vacía: cabeza = cola=NULL;
- La función crearLista() construye iterativamente insertando por la cabeza.
- La función visualizar() recorre cada nodo mostrando su dirección, dato y dirección anterior y posterior a donde apunta.
- Las demás funciones básicas se declaran similarmente.

```
//La clase ListaDoble
class ListaDoble
{
    private:
    NodoDoble* cabeza;
    NodoDoble* cola;
    public:
    ListaDoble() //Constructor
    {
        cabeza = cola=NULL;
    }
    void crearLista();
    void visualizar();
    void insertarCabezaLista(int);
    void insertarDespues(int, int);
    void eliminar(int);
    //Continua las demás funciones
};
```

Modificando insertarCabezaLista()

- Al ingresar cualquier dato colocamos delante del nuevo la cabeza, al principio cabeza es NULL.
- La primera vez que ingresamos un dato, la cabeza es NULL, entonces el nuevo dado será la cabeza y la cola. Ese dato será la cola: cola=nuevo;
- Cuando se ingresa el segundo dato, cabeza ya no es NULL, y se coloca detrás del nuevo dato.
- Finalmente cabeza es el dato nuevo.

```
void ListaDoble::insertarCabezaLista(int dato)
{
    NodoDoble* nuevo;
    nuevo = new NodoDoble (dato);
    nuevo -> ponerAdelante(cabeza);
    if (cabeza != NULL )
        cabeza -> ponerAtras(nuevo);
    else
        cola=nuevo;
    cabeza = nuevo;
}
```

Modificando insertarDespues()

- Se debe indicar que si el nuevo dato se inserto al final de la lista, este debe ser la nueva cola.

```
void ListaDoble::insertaDespues(int datoAnterior, int dato)
{
	NodoDoble* nuevo;
	NodoDoble* anterior;
	nuevo = new NodoDoble(dato);
	// Bucle de búsqueda del anterior
	NodoDoble* indice;
	indice = cabeza;
	while (indice != NULL)
	{
		if encontrado = (indice -> datoNodo() == datoAnterior);
			break;
		if (!encontrado)
			indice = indice -> adelanteNodo();
	}
	//Inserta despues
	if(indice != NULL)
	{
		anterior=indice;
		nuevo -> ponerAdelante(anterior -> adelanteNodo());
		if (anterior -> adelanteNodo() != NULL)
			anterior -> adelanteNodo() -> ponerAtras(nuevo);
		else
			cola=nuevo;
		anterior-> ponerAdelante(nuevo);
		nuevo -> ponerAtras(anterior);
	}
}
```

Codificación de insertarFinLista()

- El dato nuevo apunta hacia adelante a NULL (por creación).
- El dato nuevo apunta hacia a atrás a la cola.
- La cola apunta hacia adelante al dato nuevo.
- Solo en el caso que la lista este vacía la cabeza=cola=nuevo.
- Finalmente el dato nuevo es la cola.

```
void ListaDoble::insertarFinLista(int dato)
{
    NodoDoble* nuevo;
    nuevo = new NodoDoble (dato);

    nuevo -> ponerAtras(cola);

    if (cola != NULL )
        cola -> ponerAdelante(nuevo);
    else
        cabeza=cola=nuevo;
    cola = nuevo;
}
```

Implementando insertarAntes()

- La clase NodoDoble define para cada objeto nodo dos enlaces, uno adelante y uno atrás.
- Cada nodo creado apunta al inicio a NULL por ambos lados.
- El método datoNodo() devuelve el valor contenido en el nodo.
- Los métodos adelanteNodo() y atrasNodo() devuelven los nodos de adelante y atrás respectivamente.
- Los métodos ponerAdelante() y ponerAtras() colocan un nodo antes o después respectivamente.

```
void ListaDoble::insertarAntes(int datoPosterior, int dato)
{
	NodoDoble* nuevo;
	NodoDoble* posterior;
	nuevo = new NodoDoble(dato);
	// Bucle de búsqueda del posterior
	NodoDoble* indice;
	indice = cabeza;
	while (indice != NULL)
	{
		if(indice -> datoNodo() == datoPosterior)
			break;
		else
			indice = indice -> adelanteNodo();
	}
	//Inserta antes
	if(indice != NULL)
	{
		posterior=indice;
		nuevo -> ponerAdelante(posterior);
		if (posterior -> atrasNodo() != NULL)
			posterior -> atrasNodo() -> ponerAdelante(nuevo);
		else
			cabeza=nuevo;
		nuevo -> ponerAtras(posterior -> atrasNodo());
		posterior-> ponerAtras(nuevo);
	}
}
```

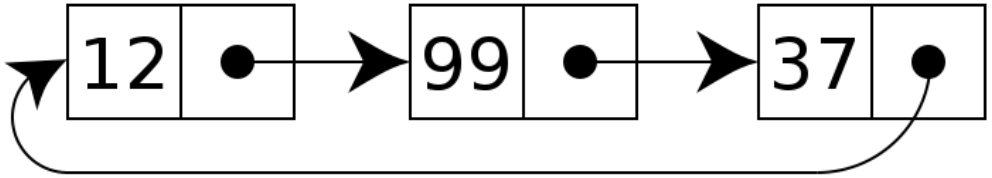
Implementando modificar()

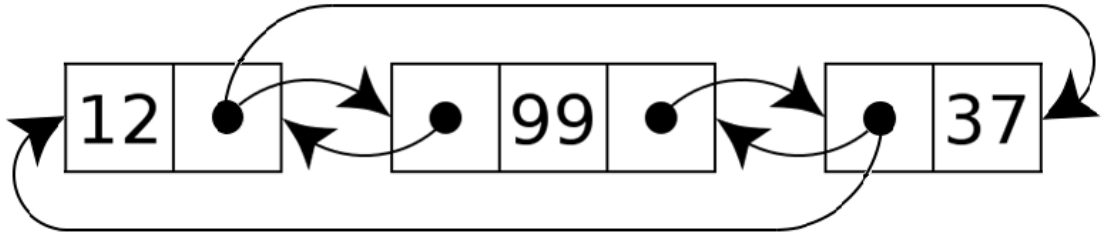
- Como vamos a cambiar un dato, utilizaremos el método fijarDato() de la clase NodoDoble.
- La función modificar() ubica el dato que será reemplazado (datoAntiguo) y luego la función fijarDato() lo reemplaza por el nuevo dato.

```
void ListaDoble::modificar(int datoAntiguo, int dato)
{
    // Bucle de búsqueda del antiguo
    NodoDoble* indice;
    indice = cabeza;
    while (indice != NULL)
    {
        if (indice -> datoNodo() == datoAntiguo)
            break;
        else
            indice = indice -> adelanteNodo();
    }
    //Reemplaza el dato
    if(indice != NULL)
    {
        indice->fijarDato(dato);
    }
}
```

Listas circulares

- En las listas lineales simples o en las dobles siempre hay un primer nodo (*cabeza*) y un último nodo (*cola*). Una lista circular, por propia naturaleza, no tiene ni principio ni fin.
- Sin embargo, resulta útil establecer un nodo de acceso (entrada) a la lista y, a partir de él, al resto de sus nodos, esta entrada puede ser la cabeza o la cola.





- Las operaciones que se realizan sobre una lista circular son similares a las operaciones sobre listas lineales, teniendo en cuenta que no hay primero ni último nodo, aunque sí un nodo de acceso.

La clase NodoCircular

- La construcción de una lista circular se puede hacer con enlace simple o enlace doble entre sus nodos.
- A continuación se implementa utilizando un enlace simple.
- El NodoCircular varía respecto al de las listas no circulares, el campo enlace, en vez de inicializarse a NULL, se inicializa para que apunte a sí mismo, de tal forma que es una lista circular de un solo nodo.
- La funcionalidad (la interfaz) de la clase NodoCircular es la misma que la de un Nodo de una lista enlazada.

```
//La clase NodoCircular
class NodoCircular
{
	private:
	int dato;
	NodoCircular* enlace; // puntero al siguiente Nodo
	public:
	NodoCircular (int t)
	{
		dato = t;
		enlace = this; // al principio se apunta a sí mismo
	}
	int datoNodo() // devuelve el dato
	{
		return dato;
	}
	void fijarDato(int a) // fija el dato
	{
		dato=a;
	}
	NodoCircular* enlaceNodo() // devuelve nodo al que apunta
	{
		return enlace;
	}
	void ponerEnlace(NodoCircular* sgte)
	{
		enlace = sgte; // enlaza con el nodo sgte
	}
};
```

La clase ListaCircular

- La clase ListaCircular dispone del puntero de acceso a la lista, junto a las funciones que implementan las operaciones.

```
//La clase ListaCircular
class ListaCircular
{
    private:
    NodoCircular* acceso; // Por donde se accede a la lista
    public:
    ListaCircular() // Constructor
    {
        acceso=NULL; // Al principio la lista esta vacia
    }
    void crearLista();
    void visualizar();
    void insertarAcceso(int);
    void insertarDespues(int,int);
    void insertarAntes(int,int);
    void eliminar(int);
    void modificar(int,int);
};
```

La función crearLista()

- El algoritmo empleado para insertar un elemento en una lista circular varía dependiendo de la posición en que se desea insertar.
- La implementación realizada considera que acceso tiene la dirección del último nodo, e inserta un nodo en la posición anterior a acceso, esto lo realizará la función insertarAcceso().

```
//Crear una lista circular
void ListaCircular::crearLista()
{
    int x;
    cout << "Termina con -1" << endl;
    do
    {
        cin >> x;
        if (x != -1)
        {
            insertarAcceso(x);
        }
    }while (x != -1);
};
```

La función insertarAcceso()

- La primera vez que ingresamos un dato, el acceso es NULL, entonces el nuevo dado será el acceso: acceso=nuevo;
- Cuando se ingresa el segundo dato, acceso ya no es NULL, el dato nuevo apunta a lo que apuntaba acceso, y acceso apunta a nuevo, cerrándose la lista circular.
- Finalmente acceso es el dato nuevo.

```
//Inserta un dato
void ListaCircular::insertarAcceso(int dato)
{
    NodoCircular* nuevo;
    nuevo = new NodoCircular (dato);
    if (acceso != NULL )
    {
        nuevo -> ponerEnlace(acceso -> enlaceNodo());
        acceso -> ponerEnlace(nuevo); //Cierra el circulo
    }
    acceso = nuevo; //Ahora nuevo es el acceso
};
```

La función visualizar()

- En una lista circular el recorrido puede empezar en cualquier nodo, a partir del cual se procesa cada nodo hasta alcanzar el nodo de partida.
- La función visualizar() inicia el recorrido en el nodo acceso a la lista y termina cuando alcanza de nuevo al nodo acceso.

```
//Visualiza la lista
void ListaCircular:: visualizar()
{
    NodoCircular* indice;
    if (acceso != NULL)
    {
        indice = acceso; // siguiente nodo al de acceso
        cout<<"DirDelDato"<<"\t"<<"Dato"<<"\t"<<"Siguiente"<<endl;
        do {
            cout <<indice<< "\t" <<indice->datoNodo()<<"\t"<<indice->enlaceNodo()<<endl;
            indice = indice -> enlaceNodo();
        }while(indice != acceso);
    }
};
```

Ejercicio 1

- Implemente en el programa de lista circular simplemente enlazada las funciones:
  - insertarDespues()
  - insertarAntes()
  - eliminar()
  - modificar()

Ejercicio 2

- Cree un programa que implemente una lista circular doblemente enlazada con las funciones fundamentales (crear, visualizar, insertar, etc.)

FIN