Universidad Mariano Gálvez de Guatemala
Facultad de Ingeniería en Sistemas de Información
Programación III
Ing. José Luis Xiloj

Nombre	Carné	Colaboración
Ashley Deydania Arredondo Salazar	1590-21-7193	100%
Angela Lolita Villalta García	1590-22-15807	100%
Juan Pablo Lopez Montoya	1590-16-11594	100%
Ludim Abisai Agreda Guzmán	1590-11-6130	0%
Sergio José Granados Mejía	1590-19-19917	100%

Sección “B”
3 de abril del 2024

Algoritmo para quitar nodo al inicio de la lista

1. Verificar si la lista está vacía.
    a) Retornar mensaje “Lista vacía”
2. Si la lista contiene un solo elemento
    a) Asignar NULL a primero y ultimo nodo de la lista.
3. Si la lista contiene más de un elemento
    a) Llevar registro del primer nodo en un nodo temporal.
    b) Asignar al primer nodo el contenido de su liga, es decir, el siguiente nodo
    c) Asignar nodo al nodo temporal.
4. Retornar mensaje

Algoritmo para quitar nodo al final de la lista

1. Verificar si la lista está vacía.
    a) Retornar mensaje “Lista vacía”
2. Si la lista contiene un solo elemento
    a) Asignar NULL a primero y ultimo nodo de la lista.
3. Si la lista contiene más de un elemento
    a) Crear nodoActual para tener la referencia del inicio de la lista
    b) Recorrer la lista mientras exista nodo después del siguiente nodo
    c) Crear nodo temporal
    d) Asignar al nodo temporal la referencia o liga del nodoActual (siguiente nodo)
    e) Asignar NULL a la liga del nodoActual
    f) Asignar al UltimoNodo la referencia del nodoActual
    g) Asignar nulo al nodo temporal.
4. Retornar mensaje

Algoritmo para eliminar un elemento con información X:

1. Comenzar desde el primer nodo de la lista y recorrer la lista hasta encontrar el nodo con la información que coincida con X.
2. Si se encuentra un nodo con la información X:
    a) Verificar si el nodo es el primer nodo de la lista.
        i. Si lo es, eliminar el nodo utilizando el método EliminarNodoAlInicio.
    b) Si no es el primer nodo:
        i. Iterar sobre la lista nuevamente, manteniendo una referencia al nodo anterior al nodo que coincide con la información X.
        ii. Una vez encontrado el nodo, enlazar el nodo anterior directamente con el nodo siguiente al nodo con información X, saltando el nodo con la información X.
        iii. Eliminar el nodo con información X liberando su memoria.
3. Si no se encuentra ningún nodo con la información X, indicar que la información no fue encontrada en la lista.

Algoritmo para eliminar nodo antes de un Dato de Referencia (X)

1. Crear 3 nodos temporales(punteros) para llevar la referencia del nodo anterior, nodo anterior y el nodo actual iniciando con la referencia del primer nodo de la lista en 3 nodos.
2. Recorrer la lista mientras el nodoActual sea diferente a NULL y el valor del nodo actual sea diferente al valor de referencia
    a) En cada recorrido llevar el control del nodo ante-anterior, anterior y actual.
3. Después del recorrido se verifica si el nodoActual es igual a NULL entonces significa que el valor de referencia no fue encontrado y se retorna el mensaje.
4. Caso contrario el valor de referencia fue encontrado
    a) Si el dato se encontró en el primer nodo
        i. Retornar mensaje para advertir que no existe nodo para eliminar.
    b) Si solo existe un nodo antes que el nodo actual
        i. Aplicar la operación de eliminar al inicio de la lista
    c) Caso contrario
        i. Asignar a la liga del nodo ante-anterior la liga del nodo anterior
        ii. Asignar NULL al nodo anterior (eliminar el nodo)
5. Retornar mensaje que el nodo fue eliminado.

Algoritmo para eliminar nodo después de un Dato de Referencia (X):

1. Crear tres punteros (nodos temporales) para llevar la referencia del nodo anterior, el nodo actual y el siguiente nodo, comenzando con la referencia del primer nodo de la lista en los tres nodos.
2. Recorrer la lista mientras el nodoActual no sea nulo y el valor del nodo actual sea diferente al valor de referencia.
    a) En cada recorrido, llevar el control del nodo anterior, el nodo actual y el siguiente nodo.
3. Después del recorrido, verificar si el nodoActual no es nulo, lo que significa que el valor de referencia se encontró.
4. Si se encontró el valor de referencia:
    a) Verificar si el nodoActual tiene un nodo siguiente.
        i. Si no tiene un nodo siguiente, retornar un mensaje indicando que no hay nodo para eliminar después del dato de referencia.
        ii. Si tiene un nodo siguiente, proceder con la eliminación.
5. Retornar un mensaje indicando que el nodo después del dato de referencia ha sido eliminado.

Algoritmo de Ordenamiento de Inserción para una Lista Simple:

1. Comenzar con el segundo nodo de la lista, ya que el primer nodo se considera trivialmente ordenado.
2. Iterar sobre el resto de la lista:

a) Para cada nodo actual, recorrer hacia atrás en la lista ordenada hasta encontrar la posición correcta para insertar el nodo actual.

b) Comparar el valor del nodo actual con el valor del nodo actual en la lista ordenada.

c) Si el valor del nodo actual es menor que el valor del nodo actual en la lista ordenada, insertar el nodo actual antes del nodo actual en la lista ordenada.

3. Repetir hasta que se hayan recorrido todos los nodos de la lista**.**

```
        public string EliminarNodoAlInicio()
        {
            if (EsListaVacia()) return "la lista esta vacía, no puede
continuar!!!";

            if (PrimerNodo == UltimoNodo)
            {
                PrimerNodo = UltimoNodo = null;
            }
            else
            {
                Nodo NodoTemporal;
                NodoTemporal = PrimerNodo;

                //Ligar El Segundo Como Primer Nodo
                PrimerNodo = PrimerNodo.Liga;
                NodoTemporal = null;
            }
            return "Nodo Eliminado!!!";
        }

        public string EliminarNodoAlFinal()
        {
            if (EsListaVacia()) return "la lista esta vacía, no puede
continuar!!!";

            if (PrimerNodo == UltimoNodo)
            {
                PrimerNodo = UltimoNodo = null;
            }
            else
            {
                Nodo NodoActual;
                Nodo NodoSiguiente;

                NodoActual = PrimerNodo;
                NodoSiguiente =NodoActual.Liga;

                while(NodoSiguiente.Liga != null)
                {
                    NodoActual = NodoActual.Liga;
                    NodoSiguiente = NodoActual.Liga;
                }
                NodoSiguiente = null;
                NodoActual.Liga = null;
                UltimoNodo = NodoActual;
            }
            return "Nodo Eliminado!!!";
        }

        public string EliminarElementoX(object valorReferencia)
{
    Nodo nodoActual = PrimerNodo;
    Nodo nodoAnterior = null;

    while (nodoActual != null && !nodoActual.Valor.Equals(valorReferencia))
    {
        nodoAnterior = nodoActual;
```

```
            nodoActual = nodoActual.Liga;
        }

        if (nodoActual != null)
        {
            if (nodoActual == PrimerNodo)
            {
                // El nodo con información X es el primer nodo de la lista
                EliminarNodoAlInicio();
            }
            else
            {
                // El nodo con información X no es el primer nodo de la lista
                nodoAnterior.Liga = nodoActual.Liga;
                nodoActual = null;
            }
            return $"Elemento con información '{valorReferencia}' eliminado!!!";
        }
        else
        {
            return $"El elemento con información '{valorReferencia}' no fue
encontrado en la lista!!!";
        }
    }
            public string EliminarNodoAntesDeX(object valorReferencia)
            {
                Nodo nodoActual = PrimerNodo;
                Nodo nodoAnterior = PrimerNodo;
                Nodo nodoAnterior2 = PrimerNodo;

                while (nodoActual != null &&
!nodoActual.Valor.Equals(valorReferencia))
                {
                    nodoAnterior2 = nodoAnterior;
                    nodoAnterior = nodoActual;
                    nodoActual = nodoActual.Liga;
                }

                if (nodoActual != null)
                {
                    if(nodoActual == PrimerNodo)
                    {
                        return $"No hay nodos para eliminar";
                    }
                    else if (nodoAnterior == nodoAnterior2)
                    {
                        EliminarNodoAlInicio();
                    }
                    else
                    {
                        nodoAnterior2.Liga = nodoAnterior.Liga;
                        nodoAnterior = null;
                    }
                    return $"Nodo Eliminado!!!";
                }
                else
                {
```

```
                return $"El valor de referencia (valorReferencia) no fue
encontrado!!!";
            }
        }


public string EliminarNodoDespuesDeX(object valorReferencia)
{
    Nodo nodoActual = PrimerNodo;
    Nodo nodoSiguiente = null;
    while (nodoActual != null && !nodoActual.Valor.Equals(valorReferencia))
    {
        nodoActual = nodoActual.Liga;
    }
    if (nodoActual != null)
    {
        nodoSiguiente = nodoActual.Liga;


        if (nodoSiguiente == null)
        {
            return $"No hay nodos para eliminar después del dato de referencia.";
        }
        else
        {
            nodoActual.Liga = nodoSiguiente.Liga;
            nodoSiguiente = null;
            return $"Nodo después del dato de referencia eliminado!!!";
        }
    }
    else
    {
        return $"El valor de referencia no fue encontrado!!!";
    }
}
    public void Ordenar()
    {
```

```
        if (PrimerNodo == null || PrimerNodo.Liga == null)
            return;
        Nodo actual = PrimerNodo.Liga;
        Nodo previo = PrimerNodo;
        while (actual != null)
        {
            Nodo siguiente = actual.Liga;
            Nodo comparador = PrimerNodo;
            Nodo anteriorComparador = null;
            while (comparador != actual)
            {
                if ((int)comparador.Valor > (int)actual.Valor)
                {
                    previo.Liga = siguiente;
                    actual.Liga = comparador;
                    if (anteriorComparador == null)
                        PrimerNodo = actual;
                    else
                        anteriorComparador.Liga = actual;
                    break;
                }
                anteriorComparador = comparador;
                comparador = comparador.Liga;
            }
            if (comparador == actual)
                previo = actual;
            actual = siguiente;
        }
           }

public string EliminarNodoEnPosicion(int posicion)
{
    if (ListaVacia())
    {
        return "La lista está vacía, no se puede eliminar ningún nodo.";
    }
    else if (posicion < 0)
```

```
    {
        return "La posición especificada es inválida.";
    }
    else if (posicion == 0)
    {
        return EliminarNodoAlInicio();
    }
    else
    {
        Nodo nodoAnterior = null;
        Nodo nodoActual = PrimerNodo;
        int contador = 0;

        while (nodoActual != null && contador < posicion)
        {
            nodoAnterior = nodoActual;
            nodoActual = nodoActual.Referencia;
            contador++;
        }

        if (nodoActual == null)
        {
            return "La posición especificada excede el tamaño de la lista.";
        }
        else
        {
            nodoAnterior.Referencia = nodoActual.Referencia;

            if (nodoActual == UltimoNodo)
            {
                UltimoNodo = nodoAnterior;
            }

            return $"Se ha eliminado el nodo en la posición {posicion}.";
        }
    }
}public void Ordenar()
{
    if (PrimerNodo == null || PrimerNodo.Liga == null)
        return;
    Nodo actual = PrimerNodo.Liga;
    Nodo previo = PrimerNodo;
    while (actual != null)
    {
        Nodo siguiente = actual.Liga;
        Nodo comparador = PrimerNodo;
        Nodo anteriorComparador = null;
        while (comparador != actual)
        {
            if ((int)comparador.Valor > (int)actual.Valor)
            {
                previo.Liga = siguiente;
                actual.Liga = comparador;
                if (anteriorComparador == null)
                    PrimerNodo = actual;
                else
                    anteriorComparador.Liga = actual;
                break;
```

```
            }
            anteriorComparador = comparador;
            comparador = comparador.Liga;
        }
        if (comparador == actual)
            previo = actual;
        actual = siguiente;
    }
    }
```

INSERTAR NODOS

ELIMINAR NODO AL INICIO

ELIMINAR NODO AL FINAL

ELIMINAR NODO POR VALOR

ELIMINAR NODO ANTES DE VALOR X

ELIMINAR NODO DESPUES DE VALOR X

ORDENAR LISTA