

Universidad Tecnológica del Norte de Guanajuato

Tecnologías de la Información y Comunicación

Licenciatura en Ingeniería en Tecnologías de la Información e  
Innovación Digital

Estructura de Datos

Alumno: Rojas Galindo Marlon

No. Control: 1224100711

Grupo: GTID-0141

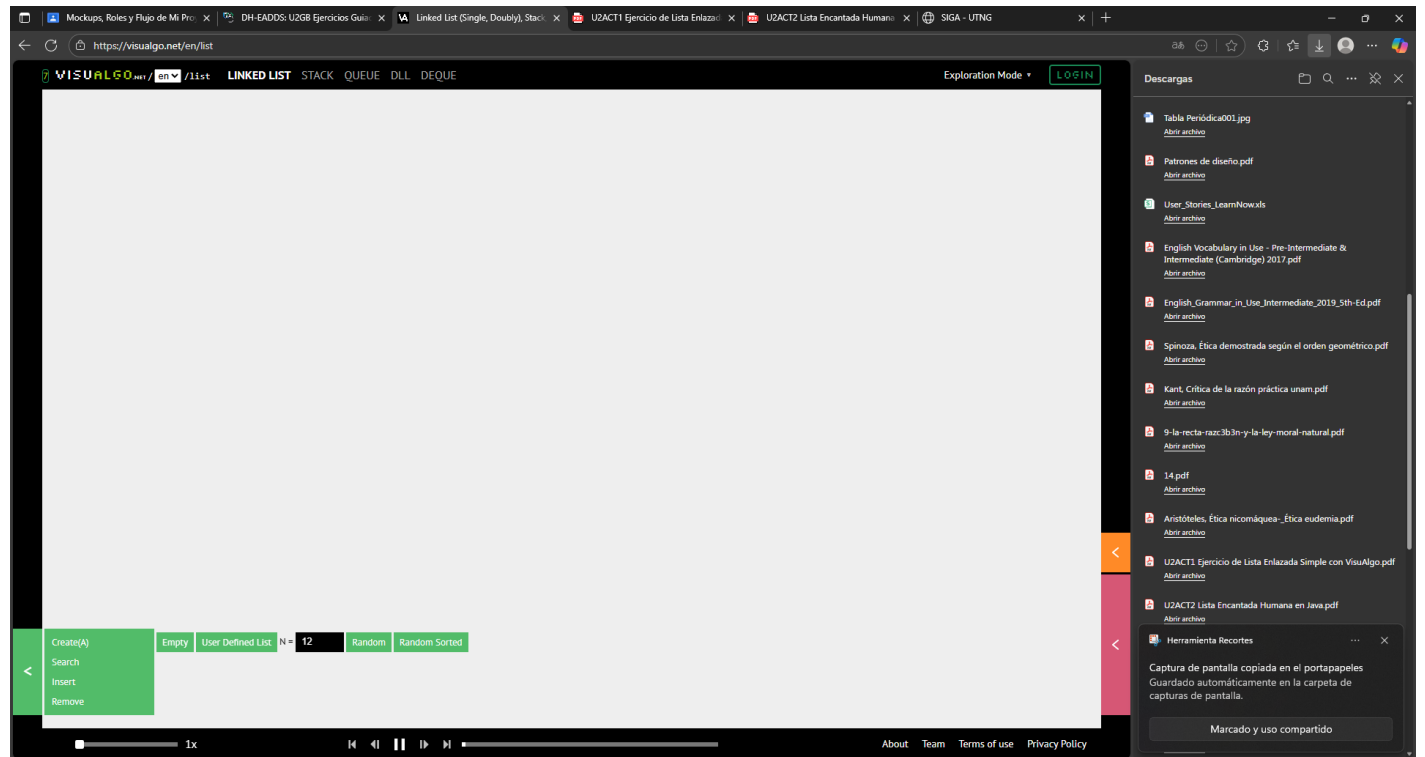
Unidad 2: Estructuras de Datos Básicas

Actividad 1

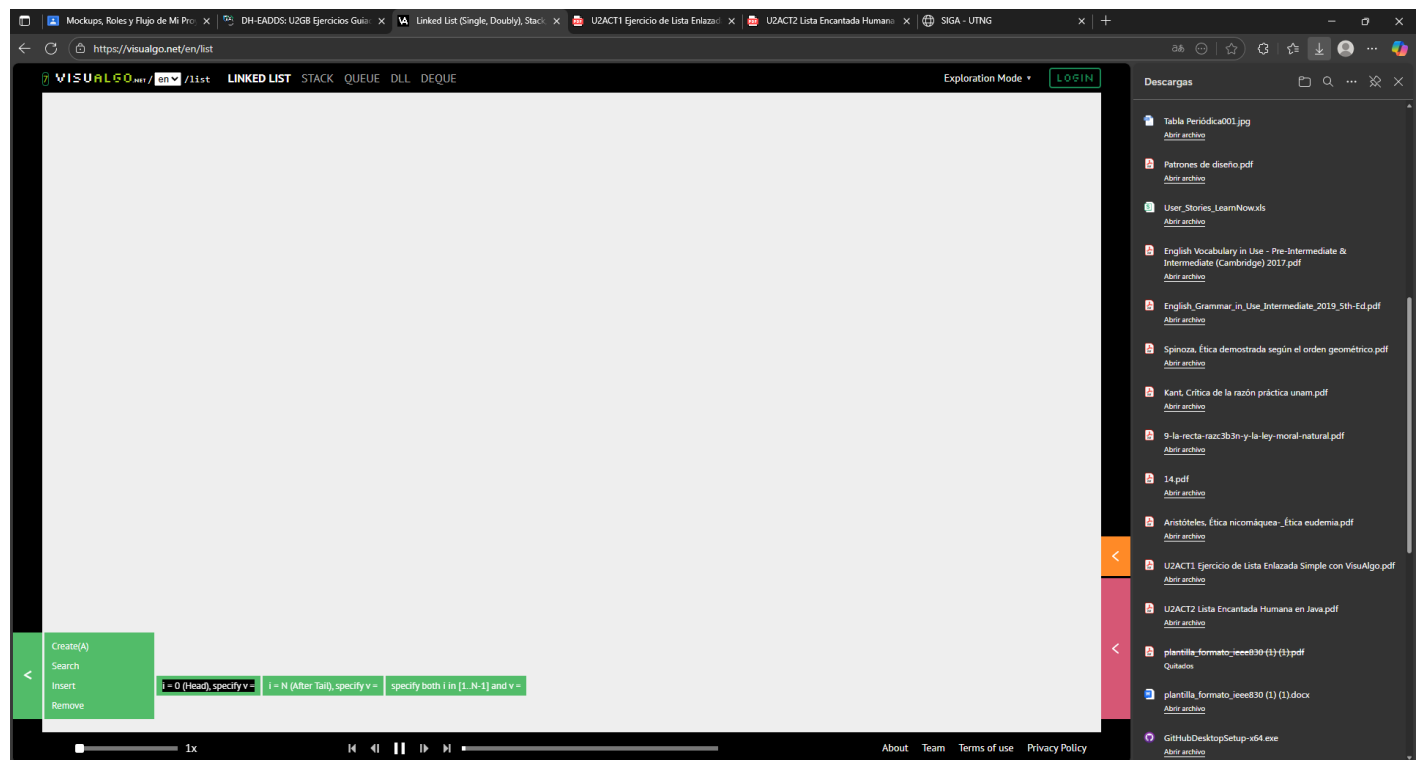
Docente: Barrón Rodríguez Gabriel

Dolores Hidalgo, C.I.N.; 7 de octubre de 2025

Se crea una lista vacía.



Se agrega un elemento a la lista.



Se inserta un elemento en la cabeza de la lista.

The screenshot shows the Visualgo interface for a linked list. The main canvas displays two nodes: 92 (green) and 18 (black). Node 92 is labeled 'head/0' and node 18 is labeled 'tail/1'. An arrow points from node 92 to node 18. On the right, a code block titled 'Insert 92 at head' shows the following steps:

```
head points to vtx.  
Vertex vtx = new Vertex(v)  
vtx.next = head  
head = vtx
```

The interface includes a top navigation bar with 'LINKED LIST', 'STACK', 'QUEUE', 'DLL', and 'DEQUE'. A 'LOGIN' button is in the top right. A sidebar on the right lists various downloadable files. The bottom of the interface has a progress bar and navigation controls.

Se inserta un elemento al final de la lista.

The screenshot shows the Visualgo interface for a linked list. The main canvas displays three nodes: 92 (black), 18 (black), and 65 (green). Node 92 is labeled 'head/0' and node 65 is labeled 'tail/2'. Arrows point from node 92 to node 18, and from node 18 to node 65. On the right, a code block titled 'Insert 65 at tail' shows the following steps:

```
tail points to vtx.  
The whole operation is O(1) if we maintain the tail pointer.  
Vertex vtx = new Vertex(v)  
tail.next = vtx  
tail = vtx
```

The interface includes a top navigation bar with 'LINKED LIST', 'STACK', 'QUEUE', 'DLL', and 'DEQUE'. A 'LOGIN' button is in the top right. A sidebar on the right lists various downloadable files. The bottom of the interface has a progress bar and navigation controls.

Se inserta un elemento entre dos elementos de la lista.

The screenshot shows the Visualgo interface with a linked list. The list contains nodes with values 92, 18, 65, and 77. The head pointer is at node 18. A tooltip displays the following code for inserting a new node at index 2:

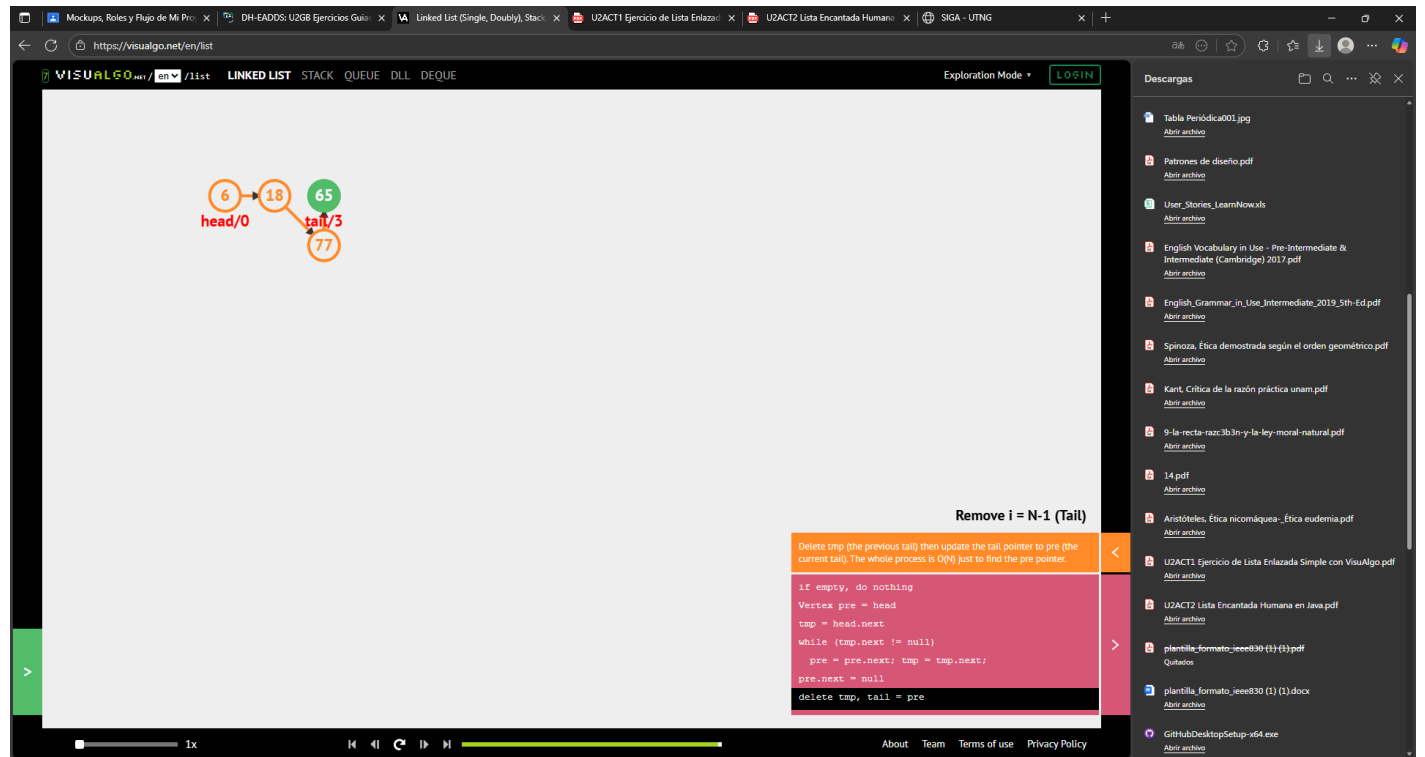
```
pre.next points to vtx.  
Vertex pre = head  
for (k = 0; k < i-1; k++)  
    pre = pre.next  
Vertex aft = pre.next  
Vertex vtx = new Vertex(v)  
vtx.next = aft  
pre.next = vtx
```

Se elimina el elemento del inicio (HEAD) y se le asigna la propiedad al elemento siguiente.

The screenshot shows the Visualgo interface with a linked list. The list contains nodes with values 18, 65, and 77. The head pointer is at node 18. A tooltip displays the following code for deleting the head element:

```
Remove i = 0 (Head)  
Delete tmp, which was the (previous) head  
if empty, do nothing  
tmp = head  
head = head.next  
delete tmp
```

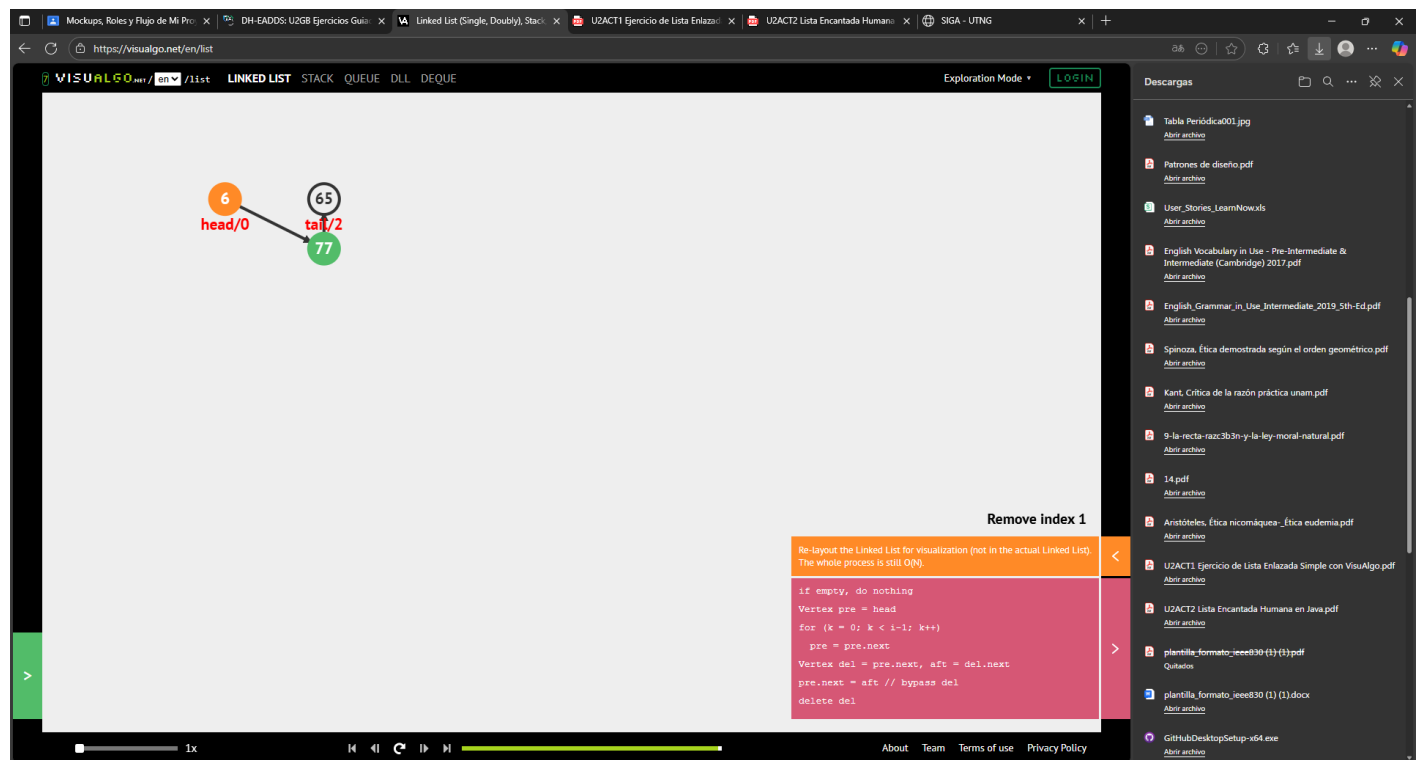
Se elimina el elemento del final.



Visualgo interface showing a linked list with nodes 6, 18, 65, and 77. The head pointer is at node 6 and the tail pointer is at node 77. The code block on the right shows the logic for removing the last element (tail):

```
Remove i = N-1 (Tail)
Delete tmp (the previous tail) then update the tail pointer to pre (the current tail). The whole process is O(N) just to find the pre pointer.
if empty, do nothing
Vertex pre = head
tmp = head.next
while (tmp.next != null)
    pre = tmp.next; tmp = tmp.next;
pre.next = null
delete tmp, tail = pre
```

Se elimina un elemento intermedio en la lista.



Visualgo interface showing a linked list with nodes 6, 65, and 77. The head pointer is at node 6 and the tail pointer is at node 77. The code block on the right shows the logic for removing an intermediate element (index 1):

```
Remove index 1
Re-layout the Linked List for visualization (not in the actual Linked List). The whole process is still O(N).
if empty, do nothing
Vertex pre = head
for (x = 0; x < i-1; x++)
    pre = pre.next
Vertex del = pre.next, aft = del.next
pre.next = aft // bypass del
delete del
```

Se busca un elemento en la lista, en este caso 64, el cual devuelve NOT\_FOUND.

Visualgo interface showing a linked list search for the value 64. The list contains nodes with values 6, 65, 91, 36, and 53. The head pointer is at 0 and the tail pointer is at 5. A temporary pointer 'tmp' is shown as null, indicating the search has reached the end of the list without finding the target value.

Search 64

```
tmp is null (we have gone past the tail after O(n) step(s)).  
We conclude that value v = 64 is NOT_FOUND in the Linked List.  
  
if empty, return NOT_FOUND  
index = 0, tmp = head  
while (tmp.item != v)  
    index++, tmp = tmp.next  
if tmp == null  
    return NOT_FOUND  
return index
```

Se busca un elemento en la lista, en este caso retorna el índice del elemento a buscar.

Visualgo interface showing a linked list search for the value 91. The list contains nodes with values 6, 65, 91, 36, and 53. The head pointer is at 0 and the tail pointer is at 5. The temporary pointer 'tmp' is now at index 3, pointing to the node with value 91, indicating the search was successful.

Search 91

```
Found value v = 91 at this highlighted vertex so we return index 3.  
The whole operation is O(n).  
  
if empty, return NOT_FOUND  
index = 0, tmp = head  
while (tmp.item != v)  
    index++, tmp = tmp.next  
if tmp == null  
    return NOT_FOUND  
return index
```

## Preguntas.

**1. ¿Qué sucede con los punteros cuando se inserta o elimina un nodo?**

En el caso de la cabeza, se asigna el puntero al nodo siguiente y en el caso de el nodo final o cola, no se tiene un puntero de referencia.

**2. ¿Cómo afecta la posición de un nodo (inicio, medio y final) al tiempo de búsqueda?**

El recorrido de la lista comienza desde la cabeza hasta la cola, dando como resultado que si se quiere buscar un elemento con un índice más alejado de la cabeza es más tardado.

**3. ¿Qué ventajas tiene recorrer una lista enlazada frente a otras estructuras como arreglos?**

Que las listas enlazadas tienen una cantidad de datos más dinámica que un arreglo, al igual que en cuestión de memoria es más eficiente.

**4. ¿Cómo podrías implementar la comprobación de una lista vacía en un lenguaje de programación como java?**

Implementar una función que regrese un valor booleano que indique si la cabeza de la lista es nula.