Estructura de Datos

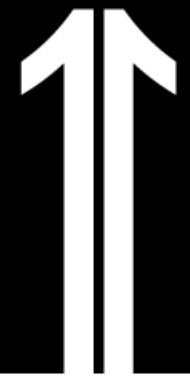
Del 20 al 24 de Mayo de 2024

UPN.EDU.PE

Semana 01

LISTAS ENLAZADAS SIMPLES

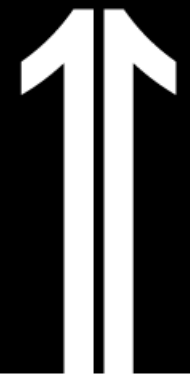
PRESENTACIÓN DE LA SESIÓN
Logro de la Sesión y Temario



Al término de la sesión, el estudiante aprende algoritmos de listas enlazadas, como representarlos y usarlos con eficacia.

- Presentación del sílabo del curso.
- Listas enlazadas Simples: Operaciones de inserción, recorrido, eliminación.

Reflexiona



- ¿Qué es una lista enlazada?
- ¿Qué es una estructura?
- ¿Qué es un nodo?

Introducción

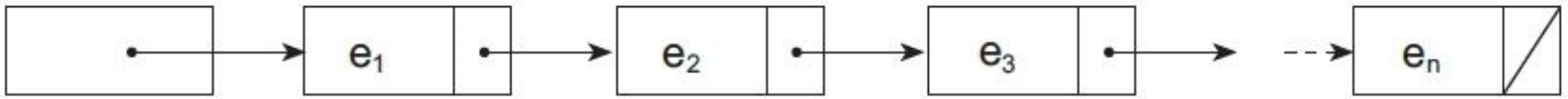
- El principal beneficio de las listas enlazadas respecto a los arreglos convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.
- Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante, pero no permiten un acceso aleatorio.

Conceptos

- Lista enlazada: Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias al nodo anterior y/o posterior.
- Cabeza: Al primer nodo de una lista enlazada
- Fin: Al último nodo de una lista enlazada

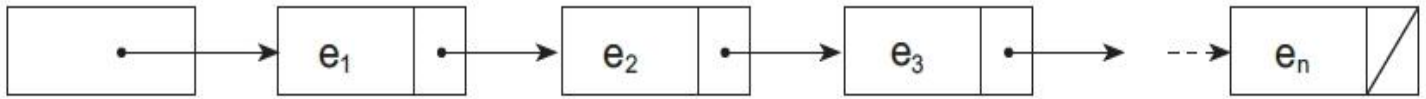
¿QUÉ ES UNA LISTA ENLAZADA?

- Una lista enlazada consta de un número de nodos con dos componentes(campos), un enlace al siguiente nodo de la lista y un valor, que puede ser de cualquier tipo.



$e_1, e_2, ...e_n$ son valores del tipo `TipoElemento`

¿QUÉ ES UNA LISTA ENLAZADA?



$e_1, e_2, ...e_n$ son valores del tipo `TipoElemento`

```
struct Nodo{
    int dato;
    Nodo *siguiente;
};
```

CLASIFICACIÓN DE LISTAS ENLAZADAS:

Las listas se pueden dividir en cuatro categorías:

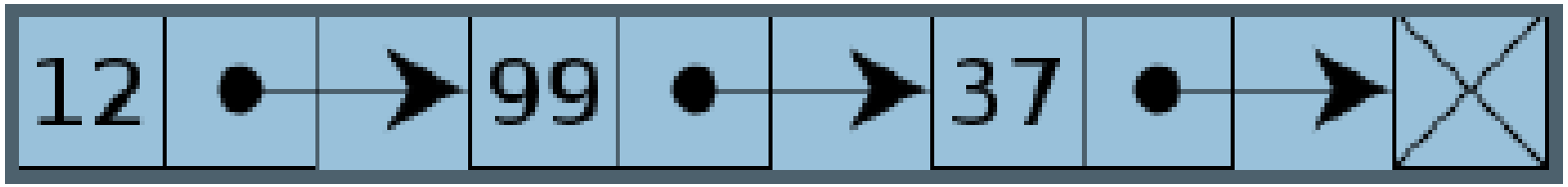
1. Listas Simplemente Enlazadas
2. Listas Doblemente Enlazadas
3. Lista Circular Simplemente Enlazada
4. Lista Circular Doblemente Enlazada

LISTAS SIMPLEMENTE ENLAZADAS

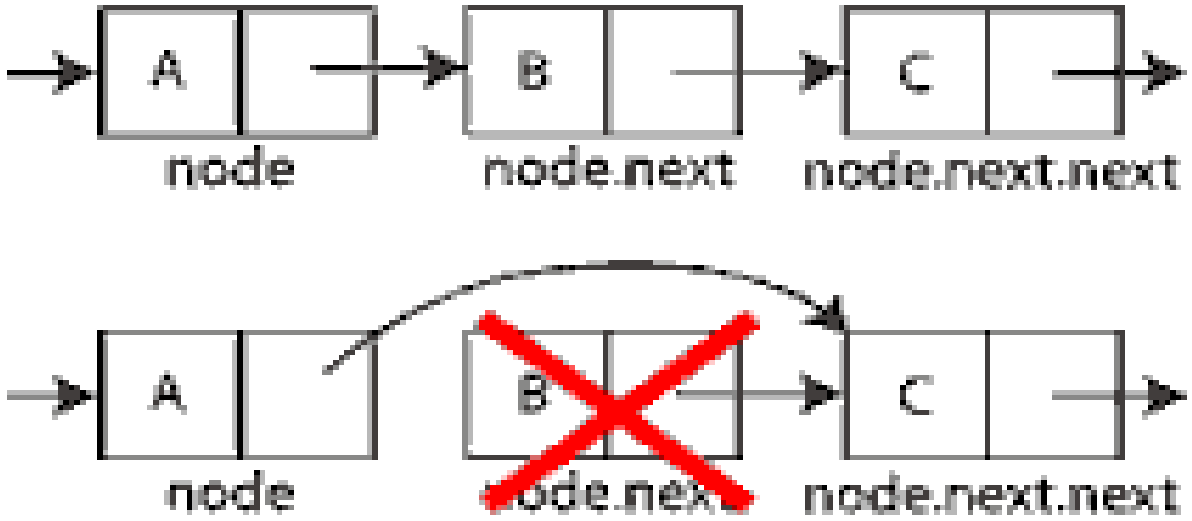
- Cada nodo (elemento) contiene un único enlace que conecta ese nodo al nodo siguiente o nodo sucesor. La lista es eficiente en recorridos directos ((<adelante»).

LISTAS SIMPLEMENTE ENLAZADAS

- Tiene un enlace por nodo. Este enlace apunta al siguiente nodo en la lista, o al valor Nulo o lista Vacía, si es el último nodo.



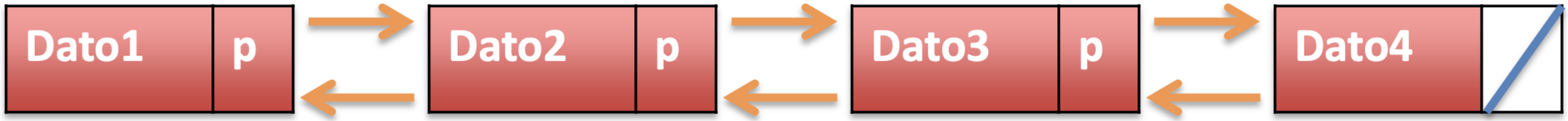
LISTAS ENLAZADAS SIMPLES



¿Qué es una lista enlazada?

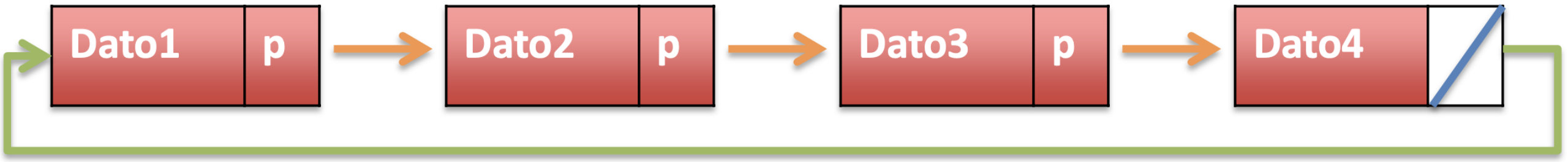
LISTAS DOBLEMENTE ENLAZADAS

- Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor. La lista es eficiente tanto en recorrido directo («adelante») como en recorrido inverso («atrás»).



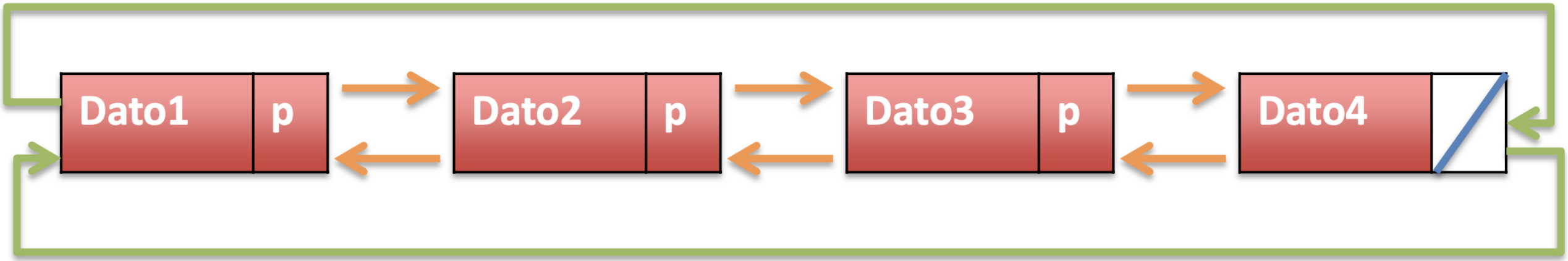
LISTA CIRCULAR SIMPLEMENTE ENLAZADA

- Una lista enlazada simplemente en la que el último elemento (cola) se enlaza al primer elemento (cabeza) de tal modo que la lista puede ser recorrida de modo circular («en anillo»).



LISTA CIRCULAR DOBLEMENTE ENLAZADA:

- Una lista doblemente enlazada en la que el último elemento se enlaza al primer elemento y viceversa. Esta lista se puede recorrer de modo circular (en anillo) tanto en dirección directa («adelante») como inversa («atrás»).



OPERACIONES EN LISTAS ENLAZADAS

- Insertar elementos en una lista enlazada
- Mostrar los elementos de una lista enlazada
- Buscar un elemento en una lista enlazada
- Eliminar un elemento en una lista enlazada

Listas enlazadas Simples. Operaciones: ordenamiento, búsqueda y mezcla

BUSCAR UN ELEMENTO EN UNA LISTA

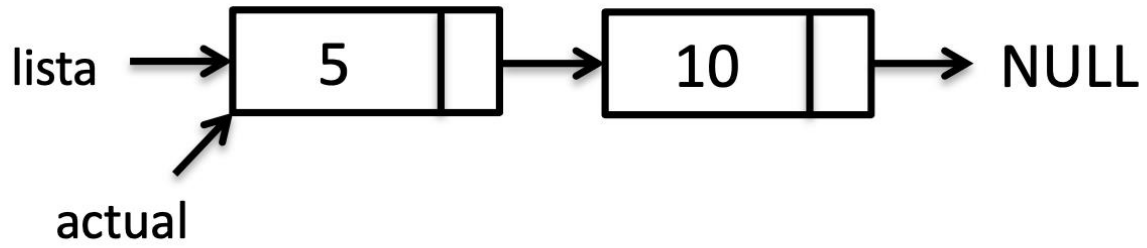
Para buscar un elemento en una lista, sólo hay que seguir 4 pasos:

1. Crear un nuevo nodo(actual).
2. Igualar ese nuevo nodo(actual) a la lista.
3. Recorrer la lista.
4. Determinar si el elemento existe o no en la lista.

1. CREAR UN NUEVO NODO(ACTUAL).

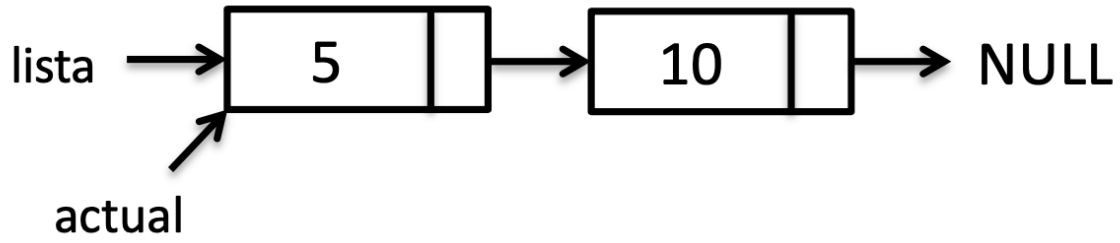

```
void buscarLista(Nodo *lista,int n){
    Nodo *actual = new Nodo();
}
```

2. IGUALAR ESE NUEVO NODO(ACTUAL) A LA LISTA.




```
void buscarLista(Nodo *lista,int n){
    Nodo *actual = new Nodo();
    actual = lista;
}
```

3. RECORRER LA LISTA.



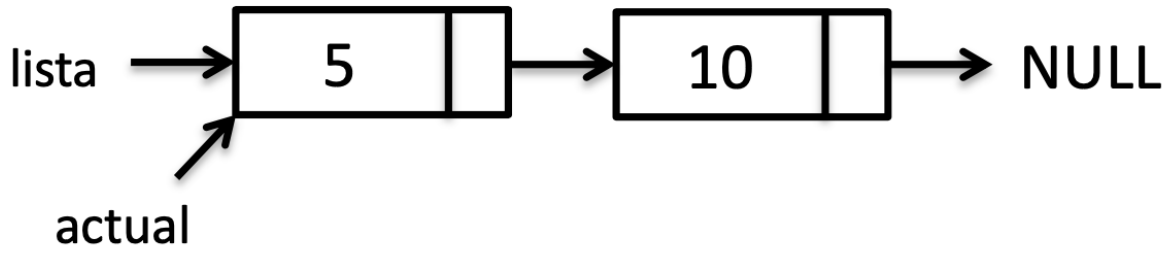
n = 10;

```
while((actual != NULL) && (actual->dato <= n)){



}
```

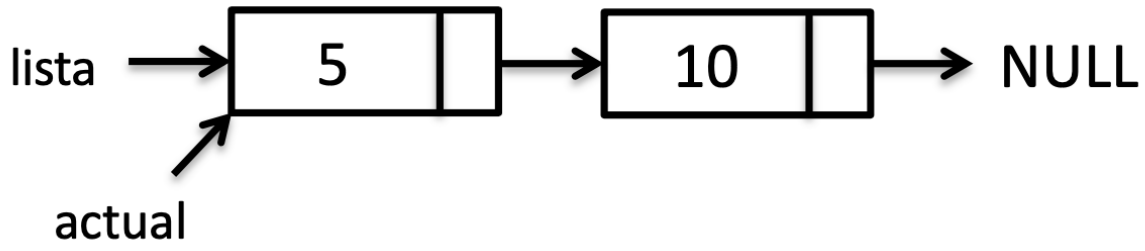
4. DETERMINAR SI EL ELEMENTO EXISTE O NO EN LA LISTA.



n = 10;

```
while((actual != NULL) && (actual->dato <= n)){
        if(actual->dato == n){
           band = true;
        }
        actual = actual->siguiente;
}
```

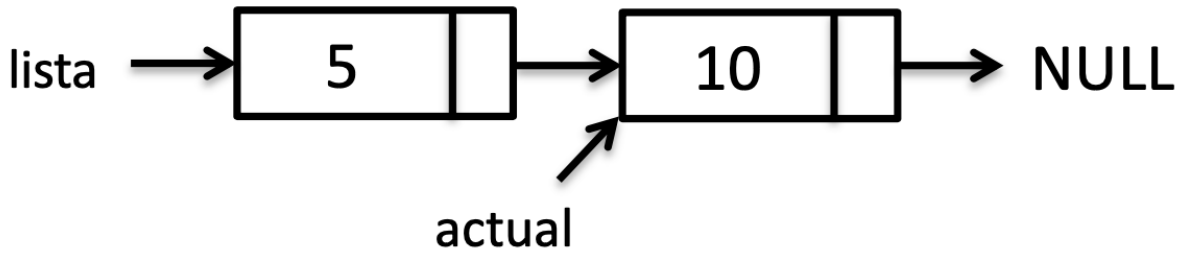
4. DETERMINAR SI EL ELEMENTO EXISTE O NO EN LA LISTA.



n = 12;

```
while((actual != NULL) && (actual->dato <= n)){
      if(actual->dato == n){
        band = true;
      }
      actual = actual->siguiente;
}
```

4. DETERMINAR SI EL ELEMENTO EXISTE O NO EN LA LISTA.



n = 12;

```
while((actual != NULL) && (actual->dato <= n)){
        if(actual->dato == n){
           band = true;
        }
        actual = actual->siguiente;
}
```

ACTIVIDAD

- Desarrollar y analizar el siguiente código:

```
void buscarLista(Nodo *lista,int n){
    bool band = false;

    Nodo *actual = new Nodo();
    actual = lista;

    while((actual != NULL) && (actual->dato <= n)){
        if(actual->dato == n){
            band = true;
        }
        actual = actual->siguiente;
    }

    if(band == true){
        cout<<"Elemento "<<n<<" SI a sido encontrado en lista\n";
    }
    else{
        cout<<"Elemento "<<n<<" NO a sido encontrado en lista\n";
    }
}
```

CONCLUSIONES

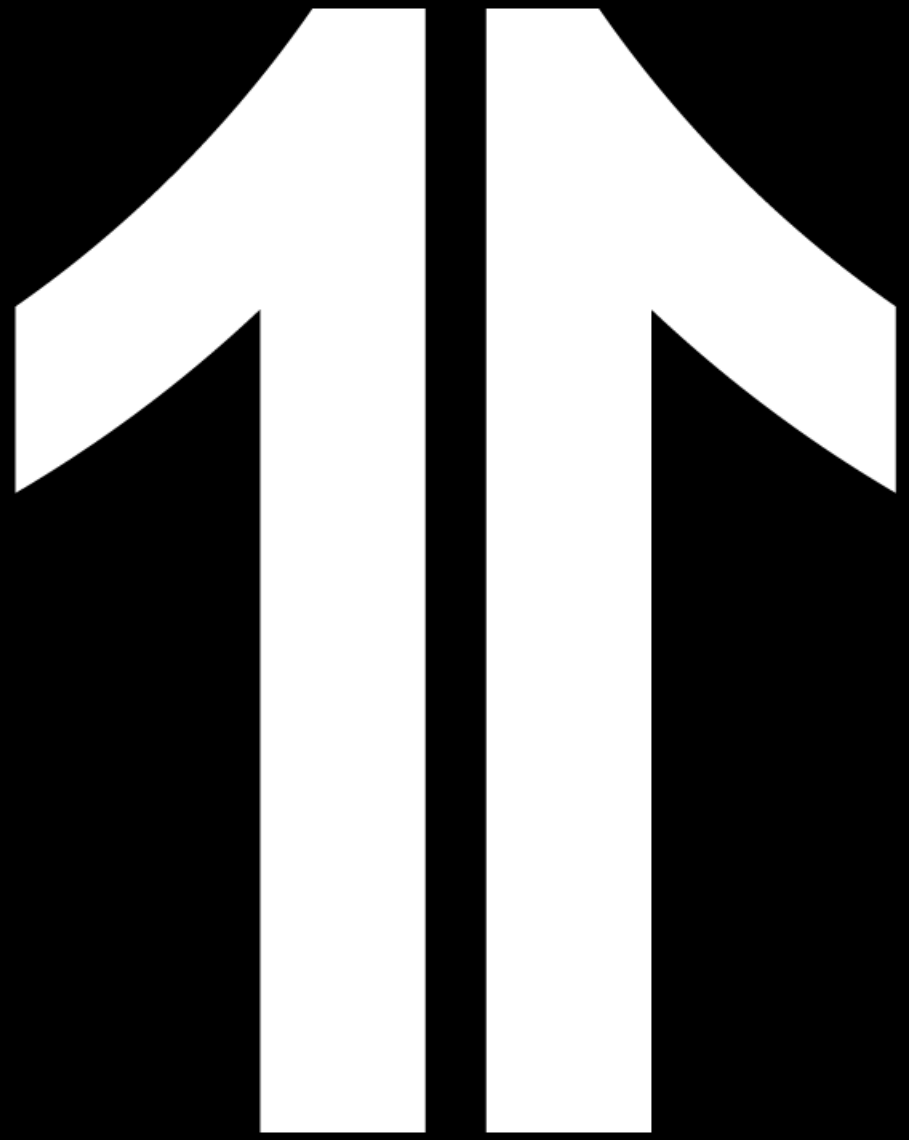
- Los elementos se pueden insertar en una lista indefinidamente mientras que un arreglo tarde o temprano se llenará o necesitará ser redimensionado, una costosa operación que incluso puede no ser posible si la memoria se encuentra fragmentada.

CONCLUSIONES

- Las listas son de acceso secuencial, y sólo puede ser recorridas en una dirección.
- El acceso secuencial es más lento en las listas.
- Se requiere de almacenamiento extra para las referencias
- Puede resultar lento asignar memoria para cada nuevo elemento.

BIBLIOGRAFIA REFERENCIAL

- Ceballos Sierra, F. Microsoft C#: Curso de Programación (2a.ed.) 2014 https://elibronet.eu1.proxy.openathens.net/es/lc/upnorte/titulos/106417
- Cesar Liza Avila; Estructura de datos con C/C++



GRACIAS


UPN
UNIVERSIDAD
PRIVADA
DEL NORTE