Fundamentos de análisis y diseño de algoritmos

Estructuras de datos

Generalidades

Pilas

Colas

Listas enlazadas

Listas doblemente enlazadas

Apuntadores y Objetos

Árboles con raíz

Estructuras de datos

Generalidades (1)

Los conjuntos son fundamentales tanto en las matemáticas como en las ciencias de la computación

Los conjuntos que son manipulados mediante algoritmos, que crecen, disminuyen o cambian con el tiempo, son llamados conjuntos **dinámicos**

Diccionario es aquel conjunto dinámico que soporta operaciones tales como insertar y borrar elementos, o chequear si un elemento pertenece al conjunto.

Estructuras de datos

Generalidades (2)

En las implementaciones típicas de un conjunto dinámico, cada elemento es representado por un objeto cuyos campos pueden ser examinados y manipulados si existe un puntero al objeto

Algunos conjuntos dinámicos asumen que uno de los campos de los objetos es un campo **llave**.

Si todas las llaves son diferentes, se puede pensar del conjunto dinámico como un conjunto de valores de llave.

Estructuras de datos

Generalidades (3)

Las operaciones en los conjuntos dinámicos pueden agruparse en: **consultay modificación.**

Algunas operaciones.

- **Search(S,k)** Retorna el elemento tal que key[x]=k o NIL
- **Insert(S,x)** Modificación del conjunto del elemento apuntado por x
- **Delete(S,x)** Borra un elemento dado un puntero x
- **Maximo(S)**, **Minimo(S)**
- **Sucesor(S,x)** Retorna el elemento siguiente en orden al apuntado por x
- **Predecesor(S,x)** Retorna el elemento anterior en orden al apuntado por x

Estructuras de datos

Pila

Una pila es una estructura de datos tipo LIFO (Last In First Out), por lo que el último elemento insertado será el primero en ser borrado

Operaciones básicas:

STACK-EMPTY(S)

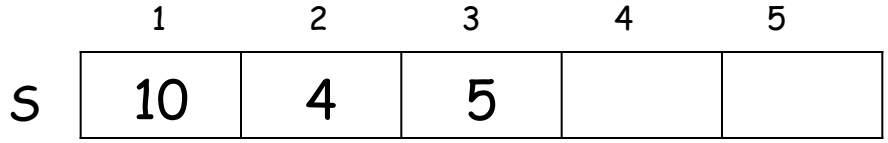
PUSH(S,x)

POP(S)

Estructuras de datos

Pila

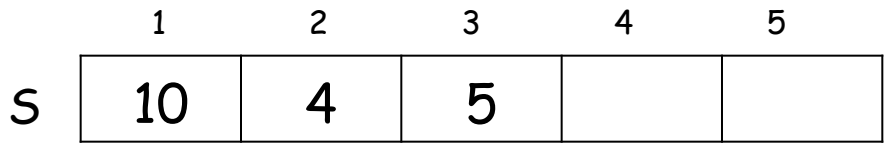
Una forma de implementar la pila es por medio de un arreglo unidimensional



Esto supone varios aspectos:

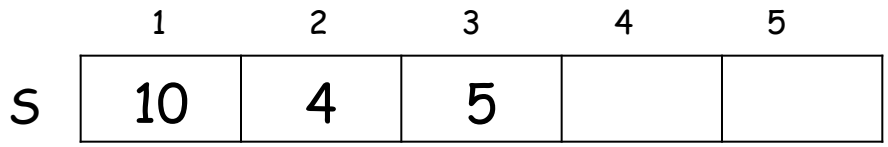
- La pila tiene una capacidad limitada
- Se cuenta con un atributo adicional, llamado top[S], que almacena el índice en el arreglo que guarda el último valor, esto es, el tope de la pila
- Cuando la pila esté vacía, top[S]=0

Estructuras de datos



top[S]=3

Estructuras de datos

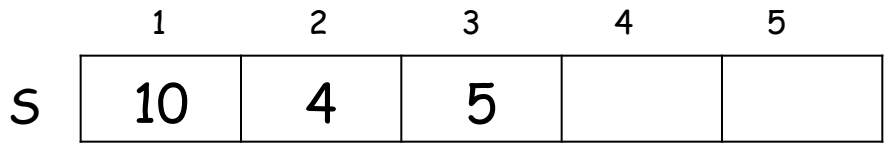


top[S]=3

Indique lo que sucede después de cada instrucción, siendo S la pila que se muestre arriba:

STACK-EMPTY(S)
PUSH(S,4)
PUSH(S,12)
PUSH(S,7)

Estructuras de datos

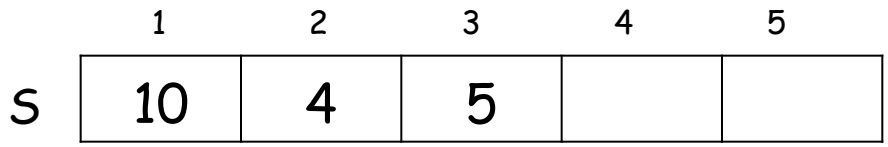


top[S]=3

Indique lo que sucede después de cada instrucción, siendo S la pila que se muestre arriba:

STACK-EMPTY(S)
PUSH(S,4)
PUSH(S,12)
PUSH(S,7) ← Overflow - desbordamiento en su capacidad máxima

Estructuras de datos



top[S]=3

Indique lo que sucede después de cada instrucción, siendo S la pila que se muestre arriba:

STACK-EMPTY(S)
POP(S)
POP(S)
POP(S)
STACK-EMPTY(S)
POP(S) ← Underflow

Estructuras de datos

Indique un algoritmo para cada operación básica, acompañado de su respectiva complejidad usando la notación O

- STACK-EMPTY(S), retorna true o false
- PUSH(S,x), adiciona x al tope, no devuelve ningún valor
- POP(S), borra el elemento que esté en el tipa y devuelve ese valor

Estructuras de datos

```
STACK-EMPTY(S)
1  if (top[S]==0)
2   then return true
3   else return false
```

Estructuras de datos

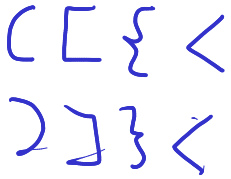
```
STACK-EMPTY(S)
1  if (top[S]==0)
2   then return true
3   else return false
```

T(n)=O(1), tiempo constante

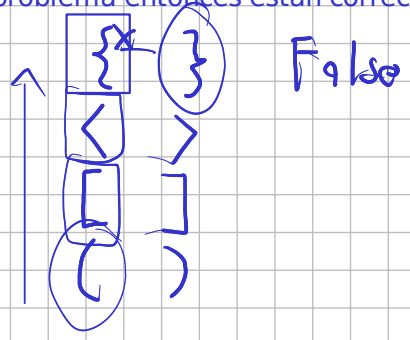
3. Write a function, where given a string of arbitrary characters, returns true if all brackets (defined as parentheses, square-brackets, curly-braces, and chevrons) are correctly paired and ordered. This is to say that all brackets, if they enclose other brackets, enclose both the paired opening and closing characters.

Examples: input -> "([<{abc123abc}>])"
output -> true

input -> "(abc[123)abc]"
output -> false



Usar una pila para incluir los de apertura y los vamos sacando a medida que llegan lo de cerradura, si la pila queda vacia al terminal el problema entonces están correctamente organizados



Estructuras de datos

Cola

Una cola es una estructura de datos tipo FIFO (First In First Out), por lo que el primer elemento que es insertado, es el primero en ser borrado

Operaciones básicas:

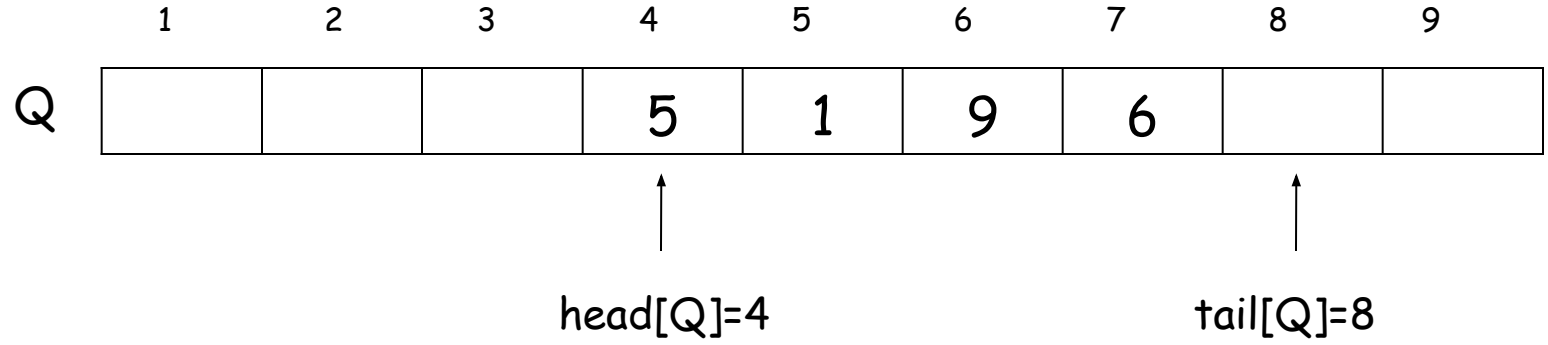
ENQUEUE(Q,x)

DEQUEUE(Q)

Estructuras de datos

Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional



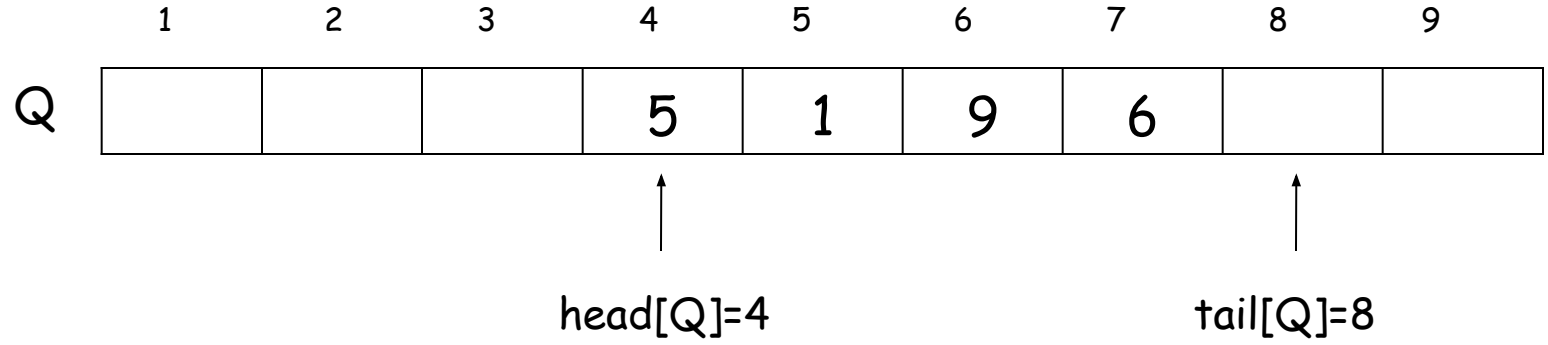
Esto supone varios aspectos:

- La cola tiene una capacidad limitada
- Se cuenta con dos atributos adicionales, head[Q] que guarda el indice de la cabeza y tail[Q] que apunta al siguiente lugar en el cual será insertado un elemento

Estructuras de datos

Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional



ENQUEUE(Q,2)

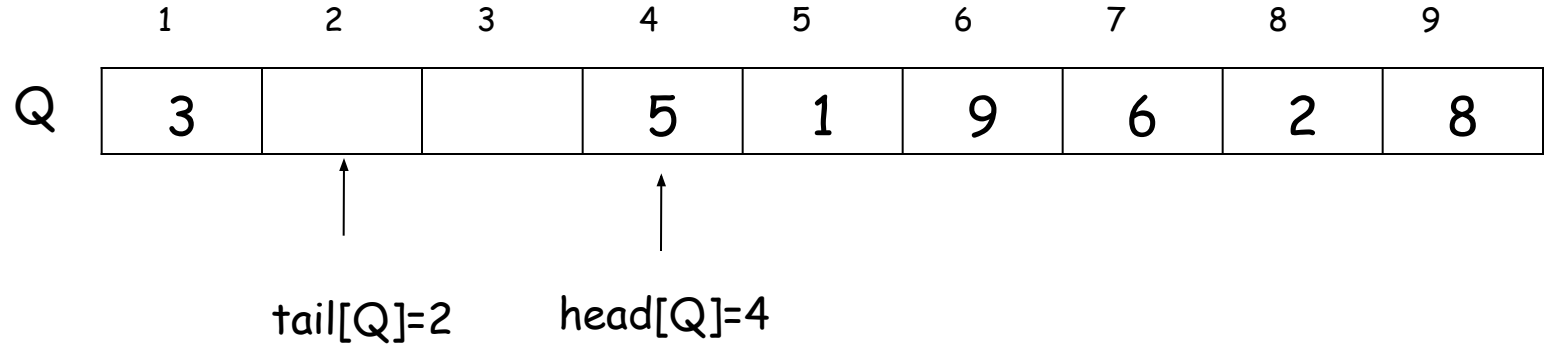
ENQUEUE(Q,8)

ENQUEUE(Q,3)

Estructuras de datos

Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional



ENQUEUE(Q,2)

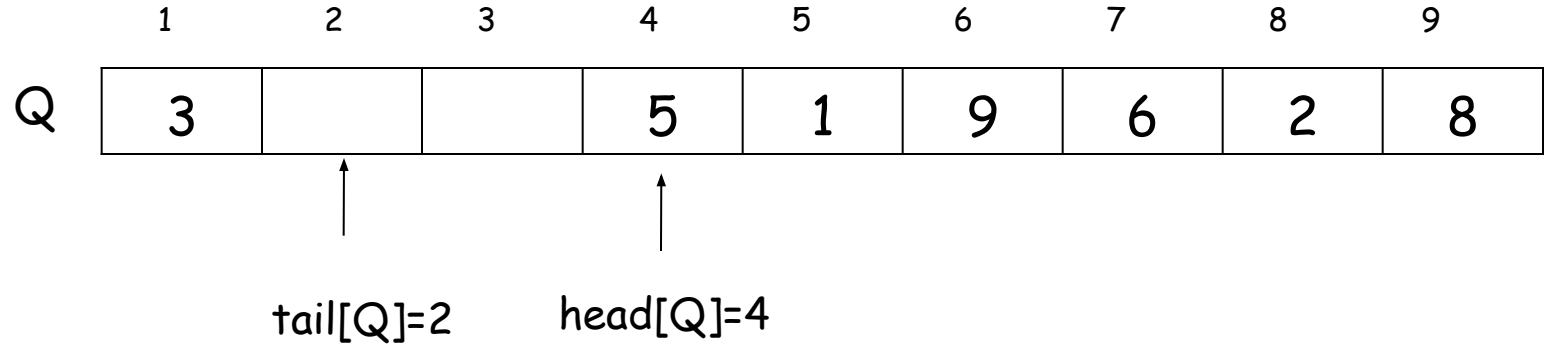
ENQUEUE(Q,8) ← Si se llega al final del arreglo, se intenta insertar en la posición 1

ENQUEUE(Q,3)

Estructuras de datos

Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional

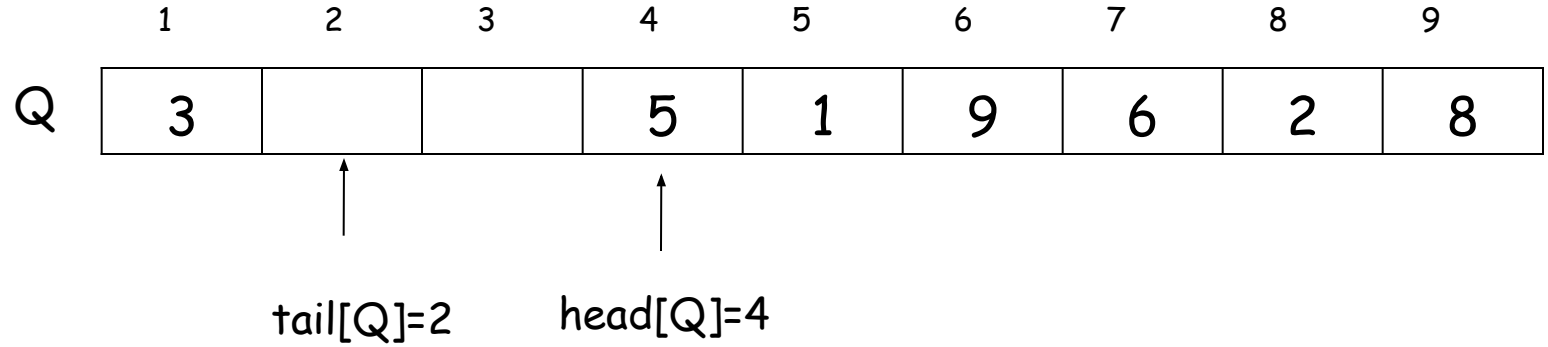


Cómo sabe que la cola está llena?

Estructuras de datos

Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional



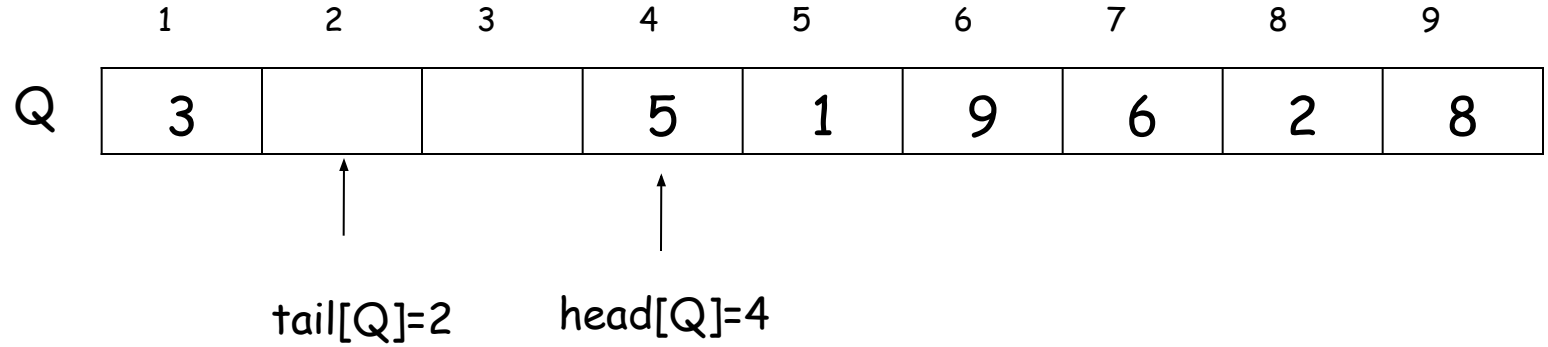
Cómo sabe que la cola está llena?

tail[Q]=head[Q]

Estructuras de datos

Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional

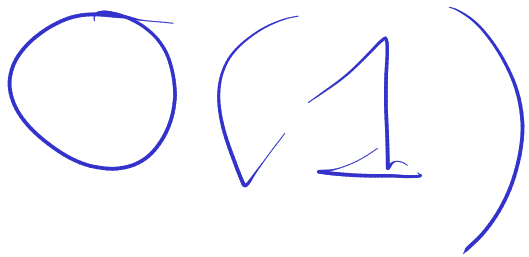


Inicialmente tail[Q]=head[Q]=1

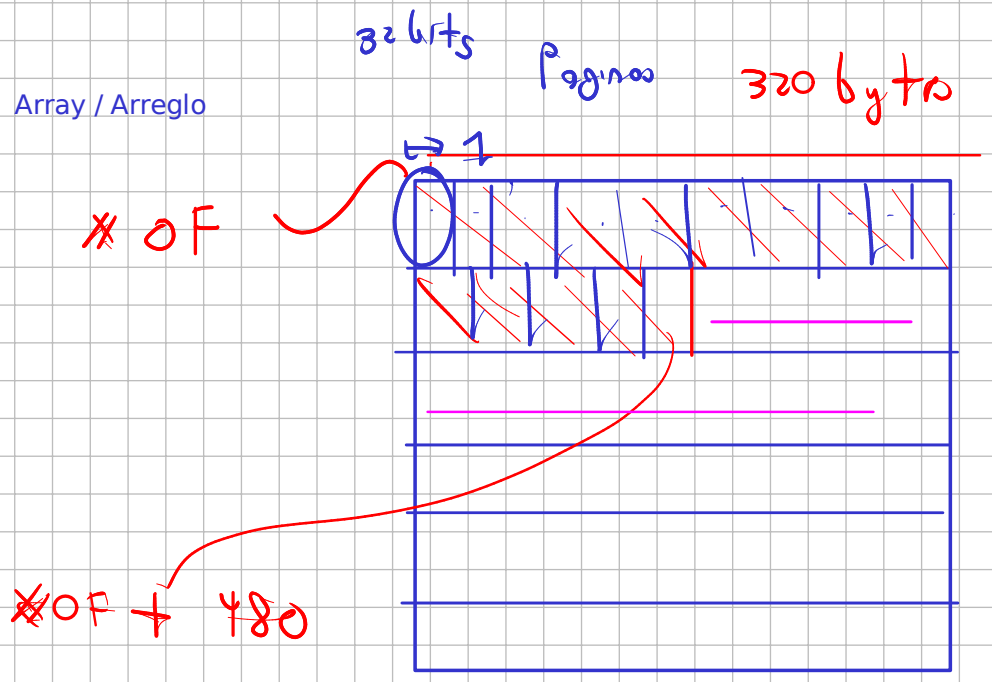
Estructuras de datos

Indique un algoritmo para cada operación básica, acompañado de su respectiva complejidad usando la notación O

- ENQUEUE(Q,x)
- DEQUEUE(Q)



Array / Arreglo



int [15] arr

Estático

-2^{31} $+2^{31}-1$

-2048 2048

arr = #0f

#0f + 0

arr[0]

arr[1]

#0f + 1x32

arr[5] arr[100]

arr[10]

O(1)

pos + tipo

Operaciones en las estructuras de datos

- Inplace: La operación MODIFICA la estructura directamente, usualmente NO RETORNA un valor

```
lst = [1,2,3,4]
lst.append(5) #Inserto al final el 5 se modifica lst

lstnew = lst.append(10) #lstnew = nulo
```

- Valor

La operacion NO MODIFICA la estructura directamente, pero nos puede RETORNAR una copia MODIFICADA

Estructuras de datos SECUENCIALES / INDEXABLES

arreglo[10]

tabla["f"]

ORDENADAS

Estructuras de datos que son SECUENCIALES o INDEXABLES

Necesitamos iteradores

Conjuntos (No tienen orden)

Conjuntas:

Son una estructura de datos en el cual

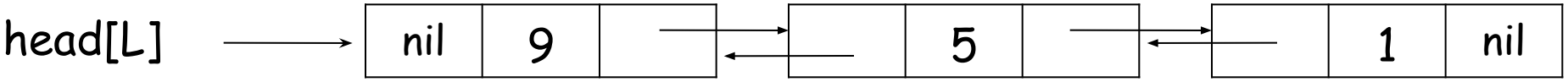
1) NO SE PUEDEN REPETIR LOS ELEMENTOS

2) NO TIENEN ORDEN

Estructuras de datos

Listas doblemente enlazadas

Es una estructura de datos en la cual los objetos son organizados en un orden lineal. A diferencia de los arreglos, el orden en las listas está dado por un puntero a cada objeto

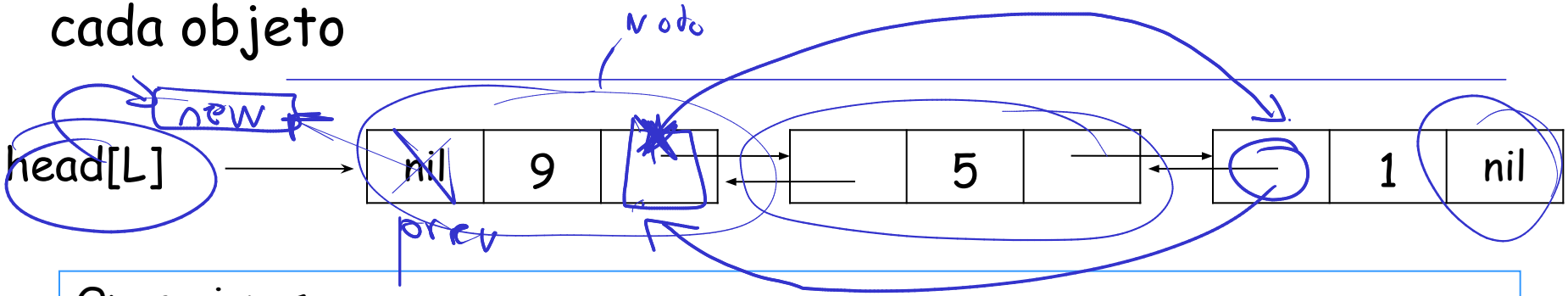


- Cada nodo en una lista doblemente enlazada tiene 3 campos: prev, key y next
- Se tiene además un puntero al primer nodo

Estructuras de datos

Listas doblemente enlazadas

Es una estructura de datos en la cual los objetos son organizados en un orden lineal. A diferencia de los arreglos, el orden en las listas está dado por un puntero a cada objeto



Operaciones

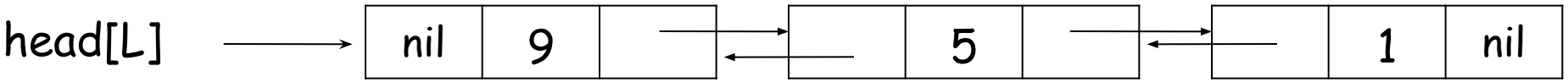
- LIST-INSERT(L,x): inserta x en la cabeza de la lista. x es un nodo tal que key[x]=k, y prev=next=nil
- LIST-DELETE(L,x): donde x es el nodo que se desea borrar
- LIST-SEARCH(L,k): busca el primer nodo que tiene llave k y retorna un puntero a ese nodo

Estructuras de datos

LIST-SEARCH busca el primer nodo que tiene llave k y retorna un puntero a ese nodo

```
LIST-SEARCH(L,k)
1.  x ← head[L]
2.  while x≠nil and key[x]≠k
3.        x ← next[x]
4.   return x
```

¿Cuál es la complejidad en el peor caso?

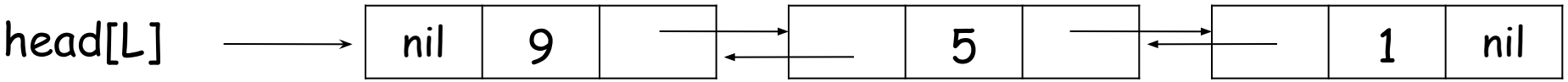


Estructuras de datos

LIST-SEARCH busca el primer nodo que tiene llave k y retorna un puntero a ese nodo

```
LIST-SEARCH(L,k)
1.  x ← head[L]
2.  while x≠nil and key[x]≠k
3.        x ← next[x]
4.   return x
```

En el peor caso será O(n)



Estructuras de datos

Indique el resultado de realizar las siguientes operaciones:

prev[z]=nil

next[z]=nil

key[z]=10

LIST-INSERT(L,z)

prev[w]=nil

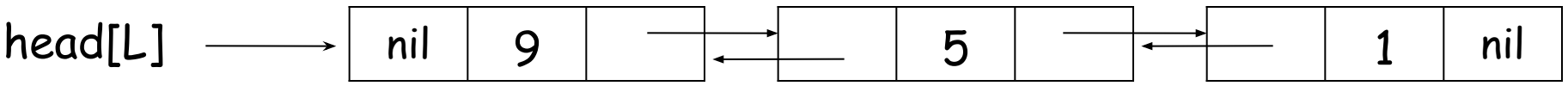
next[w]=nil

key[w]=8

LIST-INSERT(L,w)

x=LIST-SEARCH(L,10)

LIST-DELETE(L,x)



Estructuras de datos

Indique el algoritmo para las siguiente operaciones y muestre su complejidad en el peor caso:

- LIST-INSERT(L,x): inserta x en la cabeza de la lista. x es un nodo tal que key[x]=k, y prev=next=nil $O(1)$

- LIST-DELETE(L,x): donde x es el nodo que se desea borrar $O(n)$

Search(L, k) $O(n)$

LinkedList -> List doblemente enlazada, NO NECESITA SER CONTINUA EN MEMORIA

Cuando usted insert solamente se preocupa por buscar donde puede incluir el nuevo nodo

ArrayList / Vector

Array dinamico, NECESITA SER CONTINUO

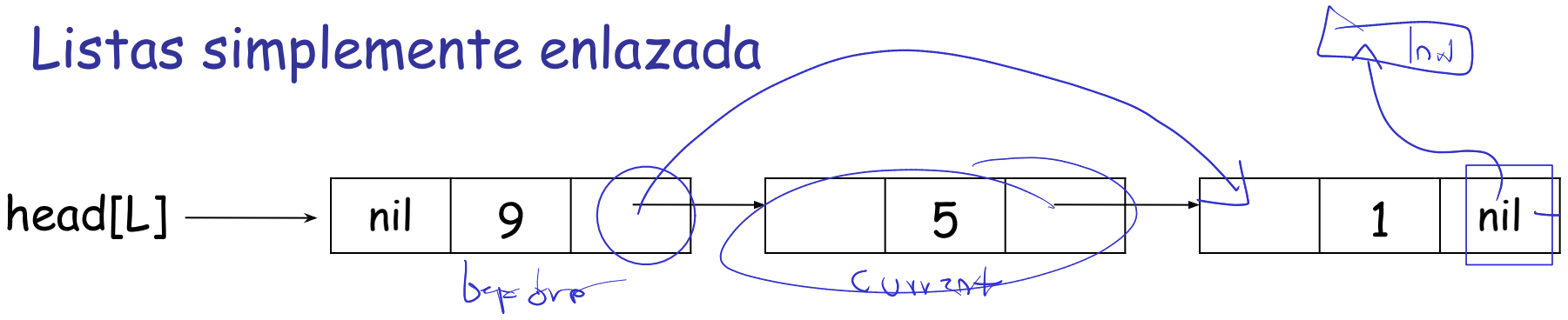
Si no cabe en la posición actual de memoria, es necesario moverlo / relocalizarlo los que es COSTOSO

Usualmente tienen una capacidad de reserva
ArrayList + 10

Vector 1x el tamaño del arreglo

Estructuras de datos

Listas simplemente enlazada



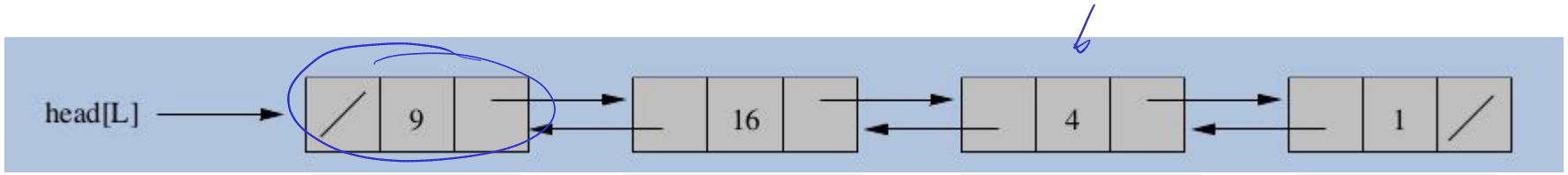
Operaciones

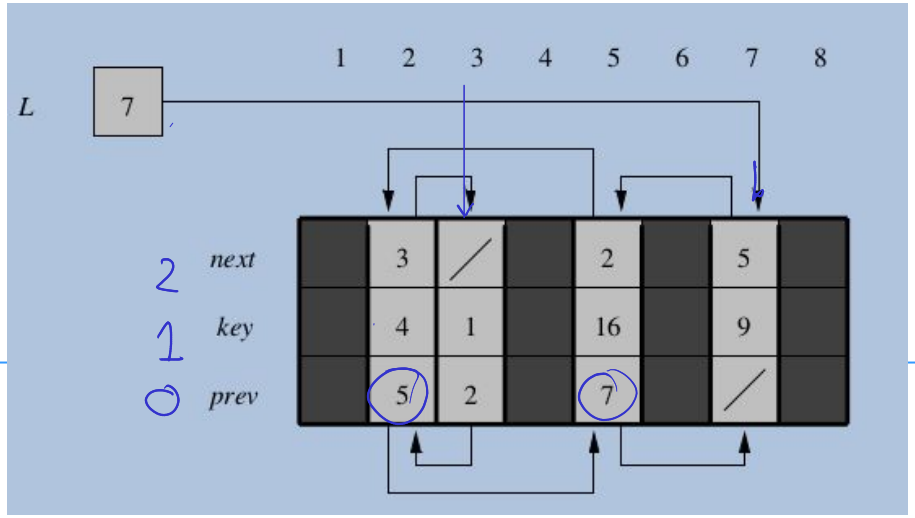
- LIST-INSERT(L,x): inserta x al final de la lista. x es un nodo tal que key[x]=k, y prev=next=nil
- LIST-DELETE(L): donde x es el nodo al final de la lista
- LIST-SEARCH(L,k): busca el primer nodo que tiene llave k y retorna un puntero a ese nodo

Estructuras de datos

Apuntadores y objetos (1)

Una colección de objetos que tienen los mismos campos se puede representar usando un arreglo para cada campo (representación con múltiples arreglos).

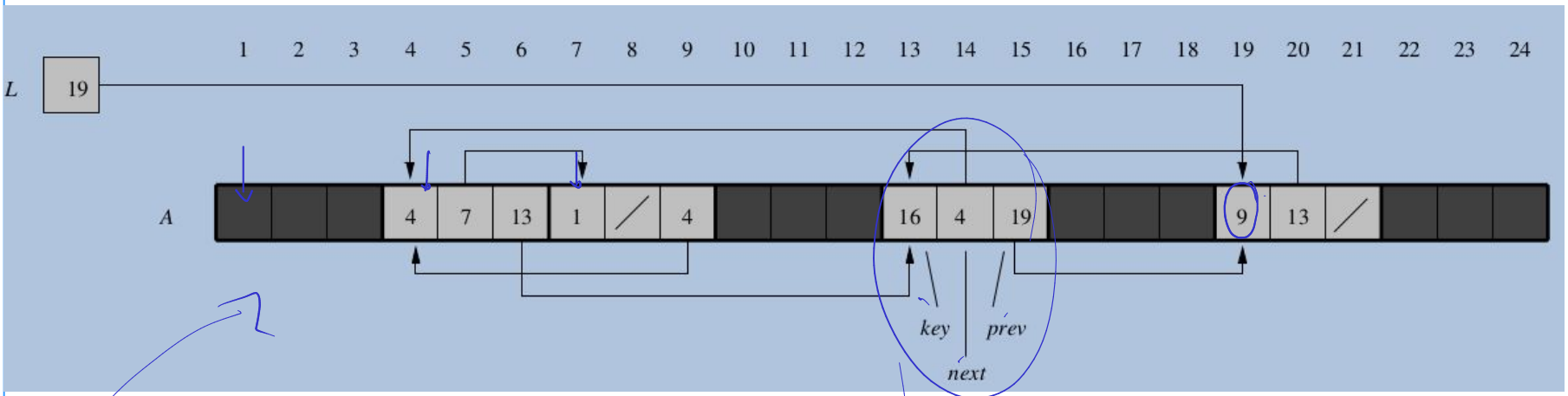




Estructuras de datos

Apuntadores y objetos (2)

Se puede usar un solo arreglo para representar los objetos.

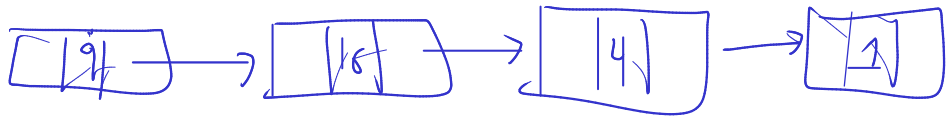


3i+1 = key

3i+2 = next

3i+3 = prev

i = índice / iterador



Estructuras de datos

Árboles con raíz

Cada nodo tiene los campos p, left y right para almacenar los punteros al padre, hijo izquierdo e hijo derecho. Además, se tiene el campo key.

Si p[x]=nil entonces x es a raíz

Si el nodo x no tiene hijo izquierdo entonces left[x]=nil

Si left[x]=right[x]=nil entonces x es una hoja

Estructuras de datos

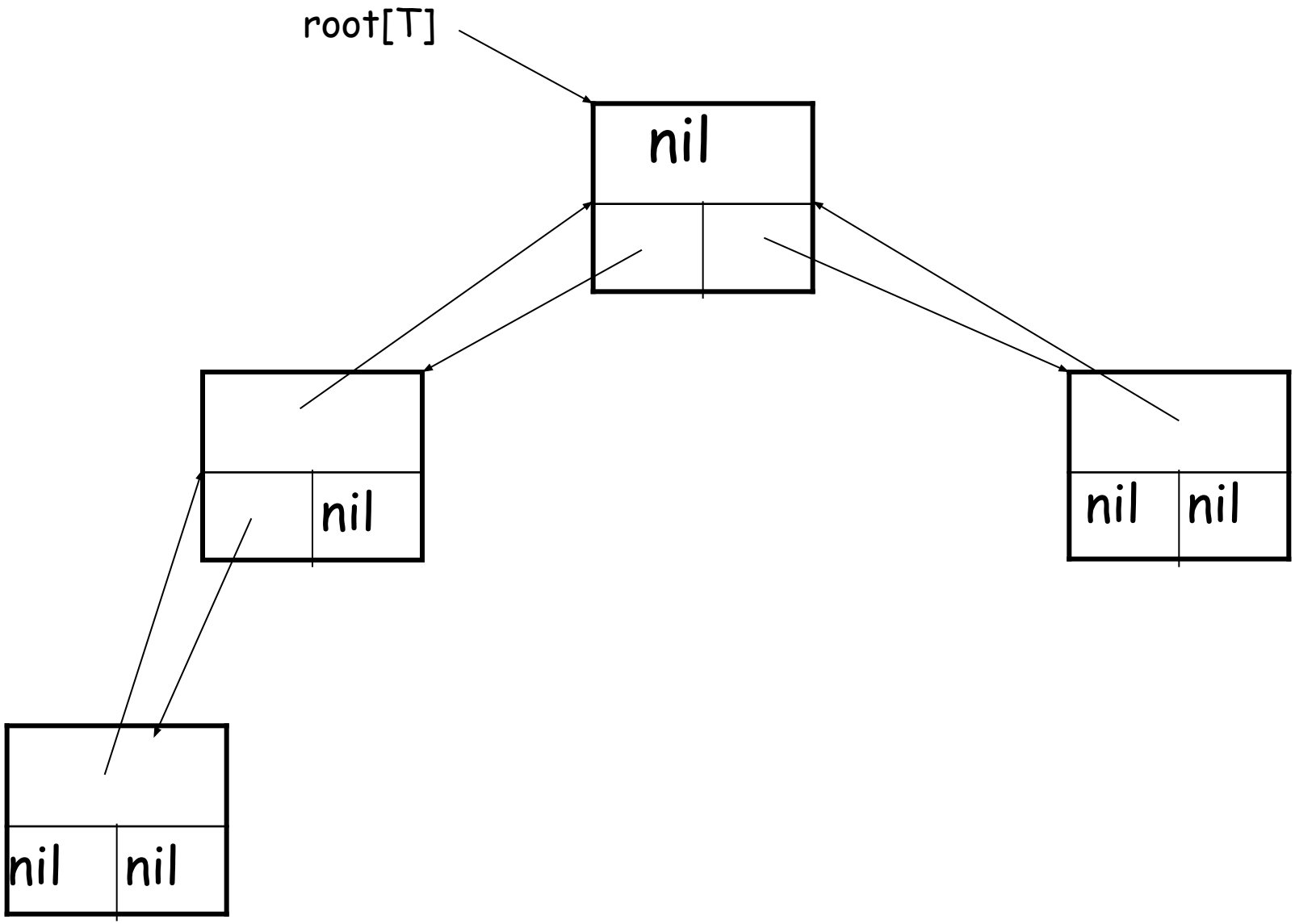
Árboles con raíz

Cada nodo tiene los campos p, left y right para almacenar los punteros al padre, hijo izquierdo e hijo derecho. Además, se tiene el campo key.

root[T] es el apuntador a la raíz del árbol

Si root[T]=nil entonces el árbol está vacío

Estructuras de datos



Referencias

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press. Chapter 10

Gracias

Próximo tema:

Estructuras de datos: Tablas Hash