Agenda

Linked Lists

Fundaments of the Linked Lists

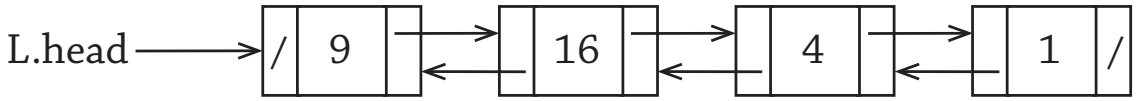
- A linked list is a data structure in which the object are arranged in a linear order. Unlike an array, however, in which the linear order is determined by the array indices, the order in a linked list is determined by a pointer in each object.

- Linked lists provide a simple, flexible representation for dynamic sets, supporting (though not necessarily efficiently) all the operations listed in.

- A list may have one of several forms. It may be either singly linked or doubly linked, it may be sorted or not, and it may be circular or not.

Linked Lists
Doubly Linked Lists

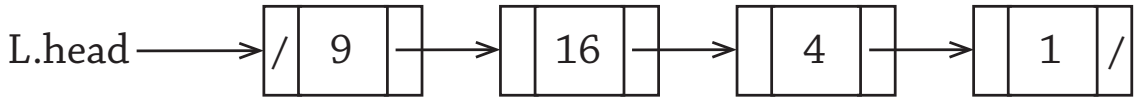
- A doubly linked list L representing the dynamic set {1, 4, 16, 9} is given in the figure below. Each element of a doubly linked list L is an object with an attribute key and two other pointer attributes: next and prev(shown by arrows). The object may also contain data.



- Given an element x in the list, x.next points to its successor in the linked list, and x.prev points to its predecessor. If x.prev=NIL, the element x has no predecessor and is there fore the first element, or head of the list.

- If x.next=NIL, the element x has no successor and is there fore the last element, or tail, of the list. An attribute L.head points to the first element of the list. If L.head=NIL, the list is empty.

- A singly linked list L representing the dynamic set {1, 4, 16, 9} is given in the figure below. Each element of a singly linked list L is an object with an attribute key and a pointer attribute: next (shown by arrows). The object may also contain data.



- If x.next=NIL, the element x has no successor and is therefore the last element, or tail, of the list. An attribute L.head points to the first element of the list. If L.head=NIL, the list is empty.

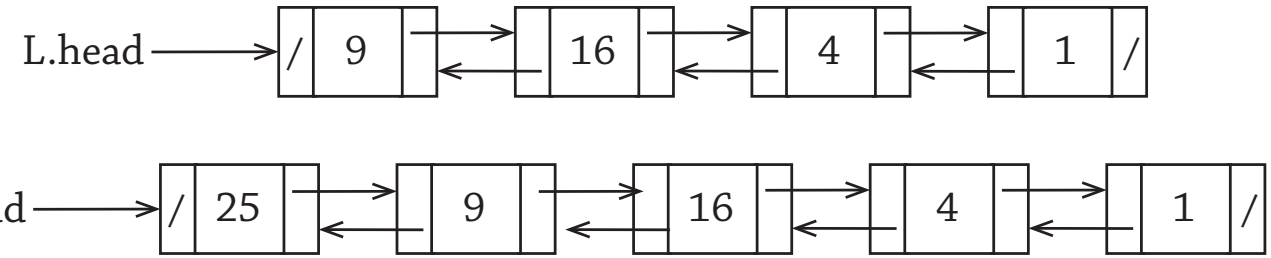
Inserting Into a Linked List

- We assume that the list with which we are working is a unsorted and doubly linked.
- Given an element x whose key attribute has already been set, the LIST-INSERT procedure splices x onto the front of the linked list.

LIST-INSERT(L, x)

```
1  x.next = L.head
2  if L.head ≠ NIL
3     L.head.prev = x
4  L.head = x
5  x.prev = NIL
```

- Following the execution of LIST-INSERT(L, x), where x.key = 25, the linked list has a new object with key 25 as the new head. This new object points to the old head with key 9. In the figure below, the new situation of the linked list is illustrated.



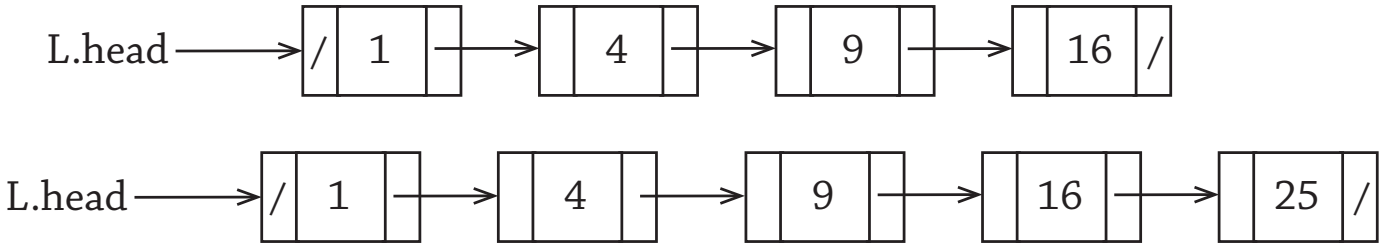
Linked Lists
Inserting Into a Linked List

- We assume that the list with which we are working is sorted and singly linked.
- Given an element x whose key attribute has already been set, the LIST-INSERT procedure finds the appropriate position by considering the key value of the x for the linked list.

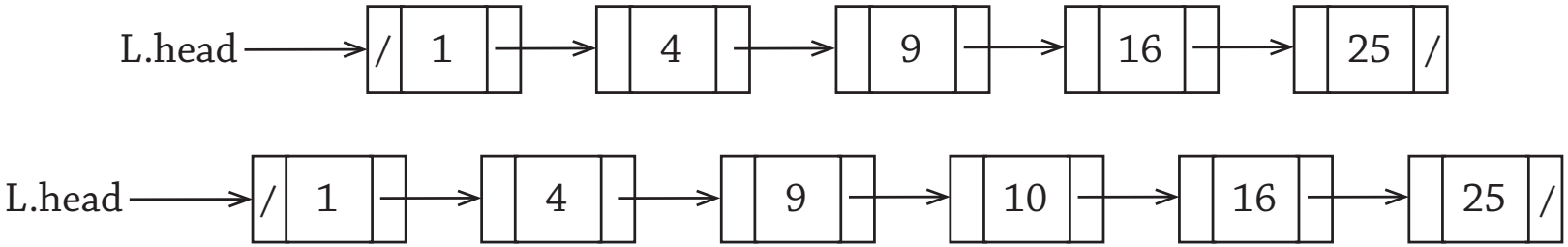
```
LIST-INSERT( L, x )
1  previous = NIL
2  current = L.head
3  while current ≠NIL and x.key > current.key
4     previous = current
5     current = current.next
6  if previous = NIL
7     x.next = L
8     L.head = x
9  else
10    previous.next = x
11    x.next = current
```

Inserting Into a Linked List

- Following the execution of LIST-INSERT(L, x), the new object with the key 25 will be inserted to the linked list. This new object becomes to the tail of the linked list. The object with the key 16 points to the newly added object with the key 25. In the figure below, the new situation of the linked list is illustrated.



- Following the execution of LIST-INSERT(L, x), the new object with the key 10 will be inserted to the linked list. Corresponding position for the object with key 10 will be determined as the location between the objects with the keys 9 and 16. In the figure below, the new situation of the linked list is illustrated.

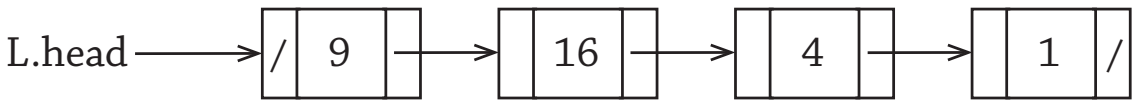


Searching a Linked List

- We assume that the list which we are working are unsorted and singly-doubly linked.
- The procedure LIST-SEARCH(L, k) finds the first element with key k in the list L by simple linear search, returning a pointer to this element. If no object with key k appears in the list, then the procedure returns NIL.

```
LIST-SEARCH( L, k )
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3     x = x.next
4  return x
```

- To search a list of n object, the LIST-SEARCH(L, k) procedure takes O(n) time in the worst case, since it may have to search the entire list. For a linked list in the figure below, the call LIST-SEARCH(L, 7) returns NIL.

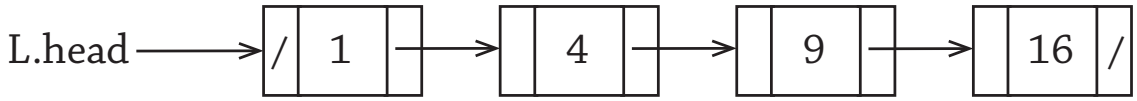


Linked Lists
Deleting from a Linked List

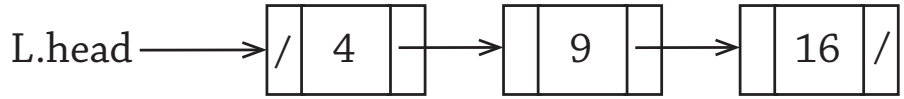
- We assume that the list which we are working are unsorted and singly linked.
- If we wish to delete an element with a given key, we must first call LIST-SEARCH(L, k) first to retrieve a pointer to the element. If we wish to delete an element with a given key, the running time of the procedure DELETE-LIST(L, k) on a list of n elements is O(n).

```
LIST-DELETE( L, k )
1  if LIST-SEARCH( L, k ) ≠ NIL
2    previous = NIL
3    current = L.head
4    while current.key ≠ k
5      previous = current
6      current= current.next
7    if previous = NIL //Deletion of the head of the list
8      L.head = current.next
9    else
10     previous.next = current.next
```

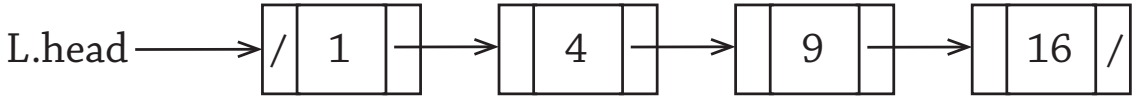
- Following the execution of LIST-DELETE(L, x), the object with the key 1 will be removed from the linked list. In the figure below, the new situation of the linked list is illustrated.



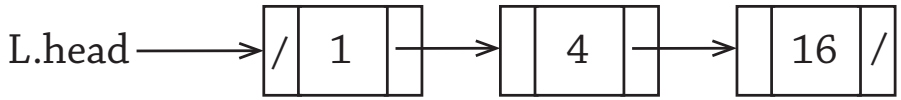
Step-1. current.key = 1, previous = NIL and *if current.key = 1* condition is satisfied.
Result. L.head = current.next



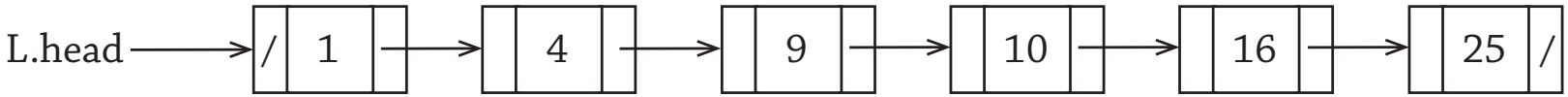
- Following the execution of LIST-DELETE(L, x), the object with the key 9 will be removed from the linked list. In the figure below, the new situation of the linked list is illustrated.



Step-1. current.key = 1, previous = NIL and *if current.key* = 9 condition is not satisfied.
Step-2. current.key = 4, previous.key = 1 and *if current.key* = *9* condition is not satisfied.
Step-3. current.key = 9, previous.key = 4 and *if current.key* = *9* condition is satisfied.
Result. previous.next = current.next



- Following the execution of LIST-DELETE(L, x), the object with the key 25 will be removed from the linked list. In the figure below, the new situation of the linked list is illustrated.



Step-1. current.key = 1, previous = NIL and *if current.key* = 25 condition is not

Step-2. current.key = 4, previous.key = 1 and *if current.key = 25* condition is not satisfied.

Step-3. current.key = 9, previous.key = 4 and *if current.key = 25* condition is not satisfied.

Step-4. current.key = 10, previous.key = 9 and *if current.key = 25* condition is not

Step-5. current.key = 16, previous.key = 10 and *if current.key = 25* condition is not satisfied.

Step-6. current.key = 25, previous.key = 16 and *if current.key = 25* condition is satisfied.

Result. current.next = NIL, previous.next = current.next