Data Structures and Algorithms

Lecture 3 – Lists

Miao Zhang

Lists

Main content

- Abstract data type of Lists
- Array-based Lists
- Linked Lists
- Stacks
- Queues
- String
- Application

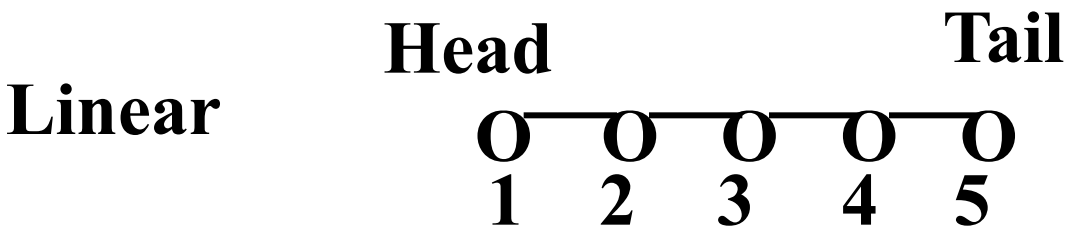
Consider Every Day Lists

- Groceries to be purchased
- Job to-do list
- List of assignments for a course
- Dean's list

- Can you name some others??

Lists

Characters of linear structure:



- **Only one head item/element**
- **Only one tail item/element**
- **One direct previous item/element (except Head)**
- **One next item/element (except Tail)**

Lists

- A list is a finite, ordered sequence of data items.
- Important concept: List elements have a position.
- Notation: $(a_1, \ldots, a_{i-1}, a_i, \ldots, a_n)$

Where a_1 is the head, a_n is the tail,
a_{i-1} is the previous element of a_i
a_i is the next element of a_{i-1}

When $2 \le i \le n$, a_i has only one previous element

When $1 \le i \le n-1$, a_i has only one next element

Properties of Lists

- Can have a single element
- Can have no elements
- There can be lists in lists

- We will look at the list as an abstract data type
  - Homogeneous
  - Finite length
  - Sequential elements

Basic Operations

- **Construct an empty list**
- **Determine whether or not empty**
- **Insert an element into the list**
- **Delete an element from the list**
- **Traverse (iterate through) the list to**
  - Modify
  - Output
  - Search for a specific value
  - Copy or save
  - Rearrange

Designing a `List` Class

- Should contain at least the following function members
  - Constructor
  - `isEmpty()`
  - `insert()`
  - `delete()`
  - `display()`
- Implementation involves
  - Defining data members
  - Defining function members from design phase

Abstract Data Type of Lists

Data set: $\{ a_0, a_1, \ldots, a_{n-1} \}$, a_i **is Data Type**

operations:

(1) Clear()

(2) Append(T value)

(3) Insert(int p, T value)

(4) Delete(int p)

(5) GetValue(int p, T& value)

(6) SetValue(int p, T value)

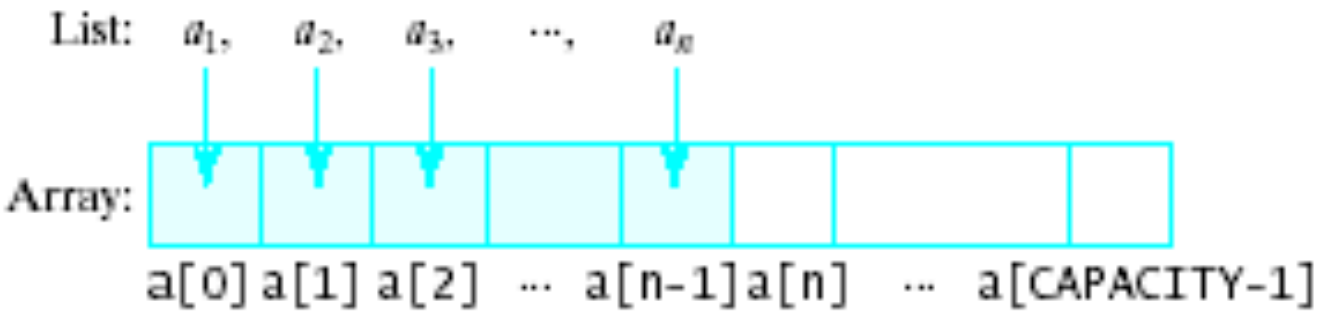
(7) GetPos(int p, T value)

Approaches to Implement Lists

- **Sequential structure**
  - Array-Based List
- **Linked structure**
  - Linked List

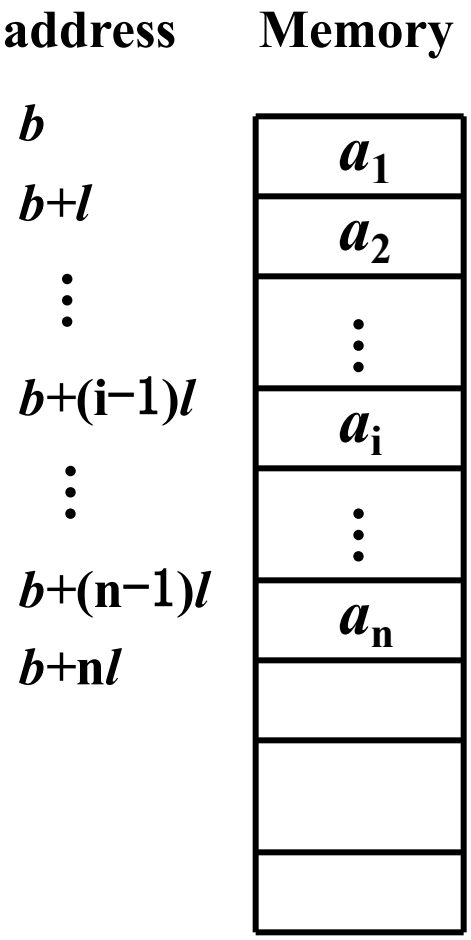
Array-Based Lists

- An array is a viable choice for storing list elements
  - Element are sequential
  - It is a commonly available data type
  - Algorithm development is easy
- Normally sequential orderings of list elements match with array elements



Array-Based List

A consecutive storage.



The memory size of each element is l

$LOC(a_i)$ is the memory address of a_i

$LOC(a_1)$ is the memory address of first element a_1, also the head of the list

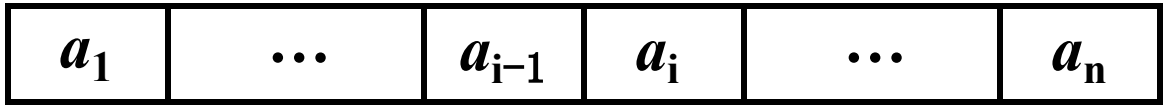
$$LOC(a_{i+1}) = LOC(a_i) + l$$

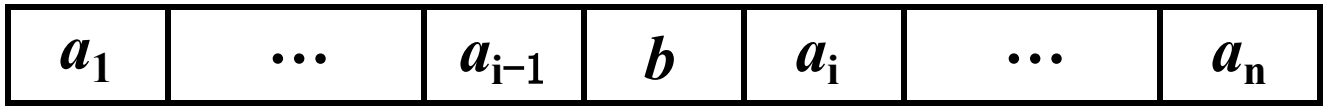
$$LOC(a_i) = LOC(a_1) + (i - 1)l$$

Implementing Operations (Insert)

INSERT

EX.， Insert b in front of a_i



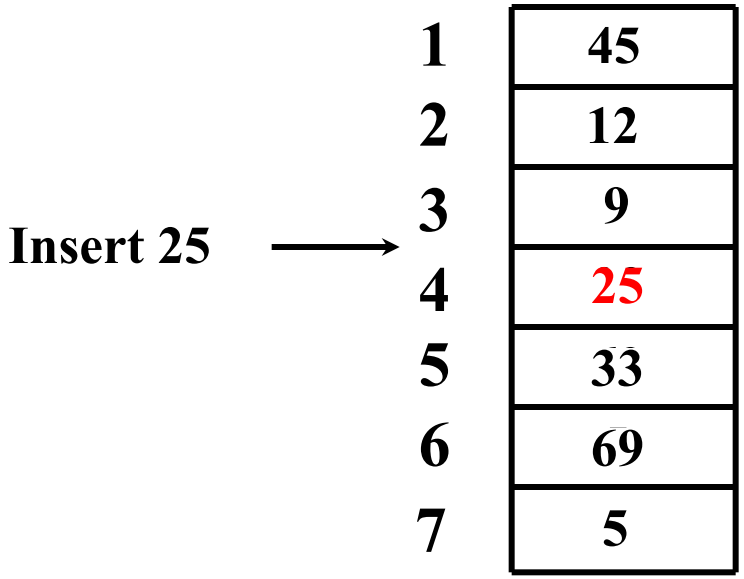


Process:

1. Shift the elements(i_{th}…n_{th}) right of insertion point
2. Insert the new element at the i_{th} position

Implementing Operations (Insert)

Ex，Insert 25 in front of the forth element



Implementing Operations (Insert)

The time complexity of insertion:

Under different conditions

i=1， move n items；

i=n+1， move 0 items；

i=i， move n – i +1 items；

Implementing Operations (Insert)

Suppose p_i is the probability of inserting a new element in front of the i_{th} item

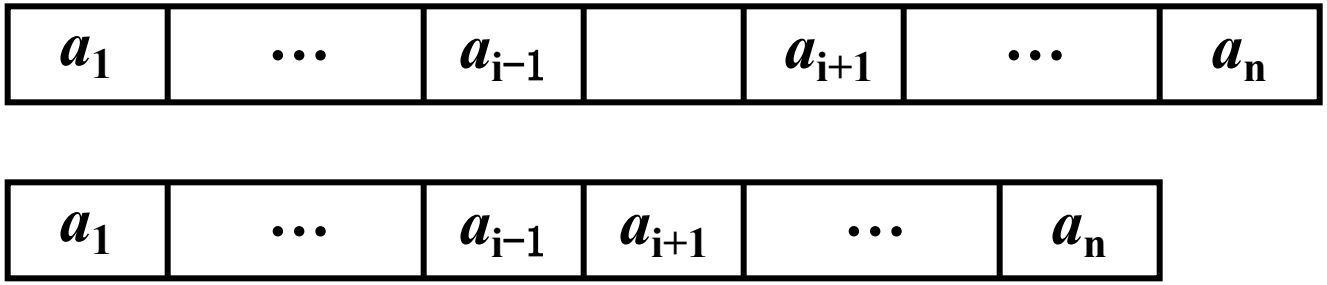
The **expectation** number of movement for inserting a new element in a list (length=n) $E_{is} = \sum_{i=1}^{n+1} p_i (n - i + 1)$

Suppose the probability of inserting an element at each position is **equal**, $p_i = \frac{1}{n+1}$

$$E_{is} = \frac{1}{n+1} \sum_{i=1}^{n+1} (n - i + 1) = \frac{n}{2} \qquad O(n)$$

Implementing Operations (Delete)

EX.2.5 Delete the i_{th} item

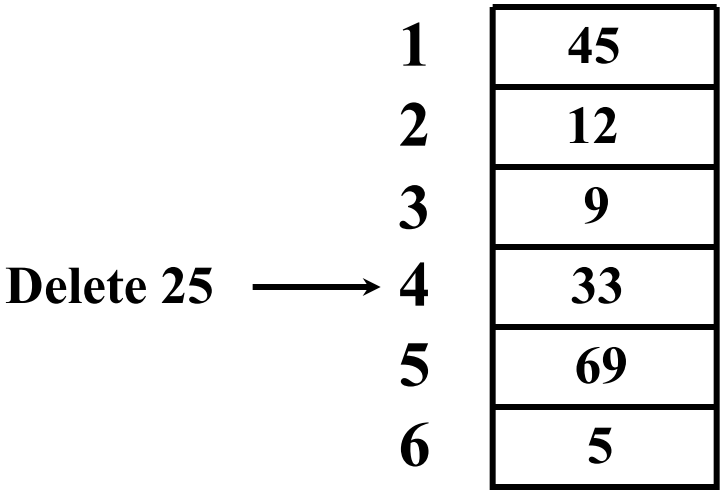


Process:

1. Delete the i_{th} element
2. Shift the elements($i+1_{th}$…n_{th}) back

Implementing Operations (Delete)

EX.， Delete the 4^{th} item 25



Implementing Operations (Delete)

The time complexity of Delete:

The number of shifted elements depends on the position of the deleted element

i=1， move n -1 elements；

i=n， move 0 element；

i=i， move n – i elements；

Implementing Operations (Delete)

Suppose q_i is the probability of delete the ith item

The expectation number of movement for deleting an item in a list (length=n) : $E_{dl} = \sum_{i=1}^{n} q_i (n - i)$

Suppose the probability of deleting an item at each position is equal， $q_i = \frac{1}{n}$

$$E_{dl} = \frac{1}{n} \sum_{i=1}^{n} (n - i) = \frac{n - 1}{2} \qquad O(n)$$

The time complexity of operations

- Insert an item : $O(n)$,
- Delete an item : $O(n)$,
- Get value of an item : $O(1)$
- Set value of an item : $O(1)$

The characteristics of array-based lists

Advantages:

- **Access elements easily and efficiently;**
- **Get the length of list directly**

Disadvantages:

- **Predefine the maximum size of the list**
- **Inserting and deleting is difficult, both of them need to shift elements**

Multidimensional Array

- Array is the sequential list with fixed size and element type
  - Setting the size and data type when the static array is defined
  - Allocate the memory space for dynamic array during the program running

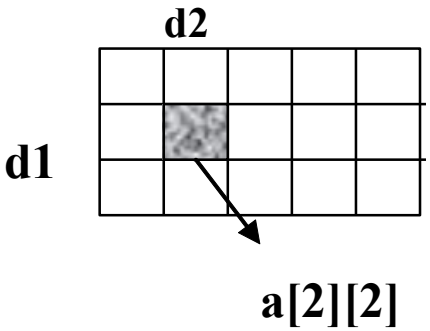
Multidimensional Array

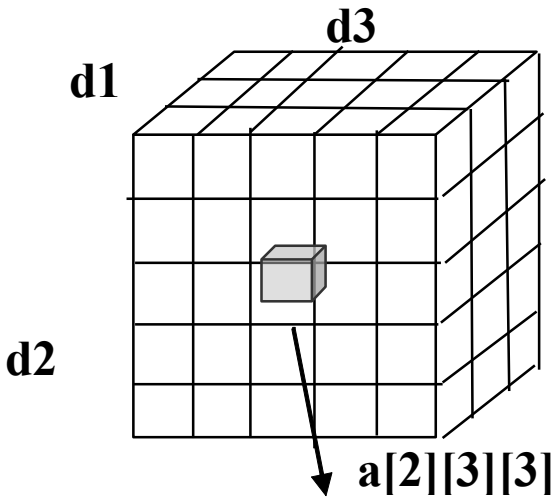
- Multi-array is the extended vector
  - The multi-array is made up of the vector of vector
  - It is represented as:

    ELEM $A[c_1..d_1][c_2..d_2]...[c_n..d_n]$
  - The number of items is:

$$\prod_{i=1}^{n}(d_i - c_i + 1)$$

Structure of array





Two-dimension array Three-dimension

d1[1..3],d2[1..5],d3[1..5]is three dimensions

The storage of array

- Logical characters of multi-array:

  One element could has many direct previous elements and next elements

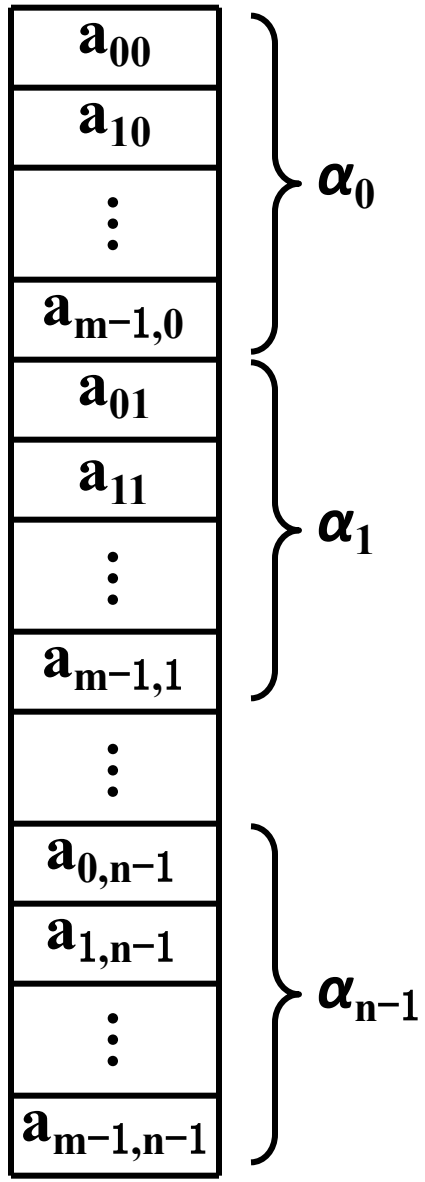
- The memory is one-dimensional. Therefore, the storage of array is one-dimension.
  - Row priority
  - Column priority

$$X=\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

Column priority

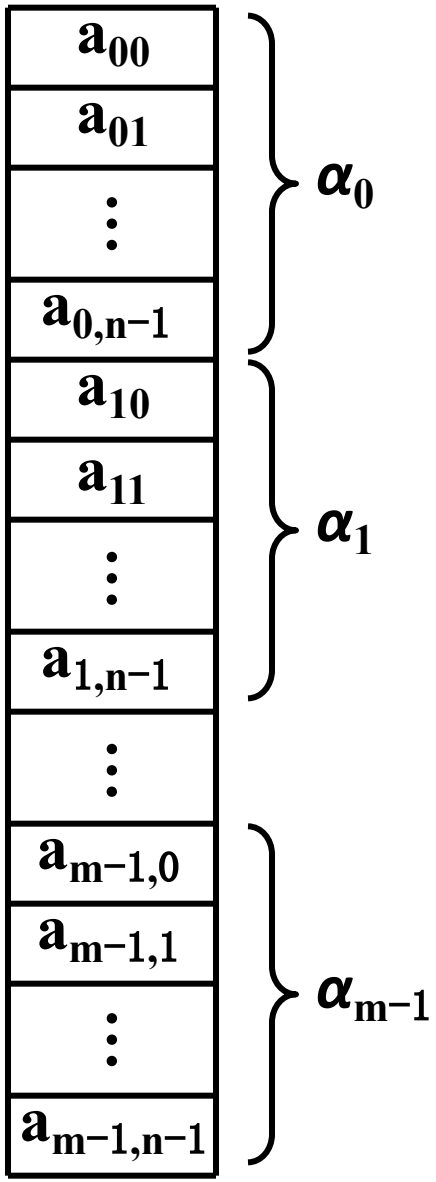
$A_{m\times n}$

$$\left[\begin{bmatrix} a_{00} \\ a_{10} \\ \vdots \\ a_{m-1,0} \end{bmatrix} \begin{bmatrix} a_{01} \\ a_{11} \\ \vdots \\ a_{m-1,1} \end{bmatrix} \begin{bmatrix} a_{02} \\ a_{12} \\ \vdots \\ a_{m-1,2} \end{bmatrix} \cdots \begin{bmatrix} a_{0,n-1} \\ a_{1,n-1} \\ \vdots \\ a_{m-1,n-1} \end{bmatrix}\right]$$

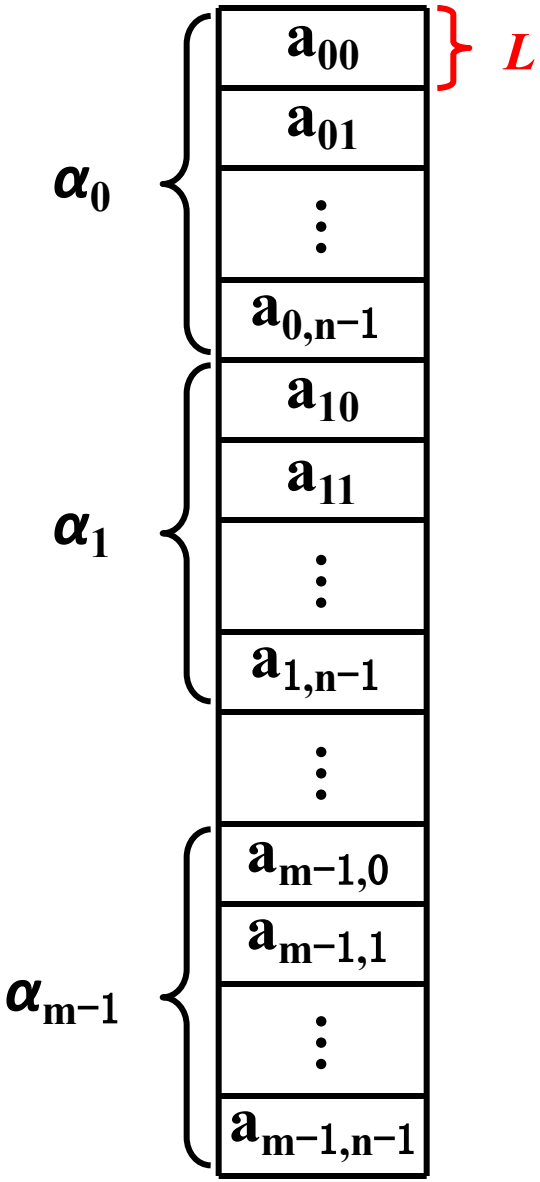


Row priority $A_{m\times n}$

$$\begin{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \end{bmatrix} \\ \begin{bmatrix} a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \end{bmatrix} \\ \vdots \quad \vdots \quad \vdots \quad \quad \vdots \\ \begin{bmatrix} a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{bmatrix} \end{bmatrix}$$



How to calculate the memory address of the item in the array



Suppose it is row priority

Array : A(m×n)

The memory address of a_{ij} is

$$LOC(i, j) = LOC(0, 0) + (n \times i + j)\, L$$

Where, LOC(0, 0) is the memory address of a_{00};

L is the length of an item in the memory

EX, $LOC(1, 1) = LOC(0, 0) + (n \times 1 + 1)\, L$

Recall Inefficiency of Array-Implemented List

- `insert()` and `delete()` functions inefficient for dynamic lists
  - Those that change frequently
  - Those with many insertions and deletions

So …

We look for an alternative implementation.

Linked List

For the array-based implementation:

1. First element is at location 0
2. Successor of the item at location i is `i + 1`
3. End is at location `size - 1`

Fix:

1. Remove requirement that list elements should be stored in consecutive location.
2. But then need a "link" that connects each element to its successor

Linked Lists !!

Linked List

The characterictics of linked structure:

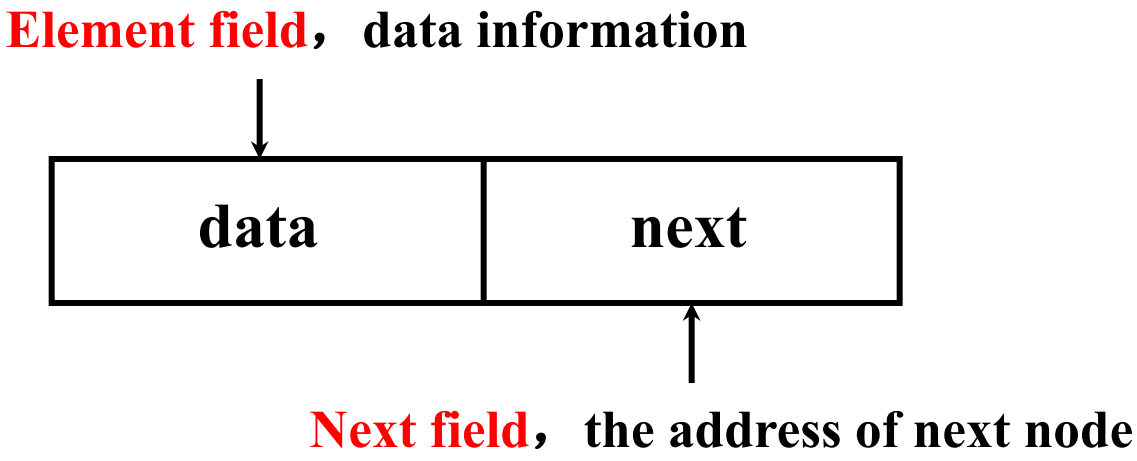
- The memory units for data elements are selected randomly.
- The memory units could be sequential or not.

Single Linked List

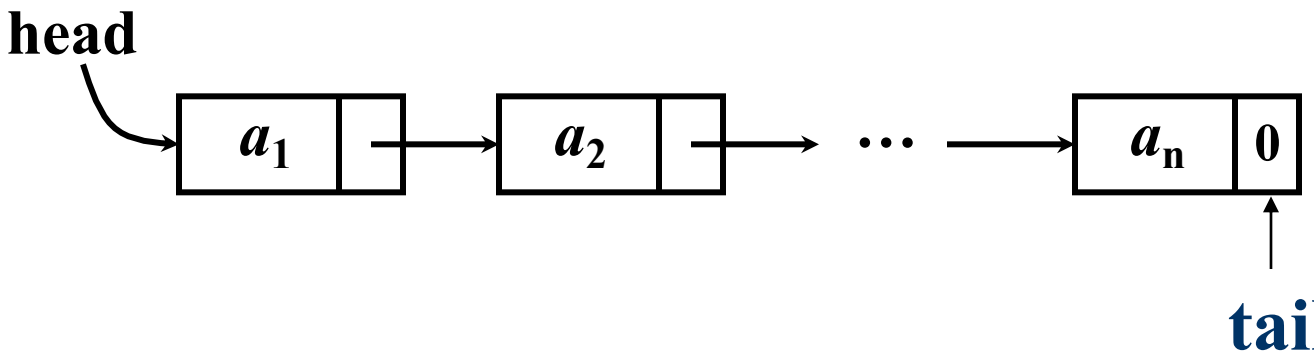
Node: Two parts，

element field for data information，

next field for the address of direct next node.



Single Linked List

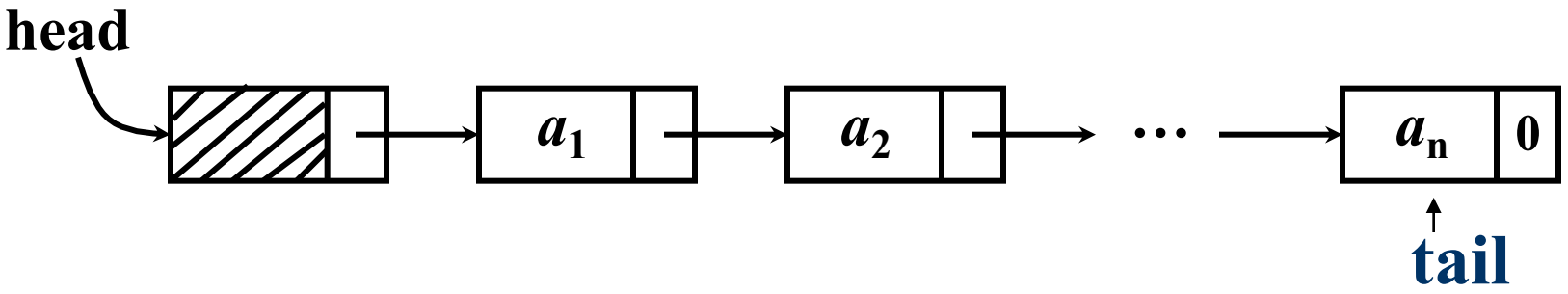


head: head pointer， points to the first node of linked list。

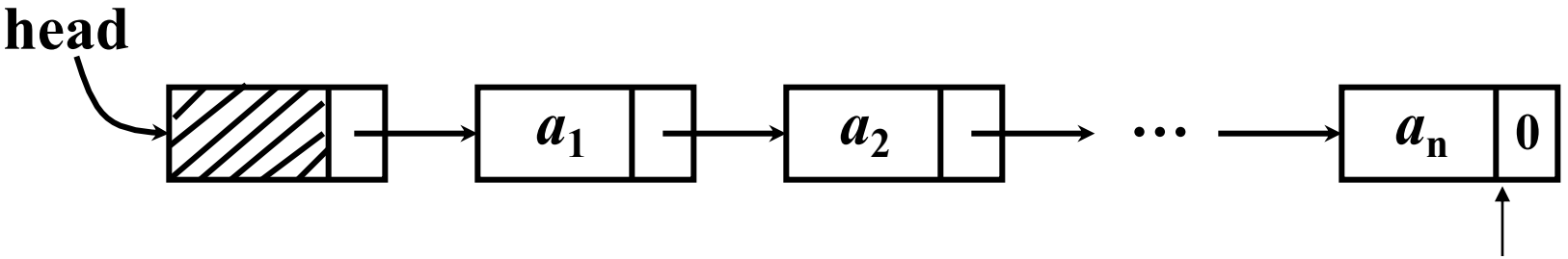
Tail: tail pointer, points to the last node of the linked list

0: Null， or “∧”

Head node: contain some information of list (length)

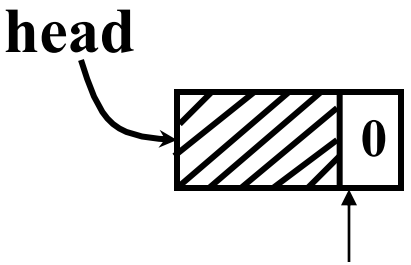


Single Linked List




```
typedef   struct   LinkNode  {
        ELEM          data ;
        ListNode      * next ;
} ;
```

Null list:



Single Linked List

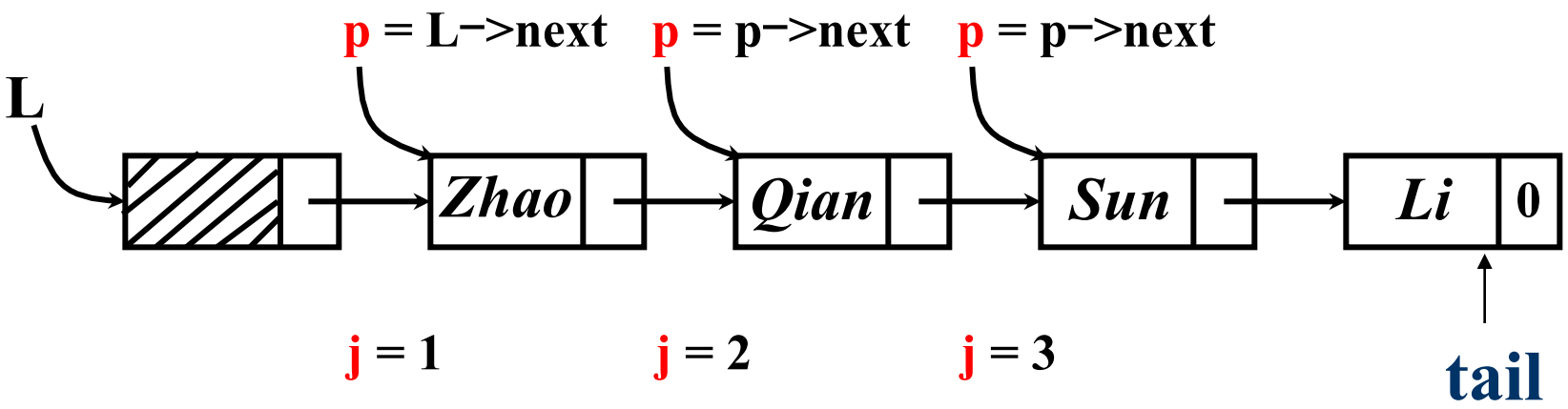
Disadvantages:

- Not able to randomly access an arbitrary data in the linked list
- Not able to directly acquire the length the the linked list

Advantages:

- Efficient to implement insert/delete operations

EX，get the third element



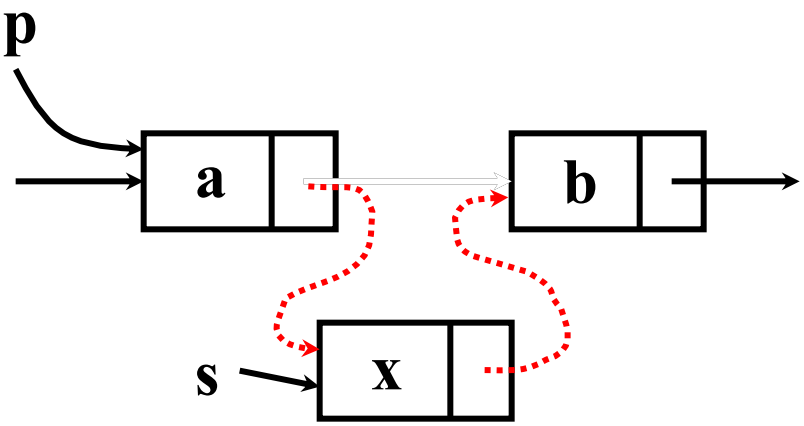
e = p->data = *Sun*

Time complexity: *O*(n)

Single Linked List

Advantages: **Efficient for inserting and deleting**

Insert x between a and b:

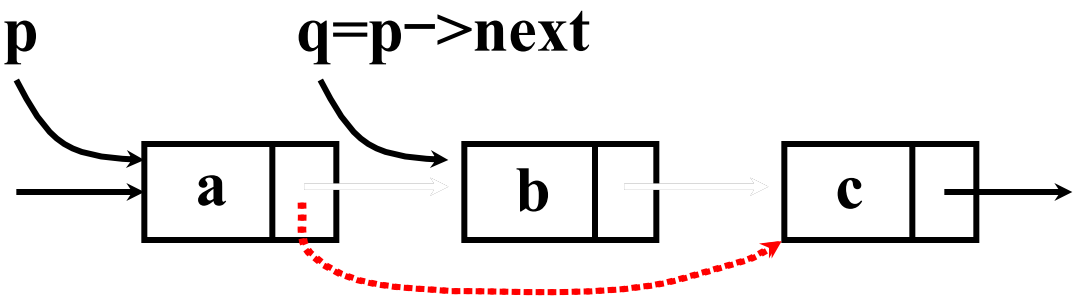


s->next = p->next

p->next= s

Single Linked List

Delete item b :



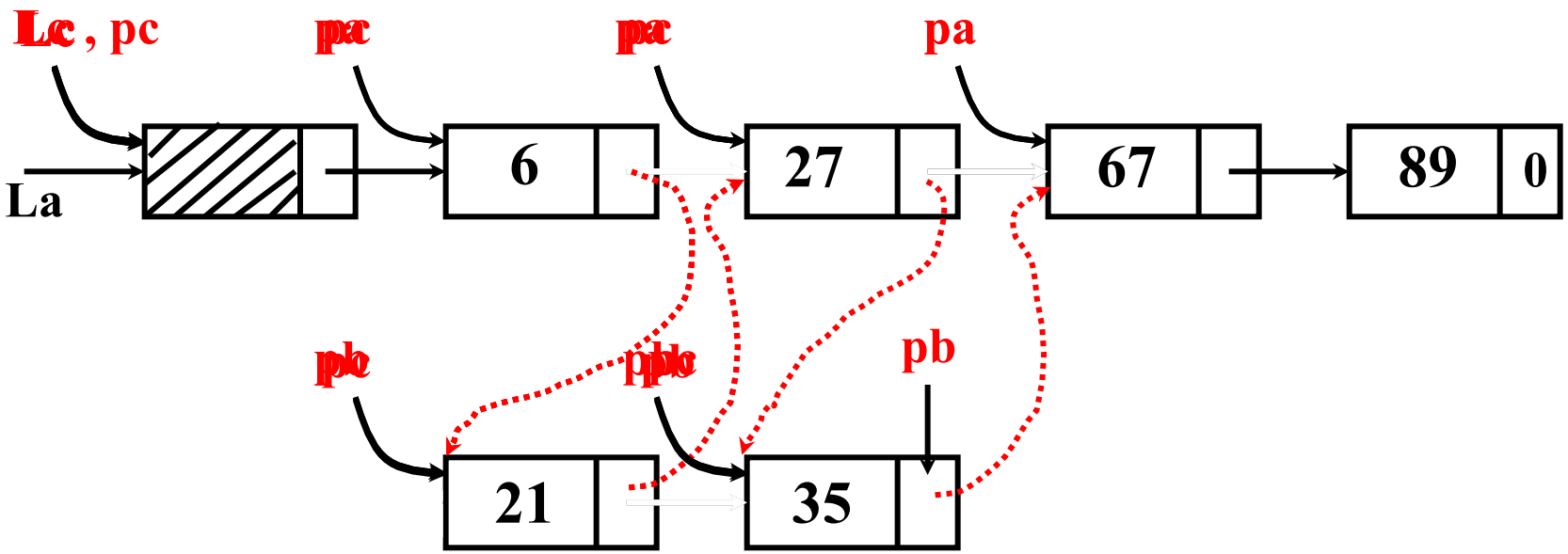
1) **p ->next = p ->next ->next**

2) **q = p->next**

p->next = q->next

The operations of single linked list

- **Merge two singe linked lists to an ordered linked lists**
  - **shift the pointer by comparing the value**

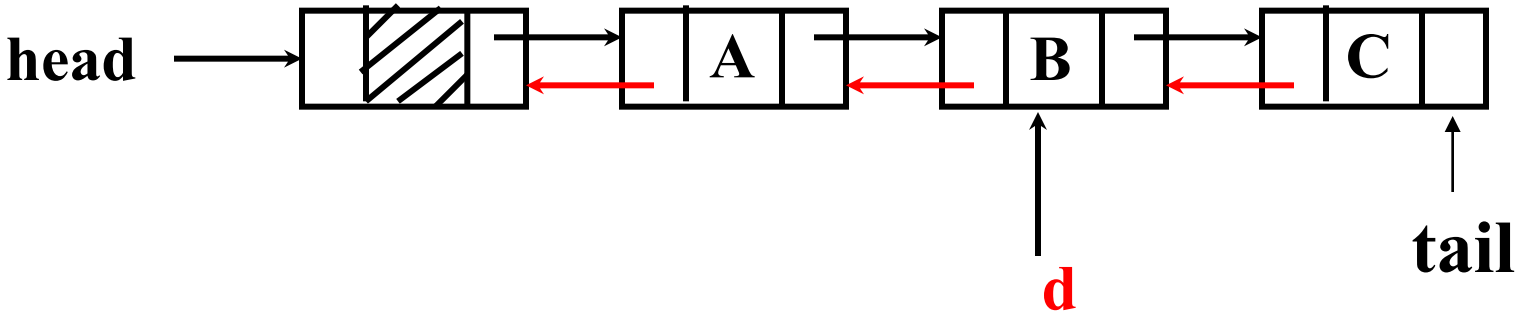


```
void   MergeList_L (LinkList &La，LinkList &Lb，LinkList &Lc) {

 pa = La->next ;   pb = Lb->next ;  //point to two first node
   Lc = pc = La ;
      while  ( pa && pb )  {
         if  ( pa->data <= pb->data )  {
               pc->next = pa ;  pc = pa ;  pa = pa->next;  }
         else  {  pc->next = pb ;  pc = pb ;  pb = pb->next ;  }
       }
   pc->next = pa ? pa : pb ;  //deal with the remain


   delete  Lb ;
}
```

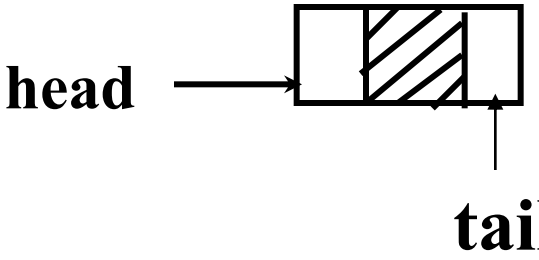
Double Linked List

Double linked list allows convenient access from a list node to the next node and also to the preceding node on the list.

Double linked list node accomplishes this in the obvious way by storing two pointers



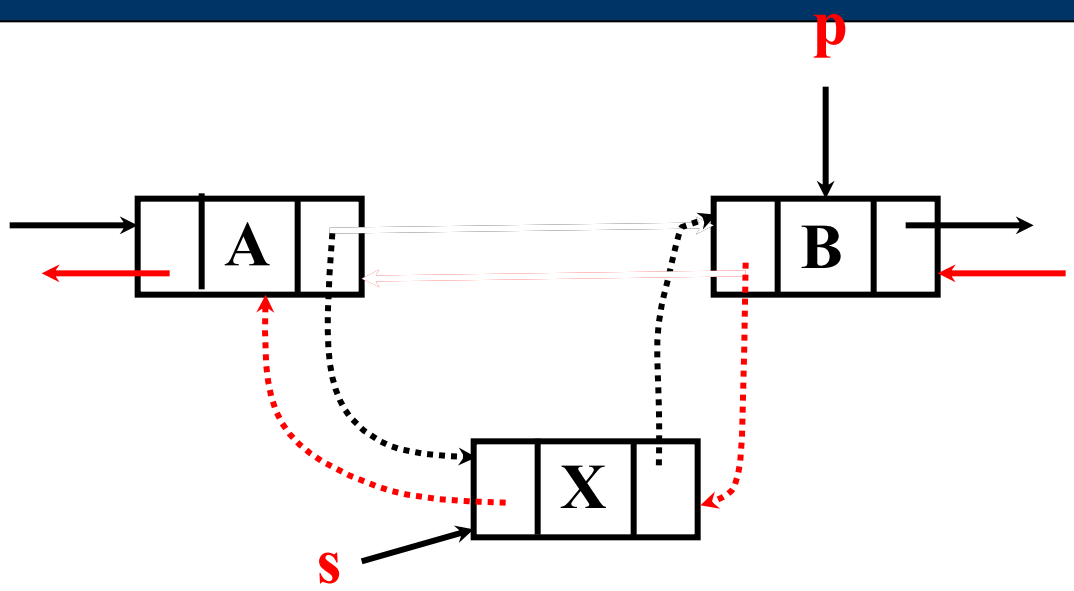
Empty list:



Property: suppose d is the pointer that points to some node, then

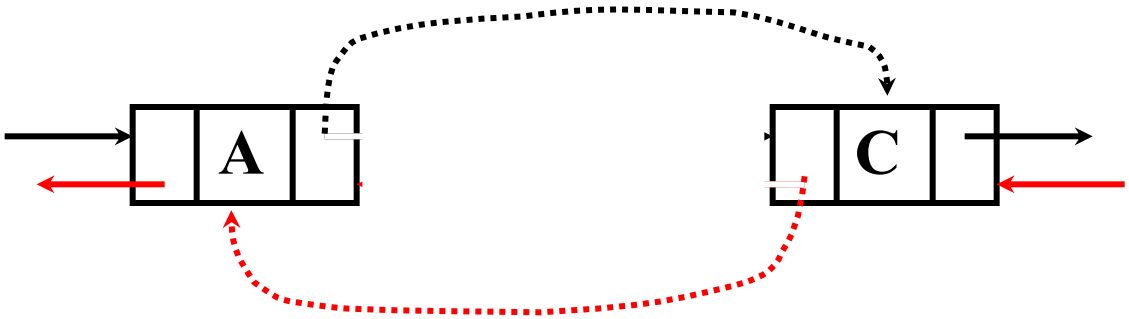
d->next->prior = d->prior->next= d

1) insert



1. Find the node before which you want to insert, p
2. s->prior= p->prior ;
3. p->prior->next= s ;
4. s->next = p ;
5. p->prior= s ;

2) Delete



1. Find the deleting node, p
2. p->prior->next = p->next ;
3. p->next->prior= p->prior ;
4. delete p ;

Homework

- Please refer to Icourse，Huawei Cloud.

- Due date for quiz: 23:30　2022/3/15
- Due date for homework: 23:30 2022/3/20
- Due data for online lab assignment:　2022/3/20　23：30