Data Structures and Algorithms (2)

Instructor: Ming Zhang

Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao

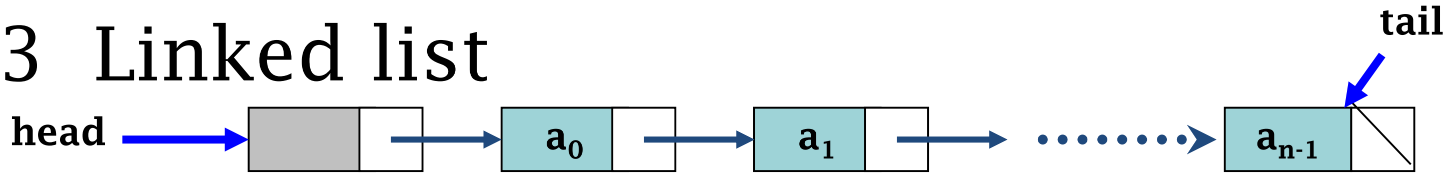
Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)

https://courses.edx.org/courses/PekingX/04830050x/2T2014/

Chapter II Linear Lists

- 2.1 Linear list $\{a_0, a_1, \dots, a_{n-1}\}$
- 2.2 Sequential list

- 2.3 Linked list



- 2.4 Comparison of sequential list and linked list

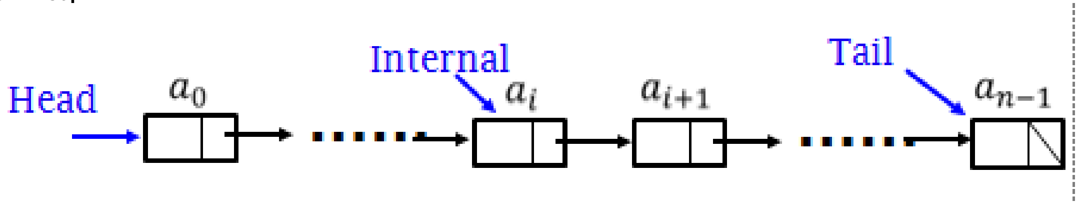
The Concepts of Linear List

- List for short, is a finite sequence of zero or more elements, usually represented as k_0 , k_1, … , k_{n-1} ($n \geq 1$)
  - Entries: elements of linear list (can contain multiple data items, records)
  - Index: i is called the "Index" of entry ki
  - Length of the list: the number of elements contained in the list n
  - Empty list: a linear list with the length of zero (n = 0)
- Features of Linear list:
  - Flexible operations
  - Dynamically changed length

Linear structure

- Tuple $B = (K, R)$ $K = \{a_0, a_1, \dots, a_{n-1}\}$ $R = \{r\}$
  - There is one and only one starting point that has no previous node and has only one successive node.
  - There is one and only one ending point that has only one previous node and has no successive node.
  - The other nodes are called internal nodes that have only one previous node and also have only one successive node.

$\langle a_i, a_{i+1}\rangle$ a_i is previous node of a_{i+1} , and a_{i+1} is the successive node of a_i



Linear structure

- Features

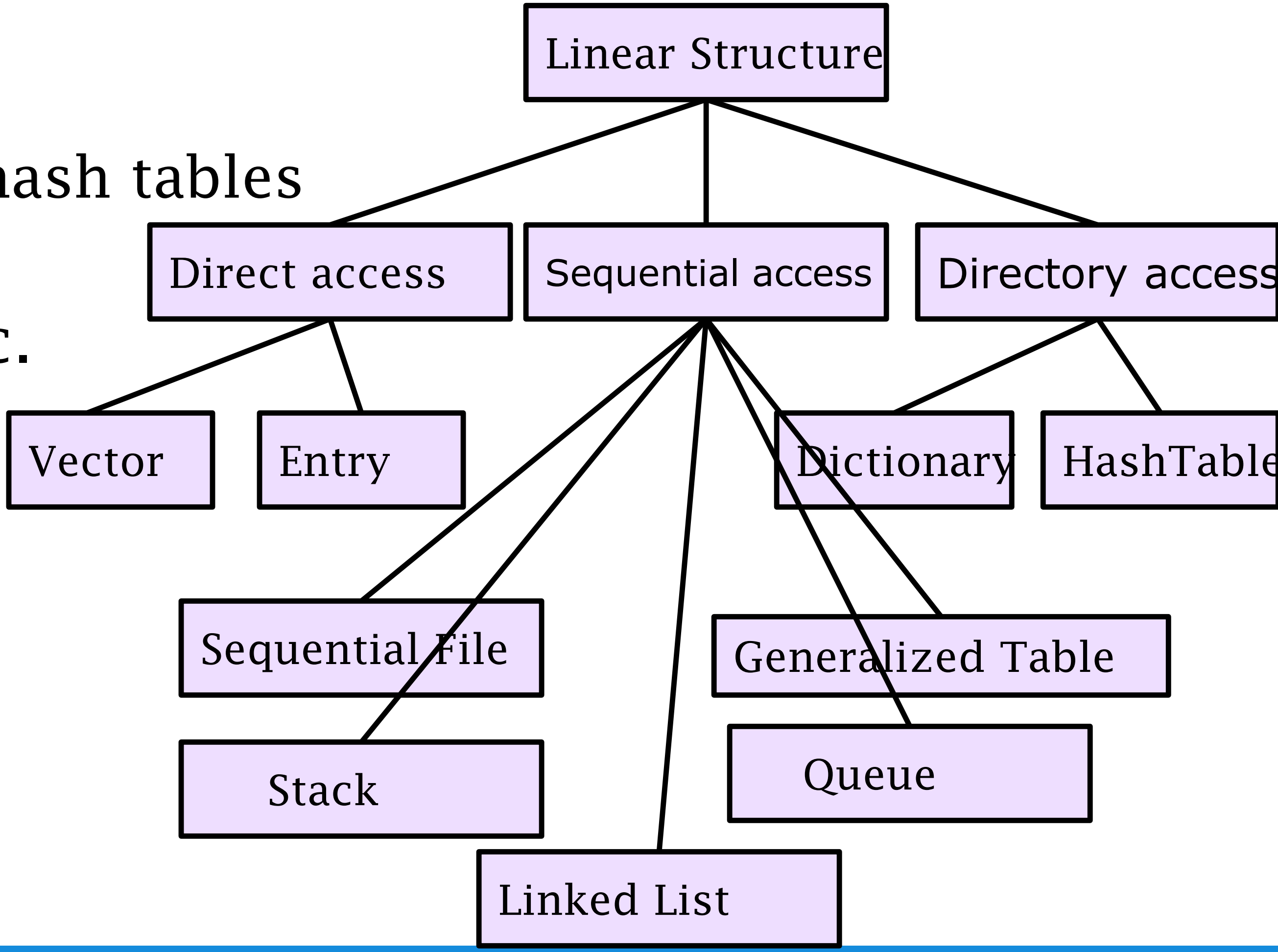
✓ Uniformity: Although the data elements of different linear lists may be diverse, but the data elements of the same linear list normally have the same data type and length

✓ Orderliness: each data element has its own position in the list and their relative positions are linear

Classification

Linear structure

- According to the complexity
  - Simple: Linear lists, stacks, queues, hash tables
  - Advanced: generalized lists, multidimensional arrays, files etc.
- Divided by access ways
  - Direct access type
  - Sequential access type
  - Contents Index type (directory access)



Linear structure

- **Classified by operation (see later)**

  -Linear List

  - All entries are nodes of the same type of linear lists
  - No need to limit the form of operation
  - Divided into: the sequential list, linked list depending on the difference of storage

  -Stack (LIFO, Last In First Out)

  Insert and delete operations are restricted to the same end of the list

  -Queue (FIFO, First In First Out)

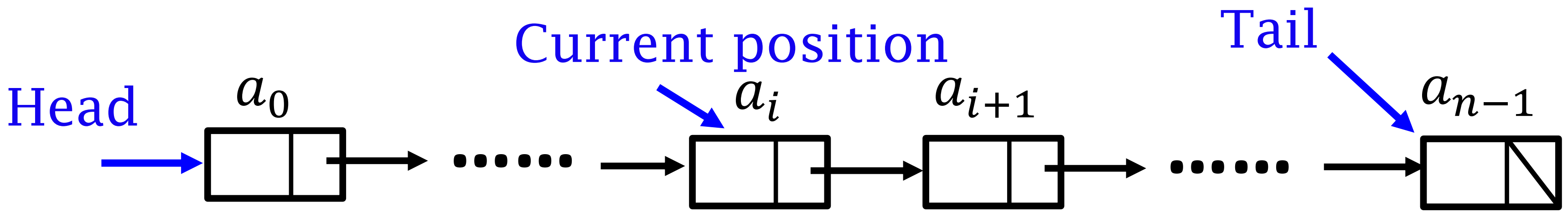
  Insert at one end of the list, while delete at the other end

2.1 Linear List

- Three aspects
  - Logical structure of the linear list
  - Storage structure
  - Operation of linear list

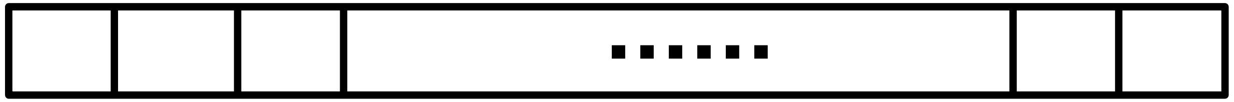
Logical structure of the linear list

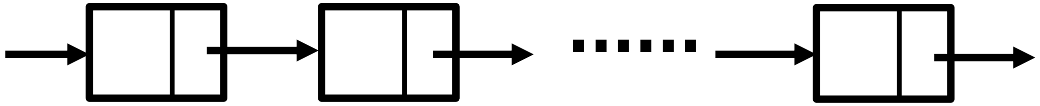
- The main properties
  - Length
  - Head
  - Tail
  - Current position



Classification (By storage)

- Linear List
  - All entries are nodes of the same type of linear lists
  - No need to limit the form of operation
  - Divided into: the sequential list, linked list depending on the difference of storage

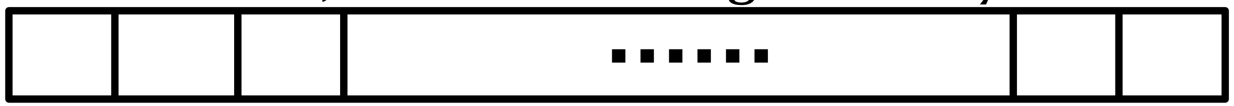




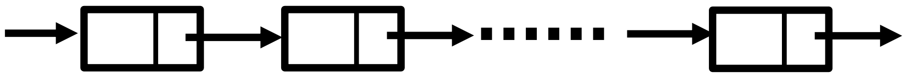
Storage Structures

- Sequential list

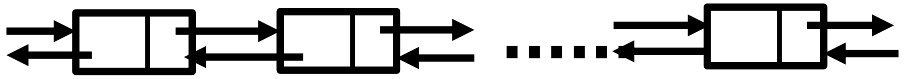
  - Store according to index values from small to large in an adjacent continuous region
  - Compact structure, and the storage density is 1



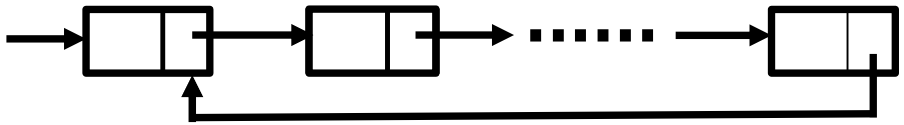
- Linked list
  - Single list



  - Double linked list



  - Circular list

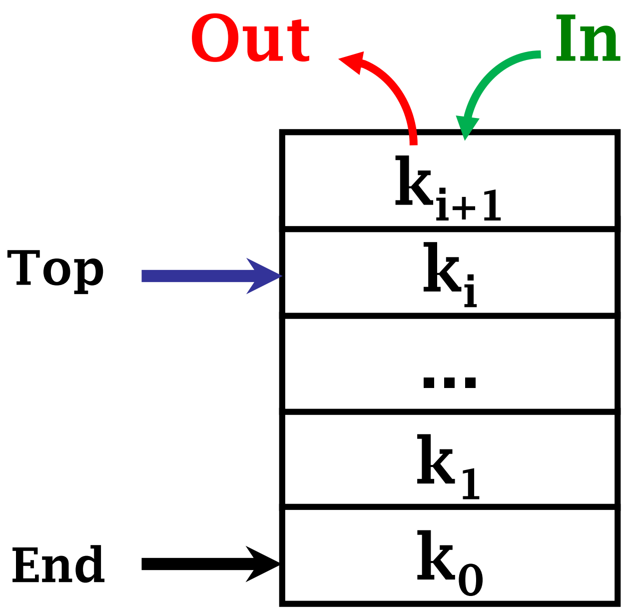


Classification (By operation)

- Linear List
  - No need to limit the form of operation
- Stack
  - At the same end
- Queue
  - At both ends

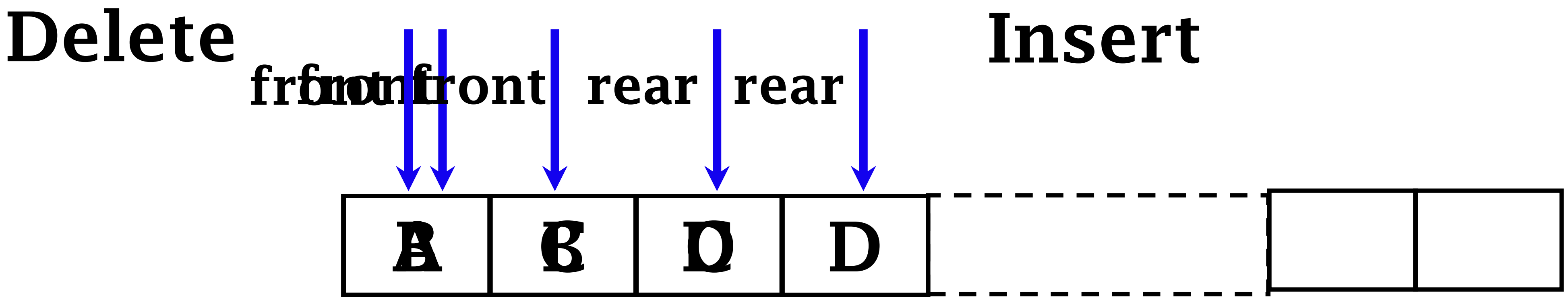
Classification (By operation)

- Stack (LIFO, Last In First Out)
  - Insert and delete operations are restricted to the same end of the list



Classification (By operation)

- Queue (FIFO, First In First Out)
  - Insert at one end of the list while delete at the other end
- Rear(true pointer)



Operation on linear Lists

- Construct a linear list
- Destruct the linear list
- Insert a new element
- Delete a specific element
- Modify a specific element
- Sort
- Search
- …

Class Template of Linear lists

```
template <class T> class List {
    void clear();          // clear the linear list
    bool isEmpty();        // When it is empty, returns true
    bool append(const T value);
                           // insert the value  at the end , length adds by 1
    bool insert(const int p, const T value);
                           // insert the value at position P , length adds  by 1
    bool delete(const int p);
                           // delete the value at position p , length decreases by 1
    bool getPos(int& p, const T value);
                           // find the value and returns its position
    bool getValue(const int p, T& value);
                           // return the element's value at  position P
                           //and assign it to the variable of value
    bool setValue(const int p, const T value);
                           // set value for position P
};
```

Thinking

- What kind of classification are there for the linear list?

- In all kinds of names of linear lists , which are related to storage structures? Which are related to operations?

Chapter II Linear List

- 2.1 Linear List
- 2.2 Sequential List
- 2.3 Linked List
- 2.4 Comparison between sequential list and linked list

2.2 Sequential List

- **Also known as the vector, fixed-length one-dimensional array is used as the storage structure**
- **Key Features**

  - Elements are of the same type

  - Elements are sequentially stored in contiguous storage space, and each element has a unique index value

  - The type of vector length is constant
- **Implemented as Array**
- **Its elements are easy to read and write, you can specify the location by using its subscript**

  - Once the starting position is got, all the data elements of the list can be random accessed

2.2 Sequential List

. The formula to calculate the elements of location is shown as below:

– $Loc(k_i) = Loc(k_0) + c \times i,\ \ c = \text{sizeof}(ELEM)$

Logical Address (Subscript)	Data elements
0	k_0
1	k_1
…	…
i	k_i
…	
n-1	k_{n-1}

Store Address	Data elements
$Loc(k_0)$	k_0
$Loc(k_0)+c$	k_1
…	…
$Loc(k_0)+i*c$	k_i
…	
$Loc(k_0)+(n-1)*c$	k_{n-1}

Sequence List's Class Definition

```
class arrList : public List<T> {       // sequential list , vector
private:                                            // value types and value space of linear list
    T * aList ;              // private variables , instance of storage for sequential list
    int maxSize;           // private variables  , maximum length of the sequential list
    int curLen;            // private variables  , current length of the sequential list
    int position;          // private variables  , current operation location
public:
    arrList(const int size) {  // construct a new list , set its length to the maximum
        maxSize = size;  aList = new T[maxSize];
                curLen = position = 0;
    }
    ~arrList() {                  // destructor function used to eliminate the instance
                delete [] aList;
    }
```

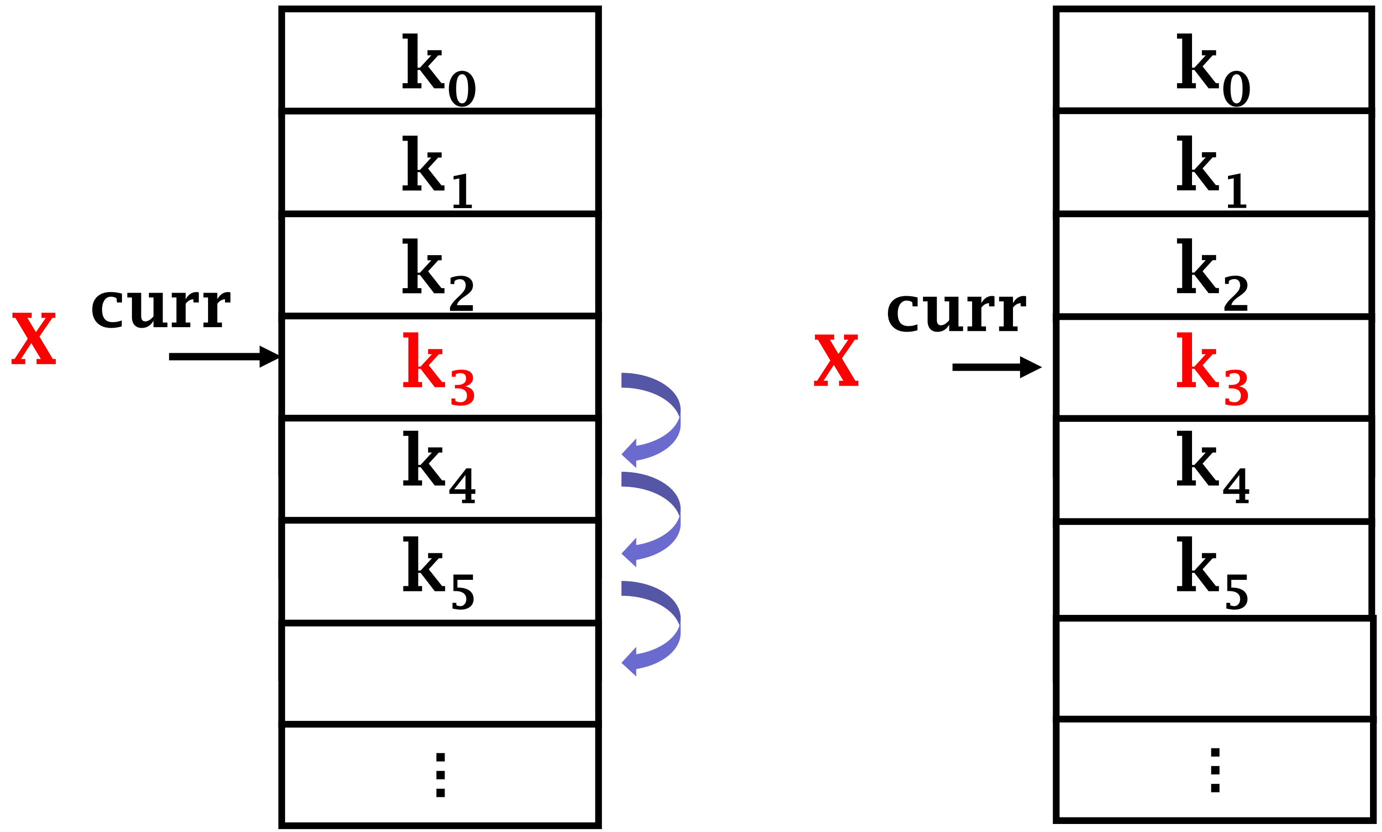
Sequence List's Class Definition

```
    void clear() {    // delete the content , becoming an empty list
        delete [] aList; curLen = position = 0;
          aList = new T[maxSize];
    }
    int  length();                                    // returns the current length
    bool append(const T value);                       // append element v at end
    bool insert(const int p, const T value);    // insert an element at P
    bool delete(const int p);                         // delete the element at P
    bool setValue(const int p, const T value);        // set the value of an element
    bool getValue(const int p, T& value);       // return the value of an element
    bool getPos(int &p, const T value);               // seek for an element
};
```

Operations in Sequential List

- Key discussions
  - Insert element operation
    - bool insert(const int p, const T value);
  - Delete element operation
    - bool delete(const int p);
- Others (Think by yourselves)

Diagram for the insertion of sequential list

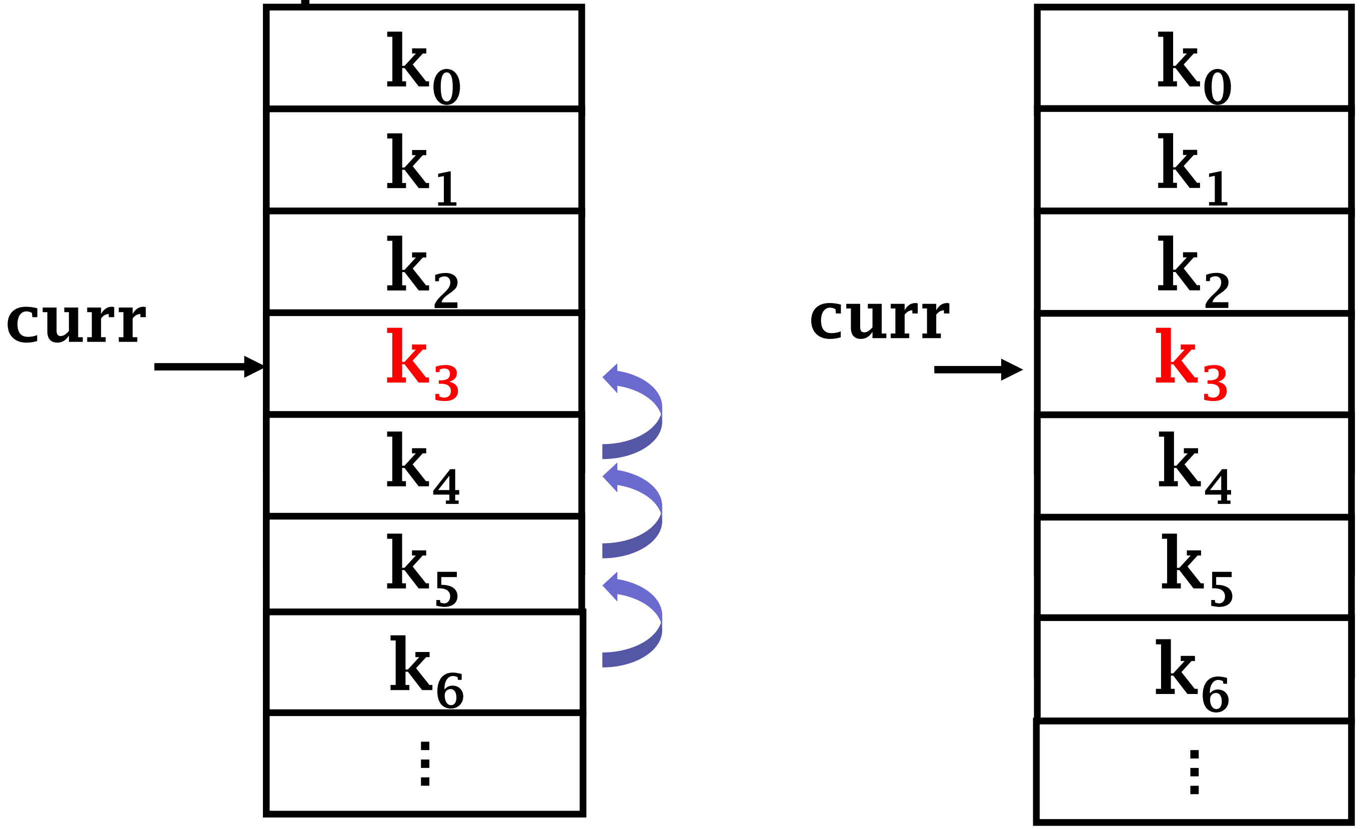


Insertion of sequential list

```
// set the element type as T， aList is the array to store Sequential list，
// maxSize is its maximum length；
// p is the insert location of the new element ，return true if succeeds，
// otherwise return false
template <class T>  bool arrList<T> :: insert (const int p, const T value) {
    int i;
    if (curLen >= maxSize) { // check if the SL is overflow
        cout << "The list is overflow"<< endl;  return false;
    }
    if (p < 0 || p > curLen) { // check if the position to insert is valid
        cout << "Insertion point is illegal"<< endl;  return false;
    }
    for (i = curLen; i > p; i--)
        aList[i] = aList[i-1];    // move right from the end curLen -1 of the
list until p
    aList[p] = value;             // insert a new element at p
    curLen++;                     // adds the current length of the list by 1
    return true;
}
```

Diagram for sequential list's delete operation

- 2.2 Sequential List



Delete operation in sequential list

```
// set the type of the element as T ; aLis is the array to store sequential list
// and p is the position of elements to delete
// returns  true when delete succeed , otherwise returns false
template <class T>                    // the  type of the elements of SL is T
bool arrList<T> :: delete(const int p)  {
    int i;
    if (curLen <= 0 )  {              // Check if the SL is empty
        cout << " No element to delete \n"<< endl;
        return false ;
    }
    if (p < 0 || p > curLen-1)  {     // Check if the position is valid
        cout << "deletion is illegal\n"<< endl;
        return false ;
    }
    for (i = p; i < curLen-1; i++)
        aList[i] = aList[i+1];        // [p, currLen) every element move left
    curLen--;            // the current length of the list decreases by  1
    return true;
}
```

Algorithm analysis of insert and delete operations in sequential list

- The movement of elements in the list
  - Insert: move $n-i$
  - Delete: move $n-i-1$
- The probability values to insert or delete in position i are respectively p_i *and* p_i'
  - The average move time for insert operation is

$$M_i = \sum_{i=0}^{n}(n-i)p_i$$

  - The average move time of delete operation is

$$M_d = \sum_{i=0}^{n-1}(n-i-1)p_i'$$

Algorithm Analysis

- If the probability to insert or delete in every location in SL is the same, namely $p_i = \frac{1}{n+1}$, $p_i' = \frac{1}{n}$

$$M_i = \frac{1}{n+1}\sum_{i=0}^{n}(n-i) = \frac{1}{n+1}(\sum_{i=0}^{n} n - \sum_{i=0}^{n} i)$$

$$= \frac{n(n+1)}{n+1} - \frac{n(n+1)}{2(n+1)} = \frac{n}{2}$$

$$M_d = \frac{1}{n}\sum_{i=0}^{n}(n-i-1) = \frac{1}{n}(\sum_{i=0}^{n} n - \sum_{i=0}^{n} i - n)$$

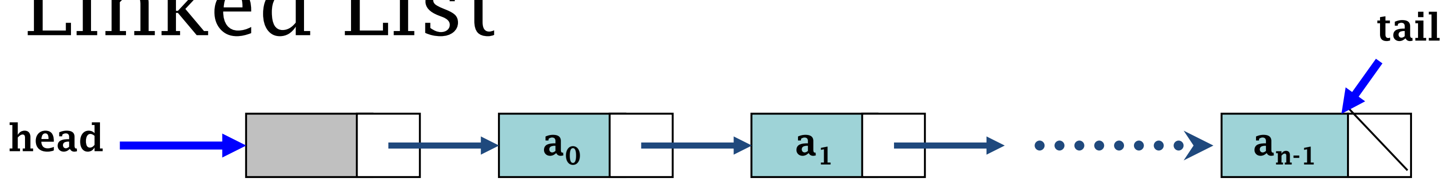
$$= \frac{n^2}{n} - \frac{(n-1)}{2} - 1 = \frac{n-1}{2}$$

Time cost is $O(n)$

Thinking

- What should you think about when doing insert or delete operations in sequential list ?
- What advantages and disadvantages does sequential list have?

Chapter II Linear List



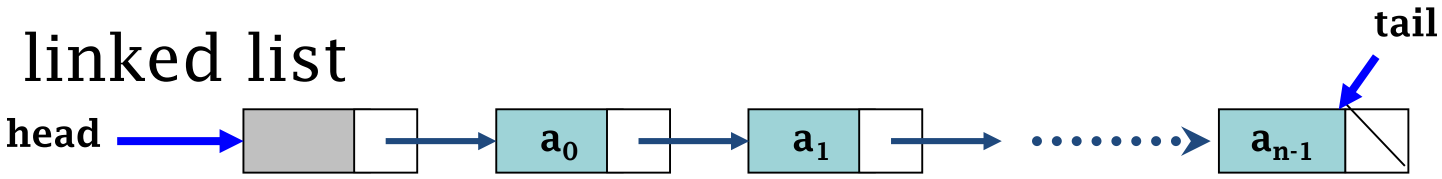
Linked List

- Link its storage nodes through pointers
  .
- Storage nodes are consisted of two parts
  - Data field + pointer field (successor address)

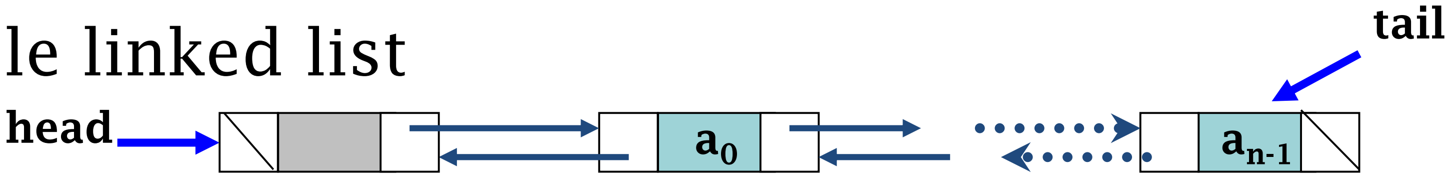
data	next

2.3 Linked List

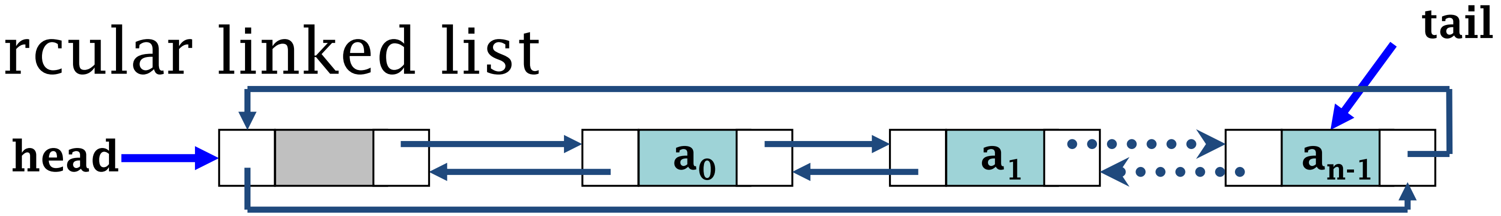
- **Classification (according to linked ways and the number of points)**
  - Single linked list

  - Double linked list

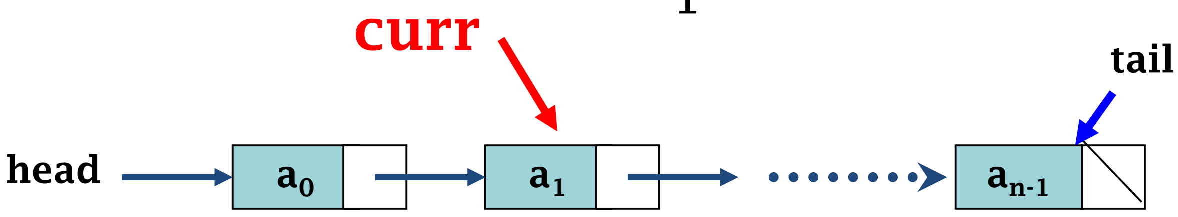
  - Circular linked list

Single linked list

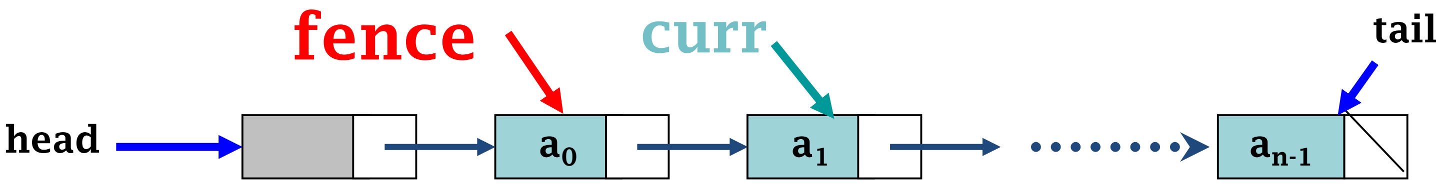
- Simple single linked list
  - The whole single linked list： head
  - The first node： head
  - The judge of empty list：

    head == NULL
  - The current node a_1：curr



Single linked list

- Single linked list with head node
  - The whole single linked list ： head
  - The first node ： head->next , head ≠ NULL
  - The judge of empty list :
    - head->next == NULL
  - The current node a_1 : fence->next (curr implied)



Node type of the single linked list

```
template <class T> class Link {
    public:
    T   data;               // to protect content of the node elements
    Link<T> * next;       // the pointer which points to successor point

    Link(const T info, const Link<T>* nextValue =NULL)  {
        data = info;
        next = nextValue;
    }
    Link(const Link<T>* nextValue)  {
        next = nextValue;
    }
};
```

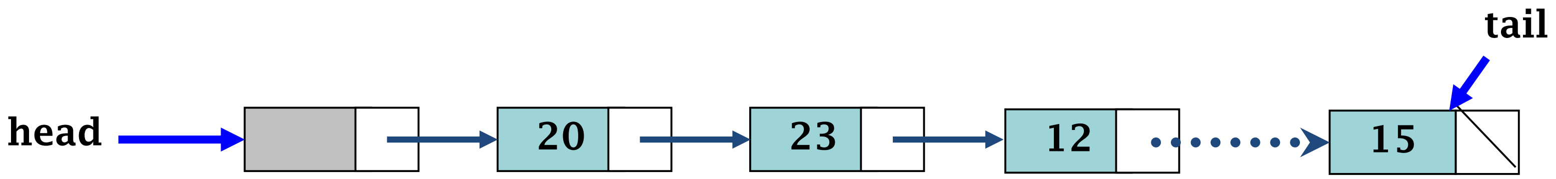
Class definition of single list

```
template <class T> class lnkList : public List<T>  {
    private:
    Link<T>  * head, *tail;                   // head and tail pointer of the single list
    Link<T>  *setPos(const int p);            // the pointer of the pth element
     public:
    lnkList(int s);                           // constructed function
    ~lnkList();                               // destructor
    bool isEmpty();                           // judge whether the link is empty
    void clear();                 // clear the link's storage and it becomes an empty list
    int length();                 // returns the current length of the sequential list
    bool append(cosnt T value);               // add an element value at the end ,
                                              // the length of the list added by 1
    bool insert(cosnt int p, cosnt T value); // insert an element at p
    bool delete(cosnt int p);                 // delete the element at p ,
                                              // the length of the list decreased by 1
    bool getValue(cosnt int p, T& value);     // get the value of the element at p
    bool getPos(int &p, const T value);       // seek for element with value T
}
```

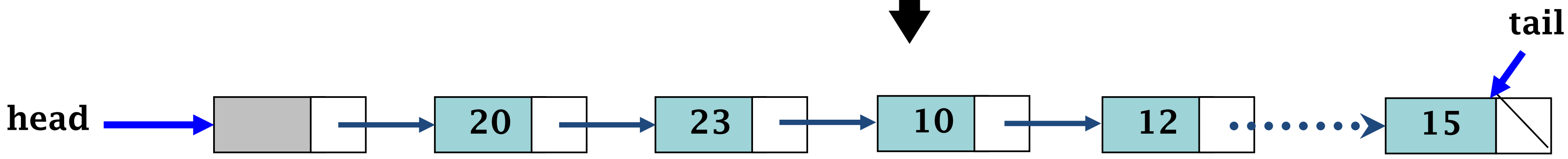
Seek the ith node in the single linked list

```
// the return value of the function is the found node pointer
template <class T>          // the element type of the linked list is P
Link<T> * lnkList <T>:: setPos(int i) {
    int count = 0;
     if (i == -1)                 // if i was -1, then locate it to the head
        return head;
    // circular location, if I was 0 then locate to the first node
    Link<T> *p = head->next;
    while (p != NULL && count < i) {
        p = p-> next;
        count++;
     };
     // points to the ith node , i＝0,1,… , when the number of
     // the nodes of the list is less than i then return NULL
    return p;
}
```

Insert operation of single linked list



Insert 10 between 23 and 12



- Create a new node
- New node points to the right node
- The left node points to new node

Insert algorithm of single linked list

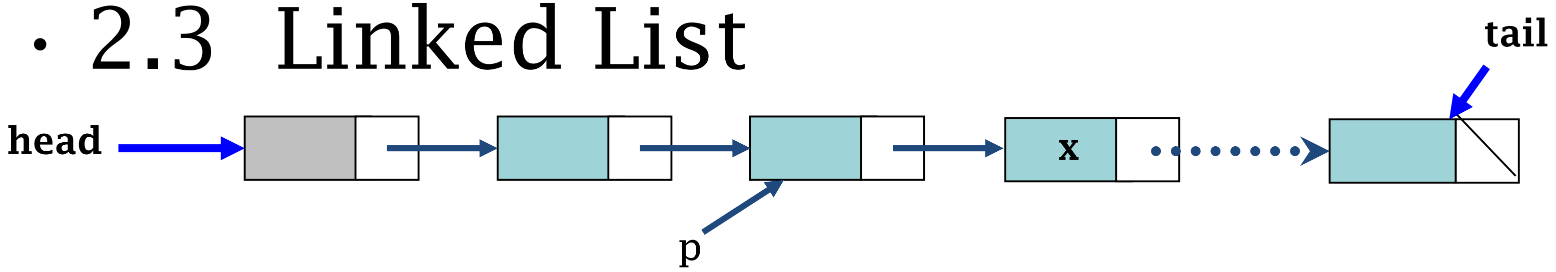
```
// insert a new node as the ith node
template <class T>
// element type of the linked list is T
bool lnkList<T> :: insert(const int i, const T value) {
    Link<T> *p, *q;
    if ((p = setPos(i -1)) == NULL) {     // p is the previous node of the ith node
        cout << " illegal insert position"<< endl;
        return false;
    }
    q = new Link<T>(value, p->next);
    p->next = q;
    if (p == tail)                        // insert position is at the tail and
                                          // the node inserted becomes the new tail
        tail = q;
    return true;
}
```

Delete operation of single linked list

- Delete the node x from linked list
  - 1. Assign p to point to the previous node of element x
  - 2. delete the node with element x
  - 3. release the space that x occupied

Example of delete operation of single linked list

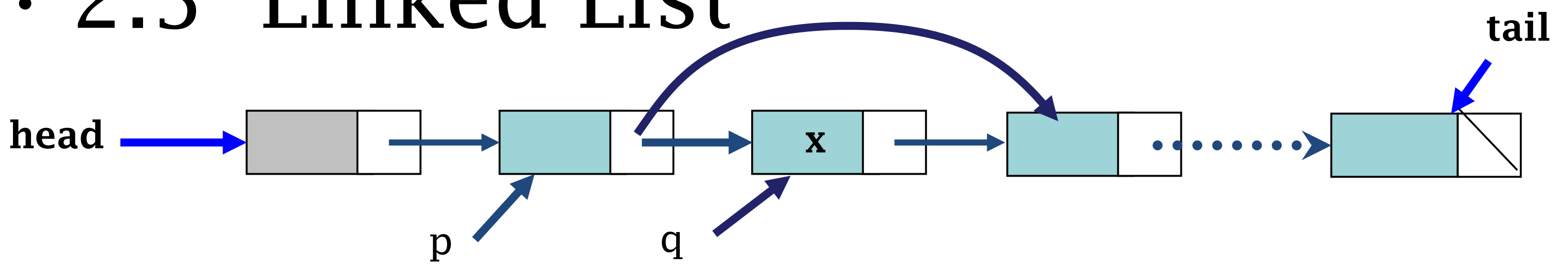
- 2.3 Linked List




```
p = head;
while (p->next!=NULL && p->next->info!= x)
        p = p->next;
```

Delete the node with value X

- 2.3 Linked List



q = p->next;
p->next = q->next;
free(q);

Delete algorithm of single linked list

```
template <class T>                    // Element type of the linked list is T
bool lnkList<T>:: delete((const int i)  {
    Link<T> *p, *q;
     // node to delete doesn't exist, when the given i is bigger than
     // the number of the current elements in the list
    if  ((p = setPos(i-1)) == NULL  ||  p == tail) {
        cout << " illegal delete position " << endl;
        return false;
    }
    q = p->next;                      // q is the real node to delete
    if (q == tail) {                  // if the node to delte is the tail,
                                      // then change the tail pointer
        tail = p;         p->next = NULL:
    }
    else                              //delete node q and change linked pointer
        p->next = q->next;
    delete q;
    return true;
}
```

Operation analysis of single linked list

- To operate on a node you must find it first, which means to get a pointer address

- To find any node in single linked list you must begin from the first node

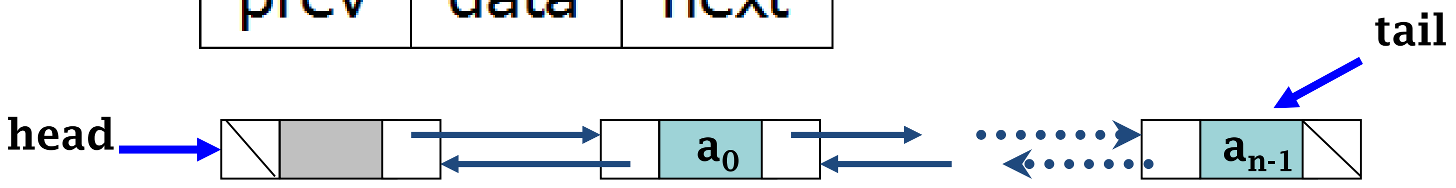
```
p = head;
while (not reaching) p = p->next;
```

- The time complexity $O(n)$
  - locating : $O(n)$
  - insert : $O(n) + O(1)$
  - delete : $O(n) + O(1)$

Double linked list

- To make up the disadvantages of single linked list, double linked list appears.
  - The next field of single linked list only points to the previous node , it can not be used to find the successive node. The same for “single prev”.
  - So, we add a pointer that points to the precursor node of it in the double linked list.

prev	data	next

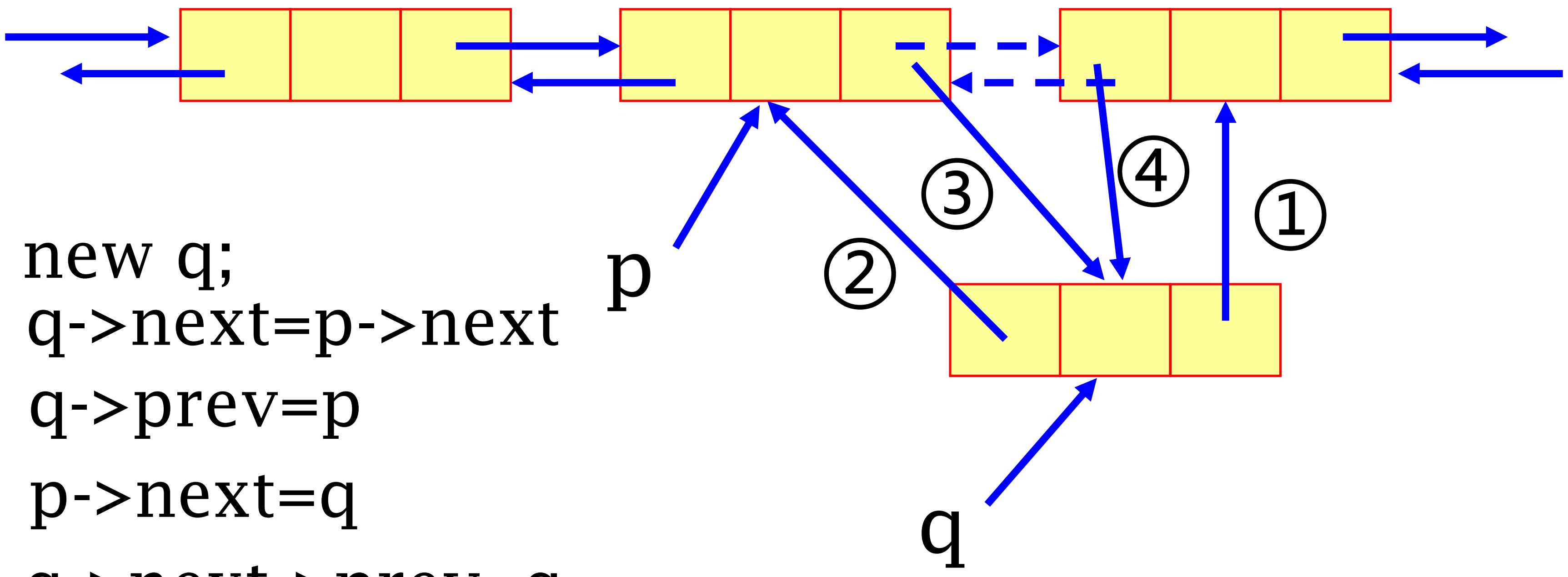


Double linked list and type of its node

```
template <class T> class Link {
    public:
    T  data;                        // used to store content of node elements
    Link<T> * next;                 // the pointer points to successor node
    Link<T> *prev;                  // the pointer points to precursor node
    Link(const T info, Link<T>* preValue = NULL, Link<T>* nextValue =
NULL) {
        // constructor with given value and precursor and successor pointers
        data = info;
        next = nextValue;
        prev = preValue;
    }
    Link(Link<T>* preValue = NULL, Link<T>* nextValue = NULL)  {
        // constructor with given value and precursor and successor pointers
        next = nextValue;
        prev = preValue;
    }
}
```

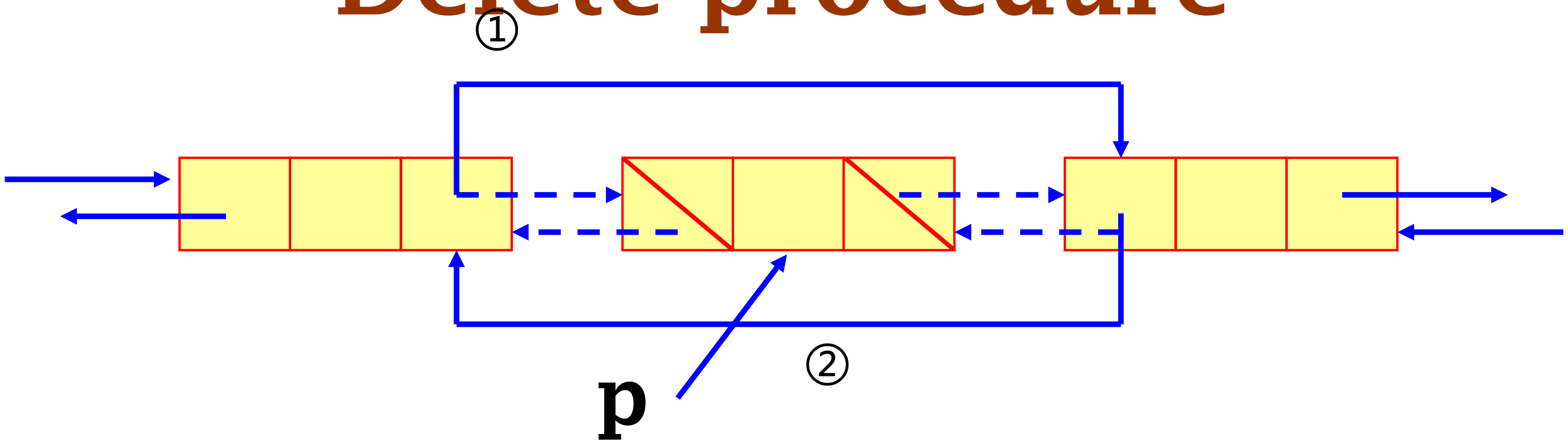
Insert procedure of double linked list (Be careful with the order)

Insert a new node after the node pointed by p




```
new q;
q->next=p->next
q->prev=p
p->next=q
q->next->prev=q
```

Delete procedure



Delete the node pointed by p

p->prev->next=p->next

p->next->prev=p->prev

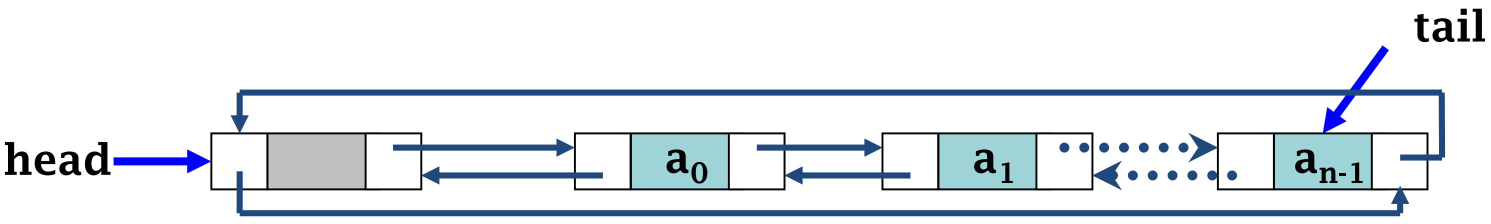
p->next=NULL

p->prev=NULL

- If you delete p immediately
  - Do not need to assign the null value

Circularly linked list

- Link the head and tail of single linked list and double linked list, and we created circular lists
- Do not increase other cost, but benefit lots of operations
  - From any node of circular list you can access all the other nodes



Boundary conditions of linked list

- Treatment of some special points
  - Treatment with the head node
  - Pointer field of the tail node of a non-circular list should be kept as NULL
  - Tail of a circular list points to its head pointer
- Treatment with linked list
  - Special treatment with empty linked list
  - When insert or delete nodes, be careful with the linking process of the related pointers
  - The correctness of points moving
    - insert
    - search or iteration

Thinking

- Think about the single linked list with head or not.
- The problems you should consider when deal with linked list.

Chapter II Linear List

- 2.1 Linear List
- 2.2 Sequential List
- 2.3 Linked list
- 2.4 Comparison between sequential list and linked list

2.4 Comparison of the implementation method of linear list

- Main advantages of sequential lists
  - No pointers, and no overhead cost
  - Read an element in a sequential list is quite easy and convenient
- Main advantages of linked list
  - No need to know the list length list before construction
  - The length of the linked list can be dynamically changed
  - Support frequent insert and delete operations
- To sum up
  - Sequential list is the best choice for storing static data
  - Linked list is a good choice for storing dynamic data

Comparison between sequential list and linked list

- Sequential list
  - Time cost of insert and delete operation is O(n) , search of ith element can be done in constant time.
  - You must apply for continuous storage space with fixed length previously
  - If the whole array is full there will be no structural storage cost
- Linked list
  - Time cost of insert and delete operation is O(1) , but the cost for finding the ith element is O(n)
  - Uses pointers for storage, you need to assign storage space dynamically to the new elements per demand
  - Every element has overhead storage cost

Storage density of Sequential list and linked list

n means the current number of elements in linear list

P means the size of the storage space of the pointer (usually 4 bytes)

E means the size of the storage space of the data element

D means the maximum number of linked list elements that can be stored in array

- Space requirement
  - Space requirement of sequential list is DE
  - Space requirement of linked list is n(P + E)
- The critical value of n , namely n > DE / (P+E)
  - The bigger the n, the higher the space efficiency of sequential list
  - If P = E , then the critical value is n = D / 2

The choice in different situations

- Situations sequential list not fit
  - Insert or delete operations are frequent
  - The maximum length of the linear list is also a vital consideration
- Situations linked list not fit
  - When read operation is more frequen than insert or delete operations
  - When the storage cost of the pointer is relatively big compared to the occupied space of the attributes of a node , think carefully.

Choice between Sequential list and linked list

- Sequential list
  - The number of nodes can be estimated
  - The nodes are relatively stable
    (insert and delete operation are not frequent)
  - $n > DE / (P + E)$
- Linked list
  - The number of nodes cannot be estimated
  - The node are dynamic
    (insert and delete operation are frequent)
  - $n < DE / (P + E)$

Thinking

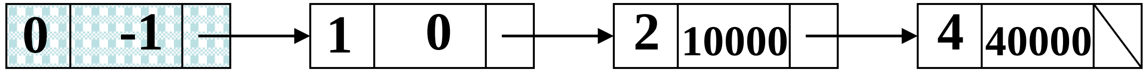
- Choose between sequential list and linked list.
  - Dynamic change of nodes
  - Storage density

Thinking： expression of polynomial with one variable

- Polynomial with one variable: $P_n(x) = p_0+p_1x+p_2x^2+\ldots+p_n x^n$
- Linear list expression：$P = (p_0, p_1, p_2, \ldots, p_n)$
- Sequential list expression：only save the coefficient（the ith element save X’s coefficient）

the situation when the data is sparse: $p(x) = 1+2x^{10000}+4x^{40000}$

- Linked list expression： node structure



Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)