第二章 线性表

一. 概念

1. 线性表
2. 单链表
3. 双链表
4. 循环表

二. 方法

5. 顺序表上实现的运算
6. ★ 链表上实现的运算(指针操作的正确性)
7. 顺序表和链表的比较

1.线性表

概念

线性表简称表，是零个或多个元素的有穷序列， 通常可以表示成

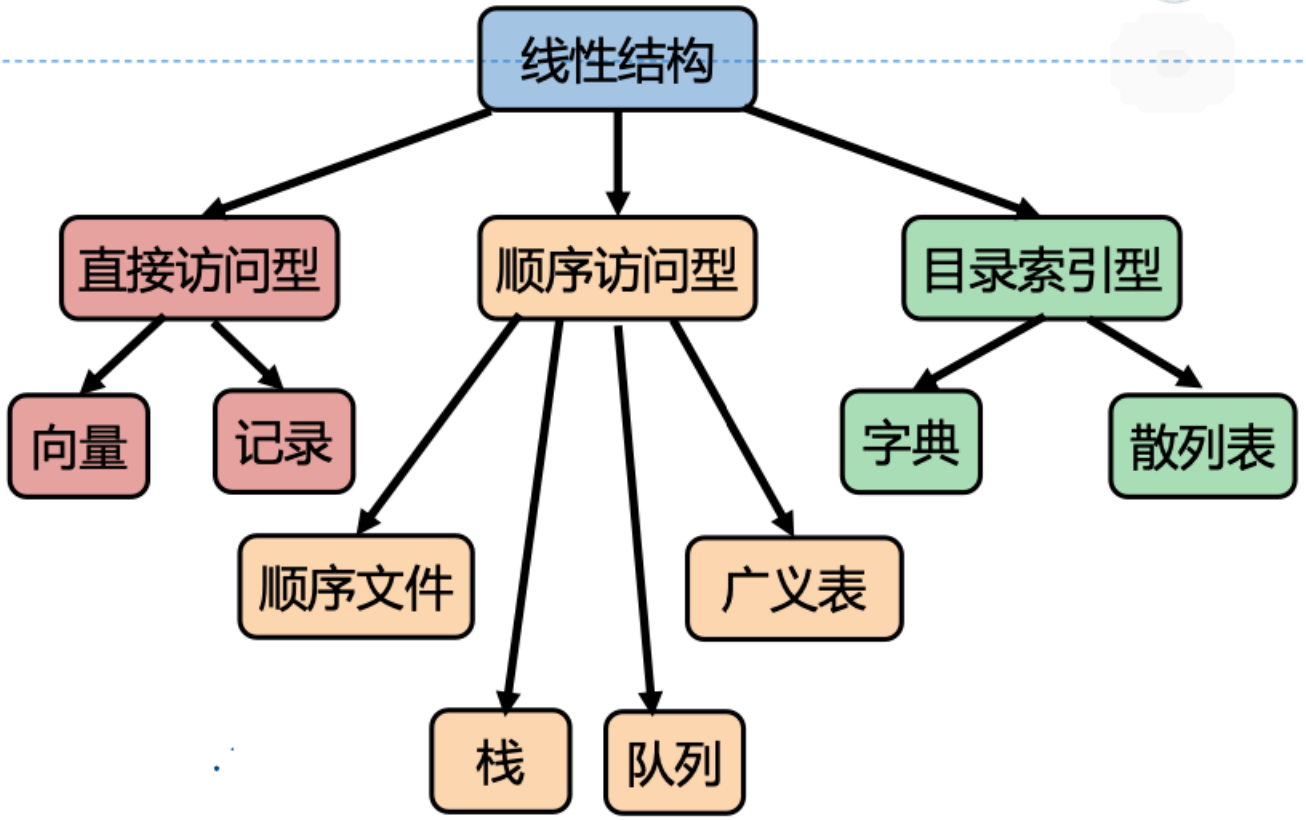
$$k_0，k_1,\ldots，k_{n-1}(n \geq 1)$$

相关概念

表目/记录，文件，索引/下标，表的长度，空表

线性表的分类

按访问方式分



按运算和操作分

线性表——不限制操作
栈——在同一端操作
队列——在两端操作

线性表的逻辑结构

长度，表头，表尾，当前位置

线性表的存储结构

顺序表

-存储在相邻的连续区域
-紧凑结构，存储密度为1

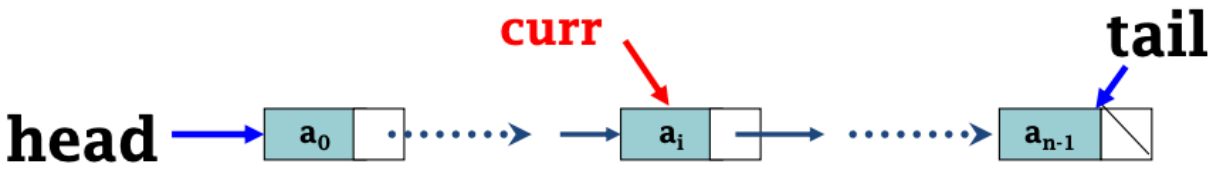
链表

-单链表
-双链表
-循环链表

2.单链表

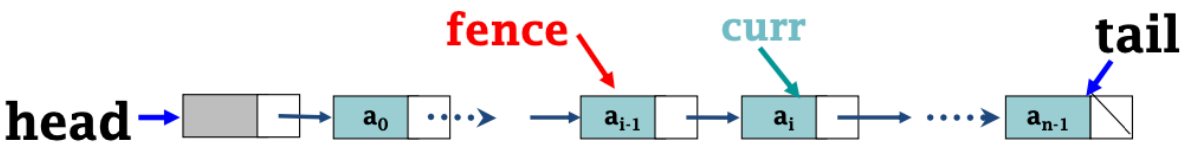
· 普通单链表

- 整个单链表： head
- 第一个结点： head
- 空表判断： head == NULL
- 当前结点 a_i： curr

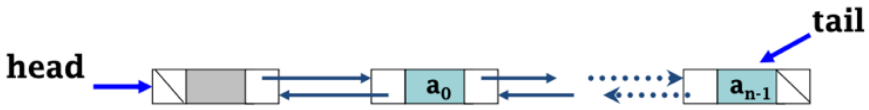


· 带头结点的单链表

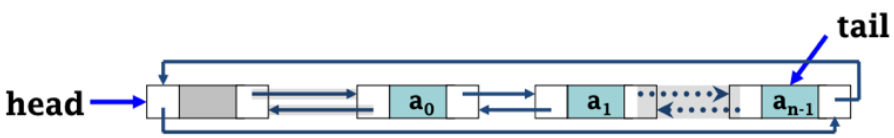
- 整个单链表： head
- 第一个结点： head->next，head ≠ NULL
- 空表判断： head->next == NULL
- 当前结点a_i： fence->next (curr 隐含)



3.双链表



4.（单/双）循环链表



不增加额外花销，却给操作带来不少方便——从循环表中任意结点出发都能访问到表中其他结点

5.顺序表上实现的运算

```
//顺序表类定义
template <class T>
class arrList{
public:
  T*aList;
  int maxLen;
  int curLen;
  arrList(int size){
    maxLen=size;
    aList=new T[maxLen];
    curlen==curpos=0;
  }
  ~arrList(){
    delete []alist;
  }

  void insert(int p,T val);
  void myDelete(int p);
};
//插入操作
template <class T>
void arrList<T>::insert(int p,T val){
  for(int i=curLen;i>p;i--)//移动curLen-p次
    aList[i]=aList[i-1];
  aList[p]=val;
  curLen++;
}
//删除操作
template <class T>
void arrList<T>::myDelete(int p){
  for(int i=p;i<curLen-1;i++)//移动curLen-p-1次
    aList[i]=aList[i+1];
  curLen--;
}
```

复杂度分析

顺序表插入和删除算法主要代价体现在表中元素的移动

-插入移动次数 n-i 次
-删除操作移动 n-i-1次

若在i 的位置上插入和删除的概率分别是 pi 和 pi'

插入的平均移动次数为

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

删除的平均移动次数为

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

考虑等概率插入，即

$$p_i = \frac{1}{n+1}, p_i^{'} = \frac{1}{n}$$

故

$$\begin{aligned} M_i &= \frac{1}{n+1}\sum_{i=0}^{n}(n-i) = \frac{1}{n+1}(\sum_{i=0}^{n}n - \sum_{i=0}^{n}i) \\ &= \frac{n(n+1)}{n+1} - \frac{n(n+1)}{2(n+1)} = \frac{n}{2} \end{aligned}$$

$$\begin{aligned} M_d &= \frac{1}{n}\sum_{i=0}^{n-1}(n-i-1) = \frac{1}{n}(\sum_{i=0}^{n-1}n - \sum_{i=0}^{n-1}(i-n)) \\ &= \frac{n^2}{n} - \frac{n-1}{2} - 1 = \frac{n-1}{2} \end{aligned}$$

所以时间复杂度为O(n)

6.链表上实现的运算——单链表

```
template <class T>
class listNode{
public:
  T val;
  listNode<T>*next;
  listNode(T v, listNode<T>*n=nullptr):val(v),next(n){}
};

//单链表类定义
template <class T>
class linkList{
```

```
public:
  listNode<T>*head;

  listNode<T>*find(int p);
  void insert(int p,T val);
  void myDelete(int p);
};
//查找操作
template <class T>
listNode<T>* linkList<T>::find(int p){
  if(!head)
    return nullptr;
  if(p==-1)//头结点特判
    return head;
  int cnt=-1;
  listNode<T>*q=head;
  while(q->next){
    q=q->next;
    cnt++;
    if(cnt==p)
      return q;
  }
  return nullptr;
}
//插入操作
template <class T>
void linkList<T>::insert(int p,T val){
  listNode<T>*q=new listNode<T>(val);
  listNode<T>*tmp=find(p-1);
  q->next=tmp->next;
  tmp->next=q;
}
//删除操作
template <class T>
void linkList<T>::myDelete(int p){
  listNode<T>*tmp=find(p-1);
  listNode<T>*q=tmp->next;
  tmp->next=q->next;
  q->next=nullptr;
  delete q;
}
```

复杂度分析

单链表运算的时间复杂度 $O(n)$

- **定位：$O(n)$**
- 插入：$O(n)+O(1)$
- 删除：$O(n)+O(1)$

6.链表上实现的运算——双链表

```
template <class T>
class listNode{
public:
  T val;
  listNode<T>*next,*prev;
  listNode(T v, listNode<T>*n=nullptr, listNode<T>*p=nullptr)
  :val(v),next(n),prev(p){}
};

//双链表类定义
template <class T>
class linkList{
public:
  listNode<T>*head;

  listNode<T>*find(int p);
  void insert(int p,T val);
  void myDelete(int p);
};

//查找操作
template <class T>
listNode<T>* linkList<T>::find(int p){
  if(!head)
    return nullptr;
  if(p==-1)
    return head;
  int cnt=-1;
  listNode<T>*q=head;
  while(q->next){
    q=q->next;
    cnt++;
    if(cnt==p)
      return q;
  }
  return nullptr;
}

//插入操作
```

```
template <class T>
void linkList<T>::insert(int p,T val){
  listNode<T>*q=new listNode<T>(val);
  listNode<T>*tmp=find(p-1);
  q->next=tmp->next;
  tmp->next->prev=q;
  tmp->next=q;
  q->prev=tmp;
}

//删除操作
template <class T>
void linkList<T>::myDelete(int p){
  listNode<T>*q=find(p);
  if(q->next)
    q->next->prev=q->prev;
  q->prev->next=q->next;
  q->prev=q->next=nullptr;
  delete q;
}
```

7.顺序表和链表的比较

	顺序表	链表
存储结构	静态结构	动态结构
	从栈中申请空间	从堆中申请空间
	存储密度高	存储密度低
运算		
定位操作	支持随机访问	需要顺链逐个查找
修改操作	元素移动	修改指针
		便于插入、删除

- **顺序表**
  - 结点总数目大概可以估计
  - 线性表中结点比较稳定（插入删除少）
  - **n > DE/(P+E)**
- **链表**
  - 结点数目无法预知
  - 线性表中结点动态变化（插入删除多）
  - **n < DE/(P+E)**

n，当前元素的数目，
P，指针大小（通常为**4bytes)**
E，数据域大小
D，数组元素的最大数目