试题构成

一 单项选择题（本题共15小题，每小题2分，共30分）

二 填空题/判断题（本题共10小题，每题1分，共10分）

三 综合题（本题共5小题，共40分）

四 程序填空与算法阅读（本题共2小题，共10分）

五 编程题（本题共10分）

第一章 绪论

数据结构基本概念

数据结构定义：是相互之间存在一种或多种特定关系的数据元素的集合。数据之间不是相互独立的，他们之间有某种特定的关系，这种数据元素之间的关系，称为“结构”。

- **结构=关系+实体**
- **程序 = 数据结构 + 算法（ + 文档)**
- **逻辑结构：分类1：集合结构、线性结构、树形结构、图状结构**
  分类2：线性结构、非线性结构
- **存储结构：顺序存储、链式存储、索引存储、散列存储**

——各类型存储结构优缺点及适用情况

算法概念和算法评价

1、算法的定义

指一系列确定的而且是在有限步骤内能完成的操作。

2、算法的五个特性

- **有穷性、确定性、0至多个输入、1至多个输出、有效性(可行性)**

3、算法设计要求

- **正确性、可读性、健壮性、高效性**

4、算法性能的评价，大O法则

- **时间复杂度**

——语句的频度

- **空间复杂度**

——存储密度d=数据本身存储量/实际所占存储量

第二章 线性表

线性表基本概念

定义：若结构是非空有限集，则有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。 可表示为：

$$(a_1, a_2, \cdots\cdots, a_n)$$

- **基本操作：** CreateList、InitList、DestroyList、ListEmpty
  ListLength、GetElem、LocateElem、ListInsert
  ListDelete
- **特殊操作：线性表合并、拆分线性表（链表）、线性表倒置（链表）**

线性表顺序存储

- **顺序存储结构：顺序表**（用“物理位置”相邻来表示线性表中数据元素之间的逻辑关系）

  存储位置：$Loc(a_i) = Loc(a_1) + (i-1)*L$

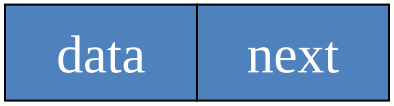
- 优点：结构简单；存储效率高，是紧凑结构；是随机存储结构，可直接存取

- 缺点：插入和删除操作需移动元素

```
#define LIST_INIT_LENTH 100 //或者N,或者是一个常数
#define LISTINCREMENT 10 //线性表存储空间的分配增量
typedef  struct {
    ElemType   *elem;
    int  length; //当前长度
    int  listsize; // 当前分配的存储容量
} SqList;
```

线性表链式存储

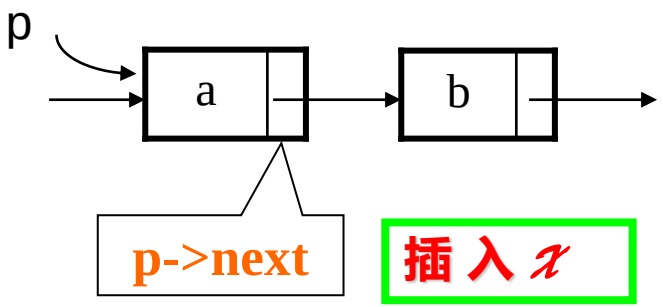
- **1、单链表：**在每个结点中除包含有数据域外，只设置一个指针域，用以指向其后继结点，这样构成的链接表称为线性单向链接表，简称单链表。

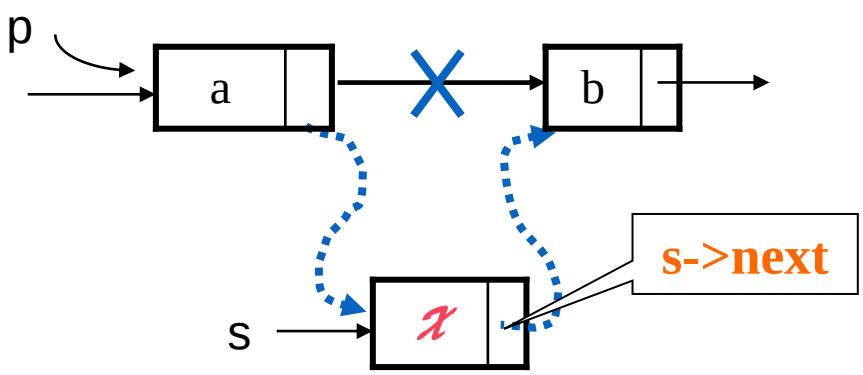



```
typedef struct LNode {
    ElemType        data;      //数据域
    struct LNode    *next;      //指针域
}LNode, *LinkList; // *LinkList为Lnode类型的指针
```

线性表链式存储

- **插入元素**　在链表中插入一个元素x的示意图如下：





链表插入的核心语句：

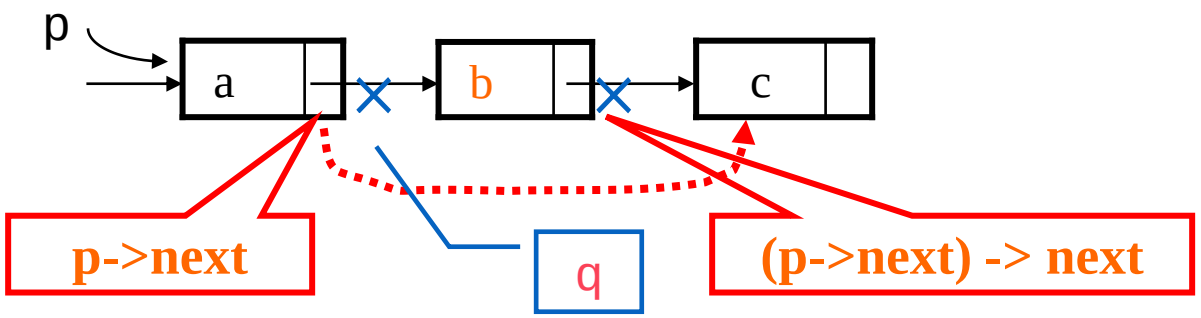
Step 1： s->next=p->next;
Step 2： p->next=s；

x 结点的生成方式：

```
s=(node*)malloc(m);
s->data=x;
s->next= ?
```

线性表链式存储

- **删除元素**　在链表中删除一个元素x的示意图如下：



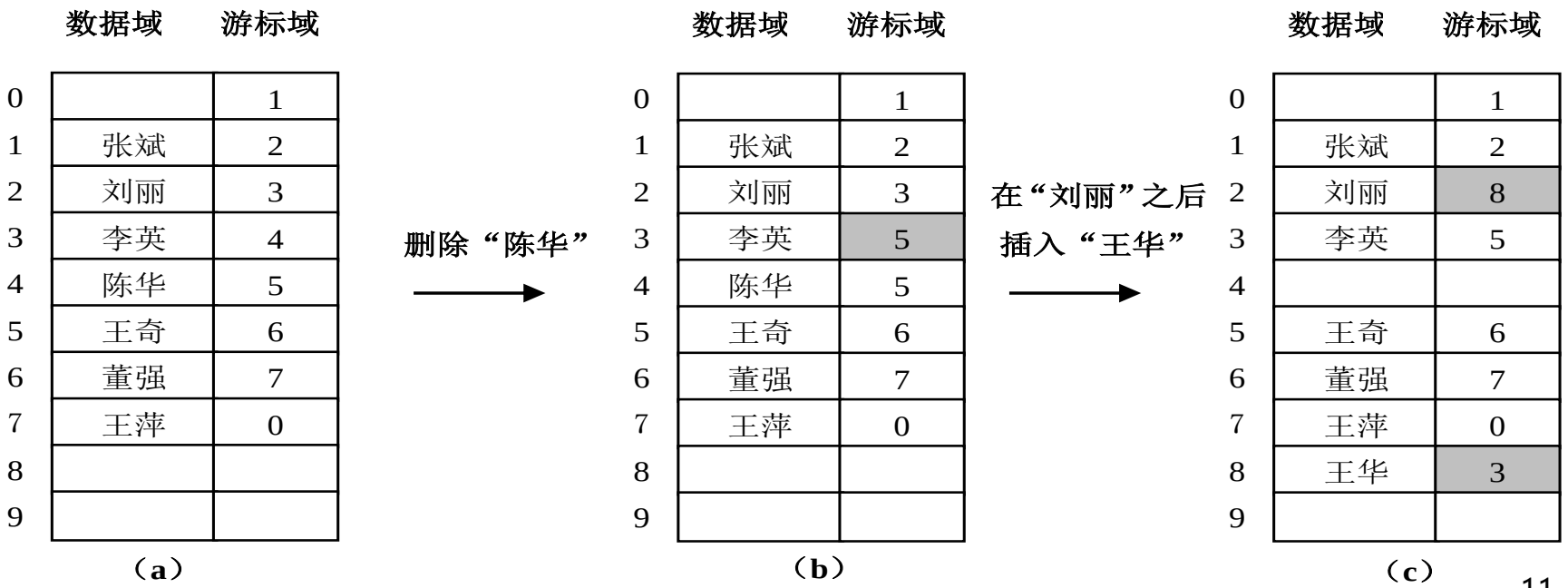
删除动作的核心语句（要借助辅助指针变量q）：

```
q = p->next;          //首先保存b的指针，靠它才能找到c;
p->next=q->next;  //将a、c两结点相连，淘汰b结点;
free(q)；             //彻底释放b结点空间
```

线性表链式存储

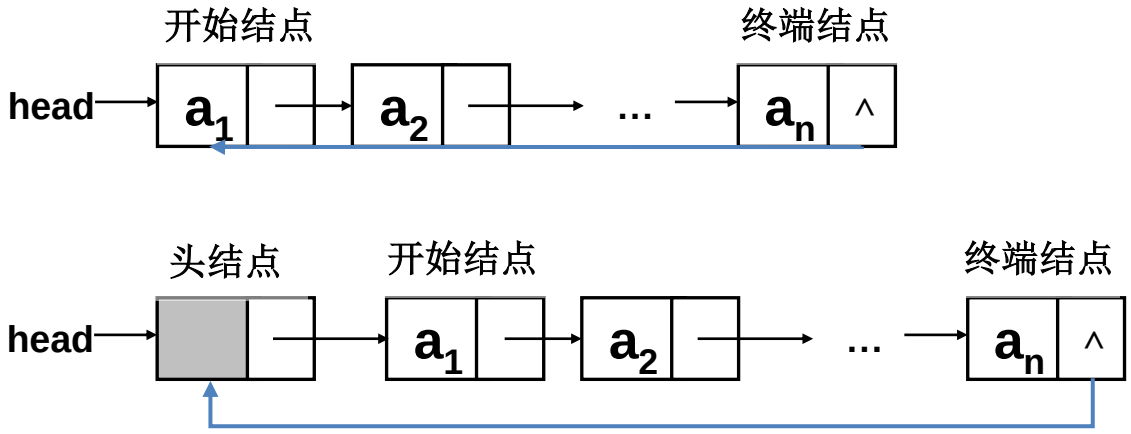
2、静态链表

```
#define MAXSIZE   1000
  //预分配最大的元素个数（连续空间）
  typedef   struct  {
        ElemType   data ;    //数据域
        int            cur ;     //指示域
   }component , SLinkList[MAXSIZE] ;
//这是一维结构型数组
```



线性表链式存储

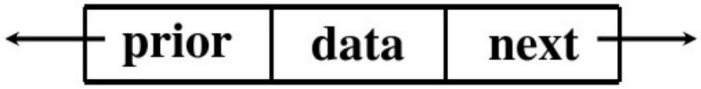
- **3、循环链表**



- 插入、删除类似单链表
- 判断链表遍历完否： p->next==head，则遍历完毕
- 判断是否空表：p->next==p，则链表为空或者只有一个元素

线性表链式存储

- **4、双向链表**

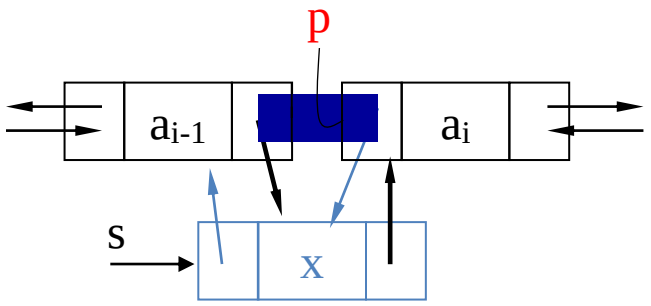



```
typedef struct DNode   {  /*定义双链表结点类型*/
     ElemType data;
     struct DNode *prior;    /*指向前驱结点*/
     struct DNode *next;      /*指向后继结点*/
 } DLinkList;
```

- 插入、删除类似单链表
- 判断链表遍历完否： p->next==head，则遍历完毕
- 判断是否空表：p->next==p，则链表为空或者只有一个元素

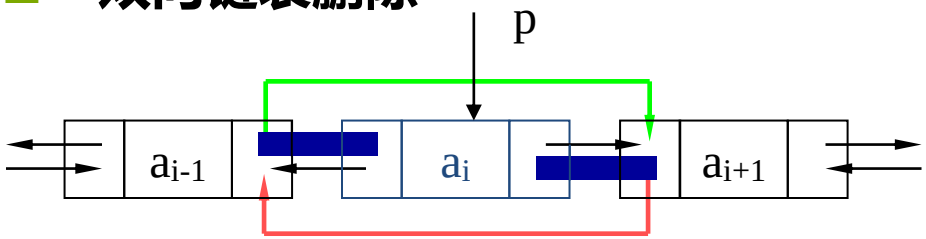
线性表链式存储

- **双向链表插入** 设p已指向第 i 元素，请在第 i 元素前插入元素 x



① a_{i-1}的后继从 a_i (指针是p)变为 x （指针是s) :
s->next = p ； p->prior->next = s ;
② a_i 的前驱从 a_{i-1} (指针是p->prior)变为 x (指针是s);
s->prior = p ->prior ; p->prior = s ;

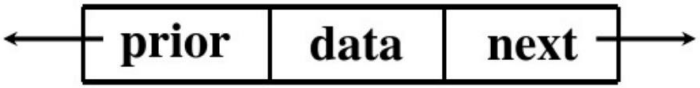
- **双向链表删除**



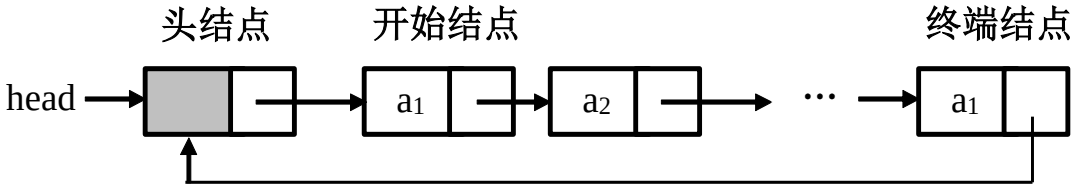
① 后继方向： a_{i-1}的后继由 a_i (指针p)变为 a_{i+1}(指针 p ->next);
p ->prior->next = p->next ;
② 前驱方向： a_{i+1} 的前驱由 a_i (指针p)变为a_{i-1} (指针 p -> prior);
p->next->prior = p ->prior ;

线性表链式存储

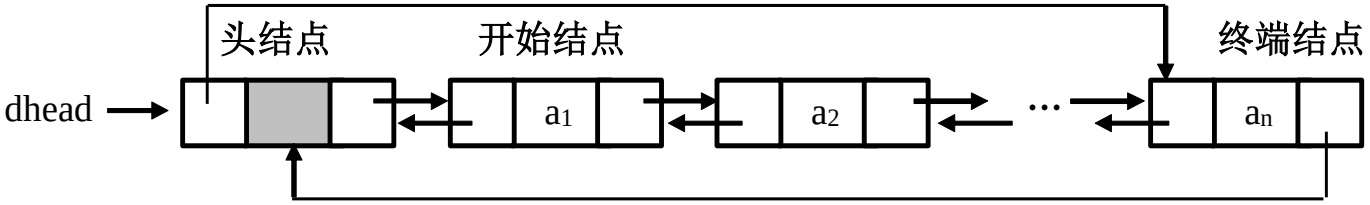
5、双向循环链表




```
typedef  struct  DuLNode{
    ElemType            data ;
    struct  DuLNode     * prior ;
    struct  DuLNode     * next ;
} DuLNode , * DuLinkList ;
```



(a)循环单链表



(b)循环双链表

第三章 栈和队列

栈

定义：栈(stack)是限制线性表中元素的插入和删除只能在线性表的同一端进行的一种特殊线性表。允许插入和删除的一端，为变化的一端，称为栈顶(Top)，另一端为固定的一端，称为栈底(Bottom)。

- **基本操作：**InitStack、Push、Pop、GetTop、IsEmptyStack、IsFullStack
- **栈是一种后进先出(Last In First Out)的线性表，简称为LIFO表，也是一种先进后出LIFO表**

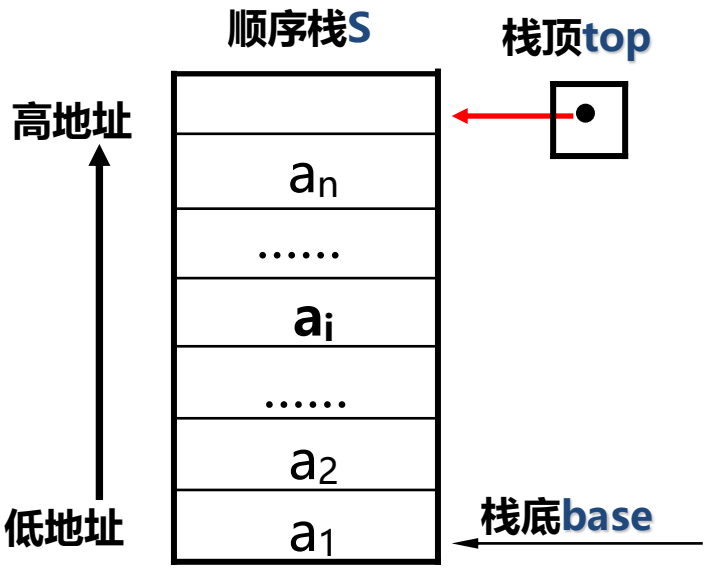
$S = (a_1, a_2, a_3, \ldots\ldots, a_{n-1}, a_n)$

a_1称为栈底元素

a_n称为栈顶元素

栈

- **顺序栈**



栈不存在的条件： base=NULL;

栈为空 的条件： base=top;

栈满的条件 ： top-base=stacksize;

一般栈顶不存放元素。

入栈口诀：堆栈指针top “先压后加” ：
S[top++]=an+1

出栈口诀：堆栈指针top “先减后弹” ：
e=S[--top]

若输入序列是 …,P_i…P_j…P_k…(i<j<k,输入序号)， 一定不存在这样的输出序列 …,P_k…P_i…P_j…

栈

- **顺序栈**
  - 栈空条件：s.top ==s.base　此时不能出栈
  - 栈满条件：s.top-s.base>=s.stacksize
  - 进栈操作：*s.top++=e;　或*s.top=e; s.top++;
  - 退栈操作：e=*--s.top;　或s.top--; e=*s.top;
  - 当栈满时再做进栈运算必定产生空间溢出，简称“上溢”；
  - 当栈空时,再做退栈运算也将产生溢出，简称为“下溢”。
- **链栈**
- **栈的应用**

队列

定义：只能在表的一端进行插入运算，在表的另一端进行删除运算的线性表。

- **基本操作：**InitStack、Push、Pop、GetTop、IsEmptyStack、IsFullStack、
- **只能在队首和队尾运算，且访问结点时依照先进先出（FIFO）的原则**
- 存储方式：顺序队列、循环队列、链队列

队列

- **顺序队列**

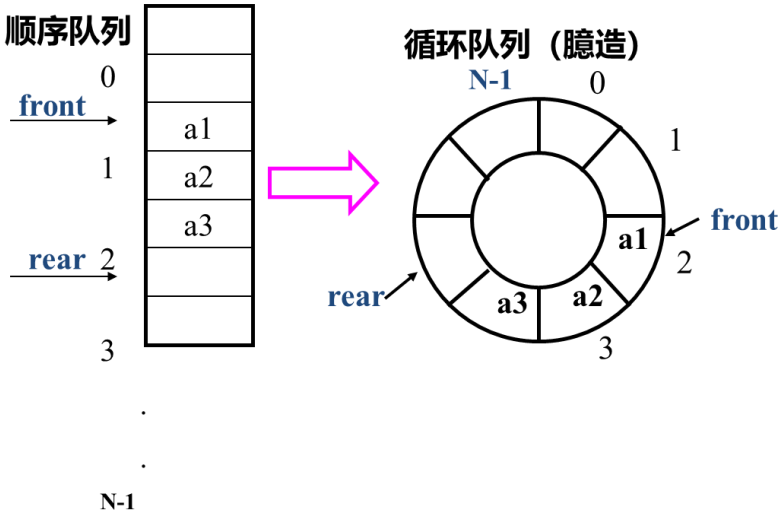
```
#define MAXSIZE 100
typedef struct {
   QElemType  *base;
   int   front;
   int   rear;
}SqQueue;
SqQueue  Q;
```

入队(尾部插入)： Q.base[rear]=e; Q.rear++ ;
出队(头部删除)： e=Q.base[front]; Q.front++;

- **空队列：** front=rear

队列

- **循环队列**



- **加设标识位**
  - 队空： front=rear
  - 队满： front=(rear+1)%N
  - 队长： L=（N＋rear－front）％N

第四章 串

串

定义：串(string)即字符串，是由零个或多个字符组成的有限序列，是数据元素为单个字符的特殊线性表。

记为： $s = ‘a_1\ a_2\ \ldots\ldots\ a_n’\ \ (n \geq 0)$

- 空串是任意串的子串；任意串S都是S本身的子串，除S本身外，S的其他子串称为S的真子串
- **串相等：**当且仅当两个串的长度相等并且各个对应位置上的字符都相同时，这两个串才是相等的
- **串的操作：**以“串的整体”为操作对象

串

- **串的操作：**StrInsert、StrAssign、SubString、StrLength、StrCompare、Concat、ClearString
- **串的模式匹配**
  - **BF算法**
  - **KMP算法**

令next[j] = k（k 与j 显然具有函数关系），则

$$next[j] = \begin{cases} 0 & \text{当 } j = 1 \text{ 时} \quad \text{//不比较} \\ \max\{k \mid 1<k<j \text{ 且 } `P_1 \cdots P_{k-1}` = `P_{j-(k-1)} \cdots P_{j-1}`\} \\ 1 & \text{其他情况} \end{cases}$$

取P首之后与Pj处之前最大的相同子串

next[j]值越小，模式串向右移动越快

第五章 数组和广义表

数组

定义：由一组名字相同、下标不同的变量构成。

- **列向量、行向量**

$$A=(\alpha_0\ \alpha_1\ \cdots\ \alpha_j\ \cdots\ \alpha_{n-1})$$

$$A_{m\times n}=\begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0j} & \cdots & a_{0n-1} \\ a_{10} & a_{11} & \cdots & a_{1j} & \cdots & a_{1n-1} \\ \vdots & \vdots & & & & \\ a_{i0} & a_{i1} & \cdots & a_{ij} & \cdots & a_{in-1} \\ \vdots & \vdots & & & & \\ A_{m-10} & a_{m-11} & \cdots & a_{m-1j} & \cdots & a_{m-1n-1} \end{pmatrix}$$

$$A_{m\times n}=\begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0j} & \cdots & a_{0n-1} \\ a_{10} & a_{11} & \cdots & a_{1j} & \cdots & a_{1n-1} \\ \vdots & \vdots & & & & \\ a_{i0} & a_{i1} & \cdots & a_{ij} & \cdots & a_{in-1} \\ \vdots & \vdots & & & & \\ A_{m-10} & a_{m-11} & \cdots & a_{m-1j} & \cdots & a_{m-1n-1} \end{pmatrix} \begin{matrix} B \\ \| \\ \beta_0 \\ \beta_1 \\ \vdots \\ \beta_i \\ \vdots \\ \beta_{m-1} \end{matrix}$$

- **基本操作：**InitArray、DestroyArray、GetValue、 SetValue

数组

- **n维数组顺序存储**

$Loc(j_1, j_2, ..., j_n) = Loc(0,0,...,0) + \sum_{i=1}^{n} c_i j_i$　　其中 $c_n = L$，$c_{i-1} = b_i \times c_i$，$1 < i \leq n$。

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

——对称矩阵顺序存储

$$k = \begin{cases} \frac{i(i-1)}{2} + j - 1 & \text{当 } i \geqslant j \\ \frac{j(j-1)}{2} + i - 1 & \text{当 } i < j \end{cases}$$

- **稀疏矩阵：**设矩阵 A 中有 s 个非零元素。令 e = t/(m*n),称 e 为矩阵的稀疏因子，一般认为 e≤0.05 时称之为稀疏矩阵
- **稀疏矩阵压缩：**三元组表示法、十字链表、行逻辑链接顺序表

广义表

定义：广义表是线性表的推广，也称为列表（lists）

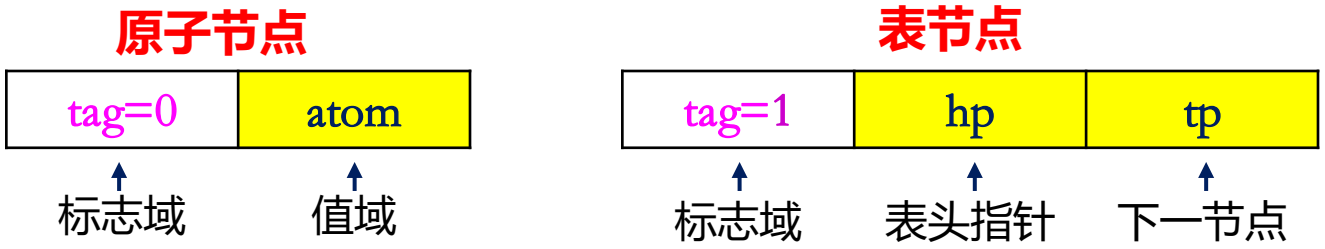
记为： $LS = (a_1, a_2, \ldots\ldots, a_n)$

广义表名　表头(Head)　表尾 (Tail)

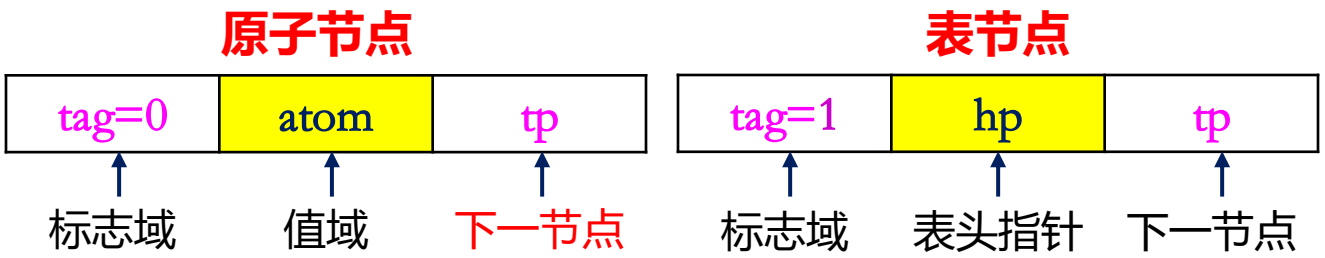
n是表长

- **操作**：GetHead、GetTail
- **存储结构**

方法一：



方法二：



第六章 树和二叉树

基本概念

定义：树(Tree)是n(n>=0)个结点构成的有限集合。

- n=0时称为空树
- 非空树满足的条件：

  ①有且仅有一个称为根（Root）的结点；

  ②n>1时，其余结点可分为m(m>0)个互不相交的有限集合 $T_1 \ldots T_m$，其中每个集合又是一棵树，称为子树

- **树的表示法：**树形、文氏图、凹入、广义表
- 基本概念：结点的度、树的度、m次树、m叉树、分支结点、叶子结点、路径、路径长度；孩子、双亲、兄弟、子孙、祖先结点；结点的层次、树的深度；有序树、无序树；森林

二叉树

定义：一棵二叉树是结点的一个有限集合，该集合或者为空，或者是由一个根结点加上两棵分别称为左子树和右子树、互不相交的二叉树组成。

- **特点：**度≤2、有序树，五种形态
- **满二叉树、完全二叉树**
- **顺序存储：**利用完全二叉树特性。会造成空间的浪费
- **二叉链表：**

lchild	data	rchild

注意：在n个结点的二叉链表中，有n+1个空指针域:2n-(n-1)=n+1

- **三叉链表：**

lchild	data	parent	rchild

- **静态二叉链表和静态三叉链表**

二叉树

- **性质：**
  - **性质1：在二叉树的第 i 层上至多有2^{i-1}个结点 ($i \geq 1$)。**
  - **性质2：深度为k 的二叉树上至多含2^k-1 个结点（k≥1）。**
  - **性质3：对任何一棵二叉树，若它含有n_0个叶子结点（度为0结点）、n_2个度为2 的结点，则必存在关系式：$n_0 = n_2+1$**
  - **性质4：具有n个(n>0)结点的完全二叉树深度为：$\lfloor \log_2 n \rfloor + 1$。**
  - **性质5：若对含n 个结点的完全二叉树从上到下且从左至右进行1 至 n 的编号，则对完全二叉树中任意一个编号为i 的结点：**

(1) 若$i=1$，则该结点是二叉树的根，无双亲，否则，编号为$\lfloor i/2 \rfloor$的结点为其双亲结点；

(2) 若$2i>n$，则该结点无左孩子；否则，编号为$2i$ 的结点为其左孩子结点；

(3) 若$2i+1>n$，则该结点无右孩子结点，否则，编号为$2i+1$ 的结点为其右孩子结点。

二叉树

- **二叉树遍历**

先序	中序	后序	层次
若二叉树为空，则空操作返回；否则： ①访问根结点； ②先序遍历根结点的左子树； ③先序遍历根结点的右子树	若二叉树为空，则空操作返回；否则： ①中序遍历根结点的左子树； ②访问根结点； ③中序遍历根结点的右子树。	若二叉树为空，则空操作返回；否则： ①后序遍历根结点的左子树； ②后序遍历根结点的右子树； ③访问根结点。	从二叉树的第一层（即根结点）开始，从上至下逐层遍历，在同一层中，则按从左到右的顺序对结点逐个访问。

- **先序、中序、后序遍历的非递归：使用栈**
- **先序+中序 或中序+后序 均可唯一地确定一棵二叉树**
- **二叉树应用：**数学表达式

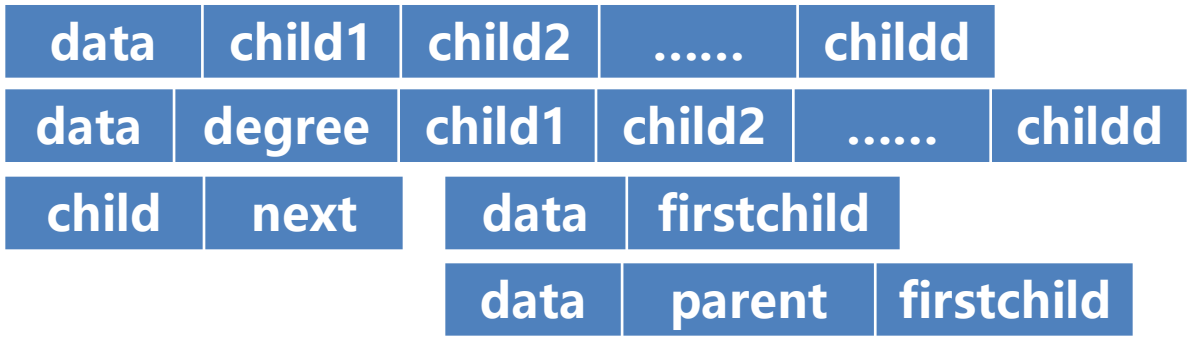
二叉树

- **二叉树线索化：**在遍历过程中修改空指针的过程：
  - ① 将空的lchild改为结点的直接前驱；
  - ② 将空的rchild改为结点的直接后继。
  - ③ 非空指针呢？仍然指向孩子结点（称为“正常情况”）
- **先序、中序**、后续线索二叉树
  - 二叉树线索化优点是便于是查找前驱结点和后继结点
  - 在线索化二叉树中，并不是每个结点都能直接找到其后继

树和森林

- **树的存储结构：**
  - 双亲表示法：

  - 孩子表示法：



  - 孩子—兄弟表示法：

- **树转化为二叉树：**加线、去线、层次调整
- **森林转化为二叉树：**1) 各森林先各自转为二叉树、依次连到前一个二叉树的右子树上；2) 森林直接变兄弟，再转为二叉树。
- 二叉树转化为树和森林：加线、去线、层次调整

树和森林

- **树的遍历：**

先根	后根	层次
①访问根结点； ②依次先根遍历根结点的每棵子树。	**①依次后根遍历根结点的每棵子树； ②访问根结点。**	**从树的第一层（即根结点）开始，从上至下逐层遍历。**
树的先根遍历等价于二叉树的先序遍历	树的后根遍历等价于二叉树的中序遍历	

- **森林的遍历：**

先序	中序	层次
若森林为空，返回。 ①访问第一棵树的根结点； ②先序遍历第一棵树的根结点的子树森林； ③先序遍历除去第一棵树之后剩余的树构成的森林。	**若森林为空，返回。 ①中序遍历森林中第一棵树的根结点的子树森林； ②访问第一棵树的根结点； ③中序遍历除去第一棵树之后剩余的树构成的森林。**	**访问森林中第一棵树的根节点；依次从左到右对森林中的每一颗树进行层次遍历。**
对应二叉树的先序遍历	对应二叉树的中序遍历	

赫夫曼树

- **结点带权的路径长度**——从该结点到树根之间的路径长度与结点上权的乘积 $WPL=\sum_{k=1}^{n} w_k l_k$
- **赫夫曼树：**给定一组具有确定权值的叶子结点，**带权路径长度最小**的二叉树。又称**最优二叉树**。
- **特点：**1)权值越大的叶子结点越靠近根结点，而权值越小的叶子结点越远离根结点；2)只有度为0（叶子结点）和度为2（分支结点）的结点，不存在度为1的结点。
- **n_0个叶子结点的哈夫曼树共有$2n_0-1$个结点**
- **赫夫曼树构造过程**——选取与合并、删除与加入
- **赫夫曼编码**

第七章 图

图

图：图G是由两个集合V(G)和E(G)组成的,记为。

$$G = (V, E)$$

- V(G)是顶点的非空有限集
- E(G)是边的有限集合，边是顶点的无序对或有序对

- 无向图、有向图
- 完全图：无向完全图（完全有向图有n(n-1)条边）

  有向完全图（完全无向图有n(n-1)/2 条边）
- 稀疏图——若边或弧的个数 e<nlogn，则称作稀疏图，否则成稠密图
- 权、带权图（网）
- 子图、邻接点
- 顶点的度：无向图（与边数相同）、有向图（入度、出度）

图

- **路径、路径长度**（非带权图的路径长度是指此路径上边的条数、带权图的路径长度是指路径上各边的权之和）
- **简单路径**：路径上各顶点 v1,v2,...,vm 均不互相重复。
- 回路（环）、简单回路
- **连通图**：针对无向图，非连通图的极大连通子图叫做连通分量。
- **强连通图**：针对有向图，非强连通图的极大强连通子图叫做强连通分量。

连通图再添加任何一个顶点就会使之变为非连通图

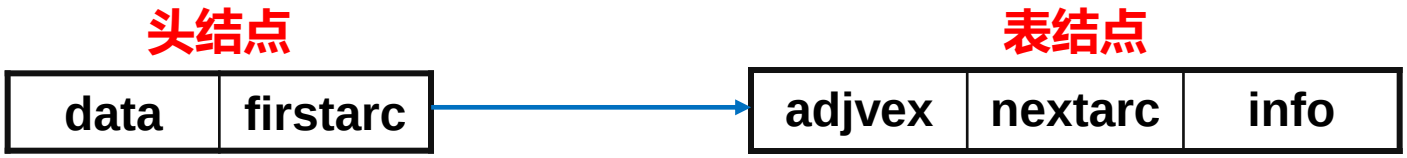
- 图生成树和森林

图的存储结构

- **顺序存储结构：数组表示法（邻接矩阵）**
  - **如果G是无向图，则：** $A[i][j]=\begin{cases}1: 若(v_i,v_j)\in VR\\0: 其他\end{cases}$
  - **如果G是有向图，则：** $A[i][j]=\begin{cases}1: <v_i,v_j>\in VR\\0: 其他\end{cases}$
  - **如果G是带权无向图，则：** $A[i][j]=\begin{cases}w_{ij}: 若v_i\neq v_j且(v_i,v_j)\in VR\\\infty: 其他\end{cases}$
  - **如果G是带权有向图，则：** $A[i][j]=\begin{cases}w_{ij}: 若v_i\neq v_j且<v_i,v_j>\in VR\\\infty: 其他\end{cases}$
- **无向图的度，有向图的入度、出度**

图的邻接表（链式）存储

- 对每个顶点vi 建立一个单链表，把与vi有关联的边的信息链接起来



邻接表不唯一，因各个边结点的链入顺序是任意的

- 无线图每条边以所依附的两个顶点的形式出现2次
- 有向图：出度： OD(Vi) = 单链出边表中链接的结点数

  入度： ID(Vi) = 邻接点域的值为i的结点个数

  度： TD(Vi) = OD(Vi) + ID(Vi)

```
typedef struct VNode {
  VertexType  data;
  ArcNode  *firstarc;
} VNode,
AdjList[MAX_VERTEX_NUM];
```

```
typedef struct ArcNode {
  int  adjvex;
  struct ArcNode  *nextarc;
  InfoType  *info;
} ArcNode;
```

```
typedef struct {
  AdjList  vertices;
  int  vexnum, arcnum;
  int  kind;
} ALGraph;
```

图的邻接表（链式）存储

- **邻接矩阵与邻接表表示法的关系**
  - 联系：邻接表中每个链表对应于邻接矩阵中的一行，链表中结点个数等于一行中非零元素的个数。
  - 区别：

① 对于任一确定的无向图，邻接矩阵是唯一的（行列号与顶点编号一致），但邻接表不唯一（链接次序与顶点编号无关）。

② 邻接矩阵的空间复杂度为$O(n^2)$,而邻接表的空间复杂度为$O(n+e)$。

- **各种存储结构的适用类型**
  - 数组(邻接矩阵)：有向图、无向图；稠密图
  - 邻接表（逆邻接表）：有向图、无向图；稀疏图
  - 十字链表：有向图
  - 邻接多重表：无向图

图的遍历

- **深度优先搜索遍历（采用递归）**

  从图中某个顶点V0 出发，访问此顶点，然后依次从V0的各个未被访问的邻接点出发深度优先搜索遍历图，直至图中所有和V0有路径相通的顶点都被访问到。

- 非连通图的深度优先搜索遍历

- **广度优先遍历（采用辅助队列）**

  - 从起始点V到其余各顶点必定存在路径，并且访问顶点的次序是按顶点的路径长度递增进行的。

1. 访问顶点 V ；
2. for (W_1、W_2、W_3)
若该邻接点W_i未被访问，
则深度优先搜索遍历该子图。

1. 访问顶点 V ；
2. for (W_1、W_2、W_3)
若该邻接点W_i未被访问，则访问W_i
3. 若W_i 的邻接点未被访问，则访问之
4. ……。

图生成树

- **连通图的生成树：**是连通图的一个极小连通子图，它含有图中全部顶点，但只有n-1条边。
  - 一个有n个顶点的连通图的生成树只有n-1条边
  - 在生成树中再加一条边必然形成回路
  - 生成树中任意两个顶点间的路径是唯一的
  - 从不同顶点出发或搜索次序不同，可得到不同的生成树
- 非连通图的生成森林：由若干棵生成树组成，含图中全部顶点，但构成这些树的边是最少的。
- **最小代价生成树：**在一个连通网的所有生成树中，各边的代价之和最小的那棵生成树称为该连通网的最小代价生成树，简称为最小生成树(MST)。

最小生成树

- **普里姆（Prim）算法**

  ①初始状态：U ={u_0}，（$u_0 \in V$），TE= φ，

  ②在u∈U,v∈V-U所有的边(u,v)∈E中，找一条代价最小的边(u0,v0)，并将边(u0,v0)并入集合TE，同时v0并入U。

  ③重复（2），直到U=V为止。

  此时TE中必有n-1条边，T = (V,{TE}) 就是最小生成树。

Colsedge[i]	adjvex	lowcost

要求：使Colsedge[i].lowcost = min((u ,vi)) u ∈ U

最小生成树

- **克鲁斯卡尔(Kruskal)算法**

  ①初始状态为只有n个顶点而无边的非连通图T=(V,{Φ})，每个顶点自成一个连通分量

  ②在E中选取代价最小的边，若该边依附的顶点落在T中两个不同的连通分量上，则将此边加入到T中；否则，舍去此边，选取下一条代价最小的边。

  ③依此类推，直至T中所有顶点都在同一连通分量上为止

算法名	Prim	Kruskal
时间复杂度	$O(n^2)$	O(eloge)
适应范围	稠密图	稀疏图

活动网络

- **AOV网**(Activity On Vertices)—用顶点表示活动的网络

  AOV网定义：若用有向图表示一个工程，在图中用顶点表示活动，用弧表示活动间的优先关系。v_i 必须先于活动 v_j 进行。则这样的有向图叫做用顶点表示活动的网络，简称 AOV。

- **拓扑排序（使用栈辅助）**

  ①输入AOV网络。令 n 为顶点个数。

  ②在AOV网络中选一个没有直接前驱的顶点, 并输出之;

  ③从图中删去该顶点, 同时删去所有它发出的有向边;

  ④重复以上 2、3 步, 直到:

  全部顶点均已输出，拓扑有序序列形成，拓扑排序完成；

  或者，图中还有未输出的顶点，但已跳出处理循环。这说明图中还剩下一些顶点，它们都有直接前驱，再也找不到没有前驱的顶点了。这时AOV网络中必定存在有向环。

活动网络

- **AOE网**(Activity On Edges)—用边表示活动的网络

  AOE网定义：如果在无环的带权有向图中，用有向边表示一个工程中的活动，用边上权值表示活动持续时间，用顶点表示事件，则这样的有向图叫做用边表示活动的网络，简称 AOE。

- **关键路径**

  ①输入e条弧（i，j），建立AOE网的存储结构。

  ②从源点v_0出发，令ve[0]＝0按拓扑有序求其余各顶点的最早发生时ve[i]（1≤i≤ n-1）。如果得到的拓扑有序序列中顶点个数小于网中顶点数n，则说明网中存在环，不能求关键路径，算法终止；否则执行步骤（3）。

  ③从汇点v_n出发，令vl[n-1]= ve[n-1]，按逆拓扑有序求其余各顶点的最迟发生时间vl[i] (n-2 ≥i≥ 2)；

  ④根据各顶点的ve和vl值，求每条弧s的最早开始时间e(s)和最迟开始时间l(s)。若某条弧满足条件e(s)=l(s)，则为关键活动。

最短路径

单源最短路径—用Dijkstra（迪杰斯特拉）算法

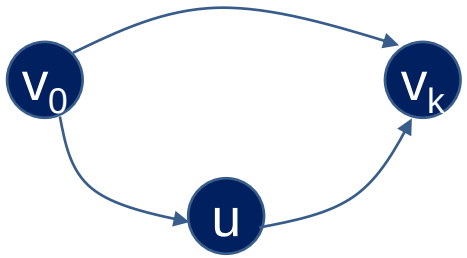
①先找出从源点v_0到各终点v_k的直达路径（v_0,v_k），即通过一条弧到达的路径；

②从这些路径中找出一条长度最短的路径（v_0,u），然后对其余各条路径进行适当调整：

若在图中存在弧（u,v_k），且（v_0,u）+（u,v_k）<（v_0,v_k），则以路径（v_0,u,v_k）代替（v_0,v_k）。其中，（v_0,u）为上次求得的v_0到u最短路径；

③在调整后的各条路径中，再找长度最短的路径，依此类推。

按路径“长度” 递增的次序来逐步产生最短路径



如果：（v_0,u）+（u,v_k）<（v_0,v_k）

则：以路径（v_0,u,v_k）代替（v_0,v_k）

最短路径

- **所有顶点间的最短路径—用Floyd（弗洛伊德）算法**

  在$D^{(k-1)}$中，对于每两个顶点u、v，在$D^{(k-1)}$中存储着一条路径u...v。试着把a加到u、v的路径上能否，得到一条更短的路径，即如果u...a+a...v<u...v的话，能够找到一条更短的路径。

$$D^{(k)}[i][j]=Min\{D^{(k-1)}[i][j], D^{(k-l)}[i][k]+D^{(k-l)}[k][j]\}$$

依次将每个点作为“中间点”去做更新

第九章 查找

基本概念

- **平均查找长度ASL**

$$ASL = \sum_{i=1}^{n} P_i \cdot C_i$$

n：记录个数；
P_i：查找第i个记录的查找概率；
C_i：找到第i个记录时所经历的比较次数

- **静态查找**
  - **顺序查找（线性查找）**
  - **折半查找（二分或对分查找）——查找不成功： high<low**

    mid=⌊(low+high)/2⌋

    若 ST.elem[mid].key < key；令：low =mid+1;重算 mid

    若 ST.elem[mid].key > key；令：high =mid–1;重算 mid

    若 ST.elem[mid].key = key，说明查找成功，元素序号=mid
  - **分块查找（索引顺序查找）**

    块间有序，块内有序与否均可以

二叉排序树

- **二叉排序树**

  或是一棵空树；或者是具有如下性质的非空二叉树

  ①左子树的所有结点均小于根的值；

  ②右子树的所有结点均大于根的值；

  ③它的左右子树也分别为二叉排序树。

- **二叉排序树的插入和删除**

  - p为叶子： 删除此结点时，直接修改*f指针域即可；
  - p只有一棵子树（或左或右）：令P_L或P_R为*f的左孩子即可；
  - p有两棵子树：情况最复杂 →

    法1：令p的左子树为 f的左子树，p的右子树接为s的右子树；
    法2：直接令s代替p，s的左子树接为P_L

平衡二叉树

- **平衡二叉排序树**

  平衡二叉树或者是一棵空树，或者是具有下列性质的二叉树：它的左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差（平衡因子）的绝对值不超过1。

- **最小不平衡子树**

- **平衡二叉树插入**
  - **LL平衡旋转（单右旋）**
  - **RR平衡旋转（单左旋）**
  - **LR平衡旋转（先左后右）**
  - **RL平衡旋转（先右后左）**

- **平衡二叉树删除**

B-树

- **B-树**

  B-树是一种平衡的多路查找树。一棵m阶的B-树，或为空树，或为满足下列特性的m叉树：

  ①树中每个结点至多有m棵子树，m-1个关键字；

  ②若根结点不是叶子结点，则至少有两棵子树；

  ③除根之外的所有非终端结点至少有$\lceil m/2 \rceil$棵子树，至少有$\lceil m/2 \rceil$ -1个关键字。

- **B-树插入：**在插入新关键字时，需要自底向上分裂结点（上溢）

- **B-树删除：**

  ①简单删除：删除前该结点中关键字个数$n \geq \lceil m/2 \rceil$；

  ②结点联合调整：若被删关键字所在结点删除前关键字个数$n=\lceil m/2 \rceil-1$，向兄弟“借”；

  ③结点合并：被删除关键字所在的结点和其相邻的兄弟结点中的关键字个数均等于$\lceil m/2 \rceil-1$。结点合并后要继续检查双亲结点是否下溢。

哈希表

- **常用的哈希函数构造方法有**

  直接定址法、数字分析法、平方取中法、折叠法、除留余数法、乘余取整法、随机数法

- **冲突处理方法**

  开放定址法：线性探测法、二次探测法、随机探测法

  链地址法（拉链法）、再哈希法（双哈希函数法）、建立一个公共溢出区

第十章 排序

排序

- **稳定性—若两个记录A和B的关键字值相等，但排序后A、B的先后次序保持不变，则称这种排序算法是稳定的。**
- **插入排序：**
  - 直接插入排序、折半插入排序、2-路插入排序
  - 表插入排序：在顺序存储结构中，给每个记录增开一个指针分量，在排序过程中将指针内容逐个修改为已经整理（排序）过的后继记录地址。
    - ✓ 优点：在排序过程中不移动元素，只修改指针。
    - ✓ 难点：重排过程（p>i，直接互换；p<i，继续顺链查找直到p>=i为止）
  - 希尔排序：先将整个待排记录序列分割成若干子序列，分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行一次直接插入排序。

交换排序

- **冒泡排序：**每趟不断将相邻两记录两两比较，并按“前小后大”（或“前大后小”）规则交换——从后往前循环
- **快速排序：**
  - 从待排序列中任取一个元素 (例如取第一个) 作为轴心(pivot)，所有比它小的元素一律前放，所有比它大的元素一律后放，形成左右两个子表；
  - 然后再对各子表重新选择轴心元素并依此规则调整，直到每个子表的元素只剩一个。此时便为有序序列了。

设待划分的序列是*r[s]~r[t]*，设参数*i*，*j*分别指向子序列左、右两端的下标*s*和*t*，令*r[s]*为轴值。

①*j*从后向前扫描，直到*r[j]<r[i]*，将*r[i]*移动到*r[i]*的位置，使关键码小(同轴值相比)的记录移动到前面去;

②*i*从前向后扫描，直到*r[i]>r[j]*，将*r[i]*移动到*r[j]*的位置，使关键码大(同轴值比较)的记录移动到后面去;

③重复上述过程，直到*i=j*。

选择排序

- **简单选择排序：**每经过一趟比较就找出一个最小值，与待排序列最前面的位置互换即可。
- **树形选择排序：**首先对 *n* 个记录的关键字进行两两比较，得到⌈*n*/2⌉个优胜者(关键字小者)，作为第一步比较的结果保留下来。然后在这⌈*n*/2⌉个较小者之间再进行两两比较，…，如此重复，直到选出最小关键字的记录为止。
- **堆排序：**
  - 堆：设有n个元素的序列 $k_1,k_2,\ldots,k_n$，当且仅当满足下述关系之一时，称之为堆。

$$\begin{cases} k_i \le k_{2i} \\ k_i \le k_{2i+1} \end{cases} \quad \text{或者} \quad \begin{cases} k_i \ge k_{2i} \\ k_i \ge k_{2i+1} \end{cases} \qquad i=1, 2,\ldots n/2$$

  - 堆的建立：从最后一个非终端结点开始往前逐步调整，让每个双亲大于（或小于）子女，直到根结点为止
  - 堆顶元素删除

归并排序

- **基本思想是：将两个（或以上）的有序表组成新的有序表**

  把一个长度为 n 的无序序列看成是 n 个长度为 1 的有序子序列，首先做两两归并，得到 $\lceil n/2 \rceil$ 个长度为 2 的有序子序列；再做两两归并，…，如此重复，直到最后得到一个长度为 n 的有序序列。

排序方法	时间复杂度	时间复杂度		稳定性	附加存储	
		最好	最差		最好	最差
直接插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	√	$O(1)$	
折半插入排序	$O(n\log_2 n)$	$n\log_2 n$		√	$O(1)$	
2-路插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	√	$O(1)$	
表插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	√	$O(n)$	
希尔排序	$O(n^{1.25})$	$O(n)$	$O(1.6n^{1.25})$	X	$O(1)$	
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	√	$O(1)$	
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	X	$O(\log_2 n)$	$O(n^2)$
简单选择排序	$O(n^2)$	n^2		X	$O(1)$	
树形选择排序	$O(n\log_2 n)$	$O(n\log_2 n)$		√	$O(n)$	
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$		X	$O(1)$	
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$		√	$O(n)$	
基数排序	O(d(n+radix))	O(d (n+radix))		√	O(n+radix）	

正在答疑