GitHub: Leslan8

数据结构背诵笔记

数据结构概念

1、数据结构及算法的相关概念和术语（务必背下来！！）

（1） 数据结构及算法的概念；

数据：所有能输入到计算机中并被计算机程序处理的符号的总称。

数据元素：数据的基本单位，一个数据项可由若干个数据项组成。

数据项：构成数据元素的不可分割的最小单位。

*** 关系：**数据>数据元素>数据项

数据对象：性质相同的数据元素的集合。

数据类型：原子类型、结构类型、抽象数据类型。

数据结构：相互之间存在一种或多种特定关系的数据元素的集合。

（2）数据的逻辑结构和存储结构；

数据结构三要素： 数据的逻辑结构、数据的存储结构、数据的运算

数据的逻辑结构：线性结构（线性表）、非线性结构（集合、树和图）。

数据的存储结构：

顺序存储：

优点：可以随机存取，每个元素占用最少存储空间

缺点：可能产生较多碎片现象

链式存储：

优点：不会出现碎片现象

缺点：每个元素占用较多存储空间，只能实现顺序存储

索引存储：

优点：检索速度快

缺点：增加了附加的索引表，会占用较多存储空间

散列存储：

优点：检索、增加和删除结点的操作都很快

缺点：可能出现存储单元的冲突，解决冲突会增加时间和空间的开销

（3）算法的定义及特性；

算法：对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作。

算法的特性：有穷性、确定性、可行性、输入、输出

算法设计的要求：正确性、可读性、健壮性、效率与低存储量需求

* 同一个算法，实现语言的级别越高，执行效率就越低

（4）算法时间复杂度和空间复杂度的分析方法。

时间复杂度：主要分析语句频度（一条语句在算法中被重复执行的次数）之和 T(n)的数量级

加法原则：$T(n)=T1(n)+T2(n)=O(max(f(n),g(n)))$

乘法原则：$T(n)=T1(n)\cdot T2(n)=O(f(n)\cdot g(n))$

*** 常见时间复杂度比较：**$O(1)<O(n)<O(nlogn)<O(n^2)<O(n^3)<O(2^n)<O(n!)<O(n^n)$

空间复杂度：算法所耗费的存储空间，$S(n)=O(g(n))$

算法原地工作是指算法所需辅助空间是常量，即 O(1)

线性表

1、顺序表的特点：

可以进行随机存取，即通过首地址和元素序号可以在 O(1)的时间内找到指定元素，但插入和删除元素操作需要移动大量元素

2、线性表的链式存储（单链表）：

单链表结构的定义：（其实就是多个指针）

```
typedef struct LNode{
    Elemtype data;
    struct LNode *next;
}LNode,*LinkList;
```

引入头结点的单链表：（特别要注意链表是否带头结点，无头节点的链表首个结点要特殊处理！）

头结点的数据域可以不设任何信息，也可以记录表长等信息，头结点的指针域指向线性表的第一个结点。

*** 引入头结点的两个优点：**

(1) 由于开始结点的位置被存放在头结点的指针域中，所以在链表的第一个位置上的操作就和在表的其它位置上操作一致，无须进行特殊处理

(2) 无论链表是否为空，其头指针是指向头结点的非空指针（空表中头结点的指针域空），因此空表和非空表的处理也就统一了

*** 动态分配语句：L.data=(ElemType*)malloc(sizeof(ElemType)*InitSize);**

（这个实在是太重要了，背下来！！！）

PS.ElemType 就是类型如 int 类型等；InitSize 是大小，如果是动态生成一个结点那就是 1，若要生成 50 个结点那就是 50。

3、循环链表。

- **循环单链表：（重点了解！）**

表尾结点的 next 域指向 L，表中没有指针域为 NULL 的结点，因此循环单链表的判空条件是看它的头结点的指针域是否等于头指针；循环单链表在任何一个位置上插入和删除操作都是等价的，无需判断是否表尾；循环单链表可以从表中任一结点开始遍历整个链表。循环单链表不设头指针而只设尾指针，可使一些操作效率更高。

- **循环双链表：**

与循环单链表不同的是，循环双链表中头结点的 prior 指针还指向表尾结点；在循环双链表 L 中，某结点*p 为尾结点时，p->next==L；当循环双链表为空时，对其头结点*p，有 p->prior=p->next==L。

- **双向链表：**

其结点有两个指针域，一个指向直接后继，另一个指向直接前趋，双向链表的插入、删除结点算法的时间复杂度为 O(1)。

简答题 1：为什么在单循环链表中设置尾指针比设置头指针更好？（考试重点考察带尾指针的单链表，可以构造队列！入队出队都很方便）（这也给我们一个解题的思路，从反方向解题！）

答：尾指针是指向终端结点的指针，用它来表示单循环链表可以使查找链表的开始结点和终端结点都很方便。设置一个带头结点的单循环链表，其尾指针是 rear，则开始结点和终端结点分别为指针 rear 所指结点的后继结点的后继结点和指针 rear 所指结点，即 rear->next->next 和 rear，查找时间均为 O(1)。若用头指针来表示该链表，则查找开始结点为 O(1)，终端结点为 O（n）。

栈与队列

1、栈

限定在表尾进行插入或删除操作（头进尾出）的线性表，表尾称为栈顶（top），表头称为栈底（bottom），不含任何元素的栈称为空栈，栈又被称为后进先出（LIFO）线性表。

栈顶指针： S.top，初始时设置 S.top = -1

栈顶元素： S.data[S.top]

栈空条件： S.top == -1

栈满条件： S.top == MaxSize － 1

栈长： S.top + 1

栈操作：（记住顺序，哪个指针先增加）

- 入栈时栈顶指针先加 1，再送值到栈顶元素
- 出栈时先取栈顶元素值，再将栈顶指针减 1

简答题 2：描述堆栈和递归的关系。（题目中要求不能使用递归，那么就自己使用堆栈来完成！！十分重要的思路）

答：递归过程是一种调用自身的函数，在调用的过程中存在转入子程序的过程。在每次转入子程序前需要保护现场，则将相应参数和中间结果压入系统堆栈，而在子程序返回的时候，需要恢复现场，则将之前压栈的数据从系统堆栈弹出，因此，递归过程存在隐含的堆栈操作，而且子程序的调用过程满足堆栈先进后出的特性。

简答题 3：栈和队列各有什么特点，什么情况下用到栈，什么情况下用到队列？

答：栈的主要特点是“后进先出”，栈的应用十分广泛，通常将递归算法转换成非递归算法时需要使用栈，栈的应用有括号匹配、算术表达式求值和迷宫问题求解等。
队列的主要特点是“先进先出”，队列的应用也十分广泛，特别在操作系统资源分配和排队论中大量地使用队列，还有队列在层次遍历中的应用。

2、队列（必须分清顺序队列和循环队列，不要记乱结论！）

- **顺序队列**

只允许在表的一端进行插入，另一端进行删除操作（尾进头出），允许插入的一端叫做队尾（rear）， 允许删除的一端叫做队头（front），是一种先进先出（FIFO）的线性表。

初始条件： Q.front = Q.rear = 0

队空条件： Q.front == Q.rear

- **队列操作（不管是出队还是入队都是指针先自加！）**

进队操作： 队不满时，++ Q.rear;Q.data[Q.rear] = x;

出队操作： 队不空时，++ Q.front;x = Q.data[Q.front];

循环队列的判满、判空方法

- **牺牲一个存储单元法：(考试重点务必背下来！)**

  队满条件：(Q.rear + 1)%MaxSize == Q.front

  队空条件：Q.front == Q.rear

  队列中元素个数：(Q.rear － Q.front + MaxSize)%MaxSize

- **计数器法：**

  队满条件：Q.size == MaxSize

  队空条件：Q.size == 0

- **标志位法：**

  队满条件：tag == 1

  队空条件：tag == 0

  若因插入导致 Q.front == Q.rear，置 tag = 1，若因删除导致 Q.front == Q.rear，置 tag=0。

PS.820 真题中从来没考过队和栈的链式存储，820 有向 408 难度靠拢的趋势！不得不防必须掌握！！请放在考试前背诵，平时了解即可。若实在不想看也行，考的几率低。

- **栈的链式存储（链栈）** 便于多个栈共享存储空间和提高其效率，不存在栈满上溢的情况，规定链栈没有头结点，Lhead 指向栈顶元素。

  基本操作：

  InitStack(&S)　　//初始化

  p->next=S->next;　S->next=p;

  Push(&S,e)　　//入栈

  p->data=e;　　S->next=p;　　p->next=S->next;

  Pop(&S,&e)　　//出栈

  p=S->next;　　e=p->data;　　S->next=p->next;　free(p);

  DestroyStack（&S）　　//销毁栈

  p=S;　　S=S->next;　　free(p);

- **队列的链式存储（链队列）**

  基本操作：

  InitQueue(&Q)　　//初始化

  Q.front=Q.rear=(QueuePtr)malloc(sizeof(QNode));　　Q.front->next=NULL;

  DestroyQueue(&Q)　　//销毁队列

  Q.rear=Q.front->next;　Free(Q.front);　Q.front=Q.rear;

  EnQueue(&Q,e)　　//入队

  p->data=e;　　p->next=NULL;　　Q.rear->next=p;　　Q.rear=p;

  DeQueue（&Q,&e）　　//出队

  p=Q.front->next;　e=p->data;　　Q.front->next=p->next;

  if(Q.rear==p) Q.rear=Q.front;　　free(p);

3、特殊矩阵的压缩储存　（背个结论即可！）

数组的存储结构：（务必对应好行与列，不必要背公式，画个表自己对应就好）

行优先：$LOC(i,j)=LOC(0,0)+(b_2 \cdot i+j)L$

列优先：$LOC(i,j)=LOC(0,0)+(b_2 \cdot j+i)L$

特殊矩阵的压缩存储：

- **对称矩阵：**

将 n^2 个元素压缩存储到 n(n+1)/2 个元素的空间中，可以行序为主序存储基下三角（含对角线）中的元素，以一维数组 sa[n(n+1)/2]作为 n 价对称矩阵 A 的存储结构，则 sa[k]和矩阵元 a_{ij} 之间存在着一一对应关系（考前背诵）：

$$k=\begin{cases}\dfrac{i(i-1)}{2}+j-1, & i\geq j\ (\text{下三角区和主对角线元素})\\ \dfrac{j(j-1)}{2}+i-1, & i<j\ (\text{上三角区元素}a_{ij}=a_{ji})\end{cases}$$

- **稀疏矩阵：**

假设在 m×n 的矩阵中，有 t 个元素不为零，令称 $\delta=\frac{t}{m\times n}$，δ 为矩阵的稀疏因子，通常认 δ ≤0.05 时称为稀疏矩阵，存储时只存储稀疏矩阵的非零元，可用三元组顺序表和十字链表两种方式存储。

三元组表格式：（必须了解！）

行号	列号	元素值
i	j	e

其中 i、j 和 e 这 3 个域分别表该非零元所在的行、列和非零元的值。

5、广义表的基本概念、存储结构和基本操作（表头为元素，表尾为表！）

广义表的定义：广义表一般记作

$$LS=(\alpha_1,\ \alpha_2,\ \cdots,\ \alpha_n)$$

其中，LS 是广义表$(\alpha_1,\ \alpha_2,\ \cdots,\ \alpha_n)$的名称，n 是它的长度，$\alpha_i$ 可以是单个元素，也可以是广义表，分别称为 LS 的原子和子表，当广义表 LS 非空时，称第一个元素 α_1 为 LS 的表头(Head)，称其余元素组成的表$(\alpha_2,\ \alpha_3,\ \cdots,\ \alpha_n)$是 LS 的表尾(Tail)

几类广义表：（记住！）

A=()——A 是一个空表，它的长度为零

B=(e)——列表 B 只有一个原子 e，B 的长度为 1

C=(a,(b,c,d))——列表 C 长度为 2，两个元素分别为原子 a 和子表(b,c,d)

D=(A,B,C)——列表 D 的长度为 3，3 个元素都是列表，代入 A、B 和 C 的值后，有 D=((),(e),(a,(b,c,d)))

E=(a,E)——列表 E 是一个递归的表，它的长度为 2，深度为∞

广义表的主要操作：

GetHead(LS) 取表头（第一个元素）

GetTail(LS) 取表尾（除第一个元素外其余元素组成的表）

树和二叉树

树是一种递归的数据结构，树作为一种逻辑结构，同时也是一种分层结构。

1、树的一些基本术语

结点的度：树中一个结点的子结点数

树的度：树中结点的最大度数

分支结点：度大于 0 的结点（非终端结点）

叶子结点：度为 0 的结点（终端结点）

结点的层次：从根开始定义起，根为第一层，根的孩子为第二层，若某结点在第 l 层，则其子树的根就在第 l+1 层

树的深度（高度）：树中结点的最大层次

森林：m(m>0)棵互不相交的树的集合，对树中的每个结点而言，基子树的集合即为森林

* 注：由于树中的分支是有向的，所以树中的路径是从上到下的，同一双亲结点的两个孩子结点之间不存在路径

2、树的性质（超级无敌重点，考试前必须背！！）

1）树中结点数=所有结点的度数(树中的边数)+1

2）度为 m 的树中第 i 层上至多有 m^{i-1} 个结点(i≥1)

3）高度为 h 的 m 叉树至多有$(m^h-1)/(m-1)$个结点

4）具有 n 个结点的 m 叉树的最小高度为$\langle \log_m(n(m-1)+1)\rangle$（<x>表示不小于 x 的最小整数）

注：特别的，对于二叉树而言，有 n0=n2+1,即叶子结点的个数等于度为 2 的结点总数加 1.

5）给定 N 个节点，能构成 h(N)种不同的二叉树。（这个考点也不用说了，无敌重要！！）

*注：h(N)为卡特兰数的第 N 项。计算方法为：

$$h(n)=\frac{C_{2n}^{n}}{n+1}$$

3、二叉树前序、中序、后序、层序的递归和非递归算法必须掌握，要能写能背诵，算法大题要靠它。需要背诵的数据结构代码附在另一个文件。

简答题 4：回答满足后序序列和前序序列正好相同的二叉树的树型和正好相反的二叉树的树型各是什么？

答：相同：只有一个根节点的二叉树。

相反：任意结点均无左孩子的二叉树（仅有右孩子的二叉树）

或任意结点均无右孩子的二叉树（仅有左孩子的二叉树）

简答题 5：什么样的二叉树，对它采用任何次序的遍历，结果都相同？（简答题必须理解！只有一个结点或没有结点的树常考！）

答：仅有根结点的二叉树或空二叉树。

4、由遍历序列构造二叉树（必须背！）

1）先序遍历中，第一个结点一定是二叉树的根结点

2）中序遍历中，根结点必然将中序序列分割成两个子序列

3）后序序列的最后一个结点一定是二叉树的根结点

4）由二叉树的遍历能否唯一地确定一棵二叉树

	先序	中序	后序	层序
先序	—	√	×	×
中序	√	—	√	√
后序	×	√	—	×
层序	×	√	×	—

5、树的存储结构（必须了解！）

- **双亲表示法**

以一组连续空间存储树的结点，同时在每个结点中附设一个指示器指示其双亲结点在链表中的位置。缺点：求结点的孩子时需要遍历整个结构。

- **孩子表示法**

除根结点外，将每个结点的孩子结点都用单链表链接起来形成一个线性结构，则 n 个结点就有 n 个孩子链表，而 n 个头指针又组成一个线性表。特点：寻找双亲的操作需要遍历 N 个结点中的孩子链表指针域所指向的 N 个孩子链表。

- **孩子兄弟表示法**

二叉树表示法、或称二叉链表表示法。以二叉链表作为树的存储结构，链表中结点的两个链域分别指向该结点的第一个孩子结点和下一个兄弟结点，分别命名为 firstchild 域和 nextsibling 域。优点：可以方便地实现树转换为二叉树的操作，易于查找结点的孩子。缺点：从当前结点查找其双亲结点比较麻烦。

6、遍历

- **树的遍历（只和二叉树的前序和中序对应）**

先根遍历：访问顺序与这棵树相应二叉树的先序遍历顺序相同

后根遍历：访问顺序与这棵树相应二叉树的中序遍历顺序相同

- **森林的遍历：**森林的先序和中序遍历即为其对应的二叉树的先序和中序遍历

7、平衡树中含有最少的结点数（背下来！）

假设以 N_h 表示深度为 h 的平衡树中含有的最少结点数，显然 $N_0=0$，$N_1=1$，$N_2=2$，并且 $\mathbf{N_h=N_{h-1}+N_{h-2}+1}$。

8、哈夫曼树性质

1）每个初始结点最终都成为叶结点，且权值越小的结点到根结点的路径长度越大

2）构造过程中一共新建了 n-1 个结点，因此，哈夫曼树中的结点总数为 2n-1

3）哈夫曼树中不存在度为 1 的结点

图

1、图的基本概念和术语（图不能为空！至少一个顶点）（图的概念比较多，建议平时多看）

图不可以是空图，即顶点集不能为空，但边集可以为空。

含有 n 个顶点的连通无向图，边至少为 n-1 条;对于连通无向图而已，边至少 n 条。

- **邻接矩阵存储的特点：（有向图和无向图的度是不同的！）**

1）无向图的邻接矩阵一定是一个对称矩阵（而且唯一）

2）无向图邻接矩阵的第 i 行(列)非零(非∞)元素的个数正好是第 i 个顶点的度 $TD(v_i)$

3）有向图邻接矩阵的第 i 行（列）非零（非∞）元素的个数正好是第 i 个顶点的出度 $OD(v_i)$(入度 $ID(v_i)$)，总的度数=入度 $ID(v_i)$+出度 $OD(v_i)$。

4）用邻接矩阵存储图，容易确定图中任意两个顶点之间是否有边相连，但是要确定图中有多少条边，必须按行、按列对每个元素进行检测，花费时间代价很大，这是其局限性

5）稠密图适合用邻接矩阵表示

6）**设图 G 的邻接矩阵为 A，则 A^n[i][j]表示由顶点 i 到顶点 j 的长度为 n 的路径的数目（重点！）**

- **邻接表存储的特点：**

1）如果无向图 G 中有 n 个顶点、e 条边，则其邻接表需 n 个头结点和 2e 个表结点。显然，在边稀疏（e<<n(n-1)/2）的情况下，用邻接表表示图比邻接矩阵节省存储空间。

2）在无向图的邻接表中，顶点 v_i 的度恰为第 i 个链表中的结点数，而在有向图中，第 i 个链表中的结点个数只是顶点 v_i 的出度。

3）在邻接表中，给定一顶点能很容易地找出它所有的边，因为只需读取它的邻接表就可以了，花费的时间为 O(1)，而同样的操作在邻接矩阵中要耗费 O(n)的时间。但是若要确

定给定两个顶点间是否存在边，则在邻接矩阵里可以立刻查到，在邻接表中则需要在相应结点对应的边表中查找另一结点，效率较低

4）在有向图的邻接表中，求一个给定顶点的出度只需计算其邻接表中的结点个数即可，但如果要求其入度，则需要遍历全部的邻接表

5）由于各边结点的链接次序可以是任意的，所以图的邻接表表示并不唯一，该次序于建立邻接表的算法及边的输入次序

＊注： 由于图的邻接矩阵表示是唯一的，但对于邻接表来说，如果边的输入次序不同，生成的邻接表也不同因此，对于同一个图，基于邻接矩阵的遍历所得到 的 DFS 序列和 BFS 序列是唯一的，基于邻接表的遍历所得到的 DFS 序列和 BFS 序列是不唯一的。**（重点！）**

2、最小生成树（普利姆算法和克鲁斯卡尔算法）

1）最小生成树不是唯一的

2）最小生成树的边的权值之和问题唯一的，而且是最小的（代价唯一）

3）最小生成树的边数为顶点数减 1

3、关键路径：从源点到汇点的具有最大路径长度的路径。关键路径上的活动称为关键活动，辨别关键路径就是要找 l(i)=e(i)的活动。

＊网中的关键路径并不唯一，且对于有几条关键路径的网，只提高一条关键路径上的关键活动速度并不能缩短整个工程的工期

简答题 6：对有向图和无向图，分别描述如何判定图中是否存在回路。

答：对有向图，可以用拓扑排序算法来判定图中是否存在回路，当输出顶点数小于顶点数时，图中存在回路，否则图中无回路。

简答题 7：什么叫无向图的连通分量和生成树？

答：无向图的极大连通子图称为连通分量；图的极小连通子图称为生成树（包含图中所有 n 个顶点，但只有 n-1 条边）。

简答题 8：对 n 个顶点的无向图，采用邻接表表示时，如何判别下列有关问题？

（1）图中有多少条边？

（2）任意两个顶点 i 和 j 是否有边相连？

（3）任意一个顶点的度是多少？

答：（1）图中的边数=邻接表链表结点总数的一半

（2）任意两顶点间是否有边相连，可看其中一个顶点的邻接表，若链表中的 adjvex 域有另一顶点位置的结点，则表示有边相连。

（3）任意一个顶点的度等于该顶点的链表中的结点个数。

简答题 9：回答 AOV 网和 AOE 网能解决的主要问题。

答：AOV 网：（1）判定工程的可行性。有回路，整个工程就无法结束。

（2）确定各项活动在整个工程执行中的先后顺序。

AOE 网：（1）估算工程的最短工期（从源点到汇点至少需要多少时间）。

（2）找出哪些活动是影响整个工程进展的关键。

简答题 10：什么样的连通图其最小生成树是唯一的？

答：具有 n 个顶点，n-1 条边的连通图其生成树是唯一的；或者每条边的权值均不相同的连通图其最小生成树是唯一的。

简答题 11：缩短某关键活动的持续时间是否总能缩短整个工程的工期？为什么？

答：不一定能。因为任何一项活动持续时间的改变都可能会导致关键路径的改变，所以只有在不改变关键路径的情况下，缩短关键活动的持续时间才有效；在有多条关键路径时，缩短不是所有关键路径共有的关键活动的持续时间，并不能缩短整个工期。

查找和排序

1、查找表

静态查找表：

若在查找表中只作前两种统称为“查找”的操作，则称此类查找表为静态查找表。

动态查找表：

若在查找过程中同时插入查找表中不存在的数据元素，或者从查找表中删除已存在的某个数据元素，则称此类查找表为动态查找表。

2、Hash 函数构造和处理冲突（必须了解！）

- **哈希函数：**

  一个把查找表中的关键字映射成该关键字对应的地址的函数，记为 Hash(key)=Addr

  1）哈希函数的设定很灵活，只要使得任何关键字由此所得的哈希函数值都落在表长允许范围之内即可

  2）对不同的关键字可能得到同一哈希地址，这种现象称为冲突。冲突只能尽可能地少，而不能完全避免，　这就需要在建立哈希表时不仅要设定一个好的哈希函数，而且要设定一种处理冲突的方法。

- **哈希表：**

  根据设定的哈希函数 H(key)和处理冲突的方法奖一组关键字映像到一个有限的连续的地址集（区间）上，并以关键字在地址集中的“像”作为记录在表中的存储位置，这种表便称为哈希表，这一映像过程称为哈希造表或散列，所得存储位置称哈希地址或散列地址。

- **哈希函数的构造方法：**

  1）直接定址法：取关键字或关键字的某个线性函数值为哈希地址，即：

  $$H(key)=key \text{ 或 } H(key)=a \cdot key+b$$

其中 a 和 b 为常数。由于直接定址所得地址集合和关键字集合的大小相同，因此，对于不同的关键字不会发生冲突，但实际中能使用这种哈希函数的情况很少。这种方法适合关键字分布基本连续的情况，若关键字分布不连续，空位较多，将造成存储空间的浪费。

2）数字分析法：假设关键字是以 r 为基的数（r 进制数），并且哈希表中可能出现的关键字都是事先知道的，则可取关键字的若干数位组成哈希地址。

3）平方取中法：取关键字平方后的中间几位为哈希地址。取的位数由表长决定。这种方法产生的哈希地址与关键字的每一位都有关系，使得哈希地址分布比较均匀，适用于关键字的每一位取值都不够均匀或均小于哈希地址所需位数的情况。

4）折叠法：将关键字分割成位数相同的几部分（最后一部分的位数可以不同），然后取这几部分的叠加和（舍去进位）作为哈希地址。适用于关键字位数很多，且关键字中每一位上数字分布大致均匀的情况。

5）除留取余法：取关键字备某个不大于哈希表长 m 的数 p 除后所得余数为哈希地址。即

$$H(key)=key \ MOD \ p, \ p \leqslant m$$

这是一种最简单也最常用的构造哈希函数的方法。由众人的经验得知：一般情况下，可以选 p 为质数或不包含小于 20 的质因数的合数。

6）**随机数法：**选择一个随机函数，取关键字的随机函数值为它的哈希地址，即 H(key)=random(key)，其中 radom 为随机函数。通常，当关键字长度不等时采用此法构造哈希函数较恰当。

- **处理冲突的方法：**

1）**开放定址法：**

$$H_i=(H(key)+d_i) \ MOD \ m \ \ i=1,2,\cdots,k(k\leqslant m-1)$$

其中：H(key)为哈希函数；m 为哈希表表长；d_i 为增量序列，又以下 3 种取法：

a)**线性探测再散列：**

$$d_i=1,2,\cdots,m-1$$

特点：容易发生二次聚集，但可以保证做到只要哈希表未填满，总能找到一个不发生冲突的地址 H_k。

b)**二次线性探测再散列：**

$$+1^2,-1^2,2^2,\cdots,\pm k^2(k\leqslant m/2，\ m\ 必须是一个可以表示成\ 4k+3\ 的质数)$$

特点：可以避免出现二次聚集的问题，缺点是不能探测到散列表上的所有单元，但至少能探测到一半单元。

c)**伪随机探测再散列：**

$$d_i=伪随机数序列$$

注：在开放定址情形下，不能随便删除表中已有元素，因为若删除元素将会截断其他具有相同哈希地址的元素的查找地址。所以若想删除一个元素时，给它做一个删除标记，进行逻辑删除。

2)**再哈希法：**

$$H_i=RH_i(key) \qquad i=1,2,\cdots,k$$

RH_i 均是不同的哈希函数，即在同义词产生地址冲突时计算另一个哈希函数地址，直到冲突不再发生。这种方法不易产生聚集，但增加了计算时间，最多经过 m-1 次探测会遍历表中所有位置，回到 H_0 位置。

3）**链地址法：**将所有关键字为同义词的记录存储在同一线性链表中。假设某哈希函数产生的哈希地址在区间[0,m-1]上，则设立一个指针型向量 chain ChainiHash[m]；其每个分量的初始状态都是空指针。凡哈希地址为 i 的记录都插入到头指针为 ChainHash[i]的链表中。在链表中的插入位置可以在表头或表尾，也可以在中间，以保持同义词在同一线性链表中按关键字有序。此法适用于经常进行插入和删除的情况。

4）**建立一个公共缓冲区：**设 HashTable[0…m-1]为基本表，每个分量存放一个记录，另设 OverTable[0…v]为溢出表，所有关键字和基本表中关键字为同义词的记录，不管它们由哈希函数得到的哈希地址是什么，一旦发生冲突，都填入溢出表。**（尤为关注！）**

- **哈希表的查找效率取决于三个因素：哈希函数、处理冲突的方法和装填因子**

* 装填因子：一般记为 α，定义为一个表的装满程度，即

$$\alpha = \frac{\text{表中记录数n}}{\text{哈希表长度m}}$$

哈希表的平均查找长度依赖与哈希表的装填因子α，而不直接依赖于 n 或 m。α 越大，表示装填的记录越满，发生冲突的可能性就越大。

3、排序算法的选择

1）若 n 较小（n≤50），则可以采用直接插入排序或简单选择排序。

2）若文件的初始状态已按关键字基本有序，则选用直接插入排序或起泡排序为宜。

3）若 n 较大，则应采用时间复杂度为 O(nlogn)的排序方法：快速排序、堆排序或归并排序。快速排序被认为是目前基于比较的内部排序方法中最好的方法，当待排序的关键字是随机分布时，快速排序的平均时间最短。

4）若 n 很大，记录的关键字位数较少且可以分解时，采用基数排序较好。

简答题 12：试分析线性探测法和二次探测法解决哈希地址冲突时，可能存在的不足。

答：**线性探测法：**容易产生“堆积”问题，这是由于当连续出现若干个同义词后，会产生连续多个地址被占用，而导致随后映射到这些地址的关键词由于前面的同义词堆积而产生冲突，尽管随后的这些关键词并没有同义词。

二次探测法：不能充分利用空间，它不能探测到哈希表上的所有单元，但至少能探测到一半单元。

简答题 13：若输入数据存储在带头结点的双向循环链表中，下面各种排序算法是否仍然适用？为什么？

（1）快速排序；（2）直接插入排序；（3）简单选择排序；（4）堆排序

答：（1）快速排序适用。因为可以快速定位到第一个元素和最后一个元素结点，然后通过 1 个指针从头部向后移动，另外一个指针从尾部向前移动，逐一与基准元素进行比较并能够通过修改指针完成结点交换操作。

（2）插入排序适用。因为可以方便地找到前驱后继和通过修改指针完成结点交换操作。

（3）选择排序适用。因为只需要移动指针遍历链表并通过修改指针完成结点交换。

（4）堆排序不适用。因为双向循环链表无法方便地找到完全二叉树的双亲与孩子结点。

填空题

1. 数据的物理结构主要包括 顺序存储结构 和 链式存储结构 两种情况。
2. 数据的逻辑结构是对数据之间关系的描述，主要有 线性结构 和 非线性结构 两大类。
3. 线性结构主要包括以下几种数据结构

（1） 线性表的顺序和链式结构 （2） 栈和队列 （3） 串 （4） 数组和广义表

4. 一个“好”的算法应考虑达到以下目标： 正确性 、 可读性 、 健壮性 和 效率与低存储量需求 。
5. 算法是 指令的有限序列 。
6. 数组结构的特性是 它是线性表的扩充 和 可进行随机访问 。
7. 读取数组给定下标的数据元素的操作，称为 取值 操作；存储或修改数组给定下标的数据元素的操作，称为 赋值 操作。
8. 判定循环队列的满与空，有三种方法，它们是 计数器法 ， 标志位法 和 牺牲一个存储单元法 。
9. 广义表难以用 顺序 存储结构，适合编写递归算法的广义表的存储结构是 表头表尾链 。
10. 对广义表进行操作，结果总是表的基本操作是 取表尾 操作。
11. 遍历二叉树实质上是对一个非线性结构进行 线性化 操作。
12. 设图中顶点数为 n，则其生成树有 n-1 条边；若图的边数大于 n-1，则一定是 有环 图；若图的边数小于 n-1，则一定是 非连通 图。
13. 连通分量是无向图中的 极大 连通子图。
14. 要保证连通具有 10 个顶点的无向图，至少需要 37 条边。（9 个顶点构成完全图（(9*8)/2）=36，再加一条边连通第 10 个顶点）
15. Floyd 最短路径算法中，表示从顶点 Vi 到顶点 Vj 中间顶点序号 不大于 k 的最短路径长度。
16. 在有向图的邻接矩阵中，若主对角线以下的元素均为零，则该图的拓扑有序序列是 存在 的。
17. 在 AOE 网中，从源点到汇点所经历的边的权值之和最小的路径，称为 最短路径 ；从源点到汇点所经历的边的权值之和最大的路径，称为 关键路径 。
18. AOV 网是一种 有向无回路 的图。
19. 对 DAG 图表示的整个工程和系统，人们最关心的是两个方面的问题
（1） 工程能否顺利进行 （2） 估算整个工程完成所必须的最短时间
20. 对查找表除进行查找操作外，可能还要进行向表中插入数据元素，或删除表中数据元素的表，称为 动态查找表 。
21. 填入哈希表中的元素个数与哈希表的长度的比值，称为哈希表的 装填因子 。
22. 为了能有效地应用 HASH 查找技术，必须解决的两个问题是 构造一个好的 HASH 函数 和确定解决冲突的方法 。
23. 散列表中解决冲突的两种方法是 开放地址法 和 链地址法 。
24. 设某散列表的长度为 100，散列函数 H(k)=k%P，则 P 通常情况下最好选择 97 。（比 P 小的最大素数）
25. 每次直接或通过基准元素间比较两个元素，若出现逆序排列就交换它们的位置，这种排序方法叫做 交换 排序。
26. 堆排序关键两步为 建立初始堆 和 筛选（调整为堆） 。
27. 实现基数排序算法时，最适合的数据结构是 链队列 。

考前临门一脚必背

常见图算法的复杂度

深度优先搜索 时间复杂度 领接矩阵表示 **$O(n^2)$** 领接表表示 **O(n+e)**

	空间复杂度	**O(e)**			
广度优先搜索	时间复杂度	领接矩阵表示	**O(e^2)**	领接表表示	**O(n+e)**
	空间复杂度	**O(n)**			
普利姆算法	时间复杂度	**O(n^2)**	适用于边稠密		
克鲁斯卡尔算法	时间复杂度	**O(eloge)**	适用于边稀疏		
迪杰斯特拉算法	时间复杂度	**O(n^2)**			
弗洛伊德算法	时间复杂度	**O(n^3)**			
拓扑排序(AOV)	时间复杂度	**O(n+e)**			
关键路径(AOE)	时间复杂度	**O(n+e)**			

注：这几个图的算法很容易弄乱，建议按顺序背。“PKDFVE”，没事就读 10 次，一定不会再错了。P-Prime;K-Kruskal;D-Dijkstra;F-Floyd;V-AOV;E-AOE.

常见排序算法在平均情况下的复杂度

1、时间复杂度

“快些以 $n\log_2 n$ 的速度归队”

快——**快速排序** 些——**希尔排序** 归——**归并排序** 队——**堆排序**

其他的都是 O(n^2),除了基数排序 O(d(n+r_d))

2、空间复杂度

快速排序 O($\log_2 n$) 归并排序 O(n) 基数 O(r_d) 其他都是 O(1)

3、稳定性

“心情不稳定，快些选一堆好友来聊天”

快——**快速排序** 些——**希尔排序** 选——**直接选择排序** 堆——**堆排序**

这四个都是不稳定的，其他都是稳定的。

4、一些特殊情况

- 一趟排序后，能够保证一个元素能达到最终位置：**起泡排序、快速排序、直接选择排序、堆排序**。
- 比较次数和序列无关：**直接选择排序、折半插入排序**
- 比较次数和序列有关：**起泡排序、快速排序**
- 输入数据有序，最好的排序算法：**起泡排序、直接插入排序**
- 不随关键字分布而改变：**堆排序、归并排序、直接选择排序**