第一章　数据结构概述

基本概念与术语

1．**数据**：数据是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被计算机程序所处理的符号的总称。

2.**数据元素**：数据元素是数据的基本单位，是数据这个集合中的个体，也称之为元素，结点，顶点记录。

（补充：一个数据元素可由若干个**数据项**组成。数据项是数据的不可分割的最小单位。）

3．数据对象：数据对象是具有相同性质的数据元素的集合，是数据的一个子集。（有时候也叫做属性。）

4．**数据结构**：数据结构是相互之间存在一种或多种特定关系的数据元素的集合。

（1）数据的逻辑结构：数据的逻辑结构是指数据元素之间存在的固有逻辑关系，常称为数据结构。

数据的逻辑结构是从数据元素之间存在的逻辑关系上描述数据与数据的存储无关，是独立于计算机的。

依据数据元素之间的关系，可以把数据的逻辑结构分成以下几种：

1.**集合**：数据中的数据元素之间除了“同属于一个集合“的关系以外，没有其他关系。

2.**线性结构**：结构中的数据元素之间存在“一对一“的关系。若结构为非空集合，则除了第一个元素之外，和最后一个元素之外，其他每个元素都只有一个直接前驱和一个直接后继。

3.**树形结构**：结构中的数据元素之间存在“一对多“的关系。若数据为非空集，则除了第一个元素（根）之外，其它　　每个数据元素都只有一个直接前驱，以及多个或零个　直接后继。

4.**图状结构**：结构中的数据元素存在“多对多”的关系。若结构为非空集，折每个数据可有多个（或零个）直接后继。

（2）数据的存储结构：数据元素及其关系在计算机内的表示称为数据的存储结构。

想要计算机处理数据，就必须把数据的逻辑结构映射为数据的存储结构。逻辑结构可以映射为以下两种存储结构：

1.**顺序存储结构**：把逻辑上相邻的数据元素存储在物理位置也相邻的存储单元中，借助元素在存储器中的相对位置来表示数据之间的逻辑关系。

2.**链式存储结构**：借助指针表达数据元素之间的逻辑关系。不要求逻辑上相邻的数据元素物理位置上也相邻。

5.**时间复杂度分析**：1.常量阶：算法的时间复杂度与问题规模 n **无关系** T(n)=O(1)

2.线性阶：算法的时间复杂度与问题规模 n **成线性关系** T(n)=O(n)

3.平方阶和立方阶：一般为**循环的嵌套**，循环体最后条件为 i++

时间复杂度的大小比较：

O(1)< O(log 2 n)< O(n)< O(n log 2 n)< O(n2)< O(n3)< O(2 n)<O(n!)<O(n n)

6.算法与程序：

（1）算法的 5 个特性

1、 输入：有零个或多个输入

2、 输出：有一个或多个输出

3、有穷性：要求序列中的指令是有限的；每条指令的执行包含有限的工作量；整个指令序列的执行在有限的时间内结束。（程序与算法的区别在于，程序不需要有有穷性）

4、确定性：算法中的每一个步骤都必须是确定的，而不应当含糊、模棱两可。没有歧义。

5、可行性：算法中的每一个步骤都应当能被有效的执行，并得到确定的结果。

（2）.算法设计的要求：

1、正确性（达到预期效果，满足问题需求）

2、健壮性（能处理合法数据，也能对不合法的数据作出反应，不会产生不可预期的后果）

3、可读性（要求算法易于理解，便于分析）

4、可修改可扩展性

5、高效率（较好的时空性能 ）

补充内容：

1、名词解释：数据结构、二元组

数据结构就是相互之间存在一种或多种特定关系的数据元素的集合。

二元组就是一种用来表示某个数据对象以及各个元素之间关系的有限集合。

2、根据数据元素之间关系的不同，数据的逻辑结构可以分为集合、线性结构、树形结构和图状结构四种类型。

3、常见的数据存储结构一般有两种类型，它们分别是顺序存储结构、链式存储结构

6.在一般情况下，一个算法的时间复杂度是问题规模的函数

7.常见时间复杂度有：常数阶 O（1）、线性阶 O（n）、对数阶 O（log 2 n）、平方阶 O(n^2)、指数阶 O(2^n)。通常认为，具有常数阶量级的算法是好算法，而具有指数阶量级的算法是差算法。

第二章 线性表

定义：**线性表**是 n 个数据元素的有限序列。 一个数据元素可由若干个**数据项**组成。

1. 顺序表结构

线性表的顺序存储是指在内存中**用地址连续的**一块存储空间顺序存放线性表的各元素，用这种存储形式存储的线性表称为顺序表。

2. 单链表

（1） 链表结点结构

线性表中的数据元素可以**用任意的一组存储单元**来存储，用指针表示逻辑关系逻辑相邻的两元素的存储空间可以是不连续的。

（2） 链表操作算法：初始化、插入、输出、删除、遍历

初始化：p=(struct student *)malloc(sizeof(struct student));

插入： p->next=head->next; head->next=p;
输出：printf(“%d”,p->data);
删除：q=p->next; p->next = q->next ; free(q);
结点遍历： for(p=head;p;p=p->next);

补充内容：
1、线性表中，第一个元素没有直接前驱，最后一个元素没有直接后驱。
2、在一个单链表中，若 p 所指结点是 q 所指结点的前驱结点，则删除结点 q 的操作语句为
P->next = q->next ; free(q);
3、在长度为 N 的顺序表中，插入一个新元素平均需要移动表中 N/2 个元素，删除一个元素平均需要移动（N-1）/2 个元素。
4、若线性表的主要操作是在最后一个元素之后插入一个元素或删除最后一个元素，则采用顺序表存储结构最节省运算时间。
5、已知顺序表中每个元素占用 3 个存储单元，第 13 个元素的存储地址为 336，则顺序表的首地址为 300。（第 n 个元素的地址即首地址+(n-1)*每个元素的存储空间，如 a[12]（第 13 个元素）的地址=a[0]+12*3）
6、设有一带头结点单链表 L，请编写该单链表的初始化，插入、输出和删除函数。（函数名自定义）
结点定义：

```
typedef int datatype;      //结点数据类型，假设为 int
typedef struct node {                  //结点结构
    datatype data;
   struct node *next;                  //双向链表还应加上*previous
} Lnode, * pointer ; //结点类型,结点指针类型
typedef pointer lklist;    //单链表类型，即头指针类型
```

1.初始化：

```
lklist initlist() {
    pointer head;
    head=new node;//这是 C++做法
    //head=( pointer)malloc(sizeof(Lnode));          这是 C 语言做法
    head->next=NULL;                    //循环链表则是 head->next=head;
                                        //双向链表应加上 head->previos=NULL;
    return head;
}
```

2.插入：（C 语言中需要把 head 转化为全局变量才能实现此程序）

```
int insert(lklist head,datatype x,int i){
   pointer q,s;
   q=get(head,i-1);  //找第 i-1 个点
   if(q==NULL)          //无第 i-1 点，即 i<1 或 i>n+1 时
      {
       cout<<”非法插入位置!\n”; //这是 C++做法，即 C 语言中的   printf(“非法插入位置!\n”);
       return 0;
```

```
        }
    s=new node;//生成新结点     即 C 语言中的   s=( pointer)malloc(sizeof(Lnode));
    s->data=x;
    s->next=q->next;   //新点的后继是原第 i 个点
    q->next=s;         //原第 i-1 个点的后继是新点
    return 1;          //插入成功
}
```

3.删除：（C 语言中需要把 head 转化为全局变量才能实现此程序）

```
int delete(lklist head,int i) {
    pointer p,q;
    q=get(head,i-1);              //找待删点的直接前趋
    if(q==NULL || q->next==NULL)     //即 i<1 或 i>n 时
       {cout<<”非法删除位置!\n”;return 0;}
    p=q->next;                    //保存待删点地址
    q->next=p->next;                  //修改前趋的后继指针
    delete p;                     //释放结点        即 C 语言中的  free(p);
    return 1;                     //删除成
```

1. 不带头结点的单链表 head 为空的判定条件是(A)

A. head=NULL B. head->next=NULL C. head->next=head D. head!=NULL

2. 带头结点的单链表 head 为空的判定条件是(B)

A. head=NULL B. head->next=NULL C. head->next=head D. head!=NULL

3. 在一个单链表中，若 p 所指结点不是最后结点，在 p 之后插入 s 所指结点，则执行(B)

A. s->next=p; p->next=s; B. s->next=p->next; p->next=s;

C. s->next=p->next; p=s; D. p->next=s; s->next=p;

4. 在一个单链表中，若删除 p 所指结点的后续结点，则执行(A)

A. p->next=p->next->next;

B. p=p->next; p->next=p->next->next;

C. p->next=p->next

D. p=p->next->next

5. 从一个具有 n 个结点的有序单链表中查找其值等于 x 结点时，在查找成功的情况下，需平均比较（B ）个结点。

A. n B. n/2 C. (n-1)/2 D. O(n log 2n)

6. 给定有 n 个元素的向量，建立一个有序单链表的时间复杂度（B）

A.O(1) B.O(n) C.O(n2) D.O(n log 2n)

7.在一个具有 n 个结点的有序单链表中插入一个新结点并仍然有序的时间复杂度是(B)

A.O(1) B.O(n) C.O(n2) D.O(n log 2n)

8. 在一个单链表中删除 q 所指结点时，应执行如下操作：

```
q=p->next;
p->next=( p->next->next );
free(q);//这种题目靠一根指针是没有办法完成的，必须要借助第二根指针。
```

9. 在一个单链表中 p 所指结点之后插入一个 s 所指结点时，应执行：

```
s->next=( p->next )
 p->next=(s)操作。
```

10. 对于一个具有 n 个节点的单链表 ,在已知所指结点后插入一个新结点的时间复杂度是(O（1）)；在给定值为 x 的结点后插入一个新结点的时间复杂度是（O(n)）。
11.问答题
线性表可用顺序表或链表存储。试问：

(1) 两种存储表示各有哪些主要优缺点?
顺序表的存储效率高，存取速度快。但它的空间大小一经定义，在程序整个运行期间不会发生改变，因此，不易扩充。同时，由于在插入或删除时，为保持原有次序，平均需要移动一半(或近一半)元素，修改效率不高。
链接存储表示的存储空间一般在程序的运行过程中动态分配和释放，且只要存储器中还有空间，就不会产生存储溢出的问题。同时在插入和删除时不需要保持数据元素原来的物理顺序，只需要保持原来的逻辑顺序，因此不必移动数据，只需修改它们的链接指针，修改效率较高。但存取表中的数据元素时，只能循链顺序访问，因此存取效率不高。

(2) 若表的总数基本稳定，且很少进行插入和删除，但要求以最快的速度存取表中的元素，这时，应采用哪种存储表示？为什么？
应采用顺序存储表示。因为顺序存储表示的存取速度快，但修改效率低。若表的总数基本稳定，且很少进行插入和删除，但要求以最快的速度存取表中的元素，这时采用顺序存储表示较好。

第三章　栈和队列

1.　栈
（1）　栈的结构与定义
定义：限定仅在表尾进行插入或删除操作的线性表。
结构：

```
typedef    struct list{
        int listsize;                //栈的容量
        struct list *head;          //栈顶指针
        struct list *base;          //栈底指针
}
```

（2）　顺序栈操作算法：入栈、出栈、判断栈空等（这个是使用数组进行操作的，具体内容参照书本 P46-47）
（3）　链栈的结构与定义
2.　队列
（1）　队列的定义
定义：只允许在表的一端进行插入，而在另一端删除元素。

补充内容：
1、一个栈的入栈序列为“ABCDE”，则以下不可能的出栈序列是（B）
A. BCDAE　　B. EDACB　　C. BCADE　　D. AEDCB
2、栈的顺序表示中，用 TOP 表示栈顶元素，那么栈空的条件是（D）
A. TOP==STACKSIZE　　B. TOP==1　　C. TOP==0　　D. TOP==-1

3、允许在一端插入，在另一端删除的线性表称为队列。插入的一端为表头，删除的一端为表尾。
4、栈的特点是先进后出，队列的特点是先进先出。
5、对于栈和队列，无论他们采用顺序存储结构还是链式存储结构，进行插入和删除操作的时间复杂度都是 O(1)（即与已有元素 N 无关）。
6、已知链栈 Q，编写函数判断栈空，如果栈空则进行入栈操作，否则出栈并输出。（要求判断栈空、出栈、入栈用函数实现）（详看考点 2）
7.出队与取队头元素的区别：出队就是删除对头的数据元素，取队头元素是获取对头的数据元素值，不需要删除。
8.链栈与顺序栈相比，比较明显的优点是：（D）
A.插入操作比较容易　　B.删除操作比较容易
C.不会出现栈空的情况　　D.不会出现栈满的情况

考点 1：队列的编程：
结构：

```
typedef struct QNode{
        int date;
        struct QNode *next;
        }QNode,*QueuePtr;
typedef struct{
        QueuePtr front;
        QueuePtr rear;
        }LinkQueue;
```

创建：

```
LinkQueue InitQueue(LinkQueue Q)
{
    Q.front=Q.rear=(QueuePtr)malloc(sizeof(QNode));
    Q.front->next=NULL;
    return (Q);
}
```

入队：

```
LinkQueue EnQueue(LinkQueue Q,int e)
{
    QueuePtr p;
    p=(QueuePtr)malloc(sizeof(QNode));

    p->date=e;
    p->next=NULL;
    Q.rear->next=p;
    Q.rear=p;
    return (Q);
    }
```

出队：

```
LinkQueue DeQueue(LinkQueue Q)
{
        int e;
        QueuePtr p;

        p=Q.front->next;
        e=p->date;
        Q.front=p->next;
        printf("%d",e);
        if(Q.rear==p)Q.rear=Q.front=NULL;
        free(p);
        return (Q);
}
```

考点 2：栈的编程：

创建：

```
struct list *creat()
{
    struct list *p;
    p=(struct list *)malloc(LEN);
    p->next=NULL;
    return(p);
}
```

入栈：

```
struct list *push(struct list *head,int a)
{
  struct list *p;
  p=(struct list *)malloc(LEN);
  p->num=a;
  p->next=head;
  return(p);
}
```

出栈：

```
struct list *pop(struct list *head)
{
    struct list *p;
    p=head->next;
    free(head);
    return(p);
}
```

判断栈空：

```
int listempty(struct list *head)
{
if(head->next)return 0;
else return 1;
}
```

第四章 串 （不是重点内容）

1.串是由零个或多个字符组成的有限序列

2.串的赋值：x=’abc’;或 x[ ]=’abc’;

第五章 数组和广义表 （不是重点内容）

1. 多维数组中某数组元素的 position 求解。一般是给出数组元素的首元素地址和每个元素占用的地址空间并组给出多维数组的维数，然后要求你求出该数组中的某个元素所在的位置。

2. 明确**按行存储**和**按列存储**的区别和联系，并能够按照这两种不同的存储方式求解 1 中类型的题。

3. 将特殊矩阵中的元素按相应的换算方式存入数组中。这些矩阵包括：对称矩阵，三角矩阵，具有某种特点的稀疏矩阵等。熟悉稀疏矩阵的三种不同存储方式：**三元组**，带辅助行向量的二元组，**十字链表**存储。掌握将稀疏矩阵的三元组或二元组向十字链表进行转换的算法。

补充内容：

三元组：

结构：

```
typedef struct{
        int i,j;        //元素行下标及列下标
        int e;          //元素值
        }Triple;
typedef struct{
        int mu,nu,tu;                       //矩阵的行数、列数、非零元素个数
        Triple data[MAXSIZE+1];             //矩阵包含的三元组表，data[0]未用
        }TSMatrix;
```

十字链表：

```
typedef struct OLNode{
        int i,j;        //元素行下标及列下标
        int e;          //元素值
        struct OLNode *right,*down;   //行的后继以及列的后继
        } OLNode，*OLink;
typedef struct{
        int mu,nu,tu;                       //矩阵的行数、列数、非零元素个数
        OLink   *rhead,*chead;          //行和列的表头指针组的首地址
        }CrossList;
```

```
CrossList Creat（CrossList M）{
        int m,n,t;
        scanf(“%d%d%d”,&m,&n,&t);
        M.mu=m;M.nu=n;M.tu=t;
        M.rhead=( OLink *)malloc((m+1)*sizeof(OLink));     //开辟行表头指针组
        M.chead=( OLink *)malloc((n+1)*sizeof(OLink));     //开辟行列头指针组
        M.rhead[]=M.chead[]=NULL;                          //初始化
        ……                                                 //接下来就是赋值和入链
}
```

第六章 树和二叉树

1． 树

（1） 树的概念及术语

树：n（n≥0）个结点的有限集合。当 n＝0 时，称为空树；任意一棵非空树满足以下条件：

⑴ 有且仅有一个特定的称为根的结点；

⑵ 当 n＞1 时，除根结点之外的其余结点被分成 m（m>0）个互不相交的有限集合 T1,T2,… ,Tm，其中每个集合又是一棵树，并称为这个根结点的子树。

（2） 结点的度：结点所拥有的子树的个数。

树的度：树中所有结点的度的最大值。

（3） 叶子结点：度为 0 的结点，也称为终端结点。

分支结点：度不为 0 的结点，也称为非终端结点。

（4）孩子、双亲：树中某结点的子树的根结点称为这个结点的孩子结点，这个结点称为它孩子结点的双亲结点；

兄弟： 具有同一个双亲的孩子结点互称为兄弟。

（5）路径： 如果树的结点序列 n1, n2, …, nk 有如下关系：结点 ni 是 ni+1 的双亲（1<=i<k），则把 n1, n2, …, nk 称为一条由 n1 至 nk 的路径；路径上经过的边的个数称为路径长度。

（6）祖先、子孙：在树中，如果有一条路径从结点 x 到结点 y，那么 x 就称为 y 的祖先，而 y 称为 x 的子孙。

（7）结点所在层数：根结点的层数为 1；对其余任何结点，若某结点在第 k 层，则其孩子结点在第 k+1 层。

树的深度： 树中所有结点的最大层数，也称高度。

（8）层序编号：将树中结点按照从上层到下层、同层从左到右的次序依次给他们编以从 1 开始的连续自然数。

（9）有序树、无序树：如果一棵树中结点的各子树从左到右是有次序的，称这棵树为有序树；反之，称为无序树。数据结构中讨论的一般都是有序树

（10）树通常有前序（根）遍历、后序（根）遍历和层序（次）遍历三种方式（树，不是二叉树，没中序遍历。)

2． 二叉树

（1）二叉树的定义：二叉树是 n（n≥0）个结点的有限集合，该集合或者为空集（称为空二叉树），或者由一个根结点和两棵互不相交的、分别称为根结点的左子树和右子树的二叉树组成。

满二叉树：在一棵二叉树中，如果所有分支结点都存在左子树和右子树，并且所有叶子都在同一层上。
（满二叉树的特点：叶子只能出现在最下一层；只有度为 0 和度为 2 的结点。）

完全二叉树：对一棵具有 n 个结点的二叉树按层序编号，如果编号为 i（1≤i≤n）的结点与同样深度的满二叉树中编号为 i 的结点在二叉树中的位置完全相同。
完全二叉树的特点：

1.在满二叉树中，从最后一个结点开始，连续去掉任意个结点，即是一棵完全二叉树。
2.叶子结点只能出现在最下两层，且最下层的叶子结点都集中在二叉树的左部；
3. 完全二叉树中如果有度为 1 的结点，只可能有一个，且该结点只有左孩子。
4. 深度为 k 的完全二叉树在 k-1 层上一定是满二叉树。

（3）二叉树的性质：

性质 1：二叉树的第 i 层上最多有 2i-1 个结点（i≥1）。

性质 2： 一棵深度为 k 的二叉树中，最多有 2k-1 个结点，最少有 k 个结点。深度为 k 且具有 2k-1 个结点的二叉树一定是满二叉树

性质 3：在一棵二叉树中，如果叶子结点数为 n0，度为 2 的结点数为 n2，则有: n0＝n2＋1。（一个结点的度就是指它放出的射线）

性质 4：具有 n 个结点的完全二叉树的深度为 log2n +1。

性质 5： 对一棵具有 n 个结点的完全二叉树中从 1 开始按层序编号，则对于任意的序号为 i（1≤i≤n）的结点（简称为结点 i），有：
（1）如果 i＞1，则结点 i 的双亲结点的序号为 i/2；如果 i＝1，则结点 i 是根结点，无双亲结点。
（2）如果 2i≤n，则结点 i 的左孩子的序号为 2i；如果 2i＞n，则结点 i 无左孩子。
（3）如果 2i＋1≤n，则结点 i 的右孩子的序号为 2i＋1；如果 2i＋1＞n，则结点 i 无右孩子。

3. 二叉树的遍历（递归调用与访问的顺序不同而产生不同的遍历方法）
（1） 先序遍历

```
void XianXu(BiTree T){
  if(T){
      printf("%c",T->data);            //先访问
      XianXu(T->lchild);               //再继续遍历
      XianXu(T->rchild);
  }
}
```

（2） 中序遍历
（3） 后序遍历

4. 森林与二叉树的转换
（1）同级以左为亲，即左一结点的右孩子是与它同级的右一结点
（2）只认最左路线为亲子路线，即结点的左孩子是它下一级结点的最左的元素

5. 哈夫曼树
（1）哈夫曼树的基本概念：

哈夫曼树：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。

（2）哈夫曼树的特点：

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.

（3）哈夫曼树的构造算法思想及构造过程（森林与**哈夫曼编码**）

就是求各权值和路径相乘之后叠加的最小值。

1、已知一棵完全二叉树有 47 个结点，则该二叉树有（C）个叶子结点。

A. 6　　B. 12　　C. 24　　D.48

解法如下：

1+2+4+8+16=31　　计算从第一层到 n-1 层的结点个数

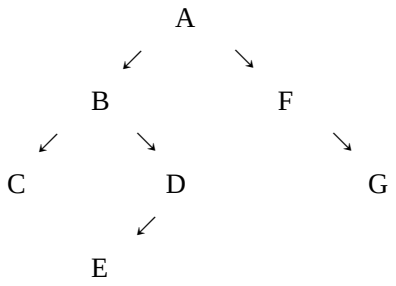
47-31=16　　计算第 n 层的叶子结点个数

16-16/2=8　　计算第 n-1 层的叶子结点个数

所以，叶子结点数=16+8=24　　计算第 n 层和第 n-1 层的总叶子结点数

2、已知遍历一棵二叉树的前序序列 ABCDEFG 和中序序列 CBEDAFG，那么是下面哪棵树（C ）。

C 图如下：



4、完全二叉树必须满足的条件为: ：一棵具有 n 个结点的二叉树，它的结构与满二叉树的前 n 个结点的的结构相同。

5、哈夫曼树不存在度为 1 的结点。

6、有 5 个带权结点，其权值分别为 2，5，3，7，11，根据哈夫曼算法构建该树，并计算该树的带权路径长度。（构建哈夫曼树，很简单，从小开始，计算相加，然后把所有叶子结点乘以等级数字然后相加。也即是：带权路径长度=叶结点的权值*路径长度）

7.试找出分别满足下列条件的所有二叉树：

⑴ 前序序列和中序序列相同：只有右子树

⑵ 中序序列和后序序列相同：只有左子树

⑶ 前序序列和后序序列相同：只有根，空二叉树

第七章 图

1. 图的基本概念：图的基本术语及推论

图的结点之间的关系可以是任意的，图中任意两个数据元素之间都可能相关。

设图有 n 个顶点，则：

有 1/2 n(n-1)条边的无向图称为**完全图**

有 n（n-1）条弧的有向图称为**有向完全图**

元素被多少条弧的箭头所指，它的**入度**就为多少；反之，出度。

第一个顶点和最后一个顶点相同的路径叫做**回路**或**环**

顶点不重复出现的路径叫**简单路径**

若图中任意两个顶点之间存在路径（不一定是直接相连），则称作**连通图**

2. 邻接矩阵：

$$\text{邻接矩阵的定义：}\ A[i][j]=\begin{cases} W_{i,j} & <V_i，V_j> \in VR \\ 0 & \text{即 VR 中不存在} <V_i，V_j> \text{时} \end{cases}$$

3. 图的遍历

（1）深度优先遍历

步骤：1.从任意顶点开始访问。

2.访问后把该元素对应的访问标志赋值为 1 表示已访问该数据元素

3.寻找与其有关未被访问的所有邻接顶点，并从该顶点开始进行访问

4. 重复 2、3 步骤直到该连通图的所有顶点均已访问完毕

（2）广度优先遍历

步骤：1.从任意顶点开始访问。

2.访问后把该元素对应的访问标志赋值为 1 表示已访问该数据元素

3.寻找与其有关未被访问的邻接顶点，并按顺序入列直到所有邻接顶点均已访问完毕

4.把最先入列的顶点出列，以它为顶点开始访问

5. 重复 2、3、4 步骤直到该连通图的所有顶点均已访问完毕

第八九十章

查找表

是由同一类型的数据元素（或记录）构成的集合

对查找表的操作有：

（1） 查询某个“特定的”数据元素是否在查找表中；

（2） 检索某个“特定的”数据元素的各种属性

（3） 在查找表中插入一个数据元素；

（4） 从查找表中删去某个特定元素

静态查找表

只进行前两种“查找”操作的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，则成为动态查找表

排序

其功能是将一个数据元素（或记录）的任意序列，重新排列成一个按关键字有序的序列。