# 第一部分·易混、易漏常考知识点

## 第一章·概论

基本语句的执行次数T(n)：函数表达式

时间复杂度O(n)：T(n)取最高次

注：对于两个顺序执行的循环语句，基本语句的执行次数应是它们相加。

## 第二章·顺序栈

①合法的出栈序列数：

②什么出栈序列不合法？

若输入序列为1,2,...,n，用数组来表示对应的输出序列。

即可得到：若i＜j＜k，则断然不可能出现a[i]＞a[k]＞a[j]的情况，即若 最后进栈的最先出栈，最先进栈的不可能夹在中间出栈。

## 第三章·队列

①约定：顺序存储方式存储的是顺序队列。用一个数组data[maxlen]来存储元素，将该数组前面的元素作为队头，后面的元素作为队尾。设一尾指针rear指向队列的最后一个元素（即队尾），同时设一头指针指向队头的前一个位置，这样在删除时仅仅修改队头指针所指示的位置即可。

②循环队列的入队与出队操作：

入队操作：rear=(rear+1)%maxlen;

data[rear]=x;

出队操作：front=(front+1)％maxlen;

③循环队列的判空与判满：

法一·设置标志法：设置一标志，若最后一次操作是入队，则front==rear表示满；若最后一次操作是出队，则front==rear表示空。

法二·空一个存储空间法：

队空（同时也是初始状态）：front==rear

队满：count==maxlen-1 或

(rear+1)%maxlen==front为队满。

## 第四章·链栈与链队列

1、静态链表与动态链表

静态链表：即数组

动态链表：存储表的首元素a₁的结点为首元素结点

2、链栈与链队列

①链栈：采用链表存储的栈称为链栈，但是只能用链表表头存储栈顶元素，并假设头指针为top。

②链队列：队头在链首，队尾在链尾。

为使入队操作在队列为空和不空时的操作保持一致，设置一个不存放元素值的附加结点（头结点）。

判空：front==rear

## 第五章·线性表

1、线性表的约定：由n个元素组成的有限序列。将元素a₁、a₂、....等n个元素存放到数组中前面n个元素中（下标从0~n-1）。最多可存放maxlen个元素。

线性表只有一个直接前驱和直接后继，这是线性表的特点。栈和队列是特殊的线性表。

|  |  |  |
| --- | --- | --- |
| 其他形式的链表结构 | | |
| 链表 | 说明 | 特点 |
| 单循环链表 | 分带头结点和不带头结点的单循环链表，最后一个元素都是指向第一个节点（而不是首元素结点）。 | 由于只能从头结点开始访问，不太方便对链尾操作。 |
| 带尾指针的单循环链表 | 在单循环链表的基础上，增设一尾指针指向最后一个结点。 | 可从末元素结点开始访问，对链尾的插入以及首元素结点较便利，但不便于删除链尾元素。 |
| 双链表结构 | 既有前驱，也有后继。可以带头结点，也可以是循环的。 |  |

2、特殊的线性表——串

①串是由有限个字符a₁,a₂....组成的序列，记作S="a₁a₂...."

S称作串名，等号右边为其串值，元素个数n称为串的长度。当n=0称之为空串。称S中的一个连续段所组成的串为S的子串。

②顺序串：为减少空间浪费，可采用压缩的方法来存储，将每个内存单元中尽可能多地存放字符，由此得到紧凑格式的顺序串。

③链串：同。紧凑格式存储。

## 第六章·递归

递归程序转非递归程序

原理：递归的返回无非就是栈的应用，能试探，能回溯。因此，模拟这一过程就好。

转换规则：

①设置一个栈（不妨用S表示），并在开始时置为空。

②在子程序入口处设置一个标号（不妨设为L₀）。

③对子程序中的每一递归调用，用以下几个等价操作来替换。

保留现场：开辟栈顶存储空间（用于保存返回地址）、调用层中的形参和局部变量的值。

准备数据：为被调用的子程序准备数据，即计算实参的值，并赋值给形参。

转入子程序执行：goto L₀;

④对返回语句，用以下几个等价操作替换。

若栈不空，则依次执行以下操作，否则结束本子程序，返回。

回传数据：若函数需要返回值，将其值保存在回传变量中。

恢复现场：从栈顶取出返回地址及其各变量、形参的值，并退栈。

返回：按返回地址返回，即执行goto X;

⑤对其中的非递归调用和返回操作可照搬。

注·简化规则：

①如果递归程序中只有一处递归调用，则在转换时，返回地址不必入栈。

②在模拟尾递归调用时，不必执行入栈操作。

③按程式写出基本程序，再按流程图化简成while循环！

实例：

①源代码

|  |
| --- |
| void P(int N)  {  if(N＞0）  {  P(N-1);  cout<<N;  P(N-1);  }  } |

②优化后的范式程序

|  |
| --- |
| void P(int N)  {  Stack S;  L₀:  if(N＞0)  {  S.push(N);  N=N-1;  goto L₀;  L₁:  cout<<N;  N=N-1;  goto L₀;  L₂:  }  if(S.empty())  { S.pop(W);  goto L₁;  }  } |

补充：

①在判断递归函数功能时，可自己先层层推导找出共性，一般都不会太难，诸如排列数、组合数等。

②递归函数执行时，可用树形分析法+简化推导一边推断。因为另一半和它可能是完全一样的结果，重复就好。

③阿克曼函数是原始递归函数，没有非递归表现形式。

## 第七章·数组和广义表

1、二维数组的行优先存储和列优先存储

①行优先存储：列下标变化速度最快。

下标从0开始，则Num[i,j]=i\*n+j+1

下标从1开始，则Num[i,j]=(i-1)\*n+j

②列优先存储：行下标变化速度最快。

下标从0开始，则Num[i,j]=j\*n+j+1

下标从1开始，则Num[i,j]=(j-1)\*n+i+1

注：若给定存储区的起始地址为Addr0，每个元素占c个单元，则元素A[i,j]在内存中的地址为Loc(i,j)=Addr0+(Num(i,j)-1)\*c

2、矩阵的压缩存储

①对称矩阵和三角矩阵。

对于对称矩阵：只存储矩阵的下三角或者上三角（包括对角线）部分的元素。将n(n+1)/2个元素存到数组SA[n(n+1)/2]中。

对于给定的下三角元素，其在SA中的序号由下式确定（数组和矩阵序号都从1开始记）：

Num(i,j)=1+2+3+...+(i-1)+j=i(i-1)/2+j

上三角元素互换i、j即可。

对于三角矩阵：同理。三角矩阵是指对角线以上或以下的元素全为0，或全为同一值。稍微调整就好。

②对角矩阵：除了主对角线和紧靠主对角线的上下若干条对角线外，其余元素全为0。（三对角矩阵：除了主对角线及其上下对角线上有非0元素外，其余位置全为0的矩阵）。

Num[i,j]=[3(i-1)-1]+[j-i+2]=2i+j-2

③稀疏矩阵的压缩存储：使用三元组表（行数、列数、非零元素个数）。

3、广义表

①广义表L是n个元素组成的有限序列，其中每个元素可以是不可分割的原子，也可以是广义表（称为子表）。另外，称元素个数n≥0为表长度；当n=0为空表时，记作L=()。

约定：在书写时，一般用小写字母表示原子，用大写字母表示广义表。

②广义表的基本操作：

取表头head(L)：把第一个元素取出来。

取表尾tail(L)：即“去表头”运算。

例.A=((a,b),c,(c,d),e)

则head(A)= ((a,b),c,(c,d))

tail(A)=(c,d)

## 第八章·树

1、树与树的相关概念

①树：树是n个结点构成的有限集合（n＞0，即没有空树），其中有一个结点叫做根。

②一个结点的度是指该结点的孩子数目。若一个结点的度为0，则称该节点为叶子结点，否则称为分支结点。

一棵树的度是树中最大的结点度。

③根的层次为1，其余结点的层次为父结点的层次数加1。最大的层次称为树的高度或深度。

2、二叉树与二叉树的相关性质

①二叉树是n个结点的有限集合，其中n≥0。

②二叉树有5种形态：空树、单结点二叉树、左子树为空右子树不空、右子树为空左子树不空、左右子树均不空。

③性质：

在二叉树的第i层上的结点数≤2ⁱ⁻¹

深度为k的二叉树的结点数≤2ᴷ-1

对任一棵非空的二叉树T，如果其叶子数为n₀，度为2的结点数为n₂，则有下面的关系式成立：n₀=n₂+1

证明：有 n=n₀+n₁+n₂

n-1=n₁+2n₂

（n个结点有n-1条边，分别是n₁和n₂贡献的）

有n个结点的完全二叉树（n＞0）的深度为log n₂向下取整再加1。

例题.已知完全二叉树有100个结点，则该二叉树有多少个叶子结点？

答：若认为每个结点均已编号，则最大的编号为100，其父结点的编号为50。则，从51到100均为叶子，因为叶子数为100-50=50。

④证明：假设一棵二叉树结点的个数为n，即该棵二叉树的前序遍历序列为q₁,q₂,q₃,...,qn，中序遍历序列为z₁,z₂,z₃,...zn，证明由二叉树的先序序列和中序序列能唯一确定一棵二叉树。

分析：算法的证明题一般采用数学归纳法，即当k=1（初始值）时成立，并假设k=n时成立推导出k=n+1也成立。

3、二叉树的存储结构及其遍历

①顺序存储结构：只适用于完全二叉树，否则会浪费很多空间。

②二叉链表存储结构及其遍历：

前序遍历

中序遍历

后序遍历

层次遍历

int BiTree::high(bnode \*T)

{ if(T==null) return 0;

else return max(high(T->lchild),high(T->rchild))+1;

}

4、二叉树与线索二叉树

①约定：

ltag=0：lchild指示该结点的左孩子

ltag=1：lchild指示该结点的前驱

rtag=0：rchild指示该结点的右孩子

rtag=1：rchild指示该结点的后继

②中序线索二叉树中求解先序前驱和先序后继。

前驱：

若T->ltag=1，直接可以求得前驱

若T->ltag=0，说明有左子树：

由中序遍历知，根结点的前驱是其左子树最右下处的叶子节点。

由此可得，中序线索二叉树的中序前驱：

bnode \*insta(bnode \*p)

{ bnode \*q=p->lchild;

if(p->ltag==1) return q;

else

{ while(q->rtag==0)

{ q=q->rchild;

}

return q;

}

}

同理得，中序线索二叉树的中序后继：

bnode \*innext(bnode \*p)

{ bnode \*q=p->rchild;

if(q->rtag==1) return q;

else

{ while(q->ltag==0)

{ q=q-lchild;

}

return q;

}

}

③先序线索二叉树的前序前驱与后继。

先序前驱：很难找，而且建立在三叉链表能追溯父结点的基础上才能实现。

先序后继：如果结点有左子树，那么该结点的左孩子是它的后继；如果结点没有左子树，但是有右子树，则右孩子是它的后继。

bnode \*presuc(bnode \*p)

{ if(p->ltag==0) return p->lchild;

else return p->rchild;

}

④设计出按先序次序遍历先序线索二叉树的非递归算法。

分析：若没有线索可用，则非递归算法需要借助于栈；若有线索可用，则可借助于线索关系实现。

代码：

void preorder(bnode \*T)

{ bnode \*p=T;

while(p!=null)

{ visite(P);

P=presuc(P);//求后继

}

}

⑤后序线索二叉树的后序后继与前驱

前驱比较好找，如果它有右子树，则右子树的根结点是它的前驱，如果没有右子树，则左子树的根结点是它的前驱。

5、树的存储结构

①双亲表示法：便于搜索相应结点的父结点及祖先结点，但若要搜索孩子结点及其后代结点，则需搜索整个表，因而较费时间。

②孩子链表表示法：将每个结点的孩子结点连成一个链表，然后把各表头指针放在一个表中构成一个整体结构。便于搜索任意结点的孩子结点及后代结点，但不便于搜索各结点的双亲结点和祖先结点。

③孩子-兄弟链表表示法（又叫二叉链表表示法）：对树中每个结点用一个链表结点来存储，每个链表结点中除了存放结点的值外，还有两个指针，一个用于指示该结点的第一个孩子结点，另一个用于指示该节点的下一个兄弟结点。

6、树与二叉树的转换

①森林（树）到二叉树：同级的结点视作右孩子，后代结点视作左孩子。连接为二叉树。

②二叉树到森林（树）：右孩子还原为同级的兄弟结点，左孩子还是自己的孩子，还原组织结构就行。

7、树的遍历方式

①先序遍历森林：相当于二叉树的前序遍历。多棵树的遍历按顺序一棵树一棵树来。

②后序遍历森林：相当于对应二叉树的中序遍历（直接后序可能出错 建议转换后再说明顺序）。

8、哈夫曼树

①应用：缩短字符编码。

②构造：选取这一堆中根植最小的两个二叉树构造一棵新的二叉树。

③带权路径长度WPL：其等于所有分支结点的值。

## 第九章·图

1、图的相关概念

①每个对象用“顶点”表示；两个元素间的关系表示为一条弧或边，弧是单向的< >，边是双向的()。

相邻接，邻接点。A邻接到B，B邻接于A。

②若给每条边/弧再附加一个数值作为权值，这样的带权图就称为网络。

③若无向图中任意两点间都有一条边，则整个图有n(n-1)/2条边，则称该图为无向完全图。

若有向图中任意一个顶点到其余各点间均有一条弧，则该图为有向完全图。

④简单路径：某路径中所经过的顶点不重复

回路（环）：某路径的首尾顶点相同。

⑤若无向图G中任意两点之间均存在路径，则称G为连通图，否则不联通，就存在若干连通分量。

若有向图中任意两点间可以相互到达，则称为强连通图。

⑥与一个顶点相邻接的顶点数称为该顶点的度。在有向图中，进入一个顶点的弧数称为该顶点的入度，从一个顶点发出的弧数称为该顶点的出度，度=入度+出度。

注：

①树的三种等价表述：联通的无环图；有n-1条边的连通图；有最少边数的连通图。

②若有向图中仅有一个顶点的入度为0，其余顶点的入度为1，则称该图为有向树，并称其中入度为0的顶点为其有向根。

|  |  |
| --- | --- |
| 易混概念总结 | |
| 连通图 | 无向图连通 |
| 强连通图 | 有向图连通 |
| 无向完全图 | 无向图两两之间存在路径 |
| 有向完全图 | 有向图两两之间存在双向路径 |

2、图的描述及其存储结构

①邻接矩阵表示：n个顶点的图，用A[n][n]来表示。（不存在的边用0或者∞表示）

②邻接表表示：将每个顶点的邻接点连成链表，并将各链表的表头指针合在一起（用数组或链表均可），其中每个头指针与该结点的信息合为一个整体结点。

逆邻接表：将邻接表中各顶点的邻接表变为其前驱顶点表即可。

3、图的深度优先遍历算法和广度优先遍历算法

①深度优先遍历借助于栈，一般更为广泛。不断找某结点的邻接点，未访问的邻接点成为新的结点继续找下去，直到找不到才通过函数断点返回。

void DFS(Graph G,int v){

int w;

for(int i=1;i＜=n;i++){

visited[i]=false;

}

//这里表示访问了新的顶点

visit(v);

visited[v]=true;

w=firstadj(G,v);

while(w!=0){

//这里表示找到了边

if(visited[w]==false) DFS(G,w);

w=nextadj(G,v,w);

}

}

②广度优先遍历借助于队列，一般用于求解最短路径问题。其思想是每访问一个结点都将其加入到队列中，然后，只要队列不为空，就依次取元素，找这些元素的邻接点。

void BFS(Graph G,int v0){

int w;

Queue q;

init(q);

for(int i=1;i＜=n;i++){

visited[i]=false;

}

visit[v0];

Enqueue(q,v0);

visited[v0]=true;

while(!empty(q)){

Dequeue(q,v);

w=firstadj(G,v);

while(w!=0){

if(!visited[w]){

visited[w]=true;

Enqueue(q,w);

}

w=nextadj(G,v,w)

}

}

}

注：对图的操作都是建立在深度遍历和广度遍历之上的，只是具体visit操作有差异。

|  |  |  |  |
| --- | --- | --- | --- |
| 时间复杂度总结 | | | |
| 算法名称 | 存储方式 | 时间复杂度 | 说明 |
| DFS | 邻接矩阵 | o(n²) | 对每个顶点来说，搜索其所有邻接点需要搜索矩阵中对应的整个一行，因此对整个图来说时间复杂度是o(n²) |
| DFS | 邻接表 | o(n+e) | 若邻接表有n个结点和e条边，对每个顶点来说，需要搜索邻接表对应的链表的各结点，时间复杂度是o(n+e) |
| BFS | 邻接矩阵 | o(n²) | 同深度遍历，o(n²) |
| BFS | 邻接表 | o(n+e) | 每个顶点均需搜索一次，时间复杂度是o(n)，在搜索任一顶点的邻接点时，每条边至少访问一次，故时间复杂度为o(e)，算法总时间复杂度为o(n+e) |
| 最小生成树·Prim | 邻接矩阵、邻接表 | o(n²) |  |
| 最小生成树·Kruskal | 邻接矩阵、邻接表 | 不讨论 |  |
| 最短路径·Dijkstra | 邻接矩阵、邻接表 | o(n³) | 整个算法的主体部分要循环n-1次以求解从给定顶点到其余n-1个顶点的最短路径，这需要搜索整个最短路径长度数组dist的n个元素，因此无论采用什么存储结构，其时间复杂度都是o(n²) |
| 最短路径·Floyd | 邻接矩阵、邻接表 | o(n³) | 借中间顶点判断路径。用矩阵，下标从1开始。借用矩阵来判断，时间复杂度都是o(n³)。 |
| 拓扑排序算法 | 邻接矩阵、邻接表 | o(n+e) | 若AOV网有n个顶点、e条边，在拓扑排序的过程中，搜索入度为0的顶点所需的时间是o(n)。在正常情况下，每个顶点入一次栈，出一次栈，所需时间o(n)。每个顶点入度减1的运算共执行了e次。所以总的时间复杂度是o(n+e)。 |

## 第十章·查找

|  |  |
| --- | --- |
| 几种查找方式辨析 | |
|  | 说明 |
| 简单顺序查找 | ①简单顺序查找对数据表的特性没有要求，无论是有序还是无序，都行。  ②C语言中数组的下标是从0开始的，但是考虑到简单顺序查找算法的特点，将数组中存储元素的下标范围定为1～n，A[0]存放哨兵。因此，存储数组需要描述为A[n+1]。  ③实现查找时，搜索方向可以是1～n，也可以是n～1，方便起见还是从后往前搜索好。  ⑤假设每个元素的查找概率相同，则由于各元素的查找长度依次取值为从1到n。  查找成功的平均查找长度：  查找成功的话，第1个元素比较1次，..，第n个元素比较n次，它们之和除以数量n，就是平均查找长度。  ASL=(1+2+3+...+n)/n=(n+1)/2  查找失败的查找长度：由于这里是使用哨兵的简单顺序查找，易知，其查找失败的平均查找长度是n+1。  注：如果该顺序表是有序的，查找长度都会改变。 |
| 有序表的二分查找 | ①查找过程：设查找区域的首尾下标分别用变量名low和high表示（初值分别为0和n-1），其中数组的下标从0开始，这与简单顺序查找中略有不同哦。  ②什么时候查找失败？low＞high的时候查找失败。  ⑤二分查找判定树：其中，其结点的值并不意味着它的权值，这只是个序号而已，即被查找的次序。  ⑥在有序表中查找任一元素的查找长度取决于该结点在相应的判断树上的层次数，因而不会超过该树的深度。  在等概率情况下，二分查找算法的平均查找长度为log₂(n+1)-1。  查找成功的平均查找长度  查找失败的平均查找长度 |
| 索引顺序表的查找 | ①分块有序：整个表中的元素未必有序，但若划分为若干块后，每一块中的所有元素均小于（或大于）其后面块中的所有元素。这种特性为分块有序。  ②对分块有序表的查找，可为其建立一个索引表，索引表中为每一块设置一索引项，每一索引项包括两部分：该块的起始地址和该块中最大（或最小）关键字的值。  ③查找两步走：首先要通过在索引表中查找以确定元素所在的块，然后在所确定的块中进行查找。由于索引表按关键字递增（递减）有序，因此，在索引表中的查找既可以采用简单顺序查找，也可以采用二分查找，这取决于索引表的项数。  ④其时间性能介于顺序查找和二分查找之间。 |
| 二叉排序树及其查找 | 二叉树的删除操作  若删除结点是叶子结点，直接删掉  若删除结点只有左子树或者只有右子树，直接删掉该结点，然后令所删除结点的子树直接连接在被删除结点的双亲结点即可。  若删除结点既有左子树又有右子树，先沿着p的左子树根结点的右指针一直往右走，直到来到其右子树的最右边的一个结点。然后把删除结点的关键字赋为它，再删除这个结点（依据实际情况遵照第一种或第二种情况删除）。 |

## 第十一章·排序

几种排序分类

①增排序和减排序

②内部排序和外部排序

③稳定排序和不稳定排序

各大排序算法与时间性能总结

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 排序算法相关汇总 | | | | |
| 排序算法的分类：  插入排序：直接插入排序、希尔排序  交换排序：冒泡排序、快速排序  选择排序：直接选择排序、堆排序  归并排序  基数排序 | | | | |
| 排序算法 | 最好情况时间复杂度 | 最坏情况时间复杂度 | 平均时间复杂度 | 是否稳定 |
| 直接插入排序 | o(n) | o(n²) | o(n²) | 稳定 |
| 希尔排序 | o(nlog₂n) | o(nlog₂n) | o(nlog₂n) | 不稳定 |
| 冒泡排序 | o(n) | o(n²) | o(n²) | 稳定 |
| 快速排序 | o(nlog₂n) | o(n²) | o(n²) | 不稳定 |
| 直接选择排序 | o(n²) | o(n²) | o(n²) | 不稳定 |
| 堆排序 | o(nlog₂n) | o(nlog₂n) | o(nlog₂n) | 不稳定 |
| 归并排序 | o(nlog₂n) | o(nlog₂n) | o(nlog₂n) | 稳定 |
| 1、o(nlog₂n)基本是因为要进行log₂n趟，每趟处理n个元素。  2、快速排序在乱序的时候最好用，但若原有数据基本有序，它的时间复杂度将大大上升。  3、归并排序是对顺序表的操作，由于在顺序表中不能进行就地归并，因而需另外开辟存储区以存放归并结果，所以归并排序需要与原表等量的辅助存储空间。  4、只有堆排序不受初始数据影响。  5、时间复杂度比较：  O(1)＜O(log₂ n)＜O(n)＜O(nlog₂n)＜O(n²)＜O(n³) | | | | |
| 插入排序·直接插入排序：常规、监视哨  ◎空间性能：仅需要一个记录的辅助存储空间  ◎时间性能：整个算法循环n-1次，每次循环基本操作是比较和移动元素，这取决于数据表的初始特性。  最好情况：数据表开始时已经有序，每次循环只需要比较1次，移动2次。所谓移动两次，一次是比较之前必须赋值给temp或者哨兵，另一次是比较之后A[j+1]=temp或者A[0]；最坏情况：数据表开始时是逆序，每次循环都要比较i次，移动i+1次。这里所说的移动i+1次，除了上述提到的必备2次，还有移动i-1个元素的次数。  最好情况 最坏情况 平均  比较次数 n-1 (n+2)(n-1)/2  移动元素次 2(n-1) (n+4)(n-1)/2  时间复杂度 O(n) O(n²) O(n²) | | | | |
| 插入排序·希尔排序  ①思想：将待排序列划分为若干组，在组内进行直接插入排序，以使整个序列基本有序，然后再对整个序列进行直接插入排序。  ②如何分组？给定一个步长d，将步长相差为d的元素划为一组，然后每一轮d=d/2。  ③时间性能：每趟在上一趟的基础上完成，可认为是基本有序，因而每趟的时间复杂度为O(n)。由于每次按上一次步长一半的方式进行，故需要log₂n趟排序。  故，时间复杂度：O(nlog₂n) | | | | |
| 交换排序·冒泡排序  ①时间性能：每趟比较n-1,n-2,...,1，故时间复杂度为O(n²)  ②改良后的冒泡算法（若是一趟下来没有交换元素，说明已经有序，停止比较）：  最好情况：比较n-1次，交换0次，时间复杂度O(n)  最坏情况：比较次数和交换次数都是n(n-1)/2，时间复杂度O(n²) | | | | |
| 交换排序·快速排序  ◎时间性能：  最好情况：o(nlog₂n)  最坏情况：o(n²)  平均情况：o(nlog₂n) | | | | |
| 选择排序·直接选择排序  ◎时间性能：比较n(n-1)/2次，交换次数最多为n-1次，因而时间复杂度为O(n²) | | | | |
| 选择排序·堆排序  ①初始建堆，先按序列建立完全二叉树，再逐个调整。插入插到最右下面，再逐个调整。  ②时间复杂度：o(nlog₂n) | | | | |
| 归并排序  ①思想：将整个表看做是n个有序子表，每个子表的长度为1，然后两两归并，得到n/2个长度为2的有序子表。然后再两两归并，直至得到一个长度为n的有序表为止。  ②一趟排序的时间复杂度为o(n)，又因需要log₂n趟归并，故时间复杂度为o(nlog₂n) | | | | |
| 基数排序：按不同关键字进行排序，如先按个位数字、再按十位数字、最后按百位数字。 | | | | |

# 第二部分·重点考察

|  |  |  |
| --- | --- | --- |
| 章节 | 非算法题考点 | 算法题考点 |
| 概论 | 时间复杂度的计算 |  |
| 栈与队列 | 1、出栈序列（有多少种、是否合法）  2、顺序栈、链栈、顺序队列、循环队列的判空、判满、入栈、出栈操作 | 符号匹配（可能考察） |
| 线性表 | 1、带头结点与不带头结点的单链表、单循环链表、双循环链表判空、判满、插入、删除操作  2、广义表中取表头和取表尾操作  压缩存储中二维数组、上下三角元素的存储 | 1、递增有序顺序表求交集、并集：  ◎A=A∩B（2015）  ◎C=A∩B（2018）  ◎A=A∪B（2021：单循环链表、不带头结点）  可能考察：C=A∪B  2、在递增有序顺序表中插入结点，使其仍然有序：  ◎假设递增有序的链表L表示一个集合，试设计算法在表中插入一个值为x的元素结点，使其仍然保持递增有序。（2016、2019：带头结点、不带头结点）  3、单链表的逆置（2017）  可能考察：顺序表的逆置  4、删除重复出现的元素：  已知顺序表A中元素按非降次序排列，请设计一个高效的算法以删除A中重复出现的元素，并分析算法的时间复杂度（2020） |
| 树 | 树的形态问题：什么样的树结点数等于高度数、有着n个结点的树/二叉树有几种形态、先序序列和中序序列相同的二叉树特点、先序序列和后序序列相反的二叉树特点  已知结点数，推测完全二叉树的可能形态  二叉树的线索化及线索二叉树的空链域问题  根据访问的完全序列或部分序列构造二叉树  根据树的前序和后序序列访问构造树  哈夫曼树的性质(以只有度为0和2的结点为纲，去推)  哈夫曼树的构造  可能考察：证明：假设一棵二叉树结点的个数为n，即该棵二叉树的前序遍历序列为q₁,q₂,q₃,...,qn，中序遍历序列为z₁,z₂,z₃,...zn，证明由二叉树的先序序列和中序序列能唯一确定一棵二叉树。 | 1、二叉树前序、中序、后续遍历的递归与非递归实现、线索二叉树的建立、线索二叉树前驱和后继结点的查找：  ◎设计一个非递归算法以输出二叉树t中先序序列中最后一个结点的值（2014）  ◎设计一个按先序次序遍历先序线索二叉树的非递归算法，并不许用栈。（2019）  ◎设计算法按先序次序输出二叉树中每个结点的值及其所对应的层次数（2015）  ◎设计算法输出二叉树T的每个结点的值及其对应的层次数。（2021）  2、完全二叉树二叉链表结构与顺序存储结构的转化：  ◎设计算法将以二叉链表T存储的二叉树转换为对应的顺序存储结构A[max]中。要求：空的元素用null表示，并返回所存储的最大的元素下标。（2016）  ◎设计算法将二叉树的二叉链表存储结构转换为对应的顺序表结构，并存储到数组A[n]中。（2018）  3、判断是否是树、二叉排序树：  ◎已知给定的二叉树T中各结点的值两两不同，设计算法以判断T是否是二叉排序树（2020）  ◎设计算法以判断有向图G是否是一棵以V0为根的有向树。若是有向树，则返回TRUE，否则返回FALSE。（2021） |
| 图 | 1、最短路径问题：Dijkstra算法（考察频率最高）、Floyd算法（近年趋势）  2、关键路径问题（近年趋势）  3、最小生成树问题：Prim算法、kruscal算法  4、拓扑排序问题、逆拓扑排序排序问题、代码下的拓扑排序  5、连通图、强连通图、无向完全图、有向完全图的概念、要求辨析 | 1、深度优先遍历的应用：  ◎遍历并以(a,b)形式输出每个联通分量（2014）  ◎判断无向图、有向图中vi到vj是否存在路径（2015、2017）  ◎判断无向图是否是树（2016）  ◎判断有向图、无向图是否联通（2018、2019）  2、广度优先遍历的应用：  ◎设计算法以递增有序数组int A[n]中元素为输入数据，构造一棵平衡的二叉树。（2017）  ◎在连通无向图G中，求出距离顶点v最短路径长度为最远的一个顶点（2020） |
| 查找 | 1、线性探测法、拉链法构造哈希表，及其等概情况查找成功、失败的平均查找长度计算  2、折半查找判定树的构造、等概查找成功时的平均查找长度（考察最多）、查找失败时的平均查找长度（可能考察）  3、二叉排序树的构造、等概查找成功时的平均查找长度（考察较多）、查找失败时的平均查找长度（可能考察）  4、平衡二叉树的构造、某一层最小平衡二叉树的结点个数  5、简单顺序查找、二分查找的特点  可能考察：B树的性质 | 可能考察：简单顺序查找的实现、二分查找的递归实现、非递归实现 |
| 排序 | 1、插入排序（直接插入排序、希尔排序）、交换排序（冒泡排序、快速排序）、选择排序（直接选择排序、堆排序）、归并排序的算法特点、元素交换次数、时间复杂度、是否稳定、排序过程复现  2、曾经考察课本之外的：基数排序 | 快速排序的实现（2014）  可能考察：所有常见算法的代码实现 |