子串“ABC”在主串“AABCABCD”中的位置为2或5。 （F）

原因：第一次出现的位置

线性表采用链表存储时，查找第i个元素的时间与i的值无关。 （F）

图的深度优先遍历序列和广度优先遍历序列不是唯一的。 （F）

采用线性探测法处理散列时的冲突，当从哈希表删除一个记录时，不应将这个记录的所在位置置空，因为这会影响以后的查找。 （T）

1．设有一个10阶的对称矩阵A，采用压缩存储方式，以行序为主存储，a11为第一元素，其存储地址为1，每个元素占一个地址空间，则a85的地址为（ B ）。

A. 13 B. 33 C. 18 D. 40

3．循环队列存储在数组A[0..m]中，则入队时的操作为（ D ）。

A. rear=rear+1 B. rear=(rear+1) mod (m-1)

C. rear=(rear+1) mod m D. rear=(rear+1)mod(m+1)

**插入排序与选择排序的区别：**

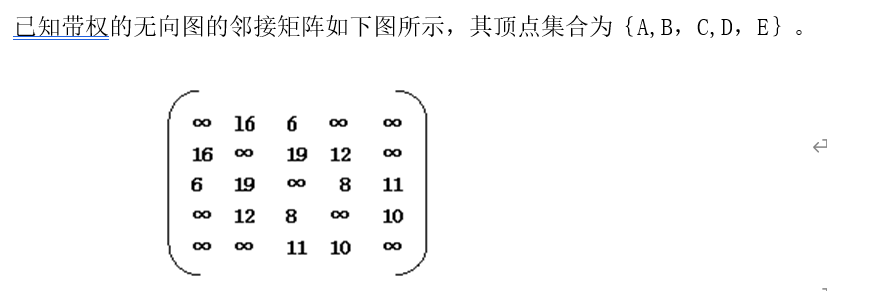
[**https://blog.csdn.net/vpurple\_/article/details/126568614**](https://blog.csdn.net/vpurple_/article/details/126568614)

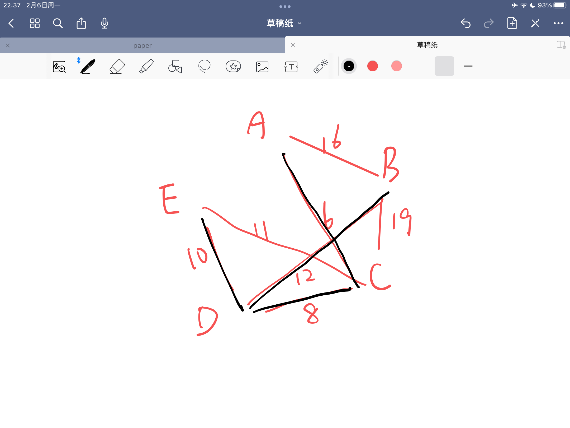
具有10个顶点的连通图，其最小生成树的边数为\_\_\_9\_\_\_。

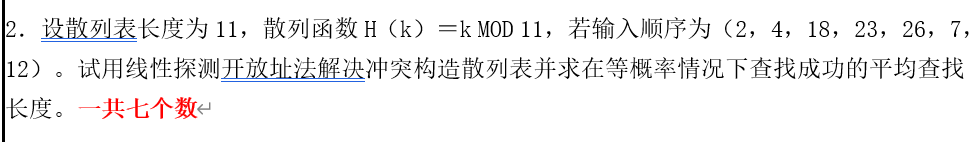
在一棵m阶B-树中，每个非叶子结点至多有\_\_ \_m-1\_\_\_ 个关键字。 最少[m/2]（向上取整）-1个

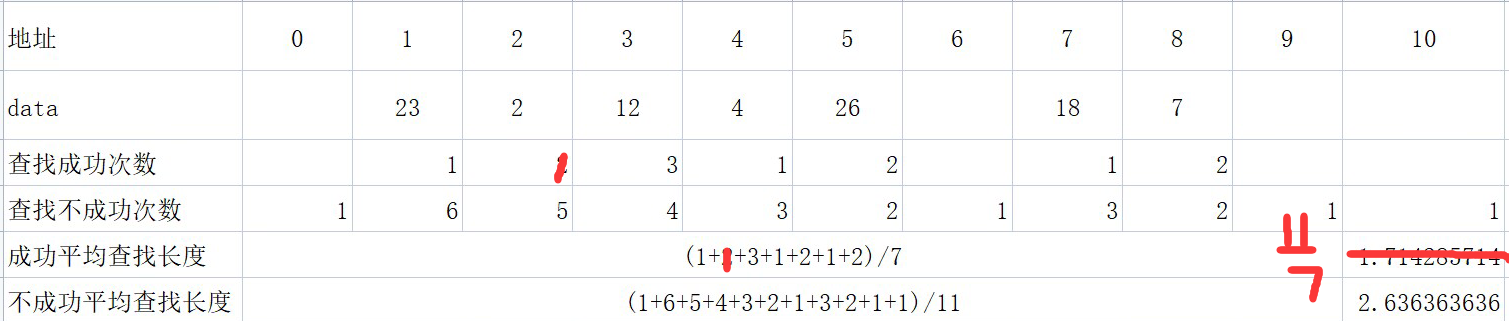
快速排序平均时间复杂度为O(nlogn) ,最坏情况下时间复杂度是\_\_\_\_O(n^2)\_\_\_\_ 。

**（纯有序的时候最慢了）**



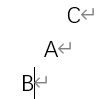






***只有前序和后序遍历无法还原二叉树！（因为无法确定左右子树）***

Eg: 前序和后序都是CAB

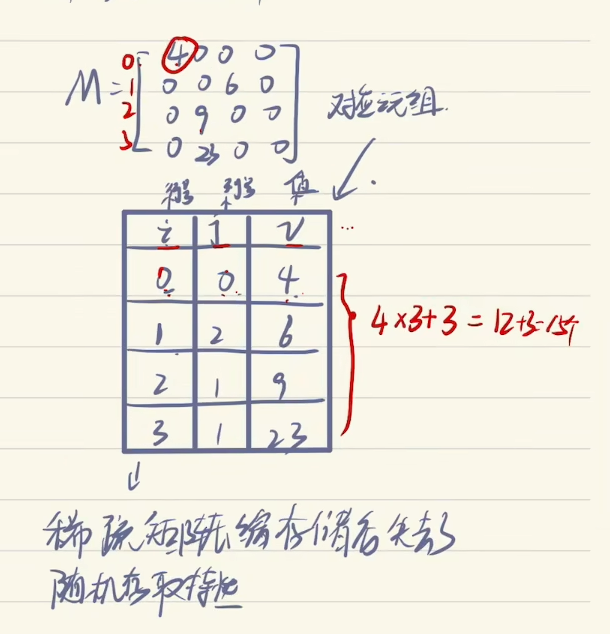


C

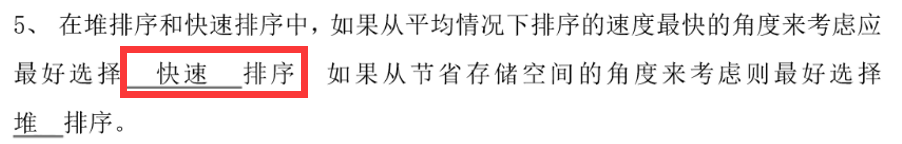
A B

一个n维数组可以看成由若干个n-1维数组组成的线性表。 （T）

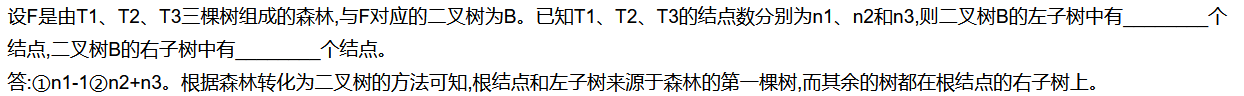
稀疏矩阵的压缩存储可以用一个三元组表来表示稀疏矩阵中的非0元素。 （T）





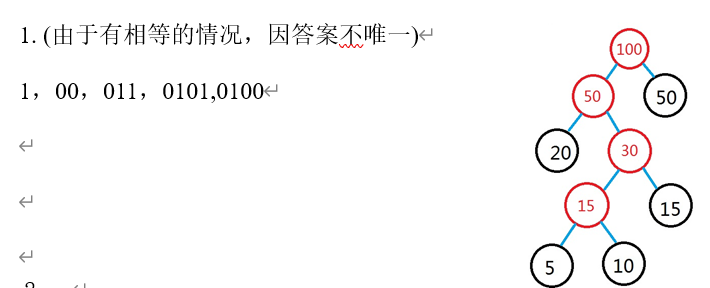


n在很大的时候，O(n)>O(logn)

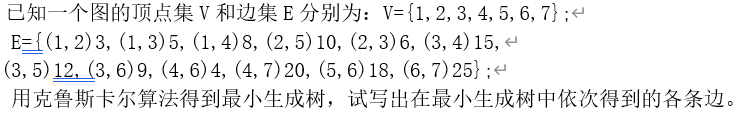


设某无向图中有n个顶点（表头结点）e条边（表结点），则建立该图邻接表的时间复杂度为（O(n+e) ）。

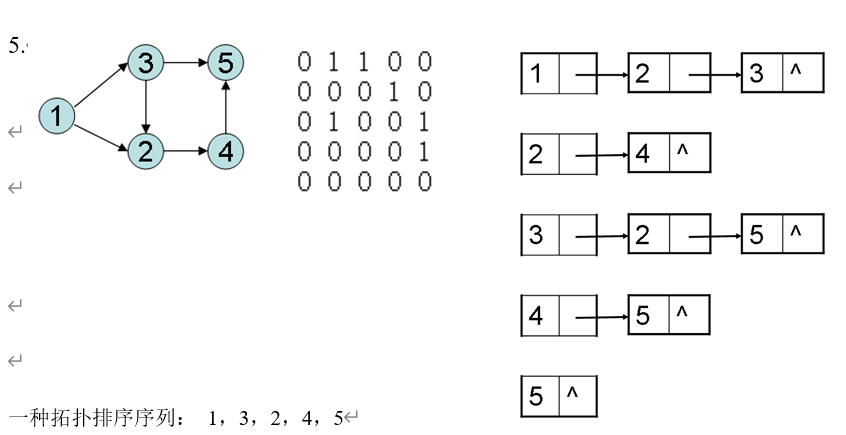
设在某通信系统中使用了5个字符，它们出现的次数分别为10，20，50，15，5，试构造一棵赫夫曼树，并给出赫夫曼编码。（左0右1，左小右大）



设一组有序的记录关键字序列为(13，18，24，35，47，50，62，83，90)，查找方法用二分查找，要求计算出查找关键字62时的比较次数，给出head，tail，mid的位置变化，并计算出查找成功时的平均查找长度 = 25 / 9。

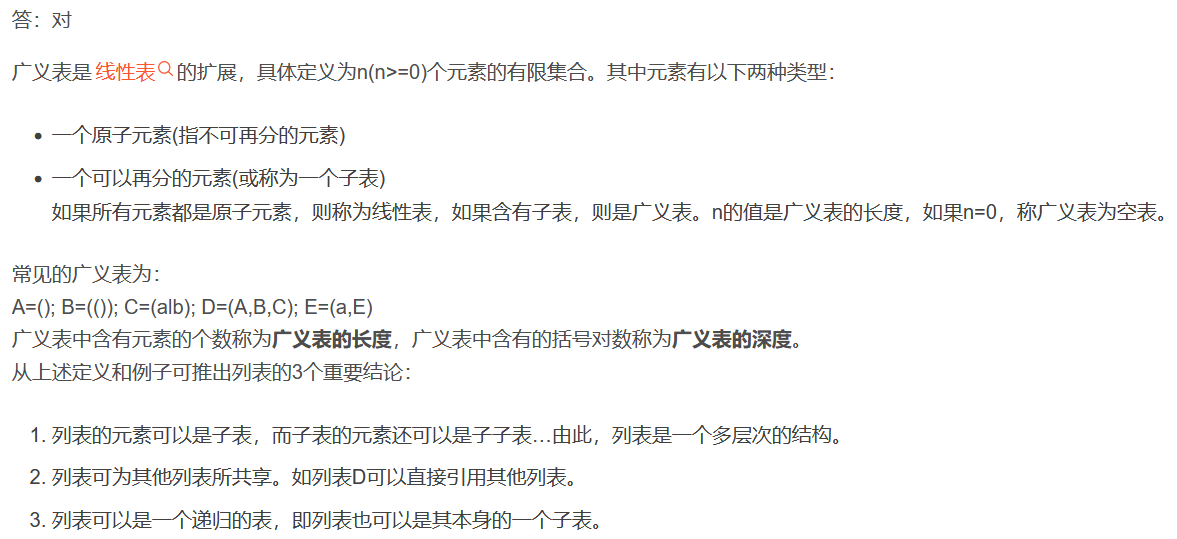
看一眼邻接表和邻接矩阵的样子:



冒泡排序：



广义表的同级元素（直属于同一个表中的各元素）具有线性关系。 （ √ ）



设一个链表最常用的操作是在末尾插入结点和删除尾结点，则选用( D )最节省时间。

A. 单链表 B.单循环链表 C. 带尾指针的单循环链表 D.带头结点的双循环链表

假设以行序为主序存储二维数组A=array[1..100，1..100]，设每个数据元素占2个存储单元，基地址为10，则LOC[5，5]=（ 818 ）。

具有10个叶结点的二叉树中有（ B ）个度为2的结点。

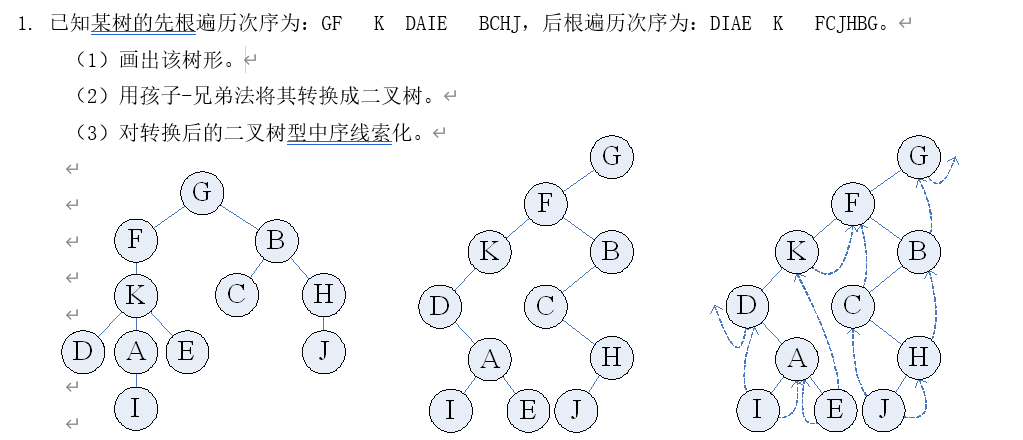
A．8 B．9 C．10 D．ll

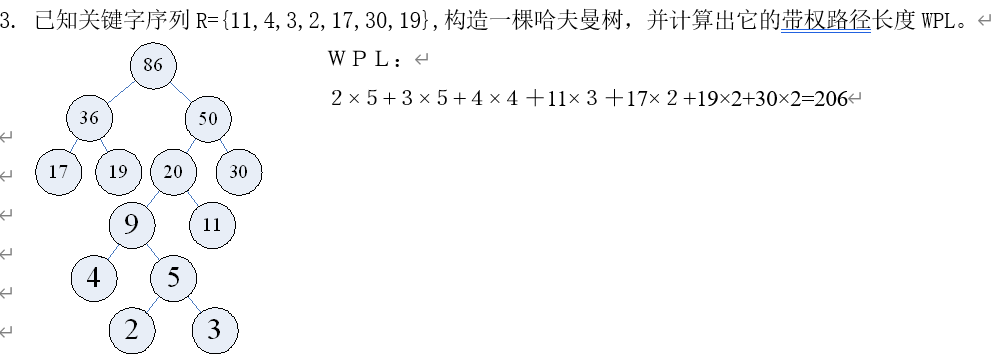
原因：一棵树中的结点个数相同：n0 + n1 + n2 = 2n2 + n1 + 1（根节点）

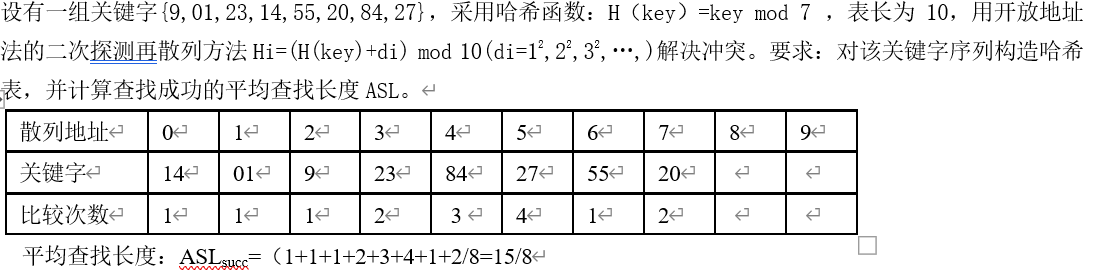
**设循环队列存放在向量sq.data[0:M]中，则队头指针sq.front在循环意义下的出队操作可表示为sq.front=(sq.front+1)%(M+1); return (sq.data(sq.front))；，若用牺牲一个单元的办法来区分队满和队空（设队尾指针sq.rear），则队满的条件为\_(sq.rear+1)%(M+1)==sq.front;。**

设只含根结点的二叉树的高度为0，则高度为k的二叉树的最大结点个数为\_2^(k+1)-1\_,最小结点个数为\_k+1\_。

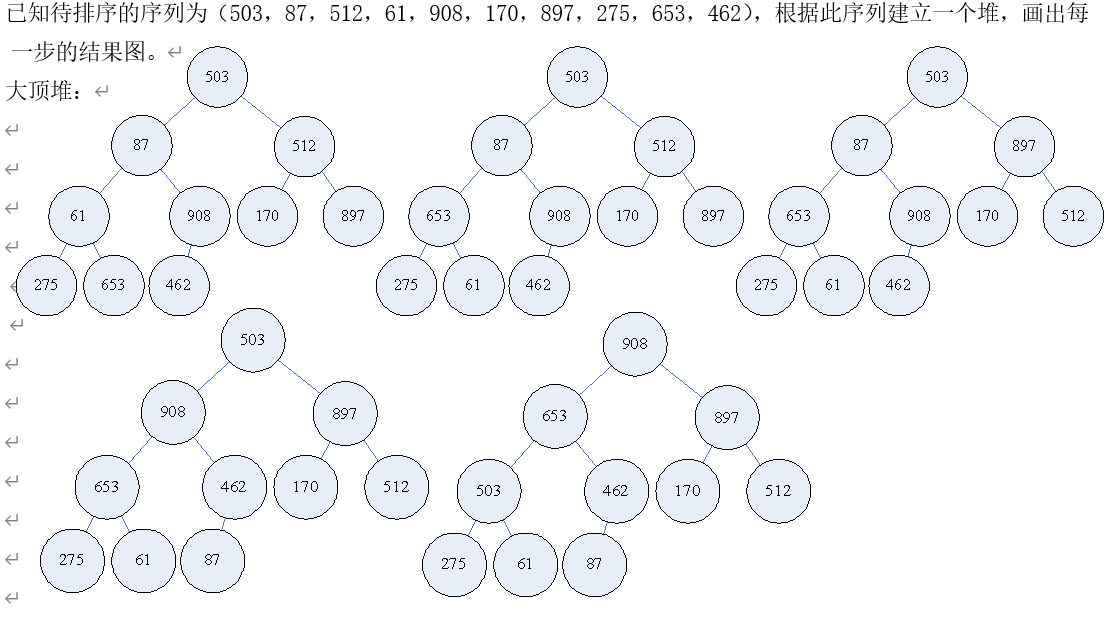
在图G的邻接表表示中，每个顶点邻接表中所含的结点数，对于无向图来说等于该顶点的\_\_度\_\_\_\_\_；对于有向图来说等于该顶点的\_\_\_\_出度\_\_\_\_\_\_\_。



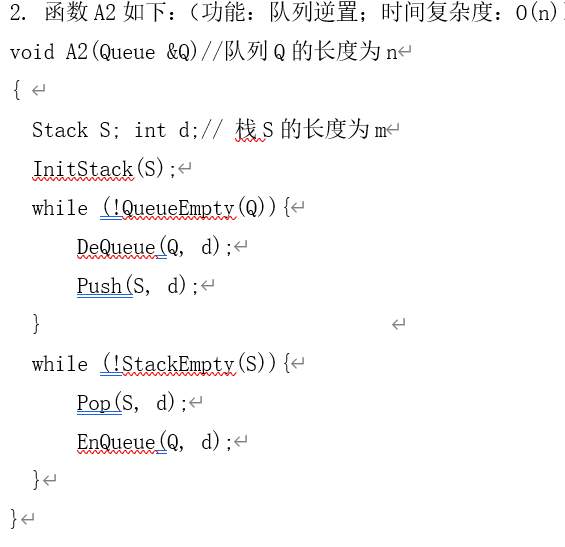


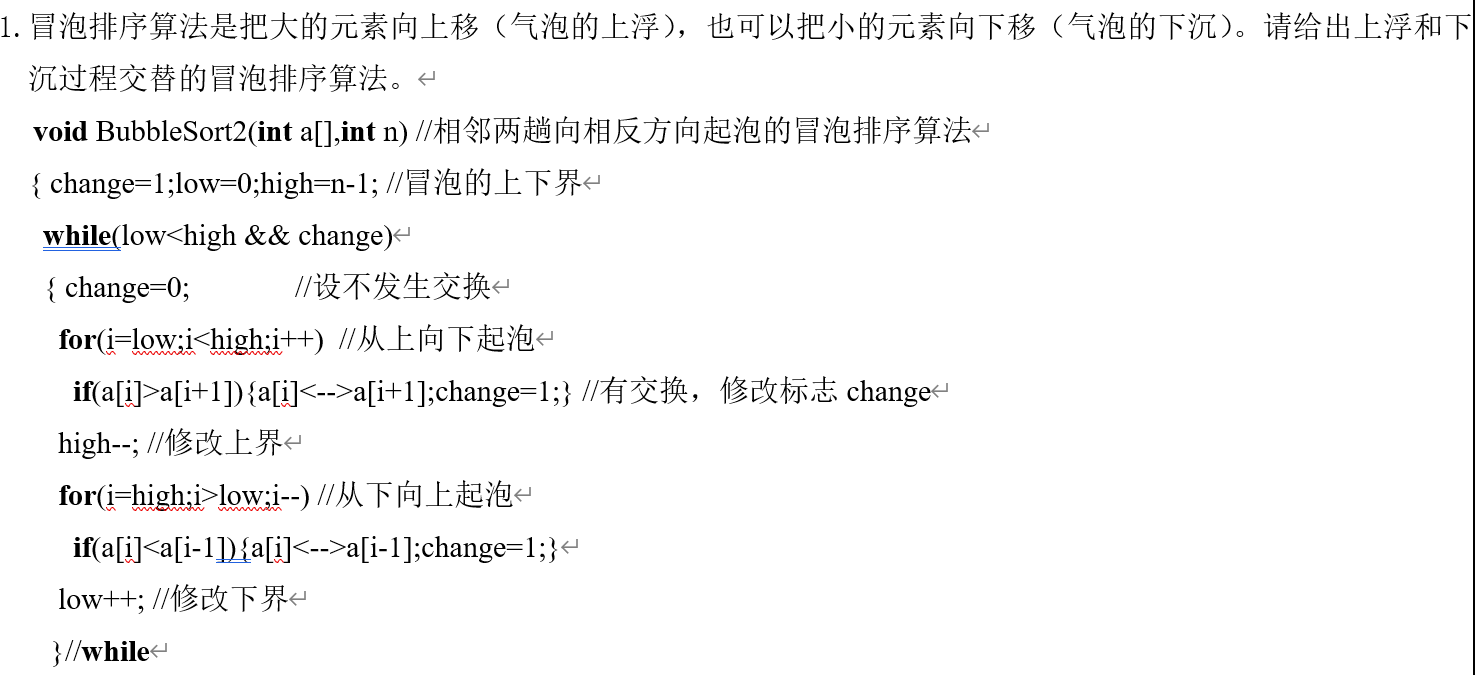


**这个就是大顶堆先建立再调整的例子，但是考试的时候建议插入一个调整一个。**



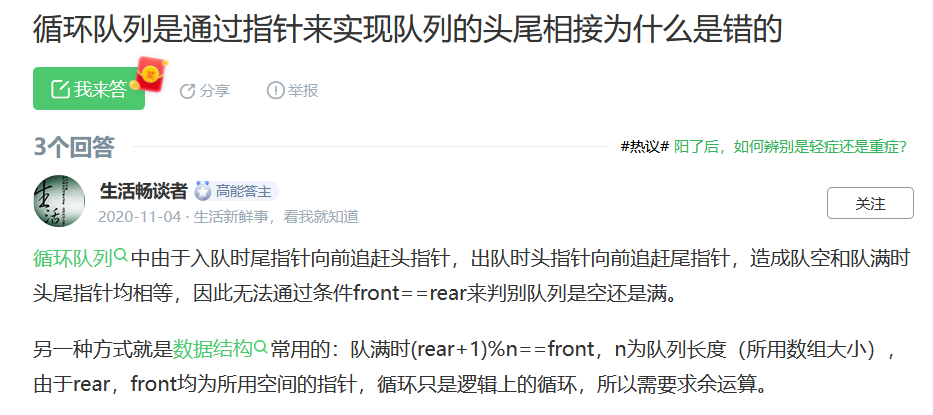
空间换时间



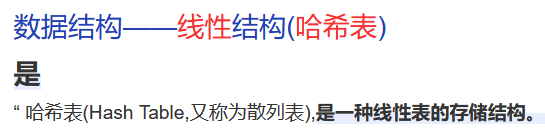


**装载因子的计算公式是: 散列表的装载（负载）因子 = 填入表中的元素个数/散列表的长度**

有向图中结点的度 等于 出度 + 入度



队空时，头尾指针应该相等；队满时，头指针应该正好在尾指针后1位；入队时，尾指针加1，出队时，头指针加1；对Maxsize取模是因为是循环队列，并防止溢出。



若长度为n的线性表采用顺序存储结构，在其第i个位置插入一个新元素的算法的时间复杂度为 O(n)

循环队列A[0..m-1]存放其元素值，用front和rear分别表示队头和队尾，则当前队列中的元素数是 (rear-front+m)%m

设有两个串p和q，其中q是p的子串，求q在p中首次出现的位置的算法称为 匹配

具有12个关键字的有序表，折半查找的平均查找长度为 3.1

**直接选择排序也称简单选择排序**，它的基本思想是：从一列数中找出最小的，和第一个交换；剩下的重新找出最小的，和这列数的第二个交换，......一直进行n-1次比较之后，该数列已经为有序数列了。

顺序栈用data[1..n]存储数据，指向实际栈顶元素的栈顶指针是top，则值为x的元素入栈的操作是：data[++top] = x

组成串的数据元素只能是（字符）

将整型数组A[1..8，1..8]按行优先次序存储在起始地址为1000的连续的内存单元中，则元素A[7，3]的地址是（1100）。= 1000 + 48 \* 2 + 2 \* 2

数组A[1..5,1..6]每个元素占5个单元，将其按行优先次序存储在起始地址为1000的连续的内存单元中，则元素A[5,5]的地址为: 1140

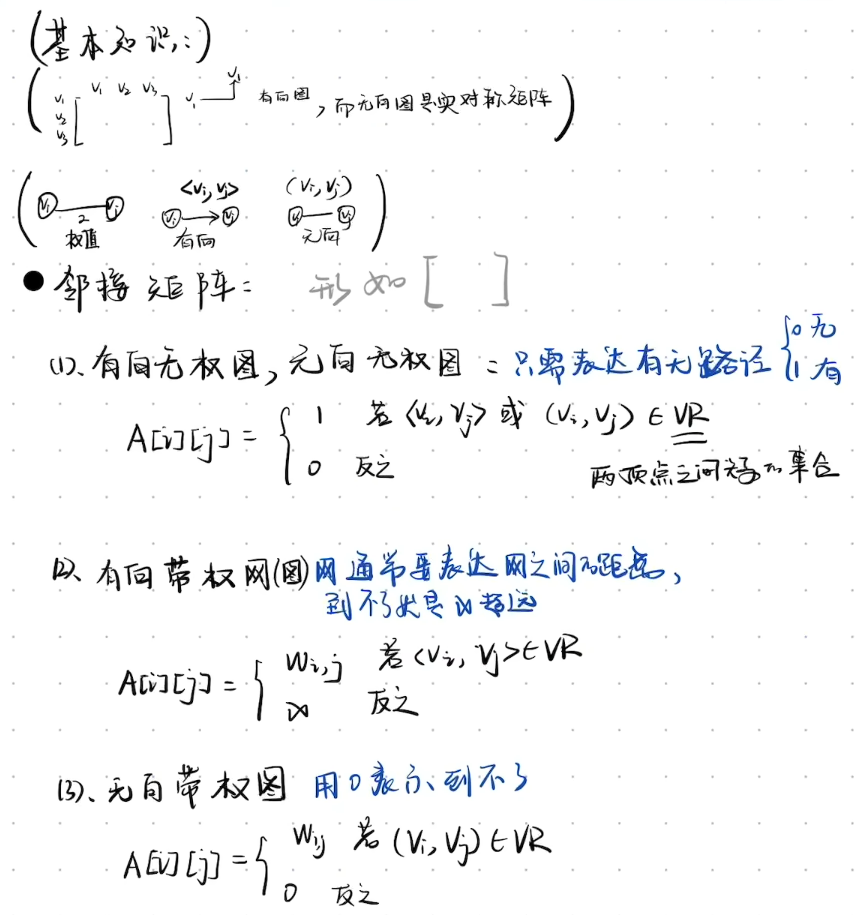
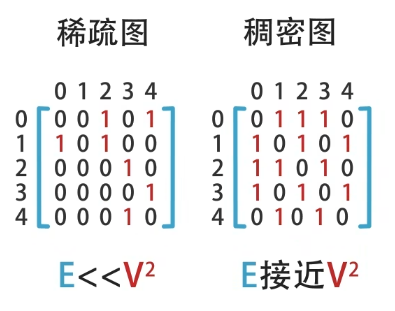
G是一个非连通无向图，共有28条边，则该图至少有（ 9 ）个顶点。

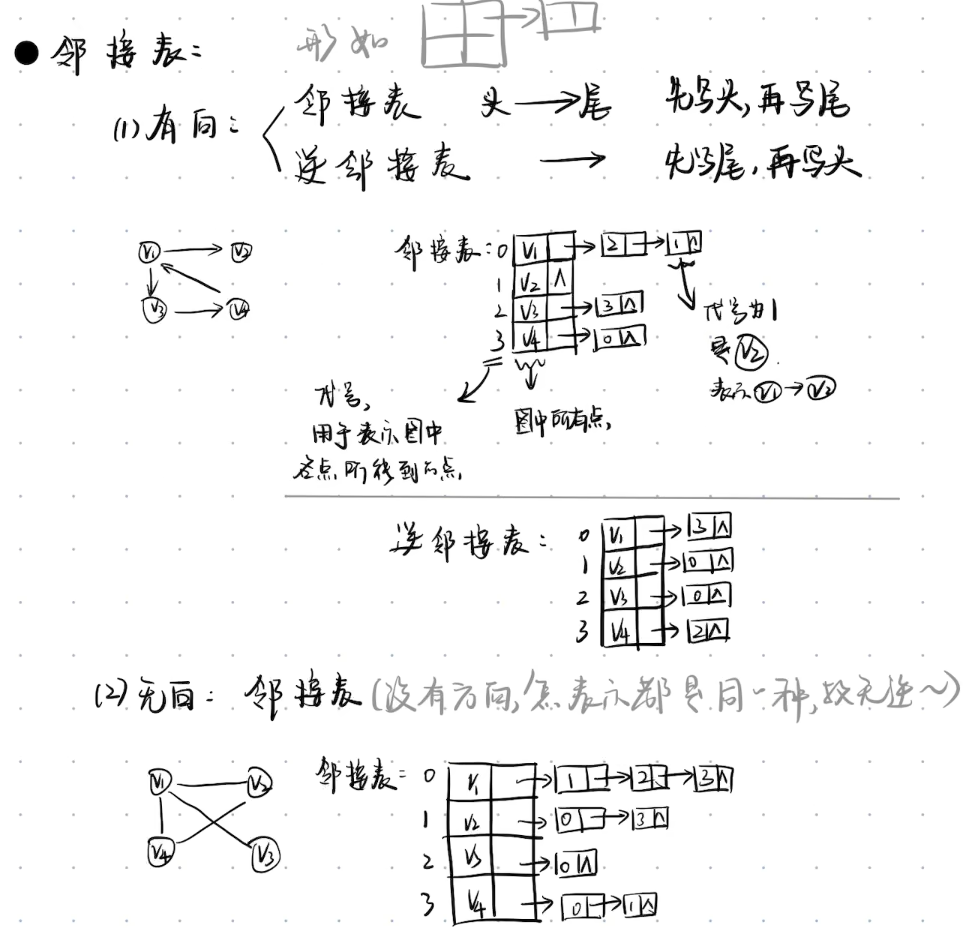
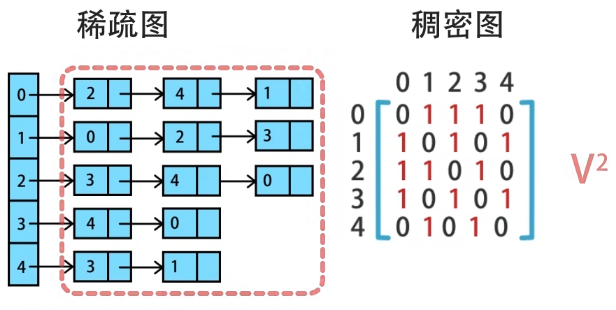
127阶B-树中每个结点最多有（126）个关键字。

若不考虑基数排序，则在排序过程中，主要进行的基本操作是关键字的（ 比较 ）和记录的移动

若用冒泡排序方法对序列{10,14,26,29,41,52}从大到小排序，需进行 （ 15 ）次比较

若a=1，b=2，c=3，d=4，则后缀式db/cc\*a-b\*+的运算结果为（ 18 ）。



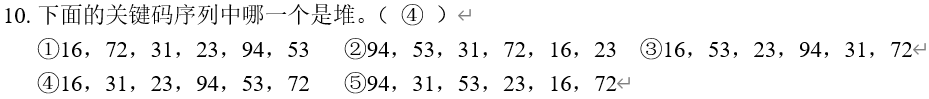




（Q，A，C，S，Q，D，F，X，R，H，M，Y）

设循环队列用数组A[1..M]表示，队首、队尾指针分别是FRONT和TAIL，判定队满的条件为（（TAIL+1）% M=FRONT）

高度为5（除叶子层之外）的三阶B-树至少有（ 31 ）个结点。 = 1 + 2 + 4 + 8 + 16



一棵树中，最大的节点的度称为树的度。

任何一个递归过程都可以转换成非递归过程。

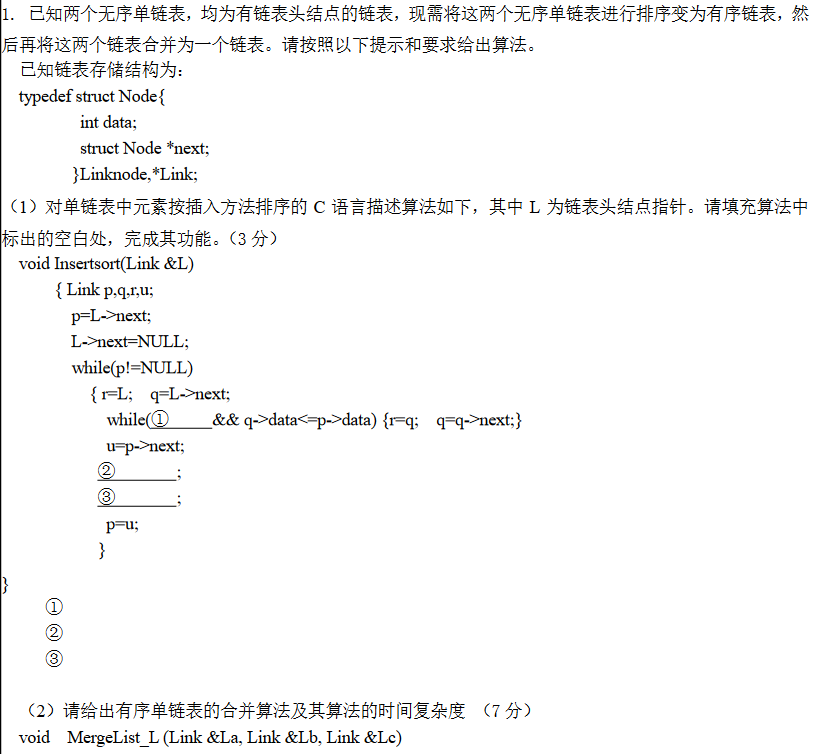
从逻辑结构上看，n维数组的每个元素均属于n个向量。

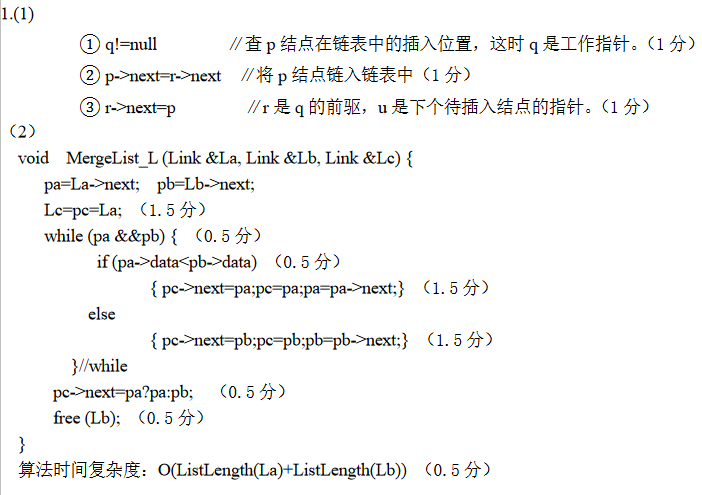
直接选择排序的时间复杂度是固定的O(n^2)

AOV网是一种有向无环图

数据的基本单位为数据元素，不可分割的最小单位为数据项

这题的逻辑小怪，但能做对。





完全二叉树的一个结点若无右孩子，则此结点必为叶子结点。 错

C



**记得关键字63也算比较的（最后多加一次）**

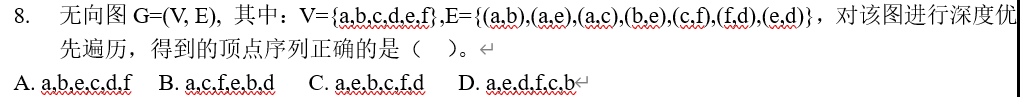
数据的逻辑结构是指数据的各数据项之间的逻辑关系。（ × ）

——逻辑结构就是数据元素间的逻辑关系,而不是数据元素内部的数据项之间的关系

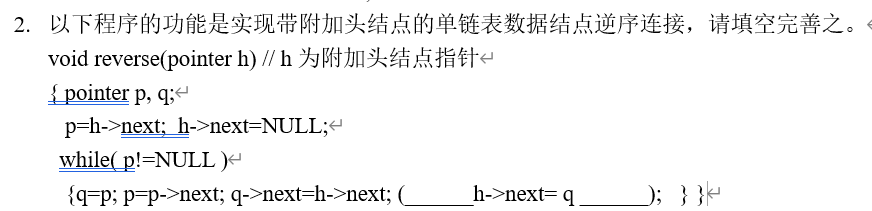
数组可看成线性结构的一种推广，因此与线性表一样，可以对它进行插入、删除等操作。（×）

对一个AOV网，从源点到终点的路径最长的路径称作关键路径。（×）

折半查找法的查找速度一定比顺序查找法快。（ × ）

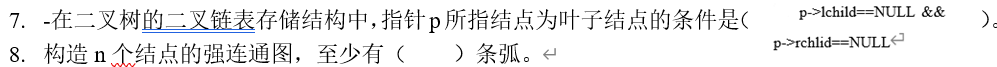


答案：abedfc





串是一种特殊的线性表，其特殊性表现在其数据元素都是字符

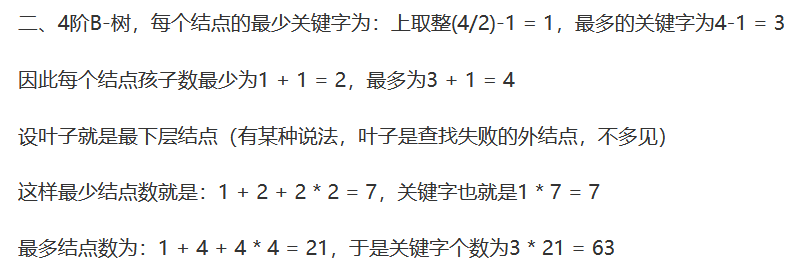


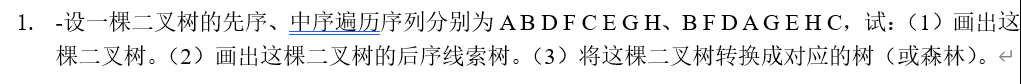
n

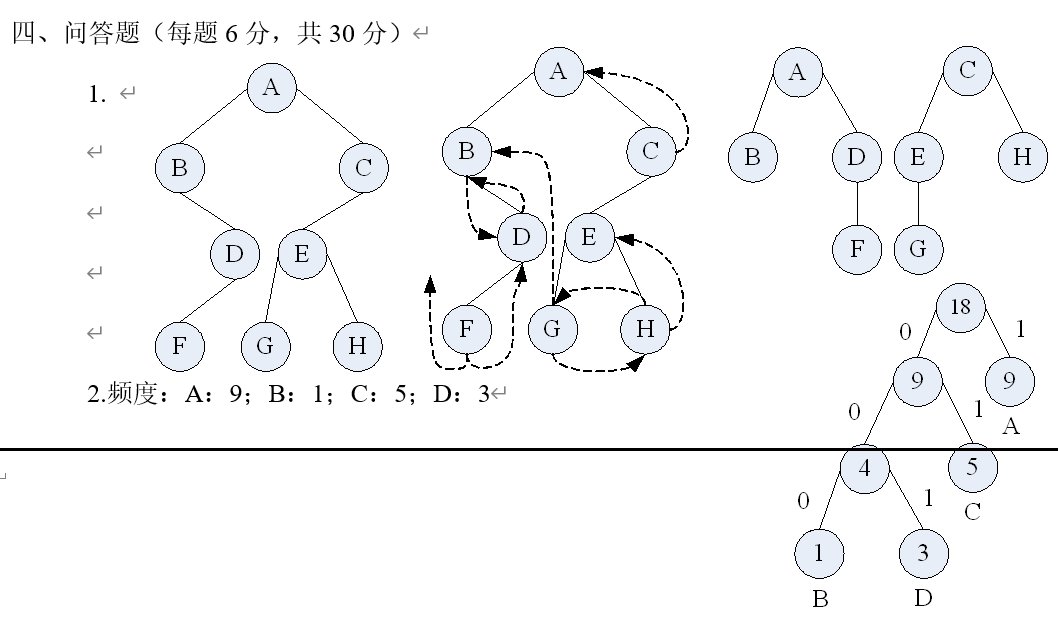
高度为4的3阶B-树中，最多有（ 26 ）个关键字。

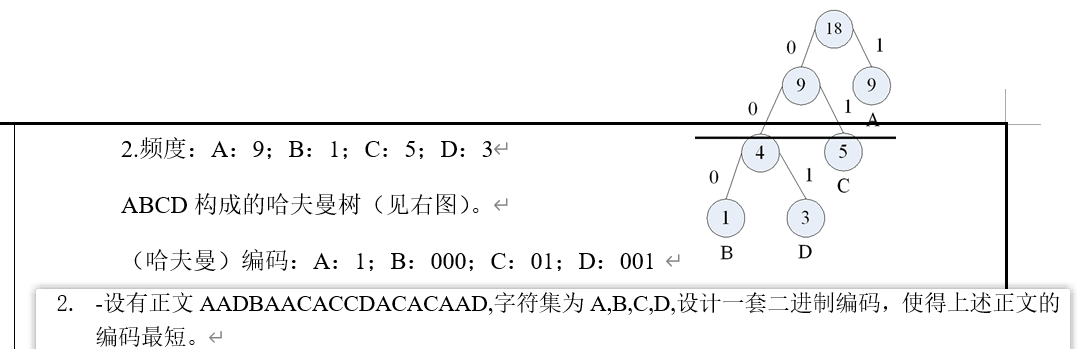
3阶B-树，每个结点最多2个关键字，最多3个孩子；26 = 2 \* （1 + 3 + 9）

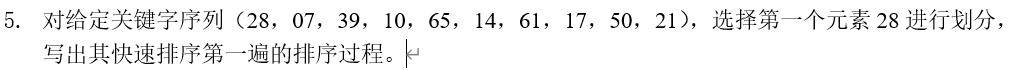
类比：

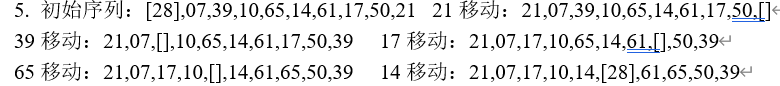




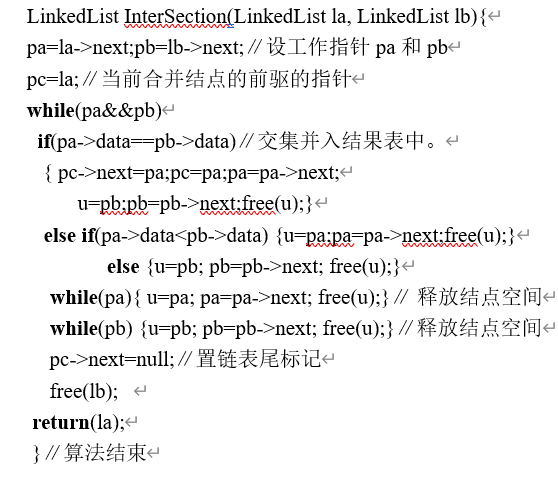


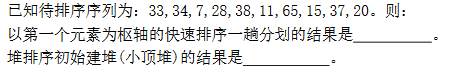






已知两个链表A和B分别表示两个集合，其元素递增排列。编一函数，求A与B的交集，并存放于A链表中。





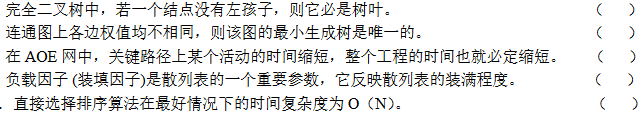




出度；入度



对



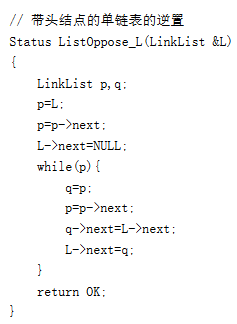
对

对

***错（万一关键路径还是最大值）***

对

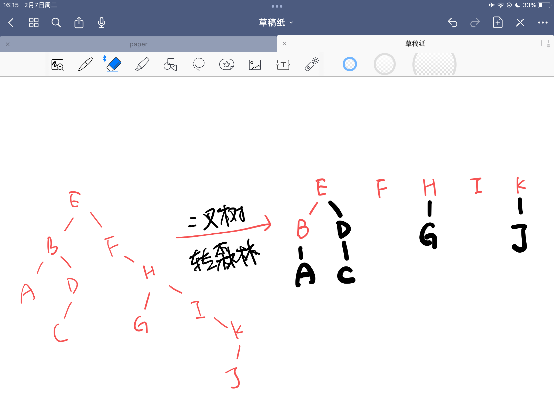
错

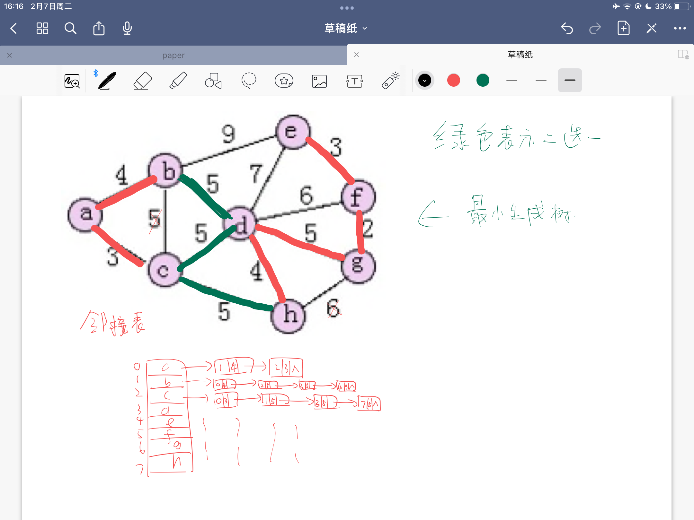




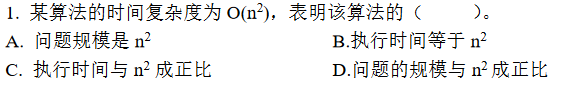




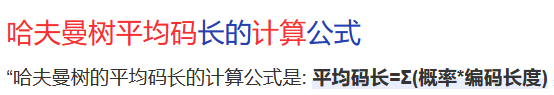




**错，原因：普利姆算法的时间复杂度为O(n2),与网中的边数无关,因此适用于求边稠密的网的最小生成树；克鲁斯卡尔算法时间复杂度为O(eloge)(e为网中的边数),所以适合于求边稀疏的网的最小生成树**

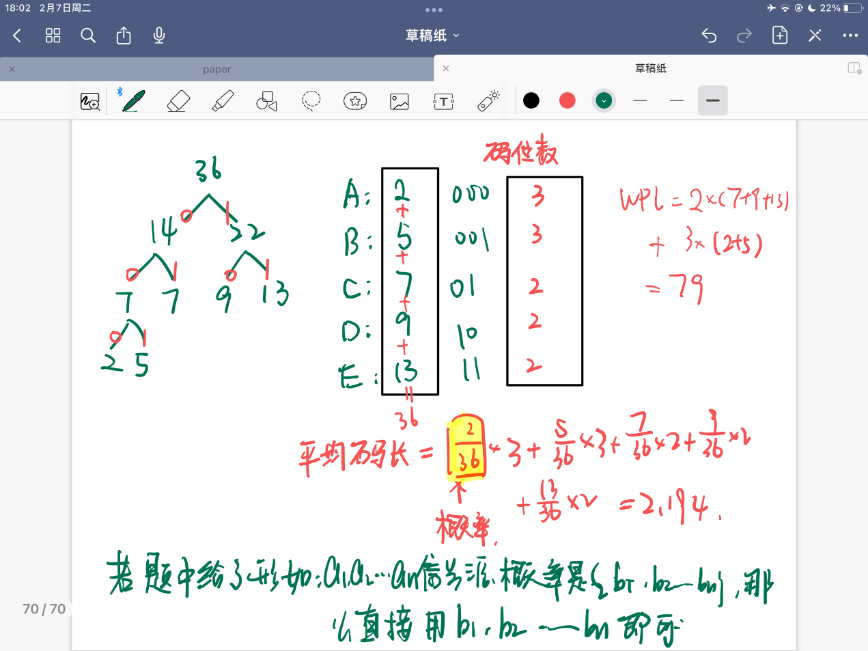


哈夫曼树构造的时候可以自己规定把新加入的元素放在右手边

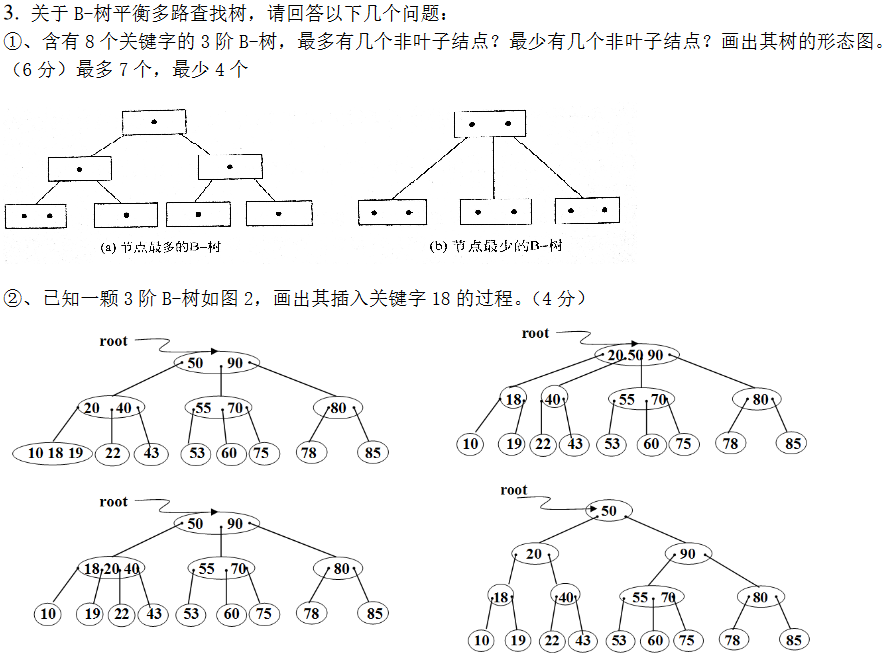


**一定要看：**

**https://blog.csdn.net/weixin\_46838716/article/details/125690610**



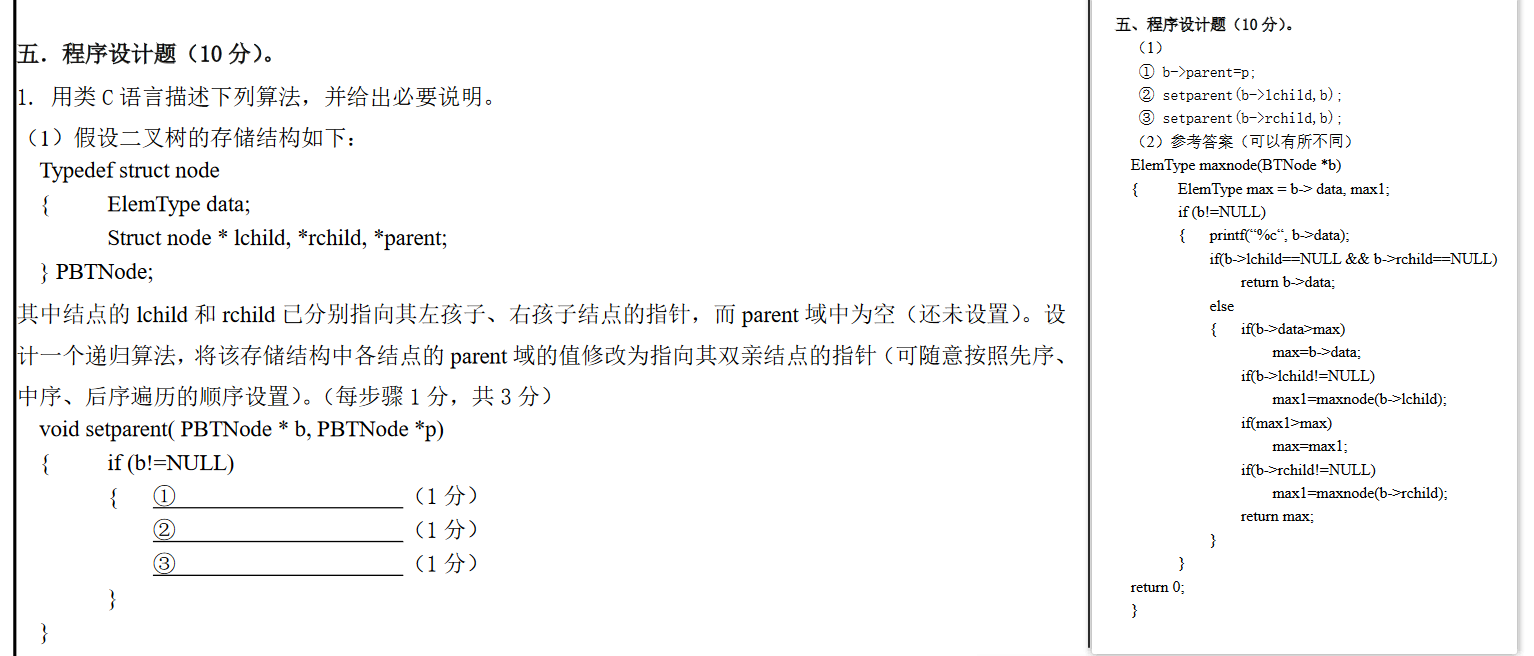




答案：基数排序

第二道题是计算二叉树中结点的最大值：



书P164 7.2.3 上面的话看一下

最小生成树的代价是指树的总权值，将最小生成树的全部边权值加起来就好了。

堆的形状是一棵完全二叉树

假设 B 是一棵树，B′是对应的二叉树。则 B 的后根遍历相当于 B′的中序遍历。 正确

一个无向连通图的生成树是其极小的连通子图。 正确

**书P238**

**对于一棵m阶的B-树，每个除根外非终端结点至少有┌m/2┐ - 1关键字，至多有m棵子树**

对于任何待排序序列来说，快速排序都是最快的。 错误

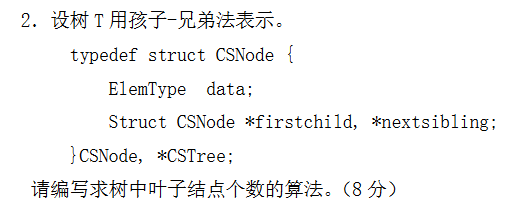
选择类排序是不稳定的。 正确

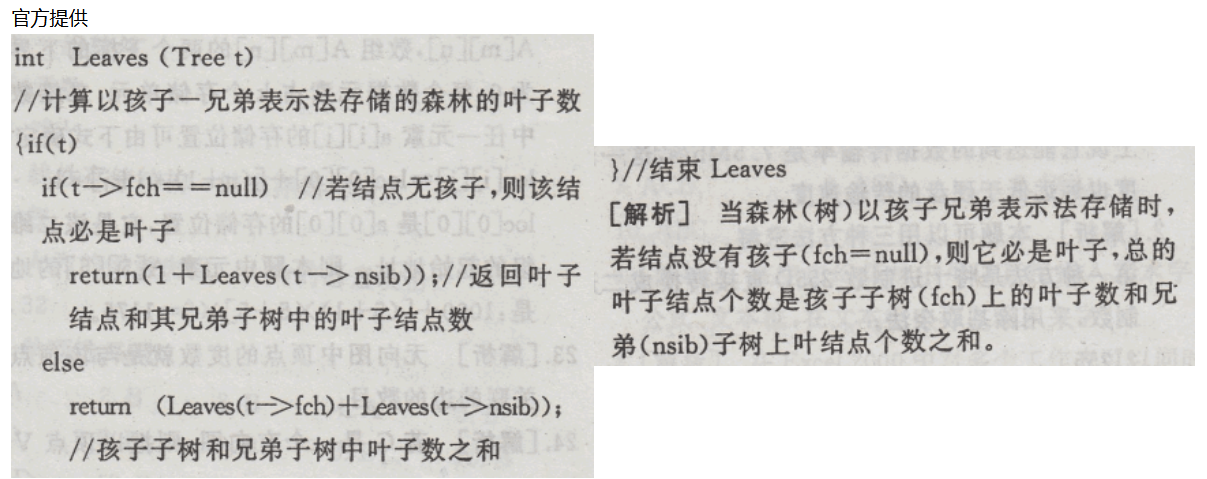
下列算法中（ Dijkstra 算法 ）适用于求图的单源最短路径。

在文件“局部有序”(待排序元素序列基本有序)的情况下,最佳内部排序算法是 直接插入排序

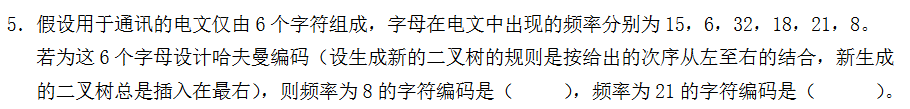
静态查找表与动态查找表的根本区别在于 施加在其上的操作不同。

**基数排序看一下是什么东西（就是按位排序）**





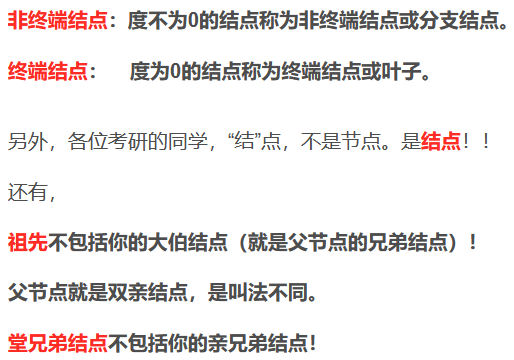
根据要求建立哈夫曼树，Ex：

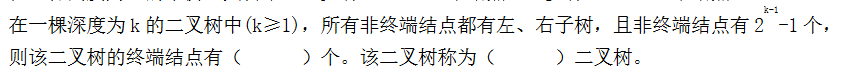


一棵高度为 4 的平衡二叉树中，至少有（ 7 ）个结点，至多有（ 15 ）个结点。

参考：

[**https://blog**](https://blog)**.csdn.net/JMW1407/article/details/108203688?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522167576720516782428624261%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request\_id=167576720516782428624261&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~first\_rank\_ecpm\_v1~rank\_v31\_ecpm-1-108203688-null-null.142^v73^insert\_down1,201^v4^add\_ask,239^v1^control&utm\_term=%E4%B8%80%E6%A3%B5%E9%AB%98%E5%BA%A6%E4%B8%BA%204%20%E7%9A%84%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%A0%91%E4%B8%AD%EF%BC%8C%E8%87%B3%E5%B0%91%E6%9C%89%EF%BC%88%20%EF%BC%89%E4%B8%AA%E7%BB%93%E7%82%B9&spm=1018.2226.3001.4187**





答案：2 \* ┌(2^(k - 1) – 1)/2┐ or 2^（k-1）, 满二叉树

**排序小总结：**

**https://www.cnblogs.com/linfangnan/p/13430897.html**

基数排序使用条件有严格的要求：需要知道各级关键字的主次关系和各级关键字的取值范围。

利用双向链表作线性表的存储结构的优点是：节省空间……（我也不知道为什么）

对算法进行分析的前提是 算法必须正确

数组通常具有的两种基本操作是：查找和修改

数组通常具有的操作是：直接存取

C

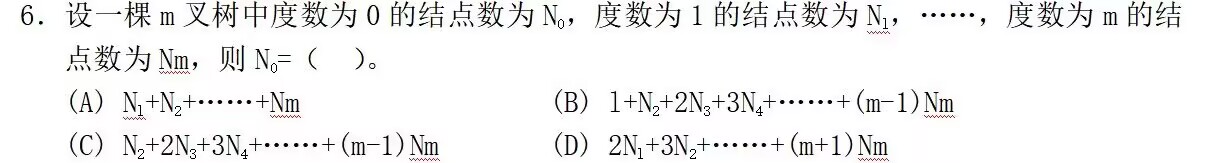
给定一个有n个元素的有序线性表。若采用顺序存储结构，则在等概率前提下，删除其中的一个元素平均需要移动（(n-1)/2）个元素。

设A是一个线性表(a1a2…an),采用顺序存储结构,则在等概率的前提下,平均每插入一个元素需要移动的元素个数为(0+1+2+…+n)/(n+1)=n/2；若元素插在ai与ai+1之间(0<=I<=n-1)的概率为2(n-i)/(n(n+1)),则平均每插入一个元素所要移动的元素个数又是(2n+1)/3

实现二叉树的后序遍历的非递归算法而不使用栈，最佳方案是二叉树的存储结构采用 三叉链表

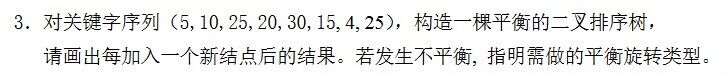
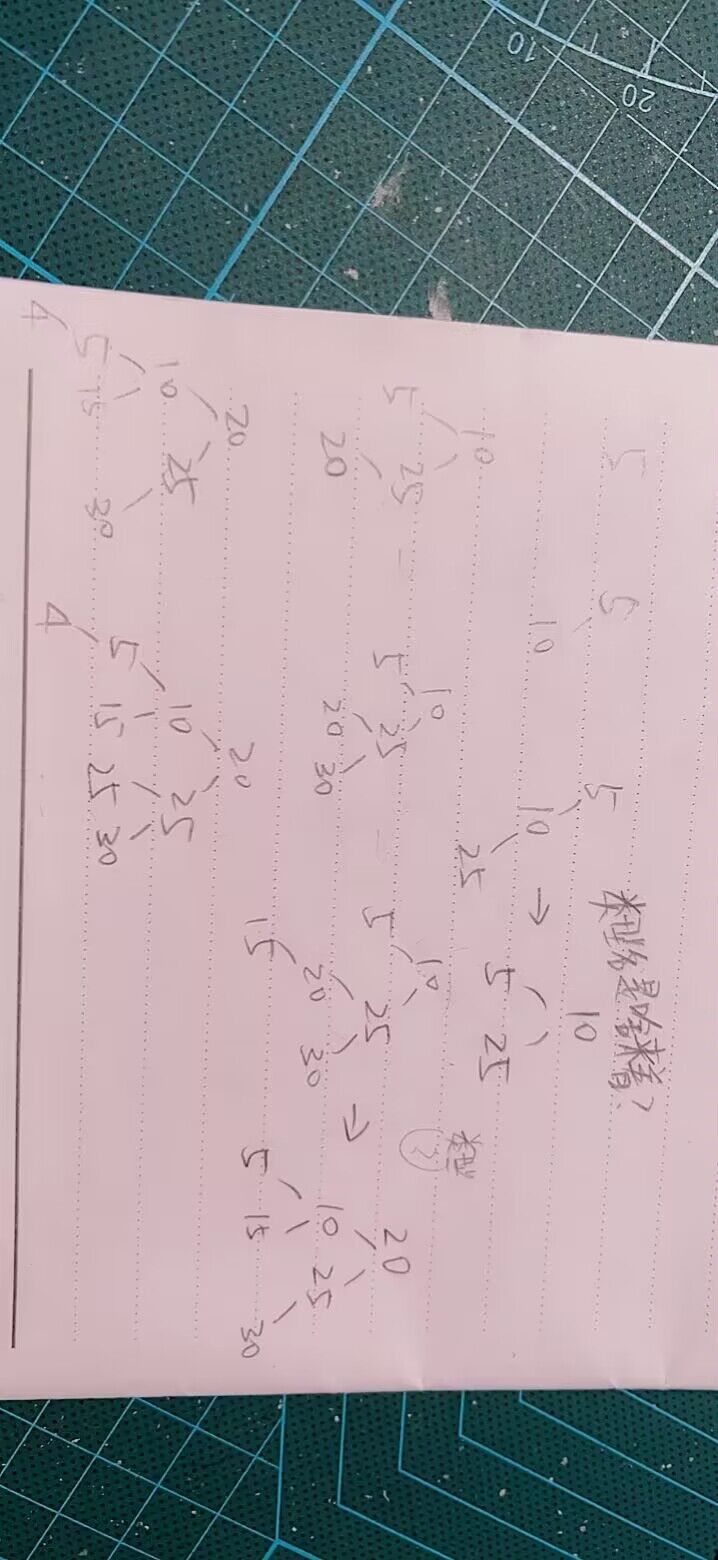
**书p190 7.6.2**

**在一个有n个顶点，e条弧的带权有向网中，求最短路径的Dijkstra算法的时间复杂度为O(n^2)**

B  


https://blog.csdn.net/m0\_37897437/article/details/84666614  



https://www.bilibili.com/video/BV16m4y1F7do/?spm\_id\_from=333.337.search-card.all.click&vd\_source=8ab2a59a663d25da4d86a9e128398659

