9.3.2　非稠密索引分块文件

非稠密索引文件也称为非稠密索引分块文件。它将基本文件中的记录分成若干块，每一块内的记录不必按照关键字值排序，但块与块之间的记录必须保持按照关键字值大小有序，即前一块中所有记录的关键字值都小于后一块中所有记录的关键字值。这样，建立的索引表只须对每一块建立一个索引项，该索引项分别给出相应一块的最大关键字值以及该块的首地址，如图 9.8(b)所示。具有这种特点的索引称为**非稠密索引**。非稠密索引及基本文件合称为**非稠密索引分块文件**。如果文件中的记录是按照关键字值大小排序的，则成为非稠密索引分块文件的一种特例，如图 9.9 所示。

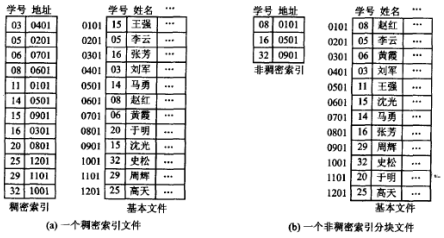

图 9.8　索引文件

对非稠密索引分块文件可以采用分块查找方法，即首先在索引表中确定要查找记录所在的块，然后把该块的所有记录读入内存，之后再在该块中具体查找记录。对索引表可采用顺序查找，也可采用折半查找。如果基本文件也是按关键字值排序的，则在已找到的块中也可采用顺序查找方法或者折半查找方法。若基本文件只是在块与块之间按关键字值有序，则在块中查找具体记录时只能采用顺序查找方法。

假设要查找的记录的关键字值为 k，当在索引表的关键字值集合中查找时如果满足关系 $k \leqslant key_i (i=1,2,\cdots)$，则说明如果要找的记录存在，它应该在第 i 块中，然后再到第 i 块中查找相应记录。

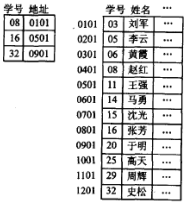

图 9.9　另一个非稠密索引分块文件

在图 9.8(b)所示的非稠密索引分块文件中，基本文件被分成 3 块，因而索引表中包含 3 个索引项。在查找关键字值 k=16 的记录时，首先在索引表中确定 k 所在的块，有关系 $k \leqslant key_1$，说明对应 k 的记录若存在应该在第 1 块中，然后再到基本文件的第 1 块中采用顺序查找方法找到记录(对于图 9.9 所示的文件则可以采用折半查找方法)。

在非稠密索引分块文件中的查找方法也称分块查找方法。其平均查找长度 ASL_{bs} 由查找块地址(即确定所在的块)的平均查找长度 L_b 与在块中查找记录的平均查找长度 L_w 两部分组成，即

$$ASL_{bs}=L_b+L_w$$

若将长度为 n 的基本文件分为 b 块，每块有 s 个记录，则 b=n/s。若每一块的查找概率为 1/b，块内查找任一记录的概率为 1/s，则当采用顺序查找方法时，有

$$ASL_{bs}=L_b+L_w=\frac{1}{b}\sum_{j=1}^{b}j+\frac{1}{s}\sum_{i=1}^{s}i=\frac{b+1}{2}+\frac{s+1}{2}=\frac{1}{2}\left(\frac{n}{s}+s\right)+1$$

当 $s=\sqrt{n}$ 时，将使得 ASL_{bs} 达到最小。若采用折半查找方法确定块地址，则平均查找长度近似为

$$ASL_{bs}=\log_2\left(\frac{n}{s}+1\right)+\frac{s}{2}$$

在非稠密索引分块文件中，不仅可以比较方便地查找单个记录，而且更便于查找一块中的全部记录。当需要对一块中的全部记录依次进行处理时，只要从索引表中找到该块的起始位置，然后再由此位置开始依次取得该块的每一个记录。

*9.3.3　多级索引文件

当索引表本身很庞大时，在索引表中确定记录或者确定记录所在的块需要花费的时间可能也会变得很多，这时可以对索引表进行分块，即建立索引表的索引。如果索引表的索引还很大，则可以继续对索引表的索引进行分块，建立索引表的索引表的索引，这样就形成了一种树形结构的多级索引。

1. 二叉排序树多级索引

此索引结构第 i 层的索引分块数为 2^{i-1}，即二叉树中第 i 层的结点数；每个结点表示一个索引块，该结点的构造如图 9.10 所示。其中，key 为索引树中的最大关键字；llink 为左指针，

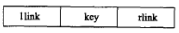

图 9.10　二叉排序树多级索引的结点构造

当sqr(625)时，
分块查找的平均查找长度ASL取最小值sqr(625)+1，
因为进行分块查找时，可将长度为N的表均匀地分成B块，每块含有S个记录，即有B=N/S，
有等概率的情况下，块内查找的概率为1/S，每块的查找概率为1/B，若按顺序查找确定元素所在的块，则分块查找的平均查找长度为：
ASL（BS）=L（B）+L（S）=（B+1）/2+（S+1）/2=1/2（N/S+S）+1
可以求得当S取sqr(N）时，ASL取最小值sqr(N)+1

c语言用字符数组和字符指针删除相同字符

如何用字符数组和字符指针做参数删除与某字符相同的字符
1.用字符数组

```
char *DelLetter1(char str1[],char a)   //用字符数组删除
{
    int i = 0;
    int c = 0;
    char *b = (char*)malloc((strlen(str1))*sizeof(char));   //定义动态数组
    for (i = 0; i < (strlen(str1)+1); i++)
    {
```

```
        if (str1[i]!=a)                    //把不是指定字符的元素复制进入新的数组
        {
            b[c] = str1[i];
            c++;
        }
        else {
            continue;
             }
    }
    return b;                              //返回新的数组
}
```

2.用字符指针

```
char *DelLetter2(char str1[], char a)
{
    char *c = str1;
    char *e = (char*)malloc((strlen(str1)) * sizeof(char));        //定义动态数组
    char *f = e;                                                   //记录新的指针开始值
    while (*c != '\0')
    {
        if (*c != a)                                               //复制进入新的数组
        {
            *e = *c;
            e++;
        }
        c++;
    }
    return f;                  //返回开始地址
```

树的遍历方法，及与其对应二叉树的遍历方法的关系。

树结构有两种次序遍历树的方法：

1.先根遍历：先访问树的根节点，再依次先根遍历子树；

2.后根遍历：先依次后根遍历子树，再访问树的根节点。

不曾看到过树的‘中根遍历’的概念，因为树并不一定是二叉树，‘中’的概念不好定义，比如对于一个拥有3个子树的根节点来说，根节点除了先根和后根两种遍历方式之外还有另外两种次序，如一种次序是先遍历根节点的第一棵子树，再访问根节点，之后再依次遍历剩余子树，另一种次序是，先遍历根节点的前两棵子树，再访问根节点，最后访问第三棵子树。对于拥有更多子树的根节点来说，依次遍历的方法更多。

树的先根遍历和后根遍历可分别借用对应二叉树的先序遍历和中序遍历实现。以上图（a）中的树和其对应的（d）中的二叉树为例：

对树进行先根遍历：A B C D

对树进行后根遍历：B C D A

对二叉树进行先序遍历：A B C D（与树的先根遍历一致）

对二叉树进行中序遍历：B C D A（同树的后根遍历）

接下来说一说森林的遍历方法，及与其对应的二叉树的遍历方法的关系。

森林的两种遍历方法：

1.先序遍历森林：

（1）访问森林中第一棵树的根节点；

（2）先序遍历第一棵树中根节点的子树森林；

（3）先序遍历除去第一棵树之后剩余的树构成的森林。

2.中序遍历森林

（1）中序遍历森林中第一棵树的根节点的子树森林；

（2）访问第一棵树的根节点；

（3）中序遍历除去第一棵树之后剩余的树构成的森林。

在森林的中序遍历方法中需要注意，森林的中序遍历与二叉树的中序遍历方法的定义不同，二叉树的中序遍历是按照LDR的顺序进行遍历，而森林的中序遍历是要先中序遍历第一棵树的所有子树，再访问这棵树的根节点，对于这棵树来说，根节点的访问次序其实是整棵

树遍历的最后（类似于二叉树的后序），这里经常与二叉树的中序遍历混淆，傻傻分不清楚~

下面，看看森林遍历方法和其对应的二叉树遍历方法的对应关系。当森林转换成二叉树时，其第一棵树的子树森林转换成左子树，剩余树的森林转换成右子树，则森林的先序和中序遍历即对应二叉树的先序和中序遍历。以上图中（a）（b）（c）构成的森林和对应的二叉树（g）为例：

对森林进行先序遍历：A B C D E F G H I J

对森林进行中序遍历：B C D A F E H J I G

对二叉树进行先序遍历：A B C D E F G H I J（与森林先序遍历一致）
对二叉树进行中序遍历：B C D A F E H J I G(同森林中序遍历)

在顺序表中插入和删除一个结点平均移动多少个结点

解析
在等概率情况下，顺序表中插入一个结点需要平均移动n/2个结点。删除一个结点需要平均移动（n-1）/2个结点。具体的移动次数取决于顺序表的长度n以及需插入或删除的位置i,i越近n,则所需移动的结点数越少。

用数学归纳法证明:1*n+2*(n-1)+3*(n-2)+...+n*1=1/6*n(n+1)(n+2)?
数学作业帮用户2017-10-19

优质解答
n=1时,左边=1*1=1
右边=1/6*1*2*3=1
左边=右边,等式成立!
假设n=k时成立 (k>1)即:
1*k+2(k-1)+3(k-2)+...+(k-1)*2+k*1=(1/6)k(k+1)(k+2)
当n=k+1时;

左边
=1*(k+1)+2(k+1-1)+3(k+1-2)+...+(k+1-1)*2+(k+1)*1
=1*k+1*1+2(k-1)+2*1+...+k*1+k+(k+1)
=[1*k+2(k-1)+...+(k-1)*2+k*1]+1+2+3+...+k+(k+1)
=(1/6)k(k+1)(k+2)+1+2+3+...+k+(k+1)
=(1/6)k(k+1)(k+2)+1/2*(k+1)*(k+2)
=(1/6)(k+1)(k+2)(k+3)
=(1/6)(k+1)[(k+1)+1][(k+1)+2]
=右边
原式也成立!
综上可知,原式为真!

元素的移动次数与关键字的初始排列次序无关的是：基数排序

元素的比较次数与初始序列无关是：选择排序

算法的时间复杂度与初始序列无关的是：直接选择排序

关键在待排序的元素序列基本有序的前提下，效率最高的排序方法是 。

A．直接插入排序
B．选择排序
C．快速排序
D．归并排序

正确答案
A

答案解析
[解析] 本题考查各种排序方法，直接插入排序是将第i个元素插入到已经排序好的前i-1个元素中；选择排序是通过n-i次关键字的比较，从n-i+1个记录中选出关键字最小的记录，并和第i个记录交换，当i等于n时所有记录都已有序排列；快速排序是通过一趟排序将待排序的记录分割为独立的两部分，其中一部分记录的关键字均比另一部分记录的关键字小，然后再分别对这两部分记录继续进行排序，以达到整个序列有序；归并排序是把一个有n个记录的无序文件看成由n个长度为1的有序子文件组成的文件，然后进行两两归并，得到[n/2]个长

度为2或l的有序文件，再两两归并，如此重复，直至最后形成包含n个记录的有序文件为止。
通过上面的分析，可知，在待排序元素有序的情况下，直接插入排序不再需要进行比较，而其他3种算法还要分别进行比较，所以效率最高为直接插入排序。

如何证明有n个点n-1条边的无向连通图一定是一颗树（即没有回环

因为是无向连通图，所以存在一棵生成树，它是这个无向连通图的子图，含有无向连通图的全部顶点（n个）以及n-1条边。换句话说。一个连通图的生成树就是该连通图擦掉若干条边后的一个子图。
上面是铺垫，下面用反证法证明：
证明：首先可知连通图必含生成树，下面证明只有n个顶点和n-1条边的连通图本身就是一棵树。
由于无向连通图本身只有n个顶点和n-1条边，又一定存在生成树，如果无向图本身不是一棵生成树（假设），那么它就需要擦去若干条边。由于擦去任意一条边后的图一定没有生成树，因为边的个数不足以拥有生成树这个子图（因为生成树至少需要n-1条边），说明该无向连通图没有生成树（本身不是，擦去后又不存在），这与无向连通图必有生成树矛盾，所以可知假设错误，该无向图本身一定是一棵生成树。

无向图变连通至少边数：n-1
有向图变连通图,至少需要边数：n
数学归纳法

所谓连通图一定是无向图,有向的叫做强连通图 连通n个顶点,至少只需要n-1条边就可以了,或者说就是生成树 由于无向图的每条边同时关联两个顶点,因此邻接矩阵中每条边被存储了两次（也就是说是对称矩阵）,因此至少有2(n-1)个非零元素

有n个顶点的强连通图最多有多少条边,最少有多少条边
我来答
分享 举报 浏览 10437 次
3个回答 #热议# 袁隆平获未来科学大奖，他的成就究竟有多高?
最佳答案 oncforever
来自科学教育类芝麻团 2015-08-20
有n个顶点的强连通图最多有n（n-1）条边，最少有n条边。

解释如下：
强连通图（Strongly Connected Graph）是指一个有向图（Directed Graph）中任意两点v1、v2间存在v1到v2的路径（path）及v2到v1的路径的图。

最多的情况：

即n个顶点中两两相连，若不计方向，n个点两两相连有n（n-1）/2条边，而由于强连通图是有向图，故每条边有两个方向，n（n-1）/2×2=n（n-1），故有n个顶点的强连通图最多有n（n-1）条边。

最少的情况：

即n个顶点围成一个圈，且圈上各边方向一致，即均为顺时针或者逆时针，此时有n条边。

某二叉树的前序序列和后序序列正好相反,则该二叉树一定是()的二叉树
空或只有一个结点
高度等于其结点数
任一结点无左孩子
任一结点无右孩子
查看正确选项
添加笔记
求解答(14)
邀请回答
收藏(227)
分享
纠错
10个回答 添加回答
8

Breeze Mao
一棵具有N个结点的二叉树的前序序列和后序序列正好相反，则该二叉树一定满足该二叉树只有左子树或只有右子树，即该二叉树一定是一条链（二叉树的高度为N，高度等于结点数）
发表于 2016-05-06 10:41:35回复(1)

Cookie1997 ： 前序序列和后序序列正好相反，高度等于其结点数；
前序序列和中序序列正好相反，任一结点无右孩子；
中序序列和后序序列正好相反，任一结点无左孩子。
2018-05-11 01:08:56回复赞(4)

回复
5

StrongYoung
虽然，ACD都满足“二叉树的前序序列和后序序列正好相反”
但是题目问的是“一定”。
B其实是ACD的概括。
发表于 2015-09-30 19:47:28回复(0)
5

相信自己!
前序遍历和后序遍历相同，只有二叉树是一条链的情况下，一条链时，二叉树有下列特点：
1）任一结点无左孩子或者任一结点无右孩子
2）结点的个数等于树的高度
C和D表述不全面

共用体与结构体的区别
共用体：
使用union 关键字
共用体内存长度是内部最长的数据类型的长度。
共用体的地址和内部各成员变量的地址都是同一个地址

结构体大小：
结构体内部的成员，大小等于最后一个成员的偏移量+最后一个成员大小+末尾的填充字节数。
结构体的偏移量：某一个成员的实际地址和结构体首地址之间的距离。
结构体字节对齐：每个成员相对于结构体首地址的偏移量都得是当前成员所占内存大小的整数倍，如果不是会在成员前面加填充字节。结构体的大小是内部最宽的成员的整数倍。

共用体

```
#include <stdio.h>
//gcc让不同类型的变量共享内存地址 ,同一时间只有一个成员有效
union data{
    int a;
    char b;
    int c;
};


int main(){
    union data data_1 = {1};//初始化时只填写一个值。(同一时间只有一个成员有效)
    data_1.b = 'c';
    data_1.a = 10;//后赋值的才有效。前面的赋值被覆盖
    //打印地址，发现指向同一个地址
    printf("%p\n%p\n%p\n",&data_1.a,&data_1.a,&data_1.a);
    return 0;
}
```

可以是任意数值，但实际上编译系统会把这个表达式强制转成逻辑值，一般地false都为0；1或任何非零数值都是true，具体要看什么语言的。

判断题

1.if语句中的表达式不限于逻辑表达式，可以是任意的数值类型。【Y 】

2.switch语句可以用if语句完全代替。【Y 】

3.switch语句的case表达式必须是常量表达式。【 Y】

4.if语句，switch语句可以嵌套，而且嵌套的层数没有限制。【Y 】

5.条件表达式可以取代if语句，或者用if语句取代条件表达式。【N 】

6.switch语句的各个case和default的出现次序不影响执行结果。【N 】

7.多个case可以执行相同的程序段。【Y 】

8.内层break语句可以终止嵌套的switch，使最外层的switch结束。【 N】

9.switch语句的case分支可以使用{ }复合语句，多个语句序列。【Y 】

10.switch语句的表达式与case表达式的类型必须一致。【Y 】

11.在switch多分支中，break语句可使流程立即跳出switch语句体。【Y 】

12.if (a=<b) x++;是合法的C语句。【 N】

13.if必须有else与之配对。【N 】

若用STACK[n]表示某堆栈采用顺序存储结构，则下列关于堆栈及堆栈操作的叙述中正确的是（ ）。

堆栈的大小为n；
堆栈为空时n=0;
最多只能进行n次进栈和出栈操作；
n各元素依次进栈后，它们的出栈顺序与进栈顺序相反。

牛客4414733号
1，栈空时top=-1；
2，可以进行无限次进栈和出栈；
3，每个元素入栈后立马出栈，则出栈顺序与入栈顺序相同。
发表于 2017-11-30 21:37:15回复(0)

回复
更多回答
0

雅尔达winniebaby92
A
B 堆栈为空时n=-1
C 可以进行无限次进栈出栈
D 每个元素进栈后又立马出栈，就可以保证元素依次进栈，且进栈顺序与出栈一致

快速排序算法的递归深度
题目：对n个记录的线性表进行快速排序为减少算法的递归深度，以下叙述正确的是（）

A.每次分区后，先处理较短的部分

B.每次分区后，先处理较长的部分

C.与算法每次分区后的处理顺序无关

D.以上三者都不对

答案：A

解析：在快速排序中，需要使用递归来分别处理左子段和右子段，递归的深度可以理解为系统栈保存的深度，先处理短的分段再处理长的分段，可以减少时间复杂度。

如果按长的递归优先的话，那短的递归会一直保存在栈中，直到长的分段处理完成。短的优先的话，长的递归调用没有进行，它是作为一个整体保存在栈中的，所以递归栈中保留的递归数据会少一些。

对n个记录的线性表进行快速排序为减少算法的递归深度,以下叙述正确的是（ A ） [北方交通大学2000]
A.每次分区后,先处理较短的部分
B.每次分区后，先处理较长的部分
C.与算法每次分区后的处理顺序无关
D.以上三者都不对

10、采用递归方式对顺序表进行快速排序，下列关于递归次数的叙述中，正确的是（D ）[2010计算机]
A、递归次数于初始数据的排列次数无关
B、每次划分后，先处理较长的分区可以减少递归次数
C、每次划分后，先处理较短的分区可以减少递归次数
D、递归次数与每次划分后得到的分区处理顺序无关

C语言中变量的地址是什么类型的

变量的地址，在C语言中，一般写作指针类型。
不同类型的变量地址，用不同的指针进行保存。

比如，char 类型的地址，使用char*保存，而int型地址，用int *保存。
除此外，部分情况下也会采用整型类型来保存变量地址，具体使用何种整型类型，取决于编译器：
1 16位编译器，地址占16位，2字节，可以使用short或者int保存。
2 32位编译器，地址占32位，4字节，可以使用int或long保存。
3 64位编译器，地址占64位，8字节，可以使用long保存。
不过不推荐使用整型类型保存地址，会带来移植上的不通用。

为实现快速排序算法，待排序序列宜采用的存储方式是链式存储
链式存储不适用于需要随机读取的算法（链表无法进行随机读取，只能从头读到尾），可以用顺序存储的方式

待排序记录序列是线性结构，可以用顺序存储结构和链式存储结构表示。 对于顺序存储结构进行排序时，是对序列中的记录本身进行物理重排（通过关键字之间的比较判断，将记录移动到合适的位置）。 而对以链表作为存储结构的序列进行排序时，无序移动记录，仅需修改指针。