西安交通大学 2018 年招收攻读硕士学位研究生入学考试试题答案

college of design

主要讲解真题及重点

团队客服扣扣 871729782

西安交通大学 2018 年招收攻读硕士学位研究生入学考试试题答案

科目代码：915　　科目名称：计算机软件基础

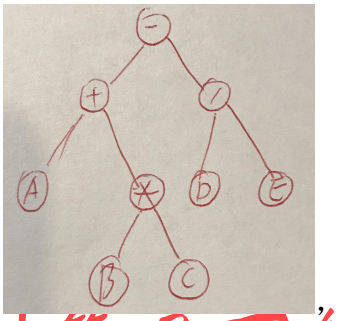
一、选择题

1、B　100，时间复杂度是（n(n-1)/2）　选项电子版打印时候工作同学手误。

2、A 卡特兰数。即一共有：h(n)=c(2n,n)/(n+1) 种合法的出栈顺序。对于出栈序列中的每一个数字，在它后面的、比它小的所有数字，一定是按递减顺序排列的。比如入栈顺序为：1 2 3 4。出栈顺序：4 3 2 1 是合法的，对于数字 4 而言，比它小的后面的数字是：3 2 1，且这个顺序是递减顺序。同样地，对于数字 3 而言，比它小的后面的数字是： 2 1，且这个顺序是递减的。出栈顺序：1 2 3 4 也是合法的，对于数字 1 而言，它后面没有比它更小的数字。同样地，对于数字 2 而言，它后面也没有比它更小的数字。出栈顺序：3 2 4 1 也是合法的，对于数字 3 而言，它后面比 3 小的数字有： 2 1，这个顺序是递减的；对于数字 2 而言，它后面的比它 小的数字只有 1，也算符合递减顺序；对于数字 4 而言，它后面的比它小的数字也只有 1，因此也符合递减顺序。出栈顺序：3 1 4 2 是不合法的，因为对于数字 3 而言，在 3 后面的比 3 小的数字有：1 2，这个顺序是一个递增的顺序(1-->2)。

3、D （少个图）打印 越界，没打印出来，权值最大的，权值就是定义的路径上面的值。可以这样理解为节点间的距离。通常指字符对应的二进制编码出现的概率。至于霍夫曼树中的权值可以理解为：权值大表明出现概率大！一个结点的权值实际上就是这个结点子树在整个树中所占的比例.

4、B



5、D ，2017 年选择题出现过原题

6、A 深度为 k 的完全二叉树的叶子结点都出现在第 k 层或 k－1 层。完全二叉树有两种形式，所以不管哪种形式，必定一定会出现在第 K 层

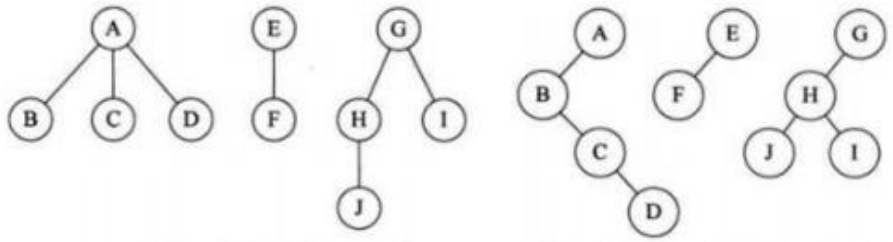
7、A 在第 i 个位置上插入一个元素，总个数变成 n+1，移动次数是 n+1-i

8、D 快速排序法定义了，以 46 为基准数，把比它大的放在右边，小的放在左边。所以第一次得到的结果是 40,38,46,56,79,84

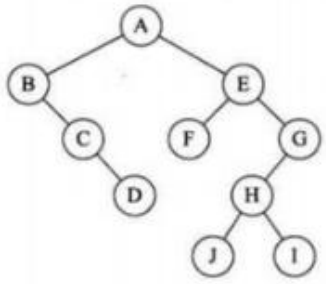
9、B 森林和二叉树的转化，右子树的节点个数。高分笔记树部分有详细讲解。

1. 把每棵树转换为二叉树。

2. 第一棵二叉树不动，从第二棵二叉树开始，依次把后一棵二叉树的根结点作为前一棵二叉树的根结点的右孩子，用线连接起来。



拥有三棵树的森林　　步骤1:森林中每棵树转换为二叉树



步骤2：将所有二叉树转换为一棵二叉树

n1=4, n2=2. n3=4 。看图便可知道结果

10、A 传统的排序算法一般指内排序算法，针对的是数据可以一次全部载入内存中的情况。但是面对海量数据，即数据不可能一次全部载入内存，需要用到外排序的方法。外排序采用分块的方法（分而治之），首先将数据分块，对块内数据按选择一种高效的内排序策略进行排序。然后采用归并排序的思想对于所有的块进行排序，得到所有数据的一个有序序列。

二、判断题

1、 错 2、对 3、错（对线性表进行折半查找时，要求线性表必须以链式方式存储，且结点按关键字有序排列） 4、对 5、对（链表在插入删除时不需要移动元素，只改变指向元素的链就好，一般在删除时只需要把 next 指向 next 的 next 节点，则节点删除。）

三、填空题：

1 不相同，如果输入序列是随机的，处理时间还是可以接受的。如果数组已经有序时，此时的分割就是一个非常不好的分割，因为每次划分只能使待排序序列减一，此时为最坏情况，快速排序沦为起泡排序，时间复杂度为 Θ(n^2)。而且，输入的数据是有序或部分有序的情况是相当常见的。因此，使用第一个元素作为基元是非常糟糕的。
2T(n/2)+n

2 队列 广度优先用队列，深度优先用栈。
广度优先：当一个节点被加入队列时，要标记为已遍历，遍历过程中，对于队列第一个元素，遍历其所有能够能一步达到的节点，如果是标记未遍历的，将其加入队列，从第一个元素出发所有能一步直接达到的节点遍历结束后将这个元素出列。
深度优先：当遍历到某个节点 A 时，如果是标记未遍历，将其入栈，遍历它能够一步直接达到的节点，如果是标记未遍历，将其入栈且标记为已遍历，然后对其进行类似 A 的操作，否则找能够一步直接达到的节点进行类似操作。直到所有能够一步直接达到的节点都已遍历，将 A 出栈。

3 2017 年原题，拓扑的排序的定义。 就是有向图的最长路径问题，如果图中存在环，则最长路径是无法求得的，所以有拓扑序列的有向图不可以存在环

4 n 个节点有 2n 个指针，n 个节点用 n-1 个线就可以链接起来，剩下的不就是 2n-(n-1)=n+1 个空指针

5 O(d(r+n)) 高分笔记上有原题，分配需要 O(n),收集为 O(r),其中 r 为分配后链表的个数，以 r=10 为例，则有 0～9 这样 10 个链表来将原来的序列分类。而 d，也就是位数(如最大的数是 1234，位数是 4，则 d=4)，即"分配-收集"的趟数。因此时间复杂度为 O(d*(n+r))。

四、综合题

1、对于某些问题，一些算法更适合于用小规模的输入，而另一些则相反。幸运的是，在评价算法运行效率时，我们往往可以忽略掉其处理小规模问题时的能力差异，转而关注其在处理大规模数据时的表现。道理是显见的，处理大规模的问题时，效率的些许差异都将对实际执行效率产生巨大的影响。这种着眼长远，更为关注时间复杂度的总体变化趋势和增长速度的策略和方法，即所谓的渐进分析（asymptomatic analysis）。

渐进时间复杂度是指对于一个算法来说，我们常常需要计算其复杂度来决定我们是否选择使用该算法。

对于一个算法，假设其问题的输入大小为 n，那么我们可以用 O(n) 来表示其算法复杂度(time complexity)。那么，渐进时间复杂度（asymptotic time complexity）就是当 n 趋于无穷大的时候，O（n）得到的极限值。

可以理解为：我们通过计算得出一个算法的运行时间 T(n)，与 T(n)同数量级的即幂次最高的 O(F(n))即为这个算法的时间复杂度。例如：某算法的运行时间 T(n) = n+10 与 n 是同阶的（同数量级的），所以称 T(n)=O(n)为该算法的时间复杂度。

算法的渐进分析就是要估计：n 逐步增大时资源开销 T(n)的增长趋势。

2、(1)高度表示结点数：n = 2^h - 1，当 h >= 0
结点数表示高度：h = log2(n + 1)，当 n>= 0
(2)根结点编号：2^(h-1) -1,当 h >= 1 根的左孩子编号：2^(h-2) - 1，根的右孩子编号 3*2^(h-2) - 1，这两个的前提是 h >= 2

4 （1）完全二叉树是由满二叉树而引出来的。 对于深度为 K 的，有 n 个结点的二叉树，当且仅当其每一个结点都与深度为 K 的满二叉树中编号从 1 至 n 的结点一一对应时称之为完全二叉树。一棵二叉树至多只有最下面的两层上的结点的度数可以小于 2，并且最下层上的结点都集中在该层最左边的若干位置上，则此二叉树成为完全二叉树。

（2）假设该完全二叉树的深度为 k,则根据完全二叉树的定义和性质 2 有：

2 ^(k-1)－1< n ≤2^k－1 或 2^(k-1)≤ n <2^k

所以有：k－1≤ log2n<k

又因为 k 是整数,所以,k＝ log2n 向下取整 ＋1

具有 n 个结点的完全二叉树的深度为 log2n(2 是下标)+1。

一个完全二叉树的节点个数为整数 N（不管 N 多么大），其叶子节点的个数均为[（N+1 ）/2]取整。

设叶子节点个数为 n0,度为 1 的节点个数为 n1,度为 2 的节点个数为 n2

侧有

n0+n1+n2=m (1)

对于二叉树有：

n0=n2+1 (2)

由(1)(2) ==>

n0=(m+1-n1)/2 (3)

由完全二叉树的性质可知：n1=0 或 1

总结：

(a):当 n1=0 时(即度为 1 的节点为 0 个时,此时 n 为奇数）或者 n 为奇数时

n0= (m+1)/2;

(b):当 n1=1 时(即度为 1 的节点为 1 个时,此时 n 为偶数）或者 n 为偶数

n0= m/2;

综合(a)(b)可得：

(结论):一个具有 m 个节点的完全二叉树,其叶子节点的个数 n0 为:m/2 向上取整，或者(m+1)/2 向下取整.

1) 在非空二叉树中，第 i 层的结点总数不超过 2i-1, i>=1;

2) 深度为 h 的二叉树最多有 2h-1 个结点(h>=1)，最少有 h 个结点;

3) 对于任意一棵二叉树，如果其叶结点数为 N0，而度数为 2 的结点总数为 N2，则 N0=N2+1;

4) 具有 n 个结点的完全二叉树的深度为 log2(n+1);

5)有 N 个结点的完全二叉树各结点如果用顺序方式存储，则结点之间有如下关系：

若 I 为结点编号则 如果 I>1，则其父结点的编号为 I/2;

如果 2I<=N，则其左儿子（即左子树的根结点）的编号为 2I；若 2I>N，则无左儿子;

如果 2I+1<=N，则其右儿子的结点编号为 2I+1；若 2I+1>N，则无右儿子。

6)给定 N 个节点，能构成 h(N)种不同的二叉树，其中 h(N)为卡特兰数的第 N 项，h(n)=C(2*n, n)/(n+1)。

7)设有 i 个枝点，I 为所有枝点的道路长度总和，J 为叶的道路长度总和 J=I+2i。

综合上述结果，（1）N(max)=2^h -1 N(min)=h

(2)h（max）=n h(min) = [log2^(n+1)]

3、书上原题，去年给内部群学员单独画过这个题，正好考试也考了。

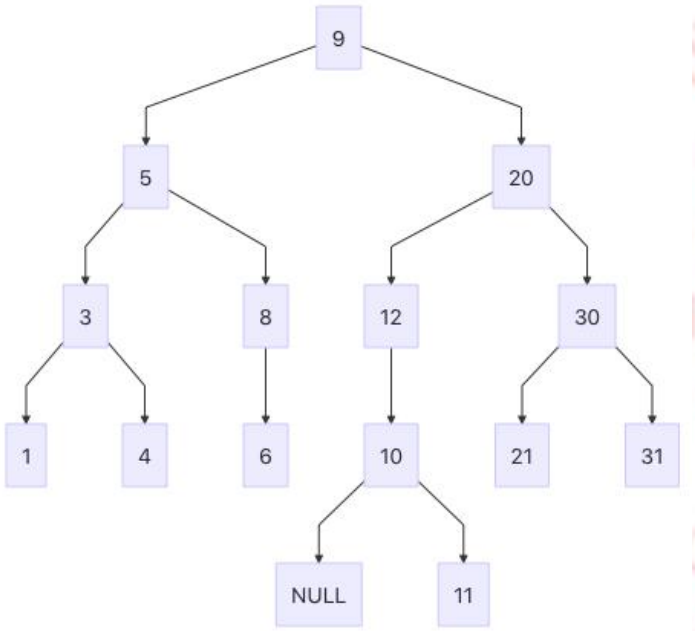
（1）栈满的条件： top[1]+1=top[2] 栈空的条件 top1=-1 top2=m 师 栈 2 为空。

（2）栈 2 的栈顶是 top2,栈 2 入栈操作为 top2 先自减 1，然后存入元素，出栈曹组偶为先取出栈顶的元素，然后 top2 再自增 1。

其中入栈出栈代码必须掌握，相关代码见《高分笔记数据结构》第 72 页。其中王道《数据结构》第 64 页也有原题。

4、二叉线索树的遍历与普通树的遍历不相同，因为二叉线索树有空指针存在，所以遍历时候必须考虑空指针的情况。类似于链表，这是一个知识点，必须记住和会。

(1)根据线索二叉树画出的二叉树如下图所示。



（2）中序遍历：

null, 1, null, 3, null, 4, null, 5, null, 6, null, 8, null, 9, null, 11, null, 12, null, 20, null, 21, null, 30, null, 31, null

后序遍历：

null, null, 1, null, null, 4, 3, null, null, 6, null, 8, 5, null, null, 11, null, 12, null, null, 21, null, null, 31, 30, 20, 9

（3）散列表的优点很明显，查询时间为常数 1，最快的查询速度。二叉树的查询速度也很快，为 log(n)但是慢于散列表。

但是二叉树相对于散列表的优点是，对于一个二叉查找树，即 binary search tree，其中元素是排序的，而散列表是不排序的。那么问题就来了，在你手机中，显然你希望联系人是按姓氏排序的，那么如果你使用散列表，你就需要额外的内存空间进行排序，而二叉查找树本身就是排好序的，因此节省了宝贵的空间。

结论就是，在空间不受限制时，且不需要高频率的排序操作时，二叉查找树不如散列表。反之二叉查找树优于散列表。

另外给大家一个博客，讨论二叉树（左子树小于根节点）和哈希表的对比。https://www.cnblogs.com/bjwu/p/9823531.html

5、（1）本题是给定图并给定图的权值，在给定图中求最小生成树的权值范围。因为图中已经给定了最小支撑边，所以最小生成树的权值必须满足图中所有的最小支撑边，给定三个 xyz 的边，如果取值不当的话，就无法满足给定的最小生成树所有的支撑边，以克鲁思卡尔的算法为例。

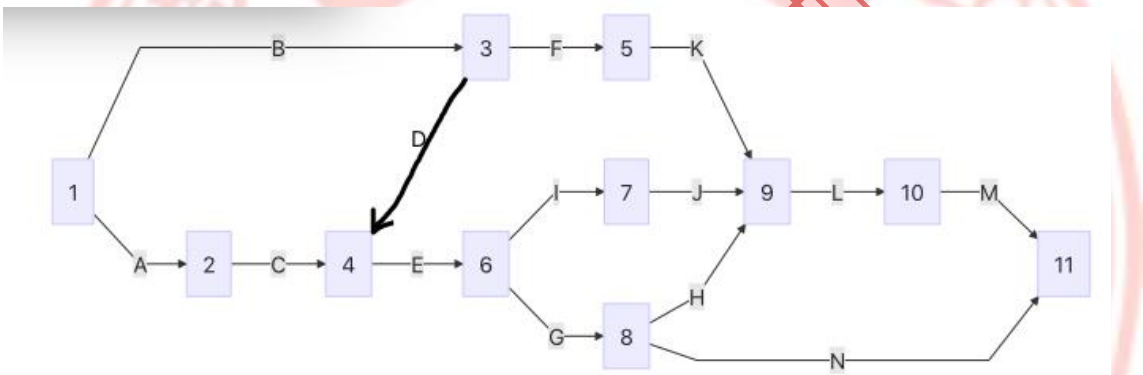
1<=y<=6　　x>=9　　z>=12　，使用其他算法比如普利姆算法生产结果可能不一样。

（2）给定所有权值，按照题目要求选邻接点。

给出的从小到大的顶点顺序是：C D A B E H G F

给出的从大到小的顶点顺序是：C E F D G H A B

6、本题是资料第 47 页一个题目的原题，几乎一样，只是工序的时间不一样，比较难，去年很多人没做出来，因为我们资料 45 页正好没答案，所有当时很多人遇到原题后发现没发作答。这个题在 1800 题的资料书上也有原题，答案写的比较抽象。　另外题目中 F，G 改成 F，I ，HGK 改成 HJK。



(2)给出各个工序最早和最迟发生时间，最好以表格形式列出来，比较清楚。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
最早发生	0	0	4	5	11	5	15	21	15	18	10	28	30	21
最迟发送	3	0	7	5	11	21	15	21	21	24	26	28	30	31

（3）关键路径：BDEGHLM　关键路径使用时间：5+6+4+6+7+2+3=33

7、散列表是数据结构中重点知识点。其中再散列和公共溢出法书上几乎没怎么讲，需要大家自己找相关博客来做。

（1）第一问，普通的散列函数，进行散列直接做。

	0	1	2	3	4	5	6
关键字		22	15	24	25	3	12
查找次数		1	2	1	1	3	2

散列函数见上表。其中平均成功查找长度是：1+2+1+1+3+2 / 6=5/3

(2) 在 hash 法是在原有的哈希函数冲突的情况下，使用新的散列函数进行散列，如果新的散列函数继续冲突，则正常利用线性探查法解决冲突。其中 h(x)=x/7 +1 求的是增量 D，新的地址是：[h(key)+nd]%d 。

相关知识点见：大家请到相关博客自己查看，学习一下二次探查法再散列

（1） https://blog.csdn.net/jnu_simba/article/details/9664053

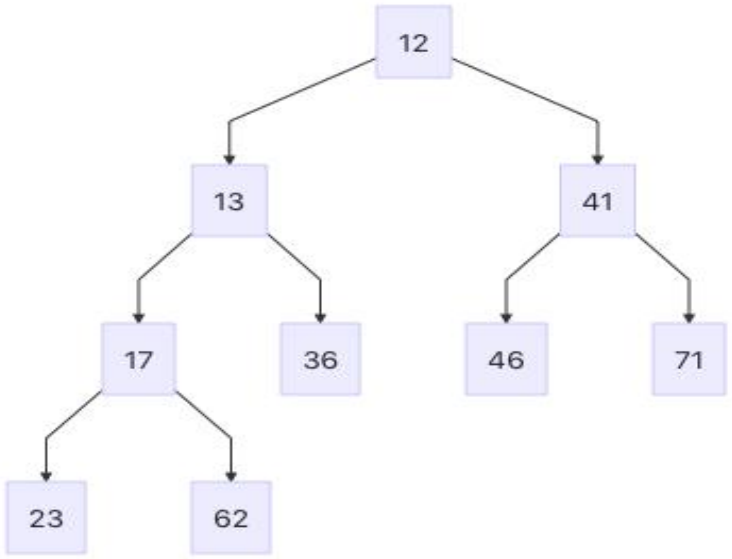
（2） https://blog.csdn.net/jnu_simba/article/details/9668369

	0	1	2	3	4	5	6
关键字		15	22	12	24	25	3
查找次数		3	1	2	1	1	3

其中平均查找长度是：3+1+2+1+1+3/6 =11/6

8、本题是堆、二叉树、排序所有知识点结合一个题，出题水平比较高。结合了好几个知识点，大家知识点需要融会贯通。堆是一颗完全二叉树。满足：任何一个非叶子结点的额值都不大于或者不小于他左右孩子结点的值，其中小顶堆是父亲小孩子大。堆排序思想，怎么建立初试堆等相关知识点。

（1）



（2） 希尔排序结果很简单：17 13 12 41 23 46 62 36 71

（3） 1 次。直接看直接插入排序代码，并手动执行判断。

五、编程题

编程题遵守三个原则：不管题目要求是否，都最好写算法思想，及解决问题及分析问题的思路。 写出一个算法的主函数体就行，不必要写其他无用的。最后最好写出本代码的时间复杂度和优点。其中算法思想必须写。不然会扣分。因为老师没太多精力详细看代码。实在写不出来代码可以用少量的伪代码描述，但会扣分很多。

1、这个题是一个 17 年的一个类似考题。稍微变动一下，就能统计结果。

```
#include<stdio.h>
void fun(char*str,int*times){
  int i;
  for(i=0;str[i];i++){
    if(str[i]-'0'>=0&&str[i]-'0'<=9)
      times[str[i]-'0']++;
  }
}
int main(){
  char str[100]={0};
  int i,j,times[10]={0};
  scanf("%s",str);
  fun(str,times);
  for(i=0;i<10;i++)
    printf("%d%s%4d",i,":",times[i]);
  return 0;
}
```

2、(题库的题) 二维数组、结构体等都能实现

```
/* 成绩排序 */
#include "stdio.h"
#include "conio.h"
typedef struct {
    int   score;
    char name[20];
}datatype;
void process( datatype* );
int main( void )
{
    int        i          = 0;
    datatype data[10] = { 0 };
    printf( "以空格隔开姓名和成绩 以回车结尾 按任意键开始输入" );
    getch();
    printf( "\n" );
```

```
    for ( i = 0; i < 10; i++ )
    {
        printf( "姓名  成绩", i + 1 );
        fflush( stdin );
        scanf( "%s %d", &data[i].name, &data[i].score );
        printf( "\n" );
    }
    process( data );
    printf( "\n 当前排序为：\n" );
    for ( i = 0; i < 10; i++ )
    {
        printf( "%s %d\n", data[i].name, data[i].score );
    }
    return(0);
}
void process( datatype* data )
{
    int   i     = 0;
    int   j     = 0;
    int   temp      = 0;
    for ( i = 10 - 1; i > 0; --i )
    {
        for ( j = 0; j < i; ++j )
        {
            if ( data[j + 1].score > data[j].score )
            {
                temp                = data[j].score;
                data[j].score       = data[j + 1].score;
                data[j + 1].score   = temp;
            }
        }
    }
}
```


2、题库题的改变题几乎一样。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int isLeapYear(int year);
int sumupDays;
int year,month,day,a=0,i;//a 值初始化为 0
int b[12]={31,28,31,30,31,30,31,31,30,31,30,31};
scanf("%d",&year);
scanf("%d",&month);
```

```
scanf("%d",&day);
for (i=1;i<month;i++)
a=a+b[i-1];//
sumupDays=(month>2?isLeapYear(year):0)+day+a;//
printf("%d",sumupDays);
}

int isLeapYear( int year )
{
    int leap;
    if（ (year % 4 = 0) && (year % 100 != 0)          /* 能被 4 整除而不能被 100 整除 */
    leap = 1;
    else if ( year % 400 = 0 )                        /* 能被 400 整除 */
        leap = 1;
    else leap = 0;
    return(leap);
}
```


3、数据结构中的动态规划问题。严蔚敏的书上写过。一般面试题笔试题就是这种题。类似于背包问题。给定 n 种物品和一背包，物品 i 的重量是 wi，价值为 vi，背包的容量为 C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大？在选择装入背包的物品时，对每种物品 i 只有两种选择，即装入背包或不装入背包，不能将物品 i 装入背包多次，也不能只装入部分的物品 i。

这是拔高题，如果参加过 ACM 训练或者程序训练的优秀者，一般会了解动态规划这种问题。这个题在各种公司的校招笔试题中也是常见题，经过多次训练一般都会写个差不多，今年一定会出这种拔高题，提前有思想准备。

贪心算法（多机调度）问题要求给出一种作业调度方案，使所给的 n 个作业在尽可能短的时间内由 m 台机器加工处理完成。 这个问题是 NP 安全问题，到目前为止还没有有效的解法。对于这一类问题，用贪心选择策略有时可以设计出较好的近似算法。

将 n 个作业按时间长到时间短排序，把时间最长的作业放入机器，再依次把次长的放入机器，直到机器都在工作，再取作业放入作业时间最短的机器上，最后工作时间持续最长的机器即为最短的处理时间

伪代码为：

```
#include <stdio.h>
#include <malloc.h>

int f( int n, int m )
{
    if ( m == 1 || n == 0 )
        return(1);
    if ( m > n )
        return(f( n, n ) );
    return(f( n, m - 1 ) + f( n - m, m ) );
}
```

```
int main()
{
    int   a, b, j, result;                          /* a,b 分别是题目中的 n,m;j 是下
边 for 循环的循环标志，result 是最后的结果 */
    int   a1   = 0;
    int   b1   = 0;                                 /* 通过 a1,b1 是否与上一组 a,b 值
相等来控制输入的结束（这个只能通过测试用例分析出来，差点坑死） */
    int   * store;                                  /* 用来暂存所有的输入 */
    int   i = 0;                                    /* 标记总共有多少个数据，2
个一组 */
    store = (int *) malloc( sizeof(int) );
    scanf( "%d %d", &a, &b );
    *(store + i) = a;
    i++;
    *(store + i) = b;
    i++;
    while ( (a == a1) ? ( (b == b1) ? 0 : 1) : 1 )   /* a!=a1 && b!=b1 判断是错的 */
    {
        a1   = a;
        b1   = b;
        scanf( "%d %d", &a, &b );
        *(store + i) = a;
        i++;
        *(store + i) = b;
        i++;
    }
    for ( j = 0; j < i - 2; )
    {
        a = *(store + j);
        j++;
        b = *(store + j);
        j++;
        result = f( a, b );
        printf( "%d\n", result );
    }

    return(0);
}
```

将数组 t[n]由大到小排序，对应的作业序号存储在数组 p[n]中；

2．将数组 d[m]初始化为 0；

3．for (i=1; i<=m; i++)

3.1 S[i]={p[i]};　//将 m 个作业分配给 m 个机器

3.2 d[i]=t[i];

4．for (i=m+1; i<=n; i++)

软件科创团队版权所有，侵权必究。团队唯一合法扣扣：871729782

4.1 j=数组 d[m]中最小值对应的下标； //j 为最先空闲的机器序号
4.2 S[j]=S[j]+{p[i]}; //将作业 i 分配给最先空闲的机器 j
4.3 d[j]=d[j]+t[i]; //机器 j 将在 d[j]后空闲

```
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
struct Data
{
   int data;
  int index;
};
Data t[100];
Data d[100];
bool cmp(Data a,Data b){  //实现结构体数组按照 data 项进行从大到小排序
 return a.data > b.data ;
}
int   sortmin(Data d[], int n){
 int i,indexs=0;
 for(i=0; i<n; i++){
 if(d[i].data < d[indexs].data )
 indexs=i;
 }
 return indexs;
}
void duoji(Data t[],   int n, int m){
 int i,j;
 int S[100][100], p[100];
 int h[100];
 Data d[100];
 memset(h,0,sizeof(h));

 sort(t,t+n,cmp);
 cout<<endl<<"排序后的作业序号为：";
 for(i=0; i<n; i++){
 p[i]=t[i].index;  //1、将 t[]排序后，按照从大到小将作业号存储在 p[]数组中
   cout<<p[i]+1<<" ";
 }
 cout<<endl;
 for(i=0; i<m; i++){  //2、初始化 d[]数组，d[]数组记录的是 m 台机器的空闲时间
 d[i].data=0;
 d[i].index=i;
 }
```

```
if(n<m){    //A、当作业数 n <  机器数 m 的情况
for(i=0; i<n; i++){  //记录第一次的分配
S[i][0]=p[i] ;
d[i].data = t[i].data ;
cout<<"第"<<i<<"台机器处理的作业号为："<<S[i][0]<<";  "
<<"所需时间分别为: "<<d[i].data<<endl;
}
}
else{  //B、当作业数 n >  机器数 m 的情况
for(i=0; i<m; i++){  //记录第一次的分配
S[i][0]=p[i] ;
d[i].data = t[i].data ;
cout<<"第"<<i+1<<"台机器处理的作业号为："<<S[i][0]<<";  "
<<"所需时间为: "<<d[i].data<<endl;
//cout<<S[i][0]<<": "<<d[i].data<<endl;
}
for(i=m; i<n; i++){
int j=sortmin(d,m);
cout<<"下一次为机器  "<<j+1<<"  来处理作业";  //j 验证第一次是正确的
S[j][++h[j]]=p[i];
cout<<"   ,并且此次机器"<<j+1<<"处理的作业为：  "<<S[j][h[j]]+1<<" ";
d[j].data += t[i].data;
cout<<endl;
}
}
}
int main()
{
   int i,n,m;
   cout<<"输入作业个数 n 和机器个数 m: ";
   cin>>n>>m;
   cout<<"输入完成各作业所需要的处理时间："<<endl;
   for(i=0; i<n; i++){
   cin>>t[i].data;
   t[i].index=i;
   }
   cout<<endl<<"各机器所处理的作业情况为："<<endl;
   cout<<endl;
   duoji( t, n,   m);
   return 0;
}
```

输出结果为：

软件科创团队版权所有，侵权必究。团队唯一合法扣扣：871729782

输入作业个数 n 和机器个数 m: 7 3
输入完成各作业所需要的处理时间：
2 14 4 16 6 5 3
各机器所处理的作业情况为：
排序后的作业序号为：4 2 5 6 3 7 1
第 1 台机器处理的作业号为：3; 所需时间为: 16
第 2 台机器处理的作业号为：1; 所需时间为: 14
第 3 台机器处理的作业号为：4; 所需时间为: 6
下一次为机器 3 来处理作业 ,并且此次机器 3 处理的作业为：6
下一次为机器 3 来处理作业 ,并且此次机器 3 处理的作业为：3
下一次为机器 2 来处理作业 ,并且此次机器 2 处理的作业为：7
下一次为机器 3 来处理作业 ,并且此次机器 3 处理的作业为：1

软件科创团队版权所有，侵权必究。团队唯一合法扣扣：871729782