MATLAB程序设计方法及若干程序实例

樊双喜

(河南大学数学与信息科学学院　开封　475004)

摘 要　本文通过对 MATLAB程序设计中的若干典型问题做简要的分析和总结 , 并在此基础上着重讨论了有关算法设计、 程序的调试与测试、 算法与程序的优化以及循环控制等方面的问题. 还通过对一些程序实例做具体解析 , 来方便读者进行编程训练并掌握一些有关 MATLAB程序设计方面的基本概念、基本方法以及某些问题的处理技巧等. 此外, 在文章的最后还给出了几个常用数学方法的算法程序 , 供读者参考使用. 希望能对初学者进行 MATLAB编程训练提供一些可供参考的材料, 并起到一定的指导和激励作用 , 进而为 MATLAB编程入门打下好的基础.

关键字　算法设计；程序调试与测试；程序优化；循环控制

1　算法与程序

1.1　算法与程序的关系

算法被称为程序的灵魂 , 因此在介绍程序之前应先了解什么是算法 . 所谓算法就是对特定问题求解步骤的一种描述 . 对于一个较复杂的计算或是数据处理的问题, 通常是先设计出在理论上可行的算法 , 即程序的操作步骤, 然后再按照算法逐步翻译成相应的程序语言 , 即计算机可识别的语言 .

所谓程序设计, 就是使用在计算机上可执行的程序代码来有效的描述用于解决特定问题算法的过程 . 简单来说, 程序就是指令的集合 . 结构化程序设计由于采用了模块分化与功能分解, 自顶向下, 即分而治之的方法, 因而可将一个较复杂的问题分解为若干子问题 , 逐步求精. 算法是操作的过程, 而程序结构和程序流程则是算法的具体体现 .

1.2　MATLAB语言的特点

MATLAB语言简洁紧凑, 使用方便灵活, 库函数极其丰富 , 其语法规则与科技人员的思维和书写习惯相近 , 便于操作.MATLAB　程序书写形式自由, 利用其丰富

的库函数避开繁杂的子程序编程任务 , 压缩了很多不必要的编程工作 . 另外, 它的语法限制不严格, 程序设计自由度大. 其最大的特点是以矩阵运算为最强 , 而数值的矩阵化又为运算和处理提供了方便 . 除此之外,MATLAB 还有着非常强大的绘图功能.

1.3 MATLAB 程序设计练习

MATLAB 有着丰富的库函数 ,一般情况下应了解并学会使用一些常用的库函数, 至少应熟悉函数库中都有哪些常用函数 , 当需要时可以现学现用 . 或者能对一些经典函数做一定的改造 , 以达到解决某一特定问题的目的 . 但, 在大多情况下还需要自己编写程序去处理形形色色的问题 . 下面就先从一些较简单的程序入手来熟悉 MATLAB 的编程方式.

例 1 一个分类统计函数的设计（分类统计 _1）

编写一个函数, 统计出一组有序（按升序或降序排列）数字中每种数字的个数, 并返回数字种类数.

分析：设待统计数组为 x, 因为 x 有序, 所以在设计算法时应抓住这个特点 .

若用 s1 记录已统计出的数字 , 则, 在对 x 中的数字进行遍历时 , 每次只需让 x(i) 与 s1 中的最后一个数字进行比较就可以了 , 若相等, 则对应计数器加 1, 若不等 , 则说明测到新数, 应开辟新的存储单元 . 其算法程序如下：

```
function [s,k]=FLTJ_1(x)
%x为待统计的一组有序数 , 返回值 s 为 2 列的数组, 第一列为不同种类的数字
%第二列为对应数字的个数 , k 记录统计出的数字种类数目
n=length(x);
s1=x(1);          % s1 记录测到的新数字, 给其赋初值为 x 的第一个数字
s2=1;             % s2 记录 s1 中每个数字的个数, 赋初值为 x(1) 的初始个数 1
k=1;              % k 记录已统计出的数字种类数 , 初值赋为 1
for i=2:n         %从第 2 项开始遍历数组 x
    if x(i)==s1(k)          % 如果 x(i) 与已测出的最后 1 个数字相同 ,
    s2(k)=s2(k)+1;          % 则对应的计数器加 1
    else                    %否则, 则说明测到新数字
```

```
        k=k+1;                  % k  值加 1
        s1=[s1;x(i)];           %将此新数并入  s1,
            s2=[s2;1];          %对应的计数器为  1
    end
end
s=[s1,s2];                      %将 s1 与 s2 拼接成一个两列的数组  s
```

程序运行如下（“ ？ ”代表回车, 下同.）

```
>> x=[1,2,2,3,3,4,5,5];          ?
>> [s,k]=FLTJ_1(x)       ?
s =
     1     1
     2     2
     3     2
     4     1
     5     2
k =
     5
```

例 2 一个数字游戏的设计

有这样一个数字游戏：在一个 20×10 的矩阵中,0~99 这 100 个数顺序排列在奇数列中（每 20 个数组成一列）, 另有 100 个图案排列在偶数列中, 这样每个数字右边就对应一个图案. 你任意想一个两位数 a, 再让 a 减去它的个位数字与十位数字之和得到一个数 b, 然后, 在上述矩阵的奇数列中找到 b, 将 b 右边的图案记在心里, 最后点击指定的按钮, 你心里的那个图案将被显示.

下面我们就来编写程序模拟一下这个小游戏 , 以[0,1] 之间的小数代替矩阵中的图案, 由 MATLAB 程序实现如下:

程序 I

```
%“ 测心术 ”游戏
format  short
a=1;t=0;
```

```
while    a
        a1=rand(100,1);
    k=3;s=[];
    while       k<=10
            a1(9*k+1)=a1(19);
       k=k+1;
            end
             a2=reshape(a1,20,5);
            a3=reshape(99:-1:0,20,5);
            for       i=1:5
                 s=[s,a3(:,i),a2(:,i)];              %生成矩阵
            end
          if      ~t
                 disp( ' //   任意想一个两位数    a,  然后将这个两位数减去它的个位数字与十位数字之和,'    );
                    disp( ' //   得到数字    b,  再在下面矩阵的奇数列中找到     b,  最后记住其右边对应的小数   c' );
                    pause(10); t=t+1;
          end
              disp(    ' '    ); disp(s);
              pause(5); disp(       ' '    );
               d=input(   '  // 确定你已经完成计算并记下了那个小数,  摁‘ Enter ’键呈现此数字\n'    );
               disp(s(19,2)); pause(3); disp(             ' '    );
                  a=input(   ' //       ‘ Enter ’退出；  => ‘ 1 ’再试一次\n'       );
          end
```

使用说明：运行程序   I 生成一个 20 ×10 的矩阵 s, 你任意想一个两位数   a, 按照上面所说的步骤操作, 然后在   s 的奇数列中找到   b, 将 b 右边的小数记在心里 ,

再调用程序 II, 则返回你所记下的那个小数. （运行演示略）

原理说明：设任意一个两位数 $a=10k_1+k_2$, 则 $a-(k_1+k_2)=9k_1=b$, 所以 b 一定是 9 的倍数, 且只可能在 9 到 81 之间. 明白了这一点, 上面程序中的各种设置就一目了然了.

例 3 折半查找算法

要从一个数组 x 中找到一个指定的数 a, 通常的做法是从 x 的第一个数开始顺序查找. 如果 x 是一个无序的数组, 的确没有比顺序查找更好的方法了, 但如果 x 有序, 设计查找算法时就要充分利用这已有的规律, 折半查找就是针对有序数组进行查找的一种效率较高的方法. 对于一个 n 维数组, 折半查找最多比较次数为 $\lfloor \log_2 n \rfloor + 1$, 而顺序查找的平均比较次数为 n/2.

写一个算法：在数组 x 中查找数字 a, 其中 x 是一个元素各异并按升序排列的一维数组. 若找到 a, 则返回 a 在 x 中的位置, 若 a 不在 x 中则返回“找不到”. 折半查找算法程序如下：

```
function  s= BinarySearch (x,a)
 %折半查找法,x 是按升序排列的一维数组,a 是待查找的数 字
n=length(x);
sign=0;        %用 sign 标记是否查找成功, 赋初值为假, 当查找成功时其值为真
top=1;bott=n;      %查找区间端点下标,top 为“顶”,bott 为“底”
if  a<x(1)|a>x(n)    %当 a 比 x 的最小值小或比 x 的最大值大时, 返回找不到此数,
   s=  'The number is not found'       ; %并终止程序, 否则在 x 的内部查找 a
     return  ;
else
    while  ((sign==0)&(top<=bott))
mid=fix((bott+top)/2);              %求中位数并取整
      if  a==x(mid)                 %若找到 a
   s=mid;                           %记录它在 x 中的下标,
   sign=1;                          %并置标志变量为真
      elseif   a<x(mid)             %否则, 改变区间端点下标
   bott=mid-1;
```

```
            else
          top=mid+1;
            end
end
  if    sign==0                          %当 sign 为假, 说明在 x 中找不到 a
  s=     'The number is not found'        ;
   end
end
```

运行情况如下：

```
>> x=[1 3 4 6 8 9 12 15];          ?
>> s= BinarySearch (x,6)           ?
s =
     4
>> s= BinarySearch (x,7)           ?
s =
The number is not found
```

例 4 对答卷中选择题的初步统计函数

当做完一项问卷调查后, 或举行完一次考试后, 在对答卷中选择题的做答情况做统计分析时, 需要知道每个选择题中每个选项的被选次数及所占的百分比 .

现假设选择题只有 4 个选项（A,B,C,D）, 某些题目允许多选, 但不得选多于 2 项.(如果题目的选项或可选项增加的话 , 方法类似.)4 个选项分别用 1,2,3,4 表示（若选择 AB,则表示为 12）, 缺选时用 0 表示. 请编程序统计每个题目中每个选项的被选次数及所占的百分比 .

分析：若一共有 n 个题目, 可用两个 4 ×n 的矩阵分别表示每个选项的被选次数与所占的百分比, 在统计的过程中用数组分别为每个题目中的每个选项计数 .

MATLAB 程序如下：

```
function     [s1,s2]=TongJi1(A)
%A为答卷结果的数据, 返回值    s1,s2  分别为每个
%选项的被选次数与所占的百分比
```

```
[m,n]=size(A);
s1=[];s2=[];
D=ones(1,n).*m;          %D 记录每个题目中每个选项被选次数总和 ,
                          %给每个元素赋初值为   m
for   j=1:n
    B=zeros(4,1);         %B为计数器, 记录每个题目中的每个选项的被选次数
 for    i=1:m
  t=A(i,j);
       if    t==0          %缺选时, 在该题的总数记录中减    1
    D(j)=D(j)-1;
       elseif    t<10    %只选一项时, 在该项的计数器上加    1
    B(t)=B(t)+1;
       else               %当选两项时, 需分离数字, 再让每项的计数器加      1,
                          %同时该题目的各项被选次数总和加    1
    t1=floor(t/10);
       t2=t-t1*10;
       B(t1)=B(t1)+1;
       B(t2)=B(t2)+1;
       D(j)=D(j)+1;
    end
  end
s1=[s1,B];                %组合矩阵
B1=B./D(j);               %求百分比
s2=[s2,B1];
end
```

附录中 表 1 是一个有 20 个学生作答记录的数据, 若记该数据矩阵为 A, 调用上面的程序运行如下：

```
?[s1,s2]=TongJi1(A)      ?
s1 =
```

3 8 0 0 2 0 0 2

8 5 5 7 12 1 7 4

4 4 5 7 2 13 7 8

4 2 9 5 3 5 5 5

s2 =

0.1579 0.4211 0 0 0.1053 0 0 0.1053

0.4211 0.2632 0.2632 0.3684 0.6316 0.0526 0.3684 0.2105

0.2105 0.2105 0.2632 0.3684 0.1053 0.6842 0.3684 0.4211

0.2105 0.1053 0.4737 0.2632 0.1579 0.2632 0.2632 0.2632

例 5 列主元 Gauss 消去法解方程组

列主元 Gauss 消去法是指在解方程组时, 未知数顺序消去, 在要消去的那个未知数的系数中找按模最大者作为主元. 完成消元后, 系数矩阵化为上三角形, 然后在逐步回代求解未知数. 列主元 Gauss消去法是在综合考虑运算量与舍人误差控制的情况下一种较为理想的算法. 其算法描述如下：

1. 输入系数矩阵 A,右端项 b

2. 测 A 的阶数 n, 对 k=1,2, ⋯,n-1 循环

a) 按列主元

$$\alpha = \max_{k \le i \le n} |a_{ik}|$$

保存主元所在行的指标 i_k.

b) 若 $\alpha = 0$, 则系数矩阵奇异, 返回出错信息, 计算停止; 否则, 顺序进行.

c) 若 $i_k = k$ 则转向 d); 否则

换行 $a_{i_k j} \leftrightarrow a_{kj}, j = k+1, \cdots, n$

$b_{i_k} \leftrightarrow b_k$

d) 计算乘子 $m_{ik} = a_{ik} / a_{kk}, i = k+1, \cdots, n$

e) 消元:

$a_{ij} = a_{ij} - m_{ik} a_{ik}, \quad i, j = k+1, \cdots, n$

$b_i = b_i - m_{ik} b_k, \quad i = k+1, \cdots, n$

3. 回代

$$b_i = \left(b_i - \sum_{j=i-1}^{n} a_{ij} b_j \right) / a_{ii}, \quad i = n, n-1, \cdots, 1$$

方程组的解 X 存放在右端项 b 中.

MATLAB 程序如下：

```
function    s=LZYgauss(A,b)
%列主元高斯消去法解方程组:A    为系数矩阵,b   为右端项
n=length(A(:,1));                          % 测量  A 一行的长度, 得出   n
for   k=1:n-1                              % 循环, 消元
    [a,t]=max(abs(A(k:n,k)));              %寻找最大值( 记为   a)
    p=t+k-1;                               %最大值所在的行记为    p
      if   a==0                            % 若  a=0,则  A为奇异阵, 返回出错信息
       error(          ‘ A is a bizarre matrix       ’   );
      else                                 %若  a 不等于  0, 换行
       t1=A(k,:);
A(k,:)=A(p,:);
A(p,:)=t1;
t2=b(k);b(k)=b(p);b(p)=t2;
A,b
        for   j=k+1:n                      % 消元过程
    m=A(j,k)/A(k,k);
    A(j,:)=A(j,:)-m.*A(k,:);
    b(j)=b(j)-m*b(k);
  end
    end
end
A,b
b(n)=b(n)/A(n,n);                          % 回代过程
for   i=1:n-1
```

```
    q=(b(n-i)-sum(A(n-i,n-i+1:n).*b(n-i+1:n)))/A(n-i,n-i);
    b(n-i)=q;
end
s=b;                                    %解放在 s 中
```

程序运行（略）

2　程序的的调试与测试

程序调试的目的是检查程序是否正确，即程序能否顺利运行并得到预期结果. 在运行程序之前，应先设想到程序运行的各种情况，测试在各种情况下程序是否能正常运行. 对初学编程的人来说，很难保证所编的每个程序都能一次性运行通过，而大多情况下都需要对程序进行反复的调试之后才能正确运行 . 所以，不要害怕程序出错，要时时准备着去查找错误，改正错误.

2.1　程序常见的错误类型

2.1.1　易犯的低级错误

a　输入错误

常见的输入错误除了在写程序时疏忽所导致的手误外，一般还有：　①在输入某些标点时没有切换成英文状态；②表循环或判断语句的关键词 “ for ”, “ while ”, “ if ”的个数与 “ end ”的个数不对应（尤其是在多层循环嵌套语句中）；③左右括号不对应等.

b　对程序的修改不彻底导致的错误

由于之前对程序中的某些部分（如变量或符号名称）做了修改，但后面又有用到它们或与之有关系的地方却没有做相应的修改 .

2.1.2　语法错误

不符合 MATLAB 语言的规定，即为语法错误. 例如在用 MATLAB 语句表示数学式 “ $k_1 \leq x \leq k_2$ ” 时，不能直接写成 “ k1<=x<=k2 ”，而应写成 “ k1<=x&x<=k2 ”. 此

外, 输入错误也可能导致语法错误 .

2.1.3　逻辑错误

在程序设计中逻辑错误也是较为常见的一类错误 , 这类错误往往隐蔽性较强, 不易查找, 错误一旦发生, 轻则使程序不够优化 , 进行很多不必要的计算 , 重则程序运行错误或无法运行 . 产生逻辑错误的原因通常是算法设计有误 , 这时需要对算法进行修改, 但也可能算法没错, 而是程序本身的问题 .

下面通过一个例子来看一下发生逻辑错误的情形

例　分类统计_2

编写一个函数, 统计出一组数字（可以是无序的）中每种数字的个数 .

分析：因为待排数组不一定有序 , 所以不能再像1.3 中例　　1 那样在对x的遍历中只让x(i)　与已测数字中的最后一个进行比较 , 而应与所有已测数字进行比较 , 当与某个数字相同时, 对应计数器加　　1, 当与所有的数字都不同时才能说明测到新数 . 如此, 可设置状态标志变量来标志在每次遍历中是否测到新数 . 看如下程序：

```
function    [s,k]=FLTJ_2(x)
% x 为待统计的一维数组 , 返回值    s 的第一列为不同种类的数字,
% s 的第二列为对应数字的个数 , k      记录统计出的数字种类数目
n=length(x);
s1=x(1);          % s1   记录测到的新数字, 赋初值为     x 的第一个数字
s2=1;             % s2  记录  s1 中数字的个数, 赋初值为    1
k=1;              % k  记录已统计出的数字种类数 , 初值赋为      1
for   i=2:n       % 从第  2 项开始遍历数组   x
  flag=0;         % 标示变量, 标记是否测到新数字 , 此趟遍历开始前 , 其值应为假
    for   j=1:k
   if   x(i)==s1(j)          %如果  x(i)  与已测出的第    j  个数字相同 ,
s2(j)=s2(j)+1;               %则对应的计数器加    1, 并结束此次遍历
       break ;
    else                     %否则, 则说明测到新数字,flag       置为真
 flag=1;
```

```
        end
    end
    if  flag                    %   当  flag  为真, 即测到新数字, 将此新数赋给
                    %s                  的第一列, 对应的计数器为    1, 同时  k 值加 1
    s1=[s1;x(i)];
    s2=[s2;1];
        k=k+1;
    end
end
s=[s1,s2];                              % 将  s1 与 s2 拼接成一个两列的数组
```

此程序乍一看似乎顺理成章, 看一下其运行结果

```
>> x=[3,2,2,4,5,4];          ?
>> [s,k]=FLTJ_2(x)       ?
s =
    3     1
    2     2
    2     1
    4     2
    5     1
    4     1
k =
    6
```

输出结果显然是不对的, 考虑一下问题出在哪了？

事实上, 当第二个“ for ”循环中的判断语句第一次不成立时并不能立即得出“ 测到新数 ”的结论, 只有在对 j 的循环结束（即循环进行到 k）时, 若判断语句仍不成立才能说明测到了新数 . 因此, 应将第二个“ for ”循环内的判断语句修改如下（注意观察虚线框内的修改部分） :

```
        if  x(i)==s1(j)
    s2(j)=s2(j)+1;
```

```
        break;
    elseif    j==k    %对 j 的循环进行到  k 时判断语句仍不成立,
                      %则说明测到新数,置  flag  为 1
        flag=1;
    end
```

其实,还可以将标志变量 flag 换一种方式使用,将程序中的循环改写如下:

```
for  i=2:n      %从第 2 项开始遍历数组  x
  flag=0;       %标示变量,标记是否测到新数字,此趟遍历开始前,其值应为假
    for  j=1:k
      if   x(i)==s1(j)          %如果 x(i) 与已测出的第  j 个数字相同,
     s2(j)=s2(j)+1;             %则对应的计数器加  1,
     flag=1;                    %同时置 flag  为 1,并结束此次遍历
          break ;
      end
  end
  if  ~flag                     %当 flag  仍为假时,即测到新数字
      s1=[s1;x(i)];
   s2=[s2;1];
       k=k+1;
    end
```

通过分析你会发现,这两种用法实际上是等效的.其实, 此程序无需使用标志变量, 可将循环直接写成如下形式.

```
for  i=2:n
     for  j=1:k
  if   x(i)==s1(j);
s2(j)=s2(j)+1;
        break;
  elseif    j==k
s1=[s1;x(i)];
```

```
            s2=[s2;1];
            k=k+1;
               end
        end
end
```

修改后的程序运行如下：

```
>> x=[3,2,2,4,5,4];          ?
>> [s,k]=FLTJ_2(x)       ?
s =
     3     1
     2     2
     4     2
     5     1
k =
     4
```

请思考一下, 第一个 ” if ” 条件下如果没有 ” break ” 程序能否运行正确？想一想为什么？

2.1.4 运行错误

程序的运行错误通常包括不能正常运行和运行结果不正确 , 出错的原因一般有：

① 数据不对, 即输入的数据不符合算法要求（见下例）.

② 输入的矩阵大小不对 , 尤其是当输入的矩阵为一维数组时 , 应注意行向量与列向量在使用上的区别.

③ 程序不完善, 只能对 “ 某些 ” 数据运行正确 , 而对 “ 另一些 ” 数据则运行错误 , 或是根本无法正常运行 , 这有可能是算法考虑不周所致 .

看下面这个例子

例 在用1.3例 3 中的折半查找程序做查找时输入一组命令运行如下：

```
>> x=[3 5 2 4 7 6 11];          ?
>> s= BinarySearch (x,4)        ?
```

```
s =
     4
>> s=BinarySearch(x,5)       ?
s =
The number is not found
```

分析：第一次对 4 的查找返回了正确的结果, 但第二次对 5 进行查找时却出了错. 原因是数组 x 并非有序数组, 不符合折半查找的要求（要求数组为升序排列）. 对 4 的查找正确, 也仅仅是一个巧合而已, 恰好在第一次“ 折半 ”中找到了它, 而对 5 的查找就会在 4 的右边进行, 自然是找不到的.

2.2 程序查错的若干方法

① 先查看程序中有无输入错误.

② 善于利用程序运行时给出的出错信息来查找错误 .

③ 检查算法是否有误.

④ 在一些适当的位置显示一些变量的值, 看其运行情况, 如果发现前面的运行正确则继续向后开设显示变量观察运行情况 .

⑤ 将循环控制变量的值设成某个常数 , 如将原来的“ i=1:n ”, 改成“ i=k ” （k为1~n 之间的一个数）, 再看其运行情况.

下面来看一个例子, 仍以折半查找算法为例.

例 如果在编写折半查找算法的程序时将“ while ”循环下的“ mid=fix((bott+top)/2) ” 语句写成了“ mid=(bott+top)/2 ”, 看一下其运行情况：

```
>> x1=[1 3 4 6 8 9 12];          ?
>> s= BinarySearch(x1,8)        ?
s =
     5
>> s= BinarySearch(x1,3)        ?
s =
     2
```

```
>> x2=[1 3 4 6 8 9 12 15];          ?
>> s=BinarySearch(x2,8)      ?
??? Attempted to access x2(4.5); index must be a positive integer or
logical.
Error in ==> BinarySearch at 12
        if a==x(mid)  %          若找到    a
```

通过上面的运行情况发现, 在 x1 中能正确查找, 但在 x2 中查找时却返回了出错信息, 说是“ 试图访问 x2(4.5), 下标必须为正整数或合乎常理 ”, 因为数组 x2 的长度为 8, 在第 1 次“ 折半 ” 时得到 (1+8)/2=4.5=mid, 而数组的下标必须为整数, 所以出错. 事实上, 当数组 x 的长度 n 是偶数时, 第 1 次 “ 折半查找 ” 就无法进行, 因为(1+ n)/2 不是整数. 当 n 是奇数时, 至少能进行一次, 特别地当 $n=2^k-1$ （k=1,2,3, …） 时, 对 x 中所有元素的查找都能正常进行 （分析一下, 为什么？）, 例如上面的 x1 中恰有 7 个元素, 为 k=3 时的情形. 所以, 这里还要强调的是： 在对程序进行测试时, 当你有幸输入了某些特殊的数据（比如 , 在此例中输入数组 x 的长度恰好为 2^k-1）发现运行正确, 不要就此以为程序已经正确无误了 .

再回到原来的话题, 通过上面的测试发现了问题所在 , 为了使程序对所有的 n 都能正常运行, 应加入对 “ (bott+top)/2 ” 取整的操作, 函数 “ fix ” 的作用是向 0 方向取整, 也可使用 “ floor ” （朝负无穷大方向取整）.

3 算法与程序的优化

如果一个程序通过各种测试都能正常运行并得到正确的结果 , 换句话说, 这个程序没有任何错误, 就能说它已经 “ 完美 ” 了吗？答案是： 不一定. 下面就来说一下有关算法和程序的优化问题 .

在设计算法和程序时往往容易只去注重算法和程序的可行性 , 而忽视对算法和程序的优化. 当遇到一些问题需要庞大的计算量时 , 如果算法或程序不够优化, 则难以在有限的时间内完成计算 . （这一点我深有体会 .）另外, 虽然 MATLAB 有强大的计算功能, 但这也给它带来了自身的缺点 ,它和其他高级程序语言相比 , 程序的执行速度较慢 ,这就更需要去注重算法的优化问题了 .即使对于那些不需要

很大计算量的程序也要尽量做到优化，因为这体现了一个人的编程素质．所以要养成这样一种好的习惯，提高程序的质量．

程序的优化本质上也是算法的优化，如果一个算法描述的比较详细的话，它几乎也就指定了程序的每一步．若算法本身描述的不够详细，在编程时会给某些步骤的实现方式留有较大空间，这样就需要找到尽量好的实现方式以达到程序优化的目的，但一般情况下认为算法是足够详细的．如果一个算法设计的足够“优”的话，就等于从源头上控制了程序走向“劣质”．

算法优化的一般要求是：不仅在形式上尽量做到步骤简化，简单易懂，更重要的是能用最少的时间复杂度和空间复杂度完成所需计算．包括巧妙的设计程序流程，灵活的控制循环过程（如及时跳出循环或结束本次循环），较好的搜索方式及正确的搜索对象等，以避免不必要的计算过程．例如在判断一个整数 m 是否是素数时，可以看它能否被 m /2 以前的整数整除，而更快的方法是，只需看它能否被 $\sqrt{m}$ 以前的整数整除就可以了．再比如，在求k_1 与 k_2 之间的所有素数时跳过偶数直接对奇数进行判断，这都体现了算法优化的思想．下面通过几个具体的例子来体会其中所包含的优化思想．

例 1 冒泡排序算法

冒泡排序是一种简单的交换排序，其基本思想是两两比较待排序记录，如果是逆序则进行交换，直到这个记录中没有逆序的元素．

该算法的基本操作是逐趟进行比较和交换．第一趟比较将最大记录放在 x[n] 的位置．一般地，第 i 趟从 x[1] 到 x[n-i+1] 依次比较相邻的两个记录，将这 n-i+1 个记录中的最大者放在了第 n-i+1 的位置上．其算法程序如下：

```
function  s=BubbleSort(x)
 % 冒泡排序,x  为待排序数组
 n=length(x);
 for  i=1:n-1                    %最多做 n-1 趟排序
    flag=0;                      %flag  为交换标志，本趟排序开始前，交换标志应为假
        for  j=1:n-i             %每次从前向后扫描,j   从 1 到 n-i
            if  x(j)>x(j+1)      %如果前项大于后项则进行交换
        t=x(j+1);
```

```
    x(j+1)=x(j);
    x(j)=t;
    flag=1;                    %当发生了交换,将交换标志置为真
       end
end
if  (~flag)                    %若本趟排序未发生交换，则提前终止程序
       break;
    end
end
s=x;
```

说明：本程序通过使用标志变量 flag 来标志在每一趟排序中是否发生了交换,若某趟排序中一次交换都没有发生则说明此时数组已经为有序 （正序）,应提前终止算法（跳出循环）.若不使用这样的标志变量来控制循环往往会增加不必要的计算量.

例 2 公交线路查询问题

设计一个查询算法,给出一个公交线路网中从起始站 s1 到终到站 s2 之间的最佳线路.其中一个最简单的情形就是查找直达线路,假设相邻公交车站的平均行驶时间（包括停站时间） 为 3 分钟,若以时间最少为择优标准,请在此简化条件下完成查找直达线路的算法,并根据附录数据 (数据 1),利用此算法求出以下起始站到终到站之间的最佳路线.

⑴ 242 → 105 ⑵ 117 → 53 ⑶ 179 → 201 ⑷ 16 → 162

分析：为了便于 MATLAB 程序计算,应先将线路信息转化为矩阵形式,导入 MATLAB（可先将原始数据经过文本导入 Excel）.每条线路可用一个一维数组来表示,且将该线路终止站以后的节点用 0 来表示,每条线路从上往下顺序排列构成矩阵 A.此算法的核心是线路选择问题,要找最佳线路,应先找到所有的可行线路,然后再以所用的时间为关键字选出用时最少的线路.在寻找可行线路时,可先在每条线路中搜索 s1,当找到 s1 则接着在该线路中搜索 s2,若又找到 s2,则该线路为一可行线路,记录该线路及所需时间,并结束对该线路的搜索.另外,在搜索 s1 与 s2 时若遇到 0 节点,则停止对该数组的遍历.

下面是实现这一算法的一个程序，看它是否做到了足够的优化.

```
function    [L,t]=DirectLineSearch(A,s1,s2)
%直达线路查询
%A为线路信息矩阵,s1,s2    分别为起始站和终到站.
%返回值  L 为最佳线路,t    为所需时间
[m,n]=size(A);
L1=[];t1=[];                    % L1 记录可行线路,t1    记录对应线路所需时间
for   i=1:m
      for   j=1:n
            if   A(i,j)==s1                    %若找到  s1, 则从下一站点开始寻找    s2
                 for   k=j+1:n
                       if   A(i,k)==0          %若此节点为  0, 则跳出循环
                            break ;
                       elseif    A(i,k)==s2    %若找到  s2, 记录该线路及所需时间,
                                               %然后跳出循环
                            L1=[L1,i];
                            t1=[t1,(k-j)*3];
                                  break ;
                       end
                 end
            end
      end
end
m1=length(L1);                                 %测可行线路的个数
if   m1==0                                     %若没有可行线路, 则返回相应信息
   L=   'No direct line'        ;
   t=    'Null'    ;
elseif    m1==1
      L=L1;t=t1;          %否则, 存在可行线路, 用   L 存放最优线路,t    存放最小的时间
```

```
    else
L=L1(1);t=t1(1);          %分别给 L 和 t 赋初值为第一条可行线路和所需时间
    for   i=2:m1
          if   t1(i)<t              %若第 i 条可行线路的时间小于   t,
               L=i;                 %则给 L 和 t 重新赋值,
      t=t1(i);
          elseif    t1(i)==t        %若第 i 条可行线路的时间等于   t,
       L=[L,L1(i)];                %则将此线路并入   L
          end
       end
end
```

首先说明, 这个程序能正常运行并得到正确结果 , 但仔细观察之后就会发现它的不足之处：一个是在对 j 的循环中应先判断节点是否为 0, 若为 0 则停止向后访问, 转向下一条路的搜索 . 另一个是, 对于一个二维的数组矩阵 , 用两层 （不是两个）循环进行嵌套就可以遍历整个矩阵 , 得到所有需要的信息 , 而上面的程序中却出现了三层循环嵌套的局面 . 其实, 在这种情况下, 倘若找到了 s2, 本该停止对此线路节点的访问, 但这里的“ break ”只能跳出对 k 的循环, 而对该线路数组节点的访问（即对 j 的循环）将会一直进行到 n, 做了大量的“ 无用功 ”.

为了消除第三层的循环能否对第二个循环内的判断语句做如下修改：

```
if   A(i,j)==s1
     continue ;
       if   A(i,k)==s2
L1=[L1,i];
t1=[t1,(k-j)*3];
          break ;
     end
end
```

这种做法企图控制流程在搜到 s1 时能继续向下走, 搜索 s2, 而不用再嵌套循环 . 但这样却是行不通的, 因为即使 s1 的后面有 s2, 也会被先被“ if

A(i,j)==s1 ”拦截,“ continue ”后的语句将不被执行. 所以, 经过这番修改后得到的其实是一个错误的程序.

事实上, 若想消除第三层循环可将这第三层循环提出来放在第二层成为与“ j ”并列的循环, 若在对 j 的循环中找到了 s1, 可用一个标志变量对其进行标志, 然后再对 s1 后的节点进行访问, 查找 s2. 综上, 可将第一个“ for ”循环内的语句修改如下：

```
flag=0;                    % 用 flag 标志是否找到 s1, 为其赋初值为假
   for j=1:n
        if A(i,j)==0       %若该节点为 0, 则停止对该线路的搜索, 转向下一条线路
            break;
        elseif  A(i,j)==s1      %否则, 若找到 s1, 置 flag 为真, 并跳出循环
      flag=1;
            break;
        end
   end
if flag              %若 flag 为真, 则找到 s1, 从 s1 的下一节点开始搜索 s2
     for k=j+1:n
          if A(i,k)==0
              break;
          elseif  A(i,k)==s2       %若找到 s2, 记录该线路及所需时间,
                                   %然后跳出循环
     L1=[L1,i];
     t1=[t1,(k-j)*3];
              break;
          end
     end
end
```

若将程序中重叠的部分合并还可以得到一种形式上更简洁的方案：

```
q=s1;       %用 q 保存 s1 的原始值
for  i=1:m
    s1=q;      %每一次给 s1 赋初值
    p=0;     %用 p 值标记是否搜到  s1 或 s2
    k=0;     %用 k 记录站点差
     for  j=1:n
      if  ~A(i,j)
            break ;
       elseif    A(i,j)==s1     %若搜到 s1, 之后在该线路上搜索   s2, 并记 p 为 1
      p=p+1;
              if  p==1
        k=j-k;
        s1=s2;
           elseif    p==2     %当 p 值为 2 时, 说明已搜到 s2, 记录相关信息,
       L1=[L1,i];
                t1=[t1,3*k];                %同时 s1 恢复至原始值,  进行下一线路的搜索
                      break ;
                   end
               end
          end
    end
```

程序运行如下：

```
?[L,t]=DirectLineSearch(242,105,A)          ?
L =
     8
t =
    24
?[L,t]=DirectLineSearch(117,53,A)          ?
```

```
L =
    10
t =
    15
?[L,t]=DirectLineSearch(179,201,A)          ?
L =
     7    14
t =
    27
?[L,t]=DirectLineSearch(16,162,A)           ?
L =
No direct line
t =
Null
```

注：在设计算法或循环控制时 , 应注意信息获取的途径 , 避免做无用的操作步骤. 如果上面这个程序不够优化 , 它将为后续转车的程序造成不良影响 .

例 3 分类统计_3 ：

统计一组数字中每种数字的个数 ,并使统计出的数字升序排列 .

分析：事实上要解决这个问题 , 完全可以先利用 “ 分类统计 _2” 进行统计后再用冒泡排序等排序算法对已统计出的数字进行排序即可 , 但, 要是能在统计的过程中又同时进行排序操作势必会比前者的计算量更少 . 如此, 可采用 “ 折半插入排序 ”的方法完成排序, 即, 将后续统计出的数字用插入排序的方法插入到已统计出的有序数组, 而插入位置则用折半查找的方式寻找 , 若测到的不是新数字只需使其对应的计数器加 1. 下面看这个算法的程序：

```
function  [s,k]=FLTJ_3(x)
%统计数组 x 中每种数字的个数, 将统计出的数字按升序排列
n=length(x);
s1=x(1);s2=1;                % s1 记录统计出的数字,s2  记录对应数字个数, 并为它们
                             %分别赋初值为： x 的第 1 个数字, 个数为 1
k=1;                         % k  记录已统计出的数字个数
```

```
for   i=2:n                    %从 x 的第 2 个元素开始遍历
      if   x(i)<s1(1)          %  若 x(i)  比 s1(1) 大, 应放在   s1(1) 前面,
                               %对应计数器位置置   1, 同时 k 值增加 1
         s1=[x(i);s1];
      s2=[1;s2];
      k=k+1;
      elseif    x(i)>s1(k)     %若 x(i)  比 s1(k) 小, 应放在   s1(k) 后面
      s1=[s1;x(i)];
      s2=[s2;1];
      k=k+1;
      else                     %否则, 应在  s1 内部查找 x(i)  或其插入位置
       flag=0;                 %标志变量, 标志是否在    s1 中找到 x(i)
      top=1;bott=k;            %查找区间下标
          while   (flag==0&top<=bott)
        mid=fix((bott+top)/2);
          if   x(i)==s1(mid)   %若在 s1 中找到 x(i),
      s2(mid)=s2(mid)+1;       %让其对应计数器加   1
      flag=1;                  %并置标志变量为真
          elseif    x(i)<s1(mid)
      bott=mid-1;
          else
      top=mid+1;
          end
      end
      if   flag==0             %若找到新数字, 则此新数应放在    s1(top)   与
                               % s1(top-1) 之间, 同时   k 值加 1
      s1=[s1(1:bott);x(i);s1(top:k)];
      s2=[s2(1:bott);1;s2(top:k)];
      k=k+1;
```

```
            end
        end
end
s=[s1,s2];
```

程序运行如下：

```
>> x=[4 5 3 4 5 7 5 1 2 3 8 3 2 1];                ?
>> [s,k]=FLTJ_3(x)      ?
s =
    1    2
    2    2
    3    3
    4    2
    5    3
    7    1
    8    1
k =
    7
```

评论：此程序除涉及前面例子中的统计方法外，还用到了插入排序及折半查找的思想，因此它着实能锻炼一个人整合各种算法的能力，即将各种算法的思想整合到一个程序中以解决特定的问题，并在这个过程中巧妙利用已有信息（或在前面的计算中所得到的信息），做到程序优化.

事实上，对于编程能力的训练，往往是先从解决一些较为简单问题入手，然后通过对这些问题修改某些条件，增加难度等不断的进行摸索，在不知不觉中自己的编程能就已经被提升到了一个新的高度.

4　有关循环控制的讨论

几乎所有实用的程序都包含循环控制，它和顺序结构、　　选择结构共同作为各种复杂程序的基本构造单元.MATLAB 中常见的循环结构是　for　语句和 while　语句.

对于 for 型循环, 须事先给循环变量赋初值或给出循环结束条件 , 通过顺序执行循环变量的值或在执行循环体的同时判断循环结束条件来完成循环控制过程 , 而 while 型循环则是针对循环变量值或循环次数事先不确定的情况 , 通过判断是否满足给定的条件来决定循环是否继续 . 但, 在具体使用 “ for ” 和 “ while ” 时, 较为灵活多变, 至于何时使用以及怎样使用往往决定循环控制实现的成败和效果 .

下面通过一个具体问题就一类循环控制问题展开讨论 .

问题: 对一个 m×n 的矩阵 A, 计算 A的每一列中相邻两个元素的比值 (前项比后项), 若某个比值落在区间 [$e^{-2/(m+1)}$, $e^{2/(m+1)}$] 之外, 则让该列所有元素加 1, 重新进行判断, 直到比值都落在区间内 . 求变化后的 A 及每列的增加值 .

分析: 第 1 种情况, 当 A中所有元素都大于 0 时, 相邻元素间的比值会随着元素值的增加而趋近于 1. 因此, 对于某一列中已落在区间 [$e^{-2/(m+1)}$, $e^{2/(m+1)}$] 内的比值, 当对此列所有元素加 1 后这些比值仍会落在区间内 . 所以, 在对各列中计算的比值从上到下依次判别时若遇到某个比值在区间之外 , 对该列元素加 1 后, 只需从变化后的当前值开始继续向下判断 , 无需回望前面已通过的值 . 这样, 循环变量值的变化不是严格递增的, 而是要在一些地方 “ 停留 ”, 不妨称这种情况为 “ 停留式 ” .

第 2 种情况, 当 A中有小于 0 的元素时, 在对各列中计算的比值依次判别时若遇到某个比值在区间之外, 对该列所有元素加 1 后可能会导致前面已通过判别的比值又跑到区间之外, 这样就需要返回该列首元素重新判断 . 不妨称这种情况为 “ 返回式 ” .

对于第 1 种情况可用下面程序完成

```
function  [s1,s2]=fei_1a(A)
%A为待输入矩阵, 返回值 s1 是变化后的 A, s2 是每列的增加值
[m,n]=size(A);
t1=exp(-2/(m+1));
t2=exp(2/(1+m));
A0=A;
for  j=1:n
      for  i=1:m-1
```

```
t=A0(i,j)/A0(i+1,j);                    %计算比值
      while  t<t1|t>t2                  %用 while 实现对“ 停留 ”的控制
   A0(:,j)=A0(:,j)+1;
   t=A0(i,j)/A0(i+1,j);
      end
end
end
s1=A0;
s2=A0(1,:)-A(1,:);
```

程序运行如下:

```
?A=[2,1,8;5,6,3;3,20,11];          ?
?[s1,s2]=fei_1a(A)       ?
s1 =
    5    17    18
    8    22    13
    6    36    21
s2 =
    3    16    10
```

说明: 因为事先并不知道应在何处停留 , 以及停留 “ 多久 ”, 因此, 可用 while 循环实现对“ 停留 ”的控制 , 将判别条件作为“ while ”的循环条件.

对于第 2 种情况似乎要麻烦一些 , 想一想该怎么办?

经过一番思考 , 你是否已经想到了 C语言中有一个 goto 语句(即无条件转向语句), 若用它来解决这问题看来是很方便的 , 于是就有了下面这个循环过程 :

```
for  j=1:n
loop  :  for  i=1:m-1
         t=A0(i,j)/A0(i+1,j);
            if  t<t1|t>t2
          A0(:,j)=A0(:,j)+1;
          goto loop;
```

```
                end
    end
end
```

但, 事实上这仅仅是一个理想,MATLAB 中不能使用无条件转向的 goto 语句. 解决这个问题的关键是动态的控制循环变量 j, 不能只让 j 从 1 到 n 严格递增的顺序取值. 如果用变化的数组来动态的控制 j 的取值, 似乎问题就容易解决了.

分析一下下面的程序段.

```
B=1:n                    %数组初值
k=n; p=0;                %k和 p 为计数器
for  j=B(1):B(k)         %用数组给 j 赋值
      for  i=1:m-1
      t=A0(i,j)/A0(i+1,j);
           if  t<t1|t>t2
        A0(:,j)=A0(:,j)+1;
        p=p+1;
       B=[B(1:j+p-1),j,B(j+p:k+p-1)];
           break;
         end
     end
end
```

和上面一样, 这种企图通过变动的数组来实现对 j 取值的动态的控制这也只是一个理想, 因为 for 循环不能使用内部重新赋值循环变量而终止, 语句“ j=B(1):B(k) ”并非是将 B 的元素依次赋给 j, 而是在一开始就被翻译为“ j=1:n ”, 所以在循环体中无论数组 B怎样变换都无法再影响到 j 的取值. 因此上面程序的运行结果将是错误的 .

其实通过和第 1 种情况类比联想一下就会发现, 第 1 中情况中的“ 停留 ”实际上指的是对 i 的停留, 而第 2 种情况中的 ” 返回 ” 也是对 i 来说的, 它又可以看成是对 j 的停留. 因为“ fei_1a ”中的 while 循环在“ i ”的下面, 因此通过类比后可将“ while ”放在“ j ”的下面, 将“ i ”放到“ while ”的内部来解决第 2 种情况. 同时 while 内的判别条件也应由 0 维(对元素的判断) 为升高到 1 维(对数组的判断). 下面程序段是这种循环的实现 :

```
for   j=1:n
      for   i=1:m-1        %用数组  a 中的元素标记此列中的比值是否通过判别条件,
                           %给  a 赋初值
          if   A0(i+1,j)==0            %若分母为  0,  让该列加   1
     A0(:,j)=A0(:,j)+1;
          end
     t=A0(i,j)/A0(i+1,j);
          if   t<t1|t>t2
     a(i)=1;
          else
     a(i)=0;
          end
    end
    while   any(a)          %      若   a 中有非  0 元,  应对此列加   1 后重新判断
  A0(:,j)=A0(:,j)+1;
       for   i=1:m-1
           if   A0(i+1,j)==0
    A0(:,j)=A0(:,j)+1;
           end
  t=A0(i,j)/A0(i+1,j);
            if   t<t1|t>t2
     a(i)=1;
            else
     a(i)=0;
                end
           end
      end
  end
```

事实上, 当从本质上对上面的循环过程加以分析, 则解决这第 2 种情况还可

以有一个更好的方法, 因为这个循环控制的本质其实是 : 下一次循环变量的值从上一次循环中获取, 这样就只需要用一个变量来动态控制 j 的取值就行了, 无需像前面那样使用数组赋值 . 其程序实现如下

```
function    [s1,s2]=fei_2b(A)
[m,n]=size(A);
t1=exp(-2/(m+1));
t2=exp(2/(1+m));
A0=A;
j=1;                          %       给  j  赋初值
   while  j<=n
          for  i=1:m-1
               if   A0(i+1,j)==0
           A0(:,j)=A0(:,j)+1;
               end
         t=A0(i,j)/A0(i+1,j);
             if   t<t1|t>t2        %若比值在区间之外, 则让该列加    1, 下次循环仍从
                                   %此列开始, 否则当此列判断结束 , 从下一列开始
 A0(:,j)=A0(:,j)+1;
 j=j;
              break ;
     elseif    i==m-1
 j=j+1;
     end
end
  end
s1=A0;
s2=A0(1,:)-A(1,:);
```

说明: 此程序中以 j 的取值作为 while 循环的条件来实现控制 . 可以看出它不但在形式上较上面的程序简洁 , 通过进一步的分析还会发现它在效率上也优于

前者.

程序运行如下:

```
?B=[2,-1,8;-5,6,3;-3,20,-11];          ?
?[s1,s2]=fei_2b(B)      ?
s1 =
   18    15    41
   11    22    36
   13    36    22
s2 =
      16    16    33
```

同理, 第 1 种情况的循环体也可做如下形式的 “ 改版 ” :

```
   for   j=1:n
    i=1;
        while   i<=m-1
       t=A0(i,j)/A0(i+1,j);
            if   t<t1|t>t2
        A0(:,j)=A0(:,j)+1;
        i=i;
            else
        i=i+1;
            end
      end
end
```

说明：在分析上面这些程序时要注意理解里面的循环控制框架，至于一些细节问题无需特别关注，例子只是为了更方便的说明问题 . 应能根据问题条件的改变灵活变动程序中的某些设置 .

总结：在对一个循环控制过程进行分析时 , 应善于抓住其本质，注意从不同的角度灵活使用 “ for ”, “ while ” 与 “ if ” 以及它们的不同组合方式 .

5　MATLAB 编程过程中通常要注意的问题

5.1　有关程序的拆分与组合

在编写一个蕴含多个程序模块的较大程序时 , 通常还要考虑什么时候应将各个程序模块 （子程序） 分开来写, 便于其他函数调用, 什么时候又适合用一个函数整体完成流程.

对于一些较为典型的算法, 或者某个独立的计算过程会在以后的计算中多次被用到, 最好将这样的计算过程写成独立的函数 , 以便被其它函数调用 , 或是被后续的计算过程使用 . 若是在被其他函数调用时需要对某些参数进行修改 , 最好将这些参数设置为从函数输入的形式 , 这就要求在编写函数或函数模块时应尽量考虑其通用性.

另外, 在编程过程中能用矩阵操作完成的尽量不用循环, 因为在 MATLAB 中, 矩阵语句被直接翻译成逻辑变量（0,1 变量）执行, 而循环语句则是采用逐行翻译的方式首先翻译成 MATLAB 的母语句, 执行速度较前者会大大降低.

5.2　其它

1. 对于一个较复杂的程序最好是先写出较详细的算法步骤 , 然后在算法的指导下进行编程, 以免直接进行编程时很多地方考虑不周 , 以至寸步难行.

2. 若某个变量值只是在程序运行时被显示 , 而并非作为函数输出值, 则这个变量值不能作为其它函数的输入被直接使用 , 若想使用它需将其包含在函数的输出项中.

3. 要养成写注释的习惯, 以增强程序的可读性.

4. Excel 与 MATLAB 的连接

当所要处理的数据在 Excel 表中（或着这些数据可以导入 Excel）, 尤其是当数据量较大时, 将 Excel 表导入 MATLAB 就显得较为必要. 其导入方法如下：

①启动 excel;

②单击 “ 工具 ” 加载宏命令, 在弹出的加载宏对话框中单击 “ 浏览 ” 按钮 ;

③找到 MATLAB7\toolbox\exlink.xla;

④返回加载宏对话窗口, 单击确定.

建立连接后, 可利用 ”putmatriax ” 与 ”getmatriax ” 实现 Excel 与 MATLAB之间的数据交换.

5. 文本数据导入 Excel 的方法：

①启动 excel;

②单击 “ 数据 ” 导入外部数据\ 外部数据命令, 在弹出的选择数据源对话框中找到文本数据, 单击 “ 打开 ” 按钮;

③设置分隔符及其它操作;

④单击完成及确定.

6 综合训练

通过前面的学习, 下面来看一个较为综合性的程序设计题目 , 练习一下从分析问题到算法设计的全过程.

例 自习室开放的优化管理

问题叙述：

近年来, 大学用电浪费比较严重, 集中体现在学生上晚自习上, 一种情况是去某个教室上自习的人比较少, 但是教室内的灯却全部打开, 第二种情况是晚上上自习的总人数比较少, 但是开放的教室比较多, 这要求我们提供一种最节约、 最合理的管理方法.

某学校收集了部分数据（数据见附录表 2）, 请完成下面的问题：

管理人员只需要每天晚上开一部分教室供学生上自习 , 每天晚上从 7:00--10:00 开放（如果哪个教室被开放, 则假设此教室的所有灯管全部打开） .

假如学校有 8000名同学, 每个同学是否上自习相互独立 , 上自习的可能性为 0.7. 要使需要上自习的同学满足程度不低于 95%,开放的教室满座率不低于 4/5, 同时尽量不超过 90%问该安排哪些教室开放, 能达到节约用电的目的.

分析：因为题中假设 ,“ 如果哪个教室开放, 则此教室的所有灯管全部打开 ” , 所以原始数据的第 4 列（开关数）和第 5 列（一个开关控制的灯管数）显然是无用的. 既然题目要求安排哪些教室能达到节约用电的目的 , 试想： 若将每个教室的

灯管的总功率除以该教室的座位数得到每个教室中平均每个座位所占的功率 $\overline{w_i}$，$\overline{w_i}$ 越小说明该教室的能源利用效率越高，则这样的教室越应该被优先选为开放教室. 所以若将教室按 $\overline{w_i}$ 升序排列，从前向后选择教室，同时看是否满足上自习同学的满足程度及满座率的限制条件 . 当条件满足时, 则找到一个较优的可行方案, 但并不能保证是最优的, 例如, 当被选中的最后一个教室比较大（其灯管的总功率也较大），而它若用其后面的一个较小的教室（灯管的总功率也较前者小）代替的话仍能满足条件 . 这样的话, 这个新方案就会比初始方案更优 . 同理, 可让原来可行方案中的最后两个教室用剩余教室中的某两个代替，看能否找到更优的方案. 但, 这样的替换一般不会进行太多, 因为越往下进行限制条件越难以得到满足, 所以一般会在经过较少的几步替换操作后得到最优解 .

通过上面的分析可知, 事实上我们只需直接使用如下数据信息：①教室号；②座位数；③每个教室的总功率；④每个教室中平均每个座位所占的功率 .

因此可先对原始数据进行如下预处理：

⑴. 第 1 和第 2 列不动, 删除第 4 与第 5 列；

⑵. 计算每个教室灯管的总功率 $w_{i总}$（即, “ 灯管数 ” 一列与 “ 每只灯管的功率 ” 一列对应项乘积）放入第 3 列；

⑶. 计算每个教室中平均每个座位所占的功率 $\overline{w_i}$（$\overline{w_i} = w_{i总}$ / 座位数）放入第 4 列；

⑷. 将矩阵按 $\overline{w_i}$ 升序排列 . 如此建立一个 4 列的矩阵记为 A.

（注：上述操作可通过 Excel 直接完成, 也可编写一个 MATLAB 程序完成, 其中的排序过程可使用 MATLAB 库函数 sort, 也可仿照冒泡排序法编一个函数 , 以 $\overline{w_i}$ 列为关键字进行比较，交换时须整行交换 .）

经过预处理后的数字信息矩阵 A 为一个 4 列矩阵, 各列数据分别是： 教室号 c_i；每个教室座位数 s_i；每个教室的总功率 $w_{i总}$；教室 i 中平均每个座位所占的功率 $\overline{w_i}$. 下面写出对问题求解的具体算法：

Step1：输入矩阵 A,行数 n, 计算上自习人数的期望值 $8000\times0.7=5600$, 计算至少应满足的上自习的同学人数 $N=5600\times95\%$.

Step2：用 S1 记录入选教室的座位总数, 并为其赋初值 $S1=0$.

Step3：对 i =1,2,3, …,n 循环

a) 把第 i 个教室的座位数加进 S1, 即 S1 = S1+A(i ,2);

b) 如果满足条件 $\frac{4}{5}$?S1 ≤N ≤90% ?S1 , 则将前 i 个教室编号放入 c（可行方案 1）, 计算前 i 个教室灯管的总功率之和 W1, 并记录此时的 i 值（令 p=i ）, 然后跳出循环.

Step4：对 p 以后的教室按每个教室的总功率 $w_{i总}$（第 3 列）升序排列, 排列后的矩阵放入 B中. 设置标志变量 flag1 标记是否有优于方案 1 的可行方案, 并给 flag1 赋初值为 0.

Step5：对 i =1,2, …,n-p 循环

用 B中的第 i 个教室替换 c 中的最后一个教室并计算替换后的座位总数 S2及教室灯管的总功率之和 W2 , 若满足条件 $\frac{4}{5}$?S2 ≤N ≤90% ?S2 和W2 <W1, 则找到比方案 1 更优的可行方案(记为方案 2), 置 flag1 的值为 1, 记录相关数据, 跳出循环.

Step6：判断 flag1 的值, 若 flag1 为 0, 说明方案 2 不存在, 无需再寻找方案 3, 程序停止, 否则程序顺序进行, 寻找是否有更优的方案.

Step7：将 c 中的最后两个教室去掉加入剩余的教室中按 $w_{i总}$ 升序排列, 排列后的矩阵放入 D 中, 设置标志变量 flag2 标记是否有优于方案 2 的可行方案, 给 flag2 赋初值为 0.

Step8：对 i =1,2, …,n-p+2 循环

对 j =i +1,…,n-p+2 循环, 用 D中的第 i 个教室与第 j 个教室替换 c 中的最后两个教室并计算替换后的座位总数 S3 及教室灯管的总功率之和 W3 , 若满足条件 $\frac{4}{5}$?S3 ≤N ≤90% ?S3 和W3 < W2 , 则找到比方案 2 更优的可行方案 (记为方案 3), 置 flag2 的值为 1, 记录相关数据, 跳出循环. 返回最优解.

MATLAB程序如下:

```
function    c=ZXSGL(A)
%矩阵 A 为经过预处理后的数据信息矩阵
%返回值 c 为最优解, 存放所选教室的编号
n=length(A(:,1));
```

```
S1=0;                          % S1  记录入选教室的座位总数, 赋初值    0.
N=8000*0.7*0.95;               % 计算至少应满足的上自习的同学人数
for  i=1:n                     %寻找方案  1
   S1=S1+A(i,2);                        % 计算前  i  个教室的座位数
      if  0.8*S1<=N&N<=0.9*S1   %当条件满足时存在可行解
      c=A(1:i,1);                       %将前  i  个教室编号放入  c, 得到可行方案  1
      W1=sum(A(1:i,3));                 %计算前  i  个教室灯管的总功率之和
      p=i;                              % 记录此时的  i  值
          break;                        % 当条件第一次满足时跳出循环
      end
end
c=c';                          %将所选教室编号以行向量输出
B=A(p+1:end,:);                %将 A 中在 p 以后的教室放入  B 中
[Y,I]=sort(B(:,3));            %对 B 中的教室按灯管的总功率升序排列,
                  %Y              存放排序后的值,I    存放相应的下标
flag1=0;          %flag1  标记是否有优于方案  1 的可行方案（方案  2）, 赋初值为  0
for  i=1:n-p      %寻找方案  2
   S2=sum(A(1:p-1,2))+B(I(i),2);            %计算替换后的座位总数及
   W2=sum(A(1:p-1,3))+Y(i);                 %教室灯管的总功率之和
      if  (0.8*S2<=N&N<=0.9*S2)&(W2<W1)%当条件满足时存在优于方案   1
                                           %的可行解
      c=[c(1:end-1),B(I(i),1)];            %记录替换后的教室
      flag1=1;
          break;
      end
end
if   flag1==0      %若 flag1  为 0, 说明方案  2 不存在, 返回  flag1  的值, 无需再寻找
                   %方案 3, 程序停止
 flag1
```

```
    return  ;
else                    %否则, 继续寻找是否有更优的方案
   D=A(p-1:end,:);
   [Y,I]=sort(D(:,3));
   flag2=0;        %flag2  标记是否有优于方案   2 的可行方案（方案   3）, 赋初值为  0
     for   i=1:n-p+2          %寻找方案  3
       for   j=i+1:n-p+2
      S3=sum(A(1:p-2,2))+D(I(i),2)+D(I(j),2);
      W3=sum(A(1:p-2,3))+Y(i)+Y(j);
              if   (0.8*S3<=N&N<=0.9*S3)&(W3<W2)
       c=[c(1:end-2),D(I(i),1),D(I(j),1)];
       flag2=1;
              end
      end
end
end
 flag1                    %  返回标志变量的值
 flag2
```

程序运行如下：

```
?c=ZXSGL(A) ?
flag1 =
     1
flag2 =
     0
c =
     24  32  40  14  28  36  39  13  27  29  35  37  38  30 18  3  4
17  5  19  43  9  23  31  26  34  6  8  10  20  22  12  7  21  25
```

由运行结果中可以看出：flag1=1　说明第 1 次替换发生了, 而 flag2=0 则说明第 2 次替换并没有发生, 即方案　2 是最优解,c　为由最优解所确定的应开放教

室.

注：如果仅仅是为了得到问题的结果 , 本题亦可通过 Lingo 等优化软件直接求解, 而本例则给了出一个具体的具有针对性的求解方法 .

7 几个常用数学方法的算法程序

1. 雅可比（Jacobi）迭代算法

该算法是解方程组的一个较常用的迭代算法 .

```
function    x=ykb(A,b,x0,tol)
 % A为系数矩阵,b   为右端项,x0( 列向量) 为迭代初值,tol       为精度
D=diag(diag(A));                    %将 A 分解为 D,-L,-U
L=-tril(A,-1);
U=-triu(A,1);
B1=D\(L+U);
f1=D\b;
q=norm(B1);
d=1;
while   q*d/(1-q)>tol                % 迭代过程
   x=B1*x0+f1;
   d=norm(x-x0);
   x0=x;
end
```

2. 拉格朗日（Lagrange）插值函数算法

该算法用于求解插值点处的函数值 .

```
function    y=lagr1(x0,y0,x)
% x0,y0 为已知点列,x   为待插值节点（可为数组）
%当输入参数只有  x0, y0 时, 返回 y 为插值函数
%当输入参数有  x 时返回 y 为插值函数在  x 处所对应的函数值
n=length(x0);
```

```
if   nargin==2
     syms x
     y=0;
     for  i=1:n
          L=1;
           for   j=1:n
             if    j~=i
          L=L*(x-x0(j))/(x0(i)-x0(j));
          end
     end
     y=y+L*y0(i);
     y=simple(y);
end
x1=x0(1):0.01:x0(n);
y1=subs(y,x1);
      plot(x1,y1);
else
      m=length(x);
     for   k=1:m                          %  对每个插值节点分别求值
          s=0;
            for   i=1:n
             L=1;
                 for   j=1:n
                      if    j~=i
                 L=L*(x(k)-x0(j))/(x0(i)-x0(j));
                      end
    end
    s=s+L*y0(i);
end
```

```
        y(k)=s;
      end
end
```

注：以上两个算法属数值计算类, 注意对比其解析表达式与用程序进行数值计算时在操作方式上的不同.

3. 图论相关算法

1) 最小生成树

```
function    [w,E]=MinTree(A)
%避圈法求最小生成树
% A为图的赋权邻接矩阵
% w记录最小树的权值之和,E    记录最小树上的边
n=size(A,1);
for   i=1:n
    A(i,i)=inf;
end
s1=[];s2=[];        % s1,s2  记录一条边上的两个顶点
w=0; k=1;           % k 记录顶点数
T=A+inf;
T(1,:)=A(1,:);
A(:,1)=inf;
while   k<n
    [p1,q1]=min(T);                 % q1 记录行下标
    [p2,q2]=min(p1);
     i=q1(q2);
    s1=[s1,i];s2=[s2,q2];
    w=w+p; k=k+1;
    A(:,q2)=inf;                    %若此顶点已被连接，则切断此顶点的入口
    T(q2,:)=A(q2,:);                % 在  T 中并入此顶点的出口
    T(:,q2)=inf;
```

```
end
E=[s1;s2];                          % E记录最小树上的边
```

2) 最短路的 Dijkstra 算法

```
    function   [d,path]=ShortPath(A,s,t)
    % Dijkstra  最短路算法实现,A  为图的赋权邻接矩阵
    %当输入参数含有  s 和 t 时，求 s 到 t 的最短路
    %当输入参数只有  s 时，求 s 到其它顶点的最短路
    %返回值 d 为最短路权值，path  为最短路径
    if  nargin==2
       flag=0;
    elseif   nargin==3
       flag=1;
    end
    n=length(A);
    for  i=1:n
       A(i,i)=inf;
    end
    V=zeros(1,n);          % 存储 lamda(由来边) 标号值
    D=zeros(1,n);          % 用 D记录权值
    T=A+inf;               % T  为标号矩阵
    T(s,:)=A(s,:);         % 先给起点标号
    A(:,s)=inf;            %关闭进入起点的边
    for  k=1:n-1
       [p,q]=min(T);       % p 记录各列最小值， q 为对应的行下标
       q1=q;               % 用 q1 保留行下标
       [p,q]=min(p);       % 求最小权值及其列下标
       V(q)=q1(q);         % 求该顶点 lamda 值
```

```
if     flag&q==t
  d=p;                    % 求最短路权值
       break;
else                          % 修改  T 标号：
  D(q)=p;                     % 求最短路权值
  A(:,q)=inf;                 % 将  A中第  q 列的值改为   inf
       T(q,:)=A(q,:)+p;            %同时修改从顶点   q 出去的边上的权值
       T(:,q)=inf;                 %顶点  q 点已完成标号，将进入   q 的边关闭
     end
end
if   flag             %输入参数含有   s 和 t，求 s 到 t  的最短路
   path=t;                                % 逆向搜索路径
  while     path(1)~=s
     path=[V(t),path];
     t=V(t);
   end
else       %输入参数只有   s，求 s 到其它顶点的最短路
   for       i=1:n
        if    i~=s
       path0=i;v0=i;                          %逆向搜索路径
            while     path0(1)~=s
         path0=[V(i),path0];
         i=V(i);
            end
   d=D; path(v0)={path0};              %将路径信息存放在元胞数组中
   %       在命令窗口显示权值和路径
   disp([int2str(s),              '->'   ,int2str(v0),       ' d='    ,...
   int2str(D(v0)),              ' path= '     ,int2str(path0)]);
end
```

```
    end
  end
```

3) Ford 最短路算法

该算法用于求解一个赋权图中 v_s 到 v_t 的最短路，并且对于权值 $w_{ij} \leq 0$ 的情况同样适用.

```
function   [w,v]=Ford(W,s,t)
% W为图的带权邻接矩阵,s   为发点,t   为终点
%返回值  w为最短路的权值之和,v   为最短路线上的顶点下标
n=length(W);
d(:,1)=(W(s,:))';          %   求  d(vs,vj)=min{d(vs,vi)+wij}        的解,   用     d 存放
                              % d(t)(v1,vj),      赋初值为   W的第  s 行, 以列存放
j=1;
while   j
      for   i=1:n
       b(i)=min(W(:,i)+d(:,j));
      end
      j=j+1;
   d=[d,b'];
      if   d(:,j)==d(:,j-1)          %  若找到最短路,  跳出循环
           break ;
       end
end
w=d(t,j);                          %  记录最短路的权值之和
v=t;                    %              用数组    v 存放最短路上的顶点, 终点为    t
while   v(1)~=s
          for   i=n:-1:1
               if   i~=t&W(i,t)+d(i,j)==d(t,j)
                    break ;
```

```
            end
        end
    v=[i,v];
    t=i;
end
```

4. 模糊聚类分析算法程序（组）

在模糊聚类分析中, 该算法中的‘程序_3’用于求解模糊矩阵、模糊相似矩阵和模糊等价矩阵．‘程序_4’用来完成聚类．‘程序_1’和‘程序_2’是为‘程序_3’服务的子程序.（有关模糊聚类分析的有关知识可查阅相关资料.）

程序_1 求模糊合成矩阵的最大最小法

```
function   s=mhhc(R1,R2)                %          模糊合成
[m,n]=size(R1);
[n,n1]=size(R2);
for  i=1:m
    for  j=1:n1
    s(i,j)=max(min(R1(i,:),(R2(:,j))'));                    %最大最小法
    end
end
```

注：此函数被‘程序_2’调用

程序_2 求模糊传递包的算法

```
function   s=mhcdb(R)
 % 求模糊传递包
while  sum(sum(R~=mhhc(R,R)))   %调用模糊合成函数‘ mhhc’
   R=mhhc(R,R);
end
s=R;
```

注：此函数被‘程序_3’调用

程序_3

```
function    [s1,s2,s3]=mhjl(x)
% x为原始数据矩阵, 行为分类对象, 列为性状指标
%返回值s1表示模糊矩阵,s2 表示模糊相似矩阵,s3 表示模糊等价矩阵
[m,n]=size(x);
x0=sum(x)/m;
for   j=1:n
  s1(j)= sqrt(sum((x(:,j)-x0(j)).^2)/m);                % 对x做平移——标准差变换
  x(:,j)=(x(:,j)-x0(j))/s1(j);
  x1(:,j)=(x(:,j)-min(x(:,j)))/(max(x(:,j))-min(x(:,j)));
                                                        %平移——极差变换
end
 s1=x1;                                  % s1   表示模糊矩阵
 R=eye(m);
 M=0;                                     %相似系数r 由数量积法求得
  for   i=1:m
        for   j=i+1:m
              if  (sum(x1(i,:).*x1(j,:))>M);
            M=sum(x1(i,:).*x1(j,:));
              end
        end
end
  for   i=1:m
        for   j=1:m
              if  (i~=j)
            R(i,j)=(sum(x1(i,:).*x1(j,:)))/M;
              end
        end
   end
s2=R;                    % R为模糊相似矩阵
```

s3=mhcdb(R);　　% s3表示模糊等价矩阵，此处调用‘ mhcdb’求模糊传递包

注：本程序中若想用“ 夹角余弦法 ” 求相似系数r, 可将上面程序中的第14行（M=0;）至第28行（倒数第3行）用下面的程序段替换.

```
for   i=1:m                    %夹角余弦法求相似系数r
    for   j=1:m
    M1=sqrt(sum(x1(i,:).^2)*sum(x1(j,:).^2));
        R(i,j)=(sum(x1(i,:).*x1(j,:)))/M1;
        end
   end
```

程序 _4

```
function    [L1,s]=Lamjjz(x,lam)
% 求λ- 截矩阵并完成聚类,x 为模糊等价矩阵
%(即程序_3中求得的s3),lam 为待输入的λ值
n=length(x(1,:));
for   i=1:n
        for   j=1:n
            if   x(i,j)>=lam
              L1(i,j)=1;                % x1为λ- 截矩阵
            end
        end
  end
A=zeros(n,n+1);
for   i=1:n
       if   ~A(i,1)
A(i,2)=i;                 % A的第一列为标示符其值为0或 1
        for   j=i+1:n
            if   x1(i,:)==x1(j,:)
        A(i,j+1)=j;
        A(j,1)=1;
```

```
                end
            end
        end
end
for   i=1:n
      if   ~A(i,1)
a=[];
     for   j=2:n+1
          if   A(i,j)
     a=[a,A(i,j)];                    % a表示聚类数组
          end
     end

disp(a)                               %将聚类数组依次显示
end
```

5.　层次分析——求近似特征向量算法

在层次分析中, 该算法用于根据成对比较矩阵求近似特征向量 .

```
function    [w,lam,CR]=ccfx(A)
% A为成对比较矩阵, 返回值   w为近似特征向量,
% lam 为近似最大特征值   λmax ,CR 为一致性比率
n=length(A(:,1));
a=sum(A);
B=A;                                  % 用 B 代替 A 做计算
for   j=1:n                           % 将 A 的列向量归一化
   B(:,j)=B(:,j)./a(j);
end
s=B(:,1);
for   j=2:n
```

```
    s=s+B(:,j);
end
c=sum(s);                                  % 和法计算近似最大特征值   λmax
w=s./c;
d=A*w;
lam=1/n*sum((d./w));
CI=(lam-n)/(n-1);                          %一致性指标
RI=[0,0,0.58,0.90,1.12,1.24,1.32,1.41,1.45,1.49,1.51];
                                           % RI 为随机一致性指标
CR=CI/RI(n);                               % 求一致性比率
if   CR>0.1
   disp(       ' 没有通过一致性检验 ' );
else   disp(    ' 通过一致性检验 ' );
end
```

6. 灰色关联性分析——单因子情形

当系统的行为特征只有一个因子 x_0, 该算法用于求解各种因素 x_i 对 x_0 的影响大小.

```
function   s=Glfx(x0,x)              % x0(行向量) 为因子,x   为因素集
[m,n]=size(x);
B=[x0;x];
k=m+1;                               % k 为 B的行数
c=B(:,1);                            %  对序列进行无量纲化处理
  for   j=1:n
     B(:,j)=B(:,j)./c;
  end
for   i=2:k                          % 求参考序列对各比较序列的绝对差
    B(i,:)=abs(B(i,:)-B(1,:));
end
```

```
A=B(2:k,:);                          % 求关联系数
a=min(min(A));
b=max(max(A));
for   i=1:m
        for   j=1:n
        r1(i,j)=r1(i,j)*(a+0.5*b)/(A(i,j)+0.5*b);
        end
end
s=1/n*(r1*ones(m,1));                %比较序列对参考序列   x0 的灰关联度
```

7. 灰色预测——GM(1,1)

该算法用灰色模型中的 GM(1,1)模型做预测.

```
function   [s,t]=huiseyc(x,m)
% x 为待预测变量的原值, 为其预测    m个值
[m1,n]=size(x);
if   m1~=1                %若 x 为列向量, 则将其变为行向量放入    x0
   x0=x';
else
   x0=x;
end
n=length(x0);
c=min(x0);
if   c<0                  %若 x0 中有小于  0 的数, 则作平移, 使每个数字都大于    0
   x0=x0-c+1;
end
x1=(cumsum(x0))';        % x1 为 x0 的 1 次累加生成序列, 即   AGO
for   k=2:n
   r(k-1)=x0(k)/x1(k-1);
end
rho=r,                        %   光滑性检验
```

```
for   k=2:n
    z1(k-1)=0.5*x1(k)+0.5*x1(k-1);
end
B=[-z1',ones(n-1,1)];
YN=(x0(2:n))';
a=(inv(B'*B))*B'*YN;
y1(1)=x0(1);
for   k=2:n+m                %       预测   m个值
    y1(k)=(x0(1)-a(2)/a(1))*exp(-a(1)*(k-1))+a(2)/a(1);
end
y(1)=y1(1);
for   k=2:n+m
    y(k)=y1(k)-y1(k-1);              % 还原
end
if   c<0
    y=y+c-1;
end
y;
e1=x0-y(1:n);
e=e1(2:n),                          % e   为残差
for   k=2:n
    dd(k-1)=abs(e(k-1))/x0(k);
end
dd;
d=1/(n-1)*sum(dd);
f=1/(n-1)*abs(sum(e));
s=y;
t=e;
```

注：以上程序实例仅供参考练习, 要想使自己的编程水平得到根本性的提高，

除了学习相关知识和经验外, 更重要的是, 一定要自己主动去编写程序, 可先从编写一些较简单的程序入手, 然后逐渐增大难度 ,多加练习, 善于摸索一些特殊问题的处理方法和技巧, 或者通过查书、 与别人讨论等方式来丰富自己的经验, 提升自己的编程能力. 更重要的是, 要对自己有信心, 勇于跨过那个门坎 ,就能进入一种更高的境界！

参考文献

[1] 谭浩强,C 程序设计题解与上机指导（第二版）, 清华大学出版社,2000.

[2] 严蔚敏, 吴伟民, 数据结构（C 语言版）, 清华大学出版社,2006.

[3] 姜启源, 大学数学实验， 清华大学出版社,2005.

[4] 《运筹学》教材编写组， 运筹学， 清华大学出版社,2005.

[5] 徐翠微, 孙绳武, 计算方法引论, 高等教育出版社,2007.

附 录

数据 1 公交线路信息
线路 1
219-114-88-48-392-29-36-16-312-19-324-20-314-128-76-113-110-213-14-301-115-34-251-95-184-92
线路 2
348-160-223-44-237-147-201-219-321-138-83-161-66-129-254-331-317-303-127-68
线路 3
23-133-213-236-12-168-47-198-12-236-113-212-233-18-127-303-117-231-254-129-366-161-133-181-132
线路 4
201-207-177-144-223-216-48-42-280-140-238-236-158-53-93-64-130-77-264-208-286-123
线路 5
217-272-173-25-33-76-37-27-65-274-234-221-137-306-162-84-325-97-89-24
线路 6
301-82-79-94-41-105-142-118-130-36-252-172-57-20-302-65-32-24-92-218-31
线路 7
184-31-69-179-84-212-99-224-232-157-68-54-201-57-172-22-36-143-218-129-106-101-194
线路 8
57-52-31-242-18-353-33-60-43-41-246-105-28-33-111-77-49-67-27-8-63-39-317-168-12-163

线路 9
217-161-311-25-29-19-171-45-71-173-129-219-210-35-83-43-139-241-78-50
线路 10
136-208-23-117-77-130-68-45-53-51-78-241-139-343-83-333-190-237-251-2
91-129-173-171-90-42-179-25-311-161-17
线路 11
43-77-111-303-28-65-246-99-54-37-303-53-18-242-195-236-26-40-280-142
线路 12
274-302-151-297-329-123-122-215-218-102-293-86-15-215-186-213-105-128
-201-122-12-29-56-79-141-24-74
线路 13
135-74-16-108-58-274-53-59-43-86-85-47-246-108-199-296-261-203-227-14
6
线路 14
224-22-70-89-219-228-326-179-49-154-251-262-307-294-208-24-201-261-19
2-264-146-377-172-123-61-235-294-28-94-57-226-18
线路 15
189-170-222-24-92-184-254-215-345-315-301-214-213-210-113-263-12-167-
177-313-219-154-349-316-44-52-19
线路 16
233-377-327-97-46-227-203-261-276-199-108-246-227-45-346-243-59-93-27
4-58-118-116-74-135

表 1 学生作答记录数据

问题 1	问题 2	问题 3	问题 4	问题 5	问题 6	问题 7	问题 8	问题 9	问题 10
1	1	4	3	2	4	4	3	3	3
2	1	2	2	1	3	2	2	3	1
4	4	4	4	4	4	4	1	1	1
3	1	2	4	4	3	3	3	24	2
2	2	2	3	2	3	34	3	2	1
3	1	2	3	2	3	23	4	4	1
2	3	3	2	2	3	2	4	3	2
4	4	3	4	3	4	3	4	1	1
4	3	3	2	2	3	3	4	23	2
2	1	4	3	2	3	2	23	4	2
3	3	2	3	2	3	2	3	4	2
4	2	3	4	2	3	34	3	1	3
2	3	3	3	3	2	2	2	2	2
1	1	4	2	1	4	2	3	34	1
2	2	4	2	2	3	3	12	4	3
2	2	4	2	2	3	23	2	4	2
2	1	4	4	4	4	4	4	1	1

3	2	4	3	2	3	13	2	3	3
1	1	4	2	2	3	4	3	4	3
4	2	4	4	2	4	4	1	2	23

表 2　教室相关数据

教室	座位数	灯管数	开关数	一个开关控制的灯管数	灯管的功率/每只
1 64		42	3	14	40w
2 88		42	3	14	40w
3 193		48	4	12	50w
4 193		50	5	10	48w
5 128		36	2	18	45w
6 120		36	2	18	45w
7 120		36	4	9	48w
8 120		36	3	12	45w
9 110		36	3	12	40w
10 120		36	4	9	45w
11 64 27			3	9	40w
12 247		75	5	15	45w
13 190		48	3	16	48w
14 210		50	5	10	50w
15 70 42			3	14	40w
16 85 42			3	14	40w
17 192		48	4	12	50w
18 195		50	5	10	48w
19 128		36	2	18	45w
20 120		36	2	18	45w
21 120		36	4	9	48w
22 120		36	3	12	45w
23 110		36	3	12	40w
24 160		36	4	9	45w
25 70 27			3	9	40w
26 256		75	5	15	45w
27 190		48	3	16	48w
28 210		50	5	10	50w
29 190		48	3	16	48w
30 205		50	5	10	50w
31 110		36	3	12	40w
32 160		36	4	9	45w
33 70 27			3	9	40w
34 256		75	5	15	45w

35 190		48	3	16	48w
36 210		50	5	10	50w
37 190		48	3	16	48w
38 190		48	3	16	48w
39 210		50	5	10	50w
40 200		48	3	16	48w
41 150		50	5	10	50w
42 150		48	3	16	48w
43 180		48	3	16	48w
44 70 25			5	5	50w
45 120		45	3	15	48w