中国科学技术大学
University of Science and Technology of China

程序设计II

第3讲 简单程序设计

计算机学院 黄章进

zhuang@ustc.edu.cn

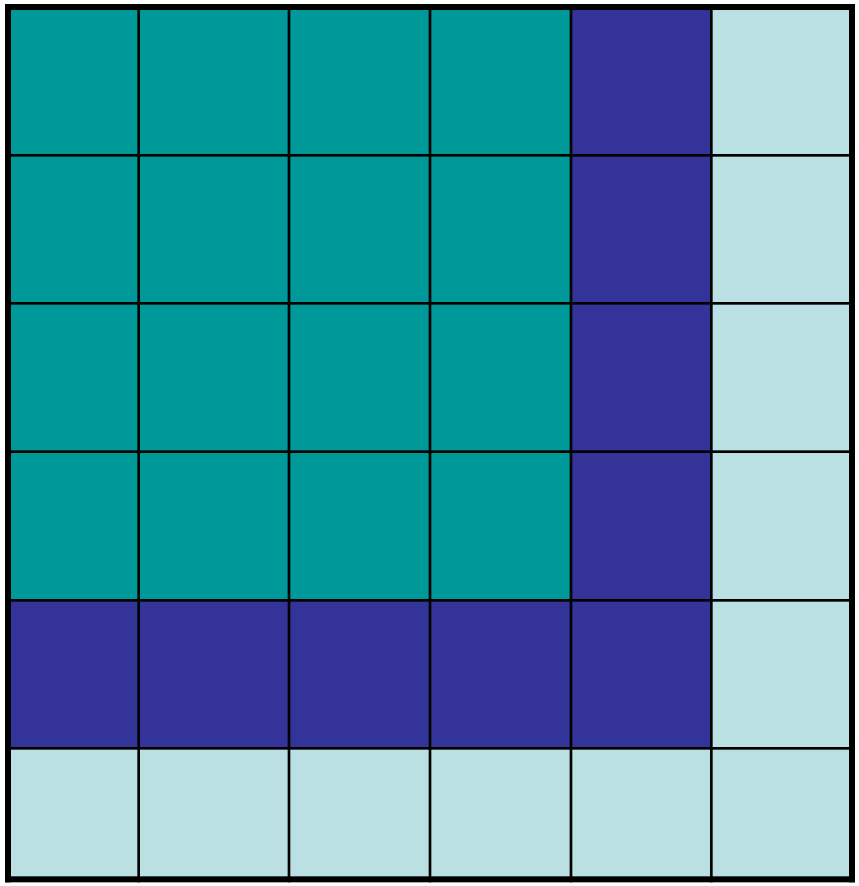
内容

- 例题1：装箱问题 1017
- 例题2：校门外的树 2808
- 例题3：生理周期 2977
- 例题4：确定进制 2972
- 例题5：日历问题 2964
- 例题6：棋盘上的距离 1657

例题1：装箱问题

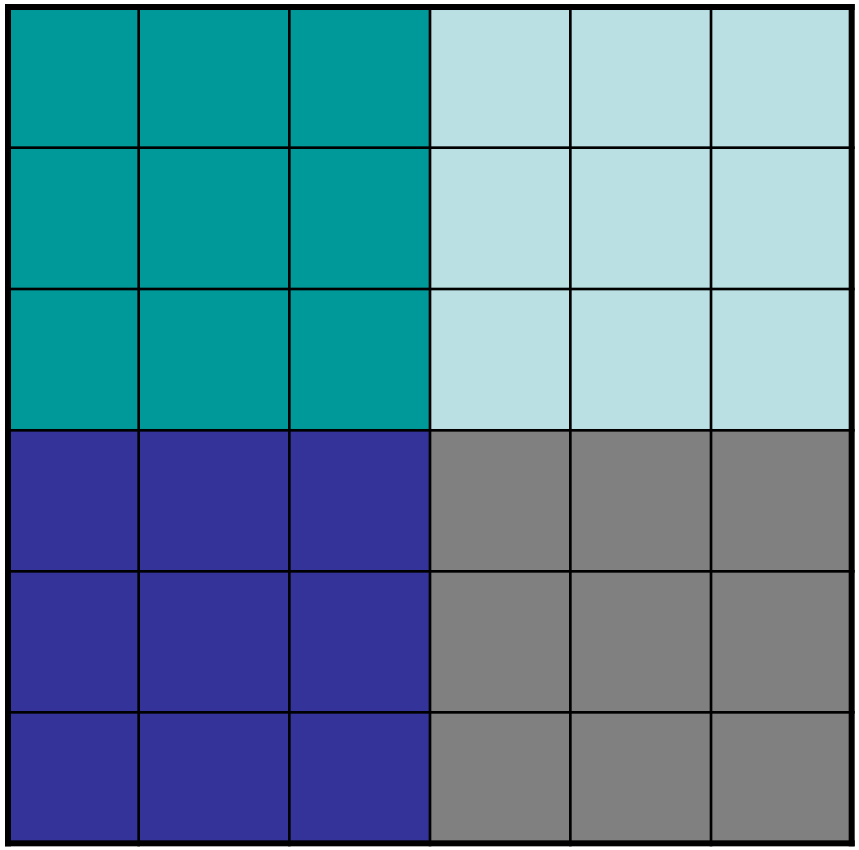
- 题意
  - 已知：有6*6 的大箱子和 1*1，2*2，3*3，4*4，5*5，6*6 的木块，箱子高度和木块一样
  - 问：给定各种木块的数目，求最少需要多少个大箱子来装
- 例如：
  - 输入：0 0 4 0 0 1 -〉 输出 2
  - 输入：7 5 1 0 0 0 -〉 输出 1
  - 输入：0 0 0 0 0 0 程序结束
- 解题思想： 先放大的，后放小的

Packets



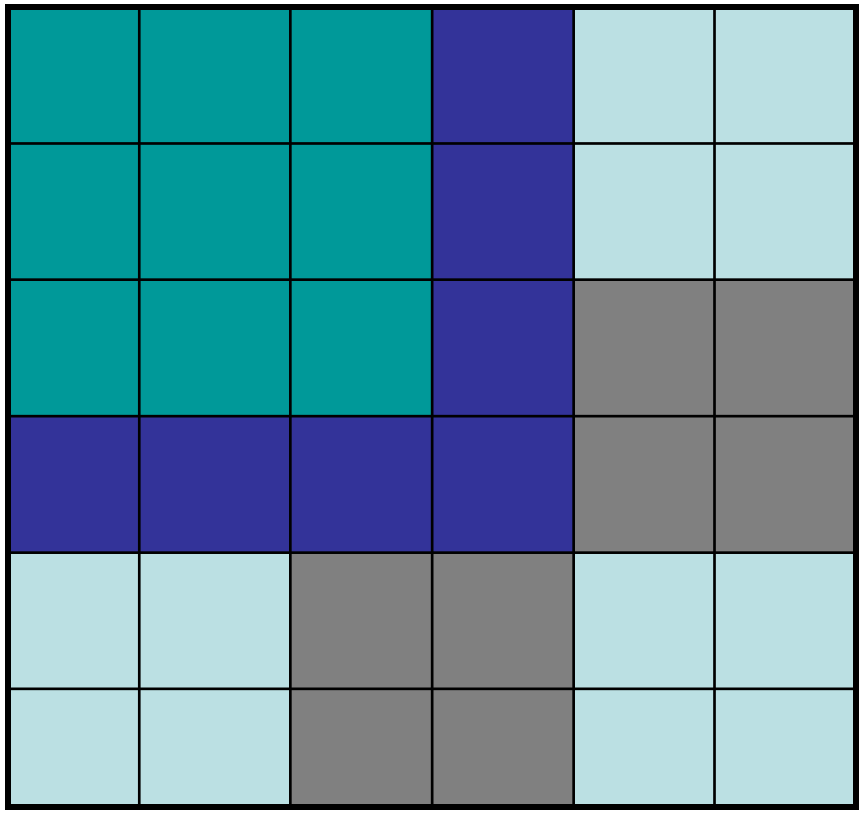
- 6*6的块单独占一个箱子
- 5*5的块单独占一个箱子，还能放11个1*1木块
- 4*4的块单独占一个箱子，还能放5个2*2木块

装箱问题



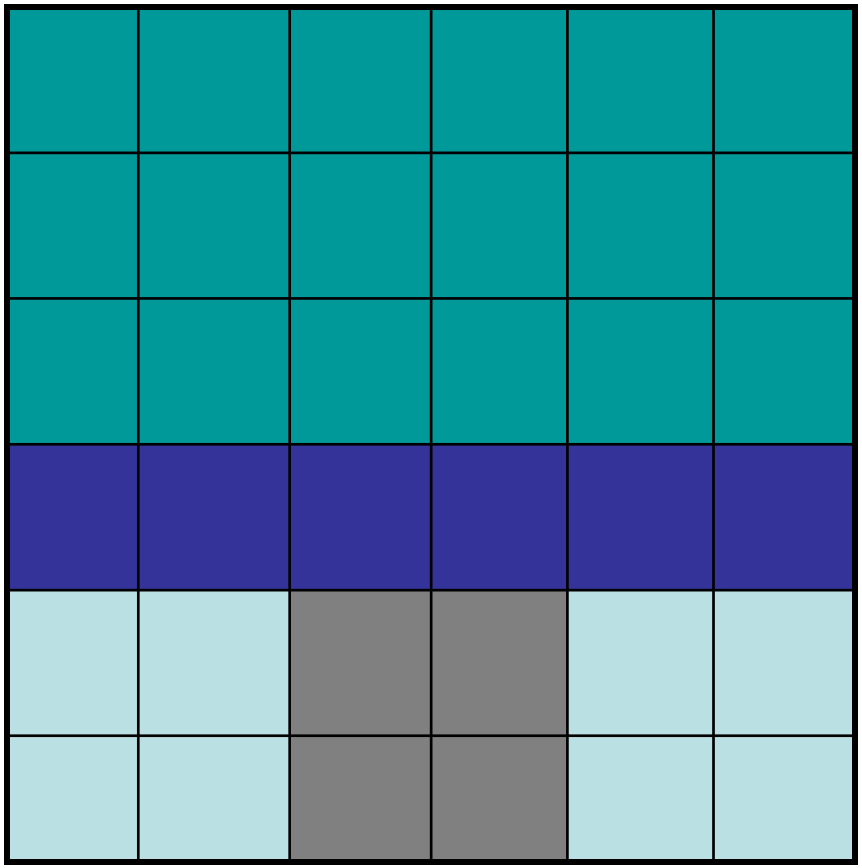
- 3*3的块，每4块占一个箱子,余下的再占一个箱子

装箱问题



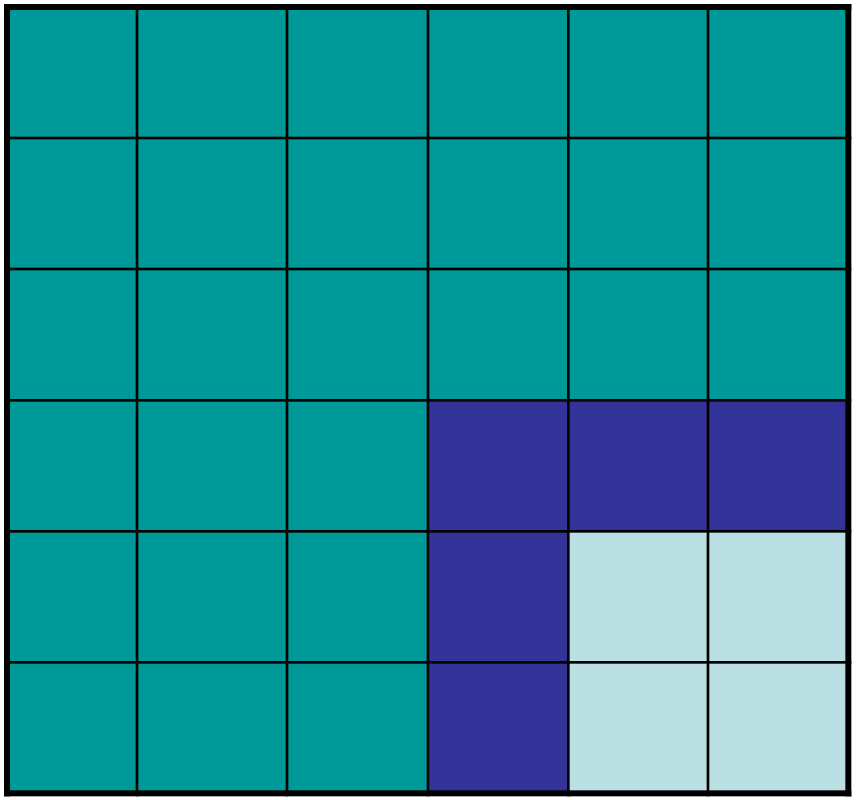
- 如果箱子里放1个3*3木块，那么还能放
- 5个2*2木块,以及7个1*1木块

装箱问题



- 如果箱子里放2个3*3木块，那么还能放3个2*2木块,以及6个1*1木块

装箱问题



- 如果箱子里放3个3*3木块，那么还能放1个2*2木块,以及5个1*1木块

装箱问题

- 解题思想： 先放大的，后放小的
  1. 6*6的木块每个占用一个新箱子；
  2. 5*5的木块每个占用一个新箱子，余下11个1*1的空格；
  3. 4*4的木块每个占用一个新箱子，余下5个2*2的空格；
  4. 3*3的木块每4个占用新一个箱子，不足4个也占一个新箱子，不同情况余下不同数目的空格；
  5. 2*2的木块先填空格，空格不足开新箱子，每9个2*2的木块占一个新箱子；
  6. 1*1的木块先填空格，空格不足开新箱子，每36个占一个新箱子。

装箱问题

- 参数定义：

6*6，5*5，4*4，3*3，2*2，1*1

个数：b6　b5　b4　b3　b2　b1

需用箱子数：nTotal

1）先放好所有 6*6, 5*5, 4*4 和3*3的木块

nTotal = b6 + b5 + b4 + (b3+3)/4

– 4*4, 5*5, 6*6 单独开箱子
– 3*3　每4个占一个箱子，余下的占一个箱子

装箱问题

2) 再把2 * 2的塞到放有3*3木块的箱子里

设一个数组：

int Contain2[4] = { 0, 5, 3, 1 };

Contain2[i] 表示当3*3木块的数目除以4的余数分别是0,1,2,3时，会产生多少个能放2*2木块的空格。

用数组纪录某些事实，比写 if else 方便。

放完2*2的木块后，再算有多少1*1的空格，能否把1*1的木块都填进去，如果不能，计算还要加多少个箱子

装箱问题

3) 计算放好6*6,5*5,4*4,3*3后留下多少空格能放2*2木块

```
c2 = 5 * b4 + Contain2[b3 % 4];
```

4) 在放好2*2的木块后，算留下多少空格能放1*1木块

```
c1 = 36 * nTotal - 36 * b6 - 25 * b5 - 16 * b4 - 9 * b3 - 4 * b2;
```

1017 装箱问题


```
#include <stdio.h>
int main()
{
   int b6,b5,b4,b3,b2,b1; //不同大小的木块个数
   int nTotal = 0; //最少需要的箱子数目
   int c1;  //当前能放 1*1 木块的空格数目
   int c2;  //当前能放 2*2 木块的空格数目
   int Contain2[4] = { 0, 5, 3, 1 };

   while(1){
      scanf("%d%d%d%d%d%d", &b1, &b2, &b3, &b4, &b5, &b6);
      if (b1 == 0 && b2 == 0 && b3 == 0 && b4 == 0 && b5 == 0 && b6 == 0)     break;

      nTotal = b6 + b5 + b4 + (b3 + 3)/4;   //小技巧：(b3+3)/4 正好等于b4除以4向上取整的结果
      c2 = 5 * b4 + Contain2[b3 % 4];
      if (b2 > c2)          nTotal += (b2 - c2 + 8) / 9;
      c1 = 36 * nTotal - 36 * b6 - 25 * b5 - 16 * b4 - 9 * b3 - 4 * b2;
      if (b1 > c1)          nTotal += (b1 - c1 + 35) / 36;
      printf("%d\n", nTotal);
   }

   return 0;
}
```

便于调试测试数据的办法

- 先在本地机器上调试程序，输入测试数据，当能得到正确运行结果后，才将程序提交到oj中。
- 使用freopen函数可以解决测试数据输入问题，避免重复输入

FILE *freopen(const char *path, const char *mode, FILE *stream);

- 参数说明：
  - path: 文件名，用于存储输入输出的自定义文件名。
  - mode: 文件打开的模式。和fopen中的模式（如r-只读, w-写）相同。
  - stream: 一个文件，通常使用标准流文件。
- 返回值：成功，则返回一个path所指定文件的指针；失败，返回NULL。（一般可以不使用它的返回值）

便于调试测试数据的办法

- freopen功能：实现重定向，把预定义的标准流文件定向到由path指定的文件中。标准流文件具体是指stdin、stdout和stderr。
  - stdin是标准输入流，默认为键盘；
  - stdout是标准输出流，默认为屏幕；
  - stderr是标准错误流，一般把屏幕设为默认。

便于调试测试数据的办法


```
#include <stdio.h>

int main()
{
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);

    /* 同控制台输入输出 */

    fclose(stdin);
    fclose(stdout);

    return 0;
}
```

```
#ifdef ONLINE_JUDGE
#else
    freopen("in.txt", "r", stdin);
#endif
```

可以不使用输出重定向，仍然在控制台查看输出。
程序调试成功后，提交到oj时不要忘记把与重定向有关的语句注释或删除掉。

例题2：校门外的树

- 某校大门外长度为L的马路上有一排树，每两棵相邻的树之间的间隔都是1米。
- 可以把马路看成一个数轴，马路的一端在数轴0的位置，另一端在L的位置；数轴上的每个整数点，即0，1，2，……，L，都种有一棵树。
- 由于马路上有一些区域要用来建地铁。这些区域用它们在数轴上的起始点和终止点表示。已知任一区域的起始点和终止点的坐标都是整数，区域之间可能有重合的部分。
- 现在要把这些区域中的树（包括区域端点处的两棵树）移走。你的任务是计算将这些树都移走后，马路上还有多少棵树。

校门外的树

- 输入数据
  - 输入的第一行有两个整数L（1 <= L <= 10000）和M（1 <= M <= 100），L代表马路的长度，M代表区域的数目，L和M之间用一个空格隔开。接下来的M行每行包含两个不同的整数，用一个空格隔开，表示一个区域的起始点和终止点的坐标。
- 输入样例

500 3

150 300

100 200

470 471

校门外的树

- 输出要求
  - 输出包括一行，这一行只包含一个整数，表示马路上剩余的树的数目。
- 输出样例

298

校门外的树

- 思路一
  - 开一个有L+1个元素的数组，每个元素对应一棵树，全部初始化为1，表示各个位置上都有树
  - 然后每读入一个区间，就将该区间对应的数组元素都变成0，表示该区间的树都被砍了
  - 最后算一下还有几个1，就是还剩几棵树了

2808 校门外的树


```
#include <stdio.h>
#include <stdbool.h>  // bool类型支持
int main()
{
   int L, i, j, n; //L 为区间的长度，n 为区间的个数，i 和 j 是循环变量。
   bool trees[10001]; //  用一个布尔数组模拟树的存在情况。
   for (i = 0; i < 10001; i++) //  赋初值
      trees[i] = true;
   scanf("%d%d", &L, &n);
   for (i = 0; i < n; i++) {
      int begin, end;  //  用 begin,end 存储区间的起止位置。
      scanf("%d%d", &begin, &end);
      for (j = begin; j <= end; j++)  //  将区间内的树移走，即赋值为 false。
         trees[j] = false;
   }
   int count = 0; //用 count 计数，数数剩余的树的数目。
   for (i = 0; i <= L; i++)
      if (trees[i])  count ++;
   printf("%d\n", count);
   return 0;
}
```

校门外的树

- 使用变长数组(C99，POJ不支持)

```
#include <stdio.h>
#include <stdbool.h>
int main()
{
    int L, i, j, n; //L 为区间的长度，n 为区间的个数，i 和 j 是循环变量。
    scanf("%d%d", &L, &n);
    bool trees[L+1]; //  用一个布尔数组模拟树的存在情况。
    for (i = 0; i <= L; i++) //  赋初值
        trees[i] = true;
    // 其他代码同前
    return 0;
}
```

校门外的树

- 思路二
  - 将区间按起点排序，然后把所有区间遍历一遍，就把所有的树都砍了。
  - 不用开设L+1个元素的数组，但是要开设数组将所有区间的起点，终点保存下来
- 思路三
  - 区间合并

例题3：生理周期

- 人生来就有三个生理周期，分别为体力、感情和智力周期，它们的周期长度为23天、28天和33天。
- 每一个周期中有一天是高峰。在高峰这天，人会在相应的方面表现出色。例如，智力周期的高峰，人会思维敏捷，精力容易高度集中。
- 因为三个周期的长度不同，所以通常三个周期的高峰不会落在同一天。
- 对每个周期，我们都给出了某一个高峰出现的日子（用高峰日是当年的第几天表示，1月1日算第0天)。

生理周期

- 现给定一个日子（也用当年的第几天表示），要求我们输出从给定日子开始（不包括给定日子）下一次三个高峰落在同一天的时间（用距给定日子的天数来表示）。
  - 例如：给定日子为当年第10天，如果下次出现三个高峰同日的时间是当年第12天，则输出2（注意这里不是3）。

生理周期

- 输入
  - 输入四个整数：p, e, i和d。
  - p, e, i分别表示体力、情感和智力高峰出现的时间（即第p天，第e天和第i天）。
  - d 是给定的日子（第d天），可能小于p, e, 或 i 。 d 是非负的并且小于365, 数据保证所求的日子会在第21252天前。数据以一行4个-1结束。
- 输出
  - 从给定日子d起，下一次三个高峰同天的日子距离给定日子的天数。

生理周期

- 问题分析
  - 设所求的时间为第x天，则x具有如下性质：

  1) d < x <= 21252

  2) (x-p)%23=0

  3) (x-e)%28=0

  4) (x-i)%33=0
- 简单思路
  - 一个最简单直观的做法就是枚举从d+1 到21252 之间所有的数字，寻找第一个满足条件2）3）4）的数字，注意输出时间减去d

生理周期

- 可以做的进一步改进是从d+1开始逐一枚举寻找满足条件2 的数a；
- 从a开始每步加23寻找满足条件3的数b（这样的b自然也满足条件2)；
- 然后再从b开始每步加23*28寻找满足条件4的数x(这样的x同时满足条件2,3)。
- x就是我们要找的数，输出时输出x-d。

生理周期

- 程序设计

// 读入p, e, i, d
// j从d+1 循环到21252，
　　如果 (j-p)%23==0，跳出循环
// j从上次跳出循环的值循环到21252，步长为23
　　如果 (j-e)%28==0，跳出循环
// j从上次跳出循环的值循环到21252，步长为23*28
　　如果 (j-i)%33==0，跳出循环
// 输出j-d

2977 生理周期


```
#include <stdio.h>
int main()
{
    int p, e, i, d, j, no = 1;
    scanf("%d%d%d%d", &p, &e, &i, &d);
    while (p != -1 && e != -1 && i != -1 && d != -1) {
        for(j = d+1; j <= 21252; j++)
            if ((j-p) % 23 == 0) break;
        for( ; j <= 21252; j = j + 23)
            if ((j-e) % 28 == 0) break;
        for( ; j <= 21252; j = j + 23*28)
            if ((j-i) % 33 == 0) break;
        printf("Case %d", no);
        printf(": the next triple peak occurs in %d days.\n", j - d);
        scanf("%d%d%d%d", &p, &e, &i, &d);
        no++;
    }
    return 0;
}
```

例题4：确定进制

- 6*9 = 42 对于十进制来说是错误的，但是对于13进制来说是正确的。即, 6(13) * 9(13) = 42(13)， 而 $42(13) = 4 * 13^1 + 2 * 13^0 = 54(10)$。
- 我们的任务是写一段程序读入三个整数p、q和 r，然后确定一个进制 B(2<=B<=16) 使得 p * q = r.

2972 确定进制

- 如果没有合适的进制，则输出 0。
- 如果 B有很多选择, 输出最小的一个。
  - 例如： p = 11, q = 11, r = 121.
  - 11(10) * 11(10) = 121(10)
  - 11(3) * 11(3) = 121(3)
    - 因为 $11(3) = 1 * 3^1 + 1 * 3^0 = 4(10)$ 和 $121(3) = 1 * 3^2 + 2 * 3^1 + 1 * 3^0 = 16(10)$。
  - 这种情况下，应该输出 3。

确定进制

- 输入
  - 输入有 T组测试样例， T在第一行给出。每一组测试样例占一行，包含三个整数p、q、r。 p、q、r的所有位都是数字，并且1 <= p、q、r <= 1,000,000。
- 输入样例

3

6 9 42

11 11 121

2 2 2

确定进制

- 输出
  - 对于每个测试样例输出一行。该行包含一个整数：即使得p * q = r成立的最小的B。如果没有合适的B，则输出 0。

- 输出样例

13

3

0

确定进制

- 解题思路
  - 此问题很简单。选择一个进制 B，按照该进制将被乘数、乘数、乘积分别转换成十进制。
  - 然后判断等式是否成立。使得等式成立的最小 B 就是所求的结果。

2972 确定进制


```
#include <stdio.h>
#include <string.h>
int  b2ten(int x,int b);   // 计算b进制数x对应的十进制值
int  main( )
{
    int p, q, r, n, b;
    scanf("%d", &n);
    while (n--) {
        scanf("%d%d%d", &p, &q, &r);
        for (b = 2; b <= 16; b++) {
            int p2 = b2ten(p, b);
            int q2 = b2ten(q, b);
            int r2 = b2ten(r, b);
            if  (p2==-1 || q2==-1 || r2==-1) continue;
            if  (p2 * q2 == r2) {    printf("%d\n",b);  break;     }
        }
        if (b == 17) printf("0\n");
    }
    return 0;
}
```

2972 确定进制


```
int b2ten(int x, int b)
{
    char tmp[100];
    int ret = 0;

    sprintf(tmp, "%d", x);
    int len = strlen(tmp);
    int i;
    for (i = 0; i < len; i++) {        // POJ不支持for语句头中声明变量
        if (tmp[i] - '0' >= b)    return -1;   // x不是b进制数
        ret *= b;
        ret += tmp[i] - '0' ;
    }

    return ret;
}
```

例题5：日历问题

- 在我们现在使用的日历中, 闰年被定义为能被4整除的年份，但是能被100整除而不能被400整除的年是例外，它们不是闰年。
  - 例如：1700, 1800, 1900 和 2100 不是闰年，而1600, 2000 和 2400是闰年。
- 给定从公元2000年1月1日开始逝去的天数，给出这一天是哪年哪月哪日星期几。

日历问题

- 输入
  - 输入包含若干行，每行包含一个正整数，表示从2000年1月1日开始逝去的天数。
  - 输入最后一行是−1, 不必处理。假设结果的年份不会超过9999
- 输入样例

1730
1740
1750
1751
-1

日历问题

- 输出
  - 对每个测试样例，输出一行，该行包含对应的日期和星期几。
  - 格式为“YYYY-MM-DD DayOfWeek”, 其中“DayOfWeek”必须是下面中的一个： "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday" 或 "Saturday“
- 输出样例

2004-09-26 Sunday

2004-10-06 Wednesday

2004-10-16 Saturday

2004-10-17 Sunday

日历问题

- 解题思路
  - 确定星期几
    - 知道 2000 年 1 月 1 日是星期几后，只要用给定的日期对 7 取模
  - 确定年月日
    - 输入一个整数 n，如果 n 大于等于一年的天数，就用 n 减去一年的天数，直到 n 比一年的天数少（这时假设剩下天数为 m）
    - 如果 m 大于等于一个月的天数，就用 m 减去一个月的天数，直到 m 比一个月的天数少（这时假设剩下的天数为 k）
    - 这时 k 为从当月开始逝去的天数

2964 日历问题


```
#include <stdio.h>

int type(int ); //判断是否为闰年的函数

char week[7][10] = {"Saturday", "Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday"};

//year[0]表示非闰年的天数，year[1]表示闰年的天数
int year[2] = {365, 366};

//month[0]表示非闰年里每月的天数，month[1]表示闰年里每月的天数
int month[2][12] = {31,28,31,30,31,30,31,31,30,31,30,31,
                    31,29,31,30,31,30,31,31,30,31,30,31
                   };
```

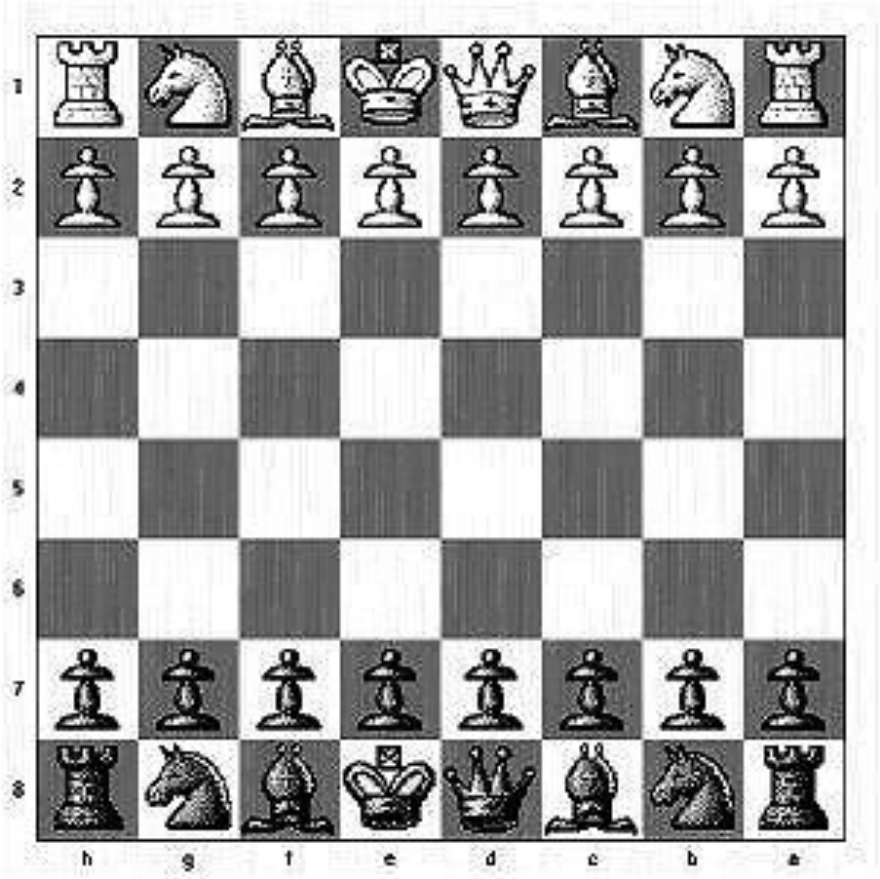
2964 日历问题


```
int  main()
{
   //days 表示输入的天数，dayofweek表示星期几。
   int days, dayofweek;
   int i = 0, j = 0;

   while (scanf("%d", &days) && days != -1) {
      dayofweek = days % 7;
      for (i = 2000; days >= year[type(i)]; i++)
         days -= year[type(i)];
      for (j = 0; days >= month[type(i)][j]; j++)
         days -= month[type(i)][j];
      printf("%d-%02d-%02d %s\n",
            i, j + 1, days + 1, week[dayofweek]);
   }

   return 0;
}
```

2964 日历问题


```
int type(int m)
{
    //判断第m年是否是闰年，是则返回1，否则返回0
    if (m % 4 != 0 || (m % 100 == 0 && m % 400 != 0))
        return  0;  //不是闰年
    else
        return 1;   // 是闰年
}
```

例题6：棋盘上的距离



国际象棋的棋盘是黑白相间的8＊8的方格，棋子放在格子中间。

王、后、车、象的走子规则如下：

- 王：横、直、斜都可以走，但每步限走一格。
- 后：横、直、斜都可以走，每步格数不受限制。
- 车：横、竖均可以走，不能斜走，格数不限。
- 象：只能斜走，格数不限。

写一个程序，给定起始位置和目标位置，计算王、后、车、象从起始位置走到目标位置所需的最少步数

棋盘上的距离

- 输入
  - 第一行是测试数据的组数t（0 <= t <= 20）
  - 以下每行是一组测试数据，每组包括棋盘上的两个位置，第一个是起始位置，第二个是目标位置。
  - 位置用"字母-数字"的形式表示，字母从"a"到"h"，数字从"1"到"8"
- 输入样例

2

a1 c3

f5 f8

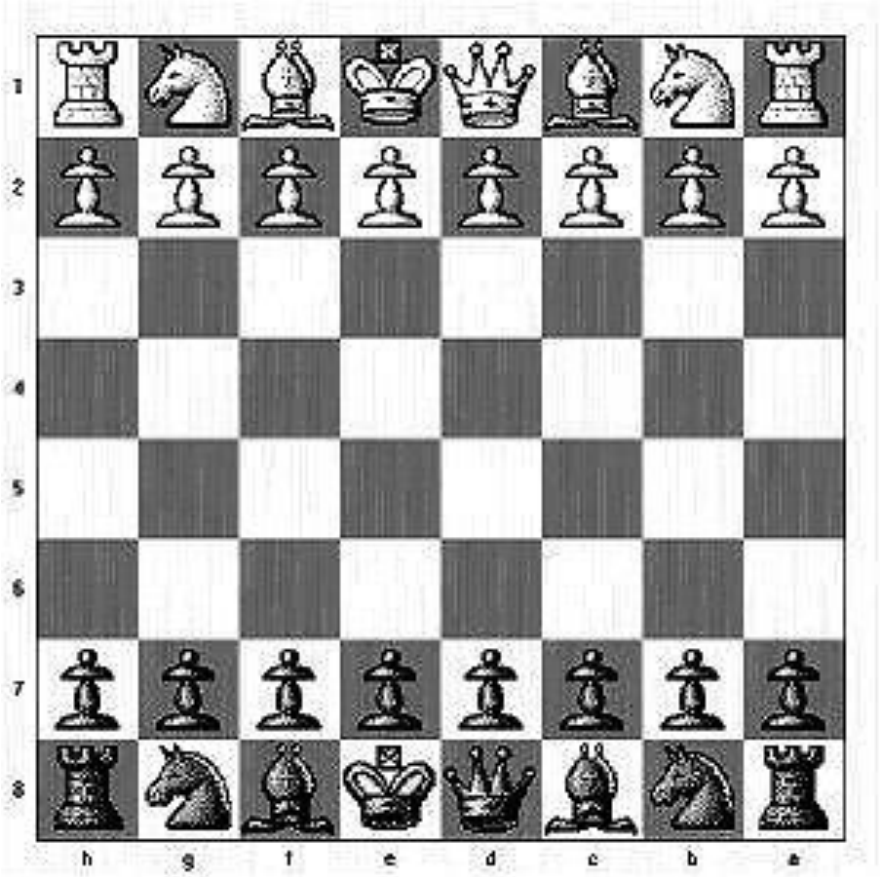
- 输出
  - 对输入的每组测试数据，输出王、后、车、象所需的最少步数。
  - 如果无法到达，就输出"Inf"
- 输出样例

2 1 2 1

3 1 1 Inf

棋盘上的距离

- a1 c3

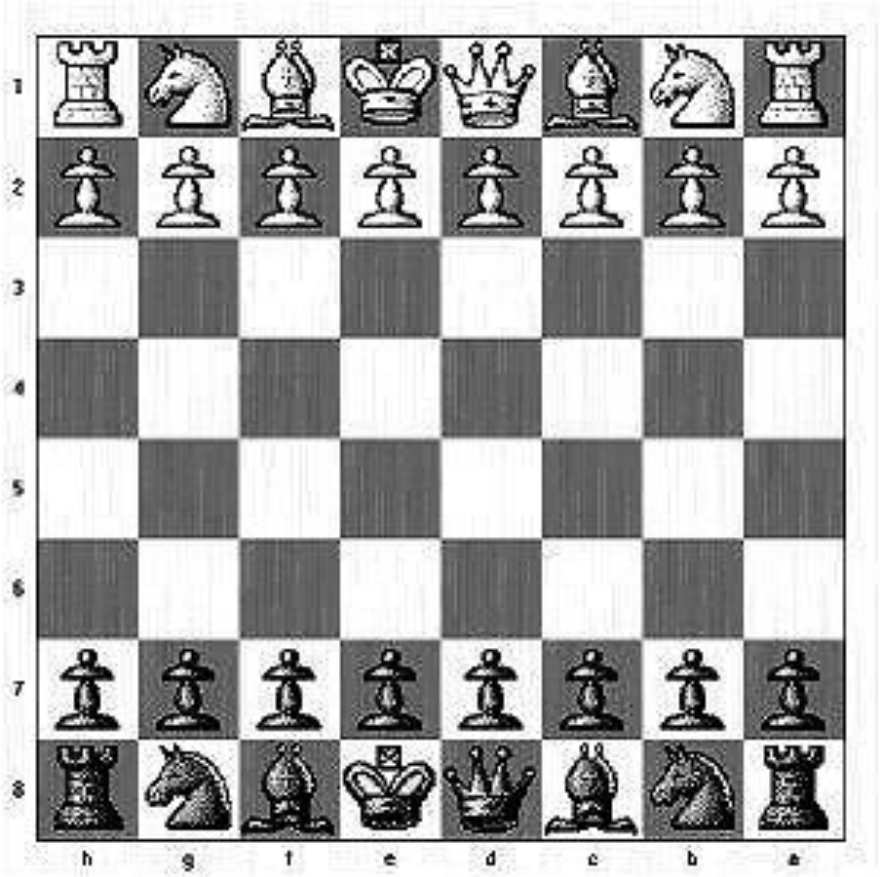


- 假设起止位置在水平和竖直方向的距离分别是x和y
- 王：横、直、斜都可以走，但每步限走一格
- 最少步数是

min(x,y)+abs(x-y) =max(x,y)

棋盘上的距离

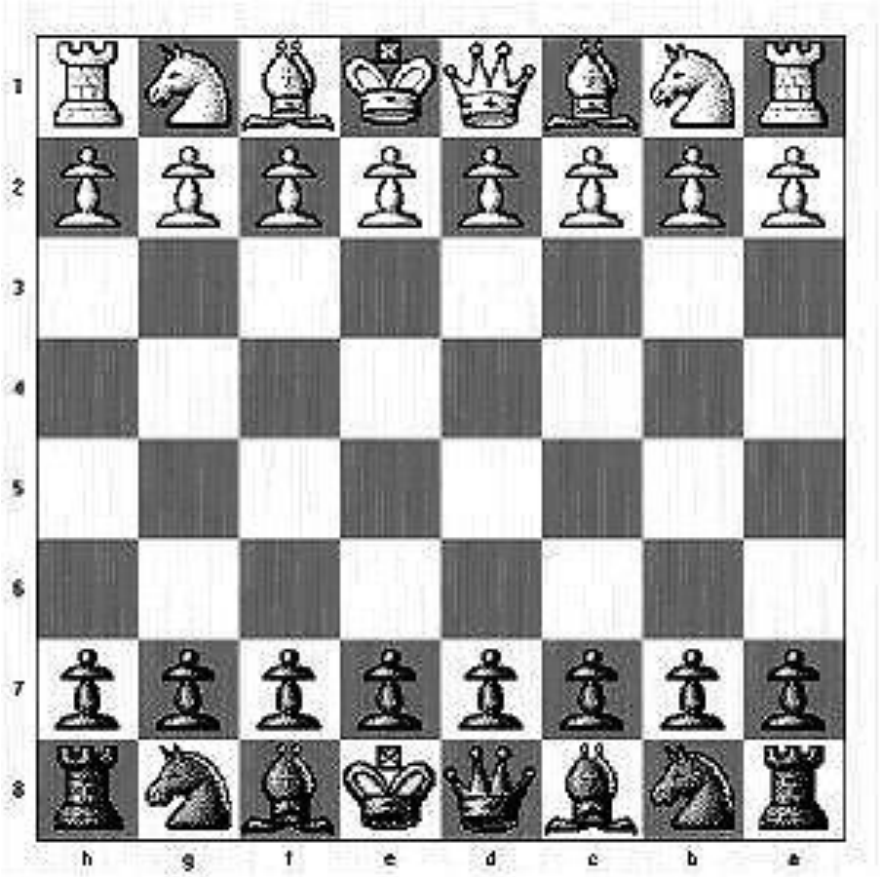
- a1 c3



- 后：横、直、斜都可以走，每步格数不受限制。
- 最少步数是
  1：x==y || x==0 || y==0
  2：其他

棋盘上的距离

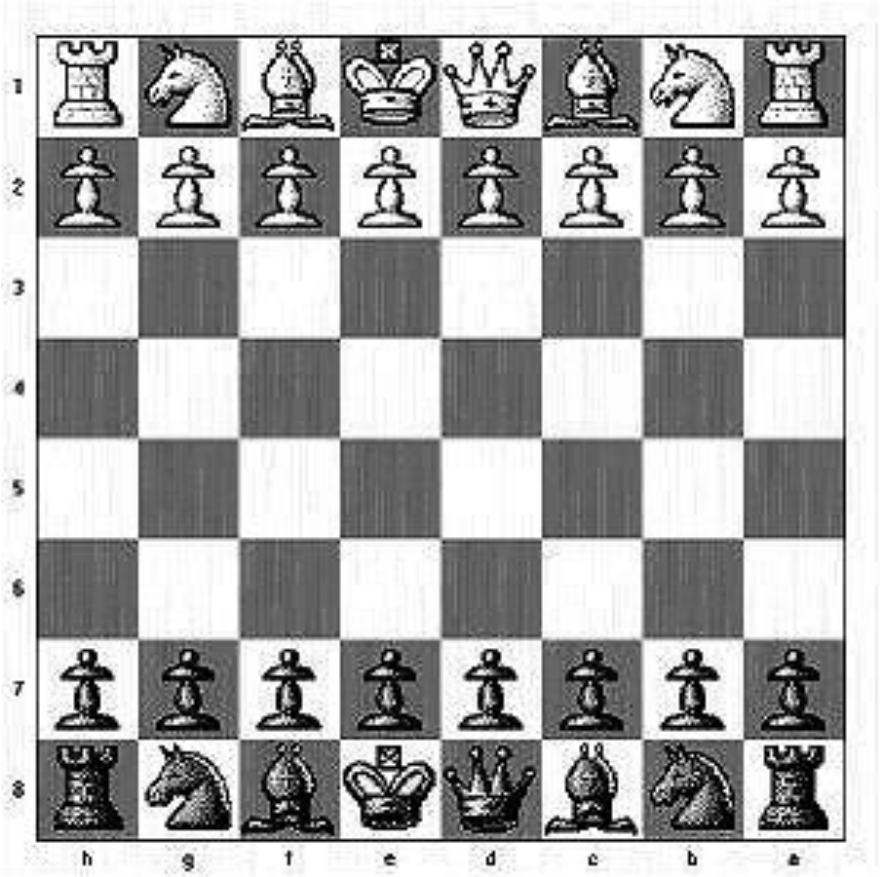
- a1 c3



- 车：横、竖均可以走，不能斜走，格数不限
- 最少步数是
  1：x==0 || y==0
  2：其他

棋盘上的距离

- a1 c3



- 象：只能斜走，格数不限

- 最少步数是

  Inf：abs(x-y) % 2 != 0

  1：x == y

  2：x != y

1657 棋盘上的距离


```
#include <stdio.h>
#include <stdlib.h>    /* abs */

int main( )
{
    int nCases, i;
    scanf("%d", &nCases);
    for(i = 0; i < nCases; i++) {
        char begin[3], end[3]; //用 begin 和 end 分别存储棋子的起止位置。
        scanf("%s %s", begin, end);

        int x, y; //用 x 和 y 分别存储起止位置之间 x 方向和 y 方向上的距离
        x = abs(begin[0] - end[0]);
        y = abs(begin[1] - end[1]);
```

1657 棋盘上的距离


```
if (x == 0 && y == 0) printf("0 0 0 0\n"); //起止位置相同
        else {
            if (x < y) printf("%d", y);      //  王的步数
            else    printf("%d", x);
            if (x == y || x == 0 || y == 0) printf(" 1");//  后的步数
            else    printf(" 2");
            if (x == 0 || y == 0)    printf(" 1"); //  车的步数
            else            printf(" 2");
            if (abs(x - y) % 2 != 0)    printf(" Inf\n"); //  象的步数
            else if (x == y)     printf(" 1\n");
            else                printf(" 2\n");
        }
    }
    return 0;
}
```

作业

- 2807：两倍
  - 给定2到15个不同的正整数，计算这些数里面有多少个数对满足：数对中一个数是另一个数的两倍。
- 2800：垂直直方图
  - 输入4行全部由大写字母组成的文本，输出一个垂直直方图，给出每个字符出现的次数
- 2967：特殊日历计算