15. 基础4——例题练习与讲解

授课教师：游伟 副教授、孙亚辉 副教授、王文轩 讲师

授课时间：周二14:00 – 15:30，周四14:00 – 15:30（教学一楼1603）

上机时间：周四18:00 – 21:00（明德楼地下机房F）

课程主页：https://www.youwei.site/course/programming

YOJ-147. 教室排课

题目描述

信息学院有四个专业A、B、C、D，各专业入学新生人数分别是Na, Nb, Nc，Nd人。新学期开始有一门公共课，按专业划分成四个教学班，四个班在某个相同的时间段上课。已知该时间段还剩余8间教室可用，编号从1到8，每个教室能容纳的人数分别为120，40，85，50，100，140，70，100。试编一个程序，为上述四个教学班分配教室。　　找出所有可行的分配方案，对于每个方案依次输出为专业A、B、C、D分配的教室编号，按照字典顺序输出所有方案。

输入格式

一行，包含4个整数Na, Nb, Nc，Nd (20≤Na, Nb, Nc，Nd≤120)，每2个整数之间用一个空格隔开。

输出格式

如果存在分配方案，输出若干行，每行表示一种教室分配方案，包含4个整数，依次表示A、B、C、D四个专业分配的教室编号。
注意：按照字典序输出所有方案。
如果不存在分配方案，输出-1。

输入样例

109 87 120 81

输出样例

1 5 6 3
1 5 6 8
1 8 6 3
1 8 6 5
6 5 1 3
6 5 1 8
6 8 1 3
6 8 1 5

YOJ-147. 教室排课

- 基本思路：
  - 遍历所有可能：因为教室数量少（8 间），可以通过四层循环遍历所有可能的 4 间教室组合（i、j、k、l 分别代表 A、B、C、D 的教室编号）。
  - 筛选有效方案：对每个组合，检查：
  - 四个编号是否互不相同（教室不重复）。
  - 每个教室的容量是否≥对应专业的人数。
  - 按字典序输出：由于循环是从 1 到 8 依次遍历，自然会按字典序（先比较第一个编号，再第二个，以此类推）输出方案。
  - 判断无方案情况：若没有符合条件的组合，输出 -1

YOJ-147. 教室排课

```
int main() {
    bool f = false;  // 标记是否有可行方案，初始为false
    int a, b, c, d;  // 存储四个专业的人数
    // 教室容量数组：e[1]是1号教室容量，e[2]是2号教室容量，以此类推
    int e[] = {0, 120, 40, 85, 50, 100, 140, 70, 100};
    // 输入四个专业的人数
    cin >> a >> b >> c >> d;
    // 四层循环遍历所有可能的教室分配（i:A, j:B, k:C, l:D）
    for (int i = 1; i <= 8; i++) {  // A专业的教室编号i
        for (int j = 1; j <= 8; j++) {  // B专业的教室编号j
            for (int k = 1; k <= 8; k++) {  // C专业的教室编号k
                for (int l = 1; l <= 8; l++) {  // D专业的教室编号l
                    // 检查四个教室编号是否互不相同（不能重复使用教室）
                    if (i == j || i == k || i == l || j == k || j == l || k == l)
                        continue;  // 有重复则跳过当前组合
                    // 检查每个教室容量是否足够（教室容量≥对应专业人数）
                    if (a <= e[i] && b <= e[j] && c <= e[k] && d <= e[l]) {
                        // 输出可行方案
                        cout << i << " " << j << " " << k << " " << l << endl;
                        f = true;  // 标记存在可行方案
                    }
                }
            }
        }
    }
    // 如果没有找到任何可行方案，输出-1
    if (!f)
        cout << "-1" << endl;
    return 0;
}
```

YOJ-134：猜数字

题目描述

猜数字游戏是文曲星上的一款打发时间的小游戏。游戏的规则是这样的：计算机随机产生一个四位数，然后玩家猜这个四位数是什么。每猜一个数，计算机都会告诉玩家猜对几个数字，其中有几个数字在正确的位置上。　　比如计算机随机产生的数字为1122。如果玩家猜1234,因为1,2这两个数字同时存在于这两个数中，而且1在这两个数中的位置是相同的，所以计算机会告诉玩家猜对了2个数字，其中一个在正确的位置。如果玩家猜1111,那么计算机会告诉他猜对2个数字，有2个在正确的位置。　　现在给你一段猜数字的过程，你的任务是根据这段对话确定这个四位数是什么。

输入格式

　　输入第一行为一个正整数N(1<=N<=10)，表示在这段对话中共有N次问答（不允许出现重复问答）。在接下来的N行中，每行三个整数A,B,C。游戏者猜这个四位数为A，然后计算机回答猜对了B个数字，其中C个在正确的位置上。

输出格式

　　一行，如果根据这段对话能确定这个四位数，则输出这个四位数，若不能，则输出"Not sure"。

输入样例

```
6
4815 2 1
5716 1 0
7842 1 0
4901 0 0
8585 3 3
8555 3 2
```

输出样例

```
3585
```

YOJ-134：猜数字

- 基本思路：
  - 枚举+验证
  - 枚举 1000-9999，依次验证是否满足所给 n 个条件
- 注意事项：
  - 预处理：将输入的和待验证的四位数处理成四位单独的数字
  - 最终输出：答案唯一时输出答案，0个或多个时均需要输出“Not sure”
  - 如何计算猜对的数字个数（不要求位置相同）：

    例：两数 a = 1123 , b = 1113

    猜对的 1 的个数由 1 较少的 1123 决定

    所以 猜对的数字个数 = 1 较少的数中1的个数 + 2较少的数中2的个数 + 3较少的数中3的个数 + …

    = 2 + 0 + 1 + …

    = 3

YOJ-134：猜数字

方法1-主函数部分-读入和预处理

```
// 全局变量定义
int n,                  // 问答次数N（1<=N<=10）
    a[20][20],          // 存储每个猜测数字的各位数字，a[i][0]-a[i][3]表示第i个猜测的个位到千位
    b[20],              // 存储每个猜测的B值（猜对的数字个数）
    c[20],              // 存储每个猜测的C值（位置正确的数字个数）
    cnt[20][20];        // 统计每个猜测中数字0-9的出现次数，cnt[i][j]表示第i个猜测中数字j出现的次数
int main()
{
    int t,              // 临时存储输入的猜测数字
        ans = -1;       // 存储最终答案，-1表示未找到
    cin >> n;           // 读取问答次数N

    // 读取并处理N次问答数据
    for(int i = 1; i <= n; i++)
    {
        scanf("%d %d %d", &t, &b[i], &c[i]);  // 读取猜测数字A、B值、C值

        // 将猜测数字x分解为各位数字，并统计各数字出现次数
        for(int j = 0; j <= 3; j++){
            a[i][j] = t % 10;           // 存储各位数字（从个位开始存储）
            cnt[i][a[i][j]]++;          // 对应数字计数加1
            t /= 10;                    // 去掉最低位，处理下一位
        }
    }
```

YOJ-134：猜数字

方法1-主函数部分-枚举+输出

```
39      bool book = 0;  // 标记是否已找到有效解（0未找到，1已找到）
40      int x[4];       // 存储候选数字t的各位数字（个位、十位、百位、千位）
41
42      // 枚举所有可能的四位数 1000-9999
43      for(x[0] = 0; x[0] <= 9; x[0]++)
44      {
45          for(x[1] = 0; x[1] <= 9; x[1]++)
46          {
47              for(x[2] = 0; x[2] <= 9; x[2]++)
48              {
49                  for(x[3] = 1; x[3] <= 9; x[3]++)
50                  {
51                      if(check(x))
52                      {  // 检查当前数字i是否满足所有条件
53                          if(book)
54                          {
55                              // 如果已找到过一个解，又找到第二个解，则解不唯一
56                              printf("Not sure");
57                              return 0;
58                          }
59                          book = 1;  // 标记已找到解
60                          ans = x[3] * 1000 + x[2] * 100 + x[1] * 10 + x[0];// 记录当前解
61                      }
62                  }
63              }
64          }
65      }
66      // 输出结果
67      if(ans == -1)
68          printf("Not sure");  // 未找到任何解
69      else
70          printf("%d", ans);   // 输出唯一解
71      return 0;
```

YOJ-134：猜数字

方法1-验证函数部分-预处理

```
77  bool check(int x[])// 存储候选数字t的各位数字（个位、十位、百位、千位）
78  {
79      int cnt1[20],               // 存储候选数字t中各数字(0-9)的出现次数
80          b1, c1;                 // 计算出的B值（猜对数字个数）和C值（位置正确个数）
81
82      for(int i = 0; i < 20; i++)  // 初始化cnt1数组为0
83          cnt1[i] = 0;
84
85      // 统计候选数字各数字出现次数，用于计算B值
86      for(int i = 0; i <= 3; i++)cnt1[x[i]]++;       // 对应数字计数加1
```

YOJ-134：猜数字

方法1-验证函数部分-检查是否满足条件

```
86      // 检查候选数字是否满足所有N次问答的条件
87      for(int i = 1; i <= n; i++){
88          b1 = c1 = 0;  // 初始化B值和C值计算器
89
90          // 计算B值：猜对的数字个数（不考虑位置）
91          // 方法：对于每个数字0-9，取其在猜测和候选数字中出现次数的较小值，然后求和
92          for(int j = 0; j <= 9; j++)
93              b1 += min(cnt[i][j], cnt1[j]);
94
95          // 计算C值：位置正确的数字个数
96          // 方法：逐位比较猜测数字与候选数字的对应位置
97          for(int j = 0; j <= 3; j++)
98              c1 += (a[i][j] == x[j]);  // 位置相同则计数加1
99
100         // if(x[0]==0&&x[1]==1&&x[2]==9&&x[3]==9){
101         //     printf("%d %d\n",b1,c1);
102         // }
103
104         // 如果与任一问答条件不符，则该候选数字无效
105         if(b1 != b[i] || c1 != c[i])
106             return false;
107     }
108
109     return true;  // 通过所有检查，候选数字有效
```

YOJ-134：猜数字

方法2-主函数部分-读入和预处理（同方法1）

```
// 全局变量定义
int n,                  // 问答次数N（1<=N<=10）
    a[20][20],          // 存储每个猜测数字的各位数字，a[i][0]-a[i][3]表示第i个猜测的个位到千位
    b[20],              // 存储每个猜测的B值（猜对的数字个数）
    c[20],              // 存储每个猜测的C值（位置正确的数字个数）
    cnt[20][20];        // 统计每个猜测中数字0-9的出现次数，cnt[i][j]表示第i个猜测中数字j出现的次数
int main()
{
    int x,              // 临时存储输入的猜测数字
        ans = -1;       // 存储最终答案，-1表示未找到
    cin >> n;           // 读取问答次数N

    // 读取并处理N次问答数据
    for(int i = 1; i <= n; i++)
    {
        scanf("%d %d %d", &x, &b[i], &c[i]);  // 读取猜测数字A、B值、C值

        // 将猜测数字x分解为各位数字，并统计各数字出现次数
        for(int j = 0; j <= 3; j++){
            a[i][j] = x % 10;           // 存储各位数字（从个位开始存储）
            cnt[i][a[i][j]]++;          // 对应数字计数加1
            x /= 10;                    // 去掉最低位，处理下一位
        }
    }
```

YOJ-134：猜数字

方法2-主函数部分-枚举+输出（枚举方式不同于方法1）

```
    bool book = 0;  // 标记是否已找到有效解（0未找到，1已找到）

    // 枚举所有可能的四位数 1000-9999
    for(int i = 1000; i <= 9999; i++)
    {
        if(check(i))
        {  // 检查当前数字i是否满足所有条件
            if(book)
            {
                // 如果已找到过一个解，又找到第二个解，则解不唯一
                printf("Not sure");
                return 0;
            }
            book = 1;  // 标记已找到解
            ans = i;   // 记录当前解
        }
    }

    // 输出结果
    if(ans == -1)
        printf("Not sure");  // 未找到任何解
    else
        printf("%d", ans);   // 输出唯一解

    return 0;
}
```

YOJ-134：猜数字

方法2-验证函数部分-预处理（需要对t进行拆分）

```
67  bool check(int t)
68  {
69      int cnt1[20],               // 存储候选数字t中各数字(0-9)的出现次数
70          x[4],                   // 存储候选数字t的各位数字（个位、十位、百位、千位）
71          b1, c1;                 // 计算出的B值（猜对数字个数）和C值（位置正确个数）
72
73      for(int i = 0; i < 20; i++)  // 初始化cnt1数组为0
74          cnt1[i] = 0;
75
76      // 将候选数字t分解为各位数字，并统计各数字出现次数
77      for(int i = 0; i <= 3; i++){
78          x[i] = t % 10;          // 获取当前最低位数字
79          cnt1[x[i]]++;           // 对应数字计数加1
80          t /= 10;                // 去掉最低位，处理下一位
81      }
```

YOJ-134：猜数字

方法2-验证函数部分-检查是否满足条件（同方法1）

```
83      // 检查候选数字t是否满足所有N次问答的条件
84      for(int i = 1; i <= n; i++){
85          b1 = c1 = 0;  // 初始化B值和C值计算器
86
87          // 计算B值：猜对的数字个数（不考虑位置）
88          // 方法：对于每个数字0-9，取其在猜测和候选数字中出现次数的较小值，然后求和
89          for(int j = 0; j <= 9; j++)
90              b1 += min(cnt[i][j], cnt1[j]);
91
92          // 计算C值：位置正确的数字个数
93          // 方法：逐位比较猜测数字与候选数字的对应位置
94          for(int j = 0; j <= 3; j++)
95              c1 += (a[i][j] == x[j]);  // 位置相同则计数加1
96
97          // 如果与任一问答条件不符，则该候选数字无效
98          if(b1 != b[i] || c1 != c[i])
99              return false;
100     }
101
102     return true;  // 通过所有检查，候选数字有效
103 }
```

YOJ-290：购物车共同性

by张涵

```
/*总体思路:以第一组数据为基准,如果这组数据中的一个元素总共
出现了n次,那么这个元素就是我们要找的元素,再从小到大排序*/
#include<stdio.h>
#include<stdlib.h>//里面有qsort函数
int f(const void*a,const void*b){
    return *(int*)a-*(int*)b;
}//制作快排函数
int main(void){
    int n;
    scanf("%d",&n);//输入n
    int a1;
    scanf("%d",&a1);
    int t[a1][2];
    for(int i=0;i<a1;i++){
        scanf("%d",*(t+i));//将第一行的元素读如数组
        t[i][1]=1;//将出现次数初始化为1
    }
    for(int i=0;i<n-1;i++){//遍历剩下每一行元素
        int j;scanf("%d",&j);
        for(int k=0;k<j;k++){
            int temp;scanf("%d",&temp);
            for(int u=0;u<a1;u++){
                if(temp==t[u][0]){//如果找到了就将出现次数加1
                    t[u][1]++;
                }
            }
        }
    }
    int answer[a1];int sum=0;
    for(int i=0;i<a1;i++){
        if(t[i][1]==n)answer[sum++]=t[i][0];//如果某个元素出现了n次,就将它存入结果数组
    }
    if(sum==0)printf("NO\n");//如果结果数组为空,输出NO
    else{
        qsort(answer,sum,sizeof(int),f);//如果数组不为空,对结果数组进行排序
        for(int i=0;i<sum;i++){
            printf("%d ",answer[i]);//打印结果数组
        }
        printf("\n");
    }
    return 0;
}//ZhangH
```

by李昊森

YOJ-291：大整数加减

- 题目描述：

比利经常会碰到超大整数的加减法运算，而普通的计算器上无法进行。因此他想你帮他写一个程序来计算结果。

输入格式

输入共三行。 第一行为一个符号，表示要进行的计算（“+”表示要进行加法运算， “-”表示要进行减法运算）。 第二行第三行每行为一个大整数（长度小于2000），表示要进行加减操作的两个数。注意：整数既可以为正数也可以为负数。

输出格式

加减运算后的结果。

YOJ-291：大整数加减

基本思路：

- 1. 首先加法和减法就是一念之间的事情，我们完全可以把减法改成加(-b)
- 2. 其实加法无非就是四种情况，左右正负2*2拼起来，可以直接分类讨论，并且有两种情况是对称的
- 3. 那就分出来两种情况：正+正，正+负（正-正）

- 顺带提一句我当时这个题是c++写的我直接把代码搬过来了，你们真的想写C的话我就说一下咋写

YOJ-291：大整数加减

提前处理

```
cin >> a;
cin >> b >> c;
reverse(b.begin(), b.end()), reverse(c.begin(), c.end());
// 将b和c从头到尾翻转过来，因为输入的时候是从高位到低位输入的，第0位对应的
最高位这样不利于进位
if (a == '-') {
    if (c[c.size() - 1] == '-')
        c.pop_back();  // 扔掉最后一个位置
    else
        c.push_back('-');  // 最后的地方放上一个负号
}  // 减法变加法
x = b.size(), y = c.size();
int f1 = (b[b.size() - 1] == '-'), f2 = (c[c.size() - 1] == '-');
// cout << f1 << ' ' << f2 << endl;
```

YOJ-291：大整数加减

正+正

```
if (!f1 && !f2) {
    for (int i = 0; i < x; i++)
        l[i] = b[i] - '0';
    for (int i = 0; i < y; i++)
        r[i] = c[i] - '0';
    // char 转 int
    int fin = 0;
    for (int i = 0; i <= max(x, y); i++) {
        ans[i] += l[i] + r[i];
        // 注意+=
        if (ans[i] >= 10)
            ans[i] -= 10, ans[i + 1]++;
        if (ans[i])
            fin = i;
        // 进位，fin表示最后有多少位
    }
    for (int i = fin; i >= 0; i--)
        cout << ans[i];
}
```

YOJ-291：大整数加减

正+负（正-正）

```
else if (!f1 && f2) {
    for (int i = 0; i < x; i++)
        l[i] = b[i] - '0';
    for (int i = 0; i < y - 1; i++)
        r[i] = c[i] - '0';
    y--;            // 只取绝对值，去掉负号
    int flag = 0;  // 判断两个数的绝对值大小
    if (y > x)
        flag = 1;
    else if (x == y) {
        for (int i = x - 1; i >= 0; i--)
            if (l[i] < r[i]) {
                flag = 1;
                break;
            }
            else if (r[i] < l[i])
                break;
    }  // 比较两个数大小的时候先判断位数，再判断字典序（从高到低位，如果出现不相等的位置就按这个位置判断大小）
    if (flag)
        swap(l, r);
    // 如果绝对值b<c就换一下，因为我现在求|b-c|
```

YOJ-291：大整数加减

正+负（正-正）

```
    if (flag)
        swap(l, r);
    // 如果绝对值b<c就换一下，因为我现在求|b-c|
    int fin = 0;
    for (int i = 0; i <= max(x, y); i++) {
        ans[i] += l[i] - r[i];
        if (ans[i] < 0)
            ans[i] += 10, ans[i + 1]--;
        if (ans[i])
            fin = i;
    }
    if (flag)
        cout << '-';
    // 对于绝对值来说，判断正负
    for (int i = fin; i >= 0; i--)
        cout << ans[i];
}
```

by吴致楷

YOJ-296：班会时间

【问题描述】

由于班上同学所选课程和上课时段各不相同。为了找到一个合适的时间开班会，班长小A收集了班上所有同学的课表。

请你编写程序，帮助小A处理所有同学的课表，并得到开班会的合适时间段，时间段按照中国人民大学的课表划分（每个时间段的表示法如下表所示）。要求程序根据输入的所有课表，统计出每个时间段有课的学生人数，并将时间段按照上课人数从小到大排序，如果某两个时间段的上课人数一样，则按照时间段的先后顺序排序（时间在前的排在前面），输出有课人数前k（1≤≤k≤49）少的时间段。

【输入格式】

第一行包含2个正整数n和k，分别表示班级的人数n和题目中要求的k。

接下来的n行，每行表示一个学生的课表，具体格式如下：

201710123410 2.1 1.2 4.3 2.4 2.5 5.1 5.2 1.6 4.2 4.4

节次	时间	周一	周二	周三	周四	周五	周六	周日
1	8:00-9:30	1.1	2.1	3.1	4.1	5.1	6.1	7.1
2	10:00-11:30	1.2	2.2	3.2	4.2	5.2	6.2	7.2
3	12:00-13:30	1.3	2.3	3.3	4.3	5.3	6.3	7.3
4	14:00-15:30	1.4	2.4	3.4	4.4	5.4	6.4	7.4
5	16:00-17:30	1.5	2.5	3.5	4.5	5.5	6.5	7.5
6	18:00-19:30	1.6	2.6	3.6	4.6	5.6	6.6	7.6
7	19:40-21:10	1.7	2.7	3.7	4.7	5.7	6.7	7.7

其中第1个十位整数m表示这个学生的学号，第2个正整数p表示这个学生有p个时间段有课，之后跟着p个小数表示有课的时间段，例如2.1表示周二第1节有课。每2个数字之间用一个空格隔开。

【输出格式】

输出k行，每行包含一个小数和一个整数，分别表示时间段和该时间段有课的学生人数，按照有课人数从小到大的顺序输出前k个，如果某两个时间段的上课人数一样，则时间在前面的先输出。

【输入样例】

3 5

2017101000 10 2.1 1.2 4.3 2.4 2.5 5.1 5.2 1.6 4.2 4.4

2017101001 9 2.1 2.2 2.5 3.2 1.1 1.3 4.3 4.4 5.2

2017101002 10 1.4 1.5 2.1 2.2 3.4 3.4 4.1 4.6 5.4 5.5

【输出样例】

1.7 0

2.3 0

2.6 0

2.7 0

3.1 0

YOJ-296：班会时间

基本思路：

- 统计每个时间段的人数再用qsort排序。
- 那我们一步步来看，在写的过程中会有什么问题。
- 在第一步我们就遇到了一个小问题，类似”3.2”这种东西，我们该怎么读入呢？

- 一种看起来不错的方式是，把他当成小数，用double读入。
- 那读入到double中怎么把两位分离开呢？
- 可以定义int _s = s * 10。这样就变成了两位数，要分离两位数的十位，则可以int x = _s/10，个位则int y = _s%10

YOJ-296：班会时间

- 读入完后我们遇到了第二个问题，我们的每个时间有两维：周几，第几节。
- 难道我们要用一个二维数组去存储每个时间的人数吗？
- 其实没必要，写起来也很麻烦，不难发现，对于

1.1	2.1	3.1	4.1	5.1	6.1	7.1
1.2	2.2	3.2	4.2	5.2	6.2	7.2
1.3	2.3	3.3	4.3	5.3	6.3	7.3
1.4	2.4	3.4	4.4	5.4	6.4	7.4
1.5	2.5	3.5	4.5	5.5	6.5	7.5
1.6	2.6	3.6	4.6	5.6	6.6	7.6
1.7	2.7	3.7	4.7	5.7	6.7	7.7

- 我们可以把这49个时间，映射到数字0，1，2，…，48上，映射方式是z = （x - 1）* 7 + （y - 1），可以停下想想为什么是这样映射

YOJ-296：班会时间

不难发现，这样映射的本质就是，按时间的先后顺序，最早的编号为0，然后依次编号2, 3, …, 48。

所以映射后，大小关系不变。也就是说，映射前有(x, y)<(_x, _y)，

映射后就会有z<_z

- 映射完后，我们只需要开一个大小49的数组t[49]，t[i]存储的是时间i的人数。

- 这种把多维映射到一维的方式很常用，且能有效减少代码量。

YOJ-296：班会时间

- 更普遍的，对于两维 x 在 [1,n]中,y 在 [1,m]中。
- 则所有的(x,y)可以被依次编号为z = (x - 1) * m + (y - 1)。

- 按照先比x，再比y的比大小方式，依次被编号为:0，1，2，…，n * m - 1

- 扩展思考：如果对于 x 在 [0,n-1]中,y 在 [0,m-1]中,如何编号。

YOJ-296：班会时间

```
scanf("%d%d",&n,&k);
for(int i = 1;i <= n;++i){
    int id,len;
    scanf("%d%d",&id,&len);//读入学号，以及该同学的总课数
    for(int j = 1;j <= len;++j){
        int x,y;
        double s;//按double读入"x.y"这种格式的输入
        scanf("%lf",&s);
        int _s = s * 10;
        x = _s / 10;
        y = _s % 10;//分离出x和y

        int z = (x - 1) * 7 + (y - 1);//把(x,y)二元组映射到一维z上
        t[z]++;//对应的时间人数加一
    }
}
```

YOJ-296：班会时间

- 然后就到了排序环节，一种方法是把时间i和该时间的人数t[i]放在一个结构体里，不难，但有没有一点更好写的方法？
- 注意到，结构体的排序需要使用自定义一个比大小函数，那对int的排序能不能也让我们自己来比较大小呢？
- 其实是ok的，我们写出这样的一个cmp函数：

```
int cmp(const void* _x,const void* _y){//比较两个时间，谁在前面。
    int x = *(int*)_x;
    int y = *(int*)_y;
    if(t[x] == t[y]){//如果两个时间人数一样，则更前的（编号小的）时间在前面。
        if(x<y)return -1;
        else return 1;
    }
    if(t[x] < t[y]) return -1;//否则，人数小的在前面
    return 1;
}
```

YOJ-296：班会时间

- 接下来，我们用一个数组存下0~48每个时间，然后对它进行排序。

```
for(int i = 0;i < 49;++i) a[i] = i;
qsort(a,49,4,cmp);
```

- 这样，前k个适合开班会的时间就是：a[0], a[1], …, a[k-1]

YOJ-296：班会时间

我们以及知道了前k个适合开班会的时间的编号，现在，我们要把时间转化回两维：（周几，第几节）

- 对于时间z，现在我们想知道(x, y)是多少。
- 不难发现，按照刚刚的映射方式，有：
- x-1 = z / 7, y-1 = z % 7;
- 一种理解方式是，把z看成一个7进制数，y-1是个位，x-1是十位。

```
for(int i = 0;i < k;++i){
    int z = a[i];//当前时间z为a[i]
    int x = z / 7 + 1;
    int y = z % 7 + 1;
    printf("%d.%d %d\n",x,y,t[z]);//依次输出x，y和时间z的人数t[z]
}
```

by江易洋

YOJ-486. 碱基串排序

【问题描述】
脱氧核糖核酸（DNA）是由以下四种碱基组成的双螺旋结构：A(ADENINE腺嘌呤)、T(THYMINE胸腺嘧啶)、G(GUANINE鸟嘌呤)、C(CYTOSINE胞嘧啶)。现给定N个长度相同的碱基串（注意：串由多个有序的碱基组成，每个碱基只能为ACGT中的一个），请你对它们按用户指定的优先次序进行排序。
【输入格式】
第1行为一个正整数N，表示需要排序的碱基串的个数。
第2行为用户指定的排序规则，共包含4个字符，为A、C、G、T四个碱基的一个全排列，表示单个碱基的优先次序，出现的越靠前则越优先。根据单个碱基的优先次序，按照如下规则比较两个碱基串a与b：从a和b的第一个碱基开始比较，如果前者比后者优先，则a比b优先；如果后者比前者优先，则b比a优先；如果二者相等，则比较后一个碱基进行判断，以此类推。
接下来N行，每行一个碱基串。注意每个碱基串长度相同。
【输出格式】
共输出N行，每行一个碱基串，按照用户指定的优先规则排好顺序。

YOJ-486. 碱基串排序

- 基本思路：
  - 读入碱基串后进行冒泡排序.
  - 定义函数compare(a, b)返回a和b的顺序, 若a排在b前(a<b)则返回true, 否则返回false.
    - 定义映射表rank[c]表示c碱基的排名.
    - 找到a和b第一个不同的位置i, 比较rank[a[i]]和rank[b[i]].

YOJ-486. 碱基串排序

```cpp
#include <bits/stdc++.h>

int n, m; // 字符串数量和长度
char s[1000][21];
int rank[128]; // 碱基的排名

bool compare(const char *a, const char *b)
{
    for (int i = 0; i < m; ++i) {
        // 找到第一个不同的碱基
        if (a[i] != b[i]) {
            return rank[(int)a[i]] <
rank[(int)b[i]];
        }
    }
    // a 和 b 完全相同
    return false;
}

int main() {
    std::cin >> n;
    char order[5];
    std::cin >> order;
    for (int i = 0; i < 4; ++i) {
        rank[(int)order[i]] = i;
    }

    for (int i = 0; i < n; ++i) {
        std::cin >> s[i];
    }
    m = std::strlen(s[0]);

    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            // s[j] < s[i] 则交换
            if (compare(s[j], s[i])) {
                std::swap(s[i], s[j]);
            }
        }
    }

    for (int i = 0; i < n; ++i) {
        std::cout << s[i] << "\n";
    }

    return 0;
}
```

by张梓桐

YOJ-506. 排值日表

- 有若干一班的学生，由输入指定其名字。每个名字为中间无空格的拼音，长度小于 64 个字符，不同名字之间有空格。还有若干二班的学生，由输入指定其名字，两组学生的数量相等。为了加强同学间的相互了解，两班各取一名学生组成一个两人值日小组，按照名字的字典序分别排列两组学生名单，然后按照两个序列的顺序，一个一班的学生与一个二班学生组合输出，安排做值日表。
- 学生人数 n 满足 1<=n<=200

YOJ-506. 排值日表

回顾冒泡排序，实际上我们要实现的是两个功能：

- 比较两个字符串的大小（字典序）
- 交换两个字符串

两个字符串 s1, s2 如何比较字典序?

- 设n= min{|s1|, |s2|}，比较两个字符串的前n个字符，如果存在某个字符是不同的，那么字符更小的那个字符串字典序更小。

```
int cmp(char *s1,char *s2)
{
    int l1=strlen(s1),l2=strlen(s2),n=l1<l2?l1:l2;
    for(int i=0;i<n;i++)if(s1[i]!=s2[i])return s1[i]<s2[i];
    return l1<l2;
}
```

YOJ-506. 排值日表

如何交换两个字符串？

- 当然可以一个字符一个字符交换，这里介绍一个更加快速、通用的方法
- 我们不动字符串在数组中的位置，而是构建一个 id[]数组，id[x]表示在排序过程中在位置x的是原数组里的第几个字符串，初始时 id[i]=i
- 可以把 id[]理解成“座位安排表”
- 交换两个字符串时，只需要交换其 id 数组即可

YOJ-506. 排值日表

```
//前面已经初始化了 id1[i]=i
for(int i=1;i<n;i++)
{
    for(int j=1;j<n;j++)
    {
        /*
        字典序：s1[id1[j+1]] < s1[id1[j]]
        根据冒泡排序，此时需要交换这两个位置
        */
        if(cmp(s1[id1[j+1]],s1[id1[j]]))
        {
            int tmp=id1[j+1];
            id1[j+1]=id1[j],id1[j]=tmp;
        }
    }
}
```

by段宇哲

YOJ-668. 变位词

题目让我们求不同的变位词数量一个简单的想法是对于每个新输入的词，判断一遍前面有没有一个词和这个词互为变位词。那么我们需要解决的问题就转变成了如何判断两个词是否互为变位词。

有两种不难想到的可行的思路

- 1.两个词互为变位词当且仅当 26 个小写字母在两个词中出现的次数均相同。
- 2.两个词互为变位词当且仅当这两个词分别按照字母从小到大排序之后相同。

YOJ-668. 变位词

- 这两种思路都不难实现。这里给出基于第一种思路的代码：

```
#include<bits/stdc++.h>
using namespace std;
int N, tot[30][26]; char s[30];
bool check(int x, int y) {
        for(int i=0;i<26;i++)
                if(tot[x][i]!=tot[y][i]) return 0;
        return 1;
}
int main() {
        cin>>N; int ans=0;
        for(int i=1;i<=N;i++) {
                scanf("%s",s+1); int len=strlen(s+1);
                for(int j=1;j<=len;j++) tot[i][s[j]-'a']++;
                bool flag=0;
                for(int j=1;j<i;j++) if(check(i,j)) flag=1;
                if(!flag) ans++;
        } cout<<ans<<endl;
        return 0;
}
```

YOJ-668. 变位词

- 数据范围很小，这样做足够通过。但是大家可能想到，如果我每次输入一个词，都要把前面输入过的所有词扫一遍，效率是很低的。我们能不能做得更好呢?

- 答案是肯定的。我们考虑使用上面的思路1对每次输入的串排序，然后我们就只需要知道每次输入的串排完序后的串是否出现过。如果我们有办法将一个字符串映射成一个整数，每次就相当于只需要查这个整数是否出现过，开一个bool类型的数组存一下就行了。这个办法被称为字符串哈希。

YOJ-668. 变位词

```cpp
#include<bits/stdc++.h>
using namespace std;
int N; map<string,bool> m;
int main() {
        cin>>N; string x; int tot=0;
        for(int i=1;i<=N;i++) {
                cin>>x;
                sort(x.begin(),x.end());
                if(!m[x]) tot++, m[x]=1;
        } cout<<tot<<endl;
        return 0;
}
```

by施涵森

YOJ-669. 矩形相交

- 如何存储矩形：采用结构体，存储其左上角和右下角的坐标。
- 如何将读入转化为存储：很显然，读入两个横坐标的较小值对应 左上角，较大值对应右下角。

```cpp
// 定义矩形结构体，存储规范化后的坐标
// (x1, y1) 为左下角，(x2, y2) 为右上角
struct Rect {
    int x1, y1, x2, y2;
};

// 辅助函数：根据对角线两个点生成规范化矩形
Rect makeRect(int xa, int ya, int xb, int yb) {
    Rect r;
    r.x1 = min(xa, xb);
    r.x2 = max(xa, xb);
    r.y1 = min(ya, yb);
    r.y2 = max(ya, yb);
    return r;
}
```

YOJ-669. 矩形相交

- 不难想到，通过对重叠区域进行分析，来判断输出。
- 求出重叠部分是简单的，只需要对左右上下边界取最大/最小值即 可。

YOJ-669. 矩形相交

- 1. 计算重叠区域的边界
- 重叠区的左边界是两个矩形左边界的较大值
- 重叠区的右边界是两个矩形右边界的较小值
- 上下同理

```
int ix1 = max(r1.x1, r2.x1);
int iy1 = max(r1.y1, r2.y1);
int ix2 = min(r1.x2, r2.x2);
int iy2 = min(r1.y2, r2.y2);
```

YOJ-669. 矩形相交

- 2. 计算重叠区域的宽和高

```
int w = ix2 - ix1;
int h = iy2 - iy1;
```

YOJ-669. 矩形相交

- 无面积重叠的情形
- 情形(1): 完全不相交(宽或高小于0)
- 情形(2): 仅1个顶点相连(宽和高都恰好为0)
- 情形(3): 仅在边上靠在一起(宽或高有一个为0，但不能同时为0)

```
// 情形 (1): 完全不相交 (宽或高小于0)
if (w < 0 || h < 0) return 0;

// 情形 (2): 仅1个顶点相连 (宽和高都恰
if (w == 0 && h == 0) return 1;

// 情形 (3): 仅在边上靠在一起 (宽或高
if (w == 0 || h == 0) return 2;
```

YOJ-669. 矩形相交

- 有面积重叠的情形(w > 0 && h > 0)
- 我们需要判断在X 轴和Y 轴上的包含关系
- 设r1_holds_r2_x 表示：在X轴方向上，矩形1 是否完全包含矩形 2
- Y 轴同理

```
bool r1_holds_r2_x = (r1.x1 <= r2.x1 && r1.x2 >= r2.x2);
bool r2_holds_r1_x = (r2.x1 <= r1.x1 && r2.x2 >= r1.x2);
```

YOJ-669. 矩形相交

- 有面积重叠的情形
- 情形(8): 完全重叠(12)互为包含关系
- 情形(6): 一个在另一个内部(8) r1包含r2，或者r2包含r1
- 情形(7): 十字交叉(10) 一个在X轴包含对方，另一个在Y轴包含对方
- 情形(5): 一条完整的边在内部(6) 如果前面的"完全重叠"、"内部"、 "十字" 都不满足，但仍有任意一个轴存在包含关系，则说明是T 字型或者一边嵌入型
- 思考为什么按照这个顺序判

YOJ-669. 矩形相交

- 有面积重叠的情形

```
if (r1_holds_r2_x && r2_holds_r1_x && r1_holds_r2_y && r2_holds_r1_y) {
    return 12;
}

bool r1_contains_r2 = r1_holds_r2_x && r1_holds_r2_y;
bool r2_contains_r1 = r2_holds_r1_x && r2_holds_r1_y;
if (r1_contains_r2 || r2_contains_r1) {
    return 8;
}

if ((r1_holds_r2_x && r2_holds_r1_y) || (r2_holds_r1_x && r1_holds_r2_y)) {
    return 10;
}

if (r1_holds_r2_x || r2_holds_r1_x || r1_holds_r2_y || r2_holds_r1_y) {
    return 6;
}
```

YOJ-669. 矩形相交

- 有面积重叠的情形
- 情形(4): 仅1个顶点在内部(4)
- 以上所有包含关系都不满足，说明X轴和Y轴都是部分重叠
- 直接在函数最后返回4 即可。

YOJ-1567. 擂台我最强

By颜黎

- 首先用一个结构题存放每一位选手的信息，包括选手的编号，初始战斗力，当前的基础得分和总胜场。 然后用 k 和 w0 分别表示当前擂主和其战斗力。 依次模拟擂台的更替过程即可。 最后用胜场数统一计算额外得分。
- 排序后，计算排名。

```
#include <bits/stdc++.h>

#define N 100005

typedef long long ll;

using namespace std;

struct node
{
    ll id,w,s,z; // 编号，初始战斗力，当前的基础得分和总胜场
}a[N];

bool cmp(node a,node b){return a.s≠b.s?a.s>b.s:(a.z≠b.z?a.z>b.z:a.id<b.id);} // 按照
题目规则排序
```

YOJ-1567. 擂台我最强

```
int main()
{
    ll i,n;
    cin>>n;
    for(i=1;i<=n;++i)
        cin>>a[i].id>>a[i].w;
    ll k=1;long double w0=a[1].w; // 当前擂主和其战斗力
    for(i=2;i<=n;++i)
    {
        if(w0>a[i].w) // 守擂赢
        {
            ++a[k].z;
            a[k].s+=5;
            w0*=0.75;
        }
        else // 攻擂赢
        {
            ++a[i].z;
            a[i].s+=max(1ll,5-a[k].z);
            w0=a[i].w*0.75;k=i;
        }
    }
```

YOJ-1567. 擂台我最强

```
    for(i=1;i≤n;++i)a[i].s+=a[i].z≥6?5*a[i].z-15:a[i].z*(a[i].z-1)/2; // 根据胜场计算
额外得分
    sort(a+1,a+n+1,cmp);
    ll rk=1;
    for(i=1;i≤n;++i)
    {
        while(a[rk].s>a[i].s)++rk; // 排名为分数比其小的人数+1
        printf("%lld %08lld %lld\n",rk,a[i].id,a[i].s);
    }
    return 0;
}
```

YOJ-1567. 擂台我最强

这个题目的浮点数比较并没有用到eps，是因为如果当前战斗力还为整数时，是不存在精度损失的问题，而当某一刻开始不再为整数时，后面都不会再变为整数，而与之比较的永远是一个整数，所以不存在两个数相同，而因为精度损失导致判断结果有误。

事实上，这个题的精度需求可以近似的看做 $\frac{1}{m}$，其中 m 为值域。

具体来说要算这个题的精度要求，相当于找到 $a \in N, b \in N$ 时，$e = |(\frac{3}{4})^k a - b|$ 的最小值级别。

$$e = |(\frac{3}{4})^k a - b| = (\frac{3}{4})^k |a - (\frac{4}{3})^k b| = (\frac{3}{4})^k |(\frac{4}{3})^k b - [(\frac{4}{3})^k b]|$$

当 $b = 1$ 时，$(\frac{3}{4})^k$ 最小可以达到 $\frac{1}{m}$ 的级别，而 $|(\frac{4}{3})^k b - [(\frac{4}{3})^k b]| < 1$，所以 e 可以达到 $\frac{1}{m}$ 的级别。

所以说如果以这样的形式判断浮点数 $w0 + eps > a[i].w$，而且随手将 eps 定为 $1e-9$，那么就会有问题，因为本题的值域为 10^{15}。

YOJ-1567. 擂台我最强

具体来说可以参考这样两组数据：

```
200
00000001 983214767807684
00000002 1
00000003 1
......
```

```
200
00000001 983214767807685
00000002 1
00000003 1
......
```

前者标准输出第一行为 1 00000001 1185，后者为 1 00000001 1195，而错误的代码会输出一样的值。

数据中的 983214767807684 即为 $\lfloor(\frac{4}{3})^{120}\rfloor$，另一个为 $\lceil(\frac{4}{3})^{120}\rceil$。

YOJ-1571.可翻转等式

By房子珈

题意

求有多少个三元组 (A, B, C) 满足以下条件：

- $1 \leq A \leq B < N, M < C < N$。
- $A + B = C$ 且 $\mathrm{rev}(A) + \mathrm{rev}(B) = \mathrm{rev}(C)$。

$2 \leq N, M \leq 10^3$。

做法

直接枚举 A, B，则 $C = A + B$。检查是否满足 $M < A + B < N$ 并且是否有 $\mathrm{rev}(A) + \mathrm{rev}(B) = \mathrm{rev}(A + B)$。

实现整数翻转函数即可。

时间复杂度 $\mathcal{O}(n^2 \log n)$。

预处理翻转后的值，可以做到 $\mathcal{O}(n^2)$。

YOJ-1571.可翻转等式

代码

这是一份复杂度为 $\mathcal{O}(n^2 \log n)$ 的实现。

```
#include<cstdio>
int n,m,ans;
inline int rev(int x){//整数翻转
    int res=0;
    while(x) res=res*10+x%10,x/=10;
    return res;
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<n;i++){//A
        for(int j=i;j<n;j++){//B
            if(i+j<=m||i+j>=n) continue;//要求满足M<C<N
            if(rev(i)+rev(j)!=rev(i+j)) continue;//要求满足rev(A)+rev(B)=rev(C)
            ans++;
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

YOJ-1571.可翻转等式

这是一份复杂度为 $\mathcal{O}(n^2)$ 的实现。

```cpp
#include<cstdio>
int n,m,ans,r[2005];
inline int rev(int x){//整数翻转
    int res=0;
    while(x) res=res*10+x%10,x/=10;
    return res;
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) r[i]=rev(i);
    for(int i=1;i<n;i++){//A
        for(int j=i;j<n;j++){//B
            if(i+j<=m||i+j>=n) continue;//要求满足M<C<N
            if(r[i]+r[j]!=r[i+j]) continue;//要求满足rev(A)+rev(B)=rev(C)
            ans++;
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

By孙奕豪

YOJ-1575.餐厅用餐

- 某餐厅推出套餐优惠活动，有 5 种主菜（编号1-5）和 3 种饮料（编号1-3）可供选择。顾客需要选择恰好 2 道主菜和 1 种饮料组成套餐。
- 优惠规则如下：
  - 价格限制：套餐总价不能超过预算上限
  - 主菜互斥：主菜1和主菜3不能同时选择（口味冲突）
  - 饮料搭配：如果选择主菜5，则必须选择饮料3（最佳搭配）
  - 健康选择：如果选择主菜2，则不能选择饮料1（热量过高）
- 请给出满足条件的套餐组合数量，并输出总价最低的套餐方案。

YOJ-1575.餐厅用餐

- 主菜和饮料的数量都很少，可以直接枚举每一个方案，判断是否符合要求
- 具体地，枚举 i, j 表示选择第 i 种主菜和第 j 种主菜，再枚举一个 k 表示选择第 k 种饮料
- 注意两个问题：
  - $i \neq j$
  - (i, j) 是一个无序对，也就是说，i=1, j=3 和 i=3,j=1 实际上是同一种方案
- 解决方案是在枚举的时候直接要求 $i < j$，这样就可以不重不漏地枚举到所有方案

YOJ-1575.餐厅用餐

- 设a[i] 表示第 i 种主菜的价格，b[i] 表示第 i 种饮料的价格，预算上限为c，那么判断一个方案合法就可以写作下面的几个逻辑表达式，它们要同时成立
  - a[i]+a[j]+b[k]≤ c
  - i != 1 || j != 3
  - j != 5 || k ==3
  - (i != 2 && j != 2) || k != 1
- 或者等价为以下逻辑表达式均不成立：
  - a[i]+a[j]+b[k]>c
  - i == 1 && j == 3
  - j == 5 && k != 3
  - (i == 2 || j == 2) && k == 1

YOJ-1575.餐厅用餐

```
int cnt = 0, min_price = c + 1, u, v, w;
    // cnt:= 方案数, min_price:= 最低合法价格
    // 选择第 u, v 个主菜和第 w 个饮料时价格最低
    for(int i = 1; i <= 5; i ++)
        for(int j = i + 1; j <= 5; j ++)
            for(int k = 1; k <= 3; k ++) {
                // 尝试选择第 i, j 个主菜和第 k 个饮料, 钦定 i < j 保证不重复
                if(i == 1 && j == 3) continue;  // 主菜互斥
                if(j == 5 && k != 3) continue;  // 饮料搭配
                if((i == 2 || j == 2) && k == 1) continue;  // 健康选择
                if(a[i] + a[j] + b[k] > c) continue;    // 价格超出预算
                cnt ++; // 枚举到了一个新的合法方案
                if(a[i] + a[j] + b[k] < min_price) {    // 尝试更新最低价格
                    min_price = a[i] + a[j] + b[k];
                    u = i, v = j, w = k;
                }
            }
    if(cnt != 0)    // 如果有解, 输出方案数和方案
        printf("%d\n%d %d %d %d\n", cnt, u, v, w, min_price);
    else    // 无解输出 No solution
        puts("No solution");
```

YOJ-1589.贪吃的梅比乌斯

By李抗抗

【题目描述】

刚做完实验的梅比乌斯摸了摸咕咕叫的肚子，准备狩猎一些食物。她发现了一个神奇的星球，这个星球上面遍布着m*n的山洞，每个山洞里面都有一份食物，每一个山洞都有且仅有一条单向通路能通向一个与之相邻的山洞，梅比乌斯开心地钻进了好几个山洞，但屡屡碰壁，于是气鼓鼓地将这个星球的山洞分布发给了你。梅比乌斯现在生气了，只想最后进入一次山洞，如果你没有告诉她那些能吃到最多食物的山洞，她就会回来吃掉你，为了保住你的小命，请编写一个程序，计算出那个能吃到最多食物的洞口吧。

【提示】

1. 梅比乌斯能够沿着山洞之间的通路通行，这条通路上的食物她都能吃到
2. 相邻指的是上下左右四个方向
3. 如果一条通路构成了环，一个洞口的食物只能计算一次

【输入格式】

n+1行，第一行两个数m, n，表示这个m*n矩阵的大小，有m列，n行；第2~n+1行，每行m个数，每个数表示这个山洞能连通的方向，1表示向左，2表示向右，3表示向上，4表示向下

【输出格式】

K行，表示有k个符合要求的洞口。每行两个数，由空格分隔，表示洞口的坐标，第一个数是横坐标，第二个数是纵坐标，参考输入格式，矩阵的最左上角为(0, 0)，向右横坐标递增，向下纵坐标递增。如果有多个符合要求的洞口，则分行输出，且按横坐标递增优先、纵坐标递增其次排序。

YOJ-1589.贪吃的梅比乌斯

- 70分：O(N^4)
- 考虑朴素的枚举，从每一个点出发尝试继续走，同时记录经过的点。当走到界外或者遇到记录过的点时（即出现了环），枚举结束，得到该点出发能吃到的食物数量。

YOJ-1589.贪吃的梅比乌斯

- for i,j:
- init()
- while(1):
- if(invalid(x,y)||vis(x,y))break;
- cnt[i][j]++;
- vis[x][y]=true;
- newx->x newy->y

YOJ-1589.贪吃的梅比乌斯

- 100分：O(N^2)
- 考虑分析刚才的思路，可以发现，我们的算法重复了很多枚举，比如一条从左到右的链，我们在第一个点的枚举过程中其实已经将这条链上每一个点的答案得到了，无需再从第2、3…个重复这一过程。
- 也即，我们在枚举一条路径时，应该将路径上所有点的答案都进行计算，同时在枚举过程中遇到之前算过的点就直接利用这部分结果、不再枚举下去。这样，每个点访问次数为O(1)次，包含初次枚举和可能的以后被相邻节点利用（至多四个方向四次），复杂度O(N^2)

YOJ-1589.贪吃的梅比乌斯

- 从(0,0)开始枚举出发点，如果当前点已经有答案则跳过，否则开始沿箭头走，沿途存下这次走过的点以统一计算他们的答案。如果指向的下一个点越界则当前点答案为1，结束本次枚举，往回一步步+1计算前面点的答案；如果下个点已经有答案，将该点答案cnt[x][y]赋值为cnt[next_x][next_y]+1，回退、更新；如果下个点访问过，且没有答案，则说明出现了环，环上的点答案为环长，为了方便的计算环长，我们补充一个int数组dep[x][y]来替代“是否访问过”，它的含义是(x,y)在一次枚举中被走到时是第几个点，即满足dep[x][y]+1=dep[next_x][next_y]，那么出现环时（假设（next_x,next_y）为访问过的、找到环的点）环长为dep[x][y]-dep[next_x][next_y]+1，把(next_x,next_y)到(x,y)之间的点答案都赋值为环长，随后再步步+1赋值环外的点。

YOJ-1589.贪吃的梅比乌斯

```
//dx,dy为方向数组，tmpx,tmpy记录走过的点的横纵坐标，num表示已经记录的点的数量（因为0开始标号了所以其实是数量-1）
int main(){
    scanf("%d%d",&m,&n);
    for(int i=1;i<=n;i++)for(int j=1;j<=m;j++)scanf("%d",&a[i][j]);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            if(!dep[i][j]){
                dep[i][j]=1,tmpx[0]=i,tmpy[0]=j;
                int x=i,y=j,tx,ty,num=0;
                while(1){
                    tx=x+dx[a[x][y]],ty=y+dy[a[x][y]];//dx[5]={0,0,0,-1,1},dy[5]={0,-1,1,0,0}
                    if(!(tx<=n&&tx>=1&&ty<=m&&ty>=1)){cnt[x][y]=1;break;}
                    if(cnt[tx][ty]){
                      cnt[x][y]=cnt[tx][ty]+1;
                      break;
                    }
```

YOJ-1589.贪吃的梅比乌斯

```
                if(/*!cnt[tx][ty]&&*/dep[tx][ty]){
                   cnt[x][y]=dep[x][y]-dep[tx][ty]+1;
                   while(!(tmpx[num]==tx&&tmpy[num]==ty))cnt[tmpx[num]][tmpy[num]]=cnt[x][y],num--;
                   cnt[tmpx[num]][tmpy[num]]=cnt[x][y];
                   break;
                }
                dep[tx][ty]=dep[x][y]+1;
                tmpx[++num]=tx;tmpy[num]=ty;
                x=tx,y=ty;
             }
             while(num>0)cnt[tmpx[num-1]][tmpy[num-1]]=cnt[tmpx[num]][tmpy[num]]+1,num--;
          }
       }
    }
    int ans=0;
   for(int i=1;i<=n;i++)for(int j=1;j<=m;j++)ans=max(ans,cnt[i][j]);
    for(int j=1;j<=m;j++)for(int i=1;i<=n;i++)if(cnt[i][j]==ans)printf("%d %d\n",j-1,i-1);
}
```