本章主要考查基本的最短路问题

译题 - ttwo

The Tamworth Two

两只塔姆沃斯牛

译 by Jure 修订 by ForEverLeeR BOI '98? - Richard Forster

题目描述

两只牛逃跑到了森林里。农夫 John 开始用他的经验追捕这两头牛。你的任务是模拟他们的行为(牛和 John)。

追击在 10x10 的平面网格内进行。一个格子可以是：

一个障碍物，两头牛(它们总在一起)，或者 农民 John. 两头牛和农民 John 可以在同一个格子内(当他们相遇时)，但是他们都不能进入有障碍的格子。

一个格子可以是：

. 空地
* 障碍物
C 两头牛
F 农民 John

这里有一个地图的例子：

```
*...*.....
......*...
...*...*..
..........
...*.F....
*.....*...
...*......
..C......*
...*.*....
.*.*......
```

牛在地图里以固定的方式游荡。每分钟，它们可以向前移动或是转弯。如果前方无障碍且不会离开地图，它们会按照原来的方向前进一步。否则它们会用这一分钟顺时针转 90 度。

农民 John 深知牛的移动方法，他也这么移动。

每次(每分钟)农民 John 和两头牛的移动是同时的。如果他们在移动的时候穿过对方，但是没有在同一格相遇，我们不认为他们相遇了。当他们在某分钟末在某格子相遇，那么追捕结束。

读入十行表示农夫 John,两头牛和所有障碍的位置的地图。每行都只包含 10 个字符，表示的含义和上面所说的相同，你可以确定地图中只有一个'F'和一个'C'.'F'和'C'一开始不会处于同一个格子中。

计算农夫 John 需要多少分钟来抓住他的牛，假设牛和农夫 John 一开始的行动方向都是正北（即上）。 如果 John 和牛永远不会相遇，输出 0。

PROGRAM NAME: ttwo

INPUT FORMAT

第 1-10 行:

每行 10 个字符，表示如上文描述的地图。

SAMPLE INPUT (file ttwo.in)

```
*...*.....
......*...
...*...*..
..........
...*.F....
*.....*...
...*......
..C......*
...*.*....
.*.*......
```

OUTPUT FORMAT

输出一个数字，表示 John 需要多少时间才能抓住牛们。输出 0，如果 John 无法抓住牛。

SAMPLE OUTPUT (file ttwo.out)

题解 - ttwo

分析

简单的模拟

需要记录方向和坐标，进行模拟，直到相遇。如何判断永远不相遇呢？

状态只有(10*10*4)400 种,两个人最多(400*400)160000 种,也就是说,如果超过 160000 步，那么肯定会出现有的状态出现了 2 次以上，那么就肯定是一个死循环，永远不会相遇。

判断无解也可以通过判断当前状态是否出现过——使用源码中 until 的前两个条件即可。

优化

利用运动周期 大大降低循环次数(code 已给出)

易证明:奶牛及约翰各自最终运动路线为一循环.

故:只需模拟时间为二者运动周期最小公倍数之前的所有状态即可.

开数组(以奶牛为例) cow[cowx,cowy,cowface]记录奶牛第一次到达格子[x,y]及朝向状态的时间

当奶牛第二次处于此状态时 显然 奶牛运动周期为 cyclecow=time(当前时间)-cow[cowx,cowy,cowface].

补:因为只有四个方向,所以每个点经过三次以上(4 次)便不行了,也可以用这种方法判断能否找到,也是这道题的难点.

译题 - mazel

Overfencing 穿越栅栏

Kolstad and Schrijvers

译 by lyl

描述

农夫 John 在外面的田野上搭建了一个巨大的用栅栏围成的迷宫。幸运的是，他在迷宫的边界上留出了两段栅栏作为迷宫的出口。更幸运的是，他所建造的迷宫是一个“完美的”迷宫：即你能从迷宫中的任意一点找到一条走出迷宫的路。 给定迷宫的宽 W(1<=W<=38)及长 H(1<=H<=100)。 2*H+1 行，每行 2*W+1 的字符以下面给出的格式表示一个迷宫。然后计算从迷宫中最“糟糕”的那一个点走出迷宫所需的步数。（即使从这一点以最优的方式走向最靠近的出口，它仍然需要最多的步数）当然了，牛们只会水平或垂直地在 X 或 Y 轴上移动，他们从来不走对角线。每移动到一个新的方格算作一步（包括移出迷宫的那一步）这是一个 W=5,H=3 的迷宫：

```
+-+-+-+-+-+
|         |
+-+ +-+ + +
|     | | |
+ +-+-+ + +
| |     |
+-+ +-+-+-+
```

如上图的例子，栅栏的柱子只出现在奇数行或奇数列。每个迷宫只有两个出口。

格式

PROGRAM NAME: maze1

INPUT FORMAT:

(file maze1.in)

第一行： W 和 H（用空格隔开）

第二行至第 2*H+2 行： 每行 2*W+1 个字符表示迷宫

OUTPUT FORMAT:

(file maze1.out)

输出一个单独的整数，表示能保证牛从迷宫中任意一点走出迷宫的最小步数。

SAMPLE INPUT

```
5 3
+-+-+-+-+-+
|         |
+-+ +-+ + +
|     | | |
```

```
+ +-+-+ + +
| |     |
+-+ +-+-+-+
```

SAMPLE OUTPUT

9

题解 - maze1

分析

我们用一个数组记录每个格子四面连通情况，然后从输入文件中读入每条边（判断），维护这个数组。然后分别从两个出口做 Flood fill，记录每个格子的最短距离。所有格子的最短距离的最大值为所求。

可以将整个图用 2*w+1,2*h+1 的布尔数组表示，门口处记为一步，最后统计所有偶数行列，输出 max div 2

小提示：使用广搜实现 Flood Fill 更快一点,最好 bfs,dfs 只过 4 个.

译题 - cowtour

Cow Tours 牛的旅行

译 by Jeru

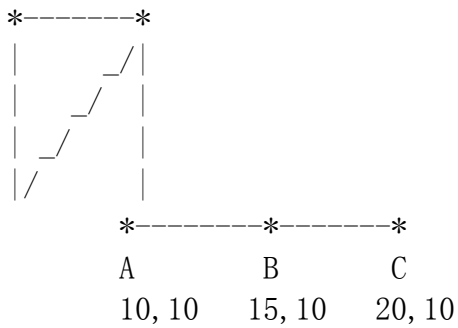
描述

农民 John 的农场里有很多牧区。有的路径连接一些特定的牧区。一片所有连通的牧区称为一个牧场。但是就目前而言，你能看到至少有两个牧区不连通。这样，农民 John 就有多个牧场了。

John 想在农场里添加一条路径(注意，恰好一条)。对这条路径有以下限制：

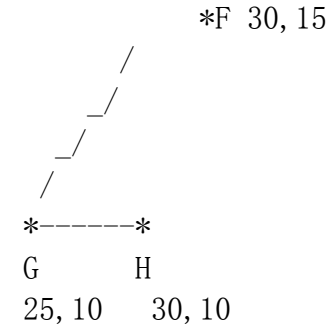
一个牧场的直径就是牧场中最远的两个牧区的距离(本题中所提到的所有距离指的都是最短的距离)。考虑如下的有 5 个牧区的牧场，牧区用“*”表示，路径用直线表示。每一个牧区都有自己的坐标：

```
                15,15    20,15
 D        E
```



这个牧场的直径大约是 12.07106，最远的两个牧区是 A 和 E，它们之间的最短路径是 A-B-E。

这里是另一个牧场：



这两个牧场都在 John 的农场上。John 将会在两个牧场中各选一个牧区，然后用一条路径连起来，使得连通后这个新的更大的牧场有最小的直径。

注意，如果两条路径中途相交，我们不认为它们是连通的。只有两条路径在同一个牧区相交，我们才认为它们是连通的。

合并后的两个牧场的图片

输入文件包括牧区、它们各自的坐标，还有一个如下的对称邻接矩阵：

	A	B	C	D	E	F	G	H
A	0	1	0	0	0	0	0	0
B	1	0	1	1	1	0	0	0
C	0	1	0	0	1	0	0	0
D	0	1	0	0	1	0	0	0
E	0	1	1	1	0	0	0	0
F	0	0	0	0	0	0	1	0
G	0	0	0	0	0	1	0	1
H	0	0	0	0	0	0	1	0

输入文件至少包括两个不连通的牧区。

请编程找出一条连接两个不同牧场的路径，使得连上这条路径后，这个更大的新牧场有最小的直径。

格式

PROGRAM NAME: cowtour

INPUT FORMAT:

(file cowtour.in)

第 1 行: 一个整数 N (1 <= N <= 150), 表示牧区数

第 2 到 N+1 行: 每行两个整数 X,Y (0 <= X ,Y<= 100000), 表示 N 个牧区的坐标。注意每个 牧区的坐标都是不一样的。

第 N+2 行到第 2*N+1 行: 每行包括 N 个数字(0 或 1) 表示如上文描述的对称邻接矩阵。

OUTPUT FORMAT:

(file cowtour.out)

只有一行，包括一个实数，表示所求答案。数字保留六位小数。

SAMPLE INPUT

```
8
10 10
15 10
20 10
15 15
20 15
30 15
25 10
30 10
01000000
10111000
01001000
01001000
01110000
00000010
00000101
00000010
```

SAMPLE OUTPUT

22.071068

题解 - cowtour

分析

用 Floyd 求出任两点间的最短路，然后求出每个点到所有可达的点的最大距离，记做 mdis[i]。

r1=max(mdis[i])

然后枚举不连通的两点 i,j，把他们连通，则新的直径是 mdis[i]+mdis[j]+(i,j)间的距离。

r2=min(mdis[i]+mdis[j]+dis[i,j])
re=max(r1,r2)

re 就是所求。

译题 - comehome

Bessie Come Home

回家

Kolstad & Burch

译 by tim green

描述

现在是晚餐时间,而母牛们在外面分散的牧场中。 农民约翰按响了电铃,所以她们开始向谷仓走去。 你的工作是要指出哪只母牛会最先到达谷仓(在给出的测试数据中,总会有且只有一只速度最快的母牛)。 在挤奶的时候(晚餐前),每只母牛都在她自己的牧场上,一些牧场上可能没有母牛。 **每个牧场由一条条道路和一个或多个牧场连接(可能包括自己)。 有时，两个牧场(可能是自我相同的)之间会有超过一条道路相连。 至少有一个牧场和谷仓之间有道路连接。** 因此,所有的母牛最后都能到达谷仓,并且母牛总是

走最短的路径。 当然,母牛能向着任意一方向前进,并且她们以相同的速度前进。 牧场被标记为'a'..'z'和'A'..'Y',在用**大写字母表示的牧场中有一只母牛,小写字母中则没有**。 谷仓的标记是'Z',注意没有母牛在谷仓中。

格式

PROGRAM NAME: comehome

INPUT FORMAT

第 1 行: 整数 P(1<= P<=10000),表示连接牧场(谷仓)的道路的数目。

第 2 ..P+1 行: 用空格分开的两个字母和一个整数:

被道路连接牧场的标记和道路的长度(1<=长度<=1000)。

SAMPLE INPUT

(file comehome.in)

```
5
A d 6
B d 3
C e 9
d Z 8
e Z 3
```

OUTPUT FORMAT

单独的一行包含二个项目: 最先到达谷仓的母牛所在的牧场的标记,和这只母牛走过的路径的长度。

SAMPLE OUTPUT

(file comehome.out)

```
B 11
```

分析

这道题的规模很小（52），所以可以用 Floyd-Warshall 算法求出所有点对的最短路，然后找出其中到'Z'距离最短的点就可以了。

当然本题也可以用 Dijkstra 来做，会比 Floyd 快一些，不过编程复杂度要高一些。
思路二：借鉴广度优先搜索的思想。code 已给出。 建立图的邻接表存储，每次访问展节点 c 时，对其相连的顶点 v 进行检查，若有 dis['Z',c]+dis[c,v]<dis['Z',v] 则 dis['Z',v]:=dis['Z',c]+dis[c,v]; 如果节点 v 未访问 则进队。只需进行一次宽度优先搜索即可。值得注意的是，由于有 10000 条边，而总共只有 52 个节点，也就是说，输入中的许多边都是没有用的，所以读入的时候应该判断一下这条边已有的权值是否比读入的权值要小，如果小的话，再赋值。

译题 - fracdec

Fractions to Decimals

分数化小数

译 by tim green

描述

写一个程序，输入一个形如 N/D 的分数(N 是分子，D 是分母)，输出它的小数形式。 如果小数有循环节的话，把循环节放在一对圆括号中。

例如, 1/3 =0.33333333 写成 0.(3), 41/333 = 0.123123123... 写成 0.(123), 用 xxx.0 等表示整数。 典型的转化例子：

1/3 = 0.(3)
22/5 = 4.4
1/7 = 0.(142857)
2/2 = 1.0
3/8 = 0.375
45/56 = 0.803(571428)

PROGRAM NAME

fracdec

INPUT FORMAT

单独的一行包括被空格分开的 N 和 D， 1 <= N,D <= 100000。

SAMPLE OUTPUT

(file fracdec.in)

45 56

OUTPUT FORMAT

按照上面规则计算出的小数表达式.如果结果长度大于 76,每行输出 76 个字符.

SAMPLE OUTPUT

(file fracdec.out)

0.803(571428)

分析

（官方题解 译 by 逆铭）
记得长除法吗？我们知道只有当出现了曾经出现过的余数时，小数部分才会出现重复。重复的部分就是自从我们上次见到同样的余数之后计算出的部分。
我们先读入并打印整数部分。接下来，我们对剩下的真分数部分进行长除直到我们发现了重复的余数或余数变为 0。如果我们发现了重复的余数，即出现了循环节，就分别恰当地打印重复的部分和不重复的部分。如果余数变为 0，即已经除尽，就打印整个小数部分。如果小数位根本没有被生成，那么打印一个 0 就是正确答案了。（此方法源码见：参考代码-> C/C++ -> 官方源码 1）
下面是另一个更加优美的解法，来自 Anatoly Preygel。
计算循环开始前的小数位数，这样你甚至无需保存各个小数位和余数，程序的空间花费将大幅减小，而运行速度也能有所提高。我们知道 2 和 5 的幂是仅有的两种不导致循环的数，因此要找到循环前的各数位，我们只需分别找到分子分母中包含的因子 2 和 5 个数的差，再取两者的最大值（详见代码片段）。然后我们仅使用第一个余数，在计算时输出各个数位即可

我们知道，a/b 的每位运算所得的余数只可能是 0..b-1，如果在某处出现一个余数之前曾经出现过(在小数位上),那么可以肯定此时从该处到上次用出现这个这个商之间存在循环节。

用 m 记录每种余数是否曾经出现过，ss 记录余数第一次出现的位置，如果余数为 0 就是整除，否则就找到循环节，输出

实际上,后面的小数内容完全取决于试除这一位时的商和余数.

另外,注意输出格式.