比较精妙的思想的进一步体现
Translate:USACO/buylow

Buy Low, Buy Lower 逢低吸纳

译 by Twink

描述

“逢低吸纳”是炒股的一条成功秘诀。如果你想成为一个成功的投资者，就要遵守这条秘诀:

"逢低吸纳,越低越买"

这句话的意思是：每次你购买股票时的股价一定要比你上次购买时的股价低.按照这个规则购买股票的次数越多越好，看看你最多能按这个规则买几次。

给定连续的 N 天中每天的股价。你可以在任何一天购买一次股票，但是购买时的股价一定要比你上次购买时的股价低。写一个程序，求出最多能买几次股票。

以下面这个表为例, 某几天的股价是:

```
天数 1  2  3  4  5  6  7  8  9  10 11 12
股价 68 69 54 64 68 64 70 67 78 62 98 87
```

这个例子中, 聪明的投资者(按上面的定义)，如果每次买股票时的股价都比上一次买时低，那么他最多能买 4 次股票。一种买法如下(可能有其他的买法):

```
天数 2  5  6  10
```

```
股价 69 68 64 62
```

格式

PROGRAM NAME: buylow

INPUT FORMAT:

(file buylow.in)

第 1 行: N (1 <= N <= 5000), 表示能买股票的天数。

第 2 行以下: N 个正整数 (可能分多行) ，第 i 个正整数表示第 i 天的股价. 这些正整数大小不会超过 longint(pascal)/long(c++).

OUTPUT FORMAT:

(file buylow.out)

只有一行，输出两个整数：

能够买进股票的天数 长度达到这个值的股票购买方案数量

在计算解的数量的时候，如果两个解所组成的字符串相同，那么这样的两个解被认为是相同的（只能算做一个解）。因此，两个不同的购买方案可能产生同一个字符串，这样只能计算一次。

SAMPLE INPUT

```
12
68 69 54 64 68 64 70 67
78 62 98 87
```

SAMPLE OUTPUT

```
4 2
```

分析：

For simplicity, let a_i be the i-th price, and l_i be the length of the longest sequence of prices that ends with the i-th price. Clearly, $l_1 = 1$. More generally, $l_i = g+1$, where g is the length of the longest sequence which ends before the i-th price and whose last element is higher than a_i (0 if no such number exists). These can be calculated using dynamic programming.

To determine the number of (distinct) maximum sequences, the same trick will work. For each value j such that $j < i$, $a_j > a_i$, and $l_j = l_i - 1$, for each sequence of maximum length that ends with the j-th price, the i-th price can be postpending to create a maximum length sequence that ends with the i-th price. To avoid double counting sequences, note that is $k < i$, $a_k = a_i$, and $l_k = l_i$, then any sequence which k can be postpended to and create a maximum sequence, i can be postpended to as well. Thus, for each value, add in only the count for the most recent occurence. Determining the number of maximum length sequences can be done in the same manner.

Translate:USACO/prime3

The Primes 素数方阵

IOI'94

译 by Felicia Crazy

描述

在下面的方格中，每行，每列，以及两条对角线上的数字可以看作是五位的素数。方格中的行按照从左到右的顺序组成一个素数，而列按照从上到下的顺序。两条对角线也是按照从左到右的顺序来组成。

```
+---+---+---+---+---+
| 1 | 1 | 3 | 5 | 1 |
+---+---+---+---+---+
| 3 | 3 | 2 | 0 | 3 |
```

```
+---+---+---+---+---+
| 3 | 0 | 3 | 2 | 3 |
+---+---+---+---+---+
| 1 | 4 | 0 | 3 | 3 |
+---+---+---+---+---+
| 3 | 3 | 3 | 1 | 1 |
+---+---+---+---+---+
```

这些素数各个数位上的和必须相等。

左上角的数字是预先定好的。

一个素数可能在方阵中重复多次。

如果不只有一个解，将它们全部输出（按照这 25 个数字组成的 25 位数的大小排序）。

一个五位的素数开头不能为 0（例如：00003 不是五位素数）

格式

PROGRAM NAME: prime3

INPUT FORMAT:

(file prime3.in)

一行包括两个被空格分开的整数:各数位上的数字的和 以及左上角的数字。

OUTPUT FORMAT:

(file prime3.out)

对于每一个找到的方案输出 5 行，每行 5 个字符，每行可以转化为一个 5 位的质数.在两组方案中间输出一个空行. 如果没有解就单独输出一行"NONE"。

SAMPLE INPUT

```
11 1
```

SAMPLE OUTPUT

上面的例子有三组解。

```
11351
14033
30323
53201
13313

11351
33203
30323
14033
33311

13313
13043
32303
50231
13331
```

分析：

First of all, we need to pre-compute all 5-digit primes given the digit sum we're interested in. This can be done in several ways, the way we choose depends on conditions. When solving the problem on a time-limited (5 hours) competition, we choose the simplest way (KISS Principle): for every number ranging from 10000 to 99999 we check whether is properly divisible by any number I (I ranges from 2 to sqrt(99999)); if so the number is definitely not a prime; otherwise a prime number is found. The first simple extension is to check only odd numbers since we're interested in 5-digit numbers (and the only even prime is 2).

When the running time is crucial, we extend the previous idea even more - we check whether the numbers 6x+1, 6x-1 that are in the range from 10000 to 99999 are properly divisible by any prime P ranging from 5 to sqrt(99999) (namely primes: 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313).

Since all solutions (given the digit sum and the top-left digit) have to be found, we must perform an exhaustive search. The "stupidest" idea is to fill the prime square

row-by-row with 5-digit primes and when the entire square is filled we just test whether the columns and both diagonals are primes. This obviously does not lead to a low running time since (for the worst test case: digit sum 23, and top-left digit 3) there are (at most) 757 prime numbers to be filled-in each row, which results to 757^5 (rough estimate: 250 x 10^12) operations - impossible even on today's 1Ghz processors.

The obvious way to fill the numbers in the square is to fill them in such way that as much as possible is known in advance about the filling prime - thus reducing the number of possibilities. One of the best ways to do so is given below.

Note: All further complexity estimates will be done on the "worst possible" test case: digit sum 23, top-left digit 3; I wonder why this test case was not used in the RECORD section).

Before all computation is done, we pre-compute so-called prime-patterns. This means that wherever we need to fill a prime whose any digits are know in advance, we pre-compute the primes when the given digits are arbitrary.

First, fill-up the main diagonal (from top-left corner to the bottom-right). Since we know only the first digit we have about 90 possible primes to be filled.

Second, (for all selections of the main diagonal) fill up the other diagonal. Since we know the middle digit in advance we have about 90 possibilities but we also know that the first digit is definitely not even (because the first column must be also a prime) - thus halving the number of primes - we have 45 possibilities.

Third, (for all possible previous selections) fill-up the first row - we have about 20 possibilities, then fill the second and fourth columns, and compute the middle digit in the last row (we know all other digits and the digit sum) and verify that last row is prime. From this point on only a small number of selections will pass, since it is not very common that the last row will be prime. The filling process continues filling the third column, second and fourth row and final verification. The table below shows the order in which the single digits are filled-in.

1	3	3	3	2
8	1	7	2	8
10	4	1	5	10
9	2	7	1	9
2	4	6	5	1

Finally the solutions need to be sorted because we need to output them in a sorted order - of course we use the built-in QSORT function.

The total operations required (worst test case) are 90*45*20*5*5*5*5*5 - which is only theoretical, since number of possibilities drastically lowers the final fives.

The idea of this approach can be extended to use symmetry to lower the running time eve more. (When square filled) when rows are exchanged by columns the main diagonal remains the same, rows are columns, columns are rows (all are valid primes) and the other diagonal reverts itself, thus if the reverted diagonal is prime we have another valid solution. For the worst test case 23,3 there are 79 (from total of 152 solutions) solutions with the other diagonal as a "double prime" - some of them are symmetric by the main diagonal and the others (about 70) form "pairs" where one member could be obtained from the another one by exchanging rows by columns since the second diagonal is "double-prime". To lower the running time, when filling the square in the STEP 2, when a double-prime is about to be inserted, we insert it only the one way and to get solutions for the second way we perform the operation of exchanging the rows by columns (e.g., 1. row <-> 1. column, 2.row <-> 2.column, etc.). Thus reducing the total running time.

Translate:USACO/race3

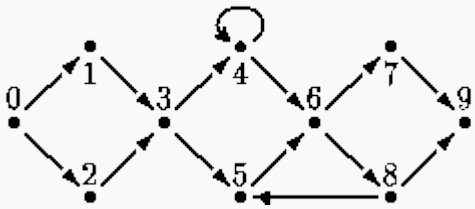
Street Race 街道赛跑

IOI'95

译 by Felicia Crazy

描述

图一表示一次街道赛跑的跑道。可以看出有一些路口（用 0 到 N 的整数标号），和连接这些路口的箭头。路口 0 是跑道的起点，路口 N 是跑道的终点。箭头表示单行道。运动员们可以顺着街道从一个路口移动到另一个路口（只能按照箭头所指的方向）。当运动员处于路口位置时，他可以选择任意一条由这个路口引出的街道。



图一：有 10 个路口的街道 一个良好的跑道具有如下几个特点：

每一个路口都可以由起点到达。

从任意一个路口都可以到达终点。

终点不通往任何路口。

运动员不必经过所有的路口来完成比赛。有些路口却是选择任意一条路线都必须到达的（称为“不可避免”的）。在上面的例子中，这些路口是 0，3，6，9。对于给出的良好的跑道，你的程序要确定“不可避免”的路口的集合，不包括起点和终点。

假设比赛要分两天进行。为了达到这个目的，原来的跑道必须分为两个跑道，每天使用一个跑道。第一天，起点为路口 0，终点为一个“中间路口”；第二天，起点是那个中间路口，而终点为路口 N。对于给出的良好的跑道，你的程序要确定“中间路口”的集合。如果良好的跑道 C 可以被路口 S 分成两部分，这两部分都是良好的，并且 S 不同于起点也不同于终点，同时被分割的两个部分满足下列条件：（1）它们之间没有共同的街道（2）S 为它们唯一的公共点，并且 S 作为其中一个的终点和另外一个的起点。那么我们称 S 为“中间路口 ”。在例子中只有路口 3 是中间路口。

格式

PROGRAM NAME: race3

INPUT FORMAT:

(file race3.in)

输入文件包括一个**良好**的跑道，最多有 50 个路口，100 条单行道。

一共有 N+2 行，前面 N+1 行中第 i 行表示以编号为(i-1)的路口作为起点的街道，每个数字表示一个终点。行末用 -2 作为结束。最后一行只有一个数字 -1。

OUTPUT FORMAT:

(file race3.out)

你的程序要有两行输出：

第一行包括：跑道中“不可避免的”路口的数量，接着是这些路口的序号，序号按照升序排列。

第二行包括：跑道中“中间路口”的数量，接着是这些路口的序号，序号按照升序排列。

SAMPLE INPUT

```
1 2 -2
3 -2
3 -2
5 4 -2
6 4 -2
6 -2
7 8 -2
9 -2
5 9 -2
-2
-1
```

SAMPLE OUTPUT

```
2 3 6
1 3
```

分析：

一道与图的连通有点关系的题目，主要就是在一个单向（非严格）有向图中寻找一个可以将图断开的点（ms 不是割点），第一问枚举每个点，将其删掉，然后查看连通性。第二问其实是第一问的一个子集。这样可以减少搜索的次数（也可以通盘枚举），需要先建立分层图，然后从枚举点开始 DFS，只要找到一个点的层次小于检测点，那么就查找失效，否则成功。

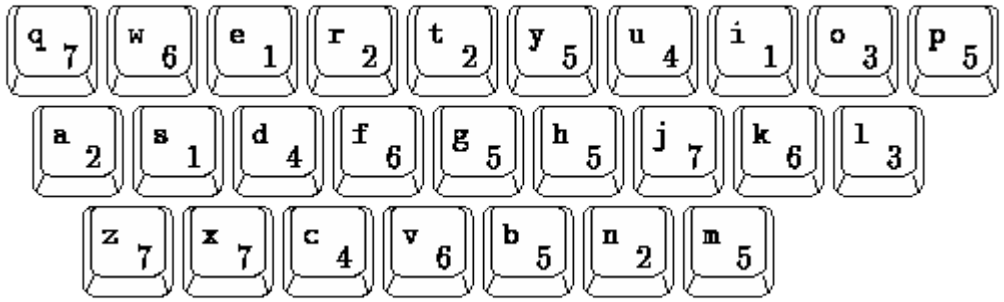
Translate:USACO/lgame

Letter Game 字母游戏

IOI 1995

译 by kd

描述



在家里用电视机做字母游戏是很流行的，其中一种玩法是:每一个字母有一个数值与之对应.你收集字母组成一个或多个字以得到尽可能高的得分.除非你已有了 “找字的方法”(“a way with words”)，你会把你知道的字都试一遍.有时你也许会查阅其拼写，然后计算得分。显然可以用计算机更为准确地完成此

任务。上图示出了英文字母及其所对应的值，当给出英文字(word) 的列表及收集的字母时，请找出所能形成的得分最高的字或字对(pairs of words)。

格式

PROGRAM NAME: lgame

INPUT FORMAT:

(file lgame.in)

输入文件 lgame.in 中有一行由小写字母(`a'到`z')组成的字符串， 这就是收集到字母(就是可以使用的字母)，字符串由至少 3 个字母至多 7 个字母(以任意顺序) 组成。

(file lgame.dict)

词典文件 lgame.dict 由至多 40，000 行组成，文件的最后一行有'.' 表示文件的结束。其它各行每一行都是由至少 3 个小写字母，至多 7 个小写字母组成的字符串。文件中的词已按字典顺序排序。

OUTPUT FORMAT:

(file lgame.out)

在文件 lgame.out 的第一行，你的程序应写上最高得分(子任务 A).使用上面图形中给出的字母-值对应表。

随后的每一行是由文件 lgame.dict 中查到的具有这个得分的所有的词和或词对(word pairs)(子任务 B)。要利用图中给定的字母的值。

当**两个词**能够形成 一个组合(具有给定的字母)时，这两个词应该打印到同一行，两个词中间用一个空格隔开。不许重复表示词对，例如'rag prom'和'prom rag'是同样的词对，只输出字典顺序较小的那个。

输出要按照字典顺序排序，如果两个词对第一个单词的顺序相同，则按照第二个单词。

SAMPLE INPUT(file lgame.in)

```
prmgroa
```

SAMPLE INPUT(file lgame.dict)

```
profile
program
prom
rag
ram
rom
.
```

SAMPLE OUTPUT

```
24
program
prom rag
```

分析：

这好像是第二道带字典的题目了。方法是:枚举+位运算优化。最多只有 7 个字母，那么就用一个 long 类型记录用过的字母，所以对于没有用过的字母的判断方式是：两个单词的记录的按位与等于 0。当然这样判断还有漏洞，这要查找时记录两种位置就行了：1.正序扫描的，2.反序扫描的。(可以避免位记录重复)。这样用 O(C*N)的复杂度构造完成所有单个字串，然后查找一对的字串再次单调扫描即可。最后需要排序。