Univerza v Ljubljani

Prvenstvo UNIVERZE V LJUBLJANI v programiranju

2002

2. poskusno tekmovanje

Fakulteta za matematiko in fiziko

Pokrovitelji:

INNOVATIVE SOFTWARE SOLUTIONS
HERMES SoftLab

Prvenstvo v programiranju 2002

poskusno tekmovanje 20. maj 2002

Prvenstvo v programiranju 2002

poskusno tekmovanje 20. maj 2002

NAVODILA

Prijava: Prijavite se lahko le na en računalnik. Če bodo med tekmovanjem z njim slučajno težave, javite vodstvu tekmovanja. V primeru, da boste hkrati prijavljeni na dveh računalnikih, bo ekipa diskvalificirana.

Uporabniško ime je sestavljeno iz besede acm in številke računalnika.

Primer: uporabniško ime na računalniku št. 1 je: acm01

uporabniško ime na računalniku št. 12 je: acm12

Geslo je sporočeno pred začetkom tekmovanja.

Vaše domače področje na disku je `/home/acmXX` kjer je XX številka računalnika

Priprava delovnega okolja

Po uspešni prijavi morate odpreti dva terminala in program Netscape.

Terminal št.1: v njem boste pisali programe, prevajali programe in pošiljali programe sodniku.

Terminal št.2: v njem boste dobivali obvestila o pravilnosti rešitev in pošiljali morebitna vprašanja sodniku.

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Pisanje programov:

Na voljo so urejevalniki besedil: *vi, joe, jed, mcedit, emacs*, *gedit in kedit*

Ime programov je točno določeno pri opisu nalog. Pazite na `return 0,` ki mora zaključiti vsak uspešno izveden C program.

Prevajanje programov:

Za prevajanje programov morate uporabljati naslednje prevajalnike:

```
C       (*.c)         gcc    primer:   gcc -o program program.c
C++     (*.cpp)       g++    primer:   g++ -o program program.cpp
Pascal (*.pas)        gpc    primer:   gpc -o program program.pas
```

Pošiljanje programov:

Za pošiljanje programov sodniku uporabljate ukaz `submit`.

Primer: `submit program.c`

Sporočila o napakah – PE in WA

Zaradi različnih razlogov je zelo težko razlikovati med `Presentation Error` in `Wrong Answer`. Ker preverjanje poteka več ali manj avtomatsko, program pogosto ne zna razlikovati med tem, ali ste le pozabili piko v odgovoru, ali pa ste napačno zračunali. Zato tudi `Wrong Answer` včasih lahko pomeni le manjkajoč znak v odgovoru. Praviloma naj bi sporočilo `PE` dobili le v primeru napačno postavljenih presledkov ali ločil in praznih vrstic, ni pa to nujno!

Komunikacija sodniki – ekipa:

V terminalu 2 izvedete ukaz: `telnet svarun`

Za prijavo na računalnik svarun uporabite isto uporabniško ime in geslo, kot ste ga uporabili za prijavo na vaš računalnik. V tem terminalskem oknu se bodo izpisovala sodnikova sporočila vam. Če želite poslati sporočilo sodniku, morate v tem terminalu napisati ukaz:

```
write sodnik
```

in potem sporočilo, ki ga zaključite s kombinacijo tipk CTRL + D na začetku prazne vrstice. Prosimo, da **ne** pošiljate sporočil tipa `... Sedaj smo poslali program ...` in `... Naš program zagotovo dela, pa dobimo WA ...` Ekipa, ki bo pošiljala tovrstna sporočila, je lahko tudi diskvalificirana.

Trenutni rezultati:

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Po zagonu programa greste na naslov:

http://svarun.fmf.uni-lj.si/index.html

Kadar želite videti trenutno stanje, morate narediti »reload«.

Pritožbe:

So možne le takoj po zaključku tekmovanja. Kasneje se rezultati lahko spremenijo le v primeru drastične sodniške napake.

Preklapljanje med angleško in slovensko tipkovnico:

(Velja samo za okolje X windows)

Če po prijavi v okolje X windows desno spodaj ne vidite sličice za izbiro tipkovnice, morate narediti sledeče:

V K-meniju izberete:

`KDE Control Center -> Input Device -> International Keyboard`

In dodaste hrvaško tipkovnico (ker slovenske še ni), potem izberete še zavihek `Startup` in izberete obe možnosti: `Autostart` in `Docked`

Odjavite se iz X okolja in po ponovni prijavi boste desno spodaj imeli možnost izbire tipkovnice.

Izbrana tipkovnica potem velja za celotno X okolje.

Parencodings

Program PARENC.C, PARENC.CPP, PARENC.PAS

Let $S = s_1\ s_2\ \ldots\ s_{2n}$ be a well-formed string of parentheses. S can be encoded in two different ways:

- By an integer sequence $P = p_1\ p_2\ \ldots\ p_n$ where p_i is the number of left parentheses before the ith right parenthesis in S (P-sequence).
- By an integer sequence $W = w_1\ w_2\ \ldots\ w_n$ where for each right parenthesis, say a in S, we associate an integer which is the number of right parentheses counting from the matched left parenthesis of a up to a. (W-sequence).

Following is an example of the above encodings:

S	(((()()())))
P-sequence	4 5 6666
W-sequence	1 1 1456

Write a program to convert P-sequence of a well-formed string to the W-sequence of the same string.

Input

The first line of the input contains a single integer t ($1 <= t <= 10$), the number of test cases, followed by the input data for each test case. The first line of each test case is an integer n ($1 <= n <= 20$), and the second line is the P-sequence of a well-formed string. It contains n positive integers, separated with blanks, representing the P-sequence.

Output

The output consists of exactly t lines corresponding to test cases. For each test case, the output line should contain n integers describing the W-sequence of the string corresponding to its given P-sequence.

Sample Input

```
2
6
4 5 6 6 6 6
9
4 6 6 6 6 8 9 9 9
```

Sample Output

```
1 1 1 4 5 6
1 1 2 4 5 1 1 3 9
```

Prvenstvo v programiranju 2002

poskusno tekmovanje 20. maj 2002

Illusive Chase

Program CHASE.C, CHASE.CPP, CHASE.PAS

Tom the robocat is presented in a Robotics Exhibition for an enthusiastic audience of youngsters, placed around an *m* x *n* field. Tom which is turned off initially is placed in some arbitrary point in the field by a volunteer from the audience. At time zero of the show, Tom is turned on by a remote control. Poor Tom is shown a holographic illusion of Jerry in a short distance such that a direct path between them is either vertical or horizontal. There may be obstacles in the field, but the illusion is always placed such that in the direct path between Tom and the illusion, there would be no obstacles. Tom tries to reach Jerry, but as soon as he gets there, the illusion changes its place and the chase goes on. Let's call each chase in one direction (up, down, left, and right), a *chase trip*. Each trip starts from where the last illusion was deemed and ends where the next illusion is deemed out. After a number of chase trips, the holographic illusion no more shows up, and poor Tom wonders what to do next. At this time, he is signaled that for sure, if he returns to where he started the chase, a real Jerry is sleeping and he can catch it.

To simplify the problem, we can consider the field as a grid of squares. Some of the squares are occupied with obstacles. At any instant, Tom is in some unoccupied square of the grid and so is Jerry, such that the direct path between them is either horizontal or vertical. It's assumed that each time Tom is shown an illusion; he can reach it by moving only in one of the four directions, without bumping into an obstacle. Tom moves into an adjacent square of the grid by taking one and only one step.

The problem is that Tom's logging mechanism is a bit fuzzy, thus the number of steps he has taken in each chase trip is logged as an interval of integers, e.g. 2 to 5 steps to the left. Now is your turn to send a program to Tom's memory to help him go back. But to ease your task in this contest, your program should only count all possible places that he might have started the chase from.

Input

The first line of the input contains a single integer t ($1 <= t <= 10$), the number of test cases, followed by the input data for each test case. The first line of each test case contains two integers m and n, which are the number of rows and columns of the grid respectively ($1 <= m, n <= 100$). Next, there are m lines, each containing n integers which are either 0 or 1, indicating whether the corresponding cell of the grid is empty (0) or occupied by an obstacle (1). After description of the field, there is a sequence of lines, each corresponding to a chase trip of Tom (in order). Each line contains two positive integers which together specify the range of steps Tom has taken (inclusive), followed by a single upper-case character indicating the direction of the chase trip, which is one of the four cases of R (for right), L (for left), U (for up), and D (for down). (Note that these directions are relative to the field and are not directions to which Tom turns). This part of the test case is terminated by a line containing exactly two zeros.

Output

For each test case, there should be a single line, containing an integer indicating the number of cells that Tom might have started the chase from.

Sample Input

```
2
6 6
0 0 0 0 0 0
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 1 0 1
0 0 0 0 0 1
1 2 R
1 2 D
1 1 R
0 0
3 4
0 0 0 0
0 0 0 0
0 0 0 0
1 2 R
3 7 U
0 0
```

Sample Output

```
10
0
```

Pseudo-Random Numbers

RANDOM.C RANDOM.PAS RANDOM.CPP

Computers normally cannot generate really random numbers, but frequently are used to generate sequences of pseudo-random numbers. These are generated by some algorithm, but appear for all practical purposes to be really random. Random numbers are used in many applications, including simulation.

A common pseudo-random number generation technique is called the linear congruential method. If the last pseudo-random number generated was *L*, then the next number is generated by evaluating $(Z \times L + I) \bmod M$, where *Z* is a constant multiplier, *I* is a constant increment, and M is a constant modulus. For example, suppose *Z* is 7, *I* is 5, and *M* is 12. If the first random number (usually called the seed) is 4, then we can determine the next few pseudo-random numbers are follows:

Last random number, L	*(Z x L + I)*	*Next random number, (Z x L + I) mod M*
4	33	9
9	68	8
8	61	1
1	12	0
0	5	5
5	40	4

As you can see, the sequence of pseudo-random numbers generated by this technique repeats after six numbers. It should be clear that the longest sequence that can be generated using this technique is limited by the modulus, *M*.

In this problem you will be given sets of values for *Z*, *I*, *M*, and the seed, *L*. Each of these will have no more than four digits. For each such set of values you are to determine the length of the cycle of pseudo-random numbers that will be generated. But be careful: the cycle might not begin with the seed!

Input

Each input line will contain four integer values, in order, for Z, I, M, and L. The last line will contain four zeroes, and marks the end of the input data. L will be less than M.

Output

For each input line, display the case number (they are sequentially numbered, starting with 1) and the length of the sequence of pseudo-random numbers before the sequence is repeated.

Sample Input

```
7 5 12 4
5173 3849 3279 1511
9111 5309 6000 1234
1079 2136 9999 1237
0 0 0 0
```

Sample Output

```
Case 1: 6
```

```
Case 2: 546
Case 3: 500
Case 4: 220
```

The Cat in the Hat

Program CAT.C,CAT.CPP,CAT.PAS

Background

(An homage to Theodore Seuss Geisel)

> The Cat in the Hat is a nasty creature,
> But the striped hat he is wearing has a rather nifty feature.
>
> With one flick of his wrist he pops his top off.
>
> Do you know what's inside that Cat's hat?
> A bunch of small cats, each with its own striped hat.
>
> Each little cat does the same as line three,
> All except the littlest ones, who just say ``Why me?''
>
> Because the littlest cats have to clean all the grime,
> And they're tired of doing it time after time!

The Problem

A clever cat walks into a messy room which he needs to clean. Instead of doing the work alone, it decides to have its helper cats do the work. It keeps its (smaller) helper cats inside its hat. Each helper cat also has helper cats in its own hat, and so on. Eventually, the cats reach a smallest size. These smallest cats have no additional cats in their hats. These unfortunate smallest cats have to do the cleaning.

The number of cats inside each (non-smallest) cat's hat is a constant, N. The height of these cats-in-a-hat is [1/(N+1)] times the height of the cat whose hat they are in.

> The smallest cats are of height one;
> these are the cats that get the work done.

All heights are positive integers.

Given the height of the initial cat and the number of worker cats (of height one), find the number of cats that are not doing any work (cats of height greater than one) and also determine the sum of all the cats' heights (the height of a stack of all cats standing one on top of another).

The Input

The input consists of a sequence of cat-in-hat specifications. Each specification is a single line consisting of two positive integers, separated by white space. The first integer is the height of the initial cat, and the second integer is the number of worker cats.

A pair of 0's on a line indicates the end of input.

The Output

For each input line (cat-in-hat specification), print the number of cats that are not working, followed by a space, followed by the height of the stack of cats. There should be one output line for each input line other than the ``0 0'' that terminates input.

Sample Input

```
216 125
5764801 1679616
0 0
```

Sample Output

```
31 671
335923 30275911
```

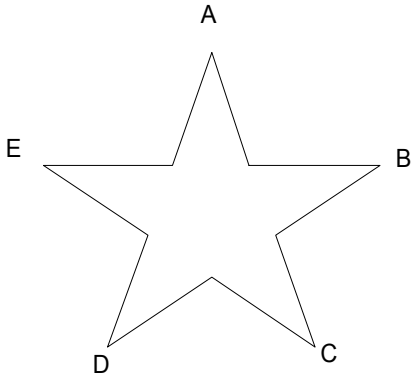
Pointing the Way

Program POINT.C, POINT.CPP, POINT.PAS

A compass is usually thought of as having 4 primary points at North, East, South, and West. These directions correspond to points on a circle, with North at 0 degrees, East at 90 degrees, South at 180 degrees, and West at 270 degrees. There can be many points between the primary points. For example, the point North-East is halfway between North and East, at 45 degrees and East-North-East is halfway between East and North-East at 67.5 degrees.

This plan has been determined to be too restrictive for space travel, so you have been asked to create a more general compass, with anywhere from 3 to 26 primary points.

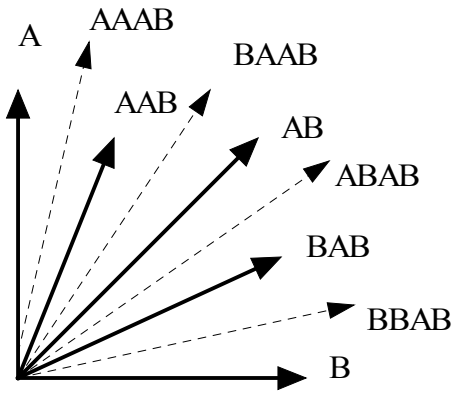
For example, if a compass were to have 5 points, for simplicity sake, name them with the first five letters of the alphabet as below:



The point A is oriented with 0 degrees. This means B is at 72 degrees, C is at 144 degrees, D is at 216 degrees, and E is at 288 degrees. At 36 degrees, halfway between A and B, is the compass point AB. At 18 degrees, halfway between A and AB is the compass point AAB. At 54 degrees, halfway between AB and B is the point BAB.

To name compass points, follow the example in the chart below for a compass with 4 points and the rules below:

– The major compass points are called *1-points*. The name of a 1-point is a single alphabetic character, where the first 1-point is 'A' (at 0 degrees) and others are B, C, ... in clockwise order.



- 2-points are halfway between two one points. The name of the 2-point between two contiguous 1-points, P and Q (specified in clockwise order,) is PQ.
- A *k*-point (where $k >= 3$) bisects an angle determined by a (*k*-1)-point and an *n*-point (where $n < k$-1). For instance, ABAB is a 4-point which bisects the angle of the 3-point BAB and the 2-point AB. The name of a *k*-point between *a* and b is *a*b, where b is the name of the nearest (*k*-1)-point and *a* is the name of the 1-point nearest to *a*b in the direction opposite of b.

Your program must take a number of compass points and a group of compass points for each and return the degree measure of the compass point. For example, for the example above, a 5 pointed compass would place BAB at 54 degrees and CD at 180 degrees.

Input

The input will have information about a number of different input sets. The first line of each set will be a number of compass points, *n*, where 3 <= *n* <= 26. The second line of the data set will have a number of compass points to be analyzed, *k*. The next *k* lines will each have compass point, where each point is a list of between 1 and 8 upper case letters. You may assume each string represents a valid compass point for the given compass. There will be no leading or trailing blanks on a line.

The end of input will be indicated by a compass with 0 points. This data should not be processed.

Output

Output the number of points of the compass at the beginning of the output for each set. For each of the k compass points, echo the compass point and give the degree measurement it represents to 2 decimal places of precision.

Have a blank line after each data set.

Sample Input

```
5
3
B
BBC
AAEA
26
1
W
4
1
BBAB
0
```

Sample Output

```
Compass with 5 points:
B : 72.00 degrees
BBC : 90.00 degrees
AAEA :  351.00 degrees

Compass with 26 points:
W : 304.62 degrees

Compass with 4 points:
BBAB : 78.75 degrees
```