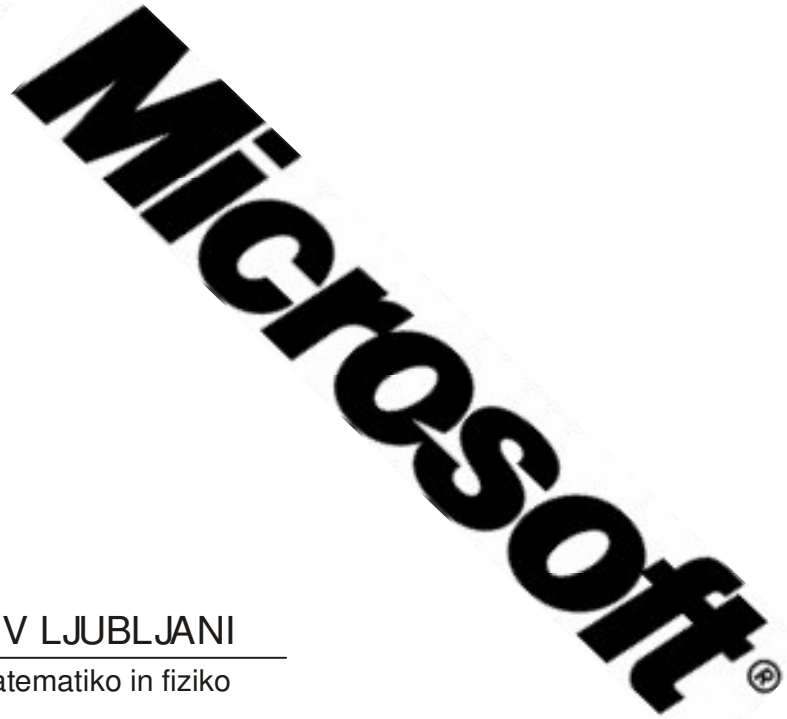
Odprto ekipno prvenstvo UNIVERZE V LJUBLJANI v programiranju

2003

1. krog

Glavni pokrovitelji



UNIVERZA V LJUBLJANI

Fakulteta za matematiko in fiziko

Pokrovitelji

anni

atlantis
V SREDIŠČU INFORMACIJSKEGA SPLETA

avtenta.si

halcom

PCFormat

Peritech

Panda Software
Protivirusne rešitve

MARAND
Napredna računalniška hiša

Milacom

IGEA

IBM

ORIA COMPUTERS

Joker

Šolski servis
Janez Vrankar

Monitor

DATALAB
SPREMENITE PODATKE V DOBIČEK!

dMFA
ZALOŽNIŠTVO

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, kwrite, 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
Java   (*.java)     gcj    primer:   gcj -o program --main=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: `ssh svarun.fmf.uni-lj.si`

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:

Z ukazom v terminalu:

sexkbmap si

setxkbmap us

Nastavitve veljajo za novo odprte programe.

The 3n+1 problem

Program problem.c, problem.cpp, problem.java, problem.pas

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

Consider the following algorithm:

```
1.          input n
2.          print n
3.          if n = 1 then STOP
4.                if n is odd      then  n := 3n + 1
5.                            else  n := n / 2
6.          goto 2
```

Given the input 22, the following sequence of numbers will be printed *22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1*.

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 < n < 1{,}000{,}000$ (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed (including the 1). For a given n this is called the *cycle-length* of n. In the example above, the cycle length of 22 is 16.

For any two numbers i and j you are to determine the maximum cycle length over all numbers between i and j (inclusive).

Input

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j.

Output

For each pair of input integers *i* and *j* you should output *i*, *j*, and the maximum cycle length for integers between and including *i* and *j*. These three numbers should be separated by one space with all three numbers on one line and with one line of output for each line of input. The integers *i* and *j* must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

```
1 10
100 200
201 210
900 1000
```

Sample Output

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

Parking Lot

Program parking.c, parking.cpp, parking.java, parking.pas

At a certain college, a small parking lot is arranged in a rectangular shape, with 20 spaces numbered 1, 2, 3..... 19, 20. Traffic flow is one way in a counter-clockwise direction. The lot looks something like this:

```
     ____________________________________________
    |                                            |
    |  20  19  18  17  16  15  14  13  12  11    |
Exit <<<<  <   <   <   <   <   <   <   <   <     |
    |  v                                   ^     |
Entrance >>>>  >   >   >   >   >   >   >   >     |
    |  1   2   3   4   5   6   7   8   9   10    |
    |____________________________________________|
```

Note that the first position encountered upon entering is 1 and the last is 20. Cars may exit or continue to drive in a counter-clockwise direction. The following assumptions apply to this problem:

- At the start, class is in session and the lot is full (all 20 spaces are occupied by parked cars).
- In addition to the (20) cars already parked in the lot, K autos are in the lot waiting for positions to become available. (1 <= K <= 20)
- Each waiting auto is positioned behind one of the occupied spaces. When a position empties, the space is filled either by the car waiting at that position or, if no car is waiting at that position, by the closest car, bearing in mind that the traffic flow is one way. (There is sufficient room at each position for the car parked in that position to leave and the car waiting at that position to then park.)
- When an auto advances N positions to a free spot, all other cars advance N positions. Since the lot is circular, advancing 4 positions from position 18 means advancing to position 2.
- None of the waiting cars exits.

Input

Input data is in two parts. The first part consists of integers, one per line beginning in column 1, representing initial positions of waiting autos. An integer 99 signals the end of this part of the data. The second part consists of integers, in the same format, representing positions vacated.

Positions are vacated in the order in which their numbers appear in the second part of the data. Read these until there's no more input.

Output

The output of your program should consist a series of lines giving, for each initial (waiting) car position, the initial position and the final position of that car based on the description and assumptions stated above. The output lines must appear in the same order as the order of the initial positions given in the input.

Sample input

```
6
19
17
13
1
99
1
3
20
16
```

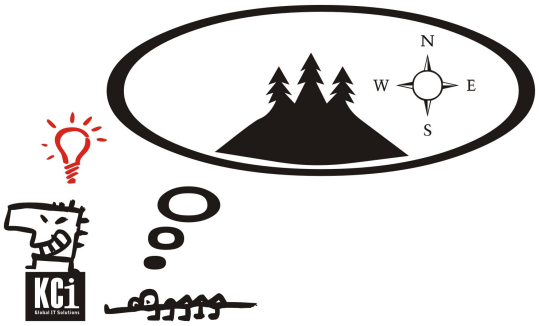
Sample output

```
Original position 6 parked in 16
Original position 19 parked in 3
Original position 17 did not park
Original position 13 parked in 20
Original position 1 parked in 1
```

Relief [KCi naloga]

Program relief.c, relief.cpp, relief.java, relief.pas

Kartografi pri sestavljanju zemljevida najprej sestavijo višinsko karto relifa. Višinska karta je namišljena pravokotna mreža velikosti n x m nad pokrajino, z izmerjeno višino na vsakem oglišču mreže (včasih so višine dobili z merjenjem kotov, danes pa jih natančneje izmerijo sateliti).



Naslednji korak je identifikacija *značilnosti reliefa* – gorovij in dolin. V našem modelu bodo vsa gorovja in doline ravne črte, ki ležijo v eni izmed naslednjih smeri: sever – jug, vzhod – zahod, severovzhod – jugozahod in severozahod – jugovzhod. Točka je del gorovja, če je njena višina večja ali enaka podanemu minimalnemu pragu gorovja in je del doline, če je njena višina manjša ali enaka maksimalnemu pragu doline. Značilnost terena mora biti daljša od dane vhodne vrednosti $d > 1$ (torej vsak kucelj še ni gorovje). V naslednjem primeru je prag gorovja 0.8km, prag doline 0.2km in $d = 3$. Vidimo, da obstajata dve dolini in eno gorovje:

```
0.4 0.2  0.21 0.57 0.9
0.9 0.85 0.8  0.15 0.1
0.8 0.3  0.2  0.21 0.05
0.4 0.1  0.8  0.85 0.15
```

Vhodni in izhodni podatki

Podatki bodo v naslednji obliki: v prvi vrsti je velikost mreže (n = zahod—vzhod in m = sever—jug), minimalna dolžina značilnosti d, ter prag gorovja in prag doline. V naslednjih m vrstah sledi višinska karta reliefa.

Poišči značilnosti reliefa ter za vsako značilnost izpiši začetno in končno koordinato ter tip značilnosti (dolina, gorovje). Koordinati naj se izpišeta v vrstnem redu sever—jug, zahod—vzhod.

Primer vhodnih podatkov

```
5 4 3 0.8 0.2
0.4 0.2  0.21 0.57 0.9
0.9 0.85 0.8  0.15 0.1
0.8 0.3  0.2  0.21 0.05
0.4 0.1  0.8  0.85 0.15
```

Primer izhodnih podatkov

```
1 2 3 2 gorovje
5 2 5 4 dolina
2 4 4 2 dolina
```

Artificial Intelligence?

Program artificial.c, artificial.cpp, artificial.java, artificial.pas

Physics teachers in high school often think that problems given as text are more demanding than pure computations. After all, the pupils have to read and understand the problem first!

So they don't state a problem like *"U=10V, I=5A, P=?"* but rather like *"You have an electrical circuit that contains a battery with a voltage of U=10V and a light-bulb. There's an electrical current of I=5A through the bulb. Which power is generated in the bulb?"*.

However, half of the pupils just don't pay attention to the text anyway. They just extract from the text what is given: *U=10V, I=5A*. Then they think: "Which formulae do I know? Ah yes, *P=U*I*. Therefore *P=10V*5A=500W*. Finished."

OK, this doesn't always work, so these pupils are usually not the top scorers in physics tests. But at least this simple algorithm is usually good enough to pass the class. (Sad but true.)

Today we will check if a computer can pass a high school physics test. We will concentrate on the P-U-I type problems first. That means, problems in which two of power, voltage and current are given and the third is wanted.

Your job is to write a program that reads such a text problem and solves it according to the simple algorithm given above.

Input

The first line of the input file will contain the number of test cases.

Each test case will consist of one line containing exactly two data fields and some additional arbitrary words. A data field will be of the form I=xA, U=xV or P=xW, where x is a real number.

Directly before the unit (A, V or W) one of the prefixes m (milli), k (kilo) and M (Mega) may also occur. To summarize it: Data fields adhere to the following grammar:

```
DataField ::= Concept '=' RealNumber [Prefix] Unit
Concept   ::= 'P' | 'U' | 'I'
Prefix    ::= 'm' | 'k' | 'M'
Unit      ::= 'W' | 'V' | 'A'
```

Additional assertions:

- The equal sign (`=') will never occur in an other context than within a data field.
- There is no whitespace (tabs,blanks) inside a data field.
- Either P and U, P and I, or U and I will be given.

Output

For each test case, print three lines:

- a line saying ``Problem #k" where k is the number of the test case

- a line giving the solution (voltage, power or current, dependent on what was given), written without a prefix and with two decimal places as shown in the sample output
- a blank line

Sample Input

```
3
If the voltage is U=200V and the current is I=4.5A, which power is generated?
A light-bulb yields P=100W and the voltage is U=220V. Compute the current,
please.
bla bla bla lightning strike I=2A bla bla bla P=2.5MW bla bla voltage?
```

Sample Output

```
Problem #1
P=900.00W

Problem #2
I=0.45A

Problem #3
U=1250000.00V
```

Expressions

Program expressions.c, expressions.cpp, expressions.java, expressions.pas

Let X be the set of *correctly built parenthesis expressions*. The elements of X are strings consisting only of the characters '(' and ')'. The set X is defined as follows:

- an empty string belongs to X
- if A belongs to X, then (A) belongs to X
- if both A and B belong to X, then the concatenation AB belongs to X.

For example, the following strings are correctly built parenthesis expressions (and therefore belong to the set X):

```
()(())()
(()(()))
```

The expressions below are not correctly built parenthesis expressions (and are thus not in X):

```
(()))(()
())(()
```

Let E be a correctly built parenthesis expression (therefore E is a string belonging to X).

The *length* of E is the number of single parenthesis (characters) in E.

The *depth* D(E) of E is defined as follows:

```
      ⎧ 0 if E is empty
D(E)= ⎨ D(A)+1 if E = (A), and A is in X
      ⎩ max(D(A),D(B)) if E = AB, and A, B are in X
```

For example, the length of "()(())()" is 8, and its depth is 2.

What is the number of correctly built parenthesis expressions of length n and depth d, for given positive integers n and d? For example, there are exactly three correctly built parenthesis expressions of length 6 and depth 2:

(())()
()(())
(()())

Write a program which computes the number of correctly built parenthesis expressions of length n and depth d.

Input data

Input consists of lines of pairs of two integers - n and d, at most one pair on line, $2 \le n \le 300$, $1 \le d \le 150$.

Output data

For every pair of integers in the input write single integer on one line - the number of correctly built parenthesis expressions of length n and depth d. The output number has at most 9 digits.

Sample Input

```
6 2
300 150
```

Sample Output

```
3
1
```