Problems Index Tue Oct 15 22:14:49 EDT 2013

BOSPRE 2013 PROBLEMS
------ ---- --------

The problems are in approximate order of difficulty,
easiest first.

problems/soundex
If it sounds alike, it is alike.
Boston Preliminary 2013

problems/transition
Population pop.
Boston Preliminary 2013

problems/pulleys
How long is your belt?
Boston Preliminary 2013

problems/crates
Nesting is not just for the birds.
Boston Preliminary 2013

problems/onlinestring
Hop, skip, and jump your way through.
Boston Preliminary 2013

problems/localcolor
Myopia is tough.
Boston Preliminary 2013

problems/eulerian
Bigger cycles are better cycles.
Boston Preliminary 2013

problems/polyellipse
It takes two to tangle.
Boston Preliminary 2013

American SOUNDEX

SOUNDEX is a system of translating words into crude phonetic approximations that may be used to suggest spellings for misspelled words. For example, ‘Tymczak’ and ‘Timsack’ both translate to ‘T522’, and so if a writer writes ‘Timsack’ and that is not in the dictionary, but ‘Tymczak’ is, a spell checker can suggest ‘Tymczak’ as a possible correct spelling.

The SOUNDEX rules for translating a word are as follows:

(1) Apply a translation table to translate every character to a digit, -, or *.
(2) Delete all occurrences of *.
(3) Delete any digit that follows an identical digit. (Alternatively, replace any string of identical digits by a single digit.)
(4) Delete all occurrences of -.
(5) If the first letter of the original word translated to a digit, replace that digit, which now begins the translated word, by the letter; otherwise prepend the letter to the translated word (prepend is to ‘beginning’ as append is to ‘end’).
(6) The translated word now consists of a letter followed by zero or more digits. Discard all digits but the first three. If there are less than 3 digits, append ‘0’s until there are 3 digits.

The translation table used for American English is

    a, e, i, o, u, and y each translate to -
    h and w each translate to *
    b, f, p, and v each translate to 1
    c, g, j, k, q, s, x, and z each translate to 2
    d and t each translate to 3
    l translates to 4
    m and n each translate to 5
    r translates to 6

If you study these rules, you will find that sequences of similar consonants are translated to a single digit unless they are separated by vowels, vowels are otherwise ignored, and ‘h’ and ‘w’ are completely ignored. In addition, the first character is kept, and only the first three dissimilar or vowel separated consonant sequences are kept.

Input

For each of several test cases, one line containing just the word to be translated. To keep things simple, each word contains only lower case letters and has at most 25 letters.

Input ends with an end of file.

Output

For each test case, one line of the form

        WORD-TO-BE-TRANSLATED => TRANSLATED-WORD

```
Sample Input
------ -----

robert
rupert
rubin
ashcraft
ashcroft
tymczak
timsack
pfister
how

Sample Output
------ ------

robert => r163
rupert => r163
rubin => r150
ashcraft => a261
ashcroft => a261
tymczak => t522
timsack => t522
pfister => p236
how => h000

Tips:
----

Input consists of lines read from the standard input.
Input ends when an end-of-file is read from the standard
input.  Output consists of lines written to the standard
output.  For example input/output code see

    ~/demos/solutions/reverser/reverser.EXT

where EXT is c, cc, or java.
```

```
You may find it beneficial to add code so that if your
program is being run in debug mode it outputs

    WORD => TWORD1 => TWORD2 => ... => TWORD6

where WORD is the original word, TWORD1 the word after
the first translation step, TWORD2 the word after the
second step, and TWORD6 the word after the 6'th and
final step.  Some examples from 'make debug' using the
judge's solution are

robert => 6-1-63 => 6-1-63 => 6-1-63 => 6163
    => r163 => r163
rubin => 6-1-5 => 6-1-5 => 6-1-5 => 615
    => r15 => r150
ashcraft => -2*26-13 => -226-13 => -26-13 => 2613
    => a2613 => a261
tymczak => 3-522-2 => 3-522-2 => 3-52-2 => 3522
    => t522 => t522
pfister => 11-23-6 => 11-23-6 => 1-23-6 => 1236
    => p236 => p236
how => *-* => - => - =>
    => h => h000

where we added some extra line feeds to conform to this
current document's 56 column width limit.
```

```
Reference
---------

    http://en.wikipedia.org/wiki/Soundex


File:      soundex.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Mon Oct  7 11:18:31 EDT 2013

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.

RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2013/10/07 15:19:23 $
    $RCSfile: soundex.txt,v $
    $Revision: 1.6 $
```

Demographic Transition

Almost all countries have undergone or are undergoing a 'demographic transition' in which the population explodes because improved public health measures and medicine cause the death rate to plummet but the birth rate remains high for a time before dropping to match the death rate. You have been asked to compute the approximate start and stop years of a country's demographic transition, and the multiplier of the transition, which is defined as the population at the stop year divided by the population at the start year.

To compute the start and stop years, take the yearly growth averaged over every 10 year period, define periods with average growth of at least 1% as 'high growth', take the first year of the first high growth period as the start year and the LAST(!!) year of the last high growth period as the stop year.

A country can have only one demographic transition (at least in historic times) which begins when improved public health measures and medicine cause the death rate to go down. But not all periods within a demographic transition are high growth, and in fact, population may decrease within the transition period due to famine or war.

To be a bit more precise, if PA is the population for year YA, the first of the 10 year period, and PB the population for year YB = YA + 10, then the period is high growth if and only if (1.01)**10 <= PB/PA. Note YA and YB are the endpoints of a sequence of 11 (not 10) year/population pair values, and YB is the LAST year of the high growth period.

However, it is uncertain whether 10 years is the right period duration and whether 1% is the right growth level for defining 'high growth periods', so these need to be parameters that can be changed.

Input

For each of several test cases, first a line containing just the test case name (which typically is the country name), and then one line containing the raw data and having the format

```
Y N P1 P2 P3 ... PN
```

This specifies the populations P1, ..., PN for N years beginning with year Y (thus the last year is Y + N - 1). This line is followed by one or more lines of the form

```
G D
```

where a high growth period is defined as a period of duration D consecutive years with effective yearly growth G %. The lines of this last format are followed by a line containing just the character '*'.

To be precise, if P[YA] denotes the population at year YA, a D year period beginning at year YA and ending at year YA + D is high growth if and only if (1+G/100)**D <= P[YA+D]/P[YA].

The input is such that

```
0 <= Y <= Y + N - 1 <= 3,000
2 <= N <= 3,000
1 <= D <= N - 1
0 <= G <= 100
0 <= Pi <= 2,000,000 for i = 1, 2, ..., N
```

Populations are actual populations in units of 1,000
of the country named in the test case name line.

All input numbers are integers except G, which is
floating point. The line containing the raw population
data may be very long, but other lines will be at most
80 characters.

Input ends with an end of file.

Output

For each test case, first a copy of the test case name
input line, then a line containing

        data is for years Y1 through Y2

where

    Y1 equals Y, the first year of raw input data
    Y2 equals Y + N - 1, the last year of raw input data

Lastly for each input line of the form

        G D

one output line of the form

    M: YA/PA through YB/PB: G% for D years

where

    M is the ratio PB / PA,
      with exactly 2 decimal places
    YA is the start year, the first year of the first
       high growth period
    PA is the population at the start year
    YB is the stop year, the last year of the last
       high growth period
    PB is the population at the stop year
    D, G are from the input line,
       but G is printed with exactly 2 decimal places

However, if there were no high growth periods, output
instead a line of the form

    no transition found: G% for D years

Note that you may get YA == Y1, which suggests that the
demographic transition actually starts before the given
data, or YB == Y2, which suggests that the transition is
not yet finished.

```
Sample Input
------ -----

-- France --
1820 190 31250 31460 31685 [... see sample.in]
1.5 10
1.2 10
1.0 10
0.8 10
0.7 10
0.6 20
0.5 20
*
-- Germany --
1820 190 24905 25260 25620 [... see sample.in]
1.5 10
1.2 10
1.0 10
0.8 10
0.7 10
0.6 20
0.5 20
*
```

```
Sample Output
------ ------

-- France --
data is for years 1820 through 2009
no transition found: 1.50% for 10 years
no transition found: 1.20% for 10 years
1.34: 1943/39000 through 1971/52432: 1.00% for 10 years
1.37: 1942/39400 through 1975/53955: 0.80% for 10 years
1.37: 1941/39600 through 1977/54378: 0.70% for 10 years
1.38: 1940/41000 through 1986/56725: 0.60% for 20 years
2.06: 1820/31250 through 2009/64420: 0.50% for 20 years
-- Germany --
data is for years 1820 through 2009
1.21: 1894/49703 through 1907/60341: 1.50% for 10 years
1.44: 1887/46001 through 1915/66230: 1.20% for 10 years
2.65: 1820/24905 through 1916/66076: 1.00% for 10 years
2.83: 1820/24905 through 1956/70603: 0.80% for 10 years
3.12: 1820/24905 through 1970/77783: 0.70% for 10 years
3.17: 1820/24905 through 1974/78966: 0.60% for 20 years
3.14: 1820/24905 through 1976/78299: 0.50% for 20 years


Tips:
----

Input consists of lines read from the standard input.
Input ends when an end-of-file is read from the standard
input.  Output consists of lines written to the standard
output.  For example input/output code see

    ~/demos/solutions/summer/summer.EXT

where EXT is c, cc, or java.
```

If you want to you can compute the growth g of any D year period using the formulae

```
    double m = (double) P[YA+D] / P[YA];
    double g = 100 * ( exp ( log ( m ) / D ) - 1 );
```

Some output from running ‘make debug’ on the judge’s solution using this formula is

```
-- France --
data is for years 1820 through 2009
10 YEAR PERIOD GROWTHS:
 1820: 0.64 0.61 0.59 0.55 0.53 0.50 0.49 0.49 0.48 0.48
 1830: 0.47 0.47 0.47 0.48 0.49 0.49 0.47 0.46 0.44 0.42
 1840: 0.41 0.40 0.39 0.38 0.37 0.36 0.34 0.32 0.30 0.28
 1850: 0.26 0.25 0.25 0.26 0.27 0.27 0.27 0.30 0.31 0.43
 1860: 0.30 0.09 0.04 0.05 0.05 0.05 0.08 0.09 0.11 0.00
 1870: 0.16 0.38 0.43 0.41 0.41 0.39 0.37 0.34 0.29 0.28
 1880: 0.25 0.20 0.17 0.14 0.11 0.09 0.08 0.11 0.14 0.13
 1890: 0.14 0.16 0.18 0.19 0.20 0.20 0.19 0.15 0.14 0.14
 1900: 0.15 0.16 0.16 0.16 0.15-0.10-0.26-0.41-0.63-0.60
 1910:-0.55-0.51-0.48-0.39-0.28 0.03 0.24 0.41 0.63 0.64
 1920: 0.65 0.65 0.60 0.49 0.40 0.32 0.25 0.24 0.22 0.16
 1930:-0.15-0.55-0.60-0.71-0.75-0.55-0.39-0.30-0.20-0.10
 1940: 0.36 0.79 0.92 1.10 1.20 1.08 1.03 1.05 1.05 1.07
 1950: 0.92 0.95 1.08 1.16 1.20 1.20 1.19 1.16 1.12 1.10
 1960: 1.09 1.07 0.96 0.89 0.85 0.80 0.75 0.71 0.68 0.64
 1970: 0.60 0.55 0.52 0.48 0.47 0.46 0.46 0.47 0.48 0.53
 1980: 0.54 0.55 0.55 0.56 0.55 0.56 0.56 0.56 0.56 0.51
 1990: 0.50 0.50 0.49 0.50 0.51 0.52 0.54 0.55 0.57 0.58
1.34: 1943/39000 through 1971/52432: 1.00% for 10 years
```

Commentary

Populations are estimated for European countries from 1820 on by Madison at

    www.ggdc.net/maddison/Historical_Statistics/
                            horizontal-file_02-2010.xls

and for other countries from 1950 on by the UN at

    esa.un.org/unpd/wpp/Excel-Data/EXCEL_FILES/
      1_Population/
      WPP2012_POP_F01_1_TOTAL_POPULATION_BOTH_SEXES.XLS

It is actually not easy to define the start and stop years of the demographic transition in a meaningful way. The data needed to precisely define the start is generally unavailable. Populations also grow as the food supply grows (as happened when the amount of land under cultivation grew in America), and may shrink as the culture changes (as is happening now in Europe), so the demographic transition is superimposed on a slower long-term growth curve.

We have kept this problem simple even though the results our solutions compute are not that close to the best that can be computed. The book

    A Concise History of World Population,
    by Massimo Livi-Bacci, English Translation, 2012

gives the following data (taken from a paper by J,-C. Chesnais) on page 118:

Country	Start and Stop Year	Multiplier
Sweden	1810 - 1960	3.83
Germany	1876 - 1965	2.11
USSR	1896 - 1965	2.05
France	1785 - 1970	1.62
China	1930 - 2000	2.46
Mexico	1920 - 2000	7.02

File: transition.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Tue Oct 15 15:18:13 EDT 2013

RCS Info (may not be true date or author):

$Author: walton $
$Date: 2013/10/15 19:48:46 $
$RCSfile: transition.txt,v $
$Revision: 1.16 $

Pulley Belts
------ -----

You have been asked to determine the length of a belt that connects two pulleys from the XY coordinates of the pulley centers and the radii of the pulleys.

Input

For each of several test cases, first one line containing just the test case name, and then one line with six floating point numbers in the format:

```
X1 Y1 R1 X2 Y2 R2
```

where the first pulley has center at (X1,Y1) and radius R1, and the second pulley has center at (X2,Y2) and radius R2.

```
-10,000 <= X1, Y1, X2, Y2 <= +10,000
0 < R1, R2 <= 1,000
distance between (X1,Y1) and (X2,Y2) > R1 + R2
```

Input ends with an end of file.

Output

For each test case, first a copy of the test case name line from the input, and then one line with the belt length as a floating point number with exactly 3 decimal places. The belt may NOT cross itself.

Sample Input
------ -----

```
-- SHOULD BE PI + 10 --
0 0 0.5 0 5 0.5
-- SHOULD BE PI + 10 --
8 -7.5 0.5 4 -4.5 0.5
-- SHOULD BE NEAR PI --
0 0 0.5 0 0.500002 0.000001
-- RANDOM SAMPLE 1 --
8.56 -0.23 3.289 -5.738 17.290 7.387
-- RANDOM SAMPLE 2 --
356 782 256 753 -219 74
```

Sample Output
------ ------

```
-- SHOULD BE PI + 10 --
13.142
-- SHOULD BE PI + 10 --
13.142
-- SHOULD BE NEAR PI --
3.142
-- RANDOM SAMPLE 1 --
79.512
-- RANDOM SAMPLE 2 --
3221.263
```

Tips

Input consists of lines read from the standard input. Input ends when an end-of-file is read from the standard input. Output consists of lines written to the standard output. For example input/output code see

    ~/demos/solutions/summer/summer.EXT

where EXT is c, cc, or java.

To solve the problem you, of course, draw a diagram, draw one or more extra lines, and apply the fundamental ideas of plane geometry and trigonometry to compute a bunch of line segment lengths and angles. Thus you can add debugging code to print these. Some of what is output when we run 'make debug' on the judge's solution is:

```
-- SHOULD BE PI + 10 --
... |BD| = 5 CAB = 1.5708 ACD = 1.5708
13.142
-- SHOULD BE PI + 10 --
... |BD| = 5 CAB = 1.5708 ACD = 1.5708
13.142
-- SHOULD BE NEAR PI --
... |BD| = 0.00173205 CAB = 0.0034641 ACD = 3.13813
3.142
```

where A is the center of pulley 1, B is the point where the belt is tangent to pulley 1, C is the center of pulley 2, D is the point where the belt is tangent to pulley 2, and ... denotes output we omitted because it might tell you where we drew the extra lines.

```
File:      pulleys.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct 15 20:33:08 EDT 2013
```


RCS Info (may not be true date or author):

```
    $Author: walton $
    $Date: 2013/10/16 00:37:35 $
    $RCSfile: pulleys.txt,v $
    $Revision: 1.7 $
```

Nested Crates
------ ------

Given the length, width, and depth of each of a set of crates, determine the maximum number that can be nested inside each other. Because the foam used to pack crates inside each other consists of rectilinear blocks, a nested crate must have sides parallel to its containing crate. Thus a crate fits in another if and only each dimension of the smaller is less than the corresponding dimension of the larger (where crates can be rotated, of course).

Input

For each of several test cases, first one line containing just the test case name, and then one line containing just the number N of crates, and then N lines each containing

H W D

where H, W, and D are the height, width, and depth of one crate. All numbers are integers. Some of the crates are very small and some very large; a bit strange, but true.

2 <= N <= 100
1 <= H,W,D <= 1,000

Input ends with an end of file.

Output

For each test case, first a copy of the test case name line from the input, and then one line containing just the maximum number of crates that can be nested inside each other. Note that the largest crate in a set of nested crates is counted, so if no two crates nest, the answer is ‘1’.

Sample Input
------ -----

```
-- SAMPLE 1 --
2
1 1 1
4 6 1
-- SAMPLE 2 --
4
1 1 1
2 2 2
3 3 3
1 2 3
-- SAMPLE 3 --
5
2 2 2
3 3 3
4 4 4
2 2 3
1 1 2
```

```
Sample Output
------ ------

-- SAMPLE 1 --
1
-- SAMPLE 2 --
3
-- SAMPLE 3 --
3


Tips
----

Input consists of lines read from the standard input.
Input ends when an end-of-file is read from the standard
input.  Output consists of lines written to the standard
output.  For example input/output code see

    ~/demos/solutions/summer/summer.EXT

where EXT is c, cc, or java.

You can add debugging code that prints which crates
can be nested inside which other crates.  The output
of running ‘make debug’ on the judge’s solution is:
```

```
-- SAMPLE 1 --
(1,1,1)
1
-- SAMPLE 2 --
(1,1,1)<(2,2,2)
(1,1,1)<(3,3,3)
(2,2,2)<(3,3,3)
(1,1,1)<(2,2,2)<(3,3,3)
3
-- SAMPLE 3 --
(2,2,2)<(3,3,3)
(2,2,2)<(4,4,4)
(3,3,3)<(4,4,4)
(2,2,3)<(4,4,4)
(1,1,2)<(3,3,3)
(1,1,2)<(4,4,4)
(1,1,2)<(2,2,3)
(1,1,2)<(3,3,3)<(4,4,4)
3

Here the judge, being a bit of a debugging fanatic, has
taken the trouble to compute and output one of the
possible maximum nestings, which is output on the line
just before the maximum nesting length.
```

```
File:      crates.txt
Author:    Shai Simonson <shai@stonehill.edu>
           with minor revisions by
           Bob Walton <walton@seas.harvard.edu>
Date:      Sat Oct  5 06:10:25 EDT 2013

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.

RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2013/10/05 12:05:21 $
    $RCSfile: crates.txt,v $
    $Revision: 1.3 $
```

Online String Matching

The ‘online string matching’ problem is finding all the substrings of a text string that match a given pattern string, assuming the pattern is given first, and is much shorter than the text string, or one pattern is matched against many text strings, so the pattern string may be preprocessed to optimize the match.

Assume there is a single text string t of length n with characters t[0], ..., t[n-1], a pattern string p of length m with characters p[0], ..., p[m-1], and you are asked to find all s >= 0 such that t[s,...,s+m-1] is an exact match to p. Here s is called the ‘shift’, and you are asked to find all the ‘matching shifts’, i.e., all s for which p matches t[s,...,s+m-1].

A crude algorithm will check all values of s with simple substring comparison and have at worst case O(mn) character comparisons. The Knuth algorithm improves this by having exactly n character table lookups, but there are algorithms with O(mn) worst case character comparisons or table lookups that outperform the Knuth algorithm in typical real world cases. You are being asked to code and evaluate several of these.

All of the algorithms you are being asked to evaluate work by comparing characters from the end of the pattern p toward the beginning of p. When this comparison is completed, either by a character mis-match or by exhaustion of p when s is a matching shift, s is incremented by a value that is a function of p and one or two characters read from t. This shift increment is taken from a precomputed table S indexed by the characters read.

For example, if for particular p and t the first character comparison for each s always fails, c = t[s+m-1] is the character looked up in S, and c is never in p at all, then the shift increment can always be m, and there will be only n/m character comparisons and n/m table lookups, for a total of 2n/m operations.

The ways of computing S that you are to study are-

1. Horspool Algorithm. c = t[s+m-1]. j = S[c] is chosen as the smallest j such that 1 <= j < m and p[m-1-j] = c, or j = m.

2. Quick-Search Algorithm. c = t[s+m]. j = S[c] is chosen as the smallest j such that 1 <= j <= m and p[m-j] = c, or j = m+1.

3. Berry-Ravindran Algorithm. c1 = t[s+m-1], c2=t[s+m]. j = S[c1,c2] is the smallest j such that 1 <= j < m and p[m-1-j] = c1 and p[m-j] = c2, or j = m and p[0] = c2, or j = m+1.

You are given pairs t and p are are being asked to compute the number of character comparisons plus the number of table lookups required for each pair and each algorithm to find all the matching shifts. The Knuth algorithm whose number of comparisons is always 0 and whose number of table lookups is always n is also to be included in the output.

Note that we assume that for each s you begin with a comparison of t[s+m-1] and p[m-1] and continue with comparisons from right to left until you get a mismatch or p runs out of characters. We also assume that initially s = 0 and you stop only when the next comparison or table lookup cannot be done because it would require reading a character beyond the end of t.

Input

For each of several test cases, first a line containing just the test case name, then one or more lines containing the text t, followed by a line containing just 'END', and then one or more lines each containing a pattern p, followed by a line containing just 'END'. Thus each test case has a single text and one or more patterns.

The text t is made by putting a line feed at the end of each text line and concatenating the lines, so the lines are separated by line feeds and the last line ends in a line feed. Each pattern is on a line by itself.

The maximum length of every line is 80 characters, and the total maximum length of the text t with line feeds is 1,000,000 characters. Each line contains only graphic ASCII characters (ASCII characters with codes in the range 33 .. 126) or single spaces (ASCII code 32). None of the texts or patterns are empty.

Input ends with an end of file.

Output

For each test case, first a line containing the test case name, and then for each pattern line 6 lines, first an exact copy of the pattern input line, and then the following 5 lines:

    Pattern Length = #, Text Length = #, Matches = #
    Knuth: # = # comparisons + # lookups
    Horspool: # = # comparisons + # lookups
    Quick-Search: # = # comparisons + # lookups
    Berry-Ravindran: # = # comparisons + # lookups

where # denotes an integer >= 0. The first two #'s are m and n respectively, and the 4'th though 6'th #'s are n, 0, and n respectively. The 'Matches' # is the number of matching shifts and can be computed by any of the three algorithms being studied, as all three algorithms agree.

Sample Input
------ -----

```
-- SAMPLE 1 --
abcaaaabcaaaabc
abcaaabcaaabc
END
abc
aaa
END
-- SAMPLE 2 --
Hopping hippers flipping,
Flipping floppers sticking,
Sticking stoppers clicking,
Clicking cloppers hopping.
END
ing
pers
ick
END
```

```
Sample Output
------ ------

-- SAMPLE 1 --
abc
Pattern Length = 3, Text Length = 30, Matches = 6
Knuth: 30 = 0 comparisons + 30 lookups
Horspool: 38 = 25 comparisons + 13 lookups
Quick-Search: 32 = 22 comparisons + 10 lookups
Berry-Ravindran: 32 = 22 comparisons + 10 lookups
aaa
Pattern Length = 3, Text Length = 30, Matches = 6
Knuth: 30 = 0 comparisons + 30 lookups
Horspool: 51 = 35 comparisons + 16 lookups
Quick-Search: 48 = 35 comparisons + 13 lookups
Berry-Ravindran: 41 = 30 comparisons + 11 lookups
-- SAMPLE 2 --
ing
Pattern Length = 3, Text Length = 109, Matches = 8
Knuth: 109 = 0 comparisons + 109 lookups
Horspool: 94 = 55 comparisons + 39 lookups
Quick-Search: 78 = 47 comparisons + 31 lookups
Berry-Ravindran: 78 = 47 comparisons + 31 lookups
pers
Pattern Length = 4, Text Length = 109, Matches = 4
Knuth: 109 = 0 comparisons + 109 lookups
Horspool: 74 = 43 comparisons + 31 lookups
Quick-Search: 64 = 39 comparisons + 25 lookups
Berry-Ravindran: 61 = 37 comparisons + 24 lookups
ick
Pattern Length = 3, Text Length = 109, Matches = 4
Knuth: 109 = 0 comparisons + 109 lookups
Horspool: 84 = 46 comparisons + 38 lookups
Quick-Search: 66 = 37 comparisons + 29 lookups
Berry-Ravindran: 66 = 37 comparisons + 29 lookups
```

```
Reference
---------

For a whole host of online string matching algorithms
see

   The Exact Online String Matching Problem:
   A Review of the Most Recent Results,
   Simone Faro and Thierry Lecroq,
   ACM Computing Surveys, vol 45, no 2, 2013.


File:      onlinestring.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Mon Oct 14 02:27:24 EDT 2013

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.

RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2013/10/14 06:44:06 $
    $RCSfile: onlinestring.txt,v $
    $Revision: 1.11 $
```

Local Color

You have been asked to write a program that will run as a local algorithm in a network of processors and improve a coloring of the network graph.

A network graph is an undirected graph. The vertices are called nodes and the edges are called connections. Two nodes sharing a connection are neighbors. The number of connections attached to a node is the degree of the node, and the degree of the network, denoted by D, is the maximum degree of any node. If a node has degree d, we identify its connections by integers in the range 0, ..., d-1. The number of nodes in the network is N, and we identify nodes by integers in the range 0, ..., N-1.

A coloring is an assignment of a color to each node such that NO neighbors have the same color. If there are C colors, we represent the colors by integers in the range 0, ..., C-1.

The problem is that given a coloring with C > D+1 colors, improve the coloring to one with at most D+1 colors, using a local distributed algorithm.

A local distributed algorithm is a deterministic program code that runs at each node and executes in K cycles. At the beginning of each cycle but the first each node receives a message from each of its neighbors. The node then computes and for each cycle but the last, outputs a message for each of its neighbors. At the beginning of the first cycle each node receives a single input message and at the end of the last cycle each node outputs a single output message.

Your program executes one cycle at one node, and is invoked by the judge's 'monitor' program K*N times to perform the local distributed computation and produce an answer.

Importantly, in our problem K is set equal to C, the number of original colors.

Messages are single lines of ASCII text whose maximum length is 1,000 characters. Each node has a memory that is represented as a message from the node to itself, i.e., is output by the node at the end of a cycle and input by the node at the beginning of its next cycle.

Your program is called by a monitor program in order to execute each cycle of each node. The monitor program, which is named 'localcolor_network', is provided by the judge. Your program is named 'localcolor'. You can invoke these programs by the command

```
localcolor_network localcolor
```

Input to Your Program

For each cycle of each node, first a line containing the following information:

```
N K node_id node_degree cycle_number
```

where N is the total number of nodes, K the total number of cycles, the N nodes have node_id's in the range 0, ..., N-1, and the K cycles are number 0, ..., K-1.

Then for the first cycle (cycle 0) a single line.

```
    C initial_node_color
```

where C is the initial number of node colors and the node is given an initial color in the range 0, ..., C-1.

For cycles other than the first, the above first input line is followed by lines containing messages output by the previous cycle. The first of these is a line that contains the contents of the memory of the node at the end of the previous cycle (the message from the node to itself). This line is followed by a line for each connection containing the message from the connection's neighbor. These connection message lines are in order of the connection number relative to the node, from 0 through node_degree-1.

You define the format of all message lines. Message lines may not be longer than 1,000 characters and must contain only printable ASCII characters and single spaces (they need to be printable to debug, and they are not permitted to contain form feeds, line feeds, tabs, or any control character other than single space).

Input ends with an end of file, at which point your program should terminate (as there are no more test cases).

Output from Your Program

The output from your program for all cycles but the last is a sequence of message lines. First, the line containing the contents of the node's memory (the message from the node to itself). Then for each connection a line containing the message output by the cycle for the connection. These lines are in order of the connection number, from 0 through node_degree-1. Lastly one line containing just 'END' (used to detect bugs).

For the last cycle of an algorithm execution output only one line containing just an integer in the range 0, ..., D that is the color of the node after the algorithm has completed, followed by one line containing just 'END'. The monitor program will check that two neighbors do not have the same color at this point, and that all colors are in the range 0,..., D. Your algorithm will be declared to be successful for the test case if this is so.

Input to the Monitor

For each test case, first a line containing just the test case name. Then a second line containing

```
        N D C SEED
```

where N is the number of nodes, D the maximum node degree, C is the number of initial colors, and SEED is a 9-digit unsigned integer that is the seed of a pseudo-random number generator used by the monitor to generate the graph.

```
        2 <= D <= 20
        D + 1 <= N <= 1,000
        D + 1 < C <= 100

Input ends with an end of file.


Output from the Monitor
------ ---- --- -------

For each test case, first a copy of the test case input
lines, and then a line containing just 'OK' if the
solution is a coloring with D+1 colors, or a line
containing

    FAILED node n1 and neighbor n2 both colored c

if neighboring nodes n1 and n2 were both colored c,
or a line containing

    FAILED node n colored c

if node n was colored with c < 0 or c > D.  If there was
some error in message format, the output will be

    FAILED ...

where ... describes the error.
```

```
Debugging
---------

You can add debugging arguments to the program command:

    localcolor_network localcolor debug_argument ...

If you do this your program will be invoked by the
monitor using the command

    localcolor debug_argument ...

and the monitor will output the input and output for
each node and cycle in addition to the usual output
of the monitor.  The input/output for a node and cycle
is surrounded by lines containing just '*****' and the
input is separated from the output by a line containing
just '-----'.  You can put debugging information in your
messages, and used the debug arguments to tune this
information.


Sample Input
------ -----

-- SAMPLE 1 --
10 3 20 783927645
-- SAMPLE 2 --
20 10 50 259140687


Sample Output
------ ------

-- SAMPLE 1 --
OK
-- SAMPLE 2 --
OK
```

Reference

For a discussion of local distributed algorithms, more examples, and impossibility results see

A Survey of Local Algorithms, Jukka Suomela,
ACM Computing Surveys, Vol. 45, No. 2, 2013.

File: localcolor.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Mon Oct 14 02:53:43 EDT 2013

RCS Info (may not be true date or author):

$Author: walton $
$Date: 2013/10/14 07:10:46 $
$RCSfile: localcolor.txt,v $
$Revision: 1.9 $

Eulerian Cycles

A Eulerian cycle is a closed path in an undirected graph that includes every edge exactly once. The path may be non-simple: it may visit a particular vertex many times. It is easy to show that a graph has an Eulerian cycle if and only if the graph is connected and every vertex of the graph has even degree. Here the degree of a vertex is the number of edges of which the vertex is an end-point.

You have been asked to find an Eulerian cycle in each of several connected graphs. Some of the graphs are very large.

Input

For each of several test cases, first a line containing just the test case name, then M lines each containing a representation of an edge of the form

<vertex> - <vertex>

and lastly a line containing just '*'. Here <vertex> is just an integer in the range 1, ..., N where N is the total number of vertices in the graph.

No edge will be represented more than once.

3 <= N <= 100,000.
N - 1 <= M <= 1,000,000

All input graphs are guaranteed to be connected and all vertices in these graphs are guaranteed to have even degree.

Input ends with an end of file.

Output

For each test case, just two lines, the first containing an exact copy of the test case name input line, and the second a possibly very long line containing the list of vertices visited by an Eulerian cycle, in the order of visitation. There should be exactly M + 1 vertices in this list with the first vertex being repeated at the end of the list. Any one of the many possible Eulerian cycles may be output.

```
Sample Input
------ -----

-- 5 VERTEX COMPLETE GRAPH --
1 - 2
1 - 3
1 - 4
1 - 5
2 - 3
2 - 4
2 - 5
3 - 4
3 - 5
4 - 5
*
-- 6 VERTEX REGULAR DEGREE 4 GRAPH --
1 - 2
1 - 3
2 - 3
2 - 4
3 - 4
3 - 5
4 - 5
4 - 6
5 - 6
5 - 1
6 - 1
6 - 2
*


Sample Output
------ ------

-- 5 VERTEX COMPLETE GRAPH --
2 4 3 2 5 1 4 5 3 1 2
-- 6 VERTEX REGULAR DEGREE 4 GRAPH --
1 2 6 4 2 3 1 6 5 3 4 5 1
```

```
File:      eulerian.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sat Oct 12 00:23:41 EDT 2013

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.

RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2013/10/12 05:04:54 $
    $RCSfile: eulerian.txt,v $
    $Revision: 1.5 $
```

Polygon/Ellipse Intersection

You have been asked to compute the area of the intersection of a convex polygon and an ellipse.

For simplicity, the ellipse is positioned so its major semi-axis, which is of length A, is on the x-axis, and its minor semi-axis, which is of length B, is on the y-axis.

Input

For each of several test cases, first a line containing just the test case name, then a line containing

```
A B N
```

where N is the number of vertices in the polygon, and then N lines each containing

```
X Y
```

specifying that (X,Y) is a vertex. The vertices are listed in clockwise order. N is an integer, all other numbers are floating point.

```
3 <= N <= 100
0 < B <= A <= 100
-100 <= X,Y <= 100
```

Input ends with an end of file.

Output

For each test case, first a line containing the test case name, and then a line containing just the required area, printed with exactly 3 decimal places.

Sample Input

```
-- SAMPLE 1 --
4 1 3
0 0
0 2
8 0
-- SAMPLE 2 --
50 8 3
0 0
100 16
100 -16
-- SAMPLE 3 --
1 1 3
1 1
1 2
2 1
-- SAMPLE 4 --
100 100 4
-100 -100
-100 +100
+100 +100
+100 -100
-- SAMPLE 5 --
1 1 3
0 0
-0.7071067700 +0.7071067700
+0.7071067700 +0.7071067700
```

```
Sample Output
------ ------

-- SAMPLE 1 --
3.142
-- SAMPLE 2 --
314.159
-- SAMPLE 3 --
0.000
-- SAMPLE 4 --
31415.927
-- SAMPLE 5 --
0.500


File:      polyellipse.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Mon Oct 14 15:26:41 EDT 2013

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.

RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2013/10/14 19:27:17 $
    $RCSfile: polyellipse.txt,v $
    $Revision: 1.8 $
```