UIL Computer Science Competition

Invitational B 2024

JUDGES PACKET - CONFIDENTIAL

I. Instructions

1. The attached printouts of the judge test data are provided for the reference of the contest director and programming judges. Additional copies may be made if needed for this purpose.

2. This packet must remain CONFIDENTIAL. Additional copies may be made and returned to schools when other confidential contest material is returned.

II. Table of Contents

Problem #1
60 Points

1. Ajeet

Program Name: Ajeet.java **Input File: None**

Test Output to Screen:

```
                          MAY 2024
01234567890123456789012345678901234567890123456789012345678901234567890123456789
    SUN      MON      TUE      WED      THR      FRI      SAT
                               1        2        3        4
      5        6        7        8        9       10       11
     12       13       14       15       16       17       18
     19       20       21       22       23       24       25
     26       27       28       29       30       31
```

Problem #2
60 Points

2. Billy

Program Name: Billy.java **Input File: billy.dat**

Test Input File:

```
MISSOURI
15 10
PENN STATE
1 1
OLE MISS
2 1
OKLAHOMA
0 10
LSU
7 6
ARIZONA
20 18
NOTRE DAME
3 2
LOUISVILLE
5 5
```

Test Output To Screen:

```
OLE MISS
```

Problem #3
60 Points

3. Daniella

Program Name: Daniella.java **Input File: daniella.dat**

Test Input File:

```
12
RIGATONI 2
PENNE 5
FETTUCCINE 4
LASAGNA 1
MACARONI 5
SPAGHETTI 3
FARFALLE 1
BUCATINI 2
DITALINI 3
ORZO 4
GNOCCHI 5
FUSILLI 2
```

Test Output To Screen:

```
RI-GA-TO-NI
PENNE
FETT-UCCI-NE
L-A-S-A-G-N-A
MACAR-ONI
SPA-GHE-TTI
F-A-R-F-A-L-L-E
BU-CA-TI-NI
DIT-ALI-NI
ORZO
GNOCC-HI
FU-SI-LL-I
```

Problem #4
60 Points

4. Donghai

Program Name: Donghai.java **Input File: donghai.dat**

Test Input File: (*indented lines are continuation of previous line*)

```
10
Let's start with a short 1-line paragraph just to see a result!
Now, this ^second paragraph^ will be much longer which results in the sample
   input displaying with all continuation lines indented.  However, it will be a
   single unbroken line in the data file that is provided for the contest...
This is the third "paragraph" of sample data for this ~programming problem~...
Joan really enjoys her `programming` cla$$ but her @true passion is the English
   language.  However, programming has caused her to view English writings
   somewhat differently and she realizes that words are simply pieces of data
   put together carefully to create a desired meaning.  For now, she has decided
   to practice her programming skills by creating a program to perform a very
   simple analysis of relatively small pieces of prose.
Joan has big plans for analyzing {written pieces} but decided to start with a
   program that simply counts the number of words in a sample of prose and
   calculate the average length of all words.
Input:  A sample of English prose with an (unknown number of lines) containing
   an unknown number of whitespace-separated words on each line.  The number of
   lines will be in the range [2,25].  Words may consist of both uppercase and
   lowercase letters and there are no non-letter symbols or numbers.
Output:  A single line containing the number of words and the calculated average
   length of all words, formatted exactly as shown below in the sample output.
   The average length is rounded to the nearest whole number.
Donghai helped Joan solve the I>n>v>i>t>a>t>i>o>n>a>l A problem counting words
   and c+a-l*c/u%l!a&t|i#n@g\ the average size of words.  Now, he wants to work
   with letters instead of words.  He wants to count the number of times each
   letter occurs in written prose.  He will ignore punctuation marks and any
   other non-alphabetic characters.  Uppercase and lowercase letters are
   considered the same letter.  Once they have been counted, he will display
   them with the letter of the highest count at the top and the letter of the
   lowest count at the bottom.  Letters that share identical counts will be
   listed alphabetically on the same line.__Can you help Donghai solve this
   problem?
Input:  First line will contain an integer N with 0 < N <= 10 which is the
   number of paragraphs that follow.  Each paragraph will occur on a single line
   of text and will not exceed 2000 characters in total length.  Any ASCII
   character can occur in a paragraph but the only ones of interest are the
   standard alphabetic letters, both uppercase and lowercase.
Output:  Each test case must display list of letters in u-p-p-e-r-c-a-s-e that
   have non-zero counts [like 1,2,3,4,5,6,7,8,9] formatted as the count;
   followed by a colon ":" followed by the letter with no spaces.  Following
   each test case display a line containing 10 equal signs "==========".
```

~ Output starts on next page… ~

~ Donghai continued… ~

Test Output To Screen: (***read down columns*** **)**

```
A:6          A:7          A:17         A:15         A:32
E:5          B:1          B:3          B:3          B:4
G:1          D:2          C:5          C:4          C:17
H:3          E:3          D:6          D:7          D:5
I:2          F:2          E:14         E:24         E:28
J:1          G:3          F:4          F:3          F:5
L:3          H:5          G:5          G:6          G:6
N:1          I:5          H:6          H:9          H:16
O:2          L:2          I:8          I:6          I:19
P:2          M:4          J:1          L:12         L:17
R:5          N:1          L:9          M:4          M:1
S:6          O:4          M:4          N:14         N:24
T:8          P:5          N:10         O:13         O:15
U:2          R:7          O:11         P:3          P:10
W:1          S:4          P:6          R:9          R:21
==========   T:5          R:10         S:8          S:12
A:11         ==========   S:10         T:18         T:26
B:3          A:32         T:14         U:8          U:6
C:5          B:2          U:4          V:2          W:7
D:7          C:10         V:1          W:5          X:2
E:20         D:10         W:4          X:1          Y:2
F:2          E:45         Y:2          Y:1          ==========
G:4          F:8          Z:1          ==========   A:16
H:11         G:15         ==========   A:41         B:2
I:20         H:15         A:16         B:8          C:10
K:1          I:24         B:7          C:22         D:5
L:14         J:2          C:6          D:17         E:23
M:2          K:1          D:5          E:73         F:5
N:18         L:19         E:30         F:5          G:3
O:10         M:13         F:5          G:7          H:7
P:6          N:17         G:3          H:34         I:11
R:8          O:22         H:8          I:29         K:1
S:10         P:13         I:12         J:1          L:15
T:16         R:32         K:2          K:2          M:2
U:5          S:26         L:11         L:32         N:13
V:2          T:20         M:7          M:9          O:17
W:6          U:6          N:29         N:37         P:6
Y:1          V:4          O:19         O:41         Q:1
==========   W:6          P:7          P:12         R:5
             Y:9          R:16         R:28         S:16
             Z:1          S:17         S:29         T:22
             ==========   T:14         T:64         U:7
                          U:8          U:13         V:1
                          W:8          V:5          W:4
                          Y:2          W:15         Y:4
                          ==========   Y:5          Z:1
                                       Z:1          ==========
                                       ==========
```

Problem #5
60 Points

5. Hiromi

Program Name: Hiromi.java **Input File: hiromi.dat**

Test Input File: (***displayed down then across columns)***

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

Test Output to Screen:

```
Puzzle #1: false
Puzzle #2: true
Puzzle #3: false
Puzzle #4: false
Puzzle #5: true
Puzzle #6: false
Puzzle #7: false
```

Problem #6
60 Points

6. Jana

Program Name: Jana.java **Input File: jana.dat**

Test Input File:

```
20
6
9
7
15
8
20
1
2
3
4
5
50
25
34
32
31
33
76
123
27
```

Test Output To Screen:

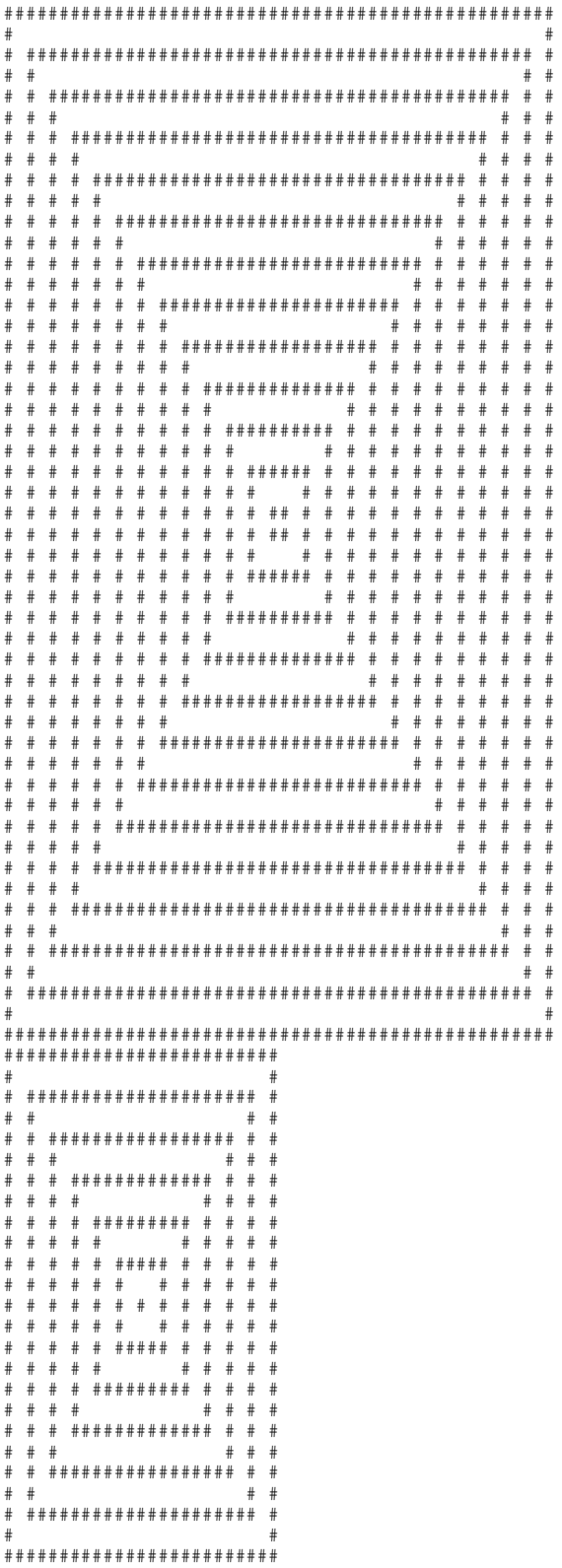
```
######
#    #
# ## #
# ## #
#    #
######
#########
#       #
# ##### #
# #   # #
# # # # #
# #   # #
# ##### #
#       #
#########
#######
#     #
# ### #
# # # #
# ### #
#     #
#######
```

~ output continues next column ~

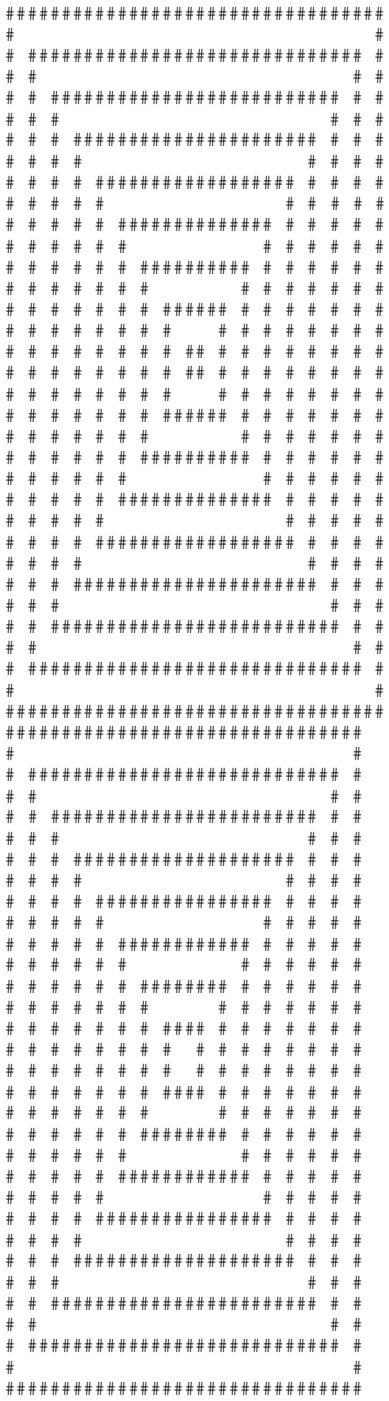
```
###############
#             #
# ########### #
# #         # #
# # ####### # #
# # #     # # #
# # # ### # # #
# # # # # # # #
# # # ### # # #
# # #     # # #
# # ####### # #
# #         # #
# ########### #
#             #
###############
########
#      #
# #### #
# #  # #
# #  # #
# #### #
#      #
########
####################
#                  #
# ################ #
# #                # #
# # ########### # #
# # #         # # #
# # # ######## # # #
# # # #       # # # #
# # # # #### # # # #
# # # # #  # # # # #
# # # # #  # # # # #
# # # # #### # # # #
# # # #      # # # #
# # # ######## # # #
# # #          # # #
# # ########### # #
# #              # #
# ################ #
#                  #
####################
#
##
##
###
# #
###
####
#  #
#  #
####
#####
#   #
# # #
#   #
#####
```

~ output continues next page ~

~ Jana output continued… ~

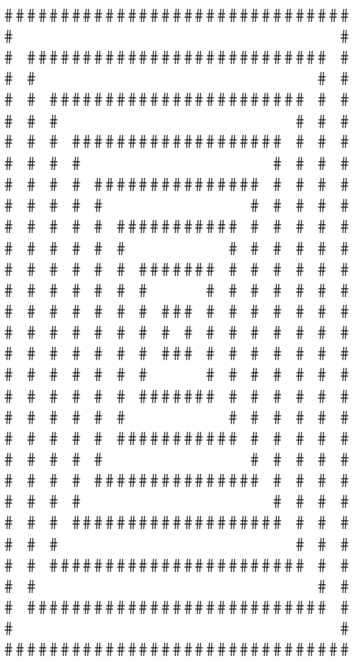


~ output continues next column ~

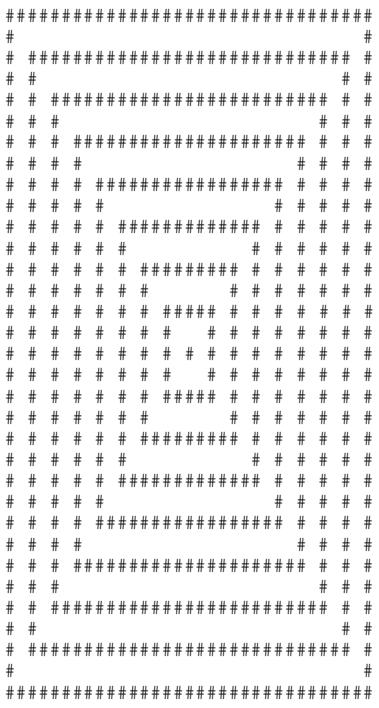


~ output continues next page ~

~ Jana output continues… ~

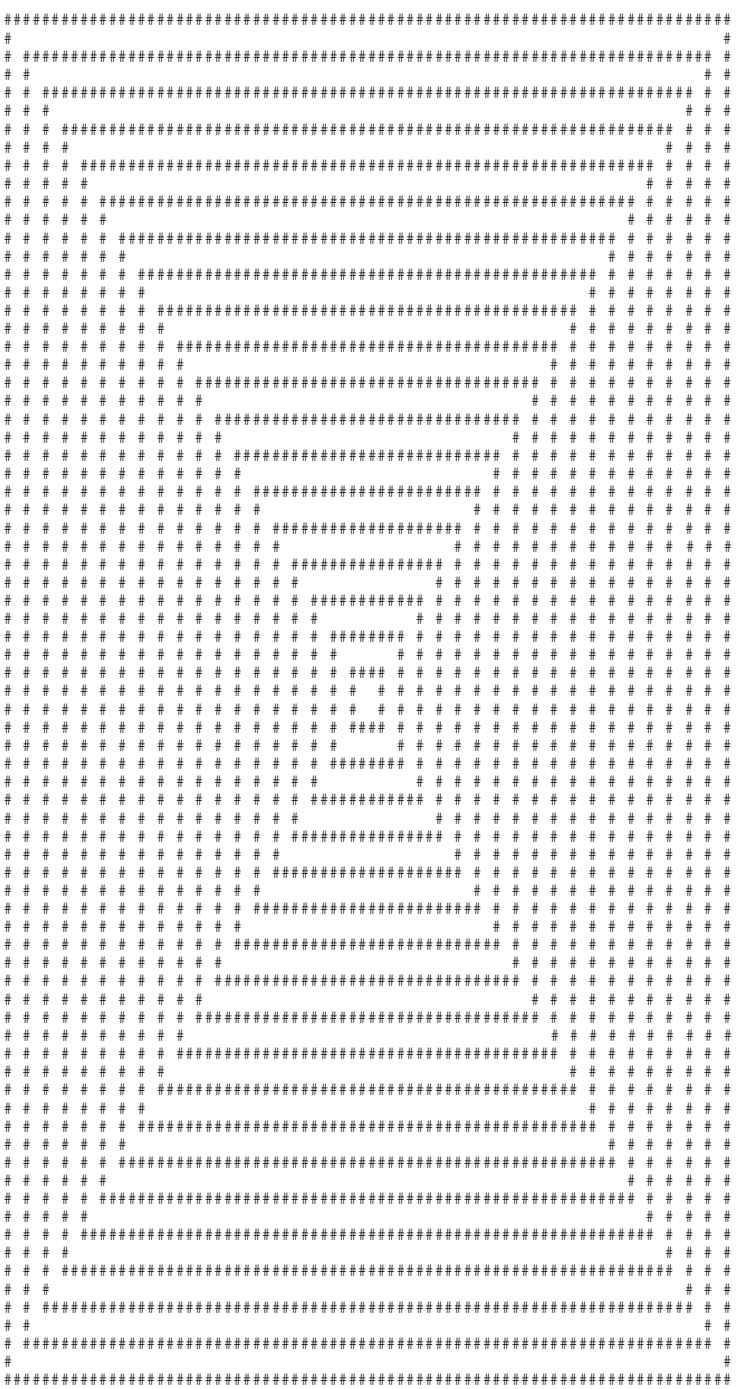


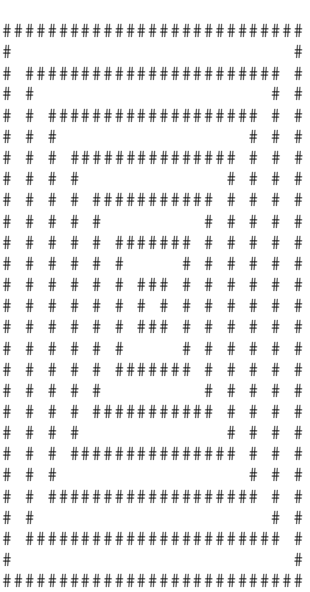
~ output continues next column ~



~ output continues next page ~

~ Jana output continues… ~





~ sign of size 123 too large to fit page width – see judge's out file ~

~ output continues next column ~

Problem #7
60 Points

7. Ksenia

Program Name: Ksenia.java **Input File: ksenia.dat**

Test Input File: (*indented lines are continuation of previous line*)

```
10
4 6
0 3,1 0,1 3,2 3,3 0,3 1
11
6 15
0 2,0 3,1 2,1 4,2 1,2 3,2 4,3 2,3 5,4 2,4 3,4 5,5 1,5 2,5 4
15
25 2
2 7,10 20
1
2 1
0 1
9
18 23
0 1,1 4,1 10,3 1,3 6,3 16,4 6,4 13,5 13,7 8,8 17,9 13,10 1,10 2,11 14,12 2,
      13 7,13 10,15 2,16 10,17 1,17 6,17 15
2
11 18
1 3,1 4,1 6,1 10,2 6,2 8,4 1,4 8,5 4,5 7,5 10,6 2,6 3,8 2,8 4,10 2,10 5,10 7
9
8 0
3
9 23
0 1,0 8,1 3,1 4,1 8,2 3,3 4,3 5,3 6,3 7,4 6,5 0,5 4,6 1,6 2,6 5,7 0,7 2,7 4,
      7 5,7 6,7 8,8 5
10
12 8
2 8,3 4,4 7,5 0,5 2,5 3,6 8,7 10
14
6 10
1 4,2 1,2 3,2 4,3 1,3 5,4 2,4 3,4 5,5 4
8
```

Test Output To Screen:

```
8192 57344
196608 1966080
50 29
1024 5120
72 164
5632 34560
64 96
9216 69632
196608 1507328
1536 8704
```

Problem #8
60 Points

8. Marta

Program Name: Marta.java **Input File: marta.dat**

Test Input File:

```
10
12 1
95 11 68 13 33 26 24 60 56 47 79 21
12 2
95 11 68 13 33 26 24 60 56 47 79 21
25 23
71 40 52 66 95 80 61 56 26 64 34 47 99 51 96 86 63 55 49 72 82 43 93 42 76
25 1
71 40 52 66 95 80 61 56 64 34 47 99 51 96 86 63 55 49 72 82 43 93 42 76 26
25 5
71 40 52 66 95 80 61 56 64 34 47 99 51 96 86 63 55 49 72 82 43 93 42 76 26
25 24
71 40 52 66 95 80 61 56 64 34 47 99 51 96 86 63 55 49 72 82 43 93 42 76 26
25 25
71 40 52 66 95 80 61 56 64 34 47 99 51 96 86 63 55 49 72 82 43 93 42 76 26
25 3
26 34 40 42 43 47 49 51 52 55 56 61 63 64 66 71 72 76 80 82 86 93 95 96 99
25 5
99 96 95 93 86 82 80 76 72 71 66 64 63 61 56 55 52 51 49 47 43 42 40 34 26
25 5
34 26 42 40 47 43 51 49 55 52 61 56 64 63 71 66 76 72 82 80 93 86 96 95 99
```

Test Output To Screen: (*lines that are indented are continuation of previous line*)

```
Test case 1 pass 1: 11 68 13 33 26 24 60 56 47 79 21 95
Test case 2 pass 2: 11 13 33 26 24 60 56 47 68 21 79 95
Test case 3 pass 21: 26 34 40 42 43 47 49 51 52 55 56 61 63 64 66 71 72 76 80 82
   86 93 95 96 99
Test case 4 pass 1: 40 52 66 71 80 61 56 64 34 47 95 51 96 86 63 55 49 72 82 43
   93 42 76 26 99
Test case 5 pass 5: 40 52 56 61 34 47 64 51 66 71 63 55 49 72 80 43 82 42 76 26
   86 93 95 96 99
Test case 6 pass 24: 26 34 40 42 43 47 49 51 52 55 56 61 63 64 66 71 72 76 80 82
   86 93 95 96 99
Test case 7 pass 25: 26 34 40 42 43 47 49 51 52 55 56 61 63 64 66 71 72 76 80 82
   86 93 95 96 99
Test case 8 pass 1: 26 34 40 42 43 47 49 51 52 55 56 61 63 64 66 71 72 76 80 82
   86 93 95 96 99
Test case 9 pass 5: 82 80 76 72 71 66 64 63 61 56 55 52 51 49 47 43 42 40 34 26
   86 93 95 96 99
Test case 10 pass 2: 26 34 40 42 43 47 49 51 52 55 56 61 63 64 66 71 72 76 80 82
   86 93 95 96 99
```

Problem #9
60 Points

9. Prateek

Program Name: Prateek.java **Input File: prateek.dat**

Test Input File: (*indented lines are continuation of previous line*)

```
25
0 9 18 19 43 56 68 73 74 86 93 99 132 542 598 607 777 949 2223 4007 4610 6032
   7529 7550 9794 9857 11435 17954 88352
10 43 593 15086 45125 86806
21 45 54 65 78 88 89 99 133 157 163 257 455 634 972 2000 5077 5307 5457 5504
   5743 6402 9321 9838 10978 12595 26462 68662 82132
3 13 18 47 50 51 63 73 93 532 797 857 1469 2628 3057 5146 7160 64135 64402
244 2891 5067 5191 7102
98 148 619
10 20 45 46 65 78 153 192 430 470 660 799 2767 5923 6409 6475 9331 10396 15662
   23701 24694 38225 40842 58629 68708 80196
31 52 59 83 530 545 727 846 980
13 75 318 689 1716 4919 83846 91863 97735
21 119 172 220 382 467 488 649 798 799 2155 4749 9140 9181 10133 64114 80226
   93403 98303
6 14 29 35 38 44 61 65 77 87 88 113 154 318 543 792 824 995 1616 2182 5328 6396
   6895 8781 16382 28073 30472 50367 69018
90 91 97 140 159 221 268 483 848 918 5001 5073 5107 6003 7723 9310 9533 69928
   76196 76219 84881 92412
0 30 39 59 68 83 141 529 609 646 700 799 860 1087 4454 4692 4803 6658 54956
   57655 81959 82759 85364
25 52 96 381 532 3865 75357 78097 80919
19 94 255 429 612 633 842 912 2757 3227 35419 79206 83108 90360 93112
37 68 74 93 652 895 1090 2503 3196 4547 4844 5170 5392 7552 7663 7917 23804
   50493 59788
0 16 55 136 416 463 973 6849 7337 10523 30764 41969 75924
47 72 398 859 5106 6108 52489 54449 60722
24 36 72 75 83 84 590 876 2699 5456 5680 7222 42988 43899 49582 59181 72629
   75895
4 22 42 51 58 59 65 71 77 167 383 455 535 847 1303 5629 33907 47253 89133 98052
   98356
3 9 22 26 67 264 509 538 687 885 5254 5640 6039 7451 16712 59269 70851 98852
131 743 877 934 1243 2440 9903 16856 95252
18 77 251 429 623 668 879 1681 2036 2436 2803 4057 5670 6314 7511 8606 8945 9418
   21288 23828 29112 30984 52011 96048
12 75 856 8019 9992 70765
25 34 46 61 63 85 90 231 303 687 3190 6446 6763 9112 9400 32709 85349
```

~ output starts on next page ~

~ Prateek continued… ~

Test Output To Screen: (*indented lines are continuation of previous line*)

```
List: 0 9 18 19 43 56 68 73 74 86 93 99 132 542 598 607 777 949 2223 4007 4610 6032 7529 7550 9794
    9857 11435 17954 88352
1-iterations: 598
2-iterations: 68 6032
3-iterations: 18 93 949 9857
4-iterations: 0 43 74 132 607 4007 7550 17954
5-iterations: 9 19 56 73 86 99 542 777 2223 4610 7529 9794 11435 88352

List: 10 43 593 15086 45125 86806
1-iterations: 593
2-iterations: 10 45125
3-iterations: 43 15086 86806

List: 21 45 54 65 78 88 89 99 133 157 163 257 455 634 972 2000 5077 5307 5457 5504 5743 6402 9321
    9838 10978 12595 26462 68662 82132
1-iterations: 972
2-iterations: 89 6402
3-iterations: 54 163 5307 12595
4-iterations: 21 78 133 455 2000 5504 9838 68662
5-iterations: 45 65 88 99 157 257 634 5077 5457 5743 9321 10978 26462 82132

List: 3 13 18 47 50 51 63 73 93 532 797 857 1469 2628 3057 5146 7160 64135 64402
1-iterations: 532
2-iterations: 50 3057
3-iterations: 13 63 857 7160
4-iterations: 3 18 51 73 797 1469 5146 64135
5-iterations: 47 93 2628 64402

List: 244 2891 5067 5191 7102
1-iterations: 5067
2-iterations: 244 5191
3-iterations: 2891 7102

List: 98 148 619
1-iterations: 148
2-iterations: 98 619

List: 10 20 45 46 65 78 153 192 430 470 660 799 2767 5923 6409 6475 9331 10396 15662 23701 24694
    38225 40842 58629 68708 80196
1-iterations: 2767
2-iterations: 78 23701
3-iterations: 45 430 6475 40842
4-iterations: 10 46 153 660 5923 10396 24694 68708
5-iterations: 20 65 192 470 799 6409 9331 15662 38225 58629 80196

List: 31 52 59 83 530 545 727 846 980
1-iterations: 530
2-iterations: 52 727
3-iterations: 31 59 545 846
4-iterations: 83 980

List: 13 75 318 689 1716 4919 83846 91863 97735
1-iterations: 1716
2-iterations: 75 83846
3-iterations: 13 318 4919 91863
4-iterations: 689 97735

List: 21 119 172 220 382 467 488 649 798 799 2155 4749 9140 9181 10133 64114 80226 93403 98303
1-iterations: 799
2-iterations: 382 10133
3-iterations: 119 488 4749 80226
4-iterations: 21 172 467 649 2155 9140 64114 93403
5-iterations: 220 798 9181 98303
```

~ Output continues on next page… ~

~ Prateek continued… ~

```
List: 6 14 29 35 38 44 61 65 77 87 88 113 154 318 543 792 824 995 1616 2182 5328 6396 6895 8781
    16382 28073 30472 50367 69018
1-iterations: 543
2-iterations: 61 6396
3-iterations: 29 88 995 28073
4-iterations: 6 38 77 154 792 2182 8781 50367
5-iterations: 14 35 44 65 87 113 318 824 1616 5328 6895 16382 30472 69018

List: 90 91 97 140 159 221 268 483 848 918 5001 5073 5107 6003 7723 9310 9533 69928 76196 76219
    84881 92412
1-iterations: 5001
2-iterations: 159 9533
3-iterations: 91 483 6003 76219
4-iterations: 90 97 221 848 5073 7723 69928 84881
5-iterations: 140 268 918 5107 9310 76196 92412

List: 0 30 39 59 68 83 141 529 609 646 700 799 860 1087 4454 4692 4803 6658 54956 57655 81959 82759
    85364
1-iterations: 799
2-iterations: 83 6658
3-iterations: 39 609 4454 81959
4-iterations: 0 59 141 646 860 4692 54956 82759
5-iterations: 30 68 529 700 1087 4803 57655 85364

List: 25 52 96 381 532 3865 75357 78097 80919
1-iterations: 532
2-iterations: 52 75357
3-iterations: 25 96 3865 78097
4-iterations: 381 80919

List: 19 94 255 429 612 633 842 912 2757 3227 35419 79206 83108 90360 93112
1-iterations: 912
2-iterations: 429 79206
3-iterations: 94 633 3227 90360
4-iterations: 19 255 612 842 2757 35419 83108 93112

List: 37 68 74 93 652 895 1090 2503 3196 4547 4844 5170 5392 7552 7663 7917 23804 50493 59788
1-iterations: 4547
2-iterations: 652 7663
3-iterations: 68 1090 5170 23804
4-iterations: 37 74 895 2503 4844 5392 7917 50493
5-iterations: 93 3196 7552 59788

List: 0 16 55 136 416 463 973 6849 7337 10523 30764 41969 75924
1-iterations: 973
2-iterations: 55 10523
3-iterations: 0 416 6849 41969
4-iterations: 16 136 463 7337 30764 75924

List: 47 72 398 859 5106 6108 52489 54449 60722
1-iterations: 5106
2-iterations: 72 52489
3-iterations: 47 398 6108 54449
4-iterations: 859 60722

List: 24 36 72 75 83 84 590 876 2699 5456 5680 7222 42988 43899 49582 59181 72629 75895
1-iterations: 2699
2-iterations: 75 43899
3-iterations: 36 84 5680 59181
4-iterations: 24 72 83 590 5456 7222 49582 72629
5-iterations: 876 42988 75895
```

~ Output continues on next page… ~

~ Prateek continued… ~

```
List: 4 22 42 51 58 59 65 71 77 167 383 455 535 847 1303 5629 33907 47253 89133 98052 98356
1-iterations: 383
2-iterations: 58 5629
3-iterations: 22 71 535 89133
4-iterations: 4 42 59 77 455 847 33907 98052
5-iterations: 51 65 167 1303 47253 98356

List: 3 9 22 26 67 264 509 538 687 885 5254 5640 6039 7451 16712 59269 70851 98852
1-iterations: 687
2-iterations: 26 7451
3-iterations: 9 264 5254 59269
4-iterations: 3 22 67 509 885 5640 16712 70851
5-iterations: 538 6039 98852

List: 131 743 877 934 1243 2440 9903 16856 95252
1-iterations: 1243
2-iterations: 743 9903
3-iterations: 131 877 2440 16856
4-iterations: 934 95252

List: 18 77 251 429 623 668 879 1681 2036 2436 2803 4057 5670 6314 7511 8606 8945 9418 21288 23828
    29112 30984 52011 96048
1-iterations: 4057
2-iterations: 668 9418
3-iterations: 251 2036 7511 29112
4-iterations: 18 429 879 2436 5670 8606 21288 52011
5-iterations: 77 623 1681 2803 6314 8945 23828 30984 96048

List: 12 75 856 8019 9992 70765
1-iterations: 856
2-iterations: 12 9992
3-iterations: 75 8019 70765

List: 25 34 46 61 63 85 90 231 303 687 3190 6446 6763 9112 9400 32709 85349
1-iterations: 303
2-iterations: 61 6763
3-iterations: 34 85 3190 9400
4-iterations: 25 46 63 90 687 6446 9112 32709
5-iterations: 231 85349
```

Problem #10
60 Points

10. Sofia

Program Name: Sofia.java **Input File: sofia.dat**

Test Input File:

```
13 10
Paris Dallas 1200
Paris Prague 650
Prague Munich 300
Paris Munich 300
Berlin Prague 400
Berlin Dallas 1400
Berlin Moscow 800
Moscow Prague 900
Casablanca Munich 800
Prague Athens 600
Athens Dubai 500
Athens Delhi 800
Dubai Delhi 400
Dallas Prague 1550
Dallas Berlin 2200
Paris Berlin 1000
Dallas Moscow 2200
Prague Moscow 1000
Prague Moscow 800
Dallas Casablanca 2300
Casablanca Dallas 2300
Dallas Delhi 3200
Paris Athens 1200
```

Test Output To Screen:

```
There's no place like home.
Berlin is always a good idea.
Berlin is always a good idea.
Moscow is always a good idea.
Moscow is always a good idea.
There's no place like home.
Casablanca is always a good idea.
Dallas is always a good idea.
Delhi is always a good idea.
Athens is always a good idea.
```

Problem #11
60 Points

11. Tyler

Program Name: Tyler.java **Input File: tyler.dat**

Test Input File: (*indented lines are continuation of previous line*)

```
44 78
1 705 385 257 2 66 3 4 708 5 6 7 72 8 456 9 906 12 13 206 16 80 403 21 88 729
   347 32 96 34 98 37 38 872 553 555 876 685 46 51 694 569 894 511
1 2 3 5 481666 6 7 3712447 8 9 32956287 589699 675204 784 405 24465262 57118
   1567753 256284 9376 75556 8888750 52394 5689319 4529 683832 46976638 59571
   5199740 90034 94901 5183864 6202 51771 77247 66240 833 194 3629428 70 583
   844869 26733532 57805 249421 77903 591 4817 86352 38354 2514 85 16599 3672 89
   24393005 3025138 92 1966146 8885978 2263167 605 1563255 4834 63074 355
   1824254 537199 873 858598 440052 7667 3625795 39415 16260225 8980979 11039700
   215039
```

Test Output To Screen:

```
Target 1: YES
Target 2: YES
Target 3: YES
Target 5: YES
Target 481666: NO
Target 6: YES
Target 7: YES
Target 3712447: NO
Target 8: YES
Target 9: YES
Target 32956287: NO
Target 589699: NO
Target 675204: NO
Target 784: YES
Target 405: YES
Target 24465262: NO
Target 57118: NO
Target 1567753: NO
Target 256284: NO
Target 9376: NO
Target 75556: NO
Target 8888750: NO
Target 52394: NO
Target 5689319: NO
Target 4529: NO
Target 683832: NO
Target 46976638: NO
Target 59571: NO
Target 5199740: NO
Target 90034: NO
Target 94901: NO
Target 5183864: NO
Target 6202: NO
```

Continues next column…

Continued from previous column…

```
Target 51771: NO
Target 77247: NO
Target 66240: YES
Target 833: NO
Target 194: NO
Target 3629428: NO
Target 70: YES
Target 583: NO
Target 844869: NO
Target 26733532: NO
Target 57805: NO
Target 249421: NO
Target 77903: NO
Target 591: NO
Target 4817: NO
Target 86352: YES
Target 38354: NO
Target 2514: NO
Target 85: NO
Target 16599: NO
Target 3672: YES
Target 89: NO
Target 24393005: NO
Target 3025138: NO
Target 92: YES
Target 1966146: NO
Target 8885978: NO
Target 2263167: NO
Target 605: NO
Target 1563255: NO
Target 4834: NO
Target 63074: NO
Target 355: NO
```

Continues next column…

Continued from previous column…

```
Target 1824254: NO
Target 537199: NO
Target 873: NO
Target 858598: NO
Target 440052: NO
Target 7667: NO
Target 3625795: NO
Target 39415: NO
Target 16260225: NO
Target 8980979: NO
Target 11039700: NO
Target 215039: NO
```

Problem #12
60 Points

12. Vika

Program Name: Vika.java **Input File: vika.dat**

Test Input File: (*indented lines are continuation of previous line*)

```
15
ABC BAC
Someone Stonecold
Applebees Geronimo
somebodyoncetoldmetheworldwasgonnarollme
   iaintthesharpesttoolintheshediwaslookingkindadumb
withafingerandathumbintheshapeofanLonherforehead
   haveyoueverheardthetragedyofdarthplagueisthewise
ithoughtnotitsnotastorythejediwouldtellyouitsasithlegend
   darthplagueswasadarklordofthesithsopowerfulandsowise
hecouldusetheforcetoinfluencethemidichlorianstocreatelifehehadsuchaknowledgeofth
   edarksidethathecouldevenkeeptheoneshecaredaboutfromdying
   thedarksideoftheforceisapathwaytomanyabilitiessomeconsidertobeunnaturalhebeca
   mesopowerfultheonlythinghewasafraidofwaslosinghispower
whicheventuallyofcoursehedidunfortunatelyhetaughthisapprentice
   everythingheknewthenhisapprenticekilledhiminhissleep
ironichecouldsaveothersfromdeathbutnothimself
   isitpossibletolearnthispowernotfromajedi
APPLESANDBANANA AEGEANSEA
albanian serbian
abcdefg abcdefg
AbCdEfG abcdefg
ABCDEFG aBcDeFg
ABCDEFG abcdefg
```

Test Output To Screen:

```
2
4
1
18
18
20
57
24
17
5
4
7
3
3
0
```

Computer Science Competition
Invitational B 2024
Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Ajeet
Problem 2	Billy
Problem 3	Daniella
Problem 4	Donghai
Problem 5	Hiromi
Problem 6	Jana
Problem 7	Ksenia
Problem 8	Marta
Problem 9	Prateek
Problem 10	Sofia
Problem 11	Tyler
Problem 12	Vika

1. Ajeet

Program Name: Ajeet.java **Input File: None**

Ajeet is making travel arrangements for his Leaguetown HS CompSci Team to go to Austin for the State UIL Meet in May 2024. To do this, he needs a calendar of May 2024.

Your job is to display the calendar for him. It should look exactly as shown below.

Input: None

Output: Duplicate the calendar exactly as shown below. Make sure to include the line below the month consisting of the six sets of digits 0-9.

Sample input: None

Sample output:

```
                          MAY 2024
01234567890123456789012345678901234567890123456789012345678
    SUN      MON      TUE      WED      THR      FRI      SAT
                                1        2        3        4
     5        6        7        8        9       10       11
    12       13       14       15       16       17       18
    19       20       21       22       23       24       25
    26       27       28       29       30       31
```

2. Billy

Program Name: Billy.java **Input File: billy.dat**

There is a lot of controversy this year because of the selections made to the Pickle Ball play-offs. There was so much discussion as to which teams made the Top Four. We will be talking about this for a long time.

Billy has been charged with finding a program that will find the winner among the eight schools in the league. He is given their names and their win-loss records. He must find the school with the highest winning percentage.

Now of course, due to a recent court decision, there will always be a clear winner after all the games are played. There is a guarantee not to have a tie for first place. So, it is all based on which team has the best winning percentage, even though teams possibly played a different number of games.

In the sample, Texas has the highest winning percentage at 90.9% with a record of 10 wins and 1 loss.

Input: There will be 16 lines of data. The odd numbered lines will be the team names. Each team name will be of length in the range [1,20]. Some teams may have spaces in their names. On the even numbered lines, there will be two integers, both in the range [0,24]. There will be one space between separating the numbers that are on each line. The first number is the number of wins. The second number is the number of losses. The win-loss record will always be on the line below the team name.

Output: Output the name of the team with the highest winning percentage.

Sample input:

```
MICHIGAN
12 2
WASHINGTON
7 14
TEXAS
10 1
ALABAMA
4 9
FLORIDA STATE
10 2
GEORGIA
1 12
OHIO STATE
4 5
OREGON
7 10
```

Sample output:

```
TEXAS
```

3. Daniella

Program Name: Daniella.java **Input File: daniella.dat**

Daniella is concerned with words. She is always thinking about words. She has always been that way. She understands language and loves to think about words.

In her job as a chef, she is in charge of preparing meals for the customers in a fancy restaurant. In that job, she cooks great meals and is in charge of every step in the process. One of those steps is chopping up the vegetables. She is known world-wide for her chopping skills.

Every once in a while, her mind wanders, and in her daydreams, she combines her two interests. Sometimes even, she imagines chopping words. Your job is to write a program to "chop" words into pieces.

You will be given a String S and an integer N. Your job is to chop S into groups containing N letters each. Between each word piece, you will place a dash, but you will not have a dash at the beginning or the end of the word. As you move across the word from left to right, place dashes after each group of N letters. The last group may be shorter than the rest if there are not enough letters to create a group of N.

Examples:

`COMPUTER 2` produces `CO-MP-UT-ER`
`GLASS 1` produces `G-L-A-S-S`
`PROGRAM 3` produces `PRO-GRA-M`
`FROG 5` produces `FROG`

Input: The first line of data will contain an integer T representing the number of data lines to follow. T will be in the range [1,20]. The next T lines of data will contain a string and an integer with one space in between. The string will have a length in the range [1,40]. It will consist of uppercase letters. The integer will be an integer with a value in the range [1,5].

Output: Each output will consist of uppercase letters separated by dashes if appropriate.

Sample input:

```
6
RIGATONI 2
PENNE 5
FETTUCCINE 4
LASAGNA 1
MACARONI 5
SPAGHETTI 3
```

Sample output:

```
RI-GA-TO-NI
PENNE
FETT-UCCI-NE
L-A-S-A-G-N-A
MACAR-ONI
SPA-GHE-TTI
```

4. Donghai

Program Name: Donghai.java **Input File: donghai.dat**

Donghai helped Joan solve the Invitational A problem counting words and calculating the average size of words. Now, he wants to work with letters instead of words. He wants to count the number of times each letter occurs in written prose. He will ignore punctuation marks and any other non-alphabetic characters. Uppercase and lowercase letters are considered the same letter. Once they have been counted, he will display the list of letters that occurred.

Can you help Donghai solve this problem?

Input: First line will contain an integer N with $0 < N \leq 10$ which is the number of paragraphs that follow. Each paragraph will occur on a single line of text and will not exceed 2000 characters in total length. Any ASCII character can occur in a paragraph but the only ones of interest are the standard alphabetic letters, both uppercase A…Z and lowercase a…z.

Output: Each test case must display list of letters in uppercase that have non-zero counts. Each output line with a letter will start with that letter followed by a colon "**:**" followed by the count with no spaces. Following each test case display a line containing 10 equal signs "==========".

Sample input: (*indented lines are continuation of previous line*)

```
3
Let's start with a short 1-line paragraph just to see a result!
Now, this second paragraph will be much longer which results in the sample input
   displaying with all continuation lines indented.  However, it will be a single
   unbroken line in the data file that is provided for the contest...
This is the third "paragraph" of sample data for this programming problem
```

Sample output:

```
A:6
E:5
G:1
H:3
I:2
J:1
L:3
N:1
O:2
P:2
R:5
S:6
T:8
U:2
W:1
==========
```

Sample output: *continued*

```
A:11
B:3
C:5
D:7
E:20
F:2
G:4
H:11
I:20
K:1
L:14
M:2
N:18
O:10
P:6
R:8
S:10
T:16
U:5
V:2
W:6
Y:1
==========
```

Sample output: *continued*

```
A:7
B:1
D:2
E:3
F:2
G:3
H:5
I:5
L:2
M:4
N:1
O:4
P:5
R:7
S:4
T:5
==========
```

5. Hiromi

Program Name: Hiromi.java **Input File: hiromi.dat**

Sudoku is a popular, logic-based game that anyone, who can count from 1-9, can play! A valid sudoku puzzle is a 9 by 9 grid of numbers such that each row, each column, and each 3x3 sub-grid region has the numbers 1-9 with no missing numbers or no duplicate numbers. A puzzle is traditionally given incomplete, and it is up to the player to fill in all the missing spaces. As the below puzzle shows, each row (horizontal), each column (vertical), and each of the nine 3x3 sub-grids, all have the numbers 1-9. See the below puzzle for a valid Sudoku puzzle, as well as the layout of the 9 sub-grids. (The darker, bolder lines denote the sub-grids.)

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

Hiromi loves solving Sudoku's but is known to make a mistake time-to-time. In an effort to ensure the final attempts are correct, Hiromi would like your help writing source code that verifies if a given attempt at a Sudoku puzzle is in fact correct, or not. Can you help with this?

Input: Input will begin with an integer N, the number of test cases. N is in the range [1,10]. Each of the following N test cases will start with 17 dashes to serve as a divider between each of the puzzles. Following the dashes will be the 9 x 9 Sudoku puzzle. All, individual numbers will be separated by a space to simplify the reading of the input. Each number present is guaranteed to be in range of [1,9].

Output: For each test case, you are to output: "`Puzzle #X: true`" if the puzzle meets the described criteria for a valid Sudoku puzzle, or "`Puzzle #X: false`", where `X` is the current test case. Remember, all nine rows, all nine columns, and all 9 sub-grids must have the numbers 1-9, to be a valid puzzle.

Sample input and output on next page…

~ Hiromi continued… ~

Sample input:

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

Sample output:

```
Puzzle #1: true
Puzzle #2: true
Puzzle #3: false
Puzzle #4: true
Puzzle #5: false
```

6. Jana

Program Name: Jana.java **Input File: jana.dat**

Jana is opening a baseball training facility and needs your help making some targets to help the pitchers practice their accuracy. You will be given the size of the target, and you need to print out the front of it to be painted onto the target. Each target will be size `ixi`, with rings of `'#'` characters, moving inwards and having every other ring be made of spaces (see sample output).

Input:
The input will begin with an integer `n`, denoting the number of test cases to follow. Each test case will consist of an integer, `i`, on its own line, denoting the height and width of the target to be printed.

Output:
Output the `ixi` target design as described above and shown in the sample output.

Sample input:
```
3
6
9
7
```

Sample output:
```
######
#    #
# ## #
# ## #
#    #
######
#########
#       #
# ##### #
# #   # #
# # # # #
# #   # #
# ##### #
#       #
#########
#######
#     #
# ### #
# # # #
# ### #
#     #
#######
```

7. Ksenia

Program Name: Ksenia.java **Input File: ksenia.dat**

Ksenia is very interested in graphs, especially what happens when you add a copy of a graph to another one. When Ksenia makes a copy of a graph and adds it to the original copy, she performs the following: for each vertex in the first copy, add an edge from that vertex to the corresponding vertex in the second copy.

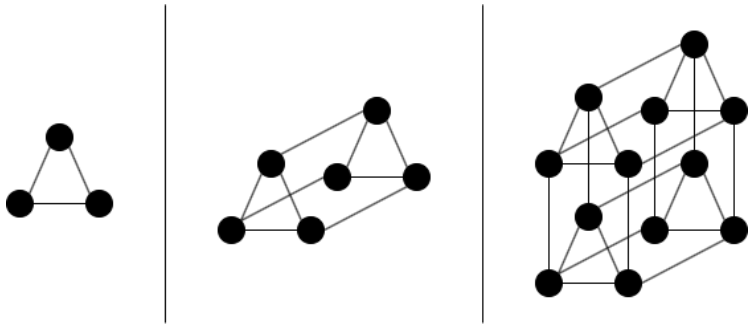


Figure 1 - Visualization of the first two copies of the first graph from the sample input

While conceptualizing what some of these graphs might look like might become difficult for graphs that have been copied multiple times, Ksenia simply is curious in the number of edges and vertices that a given graph will have once it has been copied a particular number of times. Help Ksenia by writing a program that, given a graph and the number of times that Ksenia wishes to duplicate that graph, returns the number of vertices and edges in that updated graph.

Input: The first line of input will consist of an integer n, denoting the number of queries to follow. For each of the n queries, the first line of input will consist of two integers, V and E, denoting the number of vertices and edges in the graph, where vertices are numbered 0 through $V-1$. Thet next line will consist of E comma-separated pairs of space-separated integers v_a and v_b denoting two vertices that form an edge in the graph. The final line will consist of a single integer m, denoting the number of times that Ksenia wishes to duplicate the graph.

It should be noted that $n \geq 1$, $V \geq 1$, $E \leq \frac{V(V-1)}{2}$, and $m \geq 1$. For a given query, if $E = 0$, note that there won't be a line denoting the edges. Lastly, each of the E edges will be unique, and there will never exist an edge between a given vertex and itself.

Output: For each of the n queries, print on its own line two space-separated integers V' and E' denoting the number of vertices and edges, respectively, in the final copy of the graph after it has been duplicated m times.

Sample input:

```
3
3 3
0 1,1 2,2 0
3
1 0
7
5 4
0 1,1 2,2 0,3 4
4
```

Sample output:

```
24 60
128 448
80 224
```

8. Marta

Program Name: Marta.java **Input File: marta.dat**

Marta has been studying hard for the upcoming UIL season but is having some difficulty with the classic bubble sort for arrays. She understands that every pass through the array will move smaller items one direction and larger items the other direction. An ascending sort will move smaller items towards the lower or [0] array position while larger items move towards the higher-end of the array. A descending sort would move items the opposite directions.

She would like to be able to confirm her manual sort progress after a specific number of passes to view how the arrangement should be changing. Here is an example:

Initial	95	11	68	13	33	26	24	60	56	47	79	21
Pass 1	11	68	13	33	26	24	60	56	47	79	21	95
Pass 2	11	13	33	26	24	60	56	47	68	21	79	95
Pass 11	11	13	21	24	26	33	47	56	60	68	79	95

However, the sort may not get to the requested pass count, depending on how quickly the numbers settle into their correct positions. In that case, the numbers will be fully sorted but Marta wants to know how many passes it actually took to complete the sort when that happens.

Can you help Marta modify her sort to output a list of numbers a specific pass of the sort?

Input: First line will contain an integer T with $1 \leq T \leq 10$, the number of test cases. Each test case will consist of two lines of space-separated integer data. First line will contain an integer N with $10 \leq N \leq 50$, the number of integers to be sorted, followed by another integer P with $1 \leq P < N$, the sort pass after which the array content will be displayed. The next line will contain N integers in range [10, 99], the data to be sorted into ascending order but displayed after only P or fewer passes have been completed.

Output: Each test case will produce 1 line of output that starts with "`Test case #1 pass #2: `" where `#1` is the test case number and `#2` is the actual number of passes completed which can be less than P because of the early exit technique from a proper bubble sort.

Sample input:

```
3
12 1
95 11 68 13 33 26 24 60 56 47 79 21
12 2
95 11 68 13 33 26 24 60 56 47 79 21
25 23
71 40 52 66 95 80 61 56 26 64 34 47 99 51 96 86 63 55 49 72 82 43 93 42 76
```

Sample output: (*lines that are indented are continuation of previous line*)

```
Test case 1 pass 1: 11 68 13 33 26 24 60 56 47 79 21 95
Test case 2 pass 2: 11 13 33 26 24 60 56 47 68 21 79 95
Test case 3 pass 21: 26 34 40 42 43 47 49 51 52 55 56 61 63 64 66 71 72 76 80 82 86 93
   95 96 99
```

9. Prateek

Program Name: Prateek.java **Input File: prateek.dat**

Prateek is a fan of the binary search algorithm and is looking at the efficiency of the algorithm. In particular, he is curious in the number of steps that it takes to find a given element in a sorted list using the binary search algorithm. A copy of the binary search algorithm has been provided to the right.

```
BINARYSEARCH(A, s, t)
  let A be the list of numbers
  let s be the size of the list
  let t be the target
  l ← 0
  r ← s − 1
  m ← ⌊(l+r)/2⌋
  while (l ≤ r)
    if (A[m] == t)
      return m
    else if (A[m] < t)
      r ← m − 1
    else
      l ← m + 1
    m ← ⌊(l+r)/2⌋
```

As a result, Prateek wants to classify each item in a sorted list by how quickly each item will be found using the algorithm. More specifically, for each item in the list, Prateek wants to put that element in a set, where each element in a given set share the same number of iterations of the binary search algorithm's while loop needed to be discovered.

Help Prateek solve this problem by writing a program that breaks down a sorted list into unique sorted sets where each element in a given set has the same number of iterations needed as the other elements in the set.

Input:
The first line will consist of a single integer, n, denoting the number of lines to follow. The next n lines will each consist of a space-separated list of sorted integers. It should be noted that $1 \leq n \leq 50$, and the length of any given list will be no more than 100.

Output: For each of the n sorted lists, output the analysis of the list as follows: on the first line, print the string "`List: <LIST>`", where `<LIST>` is a space-separated string representation of the sorted list. Then, for each unique set generated by Prateek's program, print out "i`-iterations: <SET>`", where i is the number of iterations needed to find the elements contained in `<SET>`, and `<SET>` is a space-separated string representation of the sorted set of elements which all take i iterations of the binary search algorithm to find. Print out the sets in increasing order in relation to the value of i.

Sample input:
```
3
2 4 6 8 9
1000
2 3 5 7 11 13 17 19 23 27 29 31
```

Sample output:
```
List: 2 4 6 8 9
1-iterations: 6
2-iterations: 2 8
3-iterations: 4 9

List: 1000
1-iterations: 1000

List: 2 3 5 7 11 13 17 19 23 27 29 31
1-iterations: 13
2-iterations: 5 23
3-iterations: 2 7 17 29
4-iterations: 3 11 19 27 31
```

10. Sofia

Program Name: Sofia.java **Input File: sofia.dat**

Sofia (who you met while on vacation with your friend Rodrigo), is going on vacation, and needs to find out if she can get from certain cities to certain other cities with the amount of money she has saved for her traveling. You will be given the prices to fly between two cities, and then two cities and an amount of money. You need to write a program to determine if that is enough money to fly between the two given cities. You need to determine the shortest path from the starting to the ending city, and find out if the amount of money Sofia has will be enough to get between the two cities.

Input: The input will begin with two integers, f, denoting the number of flights between 2 cities listed, and n, denoting the number of test cases. The next f lines will each contain 2 strings and an integer, all separated by spaces, denoting the names of the two cities and the price of the flight in either direction, respectively. The following n lines will each contain a test case, containing two strings and an integer, denoting the starting city and ending city, and the amount of money Sofia has to get from the starting to the ending city.

Output: For each test case, if there is enough money to get from the starting city to the ending city output the string "`[The name of the ending city] is always a good idea.`". Otherwise, output "`There's no place like home.`".

Sample input:

```
6 2
Paris Dallas 1200
Paris Prague 650
Prague Munich 300
Paris Munich 300
Berlin Prague 400
Berlin Dallas 2300
Dallas Prague 1550
Dallas Berlin 2200
```

Sample output:

```
There's no place like home.
Berlin is always a good idea.
```

11. Tyler

Program Name: Tyler.java **Input File: tyler.dat**

Tyler is a rich philanthropist and is a big fan of the trend of "doubling it and giving it to someone else." This trend consists of starting with a single dollar and giving participants the option of either doubling the current dollar amount and giving it to someone else, or keeping the current amount of money for themselves. This continues until someone chooses to take the money for themselves. Due to Tyler's large wealth, he has larger ambitions.

Rather than simply doubling the current amount of money and giving it to someone else, Tyler prefers giving his participants options of what to multiply the current amount of money by. As a result, he is curious whether certain dollar amounts are obtainable given a list of options to multiply the current amount of money by. You have been contracted by Tyler to develop a program which can help him automatically solve such a problem.

Input: The first line of input will consist of two integers, n and m – the number of options for multipliers and the number of queries, respectively. The next line will consist of a list of n space-separated integer multipliers. The following line will consist of a list of m space-separated integer queries indicating target dollar amounts that Tyler is interested in. It should be noted that $1 \leq n, m \leq 10^3$.

Output: For each of the m queries, print on its own line the string "Target <t>: <ANS>", where <t> is the ith target and <ANS> is the string "YES" or "NO" corresponding to whether or not the ith target is obtainable using any sequence of the m multipliers, each of which can be used any number of times.

Sample input:

```
5 10
2 3 5 7 11
2 13 5 26 121 77 15 144 154 5336100
```

Sample output:

```
Target 2: YES
Target 13: NO
Target 5: YES
Target 26: NO
Target 121: YES
Target 77: YES
Target 15: YES
Target 144: YES
Target 154: YES
Target 5336100: YES
```

12. Vika

Program Name: Vika.java **Input File: vika.dat**

You and your new friend in English class, Vika, have been playing a game. Vika gives you 2 words and its your job to find the longest sequence of letters that exists in both words. This sequence can be made up of any characters in the words, the only condition is that they appear in the same order in both words. For example, the sequence AC is common to both BAC and ABC.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of two space separated strings for you to play the game with. Determine the length of the longest common subsequence of these two strings.

Output: Output the length of the longest possible subsequence that is shared between these two strings.

Sample input:

```
3
ABC BAC
Someone Stonecold
Applebees Geronimo
```

Sample output:

```
2
4
1
```