University of Texas at Austin - High School Computer Science Contest
Problems by Jaime Rivera, Moises Holguin, and Mike Scott

General Notes:

1. Do the problems in any order you like.

2. For problems that read from a data file: When the program is judged, the data file will be in the same directory as your program. Do not include any extraneous path information in your program, just the name of the data file. Data file names are case sensitive. If you do not use the correct file name as listed in the question your program may be judged as an incorrect submission.

3. Submit your source code, the .java file, for judging via the PC^2 software. File names are case sensitive. If you do not use the correct file name for the Java file your submission may be graded as incorrect submission.

4. All problems are worth 60 points. Incorrect submissions receive a deduction of 5 points, but **ONLY** if the problem is ultimately judged correct. Problems may be reworked and resubmitted as many times as you like.

5. There is no extraneous input. All input is exactly as specified in the problem. Integer input will not have leading zeros unless specified by the problem.

6. Your program shall not print extraneous output. Follow the format exactly as given in the problem statement and as shown in example output.

7. The time limit for problems is 120 seconds. If the program has not terminated after 120 seconds on the judge's computer, the program will be judged as an incorrect submission.

Number	Problem Name
1	Chuck
2	Count
3	Cubes
4	Dimensions
5	Goals
6	Grade
7	Majority
8	Optimal
9	Rebate
10	Royal
11	Tunnels
12	Us

1. Seventy-Three is the Chuck Norris of Numbers

Program Name: Chuck.java **Input File: chuck.dat**

From *The Big Bang Theory - The Alien Parasite Hypothesis* episode:

Sheldon: What is the best number? By the way, there's only one correct answer.
Raj: 5,318,008?
Sheldon: Wrong! The best number is 73. [Short silence] You're probably wondering why.
Leonard & Howard: No no, we're good.
Sheldon: 73, is the 21st prime number, its mirror 37 is the 12th and its mirror 21 is the product of multiplying, hang on to your hats, 7 and 3. Did I lie?
Leonard: We get it! 73 is the... Chuck Norris of numbers!
Sheldon: Chuck Norris wishes! In binary, 73 is a palindrome, 1001001, which backwards is 1001001, exactly the same. All "Chuck Norris" gets you backwards is "Sirron Kcuhc"!

Write a program that finds how many of the representations of a given number in the bases from 2 to 16 are palindromes.

A palindrome is a word, phrase, or sequence that reads the same backwards and forwards. In your case, you will be looking for numbers that are palindromic, such as 12321.

Base conversions of 73_{10} from binary (base 2) to hexadecimal (Base 16)

Base 2: **1001001**
Base 3: 2201
Base 4: 1021
Base 5: 243
Base 6: 201
Base 7: 133
Base 8: **111**
Base 9: 81
Base 10: 73
Base 11: 67
Base 12: 61
Base 13: 58
Base 14: 53
Base 15: 4d
Base 16: 49

73_{10} has two palindromic representation in bases 2 to 16, 1001001_2. and 111_8.
Leading zeros may NOT be added to make a number a palindrome. For example 10, is not a palindrome.

Input

- The first line contains a single integer N that indicates the number of data sets.
- Each data set consists of a single integer x such that 0 <= x <= 2,000,000,000

Output

For each data set print out the number representations of X in bases 2 to 16 that are palindromic.

Example Input File

```
5
73
17
532
329
29
```

Example Output To Screen

```
2
3
1
0
1
```

1. Seventy-Three is the Chuck Norris of Numbers

Program Name: Chuck.java **Input File: chuck.dat**

Judges Input File

```
23
73
17
532
329
29
10
679
526
129
577
53
625
235
906
23
630
2000000000
1234554321
999999999
855222558
1111111111
0
1
```

Judges Output to Screen

```
2
3
1
0
1
9
0
0
2
0
0
0
4
1
1
1
0
1
1
1
1
15
15
```

2. Count Them Up

Program Name: Count.java Input File: count.dat

Did you ever wonder how many different ways there were to pick a 1st, 2nd, and 3rd place winner out of 10 runners? Ever want to know how many different possible hands of poker there are? If you want to figure out how many different groups of 6 you can make from a group of 30 people, you need to learn how to count using combinations and permutations.

A permutation is one of the different arrangements of a group of items where order matters. In other words the arrangement (a, b) is distinct from the arrangement (b, a). The following formula is used to calculate the number of permutations given n total elements and choosing r of them to be in a given permutation:

$$\frac{n!}{(n-r)!}$$

A combination is one of the different arrangements of a group of items where order does not matter. In other words the arrangement (a, b) is equivalent to the arrangement (b, a). The following formula is used to calculate the number of combinations given n total elements and choosing r of them to be in a given combination:

$$\frac{n!}{r!\,(n-r)!}$$

Write a program to calculate the number of different combinations or permutations given different values of n and r.

Input

- The first line of the data set is a number M that indicates the number of data sets.
- Each line contains a number $1 \leq n \leq 60$, followed by either a P or C, then a second number $0 \leq r \leq 60$. You are guaranteed $r \leq n$. No spaces will be present in a data set.
- nPr is to be read as "n pick r". Calculate the number of permutations.
- nCr is to be read as "n choose r". Calculate the number of combinations.

Output

Display the number of different combinations or permutations there are for each data set, one line of output per dataset.

Example Input File

```
6
16C3
10P2
25C13
30P4
60P10
60C10
```

Example Output To Screen

```
560
90
5200300
657720
2735898472315008 00
75394027566
```

2. Count Them Up

Program Name: Count.java Input File: count.dat

Judges Input File

```
13
16C3
10P2
25C13
30P4
60P10
60C10
16C12
20P13
25C24
20C0
10P0
25P15
60C55
```

Judges Output to Screen

```
560
90
5200300
657720
2735898472315008 00
75394027566
1820
482718652416000
25
1
1
42744736671436800 00
5461512
```

3. Cubes! Don't Assemble!!

Program Name: Cubes.java Input File: cubes.dat

Captain America has just defeated The Red Skull, and recovered a cache of powerful, but unstable power cubes. Each power cube contains an amount of dark energy. Cap has to get the power cubes back to the authorities in steel boxes. Problem is if a set of three cubes is placed in the same box and the sum of the energy of two of the cubes equals the energy of a third cube, then the set becomes unstable and explodes. The amount of dark in each cube is always an integer greater than 0.

For example cubes with energies equal to 1, 2, and 3 cannot be placed in the same box because 1 + 2 = 3. The minimum number of boxes Cap must use in this instance is 2, in order to keep those three cubes from exploding.

Another example: Cubes with energies 10, 10, 13, 16, and 19 could all go in a single box because there is no pair of cubes whose energies sum to one of the other cubes. Likewise, cubes with energies 1, 1, 1, 1, 1, 1, 1, 1, 1, and 1, can all go in the same box.

Only triples are unstable. Three cubes whose energies sum to a fourth cube's energy can be placed in the same box. Examples, cubes with energies 2, 2, 2, and 6 can be placed in the same box even though 2 + 2 + 2 = 6.

The boxes are large enough that if no deadly triples exits, all the cubes can be placed in a single box.

Input

- The first line of the data set is a number N that indicates the number of data sets.
- Each data set is a series of integers on a single line separated by spaces indicating the energy of the cubes in that data set.
- All cube energies C will have values such that 0 < C < 2,000,000,000

Output

For each data set display the minimum number of boxes Cap must use to transport the cubes such that no deadly triples exist in the same box.

Example Input File (Note, the last data set is wrapped on this page, but will be on a single line in the actual input file.)

```
8
1 2 3
1 2 3 1 2 3
10 10 13 16 19
1 1 1 1 1 1 1 1 1 1
1 3 5 7 9 3 5 19
2 2 2 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
```

Example Output To Screen

```
2
2
1
1
1
1
3
4
```

3. Cubes! Don't Assemble!!

Program Name: Cubes.java **Input File: cubes.dat**

Judges Input File (Blank lines added for clarity between data sets that are wrapped. No blank lines in actual input file.)

```
13
1 2 3
1 2 3 1 2 3
10 10 13 16 19
1 1 1 1 1 1 1 1 1 1
1 3 5 7 9 3 5 19
2 2 2 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52

18 17 5 12 9 18 5 15 5 11 10 14 14 15 3 9 11 13 13 10 4 17 14 19 9 9 3 7 6 1
17 11 16 11 1 16 1 7 6 7

1 2 3 6 10 16 100 200 300 1000 2000 3000 10000 20000 30000 100000 200000
300000

67 8 41 10 98 14 92 60 41 73 10 63 46 7 89 44 100 76 47 72 68 39 48 68 44 9
18 73 8 41 21 43 41 59 93 52 54 81 43 48 95 56 56 76 28 7 57 80 54 43

76 15 43 55 80 78 28 51 47 27 66 17 98 24 8 20 75 34 52 81 29 41 82 7 93 96
89 17 96 54 71 62 53 92 6 86 87 5 55 53 64 33 96 55 54 86 87 79 91 78 85 8 36
14 9 22 30 91 37 65

1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

Judges Output to Screen

```
2
2
1
1
1
1
3
4
3
2
3
3
2
```

4. Missing Dimensions

Program Name: Dimensions.java **Input File: dimensions.dat**

UTCS has a new building! The move took place last week. During the move all the equipment and box dimension data was lost. In order to make everything fit in the new building, we need to recover the dimension data; Some data still remains, if we can recover it. The data is just a bit cluttered.

The dimensions for each piece of equipment and box has the following form: `AxBxC` *where A, B, and C are integers, floating point numbers, or numbers in exponential form. Spaces and tabs may be present between the numbers and the separators. (x's). Separators may be upper or lower case x's.*

Floating point numbers will have a digit before the decimal point. Numbers in exponential form have the following format: `X.YeZ`, or `X.YEZ`, where `X`, `Y`, and `Z` are non-negative integers, with one or more digits.

Input

- The first line contains a single integer N that indicates the number of data sets.
- Each data set will consist of single line of text, which may or may not contain a single piece of dimensional data in the form `AxBxC` as described above.

Output

For each data set print, print out the dimension of the item in the given format with any whitespace removed. Print all separators as lower case x's and all exponent e's as lower case e's. If no dimension is present print `NO DIMENSION FOUND`.

Example Input File

```
10
1x-2X3
UTCS-Competition-Box-32x17x3-Prod#213
UPSPKG-SIZE8.11E27X 25.3329 x 16
Chocolate-xx-0.05x31x5xxxxX
No Dimension 32x32y32
xxx
-1X5.6231x14.100E10-
-1X5.6231x-14.100E10-
-1X5.6231   x    14.100E10-
333X1   x   22    aj912dskljc nsdx123x112dss
```

Example Output To Screen

```
NO DIMENSION FOUND
32x17x3
8.11e27x25.3329x16
0.05x31x5
NO DIMENSION FOUND
NO DIMENSION FOUND
1x5.6231x14.100e10
NO DIMENSION FOUND
1x5.6231x14.100e10
333x1x22
```

4. Missing Dimensions

Program Name: Dimensions.java **Input File: dimensions.dat**

Judges Input File

```
20
1x-2X3
UTCS-Competition-Box-32x17x3-Prod#213
UPSPKG-SIZE8.11E27X 25.3329 x 16
Chocolate-xx-0.05x31x5xxxxX
No Dimension 32x32y32
xxx
-1X5.6231x14.100E10-
-1X5.6231x-14.100E10-
-1X 5.6231   x    14.100E10-
333X1   x   22    aj912dskljc nsdx123x112dss
(*&(*@&#(*@(*Exsdiasdxsakjalkjsda
810x1.2323xkjallasdl0912812i8
0120930921x0213-0213Xklask1x1.2x    81.122e10201klsdlksdlk1
kksd)k2j3)DxxXXXXxx1x1x     1xhdhh12
jhskjdhsakjd193219x-12x13213x12
dskki123(*(1x-12x2112
312    232112     233211  21312
1.2e3x4.5E6X7.8E99
1.2e3x4.5E6X7.8E-99
1X-2x3
```

Judges Output to Screen

```
NO DIMENSION FOUND
32x17x3
8.11e27x25.3329x16
0.05x31x5
NO DIMENSION FOUND
NO DIMENSION FOUND
1x5.6231x14.100e10
NO DIMENSION FOUND
1x5.6231x14.100e10
333x1x22
NO DIMENSION FOUND
NO DIMENSION FOUND
1x1.2x81.122e10201
1x1x1
12x13213x12
NO DIMENSION FOUND
NO DIMENSION FOUND
1.2e3x4.5e6x7.8e99
1.2e3x4.5e6x7.8
NO DIMENSION FOUND
```

5. Anchorage or Bust

Program Name: Goals.java Input File: goals.dat

Texas 4000 for Cancer: (from Wikipedia)

"Texas 4000 was founded by Chris Condit in 2004, then a student at UT Austin, as a student organization inspired by riders of the Hopkins 4K (4K for Cancer, Inc.). Diagnosed at age 11, Condit himself is a Hodgkin's lymphoma survivor. He conceived Texas 4000 as a way to continue the fight against cancer.

Each year a new group of almost 60 University of Texas at Austin students make a 70 day, 4,687 mile bike trek from the Texas campus in Austin, Texas to Anchorage, Alaska. Each rider meets training and community service expectations as well as a $4,500 fundraising goal that goes toward Texas 4000's mission of Hope, Knowledge and Charity from Austin to Anchorage. To date, Texas 4000 has donated over $4 million to M.D. Anderson Cancer Center and the American Cancer Society. "

Write a program to determine how many days it takes a cyclist to reach their first and second mileage goals while training for the Texas 4000 and their average ride length while training.

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set consists of three lines
- The first line is a single integer that indicates the number of training rides a cyclist has completed, M. M >= 0
- The second line will be M integers each separated by a single space. Each integer, R, represents the number of miles a cyclist rode on a training ride. All R will be greater than 0.
- The third line will be two integers, G1 and G2, separated by a single space. 0 < G1 < G2. G1 represents a cyclists first goal for total miles and G2 represents the same cyclists second goal for total miles.

Output

For each data set print out a single line with three values: the number of training rides to meet or exceed the first goal, the number of training rides to meet or exceed the second goal, and the average number of miles ridden per training ride, truncated to the nearest integer. If a goal is not met print out `FAIL`. If a rider completed 0 training rides print an average mileage of 0.

Example Input File

```
5
0

50 100
3
25 30 34
30 40
7
25 30 25 30 25 40 50
50 100
1
1
10 20
10
25 25 30 50 50 100 100 40 50 30
130 500
```

Example Output To Screen

```
FAIL FAIL 0
2 2 29
2 4 32
FAIL FAIL 1
4 10 50
```

5. Anchorage or Bust

Program Name: Goals.java Input File: goals.dat

Judges Input File (The ride data for the last two data sets is on a single line in the judges input file, but is wrapped on this page.)

```
11
0

50 100
3
25 30 34
30 40
7
25 30 25 30 25 40 50
50 100
1
1
10 20
10
25 25 30 50 50 100 100 40 50 30
130 500
0

1 2
12
25 25 30 50 10 10 12 20 100 150 150 200
200 300
10
100 100 100 100 100 100 100 100 100 100
50 51
11
72 29 102 281 10 84 29 10 28 28 10
500 10000
201
121 32 103 71 21 22 41 42 101 20 25 41 35 38 45 24 86 36 10 35 10 11 66 20 73
35 103 20 80 39 12 43 37 78 15 23 109 12 35 18 25 70 35 34 35 73 36 28 31 10
80 23 75 16 77 10 25 22 21 31 27 35 10 20 42 43 37 21 34 12 31 26 42 20 32 20
10 10 33 16 41 18 41 13 10 43 26 31 18 46 42 26 29 21 46 49 24 36 26 29 31 31
22 23 133 32 76 11 31 12 18 26 36 26 75 16 27 34 26 25 90 18 18 24 26 34 23
28 28 74 38 11 53 103 40 40 70 34 43 31 42 46 36 73 10 37 25 7 105 51 54 88
15 36 26 102 33 27 25 36 20 36 36 36 39 36 36 74 27 37 22 17 31 8 20 84 20 70
19 36 36 33 36 36 73 38 51 47 28 12 45 22 17 25 25 25 17 31 32 35 19
4000 6000
251
38 24 28 36 31 58 45 16 102 24 35 51 54 15 101 43 67 109 26 33 40 28 41 71 47
36 36 37 38 37 38 74 109 25 38 38 19 33 36 59 42 70 32 26 38 9 74 15 31 14 32
29 31 23 28 41 19 33 30 31 30 40 78 37 10 41 12 24 22 16 43 38 15 32 10 30 37
27 10 33 27 34 34 29 66 27 48 28 30 52 10 40 18 37 10 80 31 15 15 17 11 30 44
28 103 28 43 28 92 15 18 25 23 15 44 22 91 7 36 15 15 10 27 5 66 28 34 32 12
22 11 26 37 30 69 36 30 40 34 36 35 38 43 35 39 31 33 17 26 41 45 49 27 37 55
15 45 56 101 20 38 24 66 21 51 40 67 20 51 65 70 20 38 39 110 7 16 35 37 38
31 23 76 62 18 23 35 20 26 23 36 17 26 27 24 37 49 66 18 21 64 20 15 25 17 25
30 32 11 32 32 10 53 10 35 53 39 30 51 52 28 22 36 40 32 33 27 33 32 61 39 35
41 12 15 101 36 33 43 36 32 15 16 100 10 45 10 16 27 19 62
3000 7500
```

Judges Output to Screen

```
FAIL FAIL 0
2 2 29
2 4 32
FAIL FAIL 1
4 10 50
FAIL FAIL 0
9 10 65
1 1 100
6 FAIL 62
107 157 37
78 207 36
```

6. Making the Grade

Program Name: Grade.java Input File: grade.dat

Many students stress about their grade, as the final exam gets closer. *What do I need to make to pass this class?* Write a program to determine the minimum grade a student needs to make on their final test to finish the class with an A, B, or C.

The grade formula for G, a student's grade in a course is:

Homework Percent*Homework Average + Quiz Percent*Quiz Average + Test Percent*Test Grade = G

The letter grade assigned is as follows: **A** ($90 \leq G \leq 100$), **B** ($80 \leq G < 90$), **C** ($70 \leq G < 80$).

Homework average is the average of the student's homework scores and quiz average is the average of the students quiz score. Homework and quiz averages are computed as floating point numbers. A student's final exam score must be between 0 and 100 inclusive.

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will consist of exactly 4 lines.
- Line 1 will be the name of the student.
- Line 2 will be the 3 percentages Homework, Quiz, and Test, respectively, used to calculate the final grade for the class. These will be integers all greater than or equal to zero and the sum of the integers shall equal 100.
- Line 3 will be the Homework grade(s) of the student. There will be at least one. All homework grades will be greater than or equal to 0 and less than or equal to 100.
- Line 4 will be the Quiz grade(s) of the student. There will be at least one. All quiz grades will be greater than or equal to 0 and less than or equal to 100.
- spaces separate the integer values in the input file

Output

For each data set display the name of the student on a single line. Then print out 3 lines with the desired grade, a tab, then the minimum grade needed on the final test to achieve the corresponding letter grade. If it is not possible to earn that letter grade then print out SORRY. When printing the necessary grade round any fractional scores up to the next integer. For example, if a student needs a 72.01 on the final test print the required score as 73.

Example Input File:

```
3
Alex
30 30 40
100 100
100 100
Stacy
25 25 50
70 80 90
60 75 90
Susan
40 40 20
100 100 100 100 99
100 100 100 100 100 99
```

Example Output To Screen

```
Alex
A  75
B  50
C  25
Stacy
A  SORRY
B  83
C  63
Susan
A  51
B  1
C  0
```

6. Making the Grade

Program Name: Grade.java **Input File: grade.dat**

Judges Input File

```
10
Alex
30 30 40
100 100
100 100
Stacy
25 25 50
70 80 90
60 75 90
Susan
40 40 20
100 100 100 100 99
100 100 100 100 100 99
Ryan
25 35 40
65 83 90 24
84 34 50 100 100
Zach
90 5 5
30 34 100
56 67 78
Ashley
45 45 10
100 100
100 100 100
Joe
40 30 30
100 100
100
Mike
20 20 60
0 0 0 0
0 0 0 0 0 0 0 0
Mean
0 0 100
100 100 100
100 100 100 100 100 100
AnotherMean
5 5 90
0 0 0 0 0
0 0 0 0 0 0 0 0
```

Judges Output to Screen

```
Alex
A   75
B   50
C   25
Stacy
A   SORRY
B   83
C   63
Susan
```

```
A   51
B   1
C   0
Ryan
A   SORRY
B   95
C   70
Zach
A   SORRY
B   SORRY
C   SORRY
Ashley
A   0
B   0
C   0
Joe
A   67
B   34
C   0
Mike
A   SORRY
B   SORRY
C   SORRY
Mean
A   90
B   80
C   70
AnotherMean
A   100
B   89
C   78
```

7. Majority

Program Name: Majority.java **Input File: majority.dat**

Write a program to determine if a majority exists within a list of candidates. A majority exists when a single candidate receives **more than half** of the total votes.

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will consist of a single String, S, which represents the list of up to a million votes.
- Each vote will be represented as a single character. Valid characters for candidates are any non-whitespace ASCII character. Upper and lower case letters represent different candidates.
- There will not be any spaces between characters and all characters will be on a single line.

Output

For each data set, if there is a majority, print the character, which represents the majority; otherwise, print `NO MAJORITY`.

Example Input File

```
5
AAAABCBC
DDDEEOO
BBBBBBBB
abba&*-aabbabbaaba*-281aaaaaaaAaaaaBBBaa
AAaa
```

Example Output To Screen

```
NO MAJORITY
NO MAJORITY
B
a
NO MAJORITY
```

7. Majority

Program Name: Majority.java **Input File: majority.dat**

Judges Input File (Note, three lines of the judges file contained over 500,000 characters each. Those are not shown.)

```
11
AAAABCBC
DDDEEOO
BBBBBBBB
abba&*-aabbabbaaba*-281aaaaaaaAaaaaBBBaa
AAaa
$
AA
Aa
<NOT SHOWN DUE TO LENGTH>
<NOT SHOWN DUE TO LENGTH>
<NOT SHOWN DUE TO LENGTH>
```

Judges Output to Screen

```
NO MAJORITY
NO MAJORITY
B
a
NO MAJORITY
$
A
NO MAJORITY
NO MAJORITY
NO MAJORITY
a
```

8. Optimal Prime

Program Name: Optimal.java **Input File: optimal.dat**

Unrelated to Optimus Prime, write a program to determine the number of primes that can be created by all combinations of particular sequences of primes.

A sequence of primes within the range [0, 7]:

2, 3, 5, 7

A combination of sequences:

Sequence 1: 2, 3, 5, 7
Sequence 2: 11, 13

Combinations from the two sequences: 211, 213, 311, 313, 511, 513, 711, 713

A combination in this example is obtained by concatenating a prime number from sequence 1 to a unique prime in sequence 2. Of those 8 numbers created from the 2 seque3nces, 3 are prime: `[211, 311, 313]`

The following can be done with more than 2 sequences.

Combination with 3 sequences:

Sequence 1: 2, 3, 5, 7
Sequence 2: 11, 13
Sequence 3: 5, 7, 11

Combinations: 2115, 2117, 21111, 2135, 2137, 21311, 3115, 3117, 31111, 3135, 3137, 31311, 5115, 5117, 51111, 5135, 5137, 51311, 7115, 7117, 71111, 7135, 7137, 71311

A prime from each sequence must be chosen. Recall the numbers are created by concatenating a number from sequence 2 to the end of a number from sequence 1 and so forth. If a sequence does not contain any primes between the upper and lower bounds, nothing is concatenated to the number for that sequence. If a series of sequences does not yield any prime numbers the sequence is considered a `NO PRIMES` data set.

Input

- The first line will contain a single integer N that indicates the number of data sets. $0 < N < 100$.
- Each data set will consist of an integer $0 < S < 10$ that represents the number of sequences for the data set.
- Each sequence will have 2 positive integers A and B (B >= A), which denote the range [A, B] (A to B inclusive). A >= 2
- All values generated from sequences will be less than 2^31 - 1.

Output

For each data set print out the number of primes that exist within the combination formed by the sequences of the data set. If there were no primes, print `NO PRIMES`.

Example Input File

```
6
2
2 7
11 13
3
2 7
11 13
5 11
1
2 100
4
100 132
999 1500
2 3
5 10
3
2 10
102 108
55 56
2
8 9
8 9
```

Example Output To Screen

```
3
2
25
134
2
NO PRIMES
```

8. Optimal Prime

Program Name: Optimal.java **Input File: optimal.dat**

Judges Input File

```
12
2
2 7
11 13
3
2 7
11 13
5 11
1
2 100
4
100 132
999 1500
2 3
5 10
3
2 10
102 108
55 56
2
8 9
8 9
4
2 99
2 99
3 10
40 60
2
900 1000
600 900
5
2 10
2 10
2 10
2 10
2 10
2
1000 1500
99999 100100
9
2 3
5 7
2 3
2 3
2 3
2 3
5 7
7 9
5 6
9
2 3
5 7
2 3
2 3
```

```
2 3
2 3
5 7
5 7
5 7
```

Judges Output to Screen

```
3
2
25
134
2
NO PRIMES
1594
81
128
33
NO PRIMES
29
```

9. Store Rebate

Program Name: Rebate.java **Input File: rebate.dat**

Buying books for classes can be very expensive especially on a college budget. Luckily, there is a new store in town that offers rebates to customers. They are willing to give students 10% back on the total amount a student spends at the store minus any returns the student makes. As a bonus, the store rounds the rebate up to the nearest 5 dollars. Write a program to calculate how much money each students get back in rebates.

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will begin with the student's name on a single line.
- The next line will have the number of transactions T the student made.
- The next T lines will be the transaction amounts. A positive number represents a purchase, while a negative number represents a return. All values will have a leading digit (possibly a zero) and 2 decimal places. It is possible to have a return before any purchases. This is not necessarily a bad data set.

Output

The name of the student followed by a space followed by the amount of their rebate. Do not include any decimal places. If a student's returns exceed their purchases, print out `BAD DATA`.

Example Input File

```
4
Samantha
3
25.25
23.00
25.15
Josh
7
21.65
56.29
251.14
129.90
541.25
433.00
-56.29
Oscar
3
300.00
150.00
-150.00
John
2
-150.16
125.99
```

Example Output To Screen

```
Samantha 10
Josh 140
Oscar 30
John BAD DATA
```

9. Store Rebate

Program Name: Rebate.java **Input File: rebate.dat**

Judges Input File

```
14
Samantha
3
25.25
23.00
25.15
Josh
7
21.65
56.29
251.14
129.90
541.25
433.00
-56.29
Oscar
3
300.00
150.00
-150.00
John
2
-150.16
125.99
Nada
10
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
10.00
Jesse
3
10.00
19.00
-29.00
Max
5
-12378.82
2231.23
19823.82
-2819.99
1827.83
Mike
4
-100.00
250.15
-200.00
-150.15
```

```
Max
8
816.29
126.72
12.42
20.19
0.23
0.00
-27.89
-72.28
Lima
1
0.01
Jack
1
0.00
terri
5
-142.21
160.12
-272.23
217.00
102.95
#$%$^
4
5.00
10.00
15.00
-5.00
Last
4
100.00
200.00
50.00
300.00
```

Judges Output to Screen

```
Samantha 10
Josh 140
Oscar 30
John BAD DATA
Nada 10
Jesse 0
Max 870
Mike BAD DATA
Max 90
Lima 5
Jack 0
terri 10
#$%$^ 5
Last 65
```

10. Honoring Coach Royal

Program Name: Royal.java Input File: NONE

Darrell K Royal passed away on November 7, 2012. D K Royal was the head coach of the Texas Longhorns from 1957 to 1976. The Longhorns football teams play their home games in Darrell K Royal–Texas Memorial Stadium, named after coach Royal since 1996.

Write a program to display a football field as shown below.

Input

- None

Output

Exactly as shown on the next page. There is no blank line before the first line of output. There are no extra spaces at the end of lines. The last line is output with a newline.


..#....#..
..#....#..
..######..
....#.....
....#.....
****
.\....\.
.\....\.
****10****
.\....\.
.\....\.
****20****
.\....\.
.\....\.
****30****
.\....\.
.\....\.
****40****
.\....\.
.\"DKR".
****50****
.\"DKR".
.\....\.
****40****
.\....\.
.\....\.
****30****
.\....\.
.\....\.
****20****
.\....\.
.\....\.
****10****
.\....\.
.\....\.
****
..#....#..
..#....#..
..######..
....#.....
....#.....

10. Honoring Coach Royal

Program Name: Count.java Input File: count.dat

Judges Input File

None

Judges Output to Screen

```
..#....#..
..#....#..
..######..
....#.....
....#.....
**********
*.\....\.*
*.\....\.*
****10****
*.\....\.*
*.\....\.*
****20****
*.\....\.*
*.\....\.*
****30****
*.\....\.*
*.\....\.*
****40****
*.\....\.*
*.\"DKR".*
****50****
*.\"DKR".*
*.\....\.*
****40****
*.\....\.*
*.\....\.*
****30****
*.\....\.*
*.\....\.*
****20****
*.\....\.*
*.\....\.*
****10****
*.\....\.*
*.\....\.*
**********
..#....#..
..#....#..
..######..
....#.....
....#.....
```

11. Steam Tunnels

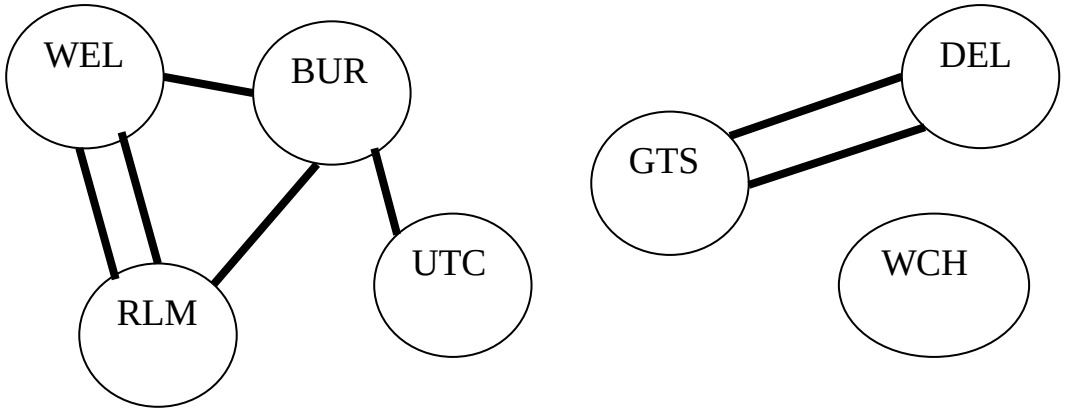
Program Name: Tunnels.java **Input File: tunnels.dat**

UT is a huge organization both in number of people and in physical size. The campus has its own power plants and utilities because it is cheaper to run them independently rather than buy power from the city of Austin. Part of the UT utility system is a network of steam tunnels that connects buildings underground. Urban legends claim students go spelunking in these tunnels. Not an advisable activity but CS students have always had an "affinity for disobedience."

Write a program that determines how many distinct parts exist in a given section of the steam tunnel system. A steam tunnel section is described as follows:

G = {WEL,BUR,RLM,GTS,DEL,UTC,WCH},{WEL,BUR},{WEL,RLM},{GTS,DEL},
{BUR,RLM},{RLM,WEL},{GTS,DEL},{UTC,BUR}

The first set of brackets contains the buildings in the section. This is followed by building pairs that indicate a tunnel exists between the two buildings in the pair. A part is any set of buildings that are connected by tunnels, directly or indirectly.



The section described above consist of 3 distinct parts, the (WEL, BUR, RLM) part, the (GTS, DEL) part, and the (WCH) part.

Input

- The first line contains a single integer N that indicates the number of data sets.
- Each data set consists of a single line. The first part of each data set is a list of the buildings in the section. Buildings will be distinct in the list for a given data set.
- The list of buildings in the data set is followed by pairs of buildings. Each pair indicates a tunnel exists between the two buildings. All buildings in the pairs will be present in the list at the start of the data set. Buildings names consist of one or more characters. Building names will not contain braces, commas, or spaces.
- There will be at least one building per data set. There may be zero tunnels for a data set. Tunnels may exist that connect a building to itself
- Brackets and commas are used to separate the initial list and pairs of buildings as shown above. There will be no extraneous spaces in the data set.

Output

For each data set print out the number of separate parts that exist in the steam tunnel section represented by the data set.

Example Input File (the last data set is wrapped on this sheet, but will be a single line in the input file)

```
7
{WEL,BUR,RLM},{WEL,BUR},{BUR,RLM},{RLM,WEL}
{JES,SZB},{JES,SZB}
{WEL,WCH,RLM}
{PAI,ACES,GATES,WEL},{GATES,WEL},{GATES,ACES},{WEL,PAI}
{WEL,wel,RLM},{wel,wel}
{WEL,wel,RLM},{wel,WEL}
{WEL,BUR,RLM,GTS,DEL,UTC,WCH},{WEL,BUR},{WEL,RLM},{GTS,DEL},
{BUR,RLM},{RLM,WEL},{GTS,DEL},{UTC,BUR}
```

Example Output To Screen

```
1
1
3
1
3
2
3
```

11. Steam Tunnels

Program Name: Tunnels.java **Input File: tunnels.dat**

Judges Input File (Blank lines added for clarity between some data sets. The blank lines are not in the actual data file.)

```
17
{WEL,BUR,RLM},{WEL,BUR},{BUR,RLM},{RLM,WEL}
{JES,SZB},{JES,SZB}
{WEL,WCH,RLM}
{PAI,ACES,GATES,WEL},{GATES,WEL},{GATES,ACES},{WEL,PAI}
{WEL,wel,RLM},{wel,wel}
{WEL,wel,RLM},{wel,WEL}
{WEL,BUR,RLM,GTS,DEL,UTC,WCH},{WEL,BUR},{WEL,RLM},{GTS,DEL},{BUR,RLM},{RLM,WE
L},{GTS,DEL},{UTC,BUR}

{Z,Y,X,W,V},{Z,Z},{Z,W},{Y,V}
{Z,Y,X,W,V},{V,Z},{Y,V},{Y,V},{W,W},{Y,Y},{Z,Z},{Y,V},{Y,Z},{Z,Z},{V,Y}
{Z,Y,X,W,V},{Y,Z},{X,W},{X,W},{V,V},{X,V},{Z,Y},{Z,Y},{X,W},{Z,V},{V,V},{Z,X}
,{W,X},{Z,Z},{W,Y},{V,Y},{W,Z},{Z,X},{X,V},{W,Z},{Y,Z}

{Z,Y,X,W,V,U,T,S,R,Q},{T,S},{Q,X},{Z,Q},{S,T},{Y,X},{Q,S},{T,Y},{U,T},{R,T},{
S,R},{U,W},{W,R},{X,U},{W,S},{X,W},{S,T},{V,Y},{S,Y},{R,R},{S,V}

{Z,Y,X,W,V,U,T,S,R,Q,P,O,N,M,L,K,J,I,H,G},{V,N},{N,T},{J,T},{P,S},{W,X},{O,M}
,{X,I},{X,T},{V,G},{R,K},{T,S},{Y,Y},{R,X},{K,M},{Z,Q},{Z,H},{O,X},{Y,X},{P,Z
},{U,H},{L,H},{T,Z},{N,M},{Y,J},{M,G},{Y,G},{Y,Q},{P,S},{W,L},{I,W}

{Z,Y,X,W,V,U,T,S,R,Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A,a,b,c,d,e,f,g,h,i,j,k,l,
m,n},{d,H},{j,G},{H,M},{V,M},{S,N},{M,g},{i,L},{h,a},{T,d},{h,l},{W,n},{G,j},
{Q,c},{m,H},{Q,H},{M,K},{J,M},{d,S},{h,i},{U,J},{D,X},{Q,m},{G,c},{O,Y},{n,W}
,{S,V},{C,C},{g,Z},{Y,n},{k,n},{X,F},{P,M},{F,J},{b,I},{g,d},{k,A},{c,O},{D,b
},{A,I},{J,J}

{Z,Y,X,W,V,U,T,S,R,Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A,a,b,c,d,e,f,g,h,i,j,k,l,
m,n,o,p,q,r,s,t,u,v,w,x,y,z},{j,A},{u,L},{D,L},{T,W},{I,i},{d,D},{m,m},{J,I},
{u,Q},{t,Q},{G,X},{l,t},{F,T},{F,s},{w,M},{w,j},{S,u},{K,T},{D,I},{R,a},{T,D}
,{x,P},{J,R},{K,g},{x,O},{f,V},{U,M},{p,U},{u,S},{O,k},{C,Z},{W,E},{u,X},{m,U
},{J,B},{y,e},{C,E},{r,f},{y,N},{n,c},{p,S},{s,X},{g,d},{f,z},{r,y},{P,N},{V,
p},{A,F},{l,m},{H,L},{k,b},{A,F}

{Z,Y,X,W,V,U,T,S,R,Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A,a,b,c,d,e,f,g,h,i,j,k,l,
m,n,o,p,q,r,s,t,u,v,w,x,y,z},{E,Y},{V,q},{K,k},{J,a},{z,S},{o,H},{e,f},{C,I},
{P,c},{L,G},{p,t},{b,W},{f,Z},{g,W},{Q,P},{v,a},{b,H},{B,z},{X,Q},{b,r},{X,d}
,{Q,x},{V,r},{U,I},{N,R},{a,P},{k,B},{S,a},{o,v},{J,V},{M,J},{H,a},{x,T},{P,t
},{k,W},{W,U},{W,x},{p,y},{n,Q},{M,l},{s,a},{r,c},{K,L},{C,Q},{t,o},{y,D},{v,
i},{s,e},{N,j},{j,x},{L,i},{o,E},{t,I},{Q,p},{t,w},{z,C},{r,y},{t,e},{b,r},{D
,V},{p,l},{d,o},{y,g},{K,R},{v,w},{G,y},{X,A},{H,Z},{t,S},{F,s},{f,a},{B,t},{
R,d},{f,R},{c,X},{k,W},{I,U},{Y,K},{R,i},{T,G},{z,X},{d,l},{h,e},{x,o},{x,m},
{v,Z},{h,E},{H,y},{f,m},{X,U},{H,Z},{H,L},{F,V},{V,f},{S,b},{D,m},{j,W},{D,h}
,{f,W},{t,S},{g,O},{y,B},{y,D},{j,E},{M,H},{q,i},{A,Y},{a,t},{T,v},{c,z},{p,A
},{Q,K},{b,A},{x,S},{J,I},{b,V},{W,B},{g,m},{N,J},{P,V},{M,O},{i,y},{U,h},{L,
w},{c,V},{U,V},{R,T},{g,p},{R,s},{Z,H},{R,K},{H,y},{V,O},{a,D},{q,Y},{W,D},{z
,G},{n,x},{a,W},{x,K},{X,I},{d,d},{U,P},{c,f},{M,u},{o,W},{L,D},{M,K},{W,J},{
Z,Y},{G,C},{P,a},{N,I},{I,F},{b,Q},{O,y},{y,D},{e,y},{Z,O},{o,O},{S,E},{J,D},
{g,G},{u,U},{n,x},{p,a},{Q,f},{V,a},{F,y},{K,G},{a,h},{g,h},{y,H},{w,h},{M,G}
,{V,s},{Q,w},{D,X},{w,B},{t,i},{c,s},{P,P},{H,h},{b,Q},{E,Y},{X,U},{t,s},{Z,F
},{P,N},{A,G},{N,O},{V,a},{O,N},{b,O},{w,M},{o,p},{z,k},{j,A},{S,g},{t,C}
```

{Z,Y,X,W,V,U,T,S,R,Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A,a,b,c,d,e,f,g,h,i,j,k,l,
m,n,o,p,q,r,s,t,u,v,w,x,y,z},{X,u},{e,f},{T,x},{v,V},{E,G},{u,G},{R,G},{J,E},
{y,G},{e,L},{L,V},{s,Y},{n,A},{n,i},{A,t},{p,s},{c,U},{q,g},{G,X},{X,r},{z,R}
,{T,q},{K,K},{I,U},{e,v},{A,K},{G,O},{T,N},{Z,F},{W,h},{H,A},{z,V},{I,r},{n,t
},{l,M},{g,u},{M,I},{b,O},{r,S},{L,Q},{U,G},{I,p},{f,r},{A,M},{C,k},{X,b},{b,
Q},{r,S},{c,L},{A,y},{o,W},{Z,h},{L,d},{D,V},{i,X},{T,W},{A,R},{p,F},{i,w},{l
,J},{K,Y},{q,o},{w,Z},{P,l},{m,I},{b,p},{Z,v},{N,E},{H,w},{G,Q},{T,Y},{r,X},{
m,E},{C,Y},{o,X},{L,h},{h,V},{Y,z},{w,U},{e,u},{D,d},{P,Y},{Y,r},{o,w},{L,n},
{h,j},{J,g},{V,T},{v,T},{w,w},{I,c},{w,o},{F,v},{C,h},{I,f},{M,T},{Q,R},{X,k}
,{j,E},{u,g},{q,A},{Q,U},{x,q},{q,O},{g,K},{S,R},{d,b},{F,Y},{Y,R},{Q,Q},{i,G
},{u,E},{e,l},{m,y},{y,L},{n,n},{o,a},{v,s},{v,M},{K,a},{D,d},{M,i},{N,C},{R,
g},{W,W},{z,k},{T,A},{p,r},{d,o},{f,k},{t,y},{w,G},{X,b},{X,w},{W,b},{I,E},{R
,C},{X,Q},{M,o},{q,p},{X,P},{U,h},{K,N},{a,V},{z,D},{E,D},{H,I},{J,y},{B,a},{
G,o},{k,A},{m,l},{L,l},{l,P},{m,I},{d,V},{S,b},{q,U},{v,x},{t,o},{C,X},{j,v},
{b,a},{z,H},{E,e},{d,R},{g,z},{N,y},{d,R},{n,N},{A,F},{I,t},{M,u},{K,b},{C,U}
,{T,v},{q,w},{P,j},{q,X},{h,l},{t,k},{d,D},{P,c},{X,o},{l,T},{V,s},{S,S},{m,k
},{V,d},{S,w},{v,P},{T,Y},{G,V},{x,g},{Q,Y},{y,U},{h,I},{j,O},{H,t},{R,r},{R,
Y},{W,e},{h,j},{J,A},{E,m},{C,F},{j,g},{o,r},{z,C},{d,C},{Y,z},{v,F},{B,m},{f
,p},{O,m},{b,k},{J,E},{D,U},{Y,f},{o,c},{k,J},{r,A},{r,x},{V,C},{c,c},{P,H},{
u,R},{q,R},{f,n},{V,o},{T,f},{x,x},{E,c},{q,U},{R,d},{g,I},{V,W},{C,X},{n,v},
{u,a},{I,b},{R,x},{V,C},{d,B},{A,u},{P,n},{R,c},{F,G},{X,F},{L,S},{H,G},{y,R}
,{M,k},{y,M},{v,S},{T,p},{O,M},{L,K},{G,I},{O,j},{D,U},{n,W},{z,k},{K,K},{r,s
},{s,f},{y,P},{h,P},{W,H},{t,X},{f,E},{g,D},{B,o},{d,d},{K,X},{k,j},{S,O},{A,
v},{k,t},{D,v},{Q,M},{Z,w},{T,X},{Y,Y},{z,p},{v,a},{i,Z},{t,i},{V,P},{Z,D},{K
,L},{V,S},{f,B},{M,c},{b,p},{t,R},{r,H},{z,y},{Y,i},{O,i},{x,Q},{E,A},{H,i},{
R,y},{u,W},{B,n},{Q,s},{w,O},{K,p},{X,L},{F,s},{s,n},{H,C},{A,P},{L,B},{h,a},
{Z,r},{G,X},{l,I},{Y,N},{a,h},{A,Z},{z,M},{T,G},{y,o},{A,j},{W,J},{J,X},{E,F}
,{u,F},{O,u},{Q,j},{J,H},{w,a},{w,i},{k,T},{p,P},{b,F},{G,h},{k,z},{F,o},{H,P
},{w,v},{v,S},{R,q},{A,O},{a,t},{Y,q},{Y,i},{E,B},{O,Q},{A,e},{m,i},{I,j},{W,
N},{x,m},{m,H},{c,d},{H,P},{K,Z},{V,f},{O,V},{w,Z},{f,Z},{F,J},{q,d},{G,A},{B
,u},{x,O},{z,P},{V,m},{p,X},{n,F},{o,B},{j,Q},{b,O},{F,y},{o,H},{X,Y},{u,n},{
h,r},{L,R},{T,f},{a,c},{W,v},{S,t},{q,e},{Z,a},{c,U},{u,Z},{Y,w},{r,G},{d,F},
{K,P},{P,X},{j,x},{f,v},{T,a},{Q,v},{U,H},{g,h},{G,v},{p,Q},{D,u},{Z,x},{t,U}
,{X,H},{r,b},{y,K},{x,K},{x,o},{R,k},{X,R},{H,D},{v,Q},{v,v},{L,Q},{F,c},{l,C
},{K,u},{j,E},{X,i},{F,t},{O,p},{K,H},{x,T},{m,O},{W,T},{z,C},{Q,l},{H,x},{v,
i},{G,w},{d,E},{T,V},{g,K},{H,S},{M,J},{c,z},{w,Z},{l,h},{t,u},{v,e},{T,Q},{p
,r},{N,T},{h,i},{d,x},{O,I},{G,D},{w,z},{k,W},{k,h},{M,X},{m,c},{h,O},{G,c},{
v,s},{n,S},{f,r},{B,s},{j,H},{b,m},{I,C},{l,N},{u,i},{W,m},{K,z},{D,f},{G,s},
{u,u},{d,V},{L,c},{z,F},{u,g},{A,j},{Q,W},{M,A},{P,z},{w,F},{I,u},{l,e},{P,K}
,{A,K},{f,F},{S,w},{Z,C},{D,a},{x,w},{y,Q},{C,u},{F,R},{z,y},{r,q},{o,A},{R,C
},{F,I},{Q,h},{j,T},{e,G},{F,W},{N,j},{Z,E},{w,U},{R,Y},{J,j},{O,Q},{l,Z},{O,
r},{p,Q},{u,i},{O,T},{h,C},{e,c},{B,l},{e,S},{z,G},{f,z},{i,p},{U,U},{N,X},{h
,O},{b,V},{q,P},{k,q},{P,S},{Y,w},{V,F},{U,F},{W,D},{T,q},{P,J},{K,M},{R,c},{
Y,q},{q,d},{p,i},{J,Y},{t,F},{E,h},{u,d},{G,J},{K,B},{S,Q},{G,e},{t,c},{Y,z},
{l,Z},{l,j},{e,T},{S,j},{F,f},{l,J},{d,F},{P,h},{l,n},{a,m},{F,n},{E,P},{R,Q}
,{W,v},{J,A},{T,L},{r,c},{R,o},{X,r},{X,z},{P,m},{m,p},{X,F},{q,b},{p,b},{o,U
},{s,g},{D,i},{s,J},{m,m},{r,q},{S,C},{R,t},{I,F},{s,X},{Z,j},{F,a},{V,W},{M,
m},{t,e},{s,l},{m,Z},{N,o},{k,r},{p,D},{z,w},{n,u},{g,N},{w,p},{q,Z},{S,x},{v
,j},{u,C},{i,h},{f,T},{j,t},{C,Q},{X,M},{G,z},{H,q},{A,v},{C,V},{r,H},{b,K},{
j,l},{e,f},{F,k},{k,X},{B,Q},{B,Z},{C,b},{o,T},{O,r},{k,S},{C,J},{J,S},{n,K},
{A,i},{Y,e},{S,l},{m,a},{X,d},{T,w},{C,T},{i,Z},{j,i},{L,L},{O,G},{Y,b},{r,u}
,{U,g},{h,D},{q,y},{Y,T},{v,U},{Z,i},{n,P},{Y,O},{f,A},{J,E},{h,I},{X,A},{c,e
},{q,c},{o,d},{b,S},{g,X},{L,K},{d,F},{k,R},{h,z},{d,u},{s,j},{V,J},{N,X},{b,
J},{B,a},{Q,N},{N,k},{S,j},{W,d},{d,M},{D,K},{g,t},{h,d},{M,h},{g,Y},{n,w},{w
,C},{i,k},{F,O},{o,r},{n,B},{t,o},{J,s},{J,m},{w,j},{k,N},{w,n},{E,t},{j,a},{
x,s},{c,r},{r,h},{f,R},{O,z},{Y,l},{q,p},{q,z},{D,R},{U,N},{r,u},{n,n},{d,Y},
{m,c},{P,K},{B,U},{f,u},{t,B},{b,t},{x,R},{g,l},{B,w},{O,s},{t,z},{e,w},{L,W}
,{t,K},{m,x},{S,L},{S,t},{n,r},{b,O},{f,D},{H,x},{F,V},{n,x},{B,j},{c,r},{A,T
},{s,Y},{v,s},{l,j},{m,p},{r,I},{y,U},{w,g},{w,s},{o,Z},{y,b},{g,Y},{q,D},{C,

q},{J,U},{r,r},{u,f},{R,v},{l,Y},{q,R},{L,W},{Z,r},{j,q},{o,Y},{E,F},{f,x},{W
,g},{i,h},{l,P},{S,L},{P,Y},{L,v},{J,x},{G,a},{b,E},{z,f},{A,u},{z,o},{I,G},{
o,h},{H,t},{V,s},{v,e},{u,c},{M,h},{L,j},{Q,M},{J,Z},{F,n},{S,f},{Y,O},{i,R},
{G,H},{b,T},{Q,a},{Z,w},{y,b},{K,H},{t,S},{M,b},{T,C},{V,B},{W,F},{n,U},{K,J}
,{O,B},{B,F},{b,D},{o,j},{k,X},{i,m},{K,A},{b,J},{M,o},{d,m},{q,E},{Z,s},{M,Y
},{k,L},{o,f},{v,x},{C,y},{C,T},{N,Q},{C,d},{H,g},{F,s},{P,c},{c,L},{U,F},{s,
c},{G,b},{i,C},{D,L},{B,c},{T,M},{g,q},{c,H},{f,g},{O,H},{L,K},{y,Y},{N,e},{M
,K},{Z,W},{D,o},{m,v},{d,g},{D,l},{x,u},{J,J},{u,R},{w,X},{F,q},{z,T},{w,l},{
W,X},{X,O},{t,a},{F,z},{F,x},{S,L},{A,x},{j,I},{a,i},{V,S},{g,I},{c,c},{H,m},
{x,F},{G,t},{F,A},{B,a},{Z,G},{B,V},{u,y},{g,F},{Y,o},{K,x},{T,X},{K,m},{L,F}
,{O,R},{S,k},{u,z},{p,P},{R,k},{j,W},{f,W},{S,D},{w,q},{z,o},{h,a},{M,c},{B,c
},{i,H},{h,p},{v,p},{p,r},{B,o},{O,i},{u,a},{U,j},{Y,q},{t,J},{b,P},{I,y},{h,
c},{p,M},{G,B},{C,p},{i,A},{h,y},{n,L},{W,Y},{r,Q},{O,H},{f,K},{W,H},{A,G},{m
,D},{o,x},{N,W},{q,v},{R,L},{J,u},{V,N},{G,s},{p,q},{q,S},{N,i},{C,v},{R,y},{
p,k},{x,X},{o,x},{i,x},{K,x},{o,X},{a,N},{B,h},{j,B},{V,d},{d,r},{i,I},{x,R},
{s,v},{M,Y},{W,I},{i,j},{b,X},{D,l},{Y,t},{u,w},{u,d},{r,c},{J,X},{B,Y},{P,I}
,{J,U},{K,j},{X,d},{g,H},{M,v},{h,J},{z,n},{q,m},{e,m},{b,u},{G,s},{h,P},{K,j
},{J,k},{O,p},{i,z},{n,D},{u,f},{f,e},{S,h},{i,T},{J,W},{Q,j},{x,S},{Y,E},{F,
b},{Y,f},{A,p},{H,j},{K,s},{k,b},{t,K},{y,J},{X,q},{j,j},{e,Z},{A,L},{A,h},{O
,A},{t,B},{i,y},{v,A},{Z,w},{K,k},{D,r},{j,y},{Z,Y},{K,A},{O,s},{a,A},{m,h},{
Z,u},{y,e},{N,B},{x,s}

{Z,Y,X,W,V,U,T,S,R,Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A,a,b,c,d,e,f,g,h,i,j,k,l,
m,n,o,p,q,r,s,t,u,v,w,x,y,z}

Judges Output to Screen

```
1
1
3
1
3
2
3
3
3
1
1
1
8
7
1
1
52
```

12. Us Numbers

Program Name: Us.java Input File: NONE

Assume an **Us** number (as opposed to a self-number) is a positive integer that can be expressed as the sum of another positive integer N and the sum of the digits of N.

The first 10 Us numbers are:

```
2 = 1 + (1) digits of 1
4 = 2 + (2) digits of 2
6 = 3 + (3)
8 = 4 + (4)
10 = 5 + (5)
11 = 10 + (1 + 0) digits of 10
12 = 6 + (6)
13 = 11 + (1 + 1) digits of 11
14 = 7 + (7)
15 = 12 + (1 + 2) digits of 12
```

Write a program that prints out the 100^{th}, 200^{th}, 300^{th}, 400^{th}, 500^{th}, 1000^{th}, 2000^{th} , $10{,}000^{th}$ and $20{,}000^{th}$ Us numbers. One per line.

Input
NONE

Output
9 lines of output. Replace each <Description> below with the proper Us number.

The first line will be `100 115`

The second to the last line will be `10000 11087`

There is a single space between the first and second number on a line.

```
100 <100th Us Number>
200 <200th Us Number>
300 <300th Us Number>
400 <400th Us Number>
500 <500th Us Number>
1000 <1000th Us Number>
2000 <2000th Us Number>
10000 <10000th Us Number>
20000 <20000th Us Number>
```

Example Input File:
```
NONE
```

Example Output To Screen
```
NOT SHOWN
```

12. Us Numbers

Program Name: Us.java **Input File: NONE**

Judges Input File

NONE

Judges Output to Screen

```
100 115
200 226
300 337
400 448
500 559
1000 1113
2000 2221
10000 11087
20000 22172
```