acm High School Programming Contest @

2017 University of Virginia High School Programming Contest

Welcome to the 2017 University of Virginia High School Programming Contest. Before you start the contest, please be aware of the following notes:

Rules

1. There are twelve (12) problems in this packet, using letters A-L. These problems are *loosely* sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Balloon Color
A	Mario Polo	dark purple
B	Coins	yellow
C	Kirby Your Enthusiasm	orange
D	Sortle!	pink
E	Power-up Probability	light purple
F	Jmp Mario	dark green
G	donKEY Kong	light blue
H	High Scores!	red
I	Collatz Chain Chomp	dark blue
J	Samus Charge	light green
K	Pokeballs	silver
L	Toad Puzzle	gold

2. Solutions for problems submitted for judging are called runs. Each run will be judged.

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Yes	Your submission has been judged correct.
No - Wrong Answer	Your submission generated output that is not correct or is incomplete.
No - Output Format Error	Your submission's output is not in the correct format or is misspelled.
No - Excessive Output	Your submission generated output in addition to or instead of what is required.
No - Compilation Error	Your submission failed to compile.
No - Run-Time Error	Your submission experienced a run-time error.
No - Time Limit Exceeded	Your submission did not terminate within one minute.

3. A team's score is based on the number of problems they solve and penalty minutes, which reflect the amount of time and number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty minutes are issued equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty minutes are added for problems

that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty minutes.

4. This problem set contains sample input and output for each problem. However, the judges will test your submission against several other more complex datasets, which will not be revealed until after the contest. One challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive a "wrong answer" judgment, you should consider what other datasets you could design to further evaluate your program.

5. In the event that you think a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient; no clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still think there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for a particular input, please include that input data in the clarification request.

   You may not submit clarification requests asking for the correct output for inputs that you provide. Sample inputs may be useful in explaining the nature of a perceived ambiguity, e.g., "There is no statement about the desired order of outputs. Given the input: ..., would not both this: ... and this: ... be valid outputs?".

   If a clarification that is issued during the contest applies to all the teams, it will be broadcast to everybody.

6. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first.

   Do not request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, **you may have a runner ask the site judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.**

   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams. This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification. This includes submitting dozens of runs within a short time period (say, within a minute or two).

Your Programs

8. All solutions must read from standard input and write to standard output. In C this is `scanf()` / `printf()`, in C++ this is `cin` / `cout`, and in Java this is `System.in` / `System.out`. The judges will ignore all output sent to standard error (`cerr` in C++ or `System.err` in Java). You may wish to use standard error to output debugging information. From your workstation you may test your program with an input file by redirecting input from a file:

```
program < file.in
```

9. All lines of program input and output should end with a newline character (`\n`, `endl`, or `println()`).

10. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

11. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line.

12. Unless otherwise specified, all numbers in your output should begin with a '-' if negative, followed immediately by 1 or more decimal digits. If it is a real number, then the decimal point should be followed by as many decimal digits as can be printed. This means that for floating point values, use standard printing techniques (cout and System.out.println). Unless otherwise noted, the judging will check your programs with 10^{-3} accuracy, so only consider the sample output up until that point.

    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure you do not round or use a set precision unless otherwise specified in the problem statement.

13. If a problem specifies that an input is a floating point number, the input will be presented according to the rules stipulated above for output of real numbers, except that decimal points and the following digits may be omitted for numbers with no fractional component. Scientific notation will not be used in input sets unless a problem statement explicitly specifies it.

Good luck, and HAVE FUN!!!

A. Mario Polo

Luigi just came up with a new game. He closes his eyes and calls out "Mario". Mario then responds with "Polo", and Luigi attempts to tag him based on where he heard Mario's voice. If Luigi says nothing, then Mario should also say nothing.

Demonstrate to Mario how to play this game. Every time Luigi calls out "Mario", respond with "Polo". If Luigi says nothing (i.e., "Silence"), the Mario should say nothing (i.e., "Silence").

Input Format

The first line of input, a positive integer, will be the number of test cases.

Each test case is on one line and consists of the word "Mario" or the word "Silence".

Output Format

For each test case, output one line containing the word "Polo" or the word "Silence".

Sample Input

```
5
Mario
Silence
Mario
Silence
Mario
```

Sample Output

```
Polo
Silence
Polo
Silence
Polo
```

B. Coins

Big fashion shifts have happened in the motorcycle world and Wario's bike is way out of style! He's been traveling across worlds to collect a bunch of coins to upgrade his bike, but he doesn't know how to count! There are three kinds of coins he can collect: Donkey Kong bananas, Toad mushrooms, and Zelda rupees. Each has a separate exchange rate for bitcoin. Specifically:

- 1 banana for 13 bitcoins
- 1 mushroom for 5 bitcoins
- 1 rupee for 462 bitcoins

Given the (non-negative) amount of each type of currency Wario has collected, print out the total amount of bitcoins Wario has. For each case, the input will be a line containing the amount of bananas, mushrooms, and rupees Wario has collected (respectively) and the output should be the total amount of bitcoin.

Input Format

The first line of input, a positive integer, will be the number of test cases.

The following n lines will each have three non-negative integers, separated by a single space each: the number of bananas, mushrooms, and rupees, respectively, for this test case. All values input (for bananas, mushrooms, and rupees) will be less than or equal to 100.

Output Format

Each case will output one line of the format; that line will contain only the number of bitcoins for that case. All results will be less than 1,000,000.

Sample Input

```
3
12 0 4
1 0 2
2 2 1
```

Sample Output

```
2004
937
498
```

C. Kirby Your Enthusiasm

Kirby is having an identity crisis! You need to help him find out what he really is after a series of actions.

Kirby can copy the identity of any other character. However, once he has copied someone, Kirby cannot copy any other character until he reverts back to his "normal" Kirby state by *releasing* his identity. If he tries to copy a new character without releasing the already copied identity, the "copy" action does not happen.

For example, if Kirby copies Link and then tries to copy Samus, Kirby will still have Link's identity. But, if Kirby copies Link, releases (and, thus, is back to normal), *then* tries to copy Samus, he will have Samus' identity. Kirby can release an identity at any time. You may assume Kirby starts off in a his "normal" Kirby state.

Given a list of Kirby's actions (either copy or release), state Kirby's final identity. An invalid action (such as trying to copy a second identity when not in the normal "Kirby" state) has no effect.

Input Format

The input will begin with a single positive integer $1 \leq T \leq 10,000$ representing the number of test cases to follow.

Each test case begins with a line containing a positive integer $1 \leq N \leq 1,000$. N lines follow, each containing one of Kirby's actions. The actions can either be "release" or "copy X", where "X" is an alphanumeric string.

Output Format

Each line of output should consist of the final identity of the test case.

Sample Input

```
3
3
copy Link
release
copy Samus
7
release
copy Link
copy Mario
release
release
copy Bowser
copy Link
4
copy Link
copy Link
copy Link
release
```

Sample Output

```
Samus
Bowser
Kirby
```

D. Sortle!

Squirtle has many friends and wants to organize his friend list. (There is no Facebook in the Kanto region, so he must do it himself.) He wants to organize his friends by type and list them in alphabetical order. Help Squirtle do this by writing a program to process his friends list!

For example, suppose Squirtle has five friends:

```
Pikachu Electric
Magikarp Water
Jolteon Electric
Vaporeon Water
Electabuz Electric
```

Your program should then organize these by type and print them in alphabetical order:

```
Electabuz Electric
Jolteon Electric
Pikachu Electric
Magikarp Water
Vaporeon Water
```

The types are alphabetized *and* the Pokemon names are alphabetized within these categories.

Input Format

The input will begin with a single positive integer representing the number of test cases to file.

Each test cast begins with a single positive integer $1 \leq n \leq 10^5$ denoting the number of friends in Squirtle's list. Each of the following n lines consists of a string describing the name of Squirtle's friend, a space, and then a string describing the type of the Pokemon. All Pokemon names in a given test case are guaranteed to be unique.

Output Format

Each line of output should consist of the string "Case n:" (where "n" is the current case number) followed by a newline. The following lines should be the sorted version of Squirtle's friend list.

Sample Input

```
2
5
Pikachu Electric
Magikarp Water
Jolteon Electric
Vaporeon Water
Electabuz Electric
```

```
8
Cress Water
Valerie Fairy
Wallace Water
Blaine Fire
Roxanne Rock
Brock Rock
Malva Fire
Glacia Ice
```

Sample Output

```
Case 1:
Electabuz Electric
Jolteon Electric
Pikachu Electric
Magikarp Water
Vaporeon Water
Case 2:
Valerie Fairy
Blaine Fire
Malva Fire
Glacia Ice
Brock Rock
Roxanne Rock
Cress Water
Wallace Water
```

E. Power-up Probability

Mario is losing in an intense game of MarioKart and wants to know whether he has a chance to catch up with the other racers. Given a list of all the power-ups that will be given out over the course of the game, tell him which power-up he is most likely to get. The most common on the course is the one he is most likely to get.

Input Format

The first line of input, a positive integer, will be the number of test cases.

Each test case is on one line, and begins with a line containing a positive integer $1 \leq n \leq 1000$ which is followed by n powerups, space separated. Each powerups is a single alphanumeric word, with no spaces.

Output Format

Each line of output should consist of the name of the power-up Mario is most likely to receive for each test case. If there are more than one power-ups equally likely to be the one Mario gets next, print "Tie" instead of a power-up name.

Sample Input

```
2
5 shell star mushroom shell shell
8 banana bomb bomb thunderbolt banana shell mushroom pow
```

Sample Output

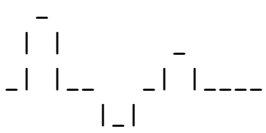
```
shell
Tie
```

F. Jmp Mario

Mario just got separated from Yoshi and needs to get back to him, but without Yoshi he can't tell when to jump! Given the vertical heights of every block in Mario's path, tell Mario at each step whether he should jump, double jump, or run (which includes falling) OR if Mario cannot traverse the landscape, output "not traversable". The landscape will be no more than 1000 blocks long.

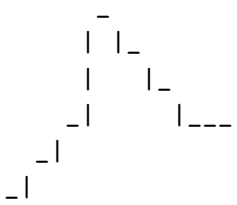
Mario has exactly three moves. He can run, which will move him one step forward; this will work properly if the landscape is the same level or if it would be a fall (he moves down if he "runs" into a pit, but is not damaged). If he needs to move up, he can jump or double jump, depending on whether he needs to increase altitude by one or two levels. He cannot triple jump, nor can he jump "over" an obstacle, such as a pit (he can possibly fall into the pit and them jump out).

The landscape will be given as a series of (at least 2) integers. For example, an input of `0 2 0 0 -1 0 1 0 0 0 0` would give the following landscape:



Mario starts on the first spot. Thus, in this landscape, Mario would want to double jump, run, run, run, jump, jump, run, run, run, run

But beware! There are some landscapes which Mario cannot traverse! For example, an input of `0 1 2 5 4 3 2 2 2` would give the following landscape:



However, Mario cannot triple jump and therefore this landscape is not traversable.

In all cases, Mario will make the minimum move required. A increase of a single level requires a jump, and not a double jump. Likewise, even though a jump will also move him forward, if there is no increase in landscape height, then a run is the correct move.

Input Format

The first line of input, a positive integer, will be the number of test cases.

Each of the next n lines will start with s, which is a positive integer that specified the number of landscape numbers to follow in this input case. There will then be $2 \leq s \leq 100$ integers on the line, space separated, denoting the height of the ground at each step. The heights may be positive or negative, and can range between -1,000 and 1,000, inclusive.

Output Format

For each test case, first output the string "Case n:" (where "n" is the current case number, starting from one) on its own line. The moves that Mario must take are then listed on the same line, space separated. Trailing whitespace on a given line is allowed.

If the landscape is not traversable, then "not traversable" should be output after the case number.

Trailing or leading whitespace (i.e., extra leading spaces/tabs or trailing spaces/tabs) on a given line is allowed.

Sample Input

```
4
11 0 2 0 0 -1 0 1 0 0 0 0
9 0 1 2 5 4 3 2 2 2
7 0 1 0 0 2 0 0
8 0 2 1 0 1 2 3 5
```

Sample Output

```
Case 1: double jump run run run jump jump run run run run
Case 2: not traversable
Case 3: jump run run double jump run run
Case 4: double jump run run jump jump jump double jump
```

G. donKEY Kong

Donkey Kong has intercepted a series of messages from the Tiki Tak Tribe describing where they hid his stash of bananas! Unfortunately, all the messages are in code and he only has the decoded version of one message. Write a program to help DK find where his stash is stashed!

DK took cryptography in jungle school, and happens to know that the cipher is a *Caesar Cipher*. This means encoding and decoding is done by substituting one letter for another, and the substitution is formed by "rotating" the alphabet by a certain offset. For example, if the offset was three letters, then the mapping is as below.

- A -> D
- B -> E
- C -> F
- ...
- T -> W
- ...
- W -> Z
- X -> A
- Y -> B
- Z -> C

Then, to encode the text "CAT", C becomes F, A becomes D, and T becomes W. The encoded word is then "FDW". To decode, we reverse the process.

As mentioned above, each letter was moved three forward in the alphabet. This is the *key*, which is the number to shift the letters forward. In the example above, each letter is shifted by 3, but we could use other keys. A key of 4, for example, would encode "CAT" as "GEX".

Donkey Kong has intercepted two messages, but he only has the decrypted form for the first one. Based on this, you will need to determine the meaning of the second message by computing the key from the first message. The key can be zero, but will not be larger than 25.

Input Format

The first line of input, a positive integer, will be the number of test cases.

Each test case will consist of three lines of text. The first line is a string of capital letters (A-Z) and underscores giving an encoded message. The second line is a string of capital letters and underscores containing the decoded form of the same message. These two strings will always be the exact same length. The final line of the test case contains an encoded string for a second message.

Messages will only contain letters and underscores; no (other) punctuation, no spaces, and no numbers. Underscores should not be encoded (meaning a underscore as input produces a underscore as output). Messages will always begin with a letter, not an underscore.

Output Format

Each line of output should consist of the string "Case n:" (where "n" is the current case number, starting from 1) followed by the decoded version of the second message.

Sample Input

```
3
FDW
CAT
KVSF
EDQDQDV_KDYH_EHHQ_VWROHQ
BANANAS_HAVE_BEEN_STOLEN
ZH_KLG_WKHP_ZHOO
NYXUOI_UYXQ_WECD_XYD_UXYG
DONKEY_KONG_MUST_NOT_KNOW
NU_WSCCOC_LKXKXKC
```

Sample Output

```
Case 1: HSPC
Case 2: WE_HID_THEM_WELL
Case 3: DK_MISSES_BANANAS
```

H. High Scores!

Miyamoto is very interested in the high scores for his video games. He thinks it might be possible that people with similar names perform better than those with different names. Given a list of high scoring players, he wants to determine the longest common prefix for each string in the list. It is possible for there to be *no* common prefix.

For example, consider the following list of high scorers:

```
Mario
Marina
Martin
Mary
Marth
Marcus
```

In this case, the longest common prefix for the entire list is "Mar". However, if we add just a few more names to the list, the common prefix changes:

```
Mario
Marina
Martin
Mary
Marth
Marcus
Matthew
Makefile
```

Now the common prefix is only "Ma".

Given a list of names, find the common prefix.

Input Format

The first line will be n, the number of test cases.

Each test case starts with a single integer $1 \leq k \leq 10^8$ denoting the number of names in the "top scorers" list. Each name will be up to 20 alphabetic characters (upper case or lower case) with no spaces.

Output Format

Each line of output should consist of the string "Case n:" (where "n" is the current case number, starting at 1) followed by the longest common prefix of the list of names. If there is no common prefix, print "No Common Prefix".

Sample Input

```
3
6
Mario
Marina
Martin
Mary
Marth
Marcus
8
Mario
Marina
Martin
Mary
Marth
Marcus
Matthew
Makefile
9
Mario
Marina
Martin
Mary
Marth
Marcus
Matthew
Makefile
HSPC
```

Sample Output

```
Mar
Ma
No Common Prefix
```

I. Collatz Chain Chomp

Tutankoopa is forming a new chain for his pet Chain Chomp and wants special numbered links. He is purchasing the links from the Collatz Chain Shop. This is a funny shop, where purchasing a link with one number means you must follow certain rules to determine the number for the next link you purchase.

If Tutankoopa purchases a link with number n, the next link he must purchase has the number:

$$\begin{cases} 3n+1 & \text{if n is odd} \\ n/2 & \text{if n is even} \end{cases}$$

The owner of the shop has verified that, if you start with any number less than 2^{61}, you must eventually purchase the chain link numbered 1. You must help Tutankoopa to determine which links he must buy if he starts with a particular chain link number.

For example, if he started with link number 5, Tutankoopa must purchase link number $3 \cdot 5 + 1 = 16$ next. After that, he must purchase $16/2 = 8$, then $8/2 = 4$, then $4/2 = 2$, and finally $2/2 = 1$.

Input Format

The first line of input, a positive integer, will be the number of test cases.

Each test case consists of a single integer, $1 < n \leq 2^{10}$, describing the number of the link that Tutankoopa purchases first. That number will be strictly greater than 1.

Output Format

Each output should consist of a space-separated list, all on one line, of the chain of links Tutankoopa must purchase. Trailing or leading whitespace (i.e., extra leading spaces/tabs or trailing spaces/tabs) on a given line is allowed. You should not print the number of the original purchase; only the remaining links.

Sample Input

```
3
5
64
3
```

Sample Output

```
16 8 4 2 1
32 16 8 4 2 1
10 5 16 8 4 2 1
```

J. Samus Charge

Space monsters are attacking! Luckily Samus is here to save the day with her laser that can instantly capture space monsters. However, her laser needs to charge up before it can capture the monsters (the greater the charge, the more effective the blast).

Space monsters move slowly, and you've acquired sensor data that accurately measures how many space monsters will be nearby at minute i. For example, the data might show that the list of space monsters present is $Z = [1, 5, 5, 1]$. On each minute i, exactly $Z[i]$ new space monsters are nearby. Note that the space monsters do not "stack" (i.e., exactly $Z[i]$ space monsters are nearby at minute i, regardless of how many were there in the last step). Also, the number of space monsters (in Z) can go down; that is, $Z[i]$ may be less than $Z[j]$ even if $j > i$.

Additionally, Samus' laser does not charge uniformly. You are given another list $C[i]$ that describes the amount of charge (in "space monster" units) that will be acquired if the gun is charged for any given minute. The charge in the gun *does* stack, so if there were already 3 charge units present, and 2 more are acquired in this minute, there are a total of 5 units to use in *this* minute. (That is, on minute i, Samus' gun first charges with $C[i]$ additional units and *then* Samus must choose whether to fire or not.)

If we call the charge at any given minute c, then up to c space monsters will be captured instantly if the gun is fired. Once fired, the gun's charge reduces to 0, and must begin to recharge again from zero. The gun starts at charge zero at the beginning as well.

For example, assume we are given $Z = [1, 5, 5, 1]$ and $C = [1, 3, 4, 2]$. We gain charge of 1 to use in minute 0, and we can use that to fire at 1 of the 1 space monsters available at step 0. If we did not fire on step 0, then we would have a total charge of 4 on step 1 (as we gain 3 charge) and could fire at 4 of the 5 available monsters. If we *did* fire on step 0, then we have only 3 charge to use in step 1.

Given these constraints, design an efficient algorithm that determines the most monsters that Samus can capture.

Input Format

The first line of input, a positive integer, will be the number of test cases.

Each test case will consist of three lines.

The first line will be a positive integer, and will list the number of steps in the functions Z and C; both sequences are the same length. There will not be more than 1,000 values in each sequence.

The second line will contain a sequence of numbers representing the amount of space monsters. The third line will contain a sequence of numbers representing the amount of charge. The values in the second and third lines will be unsigned integers between 0 and 1,000.

Output Format

The output will be one line per test case, and the only value on that line will be the maximum amount of space monsters captured (we are *not* printing "Case 1:" on the lines).

Sample Input

```
3
4
1 5 5 1
1 3 4 2
6
1 0 2 3 4 2
1 1 1 1 1 1
6
2 3 94 3 4 8
3 4 3 2 12 2
```

Sample Output

```
9
6
20
```

K. Pokeballs

Two aspiring Pokemon masters (Pokemon Master 1 and Pokemon Master 2) are playing a game. They take turns taking either 2, 3 or 5 Pokeballs from a sack containing n Pokeballs. They must take exactly 2, 3 or 5 Pokeballs from the sack on each of their turns. If a player is ever in a position where they can't take 2, 3, or 5 Pokeballs they lose. For example, if it is a players turn and there is only one Pokeball left then they lose.

The game's rules are as follows:

- Pokemon Master 1 will always go first, and the two players move in alternating turns.
- In a single move, a player can remove either 2, 3, or 5 Pokeballs from the sack.
- If a player is unable to make a move, that player loses the game and the other player is the winner. This can only occur when there are zero or one Pokeballs left.

Given the number of Pokeballs, find and print the name of the winner (i.e. First or Second) on a new line. Each player plays optimally, meaning they will not make a move that causes them to lose the game if some better, winning move exists.

Input Format

The first line of input, a positive integer, will be the number of test cases. Each test case will consist of exactly one positive integer, which is the number of Pokeballs in the sack.

Output Format

Each test case will produce one line, which is just "First" or "Second", depending on the winner of each game. Note that the case number of each input case is *not* printed.

Sample Input

```
7
1
2
3
4
5
6
7
```

Sample Output

```
Second
First
First
First
First
First
Second
```

L. Toad Puzzle

Toad has several items that might be useful for Mario and Luigi during their quest. However, he will only give items to those who are worthy; to determine worthiness, he will have them solve a *Theta Puzzle*.

The Theta Puzzle consists of a base with 6 positions at the vertices of a regular hexagon and another position at the center, connected as shown in the figure below. There are six tokens labeled A, B, C, D, E and F. A single move of the puzzle is to move a token to an adjacent empty position (along an allowed connection - the line segments in the diagram below). The idea of the puzzle is to start with an initial arrangement of tokens with the center empty and, by a sequence of moves, get to the configuration in the figure below.

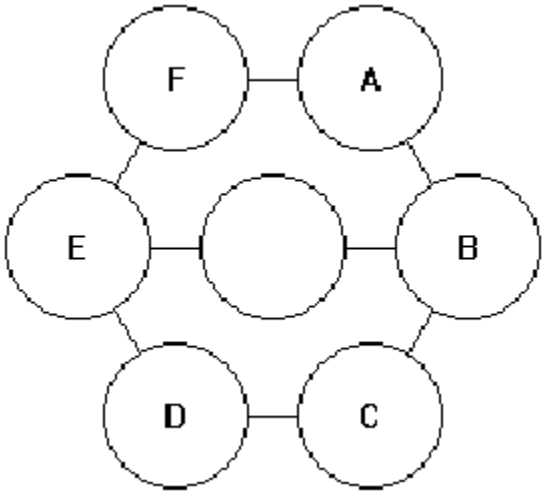


Figure 1: Theta Puzzle

An initial position for the puzzle is given by a permutation of the letters A through F. The first letter starts at A in the figure, the next at B and so on.

A sequence of moves is specified by listing the labels of tokens to be moved, in the order they are to be moved.

For example, to solve the puzzle FACDBE, use the moves BEFAB.

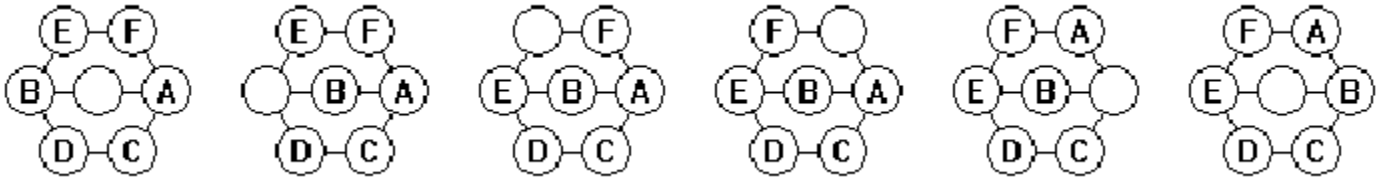


Figure 2: Example puzzle

Note: Not all starting permutations can be solved.

Write a program which, given an initial permutation, either finds number of moves in the the shortest sequence to solve the puzzle or determines that there is no solution.

Input Format

The first line of input contains a single integer P, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set is a single line that contains the data set number, followed by a space, followed by a permutation of the letters A through F giving the initial puzzle position.

Output Format

Each line of output should consist of the string "Case n:" (where "n" is the current case number, starting from 1) followed by the minimum number of moves required to solve the puzzle, or 'NO SOLUTION' if there is no solution.

Sample Input

```
12
1 FACDBE
2 ABCDEF
3 ADCEFB
4 ADCEBF
5 FEDCBA
6 FEDCAB
7 ECBFAD
8 ECBFDA
9 DCEBFA
10 DCEBAF
11 CBEADF
12 BDEAFC
```

Sample Output

```
1 5
2 0
3 19
4 NO SOLUTION
5 29
6 NO SOLUTION
7 19
8 NO SOLUTION
9 13
10 NO SOLUTION
11 21
12 16
```