A. Beat the Devil

Author: Sean Falconer, March 22, 2005

Anyone that has played cards before, is most likely familiar with some type of solitaire game. In this problem, given a description for a deck of cards, you will need to determine whether the player can win a type of solitaire called "Beat the Devil".

Beat the Devil is a very straight-forward solitaire game, where the goal is to place all the cards from the deck into various piles. The game begins with a player holding the entire deck of cards. The player takes the top card off the deck, and places it on the table, creating pile one. The player then takes the next card from the deck, and places it down beside the first card, creating a second pile. If both cards match, meaning their number or card type match (i.e. Ace of Hearts matches any other Ace, Jack of Spades matches any other Jack, 2 of Clubs matches any other 2, etc.), then take the next two cards from the deck; the first card removed goes on pile one, and the second card goes on pile two. If both cards did not match, then take the third card from the deck and create a third pile beside the second pile. Again, if this card matches any of the cards on top of any existing piles, take the next two cards from the deck and place them on top of the matching cards. You can make a maximum of 8 piles, after this, you continue placing cards as long as there are matches and you have cards. If you still have cards and there are no more matches, then you lost, if you run out of cards, then you have won.

To make this even clearer, let's play a sample game. Consider the following 15 cards making up the top part of the deck:

TS QH 9D 4S 8D KH JS KC JH 9H 2S 5S 7D 6H TH

where the first number or letter represents the card number or type (2 through 9, T = Ten, J = Jack, Q = Queen, K = King, A = Ace), and the last letter represents the suit (S = Spades, H = Hearts, D = Diamonds, and C = Clubs). We take the top card from the deck, 10 of Spades and create the first pile. Then we place the Queen of Hearts in the second pile. Since we do not have a match, we continue by placing the 9 of Diamonds in the third pile. Again, we still do not have a match, so we place the 4 of Spades in the fourth pile. We continue by placing the 8 of Diamonds in pile 5, King of Hearts in pile 6, Jack of Spades in pile 7, and finally the King of Clubs in pile 8 (see Figure 1).

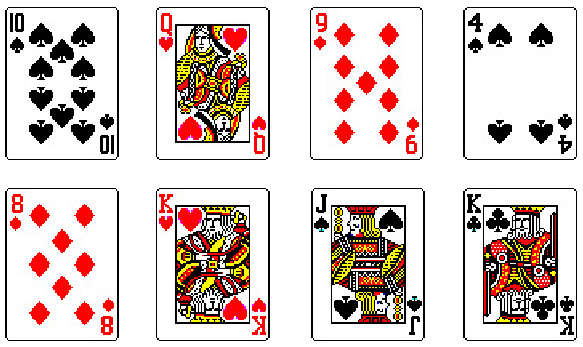

Figure 1: Example of 8 piles from Beat the Devil

We ended up creating all 8 piles with the first 8 cards, since we had no matches. However, with the 8th card, King of Clubs, we now have a match with pile 6 which has the King of Hearts on top. Therefore, we place the Jack of Hearts onto the top of pile 6, and the 9 of Hearts onto the top of pile 8. With these new cards, we now have two matches, the Jack of Hearts on pile 6 with the Jack of Spades on pile 7, and the 9 of Diamonds on pile 3 with the 9 of Hearts on pile 8. It does not matter which match you process first, let's choose pile 3 and 8 first. Thus, the 2 of Spades goes on the top of pile 3, and the 5 of Spades goes on top of pile 8. Finally, the 7 of Diamonds goes to pile 6 and the 6 of Hearts goes to pile 7. We now have no matches left on the piles, but we have the 10 of Hearts left in our hand, which means we have lost.

Although the example only uses 15 cards, all games are played with a deck of 52 cards, however, some cards may not be removed from the deck in a losing scenario.

Input

Input begins with an integer T representing the number of test cases. Each test case consists of 52 cards given in the format from the above example, that is, the card number or type, followed by the suit.

Output

For each test case, produce on a separate line the answer "Yes" if you can Beat the Devil, and "No, eternal damnation is inevitable" if you cannot.

Sample Input

```
2
TS QH 9D 4S 8D KH JS KC JH 9H 2S 5S 7D 6H TH 2D 3H AC 4H 7S 3D 9S 5D 2H 7H
QC 8C 3S 5H QD KS 4D 5C 9C 3C 7C 6S QS KD 6D 6C 8S 4C AD AH 8H JD TC TD AS
JC 2C
3C 5D 8H 5S TS 8S TC 3D 4S 4H 2C JH 4C TD AD 7H AH KC 6D 7C 9D 2H JS 6H 9S QS
KH KS 7S 7D JC KD 2S 3H QC 3S TH JD 5C 6S AC 6C QD QH 8D 5H 4D 2D 9C 8C AS 9H
```

Sample Output

```
No, eternal damnation is inevitable
Yes
```

B. Part of speech tagging

Author: Sean Falconer, March 22, 2005

Many natural language processing (NLP) applications begin by annotating the text with part of speech (POS) tagging. POS tagging is concerned with marking certain words or phrases with an appropriate label, such as noun or verb phrase. Many complicated NLP tasks such as Information Extraction begin with this phase in order to simplify the analysis of the text. There are many different ways to approach POS tagging, such as Hidden Markov models, rule-based methods, maximum likelihood, n-grams, etc., and many POS taggers have performance levels with over 95% accuracy.

In this problem, you must implement a simplified POS tagger using key-word dictionaries. You will read in dictionaries of noun phrases, verb phrases, adjectives, adverbs, and pronouns. Then, you will read in a series of plain-text messages, in which you will need to tag. Tags are indicated as follows: noun phrase = \NP, verb phrase = \VP, adjectives = \AJ, adverbs = \AV, and pronouns = \PN. For a given word in the plain-text, the corresponding tag for that word must come directly after it in your output. Words that cannot be matched, should be left as is. Word matches are case insensitive.

Consider the following example, where the noun phrases consist of the set {Ivan, princess, world, town, home}, verb phrases = {met, left, see, arrived}, adjectives = {beautiful, small}, adverb = {very}, and pronouns = {he, she}. The original plain-text is.

```
Ivan left home to see the world.  He arrived
in a small town, and met a very beautiful princess.
```

This would be tagged as:

```
Ivan\NP left\VP home\NP to see\VP the world\NP.  He\PN arrived\VP
in a small\AJ town\NP and met\VP a very\AV beautiful\AJ princess\NP.
```

Input

Input begins with the description of the dictionaries. The first line consists of 5 integers, NP, VP, AJ, AV, and PN ($0 \leq NP \leq VP \leq AJ \leq AV \leq PN \leq 100$). After this, follows NP lines consisting of the known noun phrases, then VP lines consisting of the known verb phrases, A lines consisting of the known adjectives, AV lines consisting of the known adverbs, and finally, PN lines consisting of the known pronouns. No dictionary words appear twice, all dictionary entries are single words, and all letters are lowercase.

After the dictionary words are listed, there is an integer T indicating how many test cases follow. Each test case consists of the plain-text that you must tag. The text consists of only words, spaces, end of lines, and periods. The end of one test case is indicated by a 1 on its own line.

Output

For each test case, output the annotated text. The text must be formatted the same as it was read in (i.e. spacing, punctuation, and end of lines are still in their appropriate place). Each test case must be separated by a blank line.

Sample Input

```
8 5 3 1 2
sean
home
ivan
princess
movie
yesterday
world
town
met
left
saw
arrived
see
beautiful
interesting
small
very
he
she
2
Ivan left home to see the world. He arrived
in a small town and met a very beautiful princess.
1
Sean saw an interesting movie yesterday.
1
```

Sample Output

```
Ivan\NP left\VP home\NP to see\VP the world\NP. He\PN arrived\VP
in a small\AJ town\NP and met\VP a very\AV beautiful\AJ princess\NP.

Sean\NP saw\VP an interesting\AJ movie\NP yesterday\NP.
```

C. Factorials and exponents

Author: Sean Falconer, March 22, 2005

The description of this problem is very short and simple. Given two numbers, n and m, find the largest d such that:

$$n! \geq m^d.$$

Input

Input begins with an integer T, representing the number of test cases to process. The following T lines contain two numbers, an integer $0 \leq n \leq 50000$ and a floating point number $0 \leq m \leq 50000$.

NOTE: Keep in mind that 50000! is a really really really big number.

Output

For each test case, output the integer d, such that $n! \geq m^d$. If d is equal to infinity, output "infinity" (without the quotes) as your answer.

Sample Input

```
5
10 2
2 10
10000 1
50 22.8893
50000 2
```

Sample Output

```
21
0
infinity
47
708356
```

D. Can you win the $\partial\nabla\triangle\Lambda\Theta\zeta$ game?

Author: Sean Falconer, March 22, 2005

A recent discovery on planet Terah is prime numbers. They're so fascinated by primes that all games played on Terah now involve prime numbers. Before the Terahlings (a person born on Terah) discovered primes, they were fascinated by different types of shapes. They had a game called $\partial\nabla\triangle\Lambda\Theta\Psi$, which, in English, roughly translates to "octagon rotations". The game involved six octagons, three per row. Each edge of the octagon was colored with a different color, and the goal of the game was to rotate the octagons such that adjacent octagons had matching colors.

The Terahlings have modified this game to incorporate their passion of prime numbers. The game is still played in a similar fashion, but instead of each edge having different colors, now each edge has a number associated with it. The sum of adjacent edges between octagons must sum to a prime number. The game is now called $\partial\nabla\triangle\Lambda\Theta\zeta$, which translates to "octagon rotations with primes" (sorry, the Terahlings are not very creative).

Consider Figure 2 below. On the left is the starting configuration of the game. We see that the 6 on the right side of the first octagon is lined up with the 2 on the left side of the second octagon, $6 + 2 = 8$, therefore, we need to rotate either the first or second octagon to create a prime number here. If we rotate the first octagon one step in the clockwise direction, we see that now the 1 lines up with the 2, and the sum is 3, which is prime. Also, the bottom number on the first octagon plus the top number of the fourth octagon sum to 3 $(1 + 2)$. If we check the rest of the octagons, we see that their adjacent edge sums all make prime numbers, therefore, we have won.

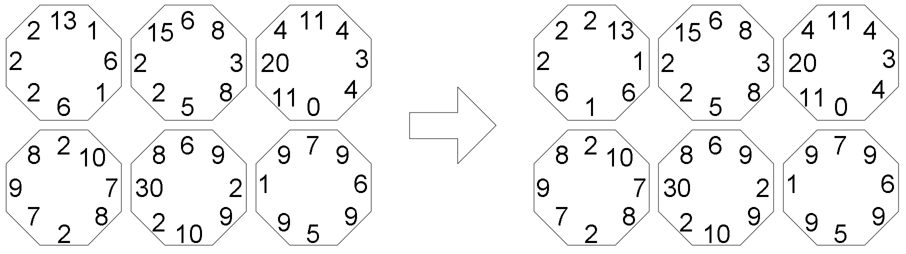

Figure 2: Example of $\partial\nabla\triangle\Lambda\Theta\zeta$ game.

You must read in an initial state of the octagons, and then find a winning configuration for the game if possible.

Input

The input begins with an single integer T, representing the number of test cases. Each test case consists of 6 lines, where each line has 8 integers, representing the integers on each side of the octagon beginning with the top number and going in a clockwise direction. The first 3 octagon descriptions represent the first row of octagons and the next 3 octagon descriptions represent the second row.

Since the Terahlings only recently discovered prime numbers, the largest prime they know of is 41, therefore, no two numbers will sum to an integer larger than 41.

Output

If it is possible to rotate the octagons such that you can win the game, then the output will consists of 6 lines. Each line describes the final configuration of the octagon, such that the octagons are in a winning state. The description of each octagon must begin with the number located in the top position, and going in a clockwise direction. The octagons should be listed in the same order as the input, that is the first 3 octagons correspond to the first row and the next 3 correspond to the second row.

If it is not possible to win the game, output "Cannot win" (without the quotes).

The output between each test case must be separated by a blank line.

Sample Input

```
2
13 1 6 1 6 2 2 2
6 8 3 8 5 2 2 15
11 4 3 4 0 11 20 4
2 10 7 8 2 7 9 8
6 9 2 9 10 2 30 8
7 9 6 9 5 9 1 9
2 2 4 6 8 10 8 4
4 6 12 4 8 10 10 2
0 0 2 2 6 6 8 8
2 4 6 8 10 12 14 16
16 14 12 10 8 6 4 2
8 8 8 8 8 8 8 8
```

Sample Output

```
2 13 1 6 1 6 2 2
6 8 3 8 5 2 2 15
11 4 3 4 0 11 20 4
2 10 7 8 2 7 9 8
6 9 2 9 10 2 30 8
7 9 6 9 5 9 1 9

Cannot win
```

E. Subsequences

Author: Sean Falconer, March 22, 2005

Most people that have been exposed to dynamic programming are familiar with the "Longest Common Subsequence" problem for two strings. More specifically, the problem involves two strings, a and b. The goal is to find the longest subsequence of letters that belongs to both strings. For example, consider $a =$ "abbaabbaa" and $b =$ "baaabaaabbaa". It is easy to see that both a and b share some common subsequences, but which is the longest one? In this case, the longest common subsequence for both a and b is "bbaabbaa".

In this problem, you will be dealing with an extension to the conventional "Longest Common Subsequence" problem. You will be given three strings, and you must determine the longest common subsequence between all three strings.

Input

Input begins with an integer T representing the number of test cases. Each case consists of 3 lines, each line contains one string with at most 50 characters.

Output

For each test case, output one line consisting of first, the length of the longest common subsequence, then a single space and then the corresponding subsequence.

If there is more than one longest common subsequence, any will do.

Sample Input

```
1
abbaabbaa
baaabaaabbaa
abba
```

Sample Output

```
4 abba
```

F. Building teams

Author: Sean Falconer, March 22, 2005

The St. Stephen elementary school is creating an intermural soccer league for students to play during lunch breaks. The teachers have decided that this is the best way to keep the kids out of trouble, however, they have a problem. The elementary school consists of several cliques (i.e. groups of kids that all hang out with each other), each clique dislikes certain other cliques, for example, the skateboarding clique may dislike everyone in the jock clique. The teachers do not like this situation, but they are scared of mixing two people from opposing cliques on the same soccer team because of all the problems with school violence lately. Instead, they decided to keep opposing cliques on different teams. Neutral cliques, that is, groups that a given clique does not dislike, can be part of the same team. Also, a clique's relationship with another clique is symmetric, that is, if clique 1 dislikes clique 2, then clique 2 also dislikes clique 1.

It is your job to figure out the minimum number of teams that are necessary in order to guarantee that no two opposing cliques are on the same team. Also, it is important to note that St. Stephen elementary school only has enough field space for a maximum of 4 teams, therefore, if you can't divide the students into at most 4 teams, then the teachers would like to know, so that they can think of a different noon hour activity for students to participate in.

Input

Input begins with an integer T, representing the number of test cases to process. Each test case begins with two numbers, $1 \leq N \leq 30$ and $1 \leq M \leq 1000$. N is the number of cliques in this example, and M is the number of students. The next N lines begins with a single word (only lower and upper case letters), representing a clique, after this follows a single space then an integer D, representing the number of cliques the current clique dislikes. After D, follows D single words, separated by a single space, each representing the names of the cliques the current clique dislikes. Finally, there are M lines, one for each student in the input. Each line consists of two words, $NAME$ and $CLIQUE$, where $NAME$ is the name of the student, and $CLIQUE$ is the name of the clique this student belongs to.

Every clique is guaranteed to contain at least one student.

Output

For each test case, print one line. This line will begin with "Case i: ", where i is replaced by the current input number starting at 1. Following this will be either a number "$TEAMS$ team(s)", representing the minimum number of teams needed to ensure no team has members from opposing cliques, or the statement "Need more teams", meaning more than 4 teams are needed.

Sample Input

```
2
3 6
jocks 1 skateboarders
cheerleaders 0
skateboarders 0
henry jocks
joe skateboarders
andrew jocks
shirley cheerleaders
theresa skateboarders
sean cheerleaders
4 10
A 2 B C
B 1 C
C 1 A
D 0
hamish A
seamus B
harry A
ron A
aaron B
hermonie C
george D
percey D
ender B
bean B
```

Sample Output

```
Case 1: 2 team(s)
Case 2: 3 team(s)
```

G. MTV slavery

Author: Sean Falconer, March 22, 2005

Canada is afraid of a US attack, so our government has started using the Canadian air force to monitor the sky. At any given time, there are N jets at various locations around Canada. Due to the size of the US military, it is probable that they may attack at many strategic locations around Canada.

The Canadian government would like your help in determining which planes to send to which locations of attack, such that each location has one plane and the average distance travelled to all locations is minimal.

Input

Input begins with an integer T representing the number of test cases that follow. Each test case begins with a integers N and M, $0 \leq N \leq 50$ and $0 \leq M \leq 50$. N represents the number of in air planes at the time of attack and M represents the number of US attack points. The next M lines contain N integers, where each line represents an attack point, and the N integers represent the time it takes for each of the N planes to reach the corresponding attack point.

Output

Output for each test case consists of one line, giving the average distance travelled by the planes going to the M locations (rounded to 2 decimal places). If no solution exists, produce one line stating "Not enough planes, we're going to be MTV slaves."

Sample Input

```
2
3 2
10 5 2
20 30 40
4 5
1 3 6 28
56 3 3 23
10 16 4 2
23 3 67 9
10 12 24 29
```

Sample Output

```
11.00
Not enough planes, we're going to be MTV slaves.
```