¿Fácil De Decir?

Un password seguro es algo delicado. Los usuarios prefieren passwords que sean fáciles de recordar (como `amigo`), pero este password puede ser inseguro. Algunos lugares usan un generador randómico de passwords (como `xvtpzyo`), pero los usuarios toman demasiado tiempo recordándolos y algunas veces lo escriben en una nota pegada en su computador. Una solución potencial es generar password “pronunciables” que sean relativamente seguros pero fáciles de recordar.

FnordCom está desarrollando un generador de passwords. Su trabajo en el departamento de control de calidad es probar el generador y asegurarse de que los passwords sean aceptables. Para ser aceptable, el password debe satisfacer estas tres reglas:

1. Debe contener al menos una vocal.
2. No debe tener tres vocales consecutivas o tres consonantes consecutivas.
3. No debe tener dos ocurrencias consecutivas de la misma letra, excepto por ‘ee’ o ‘oo’.

(Para el propósito de este problema, las vocales son 'a', 'e', 'i', 'o', y 'u'; todas las demás letras son consonantes.) Note que Estas reglas no son perfectas; habrán muchas palabras comunes/pronunciables que no son aceptables.

●

La entrada consiste en una o más potenciales passwords, uno por línea, seguidas por una línea conteniendo una palabra 'end' que señala el fin de la entrada. Cada password tiene como mínimo una y como máximo veinte letras de largo y esta formado por solo letras en minúscula. Por cada password, despliegue si es o no aceptable, usando el formato mostrado en el ejemplo de salida.

Ejemplo de entrada

```
a
tv
ptoui
```

```
bontres
zoggax
wiinq
eep
houctuh
end
```

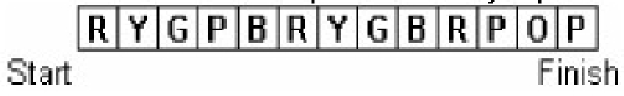
Ejemplo de salida

```
<a> is acceptable.
<tv> is not acceptable.
<ptoui> is not acceptable.
<bontres> is not acceptable.
<zoggax> is not acceptable.
<wiinq> is not acceptable.
<eep> is acceptable.
<houctuh> is acceptable.
```

Colorville

Un simple juego de niños usa un tablero que es una secuencia de cuadrados coloreados. Cada jugador tiene una pieza de juego. Los jugadores alternan turnos, sacando cartas que tienen cada una uno o dos cuadrados coloreados del mismo color. Los jugadores mueven su pieza hacia adelante en el tablero hacia el siguiente cuadrado que haga pareja con el color de la carta, o hacia adelante hasta el segundo cuadrado que haga pareja con el color de la carta que contiene dos cuadrados coloreados, o hacia adelante hasta el último cuadrado en el tablero si no hay un cuadrado con el que emparejar siguiendo la descripción anterior. Un jugador gana si su pieza está en el último cuadrado del tablero. Es posible que no exista ganador después de sacar todas las cartas.

En este problema los colores se representan las letras mayúsculas A-Z, a continuación se presenta un ejemplo.



Considere el siguiente deck de cartas: R, B, GG, Y, P, B, P, RR

Para 3 jugadores, el juego procede como sigue:

```
Jugador 1 saca R, se mueve al 1er cuadrado
Jugador 2 saca B, se mueve al 5to cuadrado
Jugador 3 saca GG, se mueve al 8vo cuadrado
Jugador 1 saca Y, se mueve al 2do cuadrado
Jugador 2 saca P, se mueve al 11vo cuadrado
Jugador 3 saca B, se mueve al 9no cuadrado
Jugador 1 saca P, se mueve al 4to cuadrado
Jugador 2 saca RR, Gano! (no hay R s al
frente de esta pieza así que va hasta el
último cuadrado).
```

Usando la misma tabla y el mismo deck de cartas, pero con 2 jugadores, el jugador 1 gana después de 7 cartas. Con 4 jugadores, no hay ganador después de utilizar todas las 8 cartas.

La entrada consiste en información de uno o más juegos. Cada juego comienza con una línea conteniendo el número de jugadores (1-4), el número de cuadrados en el tablero (1-79), y el número de cartas en el deck (1-200). Seguido por una línea de caracteres que representan los cuadrados coloreados del tablero. Seguidos por las cartas en el deck, uno el cada línea. Las Cartas pueden tener una letra o dos de las mismas letras. El final de la entrada esta señalado con una línea que tiene 0 para el número de jugadores – los otros valores son indiferentes.

Por cada juego, la salida es el jugador ganador y el número total de cartas usadas, o el número de cartas en el deck, como se muestra en el ejemplo de salida. Siempre use el plural "cards".

Ejemplo de entrada

```
2 13 8
RYGPBRYGBRPOP
R
B
GG
Y
P
B
P
RR
2 6 5
RYGRYB
R
YY
G
G
B
3 9 6
QQQQQQQQQ
Q
QQ
Q
Q
QQ
Q
0 6 0
```

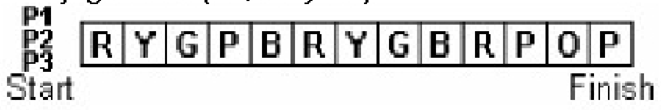
Ejemplo de salida

```
Player 1 won after 7 cards.
Player 2 won after 4 cards.
No player won after 6 cards.
```

El primer párrafo del planteamiento del problema describe claramente en que consiste el mismo.

Para quien no entendió en qué consiste este problema, bastará con ver mejor el ejemplo propuesto en el problema.

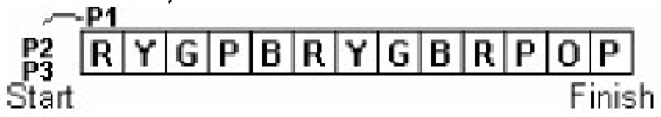
Este es nuestro tablero, y al inicio se encuentran nuestros tres jugadores (P1, P2 y P3)



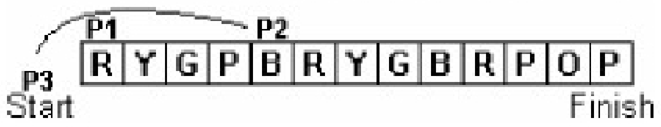
También sabemos que las cartas de nuestro deck saldrán en el siguiente orden: R, B, GG, Y, P, B, P, RR

Simulemos el juego de manera gráfica:

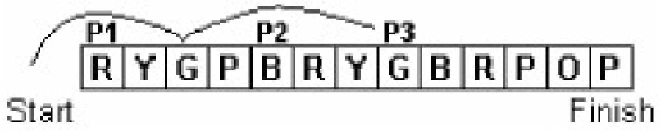
`Jugador 1 saca R, se mueve al 1er cuadrado`



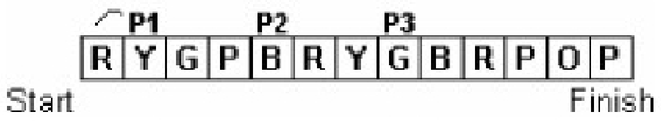
`Jugador 2 saca B, se mueve al 5to cuadrado`



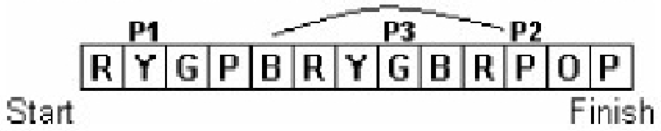
`Jugador 3 saca GG, se mueve al 8vo cuadrado`



`Jugador 1 saca Y, se mueve al 2do cuadrado`

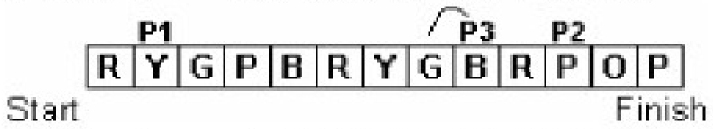


`Jugador 2 saca P, se mueve al 11vo cuadrado`

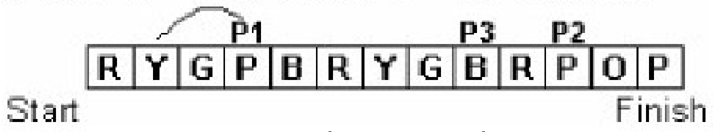


Jugador 3 saca B,   se mueve al 9no cuadrado

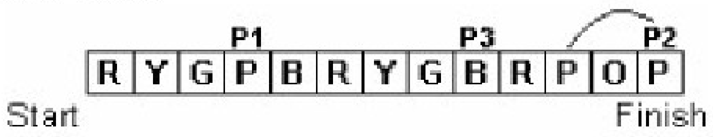
•



Jugador 1 saca P,   se mueve al 4to cuadrado



Jugador 2 saca RR, ¡Gano! (no mas hay R s al frente de esta pieza así que va hasta el último cuadrado).



Falling Leaves

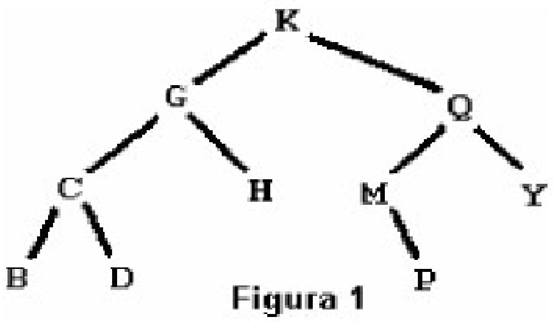


Figura 1

La Figura 1 muestra la representación gráfica de un árbol binario de letras. Lo familiarizados con los árboles binarios pueden saltarse la definición de árbol binario de letras, hojas de un árbol binario, y búsqueda en un árbol binario de letras, e ir directo al problema.

Definición.
Un **árbol binario de letras** puede ser una de dos cosas:
1. Puede estar vacía.
2. Puede tener un nodo raíz. Un nodo tiene una letra como dato y hace referencia a subárboles izquierdo y derecho. Los subárboles izquierdo y derecho son también árboles binarios de letras.

En la representación gráfica de un árbol binario de letras:
1. Un árbol vacío es omitido completamente.
2. Cada nodo esta indicado por
   Su dato letra,
   Un segmento de línea abajo a la izquierda hacia su subárbol izquierdo, si el subárbol izquierdo no está vacío,
   Un segmento de línea abajo a la derecha hacia su subárbol derecho, si el subárbol derecho no esta vacío.

Una **hoja** en un árbol binario es un nodo donde ambos subárboles están vacíos. En el ejemplo en la Figura 1, tiene cinco nodos con datos B, D, H, P, y Y.

El recorrido preorder de un árbol de letras satisface las propiedades:

1. Si el árbol esta vacío, entonces el recorrido preorder está vacío.
2. Si el árbol no esta vacío, entonces el recorrido preorder consiste en lo siguiente, en orden:
   El dato del nodo raíz,
   El recorrido preorder del subárbol izquierdo del nodo raíz,
   El recorrido preorder del subárbol derecho del nodo raíz.

El recorrido preorder del árbol de la Figura 1 es KGCBDHQMPY.

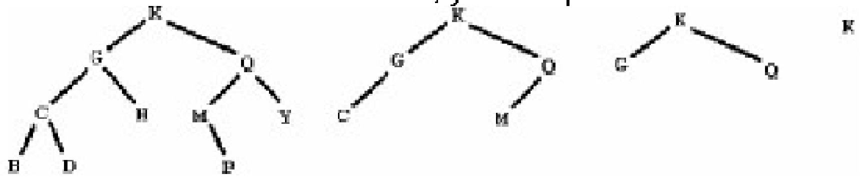
Un árbol como el de la Figura 1 es también un árbol binario de búsqueda de letras. Un árbol binario de búsqueda de letras es un árbol de letras en el cual cada nodo satisface:

1. Los datos raíz vienen después en el alfabeto que todos los datos en los nodos en el subárbol izquierdo.
2. Los datos raíz vienen antes en el alfabeto que todos los datos en los nodos en el subárbol derecho.

El problema:
Considere la siguiente secuencia de operaciones en un árbol binario de búsqueda de letras:

Borrar las hojas y listar los datos removidos
Repetir este proceso hasta que el árbol este vacío.

Empezando por el árbol de abajo a la izquierda, producimos la secuencia de árboles mostrados, y hasta que el árbol



este vacío removiendo las hojas de datos

BDHPY
CM
GQ
K

Tu problema es empezar con tales secuencias de líneas de hojas de un árbol binario de búsqueda de letras y desplegar el recorrido preorder del árbol.
La entrada contiene uno o más sets de datos. Cada set de datos es una secuencia de uno o más líneas con letras mayúsculas. Las líneas contienen las hojas removidas del árbol binario de búsqueda de la forma descrita anteriormente. Las letras en una línea están listados en orden alfabético. Los sets de datos están separados por una línea que contiene un asterisco ('*'). El último set de datos está seguido por un signo de dólar ('$'). No hay espacios en blanco ni líneas vacías en la entrada.

Por cada set de datos de entrada, hay un único árbol binario de búsqueda que puede ser producido con la secuencia de hojas. La salida es una línea que contiene solo el recorrido preorder del árbol, sin blancos.

Ejemplo de entrada

```
BDHPY
CM
GQ
K
*
AC
B
$
```

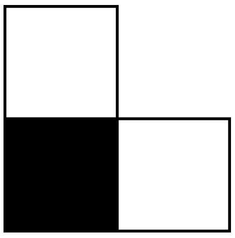
Ejemplo de salida

```
KGCBDHQMPY
BAC
```

●

D Piece it together

Tom has developed a special kind of puzzle: it involves a whole bunch of identical puzzle pieces. The pieces have the shape of three adjoint squares in an L-shape. The corner square is black, the two adjacent squares are white.



A puzzle piece

The puzzler is given a pattern of black and white squares in a rectangular grid. The challenge is to create that pattern using these pieces. The pieces can be rotated, but must not overlap.

Tom has already designed a few nice patterns, but he needs to find out if they can be constructed with the pieces at all. Rather than trying to test this for each pattern by hand, he wants to write a computer program to determine this for him. Can you help him?

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- one line with two integers n and m ($1 \leq n, m \leq 500$): the height and width of the grid containing the pattern, respectively.
- n lines, each containing m characters, denoting the grid. Each character is '`B`', '`W`', or '`.`', indicating a black, white or empty square respectively.

The grid contains at least one black or white square.

Output

Per test case:

- one line with either "`YES`" or "`NO`", indicating whether or not it is possible to construct the pattern with the puzzle pieces. You may assume that there is an infinite supply of pieces.

Sample in- and output

Input	Output

`2`
`3 4`
`BWW.`
`WWBW`
`..WB`
`3 3`
`W..`
`BW.`
`WBW`
	`YES`
	`NO`

E Please, go first

You are currently on a skiing trip with a group of friends. In general, it is going well: you enjoy the skiing during the day and, of course, the après-skiing during the night. However, there is one nuisance: the skiing lift. As always, it is too small, and can only serve one person every 5 seconds. To make matters worse, you and your friends generally don't arrive simultaneously at the lift, which means that you spend time waiting at the bottom of the mountain for the lift and at the top again for your friends.

The waiting at the top is especially inefficient. In fact, you realize that if your friends haven't arrived yet, you might as well let other people pass you in the queue. For you, it makes no difference, since otherwise you'd be waiting at the top. On the other hand, your actions might save them time if their friends have already arrived and are currently waiting for them at the top.

You are wondering how much time would be saved if everybody adopts this nice attitude. You have carefully observed the queue for a while and noticed which persons form groups of friends. Suppose someone lets another pass if doing this doesn't change his own total waiting time, but saves time for the other person. Do this over and over again until it can't be done anymore. How much time will this save, in total?

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- one line with an integer n $(1 \leq n \leq 25\,000)$: the number of people in the line for the lift.
- one line with n alphanumeric characters (uppercase and lowercase letters and numbers): the queue. The first person in this line corresponds to the person at the head of the queue. Equal characters correspond to persons from the same group of friends.

Output

Per test case:

- one line with an integer: the time saved, in seconds.

Sample in- and output

Input	Output
2 6 AABABB 10 Ab9AAb2bC2	15 45

Almost blank page

G Smoking gun

Andy: ”Billy the Kid fired first!”

Larry: ”No, I'm sure I heard the first shot coming from John!”

The arguments went back and forth during the trial after the big shoot-down, somewhere in the old wild west. Miraculously, everybody had survived (although there were serious injuries), but nobody could agree about the exact sequence of shots that had been fired. It was known that everybody had fired at most one shot, but everything had happened very fast. Determining the precise order of the shots was important for assigning guilt and penalties.

But then the sheriff, Willy the Wise, interrupted: ”Look, I've got a satellite image from the time of the shooting, showing exactly where everybody was located. As it turns out, Larry was located much closer to John than to Billy the Kid, while Andy was located just slightly closer to John than to Billy the Kid. Thus, because sound travels with a finite speed of 340 meters per second, Larry may have heard John's shot first, even if Billy the Kid fired first. But, although Andy was closer to John than to Billy the Kid, he heard Billy the Kid's shot first – so we know for a fact that Billy the Kid was the one who fired first!

Your task is to write a program to deduce the exact sequence of shots fired in situations like the above.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- one line with two integers n ($2 \leq n \leq 100$) and m ($1 \leq m \leq 1\,000$): the number of people involved and the number of observations.
- n lines with a string S, consisting of up to 20 lower and upper case letters, and two integers x and y ($0 \leq x, y \leq 1\,000\,000$): the unique identifier for a person and his/her position in Cartesian coordinates, in metres from the origin.
- m lines of the form “`S1 heard S2 firing before S3`”, where $S1$, $S2$ and $S3$ are identifiers among the people involved, and $S2 \neq S3$.

If a person was never mentioned as $S2$ or $S3$, then it can be assumed that this person never fired, and only acted as a witness. No two persons are located in the same position.

The test cases are constructed so that an error of less than 10^{-7} in one distance calculation will not affect the output.

Output

Per test case:

- one line with the ordering of the shooters that is compatible with all of the observations, formatted as the identifiers separated by single spaces.

If multiple distinct orderings are possible, output “`UNKNOWN`” instead. If no ordering is compatible with the observations, output “`IMPOSSIBLE`” instead.

Sample in- and output

Input	Output
`3`	`BillyTheKid John`
`4 2`	`IMPOSSIBLE`
`BillyTheKid 0 0`	`UNKNOWN`
`Andy 10 0`	
`John 19 0`	
`Larry 20 0`	
`Andy heard BillyTheKid firing before John`	
`Larry heard John firing before BillyTheKid`	
`2 2`	
`Andy 0 0`	
`Beate 0 1`	
`Andy heard Beate firing before Andy`	
`Beate heard Andy firing before Beate`	
`3 1`	
`Andy 0 0`	
`Beate 0 1`	
`Charles 1 3`	
`Beate heard Andy firing before Charles`	

H Tichu

Tichu is a card game played by four players. The players sit around a square table, and each player forms a team with the person sitting opposite him or her. The game is played with a standard deck of cards and four additional special cards. The basic rule of the game is as follows: the player who won the last trick can start a new trick with any legal combination of cards. Then, in turn, each next player can either pass or play the same combination of cards, but with a higher value. This continues until everyone passes, and at that point the player who played the last combination wins the trick and can start a new trick. The main goal is to get rid of all of your cards as soon as possible.

These basic rules make it a good tactic to combine the cards in such a way that they can be played in as few combinations as possible. For simplicity we consider here a slightly modified version of the game. We ignore the special cards, so that leaves a standard deck of 52 cards, ranging over the values `2` to `Ace` and over the suits `hearts`, `diamonds`, `clubs`, and `spades`. The suits are indicated by the lowercase letters `h`, `d`, `c`, and `s`, while the values are indicated in increasing order by `2`–`9`, `T`, `J`, `Q`, `K`, `A`.

The following list is a complete set of legal combinations:[2]

- any single card;
- a pair of cards of the same value;
- three cards of the same value;
- four cards of the same value;
- a full house, that is, three cards of the same value and two cards of another, same value, for example `444KK`;
- a straight of length at least five, that is, at least five cards of consecutive increasing values, for example `89TJQK`.

In this problem, your task is to determine the minimum number of combinations that your hand of 13 cards can be partitioned into.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- one line that describes your hand of 13 cards. The card descriptions are separated by single spaces. A card is described by two characters: the value followed by the suit. All cards in your hand are different.

Output

Per test case:

- one line containing an integer n: the minimum number of combinations that your hand can be partitioned into.

[2]Those who know the game of Tichu might have noticed that we removed consecutive pairs of cards as a valid combination.

- n lines that describe a minimal set of combinations of cards from your hand. Each line should contain the cards in one legal combination, in the same format as in the input. All cards from your hand must occur exactly once in one of the combinations. No specific ordering of the combinations or the cards within a combination is required.

Sample in- and output

Input	**Output**
`2`	`4`
`2h 3c 4d 5d 6s Th Qc Qs Ad Tc Ts 9c 9d`	`2h 3c 4d 5d 6s`
`2h 3h 4h 5h 6d 7s 8h 8d 8c 8s 9c Td Js`	`Th Ts Tc Qc Qs`
	`9d 9c`
	`Ad`
	`2`
	`2h 3h 4h 5h 6d 7s 8d 9c Td Js`
	`8h 8s 8c`

J Train delays

Last year, some of the judges tried to travel to NWERC'10 by train. This turned into a big disaster: on the way there, a fire in a control room caused huge delays, while on the return trip, trains in Bremen were delayed due to a terrorist threat in Hamburg. Of course, these huge delays caused other delays in the train schedule, so the big question was which trains to take: would it be better to take this slow regional train now, or wait for that intercity train, which has a big chance of being delayed?

This year, the judges have planned ahead and carefully analyzed the train schedule. They even kept track of how often trains were delayed and by how much. Now that they have all this information, they want to travel as quickly possible, minimizing the expected duration of the journey. Can you help them?

For each train connection, the judges know exactly what its scheduled departure time and duration are, as well as the probability that its arrival at the destination will be delayed. You may assume that the probabilities of delays are independent and that the judges can adapt their itinerary as they go, depending on any delays which they might already have incurred. Trains always depart on time, but may arrive late and the judges do not know whether a train's arrival will be delayed until they have boarded it. It takes judges no time to switch trains, so they can take a connecting train that departs at the same time as they arrive at a place.

The judges can choose the time of their initial departure as they wish and they want to minimize the expected duration[4] of their total trip.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- one line with the judges' place of origin and destination, these are different.
- one line with an integer n ($1 \leq n \leq 1\,000$): the number of train connections.
- n lines, each describing a train connection:
  - the origin and destination of this connection, these are different.
  - an integer m ($0 \leq m \leq 59$), the departure time in minutes after each full hour.
  - an integer t ($1 \leq t \leq 300$), the standard journey time (assuming no delays).
  - an integer p ($0 \leq p \leq 100$), the probability of delays as a percentage.
  - an integer d ($1 \leq d \leq 120$), the maximum delay in minutes.

All place names are given as strings of upper and lower case alphabetical characters, of length at most 20. If a train is delayed, then the length of the delay will be a whole number of minutes, and will be uniformly distributed in the range $[1, d]$.

[4]Given a travel plan of which trains to take (depending on previous connections and delays), the expected trip duration E is defined to be the sum of the trip duration T_i for each itinerary i possibly taken multiplied by the chance p_i of that itinerary occurring: $E = \sum_i p_i \, T_i$.

Output

Per test case:

- one line with a floating point number: the minimum expected duration of the total trip in minutes.

This number should be accurate up to 10^{-6} relative or absolute precision. Output `IMPOSSIBLE` instead if the destination is not reachable.

Sample in- and output

Input

```
3
Hamburg Bremen
3
Hamburg Bremen 15 68 10 5
Hamburg Bremen 46 55 50 60
Bremen Frankfurt 14 226 10 120
Amsterdam Rotterdam
1
Amsterdam Utrecht 10 22 5 10
BremenVegesack Utrecht
9
BremenVegesack BremenHbf 15 10 0 1
BremenVegesack BremenHbf 45 10 0 1
BremenVegesack Leer 23 140 10 15
BremenHbf Osnabruck 44 51 60 70
Osnabruck Amersfoort 55 147 38 40
Amersfoort Utrecht 24 15 30 15
Amersfoort Utrecht 54 15 10 35
Leer Groningen 45 140 5 10
Groningen Amersfoort 46 96 10 20
```

Output

```
68.3
IMPOSSIBLE
305.0532857
```

Note in the first example that it is better to take the slower train from Hamburg to Bremen, since the fast train would give an expected travel time of 70.25 minutes.