Calculate

$$R := B^P \bmod M$$

for large values of B, P, and M using an efficient algorithm. (That's right, this problem has a time dependency !!!.)

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.

Three integer values (in the order B, P, M) will be read one number per line. B and P are integers in the range 0 to 2147483647 inclusive. M is an integer in the range 1 to 46340 inclusive.

Output

For each test, the result of the computation. A single integer on a line by itself.

Sample Input

```
3
18132
17

17
1765
3

2374859
3029382
36123
```

Sample Output

```
13
2
13195
```

My birthday is coming up and traditionally I'm serving pie. Not just one pie, no, I have a number N of them, of various tastes and of various sizes. F of my friends are coming to my party and each of them gets a piece of pie. This should be one piece of one pie, not several small pieces since that looks messy. This piece can be one whole pie though.

My friends are very annoying and if one of them gets a bigger piece than the others, they start complaining. Therefore all of them should get equally sized (but not necessarily equally shaped) pieces, even if this leads to some pie getting spoiled (which is better than spoiling the party). Of course, I want a piece of pie for myself too, and that piece should also be of the same size.

What is the largest possible piece size all of us can get? All the pies are cylindrical in shape and they all have the same height 1, but the radii of the pies can be different.

Input

One line with a positive integer: the number of test cases. Then for each test case:

- One line with two integers N and F with $1 \leq N, F \leq 10000$: the number of pies and the number of friends.
- One line with N integers r_i with $1 \leq r_i \leq 10000$: the radii of the pies.

Output

For each test case, output one line with the largest possible volume V such that me and my friends can all get a pie piece of size V. The answer should be given as a oating point number with an absolute error of at most 10^{-3}.

Sample Input

```
3
3 3
4 3 3
1 24
5
10 5
1 4 2 3 4 5 6 5 4 2
```

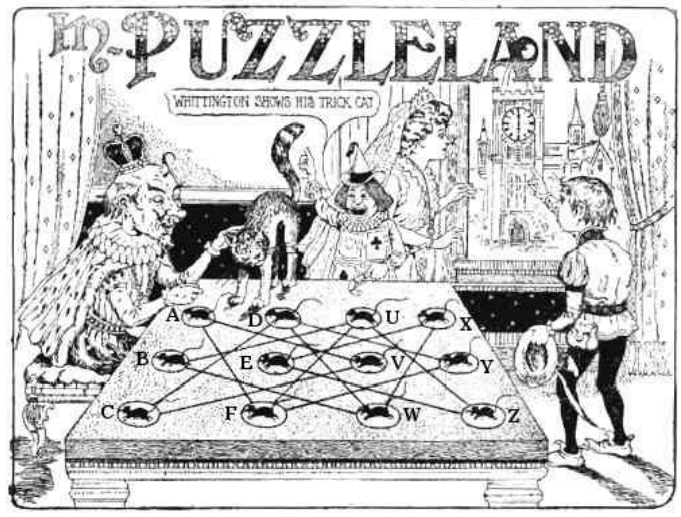
Sample Output

```
25.1327
3.1416
50.2655
```

Whittington is showing his trained cat in its surprising feat of going from A to Z, grabbing all the mice in his way while stepping on each circle just once.

You receive an arbitrary, undirected graph, where each node is identifed by a single uppercase letter. One vertex is the *source*, or starting point, and the other is the *target*, or end point.

Your job is to imitate the cat's ability, and identify a path that goes from the source to the target, visiting each node in the graph exactly once. If there is more than one valid path, choose the lexicographically lowest one.



Whittington's cat is very well trained

Input

Input starts with a positive integer T, that denotes the number of test cases.

There's a blank line at the beginning of each case. Then two integers are given in a single line: N and M, representing the number of nodes and the number of bi-directional edges in the graph, respectively. You can assume that there is at most one edge between any pair of nodes, and that each edge will be reported only once.

The next line will contain N distinct letters, separated by spaces, which are the identifers for all the nodes in the graph. The frst letter in this list will be the *source*, the last letter will be the *target*. All letters will be uppercase letters from the English alphabet.

Then M lines will be presented, describing the edges of the graph. Each of these lines contain two distinct letters, which describe two nodes that are connected by an edge.

$T \leq 60$; $2 \leq N \leq 15$

Output

For each test case, print the case number, followed by the sequence of letters that describe the path from the source to the target, visiting all nodes exactly once.

If a valid solution doesn't exist, print the word '`impossible`'.

Sample Input

```
3

12 14
A B C D E F U V W X Y Z
A F
A V
B U
B W
C D
C V
D Y
D W
E X
E Z
F U
F Y
U Z
W X

3 2
A B C
A B
A C

4 5
L N I K
L N
I L
I N
K N
K I
```

Sample Output

```
Case 1: AVCDYFUBWXEZ
Case 2: impossible
Case 3: LINK
```

A set of laboratory mice is being trained to escape a maze. The maze is made up of cells, and each cell is connected to some other cells. However, there are obstacles in the passage between cells and therefore there is a time penalty to overcome the passage Also, some passages allow mice to go one-way, but not the other way round.

Suppose that all mice are now trained and, when placed in an arbitrary cell in the maze, take a path that leads them to the exit cell in minimum time.

We are going to conduct the following experiment: a mouse is placed in each cell of the maze and a count-down timer is started. When the timer stops we count the number of mice out of the maze.

Problem

Write a program that, given a description of the maze and the time limit, predicts the number of mice that will exit the maze. Assume that there are no bottlenecks is the maze, i.e. that all cells have room for an arbitrary number of mice.

Input

The maze cells are numbered $1, 2, \ldots, N$, where N is the total number of cells. You can assume that $N \leq 100$.

The first three input lines contain N, the number of cells in the maze, E, the number of the exit cell, and the starting value T for the count-down timer (in some arbitrary time unit).

The fourth line contains the number M of connections in the maze, and is followed by M lines, each specifying a connection with three integer numbers: two cell numbers a and b (in the range $1, \ldots, N$) and the number of time units it

takes to travel from a to b.

Notice that each connection is one-way, i.e., the mice can't travel from b to a unless there is another line specifying that passage. Notice also that the time required to travel in each direction might be different.

Output

The output consists of a single line with the number of mice that reached the exit cell E in at most T time units.

Example

Input:

```
4
2
1
8
1 2 1
1 3 1
2 1 1
2 4 1
3 1 1
3 4 1
4 2 1
4 3 1
```

Output:

```
3
```

Your task is to write a program that reads a chess board configuration and answers if there's a king under attack (i.e. "*in check*"). A king is in check if it's in a square which is attacked by an oponnet's piece (i.e. it's in square which can be taken by an oponnet's piece in his next move).

White pieces will be represented by uppercase letters whereas black pieces will be represented by lowercase letters. White side will always be on the bottom of the board and black side will always be on the top of the board.

For those unfamiliar with chess, here are the movements of each piece:

Pawn (`p` or `P`): can only move straight ahead, one square at a time. But it takes pieces diagonally (and that's what concerns to you in this problem).

Knight (`n` or `N`) : have a special movement and it's the only piece that can jump over other pieces. The knight movement can be viewed as an "`L`". See the example bellow.

Bishop (`b` or `B`) : can move any number of squares diagonally (forward or backward).

Rook (`r` or `R`) : can move any number of squares vertically or horizontally (forward or backward).

Queen (`q` or `Q`) : can move any number of squares in any direction (diagonally, horizontally or vertically, forward or backward).

King (`k` or `K`) : can move one square at a time, in any direction (diagonally, horizontally or vertically, forward or backward).

Movements examples ('`*`' indicates where the piece can take another pieces):

```
Pawn        Rook        Bishop      Queen       King        Knight
........    ...*....    .......*    ...*...*    ........    ........
........    ...*....    *.....*.    *..*..*.    ........    ........
........    ...*....    .*...*..    .*.*.*..    ........    ..*.*...
........    ...*....    ..*.*...    ..***...    ..***...    .*...*..
...p....    ***r****    ...b....    ***q****    ..*k*...    ...n....
..*.*...    ...*....    ..*.*...    ..***...    ..***...    .*...*..
........    ...*....    .*...*..    .*.*.*..    ........    ..*.*...
........    ...*....    *.....*.    *..*..*.    ........    ........
```

Remember that the knight is the only piece that can jumper over other pieces. The pawn movement will depend on its side. If it's a black pawn, it can only move one square diagonally down the board. If it's a white pawn, it can only move one square diagonally up the board. The example above is a black pawn as it's a lowercase '`p`' (we say "*move*" meaning the squares where the pawn can move to when it takes another piece).

Input

There will be an arbitrary number of board configurations on the input. Each board will consist of 8 lines of 8 characters each. A '`.`' character will represent an empty square. Upper and lower case letters (as defined above) will represent the pieces. There will be no invalid characters (i.e. pieces) and there won't be a configuration where both kings are in check. You must read until you find an empty board (i.e. a board that is formed only of '`.`' characters) which should not be processed. There will be an empty line between each pair of board configurations. In all boards (except the last one which is empty) will appear both the white king and the black king (one, and only one of each).

Output

For each board configuration read you must output one of the following answers:

```
Game #d:  white king is in check.
Game #d:  black king is in check.
Game #d:  no king is in check.
```

Where d stands for the game number (starting from 1).

Sample Input

```
..k.....
ppp.pppp
........
.R...B..
........
........
PPPPPPPP
K.......

rnbqkbnr
pppppppp
........
........
........
........
PPPPPPPP
RNBQKBNR

rnbqk.nr
ppp..ppp
....p...
...p....
.bPP....
.....N..
PP..PPPP
RNBQKB.R

........
........
........
........
........
........
........
........
```

Sample Output

```
Game #1: black king is in check.
Game #2: no king is in check.
Game #3: white king is in check.
```

As you may know, balloons are handed out during ACM contests to teams as they solve problems. However, this sometimes presents logistical challenges. In particular, one contest hosting site maintains two rooms, A and B, each containing a supply of balloons. There are N teams attending the contest at that site, each sitting at a different location. Some are closer to room A, others are closer to room B, and others are equally distant. Given the number of balloons needed by each team and the distance from each team to room A, and to room B, what is the minimum total possible distance that must be traveled by all balloons as they are delivered to their respective teams, assuming they are allocated in an optimal fashion from rooms A and B? For the purposes of this problem, assume that all of the balloons are identical.

Input

There will be several test cases in the input. Each test case will begin with a line with three integers:

N A B

Where N is the number of teams ($1 \le N \le 1{,}000$), and A and B are the number of balloons in rooms A and B, respectively ($0 \le A,B \le 10{,}000$). On each of the next N lines there will be three integers, representing information for each team:

K DA DB

Where K is the total number of balloons that this team will need, DA is the distance of this team from room A, and DB is this team's distance from room B ($0 \le DA,DB \le 1{,}000$). You may assume that there are enough balloons – that is, $\Sigma(K's) \le A+B$. The input will end with a line with three 0s.

Output

For each test case, output a single integer, representing the minimum total distance that must be traveled to deliver all of the balloons. Count only the outbound trip, from room A or room B to the team. Don’t count the distance that a runner must travel to return to room A or room B. Print each integer on its own line with no spaces. Do not print any blank lines between answers.

Example

Input:
3 15 35
10 20 10
10 10 30
10 40 10
0 0 0
Output:
300

Kaunas University of Technology has bought a new light toggling system from one of the cheapest manufacturers in China. It consists of N lamps and M switches. Each switch has a subset of lights assigned to it, and when toggled, it changes the state of all the lights in the subset from on to off and vice versa. Also the system contains the main switch which is used to turn turn off all lights.

The authorities installed the switches at different locations in the univerisity. But one day the main switch went down. Now they are not able to turn off all lights by using the main switch. Unfortunately noone understands the Chineese documentation of the system, so we must wait for support from manufacturers. But we have good programmers, and we are interested in finding the minimal number of switches required to turn off all lights in the university. Initially, all lights are turned on.

Input

The first line of input contains the number of tests T ($T \leq 50$). Each test case is a set of lines. First line of each test case contains 2 positive integers N ($N \leq 15$) and M ($M \leq 100$) separated by a space character. Next M lines contain N integers K ($K_i \in \{1, 0\}$) separated by a space character (if the i-th integer is 1 then the i-th light is toggled by the switch).

Output

For each test case output a single line '`Case` T`:` N'. Where T is the test case number (starting from 1) and N is the minimal number of switches required. If it is impossible to turn off all lights N should be equal to '`IMPOSSIBLE`'.

Sample Input

```
2
2 2
0 1
1 0
3 2
1 0 1
1 1 0
```

Sample Output

```
Case 1: 2
Case 2: IMPOSSIBLE
```

You have a rectangular chocolate bar consisting of $n \times m$ single squares. You want to eat exactly k squares, so you may need to break the chocolate bar.

In one move you can break any single rectangular piece of chocolate in two rectangular pieces. You can break only by lines between squares: horizontally or vertically. The cost of breaking is equal to square of the break length.

For example, if you have a chocolate bar consisting of 2×3 unit squares then you can break it horizontally and get two 1×3 pieces (the cost of such breaking is $3^2 = 9$), or you can break it vertically in two ways and get two pieces: 2×1 and 2×2 (the cost of such breaking is $2^2 = 4$).

For several given values n, m and k find the minimum total cost of breaking. You can eat exactly k squares of chocolate if after all operations of breaking there is a set of rectangular pieces of chocolate with the total size equal to k squares. The remaining $n \cdot m - k$ squares are not necessarily form a single rectangular piece.

Input

The first line of the input contains a single integer t ($1 \le t \le 40910$) — the number of values n, m and k to process.

Each of the next t lines contains three integers n, m and k ($1 \le n, m \le 30, 1 \le k \le min(n \cdot m, 50)$) — the dimensions of the chocolate bar and the number of squares you want to eat respectively.

Output

For each n, m and k print the minimum total cost needed to break the chocolate bar, in order to make it possible to eat exactly k squares.

Examples

Input
4 2 2 1 2 2 3 2 2 2 2 2 4
Output
5 5 4 0

Note

In the first query of the sample one needs to perform two breaks:

- to split 2×2 bar into two pieces of 2×1 (cost is $2^2 = 4$),
- to split the resulting 2×1 into two 1×1 pieces (cost is $1^2 = 1$).

In the second query of the sample one wants to eat 3 unit squares. One can use exactly the same strategy as in the first query of the sample.