Perhaps you have heard of the legend of the Tower of Babylon. Nowadays many details of this tale have been forgotten. So now, in line with the educational nature of this contest, we will tell you the whole story:

> The babylonians had n types of blocks, and an unlimited supply of blocks of each type. Each type-i block was a rectangular solid with linear dimensions (x_i, y_i, z_i). A block could be reoriented so that any two of its three dimensions determined the dimensions of the base and the other dimension was the height.
>
> They wanted to construct the tallest tower possible by stacking blocks. The problem was that, in building a tower, one block could only be placed on top of another block as long as the two base dimensions of the upper block were both strictly smaller than the corresponding base dimensions of the lower block. This meant, for example, that blocks oriented to have equal-sized bases couldn't be stacked.

Your job is to write a program that determines the height of the tallest tower the babylonians can build with a given set of blocks.

Input

The input file will contain one or more test cases. The first line of each test case contains an integer n, representing the number of different blocks in the following data set. The maximum value for n is 30.

Each of the next n lines contains three integers representing the values x_i, y_i and z_i.

Input is terminated by a value of zero (`0`) for n.

Output

For each test case, print one line containing the case number (they are numbered sequentially starting from 1) and the height of the tallest possible tower in the format

'`Case` *case*`: maximum height =` *height*'

Sample Input

```
1
10 20 30
2
6 8 10
5 5 5
7
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
5
31 41 59
26 53 58
97 93 23
84 62 64
33 83 27
0
```

Sample Output

```
Case 1: maximum height = 40
Case 2: maximum height = 21
Case 3: maximum height = 28
Case 4: maximum height = 342
```

John Doe, a skilled pilot, enjoys traveling. While on vacation, he rents a small plane and starts visiting beautiful places. To save money, John must determine the shortest closed tour that connects his destinations. Each destination is represented by a point in the plane $p_i = < x_i, y_i >$. John uses the following strategy: he starts from the leftmost point, then he goes strictly left to right to the rightmost point, and then he goes strictly right back to the starting point. It is known that the points have distinct x-coordinates.

Write a program that, given a set of n points in the plane, computes the shortest closed tour that connects the points according to John's strategy.

Input

The program input is from a text file. Each data set in the file stands for a particular set of points. For each set of points the data set contains the number of points, and the point coordinates in ascending order of the x coordinate. White spaces can occur freely in input. The input data are correct.

Output

For each set of data, your program should print the result to the standard output from the beginning of a line. The tour length, a floating-point number with two fractional digits, represents the result.

Note: An input/output sample is in the table below. Here there are two data sets. The first one contains 3 points specified by their x and y coordinates. The second point, for example, has the x coordinate 2, and the y coordinate 3. The result for each data set is the tour length, (6.47 for the first data set in the given example).

Sample Input

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

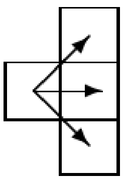
Sample Output

```
6.47
7.89
```

Problems that require minimum paths through some domain appear in many different areas of computer science. For example, one of the constraints in VLSI routing problems is minimizing wire length. The Traveling Salesperson Problem (TSP) — finding whether all the cities in a salesperson's route can be visited exactly once with a specified limit on travel time — is one of the canonical examples of an NP-complete problem; solutions appear to require an inordinate amount of time to generate, but are simple to check.

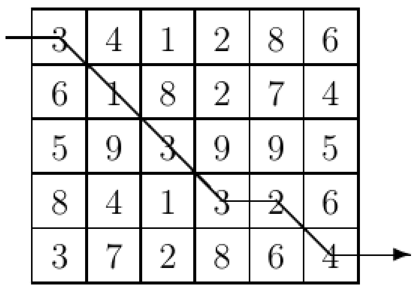
This problem deals with finding a minimal path through a grid of points while traveling only from left to right.

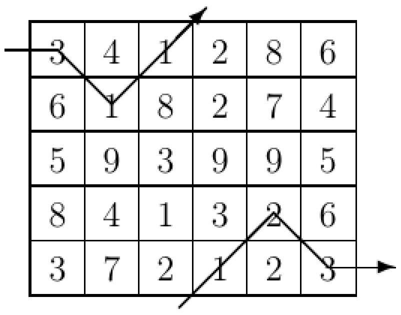
Given an $m \times n$ matrix of integers, you are to write a program that computes a path of minimal weight. A path starts anywhere in column 1 (the first column) and consists of a sequence of steps terminating in column n (the last column). A step consists of traveling from column i to column $i+1$ in an adjacent (horizontal or diagonal) row. The first and last rows (rows 1 and m) of a matrix are considered adjacent, i.e., the matrix "wraps" so that it represents a horizontal cylinder. Legal steps are illustrated on the right.



The *weight* of a path is the sum of the integers in each of the n cells of the matrix that are visited.

For example, two slightly different 5×6 matrices are shown below (the only difference is the numbers in the bottom row).





The minimal path is illustrated for each matrix. Note that the path for the matrix on the right takes advantage of the adjacency property of the first and last rows.

Input

The input consists of a sequence of matrix specifications. Each matrix specification consists of the row and column dimensions in that order on a line followed by $m \cdot n$ integers where m is the row dimension and n is the column dimension. The integers appear in the input in row major order, i.e., the first n integers constitute the first row of the matrix, the second n integers constitute the second row and so on. The integers on a line will be separated from other integers by one or more spaces. **Note:** integers are not restricted to being positive.

There will be one or more matrix specifications in an input file. Input is terminated by end-of-file.

For each specification the number of rows will be between 1 and 10 inclusive; the number of columns will be between 1 and 100 inclusive. No path's weight will exceed integer values representable using 30 bits.

Output

Two lines should be output for each matrix specification in the input file, the first line represents a minimal-weight path, and the second line is the cost of a minimal path. The path consists of a sequence of n integers (separated by one or more spaces) representing the rows that constitute the minimal path. If there is more than one path of minimal weight the path that is *lexicographically* smallest should be output.

Note: *Lexicographically* means the natural order on sequences induced by the order on their elements.

Sample Input

```
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 8 6 4
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 1 2 3
2 2
9 10 9 10
```

Sample Output

```
1 2 3 4 4 5
16
1 2 1 5 4 5
11
1 1
19
```

(If you smiled when you see the title, this problem is for you ^_^)

For those who don't know KTV, see: `http://en.wikipedia.org/wiki/Karaoke_box`

There is one very popular song called Jin Ge Jin Qu(). It is a mix of 37 songs, and is extremely long (11 minutes and 18 seconds) — I know that there are Jin Ge Jin Qu II and III, and some other unofficial versions. But in this problem please forget about them.

Why is it popular? Suppose you have only 15 seconds left (until your time is up), then you should select another song as soon as possible, because the KTV will not crudely stop a song before it ends (people will get frustrated if it does so!). If you select a 2-minute song, you actually get 105 extra seconds! ....and if you select Jin Ge Jin Qu, you'll get 663 extra seconds!!!

Now that you still have some time, but you'd like to make a plan now. You should stick to the following rules:

- Don't sing a song more than once (including Jin Ge Jin Qu).
- For each song of length t, either sing it for exactly t seconds, or don't sing it at all.
- When a song is finished, always immediately start a new song.

Your goal is simple: sing as many songs as possible, and leave KTV as late as possible (since we have rule 3, this also maximizes the total lengths of all songs we sing) when there are ties.

Input

The first line contains the number of test cases T ($T \leq 100$). Each test case begins with two positive integers n, t ($1 \leq n \leq 50$, $1 \leq t \leq 10^9$), the number of candidate songs (BESIDES Jin Ge Jin Qu) and the time left (in seconds). The next line contains n positive integers, the lengths of each song, in seconds. Each length will be less than 3 minutes — I know that most songs are longer than 3 minutes. But don't forget that we could manually "cut" the song after we feel satisfied, before the song ends. So here "length" actually means "length of the part that we want to sing".

It is guaranteed that the sum of lengths of all songs (including Jin Ge Jin Qu) will be strictly larger than t.

Output

For each test case, print the maximum number of songs (including Jin Ge Jin Qu), and the total lengths of songs that you'll sing.

Explanation:

In the first example, the best we can do is to sing the third song (80 seconds), then Jin Ge Jin Qu for another 678 seconds.

In the second example, we sing the first two (30+69=99 seconds). Then we still have one second left, so we can sing Jin Ge Jin Qu for extra 678 seconds. However, if we sing the first and third song instead (30+70=100 seconds), the time is already up (since we only have 100 seconds in total), so we can't sing Jin Ge Jin Qu anymore!

Sample Input

```
2
3 100
60 70 80
3 100
30 69 70
```

Sample Output

```
Case 1: 2 758
Case 2: 3 777
```

You are given the task to design a lighting system for a huge conference hall. After doing a lot of calculation and sketching, you have figured out the requirements for an energy-efficient design that can properly illuminate the entire hall. According to your design, you need lamps of n different power ratings. For some strange current regulation method, all the lamps need to be fed with the same amount of current. So, each category of lamp has a corresponding voltage rating. Now, you know the number of lamps and cost of every single unit of lamp for each category. But the problem is, you are to buy equivalent voltage sources for all the lamp categories. You can buy a single voltage source for each category (Each source is capable of supplying to infinite number of lamps of its voltage rating.) and complete the design. But the accounts section of your company soon figures out that they might be able to reduce the total system cost by eliminating some of the voltage sources and replacing the lamps of that category with higher rating lamps. Certainly you can never replace a lamp by a lower rating lamp as some portion of the hall might not be illuminated then. You are more concerned about money-saving than energy-saving. Find the minimum possible cost to design the system.

Input

Each case in the input begins with n ($1 \leq n \leq 1000$), denoting the number of categories. Each of the following n lines describes a category. A category is described by 4 integers - V ($1 \leq V \leq 132000$), the voltage rating, K ($1 \leq K \leq 1000$), the cost of a voltage source of this rating, C ($1 \leq C \leq 10$), the cost of a lamp of this rating and L ($1 \leq L \leq 100$), the number of lamps required in this category. The input terminates with a test case where $n = 0$. This case should not be processed.

Output

For each test case, print the minimum possible cost to design the system.

Sample Input

```
3
100 500 10 20
120 600 8 16
220 400 7 18
0
```

Sample Output

```
778
```

We say a sequence of characters is a palindrome if it is the same written forwards and backwards. For example, ‘`racecar`’ is a palindrome, but ‘`fastcar`’ is not.

A *partition* of a sequence of characters is a list of one or more disjoint non-empty groups of consecutive characters whose concatenation yields the initial sequence. For example, (‘`race`’, ‘`car`’) is a partition of ‘`racecar`’ into two groups.

Given a sequence of characters, we can always create a partition of these characters such that each group in the partition is a palindrome! Given this observation it is natural to ask: what is the minimum number of groups needed for a given string such that every group is a palindrome?

For example:

- ‘`racecar`’ is already a palindrome, therefore it can be partitioned into one group.
- ‘`fastcar`’ does not contain any non-trivial palindromes, so it must be partitioned as (‘`f`’, ‘`a`’, ‘`s`’, ‘`t`’, ‘`c`’, ‘`a`’, ‘`r`’).
- ‘`aaadbccb`’ can be partitioned as (‘`aaa`’, ‘`d`’, ‘`bccb`’).

W. A. Mozart Palindromic Duet
Part one plays from top
Part 2 plays from the bottom, upside down
Mozart Palindromic Duet Part 2
Note: From this direction, notes to be sharped will have the sharp after, not before the note.
Kitchen Musician http://members.aol.com/kitchiegal/

Can you read upside-down?

Input

Input begins with the number n of test cases. Each test case consists of a single line of between 1 and 1000 lowercase letters, with no whitespace within.

Output

For each test case, output a line containing the minimum number of groups required to partition the input into groups of palindromes.

Sample Input

```
3
racecar
fastcar
aaadbccb
```

Sample Output

```
1
7
3
```

Cars painted in different colors are moving in a row on the road as shown in Figure 1. The color of each car is represented by a single character and the distance of two adjacent cars is assumed to be 1. Figure 1 shows an example of such cars on the road. For convenience, the numbers in Figure 1 represent the locations of each car.

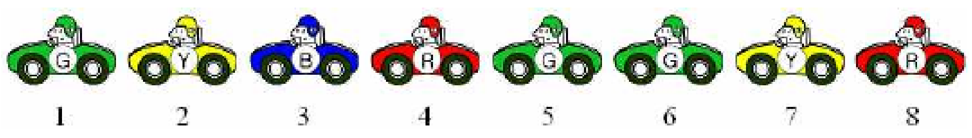


Figure 1. Cars in different colors on the road

For any color c, $location(c)$ represents set of the locations of the cars painted in color c and color length $L(c)$ is defined as follows:

$$L(c) = \max location(c) - \min location(c)$$

For example, according to Figure 1, $location(G) = \{1, 5, 6\}$, $location(Y) = \{2, 7\}$, $location(B) = \{3\}$, and $location(R) = \{4, 8\}$. Hence the color length of each color and the sum of the color lengths are as follows:

Color	G	Y	B	R	**Sum**
$L(c)$	5	5	0	4	14

In Gyeongju City, almost all the roads including the main street of the city were constructed more than 500 years ago. The roads are so old that there are a lot of puddles after rain. Visitors have complained about the bad condition of the roads for many years. Due to the limited budget, the mayor of the city decided to repair firstly the main street of the city, which is a four-lane road, two lanes for each direction.

However, since the main street is a backbone of the city, it should not be blocked completely while it is under repair, or it is expected that serious traffic jams will occur on almost all the other roads in the city. To allow cars to use the main street during the repair period, the city decided to block only two lanes, one lane for each direction. Hence, the cars in the two lanes for each direction should merge into a single lane before the blocked zone.

For instance, as shown in Figure 2, cars in the two lanes merge into a single lane as shown in Figure 3. To differentiate the cars in the same color, a unique identifier is assigned to each car.

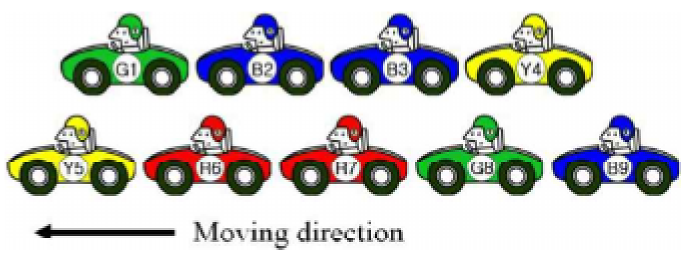
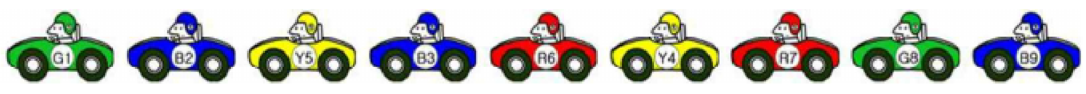


Figure 2. Cars moving in two lanes before merging

Figure 3 shows two different merging scenarios after merging the cars from the two lanes. As shown in Figure 3, cars in the two lanes do not necessarily merge one by one from each lane. The distance between two adjacent cars after merging is also assumed 1.

After merging (Scenario 1):



After merging (Scenario 2):

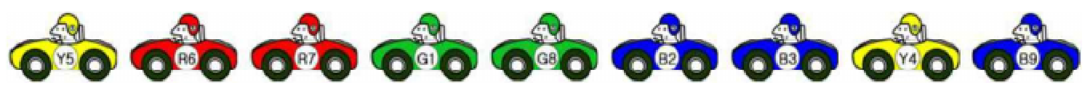


Figure 3. Two different merging scenarios

For each merging scenario shown in Figure 3, the color length for each color and the sum of the color lengths are as follows:

Color	G	Y	B	R	**Sum**
$L(c)$: Scenario 1	7	3	7	2	19
$L(c)$: Scenario 2	1	7	3	1	12

As you can imagine, there are many different ways of merging other than the two examples shown in Figure 3.

Given two character strings which represent the color information of the cars in the two lanes before merging, write a program to find the sum of color lengths obtained from the character string, which is the color information of cars after merging, such that the sum of color lengths is minimized.

Input

Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Each test case consists of two lines. In the first line, a character string of length n ($1 \leq n \leq 5,000$) that is the color information of the cars in one lane before merging is given. In the second line, a character string of length m ($1 \leq m \leq 5,000$) that is the color information of the cars in the other lane is given. Every color is represented as an uppercase letter in English, hence the number of colors is less than or equal to 26.

Output

Your program is to read from standard input. Print exactly one line for each test case. The line should contain the sum of color lengths after merging the cars in the two lanes optimally as described above.

The following shows sample input and output for two test cases.

Sample Input

```
2
AAABBCY
ABBBCDEEY
GBBY
YRRGB
```

Sample Output

```
10
12
```

You have to cut a wood stick into pieces. The most affordable company, The Analog Cutting Machinery, Inc. (ACM), charges money according to the length of the stick being cut. Their procedure of work requires that they only make one cut at a time.

It is easy to notice that different selections in the order of cutting can led to different prices. For example, consider a stick of length 10 meters that has to be cut at 2, 4 and 7 meters from one end. There are several choices. One can be cutting first at 2, then at 4, then at 7. This leads to a price of $10 + 8 + 6 = 24$ because the first stick was of 10 meters, the resulting of 8 and the last one of 6. Another choice could be cutting at 4, then at 2, then at 7. This would lead to a price of $10 + 4 + 6 = 20$, which is a better price.

Your boss trusts your computer abilities to find out the minimum cost for cutting a given stick.

Input

The input will consist of several input cases. The first line of each test case will contain a positive number l that represents the length of the stick to be cut. You can assume $l < 1000$. The next line will contain the number n ($n < 50$) of cuts to be made.

The next line consists of n positive numbers c_i ($0 < c_i < l$) representing the places where the cuts have to be done, given in strictly increasing order.

An input case with $l = 0$ will represent the end of the input.

Output

You have to print the cost of the optimal solution of the cutting problem, that is the minimum cost of cutting the given stick. Format the output as shown below.

Sample Input

```
100
3
25 50 75
10
4
4 5 7 8
0
```

Sample Output

```
The minimum cutting is 200.
The minimum cutting is 22.
```

Let us define a regular brackets sequence in the following way:

1. Empty sequence is a regular sequence.
2. If S is a regular sequence, then (S) and [S] are both regular sequences.
3. If A and B are regular sequences, then AB is a regular sequence.

For example, all of the following sequences of characters are regular brackets sequences:

`()`, `[]`, `(())`, `([])`, `()[]`, `()[()]`

And all of the following character sequences are not:

`(`, `[`, `)`, `)(`, `([)]`, `([]`

Some sequence of characters '`(`', '`)`', '`[`', and '`]`' is given. You are to find the shortest possible regular brackets sequence, that contains the given character sequence as a subsequence. Here, a string $a_1 a_2 \ldots a_n$ is called a subsequence of the string $b_1 b_2 \ldots b_m$, if there exist such indices $1 \leq i_1 < i_2 < \ldots < i_n \leq m$, that $a_j = b_{i_j}$ for all $1 \leq j \leq n$.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The input file contains at most 100 brackets (characters '`(`', '`)`', '`[`' and '`]`') that are situated on a single line without any other characters among them.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

Write to the output file a single line that contains some regular brackets sequence that has the minimal possible length and contains the given sequence as a subsequence.

Sample Input

```
1

([(]
```

Sample Output

```
()[()]
```

Triangulation of surfaces has applications in the Finite Element Method of solid mechanics. The objective is to estimate the stress and strain on complex objects by partitioning them into small simple objects which are considered incompressible. It is convenient to approximate a plane surface with a *simple polygon*, i.e., a piecewise-linear, closed curve in the plane on m distinct vertices, which does not intersect itself. A *chord* is a line segment between two non-adjacent vertices of the polygon which lies entirely inside the polygon, so in particular, the endpoints of the chord are the only points of the chord that touch the boundary of the polygon. A *triangulation* of the polygon, is any choice of $m - 3$ chords, such that the polygon is divided into triangles. In a triangulation, no two of the chosen chords intersect each other, except at endpoints, and all of the remaining (unchosen) chords cross at least one of the chosen chords. Fortunately, finding an arbitrary triangulation is a fairly easy task, but what if you were asked to find the best triangulation according to some measure?

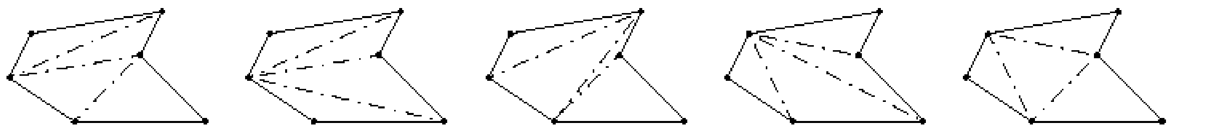


Figure I.1: Five out of nine possible triangulations of the example polygon. The leftmost has the smallest largest triangle.

Input

On the first line of the input is a single positive integer n, telling the number of test scenarios to follow. Each scenario begins with a line containing one positive integer $2 < m < 50$, being the number of vertices of the simple polygon. The following m lines contain the vertices of the polygon in the order they appear along the border, going either clockwise or counter clockwise, starting at an arbitrary vertex. Each vertex is described by a pair of integers x y obeying $0 \le x \le 10000$ and $0 \le y \le 10000$.

Output

For each scenario, output one line containing the area of the largest triangle in the triangulation of the polygon which has the smallest largest triangle. The area should be presented with one fractional decimal digit.

Sample Input

```
1
6
7 0
6 2
9 5
3 5
0 3
1 1
```

Sample Output

```
9.0
```

Dear Contestant,

I'm going to have a party at my villa at Hali-Bula to celebrate my retirement from BCM. I wish I could invite all my co-workers, but imagine how an employee can enjoy a party when he finds his boss among the guests! So, I decide not to invite both an employee and his/her boss. The organizational hierarchy at BCM is such that nobody has more than one boss, and there is one and only one employee with no boss at all (the Big Boss)! Can I ask you to please write a program to determine the maximum number of guests so that no employee is invited when his/her boss is invited too? I've attached the list of employees and the organizational hierarchy of BCM.

Best,
--Brian Bennett

P.S. I would be very grateful if your program can indicate whether the list of people is uniquely determined if I choose to invite the maximum number of guests with that condition.

Input

The input consists of multiple test cases. Each test case is started with a line containing an integer n ($1 \leq n \leq 200$), the number of BCM employees. The next line contains the name of the Big Boss only. Each of the following $n - 1$ lines contains the name of an employee together with the name of his/her boss. All names are strings of at least one and at most 100 letters and are separated by blanks. The last line of each test case contains a single '`0`'.

Output

For each test case, write a single line containing a number indicating the maximum number of guests that can be invited according to the required condition, and a word '`Yes`' or '`No`', depending on whether the list of guests is unique in that case.

Sample Input

```
6
Jason
Jack Jason
Joe Jack
Jill Jason
John Jack
Jim Jill
2
Ming
Cho Ming
0
```

Sample Output

```
4 Yes
1 No
```

A network is composed of N computers connected by $N - 1$ communication links such that any two computers can be communicated via a unique route. Two computers are said to be *adjacent* if there is a communication link between them. The *neighbors* of a computer is the set of computers which are adjacent to it. In order to quickly access and retrieve large amounts of information, we need to select some computers acting as *servers* to provide resources to their neighbors. Note that a server can serve all its neighbors. A set of servers in the network forms a *perfect service* if every client (non-server) is served by **exactly one** server. The problem is to find a minimum number of servers which forms a perfect service, and we call this number *perfect service number*.

We assume that N (≤ 10000) is a positive integer and these N computers are numbered from 1 to N. For example, Figure 1 illustrates a network comprised of six computers, where black nodes represent servers and white nodes represent clients. In Figure 1(a), servers 3 and 5 do not form a perfect service because client 4 is adjacent to both servers 3 and 5 and thus it is served by two servers which contradicts the assumption. Conversely, servers 3 and 4 form a perfect service as shown in Figure 1(b). This set also has the minimum cardinality. Therefore, the perfect service number of this example equals two.

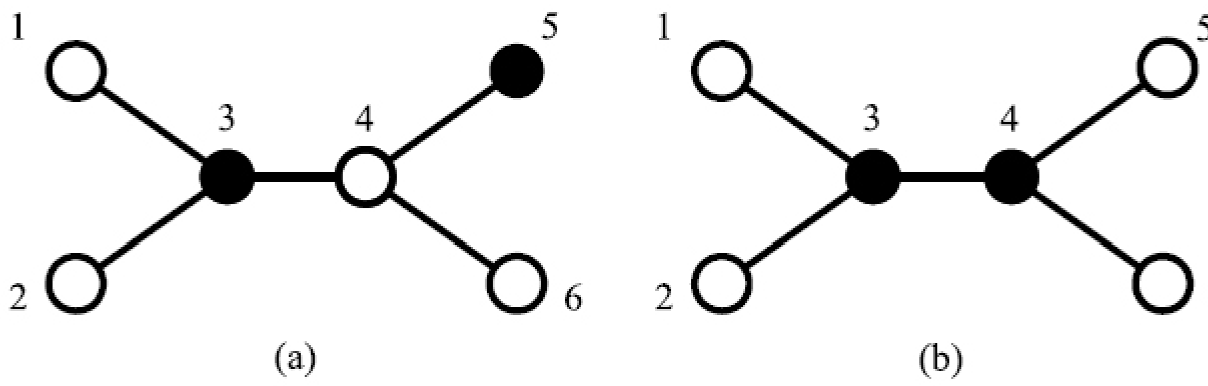


Figure 1

Your task is to write a program to compute the perfect service number.

Input

The input consists of a number of test cases. The format of each test case is as follows: The first line contains one positive integer, N, which represents the number of computers in the network. The next $N - 1$ lines contain all of the communication links and one line for each link. Each line is represented by two positive integers separated by a single space. Finally, a '`0`' at the $(N + 1)$-th line indicates the end of the first test case.

The next test case starts after the previous ending symbol '`0`'. A '`-1`' indicates the end of the whole inputs.

Output

The output contains one line for each test case. Each line contains a positive integer, which is the perfect service number.

Sample Input

```
6
1 3
2 3
3 4
4 5
4 6
0
2
1 2
-1
```

Sample Output

```
2
1
```

Consider a closed world and a set of features that are defined for all the objects in the world. Each feature can be answered with "yes" or "no". Using those features, we can identify any object from the rest of the objects in the world. In other words, each object can be represented as a fixed-length sequence of booleans. Any object is different from other objects by at least one feature.

You would like to identify an object from others. For this purpose, you can ask a series of questions to someone who knows what the object is. Every question you can ask is about one of the features. He/she immediately answers each question with "yes" or "no" correctly. You can choose the next question after you get the answer to the previous question.

You kindly pay the answerer 100 yen as a tip for each question. Because you don't have surplus money, it is necessary to minimize the number of questions in the worst case. You don't know what is the correct answer, but fortunately know all the objects in the world. Therefore, you can plan an optimal strategy before you start questioning.

The problem you have to solve is: given a set of boolean-encoded objects, minimize the maximum number of questions by which every object in the set is identifiable.

Input

The input is a sequence of multiple datasets. Each dataset begins with a line which consists of two integers, m and n: the number of features, and the number of objects, respectively. You can assume $0 < m \leq 11$ and $0 < n \leq 128$. It is followed by n lines, each of which corresponds to an object. Each line includes a binary string of length m which represent the value ("yes" or "no") of features. There are no two identical objects.

The end of the input is indicated by a line containing two zeros. There are at most 100 datasets.

Output

For each dataset, minimize the maximum number of questions by which every object is identifiable and output the result.

Sample Input

```
8 1
11010101
11 4
00111001100
01001101011
01010000011
01100110001
11 16
01000101111
01011000000
01011111001
01101101001
01110010111
01110100111
10000001010
10010001000
10010110100
10100010100
10101010110
10110100010
11001010011
11011001001
11111000111
11111011101
11 12
10000000000
01000000000
00100000000
00010000000
00001000000
00000100000
00000010000
00000001000
00000000100
00000000010
00000000001
00000000000
9 32
001000000
000100000
000010000
000001000
000000100
000000010
000000001
000000000
011000000
010100000
010010000
010001000
010000100
010000010
010000001
010000000
101000000
100100000
100010000
100001000
100000100
100000010
100000001
100000000
111000000
110100000
110010000
110001000
110000100
110000010
110000001
110000000
0 0
```

Sample Output

```
0
2
4
11
9
```

Frank is a portfolio manager of a closed-end fund for *Advanced Commercial Markets* (ACM). Fund collects money (cash) from individual investors for a certain period of time and invests cash into various securities in accordance with fund's investment strategy. At the end of the period all assets are sold out and cash is distributed among individual investors of the fund proportionally to their share of original investment.

Frank manages equity fund that invests money into stock market. His strategy is explained below.

Frank's fund has collected c US Dollars (USD) from individual investors to manage them for m days. Management is performed on a day by day basis. Frank has selected n stocks to invest into. Depending on the overall price range and availability of each stock, a *lot size* was chosen for each stock — the number of shares of the stock Frank can buy or sell per day without affecting the market too much by his trades. So, if the price of the stock is p_i USD per share and the lot size of the corresponding stock is s_i, then Frank can spend $p_i s_i$ USD to buy one lot of the corresponding stock for his fund if the fund has enough cash left, thus decreasing available cash in the fund. This trade is completely performed in one day.

When price of the stock changes to p'_i later, then Frank can sell this lot for $p'_i s_i$ USD, thus increasing available cash for further trading. This trade is also completely performed in one day. All lots of stocks that are held by the fund must be sold by the end of the fund's period, so that at the end (like at the beginning) the fund is holding only cash.

Each stock has its own volatility and risks, so to minimize the overall risk of the fund, for each stock there is the maximum number of lots k_i that can be held by the fund at any given day. There is also the overall limit k on the number of lots of all stocks that the fund can hold at any given day.

Any trade to buy or sell one lot of stock completely occupies Frank's day, and thus he can perform at most one such trade per day. Frank is not allowed to buy partial lots if there is not enough cash in the fund for a whole lot at the time of purchase.

Now, when fund's period has ended, Frank wants to know what is the maximum profit he could have made with this strategy having known the prices of each stock in advance. Your task is to write a program to find it out.

It is assumed that there is a single price for each stock for each day that Frank could have bought or sold shares of the stock at. Any overheads such as fees and commissions are ignored, and thus cash spent to buy or gained on a sell of one lot of stock is exactly equal to its price on this day multiplied by the number of shares in a lot.

Input

Input consists on several datasets. The first line of each dataset contains four numbers — c, m, n, and k. Here c ($0.01 \le c \le 100000000.00$) is the amount of cash collected from individual investors up to a cent (up to two digits after decimal point); m ($1 \le m \le 100$) is the number of days in the fund's lifetime; n ($1 \le n \le 8$) is the number of stocks selected by Frank for trading; k ($1 \le k \le 8$) is the overall limit on the number of lots the fund can hold at any time.

The following $2n$ lines describe stocks and their prices with two lines per stock.

The first line for each stock contains the stock name followed by two integer numbers s_i and k_i. Here s_i ($1 \le s_i \le 1000000$) is the lot size of the given stock, and k_i ($1 \le k_i \le k$) is the number of lots of this stock the fund can hold at any time. Stock name consists of 1 to 5 capital Latin letters from '`A`' to '`Z`'. All stock names in the input file are distinct.

The second line for each stock contains m decimal numbers separated by spaces that denote prices of the corresponding stock for each day in the fund's lifetime. Stock prices are in range from 0.01 to 999.99 (inclusive) given up to a cent (up to two digits after decimal point).

Cash and prices in the input file are formatted as a string of decimal digits, optionally followed by a dot with one or two digits after a dot.

Output

For each dataset, write to the output file $m + 1$ lines. Print a blank line between datasets.

On the first line write a single decimal number — the precise value for the maximal amount of cash that can be collected in the fund by the end of its period. The answer will not exceed 1 000 000 000.00. Cash must be formatted as a string of decimal digits, optionally followed by a dot with one or two digits after a dot.

On the following m lines write the description of Frank's actions for each day that he should have made in order to realize this profit. Write '`BUY`' followed by a space and a stock name for buying a stock. Write '`SELL`' followed by a space and a stock name for selling a stock. Write '`HOLD`' if nothing should have been done on that day.

Sample Input

```
144624.00 9 5 3
IBM 500 3
97.27 98.31 97.42 98.9 100.07 98.89 98.65 99.34 100.82
GOOG 100 1
467.59 483.26 487.19 483.58 485.5 489.46 499.72 505 504.28
JAVA 1000 2
5.54 5.69 5.6 5.65 5.73 6 6.14 6.06 6.06
MSFT 250 1
29.86 29.81 29.64 29.93 29.96 29.66 30.7 31.21 31.16
ORCL 300 3
17.51 17.68 17.64 17.86 17.82 17.77 17.39 17.5 17.3
```

Sample Output

```
151205.00
BUY GOOG
BUY IBM
BUY IBM
HOLD
SELL IBM
BUY MSFT
SELL MSFT
SELL GOOG
SELL IBM
```

It's frosh week, and this year your friends have decided that they would initiate the new computer science students by dropping water balloons on them. They've filled up a large crate of identical water balloons, ready for the event. But as fate would have it, the balloons turned out to be rather tough, and can be dropped from a height of several stories without bursting!

So your friends have sought you out for help. They plan to drop the balloons from a tall building on campus, but would like to spend as little effort as possible hauling their balloons up the stairs, so they would like to know the lowest floor from which they can drop the balloons so that they do burst.

You know the building has n floors, and your friends have given you k identical balloons which you may use (and break) during your trials to find their answer. Since you are also lazy, you would like to determine the minimum number of trials you must conduct in order to determine with absolute certainty the lowest floor from which you can drop a balloon so that it bursts (or in the worst case, that the balloons will not burst even when dropped from the top floor). A trial consists of dropping a balloon from a certain floor. If a balloon fails to burst for a trial, you can fetch it and use it again for another trial.

Input

The input consists of a number of test cases, one case per line. The data for one test case consists of two numbers k and n, $1 \leq k \leq 100$ and a positive n that fits into a 64 bit integer (yes, it's a *very* tall building). The last case has $k = 0$ and should not be processed.

Output

For each case of the input, print one line of output giving the minimum number of trials needed to solve the problem. If more than 63 trials are needed then print '`More than 63 trials needed.`' instead of the number.

Sample Input

```
2 100
10 786599
4 786599
60 1844674407370955161
63 9223372036854775807
0 0
```

Sample Output

```
14
21
More than 63 trials needed.
61
63
```

The Great Wall of China is truly one of the greatest wonders of the world. In 3-rd century BC, Emperor Qin Shi Huang connected the defensive structures built earlier by the states of Qin, Yan, and Zhao kingdoms. The purpose of the wall was to defend against raids by the barbarians from Mongolia and Manchuria. The wall was extended and renovated in later centuries, creating an impressive 6,700 km long fortification.

The centuries have left their mark on the wall, there are several sections that need immediate renovation. These sections have to be repaired as soon as possible since they deteriorate every day: if we do not fix them now, it will be more expensive to repair them later. Thus the Ministry of Monuments have designed and built the world's first Great Wall Automatic Repair Robot (GWARR), to repair the damaged sections (we are in the 21-st century, aren't we?) Your task is to write the software that will guide the robot and decide the order in which the sections are to be repaired.

For the purpose of this problem, we assume that the Great Wall is a long straight line, and every location on the wall is identified by a single number (say, the distance from one end). The GWARR is placed at some location on the wall and it can move with constant speed in both directions. For each damaged section you are given its location, how much it would cost to repair now, and how the cost would increase if repaired later. The GWARR works so efficiently that once it is at the exact location of the damaged section it can repair the wall immediately.

Input

The input contains several blocks of test cases. Each case begins with a line containing three integers: an integer $1 \leq n \leq 1000$, the number of damaged sections, an integer $1 \leq v \leq 100$, the speed of the GWARR in distance units/time units, and an integer $1 \leq x \leq 500000$, the initial position of the GWARR. The next n lines describe the n damaged sections that have to be repaired. Each line contains three integers: the location $1 \leq x \leq 500000$ of the section, the cost $0 \leq c \leq 50000$ of repairing it immediately, and $1 \leq \Delta \leq 50000$, the increase in cost per time unit. Therefore, if the section is repaired after t time units have passed, then we have to pay $c + t\Delta$ units of money. It can be assumed that the locations of the sections are all different, and the initial location of the robot is not on the list of damaged seetions.

The input is terminated by a test case with $n = v = x = 0$.

Output

For each test case, you have to output a line containing a single number, the minimum cost of repairing the wall. This number should be an integer, round *down* the result, if necessary. It can be assumed that the minimum cost is not more than 1000000000.

In the optimum solution for the first test case below, we first fix loeation 998 at the cost of 600, then the location 1010 at the cost of l400, and finally we fix the location 996 at the cost of 84, giving the total cost 2084.

Sample Input

```
3 1 1000
1010 0 100
998 0 300
996 0 3
3 1 1000
1010 0 100
998 0 3
996 0 3
0 0 0
```

Sample Output

```
2084
1138
```

Bob has n matches. He wants to compose numbers using the following scheme (that is, digit 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 needs 6, 2, 5, 5, 4, 5, 6, 3, 7, 6 matches):

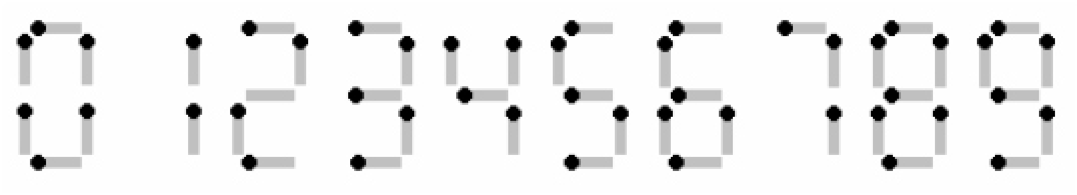


Fig 1 Digits from matches

Write a program to make a non-negative integer which is a multiple of m. The integer should be as big as possible.

Input

The input consists of several test cases. Each case is described by two positive integers n ($n \leq 100$) and m ($m \leq 3000$), as described above. The last test case is followed by a single zero, which should not be processed.

Output

For each test case, print the case number and the biggest number that can be made. If there is no solution, output '`-1`'. Note that Bob don't have to use all his matches.

Sample Input

```
6 3
5 6
0
```

Sample Output

```
Case 1: 111
Case 2: -1
```

A few kids are standing around an old tree playing a game. The tree is so huge that each kid can only see the kids close to him/her.

The game consists many 'turns'. At the beginning of each turn of the game, a piece of paper is given to a randomly chosen kid. This kid writes the letter '`B`' if he is a boy or the letter '`G`' if a girl. Then he chooses a direction to pass the paper (clockwise or counter-clockwise), and gives the paper to his neighbor in that direction. The kid getting the paper writes down his sex too, and gives the paper to his neighbor in the same direction. In this way, the paper goes through the kids one by one, until one kid stops passing the paper and announces the end of this turn.

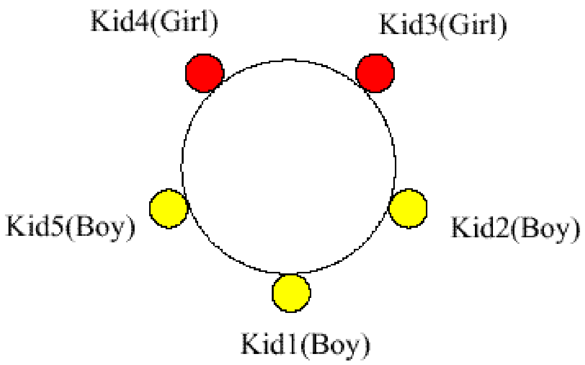


Figure-1. Five kids around the tree

For example, there are five kids around the tree, and their genders are shown in Figure-1. The paper first goes to Kid1, after writing a '`B`' he passes it to Kid2, and Kid2 to Kid3. After Kid3 writes down a '`G`', she ends up this turn, and we get the paper with a string '`BBG`'.

After N turns, we get N pieces of paper with strings of '`B`'s and/or '`G`'s. One of the kids will get all these papers, and has to figure out at least how many kids are around the tree playing the game. It's known that there are at least two kids. Please write a program to help him.

Input

There are several test cases. Each case starts with a line containing an integer N, the number of papers ($2 \leq N \leq 16$). Each of the following N lines contains a string on a paper, which is a nonempty string of letter '`B`'s and/or '`G`'s. Each string has no more than 100 letters.

A test case of $N = 0$ indicates the end of input, and should not be processed.

Output

For each test case, output the least possible number of kids in a line.

Sample Input

```
3
BGGB
BGBGG
GGGBGB
2
BGGGBBBGG
GBBBG
0
```

Sample Output

```
9
6
```

No wonder the old bookcase caved under the massive piles of books Tom had stacked on it. He had better build a new one, this time large enough to hold all of his books. Tom finds it practical to have the books close at hand when he works at his desk. Therefore, he is imagining a compact solution with the bookcase standing on the back of the desk. Obviously, this would put some restrictions on the size of the bookcase, it should preferably be as small as possible. In addition, Tom would like the bookcase to have exactly three shelves for aesthetical reasons.

Wondering how small his bookcase could be, he models the problem as follows. He measures the height h_i and thickness t_i of each book i and he seeks a partition of the books in three non-empty sets S_1, S_2, S_3 such that $(\sum_{j=1}^{3} \max_{i \in S_j} h_i) \times (\max_{j=1}^{3} \sum_{i \in S_j} t_i)$ is minimized, i.e. the area of the bookcase as seen when standing in front of it (the depth needed is obviously the largest width of all his books, regardless of the partition). Note that this formula does not give the exact area of the bookcase, since the actual shelves cause a small additional height, and the sides cause a small additional width. For simplicity, we will ignore this small discrepancy.

Thinking a moment on the problem, Tom realizes he will need a computer program to do the job.

Input

The input begins with a positive number on a line of its own telling the number of test cases (at most 20). For each test case there is one line containing a single positive integer N, $3 \leq N \leq 70$ giving the number of books. Then N lines follow each containing two positive integers h_i, t_i, satisfying $150 \leq h_i \leq 300$ and $5 \leq t_i \leq 30$, the height and thickness of book i respectively, in millimeters.

Output

For each test case, output one line containing the minimum area (height times width) of a three-shelf bookcase capable of holding all the books, expressed in square millimeters.

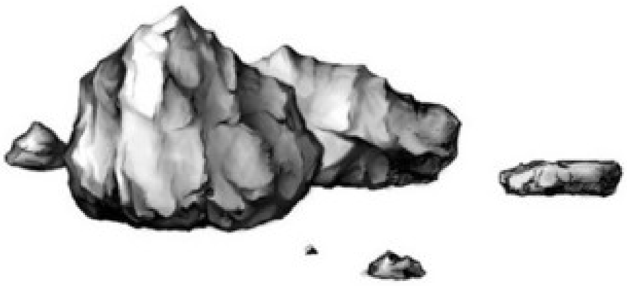
Sample Input

```
2
4
220 29
195 20
200 9
180 30
6
256 20
255 30
254 15
253 20
252 15
251 9
```

Sample Output

```
18000
29796
```

Somewhere in the neighborhood we have a very nice mountain that gives a splendid view over the surrounding area. There is one problem though: climbing this mountain is very difcult, because of rather large height differences. To make more people able to climb the mountain and enjoy the view, we would like to make the climb easier.



To do so, we will model the mountain as follows: the mountain consists of n adjacent stacks of stones, and each of the stacks is h_i high. The successive height differences are therefore $h_{i+1} - h_i$ (for $1 \le i \le n-1$). We would like all absolute values of these height differences to be smaller than or equal to some number d.

We can do this by increasing or decreasing the height of some of the stacks. The rst stack (the starting point) and the last stack (the ending point) should remain at the same height as they are initially. Since adding and removing stones requires a lot of effort, we would like to minimize the total number of added stones plus the total number of removed stones. What is this minimum number?

Input

On the rst line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with two integers n ($2 \le n \le 100$) and d ($0 \le d \le 10^9$): the number of stacks of stones and the maximum allowed height difference.
- One line with n integers h_i ($0 \le h_i \le 10^9$): the heights of the stacks.

Output

Per testcase:

- One line with the minimum number of stones that have to be added or removed or '`impossible`' if it is impossible to achieve the goal.

Sample Input

```
3
10 2
4 5 10 6 6 9 4 7 9 8
3 1
6 4 0
4 2
3 0 6 3
```

Sample Output

```
6
impossible
4
```

There is a set of jobs, say $x_1, x_2, \ldots, x_n$, to be scheduled. Each job needs one day to complete. Your task is to schedule the jobs so that they can be finished in a minimum number of days. There are two types of constraints: *Conflict constraints* and *Precedence constraints*.

Conflict constraints: Some pairs of jobs cannot be done on the same day. (Maybe job x_i and job x_j need to use the same machine. So they must be done in different dates).

Precedence constraints: For some pairs of jobs, one needs to be completed before the other can start. For example, maybe job x_i cannot be started before job x_j is completed.

The scheduling needs to satisfy all the constraints.

To record the constraints, we build a graph G whose vertices are the jobs: $x_1, x_2, \ldots, x_n$. Connect x_i and x_j by an undirected edge if x_i and x_j cannot be done on the same day. Connect x_i and x_j by a directed edge from x_i to x_j if x_i needs to be completed before x_j starts.

If the graph is complicated, the scheduling problem is very hard. Now we assume that for our problems, the constraints are not very complicated: The graph G we need to consider are always trees (after omitting the directions of the edges). Your task is to find out the number of days needed in an optimal scheduling for such inputs. You can use the following result:

> *If G is a tree, then the number of days needed is either k or $k+1$, where k is the maximum number of vertices contained in a directed path of G, i.e., a path $P = (x_1, x_2, .$ [illegible] for each $i = 1, 2, \ldots, k-1$, there is a directed edge from x_i to x_{i+1}.*

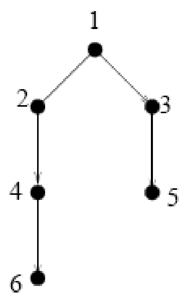


Figure 1

Figure 1 on the right is such an example. There are six jobs: 1, 2, 3, 4, 5, 6. From this figure, we know that job 1 and job 2 must be done in different dates. Job 1 needs to be done before job 3, job 3 before job 5, job 2 before job 4 and job 4 before job 6. It is easy to verify that the minimum days to finish all the jobs is 4 days. In this example, the maximum number k of vertices contained in a directed path is 3.

Input

The input consists of a number of trees (whose edges may be directed or undirected), say $T_1, T_2, \ldots, T_m$, where $m \leq 20$. Each tree has at most 200 vertices. We represent each tree as a rooted tree (just for convenience of presentation, the root is an arbitrarily chosen vertex). Information of each of the trees are contained in a number of lines. Each line starts with a vertex (which is a positive integer) followed by all its sons (which are also positive integers), then followed by a 0. Note that 0 is not a vertex, and it indicates the end of that line. Now some of the edges are directed. The direction of an edge can be from father to son, and can also be from son to father. If the edge is from father to son, then we put a letter '`d`' after that son (meaning that it is a downward edge). If the edge is from son to father, then we put a letter '`u`' after that son (meaning that it is an upward edge). If the edge is undirected then we do not put any letter after the son.

The first case of the sample input below is the example in Figure 1.

Consecutive vertices (numbers or numbers with a letter after it) in a line are separated by a single space. A line containing a single '`0`' means the end of that tree. The next tree starts in the next line. Two consecutive lines of single '`0`' means the end of the input.

Output

The output contains one line for each test case. Each line contains a number, which is the minimum number of days to finish all the jobs in that test case.

Sample Input

```
1 2 3d 0
2 4d 0
3 5d 0
4 6d 0
0
1 2d 3u 4 0
0
1 2d 3 0
2 4d 5d 10 0
3 6d 7d 11 0
6 8d 9 12 0
0
1 2 3 4 0
2 5d 0
3 6d 0
4 7d 0
5 8d 0
6 9d 0
7 10d 0
0
0
```

Sample Output

```
4
3
4
3
```

Some of you may have played a game called 'Blocks'. There are n blocks in a row, each box has a color. Here is an example: Gold, Silver, Silver, Silver, Silver, Bronze, Bronze, Bronze, Gold.

The corresponding picture will be as shown below:

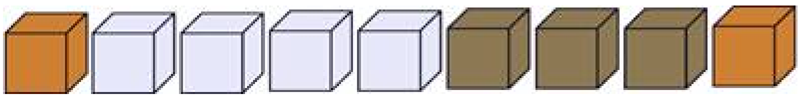


Figure 1

If some adjacent boxes are all of the same color, and both the box to its left (if it exists) and its right (if it exists) are of some other color, we call it a box segment'. There are 4 box segments. That is: gold, silver, bronze, gold. There are 1, 4, 3, 1 box(es) in the segments respectively.

Every time, you can click a box, then the whole segment containing that box DISAPPEARS. If that segment is composed of k boxes, you will get $k*k$ points. for example, if you click on a silver box, the silver segment disappears, you got $4*4 = 16$ points.

Now lets look at the picture below:

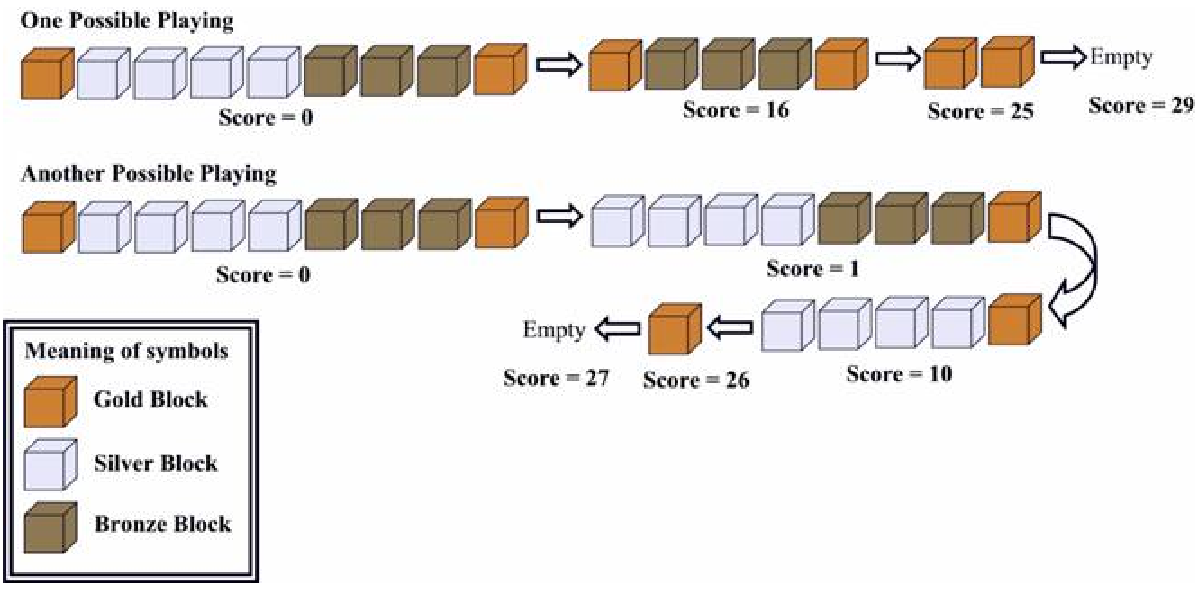


Figure 2

The first one is OPTIMAL.

Find the highest score you can get, given an initial state of this game.

Input

The first line contains the number of tests t ($1 \le t \le 15$). Each case contains two lines. The first line contains an integer n ($1 \le n \le 200$), the number of boxes. The second line contains n integers, representing the colors of each box. The integers are in the range $1 \sim n$.

Output

For each test case, print the case number and the highest possible score.

Sample Input

```
2
9
1 2 2 2 2 3 3 3 1
1
1
```

Sample Output

```
Case 1: 29
Case 2: 1
```

You're transmitting an n-bits unsigned integer k through a simulated network. The i-th bit counting from left is transmitted at time i (e.g. 4-bit unsigned integer 5 is transmitted in this order: 0-1-0-1). The network delay is modeled as follows: if a bit is transmitted at time i, it may arrive at as early as $i + 1$ and as late is $i + d + 1$, where d represents the maximal network delay. If more than one bit arrived at the same time, they could be received in any order.

For example, if you're transmitting a 3-bit unsigned integer 2 (010) for $d = 1$, you may receive 010, 100 (first bit is delayed) or 001 (second bit is delayed).

Write a program to find the number of different integers that could be received, and the smallest/largest ones among them.

Input

The input contains several test cases. Each case consists of three integers n, d, k ($1 \leq n \leq 64$, $0 \leq d \leq n, 0 \leq k < 2^n$), the number of bits transmitted, the maximal network delay, and the integer transmitted. The last test case is followed by a single zero, which should not be processed.

Output

For each test case, print the case number and the number of different integers that could be received, followed by the minimal and maximal one among them.

Sample Input

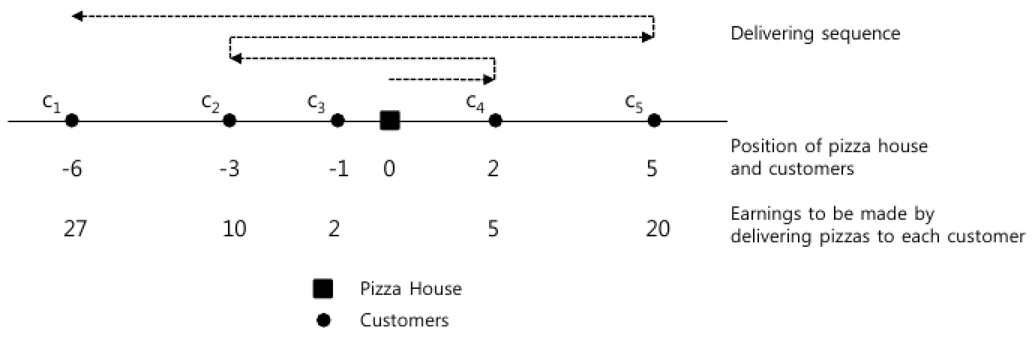
```
3 0 2
3 1 2
10 2 888
7 3 107
0
```

Sample Output

```
Case 1: 1 2 2
Case 2: 3 1 4
Case 3: 25 490 984
Case 4: 19 47 122
```

There is a pizza house located on a straight road, and there are many houses along the road which are customers to the pizza house. To attract more orders from his customers, the owner of the pizza house advertised that he will deduct a penalty for late delivery from the price of the delivered pizzas. The penalty is to be charged when some designated time period has passed after the order has been made, and the amount of penalty to be charged is 1 Korean Won per unit time thereafter. Today all houses on the road ordered pizza at the same time and the delivery of all the ordered pizzas has just started when the late delivery penalty is going to be charged. On a busy day like today, he may not deliver pizza to some customers if the late delivery penalty to be deducted for a customer is more than the earning to be made by selling pizza to the customer. Write a program to help him decide to which customers he has to deliver pizzas and which customers he may skip in order to make the greatest profit. Note that the profit made by delivering a pizza to a customer is the amount of earning from the service deducting the penalty for late delivery. You may assume that his moving velocity is one unit distance per unit time and it takes no time to hand over the pizza to the customer.

For example, the following figure shows the relative positions of five customers $\{c_1, c_2, \ldots, c_5\}$ to the pizza house and the earning to be made by selling pizzas to each customer.


Delivering sequence
c1
c2
c3
c4
c5
-6
-3
-1
0
2
5
Position of pizza house and customers
27
10
2
5
20
Earnings to be made by delivering pizzas to each customer
Pizza House
Customers

If the delivering sequence of customers is $< c_4, c_3, c_2, c_5, c_1 >$, then the amount of penalty for late delivery to each customer is 2 for c_4, 5 for c_3, 7 for c_2, 15 for c_5 and 26 for c_1. In this case the profit from each customer is 3 for c_4, -3 for c_3, 3 for c_2, 5 for c_5 and 1 for c_1. Since the profit from customer c_3 is -3, it is better not to deliver pizza to c_3. Therefore the total profit by delivering pizzas to the customers in this order is 12. The best profit the owner can make, in this case, is 32, where the delivering sequence is $< c_3, c_2, c_1, c_5 >$.

Given the relative positions of customers to pizza house, and earnings to be made by delivering pizzas to each customer, write a program to compute the maximum profits by delivering pizzas ordered from the customers.

Input

Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Each test case consists of three lines. The first line of each test case contains an integer, n ($1 \leq n \leq 100$), which is the number of customers to pizza house. The second line of each test case contains n integers p_1, p_2, $\ldots$, p_n ($p_1 < p_2 < \ldots < p_n$ and $p_i \neq 0$, $i = 1, \ldots, n$), where p_i is the relative position of the i-th customer c_i to the pizza house on the straight road. The third line of each test case contains n integers e_1, e_2, $\ldots$, e_n ($e_i > 0$, $i = 1, \ldots, n$), where e_i is the earning to be made by delivering pizzas to the customer c_i. All integers in the second and third lines are between -100,000 and 100,000 both inclusive.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain the maximum profit by delivering pizzas ordered from the customers.

The following shows sample input and output for three test cases.

Sample Input

```
3
5
-6 -3 -1 2 5
27 10 2 5 20
6
1 2 4 7 11 14
3 6 2 5 18 10
11
-14 -13 -12 -11 -10 1 2 3 4 5 100
200 200 200 200 200 200 200 200 200 200 200
```

Sample Output

```
32
13
1937
```