9th Gulf Programming Contest

March 31 – April 2, 2019

SQU, Oman

Duration: 10 am - 3 pm

Sponsored by

Problem Set

Problem #	***Problem Name***	***Balloon Color***
A	The Purifier	Orange
B	*n* Base Palindrome	Red
C	Miles of The Sky	Yellow
D	Take that Photo	Blue
E	Fair Share	White
F	Fruit Burst	Black
G	Cubenite	Golden
H	Dice of Probability	Green
I	The Secret Message	Purple
J	The Legend of Diapers	Brown
K	Pyramids	Pink
L	Table Format	Silver

A. The Purifier

Program:	purifier.(cpp|java|py)
Input:	purifier.in
Balloon Color:	Orange

Description

Professor George P. Costas is trying to catch up with the new season of his favorite show “The Purifier” that is scheduled to be released next week. However he has still to watch X hours of the previous season and there are only Y days left.

He wants to know if given his busy schedule for this week, he has enough time to watch the previous episodes. Prof. G.P.C.’s busy schedule for the next Y days is given as a list of Y lines, each containing the busy periods of a day, in the form of time pairs. You can also assume that there are 10 hours in each day devoted to sleeping, eating, etc.

For each test case you should answer with a Yes or a No whether he will be able to finish the previous season or not.

If for example X=10 and Y=2 and the schedule is as follows:

10:00-12:00 13:00-22:00

9:00-17:00

then there will not be enough time for him to catch up before the new season airs.

Input

The input file starts with a line containing the number of test cases N≤100. Each test case starts with a line containing 0<X≤100 and 0<Y≤7 separated by white-space. Y lines follow, each containing up to 10 reserved timeslots, or "-" if that day is a holiday. You can assume that the timeslots are not overlapping (but not sorted) and that they leave at least 10 hours in a day free. The timeslots are given in the form of hh1:mm1-hh2:mm2 and the times are inclusive, i.e. he is busy during hh2:mm2. All times are given in 24h format. The hours can be one or two digits, but the minutes are always two digits.

Output

For each test case you should output the number of the test case (starting from 1), followed by a ".", a space and "Yes" or "No", all in a separate line.

Sample Input / Output

purifier.in

```
2
10 2
10:00-12:00 13:00-22:00
9:00-17:00
15 4
10:00-12:00 13:00-22:00
9:00-17:00
0:00-0:00
-
```

OUTPUT

```
1. No
2. Yes
```

B. *n* Base Palindrome

Program:	palindrome.(cpp|java|py)
Input:	palindrome.in
Balloon Color:	Red

Description

A palindrome is a number that has the same value when its digits are reversed. 1, 101, and 33 etc., are examples of palindromes, while 52, 122, and 12 are not. While some numbers might not be palindromes in decimal, they can be palindromes in other bases. Given a number in decimal, and a number base, state if the given number is a palindrome in that base or not.

Example

23 is not palindrome, however, when converted into base 3 it becomes: 212 which is palindrome.

Input

The input consists of several test cases, each test case, has two integers, a number *n* in decimal, and a number base *b* in decimal. Input is terminate by 0 0.

$1 \leq n \leq 1000000$, $2 \leq b \leq 1000000$

Output

For each sequence, you are to output Yes if palindrome, No if not.

Sample Input / Output

palindrome.in

```
52 10
33 10
33 8
23 3
20 16
5 2
0 0
```

OUTPUT

```
No
Yes
No
Yes
No
Yes
```

C. Miles of The Sky

Program:	`miles.(cpp|java|py)`
Input:	`miles.in`
Balloon Color:	`Yellow`

Description

In an effort to maintain customers, most airline companies have loyalty programs where members earn 'miles' every time they fly with the airline and sometimes when they make purchases with other partner companies. In most programs, members earn two different types of 'miles' when they fly; One is 'Reward Miles' and the other is 'Tier Miles'. Reward Miles can be used a virtual currency to be exchanged for free flights, free upgrades, and in some cases can be used to make purchases in airline duty free shops. Tier Miles are only used to determine the status of the member of the loyalty program. For instance, a newly registered member can start off with a 'Bronze' level, and as they fly and ear more Tier Miles, then can move up their status to 'Silver' level, and so on. Both Reward Miles and Tier Miles have a certain validity period after which they become expired. When Reward Miles expire, they simply cannot be used to purchase flights, upgrades, etc. When Tier Miles expire, they do not contribute to the status of the member. Reward Miles are typically valid for 3 years, while Tier Miles usually expire in 1 year.

We need your help to write a program that will track loyalty program members and their flights to determine their member status. For simplicity, you will be provided with the number of Tier Miles each member earns per month, and the validity period in months. Your task is to calculate the maximum number of Tier Miles each member has achieved.

Input

The input consists of several test cases, where each test case represents data for one member. The test case begins with two integers M ($1 \le M \le 500{,}000$), and V ($1 \le V \le 500{,}000$), where M represents the months the member has been flying with the airline, and V is the number of months Tier Miles are valid for. The following line contains M integers Ti ($0 \le Ti \le 1{,}000$), representing the number of Tier Miles earned in each month, in order. The last test case is followed by a line containing two 0's.

Output

The output for each test case is in this form:

```
k. S
```

where ***k*** represents the test case number (starting at 1), and ***S*** is the maximum number of Tier Miles achieved.

Sample Input / Output

miles.in

```
4 2
10 20 20 10
6 3
10 50 10 20 40 20
7 4
15 20 10 45 20 15 7
0 0
```

OUTPUT

```
1. 40
2. 80
3. 95
```

D. Take that Photo

Program:	photo.(cpp|java|py)
Input:	photo.in
Balloon Color:	Blue

Description

A kindergarten teacher gathers up her class for a school photo. The children run and stand in one line regardless of their height. That wouldn't look nice for the photo, so she asks for your help to rearrange the children in a non-decreasing order according to their height. You then realize that the ordering can be done in more than one way since some children have the same height. Being the inquisitive person, you decide to write a program that finds out how many different photos can be taken of the children in a non-decreasing order of their height.

Input

First line of inputs contains a single integer ***n*** (***1*** <= ***n*** <= ***100***) representing the number of test cases. Each of the next ***n*** lines contains a list of integers separated by 1 or more spaces as follows:

m v_1 v_2 v_3 … v_m where ***1*** <= ***m*** <= ***20*** (***m*** being the number of children per photo)

and 0 <= v_i <= ***100***.

Output

For each test case, you will output a single line of the form

k. ans

where ***k*** is the test case number starting with ***1***, and ***ans*** is the number of different ways the integers can be arranged in a non-decreasing order.

Sample Input / Output

photo.in

```
3
4 1 2 1 4
8 1 2 1 2 1 2 2 1
6 7 5 1 4 23 11
```

OUTPUT

```
1. 2
2. 576
3. 1
```

E. Fair share

Program:	fair.(cpp|java|py)
Input:	fair.in
Balloon Color:	White

Description

Tom and Jerry found *K* cakes (1 < K <= 10), each shaped like a decimal digit. They had a fight on who should eat which cake. Daniel came to rescue. He wanted to give them a fair share. This was done as follows. Daniel first constructed two *n* digit numbers (*n* <= K/2) from the given set of cakes whose difference is minimum. "This difference is the fair share", said Daniel. He then gave one cake number to Jerry and the other to Tom. Both Tom and Jerry were happy to eat their numbers (i.e., cakes), knowing that their difference is the fair share.

Example

Given the cake set {0, 1, 3, 5, 6, 9}, and *n* =2, the fair share would be = 1, as the two 2-digit numbers that generates the fair share are 59 and 60. Note that in this example, 01 can be a valid 2-digit number.

Input

The input consists of several test cases. Each test case is given in two lines. The first line containing K (1 < K <= 10) followed by *n* (1 <= n <= K/2). The second line contains *K* distinct digits (between 0-9) separated by space(s). Input is terminated by a case having *K* = 0, which should not be processed.

Output

For each test case, you are to output one line, containing the fair share.

Sample Input / Output

fair.in

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

OUTPUT

```
2
1
1
```

F. Fruit burst

Program:	`fruit.(cpp|java|py)`
Input:	`fruit.in`
Balloon Color:	`Black`

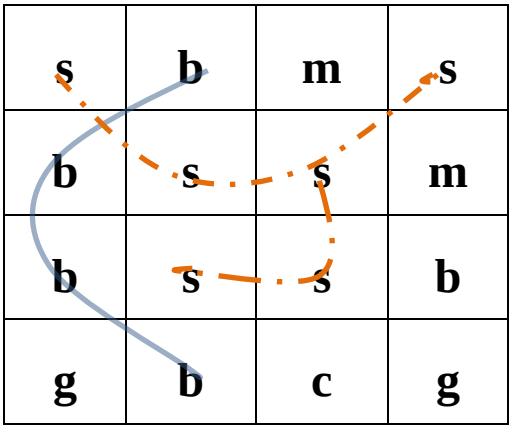
Description

Kids love to play fruit burst. It is a board game where the board is an *L* x *L* square grid, and each grid cell contains a fruit. A player connects at least *n* fruits of the same type, and bang! All those connected fruits burst, and he gets *p* * *k* points, where *p* is the point for the fruit and *k* is the number of fruits (same type) that the player connected. Your task is to find the highest point possible given the current board state.

Example

As an example, consider the following board (4x4 grid) and 5 types of fruits, *s*, *b*, *g*, c, and *m* having 10, 16, 25, 10, and 10 points, respectively. Also, *n* = 3 in this board. Two fruits are connected if they share a common edge or a corner in the grid. The board shows that maximum six *s* fruits can be connected and maximum four *b* fruits can be connected. So, when the player connects *s*, all *s* fruits burst and he gets 60 points. If he connected four *b* fruits, he would get 64. Note that *c*, *g* and *m* fruits do not burst because the player can only connect less than 3 such fruits. Therefore, they don't give any point. Therefore, the max burst is 64.

Fruit points: b = 16, g = 25 and default = 10 (i.e., s = c = m = 10)



burst(b) = 4 x 16 = 64 (connected b's shown in solid line)

burst(s) = 6 x 10 = 60 (connected s's shown in the dotted line)

burst(c) = burst(g) = burst(m) = 0

Input

The input consists of several test cases. The first line of each test case contains L (2 < L <= 100) and *n* (1 < n < *L*) followed by the grid description. The grid is given in *L* lines, each line containing *L* fruit names separated by space(s). A fruit name is a lower case character [a-z]. After the grid input, fruit

points are given in one line. In this line, a fruit name is given followed by its points, separated by spaces. The last entry in this line will be the character '#' followed by the default fruit points. Any fruit not mentioned in this line will assume the default points. All fruit points are integers > 0. Input is terminated by a case having $K = 0$, which should not be processed.

Output

For each test case, you are to output one line, containing the max burst value.

Sample Input / Output

fruit.in

```
4 3
s b m s
b s s m
b s s b
g b c g
g 25 b 16 # 10
0
```

OUTPUT

```
64
```

G. Cubenite

Program:	`cube.(cpp|java|py)`
Input:	`cube .in`
Balloon Color:	`Gold`

Description

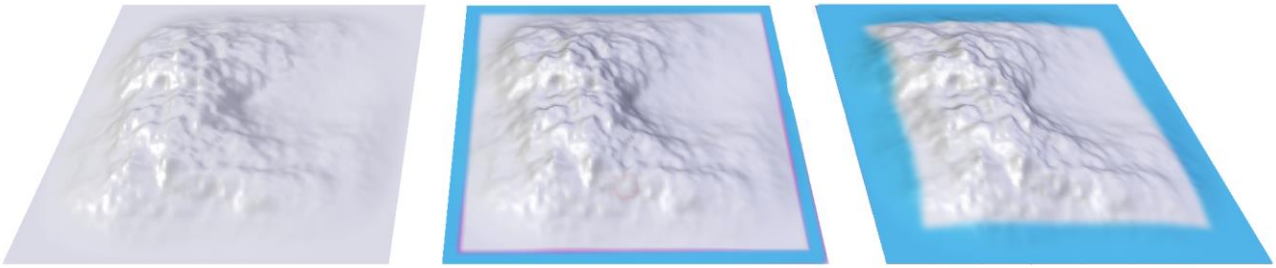
You are faced with a dilemma playing the new Cubenite battle-royale game that has just been released: how can you collect all the weapons from the sites on the map, before the storm gets to you.

As you are a keen programmer, you decide that some code would be helpful in making this decision.

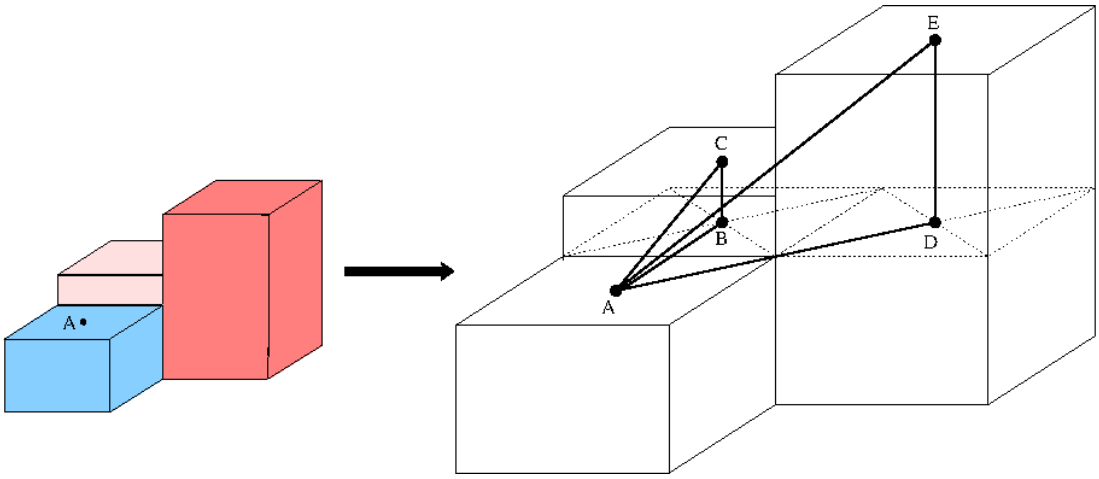
Your input consists of:

- a map of the game in the form of a grid, where the elevation of each cell is given. The grid coordinates start from (0,0), which corresponds to the top-left corner. Each cell of the grid is assumed to be a square with a 1 meter side.
- your initial coordinates
- the list of coordinates with weapon caches
- your speed (meters per second)

The storm moves from the boundaries of the grid inwards, as shown in the example below, at a steady speed of one meter per second.



During your move on the map, you can move between any two adjacent cells. The distance traveled will be equal to the hypotenuse of the triangle formed by joining the peaks of the two cells as shown below:



So if your initial position is A, moving to the top cell requires travelling the AC distance (BC is equal to the elevation difference between the two cells). Moving from A to the top-right cell would require moving across the AE segment. The lengths of all segments can be assumed to be integers (rounded-up to the closest one). If the player is to move from A to C and then to E, the total distance would be the sum of the lengths of the AC and the CE segments.

Your goal is to visit the maximum number of weapon caches, in the least amount of time, without getting caught by the storm, i.e. visiting a cell where the storm has already reached.

The output of your program should be either a list of the set of weapon caches in the order you visit them, or the string "The storm will get you" if none is reachable.

Input

The input starts with the number of test cases N (≤ 50) in a single line. Each test case start with a line containing the grid width W and height H separated by whitespace ($W, H \leq 100$). The next line contains the initial position coordinates X ($0 \leq X < W$) and Y ($0 \leq Y < H$), followed by the speed V ($1 \leq V < 100$) in meters per second, and the number of weapon depots WD ($0 < WD \leq 8$).

WD lines follow, each containing the x and y coordinates of a weapon depot.

Finally H lines follow, each containing the elevation of W cells. Each elevation is a non-negative integer less than 100.

Output

For each test case you should output in a separate line, the case number (starting from 1), followed by a '.', a space and the list of weapon depots, numbered according to their appearance order in the input file. E.g.:

```
2. 7013
```

Means that for case 2, the maximum number of depots that can be reached is 4, going in the order 7 -> 0 -> 1 -> 3. If none is reachable, you should output the string "`The storm will get you`".

Sample Input / Output

cube.in

```
2
10 10
9 8 100 3
6 7
3 3
4 4
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 3 3 1 1 1 1 1 1
1 1 3 3 1 1 1 1 1 1
1 1 1 1 8 1 1 1 1 1
1 9 9 1 8 1 1 1 1 1
1 9 9 1 8 1 1 1 1 1
1 1 1 1 8 1 1 1 1 1
10 10
0 0 1 3
3 3
4 4
6 7
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 3 3 1 1 1 1 1 1
1 1 3 3 1 1 1 1 1 1
1 1 1 1 8 1 1 1 1 1
1 9 9 1 8 1 1 1 1 1
1 9 9 1 8 1 1 1 1 1
1 1 1 1 8 1 1 1 1 1
```

OUTPUT

```
1. 012
2. The storm will get you
```

H. Dice of Probability

Program:	`dice.(cpp|java|py)`
Input:	`dice.in`
Balloon Color:	`Green`

Description

'Risk' is a board game created in the 50's and is still a popular game today. It is a war and tactics game with the eventual target of conquering the world. Risk has many rules and variants. The rule of interest for this problem relates to how a player would attack and defend their territories. Each territory has a number of armies, which can be used to attack an adjacent territory or defend against an incoming attack. Risk uses dice rolls to decide if an attack succeeds. The attacker would roll a number of dice, usually 3, and the defender would roll a number of dice, usually 2. The highest dice for both attacker and defender would be matched, then the second highest from each would be matched, and so on. For each match the highest number wins, and in case of a draw the defender wins. For each pair of dice match, the losing territory loses one of its armies. For instance, if the attacker rolls 3 dice and gets (4, 3, 3) and the defender rolls 2 dice and gets (3, 2), then the defender loses two armies. If the attacker dice are (4, 3, 3) and the defender's are (3, 3), then each would lose one army.

Example with Attacker (4,3,3), and Defender (3,2)

Given the number of dice rolled for the attacker and the defender and the value for each dice. Calculate how many armies each of them loses.

Input

The input starts with a number T ($1 \le T \le 1{,}000$) that represents the number of test cases in the file. Each test case starts with a line that contains two integers A ($1 \le A \le 1{,}000$), and D ($1 \le D \le 1{,}000$), representing the number of dice rolled by the attacker, and the number of dice rolled by the defender, respectively. Two lines follow, the first containing A integers representing the value for each dice for the attacker, and the second containing D integers representing the value for each dice for the defender. A dice value is an integer between 1 and 6, inclusive.

Output

The output for each test case is in this form:

```
k. L R
```

where ***k*** represents the test case number (starting at 1), ***L*** is the number of armies the attacker loses, and ***R*** is the number of armies the defender loses.

Sample Input / Output

dice.in

```
3
3 2
4 3 3
3 2
3 2
4 3 3
3 3
3 3
6 3 4
5 2 2
```

OUTPUT

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

I. The Secret Message

Program:	secret.(cpp|java|py)
Input:	secret.in
Balloon Color:	Purple

Description

You have crafted a new way of sharing secret messages with your friends. With your method, the sender sends a string of a mix of alphabet and numeric characters, and the receiver can reconstruct the message without having to know a predefined key. The way this works is that the receiver would use the numeric characters to re-order the alphabet characters, in a rather simple way. As the string is read, digits would appear. If this digit is even, the previous alphabet characters up to the previous digit (or up to the beginning of the text if it's the first digit) would be added to the beginning (left) of the message. If the digit is odd, they would be added to the end (right) of the message. For example, if the secret is "lo8el4Wor7H2ld3", then the message would be "HelloWorld".

Given the secret, write a program to display the message.

Input

The input starts with a number T ($1 \leq T \leq 1{,}000$) that represents the number of test cases in the file. Each test case is on a line that contains a string representing the secret. The maximum number of characters in the string is 100,000, and the minimum is 2 characters. The string always begins with an letter and ends with a digit. The string only contains alphabets and digits and has no blank spaces. Consecutive digits are possible.

Output

The output for each test case is in this form:

```
k. M
```

where ***k*** represents the test case number (starting at 1), and ***M*** is the decoded message.

Sample Input / Output

secret.in

```
2
lo8el4Wor7H2ld3
tM1re4es98sa3Sec0ge7
```

OUTPUT

```
1. HelloWorıɑ
2. SecretMessage
```

J. The Legend of Diapers

Program:	diapers.(cpp|java|py)
Input:	diapers.in
Balloon Color:	Brown

Description

The baby is here. Your life is changed forever. But first things first, you need to get diapers. This baby is going through diapers like a legend, so you will need a program to calculate how long the diapers you have will last.

Given the number of diapers you have, and how many diapers per day this legendary baby is using, your task is to write a program to calculate how many days you have until you will have to buy more diapers.

Input

The input starts with a number T ($1 \le T \le 1{,}000$) that represents the number of test cases in the file. Each test case is represented on a line that contains two integers D ($1 \le D \le 10^9$), and B ($1 \le B \le 10^9$), representing the number of diapers you have now, and the number of diapers per day the baby is using, respectively.

Output

The output for each test case is in this form:

```
k. M
```

where ***k*** represents the test case number (starting at 1), and ***M*** is the number days you have left before you will need to buy more diapers.

Sample Input / Output

diapers.in

```
3
20 5
20 9
10 10
```

OUTPUT

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

K. Pyramids

Program:	pyramids.(cpp|java|py)
Input:	pyramids.in
Balloon Color:	Pink

Description

A pyramids are discovered, each has a triangular base with equal sides. An *n* meters high pyramid is constructed by forming *n* layers of triangles using $1m^3$ cubical stones. Then layers are stacked on top of each other. The first layer is the triangle with *n* meters base, on top, *n*-1 meters base, until the top level which has only 1 stone. You are asked to calculate the number of stones needed to build a pyramid with the height *n*. WAIT!! We just discovered that some pyramids have a secret room inside. This room is also of the same shape and structure as the pyramid with *m* layers of stones removed. In order for the room to be secret, $m < n - 2$.

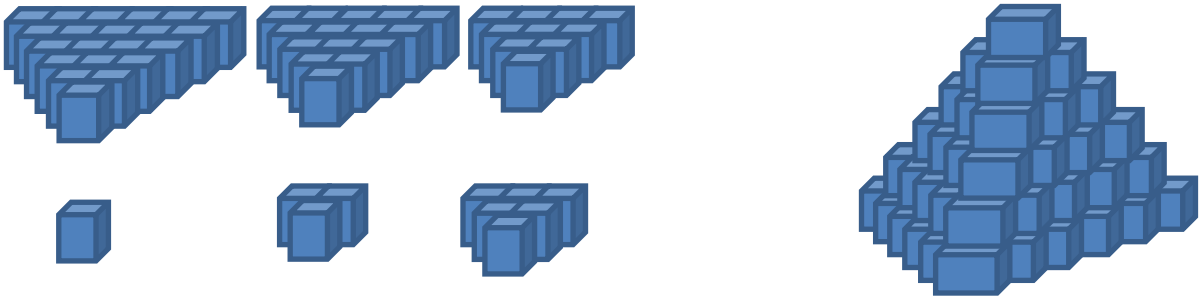


Figure 1. (a) 6 layers of stones, and (b) 6 meters high pyramid.

Example

The total number of stones needed to build a *6* meters high pyramid is = 21 + 15 + 10 + 6 + 3 + 1 = 56. If this pyramid has a *2* meters high secret room, then the total number of stones needed is 56 – 4 = 52.

Input

Each input has two integers, the first represents the pyramid height *n*, and second represents the secret room height *m*. Input is terminated by a sequence having $n = 0$, $m = 0$ which should not be processed.

$1 \le n \le 2000000000$, $0 \le m < n - 2$

Output

For each sequence, you are to output one line, containing the number of stones needed.

Sample Input / Output

rocket.in

```
6 0
6 1
6 2
6 4
100 1
0 0
```

OUTPUT

```
56
55
52
36
46120
```

L. Table Format

Program:	table.(cpp|java|py)
Input:	table.in
Balloon Color:	Silver

Description

We use tables with a number of rows and columns that are filled with text in several documents. Given a table with a fixed number of columns and number of rows, a fixed width, and a single sentence in every cell, your task is to find the minimum number of lines required so that the text fits the table.

For this problem, the number of rows is fixed to 2 and number of columns to 3.

Assume that all characters need the same space, only alphabetical characters are used, no punctuation marks, a single space is used to separate every two words, and no space needed at the beginning or the end of the statement. A word cannot be broken into two lines, if there is no room for the last word in one line, it must be moved into next line. On the other hand, if the first word in a line can fit into the end of previous line, then it must be moved to the previous line.

Example

Given the following six strings:

```
Today is Friday
Good afternoon
Java
GPC
I win this
Good morning Oman
```

If we would like to fit them in a 3x2 table with a overall width of 46 characters, we would need two lines as shown below in Table 1:

Table 1. Text arranged with width = 46, lines = 2

Today is Friday	Good afternoon	Java
GPC	I win this	Good morning Oman

In Table 1 the first column has a width of 15 (Today = 5, is = 2, Friday = 6, and 2 spaces), the second is 14 (Good = 4, afternoon = 9, and 1 space) characters wide, and the third is 17 (Good = 4, morning =7, Oman = 4, and 2 spaces). Therefore, the table is 46 characters wide. If we are given a width of 46, then the minimum number of lines needed is 2, with one line per row.

In Table 2, the width is limited to 45, therefore, one row must be stretched, allowing a sentence to span over two lines:

Table 2. Text arranged with width = 45, lines = 3

Today is Friday	Good afternoon	Java
GPC	I win this	Good morning Oman

In Table 3, with a width of 34, it is impossible to fit the text in 3 lines, therefore minimum number of lines is 4:

Table 3. Text arranged with width = 34, lines = 4

Today is Friday	Good afternoon	Java
GPC	I win this	Good morning Oman

Note that for the same table a width of 29 is sufficient to fit the text in 4 lines (8 + 9 + 12). However, if we would like to bring the word "Good morning Oman" into a single line, we must increase the width of the right column to 17. Similarly, bringing "I win this" into a single line, requires the middle column to have 10 characters, for a total width of 35, as shown in Table 4:

Table 4. Text arranged with width = 35, lines = 3

Today is Friday	Good afternoon	Java
GPC	I win this	Good morning Oman

Input

The input consists of several cases. Every case starts with a positive integer, *k* that represents the table width. Six lines follow, each line contains the statement to fit into one cell. Statements should fill the table in a row-major order (i.e. first row is filled, then the second row, going from left to right). Each statement contains at least one word, and at most 10 words. Input is terminated by having $k = 0$, which should not be processed. $min \leq k \leq 100$, where *min = longest word in column 1 + longest word in column 2 + longest word in column 3. Maximum length of any word = 20.*

Output

For each sequence, you are to output one line, containing the minimum number of lines.

Sample Input / Output

table.in

```
46
Today is Friday
Good afternoon
Java
GPC
I win this
Good morning Oman
45
Today is Friday
Good afternoon
Java
GPC
I win this
Good morning Oman
34
Today is Friday
Good afternoon
Java
GPC
I win this
Good morning Oman
15
Today is Friday
Good afternoon
Java
GPC
I win this
Good morning Oman
50
Today is Friday
The quick brown fox jumps over the lazy dog
International day
The quick brown fox jumps over the lazy dog
Today is a nice day
Good morning Oman
0
```

OUTPUT

```
2
3
4
6
6
```