Problem A:
Predictive Text Input

Source file: `predict.{c | cpp | java}`
Input file: `predict.in`

Smart mobile devices have dramatically changed the way we interact with machines. One of the smart activities of the smart software in those devices is the ability to predict the word that we are going to type. For example, as I just typed only two letters “mo”, the smart software suggested the words “mobile”, “mother”, and “motion”,. In fact, “mobile” is the word that I was going to type.

In this problem, you are asked to mimic the smart device by writing the smart predictive text software. Whenever the user types a word or a part of a word, the program will suggest the top *three* most frequent words from a dictionary of words such that the typed word is a *prefix* of the dictionary word. Prefix of a word is defined as “A substring of the word obtained by removing zero or more contiguous letters from the end of the word”. For example, *h, he, hel, hell,* and *hello* all are prefixes of *hello*. However, *hello* is not a prefix of *hell* because hello is not a substring of hell. Also, *ell* is not a prefix of *hello* because the letters removed (h and o) are not contiguous in hello and *llo* is also not a prefix of *hello* because although the letters removed (h and e) are contiguous, they are not at the end of the word.

Input

The input may contain several test cases. Each test case contains two parts: a *dictionary* and a set of *queries*. The dictionary starts with an integer N ($0 < N < 1000$) in one line denoting the number of *words* in the dictionary. The following lines contain the words. The words will be separated by spaces and spread over one or more lines. Each word shall contain only the lowercase letters (‘a’-‘z’) and shall have at least 1 letter and at most 10 letters. The frequency of the *i*-th word in the dictionary is strictly greater than the *i+1*st word. For example, the frequency of the first word > the frequency of the second word > the frequency of the third word, and so on. Therefore, the last word in the dictionary has the lowest frequency, and the first word has the highest frequency. A word in the dictionary will be listed exactly once.

The query part will contain one word per line and each word shall have at least 1 and at most 10 small letters (‘a’-‘z’). The query part will end with a line containing only the word “###”.

The input will be terminated with the value of N = 0.

Output

For each query word *Q*, you are to output one line. First you should output the query word *Q*, followed by a colon and a space and then the top three words, W1, W2, W3, separated by exactly one space. W1 is the highest frequency word in the dictionary such that *Q* is a prefix of W1. Then W2 is the next highest frequency word having Q as a prefix and W3 is the next highest frequency word having Q has a prefix. In case there are only two words in the dictionary that have Q as prefix, you should output those two words. If there is only one word in the dictionary having Q has prefix, you should output only that word. If there is no word in the dictionary that has Q as prefix, then you should output the query word itself.

Sample Input

```
34
gulf programming competition is my
favorite it the best
held every year
i come here good luck this
world finals first prize gold medal
and then gala medal important
company sponsors commends all
winners
g
i
com
pr
gol
sing
###
3
american university dubai
am
d
x
###
0
```

Output for Sample Input

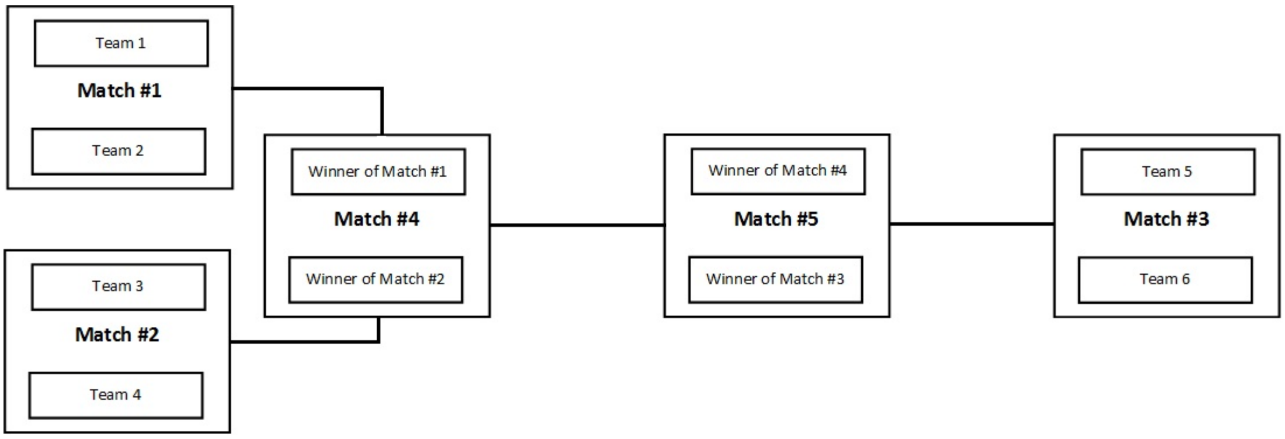
```
g: gulf good gold
i: is it i
com: competition come company
pr: programming prize
gol: gold
sing: sing
am: american
d: dubai
x: x
```

Problem B: The League

Source file: `league.{c | cpp | java}`
Input file: `league.in`

In a knockout stage of a football league, the loser of a match is out of the league and the winner advances to play with another winner of a different match. This method is adopted in the FIFA World Cup (which Germany recently won over Argentina) and many other leagues around the world. There are no draws in this stage, the game continues until a winner is decided (whether in the extra time or by shots from the penalty mark).

For the purposes of this problem, a league can contain any number of teams. Your task is to organize the matches between the teams to decide a winner while minimizing the number of matches. For instance, if a league has 6 teams, the figure below shows one way to organize the matches resulting in 5 matches to decide a winner of the league.



Given the number of teams in a league, your task is to determine the minimum number of matches required to decide a winner.

Input

The first line of the input file contains an integer N ($0 < N < 10{,}000$) indicating the number of test cases. Each test case consists of one line containing an integer T ($0 < T \leq 10^9$) representing the number of teams in the league.

Output

The output for each test case is in this form:

k. M

where ***k*** represents the test case number (starting at 1) followed by a single space, and ***M*** is the minimum number of matches required.

Sample Input

```
2
6
8
```

Output for Sample Input

```
1. 5
2. 7
```

Problem C:
Exam Graph

Source file: `graph.{c | cpp | java}`
Input file: `graph.in`

Your University has been "torturing" students for some years now, scheduling multiple final exams on the same day, as a proper scheduling algorithm has not been designed yet.

The registrar has been offering the raw data of student registrations in the form of (studentID, courseID) pairs, in the hope that one of the wiz student programmers will pick up the challenge and find a solution.

The problem is that these data cannot be used in this form. One has to first create the matrix that corresponds to the (non-weighted) graph where each node represents a course, and each edge connects courses taken together by at least one student.

Input

The input is made of several test cases. Each test case starts with a number *P* which is the number of (studentID, courseID) pairs. *P* lines follow, with the studentID and courseID separated by a single space. The input ends with a test case with *P*=0. The studentID and courseID identifiers are both strings, each having a maximum of 10 characters. The strings are case sensitive, so a course identified as CMP100 is different than cmp100.

Output

For each test case output the adjacency matrix describing the graph. The weights should be separated by a single space, with no trailing spaces at the end of the line. Test case outputs should be separated by a single empty line.
In the construction of the matrix, you should order the courses in ascending alphanumerical order.

Sample Input

```
5
10001 CMP100
10001 CMP101
10002 CMP100
10002 CMP101
10003 CMP102
0
```

Output for Sample Input

```
0 1 0
1 0 0
0 0 0
```

Problem D:
Pisano Periods

Source file: `pisano.{c | cpp | java}`
Input file: `pisano.in`

The Fibonacci sequence is a well-known (and sometimes detested by students challenged to grasp recursion) numerical sequence:

0,1,1,2,3,5,8,13,21,...

What is less known about the sequence, is that if it is divided by an arbitrary number *N*, the sequence of remainders follows a pattern with a period *T* that depends on *N*. This period is called the Pisano period. For example, if *N*=4 we have *T*=6:

You task is to find the period of the sequence of remainders, given a number *N*.

Input

The input is made of a set of test cases for different numbers $1<N<=10^6$. Each *N* resides in a different line. The input ends with a test case for *N*=0.

Output

For each *N* you should output the corresponding Pisano period in a separate line.

Sample Input

```
4
5
6
0
```

Output for Sample Input

```
6
20
24
```

Problem E:
Rank

Source file: `rank{.c| .cpp| .java}`
Input file: `rank.in`

In ACM programming contests, teams are ranked according to the following set of rules.

1. Teams are ranked in a descending order according to the number of problems they solve
2. Teams who solve the same number of problems are ranked in an ascending order by total time
3. The total time (aka score) is the sum of the time consumed for each problem solved
4. Time consumed for a solved problem is the time elapsed from the beginning of contest to the submittal of the accepted run; plus 20 penalty minutes for every rejected run (for that same problem)
5. There is no time consumed for a problem that is not solved (even if there are rejected runs for it)

The table below shows the submissions of team "Code Producers". Their score is calculated as:

I. For solving C correctly, their score is set to 60.

II. For solving A correctly, from the second run, their score is incremented by (120 + 20 = 140).

Id	Problem	Team	Time	Status
1	C	Code_Producers	60	Correct
2	A	Code_Producers	100	Incorrect
3	A	Code_Producers	120	Correct
4	B	Code_Producers	200	Incorrect

The total score for Code Producers is equal to 60 + 120 + 20 = 200. Assume that there are other teams who solved two problems: Test Team 1 (scored 160) and Test Team 2 (scored 220). Code Producers are ranked between Test Team 1 and Test Team 2.

Write a program that rank teams, based on their submissions.

Input

The input starts with a line with two integers $2 \leq N \leq 100$, which is the number of teams to be ranked, and $1 \leq M \leq 26$, which is the number of problems. Each of the remaining lines of the input file represents a team submission. Each line starts with an integer *RI*, which is the run id, followed by a character *P* (from 'A' to 'Z'), which is the problem submitted, followed by an integer *U*, which is the team number $1 \leq U \leq N$, followed by an integer *T*, which is the time of submission, and finally a character *C*, which takes one of two values, 'c' for correct submission or 'i' for incorrect submissions.

Output

You should output the ranking of the teams in ascending order, where each line shows the team rank, team number, number of correct submissions and total time score. *Ties are resolved based on team number*. That is, teams are ranked in ascending order according to their team number. If team number 1 and 2 are tied, then team number 1 comes before team number 2.

Sample Input

```
3 4
1 D 1 15 i
2 D 3 20 c
3 D 1 25 c
4 B 2 20 c
5 A 3 30 i
6 A 1 40 c
7 A 3 60 c
8 A 2 100 c
9 C 1 120 c
10 B 3 140 i
```

Output for Sample Input

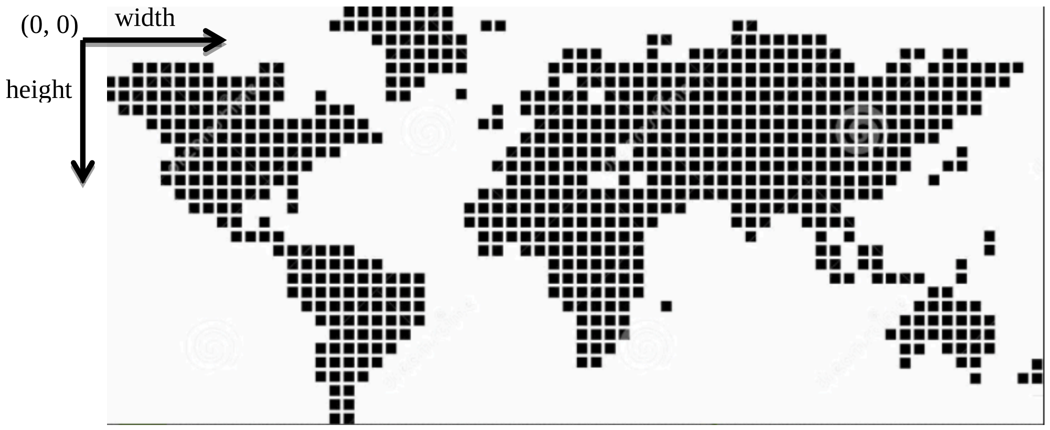
```
1 1 3 205
2 3 2 100
3 2 2 120
```

Problem F: Impact

Source file: `impact{.c| .cpp| .java}`
Input file: `impact.in`

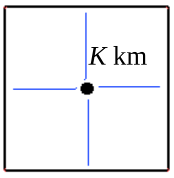
An asteroid is heading to Earth, and an impact is unavoidable. You are given the coordinates of ground zero (point where the asteroid will hit Earth) and you are asked to produce a report of the asteroid destruction. Your program is expected to read the world map as follows.

1. Each country is modeled as a set of regions, where regions maybe disconnected. Regions may not overlap.
2. Each region is described using area and population, where area is represented as a rectangle (you will be given the coordinate for the top-left corner, width and height, both in km)
3. The top-left corner of the world map is point (0, 0), whereas the bottom-right corner is the point (40075, 40008)



The program should report the casualties of the asteroid. The report should follow the guide bellow.

1. The report should have one line per affected country, where the country, which is about to receive the direct hit, should appear first. The rest should appear in alphabetical order.
2. Each line should specify the number of regions affected, and anticipated death toll.
3. The asteroid affects an area up to *K* km (will be given in input) to the north of the impact point, *K* to the south, *K* to the east and *K* to the west.
4. Any region, which overlaps with the asteroid area, is considered affected, where the death toll is computed based on the percentage of overlap. For example, if a region has a population of *P*, and *V* percent of its area overlaps with the asteroid area, then *V* percent of *P* will be dead.



5. If the asteroid does not affect a country, it should not appear in the report.
6. If a country *C* is going to lose all of its population, add the following line to the report:

   `C is wiped out`.

7. If the asteroid affects no country, the report should print:

   `No casualties!`

Input

The input consists of one test case. The input starts with an integer $M \leq 100000$, representing the number of regions. Each of the M lines describes a region. Each region-line starts with a string S, which is the name of the country that the region belongs to; followed by two integers x and y, representing the coordinates of the top left corner of the region, followed two integers w and h, representing the width and height of the region, and finally an integer p, which is the population of that region. The last line consists of three integers, which specify the coordinates of the impact x y, and the destruction parameter K.

Output

For each affected country, print a line that starts with the name of that country, followed by the number of regions affected and the death toll. The death toll should always be rounded down to nearest integer.

Sample Input

```
7
Botswana 20000 30000 300 300 1000000
South_Africa 21000 30700 500 300 5000000
South_Africa 21000 31001 1000 900 20000000
Botswana 20000 30300 400 400 1000000
Canada 0 0 3000 1000 10000000
Canada 0 2000 2000 1000 10000000
Canada 0 3000 2000 1000 10000000
20000 29500 1500
```

Output for Sample Input

```
Botswana 2 2000000
Botswana is wiped out
South Africa 1 5000000
```

Problem G:
Passengers Relocation

Source file: `passenger{.c| .cpp| .java}`
Input file: `passenger.in`

A number of passengers are to be moved to a destination. At your disposal you have different vehicle types, each with a different seating capacity and round-trip time. Your task is to find the minimum time that will be needed for the trip and the minimum associated cost. The cost for a vehicle trip is equal to the number of passengers it can accommodate (i.e. the capacity not the actual passengers moved). The seating capacity excludes the car driver.

We assume that vehicles start from the destination point, and they have to spend the round trip time for every trip. Time is the main constraint; the solution providing the smallest time should be found, regardless of the cost. Only if two solutions provide the same time, you should choose the one with the lower cost.

You can have multiple cars with the same number of passengers, and the same round trip time, or different round trip time. All numbers including inputs and outputs are integers.

Example 1

Total Passengers	Vehicle Type	# of passengers	Cost	RTT
10	Car 1	2	2	2
	Car 2	4	4	4

Solution: the minimum time is 6 units with the cost of 10
Car 1: 3 rounds, and car 2 1 round.

Example 2

Total Passengers	Vehicle Type	# of passengers	Cost	RTT
10	Car 1	2	2	4
	Car 2	4	4	2

Solution: the minimum time is 4 units with the cost of 10
Car 1: 1 round, car 2: 2 rounds.

Example 3

Total Passengers	Vehicle Type	# of passengers	Cost	RTT
10	Car 1	1	1	2
	Car 2	2	2	4
	Car 3	3	3	3
	Car 4	4	4	1

Solution: the minimum time is 3 units with the cost of 11
Car 4: 2 rounds, car 3: 1 rounds (only 2 passengers).

Example 4

Total Passengers	Vehicle Type	# of passengers	Cost	RTT
22	Car 1	7	7	6
	Car 2	5	5	5
	Car 3	4	4	3

Solution: the minimum time is 9 units with the cost of 24

Car 1: 1 rounds, car 2: 1 round, and car 3: 3 rounds (only 2 passengers in the last round).

Input

The input file will consist of several problem instances. The first line of the file will include a single integer that represents the number of instances *n*. Each problem instance starts with a line containing two positive numbers: the number of passengers *p*, followed by the number of vehicles *v*. *v* lines follow, each containing two positive integers: the first is the number of passengers a car can accommodate *c* (and cost) and the round trip time *t*. The problem variables satisfy the following conditions:

$1 \leq n \leq 100$
$1 \leq p \leq 100000$
$2 \leq v \leq 100$
$1 \leq c \leq 50$
$1 \leq t \leq 100$

Output

For each problem instance, your program should output a single line consisting of two numbers: the minimum time followed by a single space, followed by the total cost. An empty line should separate successive outputs.

Sample Input

```
4
10 2
2 2
4 4

10 2
2 4
4 2

10 4
1 2
2 4
3 3
4 1

22 3
7 6
5 5
4 3
```

Output for Sample Input

```
6 10
4 10
3 11
9 24
```

Problem H:
Hidden Number Pattern

Source file: `pattern{.c| .cpp| .java}`
Input file: `pattern.in`

Given a set of 1 digit numbers, you are required to find if any digit is hidden by forming its own pattern within a 5x3 matrix. You will have to insert the numbers into a two N x M matrix and detect and count how many times every number is hidden. Overlapping is allowed.

0	0	0
0		0
0		0
0		0
0	0	0

	1	
1	1	
	1	
	1	
1	1	1

2	2	2
		2
2	2	2
2		
2	2	2

3	3	3
		3
3	3	3
		3
3	3	3

4		4
4		4
4	4	4
		4
		4

5	5	5
5		
5	5	5
		5
5	5	5

6	6	6
6		
6	6	6
6		6
6	6	6

7	7	7
		7
		7
		7
		7

8	8	8
8		8
8	8	8
8		8
8	8	8

9	9	9
9		9
9	9	9
		9
9	9	9

Example
Input: 5x5 matrix 4511251119311388317801117
Solution: fill 5x5 matrix below then you will detect the pattern 1.

4	5	1	1	2
5	1	1	1	9
3	1	1	3	8
8	3	1	7	8
0	1	1	1	7

Input

The input file will consist of several problem instances. The first line of the file will include a single integer that represents the number of instances *k*. Each instance starts with a line containing two positive numbers that represent the matrix dimensions *n* x *m*, where *n* is the number of rows ($5 \leq n \leq 1000$) and *m* ($3 \leq m \leq 1000$) is the number of columns. The contents of the matrix follow in the form of *n x m* single digit numbers, *m* digits per line.

Output

For each problem instance, your program should output a single line consisting of 10 numbers that represents the count for every pattern starting from 0, ending with 9, and separated by space (no space after the last one).

Sample Input

```
2
5 5
45112
51119
31138
83178
01117
7 6
666010
666110
666010
666010
666111
666666
333333
```

Output for Sample Input

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

Problem I: ***Antibiotics***

Source file: `antibiotics.{c | cpp | java}`
Input file: `antibiotics.in`

Antibiotic misuse, sometimes called antibiotic abuse or antibiotic overuse, refers to the misuse or overuse of antibiotics, with potentially serious effects on health. It is a contributing factor to the development of antibiotic resistance, including the creation of multidrug-resistant bacteria, informally called "super bugs": relatively harmless bacteria can develop resistance to multiple antibiotics and cause life-threatening infections [1].

The health authority would like to implement a new system where all antibiotics prescriptions are logged to a central database. The main purpose of this is to limit the use of antibiotics in general as well as limit their use per person. Antibiotics are given in courses of a varying number of days each.

In this problem, you are required to write a program to help the health authority figure out if any patient has been prescribed a number of overlapping antibiotic courses, given the database of patients and the start and end day of their antibiotic courses.

Input

The first line of the input file contains an integer N indicating the number of test cases. Each test case represents a database and starts with an integer C ($0 < C \leq 100{,}000$) describing the number of antibiotic courses in the database. C lines follow containing three integers each P, S, E ($0 < P \leq 1{,}000$), ($0 < S < E \leq 10^8$) where P is the patient ID number and S, and E are the start day and the end day of the antibiotic course, respectively.

Output

For each test case, your program should print out a line for each patient with overlapping antibiotic courses sorted in an ascending order of the patient IDs. Each line should be in the below format:

P A

where ***P*** represents the patient ID and ***A*** is the maximum number of overlapping antibiotic courses for this patient. If no patient has overlapping antibiotic courses, print “All Clear” without the quotes instead. Print an empty line after each test case.

Sample Input

```
3
4
1 1 5
1 4 7
1 5 9
2 2 10
4
1 1 2
1 3 4
2 1 2
2 3 4
3
2 1 2
2 2 3
2 10 17
```

Output for Sample Input

```
1 2

All Clear

2 2
```

[1] Wikipedia

Problem J:
Dance (a.k.a The Circles of Evil)

Source file: `dance.{c | cpp | java}`
Input file: `dance.in`

In an effort to preserve energy, a new dance floor has been designed with a layer of energy sensors underneath it. The purpose of the new design is to use the weights and movements of the dancers to store energy in batteries that are connected to the sensors. The sensors can collect energy from dancers within a circle of a specific radius.

In this problem, you will be given the locations of the energy sensors and the locations of the dancers on the dance floor. If the dancer is within the range of only one energy sensor, he or she contributes an energy value calculated as $K \times Ei$ where *K* is a constant for the dance floor, and *Ei* is the expected energy factor for dancer *i*. If the dancer is within the range of more than one sensor, the energy contribution for each sensor is calculated as $\frac{K \times Ei}{N}$ where *N* is the number of energy sensors dancer *i* is in the range of. Your task is to calculate the total energy contribution of all dancers on the dance floor.

Input

The first line of the input file contains an integer *N* indicating the number of test cases. The first line of each test case contains three integers *D*, *S*, and *K* ($0 < D, S, K \leq 1{,}000$) where *D* and S represent the number of dancers and sensors, respectively, and *K* is a constant as described above. *D* lines follow with three integers each. The first two integers represent the *X* and *Y* coordinates of a dancer, and the third integer is the expected energy factor of the dancer *E* ($0 < E \leq 10{,}000$). S lines follow with three integers each. The first two integers represent the *X* and *Y* coordinates of a sensor, and the third integer is the range of the sensor *R* ($0 < R \leq 1{,}000$). All *X* and *Y* coordinates have an absolute maximum of 10^8. Multiple dancers can be located in the same coordinates, and also multiple sensors can be in the same coordinates.

Output

The output for each test case is in this form:

k. E

where ***k*** represents the test case number (starting at 1) followed by a single space, and ***E*** is the total energy contribution with an accuracy of two decimal points.

Sample Input

```
2
3 3 10
1 1 10
1 -2 10
5 5 10
0 1 2
0 -2 2
-5 -5 2
2 2 10
1 1 10
-1 -1 10
0 0 5
0 1 5
```

Output for Sample Input

```
1. 200.00
2. 200.00
```

Problem K: ***Elevate the Utilization!***

Source file: `elevate.{c | cpp | java}`
Input file: `elevate.in`

As electricity prices are soaring, there must be a way to reduce wastage by increasing the utilization of heavy duty machines, such as elevators. We formulate the problem as follows. There are *N* persons ($p_1,\ldots,p_N$) waiting in a queue to get into an elevator. By the way, there is only one elevator, and it travels between the *ground* floor to the *sky* floor (i.e., there are no other stoppages in between). Also, all persons are waiting at the ground floor. Just for simplicity, we are concerned only in the utilization of the ground-to-sky journey of the elevator.

Each person p_i has a positive weight w_i. The weight capacity of the elevator is M. Therefore, all persons may not be carried together. They are carried in several trips. Each person gets into the elevator according to his order in the queue. Meaning, p_1 gets in first, then p_2 and so on. Suppose at the *t*-th trip, persons p_j through p_k gets into the elevator. The *waste factor* (WF) of the trip *t* is defined as:

$$WF[t] = M - \sum_{i=j}^{k}(w_i)$$

Your task is to minimize the *total waste* (TW), which is defined as:

$$TW = \sum_{t=1}^{S-1}(WF[t])^3$$

Where *S* is the total number of trips required to carry all the people from the ground to the sky floor. Note that the WF of the last trip is not included in the TW calculation.

Input

The input may contain several cases. Each case starts with two positive integers M ($0 < M < 1000$) and N ($0 < N <= 1000$) on a single line, separated by a single space, denoting the weight capacity of the elevator, and number of persons, respectively. Following lines contain N positive integers separated by spaces and/or newline. The first integer denotes w_1, the weight of person p_1 in the queue, the second one w_2, being the weight of person p_2, and so on. Note that $0 < w_i <= 100$ and $w_i < M$, for all w_i. The input is terminated by the value 0 for M and N.

Output

For each case, you are to output one line containing the case number, followed by a colon and a space, followed by the minimum TW as described above. Note that re-ordering of the persons in the queue is not allowed, i.e., the persons must be boarded into the elevator exactly in the same order given in the input.

Sample Input

```
20 7
10 4 6 7
6 9 4
0 0
```

Output for Sample Input

```
1: 217
```