Problem A
ACM Contest Scoring
Time limit: 1 second

Our new contest submission system keeps a chronological log of all submissions made by each team during the contest. With each entry, it records the number of minutes into the competition at which the submission was received, the letter that identifies the relevant contest problem, and the result of testing the submission (designated for the sake of this problem simply as `right` or `wrong`). As an example, the following is a hypothetical log for a particular team:

```
3 E right
10 A wrong
30 C wrong
50 B wrong
100 A wrong
200 A right
250 C wrong
300 D right
```

The rank of a team relative to others is determined by a primary and secondary scoring measure calculated from the submission data. The primary measure is the number of problems that were solved. The secondary measure is based on a combination of time and penalties. Specifically, a team's time score is equal to the sum of those submission times that resulted in `right` answers, plus a 20-minute penalty for each wrong submission of a problem that is ultimately solved. If no problems are solved, the time measure is 0.

In the above example, we see that this team successfully completed three problems: `E` on their first attempt (3 minutes into the contest); `A` on their third attempt at that problem (200 minutes into the contest); and `D` on their first attempt at that problem (300 minutes into the contest). This team's time score (including penalties) is 543. This is computed to include 3 minutes for solving `E`, 200 minutes for solving `A` with an additional 40 penalty minutes for two earlier mistakes on that problem, and finally 300 minutes for solving `D`. Note that the team also attempted problems `B` and `C`, but were never successful in solving those problems, and thus received no penalties for those attempts.

According to contest rules, after a team solves a particular problem, any further submissions of the same problem are ignored (and thus omitted from the log). Because times are discretized to whole minutes, there may be more than one submission showing the same number of minutes. In particular there could be more than one submission of the same problem in the same minute, but they are chronological, so only the last entry could possibly be correct. As a second example, consider the following submission log:

```
7 H right
15 B wrong
30 E wrong
35 E right
80 B wrong
80 B right
100 D wrong
100 C wrong
300 C right
300 D wrong
```

This team solved 4 problems, and their total time score (including penalties) is 502, with 7 minutes for `H`, $35 + 20$ for `E`, $80 + 40$ for `B`, and $300 + 20$ for `C`.

Input

The input contains n lines for $0 \leq n \leq 100$, with each line describing a particular log entry. A log entry has three parts: an integer m, with $1 \leq m \leq 300$, designating the number of minutes at which a submission was received, an uppercase letter designating the problem, and either the word `right` or `wrong`. The integers will be in nondecreasing order and may contain repeats. After all the log entries is a line containing just the number -1.

Output

Output two integers on a single line: the number of problems solved and the total time measure (including penalties).

Sample Input 1

```
3 E right
10 A wrong
30 C wrong
50 B wrong
100 A wrong
200 A right
250 C wrong
300 D right
-1
```

Sample Output 1

```
3 543
```

Sample Input 2

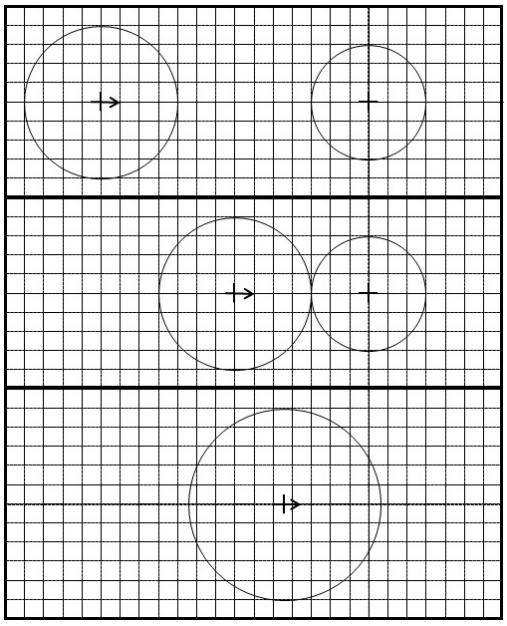
```
7 H right
15 B wrong
30 E wrong
35 E right
80 B wrong
80 B right
100 D wrong
100 C wrong
300 C right
300 D wrong
-1
```

Sample Output 2

```
4 502
```

Problem B
The Agglomerator
Time limit: 6 seconds

In this problem, we are simulating the dynamics of moving objects. The objects are droplets that are modeled in two dimensions as circles of various sizes, each moving at a constant velocity. When two circles touch they combine (i.e., agglomerate) into a single circular droplet with an area equal to the sum of the areas of the two combining droplets. The newly formed droplet's position is the *area-weighted* average position of the two droplets at the time of contact and its velocity is the *area-weighted* average velocity of the two circles. (See the following example.)



The figure to the right illustrates the process of agglomeration. In the top panel of that figure, we see the leftmost droplet with radius 4 centered at position $(-14, 0)$ and with velocity $(1, 0)$ moving toward a stationary droplet of radius 3 centered at the origin. The two droplets make contact at time $t = 7.0$ as shown in the middle panel of the figure.

The droplet with radius 4 is centered at $(-7, 0)$ at the time that the two droplets agglomerate into a single new droplet. The two original droplets have areas 16π and 9π, respectively, and thus the new droplet has area 25π and thus radius 5. The x-coordinate of the aggolomerated droplet is equal to $\frac{16}{25} \cdot (-7.0) + \frac{9}{25} \cdot 0.0 = -4.48$. The y-coordinate is $\frac{16}{25} \cdot 0.0 + \frac{9}{25} \cdot 0.0 = 0.0$. By similar calculations, the velocity of the aggolomeration is $(0.64, 0)$.

Given an initial configuration of droplets, your goal is to simulate their motion until reaching the final time at which an agglomeration occurs (if any). All test sets have been crafted to assure that:

- The original droplets do not touch each other.
- When a new droplet is formed from an agglomeration, the new droplet will not immediately be touching any other droplets. (In fact, it will be at least 0.001 away from any other droplets.)
- No two droplets will ever pass each other with only a single point of intersection. (In fact, changing the radius of any drop by ± 0.001 will not effect whether it collides with another.)
- No two pairs of droplets will ever collide at precisely the same time. (In fact, all agglomerations will be separated in time by at least 0.001.)
- No agglomerations will occur beyond time $t = 10^9$.

Input

The input consists of a description of the original configuration. The first line contains the original number of droplets, $2 \leq N \leq 100$. This is followed by N lines of data, each containing five integers, x, y, v_x, v_y, r, respectively specifying the x-coordinate of the center, the y-coordinate of the center, the x-component of the velocity, the y-component of the velocity, and the radius. These quantities are bounded such that $-10\,000 \leq x, y, v_x, v_y \leq 10\,000$ and $1 \leq r \leq 100$.

Output

Output a single line with two values k and t, where k is the number of droplets in the final configuration and t is the time at which the final agglomeration occurred. If a data set results in no agglomerations, k will be the original number of droplets and 0 should be reported as the time. The value of time should be reported to have an absolute or relative error of no more than 10^{-3}.

Sample Input 1

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

Sample Output 1

```
1 1.0
```

Sample Input 2

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

Sample Output 2

```
2 0.0
```

Sample Input 3

```
4
-8 0 2 -2 2
0 -8 -2 2 2
2 8 0 -4 3
8 2 -4 0 3
```

Sample Output 3

```
1 2.0
```

Sample Input 4

```
4
-8 3 3 0 2
-2 4 2 0 2
2 -2 2 0 2
8 -2 0 0 2
```

Sample Output 4

```
1 5.78474956
```

Problem C
Dance Recital
Time limit: 1 second

The Production Manager of a dance company has been tasked with determining the cost for the seasonal dance recital. Because of their exceptional skills, many dancers will perform in more than one routine, but this presents a problem; each dance routine incorporates a unique costume, so between routines, dancers must report backstage to a Wardrobe Specialist, who can change the dancer's costume in time to begin their next scheduled routine.

A Wardrobe Specialist does a *normal change* on a dancer when the dancer performs in two routines that are not consecutive, but when a dancer is required to perform in two consecutive routines, a *quick change* is necessary. A Wardrobe Specialist charges a flat rate per recital that covers all normal changes, but charges an exorbitant amount for each quick change. The Production Manager is responsible for keeping the show under budget, and has hired you to write a program to report the minimum number of quick changes needed for a given recital, given that the order of the dance routines could be changed.

To describe the cast of dancers that are to perform during a recital, each dancer is assigned an identifying uppercase letter. (Fortunately, there are never more than 26 dancers, so characters from `A` to `Z` suffice.) To describe a full recital, a list of individual routines is given, with a string of characters defining which dancers appear in a routine. For example, consider the following recital description:

```
ABC
ABEF
DEF
ABCDE
FGH
```

The above list describes a recital with 5 dance routines, including a total of 8 individual performers (dancers `A` through `H`). The first routine listed includes dancers {`A`, `B`, and `C`}. The second routine includes dancers {`A`, `B`, `E`, and `F`}. Notice that if these first two routines are performed in the above order, dancers `A` and `B` will require a quick change between the routines. In fact, if these five routines are scheduled in the order given above, a total of six quick changes are required. However, the schedule can be rearranged as follows:

```
ABEF
DEF
ABC
FGH
ABCDE
```

In this case, only two quick changes are required (those for `E` and `F` between the first two dances).

Input

The first line contains a single integer R, with $2 \leq R \leq 10$, that indicates the number of routines in the recital. Following that will be R additional lines, each describing the dancers for one routine in the form of a nonempty string of up to 26 non-repeating, lexicographically sorted uppercase alphabetic characters identifying the dancers who perform in that routine. Although a dancer's letter will not appear more than once in a single routine, that dancer may appear in many different routines, and it may be that two or more routines have the identical set of dancers.

Output

Output a single integer designating the minimum number of quick changes required for the recital.

Sample Input 1

```
5
ABC
ABEF
DEF
ABCDE
FGH
```

Sample Output 1

```
2
```

Sample Input 2

```
6
BDE
FGH
DEF
ABC
BDE
ABEF
```

Sample Output 2

```
3
```

Sample Input 3

```
4
XYZ
XYZ
ABYZ
Z
```

Sample Output 3

```
4
```

Problem D
Hidden Password
Time limit: 1 second

Insecure Inc. has decided to shift directions after a failed attempt at developing a new encryption standard. Their new effort is a password system used to hide a password inside another string of characters we denote as a *message*. However, it is important that the message has a certain property relative to the hidden password.

Let us assume that we denote the characters of the password as $c_1 c_2 \ldots c_P$ (although those characters need not be distinct). To be a valid message for the password, if you start from the beginning of the message and search for any character from the set $\{c_1, \ldots, c_P\}$, it must be that c_1 is the first that you find. Subsequently, if you continue looking from that point of the message for any character from the set $\{c_2, \ldots, c_P\}$, it must be that c_2 is the next that you find. Continuing in that manner, c_3 must be the next character from the set $\{c_3, \ldots, c_P\}$, and so on until reaching c_P.

For example, if the password is `ABC`, then the string `HAPPYBIRTHDAYCACEY` is a valid message.

- Notice that `A` is the first of the set {`A`, `B`, `C`} to appear in the message. (The initial `H` is not relevant.)
- Following the `A` that was found, the next occurrence from the set {`B`, `C`} is `B`.
- Following the `B` that was found, the next occurrence from the set {`C`} is indeed `C`.
  (Note that the `A` in `DAY` is not relevant, since we are only looking for a `C` at this point, and the additional `A` and `C` in `CACEY` are not relevant, because we have already completed the password with the first `C`.)

However, for the password `ABC`, the string `TRAGICBIRTHDAYCACEY` is not a valid message.

- While the `A` is the first of the set {`A`, `B`, `C`} to appear in the string,
  the next occurrence from the set {`B`, `C`} is `C` rather than `B`.

Also, the string `HAPPYBIRTHDAY` is not a valid message for the password `ABC` because the `C` never appears.

As an example with duplicate letters in the password, consider the password `SECRET`. For this password, the string `SOMECHORESARETOUGH` is a valid message. In contrast, the string `SOMECHEERSARETOUGH` is not a valid message, because an extraneous `E` is found at the point when an `R` is first expected.

Input

The input consists of a single line containing two strings. The first string is the password, having length P, with $3 \leq P \leq 8$. The second string has length S, with $10 \leq S \leq 40$. Both strings will consist solely of uppercase letters. (That is, neither string can include whitespace, lowercase letters, digits, or other special characters.)

Output

Output a single line with the word `PASS` if the second string is a valid message for the password, or `FAIL` otherwise.

Sample Input 1	Sample Output 1
ABC HAPPYBIRTHDAYCACEY	PASS

Sample Input 2	Sample Output 2
ABC TRAGICBIRTHDAYCACEY	FAIL

Sample Input 3	Sample Output 3
ABC HAPPYBIRTHDAY	FAIL

Sample Input 4	Sample Output 4
SECRET SOMECHORESARETOUGH	PASS

Sample Input 5	Sample Output 5
SECRET SOMECHEERSARETOUGH	FAIL

Problem E
Kitchen Measurements
Time limit: 21 seconds

You are making a recipe and need to measure a precise volume of liquid. There are an assortment of cups of varying volumes in your kitchen, however no cup has any markings on it other than to indicate its total volume, and none of them match the volume that you want. You start with the biggest cup full of liquid and, to make sure you know precisely how much volume you are working with at any point in time, you consider steps in which you pour from any nonempty cup into another cup, always pouring until either the cup you are pouring into becomes full, or the cup you are pouring from becomes empty (whichever occurs first). As a simple example, assume you start with a full cup having capacity 5, and you have another cup with capacity 2, but your goal is to have 3 units of the liquid in the largest cup. In this case, you can start pouring from the larger cup to the smaller, stopping when the smaller one reaches its capacity of 2. This will leave precisely 3 units in the larger cup. See Figure E.1(a).

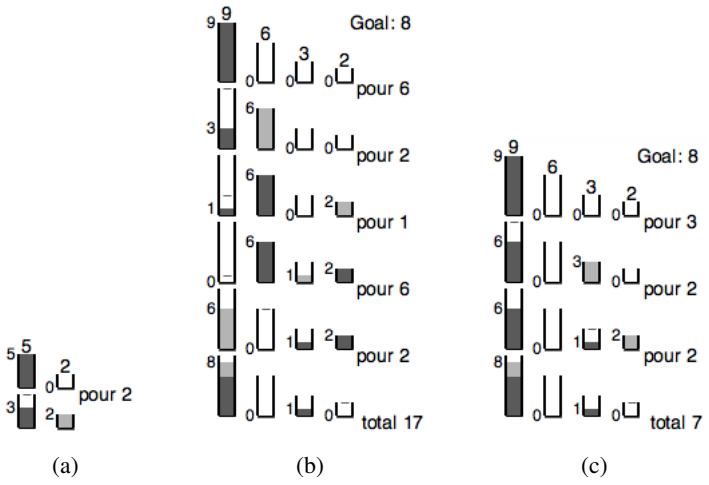
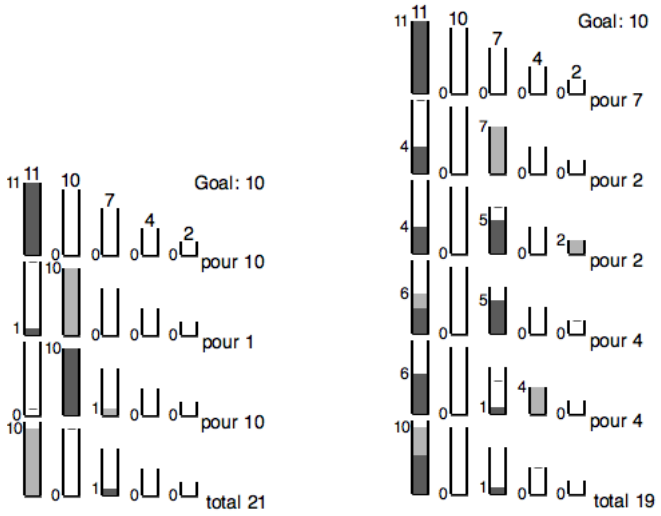


Figure E.1: Pouring between cups

As another example, consider a case in which you have 4 cups with capacities 9, 6, 3, and 2, and you start with the largest cup full, the rest empty, and a goal of ending with 8 units in the largest cup. For ease of discussion we will refer to the cup with capacity 9 as the "9-cup" and similarly for the other sizes. You notice that the 6-cup and 2-cup have combined capacity of 8, and so you could pour from the original 9-cup to fill those two cups, then dump the remaining 1 unit from the 9-cup into the 3-cup, and finally pour the full 6-cup and 2-cup back into the 9-cup. See Figure E.1(b). In implementing this strategy, the total volume of liquid poured would be $6 + 2 + 1 + 6 + 2 = 17$. You could achieve this goal in another way: pour 3 units from the 9-cup to the 3-cup (leaving 6 units in the 9-cup), then fill the 2-cup from the 3-cup (leaving 1 unit in the 2 cup), and finally pour the full 2-cup back into the 9-cup, resulting in exactly 8 units in that cup. With this strategy, the total volume poured is only $3 + 2 + 2 = 7$. See Figure E.1(c).

As a final example, you start with cups of capacities 11, 10, 7, 4, and 2, with the 11-cup full, and a goal of ending up with 10 units in the 11-cup. Obviously, you could fill the 10-cup, dump the remaining 1 unit into another cup, and then pour from the full 10-cup back into the 11-cup, as illustrated in Figure E.2(a). These three pours would mean transferring a total volume of $10 + 1 + 10 = 21$. Figure E.2(b) shows a sequence with more steps, but less liquid poured.



(a) (b)

Figure E.2: More pouring

Input

The input consists of a single line of positive integers: $n\ c_1\ c_2\ \ldots\ c_n\ V$, where there are n cups, with $2 \leq n \leq 5$, having capacities satisfying $64 \geq c_1 > c_2 > \ldots > c_n \geq 1$. The value $V < c_1$ designates the desired volume. You must start with largest cup (that with capacity c_1) full of liquid and the other cups empty, and the goal is to get exactly volume V into the *largest* cup.

Output

Output the minimum amount of liquid that must be poured to achieve the goal, or output `impossible` if the goal cannot be achieved.

Sample Input 1	Sample Output 1
2 5 2 3	2

Sample Input 2	Sample Output 2
4 9 6 3 2 8	7

Sample Input 3	Sample Output 3
5 11 10 7 4 2 10	19

Sample Input 4	Sample Output 4
2 5 2 4	impossible

Sample Input 5	Sample Output 5
5 64 45 41 28 2 63	121

Problem F
Line Them Up
Time limit: 1 second

An eccentric coach asks players on the team to line up alphabetically at the start of practice. The coach does not tell the players whether they need to line up in increasing or decreasing order, so they guess. If they guess wrong, the coach makes them run laps before practice. Given a list of names, you are to determine if the list is in increasing alphabetical order, decreasing alphabetical order or neither.

Input

The input consists of a single test case. The first line will contain the number N of people on the team ($2 \leq N \leq 20$). Following that are N lines, each containing the name of one person. A name will be at least two characters and at most 12 characters in length and will consist only of capital letters, and with no white spaces (sorry BILLY BOB and MARY JOE). Duplicates names will not be allowed on a team.

Output

Output a single word: `INCREASING` if the list is in increasing alphabetical order, `DECREASING` if it is in decreasing alphabetical order, and otherwise `NEITHER`.

Sample Input 1

```
5
JOE
BOB
ANDY
AL
ADAM
```

Sample Output 1

```
DECREASING
```

Sample Input 2

```
11
HOPE
ALI
BECKY
JULIE
MEGHAN
LAUREN
MORGAN
CARLI
MEGAN
ALEX
TOBIN
```

Sample Output 2

```
NEITHER
```

Sample Input 3	Sample Output 3
4 GEORGE JOHN PAUL RINGO	INCREASING

Problem G
Mosaic
Time limit: 1 second

In this problem, we consider a special class of tile mosaics, as exemplified in Figure G.1. Each such mosaic is built on a rectangular grid with a white background. Within each cell of the grid is either a square black tile, a triangular black tile in one of the four orientations shown in Figure G.2, or nothing, in which case the grid cell remains white. Furthermore, each mosaic is designed so that any shape that remains white is rectangular (possibly rotated).[1]

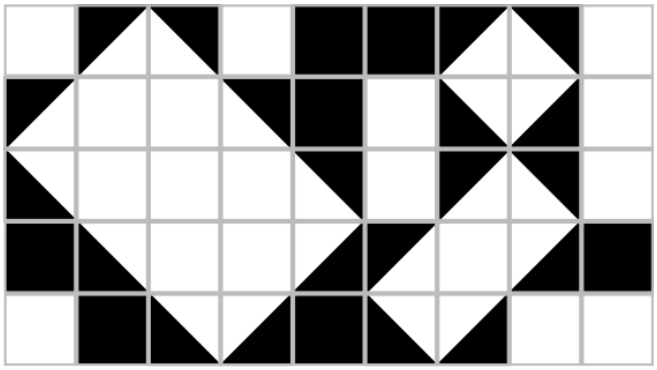


Figure G.1: An example of a mosaic

Figure G.2: Orientations for triangular tiles

To install such a mosaic, an artist starts by placing *all* the black squares. Remaining black triangles will later be added by assistants in order to complete the pattern. To ensure that the assistants complete the mosaic as envisioned, the artist marks some of the black tiles with a numeric label that indicates the number of black triangles that share an edge with that square. (Black tiles without a label may have any number of neighboring triangles.) The artists provides enough labels to ensure a unique design.

For example, Figure G.3 shows a starting configuration that uniquely defines the mosaic shown in Figure G.1. Given such a starting configuration, you are to determine the number of triangles needed to complete the mosaic.

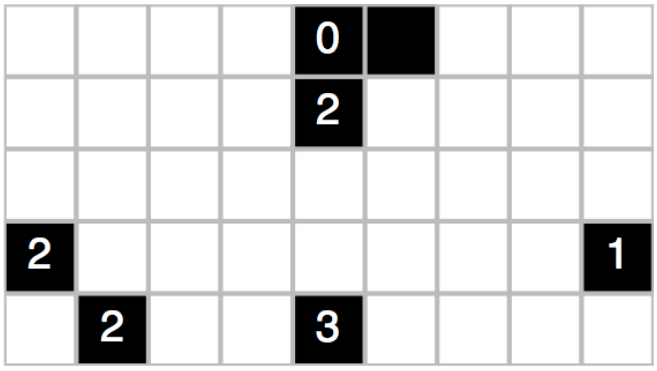


Figure G.3: A starting configuration that uniquely defines the mosaic from Figure G.1

Input

The input consists of a single test case. The first line contains two integers, $1 \leq W \leq 24$ and $1 \leq H \leq 18$, that designate the width and height of the mosaic, respectively. Following that are H additional lines, each with W characters. The characters `0`, `1`, `2`, `3`, and `4` designate black squares with the indicated constraint on the number of neighboring triangles, and the character '$\star$' designates a black square without such a constraint. All remaining locations will be designated with a '`.`' character and must either be left empty or covered with a single black triangle. Inputs have been chosen so that they define a valid and unique mosaic.

Output

Display the number of triangles used in the mosaic.

Sample Input 1

```
9 5
....0*...
....2....
.........
2.......1
.2..3....
```

Sample Output 1

```
20
```

Sample Input 2

```
5 5
2...*
.....
.....
.....
*...0
```

Sample Output 2

```
14
```

Sample Input 3

```
18 10
*1....*2.....**2..
2............3...
...4.............
....4..*.....*....
2*...*3.....3.....
.....*.....**...3*
....3.....3..3....
..............4...
...1............0
..1*2.....2*....1*
```

Sample Output 3

```
92
```

[1] Our inspiration for such mosaics is `www.nikoli.com/en/puzzles/shakashaka`.

Problem H
Pyro Tubes
Time limit: 13 seconds

The Impressively Calculable Pyrotechnics Company has been contracted to provide certain stage effects for a rock festival. In particular, fire is requested (because fire makes the crowd go wild). In order to pull this off, the main stage is outfitted with a series of tubes, each specially calibrated to deliver a flame burst of a particular luminosity. In all, there are 18 numbered tubes arranged on the stage as follows:

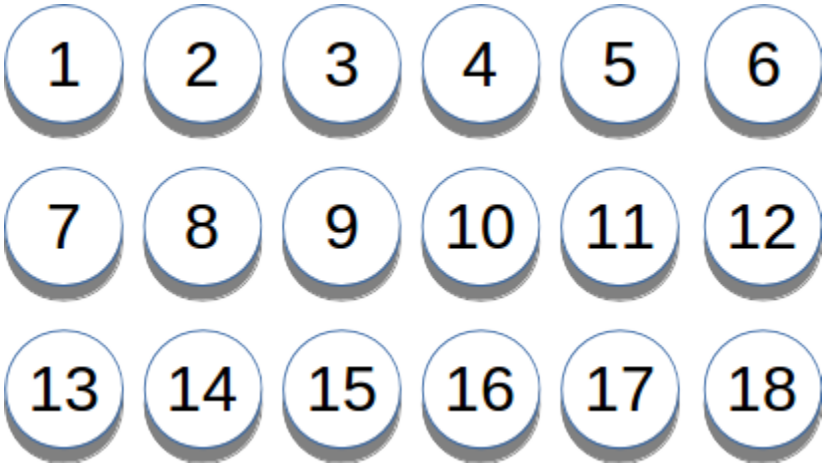


Figure H.1: Pyro tube numbering

Tube 1, when fired, produces a luminosity of 1. More generally, the luminosity produced by Tube N (when $N > 1$) is double the luminosity produced by Tube $N - 1$.

To achieve a specific luminosity, multiple tubes can be activated to fire simultaneously. The luminosity achieved is the sum of the individual luminosities of the activated tubes when fired. The total luminosity is represented by a single integer L. The software used to control the system accepts the value of L as an input argument, and determines which tubes need to be activated to achieve that luminosity (and which remain deactivated).

Figure H.2: Pyro tube states

The design of the system ensures that a given L-value can be achieved by firing a unique set of activated tubes. While the system under normal conditions can precisely generate the expected luminosity, it will occasionally encounter technical problems (sticky valves, clogged tubes, etc.) that temporarily prevent it from firing a particular requested set of tubes, so it can activate and deactivate tubes to fire a different set.

Figure H.3 shows two groups of tubes that have two differences: Tube 7 is activated on the left and deactivated on the right, while Tube 8 is deactivated on the left and activated on the right.

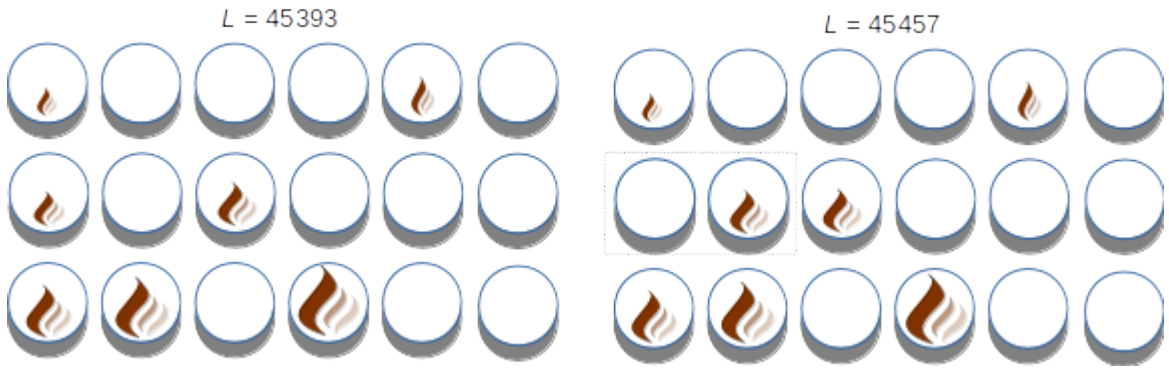


Figure H.3: Pyro tube states for requested luminosity (left) and 2 state changes (right)

Figure H.4 illustrates tube states for a requested L-value of $45\,393$ on the left, while Tube 10 is activated in the group on the right to produce an L-value of $45\,905$:

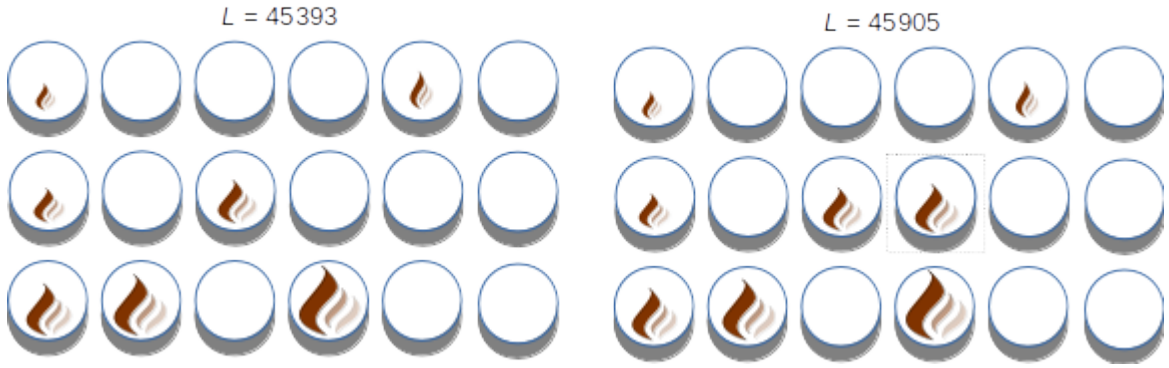


Figure H.4: Pyro tube states for requested luminosity (left) and 1 state change (right)

During the course of designing the stage effects, the producers of the rock festival have been made aware of possible technical issues that require the state of the tubes to be changed, resulting in an L-value that is different than what was originally planned. This will be OK, as long as the tube states can be quickly adjusted, and is brighter than the original L-value. Because they are obsessed with micromanagement, the producers will provide a list of L-values they have scheduled for the show, and they request a report be generated that provides, for each original value given, the count of all alternative L-values that meet the following 3 requirements:

1. The alternate L-value must be greater than the original L-value.
2. The alternate L-value must be another value in the given list.
3. The alternate L-value must be achieved by changing no more than 2 tube states from the original.

Input

The requested L-values will be indicated, one per line in increasing order, such that $1 \leq L \leq 250\,000$ for each L-value (and thus with at most $250\,000$ such values). A single -1 will appear on the last line to signify the end of input.

Output

For each original L-value, in the order given, output a line of the form L:C where integer L is the original requested luminosity value, and integer C is the quantity of alternative values meeting the 3 requirements given above.

Note: some strategies may produce correct results, but will not finish in the allotted time; a time-efficient solution is critical.

Sample Input 1

```
1
3
8
10
25
-1
```

Sample Output 1

```
1:3
3:1
8:2
10:0
25:0
```

Sample Input 2

```
2083
15093
15285
25147
31413
47797
49723
55989
58171
60085
95670
-1
```

Sample Output 2

```
2083:0
15093:3
15285:2
25147:2
31413:3
47797:2
49723:1
55989:1
58171:0
60085:0
95670:0
```

This page is intentionally left blank.

Problem I
Word Clouds Revisited
Time limit: 2 seconds

When we last visited with Tagg Johnson, he was developing software to create images known as word clouds. Although we will not consider all the details, a word cloud provides a visual representation of textual data in which the size of the bounding box for each word depends on the relative frequency of that word in the original text.

Tagg was surprised to stumble upon an oddity involving his algorithm for laying out words in rows. A maximum width is specified for the size of a word cloud, and Tagg's desire is to place the entries into the cloud in a way that minimizes the overall height. Entries must be placed in a predetermined order, with each subsequent entry either placed horizontally to the right of the previous entry, or else to the far left of a new row. The height of each row is equal to the maximum height of all entries placed in that row, and the overall height of the cloud is equal to the sum of the row heights.

Because his goal is to minimize the height, Tagg's original algorithm would place each entry in the same row as the previous entry, unless it did not physically fit because of the given limit on the width of the cloud. As an example, Figure I.1 shows the layout Tagg's original algorithm produces for a particular cloud with a maximum width of 260 units.

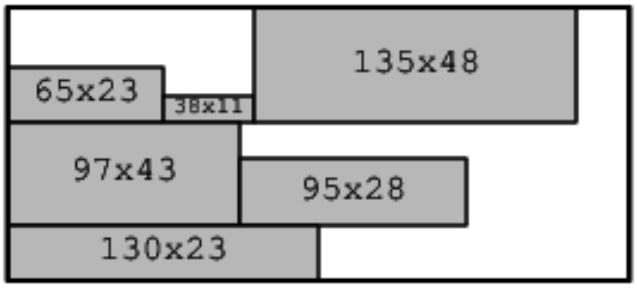


Figure I.1: Tagg's original placement of entries for his word cloud

The first, second, and third entries are placed in the first row (totaling width 238). Then the fourth and fifth entries are placed in a row together (totaling width 193). The sixth entry is by itself on a final row. The overall height of this cloud is $48 + 43 + 23 = 114$ units (as the first row has height 48, the second row has height 43, and the third row has height 23).

However, Tagg later realized that an even better cloud was possible for this same data set, as shown in Figure I.2. By placing the first and second entries in the first row, the third and fourth entries in the second row, and the fifth and sixth entries in a third row, the overall height of the cloud is only $23+48+28 = 99$ units (while still respecting the overall limit of 260 on the width of the cloud).

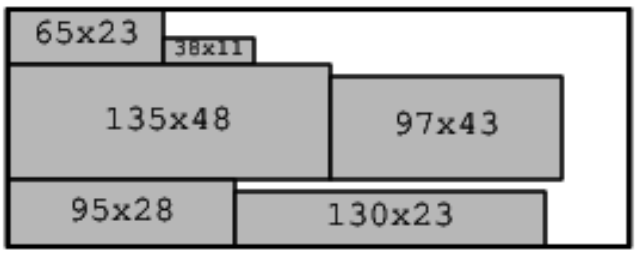


Figure I.2: An optimal placement of the entries from Figure I.1

So now Tagg wants to redesign his software so that, given a list of words and a specific maximum width, it builds a word cloud with the minimum height.

Input

The first line contains two integers, N and C, where $2 \leq N \leq 5000$ is the number of entries to be placed, and $150 \leq C \leq 1000$ is the maximum width for the cloud. Following that are N additional lines, each specifying the bounding box for one entry, in the order in which those entries must be laid out in the cloud. The line describing an entry contains two integers w and h, specifying the respective width and height of the entry, such that $10 \leq w \leq 150$ and $10 \leq h \leq 150$.

Output

Output the minimum height that is required for the word cloud under the restrictions given above.

Sample Input 1	Sample Output 1
6 260 65 23 38 11 135 48 97 43 95 28 130 23	99

Sample Input 2	Sample Output 2
3 309 150 100 10 10 150 100	200