New Zealand Programming Contest 2024

PROBLEM SET

Problem	Points	Title
A	3	Leap Year
B	3	Discounts
C	3	Crime Scene
D	3	Football Pools
E	10	Check Digits
F	10	Hex Search
G	10	Geese
H	10	Croquet
I	30	Watersheds
J	30	Snaggle
K	30	Microspikes
L	30	Crossing the Road
M	100	Music Festivals
N	100	Hotter Colder
O	100	Digging Problem
P	100	OR Game

PREAMBLE

Please note the following very important details relating to input and output:

- Read all input from the keyboard, i.e. use stdin, System.in, cin, input, Console.ReadLine or equivalent. Input will be redirected from a file to form the input to your submission.
- Do NOT prompt for input as this will appear in your output and cause a submission to be judged as wrong.
- Write all output to the screen, i.e. use stdout, System.out, cout, print, Console.WriteLine or equivalent. Do not write to stderr.
- Unless otherwise stated, all *integers* will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by a space.
- An *uppercase letter* is a character in the sequence 'A' to 'Z'. A *lower case letter* is a character in the sequence 'a' to 'z'. A *letter* is either a lower case letter or an upper case letter.
- Unless otherwise stated, a *name* is a continuous sequence of from 2 to 30 characters (printed or written letters or symbols).
- If it is stated that 'a line contains no more than *N* characters', this does not include the character(s) specifying the end of line.
- Input files are sometimes terminated by a 'sentinel' line. This line should not be processed.

Please also note that:

- The filenames of your submitted programs may need to follow a particular naming convention, for example the name of a Java file containing a public class needs to be the name of the class followed by the '.java' extension.
- DOMjudge will reject a submitted file which has any spaces in its file name.
- Problems have a time limit shown on the Problem Set page of DOMjudge. It is usually 1 second, but may be longer. A *TIMELIMIT* error will be issued for submissions that exceed that limit on a single test run.
- Each problem description takes up at least 2 pages, one of which may be empty.

New Zealand Programming Contest 2024

PROBLEM A **LEAP YEARS** **3 POINTS**

It is a tradition of the NZPC, that each leap year we have a leap year problem. So, 2024 being a leap year, here it such a problem!

You know, of course, that if a year is divisible by 4 and not by 100 it is a leap year, unless it is divisible by 400 when it is a leap year. A year that is not a leap year is a common year.

In this problem you will be given a list of years and have to report whether each one is a leap year or a common year. You must put your answer in a sentence which has the correct tense!

Input

The first line contains a single positive integer, N, which is the number of years in the list. N will be no greater than 100.

A list of N years follows, each year on a separate line. Years will be between 1582 and 2299 inclusive, one year per line.

Output

For each year in the list, output a single line which contains, in order:

The year

"`was a`", "`is a`" or "`will be a`" as appropriate.

"`common year`" or "`leap year`" as appropriate, followed by a full stop.

Sample Input

```
5
2016
1987
2025
2023
2024
```

Output for Sample Input

```
2016 was a leap year.
1987 was a common year.
2025 will be a common year.
2023 was a common year.
2024 is a leap year.
```

PROBLEM B **DISCOUNTS** **3 POINTS**

The West Harbour Drinks Company needs to boost its sales, which have not fully recovered after the Covid lockdowns. Rebekah, the manager, wants to implement a "buy three get one free" type scheme.

Your job is to write a program to help Rebekah by making it easier for customers to know how much they can save.

Input

Input consists of a product scenario. A scenario begins with the name of the product on a line of its own. The name consists of 1 or more words which will be separated by spaces.

The second line of each product scenario consists of two integers, PD and PC, separated by a space, which is the price per item of the product in dollars and cents respectively. PD is the dollar price from 0 to 100 inclusive, PC is a valid number of cents. PD and PC will not both be zero.

The next line contains an integer B which is the number of products that must be bought (from 2 to 150 inclusive) to get 1 free.

The next line is another single number, E, on a line of its own being the number of examples to follow. E is not greater than 100.

The number is followed by E lines each containing a single positive integer less than 200. Each number represents a quantity of items to be purchased. You have to use the available deal to work out the best saving the customer can make.

Output

Output starts with the name of the product on a line of its own. This line is followed by E lines, one for each example in the input. Each line will be of the format

```
Buy N, pay for P, get F free. Save $D.
```

N is the number of items bought. P is the number for which payment must be made and F is the number that are free. D is the amount saved compared to having no free items. The amount D will be in the format

```
d.dd
```

That is there will be at least one digit for the dollars, a decimal point and two digits for the cents.

Note that P + F must equal N.

[Turn over for sample input and output]

Sample Input

```
Fizzy Orange Juice
3 99
12
4
10
26
40
150
```

Output for Sample Input

```
Fizzy Orange Juice
Buy 10, pay for 10, get 0 free. Save $0.00.
Buy 26, pay for 24, get 2 free. Save $7.98.
Buy 40, pay for 37, get 3 free. Save $11.97.
Buy 150, pay for 139, get 11 free. Save $43.89.
```

Explanation

1. Buy 10 – must pay for all ten as discounts start at 12.
2. Buy 26 – Pay for 12, get 1 free, pay for another 12, get another 1 free.
3. Buy 40 – Pay for 36, 3 lots of 12, get 3 free. Pay for 1 more.
4. Buy 150 – Pay for 132, 11 lots of 12, get 11 free. Pay for 7 more. Note that buying 144 and getting 12 free would give the customer too many drinks and cost more.

PROBLEM C **CRIME SCENES** **3 POINTS**

People hunting for clues at an outdoor crime scene typically divide up the area they are examining into a grid and will record in which grid cell each item is found. It is thus quite easy to tell how many items were found in a given cell.

Input

In this problem you will be given a crime scene scenario. It begins with a line containing two integers X and Y (separated by a space) representing the length and width of the search grid. Both X and Y are positive integers not greater than 100.

The second line of the scenario is a single integer M which gives the number of items located by the search team. M is a positive integer not greater than 250.

This is followed by M lines each containing the X and Y coordinates of the grid cell in which an item was found. Note that the grid coordinate system starts at 0, 0 and that several items may be found in a particular cell, so cell coordinates may be repeated.

Following the M lines of item locations there is a list of cell references for which the total number of found items is required. The first line of this section is a single integer, N, which gives the number of cell references. N is between 1 and the number of cells in the grid (X times Y).

There follows N lines each containing the X and Y coordinates of a cell.

Output

Output consists of a single line for the scenario. It contains the total number of items found in the N cells listed.

Sample Input

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

Explanation

Cell 9,9 contains 2 items (it appears twice in the input list),

Cell 4,5 contains 1 item,

Cell 6,3 contains no items (it did not occur in the input list).

Total 3 items.

Output for Sample Input

```
3
```

PROBLEM D **FOOTBALL POOLS** **3 POINTS**

From 1922 until the advent of the UK National Lottery in 1994, one of the most popular forms of gambling in England was the football pools. During the football[1] season, pools participants tried to select those football matches that would result in a scoring draw, a game in which both teams scored at least one goal but neither team won.

A common entry was to select 8 such games and score points based on their results. 1 point was scored for a game where one of the teams won (by scoring more goals than their opponent), 2 points if the score was 0-0 and 3 points for a scoring draw. A player who scored 24 points would be a winner, potentially of a large sum of money.

In this problem you will analyse a player's entry.

Input

Input will show the 8 games selected by the player, each on a separate line. The home team will be shown first, followed by a v, followed by the away team. Team names may consist of more than one word.

After this the results of the 8 games will be shown, in the same order, each on a separate line. The home score will be first, followed by the away score. Both will be non-negative integers less than 10 and separated by a space.

Output

The first line of output will be the number of points scored by the player, an integer between 8 and 24 inclusive.

Each scoring draw in the selection must then be displayed, each on a separate line and in the order listed in the input. If there are no scoring draws, the line

```
No scoring draws
```

must be displayed instead of the scoring draws list.

Turn over for sample input and output

[1] Football is sometimes called soccer in New Zealand.

Sample Input

```
AFC Bournemouth v Tottenham Hotspur
Arsenal v Fulham
Brentford v Crystal Palace
Coventry City v Sunderland
Ipswich Town v Leeds United
West Bromwich Albion v Middlesbrough
Burton Albion v Bolton Wanderers
Stevenage v Portsmouth
0 2
2 2
1 1
0 0
3 4
4 2
1 1
0 0
```

Output for Sample Input

```
Points scored: 16
Arsenal v Fulham
Brentford v Crystal Palace
Burton Albion v Bolton Wanderers
```

Explanation

There is 1 home win (4-2) for 1 point.

There are 2 away wins (0-2 and 3-4) for 2 points in total.

There are 2 0-0 draws for 4 points.

There are 3 scoring draws (2-2, 1-1 and 1-1) for 9 points.

Total 16 points. The scoring draws are listed in order.

New Zealand Programming Contest 2024

PROBLEM E **CHECK DIGITS** **10 POINTS**

Professor Wilkerson was teaching a class on computer security. As an exercise he had each of his students generate a check digit for their student ID, and then get other students to confirm that it was correct.

The algorithm he gave to the students was as follows:

- multiply the rightmost digit by 2, the next digit by 3 and so on and then sum the numbers generated
- divide this sum by 11 and then subtract the remainder from 11
- if this number is in the range 1 to 9, it is appended to the right of the student ID
- else if this number is 11, append 0 to the right of the student ID
- else if the number is 10, the ID is rejected and the student has to ask the professor for a temporary ID!

To check that the ID with its added check digit is valid, multiply the rightmost digit by 1, the next by 2 and so on, summing the numbers as before. If the total is exactly divisible by 11, the new ID is valid.

Input

Input will be a series of numbers, one per line. Each number will contain at least 10 and no more than 15 decimal digits. The final line will be a single 0 – this marks the end of input and should not be processed.

Output

For each line of input, follow the above algorithm to calculate the required check sum. Output a single line containing the original ID, followed by the symbols ‘ -> ’, followed either by the ID with the added check sum, or the word “rejected” as appropriate.

Sample Input	**Output for Sample Input**
276320156824553	276320156824553 -> 2763201568245531
643479110054	643479110054 -> 6434791100544
6434791122	6434791122 -> rejected
0	

PROBLEM F **HEX SEARCH** **10 POINTS**

For a word search (also known as a word find), you must find all the words from a list in a 2-dimensional grid. Some word searches have an extra twist; letters left over will spell out an answer to a clue.

Mrs Robinson decided to use this idea with her computing class after teaching them how hexadecimal works. Hexadecimal is a number system where there are 16 different digits (base 16). As a digit can only occupy one position, the letters from A to F (uppercase) are used to represent decimal 10 to 15.

Input

Input will consist of a positive integer N, between 2 and 10 inclusive, representing the dimensions of a 2D grid.

There will follow N lines of N characters, each consisting of numbers, letters (see preamble) or one of the following . ? ! * Each character is space separated.

Output

Output will consist of a number of lines. The first is all the non-hexadecimal characters in the grid, if any, in the same order as they appear in the input. There will then follow the decimal equivalents of the hexadecimal numbers, also in the same order as they appear in the input. These are delimited by either a non-hexadecimal character, or the end of a line.

Sample input 1	**Sample Output 1**
3	GO!
A 2 G	162
0 F A	250
5 9 !	89

Explanation:

The only non-hexadecimal character in the first row is “G”. Similarly in the 2nd row, it’s an “O” and finally in the 3rd row an “!”.

That leaves A2 on the first line, FA on the second line and 59 on the last line.

A2 = A * 16 + 2 = 10 *16 + 2 = 162

FA = 15 * 16 + 10 = 250

59 = 5 * 16 + 9 = 89

Continued

Sample input 2

```
4
1 1 1 1
0 1 0 1
H A I .
Y O U !
```

Sample Output 2

```
HI.YOU!
4369
257
10
```

Explanation:

1111 = 1 * 16^3 + 1 * 16^2 + 1 * 16^1 + 1 = 4096 + 256 + 16 + 1 = 4369

0101 = 257

A = 10

PROBLEM G **THE GEESE OF LOOWATER** **10 POINTS**

Once upon a time, in the Kingdom of Loowater, a minor nuisance turned into a major problem. The shores of the Avon River in central Loowater had always been a prime breeding ground for geese. Due to the lack of predators, the geese population was out of control. The people of Loowater mostly kept clear of the geese. Occasionally, a goose would attack one of the people, and perhaps bite off a finger or two, but in general, the people tolerated the geese as a minor nuisance.

One day, a freak mutation occurred, and one of the geese spawned a multi-headed fire-breathing dragon. When the dragon grew up, he threatened to burn the Kingdom of Loowater to a crisp. Loowater had a major problem. The king was alarmed and called on his knights to slay the dragon and save the kingdom.

The knights explained: "To slay the dragon, we must chop off all its heads. Each knight can chop off one of the dragon's heads. The heads of the dragon are of different sizes. To chop off a head, a knight must be at least as tall as the diameter of the head. The knights' union demands that for chopping off a head, a knight must be paid a wage equal to one gold coin for each centimetre of the knight's height."

Would there be enough knights to defeat the dragon? The king called on his advisors to help him decide how many and which knights to hire. After having lost a lot of money building Mir Park, the king wanted to minimize the expense of slaying the dragon. As one of the advisors, your job was to help the king. You took it very seriously: if you failed, you and the whole kingdom would be burnt to a crisp!

Input

The first line of each test case contains two integers (H and K), both between 1 and 500 inclusive. H is the number of heads that the dragon has, and K is the number of knights in the kingdom.

The next H lines each contain a positive integer, and give the diameters of the dragon's heads, in centimetres. The following K lines each contain an integer, and specify the heights of the knights of Loowater, also in centimetres. H and K are both less than 2,000.

Output

Output a line containing the minimum number of gold coins that the king needs to pay for the knights to slay the dragon. If it is not possible to slay the dragon, output the line:

```
Loowater is doomed!
```

Sample Input 1

```
2 3
5
4
7
8
4
```

Output for Sample Input 1

```
11
```

Sample Input 2

```
2 1
5
5
10
```

Output for Sample Input 2

```
Loowater is doomed!
```

Adapted from a problem by Ondřej Lhoták

PROBLEM H **CROQUET FREE TURNS** **10 POINTS**

In a golf croquet (GC) handicap doubles event, each side may be awarded free turns according to the handicaps of the 4 players. In GC, handicaps range from -6 for the strongest players up to +20 for beginners.

To calculate free turns, the handicaps of all 4 players are needed. In a match between Team A and Team B, the stronger player in Team A is compared to the weaker player in Team B. The difference between their handicaps is halved to give the number of free turns awarded to the weaker of the two players. This is repeated for the stronger player in Team B who is compared to the weaker player in Team A. The number of free turns will be zero if two players being compared have the same handicap.

In a game where both players in one team are weaker than both players in the other team, then both sets of free turns will go to the weaker team.

Half turns are not allowed, so any half from the calculation is normally rounded up. However, if both members of the same team have half turns, one is rounded up and the other rounded down.

Input

Input will consist of 4 lines each representing a player in a match. The first and second players will be one team, the third and fourth the other.

Each line will consist of a name, as defined in the preamble, followed by a space, followed by an integer representing that player's handicap (in the range -6 to +20 inclusive). + signs will not be used.

Where two players on the same side have the same handicap, the first named player is to be considered the stronger player in that team.

Output

Two lines will be output showing the results of the two free turn calculations, starting with that for the stronger player of the first team. Both lines end with a full stop.

If one team receives two half turns, the turns of the weaker player will be rounded up, those of the stronger player rounded down.

If one player receives free turns, the line will say

`<Player 1> receives <F> free turns from <Player 2>.`

Where `<Player 1>` is the weaker player and `<Player 2>` is the stronger player, and `<F>` is the calculated number of free turns. If the number of free turns is 1, the output must read `1 free turn`.

[Continued]

If a calculation results in 0 free turns, the line will say:

`No free turns between <Player 1> and <Player 2>.`

Where `<Player 1>` is the weaker player of their team, and `<Player 2>` is the stronger player of their team.

Sample Input 1

```
Albert 8
Bernice 4
Chen 3
Dimitri 12
```

Output for Sample Input 1

```
Dimitri receives 4 free turns from Bernice.
Albert receives 3 free turns from Chen.
```

Explanation

Albert and Bernice are playing Chen and Dimitri.

Bernice is the stronger player in the first team (has the lower handicap) so is compared to Dimitri, who is the weaker player in the second team. The handicap difference is 8, which is halved to give 4 free turns for Dimitri.

Chen is now compared to Albert. The difference is 5 which is halved to 2½. This is rounded up to 3 free turns for Albert.

Sample Input 2

```
Walter 4
Xerxes 2
Yasmin 1
Zac -5
```

Output for Sample Input 2

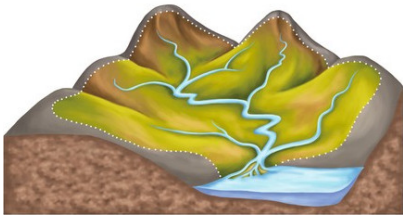
```
No free turns between Yasmin and Xerxes.
Walter receives 5 free turns from Zac.
```

Explanation

Walter and Xerxes are playing Yasmin and Zac.

Xerxes is compared to Yasmin. The handicap difference is 1 so is halved to give ½ free turn for Xerxes. Walter is compared to Zac. With a handicap difference of 9 this gives 4½ free turns for Walter. With Xerxes and Walter both receiving ½s, Xerxes' (the stronger team member) is rounded down to 0, Walter's is rounded up to 5. Yasmin is the weaker player in her team so comes first in the output.

PROBLEM I — WATERSHEDS — 30 POINTS



Geologists sometimes divide an area of land into different regions based on where rainfall flows down to. These regions are called *drainage basins*.

Given an elevation map (a 2-dimensional array of altitudes), label the map such that locations in the same drainage basin have the same label, subject to the following rules.

- From each cell, water flows down to at most one of its 4 neighbouring cells.
- For each cell, if none of its 4 neighbouring cells has a lower altitude than the current cell's, then the water does not flow, and the current cell is called a *sink*.
- Otherwise, water flows from the current cell to the neighbour with the lowest altitude.
- In case of a tie, water will choose the first direction with the lowest altitude from this list: North, West, East, South.

Every cell that drains directly or indirectly to the same sink is part of the same drainage basin. Each basin is labelled by a unique lower-case letter, in such a way that, when the rows of the map are concatenated from top to bottom, the resulting string is lexicographically smallest (in particular, the basin of the most North-Western cell is always labelled 'a'). Note that neighbouring sinks are not considered to be in the same drainage basin.

Input

The first line of the input contains two integers – H and W – the height and width of the map, in cells ($1 \le H, W \le 100$). The next H lines will each contain a row of the map, from north to south, each containing W integers, from west to east, specifying the altitudes of the cells ($0 \le altitudes < 10{,}000$). It is guaranteed that there will be at most 26 basins.

Output

Output H lines that list the basin labels for each of the cells, in the same order as they appear in the input.

Turn over for sample input and output.

Sample Input 1

```
3 3
9 6 3
5 9 6
3 5 9
```

Output for Sample Input 1

```
a b b
a a b
a a a
```

Explanation of Sample 1

The upper-right and lower-left corners are sinks. Water from the diagonal flows towards the lower-left because of the lower altitude (5 versus 6).

Sample Input 2

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

Output for Sample Input 3

```
a a a a a a a a a b
```

Sample Input 3

```
2 3
7 6 7
7 6 7
```

Output for Sample Input 3

```
a a a
b b b
```

Sample Input 4

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

Output for Sample Input 4

```
a a a a a
a a b b a
a b b b a
a b b b a
a a a a a
```

PROBLEM J **SNAGGLE** **30 POINTS**

Professor McSnort has invented a new programming language, *Snaggle*. An expression in *Snaggle* may be a positive or negative integer, or may be of the form (p e_1 e_2), where p is a floating point number between 0.0 and 1.0 (inclusive) and e_1 and e_2 are *Snaggle* expressions.

The value of a Snaggle expression is defined as follows:

- The value of an integer is the integer value itself.
- The value of the expression (p e_1 e_2) is $x+y$ with probability p and $x–y$ otherwise, where x and y are the values of the *Snaggle* expressions e_1 and e_2 respectively.

Input

Input consists of up to 25 Snaggle expressions, one per line, followed by a line containing `()`, which should not be processed.

- Integers are sequences of at most 10 digits, optionally preceded by '-'.
- Real numbers are in standard floating point format (i.e. not E format) without a sign.
- Snaggle expressions of the form (p e_1 e_2) have a single space separating the three elements and no spaces elsewhere.
- Snaggle expressions are at most 300 characters in length.

Expected value

The expected value of a random variable is the weighted average over all possible outcomes. For example, if a variable has the value n_1 with some probability p and the value n_2 otherwise, the expected value is $p \times n_1 + (1 - p) \times n_2$.

Output

Output is a single line for each Snaggle expression in the input giving the expected value of the expression to two decimal places.

Sample Input

```
7
(0.5 3 9)
(0.125 (0.5 100 200) -1000)
()
```

Output for Sample Input

```
7.00
3.00
850.00
```

PROBLEM K **MICROSPIKES** **30 POINTS**

In a study of domestic power consumption, researchers built a simulator for New Zealand homes. For each home the software simulates the power consumption of appliances. The goal of the project was to identify microspikes in power usage – short periods of time during which total power consumption rose above a specified limit.

Before a simulation starts all appliances are turned off (using no power). At various times during the simulation period appliances will increase or decrease their power usage. Every time this happens the simulator outputs a record with the appliance number, the time (seconds) since that appliance's last change (or since the start of the simulation if it's the first record for the appliance), and the change in power level (watts). For a given appliance, records are in order, but unfortunately the simulation was written in such a way that results for different appliances are randomly interleaved. In particular, it cannot be assumed that a record for appliance A, written before a record for appliance B, reports an event on appliance A that occurred before B's event.

Your task is to write a program to read files of appliance records and count the number of microspikes that occur. For any given simulation you will be given a power threshold M and a time threshold S. A microspike occurs if the power level P satisfies $P > M$ for a period of time t_s: $1 \le t_s \le S$.

Input

Your input is data from a one simulation. It begins with a line holding three integers (T, M and S), separated by single spaces. T is the total simulation time in seconds , M is the power threshold and S is the maximum spike duration.

Following are N power change lines. Each holds three integers (a, t and p), again separated by single spaces: 'a' is the appliance number, 't' is the time in seconds since that appliance's last change and 'p' is the change in power level.

A line with three zeros indicates the end of data for the simulation.

A microspike is only counted if:

1. The total power increases from at or below the threshold to above it at some time during the simulation, and
2. The total power decreases again to the threshold or below during the simulation, and
3. The duration of the spike is between 1 and S seconds inclusive.

Input limits

- $0 \le T \le 100{,}000$
- $0 \le M \le 10^9$
- $1 \le S \le 1000$
- $0 \le N \le 1{,}000{,}000$
- $1 \le a \le 100{,}000$
- $0 \le t \le T$

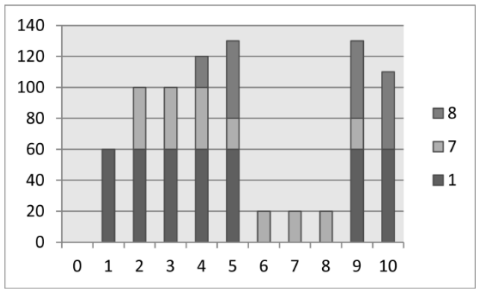
- -10,000 ≤ *p* ≤ 10,000
- You can assume that the power level of an appliance never goes negative, and that total power consumption never exceeds 1,000,000,000.

Output

For each simulation one line of output is required: the number of microspikes observed.

Sample Input (see graph on right)

```
10 100 2
8 4 20
8 1 30
1 1 60
7 2 40
7 3 -20
1 5 -60
1 3 60
7 5 -20
8 1 -50
8 3 50
0 0 0
```



Total power consumption for sample input. Labels on the horizontal axis are the times for the starts of the seconds represented by the bar. All power changes occur instantaneously at the start of a second. Note that the column at time 10 is not part of the simulation. It is included in the chart to show the power level that continues indefinitely after the simulation end.

Output for Sample Input

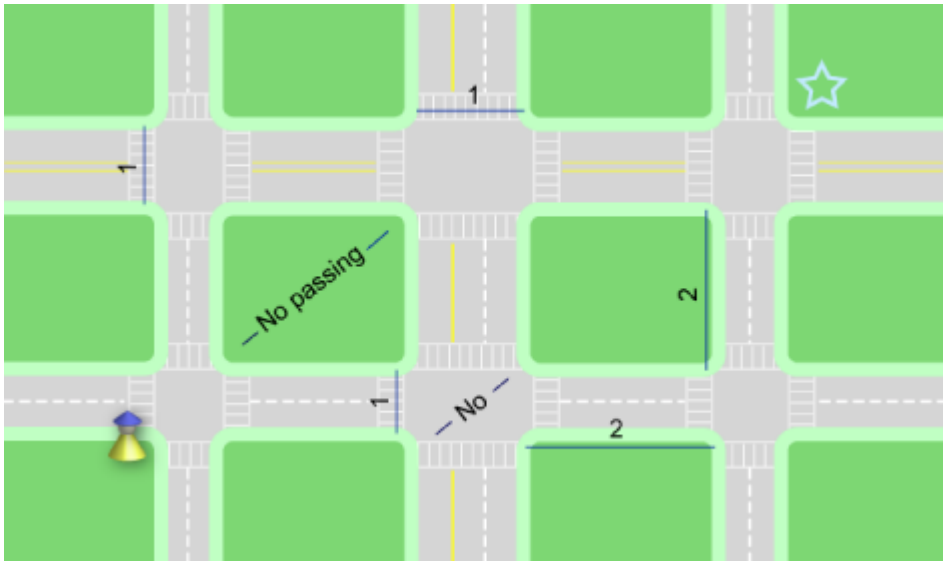
```
1
```

Problem L — Crossing the Road — 30 points

Where roads intersect, there are often traffic lights that tell pedestrians when they should cross the street. A clever pedestrian may try to optimize her path through a city based on when those lights turn green.

The city in this problem is a grid, N rows tall by M columns wide. Our pedestrian wants to get from the northeast corner of the southwest block to the southwest corner of the northeast block. Your objective is to help her find her way from corner to corner in the fastest way possible.

The pedestrian can cross a street in 1 minute, but only if the traffic light is green for the entire crossing. The pedestrian can move between two streets, along one edge of a block, in 2 minutes. The pedestrian can only move along the edges of the block; she cannot move diagonally from one corner of a block to the opposite corner.



Traffic lights follow the following pattern: at intersection i, the north-south lights stay green for S_i minutes, while the east-west lights stay red. Then the north-south lights turn red, the east-west lights turn green, and they stay that way for W_i minutes. Then they start the same cycle again. The pedestrian starts moving at $t=0$ minutes; traffic light i starts a cycle by turning green in the north-south direction at $t=T_i$ minutes. There are cycles before $t=T_i$ as well.

For example, intersection 0 could have the following values:

$S_0 = 3,\ W_0 = 2,\ T_0 = 0$

The north-south direction turns green after 0 minutes. That lasts 3 minutes, during which time the pedestrian can cross in the north-south direction and not the east-west direction. Then the lights switch, and for the next 2 minutes the pedestrian can cross in the east-west direction and not the north-south direction. Then, 5 minutes after it started, the cycle starts again. This is exactly the same as the following configuration:

$S_0 = 3,\ W_0 = 2,\ T_0 = 10$

Turn over for input and output.

Input

The first line in the input contains two space-separated integers `N M` (*1 ≤ N, M ≤ 20*), where *N* and *M* are the number of horizontal roads (rows) and vertical roads (columns), as above. This is followed by *N* lines. The *i*th of those lines contains information about intersections on the *i*th row, where the 0th row is the northernmost. Each of those lines will contain *3M* integers, separated by spaces, in the following form:

$S_{i,0}$ $W_{i,0}$ $T_{i,0}$ $S_{i,1}$ $W_{i,1}$ $T_{i,1}$ ... $S_{i,M-1}$ $W_{i,M-1}$ $T_{i,M-1}$

$S_{i,j}$, $W_{i,j}$ and $T_{i,j}$ all refer to the intersection in the ith row from the north and the jth column from the west ($0 < S_{i,j}, W_{i,j} \leq 10^7$, $0 \leq T_{i,j} \leq 10^8$).

Output

Output a single line containing the number *t*, the minimum number of minutes it takes the pedestrian to get from the southwest corner to the northeast corner.

Sample Input 1

```
1 1
3 2 10
```

Output for Sample Input 1

```
4
```

Sample Input 2

```
1 2
1 5 3 1 5 2
```

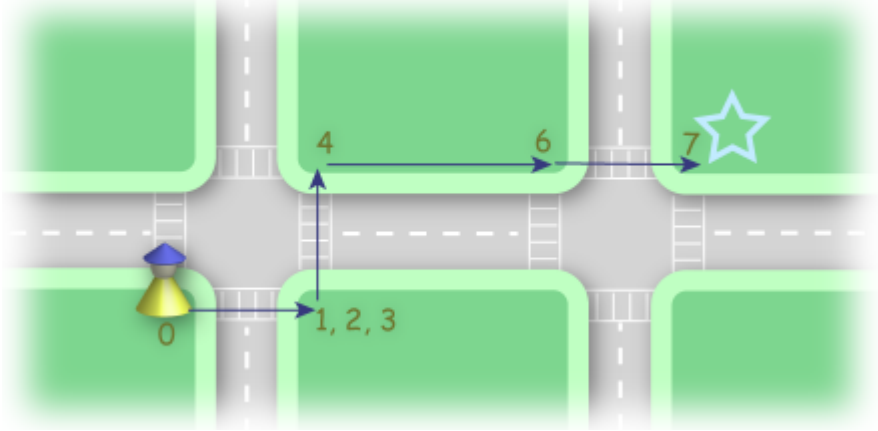
Output for Sample Input 2

```
7
```

Explanation

The first sample is described above. The pedestrian crosses to the North (1 minute), waits 2 minutes and then crosses to the East (1 minute), for a total of 4 minutes.

The second sample is depicted in the diagram below. The pedestrian crosses to the East (1 minute), waits 2 minutes and crosses to the North (1 minute). Then she walks east a block (2 minutes) and crosses to the East (1 minute) for a total of 7 minutes.



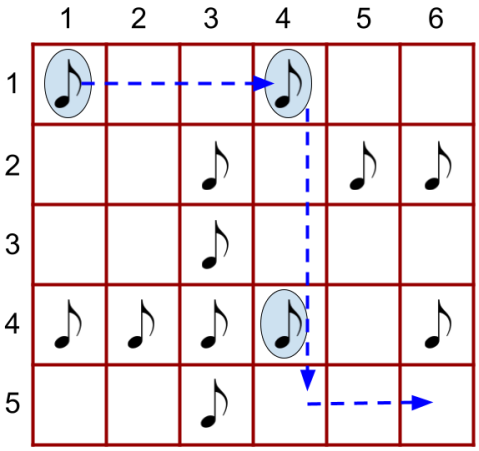
PROBLEM M **MUSIC FESTIVALS** **100 POINTS**

It is the end of winter and music festivals are popping up all over New Zealand. You want to travel from Auckland to Invercargill and visit exactly K festivals along the way.

For the purpose of this task, we model New Zealand as an $N \times M$ grid of towns. Some of the towns are hosting a music festival. Starting from the top-left town (Auckland) at the position $(1,1)$ and moving only down and right, is it possible to reach the bottom-right town (Invercargill) in position (N, M) such that you visit exactly K towns hosting music festivals along the way?

Formally, you can move from town (i,j) down to $(i+1,j)$ if $i < N$, or right to $(i,j+1)$ if $j < M$. These are the only two possible moves from town (i,j).

In the following example, New Zealand is modelled as a 5×6 grid. If $K = 3$, then by moving three times to the right, then four times down, then twice to the right, we reach Invercargill and visit exactly 3 festivals.



Input

The first line of the input contains three integers N, M, and K ($1 \le N, M \le 1500$ and $0 \le K \le 3000$). Each of the next N lines contains a string of length M. The j-th character of the i-th string is '1' if the town (i,j) is hosting a festival, and '0' otherwise.

Output

Output `1` if it is possible to go from $(1,1)$ to (N, M) by only moving down or to the right, such that along the way you visit exactly K towns which host a festival. Output `0` if it is not possible. Note that if Auckland or Invercargill are hosting a festival, they are also counted.

Turn over for sample input and output.

Sample Input 1

```
5 6 3
100100
001011
001000
111101
001000
```

Output for Sample Input 1

```
1
```

Sample Input 2

```
5 6 2
000100
001011
001010
001101
001110
```

Output for Sample Input 2

```
0
```

Sample Input 3

```
4 4 1
0000
0100
1110
0010
```

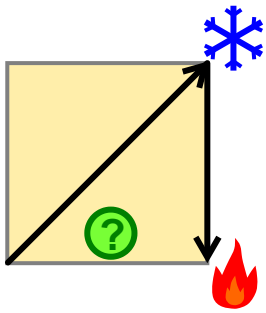
Output for Sample Input 3

```
1
```

Explanation

The first sample corresponds to the example discussed earlier. In the second sample, it is not possible to reach Invercargill and visit exactly two festivals. In the third sample, we can go twice to the right, twice down, once to the right and finally once down. This visits only the festival hosted by the town *(3,3)*.

Problem N — Hotter Colder — 100 points



The children's game *Hotter Colder* is played as follows. Player A leaves the room while player B hides an object somewhere in the room. Player A re-enters at position (0,0) and then visits various other positions about the room. When player A visits a new position, player B announces "Hotter" if this position is closer to the object than the previous position; player B announces "Colder" if it is farther and "Same" if it is the same distance.

Your task is to calculate the area of the region in which the object may have been placed.

Input

The first line of input will consist of a single integer n, the number of positions ($1 \le n \le 50$).

The following n lines each contain an x,y coordinate pair and one of the words "Hotter", "Colder", or "Same". Each pair represents a position within the room, which is a square with opposite corners at (0,0) and (10,10).

Output

For each line of input print a line giving the total area of the region in which the object may have been placed based on the information in the lines processed so far, to 2 decimal places. If there is no such region, output `0.00`.

Sample Input

```
4
10.0 10.0 Colder
10.0 0.0 Hotter
0.0 0.0 Colder
10.0 10.0 Hotter
```

Output for Sample Input

```
50.00
37.50
12.50
0.00
```

Problem O — A Digging Problem — 100 points

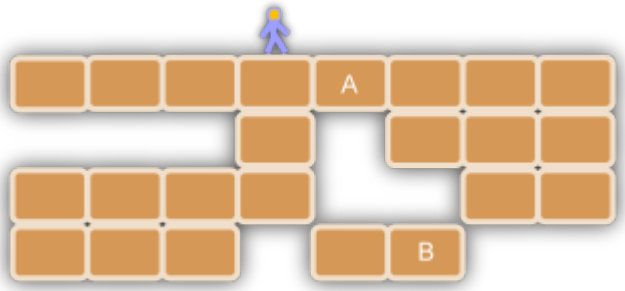
The cave is on fire and there is smoke everywhere! You are trying to dig your way to the bottom of the cave where you can breathe. The problem is that there are some air holes inside the cave, and you don't want to fall too much or you will get hurt.

The cave is represented as an $R \times C$ matrix with air holes and solid rock cells. You start at position (1, 1), which is in the top-left corner. You can move one cell at a time, left or right, if that cell is empty (an air hole). After moving, if the cell below is empty, you fall down until you hit solid rock or the bottom of the cave. The falling distance must be at most F, or you will get hurt. You must reach the bottom of the cave without getting hurt. While falling you cannot move left or right.

You can also "dig", turning a cell that contains solid rock into an air hole. The cell that you dig can be one of two cells: the one to your right and below, or the one to your left and below. The cell above the one you are digging has to be empty. While falling you cannot dig.

Your goal is not only to get to the bottom of cave, but also to "dig" as few cells as possible.

Let's describe the operations with a concrete example:



You start at (1, 1) and move right 3 times to position (1, 4), just like the picture.

You dig the rock at position (2, 5). Cell "A" becomes empty.

You move right one position and since there is no cell below you fall 3 cells to position (4, 5).

You dig the rock at position (5, 6). Cell "B" becomes empty.

You move right one position and since there is no cell below you fall 1 cell to position (5, 6).

You have reached the bottom of the cave by digging 2 cells.

Turn over for input and output.

Input

The first line is formatted as `R C F` where $2 \le R \le 50$ is the number of rows in the cave, $2 \le C \le 50$ is the number of columns in the cave, and $1 \le F < R$ is the maximum distance you can fall without getting hurt.

This is followed by R rows, each of which contains C characters. Each character can be one of two things:

- `#` for a solid rock
- `.` for an air hole

The top-left cell will always be empty, and the cell below it will be a solid rock.

Output

Output `No` if you cannot reach the bottom of the cave. Output `Yes D` if the bottom of the cave can be reached and the minimum number of cells that need digging is D.

Sample Input 1

```
2 2 1
.#
##
```

Output for Sample Input 1

```
No
```

Sample Input 2

```
3 3 1
...
###
###
```

Output for Sample Input 2

```
Yes 3
```

Sample Input 3

```
3 2 1
..
#.
..
```

Output for Sample Input 3

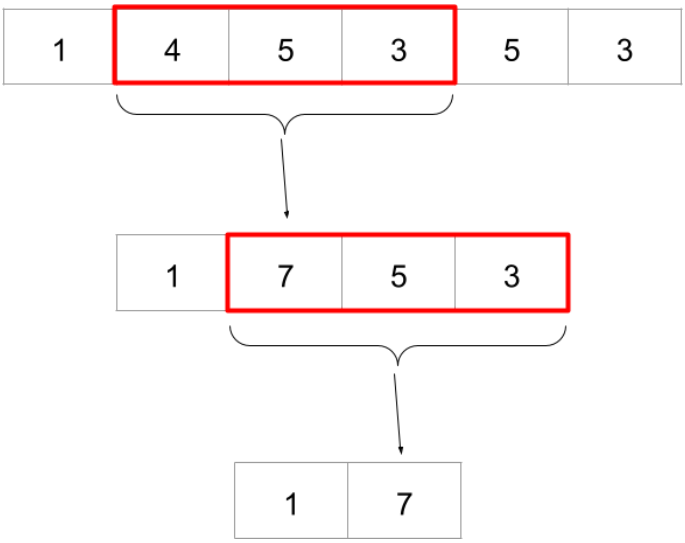
```
No
```

PROBLEM P **OR GAME** **100 POINTS**

With time left to spare until the end of the contest, you decided to play the card game Solitaire. Alas, you forgot to bring a card deck! So instead of playing Solitaire, you will have to settle for a number game called "OR Game".

Instead of cards, you are given an array A with n elements and a positive integer K. The rules of the game are as follows: As long as the array A has at least K elements, you have to select some K consecutive elements, remove them, and in their place insert their bitwise OR (definition below). Once the array's length is less than K, your final score is calculated as the sum of the elements. Your goal is to minimise this sum.

For example, suppose $A = [1,4,5,3,5,3]$ and $K = 3$. Suppose in the first step we choose the three consecutive elements starting from the second element, namely: 4, 5, 3. We remove them and replace them with 7, because 4 | 5 | 3 = 7 (in binary: `100 | 101 | 011 = 111`). Now the array is $[1,7,5,3]$. Suppose in the second step we again choose the three consecutive elements starting from the second one. We replace the numbers 7, 5, 3 with 7 because 7 | 5 | 3 = 7. This leaves us with only two numbers, 1 and 7. Thus our final score is 8. One can verify that this is the smallest possible score. Note that this score can also be achieved using a different sequence of moves.



Bitwise OR

The bitwise OR of two non-negative integers x and y is denoted x | y in most programming languages, and is performed as follows: Write down each number in binary, adding leading zeros if necessary to make them the same length. For example, $x=5$ in binary is `101`, and $y=9$ in binary is `1001`. Because the second binary number has more digits, we insert a `0` at the beginning of the first, so it becomes `0101`. Now write the two sequence of bits one below the other (see the illustration). In each column, if the two bits are both `0` then the resulting bit is `0`, otherwise it is `1`. So 5 | 9 is `1101` in binary, which is 13 in decimal.

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

The notation x_1 | x_2 | x_3 | ... | x_K means $(((\ x_1 \mid x_2\ ) \mid x_3\ ) \ldots\ ) \mid x_K$.

Turn over for input and output.

Input

The first line of the input contains two integers N and K ($2 \le K \le N \le 400,000$). The next line contains N integers $a_1, a_2, a_3, ..., a_n$ ($0 \le a_i \le 1,000,000,000$).

Output

Output a single integer which denotes the smallest possible score you can obtain.

Sample Input 1

```
6 3
1 4 5 3 5 3
```

Output for Sample Input 1

```
8
```

Sample Input 2

```
7 2
1 3 5 7 9 11 13
```

Output for Sample Input 2

```
15
```