MARATÓN DE PROGRAMACIÓN UFPS 2017

Set de Problemas

Contents

MARATÓN DE PROGRAMACIÓN UFPS 2017

Gracias a los esfuerzos del Grupo de Estudio en Programación Competitiva, el Semillero de Investigación en Linux y Software Libre (SILUX) y el Programa de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, se lleva a cabo la Maratón de Programación UFPS 2017, la cual permite poner a prueba las habilidades en algoritmia y programación de los estudiantes del programa, y gracias a la Red de Programación Competitiva (RPC) esta competencia llega a por lo menos 12 Universidades en 5 paises de Latinoamerica.

El presente conjunto de problemas ha sido escrito por:

- Milton Jesus Vera - UFPS
- Gerson Yesid Lázaro - UFPS
- Angie Melissa Delgado - UFPS
- Marcos Javier Alvarez -UFPS
- Hugo Humberto Morales - UTP Pereira
- Eddy Cael Mamani - Coderoad Bolivia

Agradecimientos especiales por su colaboración y apoyo en la realización del evento:

- Hugo Humberto Morales
- Fabio Avellaneda
- Santiago Gutierrez Alzate
- Megaterios

GRUPO DE ESTUDIO EN PROGRAMACIÓN COMPETITIVA

El grupo de estudio en Programación Competitiva hace parte del Semillero de Investigación en Linux y Software Libre (SILUX) y tiene como fin preparar y fortalecer a los estudiantes de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander en algoritmia y programación, en un entorno competitivo que favorece el aprendizaje y el trabajo en equipo.

Integrantes del grupo de estudio han participado en la Maratón Nacional de Programación desde el año 2014, logrando por 3 años consecutivos la clasificación a fase Regional Latinoamericana.

INSTRUCCIONES

Puede utilizar Java, C, C++ o Python, teniendo en cuenta:

1. Resuelva cada problema en un único archivo. Debe enviar a la plataforma únicamente el archivo .java, .c, .cpp, o .py que contiene la solución.
2. Todas las entradas y salidas deben ser leídas y escritas mediante la entrada estándar (Java: Scanner o BufferedReader) y salida estándar (Java: System.out o PrintWriter).
3. El archivo con la solución debe llamarse tal como se indica en la linea ”source file name” que aparece debajo del título de cada ejercicio.
4. En java, la clase principal debe llamarse igual al archivo (Si el Source File Name indica que el archivo debe llamarse example.java, la clase principal debe llamarse example).
5. En java, asegúrate de borrar la linea ”package” antes de enviar.
6. Su código debe leer la entrada tal cual se indica en el problema, e imprimir las respuestas de la misma forma.
7. Si su solución es correcta, en unos momentos la plataforma se lo indicará con el texto ”YES”. En caso contrario, obtendrá un mensaje de error.

REGLAS

1. Gana el equipo que resuelva mas problemas. Entre dos equipos que resuelvan el mismo número de problemas, gana el que los haya resuelto en menos tiempo (ver numeral 2).
2. El tiempo de un equipo se ajusta de la siguiente manera: Suma de los tiempos transcurrido en minutos desde el inicio de la competencia hasta el momento en que se envió la solución correcta. Habrá una penalización de 20 minutos que se suman al tiempo por cada envío erróneo (esta penalización solo se cuenta si al final el problema fue resuelto).
3. Está estrictamente prohibido el uso de internet durante la competencia. Unicamente se puede acceder a la plataforma en la cual se lleva a cabo la competencia.
4. NO se permite el uso de dispositivos electrónicos durante la competencia.
5. NO se permite la comunicación entre miembros de equipos diferentes. Cada integrante solo puede comunicarse con sus dos compañeros de equipo.
6. Se permite todo tipo de material impreso (libros, fotocopias, apuntes, cuadernos, guías) que el equipo desee utilizar.

PROBLEM A. PROBLEM WITH A RIDICULOUSLY LONG NAME BUT WITH A RIDICULOUSLY SHORT DESCRIPTION

Source file name:	`problem.c`, `problem.cpp`, `problem.java`, `problem.py`
Input/Output:	`standard`
Author('s):	`Gerson Yesid Lázaro - UFPS`

All of us hate loooooong descriptions, so here goes one short: Calculate $(66^n) mod 100$ for the given n.

INPUT

First line of input contains T, the number of test cases ($T \leq 5000$). Then, there are T lines, one for each case, containing n ($1 \leq n \leq 10^{1000}$).

OUTPUT

Print one line per case, the solution of $(66^n) mod 100$.

EXAMPLE

Input	Output
4 0 1 2 9999999999999999999999	1 66 56 36

PROBLEM B. BEAUTIFUL TRIAD

Source file name:	`beautiful.c, beautiful.cpp, beautiful.java, beautiful.py`
Input/Output:	`standard`
Author('s):	`Gerson Yesid Lázaro - UFPS`

A **numerical triad** of limit N is a set of 3 numbers A, B and C where $0 \leq A, B, C \leq N$. A numerical triad of limit N is considered a **beautiful triad** in base K, if and only if all the pairs that can be formed between their values A, B and C differ by no more than K units.

For example $(4, 4, 6)$ is a beautiful triad in base 3 because the difference between A and B is 0, the difference between A and C is 2 and the difference between B and C is 2, all differences being less than 3. However, this is not a beautiful triad in base 1, because two of their differences are greater than 1.

Knowing N and K, can you tell how many different beautiful triads of limit N in base K can be formed? Note that $(4, 4, 6)$, $(4, 6, 4)$ and $(6, 4, 4)$ are three different triads.

INPUT

The first line of the input contains an integer T, the number of test cases. Each case contains two integers N and K as described previously ($0 \leq N \leq 2 * 10^9$, $0 \leq K \leq 1000$, $K \leq N$).

OUTPUT

Print one line per test case, the number of beautiful triads of limit N in base K that can be formed. It is guaranteed that this number fits in a 64 bits signed integer.

EXAMPLE

Input	Output

5
0 0
1 0
1 1
2 1
2000000000 0
	1
	2
	8
	15
	2000000001

PROBLEM C. CHRIS' CHALLENGE

Source file name:	`challenge.c, challenge.cpp, challenge.java, challenge.py`
Input/Output:	`standard`
Author('s):	`Angie Melissa Delgado - UFPS`

Sets, sets, sets... little Ashley recently learned about sets in the school and she was fascinated with them, that's why she keeps watching sets all over around her. Her friend Chris loves mathematics and he is getting bored with Ashley's obsession, so he decided to give her a challenge.

In this challenge Ashley will have a list of consecutive integers in the range $[a, b]$ and an integer k, each integer consider as an unitary set. Then, for each pair of integers x, y in the list, she will merge the sets to which each integer belongs if the relationship between x and y is $k - beautiful$. According to Chris, two integers x and y are $k - beautiful$ if the difference between the sum of its dividers does not exceed the value k. After she groups all the integers she will have to answer two questions proposed by Chris:

- What is the size of the largest set that can be formed? And
- How many different sets will be at the end?

Given the numbers a, b and k can you solve Chris' challenge?

INPUT

The first line of the input contains an integer t, the number of test cases. Each test case is formed by a line with 3 integers a, b and k ($1 \leq a, b \leq 10^4$, $1 \leq k \leq 15000$ and $1 \leq b - a \leq 10^3$), the first number of the range, the last number of the range and the base for the beautifulness condition.

OUTPUT

For each test case print two integers, the answer for the first and the second question proposed in the Chris' challenge.

EXAMPLE

Input	Output

4
1 10 2
40 45 3
3219 3223 1962
4762 4765 5321
	6 3
	2 5
	3 2
	4 1

PROBLEM D. THE DOMINO GAME

Source file name:	`domino.c, domino.cpp, domino.java, domino.py`
Input/Output:	`standard`
Author('s):	`Milton Jesús Vera - UFPS`

In a very extrange place where arrays are not discovered yet, programmers use binary numbers powers to represent sets of data. The main idea is that each bit represents an element. Thus, they can represent any set with just one number, when the power set and its order are know.

The domino game has 28 pieces. If they are sorted and listed in ascending order by their value, you can think of represent each piece with a bit. An 1 in the i-position means that the piece is present, otherwise a 0 means that the piece is absent. The next table shows the bit that corresponds to every piece and the value of the corresponding two power.

Piece	Bit Position	Power of Two	Piece	Bit Position	Power of Two
0 - 0	0	1	2 - 3	14	16384
0 - 1	1	2	2 - 4	15	32768
0 - 2	2	4	2 - 5	16	65536
0 - 3	3	8	2 - 6	17	131072
0 - 4	4	16	3 - 3	18	262144
0 - 5	5	32	3 - 4	19	524288
0 - 6	6	64	3 - 5	20	1048576
1 - 1	7	128	3 - 6	21	2097152
1 - 2	8	256	4 - 4	22	4194304
1 - 3	9	512	4 - 5	23	8388608
1 - 4	10	1024	4 - 6	24	16777216
1 - 5	11	2048	5 - 5	25	33554432
1 - 6	12	4096	5 - 6	26	67108864
2 - 2	13	8192	6 - 6	27	134217728

TABLE 1: BITS 28, 29, 30 AND 31 ARE NOT USED AND ARE ALWAYS IN 0

If you have the pieces $0-0$, $6-6$ and $1-1$, then the bits 0, 7 and 27 will be on while the other bits would be off and the number that represents the pieces would be 134217857 ($2^0+2^7+2^{27}$). If you have the pieces $3-4$, $4-2$ and $2-5$, then the bits 19, 15 and 16 will be on while the others bits would be off and the number that represents the pieces would be 622592 ($2^{19}+2^{15}+2^{16}$). If all the pieces are on the table, all the bits will be on and the sum will be 268435455.

A domino game is considered $closed$ if there are 7 pieces of the same pint on the table and the pints of the pieces at the ends are equals. For a given domino game, could you tell if it is closed?

INPUT

The input consist of multiple test cases. Each case in conformed by a line with three integers a, b and c ($1 \leq a \leq 268435455$ and $0 \leq b, c \leq 6$), the first number represents the pieces that are on the table, the other numbers represents the pints that are at the end of the game. You should ignore how the pieces were placed.

OUTPUT

For each case print *Closed Game* if the game is closed. Otherwise prints *Game Open, keep playing*.

EXAMPLE

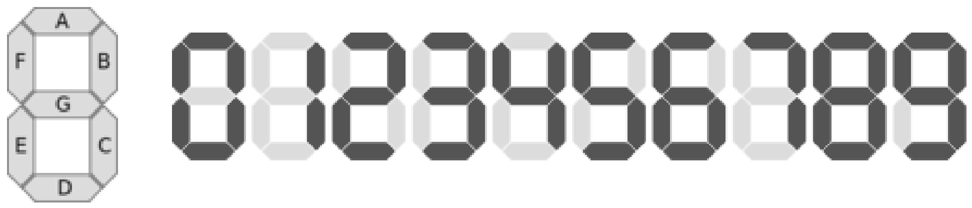
Input	Output

67633535 0 0
86643040 6 6
268435455 3 3
	Closed Game
	Game Open, keep playing
	Closed Game

PROBLEM E. SEVEN SEGMENTS DISPLAY

Source file name:	`seven.c, seven.cpp, seven.java, seven.py`
Input/Output:	`standard`
Author('s):	`Gerson Yesid Lázaro Carrillo - UFPS`

The seven segment displays are little digital devices with 7 bright leds, that using different combinations of leds on and off, can form any digit.



Louis has recently bought N of this devices and has found the nasty surprise that in some devices there are leds that do not light. He doesn't want to loose his inversion so he is making diferent experiments.Iniatially he opens every device, separating every one of its leds. His intention is to discard the leds that do not work and with those that work build new seven segments displays. Altough his plans works, he discovered that by its connection system, every led can only be placed in the same place that has in the original device (in other words, an A led only can use as an A led, never as aB or another letter led)

Knowing the quantity of seven segment displays that Louis has, and the functional and non functional leds in every one of them, cand you calculate how many enterely functional seven segments displays he can form?

INPUT

The entry begins with a number T, the number of test cases. Each case begins with a line with an integer N ($1 \leq N \leq 100$), the amount of seven segment displays bought by Louis. Next, there are N lines, each one with 7 integers separated by blank spaces, representing the state of every led from A to G. Each one of this integers can only have the values 1 (if the led works) or 0 (if the led does not work).

OUTPUT

For each test case print a line with an integer, the maximum number of seven segment displays that Louis can form.

EXAMPLE

Input	Output
3	2
3	0
1 1 1 1 1 1 1	1
0 0 0 1 1 1 1	
1 1 1 0 0 0 0	
3	
1 1 1 1 1 1 0	
1 1 1 1 1 1 0	
1 1 1 1 1 1 0	
2	
1 0 1 0 1 0 1	
0 1 0 1 0 1 0	

PROBLEM F. THE ROBOT'S GRID

Source file name:	`robot.c, robot.cpp, robot.java, robot.py`
Input/Output:	`standard`
Author('s):	`Gerson Yesid Lázaro Carrillo - UFPS`

A robot travels over a luminous grid. Initially the grid is off and every time the robot walks over a cell, this cell turns on a light that keeps lighted while the robot keeps moving.

The robot can only make to actions:

- Take a step forward
- Turn 90 degrees clockwise and advance one step forward.

The robot always starts from the lower left corner facing the top of the grid. It can do any sequence of movements a and b (it can make many consecutive a movements, or many consecutive b movements, or alternate them in any way) as long as it does not leaves the grid and don't pass over a previously lighted cell (ie, it can only pass once over every cell). Whenever it's possible, the robot will keep moving, stopping only when he reaches a cell where he can't make more movements, ending the route. How many different routes can the robot make over the grid?

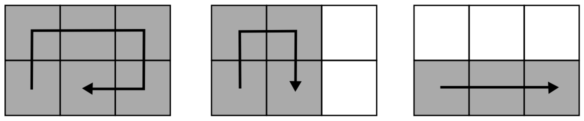


FIGURE 1: GRID OF 2 ROWS AND 3 COLUMNS, WHERE 3 DIFFERENT ROUTES CAN BE DONE

INPUT

The first line of the input contains an integer T, the number of test cases. Each case contains two integers R and C, the number of rows and columns of the grid ($2 \leq R \leq 25$, $1 \leq C \leq 25$).

OUTPUT

For each case print a line with the number of different routes that the robot can make.

EXAMPLE

Input	Output

4
2 1
2 2
2 3
3 3
	1
	2
	3
	6

PROBLEM G. GEONOSIS

Source file name:	`geonosis.c, geonosis.cpp, geonosis.java, geonosis.py`
Input/Output:	`standard`
Author('s):	`Angie Melissa Delgado - Marcos Javier Alvarez - UFPS`

After years of planning, the Galactic Empire has built the planetary fortress of Geonosis, a perfect prison for the leaders of the rebelion and a special place reserved for the evil Luke Skywalker. However, as it has been demonstrated with the failures of Death Star I and II, the fortress is not perfect and the evil Rebel Alliance plans to use a weakness detected in the communication towers surrounding Geonosis.

To achieve their evil plans, the Rebel Alliance has stolen a map with the structure of the fortress. Geonosis has a communication complex formed by n towers. Each pair of the complex's tower is connected by power lines that send messages at a w cost of energy.

The rebels want to build the Rogue Two ship to destroy the fortress and rescue its prisoners. To achieve that, they have to destroy all the towers of the fortress. They have discovered that the power necessary to destroy a tower is equal to the sum of the minimum energy needed to send messages between each pair of the fortress towers, either directly or indirectly through other towers. In other words, if we define $d(u, v)$ as the minimum energy to send a message from the tower u to the tower v, the power needed to destroy one tower will be:

$$\sum_{v,u,u!=v} d(u, v)$$

Note that once a tower is destroyed, it stops counting for this formula.

On the other hand, the commander of the Rogue Two is very capricious so he wants to destroy the towers in a given order. Knowing the information of the communication towers in Geonosis and the order in which each tower will be destroyed, can you tell what is the power needed for the Rogue Two to complete it's mission?

INPUT

The first line of the input contains an integer t, the number of test cases.

Each test case begins with a line containing an integer n ($1 \leq n \leq 500$) the number of towers in the fortress. Next n lines contain n integers each, the j_{th} number in the i_{th} line w_{ij} ($1 \leq w_{ij} \leq 10^4$ $w_{ii} = 0$) represents the amount of energy that cost to send a message from tower i to tower j.

Each test case ends with a line containing n distinct integers: $x_1, x_2, ..., x_n$ ($0 \leq x_i < n$), the order in which the towers will be destroyed.

OUTPUT

For each test case print the amount of energy that Rogue Two will need to destroy the Geonosis fortress.

EXAMPLE

Input	Output
3	96
3	41118
0 35 58	287
12 0 5	
1 2 0	
0 1 2	
3	
0 9982 1413	
8327 0 5612	
3577 7893 0	
1 0 2	
4	
0 50 10 30	
4 0 23 58	
2 1 0 5	
67 24 25 0	
0 3 1 2	

Use faster I/O methods

EXPLICATION

In the first test case, the energy needed to destroy the tower 0 is 89, to destroy the tower 1 is 7, to destroy the tower 2 is 0. So, the total energy needed by Rogue Two is 96.

PROBLEM H. HOW MANY INVERSIONS?

Source file name:	`howmany.c, howmany.cpp, howmany.java, howmany.py`
Input/Output:	`standard`
Author('s):	`Hugo Humberto Morales Peña - UTP Pereira`

Humbertov Moralov in his student days, he is attended system engineering at "University of missing hill". He was evaluated in its first course of Analysis of Algorithms (at the first half of 1997) with the following topics and questions:

Inversions :
Let $A[1 \dots n]$ an array of distinct integers of size n. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an **inversion** of A.

Given the above definition about an inversion, *Humbertov Moralov* must answer the following questions:

1. List all inversions in $\langle 3, 2, 8, 1, 6 \rangle$.
2. What array of size n, with all the numbers from the set $1, 2, 3, \dots, n$ has the largest amount of inversions? How many inversions?
3. Write an algorithm to determine the number of inversions in any permutation of n elements with $\theta(n \log n)$ in the worst case run time.

Humbertov Moralov answered questions 1. and 2. without any problem, but he was not able to solve the question 3 at time. Days later he thought the following solution:

```
inv $\leftarrow$ 0
function MERGE(A[], p, q, r)
    $n_1 \leftarrow q - p + 1$
    $n_2 \leftarrow r - q$
    create arrays L[1 $\dots n_1$ + 1] and R[1 $\dots n_2$ + 1]
    for $i = 1$ to $n_1$ do
        $L[i] \leftarrow A[p + i - 1]$
    end for
    for $j = 1$ to $n_2$ do
        $R[j] \leftarrow A[q + j]$
    end for
    $L[n_1 + 1] \leftarrow \infty$
    $R[n_2 + 1] \leftarrow \infty$
    $i \leftarrow 1$
```

```
15:     $j \leftarrow 1$
16:     **for** $k = p$ to $r$ **do**
17:         **if** $L[i] \leq R[j]$ **then**
18:             $A[k] \leftarrow L[i]$
19:             $i \leftarrow i + 1$
20:         **else**
21:             $A[k] \leftarrow R[j]$
22:             $j \leftarrow j + 1$
23:             $\textbf{inv} \leftarrow \textbf{inv} + \textbf{n}_1 - \textbf{i} + 1$
24:         **end if**
25:     **end for**
26: **end function**
27: **function** MERGESORT(A[], p, r)
28:     **if** p < r **then**
29:         $q \leftarrow \lfloor (p + r)/2 \rfloor$
30:         MERGESORT(A[], p, q)
31:         MERGESORT(A[], q + 1, r)
32:         MERGE(A[], p, q, r)
33:     **end if**
34: **end function**
```

Will this code solve the problem? Just adding the lines 1 and 23 will be enough to solve the problem?

Please help *Humbertov Moralov* to validate this solution! For this, you must implement this solution in any of the programming languages accepted by the ACM-ICPC and verify if the expected results are generated.

INPUT

The input may contain several test cases. Each input case begins with a positive integer n ($1 \leq n \leq 10^6$) denoting the length of A, followed by n distinct lines. Each line contains a positive integer from array A. For $i \in [1, n]$, will meet that $1 \leq A[i] \leq 10^8$. The input ends with a test case in which n is zero, and this case must not be processed.

OUTPUT

For each test case, your program must print a positive integer representing the total number of inversions in the array A. Each valid test case must generate just one output line.

EXAMPLE

Input	Output
5 3 2 8 1 6 5 5 4 3 2 1 1 10 0	5 10 0

Use faster I/O methods

PROBLEM I. MY PASSWORD IS A PALINDROMIC PRIME NUMBER

Source file name:	`password.c, password.cpp, password.java, password.py`
Input/Output:	`standard`
Author('s):	`Milton Jesús Vera - UFPS`

Gauss' is a clever guy who is a little obsessed with information security, that's why he is always looking for the most secure password. After a long research Gauss is choosing for his password a palindromic prime number ppm, which is ultra secure, the only problem is that he always forgets it. As a help to his bad memory, Gauss wants to write a file with the decimal number produced by the division of one (1) and his password, $1/ppm$. Gauss knows that the result is always a pure recurring (periodic) decimal number, and therefore he only wants to save the number with its period. For example:

Gauss' Password (PPM)	Decimal Number 1/PPM	Length 1/PPM (period)
101	0,0099	4
757	0,001321003963011889035667107	27
191	0,00523560209424083769633507853403141361256544502617801047120418848167539267015706806282722513089	95
11311	0,00008840951286358412165148970029175139244982760144991601096277959508443108478472283617717266377862257978958535938466979046945451330563168596941030854919989390858456369905401821235964989832906020687826010078684466448589868269825833259658739280346565290425249756873839625143665458403324197683670762974096012730969852356113517814516842012200512775174608787905578640261692158076209	377
14341	A number of 14340 decimals: 0,000069730144341398786695488459661111498500801896659926086046998117286102782232759221811589149989540478348790181995676731050833275224879...	14340
79997	A number of 39998 decimals	39998

Having the file, Gauss knows that he can execute an algorithm to remember his password. Can you help Gauss to write the file?

INPUT

The first line of the input contains an integer t, the number of test cases, followed by t lines, each line contains a palindromic prime number n $(10 < n < 10^5)$.

OUTPUT

For each test case print the pure recurring decimal number that will help Gauss to remember his password. (Use . as a decimal point)

EXAMPLE

Input	Output
2 101 757	0.0099 0.001321003963011889035667107

PROBLEM J. JULIANA AND THE STAIRS

Source file name:	`juliana.c, juliana.cpp, juliana.java, juliana.py`
Input/Output:	`standard`
Author('s):	`Gerson Yesid Lázaro Carrillo - UFPS`

Juliana is a talented ballet dancer who has a strange habit: she falls of the stairs (It is not accidental, it is her style). Her house's stair has multiple numbered steps (the bottom step has the number 1, and every step is greater than its inmediately previous step, till reach de top step. Her house has 10^5 steps). Her friends make a mark with red painting in the lowest and the highest step in which she has fallen, and they assure that she has fallen in every step between this two marks. Can you verify if this affirmation it is true? If not, you can tell in how many steps of that range she hasn't fallen yet?

INPUT

The first line of the input contains an integer T, the number of test cases. Each case is formed by two lines. The first line contains an integer N ($2 \leq N \leq 10^4$), the number of falls that Juliana has had. The second line contains N integers C_i ($1 \leq C_i \leq 10^5$) separated by blank spaces, denoting the step in which Juliana has fallen (Note that Juliana could has fall more than once in a single step).

OUTPUT

For each case print "TRUE" if the affirmation of Juliana's friends it's true, otherwise, print the number of steps in the red range in which Juliana hasn't fall yet.

EXAMPLE

Input	Output

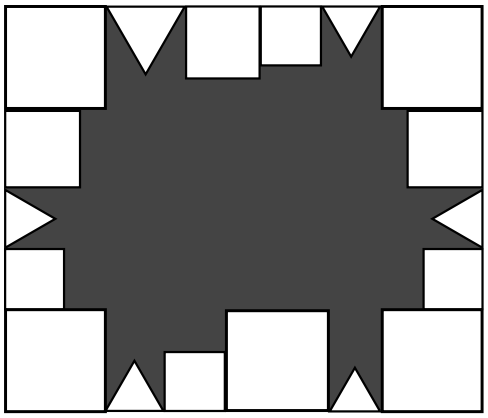
3
5
6 7 8 9 10
4
2 8 6 4
6
5 8 5 8 5 8
	TRUE
	3
	2

In the first case, Juliana has fallen in every steps in the range $[6, 10]$. In the second test case, Juliana hasn't fallen in the steps 3, 5 nor 7, so the range $[2, 8]$ isn't complete. In the third case, Juliana hasn't fall in the steps 6 nor 7 to complete the range $[5, 8]$.

PROBLEM K. POLYGONAL PARK

Source file name:	`park.c, park.cpp, park.java, park.py`
Input/Output:	`standard`
Author('s):	`Milton Vera - Melissa Delgado - Gerson Lázaro - UFPS`

When Polygonal Town was built, in its center it was left a rectangular space to built a park. However, there was a serious problems in the construction and many houses were built over the park space. More exactly, every limit of the park was invaded with two types of houses: Houses with square base and houses with equilateral triangle base, as you can see in the following picture (The white squares and triangles are houses).



The mayor of Polygonal Town has decided to left the houses in their actual location and build the park in the free space left (the grey area in the picture). Can you calculate the area of the park?

It is guaranteed that in the four corners of the park are square houses, that the consecutive houses are always together and that no house overlaps other.

INPUT

The first line of the input contains an integer T, the number of test cases. Each case begins with an integer N ($4 \leq N \leq 3100$), the number of houses located in the original park space. Next there are N lines, each one representing a house. The first line of this N lines will represent the house located at the upper left corner and all the other houses will appear clockwise. Each one of the lines that represents a house contains a character C and a number K ($1 \leq K \leq 1500$). The character C can has the value of 'S' if it is a square house, 'T' if its a triangle house or 'C' if it is a corner house (remember that if the house is corner, it will be a square house). The number K is the length in meters of one side of the house.

OUTPUT

For each test case prints a single line with the area of the park in meters. This number has to have four digits after the decimal point.

EXAMPLE

Input	Output
1 8 C 4 T 3 C 3 S 2 C 4 S 2 C 4 T 1	20.6699

PROBLEM L. THE REDUNDANT MANUSCRIPT

Source file name:	`manuscript.c, manuscript.cpp, manuscript.java, manuscript.py`
Input/Output:	`standard`
Author('s):	`Gerson Yesid Lázaro Carrillo - UFPS`

Franciscan friar William of Baskerville, a renowned monk and detective, has found in the library of the misterious abbey a very ancient manuscript. Some librarians did not want him to know the contents of the book, and so they have altered it, adding multiple words that weren't in the original version. However, William is an exceptional researcher, and in just 30 seconds he as discovered a detail: The original manuscript does not have any long word more than once (William considers a word with more of 3 letters a long word). The librarians copy words that were already in the manuscript and add them later (they can add it inmediately after, a couple of words or several sentences later but they'll never add it before the first appearence of the word in the original text). For example, they turned the sentence "in the ancient mistery abbey" into "in the ancient mistery ancient abbey mistery abbey". The short words can appear many times in the original text, but William trust that the librarians only added long words.

INPUT

The first line of the input contains an integer T, the number of test cases. Each case has two lines. The first line contains an integer N ($1 \leq N \leq 10^6$), tha amount of words in the text. The second line contains N words separated by spaces, the altered manuscript. All the words are written in lowercase, having characters only in the range $[a - z]$. There is no punctuation signs.

OUTPUT

For each case, print a line with the original version of the text.

EXAMPLE

Input	Output

`3`
`4`
`the abbey of mystery`
`4`
`a mistery a surprise`
`7`
`the abbey of abbey mystery mystery abbey`
	`the abbey of mystery`
	`a mistery a surprise`
	`the abbey of mystery`