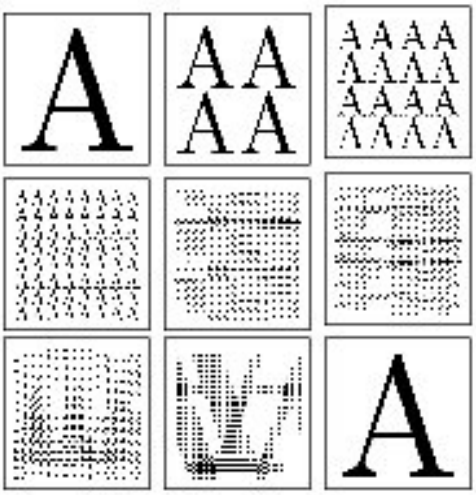
3510 - Pixel Shuffle

Europe - Southwestern - 2005/2006



Shuffling the pixels in a bitmap image sometimes yields random looking images. However, by repeating the shuffling enough times, one finally recovers the original images. This should be no surprise, since ``shuffling" means applying a one-to-one mapping (or permutation) over the cells of the image, which come in finite number.

Problem

Your program should read a number n , and a series of elementary transformations that define a ``shuffling" ϕ of $n \times n$ images. Then, your program should compute the minimal number m ($m > 0$) , such that m applications of ϕ always yield the original $n \times n$ image.

For instance if ϕ is counter-clockwise 90º rotation then $m = 4$.

Input

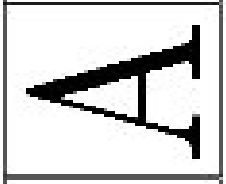
The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

Input is made of two lines, the first line is number n ($2 \leq n \leq 2^{10}$, n even). The number n is the size of images, one image is represented internally by a $n \times n$ pixel matrix (a^j_i) , where i is the row number and j is the column number. The pixel at the upper left corner is at row 0 and column 0.

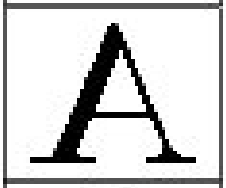
The second line is a non-empty list of at most 32 words, separated by spaces. Valid words are the keywords id, rot, sym, bhsym, bvsym, div and mix, or a keyword followed by ``-". Each keyword key designates an elementary transform (as defined by Figure 1), and key- designates the inverse of transform key. For instance, rot- is the inverse of counter-clockwise 90º rotation, that is clockwise 90º rotation. Finally, the list k_1, k_2,..., k_p designates the compound transform $\Phi = k_1 \circ k_2 \circ \cdots \circ k_p$. For instance, ``bvsym rot-" is the transform that first performs clockwise 90º rotation and then vertical symmetry on the lower half of the image.

Figure 1: Transformations of image (a^j_i) into image (b^j_i)

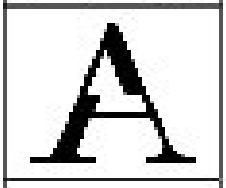
rot , counter-clockwise 90° rotation



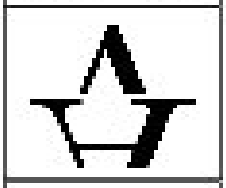
sym , horizontal symmetry : $b_i^j = a_i^{n-1-j}$



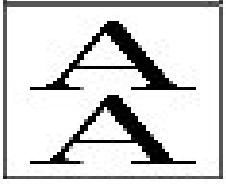
bhsym , horizontal symmetry applied to the lower half of image : when $i \geq n/2$, then $b_i^j = a_i^{n-1-j}$. Otherwise $b_i^j = a_i^j$.



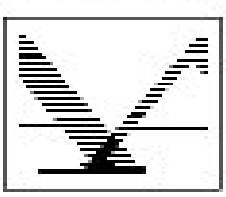
bvsym , vertical symmetry applied to the lower half of image ($i \geq n/2$)



div , division. Rows $0, 2, \ldots, n-2$ become rows $0, 1, \ldots n/2-1$, while rows $1, 3, \ldots n-1$ become rows $n/2, n/2+1, \ldots n-1$.



mix , row mix. Rows $2k$ and $2k+1$ are interleaved. The pixels of row $2k$ in the new image are $a_{2k}^0, a_{2k+1}^0, a_{2k}^1, a_{2k+1}^1, \cdots a_{2k}^{n/2-1}, a_{2k+1}^{n/2-1}$, while the pixels of row $2k+1$ in the new image are $a_{2k}^{n/2}, a_{2k+1}^{n/2}, a_{2k}^{n/2+1}, a_{2k+1}^{n/2+1}, \cdots, a_{2k}^{n-1}, a_{2k+1}^{n-1}$.



Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

Your program should output a single line whose contents is the minimal number m ($m > 0$) such that ϕ^m is the identity. You may assume that, for all test input, you have $m < 2^{31}$.

Sample Input

```
2

256
rot- div rot div

256
bvsym div mix
```

Sample Output

```
8

63457
```

Southwestern 2005-2006

PROBLEM B

B SMS Typing

Input: Standard Input
Output: Standard Output

Cell phones have become an essential part of modern life. In addition to making voice calls, cell phones can be used to send text messages, which are known as SMS for short. Unlike computer keyboards, most cell phones have limited number of keys. To accommodate all alphabets, letters are compacted into single key. Therefore, to type certain characters, a key must be repeatedly pressed until that character is shown on the display panel.

In this problem we are interested in finding out the number of times keys on a cell phone must be pressed to type a particular message.

In this problem we will assume that the key pad of our cell phone is arranged as follows.

```
---------------------
|      | abc | def  |
---------------------
| ghi  | jkl | mno  |
---------------------
| pqrs | tuv | wxyz |
---------------------
|      | <SP>|      |
---------------------
```

In the above grid each cell represents one key. Here **SP** means a space. In order to type the letter 'a', we must press that key once, however to type 'b' the same key must be repeatedly pressed twice and for 'c' three times. In the same manner, one key press for 'd', two for 'e' and three for 'f'. This is also applicable for the remaining keys and letters. Note that it takes a single press to type a space.

Input

The first line of input will be a positive integer **T** where **T** denotes the number of test cases. **T** lines will then follow each containing only spaces and lower case letters. Each line will contain at least 1 and at most 100 characters.

Output

For every case of input there will be one line of output. It will first contain the case number followed by the number of key presses required to type the message of that case. Look at the sample output for exact formatting.

Sample Input

Output for Sample Input

Sample Input	Sample Output
2 welcome to ulab good luck and have fun	Case #1: 29 Case #2: 41

Problemsetter: Shamim Hafiz
Special Thanks: Sohel Hafiz

PROBLEM C

Problem C

Nails

Time Limit: 1 Second

Arash is tired of hard working, so he wants to surround some nails on the wall of his room by a rubber ribbon to make fun of it! Now, he wants to know what will be the final length of the rubber ribbon after surrounding the nails. You must assume that the radius of nails and rubber ribbon is negligible.

Input

The first line of input gives the number of cases, **N**. **N** test cases will follow. Each test case starts with a line containing two integers, the initial length of rubber ribbon and the number of nails **0<n<=100,** respectively. Each of next **n** lines contains two integers denoting the location of a nail. There will be a blank line after each test-case.

Output

Your program must output the final length of rubber ribbon precise to 5 decimal digits.

Sample Input

```
2
2 4
0 0
0 1
1 0
1 1

5 4
0 0
0 1
1 0
1 1
```

Output for Sample Input

```
4.00000
5.00000
```

Problem setter: Ali Sharifrazavian
Alternate Solution: Hossein Azizpour

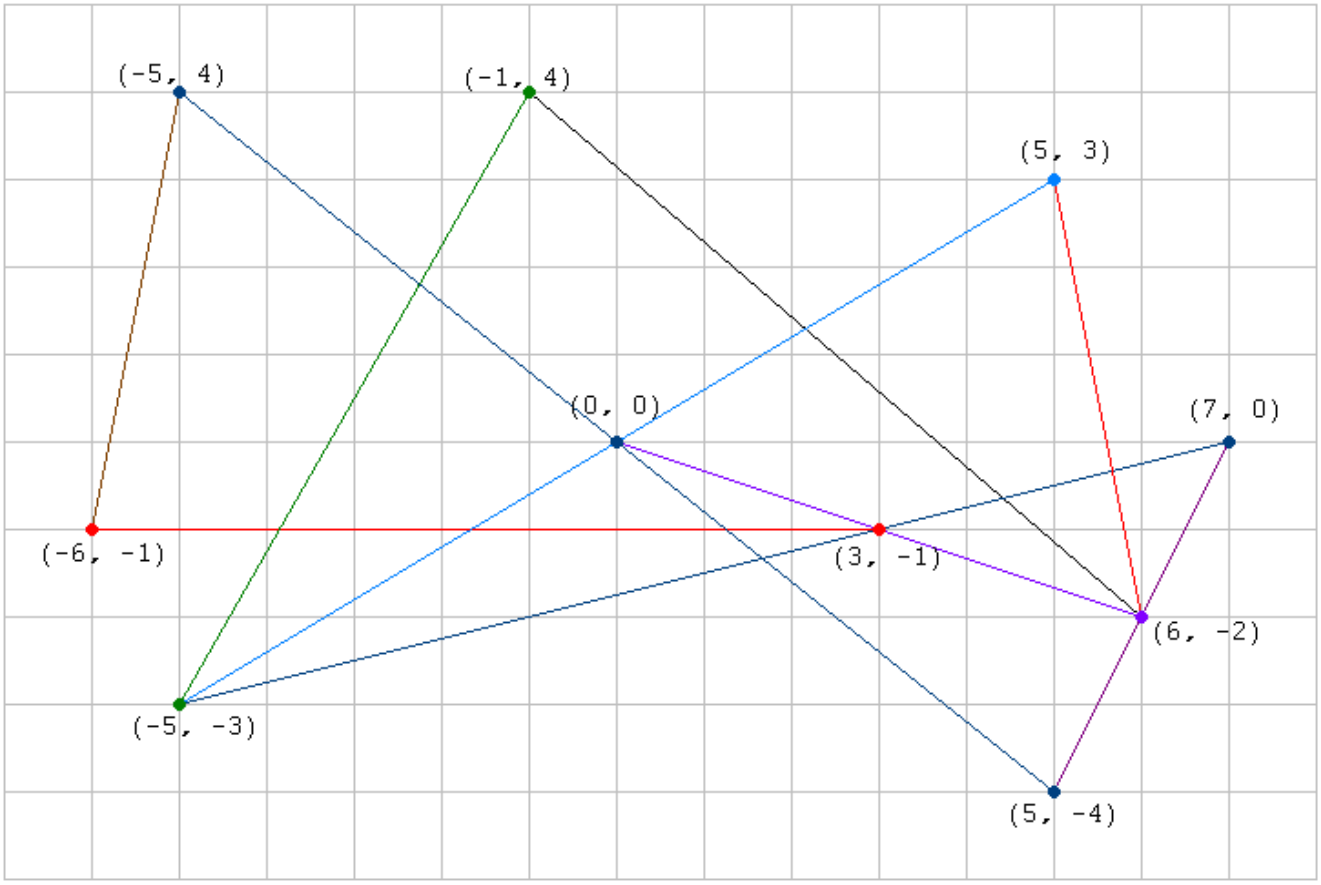
1st Amirkabir UT Annual Programming Contest Qualification Round

PROBLEM D

Problem D: How many triangles?

Time limit: 3 seconds

How many triangles are there in the following figure?



... ding! Time's up. The answer is 29.

Given at most 10 line segments, you are to **write a program** that counts the number of triangles bounded by them. Note that three colinear points do *not* make a triangle.

Input

Input consists of several test cases. Each case begins with a line containing an integer **N** ($3 \le \mathbf{N} \le 10$). The following **N** lines each gives four integers, **x1**, **y1**, **x2** and **y2**, meaning that you draw a straight line segment from (**x1**, **y1**) to (**x2**, **y2**). All the coordinates given have their absolute values no greater than 100.

Input is terminated by EOF.

Output

For each case, output the number of triangles bounded by the **N** lines.

Sample Input

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

Sample Output

```
29
```

Problemsetter: Mak Yan Kei

PROBLEM E

The Postal Worker Rings Once

Background

Graph algorithms form a very important part of computer science and have a lineage that goes back at least to Euler and the famous *Seven Bridges of Königsberg* problem. Many optimization problems involve determining efficient methods for reasoning about graphs.

This problem involves determining a route for a postal worker so that all mail is delivered while the postal worker walks a minimal distance, so as to rest weary legs.

The Problem

Given a sequence of streets (connecting given intersections) you are to write a program that determines the minimal cost tour that traverses every street at least once. The tour must begin and end at the same intersection.

The ``real-life'' analogy concerns a postal worker who parks a truck at an intersection and then walks all streets on the postal delivery route (delivering mail) and returns to the truck to continue with the next route.

The cost of traversing a street is a function of the length of the street (there is a cost associated with delivering mail to houses and with walking even if no delivery occurs).

In this problem the number of streets that meet at a given intersection is called the *degree* of the intersection. There will be at most two intersections with odd degree. All other intersections will have even degree, i.e., an even number of streets meeting at that intersection.

The Input

The input consists of a sequence of one or more postal routes. A route is composed of a sequence of street names (strings), one per line, and is terminated by the string ``*deadend*'' which is NOT part of the route. The first and last letters of each street name specify the two intersections for that street, the length of the street name indicates the cost of traversing the street. All street names will consist of lowercase alphabetic characters.

For example, the name *foo* indicates a street with intersections *f* and *o* of length 3, and the name *computer* indicates a street with intersections *c* and *r* of length 8. No street name will have the same first and last letter and there will be at most one street directly connecting any two intersections. As specified, the number of intersections with odd degree in a postal route will be at most two. In each postal route there will be a path between all intersections, i.e., the intersections are connected.

The Output

For each postal route the output should consist of the cost of the minimal tour that visits all streets at least once. The minimal tour costs should be output in the order corresponding to the input postal routes.

Sample Input

```
one
two
three
deadend
mit
dartmouth
linkoping
tasmania
york
emory
cornell
duke
kaunas
hildesheim
concord
arkansas
williams
glasgow
deadend
```

Sample Output

```
11
114
```

PROBLEM F

Statistics | Problem Statement

Problem Statement for MoneyGame

Problem Statement

Frugal Fred and Lucky Lenny were talking the other day, and Fred challenged Lenny to a game. The rules for the game were as follows:

- Lenny goes first.
- On a player's turn, he must place one coin into an initially empty pot.
- After placing a coin into the pot, the player may then remove any number of coins from the pot, such that the total value of all coins removed is less than the value of the coin placed in the pot.
- The game ends when a player cannot put a coin into the pot on their turn; that player loses. The loser must pay the winner an amount equal to the value of the coins that the winner is holding.

Lenny is afraid that Fred has stacked the game in his favor, so he has asked you to find out the outcome if both players play optimally. There are 3 types of coins. You are given a int[] **values**, where the i-th element is the value of the i-th type of coin. You are also given int[]s **lennysCoins** and **fredsCoins**, where the i-th element of each is the number of coins of the i-th type initially held by Lenny and Fred, respectively.

Return the amount of money Lenny receives in this game, assuming that both play optimally. If Lenny must lose money, return this as a negative number (e.g., if Lenny must lose 2, then this will be returned as -2).

Definition

Class:	MoneyGame
Method:	play
Parameters:	int[], int[], int[]
Returns:	int
Method signature:	int play(int[] lennysCoins, int[] fredsCoins, int[] values)
(be sure your method is public)	

Constraints

- **lennysCoins** and **fredsCoins** will each contain exactly 3 elements.
- Each element of **lennysCoins** and **fredsCoins** will be between 0 and 5, inclusive.
- **values** will contain exactly 3 elements.
- Each element of **values** will be between 1 and 1000, inclusive.

Examples

0)

```
{0,1,1}
{0,1,1}
{20,10,2}
Returns: -2
```

Optimally played, Lenny starts by playing his 2 cost coin. Fred plays his 10 cost coin, taking a 2 cost coin out of the pot. Lenny plays his 10 cost coin, and Fred (holding two 2 cost coins) plays one of them. Lenny has no coins left, and so must pay Fred 2.

1)

```
{0,1,2}
{0,1,1}
{20,10,2}
Returns: 2
```

The same game as Example 0, but in this case the extra coin makes a difference, as Lenny wins 2.

2)

```
{1,1,0}
{0,0,1}
{1,5,10}
```

Returns: 0

This game turns out to be a draw.

3)

{4,4,3}
{4,3,4}
{100,753,100}

Returns: 600

4)

{0,0,0}
{5,5,5}
{1000,1000,1000}

Returns: -15000

Lenny loses a lot of money.

5)

{3,2,1}
{0,0,0}
{6,8,14}

Returns: 42

This problem was used for:
Single Round Match 343 Round 1 - Division I, Level Two

PROBLEM G

Problem G
How Many Pieces of Land?

Input: Standard Input
Output: Standard Output
Time Limit: 3 seconds

You are given an elliptical shaped land and you are asked to choose **n** arbitrary points on its boundary. Then you connect all these points with one another with straight lines (that's **n*(n-1)/2** connections for **n** points). What is the maximum number of pieces of land you will get by choosing the points on the boundary carefully?

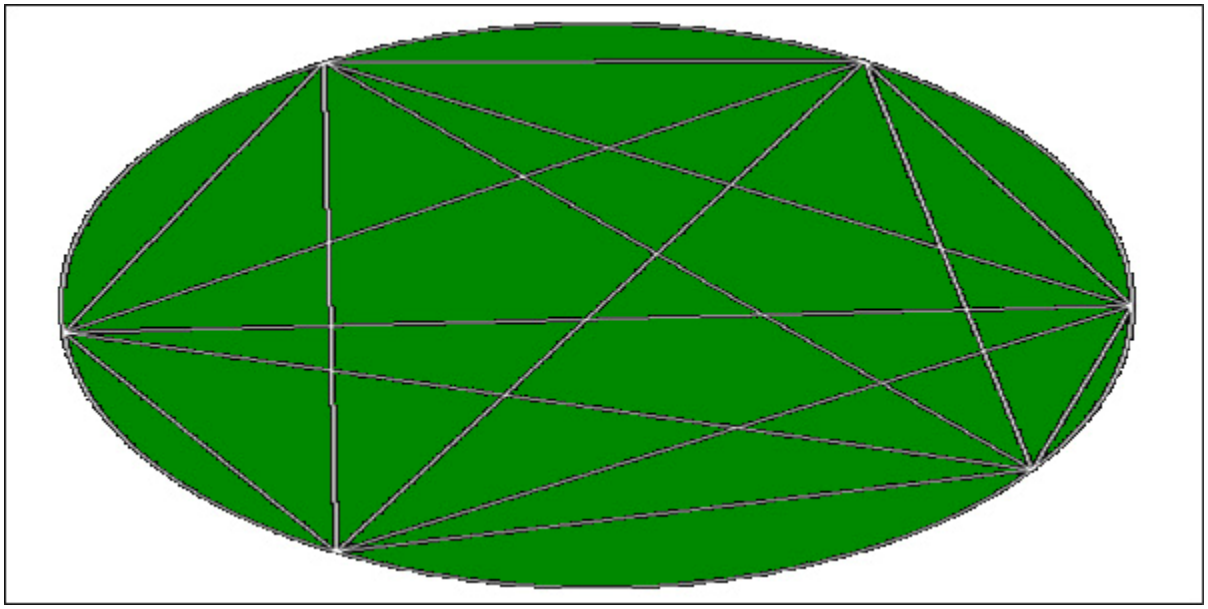


Fig: When the value of n is 6.

Input

The first line of the input file contains one integer **S (0 < S < 3500)**, which indicates how many sets of input are there. The next **S** lines contain **S** sets of input. Each input contains one integer **N (0<=N<2^31)**.

Output

For each set of input you should output in a single line the maximum number pieces of land possible to get for the value of **N**.

Sample Input:

```
4
1
2
```

```
3
4
```

Sample Output:

```
1
2
4
8
```

Shahriar Manzoor

PROBLEM H

SPOJ Problem Set (classical)

97. Party Schedule

Problem code: PARTY

You just received another bill which you cannot pay because you lack the money.
Unfortunately, this is not the first time to happen, and now you decide to investigate the cause of your constant monetary shortness. The reason is quite obvious: the lion's share of your money routinely disappears at the entrance of party localities.
You make up your mind to solve the problem where it arises, namely at the parties themselves. You introduce a limit for your party budget and try to have the most possible fun with regard to this limit.
You inquire beforehand about the entrance fee to each party and estimate how much fun you might have there. The list is readily compiled, but how do you actually pick the parties that give you the most fun and do not exceed your budget?
Write a program which finds this optimal set of parties that offer the most fun. Keep in mind that your budget need not necessarily be reached exactly. Achieve the highest possible fun level, and do not spend more money than is absolutely necessary.

Input

The first line of the input specifies your party budget and the number n of parties.
The following n lines contain two numbers each. The first number indicates the entrance fee of each party. Parties cost between 5 and 25 francs. The second number indicates the amount of fun of each party, given as an integer number ranging from 0 to 10.
The budget will not exceed 500 and there will be at most 100 parties. All numbers are separated by a single space.
There are many test cases. Input ends with 0 0.

Output

For each test case your program must output the sum of the entrance fees and the sum of all fun values of an optimal solution. Both numbers must be separated by a single space.

Example

Sample input:

```
50 10
12 3
15 8
16 9
16 6
10 2
21 9
18 4
12 4
17 8
18 9
```

```
50 10
13 8
19 10
16 8
12 9
10 2
12 8
13 5
15 5
11 7
16 2

0 0
```

Sample output:

```
49 26
48 32
```

Added by: Patryk Pomykalski
Date: 2004-07-01
Time limit: 3s
Source limit: 50000B
Languages: All
Resource: Swiss Olympiad in Informatics 2004

PROBLEM I

Prime Cuts

A prime number is a counting number ($1, 2, 3, \ldots$) that is evenly divisible only by 1 and itself. In this problem you are to write a program that will cut some number of prime numbers from the list of prime numbers between (and including) 1 and *N*. Your program will read in a number *N*; determine the list of prime numbers between 1 and *N*; and print the *C**2 prime numbers from the center of the list if there are an even number of prime numbers or (*C**2)-1 prime numbers from the center of the list if there are an odd number of prime numbers in the list.

Input

Each input set will be on a line by itself and will consist of 2 numbers. The first number ($1 \leq N \leq 1000$) is the maximum number in the complete list of prime numbers between 1 and *N*. The second number ($1 \leq C \leq N$) defines the *C**2 prime numbers to be printed from the center of the list if the length of the list is even; or the (*C**2)-1 numbers to be printed from the center of the list if the length of the list is odd.

Output

For each input set, you should print the number *N* beginning in column 1 followed by a space, then by the number *C*, then by a colon (`:`), and then by the center numbers from the list of prime numbers as defined above. If the size of the center list exceeds the limits of the list of prime numbers between 1 and *N*, the list of prime numbers between 1 and *N* (inclusive) should be printed. Each number from the center of the list should be preceded by exactly one blank. Each line of output should be followed by a blank line. Hence, your output should follow the exact format shown in the sample output.

Sample Input

```
21 2
18 2
18 18
100 7
```

Sample Output

```
21 2: 5 7 11

18 2: 3 5 7 11

18 18: 1 2 3 5 7 11 13 17

100 7: 13 17 19 23 29 31 37 41 43 47 53 59 61 67
```