Colombian Collegiate Programming League
CCPL 2017

Round 5 – May 6

Problems

This set contains 10 problems; pages 1 to 19.

(Borrowed from several sources online.)

Official site `http://programmingleague.org`

Follow us on Twitter `@CCPL2003`

A - Antenna in the Cinoc Mountains

Source file name: `antenna.c`, `antenna.cpp`, `antenna.java`, *or* `antenna.py`

Cinoc is a mountain region that offers a spectacular natural environment. The area is sometimes called the last wilderness area, but in reality the Cinoc Mountains area is a cultural landscape. The mountains of Cinoc are also an important recreational area for people from around the world and have a particular characteristic: all of them have conic form (a right cone with circular base) and are placed on a completely flat land.

In order to attend any possible emergency, the administration of the Cinoc National Park requires to install two towers of communication, to connect two distant places in the park. The connection between the towers is only possible if there is a *line of sight* between the two towers, that is to say, the highest point of one tower can be seen from the highest point of the other tower. Also, to avoid interferences, the distance from any point in the line of sight to any point on the territory must be greater than zero.

Your task is to write a program that, given a map of the park and the description of the two towers, decides if the connection is possible or not.

Input

The input consists of several test cases. For each test case: the first line contains an integer number k, $0 \leq k \leq 50$, and is followed by $k + 2$ lines. In each case the number k is the number of mountains in Cinoc Mountains region. Each of the next k lines contains four positive, integer numbers xm, ym, hm and rm, $0 \leq xm, ym, hm, rm \leq 10000$, describing a conic mountain ((xm, ym, hm) denotes the coordinates of the top of the mountain and rm denotes the radius of the base). The last two lines contain the information associated with the towers: three numbers per line: xt, yt and ht, $0 \leq xt, yt, ht \leq 10000$, ($(xt, yt, ht)$ denotes the location of the tower's top). The last case is followed by a single line containing `-1`.

The input must be read from standard input.

Output

For each test case, if there exists a valid line of sight between the corresponding top points of the towers, the following line must be printed:

```
Yes
```

In other case, the following line must be printed:

```
No
```

The answers for the different cases must preserve the order of the input.

The output must be written to standard output.

Sample Input	**Sample Output**

```
2
10 10 2 5
10 2 2 2
10 2 3
2 2 2
1
10 2 2 2
5 2 1
15 2 1
-1
```
	```
	Yes
	No
	```

B - Laszlo Babai

Source file name: `babai.c`, `babai.cpp`, `babai.java`, *or* `babai.py`

László Babai is a Hungarian computer scientist and mathematician. He is a Gödel prize winner and an outstanding researcher in the fields of the theory of computation, algorithms, combinatorics, and group theory. Last year, he proposed a subexponential-time algorithm solving Graph Isomorphism in $\exp\left((\log n)^{O(1)}\right)$-time, and the best previous result is an $\exp\left(O\left(\sqrt{n \log n}\right)\right)$-time algorithm.

Graph Isomorphism is a famous NP problem in theoretical computer science, however, you may wonder what it is. Let us explain for a bit. Given two undirected graphs $A = (V_A, E_A)$ and $B = (V_B, E_B)$, where A's vertex set is $V_A = \{a_1, a_2, a_3, \ldots, a_{nA}\}$, and B's vertex set is $V_B = \{b_1, b_2, b_3, \ldots, b_{nB}\}$, graphs A and B are isomorphic if and only if

- A and B have the same amount of vertices and edges,
- There exists a bijective (one-to-one and onto) function $f : V_A \to V_B$ such that $\{u, v\} \in E_A$ if and only if $\{f(u), f(v)\} \in E_B$.

In other words, we can relabel the vertex set of graph A to obtain graph B (and viceversa).

Graph Isomorphism is still neither known to be in P nor NP-complete. As up and coming computer scientists, we must be ambitious and never be afraid to dream big! Therefore, let us take on the challenge of testing if two 3-vertex undirected simple graphs G_1 and G_2 are isomorphic and show the world that we too can accomplish something.

Input

The first line of the input will be a single integer T ($T \leq 100$) representing the number of test cases that will follow. Every test case then starts with the number of edges m ($0 \leq m \leq 3$) in the first undirected simple graph of 3 vertices (numbered from 1 to 3), followed by m lines each containing two distinct integers $u, v (u \neq v, u, v \in \{1, 2, 3\})$ indicating that there exists an edge between vertex u and v. You may assume that there is at most one edge between any pair of vertices. After that the description of the second graph follows in the same format.

The input must be read from standard input.

Output

If the two graphs are isomorphic than output "yes" on one line. If not, output "no" instead.

The output must be written to standard output.

Sample Input	**Sample Output**

3
3
1 2
2 3
3 1
3
1 3
2 1
3 2
2
1 2
1 3
0
1
2 3
1
1 2
	yes
	no
	yes

C - Caesar Word Salad

Source file name: `caesar.c`, `caesar.cpp`, `caesar.java`, *or* `caesar.py`

The year is 46, B.C. Gaius Julius Caesar is the most powerful ruler in the world.

The newly named "imperator", busy consolidating his power in Rome and waging wars abroad, needs a secure way to send messages to his representatives around the Roman world.

The now world-famous solution to this problem is the eponymous Caesar cipher, an encoding in which every letter from the plaintext is rotated through the alphabet by a fixed *shift distance*. For example, when encoded using a shift distance of 3 , `alexandra` becomes `dohadqgud`.

Caesar is a master of strategy – he knows the value of an effective double-bluff and will not hesitate to use one. In fact, when possible, he will even send some messages with a shift distance of 0 to really confuse his enemies. However, after a run in with the soothsayer Spurina, Caesar has become a worried man. He will hear no talk of any "`i`"s of March and so, he wants to send only messages where the ciphertext contains no "`i`"s!

Given the plaintext of a message, how many distinct "i-free" shifts could be used for the encoding?

Input

The input consists of several test cases. Each test case consists of a line containing the plaintext w: it will be at least one character long and no more than 100 characters in length. It will contain only lower-case letters.

The input must be read from standard input.

Output

For each test case output a single line. If at least one "i-free" shift of w can be found, output the number of distinct shift distances that could be used. Otherwise, output `impossible`.

You may consider 0 as a valid shift distance provided it does not lead to any "i"s.

The output must be written to standard output.

Sample Input	**Sample Output**
`erratum`	`20`
`agricola`	`19`
`thequickbrownfoxjumpsoverthelazydog`	`impossible`

D - Bipartite Blanket

Source file name: `blanket.c`, `blanket.cpp`, `blanket.java`, *or* `blanket.py`

In a bipartite graph, nodes are partitioned into two disjoint sets A and B such that every edge connects a node from A with a node from B. A matching M is a set of edges where no two edges share a common node. We say that a matching M blankets a set of nodes V if every node in V is an endpoint of at least one edge in M.

We are given a bipartite graph where each node is assigned a weight – a positive integer. Weight of a set of nodes is simply the sum of the weights of the individual nodes.

Given an integer threshold t, find the number of different sets of nodes V such that the weight of V is at least t and V is blanketed by at least one matching M.

Input

The input consists of several test cases. The first line contains two integers n and m ($1 \leq n, m \leq 20$) – the number of nodes in A and B respectively. Let us denote the nodes of A with $a_1, a_2, \ldots, a_n$ and the nodes of B with $b_1, b_2, \ldots, b_m$. The following n lines contain m characters each that describe the edges of the graph. The j-th character in the i-th line is "1" if there is an edge between a_i and b_j and "0" otherwise. The following line contains n integers $v_1, v_2, \ldots, v_n$ ($1 \leq v_k \leq 10\,000\,000$) – the weights of the nodes $a_1, a_2, \ldots a_n$. The following line contains m integers $w_1, w_2, \ldots, w_m$ ($1 \leq w_k \leq 10\,000\,000$) – the weights of the nodes $b_1, b_2, \ldots b_m$. The following line contains an integer t ($1 \leq t \leq 400\,000\,000$) – the weight threshold.

The input must be read from standard input.

Output

For each test case, output the number of sets of nodes whose weight is at least t and are blanketed by at least one matching M.

The output must be written to standard output.

Sample Input

```
3 3
010
111
010
1 2 3
8 5 13
21
3 2
01
11
10
1 2 3
4 5
8
```

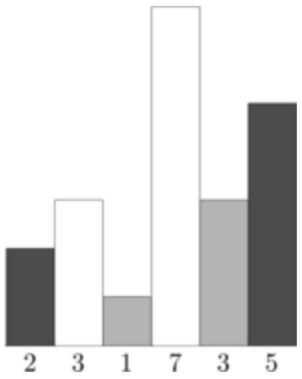
Sample Output

```
3
13
```

E - Rectangle

Source file name: `rectangle.c`, `rectangle.cpp`, `rectangle.java`, *or* `rectangle.py`

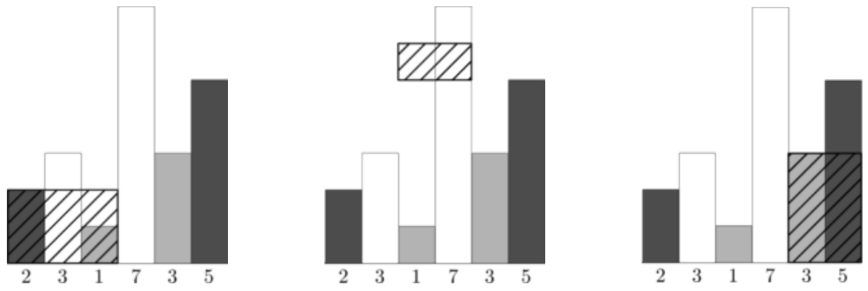
A *histogram* is a polygon composed of a sequence of rectangular bars aligned at a common base line. All bars have equal widths but may have different integer heights. Additionally, each bar is fully colored with a single color. For example, the figure below shows the histogram that consists of bars with the heights 2, 3, 1, 7, 1, 3, 5, measured in units where 1 is the width of each bar. In this case, every bar is colored with one out of three colors.



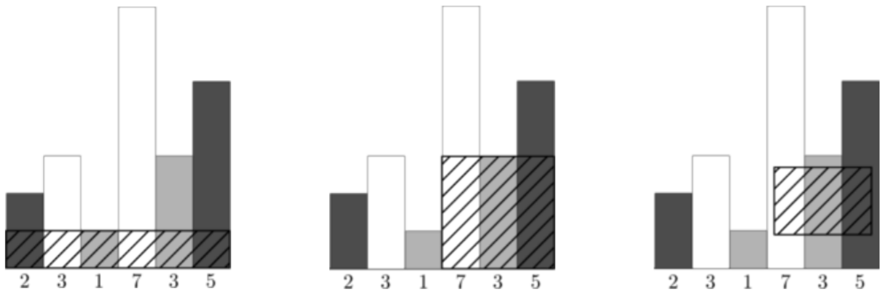
There are a lot of rectangles that can be placed inside this histogram (infinitely many, in fact). We will say that a rectangle is nice if and only if:

- It lies completely inside the histogram.
- It covers one or several regions of positive area for every color in the histogram (barely touching an area is not enough; the total area covered for each color must be strictly positive).

Here are some examples of rectangles that are not nice:



The rectangles in the two leftmost figures are not nice because they don't lie completely inside the histogram. The rectangle in the rightmost figure is not nice because the total white area covered is not strictly positive, and a nice rectangle must cover a positive area of every color. On the other hand, here are some examples of nice rectangles:



The area of the nice rectangle in the leftmost figure is $6 \times \times 1 = 6$ square units, the area of the nice rectangle in the middle figure is $3 \times 3 = 9$ square units and the area of the nice rectangle in the rightmost figure is $1.8 \times 2.6 = 4.68$ square units.

Clearly, the area of the rectangle in the middle figure is largest. In fact, it turns out it doesn't exist a nice rectangle for this histogram with a larger area!

In this problem, you are given a histogram and your task is to compute the maximum area of all the possible nice rectangles. By the way, it is not hard to prove that this area will always be an integer.

Input

The input contains several test cases (at most 50). Each test case is described by several lines. The first line contains two integer N and C, the number of bars and the total number of colors in the histogram, respectively ($1 \leq N \leq 105$ and $1 \leq C \leq min(30, N)$). The following line contains N integers h_i, the height of the i-th bar from left to right ($1 \leq h_i \leq 10^9$). The following line contains N integers c_i, the color of the i-th bar from left to right. Each color is represented as an integer between 0 and $C - 1$, inclusive. You can assume that for each histogram there will be at least one bar of every color.

The last line of the input contains two zeros and should not be processed.

The input must be read from standard input.

Output

For each test case, output a single integer on a single line – the area of the largest nice rectangle for this histogram.

The output must be written to standard output.

Sample Input

```
6 3
2 3 1 7 3 5
2 0 1 0 1 2
3 1
2 2 2
0 0 0
5 2
3 2 1 2 3
1 0 1 0 1
6 4
1 2 3 4 5 6
0 1 2 3 1 0
7 2
1 2 3 4 3 2 1
0 1 0 1 0 1 0
10 2
2 1 2 1 1 2 1 2 2 1
1 0 0 0 1 0 0 1 1 0
3 2
1000000000 999999997 999999999
0 1 1
0 0
```

Sample Output

```
9
6
5
12
10
10
2999999991
```

F - Robert Flow

Source file name: `flow.c`, `flow.cpp`, `flow.java`, *or* `flow.py`

Robert W Floyd was an American computer scientist who was also awarded the Turing Award. In the ACM-ICPC community, his most known work is probably the Floyd-Warshall algorithm which can find all-pairs shortest paths and transitive closure efficiently. It has many applications and many of them is very useful, but the following is not one of them.

Stitches is the terror of Darkshire. He emits a putrid bile along his path that is stinky, disgusting and slows you by 35% percent when walking in it. So in short, you don't want to stand in them. With Stitches' movement, he can potentially cut the map into several separated areas. Two areas are separated if there is no way one can reach the other without crossing Stitches' bile. Stitches always walks along a single direction (up, down, left or right) for a unit of length and then decides whether to change his direction or not. In order to bring peace to Darkshire, the heroes will kill him as soon as possible. You may assume that Stitches only emits bile for at most 2048 units of length, since the heroes will not allow him to walk more.

Unfortunately, Stitches' bile does not disappear after his death. As the mayor of Darkshire, you want to determine how many separated areas there are after Stitches walks through. So how can we solve this with the Floyd-Marshall algorithm and why is it not useful? Well, because of how Stitches move, we can imagine he is walking on the edges of a square grid of 4098×4098 cells. Stitches begins his walk at the center of the grid, thus he will never touch its boundary. Thus easily transform the map into a graph where neighboring blocks on the grid are connected if Stitches did not walk on its connecting edge. After running the Floyd-Marshall algorithm, we can easily determine whether two blocks are in the same area. Then, just simply counting the number of areas by using a data structure for storing disjoint sets. However, since Darkshire is not a small town and there can be more than 16 millions of blocks on the grid, this method will simply not be fast enough. Unless you have the help of bronze dragon Chromie, the Keeper of Time. Please find a better way to solve this problem. Again, cheating with the bronze dragon is not allowed.

Input

On the first line there is a single integer T ($T \leq 30$) indicating the number of test cases. For every test case, the sequence of Stitches' movement will be given on a line with 'U 'as up, 'D 'as down, 'L 'as left and 'R 'as right. The length of the sequence will not exceed 2048.

The input must be read from standard input.

Output

Output the number of separated areas on one line.

The output must be written to standard output.

Sample Input	Sample Output
4 LURD LUDR LDRDRURRDLUURURD LLUURRDDRULLLLUR	2 1 2 5

G - Convex Contour

Source file name: `convex.c`, `convex.cpp`, `convex.java`, *or* `convex.py`

A number of geometric shapes are neatly arranged in a rectangular grid. The shapes occupy consecutive cells in a single row with each cell containing exactly one shape. Each shape is either:

- a square perfectly aligned with the grid square,
- a circle inscribed in the grid square,
- or an equilateral triangle with a side corresponding to the bottom side of the grid square.

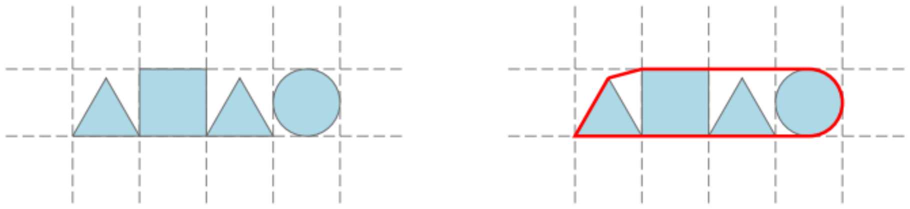


Figure 1: The shapes from the first example input and their convex contour.

Informally, the *convex contour* of an arrangement is the shortest line that encloses all the shapes. Formally, we can define it as the circumference of the convex hull of the union of all shapes.

Given an arrangement of shapes, find the length of its contour.

Input

The input consists of several test cases. The first line contains an integer n ($1 \leq n \leq 20$) - the number of shapes. The following line contains a string consisting of n characters that describes the shapes in the arrangement left to right. Each character is an uppercase letter “S”, “C” or “T” denoting a square, circle or a triangle respectively.

The input must be read from standard input.

Output

For each test case, output a single floating point number – the length of the contour rounded to 6 decimal places.

The output must be written to standard output.

Sample Input	Sample Output
4 TSTC 3 SCT	9.088434 7.509142

H - Hamiltonian Hypercube

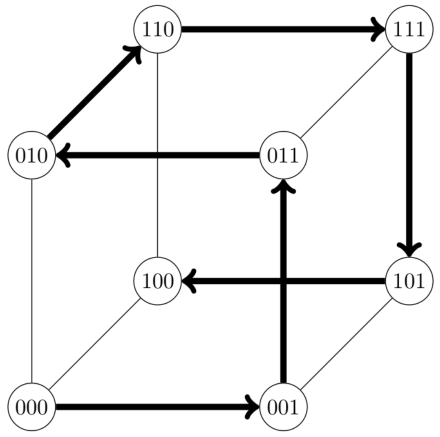
Source file name: `hamilton.c`, `hamilton.cpp`, `hamilton.java`, *or* `hamilton.py`

Hypercube graphs are fascinatingly regular, hence you have devoted a lot of time studying the mathematics related to them. The vertices of a hypercube graph of dimension n are all binary strings of length n, and two vertices are connected if they differ in a single position. There are many interesting relationships between hypercube graphs and error-correcting code.

One such relationship concerns the n-bit Gray Code, which is an ordering of the binary strings of length n, defined recursively as follows. The sequence of words in the n-bit code first consists of the words of the $(n-1)$-bit code, each prepended by a 0, followed by the same words in reverse order, each prepended by a 1. The 1-bit Gray Code just consists of a 0 and a 1. For example the 3-bit Gray Code is the following sequence:

$$000, 001, 011, 010, 110, 111, 101, 100$$

Now, the n-bit Gray Code forms a Hamiltonian path in the n-dimensional hypercube, i.e., a path that visits every vertex exactly once:



You wonder how many vertices there are between the vertices 0^n (n zeros) and 1^n (n ones) on that path. Obviously it will be somewhere between $2^{n-1} - 1$ and $2^n - 2$, since in general 0^n is the first vertex, and 1^n is somewhere in the second half of the path. After finding an elegant answer to this question you ask yourself whether you can generalise the answer by writing a program that can determine the number of vertices between two arbitrary vertices of the hypercube, in the path corresponding to the Gray Code.

Input

The input consists of several test cases. Each test case is a single line containing one integer n ($1 \leq n \leq 60$) indicating the dimension of the hypercube, and two binary strings a and b, both of length n, where a appears before b in the n-bit Gray Code.

The input must be read from standard input.

Output

For each test case, output the number of code words between *a* and *b* in the *n*-bit Gray Code.

The output must be written to standard output.

Sample Input	Sample Output

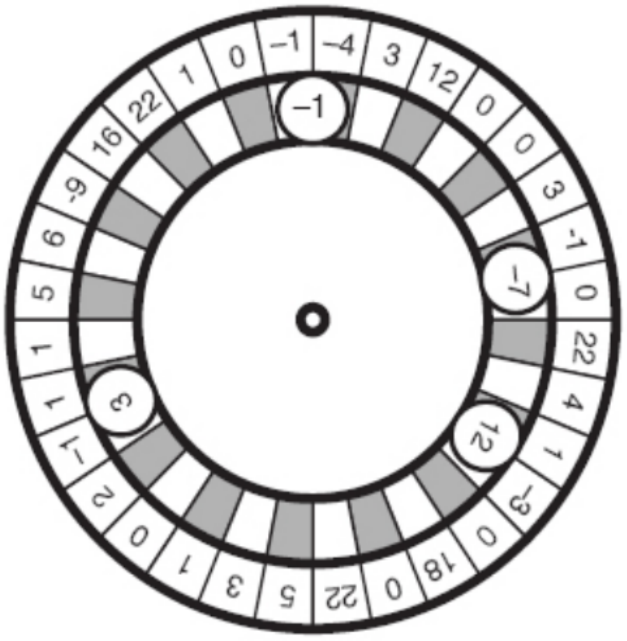
`3 001 111`
`3 110 100`
	`3`
	`2`

I - Turkish Roulette

Source file name: `turkish.c`, `turkish.cpp`, `turkish.java`, *or* `turkish.py`

Turkish Roulette is a betting game that uses a roulette with S slots, each one numbered with an integer between -64 and 64. In each turn of the game players bet on B balls, each one also numbered from -64 to 64. For each of the B balls, exactly one player will bet on it.

After spinning the roulette, the dealer throws in the B balls sequentially. When the roulette stops, each ball is lodged over two (adjacent) slots, as depicted in the figure below, which shows a roulette with thirty two slots and four balls. Notice that, as the figure illustrates, a ball occupies the space of two adjacent slots, and therefore there is room for at most $\lfloor S/2 \rfloor$ balls in the roulette.



Balls end up lodged in the same relative positions that they were thrown in the roulette. That is, if balls a, b and c are thrown in that sequence, they end up lodged such that, in clockwise direction, a is followed by b which is followed by c which is followed by a.

The value of a ball in a turn is calculated by multiplying the ball's number by the sum of the numbers of the two adjacent slots over which the ball is lodged. If a ball's value is positive, the player who bet on that ball receives that amount (the ball's value) from the dealer; if a ball's value is negative, the player who bet on that ball must pay that amount to the dealer. The profit of the dealer in a turn is the total amount received minus the total amount paid.

For example, in the figure above, the dealer pays \$5.00 for ball numbered -1, pays \$7.00 for ball numbered -7, receives \$24.00 for ball numbered 12 and does not pay nor receive anything for ball numbered 3. Therefore, in this turn the dealer makes a profit of \$12.00 (24 - 5 - 7); note that the dealer's profit in a turn may be negative (loss).

You will be given the description of the roulette, the description of the balls and the sequence in which the balls are thrown into the roulette. Write a program to determine the maximum profit that the dealer can make in one turn.

Input

Input contains several test cases. The first line of a test case contains two integers S and B which indicate respectively the number of slots in the roulette ($3 \leq S \leq 250$) and the number of balls used ($1 \leq B \leq \lfloor S/2 \rfloor$). The second line of a test case contains S integers X_i, indicating the numbers associated to the roulette's slots, in clockwise direction ($-64 \leq X_i 64$, for $1 \leq i \leq S$). The third line of a test case contains B integers Y_i, indicating the numbers associated to the balls ($-64 \leq Y_i \leq 64$, for $1 \leq i \leq B$), in the sequence the balls are thrown into the roulette (notice it is in this order that they end lodged in the roulette, in clockwise direction). The end of input is indicated by $S = B = 0$. The input must be read from standard input.

The input must be read from standard input.

Output

For each test case in the input your program must write one line of output, containing an integer indicating the maximum profit the dealer can make in one turn. The output must be written to standard output.

The output must be written to standard output.

Sample Input

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

Sample Output

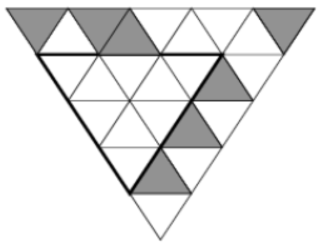
```
 4
-11
 56
 10
```

J - Triangles

Source file name: `triangles.c`, `triangles.cpp`, `triangles.java`, *or* `triangles.py`

It is always very nice to have little brothers or sisters. You can tease them, lock them in the bathroom or put red hot chili in their sandwiches. But there is also a time when all meanness comes back!

As you know, in one month it is Christmas and this year you are honored to make the big star that will be stuck on the top of the Christmas tree. But when you get the triangle-patterned silver paper you realize that there are many holes in it. Your little sister has already cut out smaller triangles for the normal Christmas stars. Your only chance is to find an algorithm that tells you for each piece of silver paper the size of the largest remaining triangle.



Given a triangle structure with white and black fields inside you must find the largest triangle area of white fields, as shown in the figure on the right.

Input

The input file contains several triangle descriptions. The first line of each description contains an integer n ($1 \leq n \leq 100$), which gives the height of the triangle. The next n lines contain characters of the set space, #, - representing the rows of the triangle, where '#'is a black and '-'a white field. The spaces are used only to keep the triangle shape in the input by padding at the left end of the lines. (Compare with the sample input. The first test case corresponds to the figure.

For each triangle, the number of the characters '#'and '-'per line is odd and decreases from $2n - 1$ down to 1. The input is terminated by a description starting with $n = 0$.

The input must be read from standard input.

Output

For each triangle in the input, first output the number of the triangle, as shown in the sample output. Then print the line **'`The largest triangle area is` *a*.'**, where *a* is the number of fields inside the largest triangle that consists only of white fields. Note that the largest triangle can have its point at the top, as in the second case of the sample input. Output a blank line after each test case.

The output must be written to standard output.

Sample Input	**Sample Output**

```
5
#-##----#
 -----#-
  ---#-
   -#-
    -
4
#-#-#--
 #---#
  ##-
   -
0
```
	```
	Triangle #1
	The largest triangle area is 9.

	Triangle #2
	The largest triangle area is 4.
	```