Problem A: Ticket Discount

Input file name: ticket.in

David, a manager of a world top amazing park – TheBEST, plans to invest $100K to develop an automatic system to calculate the price of the park ticket under various conditions.

The standard price of TheBEST is 100$ per entry normally. Moreover, to attract more and more tourists, TheBEST published many discounts for the entry ticket. So far, the discount policies include:

1. 40% discount for booking the ticket 30 days in advance (including 30 days). For example, if the ticket of Nov.1 2003 is booked before Oct.3 2003, the price is 60$.
2. 30% discount for booking the ticket 15 days in advance (including 15 days). For example, if the ticket of Nov.1 2003 is booked before Oct.18 2003, the price is 70$.
3. 10% discount for booking the ticket 7 days in advance (including 7 days). For example, if the ticket of Nov.1 2003 is booked before Oct.26 2003, the price is 90$.
4. 10% discount for booking by phone directly.
5. 20% discount for booking by travel agency.
6. 15% discount for booking by online booking system.
7. 10% discount for booking 5-19 tickets per time.
8. 20% discount for booking 20 tickets or above per time.

For one booking transaction, three discount types - advance booking (1, 2, 3), none window booking (4, 5, 6) and group booking (7, 8) can be applied for together. To illustrate the above discount policies, we have the following examples booking by today (Oct.26, 2003).

Booking 1: 1 ticket for Nov.2, 2003 by phone
(1-10%)(1-10%)*100=81$*

Booking 2: 5 tickets for Dec.1, 2003 by travel agency <China-Tour>
(1-40%)(1-20%)*(1-10%)*100=43.2$*

Booking 3: 30 tickets for Oct.27, 2003 by online system
(1-15%)(1-20%)* 100=68$*

You task is to write a program to help TheBEST calculate the exact price per ticket for each booking transaction.

Input (from ticket.in)

There are several booking transactions in the input file, each transaction per line.

A transaction is shown as following format (4 items):
BookingDate EntryDate BookingWay Quantity

And then, the format for each item is:
BookingDate: MMM.DD YYYY
EntryDate: MMM.DD YYYY
, where MMM represents the month (“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Sep”, “Oct”, “Nov” and “Dec”), DD represents the 2 digits format of the day and YYYY represents the 4 digits format of the year.

Quantity: 3 digits from 001 to 999
BookingWay: “phone” or “online” or “agency <S>”, where S is a string shows the agency information (length of S is within 100).

When Quantity equals to “000”, the input file ends.

Output

For each booking transaction, output the final price per ticket, one per line.

Sample Input

Oct.26 2003 Nov.02 2003 phone 001
Oct.26 2003 Dec.01 2003 agency <World Trade Reservation 207548> 005
Oct.26 2003 Oct.27 2003 online 030
Oct.23 2003 Oct.23 2003 online 000

Sample Output

81
43.2
68

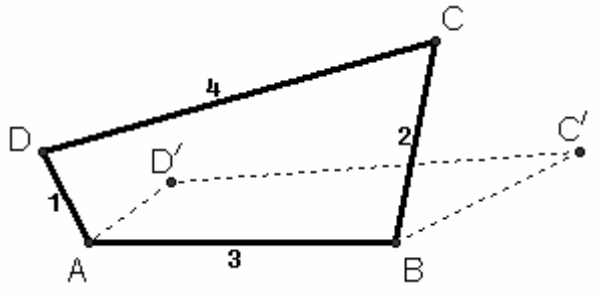
Problem B: Area of Polygon

Input file name: polygon.in

For a n-sided simple polygon, if we know the coordinates of all its n vertices, for example counterclockwise (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), the area of this polygon can be computed by so-called **the Surveyor's Formula**:

$$Area = \left| \frac{1}{2} \left(\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + \cdots + \begin{vmatrix} x_{n-1} & x_n \\ y_{n-1} & y_n \end{vmatrix} + \begin{vmatrix} x_n & x_1 \\ y_n & y_1 \end{vmatrix} \right) \right|$$

But what if we only know the lengths of all n edges of a polygon? In the case of $n = 3$ (triangle), the area can be determined by the well-known **Hero's Formula**. However, when $n \geq 4$, things become complicated because the area could be greatly affected by the angles, which is illustrated in the following figure:



Here comes our problem: given n sticks of length 1, 2, 3, …, n, you are required to construct a n-sided polygon using these n sticks as the polygonal edges. And the objective is to arrange the sticks and choose the angles so that the area of this polygon is maximized.

Input (from polygon.in)

The input for this problem contains several configurations, one per line.
For each line, there is only one integer n ($3 \leq n \leq 100$).
A line with a single -1 indicates the end of the input.

Output

The output should be one real number per line, shows the maximized area, correct to four decimal places.

Sample Input

4
6
-1

Sample Output

4.8990
29.2490

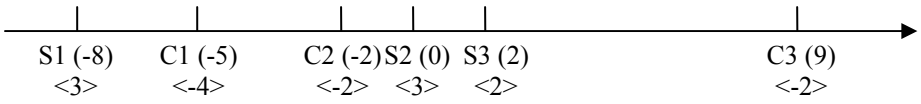
Problem C: Face Mask Transportation

Input file name: mask.in

This year, human being in the world won a tough fight with the new disease - Severe Acute Respiratory Syndrome (SARS) after long time effort. From the fight, we learnt the importance of the preparation for coming uncertain events. From the battle with SARS, the face mask becomes one of the key items in people's mind. Everyone needs face mask that results in the huge business market of face masks. Besides high quality masks, their rapid delivery from the suppliers to consumers is required as well.

Now there is a problem for you to help a transportation company to finish the delivery task for face masks as soon as possible. Assume that all suppliers and consumers are located in a linear line such as railway or ship route. Suppliers (or consumers) have various supplying capacities (consuming requirements) for face mask. We use positive (or negative) integer to represent the capacity for supplier (or requirement for consumer). Moreover, the company has only one vehicle (train or ship), starting in location zero move back and force along the line to pick up and delivery all suppliers' products to all consumers on their requirements. Finally, the vehicle should move back to its initial location zero. In addition, the vehicle has a maximum capacity, so that the total amount of the face masks on the vehicle can not exceed its maximum capacity at any time.

We illustrate an example in the following figure. There are three suppliers S1, S2 and S3 that located in location -8, 0 and 2 respectively. The supplying capacity for each supplier is 3, 3 and 2. On the other hand, there are three consumers C1, C2 and C3 that located in location -5, -2 and 9 with the requirements -4, -2 and -2. The initial location of the vehicle (assume its maximum capacity is 3) is location 0.



One possible route of the vehicle to finish the task is:

(1) Pick up mask 3 at S2
(2) Move to C3, deliver mask 2 at C3
(3) Move to S3, pick up mask 2 at S3
(4) Move to C2, deliver mask 2 at C2
(5) Move to C1, deliver mask 1 at C1
(6) Move to S1, pick up mask 3 at S1
(7) Move to C1, delver mask 3 at C1
(8) Move back to initial location 0

The total distance of the above route is 34.

If we input the locations, capacities (or requirements) of several suppliers (or consumers), would you mind help to find an optimal route solution to minimize the total distance of the route?

Input (from mask.in)

There are several test cases in the input file. Each test case has three parts. The first is a line of three integers *S*, *C* and *P* to represent the number of suppliers, the number of consumers and the maximum capacity of the vehicle ($1 \le S \le 100$, $1 \le C \le 100$, $1 \le P \le 100$), respectively. The second part includes *S* lines with two integers each. The first integer indicates the location of the supplier in the line, and the second one denotes its supplying capacity. The third part includes *C* lines with two integers each. The first integer represents the location of the consumer in the line, and the second integer indicates its requirement.

The input file ends when *P* equals zero.

Note: Assume the sum of all supplying capacities is equal to the absolute value of the total consumer requirements in all test cases.

Output

For each case in the input file, just output the minimum total distance of the optimal solution your program finds in one separate line.

Sample Input

```
3 3 3
-8 3
0 3
2 2
-5 -4
-2 -2
9 -2
0 0 0
```

Sample Output

```
34
```

Problem D: *k*-longest Common Sequence

Input file name: klong.in

Finding the longest common subsequence between DNA/Protein sequences is one of the basic problems in modern computational molecular biology. We state the problem formally as follows:

A sequence $x=x_1x_2\ldots x_n$ over finite set $P=\{A,B, C, \ldots, Z\}$ may be any combination of n characters from P, e.g. x_i in P and x in P^*. The length of x is denoted as $|x|=n$. Given a sequence x, we call another sequence $y=y_1y_2\ldots y_m$ a subsequence of x, if there exists an embedding $I=(i_1,i_2,\ldots,i_m)$ so that $1\le i_1<i_2<\ldots<i_m\le|x|$ and $x_{ik}=y_k$, for all $k=1,2,\ldots m$. In addition, we define $s(x)=\{y|\ y$ is a subsequence of $x\}$.

The problem of *k*-longest common sequence is that give *k* sequences $x(1)$, $x(2)$, …, $x(k)$, find the longest common subsequence y so that y in $s(x(i))$, for all $i=1,2,\ldots k$ and $|y|$ is maximized.

To help you understand the *k*-longest common sequence problem more clearly, there is an illustration. For two sequences ATTA and CGGC, their 2-longest common sequence is none with length 0. For three sequences GAACCACGCG, ACCGAC and GCCACCAAGC, their 3-longets common sequence is ACCAC with length 5.

Your task is to write a program to help the scientist to find the length of the *k*-longest common sequence of a given *k* DNA/Protein sequences.

Note: one subsequence of *x* may have several embeddings in *x*. For example, AG is a subsequence of AACGG, which has 4 embeddings: (1,4), (1,5), (2,4) and (2,5).

Input

The input file contains several test cases. In each case, the first line is an integer represents *k* ($1<k<100$). The following *k* lines contain the *k* sequences, one per line. The maximal length of each sequence is 500.
The input file ends when *k* is equal to 0.

Output

For each case in the input file, simply output the length of the *k*-longest common sequence, one per line.

Sample Input (from file: klong.in)

```
2
ATTA
CGGC
3
GAACCACGCG
ACCGAC
GCCACCAAGC
0
```

Sample Output

0
5

Problem E: Radius of the Holedox

Input file name: holedox.in

An ironclad disk, named *Holedox*, planned to attack the Mars. It would move in a plane, under the control of an automated program. The automated program is a sequence of (*dx*,*dy*) pairs, indicating differences of *x*-coordinates and *y*-coordinates between two contiguous positions of the disk center in the plane. For each movement, say from (*x*,*y*) to (*x*+*dx*,*y*+*dy*), the *Holedox* would keep still at (*x*,*y*) for a period, move to (*x*+*dx*,*y*+*dy*) rapidly, and then keep still again at (*x*+*dx*,*y*+*dy*) to wait for the next movements.

However, the *Holedox* would be detected by the Mars Security Bureau (MSB), who set up *m* sensors given by their coordinates (x_i,y_i) in the plane where $1 \le i \le m$. Because the *Holedox* moved incredibly fast, it could be detected only when it was keeping still. At that time, every sensor would activate its state to 1 when it was inside *Holedox*, deactivate its state to 0 when it was outside the *Holedox*, and set its state to 0 or 1 randomly when it was exactly on the *Holedox*.

On Sept. 11th of 3001, the attack began. However, after *t* movements of the *Holdedox*, the MSB intercepted its automated program. Now, based on the program and the *t*+1 previous sensor states, could you please write a program to help the MSB to calculate the minimum possible radius of the *Holedox*?

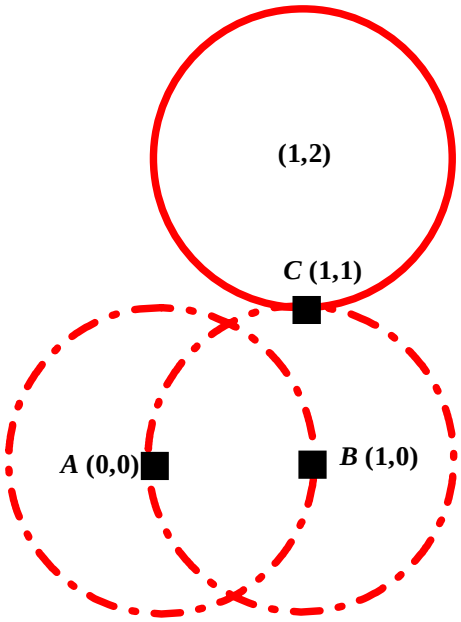


Fig 1. Sample Instance

To understand the problem clearly, let us consider the following instance described in the Sample Input. Suppose there were three sensors which are *A*(0,0), *B*(1,0) and *C*(1,1). As shown in Figure 1, accordingly to the program of "(1,0)(0,2)", a unit-radius disk moved from (0,0), to (1,0), and to (1,2) during the first three still periods. Therefore, the sensor states of *A*, *B* and *C* might become (1,1,0), (1,1,1) and (0,0,1)

respectively, consonant with the observations given in the input. Besides that, it is easy to verify that the unit size is the minimum possible radius for the sample input.

Input (from holedox.in)

The input file consists of several test cases. Each case begins with an integer m to indicate the number of sensors, whose coordinates, (x_i,y_i), are pairs of real numbers, given in the next m lines, where $1 \le m \le 50$ and $1 \le i \le$m. Then, an integer t, where $0 \le t \le 500$, is given to indicate the number of previous movements, followed by a line of m binaries representing the sensor states when the *Holedox* started to move. Among the next $2t$ lines, the $(2j\text{-}1)^{th}$ line gives (dx_j, dy_j) for the j^{th} movement in the intercepted program, and the $(2j)^{th}$ line gives m binaries which is the sensor states after the j^{th} movement of the *Holedox*, where $1 \le j \le t$. Finally, each case is terminated by an empty line, while the input file is terminated by $m=0$.

Output

For each test case, output a line of a real number, rounded to the third place after the decimal point, to indicate the minimum possible radius of the *Holedox*. If no possible radius exits for a certain case, please output -1.000 instead.

Sample Input

```
3
0 0
1 0
1 1
2
1 1 0
1 0
1 1 1
0 2
0 0 1

0
```

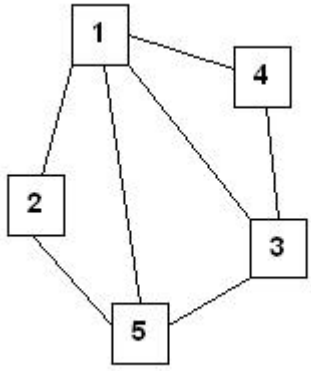
Sample Output

```
1.000
```

Problem F: Extended Graph Coloring

Input file name: egc.in

The Graph Coloring problem is a well-known NP hard problem in theoretical computer science society. Given a graph $G=(V,E)$ with $|V|=n$. C is a coloring mapping i.e. C: V-> 1, 2, …, c satisfying $C(i)\neq C(j)$ for all (i,j) in E. The graph coloring problem is to find an optimal coloring solution to minimize the size of coloring mapping i.e. minimize c. For example, in the graph in the following figure, there are 5 vertices and 7 edges. One possible optimal coloring for the graph is $C(1)=1$, $C(2)=C(3)=2$, $C(4)=C(5)=3$. That means we can use just 3 kinds of colors to construct a coloring mapping for that graph so that two vertices in one edge are colored by different colors.



Now, we extend the above graph coloring problem into another version that for a given graph $G=(V,E)$, and a fixed number of colors – c, find an optimal coloring mapping using these c colors to minimize the following objective function:

$$R(G) = \sum_{(i,j)\in\bar{E},C(i)=C(j)} 1$$

where C is a coloring mapping i.e. C: V -> 1,2, …, c satisfying $C(i)\neq C(j)$ for all (i,j) in E. $\bar{E}$ represents the set of missing edges in original graph. For example, if the coloring mapping of $c=3$ is $C(1)=1$, $C(2)=C(3)=2$, $C(4)=C(5)=3$ for the graph in above figure, we have $R(G)=2$.

You task is that calculate the minimal value of the objective function for a given graph configuration.

Input

The input file consists of several graph configurations. For each configuration, in the first line, there have two integers, v and c representing the number of vertices and colors for mapping ($0\leq v\leq 200$, $c\leq v$). Moreover, we describe the detail configuration in the following v-1 lines, where the j-th integer of the i-th line is set to 1 if (i,j) in E ($i<j$), otherwise 0.
When v equals 0, it means the end of the input file.

Assume: There exists at least one feasible coloring mapping using *c* kinds of colors for the graph given by each input configuration.

Output

For each graph configuration, just output a single line including the minimal value of the objective function for the extended coloring problem.

Sample Input

```
5 3
0 1 1 1 1
0 0 0 0 1
0 0 0 1 1
0 0 0 0 0
0 0
```

Sample Output

```
2
```

Problem G: CPCI/MCA

Input file: cpci.in

CPCI/MCA is another type of team contest, different from ACM/ICPC that one team can only consist of three team members at maximum. In CPCI/MCA, there is no restriction for the number of team members. However, the more team members, the more effort on their communication in the teamwork.

Now, you are asked to help the contest director to arrange the contest rooms. The host prepared several standard rooms on fixed capacity sized by number of people in. You should give a schedule to arrange all team members into contest rooms with the constraints that:

(1) All members from the same team should be arranged into the same room
(2) The number of peoples arranged into the same room should not exceed the fixed capacity of the standard room.

To save the budget, we must find the optimal solution with the minimum number of rooms to arrange all teams.

For instance, suppose the capacity of the standard room is 10. Moreover, we have 5 teams totally. The numbers of team members of each team are 1, 4, 10, 5 and 2. So, one of the optimal solution is using 3 standard rooms that room 1 for team 3, room 2 for team 1 and team 2, and room 3 for team 4 and 5.

Input (from cpci.in)

The format of input file is as follows:
The file line of the input file contains the number of test cases. For each test case, there have two parts:

(1) one line with two items: room capacity C and the number of teams T;
(2) T lines for the number of members of each team M_i ($1 \le i \le T$), one team per line.

Note: $1 \le C \le 1000$, $1 \le T \le 250$, $1 \le M_i \le 100$.

Output

For each test case, just output the result of the optimal solution, e.g. the minimum number of standard rooms to be used for that case. We give one line for the output of one case.

Sample Input

```
1
10 5
1
4
10
5
2
```

Sample Output

```
3
```

Problem H: Powerful Calculator

Input file name: cal.in

Today, facing the rapid development of business, ACM (Association of Calculator Management) recognizes that more powerful calculator should be studied, developed and appeared in future market shortly. ACM now invites you attending such amazing research and development work.

In most business application, the top three useful calculation operators are Addition (+), Subtraction (-) and Multiplication (*) between two given integers. Normally, you may think it is just a piece of cake. However, since some integers for calculation in business application may be very big, such as the GDP of the whole world, the calculator becomes harder to develop.

For example, if we have two integers 20000000000000000 and 4000000000000000, the exact results of the addition, subtraction and multiplication are:
20000000000000000 + 4000000000000000 = 24000000000000000
20000000000000000 – 4000000000000000 = 16000000000000000
20000000000000000 * 4000000000000000 = 80000000000000000000000000000000

Note: ACM prefers the exact format of the results rather than the float format or scientific remark format. For instance, we need "24000000000000000" rather than $2.4*10^{16}$.

As a programmer in ACM, your current task is to develop a program to obtain the exact results of the addition (*a*+*b*), subtraction (*a*-*b*) and multiplication (*a***b*) between two given integers *a* and *b*.

Input (from cal.in)

The input file consists of several test cases. Each case has two separate lines where the first line gives the integer *a* and the second gives *b* ($|a| < 10^{200}$ and $|b| < 10^{200}$)
When both *a* and *b* are equal to zero, the input file ends.

Output

For each test case in the input file, output three separate lines showing the exact results of addition (*a*+*b*), subtraction (*a*-*b*) and multiplication (*a***b*) of that case, one result per line.
Leaving a blank line between two successive test cases in the output.

Sample Input

```
20000000000000000
4000000000000000
0
0
```

Sample Output

```
24000000000000000
16000000000000000
800000000000000000000000000000000
```