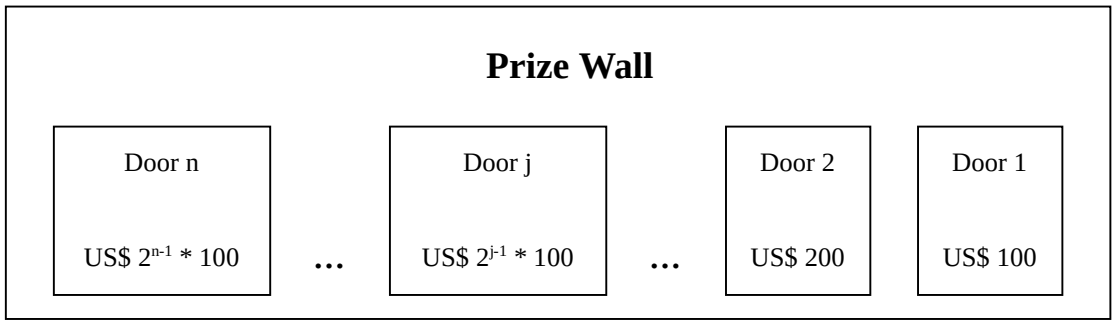
Problem A
Ranking Prizes
Input file: pa.dat

Consider the *n* lucky doors on the "prize wall" and the prize behind each door as shown in the following figure.



Suppose that all the doors are closed initially. You are allowed to open *s* doors ($s \le n$) to win all the prizes behind those *s* doors. Obviously, there are many different configurations in opening *s* doors and each configuration yields a total prize won (i.e., the summation of the prizes inside the *s* doors opened). We are interested in ranking each configuration based on the total prize won in opening *s* doors. The configurations are ranked in ascending order based on the total prize won. In other words, the configuration with the least total prize has rank 1. The configuration with the second lowest total prize has rank 2. The configuration with the grandest prize (i.e., the largest total prize) has the highest rank. For example, there are four lucky doors on the "prize wall" and you are allowed to open two doors. Obviously, there are six possible configurations:

C C O O C O C O C O O C O C C O O C O C O O C C,

where "C" and "O" mean the door is "Closed" or "Open". The total prizes associated with each of the above configurations are US$300, US$500, US$600, US$900, US$1000, US$1200, respectively. Therefore, the ranks of "CCOO", "COCO", "COOC", "OCCO", "OCOC", and "OOCC" are ranked 1, 2, 3, 4, 5, and 6, respectively. Given the number of lucky doors *n*, the number of doors that must be opened *s*, and a rank *k*, write a program to find the door-open configuration with rank *k* and the total prize won associated with this configuration. You may assume that *n* is less than or equal to 32.

Input

Each line (problem instance) has three numbers: *n s k*, separated by blanks.
A line of three zeros (0 0 0) indicates end of the input data.

Output

For each problem instance, you should print out two lines as follows:
Line 1: "door-open" configuration with rank *k*.
Line 2: the total prize associated with this configuration.
Print a blank line between each set of output.

Sample Input

```
4 2 4
5 2 6
7 3 10
16 8 10001
0 0 0
```

Sample Output

```
OCCO
US$ 900

COOCC
US$ 1200

CCOOOCC
US$ 2800

OOCCCCOCOOOCOCCO
US$ 4989700
```

Problem B
Formula Puzzle
Input file: pb.dat

Starship *Enterprise* is on a journey to explore other possible life forms in the universe. One day, the *Enterprise* was confronted by a novel alien starship. The creatures on that ship initiated a set of puzzle problems to challenge the wisdom of mankind. They demanded that the problems be solved within 30 seconds, or they would destroy the *Enterprise*. Fortunately, the problems were expressed in mathematical notations, which were understandable by both mankind and the outer-space creatures. Your mission is to save the *Enterprise* from possible destruction by solving the problems. The problems are expressed as follows:

Given a set *N* of positive integer numbers, a set *O* of binary arithmetic operators, and an integer value *f* of the final result, you are asked to determine how many possible formulas can be formed by using the numbers in set *N* and the operators in set *O* that would evaluate to the given value *f*. Note that each number in set *N* should only be used exactly once in each formula. For example, if you are given {1, 2, 3} as the number set *N*, {+, -} as the operator set *O*, and the result value *f* = 2, you can find four formulas that meet the requirements: 3-2+1, 3+1-2, 1-2+3, 1+3-2. Since each operator has been defined as a *binary* operator, a formula beginning with "+" or "–" is considered invalid. For example, -2+1+3 will be considered an invalid formula because it begins with "-". Therefore, the total number of valid formulas is 4.

Input

Each problem is represented with five lines of input data. The first line gives the total number of integers in the integer set *N*. The second line lists all the numbers in the integer set *N*. You may assume that there will be a maximum of 6 integers given for each problem, and all integers will be less than or equal to 1000. The third line represents the total number of operators in the operator set *O*, and the fourth line lists those operators. Operators are limited to + (addition), - (subtraction), * (multiplication), and / (division). The fifth line gives the value *f* of the desired result. A line with a single 0 indicates end of the input data. Engineering arithmetic precedence should be used when evaluating each formula (i.e., * and / should be performed before + and -).

Output

Each line of the output should print the total number of formulas which may be found for the respective problem.

Sample Input

```
3
1    2    3
2
+ -
2
3
1    12   3
1
+
4
2
23    43
1
+
66
0
```

Sample Output

```
4
0
2
```

Problem C
Shortest Nonzero Element
Input file: pc.dat

Let **Z** be the set of integers, and **v**=($v_1, v_2, \ldots, v_n$) be a vector in $\mathbf{Z}^n$. That is, all its components $v_1, v_2, \ldots, v_n$ are integers. The length of **v** is defined as

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{n} v_i^2}.$$

Define $\alpha\mathbf{v} = (\alpha v_1, \alpha v_2, \ldots, \alpha v_n)$, where α is an integer. Let **a** and **b** be two vectors in $\mathbf{Z}^n$. Consider the set of vectors **L** generated by the two given vectors **a** and **b**,

$$\mathbf{L}=\{\, \alpha\mathbf{a} + \beta\mathbf{b} \mid \alpha, \beta \in \mathbf{Z} \,\}.$$

A vector is *nonzero* if at least one of its components is not zero. Given **a** and **b**, write an *efficient* program to compute a shortest nonzero element in **L**. For example, when *n* = 1, both **a** and **b** are simply integers. In this case, the shortest nonzero element of **L** is the greatest common divisor of **a** and **b**.

Input

The input file contains a sequence of instances. Each instance begins with an integer *n*, and then followed by the components of **a** and **b**, each in one line. The last instance is followed by a 0 (zero), indicating end of the input file. Assume that $0 < n < 20$, and the length of each input vector is less than 2^{31}.

Output

For each instance, print **a** and **b**, and then followed by **c**, a shortest nonzero element of **L**, and the square of the length of **c**. Leave a blank line between two instances. The detailed format of the output is shown in the Sample Output section.

Sample Input

```
1
12
8
2
1  3
2  5
```

4
-21735 20335 -8995 6125
20493 -19173 8481 -5775
0

Sample Output

A = (12)
B = (8)
C = (4), |C|^2 = 16

A = (1 3)
B = (2 5)
C = (1 0), |C|^2 = 1

A = (-21735 20335 -8995 6125)
B = (20493 -19173 8481 -5775)
C = (-621 581 -257 175), |C|^2 = 819876

Problem D
Renting Conference Room
Input file: pd.dat

Your company owns a very precious conference room. For the benefit of the company, the conference room is open for rental at a particular day. Each customer can reserve the conference room but have to pay a price. Customers are allowed to bid for usage of the conference room; so the prices are proposed by the customers. The day before the conference room is actually used, all customers will state the particular time slots and the exact price that they are willing to pay. After all requests from customers are collected, you are to decide how the room shall be assigned to the customers to maximize the company's profit.

Note that the conference room can be rented to several customers as long as the reserving time slots do not *conflict* with each other. For example, a customer who needs the room from time unit 1 to time unit 100 does not conflict a customer who needs the room from time unit 200 to 400. On the other hand, a customer who needs the room from time unit 300 to 500 *does* conflict with a customer who needs the room from time unit 400 to 600; clearly we can not assign the room to both customers. To further clear up the boundary condition, we say that time unit 100 to 200 *does* conflict with time unit 200 to 300.

Input

Input file contains s*everal* sets of room rental requests. The inputs are all integers. Within each test set, the first integer (in a single line) represents the number of requests, n ($1 \leq n \leq 1{,}000$). The next n lines represent the n requests for that day. Each line contains three *positive integers*, t_1, t_2, and p, all separated by blanks, which means that the customer is willing to pay p dollars for using the room from time units t_1 to t_2. You may assume that $t_1 < t_2$. Note that the prices p's and the time units t_1's and t_2's are all arbitrary positive integers. Furthermore, $0 < t_1 < t_2 < 1{,}000{,}000{,}000$ and $0 < p < 10{,}000{,}000$.

These rental requests occur repetitively in the input as the pattern described above. A single 0 (zero) following any set of input signifies end of the input.

Output

For each set of rental requests appeared in the input, calculate the *maximum* profit that

the company can earn from these requests.

Sample Input

```
3
150    500    150
1    200    100
400    800      80
4
400    821    800
200    513    500
100    325    200
600    900    600
0
```

Sample Output

```
180
1100
```

Problem E
Broadcast Scheduling
Input file: pe.dat

The DJs of the ACM Radio Station (ARS) are facing a problem in determining their broadcast schedules. The programs are scheduled on a daily basis and, as a service to the audience, the songs or music selected for broadcasting are determined based on the requests sent by the audience on the previous day. Each day, anyone who wants to hear his/her favorite song or music on the radio can send a request to the ARS and ask for it to be broadcast on specified time of the next day. At the end of each day, the DJs collect all requests and determine the broadcast schedule for the next day accordingly. To make this service broadly available, each individual is allowed to request for one song or music per day. There is also a limit of 100 requests per day. In other words, the first 100 requests are collected and processed while all other requests are simply ignored. Anyone who couldn't make the first 100 requests today can try again tomorrow. The audience is assumed to be faithful in that each individual will start listening at the specified time once a request has been sent and will stay tuned until the specified song or music is actually broadcast. Based on the assumptions above, we can evaluate the quality of a schedule using the following criteria:

1. **Correctness** – At any given time, at most one song or music is selected and each song or music has enough time to be played to its completion (i.e. no overlapping is allowed and all songs/music must be completed before midnight).
2. **Completeness** – All requests are satisfied.
3. **Minimal Average Waiting Time** – To guarantee maximal audience satisfaction, we want to minimize the average waiting time for the requests. (The waiting time of a request is the time between the specified broadcast time and the actual broadcast time.)

Your job is to write a program to help the DJs in broadcast scheduling. The quality of your program will be judged according to the criteria described above. Note that a song or music may be broadcast more than once in a day to satisfy popular demand.

Input

The input to the program consists of a section of song/music description followed by multiple sections of requests using -1 as section separator and a 0 (zero) to indicate end of the input file. The song/music description section consists of lines of

song/music description, one entry per line. Each entry consists of a song/music ID and its length in minutes, separated by one blank. You may assume that both the IDs and the length are positive integers no more than 32000. Each request section contains all the requests collected for a particular day, one request per line. Each request is represented by a song/music ID and the desired broadcast time in military time format (i.e. 24-hour time format, such as 13:00 which stands for 1:00pm), separated by one blank.

Output

There should be an output section corresponds to each request section using one blank line as section separator. For each request section, if it is NOT possible to schedule all the requests, simply say so by displaying the message "no solution exists" without any further processing. If it is indeed possible to schedule all the requests, then the output should consist of three parts. The first part displays the number of songs scheduled for that day. The second part shows the average waiting time in minutes accurate to two digits to the right of the decimal point. The third part displays the optimal broadcast schedule in order of time. Each entry of the schedule consists of the selected time (in military time format as described above) and the ID of the chosen song/music.

Sample Input

```
1 5
2 16
3 3
4 8
5 20
6 4
7 6
8 13
-1
5 10:10
8 13:30
3 21:00
7 08:15
2 10:20
-1
```

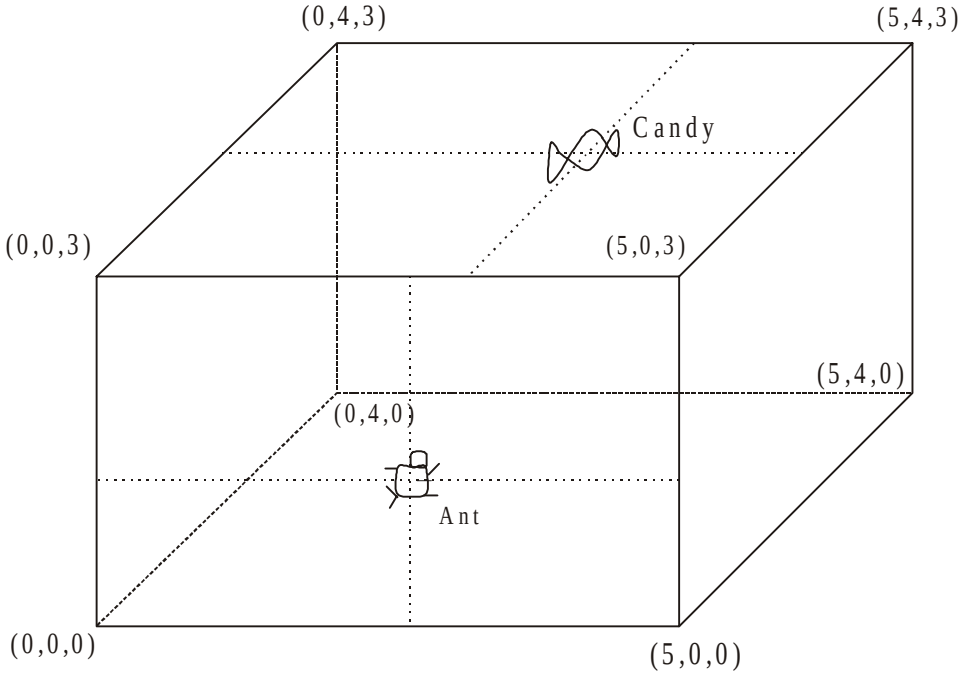
5 23:40
2 23:30
0

Sample Output

5 requests have been successfully scheduled.
The average waiting time is 2.00 minute(s).
08:15 7
10:10 5
10:30 2
13:30 8
21:00 3

no solution exists

Problem F
Ant and Candy
Input file: pf.dat



There is an ant and a candy on the surface of a rectangular box. The ant cannot fly. It can only walk on the surface of the box to the candy. Please write a program that calculates the distance of the shortest path from the ant to the candy. The coordinates of the eight corners of the rectangular box is shown in the above figure.

Input

The input consists of several pairs of the coordinates (one coordinate on each line) indicating the initial position of the ant and the candy. Each coordinate consists of three numbers and all separated by blanks. You may assume these coordinates are all on the surface of the rectangular box. A number "999" in the input line signifies the end of input.

Output

A real number represents the distance of the shortest path from the ant to the candy. Print the distance accurate to two digits to the right of the decimal point.

Sample Input

```
0  0  0
5  4  3
2.5  0  1.5
2.5  4  1.5
0  0  0.2
5  4  1.9
1.3  0  3
4.3  1.9  0
0.8  0  0.8
4.8  0  2.8
999
```

Sample Output

```
8.60
7.00
7.86
5.75
4.47
```

Problem G
Maximum Planar Subset
Input file: pg.dat

Given is a set C of n chords of a circle (see Figure (a)). We assume that no two chords of C share an endpoint. The endpoints of these chords are numbered from 0 to $2n$-1, clockwise around the circle (see Figure (b)). Let $M(i, j)$, $i \le j$, denote the number of chords in the maximum *planar subset* (i.e., no two chords overlap each other in the subset) in the region formed by the chord $\overline{ij}$ and the arc between the endpoints i and j (see Figure (d)). As the example shown in Figure (a), $M(2, 7) = 1$, $M(3, 3) = 0$, and $M(0, 11) = 3$. You are to write a program that computes the number of chords in the maximum planar subset in a circle of n chords, i.e., compute $M(0, 2n\text{-}1)$.

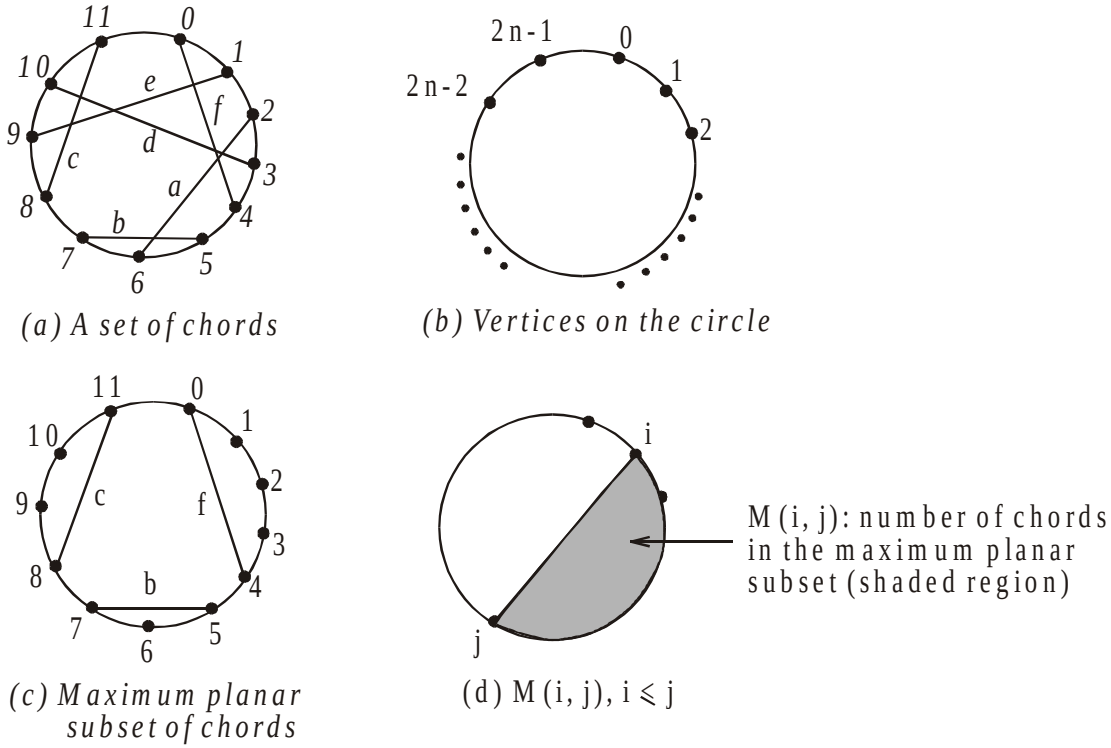


Figure: Maximum planar subset

Input

The input consists of several circles with different configurations. Configuration of each circle is specified by having an integer $2n$, $1 \le n \le 100$, on a single line, denoting

the number of vertices on a circle, followed by n lines, each containing two integers a and b ($0 \le a, b \le 2n-1$), denoting two endpoints of a chord. A single 0 (zero) in the input line at the beginning of a new configuration signifies end of the input file.

Output

For each circle configuration, print the number of chords in the maximum planar subset.

Sample Input

```
12
0  4
1  9
2  6
3  10
5  7
8  11
4
0  2
1  3
0
```

Sample Output

```
3
1
```

Problem H
Winning Poker Hands
Input file: ph.dat

A poker game has 52 cards of 4 suits: S(pade), H(eart), D(iamond) and C(lub). Each suit has 13 cards: A, K, Q, J, 10, 9, …, 3 and 2 where K, Q and J stand for 13, 12 and 11, respectively, and A can stand for either 14 or 1.
A poker hand consists of five cards. Poker hands are ranked from high to low based on their patterns (In the table below, 1 indicates the highest rank, 9 has the lowest rank.):

Rank	Poker Hands
1	Straight flush (five cards of the same suit in a sequence, e.g. SQ, SJ, S10, S9, S8)
2	Four of a kind (face values of the form (x,x,x,x,y), e.g. S5, H5, D5, C5, D7)
3	Full house (face values of the form (x,x,x,y,y), e.g. SJ, DJ, CJ, H4, D4)
4	Flush (five cards of the same suit, e.g. HK, H10, H9, H6, H3)
5	Straight (five cards in a sequence, regardless of suit, e.g. D5, H4, C3, H2, DA)
6	Three of a kind (face values of the form (x,x,x,y,z) where x, y and z are distinct, e.g. S8, H8, C8, HK, D3)
7	Two pairs (face values of the form (x,x,y,y,z)where x, y and z are distinct, e.g. S8, H8, CK, HK, D3)
8	One pair (face values of the form (w,w,x,y,z)where w, x, y and z are distinct, e.g. S8, H8, C9, HK, D3)
9	None of the above

Write a program that, given 13 cards in an input line, select and output a hand of 5 cards of the highest rank and then, from the remaining 8 cards, select and output another hand of 5 cards of the highest rank. The program processes all input lines iteratively. (If more than one hand is of the highest rank, any one can be selected.)

Input
The input file consists of an arbitrary number of input lines. Each line contains 13 cards, all separated by blanks. A line with a single 0 (zero) signifies end of the input file.

Output

For each input line of 13 cards, two output lines are produced, followed by a blank line. The first one contains a hand of 5 cards of the highest rank selected from the 13 cards. The second one contains a hand of 5 cards of the highest rank selected from the remaining 8 cards.

Sample Input

```
CJ C4 H7 S6 CK CA C9 D8 D6 HQ D3 H3 S5
DA S2 CQ H3 D3 C9 HQ D8 D5 SA C6 DQ C3
0
```

Sample Output

```
CA CK CJ C9 C4
D6 S6 H3 D3 HQ

HQ DQ CQ DA SA
C3 H3 D3 C9 D8
```