# 2012 ACM-ICPC Asia Hatyai Regional Programming Contest

## 16 November 2012

Hosted by Prince of Songkla University, Hatyai Campus
Thailand

- There are **10** problems (A-J) to solve within 5 hours.
- Solve as many problems as you can, in an order of your choice.
- Use C or C++ or Java to program at your convenience for any problems.
- **Input and output** of each program are **standard input** and **output**.

A

# Problem A
## Binary Matrix 2

You are given a matrix of size `r x c`. Each of the elements can be either 0 or 1. In each operation you can flip any element of this matrix, i.e. convert 0 to 1 or convert 1 to 0. Your goal is to convert the matrix such that -

1. Each of the rows will have the same number of 1s and
2. Each of the columns will have the same number of 1s.

What is the minimum number of operations required to achieve this?

**Input:**
Input starts with a positive integer `T` (~1000) which indicates the number of inputs.

Each case starts with two integers `m` and `n` (`1 <= r, c <= 40`), here `r` is the number of rows and `c` is the number of columns of the matrix. Each of the next m lines will have n integers each, either 0 or 1.

**Output:**
For each test case, output "`Case #: R`" in a single line, where # will be replaced by case number and `R` will be replaced by the minimum number of steps required to achieve the target matrix. Replace `R` by -1 if it is not possible to reach target matrix.

**Sample Input:**
```
3
2 3
111
111
3 3
011
011
011
2 3
001
000
```

**Sample Output:**
```
Case 1: 0
Case 2: 3
Case 3: 1
```

B

# Problem B
## Probability Through experiment

Mathematicians have often solved problems by just doing some simulation or experiments. For example, the value of pi (π) can be approximately determined by randomly plotting points in a square that inscribes a circle. Because if the square is a 250x250 square, then its area is 62500 and the area of the inscribed circle is pi*$125^2$=15625pi. As the points are plotted randomly, so it can be assumed that number of points inside the circle and total number points plotted in the square is proportional to their respective area. So, in this way, the value of pi can approximately be determined by counting how many points are inside the circle (Figure 1). The value of pi can even be determined using a more sophisticated experiment like the Buffon's needle experiment (Figure 2).

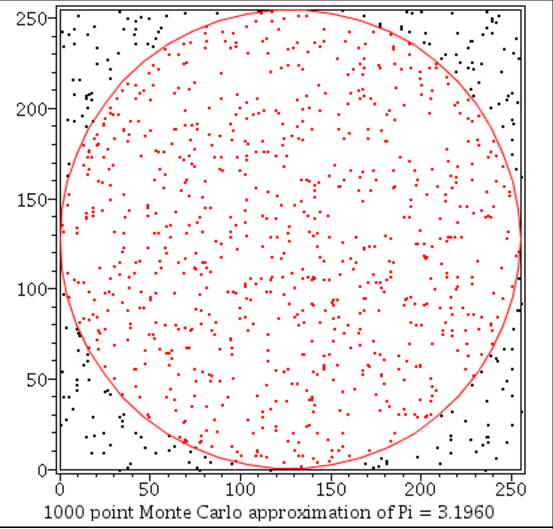

Figure 1: Pi approximation by counting the number of points inside the circle

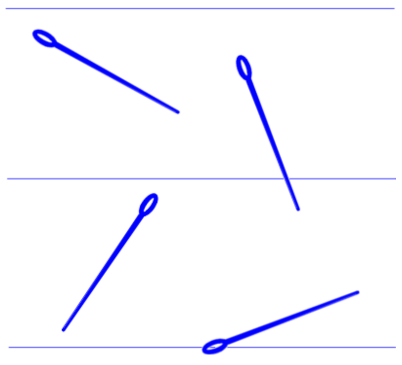

Figure 2: Buffon's needle experiment

The two experiments mentioned above to approximately determine the value of pi could be simulated by writing a computer program very easily. It would have been nice to do this sort of experiment a lot of time (Say 1 billion billion) and get an almost perfect result but for lack of time we cannot do that in real life. In this problem, you will have to write a program that will help Professor Wu to perform a similar sort of experiment but this program may not be that straightforward.

Professor Neal Wu is trying to solve a classic problem using simulation: If three points are randomly plotted on the boundary of a circle, then what is the probability that they will be the three vertex of an acute triangle? Of course, this problem can be solved analytically and the result he gets is 0.25. Now, he wants to verify this result through an experiment. The result can be found approximately by plotting three random points on a circle billions of times and counting how many times these three points form an acute triangle. The beauty of such an experiment (as mentioned above) is that if we increase the number of trials, the result will become even more accurate. But if Dr. Wu wants to repeat this process 1000 billion times, it will take 2 hours of time and if he wants to repeat it a billion billion times, it may take more than 200 years. Dr. Wu has discovered that this process can be sped up by using a different approach – generate n random points on the boundary of a circle and they form $\frac{n(n-1)(n-2)}{6}$ triangles as vertices. How many of these triangles are acute triangles? If the number of acute triangle is M and let N = $\frac{n(n-1)(n-2)}{6}$, then the desired probability is $\frac{M}{N}$. So, given the n points on the boundary, you have to assist Dr. Wu by writing a very efficient program to find the number of acute triangles.

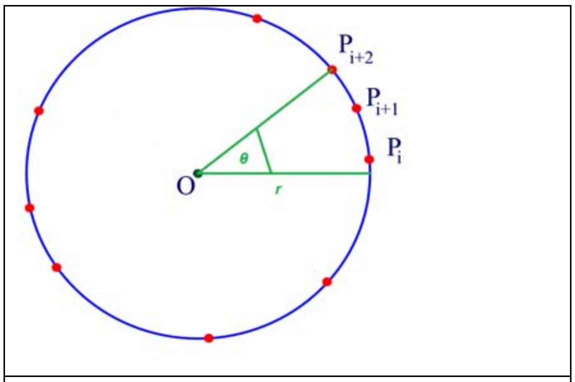

Figure 3: Example of a circle with given points on the boundary

## Input

The input file contains around **40** test cases. But most of the cases are not extreme, so the size of the input file is around 3 MB. The description of each test case is given below:

Each case starts with two positive integers n ($0 < n \leq 20000$) and r ($0 < r \leq 500$). Here, n is the total of points on the circle boundary and r is the radius of the circle. The center of the circle is always at the origin (0,0). Each of the next N lines denotes the location of one point on the boundary of the circle. Each point is P, denoted by a floating-point number θ ($0.000 \leq \theta < 360.000$). This θ is actually the angle (expressed in degree) the point P creates at the center of the circle with the positive direction of x-axis. So the Cartesian coordinate of P is ( r*cos(θ), r*sin(θ)). Value of θ will always have exactly three digits after the decimal point. No two points will be at the same location.

A line containing two zeroes terminates the input. This line should not be processed.

## Output

Each test case produces one line of output. This line contains the serial of output followed by an integer. This integer denotes how many of the $\frac{n(n-1)(n-2)}{6}$ triangles formed by these n points are actually acute triangles.

/* 20000 points generated on the boundary of a circle can actually create 20000*19999*19998/6 ~1333 billion triangles. So, of experiment for 1333 billions can be done in, say, 0.5 second. Then experiments with 1 billion billion triangles can be done in around 100 hours only (In contrast to 200 years mentioned earlier) just by repeating this experiment. Also, if we put 1817120 points on the boundary of the circle, around 1 billion billion triangles are created, and the number of acute triangles within this large number of triangles can be computed within 5 minutes. */

| **Sample Input** | **Sample Output** |
|---|---|
| <pre>4 71<br>234.600<br>33.576<br>20.375<br>84.908<br>7 7<br>11.586<br>114.435<br>248.411<br>108.640<br>287.629<br>150.224<br>340.481<br>0 0</pre> | <pre>Case 1: 2<br>Case 2: 12</pre> |

C

# Problem C
# Optimal Space Way

It is 2180 AD, the human race has just started to abandon the earth and spread the civilization in space. ACM (Association for Cosmic Machinism) is in charge of this whole relocation process and this process is funded mostly by IBM (Inter-planet Ballistic Machines). Thousands of space cities are built in the outer space and to maintain universal business hour (Sunrises and sets at the same time in all space cities) they are built on the same imaginary plane. So the location of the cities can be expressed as two-dimensional Cartesian coordinates.

The cities are located far away from one another and the only way of going from one city to another is by using shuttle rockets. But from previous experience ACM knows that traveling by roads is a lot cheaper than traveling by rockets. So they have decided to build some roads. But building roads between every pair of cities is very expensive. So they are planning to build a super space-way (SSW) that will be a straight road. When someone needs to travel from a city c1 to city c2, he first travels from c1 to the nearest point of SSW with a rocket, and moves along the SSW to the point, which is nearest to c2. Then from there he goes to c2 using another rocket. The cost of moving along the road is proportional to the distance but cost of flying on rockets is proportional to the square of the distance. So you have to build the SSW in such a position so that total cost of traveling in a calendar year by rockets is minimized. You can assume that:

1. The cities are so small compared to the distance between them that they can be considered as points on a two dimensional Cartesian coordinate system.
2. The SSW is so narrow compared to the distance between the cities that it can be considered as a straight line. The length of the SSW can be increased indefinitely in both directions if total cost of traveling by rocket decreases by doing so.
3. For simplicity you can assume that total number of incoming and outgoing rockets from each city is the same.
4. Sometimes exactly one city can be marked as the super city, which is the center of all activities. In that case the total of incoming and outgoing flights of that city is M times higher than an ordinary city.
5. Assume that all rockets travel in straight line.

Given the location of all the cities, you will have to find a location for the SSW for which the total cost of traveling by rockets is minimum. And you have to produce the minimum average cost per rocket flight assuming that cost of flying v unit distance is $v^2$.

## Input

The input file contains less than 50 test cases. The description of each test case is given below:

Each test case starts with two integers N ($0 < N \leq 10000$) and Q ($0 < Q \leq 100$). Here N is the total number cities built in outer space and Q is the total number of query. Each of the next N lines contains two floating-point numbers, $x_i, y_i$ ($0.0 \leq x_i, y_i \leq 1000.0$) which denotes the coordinate of the i-th city. Cities are numbered from 0 to N-1 in the order they appear in the input file. Each of the next Q lines contains two integers S ($0 \leq S \leq N-1$) and M ($1 < M \leq 10000$).

Here S denotes the id of the super city and M denotes that its total number of incoming and outgoing flights in the super city is M times higher than an ordinary city.

A line containing two zeroes terminates the input. This line should not be processed.

## Output

For each test case produce Q+2 lines of output. The first line contains the case number followed by a colon. The second line contains a floating-point number which denotes the minimum average cost per rocket flight for the optimal super space way (Assuming all cities are ordinary). This line is followed by Q lines, one line for each query. This line contains the serial of query followed by a floating-point number. This number denotes the minimum average cost for the optimal super space way assuming that S th city is a super city and all other cities are ordinary. All floating-point numbers in the output should have five digits after the decimal point. Look at the output for sample input for details. You can assume that for all floating-point outputs an absolute error less than $10^{-5}$ will be ignored.

## Sample Input

```
5 2
464.9900 243.2652
463.9409 772.4632
201.9822 561.6255
695.8948 933.4567
226.0628 93.1435
3 2
4 3
4 2
27.1679 304.2512
27.7639 16.2479
921.9150 863.0064
167.6203 929.5471
2 2
2 3
0 0
```

## Output for Sample Input

```
Case 1:
16172.49971
1: 14289.23473
2: 11558.37654
Case 2:
53198.72595
1: 47995.33546
2: 41543.27604
```

# Problem D
# Radiation

Nuclear power plants (NPP) are a blessing and curse of modern civilization. NPPs have some risks but still it is one of the cheapest ways to produce electricity in the developed world. In this problem we will discuss a situation related to two nuclear plants, which are not far away from each other.

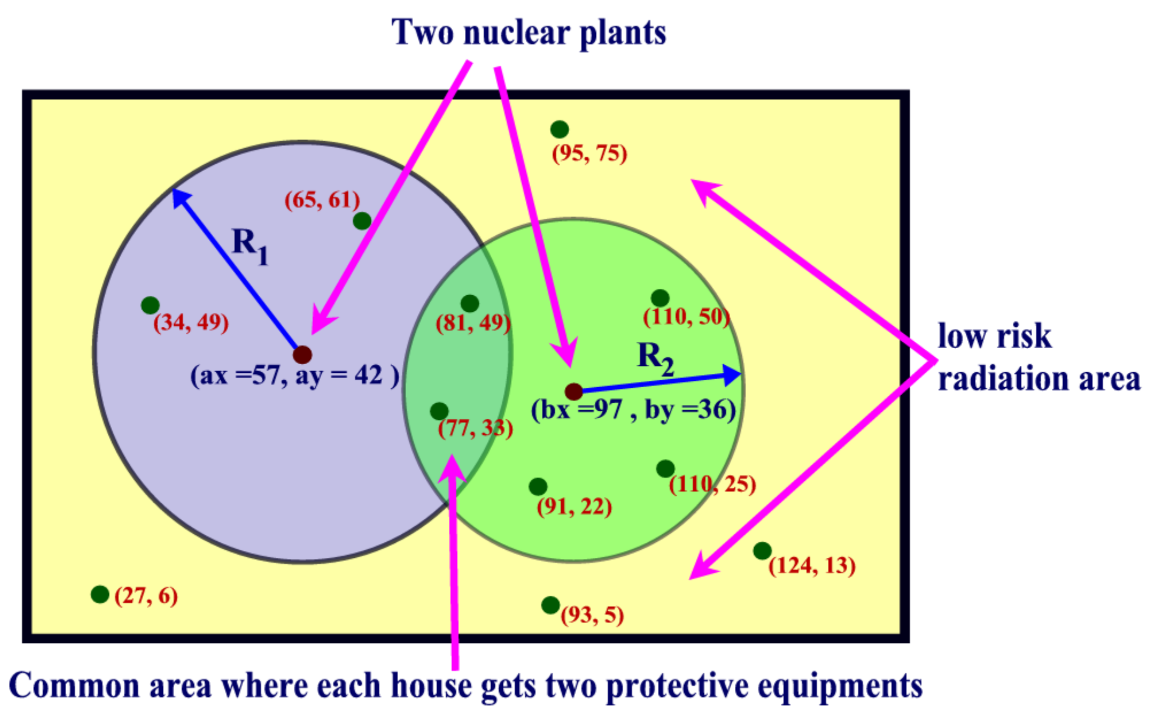

Figure 1: Two Nuclear Power Plants. Houses at (81, 49) and (77,33) are at high risk from both the plants.

We will describe the entire scenario in a flat land, so two-dimensional Cartesian coordinate system is used to denote each location. Lets assume that the coordinate of the two nuclear power plants are (ax, ay) and (bx, by). Houses that are located within distance $R_1$ (inclusive) of the power plant at (ax, ay) are under high risk of radiation. Similarly, houses that are located within distance $R_2$ (inclusive) of the power plant at (bx, by) are under high risk of radiation. So the authorities of power plant 1 and power plant 2 distribute special protective equipments to the houses that are within radius (inclusive) R1 and R2 of the respective power plants. As a result each of the houses that are endangered by both the plants actually receive two sets of equipments to protect their house, however only one set is enough for full protection. Houses that are outside the high-risk area are under low risk of radiation but they do not receive any protective equipment due to budget constraints. However, each owner of the houses that have two sets of protective equipments gives away one set of equipment to the owner of a house that has none. Still, some houses in the low-risk area remain un-protected. Given the location of the houses and the values of ax, ay, bx, by and possible values of $R_1$ and $R_2$ your job is to find out the number of houses that are without protective equipments for each pair of values of $R_1$ and $R_2$.

## Input

The input file contains at most **3** test cases. The description of each test case is given below:

A test case starts with a line containing a positive integer N ($0 < N \le 200000$) that denotes the number of houses that are under either low risk or high risk of radiation. Each of the next N lines contains two integers $x_i$, $y_i$ ($0 \le x_i, y_i \le 20000$) that denotes the coordinate of the i-th house. No two houses are at the same location. The next line contains five integers ax, ay, bx, by and q ($0 \le$ ax, ay, bx, by $\le 20000$, $0 < q \le 20000$). The meaning of ax, ay, bx and by are given in the problem statement. Here q denotes the total number of query. Each of the next q lines contains two integers, which denote the values of $R_1$ and $R_2$ ($0 < R_1, R_2 \le 13000$) respectively.

A line containing a single zero terminates input. This line should not be processed.

## Output

For each test case produce q+1 lines of output. The first line is the serial of output. For each query (given value of $R_1$ and $R_2$) determine how many houses in the low risk region remains without protective equipment. You may consider using faster IO as judge input file is large.

| Sample Input | Output for Sample Input |
|---|---|
| <pre>11<br>95 75<br>27 6<br>93 5<br>124 13<br>34 49<br>65 61<br>81 49<br>77 33<br>110 50<br>91 22<br>110 25<br>57 42 97 36 2<br>31 25<br>25 25<br>0</pre> | <pre>Case 1:<br>2<br>2</pre> |

Note: First query in the sample input corresponds to Figure 1.

# Problem E
# Version Controlled IDE

Programmers use version control systems to manage files in their projects, but in these systems, versions are saved only when you manually submit.
Can you implement an IDE that automatically saves a new version whenever you insert or delete a string?

Positions in the buffer are numbered from 1 from left to right. Initially, the buffer is empty and in version 0. Then you can execute 3 commands (vnow means the version before executing the command, and L[v] means the length of buffer at version v):

1 p s: insert string s after position p(0<=p<=L[vnow], p=0 means insert before the start of the buffer). s contains at most 1 and at most 100 letters.
2 p c: remove c characters starting at position p(p>=1, p+c<=L[vnow]+1). The remaining charactesr (if any) will be shifted left, filling the blank
3 v p c: print c characters starting at position p(p>=1, p+c<=L[v]+1), in version v(1<=v<=vnow).

The first command is guaranteed to be command 1(insert). After executing each command 1 or 2, version is incremented by 1.

## Input

There is only one test case. It begins with a single integer n (1<=n<=50,000), the number of commands.
Each of the following n lines contains a command. The total length of all inserted string will not exceed 1,000,000.

## Output

Print the results of command 3, in order. The total length of all printed strings will not exceed 200,000.

## Sample Input (no obfuscation)

```
6
1 0 abcdefgh
2 4 3
3 1 2 5
3 2 2 3
1 2 xy
3 3 2 4
```

## Sample Output

```
bcdef
bcg
bxyc
```

***Obfuscation***

In order to prevent you from preprocessing the command, we adopt the following obfuscation scheme:
Each type-1 command becomes `1 p+d s`
Each type-2 command becomes `2 p+d c+d`
Each type-3 command becomes `3 v+d p+d c+d`
Where d is the number of lowercase letter 'c' you printed, before processing this command.
After the obfuscation, the sample input would be:

```
6
1 0 abcdefgh
2 4 3
3 1 2 5
3 3 3 4
1 4 xy
3 5 4 6
```

This is the real input that your program will read.

International Collegiate Programming Contest

event sponsor

**F**

# Problem F
# Ultimate Device

[illegible]

Mr Tomisu Ghost is planning to build the ultimate device. He shared the idea with me but asked me to keep it secret. So, I am not going through the functionalities of this great device and I also do not want to describe the mechanisms to build it.

However, Mr Tomisu has planned to buy some circuits for this device from a local store and for that he went to the store. He found that there were **n** types of circuits and each of the **i**$^{\text{th}}$ circuits has the burning cycle of $\mathbf{t_i}$ seconds. burning cycle of a circuit means that if it's used in the device, it will enter to its burning state at every $\mathbf{t_i}$ seconds. If there is any other circuit in the device that is not in its burning state, then every circuit will survive and no damage will happen. But if every circuit is at its burning state at that time, all will be burned out together and the device will be malfunctioned.

For example, consider two circuits have the burning cycle of 3 and 5 seconds respectively and let both of them are used in the device together. At 3rd second, circuit 1 will be in its burning state, but since the other one is not in its burning state, it will survive. At 5th second, circuit 2 will be in burning state while circuit 1 will not be in its burning state, thus circuit 2 will also survive. At 6th second circuit 1 will be in its burning state again, but survive for the same reason. Thus at 15th second both circuits will be in their burning state and burn out. If there are three circuits with the burning cycle of 3, 4 and 5 seconds respectively and all of them are used together, they will burn out at 60th second. But if the first two circuits are used only, then they will burn out at 12th second.

Now, Mr. Tomisu wants to go through all the circuits one by one. In front of every circuit, he will flip a coin (assume that it's a fair coin). If it's a head he will select the circuit, otherwise he will reject it. After visiting the **n**$^{\text{th}}$ circuit he will have some selected circuits for his device. You have to help him by calculating the expected lifetime of his device. If no circuit is selected, then the lifetime of the machine is 0.

## Input

Input starts with an integer **T (≤ 100)**, denoting the number of test cases.

Each case starts with an integer **n (1 ≤ n ≤ 100)**, where **n** denotes the number of circuits. The next line contains **n** space separated integers, where the **i**$^{\text{th}}$ integer denotes the burning cycle of the **i**$^{\text{th}}$ circuit, $\mathbf{t_i}$ $\mathbf{(1 \le t_i \le 500)}$. You may assume that all the burning cycles for a test case will be distinct.

## Output

For each case, print the case number first. Then print $\mathbf{(r * 2^n)}$ modulo **10007**, where **r** is the expected lifetime of the device. If $\mathbf{(r * 2^n)}$ is not an integer print **"not integer"** without the quotes.

| Sample Input | Sample Output |
|---|---|
| `2`<br>`3`<br>`3 4 5`<br>`2`<br>`2 7` | `Case 1: 119`<br>`Case 2: 23` |

# Problem G
# Warp Speed II

In the not-so-distance future, space engineers can invent a new technology for traveling through space and name it as “warp-drive”. This warp-drive can make a spaceship travel faster than light speed. It works by bending an amount of distance in space and make a ship travel through that bended space in a single “hop”. To travel through space, a spaceship (that is equipped with this warp-drive) may have to perform more than one hop between the beginning and the ending points. The amount of energy/power to hop depends on the current state/configuration of the warp-drive. And some amount of energy is also consumed in order to prepare or switch to any warp-drive state.

**Goal**
Suppose that you are an engineer on a battle spaceship. Your duty is to control/configure the warp-drive so that it will consume as less energy as possible for any traveling. For each traveling, you will be provided with a sequence of hops, which you have to find a corresponding sequence of warp-drive configurations that use the lowest energy.

In order to accomplish your duty, you have to build a computer program to find the lowest energy of warp-drive configuration sequence for any given hop sequence based on two tables of warp-driver energy consumptions. The first table shows the energy for switching between any two states. The second table shows the energy consumption related to warp-drive states and hops.

**Input**
Input is a standard input which consists of 4 parts, separated by a blank line.

The first part defines the sizes of warp-drive states and hop types.

- It contains of only one line. This line contains 2 numbers separated by a space. These 2 number are
  - Size of warp-drive states(*N*), which is between 1 and 100.
    - The state id is starting from *0* to *N-1*.
    - The first state (id #0) is the idle state. At this and only this state, the warp-drive can't perform any hop. (It is the default starting and ending state for any output state sequence.)
  - Size of different types of hops(*H*), which is between 1 and 1,000.
    - The hop id from *0* to *H-1*.

The second part is a table showing the energy for switching warp-drive states. The table size is *N* rows and *N* columns, which contains $N^2$ energy values.

- There are *N* lines. Each line contains *N* energy values. These values are between 1 and 100.
- Each energy value can be indexed by its row and column. The row and column indexes are the state ids of the current and next warp-drive states respectively. Please be notified that these two indexes are starting from 0.
  - For example, the value at *(row index, column index) = (5,0)* is the energy for switching warp-drive state from number 5 to 0. (This value is the first number in the sixth line.)

The third part is a table showing the energy consumption in performing hops for each state drive.

The table size is *N* rows and *H* columns, which contains *N* by *H* values.

- There are *N* lines. Each line contains *H* energy values. These values are between 1 and 100.
- Each energy value can be indexed by its row and its column. The row index is the id of the current warp-drive state. The column index is the id of the hop. Please be notified that these two indexes are starting from 0.
    - For example, the value at *(row index, column index) = (1,9)* is the energy for warp-drive at state #1 to perform hop #9 . (This value is the tenth number in the second line.)
    - Please be notified that in the very first line of this part, which corresponds to the state #0 or the idle state, all numbers are zeros. This rather means that it can't perform any hop in this state.

The fourth part is a set of hop sequences. The number of sequences in this set is between 1 and 1,000.

- The number of lines is 1 up to 1,000.
- Each line in this part contains one hop sequences.
- The number of hops in a sequence is between 1 and 1,000.
- Each hop is a number (starting from *0* to *H-1*) and is separated with space.

The blank line after the fourth part is the termination of the input.

**Output**

For each hop sequence in the fourth part of the input, write 2 parts of output as follows

- In the first line, write the total amount of energy which must be minimum.
- In the following line, write the solution. The solution contains a sequence of warp-drive states corresponds to the given hop sequence. (The size of input and output sequence must be equal.) If there are two or more possible solutions which consumes the same amount of energy, write only the sequence which contains the lowest states comparing from left to right.

| Line no. | **Sample Input** | **Sample Output** |
|---|---|---|
| 1 | 4 5 | 9 |
| 2 | | 3 2 |
| 3 | 1 2 6 1 | 23 |
| 4 | 3 4 3 17 | 1 1 2 3 |
| 5 | 2 3 9 3 | |
| 6 | 1 21 1 8 | |
| 7 | | |
| 8 | 0 0 0 0 0 | |
| 9 | 3 3 2 4 3 | |
| 10 | 2 2 4 3 1 | |
| 11 | 4 2 2 7 7 | |
| 12 | | |
| 13 | 0 4 | |
| 14 | 1 2 3 2 | |
| 15 | | |

**More Explanations**

There are 15 lines in the sample input and 4 lines in the sample output.

In the first sample output (solution for input line #13), the minimum total energy is 9. The warp-drive state sequence is [3 2] or [0- 3 2 -0]. There are 3 state switchings from 0 → 3, 3 → 2, and 2 → 0, which consumes 1 + 1 + 2 = 4. And two hops (0 and 4) at state 3 and 2 respectively, which consumes 4 +1 = 5. So the total energy is 4 + 5 = 9.

In the last sample output, the minimum energy is 23. The solution is [0- 1 1 2 3 -0] which draws 23 energy in total.

# Problem H
# Birthdates

Write a program to identify the youngest person and the oldest person in a class.

**Input**

The number $n$ $(1 \leq n \leq 100)$ in the first line determines the number of people in a class. The following $n$ lines contain person's name and his/her birthdate.

The information in each line is of this format:

*personName dd mm yyyy*

where *personName* is a single word less than 15 letters, *dd mm yyyy* are date, month and year of the birthdate.

Suppose that no one has the same name or the same birthdate.

**Output**

Print out 2 lines containing the name of youngest person and oldest person, respectively.

**Sample Input**

```
5
Mickey 1 10 1991
Alice 30 12 1990
Tom 15 8 1993
Jerry 18 9 1990
Garfield 20 9 1990
```

**Sample Output**

```
Tom
Jerry
```

# Problem I
# Prime Substring

Given a string of digits, your task is to find the largest prime number which presents in that string. Our prime numbers are values between 2 to 100,000 only.

**Input**

Each line contains a string of digits (255 digits at most). The line contains only 0 indicates the end which will not be processed. The input does not exceed 1,000 lines.

**Output**

Print out in each line the largest prime number found in each input string.

| Sample Input | Sample Output |
|---|---|
| 11245<br>91321150448<br>1226406<br>0 | 11<br>1321<br>2 |

J

## Problem J
## Longest Word

A word is composed of only letters of the alphabet (a-z, A-Z) and may contain one hyphen (-) or more. Given a text containing words, and other characters (punctuations, numbers, symbols, etc), you are to write a program to find the longest word.

Each letter or a hyphen in a word is counted as 1. For example,

| | | |
|---|---|---|
| The length of `Apple` | is | 5 |
| The length of `son-in-law` | is | 10 |
| The length of `ACM-ICPC` | is | 8 |

**Input**

A text may contain several lines and paragraphs but the text does not exceed 10,000 characters. No word can exceed 100 characters. The word `E-N-D` indicates the end of input.

**Output**

Print out the longest word in small letters. If there exist several longest words, print only the first one found in the text.

**Sample Input**

```
  ACM International Collegiate Programming Contest (abbreviated as
ACM-ICPC or just ICPC) is an annual multi-tiered computer programming
competition among the universities of the world. The contest is
sponsored by IBM. Headquartered at Baylor University, with autonomous
regions on six continents, the ICPC is directed by Baylor Professor
William B. Poucher, Executive Director, and operates under the
auspices of the Association for Computing Machinery (ACM).

    The 2012 ACM-ICPC Asia Hatyai Regional Programming Contest is
held during 15-16 November 2012. It is hosted by Prince of Songkla
University, Hatyai campus.  E-N-D
```

**Sample Output**

```
international
```