



acm International Collegiate
Programming Contest

2006



event
sponsor

ACM International Collegiate Programming Contest 2006

South American Regional Contests

November 10-11, 2006

Contest Session

This problem set contains 9 problems; pages are numbered from 1 to 19.

This problem set is used in simultaneous contests hosted in the following countries:

- Argentina
- Bolivia
- Brazil
- Chile
- Colombia
- Peru
- Venezuela

Problem A

Weekend Lottery

Source file name: `lottery.c`, `lottery.cpp`, `lottery.java` or `lottery.pas`

Some people are against lotteries on moral grounds, some governments forbid lotteries, but with the advent of Internet this popular form of gambling, which started in China and helped finance the Great Wall, is thriving.

But the odds of winning a national lottery are tiny, and therefore your college classmates decided to organize a private lottery, with draws every Friday. The lottery is based on a popular style: a student who wants to bet chooses C distinct numbers from 1 to K and pays US\$ 1.00 (notice that traditional lotteries such as US National Lotto use $C = 6$ and $K = 49$). On Friday during lunch C numbers (also from 1 to K) are drawn. The student whose bet has the largest number of correct guesses receives the amount collected in the bets. This amount is shared in case of ties and accumulates to next week if no one guessed any of the numbers drawn.

Some of your colleagues do not believe in the laws of probability and asked you to write a program that determines the numbers that have been drawn the fewest times considering all previous draws, so that they can bet on those numbers.

Input

The input contains several test cases. The first line of a test case contains three integers N , C and K which indicate respectively the number of draws that have already happened ($1 \leq N \leq 10000$), how many numbers comprises a bet ($1 \leq C \leq 10$) and the maximum value of the numbers to be chosen in a bet ($C < K \leq 100$). Each of the next N lines contains C distinct integers X_i indicating the numbers drawn in each previous contest ($1 \leq X_i \leq K$, for $1 \leq i \leq C$). The end of input is indicated by $N = C = K = 0$.

The input must be read from file `lottery.in`.

Output

For each test case in the input your program must write one line of output, containing the set of numbers that have been drawn the fewest times. This set must be printed as a list, in increasing order of numbers. Leave one blank space between two consecutive numbers in the list.

The output must be written to standard output.

Sample input	Output for the sample input
5 4 6 6 2 3 4 3 4 6 5 2 3 6 5 4 5 2 6 2 3 6 4 4 3 4 3 2 1 2 1 4 4 3 2 1 4 3 0 0 0	1 1 2 3 4

Problem B

Lazy Jumping Frog

Source file name: frog.c, frog.cpp, frog.java or frog.pas

Mr. Frog lives in a grid-like marsh of rectangular shape, composed of equally-sized cells, some of which are dry, some of which are only watery places. Mr. Frog lives in a dry cell and can jump only from a dry cell to another dry cell on his wanderings around the marsh.

Mr. Frog wants to visit his girlfriend, Ms. Toad, who also lives in a dry cell in the same marsh. But Mr. Frog is lazy, and wants to spend the minimum amount of energy in his jumping way to Ms. Toad's home. Mr. Frog knows how much energy he spends in any of his jumps. For any single jump, Mr. Frog always uses the following figure to determine which are the possible target cells from his current position (the cell marked **F**), and the corresponding energy spent in the jump, in calories. Any other cell is unreachable from Mr. Frog's current position with a single jump.

7	6	5	6	7
6	3	2	3	6
5	2	F	2	5
6	3	2	3	6
7	6	5	6	7

Your task is to determine the minimum amount of energy that Mr. Frog needs to spend to get from his home to Ms. Toad's home.

Input

The input contains several test cases. The first line of a test case contains two integers, C and R , indicating respectively the number of columns and rows of the marsh ($1 \leq C, R \leq 1000$). The second line of a test case contains four integers C_f, R_f, C_t , and R_t , where (C_f, R_f) specify Mr. Frog's home location and (C_t, R_t) specify Ms. Toad's home location ($1 \leq C_f, C_t \leq C$ and $1 \leq R_f, R_t \leq R$). The third line of a test case contains an integer W ($0 \leq W \leq 1000$) indicating the number of watery places in the marsh. Each of the next W lines contains four integers C_1, R_1, C_2 , and R_2 ($1 \leq C_1 \leq C_2 \leq C$ and $1 \leq R_1 \leq R_2 \leq R$) describing a rectangular watery place comprising cells whose coordinates (x, y) are so that $C_1 \leq x \leq C_2$ and $R_1 \leq y \leq R_2$. The end of input is indicated by $C = R = 0$.

The input must be read from file frog.in.

Output

For each test case in the input, your program must produce one line of output, containing the minimum calories consumed by Mr. Frog to go from his home location to Ms. Toad's home location. If there is no way Mr. Frog can get to Ms. Toad's home, your program should output **impossible**.

The output must be written to standard output.

Sample input	Output for the sample input
4 4	14
1 1 4 2	impossible
2	12
2 1 3 3	
4 3 4 4	
4 4	
1 1 4 2	
1	
2 1 3 4	
7 6	
4 2 7 6	
5	
4 1 7 1	
5 1 5 5	
2 4 3 4	
7 5 7 5	
6 6 6 6	
0 0	

Problem C

Jukebox

Source file name: `jukebox.c`, `jukebox.cpp`, `jukebox.java` or `jukebox.pas`

The ICPC judges are preparing a party for the opening ceremony. For the party, they intend to add a playlist with some songs to the jukebox software (a simple MP3 player). However, there are so many songs in the computer that it is difficult to find the ones they want to add. As a consequence, they need to use the search feature many times.

In this jukebox, when you search for a string s , the software returns every music whose title or artist name contains s as a substring. String s is a substring of string t if t contains all characters of s as a contiguous sequence (for example, ‘bc’ is a substring of ‘abcd’, but ‘ac’ is not). To save their precious time, while looking for a song, they always use one of the song’s *golden string*, i.e. one of the shortest strings for which the search returns as a result only the song they want.

		Search <input type="text"/>
○ My musics	Music	Artist
	a_flor	los_hermanos
	anna_julia	los_hermanos
	quem_sabe	los_hermanos
	pierrot	los_hermanos
	azedume	los_hermanos
	johnny	massacration
	john	massacration
	johnnatan	massacration

In the example above, a possible golden string for the song ‘johnnatan’ is ‘ta’. Note that ‘ta’ is not a substring of the name of another song nor a substring of the artist of another song. Note also that there is no string of size equal to 1 that could identify uniquely the song ‘johnnatan’.

		Search <input type="text"/>
○ My musics	Music	Artist
	a_flor	
	anna_julia	
	quem_sabe	
	pierrot	
	azedume	
	johnny	
	john	massacration
	johnnatan	

They discovered that if they remove the artist fields from some of the songs they can get even smaller golden strings. For the song ‘john’, there is no golden string. However, if one removes the artist field from all other songs, the string ‘c’ becomes the golden string for the song ‘john’.

Given the song list (each song with name and artist), your job is to determine the minimum sum of the golden string sizes for all songs that can be obtained if one is allowed to remove some of the artist fields. In the figure above you can see a possible best result with the golden strings in bold. The minimum sum of the golden string sizes in this case is 10.

Input

The input contains several test cases. The first line of each test case contains one integer N ($1 \leq N \leq 30$), which indicates the number of songs. Following there will be N pairs of lines ($2 * N$ lines), one pair for each song. The first line of a pair will contain the song name, the second line will contain the artist name. Both artist and song names are strings containing only lower case letters or underlines and having at least 1 and at most 30 characters. There will be at most 6 different artists in the list.

The end of the input is given by $N = 0$.

The input must be read from file `jukebox.in`.

Output

For each test case your program must output one single line with the minimum sum of the golden string sizes. You may assume that there will always be a solution.

The output must be written to standard output.

Sample input	Output for the sample input
8 a_flor los_hermanos anna_julia los_hermanos quem_sabe los_hermanos pierrot los_hermanos azedume los_hermanos johnny massacration johnnatan massacration john massacration 4 c axc b axc d cc xc cc 0	10 5

Problem D

Bubble Maps

Source file name: `maps.c`, `maps.cpp`, `maps.java` or `maps.pas`

Bubble Inc. is developing a new technology for browsing a map at different zoom levels. Their new technology assumes that the region to be mapped is a rectangular plane surface and it divides this surface in rectangular sub-regions, which represent deeper zoom levels.

Bubble Inc. technology represents maps using a structure known as *quad-tree*. In a quad-tree, a rectangular region named x may be divided in half, both horizontally and vertically, resulting in four equal-sized rectangular sub-regions. Those sub-regions are called child regions of x , and are named xp for the top-left, xq for the top-right, xr for the bottom-right and xs for the bottom-left regions, where xc represents the concatenation of string x and character $c =$ ‘p’, ‘q’, ‘r’ or ‘s’. For example, if the base region to be mapped is called m , the child regions of m are, from top-left in clockwise order: mp , mq , mr and ms , as illustrated below.

mp	mq
ms	mr

Any region can be further subdivided. For example, the region named ms can be further divided into sub-regions msh , msq , msr and mss , as illustrated below.

msh	msq
mss	msr

As another example, the figure below shows the result of subdividing the child sub-regions of the region named msr .

msrpp	msrpq	msrqp	msrqq
msrps	msrpr	msrqs	msrqr
msrsp	msrsq	msrrp	msrrq
msrss	msrsr	msrrs	msrrr

Sub-regions with names of the same length have the same zoom level, since they represent regions of the same size. Sub-regions in the same zoom level that share a common side are said to be *neighbors*.

Anything that lies outside the base region m is not mapped and, for every zoom level, all sub-regions of m are mapped.

Bubble’s map technology provides a way for the user to navigate from a given sub-region to neighboring sub-regions in the directions up, down, left and right. Your mission is to help Bubble Inc. in finding the neighboring sub-regions of a given sub-region. That is, given the name of a rectangular sub-region, you must determine the names of its four neighboring sub-regions.

Input

The input contains several test cases. The first line contains one integer N indicating the number of test cases. Each of the following N lines represents a test case, containing the name

of a region composed by C characters ($2 \leq C \leq 5000$), the first always being the letter 'm' and the following being either 'p', 'q', 'r' or 's'.

The input must be read from file maps.in.

Output

For each test case in the input your program must produce one line of output, containing the names of the four neighboring regions of the given region in the order of direction up, down, left, right. For the neighbors that are not mapped you should output `<none>` instead of its name. Leave one blank space between two consecutive names.

The output must be written to standard output.

Sample input	Output for the sample input
2 mrspr mps	mrsrq mrssq mrsps mrsqs mpp msp <none> mpr

Problem E

Onion Layers

Source file name: onion.c, onion.cpp, onion.java or onion.pas

Dr. Kabal, a well recognized biologist, has recently discovered a liquid that is capable of curing the most advanced diseases. The liquid is extracted from a very rare onion that can be found in a country called Onionland. But not all onions of Onionland are worth to take to the lab for processing. Only those onions with an odd number of layers contain the miraculous liquid. Quite an odd discovery!

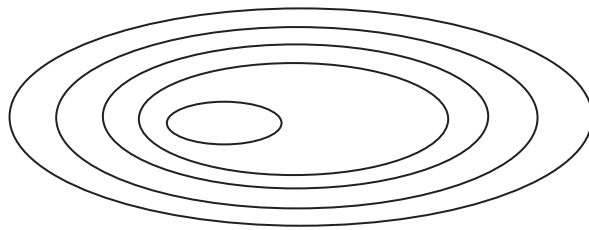


Figure 1: Onion from Onionland

Dr. Kabal has hired a lot of research assistants to collect and analyse onions for him. Since he does not want to share his discovery with the world yet, he didn't tell the assistants to look for onions with an odd number of layers. Instead, each assistant was given the task of collecting onions, and selecting points from each of the layer's outer borders, so that an approximation of the layer structure of the onion can be reconstructed later. Dr. Kabal told the assistants that the next step will be a "complicated analysis" of these points. In fact, all he will do is simply to use the points to count the number of layers in each of the onions, and select the ones with an odd number of layers.

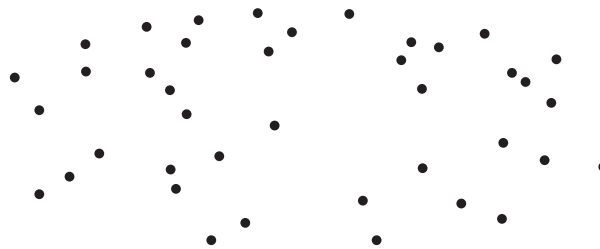


Figure 2: Points collected by an assistant

It is clear that the approximation obtained by Dr. Kabal, from the points collected, might have a different *shape* than the original onion. For instance, only some of the points of the onion shown in Figure 1 would be extracted in the process, giving rise to a set of points as shown in Figure 2. With these points Dr. Kabal will try to approximate the original layers of the onion, obtaining something like what is shown in Figure 3. The approximation procedure followed by Dr. Kabal (whose result is shown in Figure 3) is simply to recursively find nested convex polygons such that at the end every point belongs to precisely one of the polygons. The assistants have been told to select points in such a way that the *number of layers in the*

approximation, if done in this recursive manner, will be the same as in the original onion, so that is fine with Dr. Kabal. The assistants are also aware that they need at least three points to approximate a layer, even the innermost one.

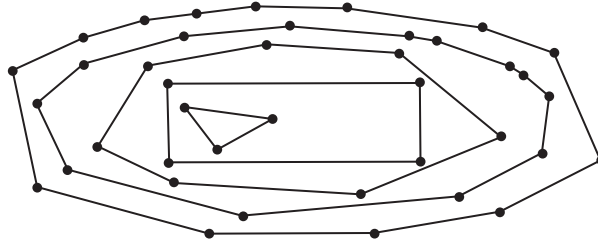


Figure 3: Dr. Kabal's approximation

Your task is to write a program that, given a set of points collected by an assistant (as shown in Figure 2), determines if the respective onion should be taken to the laboratory.

Input

The input contains several test cases. Each test case consists of an integer $3 \leq N \leq 2000$ in a single line, indicating the number of points collected by the assistants. Following, there are N lines, each containing two integers $-2000 \leq X, Y \leq 2000$ corresponding to the coordinates of each point. The input is finished by a problem with $N = 0$ points, which should not be processed.

The input must be read from file onion.in.

Output

There should be one line of output for each test case in the input. For each test case print the string

Take this onion to the lab!

if the onion should be taken to the laboratory or

Do not take this onion to the lab!

if the onion should not be taken to the laboratory.

The output must be written to standard output.

Sample input	Output for the sample input
7 0 0 0 8 1 6 3 1 6 6 8 0 8 8 11 2 6 3 2 6 6 0 0 0 11 1 1 1 9 7 1 7 9 8 10 8 0 0	Do not take this onion to the lab! Take this onion to the lab!

Problem F

Odd or Even

Source file name: odd.c, odd.cpp, odd.java or odd.pas

There are several versions of Odd or Even, a game played by competitors to decide random issues (such as “who will code this problem?”). In one of the versions, for two players, the game starts with each player calling either odds or evens. Then they count to three (some people chant “Once, twice, three, SHOOT!”). On three, both players hold out one of their hands, showing a number of fingers (from zero to five). If the fingers add to an even number, then the person who called evens wins. If the fingers add to an odd number, then the person who called odds wins.

John and Mary played several games of Odd or Even. In every game John chose odds (and, consequently, Mary chose evens). During the games each player wrote down, in small cards, how many fingers he/she showed, using one card for each game – Mary used blue cards, John used red cards. Their objective was to be able to re-check the results later, looking at the cards for each game. However, at the end of the day John dropped the deck of cards, and although they could separate the cards by color, they are now out of order.

Given the set of numbers written on red cards and on blue cards, you must write a program to determine the minimum number of games that Mary certainly won.

Input

The input contains several test cases. The first line of a test case contains an integer N representing the number of games played ($1 \leq N \leq 100$). The second line of a test case contains N integers X_i , indicating the number of fingers shown by Mary in each of the games ($0 \leq X_i \leq 5$, for $1 \leq i \leq N$). The third line of a test case contains N integers Y_i , indicating the number of fingers shown by John in each of the games ($0 \leq Y_i \leq 5$, for $1 \leq i \leq N$). The end of input is indicated by $N = 0$.

The input must be read from file odd.in.

Output

For each test case your program must write one line, containing one integer, indicating the minimum number of games that Mary certainly won.

The output must be written to standard output.

Sample input	Output for the sample input
3	0
1 0 4	3
3 1 2	
9	
0 2 2 4 2 1 2 0 4	
1 2 3 4 5 0 1 2 3	
0	

Problem G

Power Generation

Source file name: `power.c`, `power.cpp`, `power.java` or `power.pas`

Demand for electricity grew rapidly in the country over recent years, and is projected to grow even faster in the next twenty years. To cope with this increase in demand, the government is planning to privatize the country's electricity power-generation sector, ending the monopoly of the state-owned company, ICPC (Independent Circuit Power Corporation).

ICPC owns a set of power-generation plants (hydroelectric and nuclear). ICPC's plants are connected by *power lines* that cross the country. Each power line connects two distinct power plants and is constructed in a straight line. A *power path* is a sequence of power lines l_1, l_2, \dots, l_m , with each power line l_i connecting directly plants p_{i-1} and p_i , such that any two consecutive power lines l_i and l_{i+1} are linked to a common power plant p_i .

Power plants were built over several years, one at a time, due to budget restrictions. Also due to budget restrictions, every time a new power plant was built, only one new power line was constructed to integrate the new plant to the existing ICPC system. The new power line always linked the newly built power plant to the nearest power plant already in the system. If more than one such plant existed (that is, if more than one plant was located at a minimum distance from the new plant), the oldest plant was chosen.

In the privatization project, the aim is to break up the ICPC power-generation system into smaller companies, each company owning a set of power plants (each power plant will be owned by only one company). After the privatization, ICPC will cease to exist; only the new companies will own the power plants. The division of power plants among new companies must obey the following restrictions:

- the *total capacity* of every new company must be at least C , where C is a value in MW (Mega Watts) decided by the government. The total capacity of a set of power plants is the sum of capacities of those plants;
- power paths between any two plants owned by a new company must include only plants owned by that company.

You have been hired by ICPC to determine which is the largest number of new companies that can be created in the privatization process.

Input

The input contains several test cases. The first line of a test case contains two integers N and C indicating respectively the total number of power plants owned by ICPC ($1 \leq N \leq 10000$) and the minimum total capacity, in MW, that every new company must have ($1 \leq C \leq 10000$). Power plants are identified by integers from 1 to N ; plant 1 was the first to be built, plant 2 the second to be built, and so on. Each of the next N lines describes a power plant; the first line describes power plant 1, the second line describes power plant 2, and so on. Each description consists of three integers X , Y and P , where (X, Y) is the plant location ($0 \leq X \leq 1000$ and $0 \leq Y \leq 1000$) and P is the plant capacity ($1 \leq P \leq 1000$). Plants were built at different locations (that is, no two plants have the same location). The end of input is indicated by $N = C = 0$.

The input must be read from file `power.in`.

Output

For each test case in the input your program must produce one line of output, containing only one integer: the largest number of new companies that can be created in the privatization process.

The output must be written to standard output.

Sample input	Output for the sample input
2 22	1
0 0 20	2
10 20 30	0
4 430	
10 20 100	
20 10 400	
50 10 50	
25 25 500	
3 100	
10 10 33	
0 10 33	
10 0 33	
0 0	

Problem H

Report Recovery

Source file name: `report.c`, `report.cpp`, `report.java` or `report.pas`

At the end of the week, John asked Mary to send him an urgent sales report. Mary was in a hurry because she was leaving for her holiday. She then copy-pasted the sales sheet on an email, sent it to John and went out. She did not want to be annoyed with work issues, so she left without telling anyone where she would be. She announced that she would be simply not available for the next two weeks, turned off her cell phone, and left.

When John received the message he realized that the report had no spaces at all! He knew that the report should have a header line with product codes of the form `P1`, `P2`, ..., `PN` and the word `Totals` at the end. Then there would be several lines reporting product sales for the different sellers of Mary's office. Each seller was identified with a name composed by one word (only alphabetical characters). The line corresponding to a seller should begin with his/her name, followed by the number of sold products, according to the columns' report. The last line of the report should begin with the two letters `TP` followed by the totals of each column in the report (of course, no seller's name began with the letters `TP`). John knew that there were no negative numbers in the report, a zero quantity was reported with a single `0`, and there were no leading zeros when reporting a positive quantity.

At this point, John decided to reconstruct Mary's report. He knew that there could be more than one possible result, but he wanted to do it anyway with the first consistent solution that he could find (maybe he could fix any mistakes when Mary comes back).

Could you help John with the recovering of Mary's sales report?

Input

The input consists of several test cases. The first line in the input contains an integer C specifying the number of test cases. The first line of a report is a header line, containing the product codes `P1`, `P2`, ..., `PN` and the word `Totals`, as described above. The numbering of products in this header line is consecutive, from 1 to N , with $1 \leq N \leq 5$. Then there are a number of lines, each representing a row of the report, as described above. The last line of the report starts with the letters `TP` and have the format described above. Consider that each seller sold less than 1000 units of each product. There are no more than 4 sellers on each test case. Each seller name will not exceed 10 characters (only uppercase and lowercase letters).

The input must be read from file `report.in`.

Output

For each test case in the input your program must produce one possible Mary's report. Each line of the answer must be left aligned, with its items separated by a single space, and with no space at its end.

The output must be written to standard output.

Sample input	Example of output for the sample input
2 P1P2P3Totals Amanda121100131 Charles5141772 Monique14121238 TP1862629241 P1P2Totals Ingrid9519851936 Candid49212504 Peter10313 Camila000 TP145310002453	P1 P2 P3 Totals Amanda 121 10 0 131 Charles 51 4 17 72 Monique 14 12 12 38 TP 186 26 29 241 P1 P2 Totals Ingrid 951 985 1936 Candid 492 12 504 Peter 10 3 13 Camila 0 0 0 TP 1453 1000 2453

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 ($-64 \leq X_i \leq 64$, 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 file roulette.in.

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.

Sample input	Output for the sample input
4 2	4
-1 0 2 -1	-11
-1 1	56
5 2	10
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	