The 2010 ACM ASIA Programming Contest Dhaka Site

Sponsored by IBM

Hosted by North South University
Dhaka, Bangladesh

6th November 2010
You get 21 Pages
10 Problems
&
300 Minutes

Rules for ACM-ICPC 2010 Asia Regional Dhaka Site:

a) Solutions to problems submitted for judging are called runs. Each run is judged as accepted or rejected by the judge, and the team is notified of the results. Submitted codes should not contain team or University name and the file name should not have any white space.

b) Notification of accepted runs will **NOT** be suspended at the last one hour of the contest time to keep the final results secret. Notification of rejected runs will also continue until the end of the contest. But the teams will not be given any balloon and the public rank list will not be updated in the last one hour.

c) A contestant may submit a clarification request to judges. If the judges agree that an ambiguity or error exists, a clarification will be issued to all contestants.

d) Contestants are not to converse with anyone except members of their team and personnel designated by the organizing committee while seated at the team desk. **But they cannot even talk with their team members when they are walking around the contest floor to have food or any other purpose.** Systems support staff may advise contestants on system-related problems such as explaining system error messages.

e) While the contest is scheduled for a particular time length (five hours), the Chief Judge has the authority to alter the length of the contest in the event of unforeseen difficulties. Should the contest duration be altered, every attempt will be made to notify contestants in a timely and uniform manner.

f) **A team may be disqualified by the Chief Judge** for any activity that jeopardizes the contest such as dislodging extension cords, unauthorized modification of contest materials, or distracting behavior. The **external judges** will report to the chief judge about distracting behavior of any team. **The external judges can also recommend penalizing a team with additional penalty minutes for their distracting behavior.**

g) Nine, ten or eleven problems will be posed. So far as possible, problems will avoid dependence on detailed knowledge of a particular applications area or particular contest language. Of these problems at least two will be solvable by a first year computer science student, another one will be solvable by a second year computer science student and rest will determine the winner.

h) Contestants will have foods available in their contest room during the contest. So they cannot leave the contest room during the contest without permission from the external judges. The contestants are not allowed to communicate with any contestant (Even contestants of his own team) or coach while are outside the contest floor.

i) Team can bring up to 200 pages of printed materials with them but they can also bring three additional books. But they are not allowed to bring calculators or any machine-readable devices like CD, DVD, Pen-drive, IPOD, MP3/MP4 players, floppy disks etc. They are requested not to bring calculator they got on mock day at the date of actual contest.

j) With the help of the volunteers and external judges, the contestants can have printouts of their codes for debugging purposes.

k) The decision of the judges is final.

l) Teams should inform the volunteers if they don't get reply from the judges within 10 minutes of submission. Volunteers will inform the External Judge and the external judge will take further action. Teams should also notify the volunteers if they cannot log in into the PC^2 system. This sort of complains will not be entertained after the contest.

m) If you want to assume that judge data is weaker than what is stated, then do it at your own risk :).

A Emoogle Balance ☺

Input: Standard Input
Output: Standard Output

We have a very famous and popular fellow in our problemsetters' panel. He is so famous that his name is immaterial. Some of his admirers have recently given him the nickname 'Emoogle'. Let's stick to that name in our discussion for now. Being such a kind, friendly and generous person as he is, Emoogle is often known to give treats to the other problemsetters. Some times, there is a strange rumor in the air that his treats are mostly due to the fact that, if he is not sparing enough for those treats, 'problems' are likely to be created. But let's not pay heed to such nonsense!

Now, there is another word in the air that this remarkable man is going to get married soon. To observe this special occasion with proper respect, his fellow troublemakers have decided to compile a book named '99 reasons why Emoogle should give us a treat'. Every single reason mentioned in this book is denoted by a number. For example, Emoogle should give us a treat because -

1. If he does not, problems will be created. :)
2. His giveaway problem has been solved by less than 10 teams in the recent programming contest.
3. He is going to join a world famous goggles manufacturing company soon.
4. He has found a ticket of a soccer world cup game while digging his backyard garden in the morning.
5. He has just got a new Facebook fan club.
6. Having forgotten about a date with his wife-to-be which collided with a Topcoder SRM (Single Round Match), he participated in the SRM. (May God bless his soul!)
7. A programming contest (may be this one?) is being arranged celebrating his marriage.
8. He is getting engaged soon.

.....

99. Solely because he is the great and kind and sweet Emoogle.

If you have any more ideas about why he should throw a party, we would love to know. Drop us a line at emoogle.party@gmail.com.

At this point, Dear brother Emoogle might want to remind us about the number of times he has already thrown a party. Hence we introduce the term Emoogle Balance. This is defined as :

Emoogle Balance = number of times Emoogle is supposed to give a treat according to the book - number of times he has actually given the treat.

In this problem, we want you to find Emoogle Balance. We also wish that Emoogle Balance always keeps a healthy negative value and may dear brother Emoogle have a very happy married life forever.

Input

There are around **75** test cases in the input file. Each test case describes a series of events. A test case starts with an integer **N (1 ≤ N ≤ 1000)** denoting the number of events in this test case. This integer is followed by a line with **N** integers, each describing an event. These integers have values between **0** and **99** (inclusive). A value between **1** and **99** means a reason for Emoogle's giving a treat has occurred while a **0** means he has given a treat.

The end of input will be denoted by a case with **N = 0**. This case should not be processed.

Output

For each test case, print a line in the format, "**Case X: Y**", where **X** is the case number & **Y** is the Emoogle Balance for this case.

Sample Input

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

Output for Sample Input

```
Case 1: 1
Case 2: -2
Case 3: 3
```

B A Digital Satire…

Input: Standard Input
Output: Standard Output

The government of "Moderdesh" is planning to enter the digital age and so people of different profession and business are proposing different ways to enter that age successfully. The hardware vendors are saying that we need to provide a laptop for each student, the mobile companies are saying that every children needs to have a mobile phone in his small hand and talk all night long, the multimedia experts are crying for Multimedia University and so on. But very few are crying for the overall improvement of Computer Science education. We do not understand that by being only the consumer of modern digital technologies we cannot enter the real digital age.

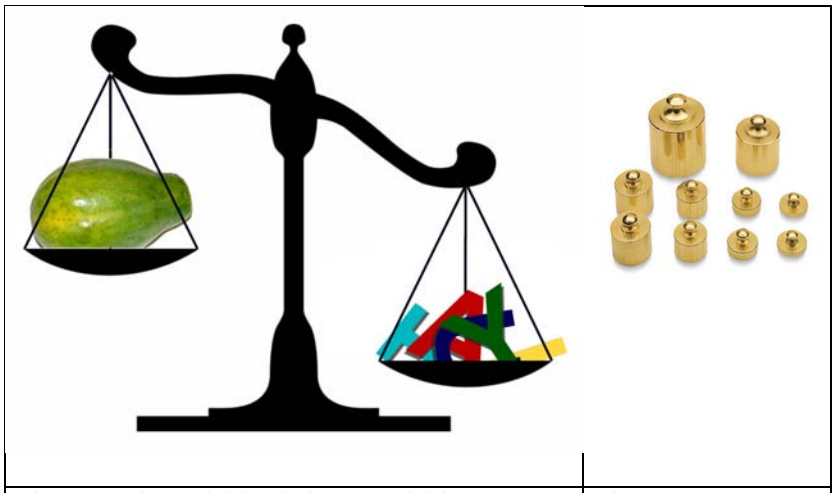


Figure 1: The weighing balance weighing a papaya using letter shaped weights.

Figure 2: Conventional weight set

Now as a protest, some local computer geeks are planning to digitize the local vegetable and grocery markets in a strange way. The local markets generally use weighing balances as shown in the figure-1 and they use conventional weight sets as shown in figure 2. The computer geeks want to introduce a new type of weight set, where each piece will have the shape of an upper case English alphabet, and strangely the weights of these pieces will be related with their ASCII values. For example the ASCII value of **'A'** is **65**, which is $\mathbf{1000001_2}$ in binary. For each **'1'** in binary representation a weight of **500** gms will be added and for each **'0'** in binary representation a weight of **250** gms will be added. So a piece with shape **'A'** actually weighs **(250*5+2*500)** gm = **2250** grams. Note that leading zeroes in binary representation are not considered. The geeks believe if others are correct about their ways to enter digital age, they are also correct about digitizing the local markets by introducing new weight sets related to ASCII characters.

Now in this problem you will be given (i) the picture of a weighing scale and the weight pieces that it contains in left pan and in the right pan (ii) You will also be informed which pan is heavier and which pan is lighter (Not necessarily correct). You will have to detect whether the given information is correct or not. If the given information is not correct you will have to rectify the picture and show it in the output.

Input

First line of the input file contains a positive integer **T (T ≤ 6000)** that denotes the number of test sets. The description of each set of input is given below:

Each set of input is given in a **(7*18)** grid. This grid actually contains the plain text description of a weighing scale. Each location of the grid is either a dot **'.'** (ASCII value **46**) or a front slash **'/'** (ASCII Value **47**) or a back slash **'\'** (ASCII value **92**) or an under score **'_'** (ASCII value **95**) or a pipe **'|'** (ASCII value **124**) an upper case English Alphabet (ASCII value **65** to **90**). The **(7*18)** grid is divided into two equal parts by two vertical lines formed with pipe character. The left part denotes the status of left pan and right part denotes the status of the right pan. The bottom of the pan is formed with **6** (six) under score characters and the ropes attached to the pans are formed with front slash and back slash. The weights on both pans are placed just above the row that denotes bottom of the pan and they are left justified. There can be maximum **6** weights on a single pan. There are three possible vertical positioning of the pans (i) Left pan is low and right pan is high (ii) Both pan is in the middle (iii) Left pan is high and right pan is low. If weight of both pan is same then they should be in state (ii), if the left pan is heavier then they should be in

C Hyper-Box

Input: Standard Input
Output: Standard Output

You live in the universe **X** where all the physical laws and constants are different from ours. For example all of their objects are **N**-dimensional. The living beings of the universe **X** want to build an **N**-dimensional monument. We can consider this **N** dimensional monument as an **N**-dimensional hyper-box, which can be divided into some **N** dimensional hyper-cells. The length of each of the sides of a hyper-cell is one. They will use some **N**-dimensional bricks (or hyper-bricks) to build this monument. But the length of each of the **N** sides of a brick cannot be anything other than fibonacci numbers. A fibonacci sequence is given below:

1, 2, 3, 5, 8, 13, 21….

As you can see each value starting from **3** is the sum of previous **2** values. So for **N=3** they can use bricks of sizes **(2,5,3)**, **(5,2,2)** etc. but they cannot use bricks of size **(1,2,4)** because the length **4** is not a fibonacci number. Now given the length of each of the dimension of the monument determine the minimum number of hyper-bricks required to build the monument. No two hyper-bricks should intersect with each other or should not go out of the hyper-box region of the monument. Also none of the hyper-cells of the monument should be empty.

Input

First line of the input file is an integer **T(1≤T≤100)** which denotes the number of test cases. Each test case starts with a line containing **N (1≤N≤15)** that denotes the dimension of the monument and the bricks. Next line contains **N** integers the length in each dimension. Each of these integers will be between **1** and **2000000000** inclusive.

Output

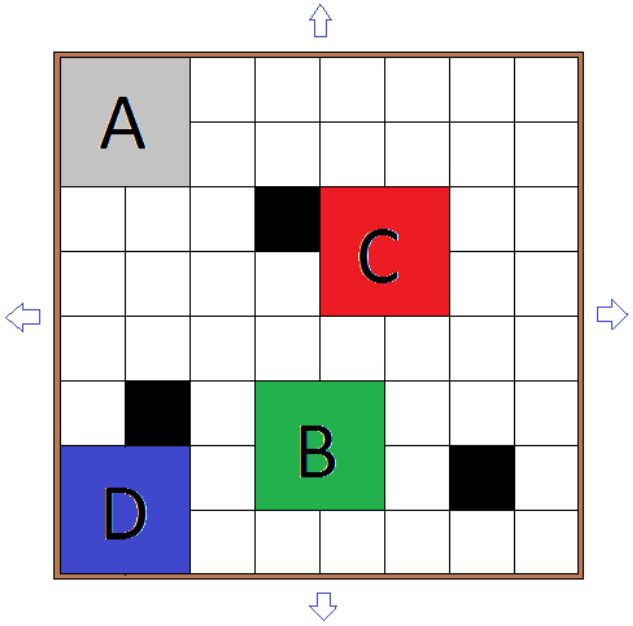
For each test case output contains a line in the format **Case x: M** where **x** is the case number (starting from **1**) and **M** is the minimum number of hyper-bricks required to build the monument.

Sample Input	Output for Sample Input
2 2 4 4 3 5 7 8	Case 1: 4 Case 2: 2

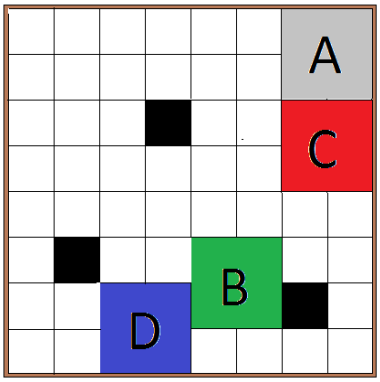
D OmniGravity

Input: Standard Input
Output: Standard Output

In this problem, we will play with four 2x2 squared pieces, an 8x8 board, few obstacles and gravity(!!!).



Initially the board contains few obstacles and the 4 colored pieces randomly placed. An example of an initial configuration is shown in the above diagram. The black squares represent the obstacles and the 4 colored pieces are shown with labels A, B, C and D - (in order to help the colorblind people). Initially there is no gravity and thus the pieces remain at a steady position. We can enable 'gravity' in any of the four directions to move the pieces. The pieces move in the direction of gravity. A piece continues to move until it hits an edge of the board, an obstacle or any other piece. The obstacles aren't affected by gravity and so remains in their initial position at all times. We can enable at most one 'gravity' at a time. The gravitational direction can only be changed when all the pieces are static.



The diagram to the left shows the positions of the pieces after the 'gravity' from the right is enabled.
How many different static states can be reached by enabling at least one 'gravity'? Two states are different if there is at least once cell that has a different color. A static state is one in which all the pieces are steady!

Input

The first line of input is an integer **T (T≤100)** that indicates the number of test cases. Each case consists of **8** lines with **8** characters in each line. Every character will be from the set **{A, B, C, D, ., #}**. Dots(.) represent empty spaces, hashes(**#**

E Halloween Costumes

Input: Standard Input
Output: Standard Output

Gappu has a very busy weekend ahead of him. Because, next weekend is Halloween, and he is planning to attend as many parties as he can. Since it's Halloween, these parties are all costume parties, Gappu always selects his costumes in such a way that it blends with his friends, that is, when he is attending the party, arranged by his comic-book-fan friends, he will go with the costume of Superman, but when the party is arranged contest-buddies, he would go with the costume of 'Chinese Postman'.

Since he is going to attend a number of parties on the Halloween night, and wear costumes accordingly, he will be changing his costumes a number of times. So, to make things a little easier, he may put on costumes one over another (that is he may wear the uniform for the postman, over the superman costume). Before each party he can take off some of the costumes, or wear a new one. That is, if he is wearing the Postman uniform over the Superman costume, and wants to go to a party in Superman costume, he can take off the Postman uniform, or he can wear a new Superman uniform. But, keep in mind that, Gappu doesn't like to wear dresses without cleaning them first, so, after taking off the Postman uniform, he cannot use that again in the Halloween night, if he needs the Postman costume again, he will have to use a new one. He can take off any number of costumes, and if he takes off **k** of the costumes, that will be the last **k** ones (e.g. if he wears costume **A** before costume **B**, to take off **A**, first he has to remove **B**).

Given the parties and the costumes, find the minimum number of costumes Gappu will need in the Halloween night.

Input

First line contains **T (T ≤ 2500)**, the number of test cases.

Each test case starts with two integers, **N** and **M(1≤N,M≤100)**, the number of parties, and number of different types of costumes. Next line contains **N** integers, c_i **(1 ≤** c_i **≤ M)**, the costume he will be wearing in party **i**. He will attend the party **1** first, then party **2**, and so on.

There is a blank line before each test case.

Output

For each test case, output the minimum number of required costumes. Look at the output for sample input for details.

Sample Input	Output for Sample Input
4 1 1 1 2 2 1 1 3 2 1 2 1 7 3 1 2 1 1 3 2 1	Case 1: 1 Case 2: 1 Case 3: 2 Case 4: 4

F Digital Matrix

Input: Standard Input
Output: Standard Output

You are given two **N x N** square matrices, **A** and **B**. Each of the elements of these matrices is an integer between **1** and **K**(inclusive). You have to convert matrix **A** into matrix **B** in minimum number of operations. In each operation you can choose one element of matrix **A** and change it to any integer between **1** and **K** (inclusive). You have to ensure that after any operation the matrix is not converted to a symmetric matrix. A square matrix is said to be symmetric if j^{th} element of i^{th} row is equal to the i^{th} element of j^{th} row for all **(i, j)** where $1 \le i \le N$ and $1 \le j \le N$. For example –

$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 2 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$
Symmetric Matrix	Non-symmetric Matrix

Input

Input will start with an integer **T (T ≤ 200),** number of test cases. Each test case starts with a line containing two integers **N (1 ≤ N ≤ 100)** and **K (1 ≤ K ≤ 9)**. This line will be followed by **2N** lines. First **N** lines will represent matrix **A** and next **N** line will represent matrix **B**. Each of these **2N** lines will contain **N** integers, all of these integers are in between **1** and **K** (inclusive).

Output

For each test case, output a single line containing the case number followed by the minimum number of operations required to convert **A** into **B**. If it is impossible to convert **A** into **B** obeying the rules, print **-1** instead. See output for sample input for exact formatting.

Sample Input	Output for Sample Input
3 3 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 2 3 1 2 1 1 1 1 3 1 2 3 1 2 3 1 1 3 2 1	Case 1: 0 Case 2: 2 Case 3: 3

Warning: Don't use ***cin***, ***cout*** for this problem, use faster i/o methods e.g ***scanf***, ***printf***.

G Knockout Tournaments

Input: Standard Input
Output: Standard Output

Knockout tournaments are sometimes heart breaking for players but still they are very much common in real world because the audience love them so much. Some tournaments have round robin format initially and then they turn into knockout tournaments in latter rounds (Like FIFA Soccer World Cup). On the other hand most lawn tennis tournaments are knockout tournaments from the very first round. Such a knockout tournament with **n** rounds has $\mathbf{2^n}$ teams. The first round can be considered as round **1** and the final match can be considered as round **n**. Bob has been participating in computer created tennis tournaments in his laptop computer for a long time now. He keeps track of how many matches he has won and how many matches he has lost and at the end of **10**, **20** or **30** years he starts to calculate his performance. But the problem is that he cannot keep track of what round he has reached in which tournament and also in how many tournaments he has participated in because he plays thousands of computer-generated matches per day. Now your job is to assist him to calculate his average performance considering that each possible scenario has equal weight.

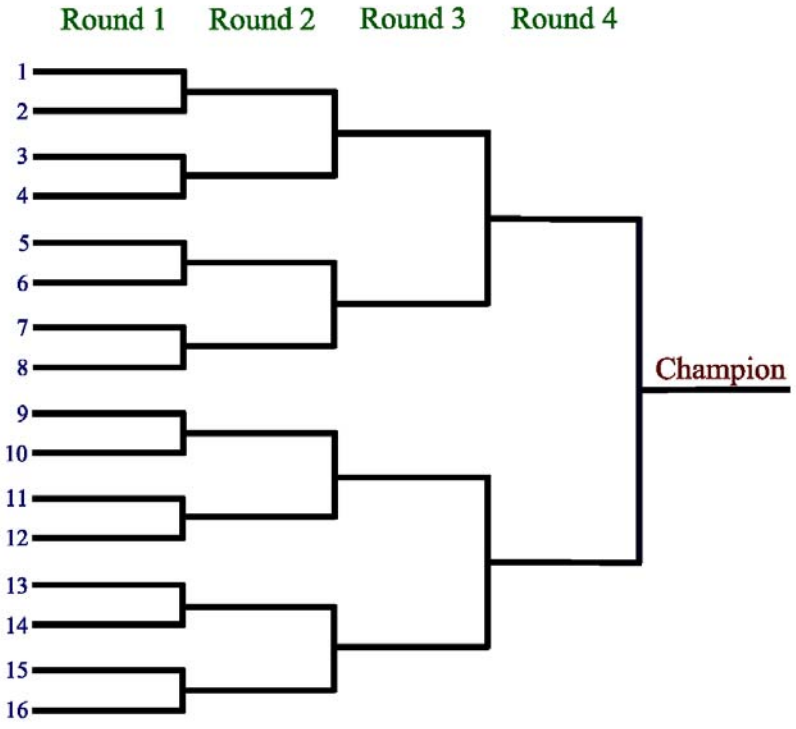


Figure: A knockout tournament structure with four rounds and 16 (2^4) players. But of course in this problem all tournaments has 8 rounds and 256 players.

You can assume the following thing for simplicity:

(a) All the tournaments that Bob participates in have exactly **8** (eight) rounds.

(b) Bob can be ousted from a tournament if and only if he loses a match. Every match has two possible outcomes (i) Winning (ii) Losing.

(c) If he wins a match at round **n** he moves to round **(n+1)** if **(n<8)**. At round **8** if he wins a match he becomes the champion but does not move to round **9** (Remains at **8**).

Your job is to write a program that tells bob about his average performance.

Input

The input file contains at most 2005 lines of inputs. Each line contains two integers **W (0≤W≤200000000)** and **L(0≤L≤200000000)**. Here **W** denotes the number of matches Bob has won in a specific time and **L** denotes the number of matches he has lost in that same time. Input is terminated by a line containing two zeroes. This line should not be processed.

Output

For each line of input produce two lines of output. First line contains the serial of output. Next line describes his average performance considering all possible scenarios. The floating-point number should be rounded to two digits after the decimal point. Two scenarios are different if and only if number of tournaments played is different. In determining average you must assume that all possible scenario has equal weight. If the given value of **W** and **L** does not illustrate any practical scenario print **"Situation Impossible."** (without the quotes) instead. Look at the output for sample input for formatting details. Judge input is such that small precision error would not cause difference in output.

Sample Input	Output for Sample Input
10 2 11 2 0 0	Case 1: On Average Bob Reaches Round 5.00 Case 2: On Average Bob Reaches Round 5.42

Illustration of 2nd sample input/output:

In the **2nd** sample input it is given that in a certain time Bob has **11** wins and **2** losses. If he participates in two tournaments then on average he reaches **6.5** round, and if he participates in three tournaments then he reaches on average round **4.33**. So considering all possible scenario he reaches round **(6.5+4.33)/2** ~ round **5.42**.

H Optimal Store

Input: Standard Input
Output: Standard Output

You live in a flat world and you have to carry some goods to three destinations **A, B, C** from a storeroom. You know the location of **A**, **B** and **C** and you have to find an optimal location **G** for the storeroom and build the storeroom at **G**. But for carrying goods you have only one truck available and that can drive through any place/location you want. The truck will initially be located at **G**. But this truck is not large enough to carry goods for more than one place at a time. So for minimum path covering what you do is:

1. Always drive from one place to another in straight line.
2. Load goods in the truck at **G**.
3. Carry these goods to the nearest destination to **G**.
4. Unload the goods at the nearest destination.
5. Drive the empty truck back to **G**.
6. Load good in the truck at **G**.
7. Carry these goods to the **2nd** nearest destination from **G**.
8. Unload the goods at the **2nd** nearest destination.
9. Drive the empty truck back to **G**.
10. Load goods in the truck at **G**.
11. Carry these goods to the farthest destination from **G**. And of course stay at **G**, as you have to carry nothing else.

If you had known the location of **G** then to find the minimum driving length would have been very easy. But for this problem your job is to find a location of **G** for which the total path length would be minimum and report this minimum driving length.

Input

The input file contains less than **11000** lines of input.

Each line contains six integer numbers $\mathbf{A_x, A_y, B_x, B_y, C_x, C_y}$. You can assume that $\mathbf{(0 \le A_x, A_y, B_x, B_y, C_x, C_y \le 1000)}$. These integers denote that the location of **A, B** and **C** in two-dimensional Cartesian coordinate system is $\mathbf{(A_x, A_y)}$, $\mathbf{(B_x, B_y)}$ and $\mathbf{(C_x, C_y)}$ respectively.

A line containing six negative numbers terminates the input.

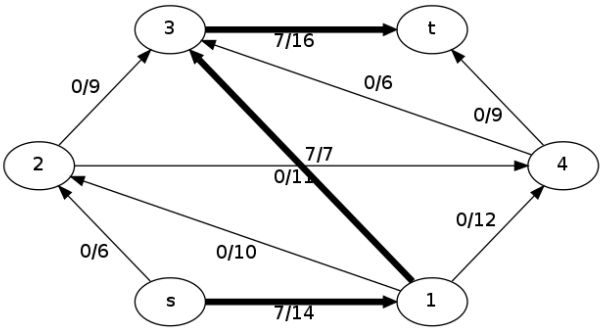
Output

For each line of input except the last one produce one line of output. This line contains the serial of output followed by a floating-point number **d**, which denotes the minimum driving length needed from the optimal location of **G**. This number should have eight digits after the decimal point. Errors less than $\mathbf{10^{-7}}$ will be ignored. Look at the output for sample input for details.

Sample Input	Output for Sample Input
0 0 15 0 8 1 -1 -1 -1 -1 -1 -1	Case 1: 22.20439337

I Network Flow

Input: Standard Input
Output: Standard Output



In a water refining plant, the flow of impure water is passed through a network consisting of straight-line water pipes. There are several pipes in the network. Each of them performs a particular operation in the refinement process. So, the impure water must pass through each of them. These pipes are placed on a 2D grid like surface and the two endpoints of each pipe can be described by a pair of integer coordinates. Every two pipes sharing an endpoint are joined at that location through a multi-way connector. Each connector has one or two pairs of openings. All the openings are connected to some pipe. In addition to the openings for joining pipes, each connector can be connected to a water tank. So, in a single refinement process, the tank full of impure water is connected to one of the connectors, all of the water is passed through the pipe networks as necessary and then purified water is brought back to the tank. It might be important to clarify that there is only one tank in the system and there is no additional pipe to send fresh water back to the tank once the purification is done. Also, in order to prevent overuse of the pipes, in a single refinement process water is allowed to pass through any pipe exactly once. This single flow through a pipe can be in any direction.

Now, waters can move freely through the pipes but if they need to change direction in the connectors, external energy must be supplied by using pumps. The amount of rotation can be expressed as a function of the pump's energy. For a network, TRA (Total Rotation Amount) can be calculated in the following way. After the flow is complete, a polygon is created by tracing the path of water flow inside the network. Then at each node of the polygon, we take the smallest angle between the two lines adjacent to it. By summing up these angle values for all the nodes, we get the TRA. A pump with **e** units of energy can provide $\mathbf{Ae^3 + Be^2 + Ce + D}$ full circle amount of TRA where **A, B, C, D** are non-negative integer constants.

You are given the description of a network and the **A, B, C, D, e** values of a pump. Your task is to determine the maximum percentage of energy that can remain unused after completing a single refinement process. Percentage can be found by the formula - **unused energy / total energy * 100.**

Input

There will be at most 40 cases in the input file. Each test case starts with a line with 6 integers, **E (3 ≤ E ≤ 20000), A, B, C, D (0 ≤ A, B, C, D ≤ 5) & e (0 ≤ e ≤ 50)**. **E** is the number of pipes in the network while **A, B, C, D** & **e** are the description of the pump as mentioned in the statement above. Each of the following **E** lines contains 4 integers: **x1, y1, x2, y2 (0 ≤ x1, y1, x2, y2 ≤ 10,000)**, denoting the coordinates of two endpoints of a pipe.

The final test case is followed by a line containing a single **0** denoting end of input.

Output

For each test case, print the case number at first. Then if it is possible to complete a flow in this network using the given pump, print the maximum possible percentage of unused energy rounded to **2** digits after decimal point. If it's not possible to complete a flow cycle, report so. Check sample input for exact formatting.

Sample Input	Output for Sample Input
4 1 1 1 1 1 1 2 2 1 2 1 1 0 1 0 0 1 0 1 1 2 4 1 1 1 0 0 1 2 2 1 2 1 1 0 1 0 0 1 0 1 1 2 0	Case 1: 75.00 Case 2: Impossible

J Hyper Knights

Input: Standard Input
Output: Standard Output

'Hyper Knights' is a puzzle game played in a 2D board. Though the game may not be too familiar to you, but in 'KnightsLand' it's a very popular game. The best thing of the game is that many people can play the game together competing others. The rules of the games are as follows:

Initially an **m x n** white board is taken and in each cell an integer is written. After that some cells are colored green and some cells are colored red. All the players are sent one copy of the board. Then each of the players starts placing Hyper Knights (like chess knights) in the board for one hour. No player can see others board.
When placing Hyper Knights, each player can use as many Hyper Knights as they want, but the jury accepts a board if the following constraints are fulfilled.

1) In each cell, at most one Hyper Knight can be placed.
2) A Hyper Knight should be placed in each green cell.
3) No Hyper Knight should be placed in the red cells.
4) No two Hyper Knights in the board should be in attacking positions.
5) A board with no Hyper Knights is rejected.


A Hyper Knight

Two Hyper Knights are said to be in attacking positions if

1) their vertical distance is **3** and horizontal distance is **1** or,
2) their vertical distance is **1** and horizontal distance is **3**

The scoring technique is quite simple. For a player's accepted board, if a cell contains a Hyper Knight, then the player is awarded a score same as the integer written in that cell. And for all his placed Hyper Knights, he sums up his scores. The player whose overall score is highest wins the game. If several players tie; all of them are declared as the winners.
Now, you are given a 'Hyper Knights' board as described, you have to find the maximum score a player can get in that board maintaining all the restrictions.

Input

Input starts with an integer **T (≤ 200)** denoting number of cases.

Each case starts with a black line. Next line contains two integers, **m** and **n (1 ≤ m, n ≤ 30)** denoting the board with **m** rows and **n** columns. The rows are numbered from **0** to **m - 1** and the columns are numbered from **0** to **n – 1**. Each of the next **m** lines contains **n** integers, separated by spaces. The **jth** integer in the **ith** line denotes the integer written in the cell in **ith** row and **jth** column. The absolute value of each integer will be less than 10^6.

Next line contains an integer **P** denoting the number of distinct green cells. Each of the next **P** lines contains two integers **i** and **j**, denoting that the cell in **ith** row and **jth** column is green.

Next line contains an integer **Q (Q < m * n)** denoting the number of distinct red cells. Each of the next **Q** lines contains two integers **i** and **j**, denoting that the cell in **i-th** row and **j-th** column is red.

If you place Hyper Knights in all green cells you are guaranteed that no two of them will be in attacking positions. And you may also assume that no cell will be colored both red and green.

Output

For each case, print a line containing the case number and the maximum score a player can make. After that you have to print the lexicographically smallest Hyper Knight placement that forms the maximum score. Print the row and column position of each Hyper Knight in separate lines. See the samples for details.

To check the lexicographical order we first make a list using each Hyper Knight placement as **[a_1, b_1, a_2, b_2 ... a_x, b_x]**, where cell **(a_i, b_i)** contains a Hyper Knight. After that we find the lexicographical order using these lists. For example, **[1, 2, 21, 5]** is smaller than **[1, 11]** and **[1, 1, 12, 13]** is smaller than **[1, 1, 12, 13, 1, 3]**.

Sample Input	Output for Sample Input
2 3 4 2 1 3 1 7 2 1 100 1 2 1 100 1 1 0 4 0 2 0 1 2 1 2 2 2 4 2 1 1 1 1 1 1 1 0 0	Case 1: 110 1 0 1 1 1 2 1 3 Case 2: 7 0 0 0 1 0 2 0 3 1 1 1 2

Warning: Don't use ***cin***, ***cout*** for this problem, use faster i/o methods e.g ***scanf***, ***printf***.

Problem A - Emoogle Balance

Stop laughing as soon as you can. Find the necessary part from the problem statement. You are half done. Count the number of non zero integers and the number of zero. Subtract the second from the first. All done.

Problem B – A Digital Satire…

This is the 2nd easy (With some doubt) problem of the set. There is nothing much to think, only issue is the lengthy implementation problem. There is also one critical case where the two pans are empty.

Problem C – Hyper-Box

For each length find the minimum number of fibonacci number whose sum is that length. This can be done greedily. Say for the dimension i this number is F_i. Then the final answer is $\prod_{i=0}^{N-1} F_i$.

Problem D - OmniGravity

This is a easy/mid simulation problem. Either BFS or DFS can be used to make transitions from different states. Since the board size is only 8x8, a state of S[8][8][8][8][8][8][8][8] to keep track of the upper left corner of each square should suffice.

Halloween Costume E - How to solve

Let's say, you need costumes: $c_1, c_2, c_3, \ldots, c_n$

Where, in i^{th} party, Gappu has to wear costume c_i. Let's say, in parties [i…j] (for some integers $i \le j$), the minimum number of costumes required is defined by MC (i, j). So, if Gappu wear the same costume in party i and party k, ($i < k \le j$), then he has to take off all the new costumes required to attend parties [i+1..j-1]. So, MC may be defined as:

$$MC(i,j) = \min_{i \le k \le j, c_i = c_k} \left(1 + MC(i+1, k-1) + MC(k+1, j)\right)$$

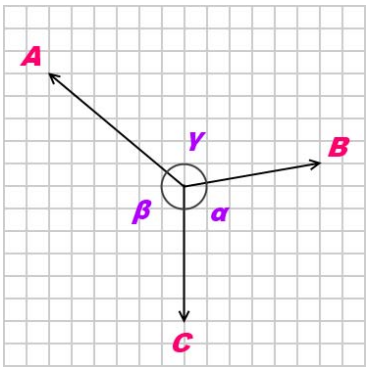
To attend party i-1, Gappu will be wearing costume ci-1. If $c_i = c_{i-1}$, you won't be needing a new costume to attend party i. You can modify the recurrence accordingly to handle this. You can solve this recurrence using Dynamic Programming.

Problem F - Digital Matrix

This is an easy looking problem. But some traps make this harder. There are few patterns of input matrices. Carefully handling all of these patterns is enough to solve this problem. So basically this is an if-else problem.

Problem G – Knockout Tournaments

Suppose with W wins and T losses the minimum and maximum number of tournaments that can be played is T_{min} and T_{max} respectively. The value of T_{min} and T_{max} can be found using some greedy strategy. It can be shown that if total T tournaments are played then always the average performance will be (L+W)/T. As all scenario (Difference in number of tournaments played creates different scenario). Have equal weight so the desired average is $\sum_{T=T_{min}}^{T_{max}} \frac{(L+W)}{T} = (L+W)\sum_{T=T_{min}}^{T_{max}} \frac{1}{T}$. So by pre-calculating some values of summation upto t-th term of the series $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{230000000}$ one can speed up the evaluation of the expression $(L+W)\sum_{T=T_{min}}^{T_{max}} \frac{1}{T}$.



Problem H – Optimal Store

Remember Lami's theorem ☺ ?

In statics, the Lami's theorem is an equation relating the magnitudes of three coplanar, concurrent and non-collinear forces, which keeps an object in static equilibrium, with the angles directly opposite to the corresponding forces. According to the law,

$$\frac{A}{\sin\alpha} = \frac{B}{\sin\beta} = \frac{C}{\sin\gamma}$$

where *A*, *B* and *C* are the magnitudes of three coplanar, concurrent and non-collinear forces, which keeps the object in static equilibrium, and *α*, *β* and *γ* are the angles directly opposite to the forces *A*, *B* and *C* respectively. This problem can also be solved using Lami's theorem.

Problem I – Network Flow

Network flow algorithm is not required by any means. What you acutally need is to find a euler cycle from the graph. If the graph is connected, it will always be possible to form a euler cycle. Try to process the nodes with degree 2 before the nodes with degree 4. In degree 4 nodes, keep the pairs of edges sorted by the angle between them and try to minimize this angle.

Problem J - Hyper Knights Solution

After constructing the graph, where each cell is a vertex and there is an edge betweens two cells if they are in attacking positions, you will find that the graph is bipartite. And your task is to find the maximum independent set in the graph which has the maximum summation. Since the graph is bipartite so, connect one end with the source and the other with the sink. And the capacity is the integer in that cell. And the capacity of other edges in the graph will be infinite. If a cell is green then capacity from source to that node (or that node to sink) is infinite. Avoid red cells. After that (total summation – min cut) is the result.

Since negative numbers and zeroes can occur in the input, you have to handle them separately. And to print a solution, it's clear that after the max flow, if the capacity of a node from source (or the node from sink) is not full, then it must be taken. After taking these nodes, try to take the nodes lexicographically,

if the capacity from source to that node (or that node to sink) is increased and the total flow towards sink increases then it can't be taken. Otherwise it can be taken. This part can be calculated without using the flow algorithm, with clever BFS; since the graph is bipartite.