UW-Madison's

2009 ACM-ICPC Individual Placement Test

October 9th, 1:00-6:00pm, CS1350

Overview:
This test consists of seven problems, which will be referred to by the following names (respective of order): causal, rsp, dollar, arctic, strings, forest, flatland.

Input/Output:
Your programs should take input from standard in (i.e., the keyboard) and output to standard out (i.e., the terminal). As is standard practice for the ICPC, you may assume that the input strictly follows the description in the problems. It is your responsibility to ensure that your output precisely matches that described in the problems, or else risk your program being rejected with a "Presentation Error".

Problem Submission:
Problem submission will be handled via the PC^2 judging software, as at the actual competition.

Clarifications:
Clarification requests will also be handled via the PC^2 software. Accepted clarification requests will be answered to all those taking the test. Experience of previous years learns that most clarification requests are rejected, and receive a simple response such as "Read the problem description".

Printing:
You may print to the printer at any time during the test.

Written Solutions:
We encourage you to spend the last half hour of the test to write down the main idea behind the solution to any of the problems for which you have not had a program accepted. Please be concise, using at most a few sentences within the space provided on the opposite side of this page. We will take these partial solutions into account along with your official ranking when composing teams, although they will have less weight.

After the Test:
The proctor will announce when time is up. Please stop working at this time and take a moment to fill out the form on the back of this sheet and turn it in to the poctor (you may keep the problems). You are invited to join us for pizza and soda after the test in 1325 CS.

Information:

Name: _____

CS Login:_____

PC^2 Login/Pass: _____ / _____

Student Status (e.g Junior, 1st-year grad):_____

Year of birth:_____, Year starting college:_____

How many ICPC regionals have you participated in?____ How many world finals?___

Which of C/C++/Java do you prefer:____ Please indicate your proficiency in each:

Which classes have you taken (or are taking) whichare relevent to the ICPC?

What do you feel your strengths are with respect to the ICPC?
Are you able to travel to the world finals in Sharm-el-Sheikh, Egypt, February 27 – March 4, 2011 (and obtain a visa / passport as necessary)?

If your team progresses to the world finals, how many hours per week could you commit to practicing, starting mid-November? ____

Are there people you would prefer to be (or not be) on the same team as?

Is there anything else we should know about you?

Written solutions for unsolved problems:

causal:

rsp:

dollar:

arctic:

strings:

forest:

flatland:

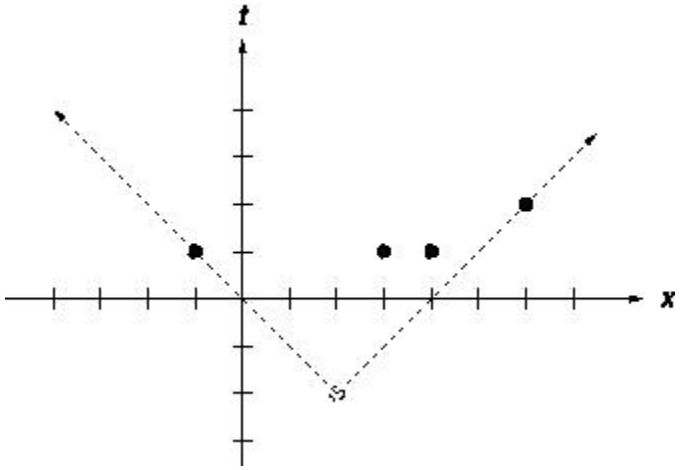
Problem B: Advanced Causal Measurements (ACM)

Causality is a very important concept in theoretical physics. The basic elements in a discussion of causality are *events*. An event e is described by its time of occurrence t, and its location, $\boldsymbol{x}$, and we write $e = (t,\boldsymbol{x})$. For our concerns, all events happen in the one dimensional geometric space and thus locations are given by a single real number x as a coordinate on x-axis. Usually, theoretical physicists like to define the speed of light to be 1, so that time and space have the same units (actual physical units frighten and confuse theorists).

One event $e_1 = (t_1,x_1)$ is a *possible cause* for a second event $e_2 = (t_2,x_2)$ if a signal emitted at e_1 could arrive at e_2. Signals can't travel faster than the speed of light, so this condition can be stated as:

$$e_1 \text{ is a } \textit{possible cause} \text{ for } e_2 \textbf{ iff } t_2 >= t_1+|x_2-x_1|$$

Thus an event at (-1,1) could cause events at (0,0), (1,2), and (1,3), for example, but could not have caused events at (1,4) or (-2,1). Note that one event can cause several others.



Recently, scientists have observed several unusual events in the geometrically one dimensional universe, and using current theories, they know how many causes were responsible for these observations, but they know nothing about the time and space coordinates of the causes. You asked to write a program to determine the latest time at which the earliest cause could have occurred (i.e. the time such that at least one cause must have occurred on or before this time). Somewhat surprisingly, all the observed events have both space and time coordinates expressed by integer numbers in the range $-1000000 \le t, x \le 1000000$.

The figure on the right illustrates the first case from input: the earliest single event as a possible cause of all four events.

The first line of input is the number of cases which follow. Each case begins with

a line containing the number n of events and the number m of causes, $1 \leq n, m \leq 100000$. Next follows n lines containing the t and x coordinates for each event.

Output consists of a single line for each case in the format as in the sample output, giving the latest time at which the earliest cause could have occurred, this will be an integer as our time units are not divisible.

Sample Input

```
4
4 1
1 -1
1 3
1 4
2 6
4 2
1 -1
1 3
1 4
2 6
4 3
1 -1
1 3
1 4
2 6
4 4
1 -1
1 3
1 4
2 6
```

Output for Sample Input

```
Case 1: -2
Case 2: 0
Case 3: 0
Case 4: 1
```

Daniel Robbins

Rock, Scissors, Paper

Bart's sister Lisa has created a new civilization on a two-dimensional grid. At the outset each grid location may be occupied by one of three life forms: *Rocks*, *Scissors*, or *Papers*. Each day, differing life forms occupying horizontally or vertically adjacent grid locations wage war. In each war, Rocks always defeat Scissors, Scissors always defeat Papers, and Papers always defeat Rocks. At the end of the day, the victor expands its territory to include the loser's grid position. The loser vacates the position.

Your job is to determine the territory occupied by each life form after *n* days. The first line of input contains *t*, the number of test cases. Each test case begins with three integers not greater than 100: *r* and *c*, the number of rows and columns in the grid, and *n*. The grid is represented by the *r* lines that follow, each with *c* characters. Each character in the grid is R, S, or P, indicating that it is occupied by Rocks, Scissors, or Papers respectively.

For each test case, print the grid as it appears at the end of the *n*th day. Leave an empty line between the output for successive test cases.

Sample Input

```
2
3 3 1
RRR
RSR
RRR
3 4 2
RSPR
SPRS
PRSP
```

Output for Sample Input

```
RRR
RRR
RRR

RRRS
RRSP
RSPR
```

Problem 1: Breaking a Dollar

Using only the U. S. coins worth 1, 5, 10, 25, 50, and 100 cents, there are exactly 293 ways in which one U. S. dollar can be represented. Canada has no coin with a value of 50 cents, so there are only 243 ways in which one Canadian dollar can be represented. Suppose you are given a new set of denominations for the coins (each of which we will assume represents some integral number of cents less than or equal to 100, but greater than 0). In how many ways could 100 cents be represented?

Input

The input will contain multiple cases. The input for each case will begin with an integer *N* (at least 1, but no more than 10) that indicates the number of unique coin denominations. By *unique* it is meant that there will not be two (or more) different coins with the same value. The value of *N* will be followed by *N* integers giving the denominations of the coins.

Input for the last case will be followed by a single integer –1.

Output

For each case, display the case number (they start with 1 and increase sequentially) and the number of different combinations of those coins that total 100 cents. Separate the output for consecutive cases with a blank line.

Sample Input	Output for the Sample Input

`6 1 5 10 25 50 100`
`5 1 5 10 25 100`
`-1`
	`Case 1: 293 combinations of coins`
	
	`Case 2: 243 combinations of coins`

Problem C: Arctic Network

The Department of National Defence (DND) wishes to connect several northern outposts by a wireless network. Two different communication technologies are to be used in establishing the network: every outpost will have a radio transceiver and some outposts will in addition have a satellite channel.

Any two outposts with a satellite channel can communicate via the satellite, regardless of their location. Otherwise, two outposts can communicate by radio only if the distance between them does not exceed D, which depends of the power of the transceivers. Higher power yields higher D but costs more. Due to purchasing and maintenance considerations, the transceivers at the outposts must be identical; that is, the value of D is the same for every pair of outposts.

Your job is to determine the minimum D required for the transceivers. There must be at least one communication path (direct or indirect) between every pair of outposts.

The first line of input contains N, the number of test cases. The first line of each test case contains 1 <= S <= 100, the number of satellite channels, and S < P <= 500, the number of outposts. P lines follow, giving the (x,y) coordinates of each outpost in km (coordinates are integers between 0 and 10,000). For each case, output should consist of a single line giving the minimum D required to connect the network. Output should be specified to 2 decimal points.

Sample Input

```
1
2 4
0 100
```

```
0 300
0 600
150 750
```

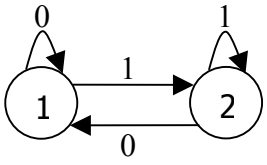
Sample Output

```
212.13
```

Problem 8: Acceptable Strings

A deterministic finite automaton has a finite set of states, with directed edges leading from one state to another. Each edge is labeled with a symbol. In this problem, we are only concerned about automata (the plural of automaton) that use the binary digits 0 and 1 as symbols. Each edge is thus labeled with 0 or 1. One state is identified as the start state, and one or more states are identified as final states.

A finite automaton is usually represented by a graph. For example, consider the finite automaton represented by the graph shown below; the states are shown as circles, and are named 1 and 2 for ease of identification. In this automaton, state 1 is the start state, and state 2 is the final state.



Each automaton in this problem accepts or rejects a string as follows. Beginning in the start state, for each symbol (0 or 1) in the input string (working from left to right in sequence), the automaton follows the one edge labeled with the input symbol from the current state to the next state. After making the transition associated with the last symbol in the input string, if the automaton is in a final state, then the input is accepted. Otherwise (that is, if the automaton is not in a final state), the input is rejected.

For the string 0101 and the automaton shown above, we start in state 1 (the start state). Since the first input symbol is 0, the edge labeled 0 from state 1 back to state 1 is followed, leaving us in state 1. The next input symbol, 1, causes a transition to state 2. The next symbol, 0, moves us back to state 1. The last input symbol, 1, causes the last transition, from state 1 to state 2. Since state 2 is a final state, the automaton accepts the string 0101. Note that the string 010 would have been rejected, since the automaton would have been in state 1 (which is not a final state) at the end of the input. This automaton happens to accept all binary strings that end with 1.

In this problem you will be given one or more automata and an integer *N*. For each of these, you are to find the number of binary strings having each length less than or equal to *N* that are accepted by the automaton. For example, for *N* = 3 with the automaton above, the output would specify 0 strings of length 0 (since state 1 is not a final state), 1 string of length 1 (1), 2 strings of length 2 (01 and 11), and 4 strings of length 3 (001, 011, 101, and 111).

Input

There will be multiple input cases. For each case the input begins with three integers N, S, and F. N (no larger than 10) specifies the maximum length of the strings that are sought. S (no larger than 100) specifies the number of states in the automaton. F (no larger than 10) specifies the number of final states. Following these three integers are S pairs of integers. Each pair specifies the labels on the edges from the states, in order, starting with state 1. The first integer in each pair specifies the state to which the edge labeled 0 connects; the second integer specifies the state to which the edge labeled 1 connects. Finally, the last F integers identify the final states. State 1 will always be the start state. The input for the last case is followed by three zeroes.

Output

For each case, display the case number (they are numbered sequentially, and start with 1). Then display the number of strings of each length (from 0 to N) accepted by the automaton, using a separate line for each length. The output must be identical in format to that shown in the examples below.

Sample Input

```
3 2 1
1 1
1 1
2

3 2 1
1 2
1 2
2

10 7 1
2 2
3 3
4 4
5 5
6 6
7 7
7 7
6

0 0 0
```

Expected Output

```
Case 1:
    Length = 0, 0 strings accepted.
    Length = 1, 0 strings accepted.
    Length = 2, 0 strings accepted.
    Length = 3, 0 strings accepted.
Case 2:
    Length = 0, 0 strings accepted.
    Length = 1, 1 string accepted.
    Length = 2, 2 strings accepted.
    Length = 3, 4 strings accepted.
Case 3:
    Length = 0, 0 strings accepted.
    Length = 1, 0 strings accepted.
    Length = 2, 0 strings accepted.
    Length = 3, 0 strings accepted.
    Length = 4, 0 strings accepted.
    Length = 5, 32 strings accepted.
    Length = 6, 0 strings accepted.
    Length = 7, 0 strings accepted.
    Length = 8, 0 strings accepted.
    Length = 9, 0 strings accepted.
    Length = 10, 0 strings accepted.
```

Problem C: A Walk Through the Forest

Jimmy experiences a lot of stress at work these days, especially since his accident made working difficult. To relax after a hard day, he likes to walk home. To make things even nicer, his office is on one side of a forest, and his house is on the other. A nice walk through the forest, seeing the birds and chipmunks is quite enjoyable.

The forest is beautiful, and Jimmy wants to take a different route everyday. He also wants to get home before dark, so he always takes a path to make progress towards his house. He considers taking a path from A to B to be progress if there exists a route from B to his home that is shorter than any possible route from A. Calculate how many different routes through the forest Jimmy might take.

Input

Input contains several test cases followed by a line containing 0. Jimmy has numbered each intersection or joining of paths starting with 1. His office is numbered 1, and his house is numbered 2. The first line of each test case gives the number of intersections N, $1 < N \leq 1000$, and the number of paths M. The following M lines each contain a pair of intersections a b and an integer distance $1 \leq d \leq 1000000$ indicating a path of length d between intersection a and a different intersection b. Jimmy may walk a path any direction he chooses. There is at most one path between any pair of intersections.

Output

For each test case, output a single integer indicating the number of different routes through the forest. You may assume that this number does not exceed 2147483647.

Sample Input

```
5 6
1 3 2
1 4 2
3 4 3
1 5 12
4 2 34
5 2 24
7 8
1 3 1
1 4 1
3 7 1
7 4 1
7 5 1
6 7 1
5 2 1
6 2 1
0
```

Output for Sample Input

```
2
4
```

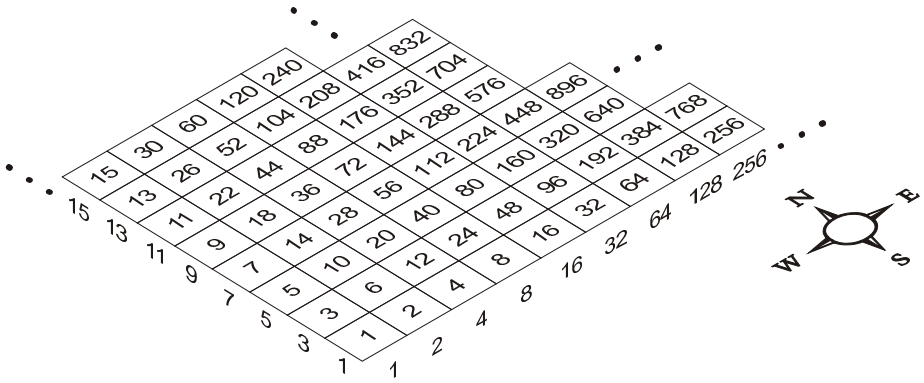
(apologies to) *Richard Krueger*

[C] City of Flatland

Program:	flatland.(c|cpp|java)
Input:	flatland.in
Output:	flatland.out

Description

In recognition to the number of famous mathematicians of its residents, the City of Flatland has decided to rename all its streets as numbers (positive integers to be more precise.) The streets of Flatland are organized as a grid. The city decided to number all its North-South streets using powers of two $(1, 2, 4, 8, \ldots)$ and all its East-West streets using odd numbers $(1, 3, 5, \ldots)$. The city also decided to re-number all its buildings so that the number of each building is the result of multiplying the numbers of the two streets the building is on. For example, building #40 is at the intersection of streets 5 and 8.



The problem with this numbering scheme is that it is not easy for the residents to determine the distance between buildings. The distance between any two buildings is the number of buildings one needs to cross to go from one building to another. One can only move parallel to the streets (no diagonals or any other shortcuts.) For example, to go from building #6 to building #40, one has to travel one building north and two buildings east, so the distance is 3. Similarly, the distance from building #80 to building #88 is 4.

Help the residents of Flatland by writing a program that calculates the distance between any two given buildings.

Input Format

The input is made of one or more pairs of building numbers. Each pair `<S,T>` appears on a single line with a single space between the two numbers. Note that `S,T < 1,000,000,000`. The end of the input is identified by the pair `<0,0>` (which is not part of the test cases.)

Output Format

For each input pair `<S,T>`, the output file should include a line of the form:

`The distance between S and T is D.`

The output file should be in the same order as the input file.

Sample Input/Output

flatland.in

```
12 14
20 30
40 50
0 0
```

flatland.out

```
The distance between 12 and 14 is 3.
The distance between 20 and 30 is 6.
The distance between 40 and 50 is 12.
```