The 2005 ACM ASIA Programming Contest Dhaka Site

Sponsored by IBM
Hosted by North South University
Dhaka, Bangladesh

22nd and 23rd September 2005
You get 14 Pages
8 Problems
Discussion on Problemset*

Problem A

Harmonic Mean

Input: a.in
Output: Standard Output

The harmonic mean (H_N) of N numbers $a_1, a_2, a_3 \ldots a_{n-1}, a_n$ is defined as below:

$$H_N = \frac{N}{\frac{1}{a_1} + \frac{1}{a_2} + \frac{1}{a_3} + \ldots + \frac{1}{a_{n-1}} + \frac{1}{a_n}}$$

So the harmonic mean of four numbers a, b, c, d is defined as

$$H_4 = \frac{4}{\frac{1}{a_1} + \frac{1}{a_2} + \frac{1}{a_3} + \frac{1}{a_4}}$$

In this problem your job is very simple: given **N (0<N<9)** integers you will have to find their harmonic mean.

Input

The first line of the input file contains an integer **S (0<S<501)**, which indicates how many sets of inputs are there. Each of the next S lines contains one set of input. The description of each set is given below:

Each set starts with an integer **N (0<N<9)**, which indicates how many numbers are there in this set. This number is followed by **N** integers $a_1, a_2, a_3, \ldots a_N \quad (0 < a_i < 101)$.

Output

For each set of input produce one line of output. This line contains the serial of output followed by two integers **m** and **n** separated by a front slash. These two numbers actually indicate that the harmonic mean of the given four numbers is $\frac{m}{n}$. You must ensure that **gcd(m, n)=1** or in other words **m** and **n** must be relative prime. The value of **m** and **n** will fit into a **64**-bit signed integer.

Sample Input	Output for Sample Input
2 4 1 2 3 4 4 2 2 3 1	Case 1: 48/25 Case 2: 12/7

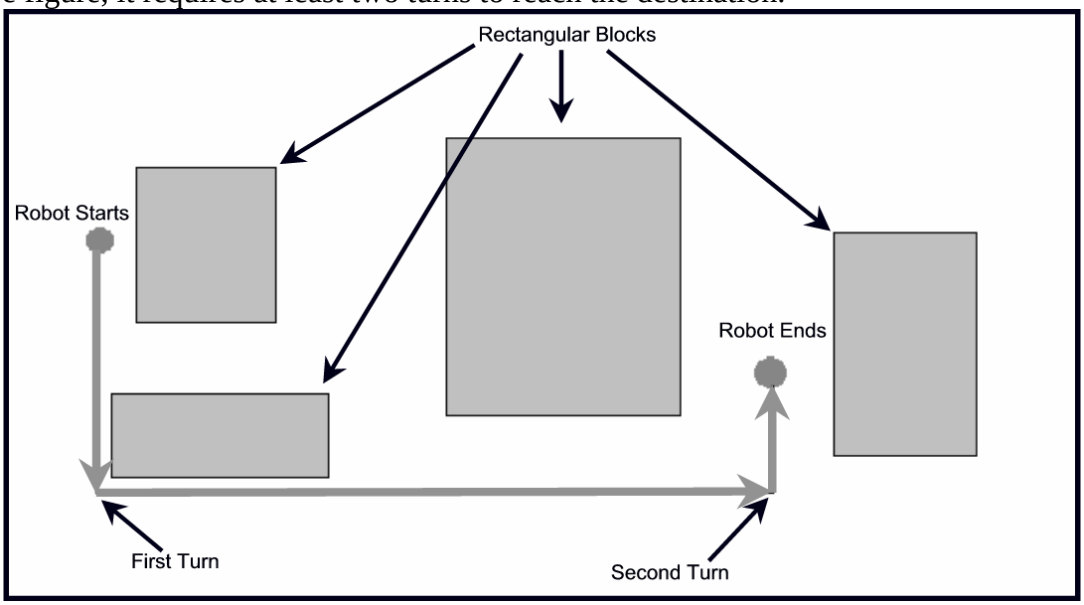
Problem B

Robots inside the Labyrinth

Input File: b.in
Output: Standard Output

Dr. Jemison is simulating the navigation system of a robot. He is using a 2D labyrinth containing several rectangular blocks. All blocks are placed either vertically or horizontally. Spaces are available around the blocks. A robot can use these empty places for its movement. While moving, the robot is not allowed to touch or pass through any block. Also, robot's movement must be either horizontal or vertical. A sample scenario is given in the following picture, where rounded blocks indicate the position of robot –

The main aspect of Jemison's experiment is to test whether a robot can turn timely in the right direction and reach its destination. Jemison has already embedded the complete map of the labyrinth and its final position inside the robot's memory. As turning is costly, Jemison wants the robot to reach its destination using minimum number of turns. For example, in the above figure, it requires at least two turns to reach the destination.



In this problem you will be given a labyrinth of infinite extent. There can be zero or more rectangular blocks inside the labyrinth. Each rectangular block will be defined by its bottom-left (**lx, by**) and top-right (**rx, ty**) corners. Rectangular blocks will not overlap one another but they can share a common border line. The starting and ending positions of the robot will be denoted by Cartesian coordinates. These positions will not touch any block or stay inside it. You have to find the minimum number of turns that the robot must make to reach its destinations from starting positions.

Input

In the first line, there will be an integer, **T** (**1=T=50**) denoting the number of tests. Each input will start with an integer, **N (0=N=50),** where N is the number of rectangular blocks. Following **N** lines will contain the description of a rectangular block. Each line will contain **4** integers **lx**, **by**, **rx (greater than lx)**, **ty (greater than by)**. Next line will contain another

integer **K (1=K=20)** which is the number of queries. Each query will contain starting and ending coordinates of the robot in a line (two positions will be distinct always). The coordinates will be positive integer and will not exceed 10^8.

Two successive input cases will be separated by a blank line.

Output

For each input set, output must start with a line "Labyrinth #**D**", where **D** is the test number starting from **1**. It will be followed by minimum number of turns for each query in a separate line. If the robot somehow cannot reach to its destination, print "**Impossible.**". See sample input output for clarification.

Sample Input	Output for Sample Input
2 0 2 10 10 20 20 10 10 10 20 1 10 10 100 100 2 9 10 101 10 1 1 1000 1000	Labyrinth #1 1 0 Labyrinth #2 2 1

Problem C

Invite Your Friends

Input File: c.in
Output: Standard Output

Rafiq lives in a strange square shaped country **FONO** where each city is equal in size and square in shape and connected to at most four cities. From the top view of the country, it looks like a grid. For simplicity, we assume that each city is recognized by two numbers: the row and column number, starting from **(0, 0)**. Each city is connected by road from those cities which share the same borderline. So from city **(i,j)**, any one can go to city **(i-1,j)** or city **(i+1,j)** or city **(i,j-1)** or **city(i,j+1)** and no one can move more than one city in a day. The rules of the country are very simple: when any one stays in a city, he must pay an amount of money to that city which covers the cost of staying in that city for one day and the cost to reach any one of the neighbor cities where he wants to go. He can stay in a city as many days as he wants but he needs to pay for each day.

4 F1	5	10	20 F2
40	30	40	10
18	53	4	32 F3
52	37	42	43

Fig: Country FONO

Every year the government of **FONO** organizes a lottery and gives a chance to one person and his **F** friends to stay in any city of the country without any cost. This lottery is valid for only **T** days. This year Rafiq has won this lottery and has decided to invite **3** of his friends. He has also decided to bear the cost to reach that city for all of his friends. So he needs to calculate which city is suitable for him to invite – that is minimum amount of money required to reach in that city by his friends. Suppose **F1** lives at City **(0, 0)**. To reach the city **(0, 3)**, he needs to pay **19**. Because after reaching city **(0, 3)**, he need not to pay any money. (Remember that, the lottery is valid for only **T** days. So all of his friends must reach in the selected city within **T** days.)

Input

The input consists of a number of cases (Less than **31**). Each case starts with a line specifying three integer numbers **N**, **F**, **T**. Here **N (0<N=25)** represents the size of the country, **F (0<F=5)** represents the number of friends Rafiq wants to invite, and **T (0<T=25)** represents the number of days within which the lottery is valid. After that, **N x N** positive numbers **(<10000)** are given which represents the cost of each city of the country. After that, there are

F lines. Each line contains **2** numbers – **x** and **y** - representing the current position of Rafiq's friends. Input is terminated by a line where **N=F=T=0**.

Output

For each test case, first print the "**Case #i:**" where **i** is the test case number, and then print "**Impossible.**" if it is not possible to find a city where each one can meet within the specified day. If it is possible, print the position of the city and the cost to reach that city. If more than one solution of minimum cost is possible, select the one with minimum row number. If still more than one solution found, print the one with minimum column number.

Sample Input Output for Sample Input

Sample Input	Output for Sample Input
4 3 3 4 5 10 20 40 30 40 10 18 53 4 32 52 37 42 43 0 0 0 3 2 3 4 3 2 4 5 10 20 40 30 40 10 18 53 4 32 52 37 42 43 0 0 0 3 2 3 0 0 0	Case #1: Selected city (0,3) with minimum cost 61. Case #2: Impossible.

Problem D

Understanding Recursion

Input File: d.in
Output: Standard Output

Understanding recursion is not easy but unfortunately to solve this problem you need to understand it quite well. Below you can see a program written in plain C, which takes as input up to **40000**, **32**- bit integers and produces an output. It continues to do so until a number set of zero elements appear. Given the input your job is to find out what output will the following program will produce.

```
#include<stdio.h>
#include<math.h>
int const MAX=40000;
long nums[MAX];
long recur(int i,int j,int N)
{
    long t1=0,t2=0,t=0;
    if(i<0 || j<0 || i>=N || j>=N) return 0;
    if(i==j) t=recur(i+1,j+1,N);
    if(i<=j) t1=(nums[i]>nums[j])+recur(i,j+1,N);
    if(i>=j) t2=(nums[i]>nums[j])+recur(i,j-1,N);
    return t1+t2+t;
}
int main(void)
{
    long int i,j,N,kase=0;
    freopen("d.in","r",stdin);
    while(1)
    {
        scanf("%d",&N);
        for(i=0;i<N;i++)
            scanf("%ld",&nums[i]);
        if(N==0) break;
        printf("Case %d: %ld\n",++kase,recur(0,0,N));
    }
    return 0;
}
```

Input

The input file contains maximum **10** sets of input. The description of each set is given below:

The first line of each set is an integer **N (0=N=40000)** which indicates how many numbers are in this set. Each of the next **N** lines contains a number. All these numbers are less than **2000000001**.

Input is terminated by a set where the value of **N** is zero.

Output

For the input file produce the output that the program above will produce (Assuming that it will run smoothly in the computer and no stack overflow will occur) for the given input file.

Sample Input	Output for Sample Input
4 1 2 3 4 2 6 1 0	Case 1: 6 Case 2: 1

Problem E

Matrissor

Input File: e.in
Output: Standard Output

Matrissor is a special kind of processor which can multiply a sequence of matrices in quick time. It has certain capacity **K** which means the maximum number of computations (multiplications here) it can perform at one step. For example if **K** is **1000**, then it can multiply **2** matrices of **10x10** dimension. But it cannot multiply a **(10x11)** matrix and another **(11x10)** matrix which require **1100** multiplications. There is a limitation of matrissor. It cannot multiply a sequence of matrices optimally. If it is to multiply **m** matrices, it processes first **(m-1)** matrices first and then multiples the resultant matrix with $\mathbf{m}^{th}$ matrix.

Your task is to multiply a sequence of matrices optimally using the matrissor with capacity **K**. Here optimality depends on one criterion. You have to use the matrissor minimum number of times. Say you have **4** matrices available - $\mathbf{M_1(10x1)}$, $\mathbf{M_2(1x10)}$, $\mathbf{M_3(10x1)}$ and $\mathbf{M_4(1x10)}$. Now if you use a **100** capacity matrissor, then you can multiply $\mathbf{M_2}$, $\mathbf{M_3}$ and $\mathbf{M_4}$ in one step and in last step you can multiply $\mathbf{M_1, (M_2, M_3, M_4)}$. This can be expressed as $\mathbf{(M_1,(M_2,M_3,M_4))}$, where $\mathbf{(M_2, M_3, M_4)}$ denotes the resultant matrix after multiplying $\mathbf{M_2}$, $\mathbf{M_3}$, $\mathbf{M_4}$.

Input

The input file contains the number of test cases **T** first, which is at most **30**. Each test case begins with a positive integer **N (2=N=50)** which is the number of matrices. Following **N** lines contain the dimensions of matrices, one line per matrix. Dimensions will be valid and any dimension will be in between **1** to **50**. Next line will contain another integer **Q(1=Q=N)** which is the number of queries, followed by the capacities of the matrissor in one line. Each test case will be followed by a blank line.

Output

For each set of input, print a line "**Matrix #D**" in first line, where **D** is the test case number starting from 1. In next **Q** lines print the minimum number of steps to multiply all the matrices. If it is not possible to multiply the matrices, then print "**Impossible.**". Put a blank line after each output set. See sample output for details.

Sample Input	Output for Sample Input
2 4 10 1 1 10 10 1 1 10 3 100 99 300 4 1 1	Matrix #1 2 Impossible. 1 Matrix #2 3 2

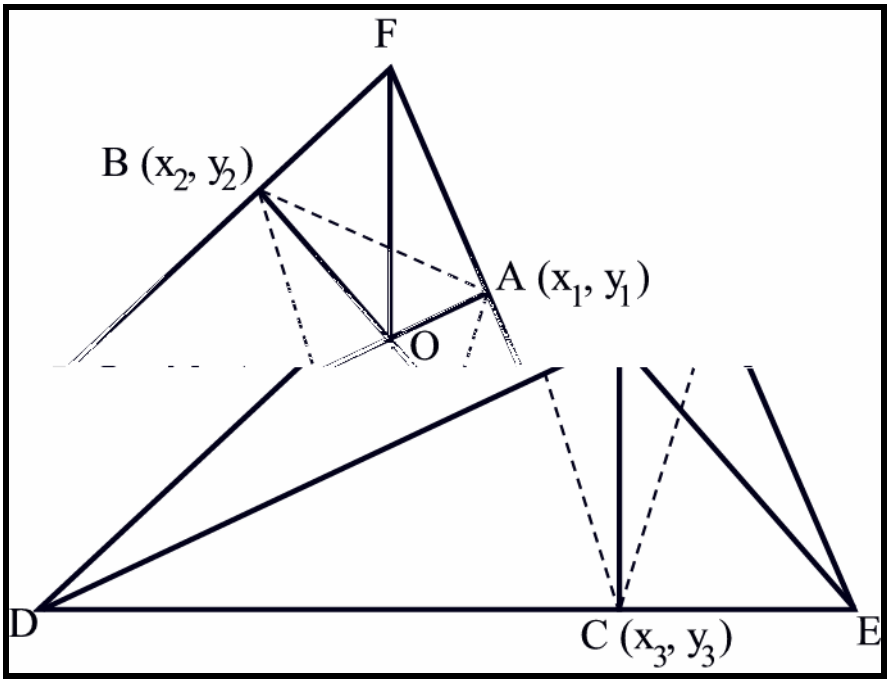
1 1 1 1 1 1 2 1 2	

Problem F

Altitude Triangle

Input File: f.in
Output: Standard Output

If **DEF** is an acute triangle and **DA**, **EB** and **FC** are its three heights on **EF**, **DF** and **DE** respectively, then the triangle **ABC** is called the altitude triangle of triangle **DEF**. It is well known that **DA**, **EB** and **FC** are concurrent and let us assume that their common point of intersection is **O**. So point **O** is called the orthocenter of triangle **DEF**. It can be proved that **O** is the in center of triangle **ABC**. In this problem you will be given the altitude triangle **ABC** and your job is to find out the corresponding acute triangle **DEF**.



Input

The input file contains at most **1001** lines of input. Each line contains six integers x_1, y_1, x_2, y_2 and x_3, y_3. These six integers denote an altitude triangle with vertex **A** (x_1, y_1), **B** (x_2, y_2) and **C** (x_3, y_3) respectively. Input is terminated by a case where all six integers are zero. The points **A**, **B** and **C** will not be collinear.

Output

For each line of input produce four lines of outputs. The description of these four lines is given below:

The first line contains the serial of output. Each of the next three lines contains two floating-point numbers, which are actually the coordinate of **D**, **E** and **F** respectively. Note that for a given altitude triangle **ABC**, there can be four possible triangles **DEF**. But you are requested only to find the one that is acute. Also note that judge data will be such that precision errors should not occur if you use double precision floating-point numbers. Absolute values of none

of the output numbers will be greater than **100000** and all the numbers should have three digits after the decimal point.

Sample Input	Output for Sample Input
682 1369 3981 1233 4333 4583 4131 734 1249 4705 2815 475 2815 475 4131 734 1249 4705 0 0 0 0 0 0	Case 1: 6539.582 3443.107 -1528.155 7610.801 1491.578 -917.367 Case 2: -1810.802 3068.269 3810.093 -82.858 6872.845 7713.274 Case 3: 6872.845 7713.274 -1810.802 3068.269 3810.093 -82.858

Problem G

The Ultimate Bamboo Eater

Input File: g.in
Output: Standard Output

Jingjing the panda lives in a forest containing **n** pieces of bamboo land. Each bamboo land is very small and can be regarded as a single point. Bamboo land **i** contains $\mathbf{L_i}$ bamboos and is associated with a ``deliciousness'' $\mathbf{W_i}$.

Jingjing eats all bamboos in a selected bamboo land every day. He has a bad habit: the deliciousness

Output

For each test case, print the case number followed by the number of days Jingjing can survive. Look at the output for sample input for details.

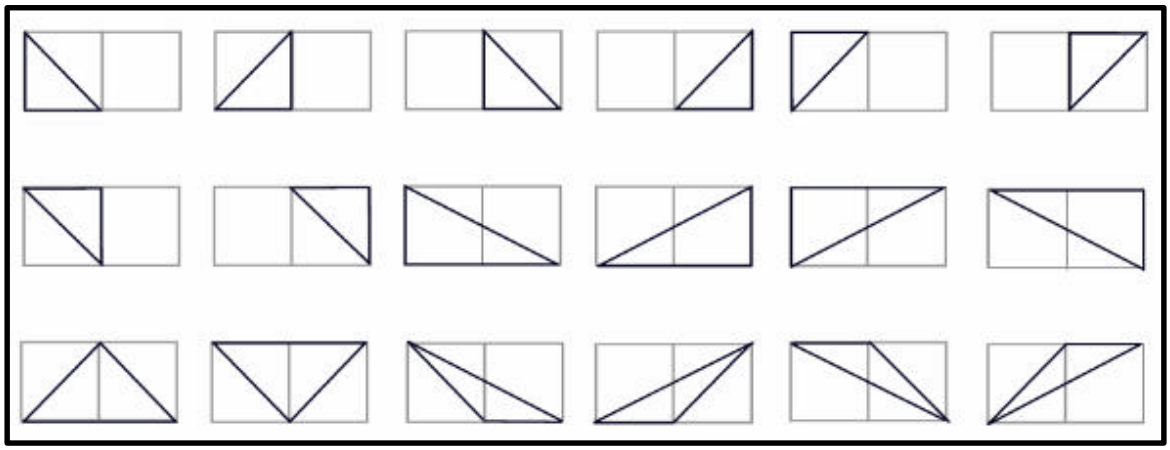
Sample Input	Output for Sample Input
2 3 0 0 3 4 2 2 2 3 5 5 1 3 3 0 0 3 4 2 2 2 3 5 5 1 3	Case 1: 2 Case 2: 2

Problem H

Counting Triangles

Input File: h.in
Output: Standard Output

Triangles are polygons with three sides and strictly positive area. Lattice triangles are the triangles all whose vertexes have integer coordinates. In this problem you have to find the number of lattice triangles in an **M*N** grid. For example in a **(1x2)** grid there are **18** different lattice triangles as shown in the picture below:



Input

The input file contains at most **21** sets of inputs.

Each set of input consists of two integers **M** and **N (0<M,N<=1000)**. These two integers denote that you have to count triangles in an **(MxN)** grid.

Input is terminated by a case where the value of **M** and **N** are zero. This case should not be processed.

Output

For each set of input produce one line of output. This output contains the serial of output followed by the number lattice triangles in the **(MxN)** grid. You can assume that number of triangles will fit in a **64**-bit signed integer.

Sample Input	Output for Sample Input
1 1 1 2 0 0	Case 1: 4 Case 2: 18

ACM ICPC Regional Contest - Dhaka Site
Discussion on Problemset
(Added After the Contest)

Problem A – Harmonic Mean

Type: This is a problem to check whether a person can code simple things in C/C++ or Java. It was expected that all teams will solve it.
Difficulty Level: 1 (Very Easy)
How to solve: Not required.

Problem B – Robots inside the Labirynth

Type: Basically a searching problem with some flavor of geometry.
Difficulty Level: 6 (Medium Hard), if you have knowledge of handling rectangles, otherwise quite difficult
How to solve: You have to form a grid with all possible x-coordinates and y-coordinates. The grid dimension will be at most (2n+2)*(2n+2), if n is the number of rectangles. Now the grids which are a part of given rectangular blocks will be inaccessible. Rest grid locations will be accessible. Apply a BFS from start grid location to the end grid location. Choose all 4 directions initially. Be careful when number of turns is zero.

Problem C – Invite Your Friends

Type: This is basically a shortest path problem and can be solved by Dijkstra shortest path algorithm, BFS or DFS.
Difficulty Level: 4 (Medium Easy).
How to Solve: Not required.

Problem D – Understanding Recursion

Type: Recursion Understanding or guesswork.
Difficulty Level: 3(Easy).
How to Solve: This problem can be solved quickly by someone if he understands recursion. But this problem can also be solved by typing the given program, testing it with different inputs and then some guess works will do fine. Someone just needs to realize quickly that memorization with the given code will not work as the memory required for that is too large.

Problem E – Matrissor

Type: A Dynamic Programming problem
Difficulty Level: 8 (Hard), Difficult as it requires some analysis, why DP approach will work in this case.
How to Solve: Consider you know the optimal answer for N matrices and also know the minimum number of multiplications you will require at last level. You can get a partition of N to (N-k)th matrices. You know the optimal answer for 1 to (N-k+1) matrices and minimum steps to multiply (N-k)th to Nth matrix. Now try to find out, what is the minimum steps if we want to multiply 1 to (N-k+1) matrices and resultant of (N-k)th to Nth matrices. Also remember, how many multiplications will be necessary in the last level of multiplication.

Problem F – Altitude Triangle

Type: Pure coordinate geometric problem
Difficulty level: 5, Easy if you have quite good conception of geometry.
How to solve: The key hint to solve this problem is the following sentence in the problem statement:

"It can be proved that O is the in center of triangle ABC."

So angle DEF is actually formed by the external bisectors of angle A, B and C.

Problem G – The Ultimate Bamboo Eater

Type: Algorithmic (Dynamic programming + 2-D Interval Tree)
Difficulty level: 9 (Very Hard)
How to solve: Construct a directed graph in which nodes and bamboo lands. Connect every node u to all nodes that are can be reached from u directly. Then the problem is to find the longest path in DAG, which is a typical problem solvable via dp(dynamic programming).

let d[u] be the length of longest path from the starting node to u. We compute d[u] in increasing order of deliciousness, with the formula d[u] = max{d[v]}+1, where there is an arc v->u. A straightforward implementation runs in O(n^2), where for each u we check every v to select the maximum.

To speed it up, notice that all nodes that needs to be checked lies in a square, thus we need a data structure to support fast maximum-in-square and insert (we are inserting every node in order of delicousness) quickly. 2D Interval Tree is such a data structure, since both operation runs in O((logn)^2) so the total
time complexity is O(n(logn)^2).

Reference:
Rujia Liu, Liang Huang, Art of Algorithm and Programming Contests, Tsinghua

University Press, 2004. (currently only Chinese edition is available)

Problem H – Counting Triangles

Type: Number theory, combinatorics, dynamic programming (Optional).
Difficulty: 7 (Hard).
How to Solve: This problem is not as easy as it looks. It can be solved in 2 or 3 ways but it has several wrong ways to solve as well. That is why this problem is tricky. The primary judge solution was written based on the idea of counting the number of triangles within a bounding box and then calculating the total result by considering all possible bounding box.