ICPC-2015好题选讲

长春赛区

**Problem A. Too Rich**  
Time limit: 3s   
You are a rich person, and you think your wallet is too heavy and full now. So you want to give me some money  
by buying a lovely pusheen sticker which costs *p* dollars from me. To make your wallet lighter, you decide to pay  
exactly *p* dollars by as many coins and/or banknotes as possible.  
For example, if *p* = 17 and you have two $10 coins, four $5 coins, and eight $1 coins, you will pay it by two $5  
coins and seven $1 coins. But this task is incredibly hard since you are too rich and the sticker is too expensive  
and pusheen is too lovely, please write a program to calculate the best solution.  
**Input**  
The first line contains an integer *T* indicating the total number of test cases. Each test case is a line with 11  
integers *p, c*1*, c*5*, c*10*, c*20*, c*50*, c*100*, c*200*, c*500*, c*1000*, c*2000, specifying the price of the pusheen sticker, and the number  
of coins and banknotes in each denomination. The number *ci* means how many coins/banknotes in denominations  
of *i* dollars in your wallet.  
*•* 1 *≤ T ≤* 20000  
*•* 0 *≤ p ≤* 109  
*•* 0 *≤ ci ≤* 100000  
**Output**  
For each test case, please output the maximum number of coins and/or banknotes he can pay for exactly *p* dollars  
in a line. If you cannot pay for exactly *p* dollars, please simply output ‘-1‘.  
**Sample**  
**standard input**  
3  
17 8 4 2 0 0 0 0 0 0 0  
100 99 0 0 0 0 0 0 0 0 0  
2015 9 8 7 6 5 4 3 2 1 0

**standard output**  
9  
-1  
36

**Problem C. Play a game**  
Time limit: 2s   
Andy and Andrew are very smart guys and they like to play all kinds of games in their spare time. The most  
amazing thing is that they always find the best strategy, and that’s why they feel bored again and again. They  
just invented a new game, as they usually did.  
At the beginning of the game, they write down one string *S* = *s*1*s*2*s*3 *. . . sk*, and then they take turns(Andy first)  
to either:  
*•* 1. Erase the leftmost character from *S*, that is, *S* = *s*2*s*3*s*4 *. . . sk*.  
*•* 2. Erase the rightmost character from *S*, that is, *S* = *s*1*s*2*s*3 *. . . sk−*1.  
Whenever *S* is empty or *S ∈ A* (*A* is a given list of strings), the player which plays next loses the game.  
For example, let *S* = *dzxx* and *A* = *{z, dz}*. If Andy erases ’x’ then Andrew can erase another ’x’, because *S* = *dz*  
and *dz* is in *A*, Andy, the next player, loses. Otherwise, Andy erases ’d’, and then Andrew can erase ’z’ result in a  
losing position for Andy.  
You are given a string *T* = *t*1*t*2*t*3 *. . . tn* and a list of string *A* = *{a*1*, a*2*, . . . , am}*. Your task is to find who is the  
winner if *S* is some substring of *T*. Andy and Andrew play so many times so you need to answer multiple queries.  
**Input**  
The first line contains an integer *t* indicating the total number of test cases. The following lines describe a test  
case.  
The first line of each case contains three integers *n*, *m*, *q*, the length of *T*, the size of *A*, and the number of queries.  
The second line contains a string, representing *T*. Next *m* lines, each line consists of a string, representing *ai*.  
Next *q* lines, each line consists of two integers *l, r*, representing a query that you should output who is the winner  
if *S* = *tltl*+1 *. . . tr*.  
*•* 1 *≤ t ≤* 21  
*•* 1 *≤ n, q ≤* 40000  
*•* 1 *≤ m ≤* 10000  
*•*

*•* 1 *≤ l ≤ r ≤ n*  
*• T* and strings in *A* consist of lowercase English letters.  
*•* There are at most 6 test cases with *n >* 5000.  
**Output**  
For each query, if Andy wins, print ”1” (without the quotes) on a single line, otherwise print ”0” (without the  
quotes) on a single line.

**Sample**  
**standard input**  
1  
10 4 10  
zzzabcdzxx  
a

z  
dz  
abcd  
1 3  
1 4  
3 6  
3 7  
3 8  
3 9  
4 4  
4 5  
5 5  
7 10

**standard output**  
0

1

1

1

0

1

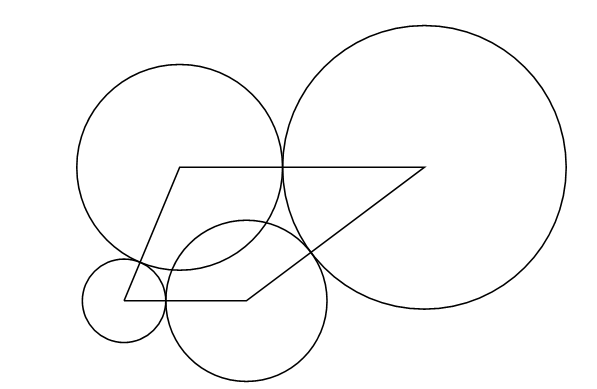
0

1

1

0

**Problem E. Rebuild**  
Time limit: 1s   
Archaeologists find ruins of Ancient ACM Civilization, and they want to rebuild it.  
The ruins form a closed path on an x-y plane, which has *n* endpoints. The endpoints locate on (*x*1*, y*1), (*x*2*, y*2),  
*. . . ,* (*xn, yn*) respectively. Endpoint *i* and endpoint *i −* 1 are adjacent for 1 *< i ≤ n*, also endpoint 1 and endpoint  
*n* are adjacent. Distances between any two adjacent endpoints are positive integers.  
To rebuild, they need to build one cylindrical pillar at each endpoint, the radius of the pillar of endpoint *i* is *ri*.  
All the pillars perpendicular to the x-y plane, and the corresponding endpoint is on the centerline of it. We call  
two pillars are adjacent if and only if two corresponding endpoints are adjacent. For any two adjacent pillars, one  
must be tangent externally to another, otherwise it will violate the aesthetics of Ancient ACM Civilization. If two  
pillars are not adjacent, then there are no constraints, even if they overlap each other.  
Note that *ri* must not be less than 0 since we cannot build a pillar with negative radius and pillars with zero radius  
are acceptable since those kind of pillars still exist in their neighbors.  
You are given the coordinates of *n* endpoints. Your task is to find *r*1*, r*2*, . . . , rn* which makes sum of base area of  
all pillars as minimum as possible.

  
Example

For example, if the endpoints are at (0*,* 0), (11*,* 0), (27*,* 12), (5*,* 12), we can choose (*r*1, *r*2, *r*3, *r*4)=(3*.*75, 7*.*25, 12*.*75,  
9*.*25). The sum of base area equals to 3*.*752*π*+7*.*252*π* + 12*.*752*π* + 9*.*252*π* = 988*.*816 *. . .*. Note that we count the  
area of the overlapping parts multiple times.  
If there are several possible to produce the minimum sum of base area, you may output any of them.  
**Input**  
The first line contains an integer *t* indicating the total number of test cases. The following lines describe a test  
case.  
The first line of each case contains one positive integer *n*, the size of the closed path. Next *n* lines, each line consists  
of two integers (*xi, yi*) indicate the coordinate of the *i*-th endpoint.  
*•* 1 *≤ t ≤* 100  
*•* 3 *≤ n ≤* 104  
*• |xi|, |yi| ≤* 104  
*•* Distances between any two adjacent endpoints are positive integers.

**Output**  
If such answer doesn’t exist, then print on a single line ”IMPOSSIBLE” (without the quotes). Otherwise, in the  
first line print the minimum sum of base area, and then print *n* lines, the *i*-th of them should contain a number *ri*,  
rounded to 2 digits after the decimal point.  
If there are several possible ways to produce the minimum sum of base area, you may output any of them.  
**Sample**  
**standard input**   
3

4  
0 0  
11 0  
27 12  
5 12  
5  
0 0  
7 0  
7 3  
3 6  
0 6  
5  
0 0  
1 0  
6 12  
3 16  
0 12

**standard output**  
988.82  
3.75  
7.25  
12.75  
9.25  
157.08  
6.00  
1.00  
2.00  
3.00  
0.00  
IMPOSSIBLE

**Problem M. Security Corporation**  
Time limit: 20s   
Due to the high crime rate of Bytecity, the new mayor has decided to employ security agents from *c* different  
security corporations. As a first step, he’ll arrange a security agent from any of the corporation at each intersection  
of two roads in Bytecity.  
The Bytecity has a very special and complex transportation system. There are *n* straight and long roads in the  
city, each of them can be treated as an **infinitely long line** on a two dimensional plane, and no 3 or more roads  
have a common intersection.  
In order to prevent agents from slacking during work, the mayor has decided to arrange agents from two different  
security corporations at any two *adjacent intersections*. We said that two intersections are *adjacent* if they are on a  
same road, and there is no other intersection between these two intersections on that road. Also, the mayor thinks  
less number of security corporations is more manageable. Can you help the mayor to distribute all the intersections  
to as less security corporations as possible?  
**Input**  
The first line of input contains an integer *T* indicating the total number of test cases. The first line of each test  
case is an integer *n*, indicating the number of roads in Bytecity. The *n* lines that follow describes roads in Bytecity,  
the *i*-th of these lines contains 4 integers *x*1*i, y*1*i, x*2*i, y*2*i*, indicating road *i* is a straight line through (*x*1*i, y*1*i*),  
(*x*2*i, y*2*i*) on the plane.  
*•* 1 *≤ T ≤* 1000.  
*•* 2 *≤ n ≤* 1000.  
*• −*103 *≤ x*1*i, y*1*i, x*2*i, y*2*i ≤* 103.  
*•* (*x*1*i, y*1*i*) *≠* (*x*2*i, y*2*i*).  
*•* There are at most 10 test cases with *n >* 100.  
*•* There are no 2 identical roads, and at least 1 intersection exists in each test case.  
**Output**  
For each test case, please output an integer *c* in the first line indicating the minimum number of different  
corporations in your arrangement. Following *n −* 1 lines, the *i*-th line of these lines should contain integers  
*ai,i*+1*, ai,i*+2*, ..., ai,n* in one line. The number *ai,j* indicates the security corporation that manages the intersection  
of road *i* and road *j*, if the intersection does not exist, *ai,j* should be *−*1.  
Note that all the corporations should be numbered as an integer from 1 to *c*, and any arrangement that satisfy the  
mayor’s requirement would be accepted.  
**Sample**  
**standard input**   
2

3  
0 0 1 0  
0 0 0 1  
1 0 0 1  
3  
0 0 1 0  
0 1 0 2  
1 0 1 1

**standard output**  
3  
1 2  
3

2  
2 1  
-1

上海赛区

**Problem D. Discover Water Tank**  
Time limit: 12.5s   
A lot of frogs are living in a water tank, but none of them know exactly how much water are there.

The water tank has an infinite height, but with a narrow bottom. The length of the tank is N, and the width is only 1.

Now N−1 boards has divided the tank into N parts, each with an 1×1 bottom. Boards may have different heights. Water cannot flow through the boards, but can run freely if the water level is higher, following the basic physical rules.

The Frog King wants to know more details of the tank, so he sent someone to choose M positions and find whether there's water at that position.

For example, each time he'll choose(x,y), checking the xth part of the tank(1≤x≤N), counting from left to right, and find whether there's water at height (y+0.5).

Now the King gets M results, but he finds some of them might be wrong. The King wants to find out the maximum possible number of the correct results.  
**Input**  
First line contains an integer T, which indicates the number of test cases.

Every test case begins with two integers N and M, which is the numbers of tanks and numbers of results.

The second line of each test case contains N−1 integers, h1, h2, ⋯, hN−1, and hi indicates the height of ith board's height.

Then M lines follow, the ith line, formated as 'x y z', indicates ith result. z is 0 if there is no water at height (y+0.5) in xth tank, otherwise z is 1.

⋅ 1≤T≤100.

⋅ For 90% data, 1≤N≤1000 and 1≤M≤2000

⋅ for 100% data, 1≤N≤105 and 1≤M≤2⋅105.

⋅ 1≤hi≤109 for all 1≤i≤N−1.

⋅ for every result, 1≤x≤N, 1≤y≤109 and z is either 0 or 1.  
**Output**  
For every test case, you should output "Case #x: y",where x indicates the case number and counts from 1, and y is the maximum possible number of the correct results.  
**Sample**  
**standard input**   
2

3 4

3 4

1 3 1

2 1 0

2 2 0

3 3 1

2 2

2

1 2 0

1 2 1

**standard output**  
Case #1: 3

Case #2: 1

**Problem E. Expection of String**  
Time limit: 3s   
Frog has just learned how to multiply two numbers. Now he wants to do some exercise.

He wrote a string on the paper, which only contains digits and a single × as the operator. If the × appears at the front or the end of the string, he regards the result as zero, otherwise he does the calculation as a normal multiplication.

After some play, he wonders a new problem: for a initial string, each time he randomly choose two characters and swap their positions. He will do this again and again, say for K times, he wants to know the expected calculation result for the newest string that he gets.

It can be shown that their can be ways for the whole swap operations, so if the expected result is x, you need to output x×as an integer.  
**Input**  
First line contains an integer T, which indicates the number of test cases.

Every test case begins with an integers K, which is the numbers of times the Frog can swap characters.

The second line of each test case contains the string Frog plays with, which only contains digits and exactly one multiplication operator, written as '∗'.

⋅ 1≤T≤100.

⋅ the string's length is L.

⋅ for 70% data, 1≤L≤10 and 0≤K≤5.

⋅ for 95% data, 1≤L≤20 and 0≤K≤20.

⋅ for 100% data, 1≤L≤50 and 0≤K≤50.  
**Output**  
For every test case, you should output "Case #x: y", where x indicates the case number and counts from 1 and y is the result.

Because y could be very large, just mod it with 109+7.  
**Sample**  
**standard input**   
2

1

1\*2

2

1\*2

**standard output**  
Case #1: 2

Case #2: 6

**Problem I. Infinity Point Sets**  
Time limit: 5s   
This story comes from an ancient book written by an old frog philosopher.

When will be the end of the world? Perhaps the best way to understand is using geometry for a calculation.

At first, you should draw some points on a piece of paper. Each time you should choose two points and connect them with a segment. When there are two segments meet and produce a new point at the cross, you add that point to the paper, and try to connect it with previous ones as before.

You should do this again and again, keep drawing segments, and adding points if possible, until there are no new segments to draw. Then is the end of the world, the old frogs will die and a new age will begin.

As you can see, different sets of initial points lead to different results. For some sets of points, the end of the world will never come, we call those sets Infinity.

Now you are given N points, and you need to find, among all the possible subsets of those N points not including empty set, so there will be 2N−1 sets in total), how many of them are NOT Infinity?  
**Input**  
First line contains an integer T, which indicates the number of test cases.

Every test case begins with an integers N, which indicates the numbers of points.

In the following N lines, the ith line contains two intergers xi and yi, indicating the coordinate of ith point is (xi,yi).

⋅ 1≤T≤10.

⋅ for 90% data, 1≤N≤100.

⋅ for 100% data, 1≤N≤1000.

⋅ 1≤xi,yi≤104.

⋅ there is no a pair of points with the same coordinate.  
**Output**  
For every test case, you should output `"Case #x: y", where x indicates the case number and counts from 1 and y is the result.

Because y could be very large, just mod it with 109+7.  
**Sample**  
**standard input**   
2

4

0 0

0 2

2 2

2 0

5

0 0

0 2

2 2

2 0

1 2

**standard output**  
Case #1: 15

Case #2: 30

沈阳赛区

**Problem E. Efficient Tree**  
Time limit: 6s   
A graph G with n×m nodes forms a n×m grid with n×m vertices.

(n−1)×m weighted edges connect the vertices of adjacent rows, and n×(m−1) weighted edges connect the vertices of adjacent columns.

A spanning tree of graph G is a subgraph that is a tree and connects all the vertices together.

A minimum spanning tree (MST) or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree.

Graph G has many different minimum spanning trees.

For each MST *T*, the *LRdeg(u)* of node u is defined as the number of nodes, in the previous column or the previous row connecting with u, plus one.

And we define *τ(T)=∏uLRdeg(u)* as the product of *LRdeg(u)* for all nodes.

Your mission is to find the weight of the minimum spanning tree of graph G, and count *τ(T)* of all minimum spanning trees. Two *MST(s)* are considered different if they contain different subsets of edges.  
**Input**  
The input contains several test cases. The first line of the input is a single integer t (1≤t≤32) which is the number of test cases. Then t test cases follow.

For each test case, the first line contains the two integers n (1≤n≤800) and m (1≤m≤7).

Each line of the next n lines contains m−1 integers, which describe the weights of edges connecting the vertices of adjacent columns.

And each line of the next n−1 lines contains m integers, which describe the weights of edges connecting the vertices of adjacent rows. The weights of edges are no more than 10.  
**Output**  
For each test case, you should output two integers in one line. The first one is the weight of the minimum spanning tree.

The second one is the sum of τ(T) for all different minimum spanning trees, modulo 109+7.  
**Sample**  
**standard input**   
2

2 5

9 8 5 6

4 6 2 3

1 7 8 3 8

5 5

8 10 5 4

1 7 7 7

5 4 5 5

3 2 2 2

8 7 8 3

8 5 7 8 6

10 3 2 4 3

8 7 2 8 9

9 4 8 3 9

**standard output**  
Case #1: 37 288

Case #2: 96 4478976

CCPC

