A. Biconnected

[Problem Description]

People are weak. Relationship between people like friendship or love is weak too. But a group of persons can have strong relationship, umm, 2-edge-connected relationship.

Suppose there are *n* persons. If two persons, A and B, are in a relationship, then we add an un-directional edge between them. In this way we can have a relationship graph, which is an un-directional graph without self-loops or multiple edges. If this graph is 2-edge-connected, then we say these persons have a strong relationship.

Now we have a group of persons without relationship between any two of them. And some pair of persons even hate each other. You will introduce some pairs of persons to know each other and set up a relationship between them to make the group of persons have a strong relationship. But notice that you can't set up a relationship between a pair of persons who hate each other. How many ways you can do that? (Two ways are different if there exist a pair of persons which have relationship in one way but not in another way).

[Input Format]

The first line contains an integer *T*, denoting the number of the test cases.

For each test case, the first line contains 2 integers *n* and *m*, denoting the number of persons in the group and the number of pairs of persons who hate each other. Then *m* lines follow, each line containing 2 integers A and B, denoting that A and B hate each other.

$T<=5$, $2<=n<=10$, $0<=m<=n*(n-1)/2$. The persons are indexed from 1.

[Output Format]

For each test case, output the answer in a line.

[Sample Input]

```
3
5 0
10 0
5 2
1 2
2 3
```

[Sample Output]

```
253
466997276
18
```

[Hint]

A 2-edge-connected graph is a graph which is connected and if you remove an edge from it, it is still connected.

Note that n>=2, so we can ignore the issue that whether a single vertex is 2-edge-connected or not :).

B. Rotate

[Problem Description]
Noting is more interesting than rotation!

Your little sister likes to rotate things. To put it easier to analyze, your sister makes n rotations. In the i-th time, she makes everything in the plane rotate counter-clockwisely around a point a_i by a radian of p_i.

Now she promises that the total effect of her rotations is a single rotation around a point A by radian P (this means the sum of p_i is not a multiplier of 2π).

Of course, you should be able to figure out what is A and P :).

[Input Format]
The first line contains an integer T, denoting the number of the test cases.

For each test case, the first line contains an integer n denoting the number of the rotations. Then n lines follows, each containing 3 real numbers x, y and p, which means rotating around point (x, y) counter-clockwisely by a radian of p.

We promise that the sum of all p's is differed at least 0.1 from the nearest multiplier of 2π.

$T<=100$. $1<=n<=10$. $0<=x, y<=100$. $0<=p<=2\pi$.

[Output Format]
For each test case, print 3 real numbers x, y, p, indicating that the overall rotation is around (x, y) counter-clockwisely by a radian of p. Note that you should print p where $0<=p<2\pi$.

Your answer will be considered correct if and only if for x, y and p, the absolute error is no larger than 1e-5.

[Sample Input]
```
1
3
0 0 1
1 1 1
2 2 1
```

[Sample Output]
```
1.8088715944 0.1911284056 3.0000000000
```

C. Overt

[Problem Description]

Carelessly designed cryptographic primitives leave your secret as bared as plain text. It is not a surprise that seemingly "random" hash functions are weak. Consider the function of the following form:

```
unsigned int hash(unsigned int x) {
    x += 0x327b7473u;
    x &= 0xffffafffu;
    x ^= 0x90283712u;
    x |= 0x00300000u;
    x += 0x89129723u;
    x ^= 0x464726ccu;
    x &= 0xfffff8ffu;
    //......
    return x;
}
```

The function maps an integer to another integer and intends to make the result random. All statements are of the form: x (some operator) (some number). Possible operators are: add (+=), subtract (-=), bitwise-and (&=), bitwise-xor (^=), and bitwise-or (|=). Due to the nature of fixed size integer, there is an implicit modulo 4294967296 operation after each statement.

However, it is a rather weak hash function from a cryptographic point of view. To demonstrate its weakness, you are requested to find an input *x* that maximizes the output.

In this example the best *x* is 1841992591 and the maximum output is 4292342015.

[Input Format]

The first line contains an integer *T*, denoting the number of the test cases.

Each test case begins with a non-negative integer *N*, the number of operations in the hash function. 0<=*N*<=40.

Then follows *N* lines, each describing an operation. Each line contains an operator and an 8-digit hexadecimal number.

[Output Format]

For each test case, output the maximum output in decimal format.

[Sample Input]

```
2
7
+= 327b7473
&= ffffafff
^= 90283712
|= 00300000
+= 89129723
^= 464726cc
&= fffff8ff
1
-= 00000001
```

[Sample Output]

```
4292342015
4294967295
```

D. Clone

[Problem Description]

After eating food from Chernobyl, DRD got a super power: he could clone himself right now! He used this power for several times. He found out that this power was not as perfect as he wanted. For example, some of the cloned objects were tall, while some were short; some of them were fat, and some were thin.

More evidence showed that for two clones A and B, if A was no worse than B in all fields, then B could not survive. More specifically, DRD used a vector v to represent each of his clones. The vector v has n dimensions, representing a clone having N abilities. For the i-th dimension, $v[i]$ is an integer between 0 and $T[i]$, where 0 is the worst and $T[i]$ is the best. For two clones A and B, whose corresponding vectors were p and q, if for $1 <= i <= N$, $p[i] >= q[i]$, then B could not survive.

Now, as DRD's friend, ATM wants to know how many clones can survive at most.

[Input Format]

The first line contains an integer T, denoting the number of the test cases.

For each test case: The first line contains 1 integer N, $1 <= N <= 2000$. The second line contains N integers indicating $T[1]$, $T[2]$, ..., $T[N]$. It guarantees that the sum of $T[i]$ in each test case is no more than 2000 and $1 <= T[i]$.

[Output Format]

For each test case, output an integer representing the answer MOD 10^9 + 7.

[Sample Input]

```
2
1
5
2
8 6
```

[Sample Output]

```
1
7
```

E. Walk

[Problem Description]

I used to think I could be anything, but now I know that I couldn't do anything. So I started traveling.

The nation looks like a connected bidirectional graph, and I am randomly walking on it. It means when I am at node *i*, I will travel to an adjacent node with the same probability in the next step. I will pick up the start node randomly (each node in the graph has the same probability.), and travel for *d* steps, noting that I may go through some nodes multiple times.

If I miss some sights at a node, it will make me unhappy. So I wonder for each node, what is the probability that my path doesn't contain it.

[Input Format]

The first line contains an integer *T*, denoting the number of the test cases.

For each test case, the first line contains 3 integers *n*, *m* and *d,* denoting the number of vertices, the number of edges and the number of steps respectively. Then *m* lines follows, each containing two integers *a* and *b*, denoting there is an edge between node *a* and node *b*.

T<=20, *n*<=50, *n*-1<=*m*<=*n**(*n*-1)/2, 1<=*d*<=10000. There is no self-loops or multiple edges in the graph, and the graph is connected. The nodes are indexed from 1.

[Output Format]

For each test cases, output *n* lines, the *i*-th line containing the desired probability for the *i*-th node.

Your answer will be accepted if its absolute error doesn't exceed 1e-5.

[Sample Input]

```
2
5 10 100
1 2
2 3
3 4
4 5
1 5
2 4
3 5
2 5
1 4
1 3
10 10 10
1 2
2 3
3 4
```

```
4 5
5 6
6 7
7 8
8 9
9 10
4 9
```

[Sample Output]

```
0.0000000000
0.0000000000
0.0000000000
0.0000000000
0.0000000000
0.6993317967
0.5864284952
0.4440860821
0.2275896991
0.4294074591
0.4851048742
0.4896018842
0.4525044250
0.3406567483
0.6421630037
```

F. Tree

[Problem Description]

You are given a tree with N nodes which are numbered by integers 1..N. Each node is associated with an integer as the weight.

Your task is to deal with M operations of 4 types:

1. Delete an edge (x, y) from the tree, and then add a new edge (a, b). We ensure that it still constitutes a tree after adding the new edge.
2. Given two nodes a and b in the tree, change the weights of all the nodes on the path connecting node a and b (including node a and b) to a particular value x.
3. Given two nodes a and b in the tree, increase the weights of all the nodes on the path connecting node a and b (including node a and b) by a particular value d.
4. Given two nodes a and b in the tree, compute the second largest weight on the path connecting node a and b (including node a and b), and the number of times this weight occurs on the path. Note that here we need the strict second largest weight. For instance, the strict second largest weight of {3, 5, 2, 5, 3} is 3.

[Input Format]

The first line contains an integer T (T<=3), which means there are T test cases in the input.

For each test case, the first line contains two integers N and M (N, M<=10^5). The second line contains N integers, and the i-th integer is the weight of the i-th node in the tree (their absolute values are not larger than 10^4).

In next N-1 lines, there are two integers a and b (1<=a, b<=N), which means there exists an edge connecting node a and b.

The next M lines describe the operations you have to deal with. In each line the first integer is c (1<=c<=4), which indicates the type of operation.
If c = 1, there are four integers x, y, a, b (1<= x, y, a, b <=N) after c.
If c = 2, there are three integers a, b, x (1<= a, b<=N, |x|<=10^4) after c.
If c = 3, there are three integers a, b, d (1<= a, b<=N, |d|<=10^4) after c.
If c = 4 (it is a query operation), there are two integers a, b (1<= a, b<=N) after c.
All these parameters have the same meaning as described in problem description.

[Output Format]

For each test case, first output "Case #x:"" (x means case ID) in a separate line.

For each query operation, output two values: the second largest weight and the number of times it occurs. If the weights of nodes on that path are all the same, just output "ALL SAME" (without quotes).

[Sample Input]

```
2
3 2
1 1 2
1 2
1 3
4 1 2
4 2 3
7 7
5 3 2 1 7 3 6
1 2
1 3
3 4
3 5
4 6
4 7
4 2 6
3 4 5 -1
4 5 7
1 3 4 2 4
4 3 6
2 3 6 5
4 3 6
```

[Sample Output]

```
Case #1:
ALL SAME
1 2
Case #2:
3 2
1 1
3 2
ALL SAME
```

G. Osu!

[Problem Description]

Osu! is a famous music game that attracts a lot of people. In osu!, there is a performance scoring system, which evaluates your performance. Each song you have played will have a score. And the system will sort all you scores in descending order. After that, the i-th song scored a_i will add 0.95^(i-1)*a_i to your total score.

Now you are given the task to write a calculator for this system.

[Input Format]

The first line contains an integer T, denoting the number of the test cases.

For each test case, the first line contains an integer n, denoting the number of songs you have played. The second line contains n integers a_1, a_2, ..., a_n separated by a single space, denoting the score of each song.

T<=20, n<=50, 1<=a_i<=500.

[Output Format]

For each test case, output one line for the answer.

Your answers will be considered correct if its absolute error is smaller than 1e-5.

[Sample Input]

```
1
2
530 478
```

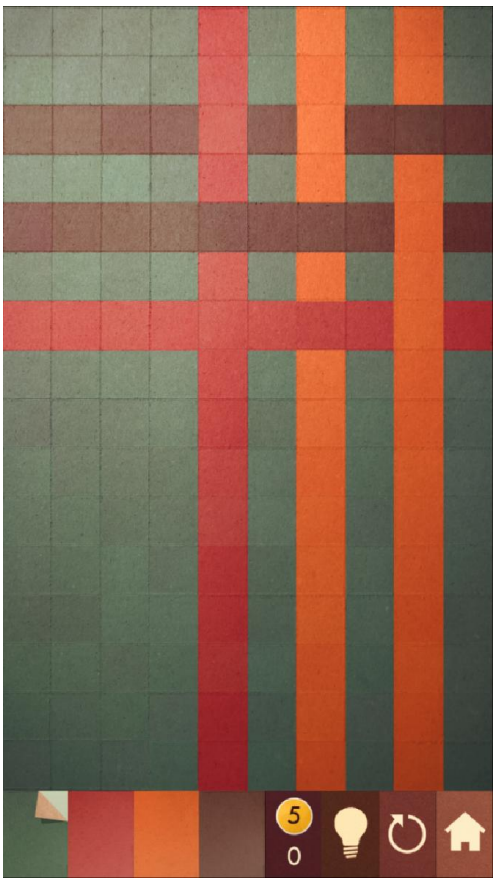
[Sample Output]

```
984.1000000000
```

H. KAMI

[Problem Description]

Little Bob found an interesting mobile game called KAMI in Google Play recently. In this game, you are given a 16*10 grid, and each cell is one of the 4 different colors.

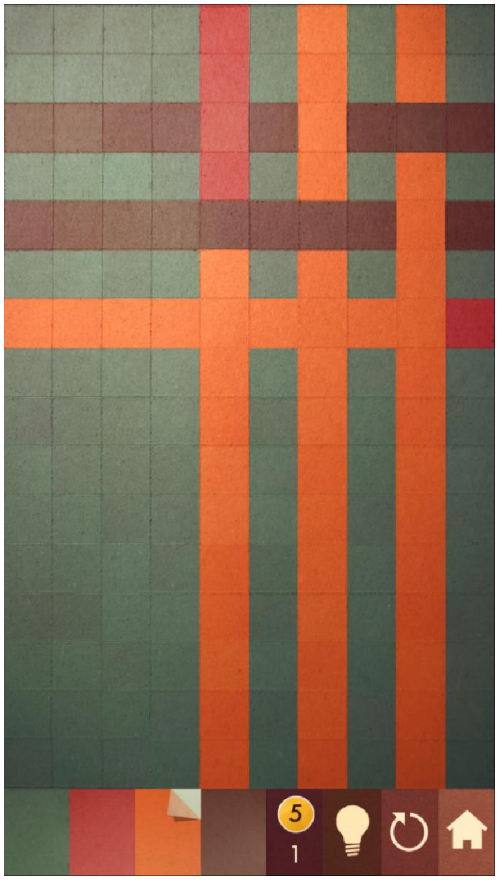


The only thing you can do in this game is to choose a cell and a color, and then the entire 4-connected component, which contains the cell and has the same color as the cell, will be painted with the chosen color.

Your goal is to make all cells painted with the same color with the least painting steps. Also note that the optimal painting steps will be given, just the same as the original game.

For your convenience, we label the colors with integers 1..4. In the grid, we index the rows with integers 1..16 from top to bottom, and index the columns with integers 1..10 from left to right. Thus the cell in the i-th row and j-th column will be denoted by (i, j).

Once you choose to paint cell (7, 1) with orange color in the above picture, we will obtain:



[Input Format]

The first line contains an integer T (T<=5), which means there are T test cases in the input.

For each test case, the first line contains a positive integer n (n<=8), which denotes the optimal painting steps of this case. We ensure this is the least painting times to achieve the goal.

The next 16 lines in each test case describe the initial colors of each cell in the grid. Each line contains a string of length 10, which consists of digits '1'~'4'. For each test case, we ensure that the 4 colors all appear in the initial state.

[Output Format]

For each test case, output "Case #x:" (x means the case ID) in a separate line first. Then you should output exactly n lines. Each line contains three integers c, x, y (1<=c<=4, 1<=x<=16, 1<=y<=10), which means you choose to paint cell (x, y) with color c in this step.

If there exists more than one optimal solutions, you can output an arbitrary one.

[Sample Input]

```
2
3
4444444444
3332222333
3111111113
3332222313
```

```
3111111113
3132222333
3111111113
3332222313
3111111113
3132222333
3111111113
3332222313
3111111113
3132222333
3332222333
4444444444
5
1111213131
1111213131
4444243444
1111213131
4444444434
1111213131
2222222232
1111213131
1111213131
1111213131
1111213131
1111213131
1111213131
1111213131
1111213131
1111213131
```

[Sample Output]

```
Case #1:
2 14 2
3 15 7
4 12 8
Case #2:
4 4 5
4 16 9
2 5 10
1 7 10
3 16 10
```

I. Compromise

[Problem Description]

Xiaoqiang and Amoeba (AMB for short) are good friends. Friends as they are, they are inevitably self-centered beings. So their life together is full of concession and compromise. To further promote their ongoing relationship, Xiaoqiang studies his interaction with AMB from a game theoretical perspective. He views his everyday life with AMB as a game.

The game has *N* "nodes" corresponding to different "states" of life. Each node has several outgoing "edges" representing "actions". A node either belongs to Xiaoqiang or AMB. The nodes and edges form a connected DAG (Directed Acyclic Graph, a graph without cycles). Each node without outgoing edges has two "utility" values for Xiaoqiang and AMB respectively.

For example, consider the following game:

- Node S: The day begins. AMB has two actions leading to A and B respectively.
- Node A: AMB goes to University P. Xiaoqiang has two actions leading to C and D respectively.
- Node B: AMB goes to University T. Xiaoqiang has two actions leading to D and E respectively.
- Node C: Xiaoqiang and AMB both go to University P. AMB's utility is 102, Xiaoqiang's utility is 99.
- Node D: Xiaoqiang and AMB go to different universities. AMB's utility is 51, Xiaoqiang's utility is 0.
- Node E: Xiaoqiang and AMB both go to University T. AMB's utility is 100, Xiaoqiang's utility is 101.

Life starts from Node S. At each node, either Xiaoqiang or AMB can take an action. Xiaoqiang's aim is to maximize his own utility (and cares nothing about AMB) and AMB's aim is also to maximize his own utility.

AMB thinks, if he goes to node A then Xiaoqiang's best response will be going to node C and he can happily live in University P with Xiaoqiang. In this case AMB's utility is 102 and Xiaoqiang's is 99. Xiaoqiang is not satisfied with this, but it seems there is nothing Xiaoqiang can do about this. This is the so called Nash Equilibrium.

Until one day Xiaoqiang claims (or, to be exact, swears): "Listen, I was attacked by a paramecium and I became insane. If I am in Node B, I will surely go to Node E. But if I am in Node A, I will go to Node D without hesitation." In this way, he reveals his whole strategy to AMB, which encodes for each node what Xiaoqiang will do if he arrives at the node. Notice that in this strategy Xiaoqiang's action (going to University T) at a node does not depend on which path is used to arrive at the node. This strategy is not rational, because at Node A Xiaoqiang's best action should be going to Node C.

But Xiaoqiang makes his claim in such a convincing voice that AMB believes that Xiaoqiang will follow this strategy at all cost however irrational it is. Then AMB finds out that his best strategy will be going to node B, and they will end up at Node E. In this case, Xiaoqiang's utility is 101. Notice that Xiaoqiang must keep his words after the claim. That is, after the claim Xiaoqiang's strategy is fixed, and AMB will make decisions to maximize his own utility provided Xiaoqiang's strategy.

Given a game, you are to determine:
(1) Xiaoqiang's best utility if he cannot make the claim.
(2) Xiaoqiagn's best utility if he can make the claim.
To make things simple, all utility values are distinct.

Remark: The real world situation is, Xiaoqiang and AMB are currently in Node D because AMB failed to get himself to University T through the entrance exam.

[Input Format]
The first line contains an integer *T*, denoting the number of the test cases.

Each test case begins with one integer *N*, the number of nodes. 1<=*N*<=200. Life starts from the first node.

Then follows *N* lines. Each line begins with an integer *M* indicating the number of actions. 0<=*M*<*N*

If *M* is nonzero, *M* integers follow. Each integer (in range [1, *N*]) represents the destination of an outgoing edge. At the end of the line is a character being either 'A' or 'X' describing whether this node belongs to AMB or Xiaoqiang.

If *M* is zero, two integers follow representing AMB's utility and Xiaoqiang's utility respectively. Utility values are in range [0, 10000] and are distinct.

[Output Format]
For each test case, output one line containing two integers representing the answers separated by a space.

[Sample Input]
```
1
6
2 5 6 A
0 102 99
0 50 0
0 100 101
2 2 3 X
2 3 4 X
```

[Sample Output]
```
99 101
```

J. Resistance

[Problem Description]

Recently DRD got a number of wires. Some of the wires have the resistance 1 ohm while the others have the resistance 0 ohm. Each wire has probability 50% to be 0 ohm or 1 ohm.

Now ATM gets a circuit board. There are *N* points in the circuit board. DRD wants to connect these points using his wires. He chooses a wire randomly and chooses two points in the board randomly. Then he will connect the two points using the wire he chooses. DRD will use *M* wires. Note that it's possible that there exist two points which are not connected by wires.

ATM is interested in the equivalent resistance between the point *S* and point *T*. Since they lack of instruments, they want you to calculate the answer.

[Input Format]

The first line contains an integer *T*, denoting the number of the test cases.

For each test case: The first line contains 4 integers: *N*, *M*, *S*, and *T*. For each of the next *M* lines, it contains 3 integers: *u*, *v*, and *c*, representing that DRD uses a wire, whose resistance is *c*, to connect the point *u* and *v*. It's guaranteed that 1 <= *N* <= 10000 and *M* = 4**N*. 1 <= *u*, *v* <= *N*. *c* is randomly chosen from {0, 1} and *u* and *v* are randomly chosen from {1, 2, ..., *N*}. Note that *u* can equal *v*.

[Output Format]

For each test case, output a real number. There must be exactly 6 digits after the decimal point. If *S* and *T* are not connected by wires, output "inf" (without quotes) instead.

[Sample Input]

```
2
10 40 6 1
9 4 1
7 3 1
10 1 0
5 2 0
6 7 1
7 3 1
3 5 1
3 6 1
8 10 0
8 3 0
7 3 1
3 9 1
2 8 1
10 5 0
10 2 1
```

```
10 9 1
9 1 0
7 5 1
10 2 0
8 1 0
2 8 0
10 2 0
1 5 0
5 4 0
7 4 1
8 5 1
4 3 1
6 1 1
5 10 0
3 9 1
5 1 0
9 2 1
3 4 1
5 8 0
3 5 1
3 4 1
2 7 0
4 4 0
1 8 0
10 10 0
10 40 2 1
5 6 1
8 10 1
3 8 1
8 5 0
6 4 1
9 9 1
3 6 1
2 4 1
9 8 0
9 3 0
7 7 1
8 6 1
10 4 1
1 3 0
2 8 1
9 8 0
8 1 1
```

```
6 4 0
3 4 0
1 4 0
1 10 0
9 10 0
6 1 1
3 1 1
5 4 0
1 2 1
7 2 1
10 9 0
6 1 0
10 3 1
8 6 1
1 4 0
1 1 0
4 3 0
1 8 0
7 10 1
10 6 0
4 5 0
2 2 0
4 2 1
```

[Sample Output]

```
0.333333333
0.222222222
```