CEOI 2019 day 2 online mirror (unrated, IOI format)

A. Amusement Park

time limit per test: 3 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

You have been hired to supervise the project of a new amusement park. The park will have a special gimmick: directed slides that can get customers from one attraction to another quickly and in an entertaining way.

The park owner has given you the current project: a list of planned attractions and a list of slides that should be built between them. However, him being a businessman, he casually envisioned the impossible: among other things, he projected a slide coming from the Haunted Castle to the Roller Coaster, another from the Roller Coaster to the Drop Tower, and a third from the Drop Tower to the Haunted Castle. As the slides can only go downhill, it is evident why this is a problem. You don't have the luxury of ignoring the laws of physics when building the park, so you have to request changes in the project. Maybe he would accept reversing the slide between the Drop Tower and the Haunted Castle?

Formally:

- The **project** is a list of attractions and a list of directed slides. Each slide starts at one attraction and ends at another attraction.
- A **proposal** is obtained from the project by reversing the directions of some slides (possibly none or all of them).
- A proposal is **legal** if there is a way to assign an elevation to each attraction in such a way that every slide goes downhill.
- The **cost** of a proposal is the number of slides whose directions were reversed.

For a given project, find and report the sum of costs all legal proposals. Since this number may be large, output it modulo $998,244,353$.

Input

The first line contains two space-separated integers n, m ($1 \le n \le 18$, $0 \le m \le n(n-1)/2$) – the number of attractions and the number of slides, respectively. The attractions are numbered 1 through n.

Then, m lines follow. The i-th of these lines contains two space-separated integers a_i, b_i ($1 \le a_i, b_i \le n$) denoting a slide from a_i to b_i.

You may assume that:

- There are no self-loops. (For each i: $a_i \neq b_i$.)
- No slide appears twice. (For all $i \neq j$: $a_i \neq a_j$ or $b_i \neq b_j$.)
- No pair of attractions is connected in both directions. (The unordered pairs $\{a_i, b_i\}$ are distinct.)

Output

Output one line with a single integer, the sum of costs of all legal proposals modulo $998,244,353$.

Scoring

Subtask 1 (7 points): $n \le 3$

Subtask 2 (12 points): $n \le 6$

Subtask 3 (23 points): $n \le 10$

Subtask 4 (21 points): $n \le 15$

Subtask 5 (37 points): no additional constraints

Examples

input
2 1 1 2

output
1

input
3 3 1 2 2 3 1 3

output
9

Note

In the first example, there are two proposals:

- The slide direction is not flipped. This proposal has cost 0.
- The slide direction is flipped. This proposal has cost 1.

As both proposals are valid, the answer is $0 + 1 = 1$.
In the second example, there are eight proposals with the slide directions as follows:

- $1 \to 2$, $2 \to 3$, $1 \to 3$ (cost 0)
- $1 \to 2$, $2 \to 3$, $3 \to 1$ (cost 1)
- $1 \to 2$, $3 \to 2$, $1 \to 3$ (cost 1)
- $1 \to 2$, $3 \to 2$, $3 \to 1$ (cost 2)
- $2 \to 1$, $2 \to 3$, $1 \to 3$ (cost 1)
- $2 \to 1$, $2 \to 3$, $3 \to 1$ (cost 2)
- $2 \to 1$, $3 \to 2$, $1 \to 3$ (cost 2)
- $2 \to 1$, $3 \to 2$, $3 \to 1$ (cost 3)

The second proposal is not legal, as there is a slide sequence $1 \to 2 \to 3 \to 1$. This means that the attraction 1 has to be strictly higher than ifself, which is clearly impossible. Similarly, the seventh proposal is not legal. The answer is thus $0 + 1 + 2 + 1 + 2 + 3 = 9$.

B. Magic Tree

time limit per test: 2 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

We have a magic tree: a rooted tree on n vertices. The vertices are numbered 1 through n. Vertex 1 is the root.

The magic tree gives us magic fruit. The fruit only grows in vertices of the tree other than the root. Each vertex contains at most one piece of fruit.

It is now day 0 and no fruit is ripe yet. Each fruit will only be ripe for a single day. For each fruit, we are given the vertex v_j where it grows, the day d_j on which it will be ripe, and the amount w_j of magic juice we can extract from it if we harvest it when it is ripe.

The fruits have to be harvested by cutting some branches of the tree. On each day, you may cut as many branches of the tree as you like. The parts of the tree you cut off will fall to the ground and you can collect all the ripe fruits they contain. All fruits that fall to the ground when they are not ripe are discarded and no magic juice is collected from them.

Formally, on each day, you may erase some edges of the tree. Whenever you do so, the tree will split into multiple connected components. You then erase all components that do not contain the root and you harvest all ripe fruits those components contained.

Given is a description of the tree together with the locations, ripening days and juiciness of all m fruits. Calculate the maximum total

amount of magic juice we can harvest from the tree.

Input
The first line contains three space-separated integers n ($2 \le n \le 100,000$), m ($1 \le m \le n-1$) and k ($1 \le k \le 100,000$) – the number of vertices, the number of fruits, and the maximum day on which a fruit may become ripe.

The following $n-1$ lines contain the integers $p_2, \ldots, p_n$, one per line. For each i (from 2 to n, inclusive), vertex p_i ($1 \le p_i \le i-1$) is the parent of vertex i.

Each of the last m lines describes one fruit. The j-th of these lines has the form "$v_j\ d_j\ w_j$" ($2 \le v_j \le n$, $1 \le d_j \le k$, $1 \le w_j \le 10^9$).

It is guaranteed that no vertex contains more than one fruit (i.e., the values v_j are distinct).

Output
Output a single line with a single integer, the maximum amount of magic juice we can harvest from the tree.

Scoring
Subtask 1 (6 points): $n, k \le 20$, and $w_j = 1$ for all j

Subtask 2 (3 points): fruits only grow in the leaves of the tree

Subtask 3 (11 points): $p_i = i - 1$ for each i, and $w_j = 1$ for all j

Subtask 4 (12 points): $k \le 2$

Subtask 5 (16 points): $k \le 20$, and $w_j = 1$ for all j

Subtask 6 (13 points): $m \le 1,000$

Subtask 7 (22 points): $w_j = 1$ for all j

Subtask 8 (17 points): no additional constraints

Example

input
6 4 10 1 2 1 4 4 3 4 5 4 7 2 5 4 1 6 9 3

output
9

Note
In the example input, one optimal solution looks as follows:

- On day 4, cut the edge between vertices 4 and 5 and harvest a ripe fruit with 1 unit of magic juice. On the same day, cut the edge between vertices 1 and 2 and harvest 5 units of magic juice from the ripe fruit in vertex 3.
- On day 7, do nothing. (We could harvest the fruit in vertex 4 that just became ripe, but doing so is not optimal.)
- On day 9, cut the edge between vertices 1 and 4. Discard the fruit in vertex 4 that is no longer ripe, and harvest 3 units of magic juice from the ripe fruit in vertex 6. (Alternately, we could achieve the same effect by cutting the edge between vertices 4 and 6.)

C. Scissors and Tape

time limit per test: 1 second
memory limit per test: 1024 megabytes
input: standard input
output: standard output

You are given a piece of paper in the shape of a simple polygon S. Your task is to turn it into a simple polygon T that has the same area as S.

You can use two tools: scissors and tape. Scissors can be used to cut any polygon into smaller polygonal pieces. Tape can be used to combine smaller pieces into larger polygons. You can use each tool multiple times, in any order.

The polygons given in the input have integer coordinates, but you are allowed to produce shapes with **non-integer coordinates** in your output.

A formal definition of the task follows.

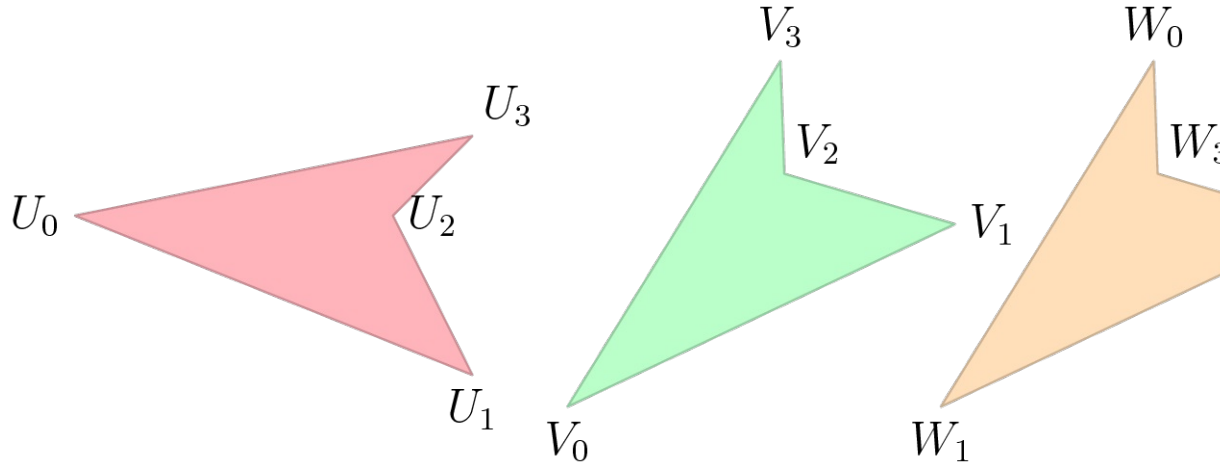
A **shape** $Q = (Q_0, \ldots, Q_{n-1})$ is a sequence of three or more points in the plane such that:

- The closed polyline $Q_0 Q_1 Q_2 \ldots Q_{n-1} Q_0$ never touches or intersects itself, and therefore it forms the boundary of a simple polygon.
- The polyline goes around the boundary of the polygon in the counter-clockwise direction.

The polygon whose boundary is the shape Q will be denoted $P(Q)$.

Two shapes are called **equivalent** if one can be translated and/or rotated to become identical with the other.

Note that mirroring a shape is not allowed. Also note that the order of points matters: the shape $(Q_1, \ldots, Q_{n-1}, Q_0)$ is not necessarily equivalent to the shape $(Q_0, \ldots, Q_{n-1})$.



In the figure on the left: Shapes U and V are equivalent. Shape W is not equivalent with them because the points of W are given in a different order. Regardless of the order of points, the fourth shape is not equivalent with the previous ones either as flipping a shape is not allowed.

In both input and output, a shape with n points is represented as a single line that contains $2n+1$ space-separated numbers: the number n followed by the coordinates of the points: $Q_{0,x}$, $Q_{0,y}$, $Q_{1,x}$, ...

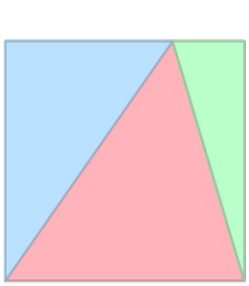
Shapes have **identification numbers** (IDs). The given shape S has ID 0, the shapes you produce in your solutions are given IDs 1,

2, 3, ..., in the order in which they are produced.

Shapes $B_1, \ldots, B_k$ form a **subdivision** of shape A if:

- The union of all $P(B_i)$ is exactly $P(A)$.
- For each $i \neq j$, the area of the intersection of $P(B_i)$ and $P(B_j)$ is zero.

The **scissors** operation destroys one existing shape A and produces one or more shapes $B_1, \ldots, B_k$ that form a subdivision of A.



In the figure: Shape A (square) subdivided into shapes B_1, B_2, B_3 (the three triangles). One valid way to describe one of the B_i is "3 3 1 6 1 5.1 4".

The **tape** operation destroys one or more existing shapes $A_1, \ldots, A_k$ and produces one new shape B. In order to perform this operation, you must first specify shapes $C_1, \ldots, C_k$ and only then the final shape B. These shapes must satisfy the following:

- For each i, the shape C_i is equivalent to the shape A_i.
- The shapes $C_1, \ldots, C_k$ form a subdivision of the shape B.

Informally, you choose the shape B and show how to move each of the existing A_i to its correct location C_i within B. Note that only the shape B gets a new ID, the shapes C_i do not.

Input

The first line contains the source shape S.

The second line contains the target shape T.

Each shape has between 3 and 10 points, inclusive. Both shapes are given in the format specified above.

All coordinates in the input are integers between -10^6 and 10^6, inclusive.

In each shape, no three points form an angle smaller than 3 degrees. (This includes non-consecutive points and implies that no three points are collinear.)

The polygons $P(S)$ and $P(T)$ have the same area.

Output

Whenever you use the scissors operation, output a block of lines of the form:

```
scissors
id(A) k
B_1
B_2
...
B_k
```

where $id(A)$ is the ID of the shape you want to destroy, k is the number of new shapes you want to produce, and $B_1, \ldots, B_k$ are those shapes.

Whenever you use the tape operation, output a block of lines of the form:

```
tape
k id(A_1) ... id(A_k)
C_1
C_2
...
C_k
B
```

where k is the number of shapes you want to tape together, $id(A_1), \ldots, id(A_k)$ are their IDs, $C_1, \ldots, C_k$ are equivalent shapes showing their position within B, and B is the final shape obtained by taping them together.

It is recommended to output coordinates of points to at least 10 decimal places.

The output must satisfy the following:

- All coordinates of points in the output must be between -10^7 and 10^7, inclusive.
- Each shape in the output must have at most 100 points.
- In each operation the number k of shapes must be between 1 and 100, inclusive.
- The number of operations must not exceed 2000.
- The total number of points in all shapes in the output must not exceed 20000.
- In the end, there must be exactly one shape (that hasn't been destroyed), and that shape must be equivalent to T.
- All operations must be valid according to the checker. Solutions with small rounding errors will be accepted. (Internally, all comparisons check for absolute or relative error up to 10^{-3} when verifying each condition.)

Scoring

A shape is called a **nice rectangle** if it has the form $((0,0),\ (x,0),\ (x,y),\ (0,y))$ for some positive integers x and y.

A shape is called a **nice square** if additionally $x = y$.

A shape A is called **strictly convex** if all inner angles of the polygon $P(A)$ are smaller than 180 degrees.

Subtask 1 (5 points): S and T are nice rectangles. All coordinates of all points are integers between 0 and 10, inclusive

Subtask 2 (13 points): S is a nice rectangle with $x > y$, and T is a nice square

Subtask 3 (12 points): S and T are nice rectangles

Subtask 4 (14 points): S is a triangle and T is a nice square

Subtask 5 (10 points): S and T are triangles

Subtask 6 (16 points): S is a strictly convex polygon and T is a nice rectangle

Subtask 7 (11 points): T is a nice rectangle

Subtask 8 (19 points): no additional constraints

Examples

input
6 0 0 6 0 6 4 5 4 5 9 0 9 4 0 0 7 0 7 7 0 7

output
scissors 0 5

```
3 0 0 3 0 3 4
3 3 4 0 4 0 0
3 3 0 6 0 6 4
3 6 4 3 4 3 0
4 0 4 5 4 5 9 0 9
tape
5 1 2 5 3 4
3 0 3 0 0 4 0
3 4 0 7 0 7 4
4 0 3 4 0 7 4 3 7
3 7 4 7 7 3 7
3 3 7 0 7 0 3
4 0 0 7 0 7 7 0 7
```

input

```
4 0 0 3 0 3 3 0 3
4 7 -1 10 -1 11 2 8 2
```

output

```
scissors
0 2
3 0 0 1 3 0 3
4 1 3 0 0 3 0 3 3
tape
2 1 2
3 110 -1 111 2 110 2
4 108 2 107 -1 110 -1 110 2
4 107 -1 110 -1 111 2 108 2
```

input

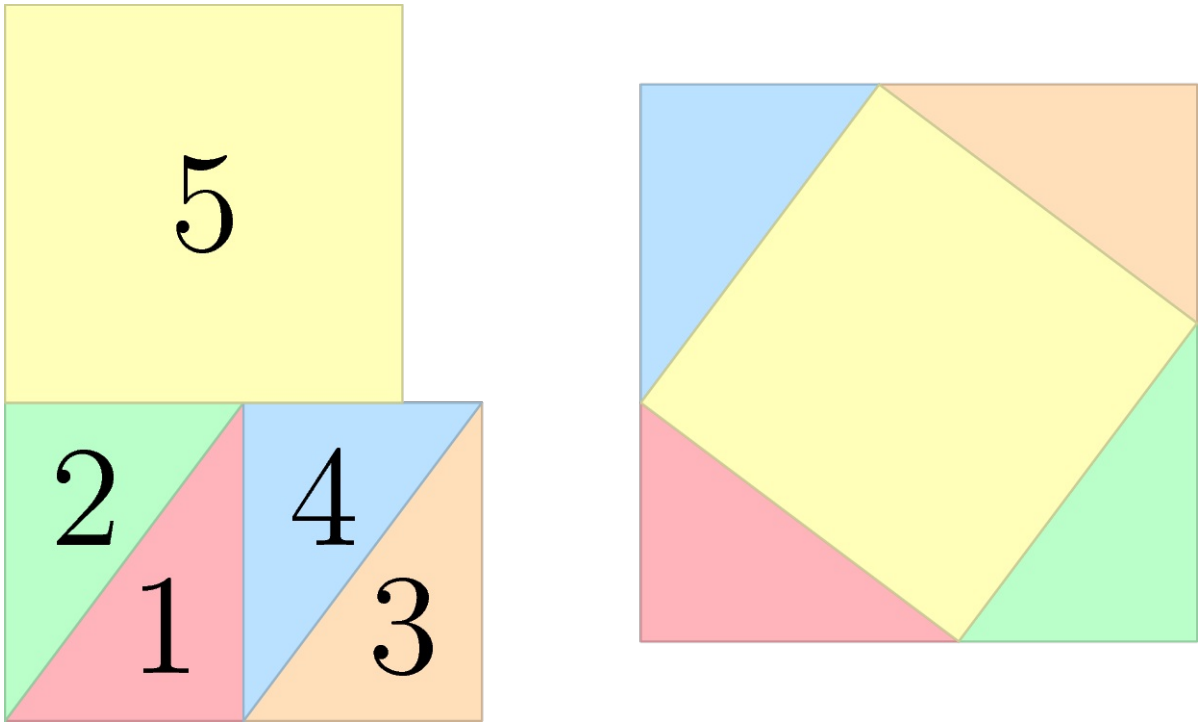
```
4 0 0 9 0 9 1 0 1
4 0 0 3 0 3 3 0 3
```

output

```
scissors
0 2
4 1.470000000 0 9 0 9 1 1.470000000 1
4 0 0 1.470000000 0 1.470000000 1 0 1
scissors
1 2
4 1.470000000 0 6 0 6 1 1.470000000 1
4 9 0 9 1 6 1 6 0
tape
2 4 3
4 3 2 3 1 6 1 6 2
4 6 1 1.470000000 1 1.470000000 0 6 0
6 1.470000000 0 6 0 6 2 3 2 3 1 1.470000000 1
scissors
5 4
4 1.470000000 0 3 0 3 1 1.470000000 1
4 3 0 4 0 4 2 3 2
4 4 2 4 0 5 0 5 2
4 5 0 6 0 6 2 5 2
tape
5 2 6 7 8 9
4 0 0 1.470000000 0 1.470000000 1 0 1
4 1.470000000 0 3 0 3 1 1.470000000 1
4 0 2 0 1 2 1 2 2
4 0 2 2 2 2 3 0 3
4 3 3 2 3 2 1 3 1
4 0 0 3 0 3 3 0 3
```

Note

The figure below shows the first example output. On the left is the original figure after using the scissors, on the right are the corresponding C_i when we tape those pieces back together.



In the second example output, note that it is sufficient if the final shape is equivalent to the target one, they do not have to be identical.

The figure below shows three stages of the third example output. First, we cut the input rectangle into two smaller rectangles, then we cut the bigger of those two rectangles into two more. State after these cuts is shown in the top left part of the figure.

Continuing, we tape the two new rectangles together to form a six-sided polygon, and then we cut that polygon into three 2-by-1 rectangles and one smaller rectangle. This is shown in the bottom left part of the figure.

Finally, we take the rectangle we still have from the first step and the four new rectangles and we assemble them into the desired 3-by-3 square.