Level 2 in computer science
INF211 (INF2112): Object-Oriented Programming (OOP)
Practice /30

Dr. Azanzi Jiomekong

Objective: The main goal of the overall exercise is to allow students to understand how to use OOP techniques to develop software. On the other hand, they will learn how to implement software using Object Oriented Programming techniques. Thus, firstly, the students will collect data on some of their daily activities. Thereafter, they will use this data to design and develop an application.

NB: The overall exercises are out of more than 30. However, students' marks will be maintained out of 30. For students having more than 30, the rest of their marks will be used for the continuous assessment bonuses.

Evaluation: Wednesday, 04th January 2023

Exercise 0: setting up the development environment **(3pts)**

1- Set up your Operating System (Linux) **0.5pt**

2- Make a table of your development environment for each programming language involved in this exercise **0.25x4=1pt**

3- Install the development environment of Java, C++, Python and Java Script programming language **0.25x4=1pt**

4- Write the "Hello World" program in your new development environment **0.5pt**

Exercice 1: Food Image annotation **(15pts)**

Adequate nutrition is an essential catalyst for economic and human development as well as for achieving Sustainable Development Goals - Goal 2: Zero Hunger and Goal 3: Ensure healthy lives and promote well-being for all at all ages. However, understanding Food information can allow people to have a healthy diet. To this end, food information engineering involves the acquisition, the processing and the diffusion of up-to-date food information to different stakeholders. These information are compiled from several data sources and used for a variety of purposes such as food recommendation, recipe substitution, food image recommendation, nutritional agendate, etc. To recommend a food to a person, a task can consist of first recognizing this food from a food image. To this end, Food images should be annotated.

Food image annotation or tagging is the assignment of metadata in the form of captioning or keywords to a digital food image or a set of food images (for instance to name a dish). Actually, many dishes may contain different kinds of foods, making it difficult to identify the food contained in a dishe.

The goal of this exercise is to apply the OOP techniques to annotate and recognize African food images. To this end, a set of images should be downloaded, relevant objects identified in each image. The annotators should identify specific objects, segment an image into relevant regions, which are specific points of interest in the food image. To ensure labeling accuracy, each member of the group should contribute to labeling the same

image. Thereafter, the image should be annotated by indicating the shape and label of each food item in the dishe and the name of the dishe.

This work can be done individually or by a group of 2-5 students. Each student or group of students should execute the following tasks:

1- Choose one country in Africa and create an image dataset of the food eaten in this country (at least 50 images per person) **0.5pt**

2- From each dish, identify food objects and build an object diagram **1pt**

3- Use the question 2 to organize foods into classes and determine the relations that are between these classes and build the class diagram **0.5+0.5+0.5 = 1.5pt**

4- From questions 2 and 3, and a tool of your choice (for instance Computer Vision Annotation Tool) to Identify objects in images and use lines, polygons, or markers to draw lines along the object's perimeter or circumference. **5pts**

5- Export the annotations in the format provided by the annotation tool that you choose (.json or .txt) **1pt**

In the rest of the exercise, we will consider the Java, C++, Java Script and Python programming languages.

6- Create the classes (with their properties and relations) in Java, C++, Java Script and Python **1pt**

7- Create objects of the classes of question 6 **1pt**

8- Identify operations that are common to all the dishes and use the appropriate OOP concept to implement it **2pt**

9- Write the final program for food recognition: it takes images characteristics and return information on the food **2pt**

Exercice 2: Nutritional agenda **(15pts)**

Following exercise 1, the goal of this exercise is to collect information on your food habit and use it to recommend healthy behavior. Actually, the Gastrointestinal (GI) track consists of the mouth, stomach, and intestines. Together with the liver, gallbladder, and pancreas, these organs work together to absorb nutrients and expel waste. Disturbances to this process can cause a range of health problems such as bloating, cramping, gas, abdominal pain, diarrhea and constipation, etc. The goal of this exercise is to build a tool that will help people know which food causes digestive disorders or allergies. To this end, the following tasks should be done:

1- Collect data on the food you are eating in a daily basis by filling the following table **2pts**

Date (Monday xxx)	Foods eaten	NB times food eaten per day	Quantity of water drank	Quantity of other liquid drank	Eating fruits and legumes	NB bowel movement	Health problem

Consider that the food eaten can be organized with an heterogeneous graph in which the nodes represent foods, and his digestive consequence.

2- Propose this graph **1.5pt**

3- Build a state-machine diagram of a person **1pt**

4- Identify objects and build an object diagram and the class diagram **1+1=2pts**

5- Use the graph of question 2 to propose a model of your eating habit in a daily basis **1.5pt**

6- Propose an algorithm that can be used to predict the food that you'll eat **1pt**

7- Propose an algorithm that can be used to describe your eating habit (this is a graph transversal algorithm) **1pt**

If we consider that our body is a bag that should be filled with food1 (x1 quantity), food2 (x2 quantity), food3 (x3 quantity), water (y quantity). Xi is the maximum quantity of foodi that the body should contain.

8- Propose the fill max bad program that can allow you to have healthy eating habit **1pt**

9- Implement your system using the POO techniques **5pts**

Exercice 3: Surviving in a competitive environment **(10pts)**

This is the game of life in which we consider two types of entities: the mouse and the cat. The mouse and the cat forage, fight and bargain their way through procedurally generated environments to outcompete other mice and cats trying to do the same. We consider in this game that the world has a beginning and an end that can be parameterized during the game initialization. At its end, a dashboard presenting the entities and their rest of life. The goal is to simulate populations of agents in procedurally generated virtual worlds. It is inspired by classic massively multiagent online role-playing games. We consider that:

- The game starts with a map made up of NxN tiles (parameterized at the beginning of the game);
- In each tile, only one agent can be present. In some cases, two agents of the same species can be present only if one is a male and another one a female. In this case, in a certain number of times, these agents give birth to babies and all the agents move to the adjacent tiles, leaving the female in the current tile.
- The maize appear randomly on the tiles
- The mice and the cats should eat to survive
- The cats eat the mice and the mice eat maize
- If a cat and a mouse meet in a tile, the cat will kill the mouse and if the cat is not full, it will eat him
- If any entity did not eat for a certain time, it will die
- Two entity of the same sex should not meet in the same tile
- Each entity can make multiple actions: move, eat, reproduce.
- Each entity can observe NxN tiles around him
- If an entity move out the map, it will die

To build this game, reply to the following questions:

1- Draw a map made up of 25 tiles, 3 cats and 5 mices **1pt**

2- Simulate the game with the environment of question 1 (you can use activity diagram or sequence diagram, etc.) **1pt**

3- From the picture of question 1, identify objects and build an object diagram - for each object, identify the messages that it can send or receive **1pt**

4- Use the question 3 to propose the class diagram **1pt**

5- Build the state-machine diagram of the main objects of the system **1pt**

6- Implement the classes of question 4 in Java, C++, Python and JS **2pts**

7- Implement the whole application **3pts**

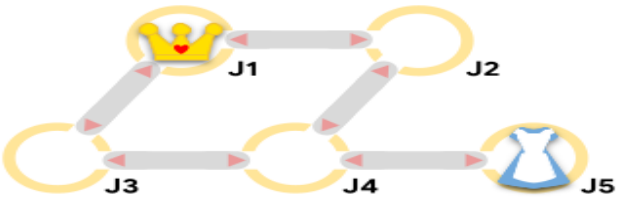
Exercice 4 10pts

Problem

Alice is trapped in Wonderland's labyrinth, being chased by the Queen of Hearts and her herald! The labyrinth is a set of $\mathbf{J}$ junctions numbered 1 through $\mathbf{J}$, connected by $\mathbf{C}$ bidirectional corridors.

Alice and the Queen of Hearts take turns making moves, and each knows the location of the other at all times. A move (by either of them) consists of either staying at the current junction or moving to another one that is connected to it by a corridor.

The Queen's herald, however, announces the next move the Queen makes in advance. That means that before anyone makes a move, he announces the Queen's first move. Then, Alice moves first. Then, each time the Queen moves, she must respect the previous announcement, and then decide her next move so the herald can announce it. Alice hears the announcements, so she always knows the Queen's next move before making her own.



If Alice and the Queen are at the same junction after either of them moves, then Alice is caught. Otherwise, the pursuit continues. After 10^9 total moves (half of them for Alice and half for the Queen), if Alice and the Queen are not in the same junction, then the Queen will give up and Alice will be safe.

Alice chooses her moves optimally to escape. If she cannot escape, she chooses her moves to maximize the total number of moves until she is caught. The Queen chooses her moves optimally to try to catch Alice in as few total moves as possible.

Given the labyrinth's layout and the initial locations of both the Queen and Alice, find out whether Alice will be caught by the Queen and, if so, in how many moves.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case starts with a line containing four integers $\mathbf{J}$, $\mathbf{C}$, $\mathbf{A}$, and $\mathbf{Q}$: the number of junctions, the number of corridors, the junction where Alice starts, and the junction where the Queen starts, respectively. Then, $\mathbf{C}$ lines follow. The i-th of these lines contains two integers $\mathbf{U_i}$ and $\mathbf{V_i}$, indicating that the i-th corridor bidirectionally connects junctions $\mathbf{U_i}$ and $\mathbf{V_i}$.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1) and y is SAFE if Alice can avoid being caught for 10^9 total moves. Otherwise, y is the total number of moves (including Alice's and the Queen's) that it takes for the Queen to catch Alice.

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{A} \leq \mathbf{J}$.
$1 \leq \mathbf{Q} \leq \mathbf{J}$.
$\mathbf{A} \neq \mathbf{Q}$.
$1 \leq \mathbf{U_i} < \mathbf{V_i} \leq \mathbf{J}$, for all i.
$(\mathbf{U_i}, \mathbf{V_i}) \neq (\mathbf{U_j}, \mathbf{V_j})$, for all $i \neq j$.

Test Set 1 (Visible Verdict)

Time limit: 10 seconds.
$2 \leq \mathbf{J} \leq 30$.
$1 \leq \mathbf{C} \leq 60$.

Test Set 2 (Hidden Verdict)

Time limit: 60 seconds.
$2 \leq \mathbf{J} \leq 10^5$.
$1 \leq \mathbf{C} \leq 2 \times 10^5$.

Sample

Sample Input

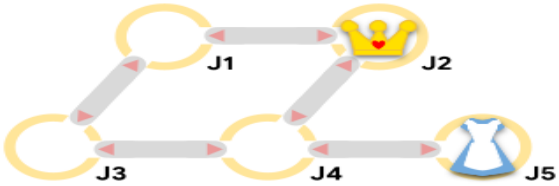
```
4
5 5 5 1
1 2
1 3
2 4
3 4
4 5
5 5 5 2
1 2
1 3
2 4
3 4
4 5
3 1 2 3
1 3
2 1 1 2
1 2
```

Sample Output

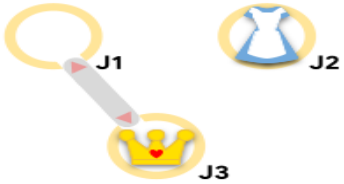
```
Case #1: SAFE
Case #2: 4
Case #3: SAFE
Case #4: 2
```

Sample Case #1 is the one pictured in the problem statement. Alice's optimal first move is to move to junction 4.

Sample Case #2 is the same as Sample Case #1 but the Queen starts at junction 2. The Queen can catch Alice by first announcing a move to junction 4. If Alice were to move to junction 4, she would be caught in 2 moves. Alice can evade capture for an extra 2 moves by staying put and waiting until the Queen then moves to junction 5 where she is located.



In Sample Case #3, the Queen cannot reach Alice no matter what she does.



In Sample Case #4, the Queen can begin by announcing that she will move to Alice's current junction. Alice has to move before then. If Alice moves to where the Queen already is, she gets caught immediately; if Alice remains in place, then she gets caught when the Queen moves. The second option is better, since it requires 2 total moves (Alice's and the Queen's) instead of 1.

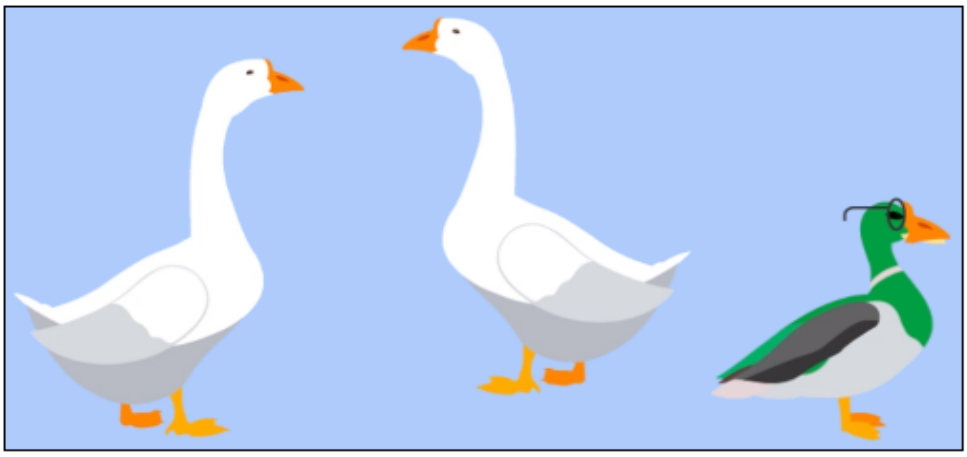
Exercise 5 10pts

Problem

The first international Geese conference just wrapped up, and even though it should have been a happy occasion, it was bittersweet. The organizers found a paper with detailed plans of a duck infiltration. Now, they are trying to identify the infiltrating group from among the attendees.

The document that they found contained a list of $\mathbf{M}$ triples of integers $(\mathbf{X_i}, \mathbf{Y_i}, \mathbf{C_i})$ meaning the ducks would meet exactly $\mathbf{C_i}$ seconds after the start of the conference at point $(\mathbf{X_i}, \mathbf{Y_i})$, which is $\mathbf{X_i}$ meters east and $\mathbf{Y_i}$ meters north of the center of the conference floor. Each goose may or may not have been at those specific points at those specific times, but every duck certainly was.

Both ducks and geese walk at a maximum speed of one meter per second, which means an attendee that is at point (x, y) at time t can reach any point of the form $(x + \Delta_x, y + \Delta_y)$ by time $t + \Delta_t$ as long as ${\Delta_x}^2 + {\Delta_y}^2 \leq {\Delta_t}^2$. Each attendee's position at time 0 can be any point, independently of the other attendees.



After the discovery, the group held a questioning session to try to identify the ducks. During that session, attendees issued a series of statements, one at a time. The j-th of those, in the order they were issued, was made by attendee $\mathbf{A_j}$, claiming that both they and attendee $\mathbf{B_j}$ were at point $(\mathbf{U_j}, \mathbf{V_j})$ exactly $\mathbf{D_j}$ seconds after the start of the conference. Points in statements may or may not be points where duck meetings happened.

Statements from geese are always true, but ducks may lie. Moreover, ducks know which attendees are ducks and which are geese. To avoid getting caught easily, ducks only make statements that are consistent with all statements previously made by geese. Note that statements made by geese are consistent with all ducks being at all duck meetings.

It may not be possible to determine all the ducks with the information provided. However, knowing the minimum number of ducks will at least provide a lower bound on the level of duck activity. Note that there was at least one duck. Find this minimum number of ducks.

Formally, a *hypothesis* H is a partition of all attendees into a set of ducks (named H-ducks) and geese (named H-geese). H is consistent with a set of statements S if there exists a path for each attendee moving at most one meter per second such that:

- all H-ducks were at all duck meetings and
- for each statement in S claiming that A saw B at point P at time T, both A and B's paths went through point P at time T.

A hypothesis H is *feasible* under a set of statements S if:

- H-ducks is not empty *(i.e., there was at least one duck)*,
- the subset of all statements from S made by members of H-geese is consistent with H *(i.e., statements from geese are always true), and*
- for each statement $s \in S$ made by a member of H-ducks, if $P \subseteq S$ is the subset of statements made by members of H-geese issued before s, there exists a hypothesis H' (which may or may not be equal to H) such that $\{s\} \cup P$ is consistent with H' *(i.e., ducks do not contradict previous statements made by geese).*

Notice that the hypotheses H such that H-ducks contains all attendees is always feasible.

Find the minimum size of H-ducks over all feasible hypotheses H.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case starts with a line containing three integers, $\mathbf{N}$, $\mathbf{M}$, and $\mathbf{S}$, representing the numbers of attendees, duck meetings, and statements, respectively. The next $\mathbf{M}$ lines each describe a different duck meeting with three integers $\mathbf{X_i}$, $\mathbf{Y_i}$, and $\mathbf{C_i}$, representing that there was a meeting at point $(\mathbf{X_i}, \mathbf{Y_i})$, held exactly $\mathbf{C_i}$ seconds after the start of the conference. Then, the last $\mathbf{S}$ lines of a test case each describe a statement. The j-th of these lines describes the j-th issued statement with five integers $\mathbf{A_j}$, $\mathbf{B_j}$, $\mathbf{U_j}$, $\mathbf{V_j}$, and $\mathbf{D_j}$, representing that attendee $\mathbf{A_j}$ stated that they and attendee $\mathbf{B_j}$ were both at point $(\mathbf{U_j}, \mathbf{V_j})$ exactly $\mathbf{D_j}$ seconds after the start of the conference.

Output

For each test case, output one line containing `Case #x: y`, where x is the test case number (starting from 1) and y is the minimum number of ducks that might have infiltrated the conference.

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 50$.
$-10^9 \leq \mathbf{X_i} \leq 10^9$, for all i.
$-10^9 \leq \mathbf{Y_i} \leq 10^9$, for all i.
$1 \leq \mathbf{C_i} \leq 10^9$, for all i.
$\mathbf{C_i} < \mathbf{C_{i+1}}$, for all i.
$(\mathbf{X_i} - \mathbf{X_{i+1}})^2 + (\mathbf{Y_i} - \mathbf{Y_{i+1}})^2 \leq (\mathbf{C_i} - \mathbf{C_{i+1}})^2$, for all i.
$1 \leq \mathbf{A_j} \leq \mathbf{N}$, for all j.
$1 \leq \mathbf{B_j} \leq \mathbf{N}$, for all j.
$\mathbf{A_j} \neq \mathbf{B_j}$, for all j.
$-10^9 \leq \mathbf{U_j} \leq 10^9$, for all j.
$-10^9 \leq \mathbf{V_j} \leq 10^9$, for all j.
$1 \leq \mathbf{D_j} \leq 10^9$, for all j.
$(\mathbf{A_j}, \mathbf{B_j}, \mathbf{U_j}, \mathbf{V_j}, \mathbf{D_j}) \neq (\mathbf{A_k}, \mathbf{B_k}, \mathbf{U_k}, \mathbf{V_k}, \mathbf{D_k})$, for all $j \neq k$.

Test Set 1 (Visible Verdict)

Time limit: 20 seconds.
$2 \leq \mathbf{N} \leq 50$.
$1 \leq \mathbf{M} \leq 50$.
$1 \leq \mathbf{S} \leq 50$.

Test Set 2 (Hidden Verdict)

Time limit: 60 seconds.
$2 \leq \mathbf{N} \leq 10^5$.
$1 \leq \mathbf{M} \leq 10^5$.
$1 \leq \mathbf{S} \leq 10^5$.

Sample

Sample Input

```
2
2 1 2
1 2 3
1 2 1 1 1
2 1 2 2 2
4 2 4
4 3 10
-4 -3 20
1 3 4 3 11
2 4 0 0 16
3 1 6 3 9
4 2 0 0 16
```

Sample Output

```
Case #1: 1
Case #2: 2
```

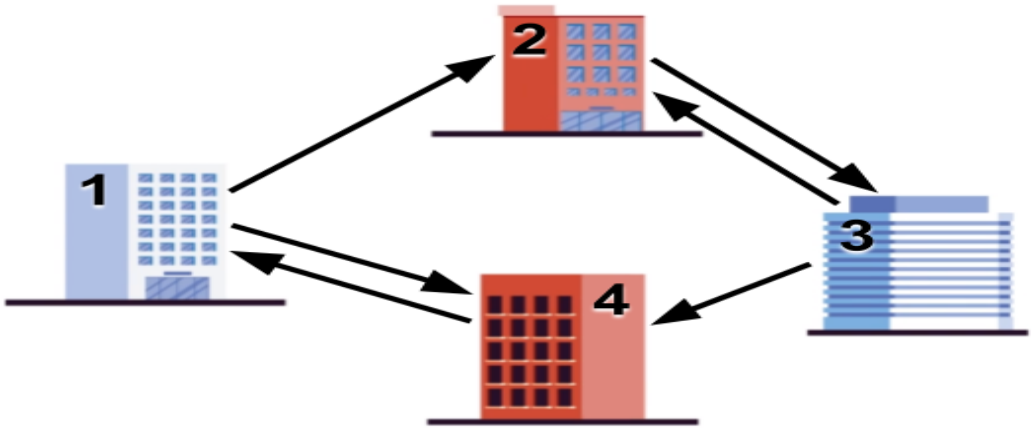
In Sample Case #1, attendee 1 being the only duck is a feasible hypothesis.

In Sample Case #2, attendees 2 and 4 being the only ducks is a feasible hypothesis. Note that there is at least one duck, so all attendees being geese is not feasible.

Exercice 6 10pts

Problem

Gooli is a huge company that owns $\mathbf{B}$ buildings in a hilly area, numbered 1 through $\mathbf{B}$. Six years ago, Gooli built slides that allowed employees to go from one building to another. Each slide allows anyone to go from the slide's origin building to the slide's destination building, but not the other way around. Gooli's CEO is very proud of their slides and wants to organize a parade through the slides. She has tasked Melek, Gooli's Head of Transportation and a problem-solving enthusiast, with designing the parade's route.



She has some requirements for the parade route in mind:

- It must start and end at building 1, where her office is located.
- It must visit each building the same number of times. Being in building 1 at the start of the route does not count as a visit.
- It must use each slide at least once.
- It must have at most 10^6 steps.

Given the layout of buildings and slides, help Melek find a route that satisfies all of the CEO's requirements, if one exists.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case starts with a line containing two integers $\mathbf{B}$ and $\mathbf{S}$: the number of buildings and slides, respectively. Then, $\mathbf{S}$ lines follow. The i-th of these lines contains two integers $\mathbf{U_i}$ and $\mathbf{V_i}$, indicating that the i-th slide goes from building $\mathbf{U_i}$ to building $\mathbf{V_i}$.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1). If there is no route that fulfills all the requirements, y must be `IMPOSSIBLE`. If there is, y must be an integer between $\mathbf{S}+1$ and 10^6+1, inclusive, representing the length of one such route you want to exhibit. In that case, output another line containing y integers $z_1\ z_2\ \ldots\ z_y$, where z_j is the j-th building in your proposed route. Notice that $z_1 = z_y = 1$ and that each building must appear the same number of times among the z_j, except for building 1, which appears exactly one extra time.

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{U_i} \leq \mathbf{B}$, for all i.
$1 \leq \mathbf{V_i} \leq \mathbf{B}$, for all i.
$\mathbf{U_i} \neq \mathbf{V_i}$, for all i.
$(\mathbf{U_i}, \mathbf{V_i}) \neq (\mathbf{U_j}, \mathbf{V_j})$, for all $i \neq j$.

Test Set 1 (Visible Verdict)

Time limit: 10 seconds.
$2 \leq \mathbf{B} \leq 10$.
$2 \leq \mathbf{S} \leq 10$.

Test Set 2 (Hidden Verdict)

Time limit: 20 seconds.
$2 \leq \mathbf{B} \leq 200$.
$2 \leq \mathbf{S} \leq 5000$.

Sample

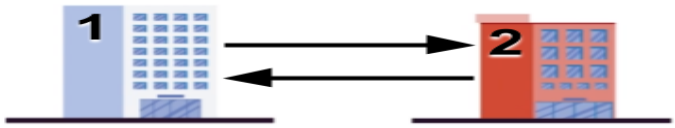
Sample Input

```
5
2 2
2 1
1 2
3 4
2 3
1 2
3 2
1 3
3 6
1 2
1 3
2 1
2 3
3 1
3 2
3 4
1 2
2 1
1 3
3 1
4 6
1 2
1 4
2 3
3 2
3 4
4 1
```

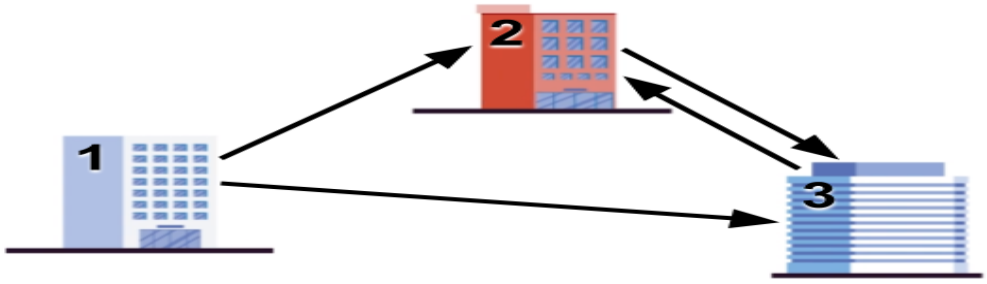
Sample Output

```
Case #1: 7
1 2 1 2 1 2 1
Case #2: IMPOSSIBLE
Case #3: 7
1 2 3 1 3 2 1
Case #4: IMPOSSIBLE
Case #5: 9
1 4 1 2 3 2 3 4 1
```

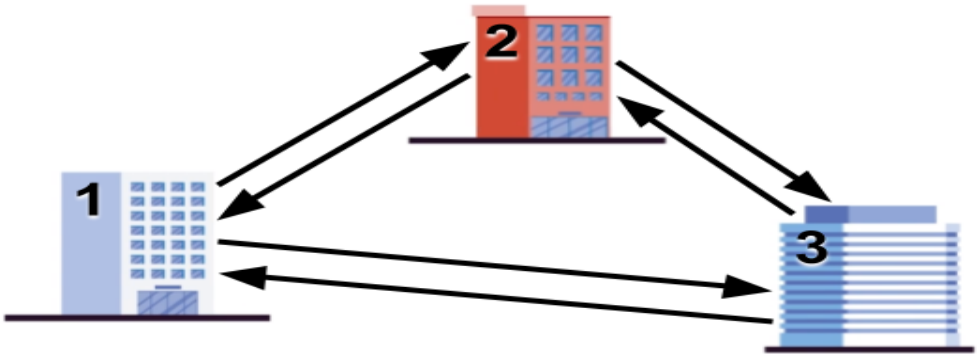
In Sample Case #1, another acceptable parade route is one that goes from building 1 to building 2 and then back for a total of 2 steps.



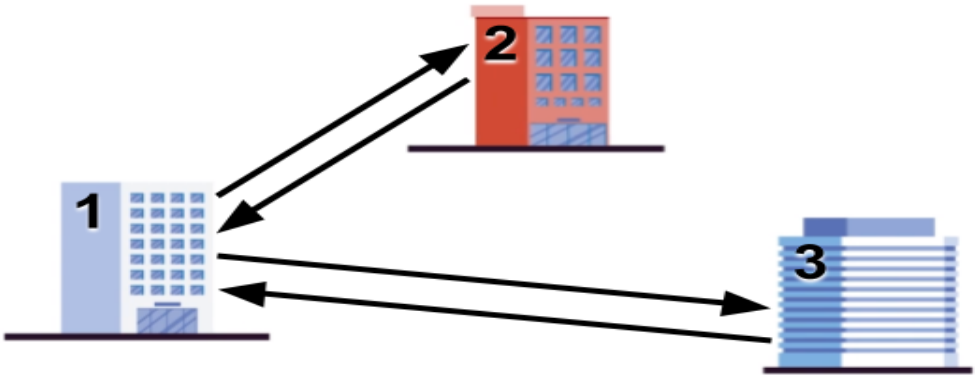
In Sample Case #2, there are no slides leading to building 1, so no valid parade can exist.



In Sample Case #3, the parade route the sample output exhibits goes through each building twice.



Sample Case #4 is pictured below.



Sample Case #5 is the one illustrated in the problem statement. In the parade route in the sample output, the slides from 2 to 3 and from 4 to 1 are used twice, but the rest of the slides are used only once each.

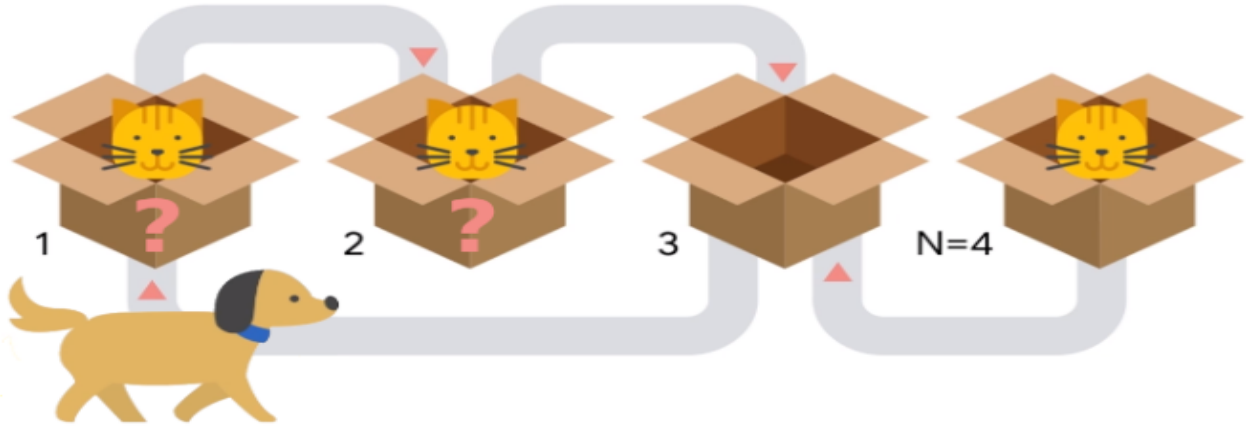
Problem

The story, all names, characters, and incidents portrayed in this problem statement are fictitious. No identification with actual persons is intended or should be inferred.

It is 1935 and a meeting between two Nobel prize winners is producing astonishing results. Schrödinger, a famous physicist, invited Pavlov, a famous physiologist, to see his experiments with cats in boxes. Pavlov brought his dog with him to keep up with his own research, and the combination proved interesting, to say the least.

Schrödinger had a row of $\mathbf{N}$ boxes. Some boxes definitely contain a cat, some boxes definitely do not contain a cat, and some boxes may or may not contain a cat. Each box is only big enough to hold a single cat. Each box is also equipped with a special quantum tunnel, that allows the cat in the box to move to some other specific box if the destination was empty. The tunnels work in a single direction.

Cats are usually mellow and quiet and do not use the tunnels unless they become startled. When a third unannounced guest rings the bell, Pavlov's dog gets excited immediately and starts running and barking. The dog starts at box 1 and runs towards box $\mathbf{N}$. As the dog runs, it passes right next to each box, one at a time. When it passes next to a box that contains a cat, the cat in that box becomes startled. The startled cat checks the available tunnel and, if the destination box is empty, uses it to escape. If the destination box is occupied, the cat stays in its current box. The same cat can be startled more than once if they move to a box the dog will get to afterwards, and will proceed in the same way every time it is startled (using only the newly available tunnel each subsequent time).



After Pavlov's dog finally stops right next to the last box, Pavlov asks Schrödinger whether there is a cat in that last box. Schrödinger, true to his fame, replies that he does not know. Pavlov notices that the answer may depend on whether or not there were cats in the unknown boxes. Moreover, he also notices that because there are k unknown boxes, there are 2^k possible *initial configurations*, one for each combination of statuses of the unknown boxes. Pavlov tells Schrödinger that they should try to calculate how many of the 2^k initial configurations would result in having a cat in the last box. You are asked to recreate that calculation. Since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime $10^9 + 7$ (1000000007).

Neither cats, nor dogs, nor Nobel prize winners were harmed in the making of this problem statement.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow, each described by exactly three lines. The first line of a test case contains a single integer $\mathbf{N}$, the number of boxes in Schrödinger's experiment. Boxes are numbered between 1 and $\mathbf{N}$, in the order Pavlov's dog passes them by. The second line of a test case contains a single string $\mathbf{S}$ of $\mathbf{N}$ characters. The i-th character of $\mathbf{S}$ (counting from left to right) represents the contents of box i: it is an uppercase 'C' if the box contains a cat, a period '.' if the box does not contain a cat and a question mark '?' if it is unknown whether the box contains a cat or not. The third line of a test case contains $\mathbf{N}$ integers $\mathbf{B_1}, \mathbf{B_2}, \ldots, \mathbf{B_N}$, representing that there is a tunnel going out of box i and into box $\mathbf{B_i}$, for all i.

Output

For each test case, output one line containing `Case #x: y`, where x is the test case number (starting from 1) and y is the number of initial configurations that would result in a cat being in the last box and unable to escape despite hearing the barking, modulo the prime $10^9 + 7$ (1000000007).

Limits

Time limit: 10 seconds.
Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 1234$.
the length of $\mathbf{S} = \mathbf{N}$.
Each character of $\mathbf{S}$ is either an upper case 'C', a period '.' or a question mark '?'.
$1 \leq \mathbf{B_i} \leq \mathbf{N}$, for all i.
$\mathbf{B_i} \neq i$, for all i.

Test Set 1 (Visible Verdict)

$1 \leq \mathbf{N} \leq 100$.
$i - 5 \leq \mathbf{B_i} \leq i + 5$, for all i. (All tunnels connect to nearby boxes.)

Test Set 2 (Hidden Verdict)

$1 \leq \mathbf{N} \leq 5000$.

Sample

Sample Input

```
4
4
??.C
2 3 1 3
4
????
2 3 1 3
6
?.????
6 6 6 6 6 5
34
??????????????????????????????
2 3 4 5 6 7 8 9 10 11 12 13 14
```

Sample Output

```
Case #1: 1
Case #2: 2
Case #3: 15
Case #4: 294967268
```

Sample Case #1 is illustrated in the problem statement. There are 4 possible configurations:

- ...C: the dog runs through the first 3 boxes without changing anything because there is no cat there. Then, when it gets to the last box, the cat hears it and escapes to box 3. Therefore, there is no cat in the last box in this case.
- C..C: when the dog barks near box 1, that startles the cat that goes through the tunnel to get to box 2, which was empty. Then, the same cat gets startled again when the dog barks near box 2 and gets to box 3. And when the dog barks next to box 3, the cat hears it and returns to box 1. Therefore, when the dog gets to box 4 and the other cats hears it, box 3 is empty so the cat escapes and the last box ends up empty.
- .C.C: This case is very similar to the previous one. After the dog goes through the first box and nothing happens, the state is the same as before, so the ultimate result is the same: last box empty.
- CC.C: In this case, the cat in the first box cannot escape when it hears the dog, so it remains in box 1. Then, when the cat in box 2 gets startled it escapes to box 3 leaving a state of C.CC. When the dog gets to the box 3, the cat currently there cannot escape to box 1 so the state remains the same. Finally, when the dog gets to the last box, the cat that is there cannot escape because box 3 is occupied this time. So, in this case, the last box ends up with a cat after the dog ends its journey.

Out of the 4 possibilities, only 1 (the last one) ends up with a cat in the last box, so the answer is 1.

In Sample Case #2, the tunnels are set up the same as in Sample Case #1. Since no tunnel ends at the last box, the configurations that start with no cat at the last box will also not end with a cat there, so we do not need to count them. Then, we have 8 additional configurations. The 4 we considered for Sample Case #1, out of which only 1 ends up with a cat at the last box. The remaining 4 configurations are: ..CC, C.CC, .CCC, CCCC. From these additional 4 configurations, only in the last one listed a cat ends up in the last box, for a total of 2 overall.

In Sample Case #3, notice that for a cat to remain in the last box after the dog barks near it, both that box and box 5 must be occupied then (otherwise, either there is no cat in the last box, or it will escape to box 5). Since there is no tunnel going into box 5, a cat must start there. As long as there is another cat in any other box, box 6 will get (or remain) occupied before the cat in box 5 gets an opportunity to escape, so all of those will end up with a cat in the last box. As we argued before, a single cat is not enough. Thus, we need to count the number of configurations with a cat in box 5 and at least one other cat. There are 2^4 configurations with a cat in box 5, and out of those, only 1 has no other cat, so the answer is $2^4 - 1 = 15$.

In Sample Case #4, in all of the 2^k ways in which the k unknown boxes may exist a cat would be left in the last box.

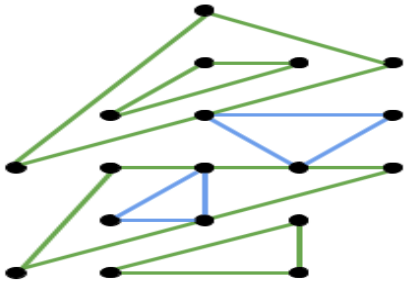
Exercice 7 10pts

Problem

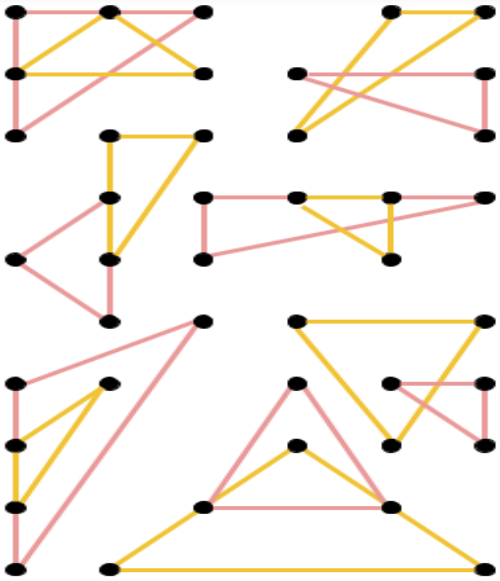
You are given a set P of $\mathbf{N}$ distinct points in the two-dimensional plane. You want to find a maximum set of triangles such that:

- Each vertex of a triangle in your set is a point from P and each point in P is a vertex of at most one triangle in your set.
- Each triangle in your set has positive area (i.e., its 3 vertices are not collinear).
- For any two sides of triangles in your set, their intersection is either empty or an endpoint of one of them.
- For any two triangles in your set, the intersection of the areas strictly inside those triangles is either empty or equal to one of them.

For example, the set of triangles depicted below meets the definition above.



On the other hand, each pair of a yellow and a red triangle in the picture below does not meet the definition.



Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case starts with a line containing a single integer $\mathbf{N}$. Then, $\mathbf{N}$ lines follow. The i-th of these lines contains two integers $\mathbf{X_i}$ and $\mathbf{Y_i}$ representing the coordinates of the i-th point.

Output

For each test case, output one line containing `Case #x: y`, where x is the test case number (starting from 1) and y is the maximum size of a set of triangles with the desired properties. Then, output y more lines. The j-th of those lines must contain $p_j\ q_j\ r_j$ representing that the j-th triangle in your proposed set has the p_j-th, q_j-th, and r_j-th points in the input as vertices. Points in the input are numbered starting from 1.

Limits

Time limit: 15 seconds.
Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$-10^9 \leq \mathbf{X_i} \leq 10^9$, for all i.
$-10^9 \leq \mathbf{Y_i} \leq 10^9$, for all i.
$(\mathbf{X_i}, \mathbf{Y_i}) \neq (\mathbf{X_j}, \mathbf{Y_j})$, for all $i \neq j$.

Test Set 1 (Visible Verdict)

$3 \leq \mathbf{N} \leq 12$.

Test Set 2 (Hidden Verdict)

$3 \leq \mathbf{N} \leq 3000$.

Sample

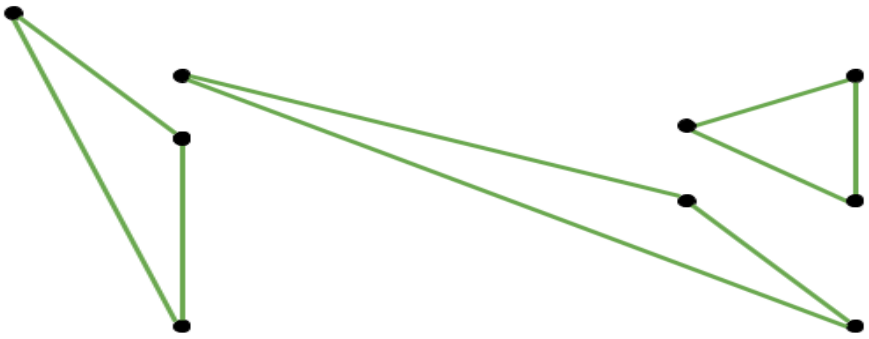
Sample Input

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

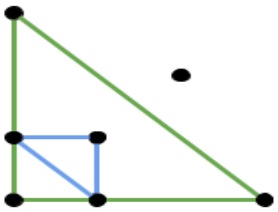
Sample Output

```
Case #1: 3
3 4 5
1 7 9
6 2 8
Case #2: 2
2 3 1
6 5 4
Case #3: 0
```

Sample Case #1 is illustrated below. Notice that there are other valid ways to construct a maximum number of triangles.



Sample Case #2 is illustrated below. As before, there are other valid ways to construct 2 triangles.



In Sample Case #3, the 3 given points are collinear, so it is not possible to make a valid triangle with them.

Exercice 8: 10pts

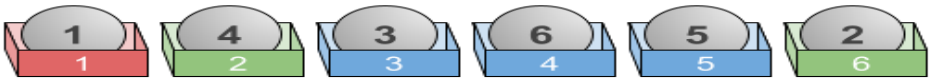
Problem

In this problem, when something is said to be chosen at random, it means uniformly at random from among all valid possibilities, and independently of any other choice.

Code Jam contestants once helped the mighty Goro sort an array of integers. (You do not need to read that problem to solve this one.) Once again, Goro needs your help. He has $\mathbf{N}$ boxes lined up on the table in a single row, numbered 1 through $\mathbf{N}$ from left to right. Each box has exactly one ball inside. Balls are also numbered 1 through $\mathbf{N}$. Goro wants ball i to end up in box i, for all i. That is, he wants to leave the balls in sorted order. Unfortunately, that is not initially the case.

When Goro bumps the table with his powerful fists, balls pop up in the air and fall back in boxes. Goro can do this so accurately that exactly one ball falls into each box. A ball may fall into the same box it came out of, or into a different one.

Better yet, Goro also has the ability to assign colors to boxes before each bump. Then, he can bump the table in such a way that balls coming out of a box of color c always fall into a box of color c. As impressive as this accuracy is, Goro does not have any more control than that. Within each color, balls end up assigned to boxes at random.



For example, suppose the balls appear in the order $1, 4, 3, 6, 5, 2$ (as seen above). He might choose — not necessarily optimally — to give the first box the color red, the second and sixth boxes the color green, and the third through fifth boxes the color blue. Then, after Goro bumps the table,

- The 1 in the first box falls back into the same box, because that is the only red box.
- The 4 and 2 in the second and sixth boxes remain in place with probability $\frac{1}{2}$, and switch places with probability $\frac{1}{2}$.
- The $3, 6, 5$ in the third, fourth, and fifth boxes end up in one of the following orders, each with probability $\frac{1}{6}$:
  - $3, 6, 5$
  - $3, 5, 6$
  - $6, 3, 5$
  - $6, 5, 3$
  - $5, 3, 6$
  - $5, 6, 3$

So, for example, the probability of the bump leaving the balls in the order $1, 2, 3, 5, 6, 4$ is $\frac{1}{12}$. If Goro got this or some other non-sorted result, he would have to designate a set of box colors for the next round, and so on, until he eventually arrives at the sorted $1, 2, 3, 4, 5, 6$. Goro can assign colors to boxes in any way before each bump, regardless of previous assignments.

Can you help Goro implement a better strategy that will efficiently sort the balls? It is guaranteed that the balls start in a random non-sorted order.

Input and output

This is an interactive problem. You should make sure you have read the information in the Interactive Problems section of our FAQ.

Initially, your program should read a single line containing three integers, $\mathbf{T}$ $\mathbf{N}$ $\mathbf{K}$: the number of test cases, the number of boxes per test case, and the total number of bumps allowed for all test cases combined. Then, $\mathbf{T}$ test cases must be processed.

Each test case begins with the judge sending one line with $\mathbf{N}$ integers, with each integer from 1 to $\mathbf{N}$ appearing exactly once, and with the list chosen at random from all non-sorted lists. Then you must engage in a series of interactions with the judge. Each interaction works as follows:

- You send one line of $\mathbf{N}$ integers $C_1, C_2, \ldots, C_{\mathbf{N}}$, in which each integer is between 1 and $\mathbf{N}$, inclusive. Each C_i represents that you are assigning color C_i to box i for the next bump. You may choose how many colors there are and how they are numbered, but you must assign a color to each box.
- The judge simulates the bump as explained in the statement. If this results in the balls being in sorted order:
  - If this interaction was the $\mathbf{K}$-th interaction across all test cases, and this was not the last test case, the judge sends one line with the integer -1 and does not output anything further.
  - Otherwise, the judge sends one line with the integer 1 and then immediately begins the next test case, if there is one. If this was the last test case, your program must exit without error and without sending anything further.
- Otherwise, the balls are not sorted, and:
  - If this interaction was the $\mathbf{K}$-th across all test cases, or if you provided an invalid line (e.g., too few integers, or color numbers out of range), the judge sends one line with the integer -1 and does not output anything further.
  - If this was not your $\mathbf{K}$-th interaction, the judge sends one line with the integer 0, and then another line with $\mathbf{N}$ integers, with each integer from 1 to $\mathbf{N}$ appearing exactly once, and in non-sorted order, representing the new order of the balls, that is, the i-th of these integers is the ball that fell into box i. Then you must begin another interaction.

As usual, if the memory limit is exceeded, or your program gets a runtime error, you will receive the appropriate judgment. Also, if your program continues to wait for the judge after receiving a -1, your program will time out, resulting in a Time Limit Exceeded error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error.

Be advised that the judge uses the same source of randomness each time, so in the absence of other errors (e.g. Time Limit Exceeded, Memory Limit Exceeded), submitting the exact same code twice will yield the same outcome twice.

Limits

Time limit: 20 seconds.
Memory limit: 1 GB.
$\mathbf{T} = 1000$.
$\mathbf{N} = 100$.

Test Set 1 (Visible Verdict)

$\mathbf{K} = 16500$.

Test Set 2 (Visible Verdict)

$\mathbf{K} = 12500$.

Test Set 3 (Visible Verdict)

$\mathbf{K} = 11500$.

Testing Tool

You can use this testing tool to test locally or on our platform. To test locally, you will need to run the tool in parallel with your code; you can use our interactive runner for that. For more information, read the instructions in comments in that file, and also check out the Interactive Problems section of the FAQ.

Instructions for the testing tool are included in comments within the tool. We encourage you to add your own test cases. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently. If your code passes the testing tool but fails the real judge, please check the Coding section of the FAQ to make sure that you are using the same compiler as us.

Download testing tool

Sample Interaction

Judge **Solution**

Number of cases, number of boxes per case, number of interactions allowed across all cases

```
2 4 8
```

Case 1

```
1 4 3 2
```

Judge gives this non-sorted list.

```
1 2 3 2
```

Solution assigns color 1 to the first box, color 2 to the second and fourth boxes, and color 3 to the third box.
The judge trivially assigns the balls in the boxes of color 1 and 3 back to their original box, and randomly chooses whether to swap the balls in the second and fourth boxes. In this case, it does not swap them.

```
0
1 4 3 2
```

```
1 2 3 2
```

Solution does the same thing as before.
This time the judge does happen to swap the 4 and the 2.

```
1
```

Judge indicates that this test case was solved, and immediately begins the next case.

Judge indicates that this test case was solved, and immediately begins the next case.

Case 2

```
2 1 4 3
```

```
4 4 4 4
```

Solution assigns all boxes color 4. Notice that it's OK to not have colors 1, 2, or 3.
The judge chooses a new random order for the balls. Wow, what luck, they come out in order!

```
1
```

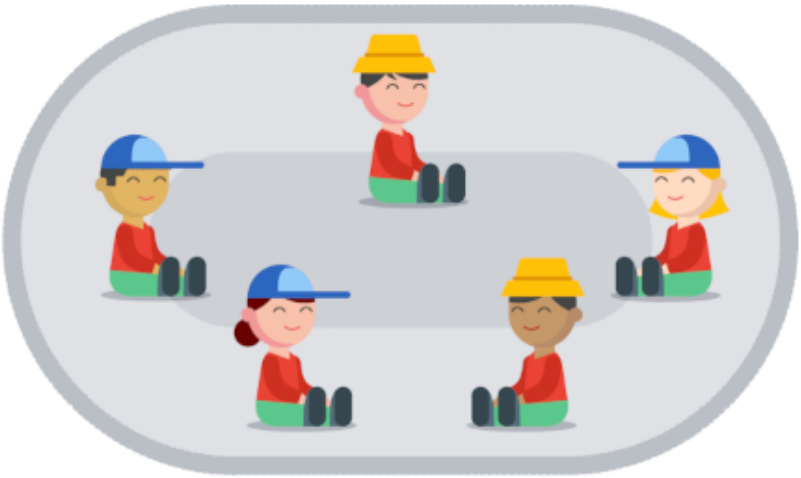
Judge indicates that this test case was solved, and sends no further output, because this was the last test case.
Contestant solution should exit (with no error) to avoid a Time Limit Exceeded error.

Note that the sample interaction does not satisfy the constraints of any of the test sets. It is only presented to clarify the input and output format.

Problem

In the game "Duck, Duck, Goose", all players but one sit on the floor and form a circle. The remaining player walks around the circle calling each player "duck" until they select one sitting player and, while touching their head, call them "goose" instead. At that point, the goose chases the selecting player and our interest in the game fades.

In the new game "Duck, Duck, Geese", the walking player instead chooses a contiguous subset of at least two (but not all) sitting players to be "geese"! Furthermore, each sitting player is wearing a hat. Each hat is one of $\mathbf{C}$ possible colors, numbered 1 through $\mathbf{C}$.



For each color i, the quantity of selected geese wearing a hat of color i must be either 0 or between $\mathbf{A_i}$ and $\mathbf{B_i}$, inclusive.

Can you help count the number of choices that fulfill these requirements? Two choices are considered different if there is some player that is included in one choice but not the other.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case starts with a line containing two integers $\mathbf{N}$ and $\mathbf{C}$: the number of sitting players and hat colors, respectively. Then, $\mathbf{C}$ lines follow. The i-th of these lines contains two integers $\mathbf{A_i}$ and $\mathbf{B_i}$, as explained above. The last line of a test case contains $\mathbf{N}$ integers $\mathbf{P_1}, \mathbf{P_2}, \ldots, \mathbf{P_N}$ representing that the j-th sitting player in clockwise order (starting from an arbitrary one) is wearing a hat of color $\mathbf{P_j}$.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1) and y is the number of sets of at least 2 and at most $\mathbf{N} - 1$ contiguously sitting players that fulfill all the color requirements.

Limits

Time limit: 20 seconds.
Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$2 \leq \mathbf{C} \leq \mathbf{N}$.
$0 \leq \mathbf{A_i} \leq \mathbf{B_i} \leq \mathbf{N}$, for all i.
$1 \leq \mathbf{P_j} \leq \mathbf{C}$, for all j.

Test Set 1 (Visible Verdict)

$3 \leq \mathbf{N} \leq 1000$.

Test Set 2 (Hidden Verdict)

$3 \leq \mathbf{N} \leq 10^5$.

Sample

Sample Input

```
3
3 2
1 1
1 1
1 1 2
5 2
1 1
1 2
1 2 1 2 2
3 3
1 2
1 2
2 2
1 1 3
```

Sample Output

```
Case #1: 2
Case #2: 9
Case #3: 1
```

In Sample Case #1, the total number of players chosen as geese must be 2. There are only three possible ways to select 2 players. The following color configurations are possible: $[1, 1]$, $[1, 2]$, and $[2, 1]$. The first one has two players wearing hats of color 1, so it is not valid, but the other two are valid. Therefore the answer is 2.

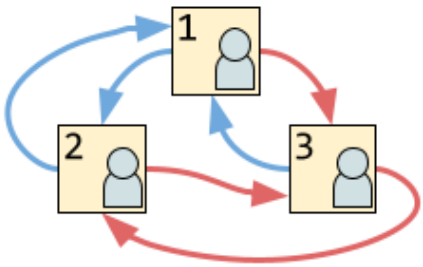
Sample Case #2 is the one illustrated in the statement, with color 1 being yellow and color 2 being blue. The total number of players chosen as geese in this case must be between 2 and 3, because selecting 4 geese would require at least one color to be out of bounds. For cases with 2 geese, the only requirement is that we do not select 2 geese both wearing hats of color 1; all 5 such selections are valid. If choosing 3 geese, the options are $[1, 2, 1]$, $[2, 1, 2]$, $[1, 2, 2]$, $[2, 2, 1]$, or $[2, 1, 2]$. All but the first one are valid, adding another 4 valid options, for a total of 9.

In Sample Case #3, notice that there can be hat colors that nobody is wearing. In this case, since there is only 1 player wearing hat color 3 and 1 is not in range, the only valid way is to pick 0 players wearing that hat color.

Problem

The Google Coding Competitions team is setting up a new theme park. As in any good theme park, we want to have actors dressed up as mascots to interact with visitors. Because we are in a rush to open, we decided to use the letters from `CODE JAM`, `KICK START`, and `HASH CODE` as mascots, for a total of 13 different mascots (the letters `ACDEHIJKMORST`).

The park's only attraction is a maze that has a set of $\mathbf{N}$ rooms numbered from 1 to $\mathbf{N}$. Each room has a left exit and a right exit. Each exit takes the visitor to another room. Exits cannot be used in reverse; for example, if room 2 has an exit to room 3, you cannot go back from room 3 to room 2 unless room 3 also happens to have an exit to room 2.



We want to place exactly one of our 13 mascots in each room. Each letter may be present in zero, one, or more rooms of the maze. To increase variety, we want to place mascots so that any three (not necessarily distinct) rooms that a visitor can visit consecutively have three different mascots.

Can you help us choose a mascot for each room such that this goal is met, or let us know that it cannot be done?

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case consists of 3 lines. The first line contains a single integer $\mathbf{N}$, representing the number of rooms in the maze. The second line contains $\mathbf{N}$ integers $\mathbf{L_1}, \mathbf{L_2}, \ldots, \mathbf{L_N}$, representing that the left exit from room i leads to room $\mathbf{L_i}$. The third and last line contains $\mathbf{N}$ integers $\mathbf{R_1}, \mathbf{R_2}, \ldots, \mathbf{R_N}$, representing that the right exit from room i leads to room $\mathbf{R_i}$.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1) and y is `IMPOSSIBLE` if there is no way to assign mascots while obeying the rules explained above. Otherwise, y is an $\mathbf{N}$ character long string. The i-th character of y should be an uppercase letter from the set `ACDEHIJKMORST`, representing that you wish to assign that mascot to the i-th room.

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$\mathbf{L_i} \neq i$, for all i. $\mathbf{R_i} \neq i$, for all i. $1 \leq \mathbf{L_i} < \mathbf{R_i} \leq \mathbf{N}$, for all i.

Test Set 1 (Visible Verdict)

Time limit: 20 seconds.
$3 \leq \mathbf{N} \leq 100$.

Test Set 2 (Hidden Verdict)

Time limit: 45 seconds.
$3 \leq \mathbf{N} \leq 10^5$.

Sample

Sample Input

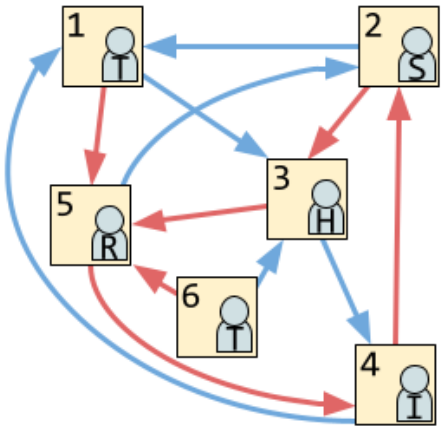
```
4
3
2 1 1
3 3 2
6
3 1 4 1 2 3
5 3 5 2 4 5
20
2 3 4 5 6 7 8 9 10 11 12 13 14
3 4 5 6 7 8 9 10 11 12 13 14 1
19
2 3 4 5 6 7 8 9 10 11 12 13 14
3 4 5 6 7 8 9 10 11 12 13 14 1
```

Sample Output

```
Case #1: IMPOSSIBLE
Case #2: TSHIRT
Case #3: HCJKSHCJKSHCJKSHCJKS
Case #4: CODEJAMROCKSTHEMOST
```

Sample Case #1 is the image in the problem statement. It is possible to visit rooms 1, 2, and 1 consecutively (which visits room 1 twice), so the case is impossible.

Sample Case #2 has the following layout (blue arrows represent the left exits and red arrows represent the right exits):



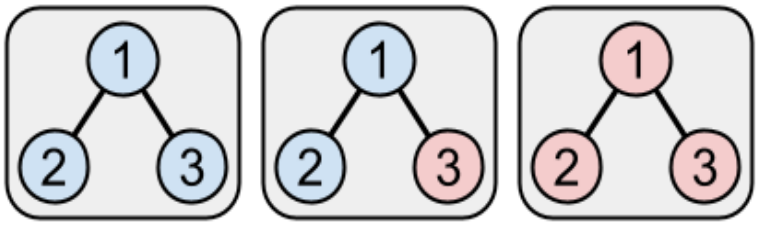
One of many valid answers is to assign mascots as indicated. Notice that although we do not need to assign two T mascots in this case, we have done so in a way that does not break the rules.

Sample Cases #3 and #4 are possible, but require the use of multiple copies of some mascots.

Problem

Ueli and Vreni are playing a game. The game's board is a tree with $\mathbf{N}$ vertices, all initially colored blue. They alternate turns, with Ueli going first. In each turn, a player must choose a blue vertex, together with any subset (possibly none or all) of its blue neighbors, and color all those vertices red. If at the start of a players' turn, all vertices are red, then that player loses the game and the other player wins the game.

In the example game below, Ueli colored vertex 3 red in their first turn. Then, Vreni chose vertex 2 for their turn and colored both it and its neighbor (vertex 1) red. Because all vertices are now red, Ueli loses and Vreni wins.



Ueli and Vreni have noticed that it is much easier for Ueli to win this game because he has the first turn. Therefore they have adopted the following procedure: first, Ueli chooses an integer $\mathbf{N}$. Then, Vreni chooses any tree with $\mathbf{N}$ vertices. And then they start playing as described above, with Ueli taking the first turn.

Vreni is hopeful that being able to choose the tree can help her overcome the disadvantage of going second. Can you demonstrate how Vreni can win games in this setup?

Input and output

This is an interactive problem. You should make sure you have read the information in the Interactive Problems section of our FAQ.

Initially, your program should read a single line containing an integer, $\mathbf{T}$, the number of test cases. Then, $\mathbf{T}$ test cases must be processed.

For each test case, your program must first read a line containing a single integer $\mathbf{N}$, the number of vertices that Ueli has chosen. Then, your program must output $\mathbf{N} - 1$ lines describing the edges of the tree Vreni should choose. The nodes of the tree are numbered 1 through $\mathbf{N}$. Each line must represent a distinct edge of the tree with 2 integers between 1 and $\mathbf{N}$: the two vertices the edge connects. The edges must represent a tree. The two integers within a line may be in either order, and the $\mathbf{N} - 1$ lines themselves may be in any order.

After that, your program must read a line containing a single integer $\mathbf{M}$, the number of games that you need to play on this tree. These games are played independently; in other words, all vertices of the tree are blue at the start of each game.

For each of the $\mathbf{M}$ games, you need to process some number of exchanges until the game is over. Each exchange consists of a turn from each player.

For each exchange, your program must read two lines describing Ueli's turn first. The first of those lines will contain an integer $\mathbf{K}$, denoting the number of blue vertices to be colored red. The second of those lines will contain $\mathbf{K}$ distinct integers $\mathbf{A_1}, \mathbf{A_2}, \ldots, \mathbf{A_K}$ describing the blue vertices to be colored red. $\mathbf{K}$ will be at least 1, and each $\mathbf{A_i}$ will be between 1 and $\mathbf{N}$, inclusive. Vertices $\mathbf{A_2}, \mathbf{A_3}, \ldots, \mathbf{A_K}$ will all be neighbors of vertex $\mathbf{A_1}$.

After that, your program must output Vreni's choice for their turn in the same format: the first line with the number of blue vertices to be colored red, followed by the second line with the numbers of those vertices, in such an order that all vertices except the first one are neighbors of the first one.

If all vertices are red after Vreni's turn, it means that Vreni has won and this game is over. The next game starts immediately if there is one. If this was the last game for this test case, then the next test case starts immediately if there is one. If this was the last test case, the judge will send no further input to your program, and the program must send no further output.

On the other hand, if all vertices are red after Ueli's move, it means that Vreni has lost and therefore your program did not pass the test case. In this case, instead of starting a new exchange by printing the last move that colors all remaining blue vertices red, the judge will print a single number -1 and will not print any further output, and will not process any further games or test cases.

If the judge receives an invalidly formatted or invalid line (like outputting an unexpected number of integers, or integers out of range, or outputting a set of edges that do not form a tree, or trying to color a vertex that is already red, or trying to color a vertex that is not a neighbor of the first vertex colored in this turn) from your program at any moment, the judge will also print a single number -1 and will not print any further output. If your program continues to wait for the judge after receiving a -1, your program will time out, resulting in a Time Limit Exceeded error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error. As usual, if the memory limit is exceeded, or your program gets a runtime error, you will receive the appropriate judgment.

The judge is deterministic. In other words, if you make two attempts that print the same numbers, you will get the same inputs from the judge. However, of course the judge can make different moves in different games on the same tree.

Limits

Time limit: 60 seconds.
Memory limit: 1 GB.
$1 \leq \mathbf{M} \leq 50$.

Test Set 1 (Visible Verdict)

$\mathbf{T} = 1$.
$\mathbf{N} = 30$.

Test Set 2 (Hidden Verdict)

$1 \leq \mathbf{T} \leq 10$.
$31 \leq \mathbf{N} \leq 40$.
No two test cases use the same value of $\mathbf{N}$.

Testing Tool

You can use this testing tool to test locally or on our platform. To test locally, you will need to run the tool in parallel with your code; you can use our interactive runner for that. For more information, read the instructions in comments in that file, and also check out the Interactive Problems section of the FAQ.

Instructions for the testing tool are included in comments within the tool. We encourage you to add your own test cases. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently. If your code passes the testing tool but fails the real judge, please check the Coding section of the FAQ to make sure that you are using the same compiler as us.

Download testing tool

Note that the testing tool just makes random choices for Ueli unless it can win in one turn. Therefore, it might be easier to win against the testing tool than against the real judge, which will try harder to win.

Sample Interaction

Judge	Solution
Number of cases	
2	
Case 1	
3	
Judge gives $\mathbf{N} = 3$.	
	1 2 1 3
	Solution outputs a tree with 3 vertices.
1	
A single game will be played on this tree.	
Case 1, Game 1 (illustrated above)	
1 3	
Judge colors the third vertex red. Note that the judge could have won immediately by coloring all vertices red, and therefore -1 would also be a possible output from the judge here.	
	2 1 2
	Solution colors the two remaining vertices red and wins, so we move to the next test case since there was just one game to play in the first test case.

Case 2

```
4
```

Judge gives $\mathbf{N} = 4$.

```
1 2
2 3
2 4
```

Solution outputs a tree, pictured below.

```
2
```

Two games will be played on this tree.

Case 2, Game 1 (illustrated below)

```
3
2 1 3
```

Judge colors the first three vertices red. Note that vertex 2 must be printed first here.

```
1
4
```

Solution colors the remaining vertex red and wins, so we move to the next game.

Case 2, Game 2 (illustrated below)

```
2
2 3
```

Judge now makes a better first turn, coloring the two middle vertices red.

```
1
1
```

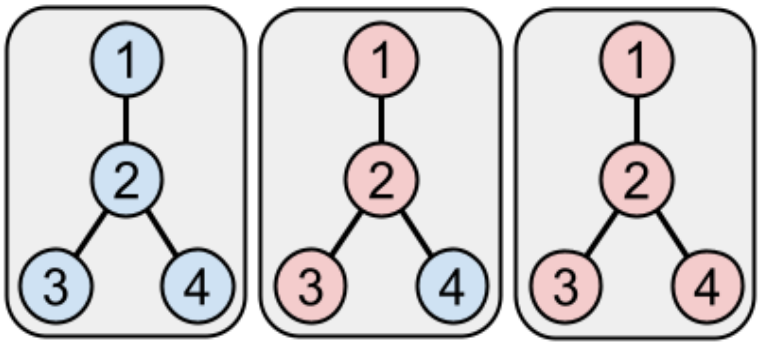
Solution colors the first vertex red.

```
-1
```

The judge can now win by coloring the last vertex red, so the solution is incorrect.

Note that the sample interaction does not satisfy the constraints of either test set, as its $\mathbf{N}$ values are too small. It is only presented to clarify the input and output format.

Below is an illustration of Case #2, Game #1 at the beginning and after each turn:



Below is an illustration of Case #2, Game #2 at the beginning and after each turn: