Problem PA
Counterfeit Money

Time limit: 3 seconds
Memory limit: 1024 megabytes

Problem Description

The banknotes of the ICPC Kingdom have anti-counterfeiting measures. Each banknote has an exclusive serial number, and this serial number is divisible by 13. In other words, if the serial number is not divisible by 13, then the banknote is counterfeit. To verify whether a number is divisible by 13, we can directly divide the number by 13. Yet, there is another method:

Partition the digits of the given decimal number into groups starting from the right, where each group has three digits. Now, treat each group as a three-digit number. Then, from the rightmost group, apply subtraction and addition operations alternately to the three-digit number and obtain the result. If the result is divisible by 13, then the original number is divisible by 13. Otherwise, it is not.

For example, for the number 123,456,789, if we apply subtraction and addition operations alternately from the rightmost group of 3 digits, we get $789 - 456 + 123 = 456$. As 456 is not divisible by 13, the original number 123,456,789 is not divisible by 13.

For another example, for the number 593,825,856, if we apply subtraction and addition operations alternately from the rightmost group of 3 digits, we get $856 - 825 + 593 = 624$. As 624 is divisible by 13 ($624 = 13 \times 48$), the original number 593,825,856 is divisible by 13.

Based on the above method, write a program to verify whether a banknote is counterfeit or not.

Input Format

The input contains several test cases. The first line stands for the number of test cases t. The next t lines will each contain a positive number. The given number may contain up to 1000 digits.

Output Format

For each input number, output the absolute value of the result when we apply the above alternate-add-subtract method. Then, on the same line, output "`YES`" if the input number is divisible by 13, and "`NO`" otherwise. There is a space between the output value and `YES`/`NO`.

Technical Specification

- $1 \leq t \leq 1000$.
- Each input number may contain up to 1000 digits.

Sample Input 1

```
2
123456789
593825856
```

Sample Output 1

```
456 NO
624 YES
```

Hint

- string and simulation
- Partition the given number into sets starting from the right, each group has three digits. We have the following two methods:
  - From the rightmost group of 3 digits apply the subtraction and addition operations alternatively and find the result. If the result is either a 0 or it can be divisible by 13 completely without leaving a remainder, then the number is divisible by 13 (simulate the statement of problem).
  - Numbered the group from the right. Let S_{odd} is the sum of groups with numbered odd, and S_{even} is the sum of groups with numbered even. If $|S_{odd} - S_{even}|$ is either a 0 or it can be divisible by 13 completely without leaving a remainder, then the number is divisible by 13.

Problem PB
Recurring Decimal to Fractions

Time limit: 3 seconds
Memory limit: 1024 megabytes

Problem Description

Given two strings of numbers representing a fraction smaller than one in recurring decimal form. The first string s_1 indicates the non-repeating part after the decimal point of the recurring decimal and the second string s_2 indicates the repeating part of the recurring decimal such as

```
1
012
```
means $0.1\overline{012}$

Return two integers n, d represent the fraction in the form of numerator and denominator. The two integers should be relatively prime.

Input Format

The first line contains an integer $T(\leq 40)$, representing the number of test cases. Each test case below contains two lines. For each test case, the first line has two integers a and b separated by a space. The second line is a string s_1 with length a and the third line is a string s_2 with length b.

Output Format

Each test case outputs two integers n and d, separated by a space. The first integer n is the numerator and the second integer d is the denominator. It is necessary to simplify the fraction so that the numerator and denominator are relatively prime.

Technical Specification

- $1 \leq a$
- $1 \leq b$
- $1 \leq a + b \leq 10$

Sample Input 1

```
2
1 3
0
012
3 1
085
3
```

Sample Output 1

```
2 1665
32 375
```

Hint

- Euclidean algorithm

Problem PC
Where the Lantern Lights are Dimming

Time limit: 3 seconds
Memory limit: 1024 megabytes

Problem Description

The Lantern Festival features many lanterns on display. In the darkness of the night, they cast beautiful shadows and reflections, attracting numerous visitors to come and go. With so many lanterns on display simultaneously, it's impossible to showcase the unique features of each. As such, the organizers switch on the lights of some lanterns while turning off others in rotation. At any given time, some lanterns are illuminated while others rest. Additionally, some lanterns remain perpetually off due to malfunctions, missing their chance to dazzle.

To engage the visitors in the Lantern Festival, the organizers also hold a scoring event. If a visitor is **very satisfied** with the festival, they will receive a pack of stickers worth 3 points each, and they will affix one 3-point sticker to each lantern on display. If they feel **satisfied**, they will receive a pack of 1-point stickers and place one on each displaying lantern. If they are **disappointed** with the festival, they will get a pack of -2 point stickers and apply one to each displaying lantern. In other words, those lanterns that are resting and not illuminated won't have an opportunity to receive any stickers from this visitor. After the festival ends, please help write a program to calculate the total points from all the stickers on the lanterns.

Input Format

The input begins with a line containing two integers, n and m. The next n lines describe the initial state of n lanterns, numbered from 0 to $n-1$. Each of these lines contains two integers: s_i and p_i. s_i represents the state of the lantern:

- 1 if the i-th lantern is initially on,
- 0 if it is off,
- -1 if it is out of order (meaning it is perpetually off and cannot be turned on)

p_i indicates the total points from stickers that are already on the i-th lantern.

The following m lines represent m events given in chronological order, each corresponding to one of two even types: switching or scoring.

- Lines beginning with the letter `W` signify a switching event. They have two subsequent integers, l_j and r_j, which imply that the state of lanterns numbered in the range $[l_j, r_j]$ (inclusive) will be toggled.

- Lines beginning with the letter `C` denote a scoring event by a visitor. These lines have a single subsequent integer, $q_j \in \{-2, 1, 3\}$, indicating the sticker score assigned by the visitor. Every lantern currently on display receives a sticker with q_j points from the visitor.

Output Format

Output a single integer that is the total points from all the lanterns after the festival.

Technical Specification

- $1 \leq n \leq 1,000,000$
- $1 \leq m \leq 1,000,000$
- $s_i \in \{-1, 0, 1\}$
- $-10,000 \leq p_i \leq 10,000$
- $0 \leq l_j \leq r_j < n$
- $q_j \in \{-2, 1, 3\}$

Sample Input 1

```
3 3
0 0
0 0
0 0
W 0 2
W 1 1
C 3
```

Sample Output 1

```
6
```

Sample Input 2

```
5 5
1 5
0 0
-1 2
1 0
0 -2
C 1
W 0 4
C -2
W 1 3
C 3
```

Sample Output 2

```
9
```

Hint

- Maintaining an integer y that is the total points. Initially, $y = \sum p_i$.
- Skip all the out-of-order lanterns and build an array A for normal lanterns only.
- Maintaining an integer x that represents the number of displaying lanterns.
- In each scoring event, y is increased by $q_j \times x$
- The states of the n lanterns are maintained in a segment tree with lazy propogation for state flipping.
- For a given input range $[l_j, r_j]$, find the exact $[l'_j, r'_j]$ indexes from A by using binary search to skip the out-of-order lanterns. Then, perform a range update within the new range.

The complexity of each scoring event is $O(1)$, and the complexity of each switching event is $O(\log n)$. The overall complexity is $O(m \log n)$.

Almost blank page

Problem PD
Quarantine Policy

Time limit: 3 seconds
Memory limit: 1024 megabytes

Problem Description

The 2019 novel coronavirus, COVID-19, can be transmitted between humans through water droplets and close contact. The transmission is especially easy and fast in relatively crowded or confined spaces, such as airplanes or trains. If someone is infected with COVID-19, then passengers occupying the adjacent seats will be infected easily.

To prevent the spread of the virus, we can take precautions, such as washing hands regularly and avoiding touching our eyes, nose, or mouth, to avoid infection. In addition, governments have also implemented special measures such as isolation and quarantine for this purpose. For instance, when someone on an airplane caught the coronavirus, the person will need to be isolated. Moreover, persons occupying the seats adjacent to the infected person will need to be quarantined. Precisely, there are two types of adjacent seats. One is directly adjacent, that is the seat is in the front, rear, left, or right of the virus seat. The other one is diagonally adjacent, that is the seat is in the front-left, front-right, rear-left, or rear-right of the virus seat. In the quarantine policy, someone whose seat is directly adjacent will be quarantined for d_1 days, and someone whose seat is diagonally adjacent will be quarantined for d_2 days. If there is more than one infected person adjacent to some seat, the number of days of quarantine will not be accumulated.

Please write a program to output which seats whose occupying persons need to be quarantined, and the number of days of quarantine. If a seat whose occupying person needs to be quarantined for different days, output the maximum of such days.

Input Format

The input contains several test cases. The first line stands for the number of test cases t. For each test case, the first line contains four integers n, m, d_1, d_2 ($0 < n, m \leq 100$, $1 \leq d_2 \leq d_1 < 10$), which stands for that there are n lines and m columns of the airplane, and a seat will be quarantined d_1 days if the seat is adjacent to the virus seat directly (i.e., front, rear, left, right), and a seat will be quarantined d_2 days if it is adjacent to the virus seat in the diagonal directions (front-left, front-right, rear-left, rear-right). The next n lines contain exactly m characters and represent the seats on the airplane.

Each healthy seat is represented by a '.' character and each virus seat is represented by a 'V' character.

Output Format

For each airplane, first print the following message in a line alone:

Airplane #z:

where z stands for the label of the airplane (starting with 1). The next n lines replace each '.' character in the input seats by the corresponding number of days to quarantine for that seat.

Technical Specification

- $1 \leq t \leq 1000$.
- $0 < n, m \leq 100$.
- $1 \leq d_2 \leq d_1 < 10$.

Sample Input 1

```
2
4 4 7 3
.V..
....
..V.
....
2 2 1 1
V.
..
```

Sample Output 1

```
Airplane #1:
7V70
3773
07V7
0373
Airplane #2:
V1
11
```

Hint

- simulation
- Similar as UVA10189 (Minesweeper): counts the total mines adjacent to a square.
- The differences are (1)There are two types of adjacent seats: adjacent directly and diagonal direction. (2) If there is more than one confirmed case adjacent to someone on the same flight, the number of days of quarantine will not be accumulated.

Problem PE
Slabstones Rearrangement

Time limit: 3 seconds
Memory limit: 1024 megabytes

Problem Description

Babara has a garden. She has bought some rectangular slabstones and worked out an initial placement of all slabstones in the garden. The shape of the garden is rectangular. An edge of a slabstone should be either parallel or orthogonal to an edge of the garden. There exists a slabstone touching the left, right, bottom, and top edges of the garden, respectively. All the slabstones are contained in the garden. Meanwhile, none of the slabstones overlap in the initial placement. Babara enjoys stepping slabstones from one to another every day. However, Babara would like to redesign her garden to make room for some other purposes. She is wondering how tight the slabstones can be packed together if they can only be shifted horizontally (i.e., left or right) without changing their vertical coordinates. Furthermore, if the vertical dimensions of any two slabstones overlap (not including touching of their ends), their relative locations in the horizontal direction should be maintained. That is, if the vertical dimensions of slabstones R and Q overlap and, before shifting, slabstone Q is on the right of slabstone R, Q should be still on the right of R after shifting or vice versa. Besides, there is a minimal horizontal spacing between two slabstones during slabstone rearrangement if their vertical dimensions overlap. The slabstones should remain non-overlapping after shifting. Nevertheless, their horizontal edges may touch. Now you are asked to help Babara calculate the largest area that can be spared for other purposes.

Input Format

The first line holds an integer specifying the number of test cases. It is then followed by the input data of the test cases. The first line of the input for each test case gives two integers. The first one specifies the number of rectangular slabstones whereas the second one gives the minimal horizontal spacing between two slabstones. Then, each of the following lines contains four integers. The first two integers specify the initial x and y coordinates of the bottom-left corner of a slabstone in the garden. The remaining two integers specify the initial x and y coordinates of the top-right corner of a slabstone. Two adjacent numbers are separated by a whitespace.

Output Format

The output of a test case takes a line. It contains the largest area saved by shifting the slabstones. If no area can be saved, just output zero.

Technical Specification

- The number of test cases is not more than 32.
- All the coordinates are 32-bit unsigned integers.
- A garden's area is not larger than the maximal 32-bit unsigned integer.
- The width and length of a slabstone are 32-bit unsigned integers. They should be larger than zero.
- The minimal horizontal spacing between any two slabstones is a 32-bit unsigned integer and should be greater than zero.
- The number of slabstones is from 4 to 100.

Sample Input 1

```
2
4 2
2 6 4 12
8 4 16 8
7 10 11 16
18 4 20 18
6 4
2 5 4 11
2 14 6 17
7 10 10 16
9 4 16 7
11 11 16 14
18 4 20 18
```

Sample Output 1

```
28
0
```

Hint

- Longest path on DAC for geometric objects
- The relative horizontal positions of rectangles (i.e., slabstones) need to be converted into a directed acyclic graph where a rectangle is treated as a vertex and the overlapping of two rectangle's vertical dimensions has to be modeled as an edge. Hence, a directed edge from rectangle X to rectangle Y means that rectangle X is positioned relaively to the left of rectangle Y and their vertical dimensions overlap. Associated with each vertex is the width of the underlying rectangle whereas associated with each edge is the minimal spacing between two rectangles. Once an drected acyclic graph is ready, a longest path algorithm can be aplied to find out the required X dimension of a garden. As a result, we can obtain the largest area that can be saved.

Problem PF
Baker's Dilemma

Time limit: 3 seconds
Memory limit: 1024 megabytes

Problem Description

A baker has N bakery orders from customers that he must fulfill, but he can only handle one order a day. For the i^{th} order, the baker needs to spend D_i ($1 \leq D_i \leq 1000$) consecutive days to complete it; however, for every day of delay, the baker must be fined S_i ($1 \leq S_i \leq 10000$). For example, if the baker receives four orders to make biscuits, the number of days required for each order is 3, 1, 2, 5, and the penalty for each day of delay is 4, 1000, 2, 5. If the baker's work order is 1 2 3 4, the penalty will be $4 \times 0 + 1000 \times 3 + 2 \times 4 + 5 \times 6 = 3038$, but if the work order is 2 1 3 4, the penalty will be $1000 \times 0 + 4 \times 1 + 2 \times 4 + 5 \times 6$. $0 + 4 \times 1 + 2 \times 4 + 5 \times 6 = 42$, so the latter penalty is less. Please write a program to help the baker to find out the sequence of work which has the least penalty.

Input Format

The first line of the input has a positive integer T representing the number of groups of data. Then, there are T groups of data. For each group, the first line has an integer N between 1 and 1000 representing the number of orders, followed by N lines, each with two integers separated by a space character, representing the number of days required for each order, D, and the penalty, S, for each day of delay, in that order.

Output Format

For each set of data, output the sequence of jobs with the smallest penalty on one line. Each job is represented by its number, separated by a blank character. If there is more than one set of answers, print the one with the smallest dictionary order. Note that each group of jobs is numbered starting with 1.

Technical Specification

- $1 \leq T \leq 1000$.
- $1 \leq N \leq 1000$.
- $1 \leq D_i \leq 1000, \forall 1 \leq i \leq N$.
- $1 \leq S_i \leq 10000, \forall 1 \leq i \leq N$.

Sample Input 1

```
2
4
3 4
1 1000
2 2
5 5
5
3 4
1 1000
8 8
2 2
5 6
```

Sample Output 1

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

Hint

First, we assume that, in a group, all orders are received at the same time (say time 0). The different permutation will have different penalty. We can sort these orders according to the product of finished time and penalty, that is, employing greedy policy.

Problem PG
A Packing Problem

Time limit: 1 second
Memory limit: 1024 megabytes

Problem Description

Jade is having fun playing around with items and boxes. She is wondering whether or not the items she has can be packed into the boxes. The scenario is described as follows.

Jade has m boxes $\mathcal{B} := \{ B_1, B_2, \ldots, B_m \}$ that are nonidentical to each other. Despite the nonidentical appearances of the boxes, they have a uniform size T. On the other hand, she has n items $\mathcal{I} := \{ I_1, I_2, \ldots, I_n \}$, where item I_j has size a_j that is either s_1 or s_2 for some s_1, s_2. In other words, $a_j \in \{s_1, s_2\}$ for all $1 \le j \le n$. Furthermore, it is known that $0 < s_1, s_2 \le T$ and $s_1, s_2 \notin (T/4,\ 3T/4)$. That is, either $s_i \le T/4$ or $s_i \ge 3T/4$ for $i \in \{1, 2\}$.

In Jade's rule of thumb for packing the items, not every item can be placed in every box. In particular, she has associated each item I_j with a subset $P_j \subseteq \mathcal{B}$ which denotes the set of boxes in which item I_j is allowed to be placed. To be precise, item I_j can be placed in box B_i if and only if $B_i \in P_j$. For convenience, for any $B_i \in \mathcal{B}$, also define $P_i^{-1} := \{ I_j \in \mathcal{I} :\ B_i \in P_j \}$ to be the set of items that are allowed to be placed in box B_i.

Under this scenario, Jade is wondering, whether or not it is possible to pack all the items in the boxes such that the total size of the items in each box is at most T. In other words, Jade is interested in knowing the existence of an assignment function $\sigma \colon \mathcal{I} \mapsto \mathcal{B}$ such that

- For any j with $1 \le j \le n$, $\sigma(I_j) = B_i$ only if $B_i \in P_j$.
- For any i with $1 \le i \le m$,
$$\sum_{I_j \in \sigma^{-1}(B_i)} a_j \ \le\ T.$$

In particular, if there exists no such assignment, then we say that the box size T is infeasible.

As a known fact, one way to certify the infeasibility of any $T \ge 0$ in the above scenario is to demonstrate a set of variables α_j and β_i for all $I_j \in \mathcal{I}$, $B_i \in \mathcal{B}$ such that the linear constraints listed below in LP-(t) with $t := T$ is satisfied.

In other words, there exists a set of valid estimations on the sizes of the items and boxes in the sense that, whenever the total size of an item combination C is at most the size of a box B_i, so is their total estimated sizes. Furthermore, the total estimated item sizes are strictly larger than the total estimated sizes of the boxes.

While knowing the above fact and feeling that the box size T may not be feasible for the items,

$$\sum_{1\le j\le n} \alpha_j \;>\; \sum_{1\le i\le m} \beta_i \qquad \text{LP-(t)}$$

$$\sum_{I_j\in C} \alpha_j \;\le\; \beta_i, \qquad \text{for any } 1\le i\le m \text{ and any } C\subseteq P_i^{-1} \text{ such that } \sum_{I_j\in C} a_i \le t.$$

Jade has a hard time finding such a set of variables. One day, Jade's best friend, Mike, dropped by and said

> The instance you give is very hard to pack! Why don't you try to prove that the box size $(\,1+3/4\,)\cdot T$ is feasible?

While it is easier to use boxes with enlarged size $(\,1+3/4\,)\cdot T$, Jade insists that each box must not contain more than one item with size strictly larger than $T/4$!

Provided the above information, your task in this problem is to help Jade compute either

1. An assignment σ for the enlarged box size $(\,1+3/4\,)\cdot T$ such that no box contains more than one item with size strictly larger than $T/4$, or
2. A set of α_j and β_i that satisfies LP-(t) with $t := T$ which shows that the box size T is infeasible for the items.

Input Format

The first line contains two integers n and m, which denote the number of items and the number of boxes. Then there are n lines, each of which describes the parameters for the n items. In particular, the j^{th} line starts with two integers a_j and p_j, the size of I_j and the cardinality of P_j. Then p_j integers follow, which denote the indexes of the boxes in P_j. The last line contains a single integer T.

You may assume that the indexes of the boxes are numbered from 1 to m.

Output Format

Depending on the resulting outcome, the output format is different.

If an assignment for box size $(\,1+3/4\,)\cdot T$ is found, then print in the first line the string "Assignment". Print in the second line n integers which denotes the indexes of the boxes in which the n items are placed. If there are multiple answers, print any of them.

On the other hand, if a set of α_j and β_i that satisfy LP-(t) with $t := T$ is found, then print in the first line the string "Proof". Print the values of α_j for all $1\le j\le n$ and β_i for all $1\le i\le m$ in two lines separately. If there are multiple solutions, print any of them that satisfies the following two conditions.

- α_j and β_i are non-negative integers for all $1 \leq j \leq n$ and $1 \leq i \leq m$.
- $\max\left(\max_{1\leq j\leq n} \alpha_j,\ \max_{1\leq i\leq m} \beta_i\right) \leq 3 \times 10^5$.

Technical Specification

- $1 \leq n \leq 100$, $1 \leq m \leq 100$.
- T is a multiple of 4. Furthermore, $1 \leq T \leq 10^5$.
- For both $i \in \{1, 2\}$, either $1 \leq s_i \leq T/4$ or $3T/4 \leq s_i \leq T$ must hold.
- $a_j \in \{s_1, s_2\}$ for all $1 \leq j \leq n$.

For the output, the following conditions must be satisfied.

- α_j and β_i are non-negative integers for all $1 \leq j \leq n$ and $1 \leq i \leq m$.
- $\max\left(\max_{1\leq j\leq n} \alpha_j,\ \max_{1\leq i\leq m} \beta_i\right) \leq 3 \times 10^5$.

Sample Input 1

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

Sample Output 1

```
Assignment
1 2 3
```

Sample Input 2

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

Sample Output 2

```
Proof
1 1 1 1
1 1 1
```

Hint

This problem is a variation of the minimum makespan scheduling problem, where the target makespan is fixed to T and the processing times of the jobs are either large, i.e., $\geq 3T/4$, or small, i.e., $\leq T/4$.

The goal of this problem is to produce either a scheduling for which the makespan is at most $(1 + 3/4)T$ or a proof showing the impossibility of scheduling to meet the target makespan T.

One default solution here is a local search algorithm but alternative methods for producing assignments and proofs may be possible and open to the contestants.

In the default solution, we pack the items one by one. When considering an item, use a local search algorithm with a fixed set of rules to arrange a slot for the item. When this process succeeds, we have a valid partial assignment for the items considered so far. When this process fails, we obtain a way to forming the proofs from the structure given by the local search algorithm.

For the remaining details, please refer to the following paper.

Klaus Jansen and Lars Rohwedder, "On the configuration-lp of the restricted assignment problem", SODA 2017.

Problem PH
Bank Deposit Challenge

Time limit: 1 second
Memory limit: 1024 megabytes

Problem Description

N banks offer their own one-year deposit activities. For bank i, (v_i, w_i) denotes the yearly deposit interest and deposit limit of its activity, respectively. When a depositor selects the activity of bank i, he/she must deposit an amount of money equal to w_i to cover a whole year. Note that a depositor can choose an activity only once. If a depositor has only cash C to participate in the deposit activities, what is the maximum deposit interest he/she can earn?

Input Format

The first line is an integer representing the cash C. The second line contains N integers, where the i-th integer indicates v_i. The third line also contains N integers, where the i-th integer indicates w_i. Note that there is a space between adjacent integers.

Output Format

An integer specifying the maximum deposit interest the depositor can earn. If no deposit interest can be earned, please output 0.

Technical Specification

- $1 \leq N \leq 100$.
- $1 \leq C \leq 1,000$. (unit: ten thousand dollars)
- $1 \leq v_i \leq 540$. (unit: one thousand dollars)
- $1 \leq w_i \leq 300$. (unit: ten thousand dollars)

Sample Input 1

```
100
10 5 15 7
20 30 50 70
```

Sample Output 1

```
30
```

Sample Input 2

```
500
72 2 2 10 12 10 10 17 13 15
120 10 5 20 25 100 80 300 50 100
```

Sample Output 2

```
144
```

Sample Input 3

```
5
1 2
10 20
```

Sample Output 3

```
0
```

Hint

- This problem can be considered a knapsack problem.
- Dynamic Programming.

Problem PI
The Pentagon Conjecture

Time limit: 10 seconds
Memory limit: 1024 megabytes

Problem Description

Counting the number $k(H, G)$ of copies of a designated subgraph H in a given undirected simple graph G efficiently can be challenging. For example, counting $k(C_3, G)$ for an n-node graph G may require $\Omega(n^{3-\delta})$ time for any constant $\delta > 0$ if your algorithm is "combinatorial." By C_k, we denote the simple cycle of length k. When the designated subgraph H becomes more complicated, the running time required to count $k(H, G)$ usually grows very quickly.

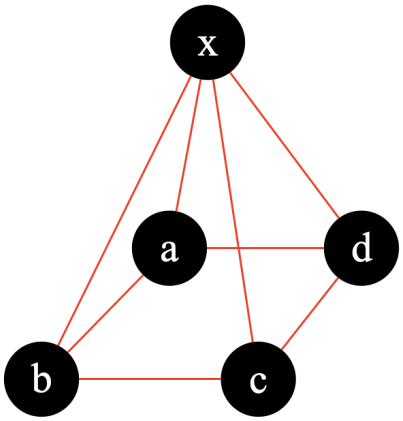

Figure 1: A pyramid P is an undirected simple graph consisting of 5 nodes and 8 edges, as depicted above. It is not hard to check that $k(C_3, P) = 4$, $k(C_4, P) = 5$, and $k(C_5, P) = 4$.

Bob is a researcher who needs to detect rare events in a given graph network. For example, in a random graph R of n nodes in which each edge is included with probability $1/2$, $k(C_3, R) = \binom{n}{3}/8$ in expectation. If he finds that $k(C_3, R)$ deviates significantly from the expectation, he may conclude that R is unlikely to be a random graph generated by the above process. Bob is studying the correlation between $k(C_3, G)$ and $k(C_5, G)$ for n-node graphs. There is a conjecture saying that, for any n-node undirected simple graph G, if

$$k(C_3, G) \geq \frac{5}{4}\left(n^{1.5} + \frac{n}{2}\right),$$

then $k(C_5, G) > 0$. Your task is to implement an efficient algorithm that verifies whether the conjecture is false.

Input Format

The first line contains exactly one integer t indicating the number of testcases. Then the testcases follow. For each testcase, the first line contains exactly two integers n and m, separated

by a space. n denotes the number of nodes in the given undirected simple graph G, and m denotes the number of C_3s given in the subsequent lines. The set of edges in G is exactly the union of the edges in the given C_3s. Each of the subsequent m lines contains exactly three distinct integers x, y, and z, separated by a space, where x, y, and z denote a distinct C_3 in G with end-nodes at x, y, and z. The nodes in G are numbered from 1 to n.

Output Format

For each testcase, output the 5 node identifiers of any pentagon in G in order on a line if $k(C_5, G) > 0$ (i.e. output a set of 5 nodes $x_1x_2x_3x_4x_5$ so that (x_i, x_{i+1}) is an edge for each $1 \leq i \leq 4$ and (x_1, x_5) also is an edge; if there are multiple solutions, output any of them); or otherwise output "-1" on a line.

Technical Specification

- $1 \leq t \leq 10$.
- $1 \leq n \leq 10^4$.
- $m = \lceil 5/4(n^{1.5} + (n/2)) \rceil$.

Sample Input 1

```
1
7 28
1 2 3
1 2 4
1 2 5
1 2 6
1 2 7
1 3 4
1 3 5
1 3 6
1 3 7
1 4 5
1 4 6
1 4 7
1 5 6
1 5 7
1 6 7
2 3 4
2 3 5
2 3 6
2 3 7
2 4 5
2 4 6
2 4 7
2 5 6
2 5 7
2 6 7
3 4 5
3 4 6
3 4 7
```

Sample Output 1

```
1 2 3 4 7
```

Hint

- It may help to check when the union of three triangles contains a pentagon.
- For graphs containing a sufficient number of triangles, it is a known fact that they must also contain at least one pentagon, as shown by Bollobás and Győri in 2007. The proof by Bollobás and Győri itself yields a deterministic $O(n^2)$-time algorithm. An alternative algorithm to solve this problem is based on the following provable fact. Any of such

graphs must have $O(\sqrt{n})$ nodes that are part of some pentagons. Consequently, one can select a random node, check in $O(m)$ time whether the node is part of any pentagon, and repeat this until a pentagon is found. This results in a randomized algorithm with expected running time of $O(n^2)$.

Problem PJ
Lead Time Estimation

Time limit: 1 second
Memory limit: 1024 megabytes

Problem Description

The lead time is critical for earning orders, and several issues would result in various lead times when preparing the products. For example, the processes switched between different working areas and the workload of each process. The production manager should accurately estimate the lead time for the target products when the order inquiry is received. Please develop a program to help the product manager estimate the lead time.

The lead time of an inquiry stands for the production time, including several jobs such as material preparation, manufacturing, quality checking, shipping, etc. The processing time of each position could be determined by an execution time while transferring the process from one job to the next requires a transmission time. For an inquiry, the production manager has three pieces of information about the target products.

- The number of jobs and transmissions, e.g. $|T^j|$ and $|T^t|$,
- the processing time of each job, where $T^j = \{t^j_0, t^j_1, t^j_2, \ldots, t^j_{|T^j|-1}\}$, and
- the transmission time between jobs, where $T^t = \{t^t_0, t^t_1, t^t_2, \ldots, t^t_{|T^t|-1}\}$.

The process may begin or end with several starting jobs. This situation could easily insert virtual starting and finishing jobs to simplify the problem. We can assume that the inquiry processes include one starting job and one finishing job.

Input Format

The input includes three parts: (1) the number of jobs and transmissions in the first row, (2) the job processing time in the second row, and (3) the transmission time in the remaining rows. The first input row consists of $|T^j|$ and $|T^t|$ with a space for the separation. The second input row should be T^j, and the comma separates each element in T^j. The transmission time information is revealed from row number three to $(2 + |T^t|)$. Each row of transmission time information involves three data: the source job, the destination job, and the transmission time.

Moreover, the test cases have some restrictions.

- There may be one transmission time for any pair of jobs at most.
- There may be multiple cases in a file.

Output Format

The total time of delivering the products that is denoted by z should be provided, and that means all jobs should be done in z. Also, the production manager is interested in the manufacturing process. If there is exactly one processing path that dominates the lead time, please output the job sequence of the processing path and the letter "M" in upper case otherwise. A comma separates each element in the output sequence. In other words, the output sequence should be like either "z,v_1,v_2,..." or "z,M", where v_1 and v_2 represent the jobs.

Technical Specification

Each inquiry includes precisely one entry and one exit. In each inquiry, at least one path will dominate the lead time, indicating no cyclical manufacturing processes. The boundaries of each variable are listed as follows.

- $2 \leq |T^j| \leq 50$.
- $1 \leq |T^t| \leq 100$.
- $1 \leq t_x^j, t_y^t \leq 50$.

Sample Input 1

```
8 11
2,7,2,6,5,1,2,7
6 7 2
0 1 4
0 2 2
1 3 6
1 4 5
1 5 3
2 4 1
3 6 2
3 7 9
4 5 2
5 7 2
```

Sample Output 1

```
41,0,1,3,7
```

Sample Input 2

```
6 7
10,8,9,10,11,12
0 1 1
1 2 2
1 3 3
1 4 4
2 5 5
3 5 6
4 5 7
6 7
10,8,9,11,11,12
0 1 1
1 2 2
1 3 4
1 4 4
2 5 5
3 5 7
4 5 7
```

Sample Output 2

```
53,0,1,4,5
53,M
```

Hint

- The lead time of first case is 53. There is only one processing path that dominates the lead time, e.g $0 \rightarrow 1 \rightarrow 4 \rightarrow 5$, so the program outputs the sequence "53,0,1,4,5".

- The lead time of second case is also 53, but there are two dominated processing paths: $0 \to 1 \to 3 \to 5$ and $0 \to 1 \to 4 \to 5$. Therefore, the output sequence is "53,M".

Problem PK
Chemical Storage

Time limit: 1 second
Memory limit: 1024 megabytes

Problem Description

International Chemical Producing Company (ICPC) is an international company that manufactures various chemicals. The company built several chemical rooms for storing chemicals. They made a short railroad to connect two chemical rooms for convenience to move the chemicals. The network of the chemical rooms and the railroads form a special tree graph in which all the nodes are within distance 2 of a central path. We label each node a number sequentially from 1. Fig. 2 gives an example of networks.

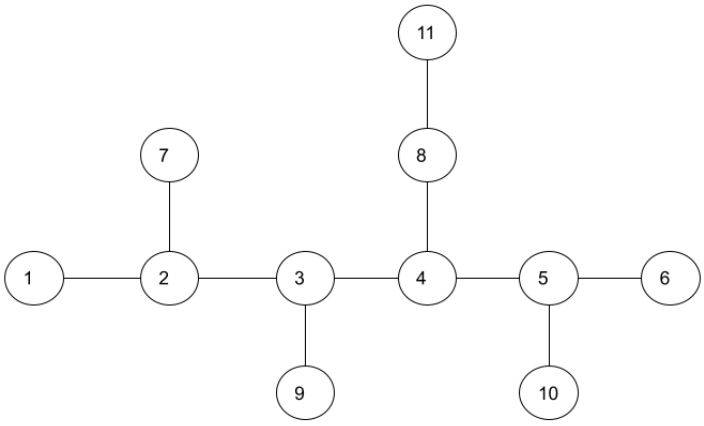

Figure 2: An example of networks with chemical rooms and railroads.

Each chemical product will be stored in a tank placed in a chemical room. Since the chemicals may leak into the air, the safety rule is that the chemicals cannot be placed in two adjacent rooms to avoid adverse chemical reactions between the chemicals. Fig. 3 gives two chemical placement network examples: (a) is safety, and (b) is unsafety.

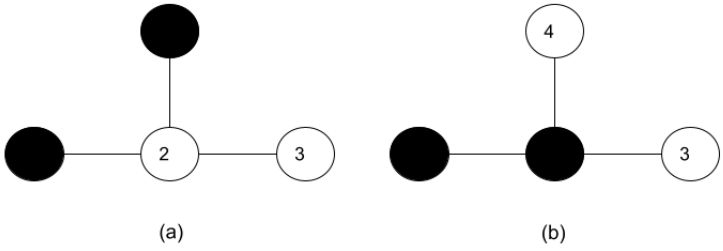

Figure 3: Two examples of chemical placement networks: (a) safety and (b) unsafety.

Sometimes, the workers must clean the tanks and move some chemicals to the other chemical rooms. Peter is the worker, and his manager will assign him a task with two safety placement networks: the source network T_s and the destination network T_d. A task is called $feasible$ if a possible strategy exists to move the chemicals from T_s to T_d following the safety rules; otherwise, it is called $infeasible$. Notice that we do not restrict chemicals to be placed in a specific room. If T_s and T_d are the same, the task is also treated as feasible. Fig. 4 and Fig. 5 show examples of feasible and infeasible tasks, respectively.

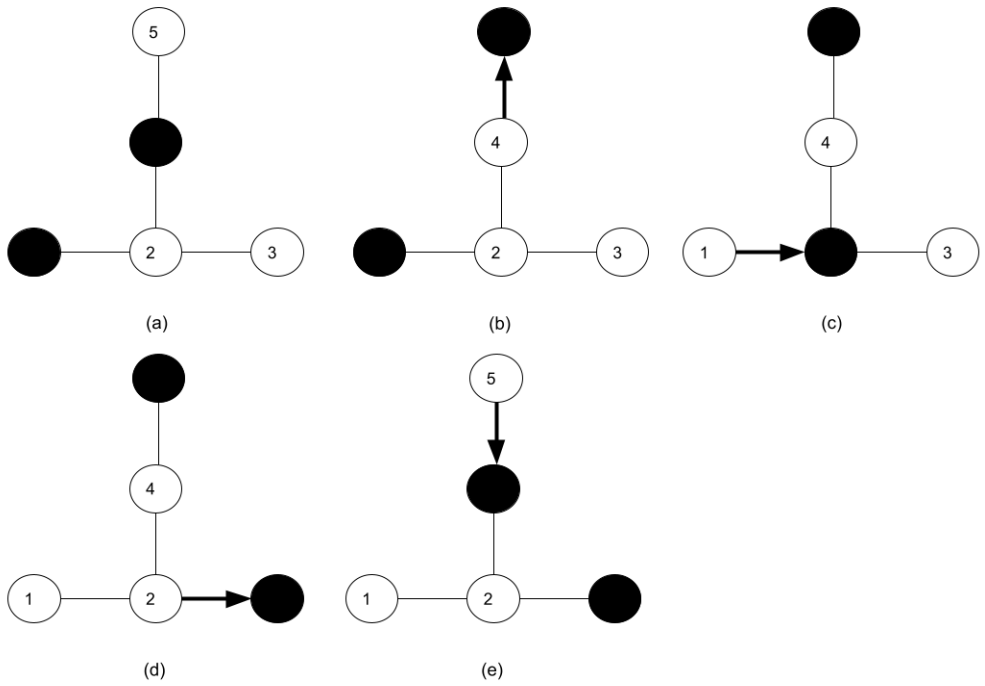

Figure 4: A feasible task: (a) the source network, (b) move the chemical from 4 to 5, (c) move the chemical from 1 to 2, (d) move the chemical from 2 to 3, (e) move the chemical from 5 to 4 to the destination network.

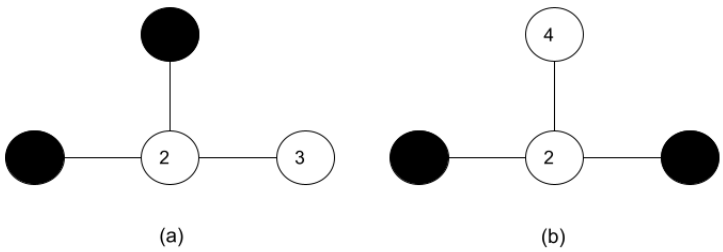

Figure 5: An infeasible task: (a) the source network, (b) the destination network.

Please write a program to help Peter judge whether a task is feasible or not.

Input Format

The first line contains exactly one integer t, which represents the number of test cases. Each test case below contains four lines. For each test case, the first line contains two integers n and m, where n represents the number of chemical rooms and m represents the number of chemicals; the second line contains $n-1$ integers $r_1, r_2, \cdots r_{n-1}$, which represents that room $i+1$ has a railroad connecting to room r_i for $1 \leq i \leq n-1$; the third line contains m integers s_j for $1 \leq j \leq m$, representing the room numbers in which chemicals are placed at the source network; and the fourth line contains m integers d_k for $1 \leq k \leq m$, representing the room numbers in which chemicals are placed at the destination network.

Output Format

Each test case outputs 1 if the task is feasible, otherwise outputs 0 in a line.

Technical Specification

- $5 \leq t \leq 10$.
- $1 \leq m \leq n \leq 10,000$.
- $1 \leq r_i < i+1$, $1 \leq i \leq n-1$.
- $1 \leq s_j \leq n$ and $s_p \neq s_q$ if $p \neq q$, $1 \leq d_j \leq n$ and $d_p \neq d_q$ if $p \neq q$.

Sample Input 1

```
6
4 2
1 2 2
1 4
3 4
4 2
1 2 2
1 4
1 4
5 2
1 2 2 4
1 4
3 4
11 4
1 2 3 4 5 2 4 3 5 8
1 6 7 8
1 3 5 8
11 4
1 2 3 4 5 2 4 3 5 8
1 3 5 8
7 8 9 10
10 5
1 2 3 4 2 3 3 6 4
2 4 7 8 9
1 5 6 7 8
```

Sample Output 1

```
0
1
1
0
1
1
```

Hint

This problem was inspired by the *sliding token problem* for trees introduced in [1], in which a linear time algorithm proposed.

A node with chemical is called *chemical node*. Let I_s and I_d be the (independent) sets of chemical nodes of T_s (source network) and T_d (destination network), respectively. For a set I of chemical nodes in a network, T, a chemical node in I is called *rigid* if it cannot move at all. We can apply the following rules to identify the rigid nodes.

1. If $|T| = 1$ with one chemical node u, then u is rigid.
2. Suppose $|T| \geq 2$. A chemical node u in the chemical node set I of T is rigid if and only if for every neighbor v of u in T, there exists a chemical node w in $I \cap T_w^v$ is rigid, where

T_w^v is the subtree containing v and w.

The algorithm is based on the following two key points.

1. If I_s and I_d have different placements of rigid nodes, then the task is unfeasible.
2. Otherwise, we obtain a forest by deleting the rigid nodes together with their neighbors. The answer is feasible as long as each tree in the forest contains the same number of chemical nodes in I_s and I_d.

Then, the following algorithm efficiently finds the rigid nodes iteratively.

1. Define and compute $degI(w) = |N(T, w) \cap I|$ for all vertices $w \in V(T)$, where $N(T, w)$ denotes the neighbors of w in T.
2. Define and compute $M = \{\ u \in I |$ there exists $w \in N(T, u)$ such that $degI(w) = 1\ \}$, that is, M is the set of chemical nodes that can be immediately slid.
3. Repeat the following steps (i)–(iii) until $M = \emptyset$.
   (i) Select an arbitrary node $u \in M$, and remove it from M and I.
   (ii) Update $degI(w) = degI(w) - 1$ for each neighbor $w \in N(T, u)$.
   (iii) If $degI(w)$ becomes one by the update (ii) above, then add the node $u \in N(T, w) \cap I$ into M.
4. Output I. Note that, since $M = \emptyset$, all chemical nodes in I are now (T, I)-rigid.

Reference

[1] E. D. Demaine, M. L. Demaine, E. Fox-Epstein, D. A. Hoang, T. Ito, H. Ono, Y. Otachi, R. Uehara, and T.akeshi Yamada, 'Linear-time algorithm for sliding tokens on trees', Theoretical Computer Science, Volume 600, Pages 132-142, 2015.

Almost blank page

Problem PL
Nine Never

Time limit: 3 seconds
Memory limit: 1024 megabytes

Problem Description

Once upon a time, there lived a great general called Tso who was well-known in arranging his soldiers into small groups to perform different tasks. If possible, Tso would divide his soldiers into as many groups as possible, so as to increase the flexibility. However, there was a little secret that only very few people knew: When Tso was young, a fortune-teller warned him that "9" would be a super unlucky number for him. So now, whenever Tso divided his soldiers into groups, he would make sure that the total number of soldiers, in any combination of these groups, would not be equal to 9.

For instance, when there are $N = 11$ soldiers, we can divide the soldiers into three groups, with 3, 4, 4 soldiers, respectively, so that any combination would not sum up to exactly 9 soldiers (the total number could only be 3, 4, 7, 8, or 11). Another way is to divide the soldiers into four groups, with 1, 3, 3, 4 soldiers, respectively, so that again, any combination would not sum up to exactly 9 soldiers (the total number could only be 1, 3, 4, 5, 6, 7, 8, 10, or 11). Since the latter way has more groups, it is a better choice than the former way.

In contrast, if we divide the soldiers into eleven groups, one for each group, then we would have even more groups; however, some combination (taking nine of these groups) would have exactly 9 soldiers in total, so this will not be an acceptable division for Tso.

As one of Tso's most reliable assistants, you are assigned the following task: Given a number N of soldiers where $N \neq 9$, help Tso to find the maximum number K of groups that can be formed.

Input Format

The input has only one line, which contains a single positive integer N.

Output Format

Let K denote the desired maximum number of groups. The output has only one line, which prints a non-negative integer X such that X is the remainder of K when divided by $10^9 + 7$. That is, $X \equiv K \pmod{10^9 + 7}$ with $0 \leq X < 10^9 + 7$.

Technical Specification

- $1 \leq N \leq 10^{15}$
- $N \neq 9$
- $0 \leq X < 10^9 + 7$

Sample Input 1	Sample Output 1
11	4

Sample Input 2	Sample Output 2
8	8

Sample Input 3	Sample Output 3
12345678901234	839407413

Hint

- When $N < 9$, the best way to divide is $1, 1, \ldots, 1$, so that $K = N$.
- When N is even and $N \geq 10$, we can show (or guess) that one of the best ways to form groups is $2, 2, 2, \ldots, 2$, so that $K = N/2$. See Tan et al. [Discrete Mathematics 2017, 340(6):1397–1404] (Corollary 2.2) for details.
- When N is odd, and N is sufficiently large, we can show that one of the best ways to form groups is $11, 2, 2, \ldots, 2$, so that there are $1 + (N - 11)/2$ groups formed. In particular, we have:

  Theorem: $N \geq 29$ is sufficient.
  Proof: Let K denote the value $1 + (N - 11)/2$. Suppose to the contrary that $N \geq 29$ is not sufficient, so that more than K groups can be formed. Since N is odd, one of the groups must be an odd number. Let z denote the smallest of such odd numbers.

  - If $z \geq 11$, then the number of groups is at most K, a contradiction.
  - If $z = 9$, then we have a group of size 9, a contradiction.
  - If $z = 7$, then we cannot have any 2 (or else we have a combination of 9). Thus, all remaining groups will have size at least 4, so that there are at most $\lfloor 1 + (N - 7)/4 \rfloor$ groups. This number is at most K when $N \geq 29$. Thus, a contradiction.
  - If $z = 5$, then we can have at most one 2. Apart from it, all remaining groups will have size at least 4. So, there are at most $\lfloor 2 + (N - 5)/4 \rfloor$ groups. This number is at most K when $N \geq 29$. Thus, a contradiction.
  - If $z = 3$, then we can have at most two groups of 2 and at most one other group

of 3. Apart from them, all remaining groups will have size at least 4. So, there are at most $\lfloor 4 + (N - 10)/4 \rfloor$ groups. This number is at most K when $N \geq 29$. Thus, a contradiction.

- If $z = 1$, then we separate the discussion into different cases.
  - Case 1: The partition has exactly one 1 or two 1s. In this case, we can have at most three groups of either size 2 or size 3. The remaining groups will have size at least 4. So, there are at most $\lfloor 5 + (N - 7)/4 \rfloor$ groups. This number is at most K when $N \geq 29$. Thus, a contradiction.
  - Case 2: The partition has exactly three 1s or four 1s. In this case, we can have at most two groups of either size 2, size 3, or size 4. The remaining groups will have size at least 5. So, there are at most $\lfloor 6 + (N - 7)/5 \rfloor$ groups. This number is at most K when $N \geq 29$. Thus, a contradiction.
  - Case 3: The partition has exactly five or six 1s. In this case, we can have at most one group of size 2 or size 3. However, we cannot have a group of size 4, 5, 6, 7, 8, 9, so that the remaining groups will have size at least 10. Thus, there are at most $\lfloor 7 + (N - 7)/10 \rfloor$ groups. This number is at most K when $N \geq 29$. Thus, a contradiction.
  - Case 4: The partition has exactly seven or eight 1s. In this case, we cannot have a group of size 2, 3, 4, 5, 6, 7, 8, 9 so that the remaining groups will have size at least 10. Thus, there are at most $\lfloor 8 + (N - 7)/10 \rfloor$ groups. This number is at most K when $N \geq 29$. Thus, a contradiction.

- For any odd $N < 29$ (and $N \neq 9$), we search by brute-force all the possible partitions of N, and select one that contains the maximum number of parts, and at the same time avoids a subset sum of 9. A possible speed up is that we will never need a number z greater than 19, for we can make more parts by replacing z with 10 and $z - 10$.

Almost blank page

Problem PM
Task scheduler

Time limit: 1 second
Memory limit: 1024 megabytes

Problem Description

A simple task scheduler schedules tasks for an embedded system. Each task has a task ID, priority value, and a function pointer to its corresponding callback function. The task ID is represented by an integer ranging from 0 to 99. The priority value is represented by an integer ranging from 0 to 255. The callback function is responsible for printing the associated task ID.

Tasks are added statically by developers. After N tasks are added, the task scheduler starts to schedule the tasks. The task with the smallest priority value will be scheduled first. If multiple tasks have the same priority value, the task added to the system earliest will be scheduled first. Now, let's see what the system prints.

Input Format

The first line contains an integer T, representing the number of test cases. Each test case below includes three lines.

For each test case, the first line is an integer $N \leq 100$, representing the number of the following two lines of integers. The second line contains N integers, each integer representing the ID of a task. The integer closer to the beginning of the second line indicates that its corresponding task was added to the system earlier. The third line also contains N integers, representing the priority value. The $i-th$ number in the third line is the priority value of the task represented by the $i-th$ number in the second line.

Output Format

Each test case outputs N integers, separated by a space. These integers are the IDs of tasks, and the order in which they are printed corresponds to the order in which they are scheduled.

Technical Specification

- $1 \leq T \leq 10$.
- $1 \leq N \leq 100$.
- $0 \leq taskID \leq 99$.
- $0 \leq priority\ value \leq 255$.

Sample Input 1

```
4
1
0
5
3
0 1 2
1 0 0
4
0 1 2 3
6 4 2 2
8
0 1 2 3 4 5 6 7
10 12 12 3 1 3 1 1
```

Sample Output 1

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

Hint

- Create a linked list.
- Add a task to the linked list at an appropriate position.
- Traverse the existing nodes of the linked list. If the priority value of the visited node is not higher than that of the task you are adding, continue to the next node. Otherwise, insert the current task before the currently visited node.
- Once all tasks are added to the linked list, traverse the linked list once again.