Problem A
Pair Game
Number of Test Cases: 14
Excution Time Limit: 10 seconds

NCPC created a new solitary game called the Pair Game. The game starts with having $2n$ cards on the table. The $2n$ cards each has a number on it. The number is between 1 and n and each number appears on exactly two cards. The cards are dealt onto the table face down forming a row of $2n$ cards. To play the game, the player continuously picks two neighboring cards from the table and turn them over. If the two cards have the same number, both cards are removed from the table; otherwise, two cards swap positions and turned face down again. Taking two cards off the table or swapping two cards on the table is considered a move. Player must try to clear all cards on the table within a given number of moves.

The game is won if all cards are removed from the table before using up the allowed number of moves. NCPC wants to ensure the game is always winnable, but also not too easy to win, so the number of moves allowed will be set to some multiple of the least (optimal) number of moves necessary to remove all cards. Given the configuration of the $2n$ cards, please find the least number of moves required to win the game so the proper limit can be set.

For example, if there are initially 6 cards with numbers 1 2 3 1 2 3 on the table. The following two sequences of six moves show that with three swaps and three remove moves, all cards can be removed from the table.

1 2 3 1 2 3 → 2 1 3 1 2 3 → 2 3 1 1 2 3 → 2 3 2 3 → 3 2 2 3 → 3 3 → *no more card*

1 2 3 1 2 3 → 2 1 3 1 2 3 → 2 1 1 3 2 3→ 2 3 2 3 → 2 3 3 2 → 2 2 → *no more card*

There are no possible sequence of fewer moves to remove all cards. Therefore, 6 is the least number of moves required to win the game.

Input File Format

There are more than one test cases in the input file. The first line contains a single integer indicating the number of test cases to follow. Each test case starts with a positive integer n ($n \leq 10,000$), followed by $2n$ positive integers denoting the numbers on the $2n$ cards that are on the table.

Output Format

For each test case, print the least number of swaps needed to clear all cards on the table on one line.

Sample Input

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

Output for the Sample Input

```
6
3
```

Solution

Mathematical Model of the Problem

Two observations:
1. Number of Remove moves = n, because every pair need to be removed.
2. In a subsequence A, n_i, n_j, n_k, A, assume $n_i \neq n_j \neq n_k$, then 3 swaps are needed to remove A. After swapping, relative order of n_i, n_j, n_k is preserved.

Algorithm and Time Complexity of the Problem

By summing the number of distinct numbers $(n_i, n_j, ...)$ between pair of numbers for all numbers, the number of swaps is the sum/2, as each swap will be counted twice. This algorithm takes $O(n^2)$ time.

With segment tree/Interval tree, it takes $O(n)$ to build the tree. Once built, it takes $O(logn)$ time to search and update the tree. To go through all n pairs, the total time is therefore bounded by $O(nlogn)$.

Problem B

Minesweeper

Number of Test Cases: 100
Excution Time Limit: 2 seconds

Problem

A minesweeper sails the ocean and detonates the mines. The minesweeper starts at the origin $(0, 0)$ and sails according to a sequence of n commands. The first part of the command is the direction – 0 for the east, 1 for the north, 2 for the west, and 3 for the south. The second part of a command is the distance. For example, if the first command is (0, 100), the minesweeper will sail from $(0, 0)$ to $(100, 0)$.

There are m mines in the ocean, each in the position (x_i, y_i), for i between 1 and m. If the minesweeper is within a *detection distance* d from a mine, then it will detonate the mine. For simplicity, we define the distance between two points as the maximum difference in the x and y coordinates of the two points. For example, if d is 3, the current position of the minesweeper is $(9, 2)$ and there is a mine in $(12, 0)$, the minesweeper will detonate it because the distance between them is $\max(|9 - 12|, |2 - 0|) = 3$, which is not greater than $d = 3$. However, if the mine is at location $(12, 6)$, the minesweeper cannot denote it because the distance between them is now $\max(|9 - 12|, |2 - 6|) = 4$, which is greater than $d = 3$.

Now we define the task. We are given the detection distance d, a sequence of n commands, and the locations of m mines, and we need to determine the number of mines that the minesweeper will detonate.

Input File Format

There is more than one test case in the input file. Each test case starts with the positive mine detonation distance d, where d is no more than 100. The input then has a positive integer n, the number of commands to the minesweeper, where n is no more than 1000. The input then has n pairs of integers. The first integer in a pair is the direction in which the minesweeper will go and the second integer is the distance it will go. The direction is from 0 to 3 and the distance is a positive integer no more than 10000. Then there is a positive integer m, the number of mines, followed by m pairs of integers that indicate the coordinates x and y of a mine. m is no more than 10000000. Note that there could be multiple mines at the same location. The coordinates of the mines and the minesweeper are always between -2147483648 and 2147483647. The last line contains only an integer 0 to indicate the end of the input.

Output Format

For each test case, print the number of mines that the minesweeper detonates on one line.

Sample Input

```
1
4
0 10 1 10 2 10 3 10
7
0 0 -1 -1 11 -1 9 9 1 9 5 5 -1 -1
1
4
0 10 2 10 0 10 2 10
7
0 0 -1 -1 11 -1 9 9 1 9 5 5 -1 -1
0
```

Output for the Sample Input

```
6
4
```

Solution

First, we realize that the course of the minesweeper is horizontal or vertical. As a result, we can use a set of rectangles to store the area the minesweeper can detect the mines. We then sort the left-lower corners, the right-upper corners of these rectangles, and the positions of the mines according to the x coordinates, and place them into an array P. It is easy to see that the sorting to build P takes $O((m+n)\log(m+n))$ time.

We then process these positions in P from left to right. If we encounter a left-lower corner. we add a segment into a segment tree S. If we encounter a right-upper corner. we remove the corresponding segment from S. Finally, if we encounter a mine position then we check if it is with any segment within S. It is easy to see that it takes $O(\log(m+n))$ time to process a point, and the number of points is $O(m+n)$. As a result, it also takes $O((m+n)\log(m+n))$ time to process the points in P. Therefore the total execution time is $O((m+n)\log(m+n))$.

Problem C
Anchor Chains
Number of Test Cases: 10
Excution Time Limit: 3 seconds

Peter is a biologist. He estimates the similarity of two genome sequences P and Q as follows. If a subsequence $P[a, b]$ of P, where $a \leq b$, and a subsequence $Q[c, d]$ of Q, where $c \leq d$, are identified to be closely related, we say that there is an *anchor* at (a, b, c, d) and give a *weight* to it. (Intuitively, anchors indicate *small matches*.) We say that an anchor (a, b, c, d) *precedes* another anchor (a', b', c', d') if $b < a'$ and $d < c'$. For example, anchor $(2, 5, 1, 9)$ precedes anchor $(9, 12, 10, 11)$. Let M be the set of the anchors of P and Q. Let $C = (A_1, A_2, ..., A_p)$ be a sequence of the anchors in a subset of M, where $p \geq 1$. We call C an *anchor chain* if A_i precedes A_{i+1} for $1 \leq i < p$ and when C is an anchor chain we define its *score* to be

$$Score(C) = \sum_{1 \leq i \leq p} w(A_i),$$

where $w(A_i)$ is the weight of A_i. For example, $C = ((2, 5, 1, 9), (9, 12, 10, 11), (14, 15, 16, 19))$ is an anchor chain and its score is $w((2, 5, 1, 9)) + w((9, 12, 10, 11)) + w((14, 15, 16, 19))$. The *similarity* of P and Q, denoted by $Similar(P, Q)$, is the maximum score of an anchor chain of P and Q. That is,

$$Similar(P, Q) = \max\{Score(C) \mid C \text{ is an anchor chain of } P \text{ and } Q\}.$$

Please write a program to help Peter.

Technical Specification

- The number of test cases is at most 10.
- The number of anchors is at most 10^6.
- For each anchor (a, b, c, d) with weight w, we have $a \leq b$, $c \leq d$, and $1 \leq a, b, c, d, w \leq 10^9$.

Input File Format

The first line contains an integer indicating the number of test cases. Each test case begins with a line containing an integer n, indicating the number of anchors. Each of the next n lines gives five integers a, b, c, d, w, indicating that there is an anchor (a, b, c, d) with weight w.

Output Format

For each test case, print $Similar(P, Q)$ in a line.

Sample Input

```
2
5
9 12 10 11 5
2 5 1 9 3
1 1 1 1 2
12 13 12 14 2
14 15 16 19 1
3
4 8 2 6 3
5 10 9 12 5
1 3 5 10 4
```

Output for the Sample Input

```
9
5
```

Solution

Algorithm and Time Complexity of the Problem

A DP algorithm is as follows. Let $M[1], M[2], \ldots, M[n]$ be the anchors in M. Presort the anchors in M such that the anchors are arranged in non-decreasing order of $M[i].a$. Let $C[i]$ be the maximum score of a chain ending at $M[i]$. Then, our problem is to compute $\max\{C[1], C[2], \ldots, C[n]\}$. Clearly, $C[1] = w(M[1])$ and for $i > 1$, we have

$$C[i] = w(M[i]) + \max\{C[j] \mid 1 \leq j < i, M[j] \text{ precedes } M[i]\}.$$

A naive implementation takes $O(n^2)$ time. An $O(n \log n)$-time implementation is described below. An RMQ (range minimum query) data structure (such as a segment tree) D is maintained for the computation. Let $E[1, 2n]$ be the sorted sequence of all the endpoints $M[i].a$ and $M[i].b$. We scan the endpoints in E and do the following.

Case 1: The current endpoint is an $M[i].a$. We query D to find $m = \max\{C[j] \mid C[j] \text{ in } D \text{ and } M[j].d < M[i].c\}$ and then compute $C[i] = w[i] + m$; and

Case 2: The current endpoint is an $M[i].b$. We store $C[i]$ into D with $M[i].d$ as its key.

Note that if several $C[j]$ in D have the same d, only keep the largest one.

Problem D
The 231 Pattern
Number of Test Cases: 20
Excution Time Limit: 1 second

Bob has a stock, whose price at time $i \in \{1, 2, \ldots, n\}$ is $A[i]$. Usually, he feels fine with rises and falls of the stock price. However, if the stock price rises from a value x and then falls to below x, then he will be upset. In detail, he will be upset if there exist indices i, j, $k \in \{1, 2, \ldots, n\}$ such that

- $i < j < k$,
- $A[i] < A[j]$, and
- $A[k] < A[i]$.

So, for example, if $A[i] = 2$, $A[j] = 3$ and $A[k] = 1$ for some $1 \leq i < j < k \leq n$, then Bob will be upset.

Please determine whether Bob will be upset. If the answer is affirmative, please also output indices i, j, $k \in \{1, 2, \ldots, n\}$ satisfying $i < j < k$ and $A[k] < A[i] < A[j]$.
Hint: There is a beautiful and asymptotically optimal algorithm using a stack. Alternatively, assuming that you have a data structure (e.g., a balanced binary search tree) T with keys $A[1]$, $A[2]$, ..., $A[i-1]$, you may want to find the maximum key in T that is less than $A[i]$. Then you may want to insert $A[i]$ into T and increase i. Note that the minimum among $A[i+1]$, $A[i+2]$, ..., $A[n]$ can be calculated in the order of decreasing i.

Input File Format

The input begins with the number of test cases, which is at most 20. Each test case is given by n, $A[1]$, $A[2]$, ..., $A[n]$, in that order. Any two consecutive integers are separated by whitespace character(s). It is known that $1 \leq n \leq 10^5$ and that $1 \leq A[i] \leq 10^8$ for all $1 \leq i \leq n$.

Output Format

For each test case:

- If Bob will not be upset, then please output "no".
- Otherwise, please output "yes", i, j and k (in that order) satisfying $1 \leq i < j < k \leq n$ and $A[k] < A[i] < A[j]$. If there are two or more correct solutions, please just output one of them arbitrarily.

Sample Input

```
4
5 20 3 30 80 10
5 1 2 3 4 5
5 4 3 6 7 8
7 2 15 4 6 40 7 80
```

Output for the Sample Input

```
yes 3 4 5
no
no
yes 2 5 6
```

Solution

Mathematical Model of the Problem

Determine whether an array contains "231."

Algorithm and Time Complexity of the Problem

Consider the following algorithm, called stack sorting:

```
1: Initialize an empty stack S;
2: for i = 1 to n do
3:    while S is nonempty and A[i] is greater than the top of S do
4:       Pop the top of S to the output;
5:    end while
6:    Push A[i] onto S;
7: end for
8: while S is nonempty do
9:    Pop the top of S to the output;
10: end while
```

Now pick $1 \le i < j \le n$ arbitrarily. Observe that if $A[i] < A[j]$, then $A[i]$ gets output before $A[j]$: If $A[i]$ remains to be in S when $A[j]$ arrives, then $A[j]$ will force $A[i]$ out of S. Assuming $A[i] > A[j]$,

(i) If $A[\ell] > A[i]$ for some $i < \ell < j$, then the arrival of $A[\ell]$ will force the output of $A[i]$ (if $A[i]$ is yet to be output), implying that stack sorting outputs $A[i]$ before $A[j]$.

(ii) If $A[\ell] \le A[i]$ for all $i < \ell < j$, then $A[i]$ will remain to be in S when $A[j]$ arrives, implying that $A[j]$ gets output before $A[i]$.

In summary,

- with a 231 pattern, item (i) will happen for some i, j and ℓ, forbidding stack sorting to sort correctly (into the ascending order).
- without a 231 pattern, item (ii) will happen for all $i < j$ such that $A[i] > A[j]$, implying stack sorting to sort correctly (into the ascending order). Note that for all $i < j$ such that $A[i] < A[j]$, $A[i]$ does get output before $A[j]$, as observed near the beginning of this subsection.

So the solution is to see whether stack sorting sorts correctly. This takes $O(n)$ time.

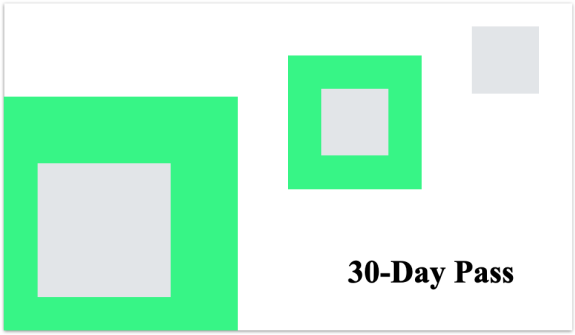
Source: I found the thing on wikipedia, which says that the real source is Knuth's *The Art of Computer Programming*.

Problem E

30-Day Pass

Number of Test Cases: 100
Excution Time Limit: 1 second

You are considering whether to purchase a 30-day pass for commuting in Taipei. The 30-day pass costs 1200 NT dollars and provides unlimited rides on all public transportation for 30 consecutive days. If you pay as you go, it will cost you x NT dollars per weekday (Monday–Friday) and y NT dollars per day on weekends (Saturday and Sunday). We are interested in determining if using the 30-day pass will save you money over the next 30 days.



You are designing a web page where users can input the values of x and y, as well as the number a of weekdays and the number b of days on weekends over the next 30 days. The goal is to determine whether buying the 30-day pass is strictly cheaper than paying as you go.

Note that the values of x, y, a, and b are input by users, so they may contain some errors. Each of these four numbers is a non-negative integer with **at most 80 digits**. You may assume that no input number will start with zero unless the number is zero. Clearly, a necessary condition is $a + b = 30$. If the combination of a weekdays and b weekend days is impossible for any 30-day period, your program should be able to detect this and report an error.

Input File Format

There are multiple test cases in the input file. The first line contains exactly one integer t ($t \leq 100$), indicating the number of test cases. Each test case consists of the four numbers x, y, a, b on a line.

Output Format

For each test case, if it is impossible to have a weekdays and b days on weekends for any 30-day period, output "Wrong Input". Otherwise, determine whether buying the 30-day pass is strictly cheaper than paying as you go. If the 30-day pass is strictly cheaper, output "30-Day Pass". If paying as you go is cheaper, output "Pay As You Go". If the costs are the same, output "Equally Good".

Sample Input

```
4
30 45 15 15
30 67 22 8
30 68 22 8
40 40 22 8
```

Output for the Sample Input

```
Wrong Input
Pay As You Go
30-Day Pass
Equally Good
```

Solution

Mathematical Model of the Problem

Algorithm and Time Complexity of the Problem

Problem F

Stats

Number of Test Cases: 20
Excution Time Limit: 3 seconds

String matching is a famous problem. In this problem, you are given a text string T and a pattern string P, and the goal is to find *all* places where P happens in T. For example, let $T =$ `aabb` and $P =$ `ab`. We can see that P happens at position 2 of T, namely the fragment with an underline in `aabb`. Notice that the position of a string starts from 1, and we use the *left position* to describe a match of P in T. A pattern may happen at several places in a text. For example, pattern `a` happens at positions 1 and 2 in text `aabb`. Therefore, given T and P, we can list the positions of all matches of P against T, order them from left to right, to get a sequence. This sequence is called the *occurrence distribution* of P relative to T, and we use $occ(T, P)$ to denote it. For example, $occ(\texttt{aabb}, \texttt{a}) = [1, 2]$.

You are a string statistician. You want to collect some information about the occurrence distributions of a text string. Take text $T =$ `aabab` as an example. You draw the following table to analyze the occurrence distributions:

pattern	distribution
`a`	1, 2, 4
`b`	3, 5
`aa`	1
`ab`	2, 4
`ba`	3
`aba`	2
`aaa`	nowhere

While building this table, you observed an interesting fact that different patterns may get the same occurrence distribution. For example, $occ(T, \texttt{aab}) = [1]$, which is equal to $occ(T, \texttt{aa})$ in the 3rd row. Similarly, $occ(T, \texttt{abab}) = [2]$, which is equal to $occ(T, \texttt{aba})$ in the 6th row. You want to find as many distinct occurrence distributions as possible for a text. Notice that we do not consider an empty string as a legal pattern. Therefore, there are totally 7 different occurrence distributions for the text `aabab`. Please write a program to count the total number of different occurrence distributions for a given string.

Input File Format

The first line contains an integer k, the total number of test cases, $k \leq 20$. In the next k lines, each line gives you one text string that has at least one character. A text string is composed of lowercase English letters. The total length of all text strings in the input file is at most 10^6.

Output Format

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

Sample Input

```
5
aaa
bbbb
abab
aaab
ncpcistttoughfffbutcccreative
```

Output for the Sample Input

```
4
5
5
7
38
```

Solution

Mathematical Model of the Problem

Let $T = T[1, n]$ be a text string. In order to get the occurrence distributions of T, we only need to consider factors (substrings) of T as the candidates of patterns. Let P be one of them. Then you can apply any string matching algorithm in order to get the occurrence distribution $occ(T, P)$. Since there are at most $O(n^2)$ factors of T, there can be at most $O(n^2)$ different occurrence distributions. However, this approach takes too much time.

Algorithm and Time Complexity of the Problem

Notice that if two patterns P and Q have the same occurrence distribution, then the shorter one must be a proper prefix of the longer one, since they can be overlapped and aligned at the same starting position. Without loss of generality, assume P is shorter than Q. We use the notation $P \lhd Q$ to signify that P is a proper prefix of Q. Observe that if there is a factor R such that $P \lhd R \lhd Q$ and $occ(T, P) = occ(T, Q)$, then $occ(T, R)$ is also identical. This means that if we collect all patterns that have the same occurrence distribution in the ascending way, they form a contiguously incremental fragments (i.e., the latter one has one more character in the end than the immediately previous one). If we construct the suffix tree of T, those patterns with the same occurrence distribution actually correspond to an edge in the suffix tree. (Actually, they are paths start from the root and end at the same edge.) Accordingly, their same occurrence distribution corresponds to the set of labels of suffixes below this edge. Therefore, the total number of different occurrence distributions is equal to the total number of edges in the suffix tree, plus one with no match. The construction of a suffix tree takes time $O(n \lg \sigma)$ where n is the length of T and σ is the alphabet size. In our case, σ is 26. A simpler way to solve this problem is to construct a suffix automaton DAWG (Directed Acyclic Word Graph) that works similarly as a suffix tree. (The code in C++ is less than 60 lines.) The time complexity is still $O(n \lg \sigma)$.

Problem G
Secrets of Polynomial Modulo Prime Power
Number of Test Cases: 20
Excution Time Limit: 1 second

Bob is a number wizard, always curious about how numbers behave. One day, he stumbles upon polynomials with integral coefficients. He is fascinated by the idea of finding their roots, the special numbers that made them equal to zero, and thinks about using the roots to hide secrets for communication. For simple polynomials with a small modulo, finding roots can be done by searching through all possibilities. But when the polynomials become bigger and the modulo is unknown, things get tricky. Bob realizes that finding roots isn't just about solving equations; it is about understanding the deeper patterns within the numbers.

For example, Bob observes the following. For a polynomial $f(x) = x^2 + x + 7$ and $p = 3$, it is clear that 1 is the unique root of $f(x) \equiv 0 \pmod{p}$ in the domain $\{0, 1, 2\}$. While, there are three roots for $f(x) \equiv 0 \pmod{p^2}$, i.e., 1, 4 and 7 from the domain $\{0, \ldots, 8\}$. It appears that for different powers of p, a polynomial may have different roots. This motivates Bob to work on how to find the roots of polynomials.

Formally, Bob is interested in the single-variable polynomial $f(x)$ modulo p^ℓ, where p is a prime number, ℓ is a positive integer, and all the coefficients are non-negative integers. He wants to find all possible $a \in \mathbf{Z}_{p^\ell} := \{0, 1, \ldots, p^\ell - 1\}$ such that $f(a)$ is a multiple of p^ℓ, i.e., $f(a) \equiv 0 \pmod{p^\ell}$.

Bob finds a way to extend a root $a \in \mathbf{Z}_{p^j}$, satisfying $f(a) \equiv 0 \pmod{p^j}$, to higher prime power with the following property:

$$f(a + tp^j) \equiv f(a) + f'(a)tp^j \pmod{p^{j+1}},$$

for any non-negative integer t, where f' is the derivative of f. Further, Bob observes that $f(a + tp^j) \equiv 0 \pmod{p^{j+1}}$ if and only if $f(a) + f'(a)tp^j \equiv 0 \pmod{p^{j+1}}$, which implies

$$\frac{f(a)}{p^j} + f'(a)t \equiv 0 \pmod{p}.$$

Based on the above, we can determine whether a can be extended to root(s) of higher power or not, by finding possible t that satisfies the above congruent equation. That is, for any t satisfying the above congruent equation, $a+tp^j$ is a root of $f(x) \equiv 0 \pmod{p^{j+1}}$. Note that, for any polynomial $q(x)$, if $q(a) \equiv 0 \pmod{p^\ell}$, then $q(a) \equiv 0 \pmod{p^{\ell-1}}$, but the other direction may not be true.

Your task is to write an efficient program to help Bob find all the possible distinct roots of an integral polynomial modulo some prime power, where all the coefficients are non-negative. If there is no root, then simply output -1.

Input File Format

The first line in the input file has a positive integer T (≤ 10), which indicates there are T test cases. Then, for each case, the first line has two positive integers, p ($< 10,000$) and ℓ (< 40) with $p^\ell < 2^{64}$, which indicates, respectively, a prime number and the power. In the following line, there is a sequence of non-negative integers: $d, c_d, \cdots, c_1, c_0$ indicating a degree d ($0 < d < 600$) polynomial, i.e., $c_d x^d + \cdots + c_1 x + c_0$, where $c_d \neq 0$ and $0 \leq c_i < 10^{12}$. All the input is guaranteed to have no more than 1000 distinct roots.

Output Format

For each test case, print, in a line, the roots of the corresponding polynomial under the prime power in **increasing order**. All the roots should be in $\mathbf{Z}_{p^\ell}$, for the corresponding p and ℓ. If the polynomial does not have root in $\mathbf{Z}_{p^\ell}$, then output -1.

Sample Input

```
4
2 4
1 1 4
3 4
2 1 1 23
3 7
3 1 2 1 6
7 3
2 1 1 47
```

Output for the Sample Input

```
12
-1
840
99 243
```

Solution

Mathematical Model of the Problem

Given a degree d polynomial $f(x) = \sum_{i=0}^{d} c_i x^i$ and a prime power p^ℓ, find the roots of $f(x) \equiv 0 \pmod{p^\ell}$ in $\mathbf{Z}_{p^\ell}$.

Algorithm and Time Complexity of the Problem

We can start by exhaustively searching the root(s) for $f(x) \equiv 0 \pmod{p}$ with $O(dp)$ time. Then we can *lift* the roots to higher power when possible. This can be done in $O(dp + \ell M(d+p))$ time, where M is the maximum number of root under a modulo.

Let $a \in \mathbf{Z}_{p^j}$ satisfy $f(a) \equiv 0 \pmod{p^j}$. We want to extend a to $a + t \cdot p^j$ such that $f(a + tp^j) \equiv 0 \pmod{p^{j+1}}$. By Taylor's expansion we can express

$$f(a + tp^j) = f(a) + f'(a)tp^j + f''(a)t^2p^{2j}/2! + \cdots .$$

Thus,

$$f(a + tp^j) \equiv f(a) + f'(a)tp^j \pmod{p^{j+1}}.$$

Note $f(a + tp^j) \equiv 0 \pmod{p^{j+1}}$ iff $f(a) + f'(a)tp^j \equiv 0 \pmod{p^{j+1}}$. So

$$\frac{f(a)}{p^j} + f'(a)t \equiv 0 \pmod{p}.$$

Then we can determine if there exists t satisfying the above congruent equation. Consider the following cases:

1. Case $f'(a) \not\equiv 0 \pmod{p}$: Apply the extended-gcd algorithm to find the inverse of $f'(a)$ and then a can be extended to $a + tp^j$ as a root of f modulo p^{j+1} with $t = -\frac{f(a)}{p^j}(f'(a))^{-1}$.

2. Case $f'(a) \equiv 0 \pmod{p}$: We have $f(a + tp^j) \equiv f(a) \pmod{p^{j+1}}$. Now there are 2 subcases: (1) if $f(a) \equiv 0 \pmod{p^{j+1}}$, then $f(a + tp^j) \equiv 0 \pmod{p^{j+1}}$, $t = 0, 1, \cdots, p-1$, i.e., a can be extended to p roots of f modulo p^{j+1}. (2) if $f(a) \not\equiv 0 \pmod{p^{j+1}}$, then a cannot be extended to $a + tp^j$ as a root of f modulo p^{j+1}.

Based on the above and the assumption $p < 10000$, we can start by searching all the root(s) of $f(x) \equiv 0 \pmod{p}$. Then extend the root(s) inductively on the power.

Problem H
Hidden Tree Reconstruction
Number of Test Cases: 1
Excution Time Limit: 1 second

Alice has an edge-weighted rooted tree $T = (V, E)$ where each edge $e \in E$ is associated with a positive weight $w_e \in \mathbf{R}_{>0}$. Alice told Bob that the tree she possesses has two very special properties that

1. The distances between the root vertex r and any leaf node are the same, i.e.,

   $$d_T(u, r) \;=\; d_T(v, r)$$

   for any two leaf nodes u, v in T, and

2. The degree of the root node is at least 2. For any non-root internal node, the degree is at least 3.

"I will tell you more information about the tree," said Alice, who then wrote down the distances between some pairs of leaf nodes in the tree.

"As a computer scientist, " Alice continued,
"I will go out a date with you, if you can find out the secret tree I have."

.....

Please write a program to help Bob find out the secret tree.

Input File Format

The first line of the input contains two integers n and m, where n is the number of leaf nodes in Alice's tree and m is the number of clues (the pairwise distances) Alice has written. Then there are m lines, where the i-th line contains three integers u_i, v_i, and ℓ_i specifying that the distance between leaf u_i and leaf v_i in the tree is ℓ_i.

You may assume that

- $2 \leq n \leq 10^5$ and $1 \leq m \leq 10^5$.
- The leaf nodes are indexed between 1 and n.
- $1 \leq u_i, v_i \leq n$ and $u_i \neq v_i$ for all $1 \leq i \leq m$.
- $2 \leq \ell_i \leq 10^9$ is always an even number for all $1 \leq i \leq m$.

Output Format

If a tree satisfying Alice's description is found, then print out its structure in the following format. In the first line print two integers k and r, the total number of nodes in the tree and the index of the root vertex. In each of the next $k-1$ lines, print three integers x_i, y_i, and w_i, indicating that there is an edge between x_i and y_i with weight w_i in the tree.

Note that, your output for this case must follow the following guidelines.

- Use the same indexes for the leaf nodes as used in Alice's description.
- Label all nodes with indexes between 1 and k.
- The distance between any pair of leaf nodes is at most 10^9.

As there may be multiple number of trees satisfying Alice's criteria, she will honestly accept the answer as long as the tree you output meets all of the conditions she has described. This says, if there are multiple answers, you may print any of them.

If there exists no tree satisfying Alice's description, then print the string

> "Sorry, buddy. She's not into you."

in a line. Note that there are 5 spaces in the above sentence in total, one after each punctuation and each word, except for the last period.

Sample Input 1

```
3 3
1 2 10
2 3 18
1 3 18
```

Sample Output 1

```
5 5
1 4 5
2 4 5
4 5 4
3 5 9
```

Sample Input 2

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

Sample Output 2

```
Sorry, buddy. She's not into you.
```

Solution

Mathematical Model of the Problem

Given a partial distance metric d, determine if there exists an ultrametric T that is consistent with d.

Algorithm and Time Complexity of the Problem

Sort the distances specified in d in non-descending order. Consider the distances one by one and construct the ultrametric accordingly. Report "-1" if inconsistency is found during the process.

Problem I
Candies
Number of Test Cases: 40
Excution Time Limit: 5 seconds

In a kindergarten with n children, a teacher has a total of m candies. The teacher wishes to give the k-th child at most $f(n,k)$ candies, where $f(n,k) = \max\{d \mid d \times \alpha = k,\ d \times \beta = n \text{ for some } \alpha, \beta, d \in \mathbf{Z}\}$. The candies are distributed starting from the first child, giving as many as possible, then moving on to the the second child, and so on. It is possible that not all the children will receive candies. The task is to determine how many children, in total, will receive candies?

For example, if there are $4(= n)$ children and the teacher wants to give them at most 1, 2, 1, and 4 candies respectively (note that $f(4,1) = 1$, $f(4,2) = 2$, $f(4,3) = 1$, and $f(4,4) = 4$), then:

1. If the teacher only has $2(= m)$ candies, the four children will receive 1, 1, 0, and 0 candies respectively, so only 2 children will receive candies.
2. If the teacher has 6 candies, the four children will receive 1, 2, 1, and 2 candies respectively, so all 4 children will receive candies.
3. If the teacher has 9 candies, the four children will receive 1, 2, 1, and 4 candies respectively, so all 4 children will receive candies, with one candy remaining.

Input File Format

The first line is the number of test cases T with $T \leq 40$. Each test case has two positive integers n and m given in one line. You can assume that $1 \leq n \leq 20241006123000$ and $0 \leq m < 2^{63}$.

Output Format

For each test case, please print the number of children who will receive candies in a single line.

Sample Input

```
10
4 0
4 1
4 2
```

```
4 3
4 4
4 5
4 6
4 7
4 8
4 9
```

Output for the Sample Input

```
0
1
2
2
3
4
4
4
4
4
```

Solution

Mathematical Model of the Problem

Ground Truth: gcd(a,b)

a,b	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	1																													
2	1	2																												
3	1	1	3																											
4	1	2	1	4																										
5	1	1	1	1	5																									
6	1	2	3	2	1	6																								
7	1	1	1	1	1	1	7																							
8	1	2	1	4	1	2	1	8																						
9	1	1	3	1	1	3	1	1	9																					
10	1	2	1	2	5	2	1	2	1	10																				
11	1	1	1	1	1	1	1	1	1	1	11																			
12	1	2	3	4	1	6	1	4	3	2	1	12																		
13	1	1	1	1	1	1	1	1	1	1	1	1	13																	
14	1	2	1	2	1	2	7	2	1	2	1	2	1	14																
15	1	1	3	1	5	3	1	1	3	5	1	3	1	1	15															
16	1	2	1	4	1	2	1	8	1	2	1	4	1	2	1	16														
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	17													
18	1	2	3	2	1	6	1	2	9	2	1	6	1	2	3	2	1	18												
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19											
20	1	2	1	4	5	2	1	4	1	10	1	4	1	2	5	4	1	2	1	20										
21	1	1	3	1	1	3	7	1	3	1	1	3	1	7	3	1	1	3	1	1	21									
22	1	2	1	2	1	2	1	2	1	2	11	2	1	2	1	2	1	2	1	2	1	22								
23	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	23							
24	1	2	3	4	1	6	1	8	3	2	1	12	1	2	3	8	1	6	1	4	3	2	1	24						
25	1	1	1	1	5	1	1	1	1	5	1	1	1	1	5	1	1	1	1	5	1	1	1	1	25					
26	1	2	1	2	1	2	1	2	1	2	1	2	13	2	1	2	1	2	1	2	1	2	1	2	1	26				
27	1	1	3	1	1	3	1	1	9	1	1	3	1	1	3	1	1	9	1	1	3	1	1	3	1	1	27			
28	1	2	1	4	1	2	7	4	1	2	1	4	1	14	1	4	1	2	1	4	7	2	1	4	1	2	1	28		
29	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	29	
30	1	2	3	2	5	6	1	2	3	10	1	6	1	2	15	2	1	6	1	10	3	2	1	6	5	2	3	2	1	30

Definition 1:
Let $f(n,u)$ and $\phi(n,u)$ be defined as follows.
$f(n,u) = \sum_{k=1}^{u} gcd(k,n)$
$\phi(n,u) = |\{k \mid k \in \mathtt{Z}, 1 \leq k \leq u, gcd(k,n) = 1\}|$
We also let $f(n) = f(n,n)$ and $\phi(n) = \phi(n,n)$.

Ground Truth 2:

n	1	2	3	4	5	6	7	8	9	10
$f(n)$	1	3	5	8	9	15	13	20	21	27
$\phi(n)$	1	1	2	2	4	2	6	4	6	4
n	11	12	13	14	15	16	17	18	19	20
$f(n)$	21	40	25	39	45	48	33	63	37	72
$\phi(n)$	10	4	12	6	8	8	16	6	18	8
n	21	22	23	24	25	26	27	28	29	30
$f(n)$	65	63	45	100	65	75	81	104	57	135
$\phi(n)$	12	10	22	8	20	12	18	12	28	8

Theorem 3:
The function f is bounded as $max(2 - \frac{1}{n}, (\frac{3}{2})^{\omega(n)}) \leq \frac{f(n)}{n} \leq 27 \times (\frac{log(n)}{\omega(n)})^{\omega(n)}$, where $n \in \mathbf{N}$ and $\omega(n)$ is the number of distinct prime numbers dividing n.
(in https://cs.uwaterloo.ca/journals/JIS/VOL4/BROUGHAN/gcdsum.pdf)

Theorem 4:
$f(p^\alpha) = (\alpha+1)p^\alpha - \alpha p^{\alpha-1}$ for a prime p and $\alpha \geq 1$.
(in https://cs.uwaterloo.ca/journals/JIS/VOL4/BROUGHAN/gcdsum.pdf)

Corollary 5:
If $n = p^{\alpha}$ for a prime p and $\alpha \geq 1$, we have $f(n) \leq (\alpha + 1)n$.

Corollary 6:
If $\omega(n) = 1$, we have $f(n) \leq (log_2(n) + 2)n$.

Result 7:
Consider $2 \leq n \leq 20241006123000$ with $\omega(n) \geq 2$. We have $0 \leq log(n) \leq 45$. Since $2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19 \times 23 \times 29 \times 31 \times 37 = 7420738134810 < 20241006123000 < 304250263527210 = 2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19 \times 23 \times 29 \times 31 \times 37 \times 41$, we have $\omega(n) \leq 12$. So, we have
$f(n) \leq 27 \times (\frac{log(n)}{\omega(n)})^{\omega(n)} \leq 27 \times (\frac{45}{2})^{12} = 4.5... \times 10^{17} < 2^{57}$

Lemma 8:
Let a, b, and c be positive integers. We have $gcd(a \times c, b \times c) = c \times gcd(a, b)$

Theorem 9:
Given n, decompose $n = p_1^{n_1} \times p_2^{n_2} \times p_3^{n_3} \cdots p_r^{n_r}$ with primes p_k. Let $S = \{p_1, p_2, p_3, ...p_r\}$. We can find that
$f(n) = \phi(n) + \sum_{A \subset S} \{(-1)^{|A|+1} \times C \times f(n/C) \mid C = \prod_{p \in A} p\}$

For $n = 8 = 2^3$, we have $S = \{2\}$.
$f(8) = 1 + 2 + 1 + 4 + 1 + 2 + 1 + 8 = 20$
$f(8) = (1 + 1 + 1 + 1) + (2 + 4 + 2 + 8)$
$= \phi(8) + 2 \times (1 + 2 + 1 + 4)$
$= \phi(8) + 2 \times f(4)$

For $n = 6 = 2 \times 3$, we have $S = \{2, 3\}$.
$f(6) = 1 + 2 + 3 + 2 + 1 + 6 = 15$
$f(6) = (1 + 1) + (2 + 2 + 6) + (3 + 6) - (6)$
$= \phi(6) + 2 \times (1 + 1 + 3) + 3 \times (1 + 2) - 6 \times (1)$
$= \phi(6) + 2 \times f(3) + 3 \times f(2) - 6 \times f(1)$

For $n = 12 = 2^2 \times 3$, we have $S = \{2, 3\}$.
$f(12) = 1 + 2 + 3 + 4 + 1 + 6 + 1 + 4 + 3 + 2 + 1 + 12 = 40$
$f(12) = (1 + 1 + 1 + 1) + (2 + 4 + 6 + 4 + 2 + 12) + (3 + 6 + 3 + 12) - (6 + 12)$
$= \phi(12) + 2 \times (1 + 2 + 3 + 2 + 1 + 6) + 3 \times (1 + 2 + 1 + 4) - 6 \times (1 + 2)$
$= \phi(12) + 2 \times f(6) + 3 \times f(4) - 6 \times f(2)$

Theorem 10:
Given n and u, we decompose $n = p_1^{n_1} \times p_2^{n_2} \times p_3^{n_3} \cdots p_r^{n_r}$ with primes p_k. Let $S = \{p_1, p_2, p_3, ...p_r\}$. We can find that
$f(n, u) = u + \sum_{A \subset S} \{(-1)^{|A|} \times u/C \mid C = \prod_{p \in A} p\}$
$+ \sum_{A \subset S} \{(-1)^{|A|+1} \times C \times f(n/C, u/C) \mid C = \prod_{p \in A} p\}$

We assume that $a//b$ is the quotient of a divided by b.

For $n = 6 = 2 \times 3$ and $u = 5$, we have $S = \{2, 3\}$.
$f(n, u) = f(6, 5) = 1 + 2 + 3 + 2 + 1 = 9$
$= (1 + 1) + (2 + 2) + (3) - (0)$
$= \phi(6, 5) + 2 \times f(3, 2) + 3 \times f(2, 1) - 6 \times f(1, 0)$
$= \phi(6, 5) + 2 \times f(6/2, 5//2) + 3 \times f(6/3, 5//3) - 6 \times f(6/6, 5//6)$

For $n = 18 = 2 \times 3^2$ and $u = 13$, we have $S = \{2, 3\}$.
$f(n, u) = f(18, 13) = 1 + 2 + 3 + 2 + 1 + 6 + 1 + 2 + 9 + 2 + 1 + 6 + 1$
$= (1 + 1 + 1 + 1 + 1) + (2 + 2 + 6 + 2 + 2 + 6) + (3 + 6 + 9 + 6) - (6 + 6)$
$= \phi(18, 13) + 2 \times (1 + 1 + 3 + 1 + 1 + 3) + 3 \times (1 + 2 + 3 + 2) - 6 \times (1 + 1)$
$= 13 - 13//2 - 13//3 + 13//6$
$+ 2 \times f(18/2, 13//2) + 3 \times f(18/3, 13//3) - 6 \times f(18/6, 13//6).$

So, we can solve this problem by following techniques.

1. Sieve of Eratosthenes. $(O(\sqrt{n}))$
2. Dynamic programming.
3. Principle of inclusion-exclusion. $(O(2^r))$ $(O(2^{\omega(n)}))$
4. Binary search on u. $(O(log(n)))$

Problem J
Big Difference
Number of Test Cases: 14
Excution Time Limit: 1 second

In the NCPC park, there are n sightseeing spots and m lanes, with each of which connecting two of the n spots. The m lanes are not always accessible because they have to be maintained from time to time. Today, there are two groups of visitors. For each group, the owner of the park makes exactly $n-1$ lanes accessible to ensure that there is a path between any two spots; namely, the $n-1$ lanes form a tree structure connecting the n spots. The owner would like to make the two tree structures as different as possible. Given the tree structure T_1 for the first group, please help the owner find a tree structure T_2 for the second group so that the lanes used in T_2 but not in T_1 are as many as possible.

Precisely, the park can be seen as a connected undirected simple graph, say G, and T_1 is a given spanning tree of G. Your task is to find another spanning tree T_2 so that $|E(T_2) - E(T_1)|$ is as large as possible, where $E(X)$ denotes the edge set of graph X.

Please compute $|E(T_2) - E(T_1)|$. Notice that T_1 and T_2 can be identical.

Input File Format

The first line is an integer t, indicating the number of test cases. For each test case, the first line contains two integers n and m ($2 \leq n \leq 1000$, $1 \leq m \leq 260,000$), which are the numbers of vertices and edges of G, respectively. The vertices are numbered from 0 to $n-1$. Each of the following m lines contains two integers, indicating the endpoints of an edge, and the first $n-1$ of the m lines specify the edges of the spanning tree T_1.

Output Format

For each test case, print an integer, which is $|E(T_2) - E(T_1)|$.

Sample Input

```
2
2 1
0 1
4 5
0 1
1 2
3 1
```

```
2 3
3 0
```

Output for the Sample Input

```
0
2
```

Solution

Algorithm and Time Complexity of the Problem

For the edges in $E(T_1)$, let each of them be weighted by 2. For the remaining edges of G, set the weights to be 1. Then find a minimum spanning tree T' of the edge weighted graph. T' is the spanning tree which share the least number of edges in common with T. Let the sum of weights of the edges of T' be W. Then $2(n-1) - W$ is the answer.

Problem K
Magic Stone Knights
Number of Test Cases: 2
Excution Time Limit: 2 seconds

Far away from the mountains, where the sky and the sea meet, lies the Kingdom of Eldoria. This prosperous realm is known for its lush forests, sparkling rivers, and, most notably, its knights. The knights are equipped with an array of finely crafted weapons made of magic stones, each designed for specific purposes in battle. Each weapon has its own sequence of stones. When untouched, they are linearly arranged on the wall. But when a knight approaches, the stones will recognize the presence of its master. Upon the knight's touch, the stones will begin to fold into the weapons. The stones can be folded from head to tail by the sequence (mathematical formulation). For a sword, the stones would fold into a blade (Figure 1(a)). For a polearm, the stones would nest within one another, expanding and lengthening, forming a lethal blade of the weapon (Figure 1(b)). Note that cross-pairing is not allowed (Figure 1(c)). i.e., given a sequence of stones $S = [s_1, s_2, \ldots, s_n]$, if s_i is paired with s_j, no stones between i and j can be paired with those outside the range $[i + 1, j - 1]$.

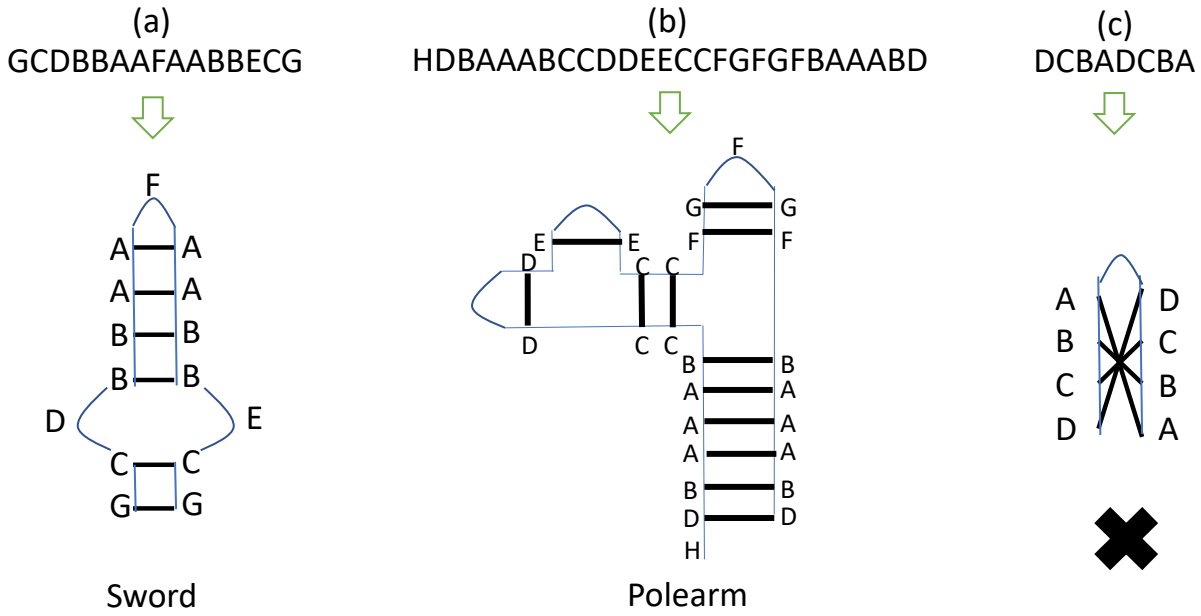


Figure 1: (a) the magic stones folded into a sword of 46 units of power; (b) the magic stones folded into a polearm of 94 units of power; (c) no cross-pairing is allowed.

The strength of a knight's weapon is not merely in its sharpness or size, but in the magic power when these stones fold together. The magic power of each weapon depends on the folding structure of the underlying stones. Assume each stone weapon is represented by a linear chain of alphabetic symbols (A-Z). When the stones are folded into their final shape, they sought to align perfectly, matching symbol to symbol from head and tail. The matching, and mismatching, and hanging stones in the folded structure will obtain magic power of +8, -1, and -1 units, respectively. For instance, the sword in Figure 1(a) obtains a total magic power of 6*(+8)-1-1=46, due to 6 pairs of matching stones (e.g., A

to A, B to B), one mismatching pair (i.e., D to E), and one hanging stone without any matched/unmatched stone (i.e., F). The polearm (Figure 1(b)) has 12 pairs of matching stones aligned perfectly and two hanging stones, thereby creating a weapon of strength of 12*(+8)+2*(-1))=94 power units. Every sequence of magic stones folds into the weapon of maximum magic power.

However, these powerful weapons become the envy of enemies who sought to seize them for their own dark purposes. To protect these weapons, the knights decide to encrypt all the weapons into a long, linear chain of rearranged stones. At first, the stones of each weapon (e.g., ABA) is appended by a special stone, marked by the ancient symbol of the number sign (e.g., ABA#). Then, the chain of stones of each weapon is circularly rotated, where each rotation shifts the stones and creates new patterns. For example, given two stone weapons (e.g., ABA# and BCACB#), ten circular rotations, ABA#, BA#A, A#AB, #ABA, BCACB#, CACB#B, ACB#BC, CB#BCA, B#BCAC, and #BCACB will be generated. Next, these patterns are sorted in lexicographical order (i.e., #ABA, #BCACB, A#AB, ABA#, ACB#BC, B#BCAC, BA#A, BCACB#, CACB#B, CB#BCA). Finally, only the last stones in the sorted patterns (i.e., ABB#CCA#BA) are stored as a linear chain in the armory. Therefore, their enemies cannot easily know what weapons are stored in the armory.

The armory is guarded by a sentinel, watching over the kingdom's most treasured weapons. For a knight to reclaim his weapon, he must correctly speak the sequence of the weapon stones (e.g., ABA) to the guardian. If the weapon exists, these stones will fold into the weapon with maximum magic power. The guardian will hand over the folded weapon to the knight and tell him the maximum magic power of this weapon (e.g., 7 for ABA). On the other hand, if a thief wants to steal the weapon and spells a false sequence of stones not existed in the armory (e.g., ABC), the guardian will tell a thief might be coming and not giving any weapon to him. Note that this armory not only hides the weapons from their enemies but also allows a more compact storage of weapons. When a knight is seeking for a smaller weapon (e.g., CAC), which is a part (a subsequence of consecutive stones) of a larger one (e.g., BCACB), the guardian would retrieve it with ease.

Given a chain of encrypted stones in the armory, and a stone sequence spelled by a knight or thief, you are asked to write a program which computes the maximum power of the weapon (if existed). Otherwise, if the weapon is non-existed, you should output "Thief" instead.

Input File Format

The first line contains the number of test cases. Each test case starts with a line containing the encrypted stone sequences of all weapons in the armory. The next line is the number of query stone strings N, $N \leq 10$. Each of the following N lines stores one query stone string from the knight or thief. The specification of each variable is given below.

1. The number of stones of all weapons in the armory ranges from 1 to 2500000.
2. The number of stones of each weapon ranges from 1 to 400

3. The stones of each weapon is from standard alphabet A, B, C, ..., Z.
4. The stones of each weapon is always appended with a number sign '#' when encripted in the armory.
5. Given a sequence of stones $S = [s_1, s_2, \ldots, s_n]$, a stone s_i can be paired with any stone s_j (either match or mismatch) where $j > i$ or hang alone without pairing to any other stone. If s_i is paired with s_j, no stone between i and j can be paired with those outside the range $[i + 1, j - 1]$ (i.e., no crossing pairs).
6. Each pair of matched stones obtains +8 units of magic power.
7. Each pair of mismatched stones obtains -1 units of magic power.
8. Each hanging-alone stone obtains -1 unit of magic power.

Output Format

For each query stone string within each test case, output the maximum magic power of the weapon if it exists. If it does not exist, your program should print "Thief" in one line.

Sample Input

```
2
ABB#CCA#BA
3
ABA
BCACB
ABC
AGDBBBFAABAAABACDFBCDAAABDDBEGCCEBHCA#CDEBDAGGCC#FF#
3
GCDBBAAFAABBECG
HDBAAABCCDDEECCFGFGFBAAABD
DCBADCBA
```

Output for the Sample Input

```
7
15
Thief
46
94
4
```

Solution

Mathematical Model of the Problem

This problem is composed of two subproblems. The first subproblem checks whether the query string is a substring of any weapon string in the armory, which are encoded by Burrows-Wheeler Transform (BWT). The first subproblem can be solved by using FM-index. Two tables, C[c] and Occ[c,i] will be first constructed. C[c] is a table that, for each character c, contains the number of occurrences of lexically smaller characters in the encoded string. The function Occ(c, k) is the number of occurrences of character c in the prefix of encoded string [1..k]. Subsequently, the existence of each query string can be known by computing the BWT interval [First, Last] via the backward-search algorithm shown below. If it exists, the frequency of each query string is the size of the BWT interval, that is, Last−First+1. Otherwise, if it does not exist, last will be less than First during the backward search.

Backward_search_algorithm$(P[1,p])$
(1) $i \leftarrow p, \quad c \leftarrow P[p], \quad First \leftarrow C[c]+1, \quad Last \leftarrow C[c+1];$
(2) while $(First \leq Last)$ and $(i \geq 2)$ do
(3) $\quad c \leftarrow P[i-1];$
(4) $\quad First \leftarrow C[c] + \text{Occ}(c, First-1) + 1;$
(5) $\quad Last \leftarrow C[c] + \text{Occ}(c, Last);$
(6) $\quad i \leftarrow i-1;$
(7) if $(Last < First)$return "Thief"
else return $\langle Last - First + 1 \rangle$.

If the weapon exists (i.e., $Last - First + 1 > 0$), the second subproblem aims to compute the maximum power of all possible folding structure, which can be solved by a dynamic programming on the substrings of the query string. Let $S[i,j]$ be the maximum power by aligning i-th stone to the j-th one. Let s be the power obtained by aligning the i-th and j-th stones, where $s = +8$ if both stones are the same (match) and -1 otherwise (mismatch or hanging).

$$S[i,j] = \max \begin{cases} S[i+1, j-1] + s, \\ S[i+1, j] - 1, \\ S[i, j-1] - 1, \\ \max\limits_{i<k<j} S[i,k] + S[k+1, j] \end{cases}$$

The construction of C[c] and Occ[c,k] takes O(nZ) time, where n is the length of BWT string and Z is the size of alphabet. The backward search takes $O(m)$ time, where m is the length of the query string. The dynamic programming of computing the maximum power takes $O(m^3)$ time. Therefore, the overall complexity is $O(nZ + m^3)$.

Problem L
Placing Rooks
Number of Test Cases: 1
Excution Time Limit: 1 second

You are given an n by n grid, where each cell at the intersection of the i-th row and j-th column is called the (i, j)-cell. For each (i, j)-cell, if i equals j, then it is a *diagonal* cell. Otherwise, we call it *non-diagonal* cell. Each non-diagonal cell is assigned one of three possible colors: white, blue, or green.

You are allowed to perform a sequence of *swap operations* on the grid. A swap operation proceeds as follows. Pick an index k in $\{1, 2, \ldots, n-1\}$. For each $i \notin \{k, k+1\}$, swap the (i, k)-cell and the $(i, k+1)$-cell, and also swap the (k, i)-cell and the $(k+1, i)$-cell. Finally, swap the $(k, k+1)$-cell and the $(k+1, k)$-cell. The illustration below depicts the swap operation for $n = 6$ and $k = 3$.

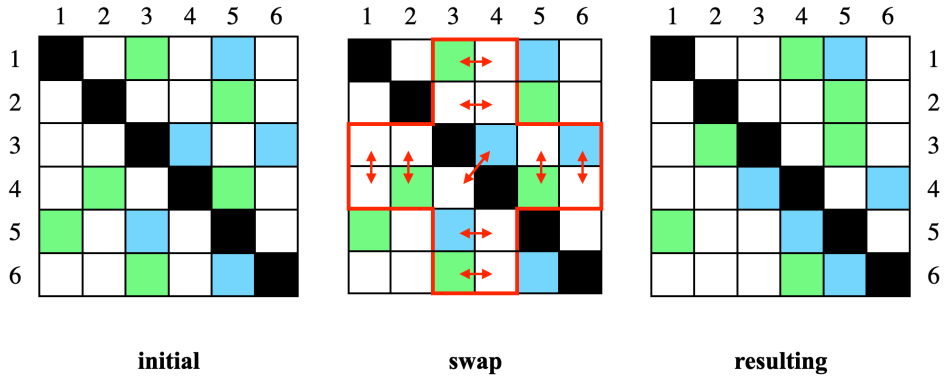


Figure 1: An illustration of the swap operation for $n = 6$ and $k = 3$.

We consider to perform a sequence of the swap operations on the grid such that the following two conditions simultaneously hold.

- Each non-diagonal cell that is not colored white must be positioned above a diagonal cell. In other words, all such non-diagonal cells must be placed in the upper-right triangle of the grid.
- There exists a way to place $n-1$ rooks, each on one of the $n-1$ selected non-diagonal cells (none of which are colored white) such that no two rooks can attack each other, and the number of rooks on blue cells is even. Formally, two rooks can attack each other if they are placed on the same row or the same column.

If such a sequence of swap operations exists, output "Yes." Otherwise, we need to count how many ways a single non-diagonal cell with any color can be colored blue such that the sequence exists. The output in this case will be an integer in the range $[0, n(n-1)]$.

Input File Format

Each test case is in a file. Each test case begins with two positive integer n and m ($n \leq 10^6$ and $m \leq 10^6$), indicating that the input is an n by n grid and m non-diagonal cells on the grid are not colored white. Then m triples of integers follow. Each of the m ordered triples (x, y, c) indicates that the (x, y)-cell is colored c where $c = 0$ for blue and $c = 1$ for green. Clearly, x and y both are integers in $\{1, 2, \ldots, n\}$.

Output Format

For each test case, output "Yes" on a line if the input is a former case. Otherwise, output ω on a line where ω denotes the number of ways a single non-diagonal cell with any color can be colored blue such that the desired sequence of swap operations exists.

Sample Input

Shown below are 3 test cases separated by blank lines.

```
3 2
1 2 1 2 3 1

3 2
1 3 0 2 3 1

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

Output for the Sample Input

Shown below are outputs for the test cases separated by blank lines.

```
Yes

1

2
```

Solution

Dynamic Programming on DAGs. See details below.

Mathematical Model of the Problem

This problem is a restatement of the problem that, given a DAG G (the n by n grid is the adjacency matrix of the DAG), count how many ways to add a blue edge to G or replace a green edge of G with color blue such that G contains a hamiltonian path with an even number of blue edges.

Algorithm and Time Complexity of the Problem

Once you find the above restatement. The solution is to implement a DP on a DAG, followed by $O(|V|)$ queries on the DP table. That is, you need to find a simple sx-path and yt-path for some x, y, source node s, and sink node t. Our goal is to join x and y with a directed edge such that there exists no yx-paths. This can be done in $O(|V|+|E|)$ time. This DP approach is not hard to find, but there is a more direct (but runtime-consuming) DP may trap the contestants for a while.

Problem M

Problem: H-index Calculation

Number of Test Cases: 20
Excution Time Limit: 3 seconds

In the fast-paced world of computer science research, impact is everything. With a constant flow of papers being published, how do we measure a scientist's contribution to the field? A balanced metric has emerged to capture both productivity and influence: H-index.
The H-index is a measure used to quantify the impact and productivity of a scientist based on their published papers.

Problem Description

The *H-index* is defined as the largest integer i such that the scientist has published i papers, in which each paper is cited at least i times.

Given the citation counts of papers for several scientists, your task is to calculate the H-index for each scientist based on their citation counts.

Input Format

The first line of input contains an integer N representing the number of scientists.

For each of the next N lines:

- The first integer M represents the number of papers published by the scientist.
- The next M integers represent the citation counts for each paper.
- For each line, consecutive two integers are separated by a space.

Output Format

The program should output N lines. For each scientist, print their H-index on a new line. If $N = 0$, please output the integer 0.

Constraints

- $0 \leq N \leq 500$
- $0 \leq M \leq 10^5$
- $0 \leq$ Citation Count $\leq 10^5$

Sample Input 1

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

Sample Output 1

```
3
2
```

Sample Input 2

```
0
```

Sample Output 2

```
0
```

Solution

The Algorithm and the Time Complexity of the problem

First, sorting the M papers in descending order based on their citation counts. Then, use binary search to efficiently find the largest index where the citation count is more than or equal to the current index (1-based). The value of this index represents the *H-index*. The time complexity mainly depends on the sorting and searching, which could be $O(nlogn)$.

Sample Explanation

In the first case, the first scientist has 5 papers with citation counts [3, 0, 6, 1, 5]. After sorting the citation counts in descending order [6, 5, 3, 1, 0], we see that the largest H such that at least H papers have been cited H times is 3.

In the second case, the second scientist has 3 papers with citation counts [4, 4, 0]. Sorting the counts gives [4, 4, 0], and the largest H such that at least H papers have been cited H times is 2.