GLOBUSSOFT
TECHNOLOGY AHEAD OF TIME

Greetings From Globussoft

- Given below are 5 Programming questions, you have to solve any 3 out of 5 questions.

- These 5 questions you can attempt in any technology like C/C++, java, .Net, PHP

- To solve these 3 questions you've max. 3 hours.

- While Solving these questions you are not allowed to use any Search Engine like Google, Yahoo, Bing ...

All the best for your test

Globussoft

QUESTION - 1

Phone List Given a list of phone numbers, determine if it is consistent in the sense that no number is the prefix of another. Let’s say the phone catalogue listed these numbers:

• Emergency 911

• Alice 97 625 999

• Bob 91 12 54 26

In this case, it’s not possible to call Bob, because the central would direct your call to the emergency line as soon as you had dialled the first three digits of Bob’s phone number. So this list would not be consistent.

Input

The first line of input gives a single integer, 1 <= *t* <= 40, the number of test cases. Each test case starts with *n*, the number of phone numbers, on a separate line, 1 <= *n* <= 10000. Then follows *n* lines with one unique phone number on each line. A phone number is a sequence of at most ten digits.

Output

For each test case, output “YES” if the list is consistent, or “NO” otherwise.

Example

```
Input:
2
3
911
97625999
91125426
5
113
12340
123440
12345
98346

Output:
NO
YES
```

QUESTION – 2

Little Ivica recently got a job delivering pizzas for the most popular pizzeria in town.

At the start of his work day, he receives a list with the locations to which he needs to deliver pizzas, inorder in which the locations are given.

The city is divided into R×C cells. The rows are numbered 1 through R, columns 1 through C.

From every cell, it is possible to move to neighbouring cells to the left and right. Moving up or down is only allowed in the first and last columns (columns 1 and C).

The pizzeria is in the top left corner (1, 1) and this is the location Ivica starts from. Ivica takes with him all the pizzas he will deliver that day so he does not have to return to the pizzeria between deliveries or after the last delivery.

For each location in the city, Ivica knows how much time he will spend every time he is in it (trying to get through the intersection, for example). Write a program that calculates the smallest amount of time for Ivica to deliver all the pizzas.

Input

The first line contains the integers R and C ($1 \leq R \leq 2000$, $1 \leq C \leq 200$), the dimensions of the city.

Each of the following R lines contains C integers. These are the times Ivica spends every time he enters a location. The times will be integers between 0 and 5000, inclusive.

The next line contains an integer D ($1 \leq D \leq 200\ 000$), the number of pizza deliveries that day. (No, it's not unrealistically large at all.) Each of the following D lines contains two integers A and B ($1 \leq A \leq R$, $1 \leq B \leq C$), the location to which a pizza must be delivered. The pizzas are given in the order in which they must be delivered. No location will be given twice in a row.

Output

Output the smallest amount of time for Ivica to deliver all the pizzas.

Sample

```
input
3 3
1 8 2
2 3 2
1 0 1
3
1 3
3 3
```

```
2 2
output
17

input
2 5
0 0 0 0 0
1 4 2 3 2
4
1 5
2 2
2 5
2 1
output
9
```

QUESTION – 3

In the nearby kindergarten they recently made up an attractive game of strength and agility that kids love. The surface for the game is a large flat area divided into N×N squares. The children lay large spongy cues onto the surface. The sides of the cubes are the same length as the sides of the squares. When a cube is put on the surface, its sides are aligned with some square. A cube may be put on another cube too. Kids enjoy building forts and hiding them, but they always leave behind a huge mess. Because of this, prior to closing the kindergarten, the teachers rearrange all the cubes so that they occupy a rectangle on the surface, with exactly one cube on every square in the rectangle. In one moving, a cube is taken off the top of a square to the top of any other square.

Write a program that, given the state of the surface, calculates the smallest number of moves needed to arrange all cubes into a rectangle.

Input

The first line contains the integers N and M ($1 \le N \le 100$, $1 \le M \le N^2$), the dimensions of the surface and the number of cubes currently on the surface.

Each of the following M lines contains two integers R and C ($1 \le R, C \le N$), the coordinates of the square that contains the cube.

Output

Output the smallest number of moves. A solution will always exist.

Sample

```
input
4 3
2 2
4 4
1 1
output
2

input
5 8
2 2
3 2
4 2
2 4
3 4
4 4
2 3
2 3
output
3
```

QUESTION – 4

One of the most fundamental data structure problems is the dictionary problem: given a set D of words you want to be able to quickly determine if any given query string q is present in the dictionary D or not. Hashing is a well-known solution for the problem. The idea is to create a function $h : \Sigma^* \rightarrow [0..n-1]$ from all strings to the integer range $0,1,..,n-1$, i.e. you describe a fast deterministic program which takes a string as input and outputs an integer between 0 and $n-1$. Next you allocate an empty hash table T of size n and for each word w in D, you set $T[h(w)] = w$. Thus, given a query string q, you only need to calculate $h(q)$ and see if $T[h(q)]$ equals q, to determine if q is in the dictionary. Seems simple enough, but aren't we forgetting something? Of course, what if two words in D map to the same location in the table? This phenomenon, called collision, happens fairly often (remember the Birthday paradox: in a class of 24 pupils there is more than 50% chance that two of them share birthday). On average you will only be able to put roughly $\sqrt{n}$-sized dictionaries into the table without getting collisions, quite poor space usage!

A stronger variant is Cuckoo Hashing. The idea is to use two hash functions h_1 and h_2. Thus each string maps to two positions in the table. A query string q is now handled as follows: you compute both $h_1(q)$ and $h_2(q)$, and if $T[h_1(q)] = q$, or $T[h_2(q)] = q$, you conclude that q is in D.

The name "Cuckoo Hashing" stems from the process of creating the table. Initially you have an empty table. You iterate over the words d in D, and insert them one by one. If $T[h_1(d)]$ is free, you set $T[h_1(d)] = d$. Otherwise if $T[h_2(d)]$ is free, you set $T[h_2(d)] = d$. If both are occupied however, just like the cuckoo with other birds' eggs, you evict the word r in $T[h_1(d)]$ and set $T[h_1(d)] = d$. Next you put r back into the table in its alternative place (and if that entry was already occupied you evict that word and move it to its alternative place, and so on). Of course, we may end up in an infinite loop here, in which case we need to rebuild the table with other choices of hash functions. The good news is that this will not happen with great probability even if D contains up to $n/2$ words

Input

On the first line of input is a single positive integer $1 \leq t \leq 50$ specifying the number of test cases to follow. Each test case begins with two positive integers $1 \leq m \leq n \leq 10000$ on a line of itself, m telling the number of words in the dictionary and n the size of the hash table in the test case. Next follow m lines of which the i:th describes the i:th word d_i in the dictionary D by two non-negative integers $h_1(d_i)$ and $h_2(d_i)$ less than n giving the two hash function values of the word d_i. The two values may be identical.

Output

For each test case there should be exactly one line of output either containing the string "successful hashing" if it is possible to insert all words in the given order into the table, or the string "rehash necessary" if it is impossible.

Example

```
Input:
2
3 3
0 1
1 2
2 0
5 6
2 3
3 1
1 2
5 1
2 5
Output:
successful hashing
rehash necessary
```

QUESTION – 5

Dave's little son Maverick likes to play card games, but being only four years old, he always lose when playing with his older friends. Also, arranging cards in his hand is quite a problem to him.

When Maverick gets his cards, he has to arrange them in groups so that all the cards in a group are of the same color. Next, he has to sort the cards in each group by their value – card with lowest value should be the leftmost in its group. Of course, he has to hold all the cards in his hand all the time.

He has to arrange his cards as quickly as possible, i.e. making as few moves as possible. A move consists of changing a position of one of his cards.

Write a program that will calculate the lowest number of moves needed to arrange cards.

Input

The first line of input file contains two integers C, number of colors ($1 \leq C \leq 4$), and N, number of cards of the same color ($1 \leq N \leq 100$), separated by a space character.

Each of the next C*N lines contains a pair of two integers X and Y, $1 \leq X \leq C$, $1 \leq Y \leq N$, separated by a space character.

Numbers in each of those lines determine a color (X) and a value (Y) of a card dealt to little Maverick. The order of lines corresponds to the order the cards were dealt to little Maverick. No two lines describe the same card.

Output

The first and only line of output file should contain the lowest number of moves needed to arrange the cards as described above.

Sample

```
CARDS.IN

2 2
2 1
1 2
1 1
2 2

CARDS.OUT

2

CARDS.IN

4 1
2 1
3 1
1 1
4 1

CARDS.OUT
```

```
0
```

CARDS.IN

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

CARDS.OUT

```
2
```