

Problem A. Bugcaaaaaat

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 256 megabytes

ID	abbr.	Description	meme
1	/button	bugcat-pressing-the-button	
2	/dead	bugcat-dead	
3	/eat	bugcat-eating	
4	/fn	bugcat-angry	
5	/shui	bugcat-sleeping	
6	/veg	broccolibugcat	
7	/you	fishbugcat	

As shown in the figure, in the land of Bugcats, there is a Bugcat chat software containing 7 types of memes. Each meme has three attributes:

- ID: an integer from 1 to 7.
- Abbreviation (abbr.): a string that starts with ‘/’ and contains only lowercase letters apart from the leading ‘/’.
- Description: a string consisting of lowercase letters and hyphens ‘-’.

Now a Bugcat sends you a string. As the maintainer of the software, you need to determine whether the string is an **abbreviation** or a **description**, and also find its corresponding ID.

Note that the string sent by the Bugcat is guaranteed to be the abbreviation or description of one of the 7 memes in the table.

Input

The input consists of a single line, representing the string sent by the Bugcat.

It is guaranteed that the string is one of the following: /button, /dead, /eat, /fn, /shui, /veg, /you, bugcat-pressing-the-button, bugcat-dead, bugcat-eating, bugcat-angry, bugcat-sleeping, broccolibugcat, fishbugcat.

Output

Output a single line.

First, output a string `abbr.` or `description` (in lowercase) indicating whether the input string is an abbreviation or a description, respectively. Then output a space followed by an integer from 1 to 7, representing the corresponding ID.

Examples

standard input	standard output
/button	abbr. 1
bugcat-angry	description 4

Problem B. Bugcat's Counting Game

Input file: `standard input`
Output file: `standard output`
Time limit: 1 second
Memory limit: 256 megabytes

There is a creature known as Bugcat who wants to play a counting game. At the start of the game, Bugcat chooses a fixed integer k . Then, starting from 1, Bugcat begins counting upwards, but it will only call out a number if it meets at least one of the following conditions:

1. The number is a multiple of k .
2. The digit k appears in the decimal representation of the number.

Bugcat wants to know what the x -th number it calls out will be. Since Bugcat is a bit slow and can only count one by one, it has tasked you with finding the answer.

Input

The input consists of two integers, k and x ($1 \leq k \leq 9, x \leq 10^5$).

Output

Print a single integer representing the x -th number that Bugcat calls out.

Examples

standard input	standard output
3 5	13
4 5	16

Note

In the first test case, the numbers Bugcat calls out are 3, 6, 9, 12, 13, ... The 5-th number in this sequence is 13.

In the second test case, the numbers Bugcat calls out are 4, 8, 12, 14, 16, ... The 5-th number in this sequence is 16.

Problem C. Bugcat's Unique Queue

Input file: **standard input**
Output file: **standard output**
Time limit: 1 second
Memory limit: 256 megabytes

Bugcat is currently managing a queue. Bugcat has a specific preference: it does not want any duplicate numbers to appear in the queue at the same time. Additionally, all numbers entering the queue must be within the range $[1, m]$.

You need to process Q operations. There are three types of operations:

- Type 1 (Push): Attempt to add the integer x to the back of the queue. If x is already present in the queue, do nothing. Otherwise, add x to the tail.
- Type 2 (Pop): Remove the element at the head of the queue. It is guaranteed that the queue is not empty when this operation is called.
- Type 3 (Query): Find the x -th number currently in the queue (indexed from 1, where 1 is the head). If the queue contains fewer than x elements, output -1 .

Your task is to output the result for every Type 3 operation.

Input

The first line contains two integers: Q (the number of operations) and m (the maximum value of elements), where $1 \leq Q, m \leq 10^6$.

The next Q lines describe the operations:

- 1 x : Attempt to add x to the queue ($1 \leq x \leq m$).
- 2: Pop the element at the head of the queue.
- 3 x : Query the x -th element in the queue ($1 \leq x \leq Q$).

Output

For each Type 3 operation, print the result on a new line.

Example

standard input	standard output
10 10	5
1 3	-1
1 5	3
2	8
3 1	
3 2	
1 3	
1 5	
1 8	
3 2	
3 3	

Problem D. Locks

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 256 megabytes

There are n elements $a_1, a_2, \dots, a_n$, each with a lock. Initially, all elements are unlocked.

Given q queries, the i -th query consists of four integers (t_s, t_e, l, r) , where $1 \leq l \leq r \leq n$. Each query performs a locking process and an unlocking process on the timeline.

Locking process: Starting from time t_s , in the next $r - l + 1$ time instants, the elements $a_l, a_{l+1}, \dots, a_r$ are processed in order. Specifically, at time $t_s + k$, element a_{l+k} is processed ($0 \leq k \leq r - l$). If this element is currently unlocked, it is locked, and the lock is marked as belonging to the current query; if it is already locked by another query, no operation is performed. That is, if when accessing this lock, it has already been locked by another locking process, this lock will not have any changes.

Unlocking process: Starting from time t_e , in the next $r - l + 1$ time instants, the elements $a_l, a_{l+1}, \dots, a_r$ are processed in order. Specifically, at time $t_e + k$, element a_{l+k} is processed ($0 \leq k \leq r - l$). If the current lock on this element was added by this query during the locking process, it is unlocked; otherwise, no operation is performed.

For all operations occurring at the same time on the same element, they are executed in ascending order of query index.

For each query, count the number of elements that were **successfully locked** during its locking process.

Input

The first line contains two integers n, q ($1 \leq n, q \leq 1000$).

The next q lines each contain four integers t_s, t_e, l, r ($1 \leq l \leq r \leq n, 1 \leq t_s < t_e \leq 10^9$), representing a query.

Output

Output q lines, where the i -th line contains the number of elements successfully locked by the i -th query.

Examples

standard input	standard output
5 5 3 5 3 4 2 5 4 5 2 4 3 5 2 5 3 5 1 3 5 5	0 1 2 0 1
5 5 1 2 5 5 2 4 3 5 3 4 1 4 1 3 3 5 2 5 1 2	1 0 2 3 2

Problem E. Bugcat's Gathering

Input file: **standard input**
Output file: **standard output**
Time limit: 1 second
Memory limit: 256 megabytes

Two Bugcats are planning to meet up for a gathering. The map they live on consists of n nodes and m directed edges. The first Bugcat starts at Node 1, and the second Bugcat starts at Node n .

Each edge i connects node x_i to node y_i . There are two ways to traverse an edge:

- Walking: This consumes z_i units of stamina.
- Taking a Taxi: This costs 1 dollar but consumes 0 stamina.

The two Bugcats have a combined total budget of w dollars. They can choose to take a taxi on any edge at any time, provided they do not exceed their total budget.

Your task is to find the minimum total stamina the two Bugcats must consume in order to meet at the same node, given their budget constraint. If it is impossible for the two Bugcats to meet, output -1 .

Input

The input uses multiple test cases.

The first line contains an integer T , the number of test cases.

For each test case, the first line contains three integers n, m, w ($\sum n, \sum m \leq 1000, 0 \leq w \leq 1000$), representing the number of nodes, the number of edges, and the total budget.

The next m lines each contain three integers x_i, y_i, z_i ($1 \leq x_i, y_i \leq n, 1 \leq z_i \leq 10^9$), representing a directed edge from x_i to y_i with a walking stamina cost of z_i .

Output

For each test case, output a single line representing the minimum combined stamina required for the two Bugcats to meet. If it is impossible for the two Bugcats to meet, output -1 .

Example

standard input	standard output
2	1
5 5 2	-1
1 2 3	
1 3 4	
5 4 2	
4 3 1	
2 3 100	
5 4 2	
1 2 3	
1 3 4	
5 4 2	
2 3 100	

Problem F. Card Game

Input file: `standard input`
Output file: `standard output`
Time limit: 2 seconds
Memory limit: 256 megabytes

Little Z is playing a card game. He has N cards stacked face-up in a pile. Each card has a single uppercase letter printed on its face.

The game requires him to rearrange the cards into a new pile, such that the sequence of letters from top to bottom of the new pile is lexicographically smallest.

Each time, Little Z can only take a card from either the top or the bottom of the current pile, and place it onto the bottom of the new pile. This operation is repeated until all cards have been moved to the new pile.

Given the initial sequence of letters from top to bottom, help Little Z find the lexicographically smallest sequence he can obtain.

Input

The first line contains a single integer N ($1 \leq N \leq 5 \times 10^5$).

The following N lines each contain a single uppercase letter, representing the letters on the cards from top to bottom in the initial pile.

Output

Output a string of length N , representing the lexicographically smallest sequence that can be obtained.

Example

standard input	standard output
6 B C B D C A	ABCBCD

Note

A string s is lexicographically smaller than a string t if and only if there exists some index i such that $s_1 = t_1$, $s_2 = t_2$, $\dots$, $s_{i-1} = t_{i-1}$, and $s_i < t_i$; or s is a proper prefix of t . Characters are compared according to alphabetical order, i.e. $A < B < \dots < Z$.

For example, $AABC < ABCA$, since at the second position $A < B$.

Problem G. Skynet

Input file: **standard input**
Output file: **standard output**
Time limit: 3 seconds
Memory limit: 1024 megabytes

This is an interactive problem.

In a layered defense system, there is an $n \times m$ grid. Rows are numbered 1 to n from top to bottom, and columns are numbered 1 to m from left to right. Some cells initially contain obstacles.

An invading drone starts from an empty cell in row n and attempts to break through upward. You need to place additional obstacles to intercept it before it reaches row 1.

Suppose the drone is currently at cell (r, c) . If $r > 1$, it moves upward one row in this round, and its target can only be one of the following three positions: $(r - 1, c - 1)$, $(r - 1, c)$, $(r - 1, c + 1)$. The cell it moves to must satisfy:

- It is within the board boundaries;
- There is no obstacle in that cell.

If the drone reaches row 1, it is considered to have successfully broken through, and your program fails.

After each move of the drone, if it is currently at (r, c) with $r > 1$, you may place **at most one** new obstacle. If you choose to place one, the obstacle must be placed in a currently empty cell (x, y) satisfying $x \in \{r - 1, r - 2\}$, $1 \leq x \leq n$, $1 \leq y \leq m$. That is, you can only add an obstacle in the first or second row above the drone's current position. Newly placed obstacles remain permanently; you may also choose not to place any obstacle in that round.

Your goal is to make the drone unable to perform a move at some moment, thereby intercepting it successfully.

It is guaranteed that for every test case, there exists at least one strategy that ensures you can intercept the drone before it reaches row 1.

Input

The first line contains two integers n and m ($2 \leq n, m \leq 100$), the number of rows and columns of the board.

The next n lines each contain a string s_i of length m , describing the initial board state:

- If $s_{i,j} = \#$, then cell (i, j) initially contains an obstacle;
- If $s_{i,j} = .$, then cell (i, j) is initially empty.

It is guaranteed that row n contains at least one empty cell.

The last line contains two integers r_0 and c_0 , the initial position of the drone (r_0, c_0) . It is guaranteed that $r_0 = n$ and that this cell is initially empty.

After reading the input, you will interact according to the following rules.

Interaction Protocol

After reading the input, you repeatedly perform the following interaction until the process ends:

1. Output two integers $x \ y$.
 - If $(x, y) = (0, 0)$, it means you choose not to place an obstacle in this round;

- Otherwise, it means you place a new obstacle at cell (x, y) . In this case the following must hold:
 - The cell is within the board boundaries;
 - The cell currently contains no obstacle;
 - If the drone is currently in row t , then you must have $x \in \{t - 1, t - 2\}$.

2. Then the interactor moves the drone one step and outputs two integers r and c :

- If $(r, c) = (0, 0)$, the drone can no longer make a move, meaning you have successfully intercepted it. Your program should terminate immediately;
- If $(r, c) = (-1, -1)$, you have failed to intercept it. This happens when the drone has already reached row 1 or your output was invalid. Your program should terminate immediately, and you will receive **Wrong answer**;
- Otherwise, the drone has moved to cell (r, c) , and the next round begins.

If your program continues interacting after reading $(0, 0)$ or $(-1, -1)$, or fails to follow the interaction protocol, you will also receive **Wrong answer**.

After outputting, flush the output buffer; otherwise you may get **Idleness limit exceeded**. For example:

- C/C++: `fflush(stdout)` or `cout.flush()`
- Java: `System.out.flush()`
- Python: `sys.stdout.flush()`

Example

standard input	standard output
5 5	
.....	
.....	
#.#..	
..#..	
.....	
5 3	
	4 4
4 2	
	3 2
0 0	

Note

The interactor can be **adaptive**. That is, the drone's moves are not necessarily fixed before the interaction starts.

Initial obstacles and obstacles you place later remain permanently and never disappear.

The drone never enters a cell containing an obstacle.

You must ensure that no matter how the interactor makes moves, your strategy eventually intercepts the drone.

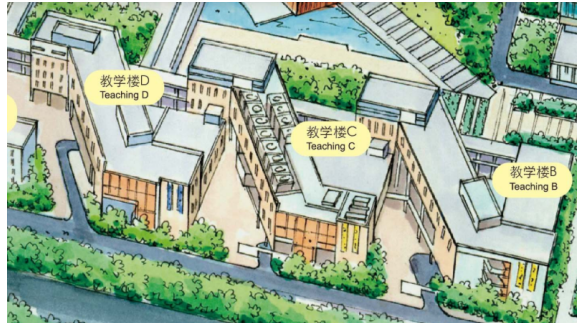
The sample interaction above proceeds as follows:

- Initially the drone is at $(5, 3)$.

- In round 1, you output 4 4, placing an obstacle at (4,4). Now row 4 has obstacles at (4,3) and (4,4), so the drone can only move to (4,2). The interactor then outputs 4 2.
- In round 2, you output 3 2, placing an obstacle at (3,2). Now cells (3,1), (3,2), (3,3) in row 3 are all unreachable. The interactor then outputs 0 0, indicating you have successfully intercepted the drone.

Problem H. Teaching Building

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 1024 megabytes



As a student of The Chinese University of Hong Kong, Shenzhen, Little B got lost when entering the teaching building for the first time (because TD is above TC, and TC is above TB). Years later, Little B became the designer of the new campus teaching building, and now he is designing the cross-sectional view of the building.

The new teaching building will have $2n$ teaching areas, numbered from 1 to $2n$. The designed footprint length is $2n$ units, and the building height is limited to $n + 1$ units, meaning that the cross-sectional view can be regarded as an $(n + 1) \times (2n)$ grid. Each grid point must contain a number from 0 to $2n$. $1 \sim 2n$ indicates the teaching area number at that position, and 0 means that the position does not belong to any teaching area. Each teaching area must be **4-connected*** in the cross-sectional view.

Let $a_{i,j}$ denote the teaching area number at the grid point in the i -th row and j -th column. We say that teaching area x is *above* teaching area y ($x \neq y$) if there exist i, j ($1 \leq i \leq n$, $1 \leq j \leq 2n$) such that $a_{i,j} = x$, $a_{i+1,j} = y$.

For structural clarity and elegance, Little B has determined $(2n - 1)$ relations. The i -th relation is of the form “teaching area x_i is above teaching area y_i ”. If we add a directed edge from x_i to y_i , the $2n$ teaching areas exactly form a **downward tree**** rooted at teaching area 1.

Now, you need to design a valid cross-sectional view (i.e., assign a teaching area number to each grid cell) such that the directed graph corresponding to this cross-sectional view is exactly the same as the tree given by Little B. (That is, the relationships specified by Little B must be satisfied, while the relationships not specified by Little B must not be satisfied.)

Please output any valid filling scheme, or report that no solution exists.

***4-connected:** A set of grid points is called 4-connected if any two points in the set can be connected by a sequence of adjacent (up, down, left, right) grid points that also belong to the set.

****Downward tree:** A rooted tree with a specified root, where every edge is directed from the parent to the child (i.e., from the upper level to the lower level). In this problem, the given $2n - 1$ relations “teaching area x_i is above teaching area y_i ” correspond to directed edges $x_i \rightarrow y_i$, and these edges form a downward tree rooted at 1.

Input

The first line contains an integer T ($1 \leq T \leq 500$), indicating the number of test cases.

For each test case, the first line contains an integer n ($1 \leq n \leq 500$).

Then $2n - 1$ lines follow, each containing two integers x_i, y_i , representing a relation. It is guaranteed that the input forms a downward tree rooted at 1.

It is also guaranteed that $\sum n \leq 2000$.

Output

For each test case:

- If a valid solution exists, output $(n + 1)$ lines, each containing $2n$ integers representing the construction. If multiple solutions exist, output any of them.
- If no solution exists, output a single line containing the string `No solution`.

Example

standard input	standard output
1	2 1 1 1
2	2 2 2 3
1 2	4 4 0 3
1 3	
2 4	

Problem I. Bugcat's Adventure

Input file: standard input
Output file: standard output
Time limit: 3 seconds
Memory limit: 256 megabytes

Bugcat is playing a game. The game provides an undirected graph where each node contains a monster. A monster may also have zero or one minion. A node is considered **unlocked** only after you defeat all enemies (including the minion, if it exists) at that node. You can move to a node if and only if it is connected to at least one already unlocked node.

However, you do not necessarily need to defeat all enemies when you visit a node. Specifically, you may choose to defeat only the monster's minion and then leave for other nodes. In this case, the node remains locked.

The player starts with an initial HP x . When attempting to kill an enemy with HP y , if $x \geq y$, the enemy is defeated and the player's HP increases by y (i.e., $x \leftarrow x + y$). Otherwise, the player dies immediately.

Maomaochong asks for your help. He provides the undirected graph and the initial HP of the monster at each node (initially, no monsters have minions). There are multiple operations of two types:

1. The monster at node x summons a minion with HP y (it is guaranteed that each monster at node x will summon at most one minion throughout the game).
2. Given a starting node x and an initial HP y , calculate the maximum possible final HP the player can achieve.

Input

The first line contains three integers n, m, q ($1 \leq n, m, q \leq 2 \times 10^5$), representing the number of nodes, edges, and queries.

The second line contains n integers representing the health of the monster at each node h_i ($1 \leq h_i \leq 10^6$).

The next m lines each contain two integers x, y representing an undirected edge between node x and y . There are no self-loops or multiple edges.

The next q lines start with an operation type op :

- If $op = 1$: The operation is modify. Two integers x, y follow ($1 \leq x \leq n, 1 \leq y < h_x$). Node x summons a minion with health y . It is guaranteed that each node x will appear in a Type 1 query at most once.
- If $op = 2$: The operation is query. Two integers x, y follow ($1 \leq x \leq n, 1 \leq y \leq 10^6$). You start at node x with health y .

Output

For each Query, output a single integer representing the maximum possible final health.

Example

standard input	standard output
5 5 5	5
1 2 6 6 8	27
1 2	4
1 3	
1 4	
4 5	
3 5	
2 2 2	
1 3 2	
2 2 2	
1 4 5	
2 4 4	

Problem J. Bugcat's Mahjong

Input file: `standard input`
Output file: `standard output`
Time limit: 2 seconds
Memory limit: 1024 megabytes

During the Lunar New Year, Bugcat has been playing a Mahjong-themed "roguelike" game (similar to Balatro). In this game, Bugcat draws tiles from a deck and discards them, and it has perfect knowledge of the tile sequence in the "wall" (the deck).

Bugcat starts wondering: if there is a Mahjong set with n distinct types of tiles, what is the expected number of tiles drawn (including the initial 13 tiles) to complete a "Four Concealed Pongs" (Sūankō) hand, assuming optimal play? Output the result modulo $10^9 + 7$.

Definitions:

- The Mahjong set: There are n types of tiles. Each type has exactly 4 identical-looking but distinct tiles. Thus, there are a total of $4n$ tiles in the deck.
- Four Concealed Pongs: A "Pong" (or triplet) is defined as 3 tiles of the same type. To complete this hand, you need four Pongs and one pair (2 tiles of the same type).

Simplified Problem Statement:

There are $4n$ tiles, categorized into n colors (types), with 4 tiles per color. Even tiles of the same color are considered distinct.

Consider all $(4n)!$ possible permutations of the deck. For a given permutation, let the "cost" be the smallest index i such that among the first i tiles in the sequence, the following two conditions are met:

- At least 4 colors have appeared 3 or more times.
- At least 5 colors have appeared 2 or more times.

Find the expected value of the cost i over all possible permutations. Output the result modulo $10^9 + 7$.

Input

A single integer n ($1 \leq n \leq 2 \times 10^6$), representing the number of tile types.

Output

A single integer representing the expected cost modulo $10^9 + 7$.

Examples

standard input	standard output
5	80495372
4	0

Problem K. Moonlit Trees

Input file: standard input
Output file: standard output
Time limit: 3 seconds
Memory limit: 1024 megabytes

A tree by the river can be described as a form with x nodes numbered $1 \sim x$:

- There is exactly one root, numbered 1.
- For each node i ($i \neq 1$), it has exactly one parent numbered f , satisfying $f < i$.

Little B cannot remember such an abstract concept as a “tree”, so he invented two ways to turn a tree into a permutation:

Algorithm 1 Transfer Tree to Permutation

Require: sons of each node *son*, interger *type* equals to 1 or 2.

Ensure: queue q .

```
1: function DFS( $u$ )
2:    $q.push(u)$ 
3:   if  $type = 1$  then
4:     for each  $v$  in  $son[u]$  in increasing order do
5:       DFS( $v$ )
6:     end for
7:   else
8:     for each  $v$  in  $son[u]$  in decreasing order do
9:       DFS( $v$ )
10:    end for
11:  end if
12: end function
13: DFS(1)
```

- **Method 1**, $type = 1$: Start from the root, visit the parent first, then the children, and children are visited in increasing order of their numbers.
- **Method 2**, $type = 2$: Start from the root, visit the parent first, then the children, and children are visited in decreasing order of their numbers.

It can be proved that after visiting all nodes, the elements in q form a permutation of $1 \sim x$.

A permutation is called **ambiguous** if it can be obtained from some tree t_1 by method 1 and also from some tree t_2 by method 2. Note that the two trees may be identical, i.e., t_1 may equal t_2 .

The bugcat only remembers the positions of $1 \sim n$ in a permutation a of length $n + m$, and asks you to determine, among all possible permutations, how many are ambiguous.

Input

The first line contains an integer T ($1 \leq T \leq 10^5$) indicating the number of test cases.

For each test case, the first line contains two integers n and m ($0 \leq m \leq 80$, $1 \leq (n + m) \leq 10^5$).

The second line contains $n + m$ integers $a_{1 \sim n+m}$ ($0 \leq a_i \leq n$). If $a_i = 0$, it means that the bugcat does not remember the value at position i . It is guaranteed that each number from 1 to n appears exactly once in a .

It is also guaranteed that $\sum(n + m) \leq 3 \times 10^5$.

Output

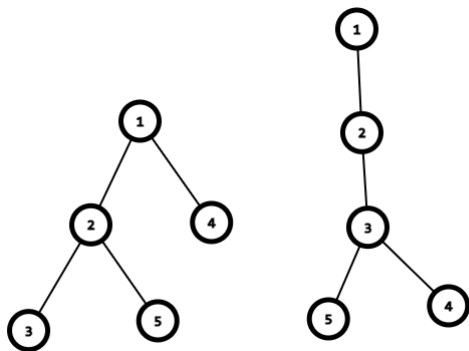
For each test case, output a single integer representing the number of ambiguous permutations, modulo $10^9 + 7$.

Example

standard input	standard output
3	1
3 0	1
1 2 3	109
4 1	
1 2 3 0 4	
3 5	
1 2 3 0 0 0 0 0	

Note

For the second sample, the only possible permutation is 1, 2, 3, 5, 4. It can be obtained from the first tree by method 1, and also from the second tree by method 2, thus it is ambiguous.



Problem L. Pairing Bugcats

Input file: `standard input`
Output file: `standard output`
Time limit: 2 seconds
Memory limit: 1024 megabytes

There are 2^n bugcats, each with a distinct attribute that can be represented as a number from 0 to $2^n - 1$.

These bugcats are to be paired into 2^{n-1} groups (each group contains exactly two bugcats) to dine in the restaurant. The bugcats are kind-hearted, so no bugcat appears in more than one group.

The restaurant has $2^{n-1} + 1$ tables for two, numbered from 1 to $2^{n-1} + 1$. It is required that the **XOR** of the attribute values of the two bugcats at each table equals the table number.

However, Little B, who came to the restaurant alone, has already taken the quietest table x . (That is, table numbered x cannot be used by the bugcats.)

Can you provide a pairing such that all bugcats in different groups can be seated at different tables for two, and Little B does not move?

Input

The input consists of a single line with two integers n, x ($1 \leq n \leq 20, 1 \leq x \leq 2^{n-1} + 1$).

Output

The first line of output should be a string YES or NO indicating whether such a pairing exists.

If it exists, output 2^{n-1} lines, each containing two integers x_i, y_i , indicating that the bugcats with attributes x_i and y_i are in the same group.

The groups can be output in any order. If multiple solutions exist, output any one.

Examples

standard input	standard output
3 1	YES 5 7 0 3 2 6 1 4
2 1	NO
3 2	NO
1 2	YES 0 1