Problem A. Aarelia Mountains

Input file:	aarelia.in
Output file:	aarelia.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

You are a powerful wizard travelling across the Aarelia kingdom. Your path lies across the land of steep hills and mountains. The mountain area can be represented as a row of n regions; each region has its *elevation* h_i.

Unfortunately, you have no climbing equipment (and, frankly, no climbing skills), and your Flight spells are exhausted. However, you will be able to travel across the area when region elevations will form a non-decreasing sequence, that is, $h_i \le h_{i+1}$ for each i from 1 to $(n-1)$ (you are travelling from right to left, and high falls are no threat for you).

The only spell that can help you is the Earthquake spell. The spell affects several adjacent regions and allows you to alter their elevation. You know m varieties of the Earthquake spell; each spell has its range l_i (that is, number of adjacent regions affected by the spell), its energy cost c_i and can either raise or lower all affected regions' elevation by 1 (a single spell can be applied either to raise or lower regions, but not both). Each spell can be applied to any range of length l_i that lies completely within the area, and can be cast an unlimited number of times (paying energy cost after each repetition). It is possible that some of the regions will have negative elevation after casting a spell.

Determine whether it is possible to alter the landscape so that you may travel safely across the area, and if it is, find the lowest total energy you have to spend on casting the spells.

Input

The first line contains two space-separated integers n and m ($1 \le n, m \le 200$).

The second line contains n space-separated integers h_i — initial elevation of the regions ($0 \le h_i \le 10^6$).

The next m lines describe varieties of the Earthquake spell you know. The i-th of these lines describes the spell variety according to format "t_i l_i c_i"; here t_i is the character "+" if the spell allows to raise elevations by 1, or "-" if it allows to lower elevations by 1; l_i and c_i are the range and the energy cost of the spell, respectively ($1 \le l_i \le n$, $1 \le c_i \le 10^6$).

Output

It it is possible to travel across the area, print the lowest total energy that must be spent on casting the spells, otherwise print -1.

Examples

aarelia.in	aarelia.out
3 2 3 2 1 + 1 1 - 1 1	2
3 1 3 2 1 + 2 1	-1

Problem B. Bar "Duck"

Input file:	barduck.in
Output file:	barduck.out
Time limit:	10 seconds
Memory limit:	256 mebibytes

You're having a party at your place tonight! Some of your friends will come over, and you're going to spend the evening playing your favourite board game called 'Bar "Duck"'. But several minutes prior to the guests' arrival you have discovered that the only table suitable for playing is covered with trash of mysterious origin. You've got no time for proper cleaning, so you're going to simply fling some of the trash pieces away so that at least a portion of the table is clean.

The table is long, but very narrow, so its surface can be simply represented as a straight segment of length L centimeters; you have chosen a coordinate system such that one of the table ends has coordinate 0 and the other one has coordinate L. There are n trash pieces on the table; the i-th piece is located at a point with coordinate x_i and has mass of m_i grams.

You can fling some of the pieces, that is, provide them with initial speed in either of two directions along the axis. After a piece is flung, it moves with the initial provided speed with no acceleration or deceleration (you're not quite sure why it acts like that) until it reaches either end of the table and falls on the floor; you may assume that pieces move independently of each other, that is, they do not collide or interfere with each other. However, under the laws of physics, applying a force requires spending energy; more precisely, to make an object of mass m grams move at a speed of v centimeters per second, you have to spend $\frac{mv^2}{2}$ units of energy.

The game will start after exactly T seconds from the present moment. The game always takes up a lot of space, so at the moment of the game start you will choose the longest segment of the table that does not contain any pieces of trash. Besides, you're tired after your day job, so you're not going to spend more than E units of energy on flinging the trash pieces. You may provide any speed to any piece or an arbitrary set of pieces (you choose speed independently for each piece) assuming the total spent energy does not exceed the limit of E units.

Find out the length of the longest clean table segment you can obtain after T seconds wasting no more than E units of energy.

Input

The first line contains four space-separated integers — n, L, T, and E ($1 \le n \le 10^5$, $1 \le L, T, E \le 10^6$).

The next n lines describe the trash pieces. The i-th line contains two space-separated integers x_i and m_i ($0 < x_i < L$, $1 \le m_i \le 10^3$) — position and mass of the i-th piece. No two pieces occupy the same place; you may assume that the pieces in the input are sorted by increasing x_i.

Output

Print one real number — the length (in centimeters) of the longest clean table segment you can obtain after T seconds spending no more than E units of energy. The answer will be judged as correct if relative or absolute error when comparing to the jury's answer will not exceed 10^{-6}.

Examples

barduck.in	barduck.out
1 4 1 1 2 1	3.4142135624
2 6 1 1 2 1 4 1	4.0000000000

Note

In the first sample we spend all energy to provide speed of $\sqrt{2}$ centimeters per second to the only piece, so after one second the longest clean segment has length of $2 + \sqrt{2}$ centimeters.

In the second sample the optimal way is to make both pieces move at speed of 1 centimeters per second in the opposite directions.

Problem C. Collections In Containers

Input file:	`collections.in`
Output file:	`collections.out`
Time limit:	1 second
Memory limit:	256 mebibytes

A *d-dimensional collection* is a set of vectors such that:

- Each coordinate of each vector in the set is a non-negative integer number.
- Let a vector $v = (v_1, \ldots, v_d)$ belong to the set. Then every vector $u = (u_1, \ldots, u_d)$ such that all u_i are non-negative integers and u does not exceed v coordinate-wise (that is, for every index i from 1 to d the inequality $0 \le u_i \le v_i$ must hold) must belong to the set as well.

For example, $\{(0,0),(0,1),(1,0)\}$ and $\{(0,0),(1,0),(2,0),(3,0)\}$ are 2-*dimensional collections*, while $\{(0,1),(1,0),(1,1)\}$, $\{(0,0),(-1,0)\}$, and $\{(0,0),(0,\frac{1}{2}),(0,1)\}$ are not.

You have to store two identical d-dimensional collections, each of size n. In order to do that, you gathered n identical d-dimensional containers each of which has a capacity given by the *capacity vector* $c = (c_1, \ldots, c_d)$. You decided to choose a container for each vector from each collection in such a way that every container would contain exactly one vector from the first collection and exactly one vector from the second collection. However, if two vectors $v = (v_1, \ldots, v_d)$ and $u = (u_1, \ldots, u_d)$ are placed in the same container, it must be verified that $v_i + u_i \le c_i$ for every i; that is, the vector $v + u$ must not exceed the capacity vector c coordinate-wise.

It is also guaranteed that each vector from each collection fits in any container by itself; that is, each vector v of each collection does not exceed the capacity vector c coordinate-wise.

Finding an arrangement to place all those vectors in containers turned out to be hard, so you decided to write a program that will solve this problem.

Input

The first line of input contains two space-separated positive integers n and d ($1 \le n \cdot d \le 10^5$) — the size and the dimension of each collection.

The second line of input contains d space-separated integers c_1, ..., c_d ($0 \le c_i \le 10^9$).

Next n lines of input contain description of the collections; i-th of these lines contains d space-separated integers v_{i1}, ..., v_{id} ($0 \le v_{ij} \le 10^9$) — coordinates of i-th vector in the collection. It is guaranteed that the described set of vectors is indeed a d-dimensional collection according to the definition above, and $v_{ij} \le c_j$.

Output

Output n lines, with two space-separated integers in each — the numbers of vectors from the first and the second collection respectively that must be put in the i-th container. The numeration of vectors in the input is 1-based. Every vector from every collection must be assigned to exactly one container.

If there is more than one possible arrangement, output any one of them. It is guaranteed that there is at least one arrangement that satisfies all the requirements.

Examples

collections.in	collections.out
4 2 1 1 0 0 0 1 1 0 1 1	1 4 2 3 3 2 4 1
4 2 2 1 0 0 1 0 2 0 0 1	1 4 2 2 3 1 4 3

Problem D. 1D Spreadsheet

Input file:	1d-spreadsheet.in
Output file:	1d-spreadsheet.out
Time limit:	10 seconds
Memory limit:	512 mebibytes

You've got a marvellous opportunity to work as an intern for a huge software company over summer. The project you are assigned to is a brand new, cutting-edge application — 1D spreadsheet.

A 1D spreadsheet is an array of n *cells* numbered from 1 to n. The cell i may contain either an integer a_i or a *link* to some other cell l_i (in which case we will say that the cell i is *linked* to the cell l_i). Cyclic link dependencies are not allowed, that is, at any moment, there is no sequence of cells i_1, i_2, ..., i_k such that the cell i_1 is linked to the cell i_2, ..., and the cell i_k is linked to the cell i_1.

Every cell can be *evaluated* to produce its *value*, which is defined as follows: if the cell i contains a number a_i then $value(i) = a_i$, otherwise $value(i) = value(l_i)$; that is, to evaluate a cell one must follow a chain of links that starts at the cell until he arrives at a cell that contains a number. As cyclic dependencies are disallowed, at any moment, $value(i)$ can be determined unambiguously for every cell i.

Your colleague Bill has just finished developing a new feature for the spreadsheet — range value summation. With this feature enabled, user may ask queries of sort "find $\sum_{i=l}^{r} value(i)$" as well as change the contents of cells. However, after some performance testing it turned out that Bill's implementation is quite slow. Your task is to optimize this feature. Write a program that processes a sequence of queries to 1D spreadsheet.

Input

The first line of input contains two space-separated integers n and q ($1 \le n, q \le 2 \cdot 10^5$) — the number of cells in the spreadsheet and the number of queries to process, respectively.

The second line contains n space-separated integers a_1, ..., a_n ($-10^4 \le a_i \le 10^4$) — the initial values of cells. Initially, no cell contains a link.

The next q lines contain descriptions of queries according to the following format:

- "`set` i x" — assign the cell i to contain the number x ($-10^4 \le x \le 10^4$)
- "`link` i j" — assign the cell i to contain link to the cell j.
- "`sum` l r" — find $\sum_{i=l}^{r} value(i)$.

It is guaranteed that cyclic dependencies will not occur at any point of time.

Output

Output the answers for "`sum` l r" queries in the same order as in the input, one per line.

Example

1d-spreadsheet.in	1d-spreadsheet.out
5 7 1 2 3 4 5 sum 1 5 link 3 5 link 1 3 link 4 2 sum 1 5 set 5 -1 sum 1 5	15 19 1

Problem E. Even Separation

Input file:	even.in
Output file:	even.out
Time limit:	1 second
Memory limit:	256 mebibytes

The good king of the Even Kingdom is dying, and two (surely, an even number) of his sons, Alfred and Brian, are in equal right of ruling the kingdom. The king has decided to separate the kingdom into two parts, so that each son can rule his part single-handedly. Of course, the separation must be *even*.

The kingdom consists of n cities, some of which are connected by bidirectional roads; no road connects a city with itself, and no two cities are connected by two or more roads. After separation, each city will belong either to Alfred or to Brian. Moreover, every road which connects cities that belong to different brothers will be destroyed. The separation will be *even* if, after those roads are destroyed, each of the cities will be **directly** connected to an even number of other cities (that is, the degree of each vertex of the graph representing the road network will be even).

Time is running short; help the good king to find the even separation of his kingdom!

Note that initially some pairs of cities may be unreachable from each other via roads, and in an even separation it may be possible that some cities in some part are unreachable from each other via remaining roads.

Input

The first line of input contains two space-separated integers n and m — the number of cities and roads in the Even Kingdom ($1 \le n \le 500$, $0 \le m \le \frac{n(n-1)}{2}$).

The next m lines describe the roads in the kingdom; i-th of these lines contain two space-separated integers — numbers of the cities connected by i-th road. Cities are numbered starting from 1. There is no more than one road connecting the every pair of cities, and no road connecting a city to itself.

Output

If the separation is possible, print one line containing n characters; the i-th character should be "A" or "B" if the i-th city should belong to Alfred or Brian, respectively. If there are several possible answers, output any of them.

If the separation is impossible, print one line "IMPOSSIBLE".

Examples

even.in	even.out
4 6 1 2 1 3 1 4 2 3 2 4 3 4	BAAA
3 3 1 2 2 3 3 1	AAA

Note

Note that it is possible that some of the brothers will not get any cities at all. Oh well, no one said it should be fair.

Problem F. Fibonacci's Nightmare

Input file:	`fibonacci.in`
Output file:	`fibonacci.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Define a *random linear recursive sequence (RLRS)* as a sequence of random variables $a_0, a_1, \ldots$ which is generated as follows. First, $a_0 = 1$. Then, for every n starting from 1, choose integers i and j independently and equiprobably from $[0;\ n-1]$, and set $a_n = a_i + a_j$ (note that at this moment, the values of a_0, ..., a_{n-1} are already determined).

For example, $a_1 = a_0 + a_0 = 2$, and a_2 is equally likely to be $a_0 + a_0$, $a_0 + a_1$, $a_1 + a_0$ and $a_1 + a_1$, thus it has 25% probability to be 2, 50% to be 3 and 25% to be 4. After that, a_3 is equiprobably chosen from $a_0 + a_0$, $a_0 + a_1$, $a_0 + a_2$, $a_1 + a_0$, $a_1 + a_1$, $a_1 + a_2$, $a_2 + a_0$, $a_2 + a_1$, $a_2 + a_2$; and so on.

You are to determine the variance of n-th term of RLRS.

The *variance* of a random variable X is defined as $\mathbf{Var}(X) = \mathbf{E}(X - \mathbf{E}(X))^2$ (here $\mathbf{E}(X)$ means *expectation* or *mean value* of the random variable X).

Input

The first line of input contains the integer n ($0 \le n \le 10^6$).

Output

Let the variance of a_n be a rational number equal to U/V when cancelled to lowest terms (that is, U and V are integers, $V > 0$ and the greatest common divisor of U and V is 1). Output the number $X = (U \cdot V^{-1}) \bmod (10^9 + 7)$. That is, X should satisfy the congruence $VX \equiv U$ modulo $(10^9 + 7)$. It is guaranteed that such X exists and is the only root of this equation with $0 \le X < 10^9 + 7$.

Examples

fibonacci.in	fibonacci.out
1	0
2	500000004
5	305555565

Note

a_1 is always equal to 2, so $\mathbf{Var}(a_1) = 0$.

$\mathbf{Var}(a_2) = \frac{1}{2}$.

$\mathbf{Var}(a_5) = \frac{263}{36}$.

Problem G. Guess The String

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

Alice and Bob are playing a game. Alice chooses a string s consisting of lowercase English letters. Bob may ask queries of sort "`? t`", which means "is the string t a **subsequence** of s?". For example, if $s =$ `abc`, the answer for the query "`? ac`" is "`YES`", while for the query "`? bb`" the answer is "`NO`". Total length of t's in Bob's queries must not be too large, otherwise the players will grow bored with the game.

At any point, Bob may claim that he knows the chosen string s, and reveal his guess. If he is right, he wins, otherwise he loses.

Your task is to write a program which interactively asks queries and wins every game while not making too much queries.

Input

At the start of the interaction there is no input for your program.

After each "`?`"-query (refer to the output format), a single line is fed — "`YES`" if t is a **subsequence** of s, and "`NO`" otherwise.

It is known that s is a non-empty string of lowercase English letters. Length of s doesn't exceed 500.

Output

Print descriptions of your queries on a single line each, according to the following format:

- "`? t`" for a subsequence query with the string t,
- "`! t`" for submitting a guess that $s = t$. After making this query, your program must terminate.

Total size of t in '`?`"-queries must not exceed $6 \cdot 10^5$.

Don't forget to flush your output after each query.

Example

standard input	standard output
YES NO NO NO	? a ? b ? c ? aa ! a

Note

The empty lines are only for clarity of the interaction order; no actual line breaks are fed.

Note that the guess in the provided sample case is surely correct if `a`, `b` and `c` letters only are allowed; for the original problem the guess would probably be judged as incorrect as there exist other strings which are in accordance with all queries, for instance, "`ad`".

A string t is a *subsequence* of a string s if it possible to erase some characters of s (possibly none) to obtain t. Changing the order of the remaining characters is not allowed.

Problem H. Hilbert's Maze

Input file:	`hilbert.in`
Output file:	`hilbert.out`
Time limit:	3 seconds
Memory limit:	256 mebibytes

You are playing a computer game "Hilbert's Maze". The point of the game is to find a way out of various mazes. The last level of the game features the most tricky maze, which you find very difficult to escape from. Frustrated by the maze's complexity, you decided to put your programming skills to use.

The maze can be described as a rectangular field divided into identical square cells, with walls along some borders of the cells. Standing in a cell, you can move to any of the adjacent cells (two cells are adjacent if they share a border segment) if there is no wall in your way.

The particular maze you're going to escape from can be constructed using the following recursive pattern: starting from the basic maze T_0 (figure 1), you apply the following transformation (figure 2) k times to obtain the resulting maze T_k. You've introduced the coordinate system as shown on figure 2, with the cell in the lower left corner having coordinates $(1, 1)$. For example, the maze T_2 is shown on figure 3.

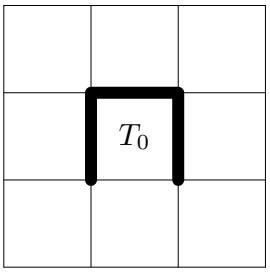


Figure 1

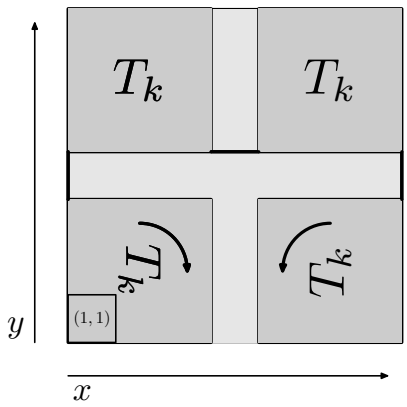


Figure 2

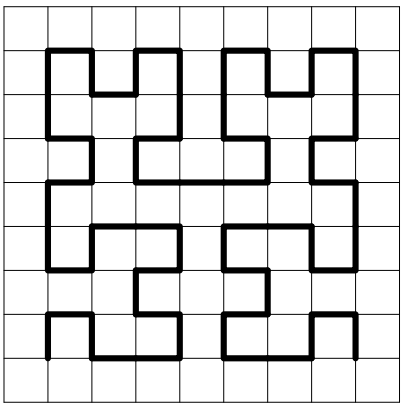


Figure 3

It is possible to escape the boundaries of the maze (that is, the square with corner cells $(1, 1)$ and $(2^{k+1} - 1, 2^{k+1} - 1)$); there are no walls outside the boundaries, and you are free to move along the border or as far as you like from the maze. You have to find the shortest way from some starting cell to some target cell. The starting and target cells are generated randomly each time you try, so you would like to write a program that finds the shortest path for many instances of the problem.

Input

The first line contains an integer T ($1 \le T \le 3 \cdot 10^5$) — the number of test cases.

Next T lines contain descriptions of the test cases. The i-th of these lines contains five space-separated integers k, x_s, y_s, x_t, y_t ($0 \le k \le 30$, $1 \le x_s, y_s, x_t, y_t \le 2^{k+1} - 1$) — the number of transformations used to construct the maze and the coordinates of the starting and the target cells. The cells may coincide.

Output

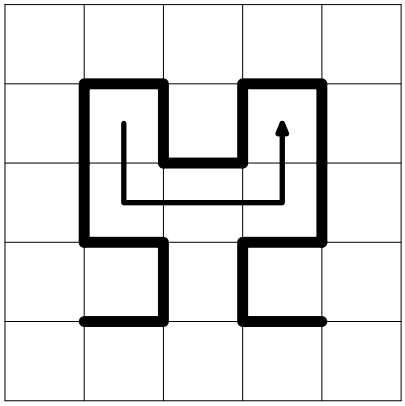
Output T lines; on the i-th of these lines print a single integer — the smallest number of moves needed to get from the starting cell to the target cell. It is guaranteed that a way between the cells exists.

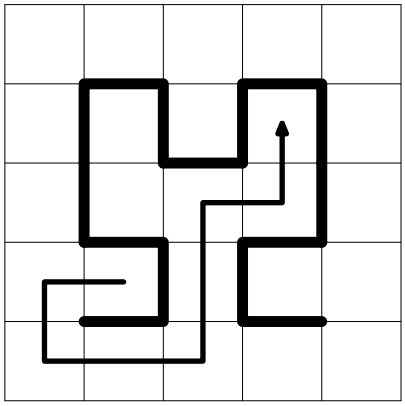
Example

hilbert.in	hilbert.out
3 0 1 1 1 1 1 1 3 3 3 1 1 1 3 3	0 4 8

Note

The second and third samples are illustrated below.





Problem I. Infinite Binary Embedding

Input file:	`infinite.in`
Output file:	`infinite.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Vasya has got himself an *infinite binary tree* T, which can be formally described as follows: T has infinitely many vertices, which are numbered starting from 1. Each vertex of T has two *children — left son* and *right son*. The left son of the vertex i is the vertex $2i$, and the right son of the vertex i is the vertex $2i + 1$.

Also, Vasya has another binary tree G (quite fortunately, a finite one). Every vertex of G is either an *internal vertex* or a *leaf*. An internal vertex has a left son and a right son (and is called the *parent* of its children), and a leaf vertex has no children. Each vertex has exactly one parent, except for one vertex which has no parent; this vertex is called the *root* of the tree G.

Vasya would like to *embed* the tree G in the infinite tree T. Formally, an *embedding* is defined as a function $f : V(G) \to V(T)$ (here $V(X)$ is the set of vertices of the tree X) with the following property: if the vertex v is a left (alternatively: right) son of the vertex u in G, then the vertex $f(v)$ must lie in the subtree of the left (alternatively: right) son of the vertex $f(u)$ in T.

Informally, an embedding is a way to put the vertices of G somewhere on T so that if we draw paths between all the embedded vertices and their children, the drawing looks like G with some of the edges possibly extended downwards (looking at the pictures for the sample cases might help to understand the notion better).

Additionally, for each leaf vertex v Vasya has chosen a number h_v — the *height* of the vertex of T that v should be embedded to (the *height* of a vertex is the number of edges between the vertex and the root of the tree). That is, $\text{height}(f(v)) = h_v$ must hold for every leaf v of the tree G.

Now, Vasya wants to know the number of different embeddings of G in T such that each leaf v is embedded to a vertex with height h_v. As the number may be quite large, find it modulo $10^9 + 7$.

Input

The first line contains one integer n — the number of vertices of G ($1 \le n \le 2000$).

The next n lines describe the tree G and the numbers h_v.

If the i-th vertex is an internal vertex, then the first number in the i-th line will be 0, followed by the numbers of its left and right sons respectively.

If the i-th vertex is a leaf, then the first number in the i-th line will be 1, followed by the number h_i for this leaf. All h_i satisfy $0 \le h_i \le 10^9$.

It is guaranteed that the root of G is the vertex 1.

Output

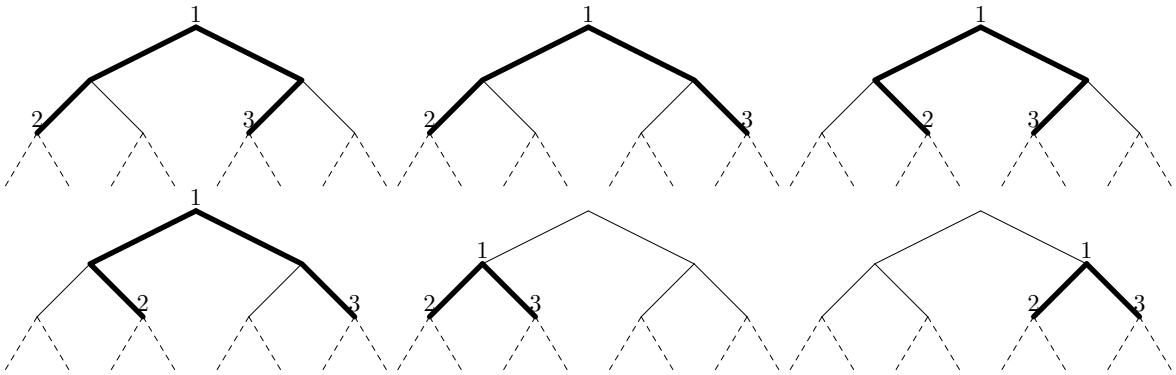
Print one integer — the number of appropriate embeddings modulo $10^9 + 7$.

Examples

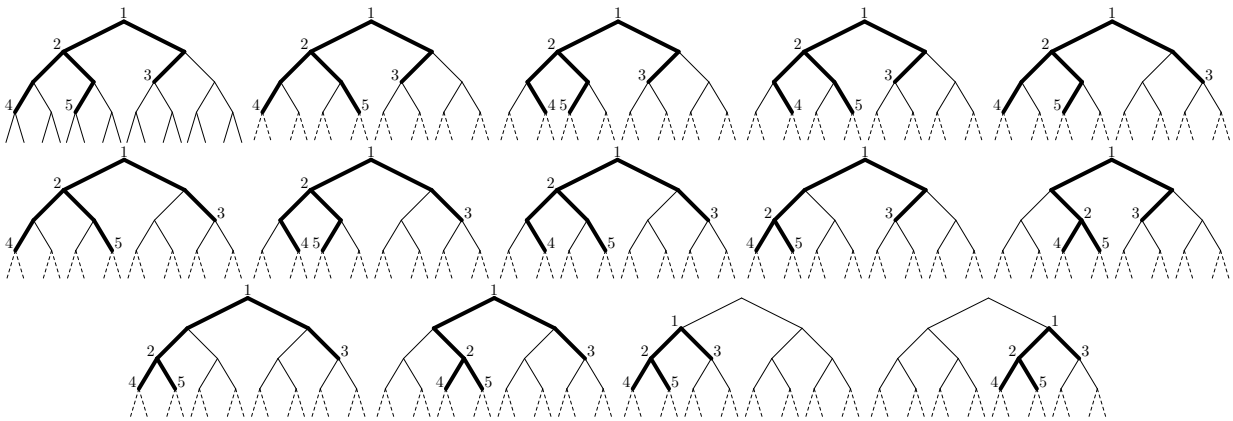
infinite.in	infinite.out
3 0 2 3 1 2 1 2	6
5 0 2 3 0 4 5 1 2 1 3 1 3	14
3 0 2 3 1 0 1 0	0
1 1 0	1

Note

All six possible embeddings for the first sample (note that the root of G does not have to coincide with the root of T):



All fourteen possible embeddings for the second sample:



Problem J. Jitterbug

Input file:	jitterbug.in
Output file:	jitterbug.out
Time limit:	1 second
Memory limit:	256 mebibytes

You have caught a fascinating insect — a jitterbug. For your own amusement, you have drawn a labyrinth (which can be represented as an undirected graph which doesn't contain self-loops or multiple edges) and have taught your bug to move in it. You noticed that after you place the bug at some vertex of the labyrinth, it uses the following moving strategy: it equiprobably chooses an edge that is incident to the current vertex and moves to the other end of this edge; then it chooses a random edge again, and so on.

Usually, you place the bug at some starting vertex in the labyrinth and observe it until it arrives at the chosen finish vertex (of course, there must be at least one path between the starting vertex and the finish vertex). You like to watch as the bug runs around the labyrinth, so you want to construct a labyrinth such that on average it takes the bug the most time to arrive to the finish point.

For example, suppose $n = 3$. It can be shown that the labyrinth with the longest expected number of moves between vertices 1 and 3 is the chain 1–2–3: the average number of moves is four.

Construct a labyrinth on 200 vertices such that it takes the bug at least 10^6 moves on average to reach the vertex 200 starting from the vertex 1.

Input

The first line of input contains two space-separated integer numbers n and b — the number of vertices and the lower limit on average number of moves (that is, your answer will be accepted if the average number of moves is at least b).

This problem has only two testcases:

1. $n = 3$, $b = 4$.
2. $n = 200$, $b = 10^6$.

Output

On the first line print a single integer m — the number of edges in your labyrinth.

Next m lines must describe edges of the labyrinth. On i-th of these lines print two integers — numbers of vertices connected by i-th edge. Remember that the edges are bidirectional.

Your graph must not have self-loops or multiple edges; vertices 1 and n must be connected by at least one path.

Example

jitterbug.in	jitterbug.out
3 4	2 1 2 2 3