icpc global
programming
tools sponsor

Official Problem Set

DO NOT OPEN UNTIL CONTEST BEGINS

2017 ACM ICPC
Asia Regional - Daejeon
Programming Contest

acm
icpc International Collegiate
Programming Contest

icpc.foundation

Problem Set

Please check that you have 12 problems and 23 pages (excluding this cover page and blank pages).

A. Broadcast Stations (2 pages)
B. Connect3 (1 page)
C. Game Map (2 pages)
D. Happy Number (1 page)
E. How Many to Be Happy? (2 pages)
F. Philosopher's Walk (2 pages)
G. Rectilinear Regions (2 pages)
H. Rock Paper Scissors (2 pages)
I. Slot Machines (2 pages)
J. Strongly Matchable (2 pages)
K. Untangling Chain (2 pages)
L. Vacation Plans (3 pages)

The memory limits for the twelve problems are all the same, 512MB.

Problem A
Broadcast Stations
Time Limit: 0.5 Seconds

There is a network of cities where broadcast stations, broadcasting identical information, should be placed on some cities. Each broadcast station has its own transmission power p so that it can broadcast to any city within distance p from it. Here, the distance between two vertices is the number of edges included in the (unique) path between them. In this problem, the network of cities is a tree T with n vertices each of which represents a city.

For a tree T with a set V of n vertices, we will assign a non-negative integer $p(v)$, called a broadcast power, to every vertex v in V such that every vertex u with $p(u) = 0$ is within distance $p(v)$ from some vertex v with $p(v) > 0$. Then we can regard the vertices v with $p(v) > 0$ as broadcast stations of transmission power $p(v)$, and a vertex u with $p(u) = 0$ can hear the broadcast of v, if u is within distance $p(v)$ from v.

The goal of the problem is to find an assignment of the broadcast powers $p(v)$ of vertices v in V described above, minimizing $\sum_{v\in V} p(v)$.

For example, in Figure A.1, two cases of an assignment of broadcast powers to vertices are shown. In the case of Figure A.1 (a), only the vertex 6 has the broadcast power 4 and the other vertices have zero. Then all the vertices with the broadcast power 0 can hear the broadcast of the vertex 6. In the case of Figure A.1 (b), two vertices 3 and 9 have the broadcast powers 2 and 1, respectively. Then all the vertices with the broadcast power 0 can hear the broadcast of either the vertex 3 or 9. This case minimizes the sum of broadcast powers.

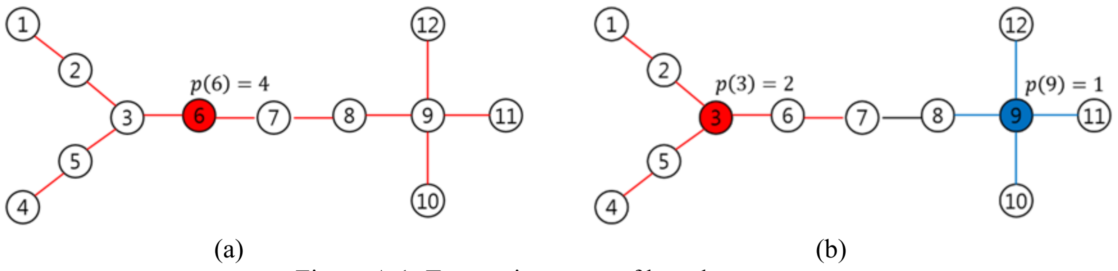


Figure A.1: Two assignments of broadcast powers

Input
Your program is to read from standard input. The first line contains one integer n ($1 \le n \le 5{,}000$) representing the number of vertices of the input tree T. The vertices of T are numbered from 1 to n. Each of the following $n-1$ lines contains two integers a and b ($1 \le a, b \le n$) representing an edge to connect two vertices a and b in T.

Output
Your program is to write to standard output. Print exactly one line which contains an integer that is the minimum sum of broadcast powers among all the possible assignments of broadcast powers to the vertices in T described above.

The following shows sample input and output for two test cases.

Sample Input 1

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

Output for the Sample Input 1

```
2
```

Sample Input 2

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

Output for the Sample Input 2

```
3
```

Problem B
Connect3
Time Limit: 0.5 Seconds

Connect3 is a simplified version of a well-known Connect4 game. Connect3 is a game for two players, black and white, who take turns placing their colored stones in a 4 x 4 grid board shown in Fig.B.1. Each square (or box) in the grid is represented by a pair of numbers (a, b) where a is for a row and b is for a column. The lower left corner of the board is (1, 1), and the upper right corner is (4, 4). Each player selects a column to place a stone which is then placed on the lowest empty square in the column. For example, square (3, 1) is to be taken only when squares (2, 1) and (1, 1) are occupied beforehand. The game ends if three stones with the same color connect in either horizontally, diagonally, or vertically in a row and the player of the color wins.

(4, 1)	(4, 2)	(4, 3)	(4, 4)
(3, 1)	(3, 2)	(3, 3)	(3, 4)
(2, 1)	(2, 2)	(2, 3)	(2, 4)
(1, 1)	(1, 2)	(1, 3)	(1, 4)

Fig.B.1. Board of Connect3. Each grid square is represented by (a, b) where a is for a row and b is for a column.

The game starts by a player placing a black stone on square $(1, x)$. If the game ends by the white player placing a stone on square (a, b), let the final state of the board be s. You are to write a program to find the number of all possible unique states of s. Note that the order of stones placed is irrelevant.

Input
Your program is to read from standard input. The input starts with a line containing an integer x $(1 \leq x \leq 4)$, representing the column of the first stone placed on the board. The next line of input shows two integers, a and b for square (a, b) which is the position of the last stone placed on the board.

Output
Your program is to write to standard output. Print exactly one number that corresponds to the answer.

The following shows sample input and output for two test cases.

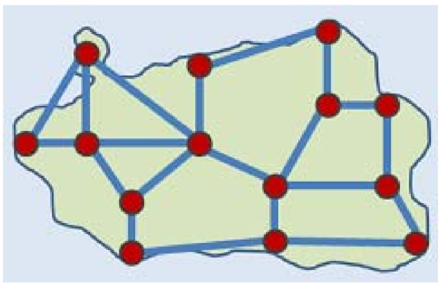
Sample Input 1	Output for the Sample Input 1
2	516
2 3	

Sample Input 2	Output for the Sample Input 2
3	177
4 4	

BLANK

PAGE

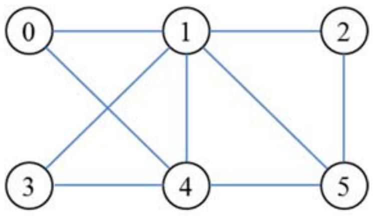
Problem C
Game Map
Time Limit: 1 Second



The ICPC-World is the most popular RPG game for ACM-ICPC contestants, whose objective is to conquer the world. A map of the game consists of several cities. There is at most one road between a pair of cities. Every road is bidirectional. If there is a road connecting two cities, they are called neighbors. Each city has one or more neighbors and all cities are connected by one or more roads. A player of the game can start at any city of the world. After conquering a city that the player stays, the player can proceed to any neighbor city which is the city the player to conquer at the next stage.

Chansu, a mania of the game, enjoys the game in a variety of ways. He always determines a list of cities which he wants to conquer before he starts to play the game. In this time, he wants to choose as many cities as possible under the following conditions: Let $(c_0, c_1, \cdots, c_{m-1})$ be a list of cities that he will conquer in order. All cities of the list are distinct, i.e., $c_i \neq c_j$ if $i \neq j$, c_i and c_{i+1} are neighbors to each other, and the number of neighbors of c_{i+1} is greater than the number of neighbors of c_i for $i = 0, 1, \dots, m-2$.

For example, let's consider a map of the game shown in the figure below. There are six cities on the map. The city 0 has two neighbors and the city 1 has five neighbors. The longest list of cities satisfying the above conditions is $(2, 5, 4, 1)$ with 4 cities.



In order to help Chansu, given a map of the game with n cities, write a program to find the maximum number of cities that he can conquer, that is, the length of the longest list of cities satisfying the above conditions.

Input
Your program is to read from standard input. The input starts with a line containing two integers, n and m $(1 \leq n \leq 100{,}000,\ \mathrm{n} - 1 \leq m \leq 300{,}000)$, where n is the number of cities on the game map and m is the number of roads. All cities are numbered from 0 to $n-1$. In the following m lines, each line contains two integers i and j $(0 \leq i \neq j \leq n-1)$ which represent a road connecting two cities i and j.

Output
Your program is to write to standard output. Print exactly one line. The line should contain the maximum number of cities which Chansu can conquer.

The following shows sample input and output for two test cases.

Sample Input 1

```
6 9
0 1
0 4
1 2
1 3
1 4
1 5
2 5
3 4
4 5
```

Output for the Sample Input 1

```
4
```

Sample Input 2

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

Output for the Sample Input 2

```
5
```

Problem D
Happy Number
Time Limit: 0.2 Seconds

Consider the following function f defined for any natural number n:

$f(n)$ is the number obtained by summing up the squares of the digits of n in decimal (or base-ten).

If $n = 19$, for example, then $f(19) = 82$ because $1^2 + 9^2 = 82$.

Repeatedly applying this function f, some natural numbers eventually become 1. Such numbers are called *happy numbers*. For example, 19 is a happy number, because repeatedly applying function f to 19 results in:

$$f(19) = 1^2 + 9^2 = 82$$
$$f(82) = 8^2 + 2^2 = 68$$
$$f(68) = 6^2 + 8^2 = 100$$
$$f(100) = 1^2 + 0^2 + 0^2 = 1$$

However, not all natural numbers are happy. You could try 5 and you will see that 5 is not a happy number. If n is not a happy number, it has been proved by mathematicians that repeatedly applying function f to n reaches the following cycle:

$$4 \to 16 \to 37 \to 58 \to 89 \to 145 \to 42 \to 20 \to 4.$$

Write a program that decides if a given natural number n is a happy number or not.

Input
Your program is to read from standard input. The input consists of a single line that contains an integer, n $(1 \leq n \leq 1{,}000{,}000{,}000)$

Output
Your program is to write to standard output. Print exactly one line. If the given number n is a happy number, print out HAPPY; otherwise, print out UNHAPPY.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1
19	HAPPY

Sample Input 2	Output for the Sample Input 2
5	UNHAPPY

BLANK

PAGE

Problem E
How Many to Be Happy?
Time Limit: 0.5 Seconds

Let G be a connected simple undirected graph where each edge has an associated weight. Let's consider the popular MST (Minimum Spanning Tree) problem. Today, we will see, for each edge e, how much modification on G is needed to make e part of an MST for G. For an edge e in G, there may already exist an MST for G that includes e. In that case, we say that e is *happy* in G and we define $H(e)$ to be 0. However, it may happen that there is no MST for G that includes e. In such a case, we say that e is *unhappy* in G. We may remove a few of the edges in G to make a *connected* graph G' in which e is happy. We define $H(e)$ to be the minimum number of edges to remove from G such that e is happy in the resulting graph G'.

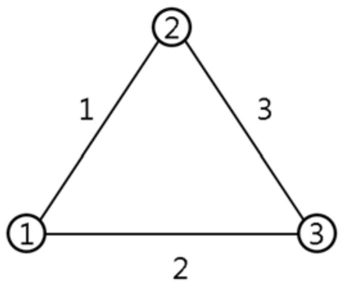


Figure E.1. A complete graph with 3 nodes.

Consider the graph in Figure E.1. There are 3 nodes and 3 edges connecting the nodes. One can easily see that the MST for this graph includes the 2 edges with weights 1 and 2, so the 2 edges are happy in the graph. How to make the edge with weight 3 happy? It is obvious that one can remove any one of the two happy edges to achieve that.

Given a connected simple undirected graph G, your task is to compute $H(e)$ for each edge e in G and print the total sum.

Input
Your program is to read from standard input. The first line contains two positive integers n and m, respectively, representing the numbers of vertices and edges of the input graph, where $n \leq 100$ and $m \leq 500$. It is assumed that the graph G has n vertices that are indexed from 1 to n. It is followed by m lines, each contains 3 positive integers u, v, and w that represent an edge of the input graph between vertex u and vertex v with weight w. The weights are given as integers between 1 and 500, inclusive.

Output
Your program is to write to standard output. The only line should contain an integer S, which is the sum of $H(e)$ where e ranges over all edges in G.

The following shows sample input and output for two test cases.

Sample Input 1

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

Output for the Sample Input 1

```
1
```

Sample Input 2

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

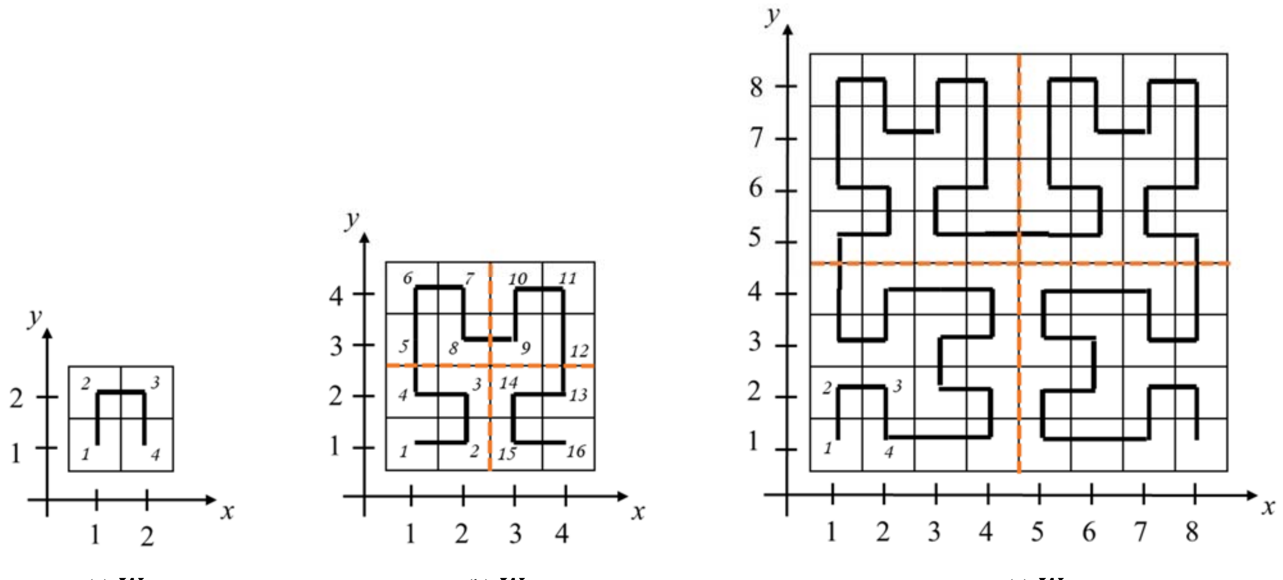
Output for the Sample Input 2

```
3
```

Problem F
Philosopher’s Walk
Time Limit: 0.5 Seconds

In Programming Land, there are several pathways called Philosopher’s Walks for philosophers to have a rest. A Philosopher’s Walk is a pathway in a square-shaped region with plenty of woods. The woods are helpful for philosophers to think, but they planted so densely like a maze that they lost their ways in the maze of woods of a Philosopher’s Walk.

Fortunately, the structures of all Philosopher’s Walks are similar; the structure of a Philosopher’s Walk is designed and constructed according to the same rule in a 2^k meter square. The rule for designing the pathway is to take a right-turn in 90 degrees after every 1-meter step when k is 1, and the bigger one for which the integer k is greater than 1 is built up using four sub-pathways with $k-1$ in a fractal style. Figure F.1 shows three Philosopher’s Walks for which k is 1, 2, and 3. The Philosopher’s Walk W_2 consists of four W_1 structures with the lower-left and the lower-right ones are 90 degree rotated clockwise and counter-clockwise, respectively; the upper ones have the same structure with W_1. The same is true for any W_k for which the integer k is greater than 1. This rule has been devised by a mathematical philosopher David Hilbert (1862 – 1943), and the resulting pathway is usually called a HILBERT CURVE named after him. He once talked about a space filling method using this kind of curve to fill up a square with 2^k sides, and every Philosophers’ Walk is designed according to this method.



(a) $\boldsymbol{W_1}$ (b) $\boldsymbol{W_2}$ (c) $\boldsymbol{W_3}$

Figure F.1. Three Philosopher’s Walks with sizes (a) $\mathbf{2^1} = 2$, (b) $\mathbf{2^2} = 4$, and (c) $\mathbf{2^3} = 8$, repectively.

Tae-Cheon is in charge of the rescue of the philosophers lost in Philosopher’s Walks using a hot air balloon. Fortunately, every lost philosopher can report Tae-Cheon the number of meter steps he has taken, and Tae-Cheon knows the length of a side of the square of the Philosopher’s Walk. He has to identify the location of the lost philosopher, the (x, y) coordinates assuming that the Philosopher’s Walk is placed in the 1st quadrant of a Cartesian plain with one meter unit length. Assume that the coordinate of the lower-left corner block is

$(1, 1)$. The entrance of a Philosopher's Walk is always at $(1, 1)$ and the exit is always $(n, 1)$ where n is the length of a side. Also assume that the philosopher have walked one meter step when he is in the entrance, and that he always go forward to the exit without back steps.

For example, if the number of meter-steps taken by a lost philosopher in the Philosopher's Walk in W_2 in Figure F.1(b) is 10, your program should report $(3, 4)$.

Your mission is to write a program to help Tae-Cheon by making a program reporting the location of the lost philosopher given the number of meter-steps he has taken and the length of a side of the square of the Philosopher's Walk. Hurry! A philosopher urgently needs your help.

Input

Your program is to read from standard input. The input consists of a single line containing two positive integers, n and m, representing the length of a side of the square of the Philosopher's Walk and the number of meter-steps taken by the lost philosopher, respectively, where $n = 2^k$ and $m \leq 2^{2k}$ for an integer k satisfying $0 < k \leq 15$.

Output

Your program is to write to standard output. The single output line should contain two integers, x and y, separated by a space, where (x, y) is the location of the lost philosopher in the given Philosopher's Walk.

The following shows sample input and output for two test cases.

Sample Input 1	**Output for the Sample Input 1**
`4 10`	`3 4`

Sample Input 2	**Output for the Sample Input 2**
`8 19`	`2 6`

Problem G
Rectilinear Regions
Time Limit: 0.5 Seconds

A rectilinear path connecting two points in the plane is a path consisting of only horizontal and vertical line segments. A rectilinear path is said to be *monotone* with respect to the x-axis (resp., y-axis) if and only if its intersection with every vertical (resp., horizontal) line is either empty or a contiguous portion of that line. A *staircase* is a rectilinear path if it is monotone to both the x-axis and the y-axis, and a staircase is *unbounded* if it starts and ends with a semi-infinite horizontal segment, i.e., a segment that extends to infinity on both ends of the x-axis. Note that staircases can be either increasing or decreasing, depending on whether they go up or down as we move along them from left to right on the x-axis. A staircase with n vertical line segments is called a staircase with n steps.

Considering two unbounded staircases L and U, there can be several or no *closed rectilinear regions* bounded by staircases L and U. Among the closed rectilinear regions, some regions are bounded by a staircase L to the bottom and by a staircase U to the top. For example, in the following figure, the two regions colored yellow are that kind of closed rectilinear regions. We would like to compute the total area of such regions.

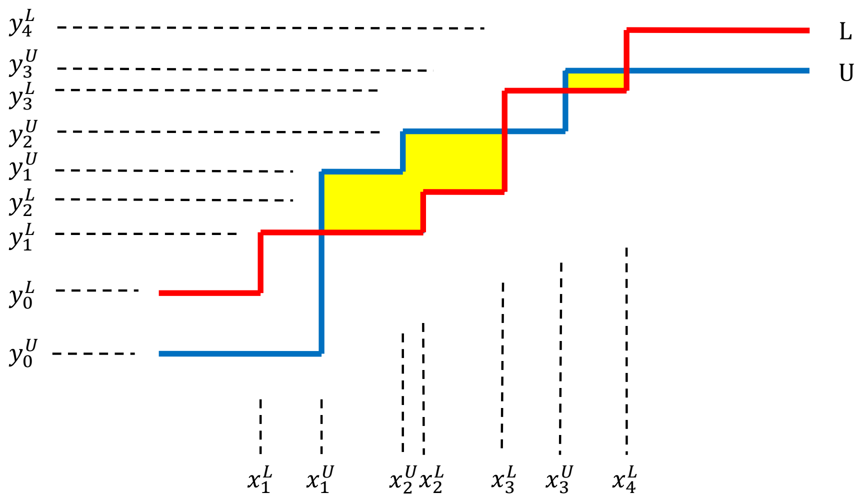


Figure G.1. Two staircases L and U, where U has 3-steps and L has 4-steps. The two yellow colored regions are closed rectilinear regions bounded by a staircase L to the bottom, and a staircase U to the top. The $\boldsymbol{x_i^L}, \boldsymbol{y_i^L}$ (resp., $\boldsymbol{x_i^U}, \boldsymbol{y_i^U}$) are the $\boldsymbol{x}$-, $\boldsymbol{y}$-coordinates of corner points of the staircase L (resp., U).

The geometry for an n-step staircase is represented by the x-, y-coordinates of corner points of the staircase in the following order:

$$y_0\ x_1\ y_1\ x_2\ y_2\ \cdots\ x_n\ y_n \qquad \text{-------------------------------} \qquad (1)$$

where $x_1 < x_2 < \cdots < x_n$ for x-coordinates of vertical line segments, and $y_0 < y_1 < \cdots < y_n$ for y-coordinates of horizontal line segments of an increasing staircase or $y_0 > y_1 > \cdots > y_n$ for a decreasing staircase.

For example, given a 4-step staircase L represented with

6 2 9 11 11 15 16 21 19

and a 3-step staircase U represented with

3 6 12 10 14 18 17

the number of bounded rectilinear regions is 2 and the total area of the regions is 32 (see figure G.1).

Given two unbounded staircases L and U that *all x-coordinates represented in (1) of corner points of both L and U are unique, and all y-coordinates represented in (1) of corner points of both L and U are unique*, compute the total area of bounded rectilinear regions that bounded by L to the bottom of the regions and by U to the top of the regions.

Input

Your program is to read from standard input. The first line contains two positive integers n and m, respectively, representing the number of steps of unbounded staircases L and U, where $1 \le n, m \le 25{,}000$. The second (resp., third) line contains $2n + 1$ (resp., $2m + 1$) integers representing the x-, y-coordinates of corner points of the staircase L (resp., U), and the integers are sequenced in the order of the notation (1). The coordinates are represented with non-negative integers less than or equal to 50,000.

Output

Your program is to write to standard output. The first line should contain two integers k and w, where k represents the number of closed rectilinear regions and w represents the total area of those regions. If there is no such regions, then your program should write 0 for both k and w.

The following shows sample input and output for two test cases.

Sample Input 1

```
4 3
6 2 9 11 11 15 16 21 19
3 6 12 10 14 18 17
```

Output for the Sample Input 1

```
2 32
```

Sample Input 2

```
4 3
9 1 7 3 5 5 3 7 1
0 2 2 4 4 6 6
```

Output for the Sample Input 2

```
0 0
```

Sample Input 3

```
1 1
1 50000 50000
0 0 49999
```

Output for the Sample Input 3

```
1 2499900000
```

Problem H
Rock Paper Scissors
Time Limit: 1 Second

There is a Rock Paper Scissors (RPS) machine which generates Rock, Paper, or Scissors randomly. You also have a similar small Rock Paper Scissors machine. Before the game, the RPS machine will generate a list of its choice of Rock, Paper, or Scissors of the length n and your machine also will generates a list of its choice of the length m. That is, you know the whole list of the RPS's choices and you have the list of your machine's choices. Of course, each choice of the machines is one of the three options (Rock, Paper, or Scissors).

Now, you start playing Rock Paper Scissors game. In every match, you compare the list of RPS's choice and the list of your machine's in sequence and decide whose machine would win. However, only you may skip some RPS's choices to find the position to get the most winning points of your machine. After you decide to start match you cannot skip the match till the end of the match. 'R' stands for Rock, 'P' stands for Paper, and 'S' stands for Scissors.

For example, suppose that the RPS's list is "RSPPSSSRRPPR" and your machine's list is "RRRR". To get the most winning points, you should start the match after skipping three RPS's choices or four RPS's choices. (See Figure H.1.) Then, you can win in three matches. The draw case is not considered.

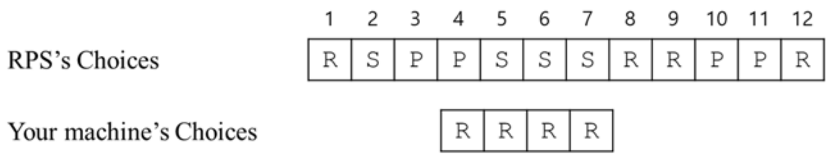


Figure H.1. The most winning position against RPS machine when $n = 12$ and $m = 4$.

Given the list of RPS's choices and the list of your choices, find the position to get the maximum number of wining matches.

Input
Your program is to read from standard input. The first line contains two positive integers n and $(1 \leq m < n \leq 100{,}000)$, where n is the length of the string for RPS machine and m is the length of the string for your machine. Following the first line contains the list of choices of RPS machine and the second line contains the list of choices of your machine.

Output
Your program is to write to standard output. The first line should contain an integer indicating the maximum number of wining matches

The following shows sample input and output for four test cases.

Sample Input 1	Output for the Sample Input 1
12 4 RSPPSSSRRPPR RRRR	3

Sample Input 2	Output for the Sample Input 2
12 3 RRRRRRRRRRRR SSS	0

Sample Input 3	Output for the Sample Input 3
12 4 PPPRRRRRRRRR RSSS	2

Sample Input 4	Output for the Sample Input 3
12 4 RRRRRRRRRSSS RRRS	3

Problem I
Slot Machines
Time Limit: 2 Seconds

Slot machines are popular game machines in casinos. The slot machine we are considering has six places where a figure appears. By combination of figures, one may earn or lose money. There are ten kinds of figures, so we will represent a figure with a number between 0 and 9. Then we can use a six-digit number $w = w_1w_2w_3w_4w_5w_6$ where $0 \le w_1, w_2, w_3, w_4, w_5, w_6 \le 9$ to represent one possible outcome of the slot machine. It is guaranteed that 000000 will never appear.

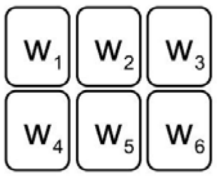


Figure I.1. The layout of a slot machine.

Old slot machines were made up with mechanical components, but nowadays they were replaced by PC-based systems. This change made one critical flaw: they are based on pseudo-random number generators and the outcome sequences of a slot machine are periodic. Let $T[i]$ be the i-th outcome of a slot machine. At first, there is a truly random sequence of length $k, T[1], T[2], \dots, T[k]$. Then there exists one positive number p such that $T[i + p] = T[i]$ for all possible values of $i(> k)$. Once an attacker can find out the exact values of k and p, he or she can exploit this fact to beat the casino by betting a lot of money when he or she knows the outcome with a good combination in advance.

For example, you have first six numbers of outcome sequences: 612534, 3157, 423, 3157, 423, and 3157. Note that we can remove first 0's. Therefore, 3157 represents 003157 and 423 represents 000423. You want to know its tenth number. If you know the exact values of k and p, then you can predict the tenth number. However, there are many candidates for k and p: one extreme case is k=5 and p=1, and another is k=0 and p=6. The most probable candidate is the one where both k and p are small. So, our choice is the one with the smallest k+p. If there are two or more such pairs, we pick the one where p is the smallest. With our example, after some tedious computation, we get k=1 and p=2.

Assume that you have n consecutive outcomes of a slot machine, $T[1], T[2], \dots, T[n]$. Write a program to compute the values of k and p satisfying the above-mentioned condition.

Input
Your program is to read from standard input. The first line contains a positive integer n ($1 \le n \le 1{,}000{,}000$), representing the length of numbers we have observed up to now in the outcome sequence. The following line contains n numbers. Each of these numbers is between zero and 999,999.

Output
Your program is to write to standard output. Print two integers k and p in one line.

The following shows sample input and output for two test cases.

Sample Input 1

```
6
612534 3157 423 3157 423 3157
```

Output for the Sample Input 1

```
1 2
```

Sample Input 2

```
9
1 2 1 3 1 2 1 3 1
```

Output for the Sample Input 2

```
0 4
```

Problem J
Strongly Matchable
Time Limit: 3 Seconds

Let G be a simple undirected graph with n vertices, whose vertex and edge sets are denoted by $V(G)$ and $E(G)$, respectively. Two edges of G are said to be *adjacent* if they share a common vertex. Similarly, two vertices of G are said to be *adjacent* if they share a common edge, in which case the common edge joins the two vertices; an edge and a vertex on that edge are called *incident*. A subset M of $E(G)$ is called a *matching* of G if no two edges in M are adjacent; M is called a *perfect matching* if every vertex of G is incident to exactly one edge of M. So, a matching M of G is perfect if and only if $|M| = \frac{n}{2}$.

The existence of a perfect matching in G can be decided in polynomial time, thanks to a polynomial-time algorithm for finding a maximum matching, a matching that contains the maximum number of edges. Besides, there are two more interesting problems on the existence of a perfect matching in G:

- Given a partition of $V(G)$ into S and T with $|S| = |T| = \frac{n}{2}$, does G has a perfect matching in which every edge joins a vertex in S and a vertex in T?
- For every partition of $V(G)$ into S and T with $|S| = |T| = \frac{n}{2}$, does G has a perfect matching in which every edge joins a vertex in S and a vertex in T?

From the well-known Hall's marriage theorem, we can derive a condition that characterizes the existence of a required perfect matching for the first question as follows: Let G' be the spanning subgraph of G with the edges joining vertices both in S or both in T being deleted, i.e., $V(G') = V(G)$ and $E(G') = \{(u, v) \in E(G) \mid \text{either } u \in S \text{ and } v \in T \text{ or } v \in S \text{ and } u \in T\}$. Then, G has a required perfect matching between S and T if and only if G' has a perfect matching. Moreover, the Hall's theorem leads to that G' has a perfect matching if and only if $|N(X)| \geq |X|$ for every subset X of S, where $N(X)$ denotes the neighborhood of X, i.e., the set of all vertices in T adjacent to some vertex of X. The question, of course, can be answered in polynomial time, also thanks to a maximum matching algorithm that runs in polynomial time.

Is there an efficient algorithm to answer the second question? A graph that admits a positive answer for the second question is called *strongly matchable*; that is, a graph G is *strongly matchable* if G has a perfect matching in which each edge joins two vertices, one in S and the other in T, for every partition of $V(G)$ into S and T with $|S| = |T| = \frac{n}{2}$. For example, the graph shown in Figure J.1 (a) is strongly matchable because there is a perfect matching for each of the three partitions up to symmetry: $M = \{(1,4), (2,5), (3,6)\}$ for $S = \{1,2,3\}$ and $T = \{4,5,6\}$; $M = \{(1,3), (2,5), (4,6)\}$ for $S = \{1,2,4\}$ and $T = \{3,5,6\}$; $M = \{(1,3), (2,5), (6,4)\}$ for $S = \{1,2,6\}$ and $T = \{3,4,5\}$. However, the graph of (b) is not strongly matchable because there is no perfect matching between $S = \{1,2,4\}$ and $T = \{3,5,6\}$. Your job is to write an efficient running program for deciding whether or not an input graph with an even number of vertices is strongly matchable.

Input

Your program is to read from standard input. The first line contains two positive integers n and m, respectively, representing the numbers of vertices and edges of the input graph, where n is even, $n \leq 100$, and $m \leq \frac{n(n-1)}{2}$. It is followed by m lines, each contains two positive integers u and v representing an edge between the vertices u and v of the input graph. It is assumed that the vertices are indexed from 1 to n.

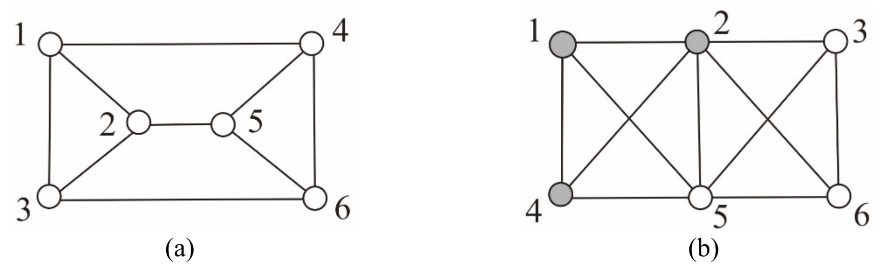


Figure J.1: The graph shown in (a) is strongly matchable, but the graph of (b) is not.

Output

Your program is to write to standard output. Print exactly one integer in a line. If the input graph is strongly matchable, the integer should be `1`; otherwise, the integer should be `-1`.

The following shows sample input and output for two test cases.

Sample Input 1

```
6 9
1 4
4 6
6 3
3 1
1 2
3 2
4 5
6 5
2 5
```

Output for the Sample Input 1

```
1
```

Sample Input 2

```
6 11
1 2
2 3
3 6
6 5
5 4
4 1
2 5
1 5
2 4
2 6
3 5
```

Output for the Sample Input 2

```
-1
```

Problem K
Untangling Chain
Time Limit: 0.5 Seconds

A rectilinear chain is an ordered sequence that alternates horizontal and vertical segments as in Figure K.1. You are given a rectilinear chain whose first segment starts from the origin $(0,0)$ and goes to the right. Such a rectilinear chain of n edges can be described as a sequence of n pairs (l_k, t_k) where l_k is the length of the k-th edge e_k and t_k denotes the turning direction from the k-th edge e_k to the $(k+1)$-st edge e_{k+1}. For $1 \leq k < n$, if the chain turns left from e_k to e_{k+1}, then $t_k = 1$, and if it turns right, then $t_k = -1$. For $k = n$, t_k is set to be zero to indicate that e_k is the last edge of the chain. For example, a rectilinear chain of six pairs shown in Figure 1(a) is described as a sequence of six pairs $(4,1), (5,-1), (2,-1), (2,-1), (4,1), (5,0)$.

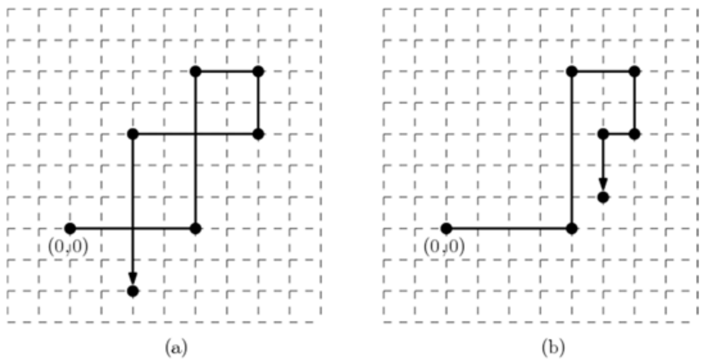


Figure K.1. (a) A non-simple chain. (b) An untangled simple chain

You would already notice that the rectilinear chain drawn by the given description is not simple. A chain is *simple* if any two edges in the chain have no intersection except at the end points shared by adjacent edges. To represent the chain in a more succinct way, you want to make it simple if it is not simple. In other words, you need to untangle a given rectilinear chain to a simple chain by modifying the length of its edges. But the length of each edge in the resulting chain must be at least 1 and at most n, and the turning directions must be kept unchanged. The chain shown in Figure K.1(b) shows one of possible modifications for the non-simple chain given in Figure K.1(a), and its description is $(4,1), (5,-1), (2,-1), (2,-1), (1,1), (2,0)$.

Given a description of a rectilinear chain, you should write a program to untangle the rectilinear chain.

Input
Your program is to read from standard input. The first line contains an integer, n $(1 \leq n \leq 10{,}000)$, representing the number of edges of an input rectilinear chain. In the following n lines, the k-th line contains two integers l_k and t_k for the edge e_k, separated by a single space, where l_k $(1 \leq l_k \leq 10{,}000)$ is the length of e_k, and t_k is the turning direction from e_k to e_{k+1}; $t_k = 1$ if it is the left turn and $t_k = -1$ if it is the right turn for $1 \leq k < n$, and $t_k = 0$ for $k = n$.

Output
Your program is to write to standard output. The first line should contain n positive integers, representing the length of the edges of your untangled simple chain according to the edge order of the input chain. Each length should be at least 1 and at most n. Note that you do not need to output the turning directions because the

turning directions of the simple chain is identical to the ones of the input chain. You can assume that any rectilinear chain described in the input can be untangled with the edge length condition.

The following shows sample input and output for two test cases.

Sample Input 1	**Output for the Sample Input 1**

6
4 1
5 -1
2 -1
2 -1
4 1
5 0
	4 5 2 2 1 2

Sample Input 2	**Output for the Sample Input 2**

6
3 1
3 1
2 1
4 1
1 1
3 0
	2 3 2 2 1 1

Problem L
Vacation Plans
Time Limit: 1 Second

A group of people plan to have a vacation in a remote island near the equator over the winter holiday. All members of the group live in different countries and the destination island is only reachable via airplane. Therefore, each member has to go to their own country's airport to take a flight to the destination island. We assume that each country has only one airport. Now, for the sake of holiday spirit, all group members agree to start the journey on the same day from their home cities. Also, they plan to be at their country's airports on the same day, which is not necessarily the first day of their travel. However, the airports might not be in each member's home city, so some members may have to travel to another city over the course of a few days. On the first day of the winter holiday, all members are in their respective home cities. Then, every day, each member has to individually decide between traveling to an adjacent city (meaning that the two cities are connected by a road), or staying the day in the city they are currently in. Since the travelling cost between two adjacent cities and the cheapest hotel price in each city are already known to the world, one knows exactly how much it will cost either to move to an adjacent city or to stay in that city for each day. All members want to have as much money as possible for the vacation on the island, so they pool their money together and decide to calculate the travel plans as a group. Their goal is that all the members end up at their designated countries' airports on the same day, while spending the least amount of money.

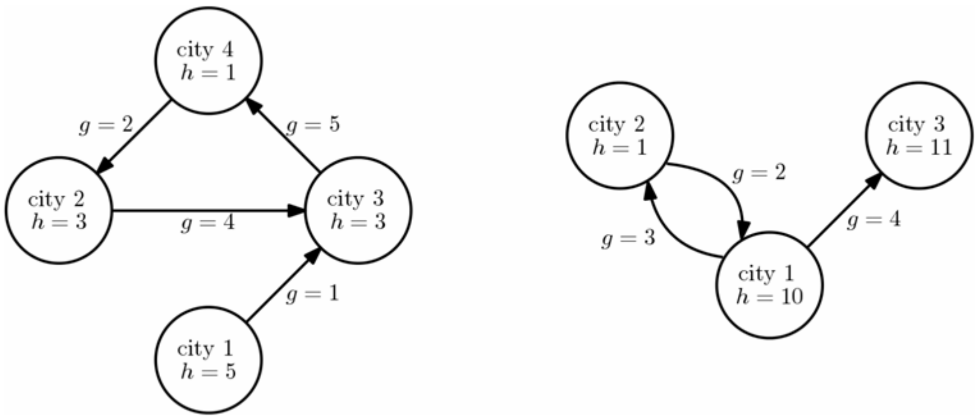


Figure L.1. Two members' country layouts, where the designated airports are in cities 4 and 3, respectively. The home city is 1 for both. $\boldsymbol{g}$ denotes the travelling cost between two cites and $\boldsymbol{h}$ denotes the cheapest hotel price.

Consider an example in Figure L.1 with two members and their designated airports being in cities 4 and 3, respectively. The cheapest travel plans with both members starting in their hometowns (always city 1) would be: (day 1) member-1 moves to city 3, member-2 moves to city 2; (day 2) member-1 moves to city 4, member-2 moves to city 1; (day 3) member-1 stays at city 4, member-2 moves to city 3. This has cost $(1 + 5 + 1) + (3 + 2 + 4) = 16$.

Note that the travelling cost between two cites is not necessary symmetric. Additionally, no city has a road connecting it to itself. You can always assume that, in each country, there is at least one path from home to the designated airport.

You should write a program that finds the minimum cost required to get all members from their home city to their country's designated airport such that everyone is at the airport on the same day. Note that every day, one has to either move to an adjacent city or stays at the current city hotel.

Input

Your program is to read from standard input. The first line contains a single integer $1 \leq p \leq 3$ denoting the number of people in the group that will be going on vacation (and therefore also the number of countries to be considered). Then the next input represents each country as follows: The next line consist of two integers $1 \leq n \leq 50$ and $n - 1 \leq m \leq 4 * n$, corresponding to the number of cities and roads in the country. The next n lines contain exactly one integer, $0 \leq h \leq 1{,}000{,}000$, representing the cheapest hotel cost of each city (the costs are given in order from city 1 to n for each country). The next m lines contain the road information. Each road is represented by three integers, separated by spaces: $1 \leq u, v \leq n$ and $0 \leq g \leq 1{,}000{,}000$, which are the cities at the start and the end of the road, respectively, and the cost to travel on the road from u to v. Finally, one more line is given containing a single integer $1 \leq a \leq n$, denoting the city containing that country's airport. In each country, city 1 is each one's home city.

Output

Your program is to write to standard output. You should output exactly one line containing a single integer equal to the minimum cost required to get all members from their home city to their country's designated airport such that everyone is at the airport on the same day.

The following shows sample input and output for two test cases.

Sample Input 1

```
2
4 4
5
3
3
1
1 3 1
2 3 4
3 4 5
4 2 2
4
3 3
10
1
11
1 2 3
1 3 4
2 1 2
3
```

Output for the Sample Input 1

```
16
```

Sample Input 2

```
2
4 4
2
8
15
1
1 2 5
2 3 7
3 4 10
4 1 3
3
5 4
```

Output for the Sample Input 2

```
32
```

1	
1	
1	
1	
1	
1 2 3	
2 3 5	
3 4 7	
4 5 1	
5	