Telefrogs

Input File	Output File	Time Limit	Memory Limit
standard input	standard output	1 second	256 MiB

Opal is a scientist studying the movement patterns of a particular species of supernatural amphibian, called the teleporting frog, or *telefrog* for short. Telefrogs are just like normal frogs, but instead of hopping, they *teleport*[1].

There is a colony of K telefrogs living together in a pond that Opal has been studying for D days. The pond contains N lily-pads numbered from 1 to N which the telefrogs like to sit on. Every frog was sitting on lily-pad 1 before Opal began studying the colony.

- At the start of each day, each telefrog *may* choose to teleport to another lily-pad in the pond.
- At the end of each day, Opal records the number of frogs on each lily-pad. In particular, there are exactly c_{ij} frogs on the j-th lily-pad during the i-th day.

No new frogs joined the colony and no frog ever went missing during her study.

At the end of her study, Opal realised that some of the K frogs might actually be *impostor frogs*, who do not have the ability to teleport! She found $N-1$ hidden one-way tunnels between pairs of lily-pads. The i-th tunnel allows impostors on lily-pad a_i to travel to lily-pad b_i ($a_i < b_i$). It is possible to travel from lily-pad 1 to any other lily-pad through a sequence of tunnels.

Every night, the impostors, as they lack the ability to teleport, may travel to another lily-pad through a sequence of tunnels.

Your task is to help Opal determine the *maximum* number of impostors there could be.

Subtasks and Constraints

For all subtasks, you are guaranteed that:

- $2 \le N \le 1000$, $1 \le K \le 10^9$ and $2 \le D \le 200$
- $0 \le c_{ij} \le K$, for all i and j.
- $c_{i1} + c_{i2} + \ldots + c_{iN} = K$, for all i. That is, the number of frogs observed on each day is K.
- $1 \le a_i < b_i \le N$, for all i.
- It is possible to travel from lily-pad 1 to any other lily-pad through a sequence of tunnels.

Additional constraints for each subtask are given below.

Subtask	Points	Additional constraints
1	14	$D = 2$ and $a_i + 1 = b_i$, for all i.
2	26	$a_i + 1 = b_i$, for all i.
3	16	$D = 2$
4	13	$a_i = 1$, for all i.
5	31	No additional constraints.

[1]Nobody knows how the frogs actually teleport. Seems sus.

Input

- The first line of input contains the integers N, K and D.
- The next $N - 1$ lines describe the one-way tunnels. The i-th line contains a_i and b_i.
- The next D lines of input contain N integers each. The j-th integer on the i-th such line is c_{ij}.

Output

Output a single integer, the maximum number of impostors that could have been among the telefrogs.

Sample Input 1

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

Sample Output 1

```
2
```

Sample Input 2

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

Sample Output 2

```
0
```

Explanation

In Sample Case 1, there could have been two impostors:

- The first impostor travels to lily-pad 2 on the first day, does nothing on the second day, and does nothing on the third day.
- The second impostor does nothing on the first day, does nothing on the second day, and travels to lily-pad 6 via lily-pad 3 on the third day.
- The first telefrog does nothing on the first day, teleports to lily-pad 2 on the second day, and teleports to lily-pad 1 on the third day.
- The second telefrog teleports to lily-pad 6 on the first day, teleports to lily-pad 2 on the second day, and teleports to lily-pad 4 on the third day.

It can be shown that there could not have been more than two impostors.

In Sample Case 2, none of the frogs could have been impostors.

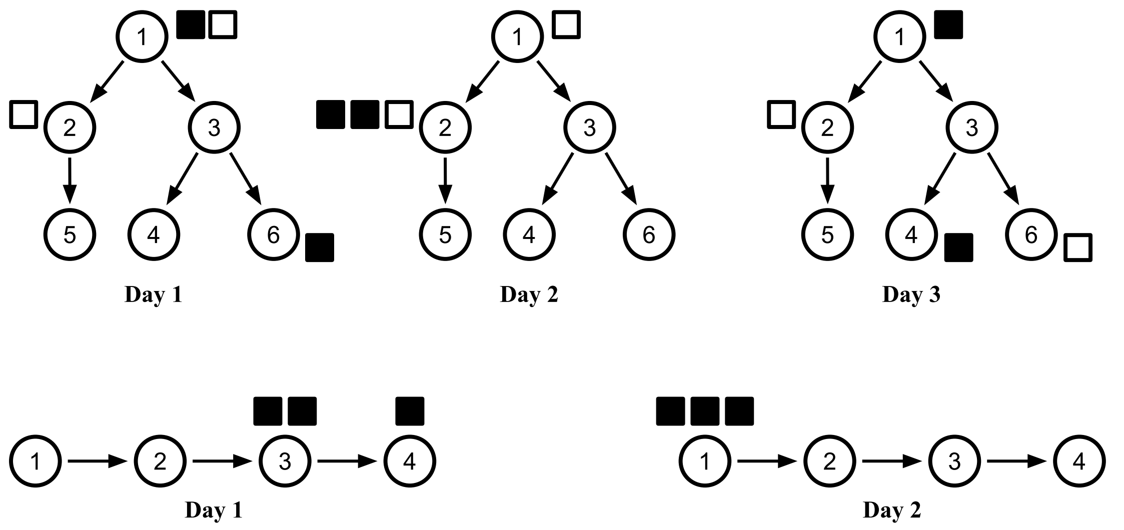


Figure 1: In each case, black squares represent telefrogs and white squares represent impostors.

River II

Input File	Output File	Time Limit	Memory Limit
standard input	standard output	1.5 seconds	256 MiB

The N residents of Ragden reside underground in dingy rectangular hollows far below the extravagant royal palaces of the Great Tree.

Tired of the constant flooding from events such as the *Great Storm* and *Lauren Forgot To Turn Off The Sprinklers*, the residents have asked you to build an artificial underground river through which the storm waters can flow.

The Underground can be described as a rectangle W metres wide, and H metres deep. The point x metres from the left edge of The Underground and y metres below the surface is denoted (x, y).

The i-th hollow is defined by the rectangle with top-left corner (a_i, b_i) and bottom-right corner (c_i, d_i). **No two hollows intersect**. Hollows may touch on their sides or at corners.

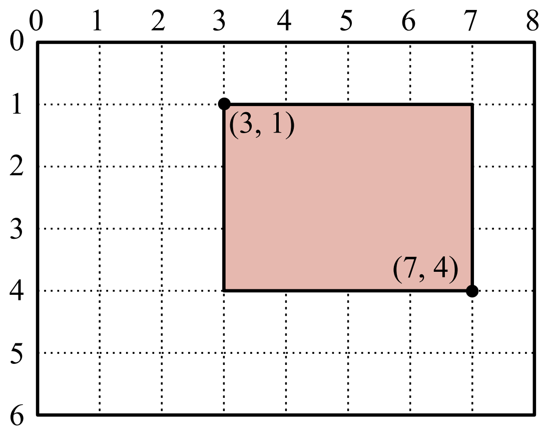


Figure 1: An example with $W = 8$, $H = 6$

The river can be thought of as a sequence of points (x_0, y_0), (x_1, y_1), (x_2, y_2), …, (x_k, y_k), that form a poly-line.

- The river must start on the surface. That is, $y_0 = 0$.
- The river must end on the bottom of The Underground. That is, $y_k = H$.
- The river must never flow upwards. That is, $y_i \le y_{i+1}$ for all i.
- The river must not intersect the interior of any hollows. The river may touch the sides or corners of the hollows.

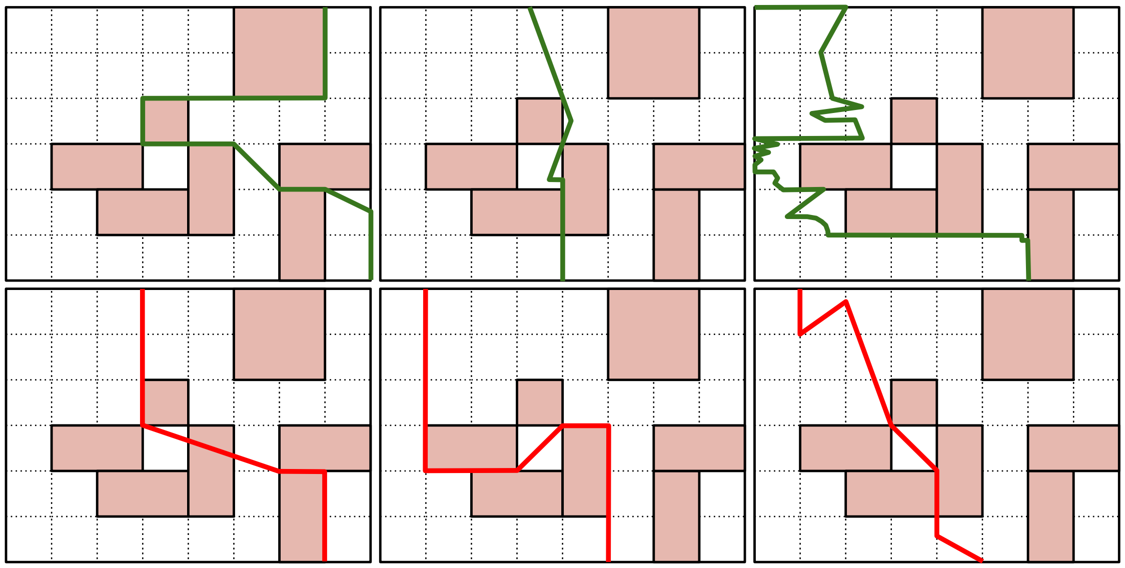


Figure 2: The top three rivers are valid. The bottom three rivers are invalid.

Notice that the river splits The Underground into a *left-side* and a *right-side*. If the i-th hollow is on the *left-side*, then the residents will generate l_i happiness points. Similarly, if the i-th hollow is on the *right-side*, then the residents will generate r_i happiness points. Note that l_i and r_i may be negative.

What is the greatest total happiness you can achieve?

Subtasks and Constraints

For all subtasks, you are guaranteed that:

- $1 \le N \le 100\,000$.
- $1 \le W, H \le 1\,000\,000$.
- $0 \le a_i < c_i \le W$, for all i.
- $0 \le b_i < d_i \le H$, for all i.
- $-10\,000 \le l_i, r_i \le 10\,000$, for all i.
- No two hollows intersect.

Additional constraints for each subtask are given below.

Subtask	Points	Additional constraints
1	6	All hollows have height 1. That is, $b_i + 1 = d_i$ for all i.
2	23	$W, H, N \le 100$.
3	14	$W, H \le 1000$.
4	25	$N \le 5000$.
5	28	All hollows have width 1. That is, $a_i + 1 = c_i$ for all i.
6	4	No additional constraints.

Input

- The first line of input contains the three integers N, W and H.
- The next N lines describe the hollows. The i-th line contains a_i, b_i, c_i, d_i, l_i and r_i.

Output

Output a single integer, the greatest total happiness you could achieve.

Sample Input

```
7 8 6
5 0 7 2 -30 -9
3 2 4 3 1 9
1 3 3 4 -3 -8
2 4 4 5 -10 10
4 3 5 5 0 2
6 3 8 4 40 6
6 4 7 6 1 3
```

Sample Output

```
30
```

Explanation

By drawing the river as shown, we can achieve a total happiness of $-9+9+-3+-10+0+40+3 = 30$.

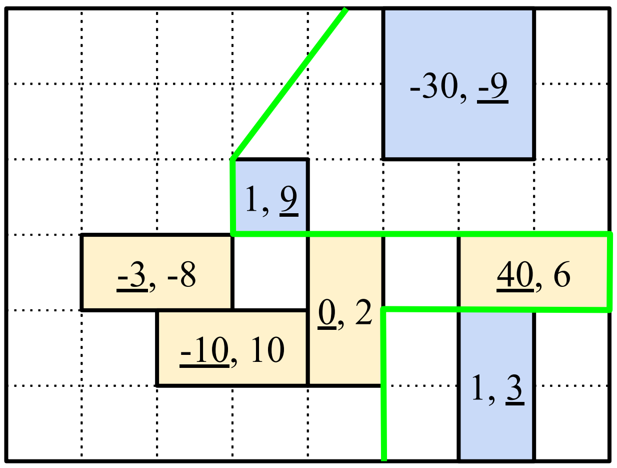


Figure 3: The sample case. Hollows on the left of the river are marked yellow, hollows on the right are blue.

Lucky Symbols

Input File	Output File	Time Limit	Memory Limit
N/A	N/A	2 seconds	512 MiB

Lucky Symbols is a game played on an R by C grid of squares. The rows are numbered from 0 to $R - 1$ from top to bottom, and the columns are numbered from 0 to $C - 1$ from left to right.

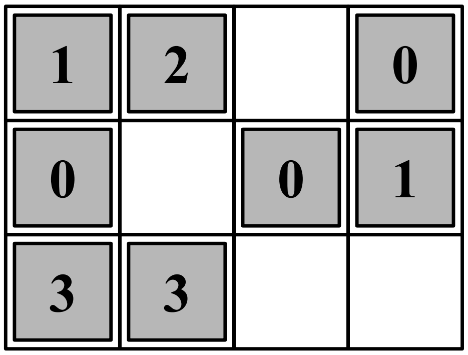
The game is played over 100 *rounds*. The goal of each round is to place symbols into the grid to maximise your score at the end of the round.

There are N types of symbols, numbered from 0 to $N - 1$. The i-th symbol has a value of a_i.

Each round is made up of 1000 *turns*. At the start of each turn, you are given one of the N types of symbols uniformly at random (each symbol is equally likely). You must then choose either to:

- *Place* the symbol in one of the empty squares of the grid.
- *Discard* the symbol without placing it in the grid. You must discard if there are no empty squares in the grid.

At the end of each turn, you receive points equal to the total sum of values of all symbols in the grid. For example, if you ended a turn with the grid below, you would receive $50 + 10 + 20 + 20 + 20 + 50 + 100 + 100 = 370$ for that turn.



Tile	Value
0	20
1	50
2	10
3	100

Bonus Pair	Points
0, 1	90
0, 2	70
0, 3	30
3, 3	100

Figure 1: An example grid with $R = 3$, $C = 4$ and $N = 4$

After 1000 turns the round ends and you get to earn some bonus points! There are E bonus symbol pairs. The i-th bonus pair gives you b_i points for each pair of adjacent (touching sides) squares, where one square contains a symbol of type x_i and the other square contains a symbol of type y_i.

For example, if you ended a round with the grid above, you would receive 400 bonus points for that round:

- Three {0, 1} bonus pairs: $3 \times 90 = 270$
- No {0, 2} bonus pairs.
- One {0, 3} bonus pair: $1 \times 30 = 30$.
- One {3, 3} bonus pair: $1 \times 100 = 100$.

After your score for the round is calculated, the board is cleared of all symbols and the next round starts. Your task is to write a program to maximise your average score over all 100 rounds.

Subtasks and Constraints

For all subtasks, you are guaranteed that:

- $0 \leq a_i$, for all i.
- $1 \leq b_i$, for all i.
- $0 \leq x_i \leq y_i < N$, for all i. Note that it is possible for $x_i = y_i$.
- No pair of symbols appears more than once as a bonus pair. That is, either $x_i \neq x_j$ or $y_i \neq y_j$ for all i and j where $i \neq j$.

Subtask	Points	R	C	N	E	Max a_i	Max b_i	Additional Constraints
1	8	1	1	1000	0	1000	N/A	
2	8	3	3	1000	0	1000	N/A	
3	18	50	50	10	10	0	1	$x_i = y_i$ for all i
4	18	15	15	2000	2000	0	1	$x_i = y_i$ for all i
5	16	20	20	10	50	10	2000	
6	16	20	20	100	500	10	2000	
7	16	20	20	1000	5000	10	2000	

In this problem, each subtask **only has one test case**. These test cases are available for download from the Attachments page. Note that in each case, the values of a_i and b_i are generated uniformly at random. Similarly, the E bonus pairs are generated such that each pair has an equal probability of being chosen.

Recall that your score for this problem is the sum of your scores for each subtask. Your score for a subtask is the maximum score obtained among any of your submissions. As such, you may wish to solve different subtasks in different submissions.

Scoring

If your program does not successfully play the game as described in the *Implementation* section, then you will receive 0% of the points for that subtask.

Otherwise, your score will be on a linear sliding scale based on two threshold scores O_{min} and O_{max} ($O_{min} < O_{max}$). Your score scales linearly from 0% to 100% between O_{min} and O_{max}. Specifically, let S be the average score achieved by your program. Then:

- If $S > O_{max}$, you will score 100% of the points.
- If $O_{min} \leq S \leq O_{max}$, you will score $\lfloor (S - O_{min})/(O_{max} - O_{min}) \times 100 \rfloor$% of the points.
- If $S < O_{min}$, you will score no points.

The parameters O_{min} and O_{max} for each subtask are in the table below. O_{max} is the judges' best score for that subtask.

Subtask	O_{min}	O_{max}
1	450000	953000
2	4500000	8120000
3	3000	3590
4	0	104
5	2500000	3620000
6	1500000	3260000
7	1500000	2580000

Implementation

Input / Output

In this task, you must not read from or write to any input/output files. Instead, your solution must interact with the functions in the header file `lucky.h`.

Do *not* output *anything* to `stdout`, or you will receive 0% points for the test case.

Functions

You **must not** implement a `main` function. Instead you should `#include "lucky.h"` and implement the functions `init`, `newRound` and `playTurn` described below:

```
void init(int R, int C, int N, int E,
    std::vector<int> a, std::vector<int> b, std::vector<int> x, std::vector<int> y)
```

where:

- `R`, `C`, `N` and `E` are the corresponding variables described above.
- For every $0 \leq i < N$, `a[i] =` a_i.
- For every $0 \leq j < E$, `b[j] =` b_j, `x[j] =` x_j, and `y[j] =` y_j.
- This function is called first to initialise your program and start the game.

```
void newRound()
```

This function is called at the start of every round. It will be called 100 times, once for every round in the game.

```
void playTurn(int symbol)
```

After each call of `newRound`, this function is called 1000 times, once for each turn in the round. `symbol` is the type of symbol you are given on that turn.

Your implementation of `playTurn` must call either `place` or `discard` as described below:

- `void place(int r, int c);` – call this function to place the symbol in the square located in the `r`-th row and `c`-th column.
- `void discard();` – call this function to discard the symbol.

The judge's grader will choose a symbol to give you uniformly at random on each turn. The grader will not adapt the symbols to your choices to place or discard symbols. The sequence of symbols chosen in each test case is fixed (that is, the symbols given to `playTurn` will be the same for every submission on that test case).

Failure conditions

Your program:

- Must not call `place` or `discard` from inside `init` or `newRound`.
- Must call either `place` or `discard` exactly once during each call of `playTurn`.
- When calling `place(r, c)`, the parameters must satisfy $0 \leq$ `r` $< R$ and $0 \leq$ `c` $< C$
- When calling `place(r, c)`, the square in the `r`-th row and `c`-th column must not have a symbol in it already.

If your program violates any of these conditions, it will be judged as `incorrect` and you will score 0% of the marks for that test case.

Experimentation

In order to experiment with your code on your own machine, first download the provided files `lucky.cpp`, `lucky.h` and `grader.cpp`, which should be placed in the same directory as your code. Please note that the grader that is used may have different behaviour to the provided grader. You should modify `lucky.cpp`, which contains stub implementations of `init`, `newRound` and `playTurn`.

Compile your solution with:

```
g++ -std=c++11 -O2 -Wall lucky.cpp grader.cpp -o lucky
```

This will create an executable `lucky`, which you can run with `./lucky`. If you have trouble compiling, please send a message in the Communication section of the contest website.

The compiled sample grader will read input from standard input in the following format:

- The first line of input contains the four integers R, C, N and E.
- The second line of input contains N integers. The i-th integer is a_i.
- The next E lines of input describe the bonus pairs. The i-th such line contains b_i, x_i and y_i.

For each test case, the sample grader will play 100 rounds. Each round begins with a call to `newRound`, followed by 1000 calls to `playTurn`.

At the end of the game, the sample grader will print your average score to standard output.

Note that the sample grader *may not* be as strict as the grader used for judging. In particular, the sample grader may not check all the failure conditions specified above.

Sample Grader Input and Sample Session

```
3 4 4 4
20 50 10 100
90 0 1
70 0 2
30 0 3
100 3 3
```

One possible sample interaction is shown below:

Grader	Student	Description
`init(3, 4, 4, 4, [20, 50, 10, 100], [90, 70, 30, 100], [0, 0, 0, 3], [1, 2, 3, 3])`		The grader initialises your program.
`newRound()`		The grader starts a new round.
`playTurn(2)`		You get a symbol of type 2.
	`place(0, 1)`	You place it in row 0, column 1.
`playTurn(3)`		You get a symbol of type 3.
	`discard()`	You discard it.
`playTurn(0)`		You get a symbol of type 0.
	`place(0, 2)`	You place it in row 0, column 2.
`playTurn(2)`		You get a symbol of type 2.
	`place(2, 3)`	You place it in row 2, column 3.
Round ends		Your score for the round is 200 points.
`newRound()`		The grader starts a new round.
`playTurn(2)`		You get a symbol of type 2.
	`discard()`	You discard it.
`playTurn(0)`		You get a symbol of type 0.
	`discard()`	You discard it.
`playTurn(2)`		You get a symbol of type 2.
	`discard()`	You discard it.
`playTurn(1)`		You get a symbol of type 1.
	`discard()`	You discard it.
Round ends		Your score for the round is 0 points.
grader terminates		The grader records your average score.

Ordinarily, the grader would play 100 rounds of 1000 turns each. However, for the sake of brevity the same session only has 2 rounds, with 4 turns each.

Your score in the first round is 200 points:

- At the end of the 1st turn, you receive 10 points.
- At the end of the 2nd turn, you receive 10 points.
- At the end of the 3rd turn, you receive 30 points.
- At the end of the 4th turn, you receive 80 points.
- At the end of the round, you score 70 bonus points.

Your score in the second round is 0 points (as you discarded everything).

Your average score at the end of the game is 100 points.

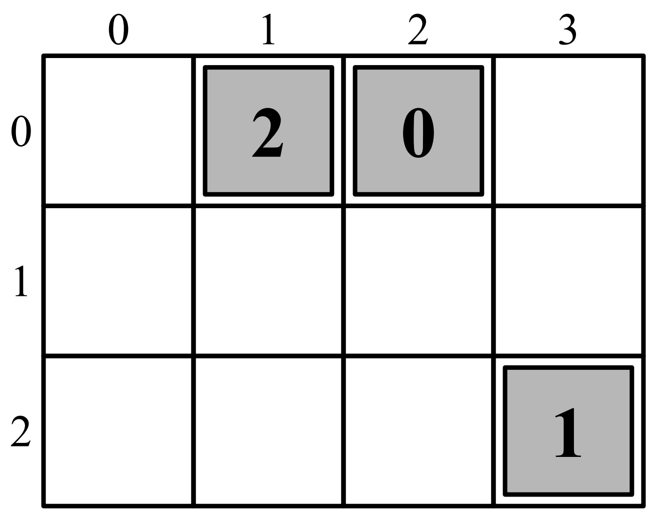


Figure 2: The state of the grid at the end of the first round of the Sample Session