Problem A: Longest Downhill Run

Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

Just as you arrived at the top of Mount Neverrest, a ski resort consisting of a vast network of slopes that are bridged by natural plateaus, the ski lift broke down. Upon examining the automatic defect report, the guy overseeing the lift quickly concluded that he wouldn't be able to fix it himself and announced that since the technician wouldn't be there before tomorrow, there would be no more rides up the mountain today.

As you are only here for this once day, this means you will only be able to ski down the mountain once. Because you were planning to ski down and ride back up all day, you at least want to make your downhill run as long as possible. To do that, you intend to take the path that consists of the largest number of individual slopes. But first, you've got to determine exactly which path this is.

Input

The input consists of

- one line containing N and M ($N \leq 10^4, M \leq 10^5$) – the numbers of plateaus (which are numerated $1, ..., N$) and slopes between them
- M lines describing the slopes, with the i-th line containing integers b_i and e_i ($1 \leq b_i, e_i \leq N$) denoting that there is a slope from plateau b_i down to plateau e_i. Of course, since all slopes lead downhill, there are no paths that use any plateau twice.

Output

Output a single integer - the maximum number of slopes of any path you could take down Mount Neverrest (as you expect this path to start very high up and end close to the mountain's foot anyway, you don't care where exactly it starts or ends).

Sample input and output

Input	**Output**

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

Problem B: Boolean Tree

Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

Let's consider a special kind of binary tree, called Boolean tree. In such a tree every level is complete, except maybe the last (the deepest) one. All vertices of the last level are aligned to the left, so there are no gaps between them. Every vertex has zero or two children (i.e. there is an even number of leaves in the last level). All leaves are assigned binary values (0 or 1).

Moreover, each internal vertex (non-leaf) is assigned a Boolean operation ("AND" or "OR"). The value of a vertex with operation "OR" is the disjunction of its children's values. Accordingly the value of a vertex with operation "AND" is the conjunction of its children's values.

We are mostly interested in the root of the tree – we want it to have the specified value $v \in \{0, 1\}$. Fortunately, we can change the Boolean operations of some vertices (change "AND" to "OR" or vice versa). Given a full description of the tree, determine the minimum number of vertex operation changes necessary to change the root's value to v or that it is impossible.

Input

The tree's vertices are numerated in such a way that the children of an internal vertex i are vertices $2i$ and $2i + 1$. The input consists of

- one line containing n and v ($1 \leq n < 10^4, v \in \{0, 1\}$) – the number of vertices in the tree and the desired value of the root. Because every vertex has an even number of children, n is odd.

- $(n - 1)/2$ lines describing the internal vertices, with the i-th line containing two integers g_i and c_i ($g_i, c_i \in \{0, 1\}$) – if $g_i = 1$ then the operation of vertex i is "AND", otherwise it is "OR", if $c_i = 1$ the operation can be changed, otherwise not

- $(n + 1)/2$ lines describing the leaves, with the i-th line containing a single integer $\ell_i \in \{0, 1\}$, denoting that vertex $(n - 1)/2 + i$ is assigned the value ℓ_i.

Output

Output a single line - the minimum number of vertex operation changes to make the root's value into v. If this isn't possible, print "IMPOSSIBLE".

Sample input and output

Input	**Output**

9 1
1 0
1 1
1 1
0 0
1
0
1
0
1
	1

5 0
1 1
0 0
1
1
0
	IMPOSSIBLE

Problem C: Slider's Paradise

Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

Aquaplex, a huge, brand new waterpark just opened its gates to the public. As it is still lacking in publicity, the owners decided to advertise based on the park's greatest attraction, *Slider's paradise*, a vast network of waterslides that covers an entire mountainside. At the end of each slide there is a platform, usually connected to the beginning of at least one other slide leading further downhill (some are not, those so-called side-exits provide a possibility to exit the attraction early). Guests start at the top platform and work their way down to the bottom platform (or one of the side exits) through one of many sequences of slides. Since they suspect the number of possibilities (in terms of which slides are used) to get from top platform to bottom platform to be extremely large, the owners want to use it in their advertisement. Can you help them calculate it?

Input

The input consists of

- one line containing N, M, s and t ($2 \leq N \leq 5 \cdot 10^4, 0 \leq M \leq 2 \cdot 10^5$, $1 \leq s, t \leq N$) – the amounts of platforms and slides, as well as the indices of top platform and bottom platform
- M lines describing the slides, with the i-th line containing integers b_i and e_i ($1 \leq b_i, e_i \leq N, b_i \neq e_i$), denoting a slide beginning in platform b_i and ending in platform e_i. It is guaranteed that no slides end in the top platform s or start in the bottom platform t and that one can't get to a platform one has already slid down from without leaving the attraction in the meantime.

Output

Output the number of possibilities specified above. It is guaranteed that the result doesn't exceed 10^{16}.

Sample input and output

Input	**Output**
4 5 1 4	3
1 2	
2 3	
1 4	
2 4	
3 4	
7 8 1 7	4
1 2	
1 3	
2 4	
3 4	
4 5	
4 6	
5 7	
6 7	
5 7 1 5	5
1 2	
1 3	
1 4	
2 3	
3 4	
3 5	
4 5	

Problem D: Dungeon Explorer 1: Maximal Net Gain

Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

In recent months, you've become an avid player of *Dungeon Explorer*, a very popular new MMORPG. At the moment, your guild is running out of gold, so you've collectively decided to raid a dungeon (that is, an extensive network of huge halls filled with monsters and treasures, connected by doors and corridors), which is known to yield a lot of gold. Through a community effort, the dungeon in question has already been thoroughly analyzed so that for each hall you know exactly what you can expect to gain from clearing it and how much this would likely cost you in terms of healing potions, attack or defense boosts etc. that you would need to use (there's some randomization involved).

Raiding a dungeon generally works as follows: All of you start together in a hall of your choice. First you have to clear that hall, then you can proceed to clearing a neighboring one, and so on. Importantly, you have to clear each hall you want to cross, you can't just cross one and skip clearing it. On the other hand, once you've cleared a hall, it will remain cleared for the rest of your stay, so that you can go back through those you already cleared and move on to clear ones neighboring them that you originally ignored. Exactly 30 minutes after entering the dungeon, its boss will teleport all of you into his throne room. Once you defeat him (or he defeats you) you will be teleported out of the dungeon entirely.

As your guild is already very numerous and strong, you are confident you'd be able to clear all halls within those 30 minutes – however, as many of them are so full of enemies that you'd need to spend more gold to clear them than you would get in return from its treasures, this might not make sense. Before you start, your guild leader has asked you to calculate the ideal proceeding.

Input

A dungeon is laid out in such a way that between any two halls there's precisely one path. As a consequence, there are only $N-1$ direct bidirectional connections between halls in a dungeon with N halls. The input consists of

- one line containing N ($1 \leq N \leq 10^5$) – the number of halls in the dungeon
- one line containing N numbers $g_1, ..., g_N$ ($-1000 \leq g_i \leq 1000$), where g_i is the expected net gain (loot minus expenses) from clearing the i-th hall.
- $N-1$ lines describing the connections between halls, with the i-th line containing integers a_i and b_i ($1 \leq a_i, b_i \leq N$), denoting a door or corridor between the halls with indices a_i and b_i.

Output

Output the maximal expected net gain from one visit to the dungeon (disregarding the loot to be gained by defeating the boss). If no strategy is expected to yield any net gain, simply output 0 (your guild won't visit the dungeon then).

Sample input and output

Input	**Output**

7
1 3 3 4 -30 10 10
1 2
1 3
1 4
4 5
5 6
5 7
	11

7
1 3 3 4 -30 10 10
1 2
1 3
1 4
4 5
5 6
6 7
	20

Problem E: Juice

Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

In the favela of Rio de Janeiro there was a flicker of light. After months of careful building, the new generator had finally been connected to the thousands of extension cords and the slum was suddenly illuminated by millions of bright lights...

However, the capacity of the extension cords was not enough to meet the energy demands of all the houses in the slum. Thus, the engineers had to carefully select which houses should be powered, and which should not, prior to connecting the power generator. Their idea was to power as many houses as possible, based on the energy demands of each house, and the capacity of the extension cords.

More specifically, the generator and each of the houses are represented by nodes and the extension cords by edges between these. Therefore, each node gets power from exactly one other node. In addition, each node except the generator node has a non-negative power demand. The generator produces an amount of energy that far surpasses the total capacity of the extension cord connected to it, and can thus be treated as an infinite energy source.

Given the same data, find out how many houses the engineers were able to cover the energy demands of.

Input

The input consists of

- one line containing n $(0 \leq n \leq 1000)$ – the number of houses in the favela
- n lines describing the house network, with the i-th line containing p_i, r_i and c_i $(0 \leq p_i \leq n,\ 0 \leq r_i \leq 100,\ 1 \leq c_i \leq 100)$ – the parent node and energy demand of house i and the capacity of the extension cord connecting it to house p_i. The power generator has index 0.

Output

Output the maximum number of power requirements that can be met.

Sample input and output

Input	**Output**
3	2
0 3 2	
0 100 100	
1 1 1	

Problem F: Funky Beats

Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

After years of passionate DJ'ing without ever gaining traction, a long standing dream of yours is finally about to come true: You managed to get booked to DJ in the world-famous club *Funky Beats* on Monday night.

Understandably, you don't want to disappoint the club's very demanding crowd, thus you plan to consult your entire database of all the tracks with funky beats and use it to create a funky playlist that is as long as possible, and, among all funky playlists of this length, also maximizes the number of tracks it uses. However, there are some difficulties in creating such a playlist: By the rules of the club, the tracks you play need to get progressively funkier – but that's not all: In order to prevent the crowd from getting bored, the funkyness increase from one track to the next needs to be higher the longer the track was. On the other hand, you shouldn't increase the funkyness too drastically, so as to avoid giving the guests a heart attack.

More specifically, each track i has a length ℓ_i as well as a funkyness level f_i and you can follow track i with track j if f_j is at least f_i plus ℓ_i in minutes (rounded up to full ones!) and strictly less than f_i plus ℓ_i in minutes plus 10 (with ℓ_i still being rounded up to full minutes).

Furthermore, of course you can't just let a track end entirely before you start playing the next one – instead you need to use the last 10 seconds of each non-final track to fade in the next one (so a playlist consisting of two tracks of length 1:30 each is considered to have length 2:50, while a playlist consisting of a single track that of length 3:00 is considered to have length 3:00).

Due to this abundance of restrictions, you are having trouble figuring out how to create the ideal playlist in your head, so you decide to write a program that helps you.

Input

The input consists of

- one line containing N ($1 \leq N \leq 10^5$) – the number of tracks in your database
- N lines containing descriptions of the tracks, with the i-th containing ℓ_i and f_i ($1{:}00 \leq \ell_i \leq 15{:}00$, $1 \leq f_i \leq 10^4$), the length and funkyness of the i-th track. It is guaranteed that each funkyness level is shared by at most 100 tracks.

Output

Output the length of the longest admissible playlist (in the format h:mm:ss) as well as the maximum number of tracks that can be used in creating it.

Sample input and output

Input	**Output**
3	0:03:40 2
2:20 1	
1:30 4	
3:00 3	
3	0:04:00 1
2:20 1	
1:30 4	
4:00 3	
5	0:10:00 3
1:39 30	
2:30 1	
2:50 6	
10:00 2	
5:00 9	