Problem A: Stars

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

In order to call a polygon a star, we require two things:

- the corners are alternating between pointing into and out of the polygon in a zig-zag pattern.
- the angles of the outer corners should be acute ($0° < \text{angle} < 90°$).

Given a polygon as a list of points describing the corners in clockwise order (neighboring and the last and first point are connected), you are to find out if it is a star.

Input

The input consists of

- one line containing N ($3 \leq N \leq 1000$) – the number of corners
- N lines describing the corners, the i-th of which contains two integers x_i and y_i ($-1000 \leq x_i, y_i \leq 1000$) – the coordinates of the i-th corner.

Output

Output `YES` if the given shape is a star and `NO` otherwise.

Sample input and output

Input	Output
6 1 -1 1 -4 -1 1 -2 3 1 2 3 3	YES
4 1 -1 -1 -1 -1 1 1 1	NO

Problem B: Border

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

To this day, there were only some independent city-states scattered across the vast plane of *Descartes*. But today, in an unprecedented gathering of all the states' rulers, they finally agreed that this had to change: As those traveling between the cities were increasingly often becoming subject to violent robberies, the rulers acknowledged that a new form of organization would be necessary to stop this.

So they decided to form a federation that centralizes – among other things – its defense efforts. Since their main objective was to patrol the area between them, they agreed that the federation should exercise control not only over all direct paths between pairs of city-states in it, but also over all the areas these paths enclose. Now they are discussing whether to create a physical wall along the external border, which would cost a lot but help them to clear their new-claimed territory from bandits. Can you help them to figure out if they can afford it?

Input

The input consists of

- one line containing N ($3 \leq N \leq 10^5$) – the number of city-states – and C ($1 \leq C \leq 10^4$) – the amount of gold necessary to build a border wall of length 1 (you can assume the cost of a wall to be proportional to its length)
- N lines describing the city-states, with each of them containing integers x, y and g ($-10^9 \leq x, y \leq 10^9$, $0 \leq g \leq 10^9$), with the first two being the coordinates of the respective city-state and the third one being the amount of gold it would be willing to spend on a border wall.

Output

If the amount all states combined are willing to spend on it suffices to build a border wall along the entire external border, output `''Build that wall!''`, otherwise output `''Maybe don't!''`.

Sample input and output

Input	Output
5 100 0 0 200 2 0 200 0 2 200 1 1 0 2 2 200	Build that wall!
3 1 0 0 0 1 0 0 1 1 0	Maybe don't!

Problem C: Colorful Polygon

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

You recently bought a 3D printer of the fused filament fabrication type and, naturally, started printing lots of useless stuff. One of the things you printed was a very large simple polygon consisting of only one layer of filament. At first you really liked it and even hung it up on your room's wall, but by now you have gotten kind of bored with it, especially because it is entirely white (your colored filaments hadn't been delivered yet) and you like things to be colorful. Also, you are not really satisfied with its ragged shape anymore and have decided that you prefer convex polygons.

Thus you decided to modify it by attaching a bunch of smaller polygons to it in such a way that no two of them have the same color, no two share a side with one another and the new polygon's shape exactly matches the convex hull of the current polygon (note that for technical reasons the polygons you attach cannot have width 0 anywhere (except at their corners of course), so you can't just print one large polygon that slings around the entire old one and completes it to its convex hull). However, you're not sure whether you still have enough of the colored filaments to do this. So let's find out!

Input

The input consists of

- one line containing n ($3 \leq n \leq 10^5$) – the number of vertices in the polygon
- n lines giving the vertices in either clockwise or counter-clockwise order, with the i-th line containing integers x_i and y_i ($-10^4 \leq x_i, y_i \leq 10^4$) – signaling that the i-th vertex has coordinates (x_i, y_i)
- one line containing m ($0 \leq m \leq 30$) – the number of colors of which you still have some filament

- one line containing m numbers $a_1, ..., a_m$ ($1 \leq a_i \leq 10^9$) – with a_i being the remaining amount of the i-th color in "area units that can still be printed with it".

Output

Output `YES` if you can print additional polygons as planned and `NO` otherwise.

Sample input and output

Input	**Output**
5 0 0 4 0 4 2 2 1 0 2 3 1 1 1	NO
5 0 0 4 0 4 2 2 1 0 2 1 3	YES
7 0 0 0 2 1 1 2 2 3 1 4 2 4 0 3 1 1 1	YES

Problem D: Defense Walls

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

At long last, the Titans are approaching even your secluded country. There is only one way to keep everyone safe: Defense walls! Of course you want to build as many walls as possible (and the supply of raw material won't be an issue), but there are some restrictions to how they can be built: Every defense wall should have polygonal shape and a guard tower at each of its corners – however, guard towers can't be built everywhere, only in a few select spots (these have already been determined). Furthermore, different defense walls shouldn't touch one another (in particular they may not have common guard towers), and, most importantly, there should be an area that is enclosed by all walls (the most vulnerable people will be moved there).

Given the coordinates where guard towers could be built, find how many walls can be constructed such that the above restrictions are satisfied.

Input

The input consists of

- one line containing n ($3 \leq n \leq 10^4$) – the number of spots where guard towers can be built
- n lines describing these spots, with the i-th line containing integers x_i and y_i ($-1000 \leq x_i, y_i \leq 1000$) – the coordinates of the i-th spot. It is guaranteed that the given points are pairwise distinct.

Output

Output the maximum number of nested defense walls behind which the villagers can hide.

Sample input and output

Input	Output
8 2 0 1 1 0 2 1 3 2 4 3 3 4 2 3 1	1
8 3 0 2 2 0 3 2 4 3 6 4 4 6 3 4 2	2

Problem E: Cycling Race

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

In a couple of days, a large cycling race is set to start in your town. You have been tasked with developing a tool for monitoring it. The tool needs to handle two kinds of queries:

- Take note of a cyclist changing its pace.
- Determine the position and index of the race leader at a given time.

As the race track leads almost exclusively downhill, the cyclists never slow down, i.e. queries of the first time will only ever inform your program about a speed increase. Initially (at time 0), each cyclist has covered 0m of the race track and is standing still (moving at speed 0m/s). You may also assume that all cyclists strictly follow the race track, so that their velocities throughout the race determine exactly how much of the race track they have covered. Naturally, the queries have timestamps and are given in the increasing order induced by them.

Input

The input consists of

- one line containing N ($2 \leq N \leq 5 \cdot 10^4$) – the number of cyclists participating in the race
- between 1 and $3 \cdot 10^5$ lines giving the queries in the following formats:
  - "t 1 c s" for the first type, where t is the timestamp ($0 \leq t \leq 10^9$), c the index of the cyclist in question ($1 \leq c \leq N$) and s its new speed in m/s ($1 \leq s \leq 10^9$)
  - "t 2" for the second type, where t is the timestamp (this query asks for the position and index of the race leader at time t, i.e. the largest distance covered by any individual cyclist up to this time and which cyclist this was).

It is guaranteed that queries of the first kind only ever report speed *increases* and that at any given time (indicated by the query timestamps) there are either

- no queries
- a single type 2 query, or
- at most one type 1 query for each cyclist.

Thus it is possible that all cyclists increase their speed at time 0, but any given cyclist cannot change its speed twice at the same point in time.

Output

For each type 2 query, print first the distance of the race leader from the start and then its index. If there are multiple possibilities for the latter, print any (see second sample).

Sample input and output

Input	**Output**
3 0 1 2 10 1 1 1 10 5 2 16 1 3 100 18 2 20 1 1 311 21 2	50 2 200 3 501 1
3 0 1 2 10 1 1 1 10 5 2 16 1 3 100 18 2 20 1 1 310 21 2	50 2 200 3 500 3

Problem F: Framing a Polygon

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

You just conceived of a stunningly beautiful simple polygon. Of course you already made it your desktop background, but it is clear to you that such beauty mustn't be confined to the digital realm. Thus you decided to print it, frame it and hang it up on your wall.

The printing shouldn't be a problem (you'll just need to buy some new toner cartridges, as usual), but the framing poses a challenge: Obviously such a perfect polygon also deserves perfect framing, but you don't know what kind of frame you will need for that. The only things you're sure about with regards to the frame are that it should be rectangular, the polygon should fit inside it (it may touch the frame's edges, but none of its area may lie outside the frame's confinements), its height shouldn't be larger than its width and its area should be minimal among all frames with these three properties.

Input

The input consists of

- one line containing N ($3 \leq N \leq 10^5$) – the number of corners in the polygon
- N lines giving the corner coordinates in either clockwise or counter-clockwise order, with the i-th line containing two integers x_i and y_i ($-10^8 \leq x_i, y_i \leq 10^8$) – the coordinates of the i-th corner.

Output

Output the ideal height and width (in this order!) with an absolute or relative precision of 10^{-3}. Of course the polygon can be rotated arbitrarily to fit the frame as described.

Sample input and output

Input	Output
4 0 0 2 0 2 3 0 3	2.0000 3.0000
3 1 0 2 2 0 1	1.4142 2.1213
16 6 4 7 0 8 4 12 2 10 6 14 7 10 8 12 12 8 10 7 14 6 10 2 12 4 8 0 7 4 6 2 2	12.9987 12.9987

Visualizations of samples 2 & 3: