Problem A. Allergen Testing

Source file name:	Allergen.c, Allergen.cpp, Allergen.java, Allergen.py
Input:	`Standard`
Output:	`Standard`

You are in a chemistry class, and you are working with some compounds, exactly one of which you are allergic to. You have a fixed number of days to figure out which compound it is. You set up a number of sites on your arm for testing. On each day, you do the following exactly once:

1. Apply each compound to some (possibly empty) subset of sites on your arm. You can apply more than one compound to the same site.
2. Wait and see which sites demonstrate an allergic reaction.

A site demonstrates an allergic reaction if and only if the compound you are allergic to is applied to that site. If a site demonstrates an allergic reaction, it cannot be used on future days.

Compute the minimum number of sites you'll need on your arm to guarantee that you can determine exactly which compound you are allergic to within the given number of days.

Input

The first line of input contains a single integer t ($1 \leq t \leq 10^4$), which is the number of test cases that follow.

Each of the next t lines contains two integers n and d ($1 \leq n,\ d \leq 10^{18}$) describing a test case, where n is the number of compounds and d is the number of days.

Output

Output t lines. On each line output a single integer, which is the minimum number of sites on your arm necessary to discover which of the n compounds is the allergen within d days for that test case. Output the answers to the test cases in the order they appear in the input.

Example

Input	Output
1 4 1	2

Problem B. Better Dice

Source file name:	Better.c, Better.cpp, Better.java, Better.py
Input:	`Standard`
Output:	`Standard`

The latest Table-Top Role Playing Game is out now: *Better Dice*. Unlike all other TTRPGs, this one is all about dice. In fact, it is all about the *better die*: decisions are made, friendships gained and lost, fights fought, battles won, all based on who has the *better die*.

This game uses special n-sided dice where each of the n faces has the same probability of being rolled. Moreover, each die has its own special set of n numbers on the faces.

While playing *Better Dice* you ended up in a very precarious situation where you must absolutely have a *better die* than your opponent, that is, you must roll higher than your opponent. Given both your die and your opponent's die, decide who is more likely to roll a higher number.

A twenty-sided die with a special set of numbers on the faces. CC BY 4.0 by hamstermann on Thingiverse

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 1000$), the number of sides on each die.
- Two lines, each with n integers d ($1 \leq d \leq 10^9$), the values on one of the dice.

Output

Output "`first`" if the first die is more likely to roll a higher number than the second die.
Output "`second`" if the second die is more likely to roll a higher number than the first die.
Output "`tie`" if they are both equally likely to come up higher than the other.

Example

Input	Output
2 4 6 5 5	tie
6 1 2 3 4 5 6 7 6 5 4 3 2	second
3 2 2 2 1 1 8	first

Problem C. Contest Construction

Source file name:	Contest.c, Contest.cpp, Contest.java, Contest.py
Input:	`Standard`
Output:	`Standard`

The ICPC NAC staff have written a number of problems and wish to construct a problem set out of them. Each problem has a positive difficulty rating.

A contest has a *Nice* difficulty distribution if, when the difficulties of the problems are sorted in ascending order, every problem after the second has a difficulty rating less than or equal to the sum of the difficulty ratings of the two problems immediately preceding that problem.

Given the difficulty ratings of a set of problems and the number of problems desired for the set, count the number of problem sets with *Nice* difficulty distributions. Two problem sets are distinct if and only if there is some problem in one problem set but not in the other. (Specifically, note that two problem sets are the same even if the problems are permuted.)

Input

The first line of input contains two integers n ($3 \leq n \leq 50$) and k ($3 \leq k \leq 18$, $k \leq n$), where n is the number of problems the judges have written, and k is the number of problems they wish to put in the problem set.

Each of the next n lines contains a single integer d ($1 \leq d \leq 10^9$). These are the problem difficulties.

Output

Output a single integer, which is the number of possible problem sets with *Nice* difficulty distributions.

Example

Input	Output
5 4 2 1 4 3 5	4
8 5 1 2 3 5 8 13 21 34	4

Problem D. Determining Duos

Source file name:	Duos.c, Duos.cpp, Duos.java, Duos.py
Input:	`Standard`
Output:	`Standard`

As a coach of $2n$ students, you are making n duos (teams of two) for the upcoming programming contest season. After the duos have been created, they will participate in r contests, each about a different topic: DP, graphs, geometry, etc. You already ran a set of internal selection contests to rank the students, and from this you were able to rank all the students with a unique integer score between 1 and $2n$ inclusive for each topic, with $2n$ being the best.

When a duo participates in a contest on a given topic, their score will be the maximum of the two scores of the two students for this topic.

You think it would be amazing if summed up over all duos and contests, your students could achieve a total score of at least $\frac{1}{2}rn(3n+1)$. Is this possible?

Input

The input consists of:

- One line with two integers n and r ($1 \leq n \leq 4000$, $1 \leq r \leq 100$), the number of duos and the number of topics.
- r lines, the ith of which contains $2n$ integers $x_{i,1}, \ldots, x_{i,2n}$ ($1 \leq x_{i,j} \leq 2n$ for each i, j) where $x_{i,j}$ is the score of student j on topic i.

Output

If it is possible to make duos such that the total score over all duos and contests is at least $\frac{1}{2}rn(3n+1)$, output "`possible`". Otherwise, output "`impossible`".

Example

Input	Output
2 2 1 2 3 4 1 2 3 4	possible
2 2 1 2 3 4 4 1 2 3	possible
2 3 1 2 3 4 4 1 2 3 1 3 2 4	impossible

Problem E. Exceeding Limits

Source file name:	Exceeding.c, Exceeding.cpp, Exceeding.java, Exceeding.py
Input:	`Standard`
Output:	`Standard`

Tim needs to reach the Binary Analog Probing Conference (BAPC) on time, but he is running late. He is not sure if he can even make it on time without exceeding the speed limit! He does not like speeding, so he would like to minimize the amount that he needs to speed and plans his route accordingly. If he decides to speed by x km/h, he will exceed the speed limit everywhere by exactly x km/h.

Help Tim find the minimal amount that he needs to speed by to get to the BAPC in time.

As an example, consider the first sample case. Without speeding, Tim will take $\frac{400}{40} + \frac{300}{20} = 25$ hours to drive from intersection 1, via intersection 3, to intersection 4. In order to arrive in time, he will need to exceed the speed limit by 10 km/h, in which case his driving time will be $\frac{400}{40+10} + \frac{300}{20+10} = 18$ hours, following the same route.

Tim's arch-nemesis: the *trajectcontrole*.
CC BY-NC-SA 2.0 by DutchRoadMovies on Flickr

Input

The input consists of:

- One line with three integers n, m, and t ($2 \leq n \leq 10^4$, $1 \leq m \leq 10^5$, $1 \leq t \leq 10^5$), the number of intersections, the number of roads, and the time within which Tim needs to reach his destination.
- m lines, each with four integers a, b, ℓ, and v ($1 \leq a, b \leq n$, $a \neq b$, $1 \leq \ell, v \leq 10^5$). Each line indicates a bidirectional road between intersections a and b with length ℓ in km and speed limit v in km/h.

The intersections are numbered between 1 and n, inclusive.

Tim will start at intersection 1 and drive to intersection n, which is guaranteed to be reachable.

Output

Output how much Tim needs to exceed the speed limit, in km/h. If Tim can reach his destination without speeding, output 0.

Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
4 4 18 1 2 800 40 1 3 400 40 4 2 500 50 4 3 300 20	10
4 3 100 1 2 300 15 2 3 500 20 3 4 300 30	0
4 4 10 1 2 200 50 2 3 300 30 2 3 400 15 3 4 500 50	56.9041576

Problem F. Finding Forks

Source file name:	Finding.c, Finding.cpp, Finding.java, Finding.py
Input:	`Standard`
Output:	`Standard`

Your cutlery drawer contains many types of forks. Each with their own purpose, and each with its own place in the cutlery drawer. After a nice dining party with all your friends, disaster struck! You put all the used forks in the dishwasher, but now you are unsure where to put back some of the forks, because at least two places in the cutlery drawer are empty! And worse, you do not remember which type of fork belongs where!

What is the minimum number of forks that must have been in the dishwasher to cause this confusion?

A small selection of the many types of forks that you own. CC BY-SA 3.0 by Mark Taff on Wikimedia Commons

Input

The input consists of:

- One line with an integer n ($2 \leq n \leq 10^5$), the number of types of forks.
- One line with n integers a ($1 \leq a \leq 10^9$), the number of forks of each type.

Output

Output the minimum number of forks that must have been in the dishwasher.

Example

Input	Output
3 4 9 5	9
10 18 39 5 12 1000000000 54 23 11 123 31415	16

Problem G. Game Show Elimination

Source file name:	Game.c, Game.cpp, Game.java, Game.py
Input:	`Standard`
Output:	`Standard`

You are running an elimination-style game show, where players are eliminated one at a time until only one remains. Based on what you know about the contestants, you are trying to predict the results.

Each week, the remaining contestants take part in a competition, where each contestant's score is based on their skill level. Because this is a silly game show, a contestant's score is a random number that falls within their unique skill range. The contestants are then ranked from highest to lowest based on their scores. (Since scores are all real numbers, there is zero probability that there is a tie.)

The winner of this week's competition chooses who is eliminated that week. However, all players believe they should pick whoever did the best after them in the week's competition, so they always choose to eliminate the second place contestant. That contestant is eliminated and goes home. The show continues, week after week, until there is only one contestant left.

The contestants' final ranks are based on when they left the competition. The last contestant to leave (the winner!) is assigned rank 1, the second-to-last gets rank 2, and so on until the first person to leave is assigned the lowest rank.

Given information about the contestants' skill levels, compute the expected ranks of each player.

Input

The single line of input contains two integers n ($2 \leq n \leq 10^3$) and k ($2 \leq k \leq 10$), where n is the number of game show contestants, and k determines the skill range of the contestants.

Contestants are numbered from 1 to n. The skill range of contestant i is from i to $i + k$ inclusive, and their score each week is a randomly assigned real number in this range.

Output

Output n lines. Each line contains a single real number, which is the expected rank of a contestant. The expected ranks must be listed in contestant number order. The answers are accepted within absolute error of at most 10^{-6}.

Example

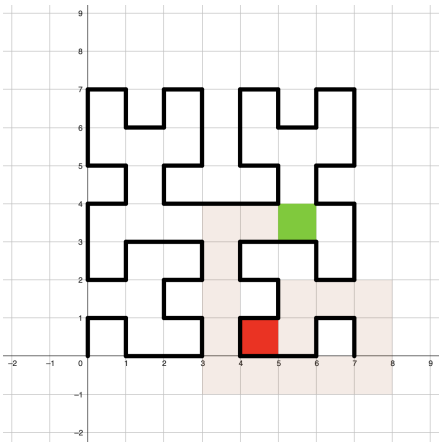
Input	Output
3 2	2.109375 2.625000 1.265625

Problem H. Hilbert's Hedge Maze

Source file name:	Hilbert.c, Hilbert.cpp, Hilbert.java, Hilbert.py
Input:	`Standard`
Output:	`Standard`

David has elaborate hedge mazes in his garden. For each of his mazes of different sizes, he wonders how long it would take to walk between any pair of locations in and around that maze. A hedge maze of order $n \geq 1$ is constructed according to very specific instructions:

1. Start with $S = A$.
2. Repeat n times:
   (a) Create a new string T by replacing each A in S with $LBFRAFARFBL$ and each B with $RAFLBFBLFAR$.
   (b) Set $S = T$.
3. Remove all A's and all B's from S.



The shortest path from the first case in the example input.

The resulting string gives the instructions for constructing the hedge maze. In the unbounded plane start at coordinates $(0, 0)$, facing towards $(1, 0)$ and for every F move forward one unit, for every R turn right 90 degrees, and for every L turn left 90 degrees.

After constructing the maze, identify position (x, y) with the unit square cell bounded by coordinates $(x, y), (x + 1, y), (x, y + 1), (x + 1, y + 1)$. We can move directly between two positions (x_1, y_1), (x_2, y_2) if and only if either $|x_1 - x_2| = 1$ or $|y_1 - y_2| = 1$, but not both, and there is no hedge between these cells. The distance between these two positions is 1.

Given a number n and a pair of positions, find the shortest distance between these positions.

Input

Input starts with a positive integer k ($1 \leq k \leq 100$), denoting the number of test cases. Each of the next k lines consists of 5 integers n, x_1, y_1, x_2, y_2, satisfying $1 \leq n \leq 50$, and $-2^{52} \leq x_1, y_1, x_2, y_2 \leq 2^{52}$. There is no interaction between the test cases. Each specifies a single hedge maze situated alone in the unbounded plane.

Output

Output k lines with one integer on each line, corresponding to the shortest distance between the two positions in each of the k test cases.

Example

Input	Output
2 3 5 3 4 0 2 4 3 0 2	16 11

Problem I. Idle Terminal

Source file name:	Idle.c, Idle.cpp, Idle.java, Idle.py
Input:	`Standard`
Output:	`Standard`

The IT Team praying that the migrations will complete successfully. From r/pcmasterrace

It is migration day at the Big Administration Processing Company: the database containing all administrative documents of all clients needs to be migrated to the latest version of the database software. Quite some things have changed over the past years, and this software has not been upgraded, so the number of migration jobs is high.

The migration jobs run in parallel on the multicore server machine in a first-come-first-serve fashion: every time a core is done running a job, it starts running the first job that has not yet started. The Idle Terminal (IT) Team is sitting huddled around the terminal window, eagerly awaiting the dopamine boost when another migration job completes successfully and a message is printed to the terminal.

The IT Team starts breaking out in sweat when nothing changes on the terminal for quite a long time. Did the connection hang? Has the server soft-locked? Did all jobs get stuck in infinite loops?? On the other hand, some migration jobs really do have a long duration, and it may simply be a coincidence that there are only long-running jobs active at the time. To calm down the IT Team, you decide to compute the longest time that goes by without seeing a new message on the terminal, starting from the moment that the first migration jobs start running.

Input

The input consists of:

- One line with two integers n and k ($1 \leq n, k \leq 10^5$), the number of migration jobs and the number of cores in the server machine.
- One line with n integers d ($1 \leq d \leq 100$), the duration of each job in the order that they are processed.

Output

Output the longest time that goes by without seeing a new message on the terminal.

Example

Input	Output
2 2 3 7	4
5 10 1 2 3 4 5	1
4 1 2 10 6 4	10
6 2 3 5 8 10 4 1	6
6 3 2 4 6 6 6 6	2

Problem J. Just a Joystick

Source file name:	Joystick.c, Joystick.cpp, Joystick.java, Joystick.py
Input:	`Standard`
Output:	`Standard`

You just got the high score when playing *Battlezone Asteroids Pac-Centipede* on an arcade machine! On the "Game Over" screen, you can enter your initials, one letter at a time. This seems to be a very modern arcade machine: whereas the original arcade machines only allowed entering three initials, this machine allows many more. However, to select the letters, you have access to just a joystick. For every letter, you need to move the joystick up or down to cycle between the letters (wrapping around between '`Z`' and '`A`', in both directions) and move it to the right to move to the next letter.

It appears that the initials of the previous high-score winner are still filled in. Entering your own initials is going to take some time, and you want to know exactly how long. How many times do you need to you move the joystick up or down to enter your own initials, if you do so in the most efficient way?

A girl playing *Battlezone Asteroids Pac-Centipede*.
CC BY-SA 2.0 by Sergiy Galyonkin on Flickr

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 10^5$), the number of letters available to enter your initials.
- One line with a string of length n, the initials of the previous high-score winner.
- One line with a string of length n, the initials that you want to enter.

The strings only consist of English uppercase letters (`A-Z`).

Output

Output the minimum number of times you should move the joystick up or down to enter your own initials. This does *not* include the number of times that you need to move the joystick to the right.

Example

Input	Output
3 RGK MPS	22
5 ABIYZ YZIAB	8

Problem K. King's Keep

Source file name:	King.c, King.cpp, King.java, King.py
Input:	`Standard`
Output:	`Standard`

King Carl's kingdom contains k keeps (commonly called castles).

Coordinates of keeps are known, and King Carl considers himself convinced that it could be convenient to choose a central keep as King Carl's residence.

Critically, King Carl considers that the average cost to carry commands from King Carl's residence to King Carl's other keeps should be small.

Compute the minimal average Euclidean distance[1] from his residence keep to the other keeps if King Carl chooses his residence optimally.

King Carl's keep of choice: created with contours of a capital 'K'. CC BY-SA 2.0 by David Dixon on geograph.co.uk, modified

Input

The input consists of:

- One line with an integer k ($2 \leq k \leq 1000$), the number of keeps.
- k lines, each with two integers x and y ($|x|, |y| \leq 1000$), the coordinates of the keeps.

It is guaranteed that all keeps are at distinct locations.

Output

Output the minimal possible average distance from the keep that is chosen as the king's residence to all other keeps.

Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
3 0 0 9 9 0 9	9
5 -3 5 6 8 1 2 5 -4 -7 -9	8.405705684

[1] The Euclidean distance between two points is the length of a straight line segment between these points.

Problem L. Losing Leaves

Source file name:	Leaves.c, Leaves.cpp, Leaves.java, Leaves.py
Input:	`Standard`
Output:	`Standard`

Here at the Benelux Advanced Phone Company (BAPC), we are the proud owners of the largest telephone network in the Benelux area. Unfortunately, our network has become too large for us to manage properly. As such, we have decided to sell part of our network.

The network of the BAPC consists of interconnected transmission nodes. One transmission node is marked as the central switch. All other nodes have exactly one upstream transmission node. For each transmission node, if we follow the upstream connections, we will finally end up at the central switch. A node is considered a customer transmission node when it is a leaf, i.e. when it has no downstream nodes.

When selling parts of our network, integrity must be maintained. That means that whenever we sell a transmission node X, we also have to sell nodes downstream of X.

Overall, BAPC decided to sell exactly k transmission nodes. While there may be many options to choose these k nodes, we actually want to make our lives as easy as possible: After selling, we want to minimize the number of customer transmission nodes in our network, while maintaining the etwork's integrity.

As an example, consider the second sample case, visualized in Figure 1. The three striped red nodes are sold, and the two bold green nodes are the remaining customer nodes.

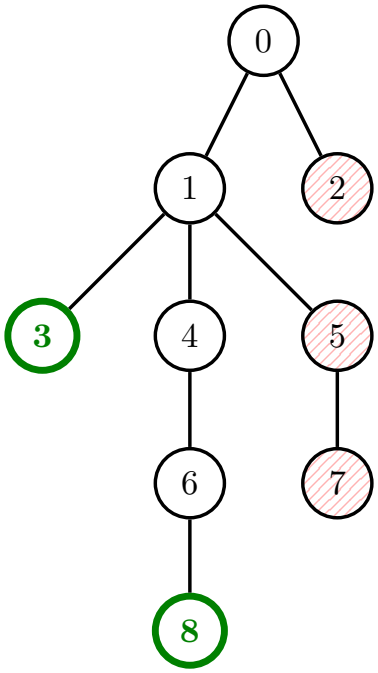


Figure 1: Visualization of the second example input.

Input

The input consists of:

- One line with two integers n and k ($0 \leq k < n \leq 10^6$), the number of transmission nodes, and the number of nodes to sell.
- $n-1$ lines, the ith of which contains one integer p_i ($0 \leq p_i < i$) indicating that transmission node i ($1 \leq i < n$) has an outgoing connection to node p_i.

The transmission nodes are numbered from 0 to $n-1$, inclusive. Node 0 is always the central switch.

Output

Output the minimum number of customer transmission nodes after selling k transmission nodes. Note that if the central switch is the only remaining node, it also counts as a customer node.

Example

Input	Output
5 2 0 0 1 1	1
9 3 0 0 1 1 1 4 5 6	2

Problem M. Monorail

Source file name:	Monorail.c, Monorail.cpp, Monorail.java, Monorail.py
Input:	`Standard`
Output:	`Standard`

A train about to enter the tunnel. From pxfuel.com

You were just about to go on a nice long holiday to the south, but as always, the trains are delayed. This time, a cargo train derailed in the Gotthard Base Tunnel through the Alps, completely taking one of the two tubes out of service for several months. Luckily, after some initial repairs, the other tube is in service again for cargo traffic.

Since it is now only a one-track connection, multiple trains in the same direction can closely follow each other, but trains in opposite directions can not pass each other. This also means that trains going north can only enter once all trains going south have exited the tunnel, and vice versa.

Today, there are n cargo trains that want to drive through the tunnel. Each train arrives at one of the ends at a given time, and takes exactly d minutes to drive through the tunnel at a constant speed.

Even though this is not your responsibility, you decide to make a schedule for today's trains. You will decide for each train how long is has to wait at its entrance portal before it can enter the tunnel. Your goal is to minimize the sum of waiting times at the entrance portals over all trains.

For simplicity, you assume that trains are short compared to the length of the tunnel and can be approximated by points travelling over a line segment.

Input

The input consists of:

- One line with an integer n and d ($1 \leq n \leq 500$, $1 \leq d \leq 10^9$), the number of cargo trains and the duration that each train needs to drive through the tunnel, in minutes.
- n lines, each containing a character s and an integer t ($s \in \{$'`N`','`S`'$\}$, $0 \leq t \leq 10^9$), indicating whether this train starts at the north or south portal, and the number of minutes after the start time at which this train arrives.

It is guaranteed that trains are sorted by arrival time. For trains with equal arrival time, the trains coming from the north are listed before the trains coming from the south.

Output

Output the minimal sum of waiting times at the tunnel entrance portals over all trains, in minutes.

Example

Input	Output
3 5 N 0 S 4 N 8	3
4 10 N 5 N 10 S 10 N 15	15
4 10 S 0 N 10 N 10 S 20	0
4 10 N 0 S 5 S 5 S 5	15