Problem A

Aspen Avenue

"Phew, that was the last one!" exclaimed the garden helper Tim as he threw the last tree plant to the ground. His employer, countess Esmeralda Hunt who owned the estate, has ordered him to arrange an avenue of aspen trees along both sides of the front road leading up to the house. The first trees in the avenue are supposed to be planted at the very beginning of the road, and the last trees of the avenue at the very end of the road. Tim, who was constantly being reminded of her sense of accuracy, knew that the countess would insist on the trees being placed in perfectly aligned tree pairs, one on each side of the road, and with exactly the same spacing between the pairs along the road. However, when bringing the tree plants to the estate, Tim had just dropped them arbitrarily along the left side of the road and was now facing the task of moving the trees to their correct positions to meet the countess's requirements. Being stronger in mind than in arms, and since it was time for a coffee break anyway before he started digging, he sat down to figure out which trees to move to which positions so as to minimize the total (Euclidean) distance that he had to move the trees.

Input specifications

The input file starts with a positive even integer N between 4 and 2000 (inclusive), giving the total number of trees in the avenue. The next line contains two integers L and W, where $1 \leq L \leq 10000$ is the length of the road, in meters, and $1 \leq W \leq 20$ is the width of the road, in meters. The next N lines each describe where Tim had dropped of the trees. Each such line contains an integer $0 \leq p \leq L$ indicating the position of a tree plant along the left side of the road, measured in meters from the start of the road.

Output specifications

For each test case, the smallest total number of meters the tree plants need to be moved. The answer should be given with an absolute or relative error of at most 10^{-6}.

Sample input 1	**Sample output 1**
4 10 1 1 0 10 10	2.4142135624

Sample input 2	**Sample output 2**
6 10 1 0 9 3 5 5 6	9.2853832858

Problem B

Best Compression Ever

Being educated in Computer Science and Mathematics is not always easy. Especially not if you have "friends" who repeatedly insist on showing you their new "proofs" that P equals NP, that the Riemann Hypothesis is true, and so on.

One of your friends recently claims to have found a fantastic new compression algorithm. As an example of its amazing performance, your friend has told you that every file in your precious collection of random bit strings after compression would be at most b bits long! Naturally, you find this a bit hard to believe, so you want to determine whether it is even theoretically possible for this to be true.

Your collection of random bit strings consists of N files, no two of which are identical, and each of which is exactly 1000 bits long.

Input specifications

The input consists of two integers N ($1 \leq N \leq 10^{15}$) and b ($0 \leq b \leq 50$), giving the number of files in your collection and the maximum number of bits a compressed file is allowed to have.

Output specifications

Output a line containing either "`yes`" if it is possible to compress all the N files in your collection into files of size at most b bits, or "`no`" otherwise.

Sample input 1	Sample output 1
`13 3`	`yes`

Sample input 2	Sample output 2
`1 0`	`yes`

Sample input 3	Sample output 3
`31415926535897 40`	`no`

Almost blank page

Problem C

Code Theft

While reviewing code recently being checked into the repository, Jim discovered that some employees now and then seemed to copy code right from the Internet into the company code base. This would be a potential disaster as the company then risks getting sued by copyright holders of the original code. The obvious solution, talking to the employees and kindly ask them not to submit any stolen code, seemed to solve the problem. Still, it was decided that a screening process should be introduced to detect newly stolen code.

The screening would work as follows: Every time new code was checked in, the full contents of the changed files where matched against a repository of known open source code. For each file the longest match, in number of consecutive lines, should be reported.

Comparison is done line by line. Empty lines, and lines only containing space, are ignored during comparison and not counted. Leading and trailing spaces should be ignored completely and consecutive space characters inside the lines are treated as one single space. The comparison is case-sensitive.

Input specifications

Test data starts with the number $0 \leq N \leq 100$ of code fragments in the repository. Then follows, for each code fragment, one line containing the file name that the fragment was fetched from and the contents of the fragment on subsequent lines. File names will neither contain whitespace nor be guaranteed to be unique. The name is at most 254 characters long. Each fragment is terminated by `***END***` on a line by its own. This line is not considered being part of the fragment.

After the fragments in the repository have all been listed, comes the actual code snippet to find matches for. This snippet is also terminated by `***END***` on a line by its own.

Lines are guaranteed to be no longer than 254 characters. No code fragment will be longer than 10000 lines. Any code and file name lines will only contain the ASCII characters 32-126. The total size of the input file will not exceed 10^6 characters.

Output specifications

For each test case, write the number of matching consecutive lines (empty lines not counted) in a longest match from the repository, followed by a space-separated list of the file names of each fragments containing a match of this length, given in the order that the matching fragments were presented in the repository description. If no fragments match, write the number 0 on a line of its own.

Sample input 1	Sample output 1

2
HelloWorld.c
int Main() {
  printf("Hello %d\n",i);
}
END
Add.c
int Main() {
  for (int i=0; i<10; i++)
  sum += i;
  printf("SUM %d", sum);
}
END
int Main() {
  printf("Hello %d\n",i);
  printf("THE END\n");
}
END
	2 HelloWorld.c

Sample input 2	Sample output 2
2 HelloWorld1.bas 10 PRINT "********************" 20 PRINT "********************" 30 PRINT "--- HELLO WORLD ---" 40 PRINT "********************" 50 PRINT "********************" ***END*** HelloWorld2.bas 10 PRINT "--------------------" 20 PRINT "********************" 30 PRINT "--- HELLO WORLD ---" 40 PRINT "********************" 50 PRINT "--------------------" ***END*** 10 REM Hello ver 1.0 (c) Acme 2008 20 PRINT "********************" 30 PRINT "--- HELLO WORLD ---" 40 PRINT "********************" 50 END ***END***	3 HelloWorld1.bas HelloWorld2.bas

Problem D

Dinner

For several years now, the Nordic Conference on Partitions and Combinatorics, NCPC, has had a growing number of participants. This year the organizing team is expecting an all time high record in the hundreds. Due to the politics of arranging this prestigious event, the conference site was decided a long time ago to be the Grand Hôtel in Stockholm. The hotel has two large dining halls, but unfortunately, each of these halls alone can only fit up to two thirds of the NCPC participants, so the participants are going to have to be divided in two groups.

This constraint calls for some thinking on behalf of the organizing team for the conference dinner: could they come up with some division of the participants in two parts, none of which is larger than 2/3 of the entire group, meeting some witty division rule suitable for the occasion, which they could tell the participants for their amusement? After all, as long as there is some grand logic rule to which of the two dining halls you are being seated in, you (as a mathematician) would be happy! They thought for a while and came up with the following idea for the division: Is there a year Y and a division of the participants in two parts such that every pair in the first part met for the first time some time *before* year Y, and every pair in the second part met for the first time some time *in or after* year Y? Now this clearly qualified as an appropriate rule to all of them, but the question was whether it would be possible?

Input specifications

The first line of input contains an integer $4 \leq n \leq 400$, the number of participants, and c, the number of known first encounters. The next c lines are each in the format $a\,b\,y$, meaning participant a and b ($1 \leq a < b \leq n$) met for the first time in year y ($1948 \leq y < 2008$). No pair of participants will appear more than once on the list, and every pair of participants not in the list is assumed to have meet only now (in the year 2008).

Output specifications

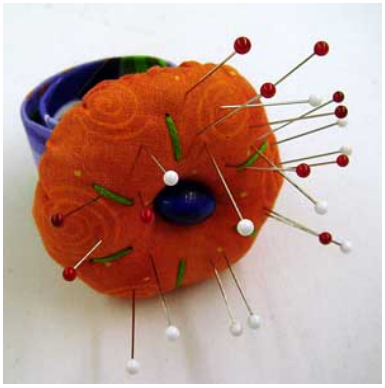
For each test case, either the smallest year Y such that it is possible to divide the participants in two parts, neither of which contains more than $2n/3$ people, such that all people in the first part first met before year Y, and all people in the second part first met in or after year Y. If there is no such year, output the string '`Impossible`'.

Sample input 1	**Sample output 1**
4 6 1 2 1987 2 3 1987 1 3 1987 2 4 1987 1 4 1987 3 4 1987	Impossible

Sample input 2	**Sample output 2**
6 3 1 2 1970 3 4 1980 5 6 1990	1971

Problem E

Event Planning



As you didn't show up to the yearly general meeting of the Nordic Club of Pin Collectors, you were unanimously elected to organize this years excursion to Pin City. You are free to choose from a number of weekends this autumn, and have to find a suitable hotel to stay at, preferably as cheap as possible.

You have some constraints: The total cost of the trip must be within budget, of course. All participants must stay at the same hotel, to avoid last years catastrophe, where some members got lost in the city, never being seen again.

Input specifications

The first line of input consists of four integers: $1 \leq N \leq 200$, the number of participants, $1 \leq B \leq 500000$, the budget, $1 \leq H \leq 18$, the number of hotels to consider, and $1 \leq W \leq 13$, the number of weeks you can choose between. Then follow two lines for each of the H hotels. The first gives $1 \leq p \leq 10000$, the price for one person staying the weekend at the hotel. The second contains W integers, $0 \leq a \leq 1000$, giving the number of available beds for each weekend at the hotel.

Output specifications

Output the minimum cost of the stay for your group, or "`stay home`" if nothing can be found within the budget.

Sample input 1	**Sample output 1**

`3 1000 2 3`
`200`
`0 2 2`
`300`
`27 3 20`
	`900`

Sample input 2	**Sample output 2**

`5 2000 2 4`
`300`
`4 3 0 4`
`450`
`7 8 0 13`
	`stay home`

Almost blank page

Problem F

Fixing the Bugs

A certain big IT company which we will not name in order not to get sued, are preparing to launch a new version of their flagship product. Having just been employed as a developer on the project, you have been given a list of open bugs that should be fixed in the new version.

Being bugs, you are not exactly certain how to fix them, even though you have some ideas. For each bug you are able to estimate your ability to quickly fix the bug. Of course, these estimates may be wrong, so if you try to fix a bug and fail, you will revise the estimate of your ability to fix this bug.

To be specific, we use the following probabilistic model for the bug fixing process: for each bug, there is an associated *fix probability* p. Every hour, you choose one bug to work on, and work on this bug for the entire hour (if you manage to fix the bug in less then an hour, you celebrate by having coffee and taunting your coworkers for the remaining part of the hour). The probability that you succeed in fixing the bug during this hour is p. If you fail to resolve the bug, the fix probability for this bug is reduced to $p \cdot f$, where $f \leq 1$ is a factor indicating how much faith you lose in your ability after a failure. The fix probabilities for the other bugs remain unchanged. The next hour, you again choose an open bug to work on, and so on. This is repeated until the new version is released, and you are allowed to go home and sleep.

In addition, each bug has a *severity* s indicating how severe the bug is (or alternatively, the value of fixing the bug). Clearly, it is possible that you will not manage to fix all the bugs before the product is released. In order to make as good an impression on your boss as possible, you would like to maximize the total severity of those bugs which you do manage to resolve, by carefully choosing which bugs to work on. What will be the expected value of the total severity of fixed bugs, provided that you, every hour, choose which bug to work on in such a way that this quantity is maximized?

Input specifications

The first line of input contains three numbers B, T and f, where $0 \leq B \leq 10$ is an integer giving the number of open bugs, $0 \leq T \leq 100$ is an integer giving the number of hours left until the new version is released, and $0 \leq f \leq 1$ is a real number as described above.

Each of the following B lines describe an open bug. Each such description contains two numbers p and s, where $0 \leq p \leq 1$ is a real number giving the initial fix probability of the bug and $0 \leq s \leq 10\,000$ is an integer giving the severity of the bug.

Output specifications

Output a line containing the expected total severity of bugs fixed, provided you work in a way which maximizes this quantity. Any answer with either absolute or relative error smaller than 10^{-6} is acceptable.

Sample input 1	**Sample output 1**
`1 2 0.950000` `0.700000 50`	`44.975`

Sample input 2	**Sample output 2**

`2 2 0.500000`
`0.750000 100`
`0.750000 20`
	`95.62500000000000000000`

Problem G

Getting Gold

We're building an old-school back-to-basics computer game. It's a very simple text based adventure game where you walk around and try to find treasure, avoiding falling into traps. The game is played on a rectangular grid and the player gets very limited information about her surroundings.

The game will consist of the player moving around on the grid for as long as she likes (or until she falls into a trap). The player can move up, down, left and right (but not diagonally). She will pick up gold if she walks into the same square as the gold is. If the player stands next to (i.e., immediately up, down, left, or right of) one or more traps, she will "sense a draft" but will not know from what direction the draft comes, or how many traps she's near. If she tries to walk into a square containing a wall, she will notice that there is a wall in that direction and remain in the position where she was.

For scoring purposes, we want to show the player how much gold she could have gotten safely. That is, how much gold can a player get playing with an optimal strategy and always being sure that the square she walked into was safe. The player does not have access to the map and the maps are randomly generated for each game so she has no previous knowledge of the game.

Input specifications

The first line of input contains two positive integers W and H, neither of them smaller than 3 or larger than 50, giving the width and the height of the map, respectively. The next H lines contain W characters each, giving the map. The symbols that may occur in a map are as follows:

- `P` – the player's starting position
- `G` – a piece of gold
- `T` – a trap
- `#` – a wall
- `.` – normal floor

There will be exactly one '`P`' in the map, and the border of the map will always contain walls.

Output specifications

Output the number of pieces of gold the player can get without risking falling into a trap.

Sample input 1	**Sample output 1**
7 4 ####### #P.GTG# #..TGG# #######	1

Sample input 2	**Sample output 2**
8 6 ######## #...GTG# #..PG.G# #...G#G# #..TG.G# ########	4

Problem H

Hard Evidence

The young reporter Janne is planning to take a photo of a secret government installation. He needs to obtain evidence of the many serious crimes against good sense that are being committed there, so as to create a scandal and possibly win a Pulitzer. Unfortunately, the base is surrounded by a high fence with high voltage wires running around. Janne does not want to risk being electrocuted, so he wants to take a photo from outside the fence. He can bring a tripod as high as the fence to take a photo, so if he wants he can stand right beside the fence and take his picture. The secret installation is a convex polygon. The fence has a form of a circle. Of course Janne wants to make a photo with maximal possible detail level. The detail level of the photo depends on the view angle of the base at the point from which the photo is taken. Therefore he wants to find a point to maximize this angle.

Input specifications

The first line of the input file contains two integer numbers: n and r — the number of vertices of the polygon and the radius of the fence ($3 \leq n \leq 200$, $1 \leq r \leq 1000$). The following n lines contain two real numbers each — the coordinates of the vertices of the polygon listed in counterclockwise order. It is guaranteed that all vertices of the polygon are strictly inside the fence circle, and that the polygon is convex. The center of the fence circle is located at the origin, $(0, 0)$.

Output specifications

Output the maximal view angle a for the photo ($0 \leq a < 2\pi$). Any answer with either absolute or relative error smaller than 10^{-6} is acceptable.

Sample input 1	**Sample output 1**
4 2 -1.0 -1.0 1.0 -1.0 1.0 1.0 -1.0 1.0	1.5707963268

Almost blank page

Problem I

Introspective Caching

In a distributed system, data is never where you need it, and fetching data over a network takes time and consumes bandwidth. The problem can be mitigated by adding a cache, where a node stores some resources locally and if those resources need to be used again, it can simply take them from its cache rather than asking someone else for them.

However, caches have a nasty tendency to fill up, so at some point, objects must be evicted from the cache to make room for new objects. Choosing what object to remove from the cache is not easy and there are several different algorithms to choose from.

The marvelous Apes in Computing Machinery have come up with a fantastic new algorithm, the Introspective Caching Algorithm, named after a city of Peru. It consists of some extra hardware (a very small, precognitive monkey) which helps making decisions. Since the monkey can see into the future, she knows exactly what objects will be accessed and in what order, and using this information she will make optimal decisions on what objects to remove from the cache. Optimality here means that she will minimize the number of times an object is read into the cache.

All object accesses go through the cache, so every time an object is accessed, it must be inserted into the cache if it was not already there. All objects are of equal size, and no writes occur in the system, so a cached object is always valid. When the system starts, the cache is empty.

You have been tasked with evaluating the monkey's performance, and feeding her the occasional banana.

Input specifications

The first line of input contains three integers, separated by single spaces, telling you how many objects fit in the cache, $0 < c \leq 10000$, how many different objects are in the system, $c \leq n \leq 100000$, and how many accesses, $0 \leq a \leq 100000$, will occur. The following a lines contain a single integer between 0 and $n - 1$ (inclusive) indicating what object is accessed. The first line corresponds to the first object accessed access and the last line to the last.

Output specifications

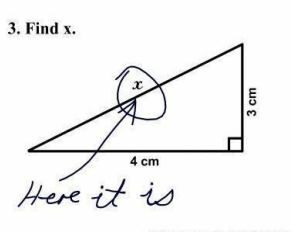
Output the least number of times an object must be read into the cache to handle the accesses listed in the input.

Sample input 1	**Sample output 1**
1 2 3 0 0 1	2

Sample input 2	**Sample output 2**
3 4 8 0 1 2 3 3 2 1 0	5

Problem J

Just A Few More Triangles!



Simon Haples is a somewhat peculiar person. Not quite hip, not quite square, he is more of a triangular nature: ever since childhood, he has had an almost unhealthy obsession with triangles. Because of his discrete nature, Simon's favorite kind of triangles are the Pythagorean ones, in which the side lengths are three positive integers a, b, and c such that $a \leq b$ and $a^2+b^2 = c^2$.

Recently, Simon has discovered the fantastic world of counting modulo some integer n. As you may imagine, he quickly realizes that there are multitudes of Pythagorean triples to which he has previously been oblivious! Simon therefore sets out to find all Pythagorean triples modulo n, i.e., all triples of integers a, b and c between 1 and $n-1$ such that $a \leq b$ and $a^2 + b^2 \equiv c^2 \pmod{n}$.

As Simon's best friend, you realize that there is not much hope in deterring Simon from his crazy plans, so you decide to help him by computing how many such triples there are, so that Simon will know when his work is done.

Input specifications

The input consists of a single integer n, satisfying $2 \leq n \leq 500\,000$.

Output specifications

Output the number of Pythagorean triples modulo n.

Sample input 1	Sample output 1
7	18

Sample input 2	Sample output 2
15	64

Problem K

Best Cow Line

FJ is about to take his N (1 <= N <= 30,000) cows to the annual "Farmer of the Year" competition. In this contest every farmer arranges his cows in a line and herds them past the judges.

The contest organizers adopted a new registration scheme this year: simply register the initial letter of every cow in the order they will appear (e.g., If FJ takes Bessie, Sylvia, and Dora in that order, he just registers BSD). After the registration phase ends, every group is judged in increasing lexicographic order (i.e., alphabetical order) according to the string of the initials of the cows' names.

FJ is very busy this year and has to hurry back to his farm, so he wants to be judged as early as possible. He decides to rearrange his cows, who have already lined up, before registering them.

FJ marks a location for a new line of the competing cows. He then proceeds to marshal the cows from the old line to the new one by repeatedly sending either the first or last cow in the (remainder of the) original line to the end of the new line. When he's finished, FJ takes his cows for registration in this new order.

Given the initial order of his cows, determine the least lexicographic string of initials he can make this way.

Input
* Line 1: A single integer: N
* Lines 2..N+1: Line i+1 contains a single initial ('A'..'Z') of the cow in the i-th position in the original line

Output
The least lexicographic string he can make. Every line (except perhaps the last one) contains the initials of 80 cows ('A'..'Z') in the new line.

6 A C D B C B	ACDBCB

Problem L

Train Timetable

A train line has two stations on it, A and B. Trains can take trips from A to B or from B to A multiple times during a day. When a train arrives at B from A (or arrives at A from B), it needs a certain amount of time before it is ready to take the return journey - this is the turnaround time. For example, if a train arrives at 12:00 and the turnaround time is 0 minutes, it can leave immediately, at 12:00.

A train timetable specifies departure and arrival time of all trips between A and B. The train company needs to know how many trains have to start the day at A and B in order to make the timetable work: whenever a train is supposed to leave A or B, there must actually be one there ready to go. There are passing sections on the track, so trains don't necessarily arrive in the same order that they leave. Trains may not travel on trips that do not appear on the schedule.

Input
The first line of input gives the number of cases, N (1 <= N <= 100). N test cases follow.
Each case contains a number of lines. The first line is the turnaround time, T (0 <= T <= 60), in minutes. The next line has two numbers on it, NA and NB (0 <= NA, NB <= 100). NA is the number of trips from A to B, and NB is the number of trips from B to A. Then there are NA lines giving the details of the trips from A to B.

Each line contains two fields, giving the HH:MM departure and arrival time for that trip. The departure time for each trip will be earlier than the arrival time. All arrivals and departures occur on the same day. The trips may appear in any order - they are not necessarily sorted by time. The hour and minute values are both two digits, zero-padded, and are on a 24-hour clock (00:00 through 23:59).
After these NA lines, there are NB lines giving the departure and arrival times for the trips from B to A.

Output
For each test case, output one line containing "Case #x: " followed by the number of trains that must start at A and the number of trains that must start at B.

2 5 3 2 09:00 12:00 10:00 13:00 11:00 12:30 12:02 15:00 09:00 10:30 2 2 0 09:00 09:01 12:00 12:02	Case #1: 2 2 Case #2: 2 0