**Oversized Pancake Flipper**

Problem

[Last year](https://code.google.com/codejam/contest/6254486/dashboard#s=p1), the Infinite House of Pancakes introduced a new kind of pancake. It has a happy face made of chocolate chips on one side (the "happy side"), and nothing on the other side (the "blank side").

You are the head cook on duty. The pancakes are cooked in a single row over a hot surface. As part of its infinite efforts to maximize efficiency, the House has recently given you an oversized pancake flipper that flips exactly **K** consecutive pancakes. That is, in that range of **K** pancakes, it changes every happy-side pancake to a blank-side pancake, and vice versa; it does *not* change the left-to-right order of those pancakes.

You cannot flip fewer than **K** pancakes at a time with the flipper, even at the ends of the row (since there are raised borders on both sides of the cooking surface). For example, you can flip the first **K** pancakes, but not the first **K** - 1 pancakes.

Your apprentice cook, who is still learning the job, just used the old-fashioned single-pancake flipper to flip some individual pancakes and then ran to the restroom with it, right before the time when customers come to visit the kitchen. You only have the oversized pancake flipper left, and you need to use it quickly to leave all the cooking pancakes happy side up, so that the customers leave feeling happy with their visit.

Given the current state of the pancakes, calculate the minimum number of uses of the oversized pancake flipper needed to leave all pancakes happy side up, or state that there is no way to do it.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with a string **S** and an integer **K**. **S** represents the row of pancakes: each of its characters is either + (which represents a pancake that is initially happy side up) or - (which represents a pancake that is initially blank side up).

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is either IMPOSSIBLE if there is no way to get all the pancakes happy side up, or an integer representing the the minimum number of times you will need to use the oversized pancake flipper to do it.

Limits

1 ≤ **T** ≤ 100.  
Every character in **S** is either + or -.  
2 ≤ **K** ≤ length of **S**.

Small dataset

2 ≤ length of **S** ≤ 10.

Large dataset

2 ≤ length of **S** ≤ 1000.

Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  ---+-++- 3  +++++ 4  -+-+- 4 | Case #1: 3  Case #2: 0  Case #3: IMPOSSIBLE |

In Case #1, you can get all the pancakes happy side up by first flipping the leftmost 3 pancakes, getting to ++++-++-, then the rightmost 3, getting to ++++---+, and finally the 3 pancakes that remain blank side up. There are other ways to do it with 3 flips or more, but none with fewer than 3 flips.

In Case #2, all of the pancakes are already happy side up, so there is no need to flip any of them.

In Case #3, there is no way to make the second and third pancakes from the left have the same side up, because any flip flips them both. Therefore, there is no way to make all of the pancakes happy side up.

**Alphabet Cake**

Problem

You are catering a party for some children, and you are serving them a cake in the shape of a grid with **R** rows and **C** columns. Your assistant has started to decorate the cake by writing every child's initial in icing on exactly one cell of the cake. Each cell contains at most one initial, and since no two children share the same initial, no initial appears more than once on the cake.

Each child wants a single rectangular (grid-aligned) piece of cake that has their initial and no other child's initial(s). Can you find a way to assign every blank cell of the cake to one child, such that this goal is accomplished? It is guaranteed that this is always possible. There is no need to split the cake evenly among the children, and one or more of them may even get a 1-by-1 piece; this will be a valuable life lesson about unfairness.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with two integers **R** and **C**. Then, there are **R** more lines of **C**characters each, representing the cake. Each character is either an uppercase English letter (which means that your assistant has already added that letter to that cell) or ?(which means that that cell is blank).

Output

For each test case, output one line containing Case #x: and nothing else. Then output **R**more lines of **C** characters each. Your output grid must be identical to the input grid, but with *every* ? replaced with an uppercase English letter, representing that that cell appears in the slice for the child who has that initial. You may not add letters that did not originally appear in the input. In your grid, for each letter, the region formed by all the cells containing that letter must be a single grid-aligned rectangle.

If there are multiple possible answers, you may output any of them.

Limits

1 ≤ **T** ≤ 100.  
There is at least one letter in the input grid.  
No letter appears in more than one cell in the input grid.  
It is guaranteed that at least one answer exists for each test case.

Small dataset

1 ≤ **R** ≤ 12.  
1 ≤ **C** ≤ 12.  
**R** × **C** ≤ 12.

Large dataset

1 ≤ **R** ≤ 25.  
1 ≤ **C** ≤ 25.

Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  3 3  G??  ?C?  ??J  3 4  CODE  ????  ?JAM  2 2  CA  KE | Case #1:  GGJ  CCJ  CCJ  Case #2:  CODE  COAE  JJAM  Case #3:  CA  KE |

The sample output displays one set of answers to the sample cases. Other answers may be possible.

**Steed 2: Cruise Control**

Problem

Annie is a bus driver with a high-stress job. She tried to unwind by going on a Caribbean cruise, but that also turned out to be stressful, so she has recently taken up horseback riding.

Today, Annie is riding her horse to the east along a long and narrow one-way road that runs west to east. She is currently at kilometer 0 of the road, and her destination is at kilometer **D**; kilometers along the road are numbered from west to east.

There are **N** other horses traveling east on the same road; all of them will go on traveling forever, and all of them are currently between Annie's horse and her destination. The i-th of these horses is initially at kilometer **Ki** and is traveling at its maximum speed of **Si**kilometers per hour.

Horses are very polite, and a horse H1 will not pass (move ahead of) another horse H2that started off ahead of H1. (Two or more horses can share the same position for any amount of time; you may consider the horses to be single points.) Horses (other than Annie's) travel at their maximum speeds, except that whenever a horse H1 catches up to another slower horse H2, H1 reduces its speed to match the speed of H2.

Annie's horse, on the other hand, does not have a maximum speed and can travel at any speed that Annie chooses, as long as it does not pass another horse. To ensure a smooth ride for her and her horse, Annie wants to choose a single constant "cruise control" speed for her horse for the entire trip, from her current position to the destination, such that her horse will not pass any other horses. What is the maximum such speed that she can choose?

Input

The first line of the input gives the number of test cases, **T**; **T** test cases follow. Each test case begins with two integers **D** and **N**: the destination position of all of the horses (in kilometers) and the number of other horses on the road. Then, **N** lines follow. The i-th of those lines has two integers **Ki** and **Si**: the initial position (in kilometers) and maximum speed (in kilometers per hour) of the i-th of the other horses on the road.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the maximum constant speed (in kilometers per hour) that Annie can use without colliding with other horses. y will be considered correct if it is within an absolute or relative error of 10-6 of the correct answer. See the [FAQ](https://code.google.com/codejam/faq.html#floating_point) for an explanation of what that means, and what formats of real numbers we accept.

Limits

1 ≤ **T** ≤ 100.  
0 < **Ki** < **D** ≤ 109, for all i.  
**Ki** ≠ **Kj**, for all i ≠ j. (No two horses start in the same position.)  
1 ≤ **Si** ≤ 10000.

Small dataset

1 ≤ **N** ≤ 2.

Large dataset

1 ≤ **N** ≤ 1000.

Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  2525 1  2400 5  300 2  120 60  60 90  100 2  80 100  70 10 | Case #1: 101.000000  Case #2: 100.000000  Case #3: 33.333333 |

In sample case #1, there is one other (very slow!) horse on the road; it will reach Annie's destination after 25 hours. Anything faster than 101 kilometers per hour would cause Annie to pass the horse before reaching the destination.

In sample case #2, there are two other horses on the road. The faster horse will catch up to the slower horse at kilometer 240 after 2 hours. Both horses will then go at the slower horse's speed for 1 more hour, until the horses reach Annie's destination at kilometer 300. The maximum speed that Annie can choose without passing another horse is 100 kilometers per hour.

**Ample Syrup**

Problem

The kitchen at the Infinite House of Pancakes has just received an order for a stack of **K**pancakes! The chef currently has **N** pancakes available, where **N** ≥ **K**. Each pancake is a cylinder, and different pancakes may have different radii and heights.

As the sous-chef, you must choose **K** out of the **N** available pancakes, discard the others, and arrange those **K** pancakes in a stack on a plate as follows. First, take the pancake that has the largest radius, and lay it on the plate on one of its circular faces. (If multiple pancakes have the same radius, you can use any of them.) Then, take the remaining pancake with the next largest radius and lay it on top of that pancake, and so on, until all **K** pancakes are in the stack and the centers of the circular faces are aligned in a line perpendicular to the plate, as illustrated by this example:



You know that there is only one thing your diners love as much as they love pancakes: syrup! It is best to maximize the total amount of exposed pancake surface area in the stack, since more exposed pancake surface area means more places to pour on delicious syrup. Any part of a pancake that is not touching part of another pancake or the plate is considered to be exposed.

If you choose the **K** pancakes optimally, what is the largest total exposed pancake surface area you can achieve?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with two integers **N** and **K**: the total number of available pancakes, and the size of the stack that the diner has ordered. Then, **N** more lines follow. Each contains two integers **Ri** and **Hi**: the radius and height of the i-th pancake, in millimeters.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the maximum possible total exposed pancake surface area, in millimeters squared. y will be considered correct if it is within an absolute or relative error of 10-6 of the correct answer. See the [FAQ](https://code.google.com/codejam/faq.html#floating_point) for an explanation of what that means, and what formats of real numbers we accept.

Limits

1 ≤ **T** ≤ 100.  
1 ≤ **K** ≤ **N**.  
1 ≤ **Ri** ≤ 106, for all i.  
1 ≤ **Hi** ≤ 106, for all i.

Small dataset

1 ≤ **N** ≤ 10.

Large dataset

1 ≤ **N** ≤ 1000.

Sample

|  |  |
| --- | --- |
| Input | Output |
| 4  2 1  100 20  200 10  2 2  100 20  200 10  3 2  100 10  100 10  100 10  4 2  9 3  7 1  10 1  8 4 | Case #1: 138230.076757951  Case #2: 150796.447372310  Case #3: 43982.297150257  Case #4: 625.176938064 |

In Sample Case #1, the "stack" consists only of one pancake. A stack of just the first pancake would have an exposed area of π × **R0**2 + 2 × π \* **R0** × **H0** = 14000π mm2. A stack of just the second pancake would have an exposed area of 44000π mm2. So it is better to use the second pancake.

In Sample Case #2, we can use both of the same pancakes from case #1. The first pancake contributes its top area and its side, for a total of 14000π mm2. The second pancake contributes some of its top area (the part not covered by the first pancake) and its side, for a total of 34000π mm2. The combined exposed surface area is 48000π mm2.

In Sample Case #3, all of the pancakes have radius 100 and height 10. If we stack two of these together, we effectively have a single new cylinder of radius 100 and height 20. The exposed surface area is 14000π mm2.

In Sample Case #4, the optimal stack uses the pancakes with radii of 8 and 9.

**Fresh Chocolate**

Problem

You are the public relations manager for a chocolate manufacturer. Unfortunately, the company's image has suffered because customers think the owner is cheap and miserly. You hope to undo that impression by offering a free factory tour and chocolate tasting.

Soon after starting the new project, you realized that the company owner's reputation is well-deserved: he only agreed to give away free chocolate if you would minimize the cost. The chocolate to be given away comes in packs of **P** pieces. You would like to open new packs for each tour group, but the owner insists that if there are leftover pieces from one group, they must be used with the next tour group before opening up any new packs.

For instance, suppose that each pack contains **P**=3 pieces, and that a tour group with 5 people comes. You will open two packs to give one piece to each person, and you will have one piece left over. Suppose that after that, another tour group with 6 people comes. They will receive the leftover piece, and then you will open two more packs to finish giving them their samples, and so you will have one piece left over again. If two groups with 4 people each come right after, the first of those will get the leftover piece plus a full pack, and the last 4 person group will get their pieces from two newly opened packs. Notice that you cannot open new packs until all leftovers have been used up, even if you plan on using all of the newly opened pack immediately.

In the example above, 2 out of the 4 groups (the first and last groups) got all of their chocolate from freshly opened packs. The other 2 groups got some fresh chocolate and some leftovers. You know that giving out leftovers is not the best way to undo the owner's miserly image, but you had to accept this system in order to get your cheap boss to agree to the project. Despite the unfavorable context, you are committed to doing a good job.

You have requests from **N** groups, and each group has specified the number of people that will come into the factory. Groups will come in one at a time. You want to bring them in in an order that maximizes the number of groups that get only fresh chocolate and no leftovers. You cannot reject groups, nor have a group get chocolate more than once, and you need to give exactly one piece to each person in each group.

In the example above, if instead of 5, 6, 4, 4, the order were 4, 5, 6, 4, a total of 3 groups (all but the 5 person group) would get only fresh chocolate. For that set of groups, it is not possible to do better, as no arrangement would cause all groups to get only fresh chocolate.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of two lines. The first line contains two integers **N**, the number of groups coming for a tour, and **P**, the number of pieces of chocolate per pack. The second line contains **N** integers **G1**, **G2**, ..., **GN**, the number of people in each of the groups.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the number of groups that will receive only fresh chocolate if you bring them in in an order that maximizes that number.

Limits

1 ≤ **T** ≤ 100.  
1 ≤ **N** ≤ 100.  
1 ≤ **Gi** ≤ 100, for all i.

Small dataset

2 ≤ **P** ≤ 3.

Large dataset

2 ≤ **P** ≤ 4.

Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  4 3  4 5 6 4  4 2  4 5 6 4  3 3  1 1 1 | Case #1: 3  Case #2: 4  Case #3: 1 |

Sample Case #1 is the one explained in the statement. Besides the possible optimal order given above, other orders like 6, 5, 4, 4 also maximize the number of groups with only fresh chocolate, although the groups that get the fresh chocolate are not necesarily the same. Notice that we only care about the number of groups that get the best experience, not the total number of people in them.

In Sample Case #2, the groups are the same as in Case #1, but the packs contain two pieces each. In this case, several ways of ordering them — for instance, 4, 4, 6, 5 — make all groups get only fresh chocolate.

In Sample Case #3, all groups are single individuals, and they will all eat from the same pack. Of course, only the first one to come in is going to get a freshly opened pack.

**Googlements**

Problem

Chemists work with periodic table elements, but here at Code Jam, we have been using our advanced number smasher to study *googlements*. A googlement is a substance that can be represented by a string of at most nine digits. A googlement of length L must contain only decimal digits in the range 0 through L, inclusive, and it must contain at least one digit greater than 0. Leading zeroes are allowed. For example, 103 and 001 are valid googlements of length 3. 400 (which contains a digit, 4, greater than the length of the googlement, 3) and 000 (which contains no digit greater than 0) are not.

Any valid googlement can appear in the world at any time, but it will eventually decay into another googlement in a deterministic way, as follows. For a googlement of length L, count the number of 1s in the googlement (which could be 0) and write down that value, then count the number of 2s in the googlement (which could be 0) and write down that value to the right of the previous value, and so on, until you finally count and write down the number of Ls. The new string generated in this way represents the new googlement, and it will also have length L. It is even possible for a googlement to decay into itself!

For example, suppose that the googlement 0414 has just appeared. This has one 1, zero 2s, zero 3s, and two 4s, so it will decay into the googlement 1002. This has one 1, one 2, zero 3s, and zero 4s, so it will decay into 1100, which will decay into 2000, which will decay into 0100, which will decay into 1000, which will continuously decay into itself.

You have just observed a googlement **G**. This googlement might have just appeared in the world, or it might be the result of one or more decay steps. What is the total number of possible googlements it could have been when it originally appeared in the world?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with a string **G**, representing a googlement.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the number of different googlements that the observed googlement could have been when it first appeared in the world.

Limits

1 ≤ **T** ≤ 100.  
Each digit in **G** is a decimal digit between 0 and the length of **G**, inclusive.  
**G** contains at least one non-zero digit.

Small dataset

1 ≤ the length of **G** ≤ 5.

Large dataset

1 ≤ the length of **G** ≤ 9.

Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  20  1  123 | Case #1: 4  Case #2: 1  Case #3: 1 |

In sample case #1, the googlement could have originally been 20, or it could have decayed from 11, which could have itself decayed from 12 or 21. Neither of the latter two could have been a product of decay. So there are four possibilities in total.

In sample case #2, the googlement must have originally been 1, which is the only possible googlement of length 1.

In sample case #3, the googlement must have been 123; no other googlement could have decayed into it.