# P1

Problem

There are a lot of great streetside food vendors in Manhattan, but without a doubt, the one with the tastiest food is the Code Jam Crepe Cart!

You want to find the cart, but you do not know where it is, except that it is at some street intersection. You believe that people from across Manhattan are currently walking toward that intersection, so you will try to identify the intersection toward which the most people are traveling.

For the purposes of this problem, Manhattan is a regular grid with its axes aligned to compass lines and bounded between 0 and **Q**, inclusive, on each axis. There are west-east streets corresponding to gridlines y = 0, y = 1, y = 2, …, y = **Q** and south-north streets corresponding to gridlines x = 0, x = 1, x = 2, …, x = **Q**, and people move only along these streets. The points where the lines meet — e.g., (0, 0) and (1, 2) — are intersections. The shortest distance between two intersections is measured via the[Manhattan distance](https://en.wikipedia.org/wiki/Taxicab_geometry) — that is, by the sum of the absolute horizontal difference and the absolute vertical difference between the two sets of coordinates.

You know the locations of **P** people, all of whom are standing at intersections, and the compass direction each person is headed: north (increasing y direction), south (decreasing y direction), east (increasing x direction), or west (decreasing x direction). A person is moving toward a street intersection if their current movement is on a shortest path to that street intersection within the Manhattan grid. For example, if a person located at (x0, y0) is moving north, then they are moving toward all street intersections that have coordinates (x, y) with y > y0.

You think the crepe cart is at the intersection toward which the most people are traveling. Moreover, you believe that more southern and western parts of the island are most likely to have a crepe cart, so if there are multiple such intersections, you will choose the one with the smallest non-negative x coordinate, and if there are multiple such intersections with that same x coordinate, the one among those with the smallest non-negative y coordinate. Which intersection will you choose?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with one line containing two integers **P** and **Q**: the number of people, and the maximum possible value of an x or y coordinate in Manhattan, as described above. Then, there are **P** more lines. The i-th of those lines contains two integers **Xi**and **Yi**, the current location (street corner) of a person, and a character **Di**, the direction that person is headed. **Di** is one of the uppercase letters N, S, E, or W, which stand for north, south, east, and west, respectively.

Output

For each test case, output one line containing Case #t: x y, where t is the test case number (starting from 1) and x and y are the horizontal and vertical coordinates of the intersection where you believe the crepe cart is located.

Limits

1 ≤ **T** ≤ 100.  
Time limit: 20 seconds per test set.  
Memory limit: 1GB.  
1 ≤ **P** ≤ 500.  
0 ≤ **Xi** ≤ **Q**, for all i.  
0 ≤ **Yi** ≤ **Q**, for all i.  
For all i, if **Xi** = 0, **Di** ≠ W.  
For all i, if **Yi** = 0, **Di** ≠ S.  
For all i, if **Xi** = **Q**, **Di** ≠ E.  
For all i, if **Yi** = **Q**, **Di** ≠ N.

Test set 1 (Visible)

**Q** = 10.

Test set 2 (Hidden)

**Q** = 105.

Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  1 10  5 5 N  4 10  2 4 N  2 6 S  1 5 E  3 5 W  8 10  0 2 S  0 3 N  0 3 N  0 4 N  0 5 S  0 5 S  0 8 S  1 5 W | Case #1: 0 6  Case #2: 2 5  Case #3: 0 4 |

In Sample Case #1, there is only one person, and they are moving north from (5, 5). This means that all street corners with y ≥ 6 are possible locations for the crepe cart. Of those possibilities, we choose the one with lowest x ≥ 0 and then lowest y ≥ 6.

In Sample Case #2, there are four people, all moving toward location (2, 5). There is no other location that has as many people moving toward it.

In Sample Case #3, six of the eight people are moving toward location (0, 4). There is no other location that has as many people moving toward it.

# P2

### Problem

Odin has some magical rings which produce copies of themselves. Each "X-day ring" produces one more X-day ring every X days after the day it came into existence. These rings come in six possible varieties: 1-day, 2-day, ..., all the way up to 6-day.

For example, a 3-day ring that came into existence on day 0 would do nothing until day 3, when it would produce another 3-day ring. Then, on day 6, each of those two rings would produce another 3-day ring, and so on.

You know that Odin had no rings before day 0. On day 0, some rings came into existence. At the end of day 0, Odin had Ri i-day rings, for each 1 ≤ i ≤ 6. You know that 0 ≤ Ri ≤ 100, for all i, and at least one of the Ri values is positive.

Fortunately, you also have access to the secret well of knowledge. Each time you use it, you can find out the *total* number of rings that Odin had at the end of a particular day between day 1 and day 500, inclusive. The well will give you the answer modulo 263, because even it can only hold so much information! Moreover, you can only use the well up to **W** times.

Your goal is to determine how many rings of each type Odin had at the end of day 0 — that is, you must find each of the Ri values.

### Input and output

This is an interactive problem. You should make sure you have read the information in the [Interactive Problems section](https://codingcompetitions.withgoogle.com/codejam/faq#interactive-problems) of our FAQ.

Initially, your program should read a single line containing two integers **T**, the number of test cases, and **W**, the number of times you are allowed to use the well of knowledge per test case. Then, you need to process **T** test cases.

In each test case, your program processes up to **W** + 1 exchanges with our judge. You may make up to **W** exchanges of the following form:

* Your program outputs one line with a single integer D between 1 and 500, inclusive.
* The judge responds with one line with a single integer: the total number of rings that Odin had at the end of day D, modulo 263. If you send invalid data (e.g., a number out of range, or a malformed line), the judge instead responds with -1.

After between 0 and **W** exchanges as explained above, you must perform one more exchange of the following form:

* Your program outputs one line with six integers R1, R2, R3, R4, R5, R6, where Rirepresents the number of i-day rings that Odin had at the end of day 0.
* The judge responds with one line with a single integer: 1 if your answer is correct, and -1 if it is not (or you have provided a malformed line).

After the judge sends -1 to your input stream (because of either invalid data or an incorrect answer), it will not send any other output. If your program continues to wait for the judge after receiving -1, your program will time out, resulting in a Time Limit Exceeded error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error. As usual, if the memory limit is exceeded, or your program gets a runtime error, you will receive the appropriate judgment.

### Limits

1 ≤ **T** ≤ 50.  
Time limit: 20 seconds per test set.  
Memory limit: 1GB.

#### Test set 1 (Visible)

**W** = 6.

#### Test set 2 (Hidden)

**W** = 2.

### Testing Tool

You can use this testing tool to test locally or on our servers. To test locally, you will need to run the tool in parallel with your code; you can use our [interactive runner](https://storage.googleapis.com/coding-competitions.appspot.com/interactive_runner.py) for that. For more information, read the [Interactive Problems section](https://codingcompetitions.withgoogle.com/codejam/faq#interactive-problems) of the FAQ.

### Local Testing Tool

To better facilitate local testing, we provide you the following script. Instructions are included inside. You are encouraged to add more test cases for better testing. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently.

If your code passes the testing tool but fails the real judge, please check the [Coding section](https://codingcompetitions.withgoogle.com/codejam/faq#coding) of our FAQ to make sure that you are using the same compiler as us.

[file\_downloadDownload testing\_tool.py](https://codingcompetitions.withgoogle.com/codejam/round/0000000000051706/0000000000122837)

### Sample Interaction

This interaction corresponds to Test set 1. Suppose that, unbeknownst to us, the judge has decided that Odin had one ring of each of the six types at the end of day 0.

t, w = readline\_int\_list() // Reads 50 into t and 6 into w

printline 3 to stdout // Asks about day 3.

flush stdout

n = readline\_int() // Reads 16 into n.

printline 1 to stdout // Asks about day 1.

flush stdout

n = readline\_int() // Reads 8 into n.

printline 1 1 1 4 0 0 to stdout

flush stdout // We make a guess even though we could have

// queried the well up to four more times.

verdict = readline\_int() // Reads -1 into verdict (judge has decided our

// solution is incorrect)

exit // Exits to avoid an ambiguous TLE error

Notice that even though the guess was consistent with the information we received from the judge, we were still wrong because we did not find the correct values.

# P3

### Problem

En garde! Charles and Delila are about to face off against each other in the final fight of the Swordmaster fencing tournament.

Along one wall of the fencing arena, there is a rack with **N** different types of swords; the swords are numbered by type, from 1 to **N**. As the head judge, you will pick a pair of integers (L, R) (with 1 ≤ L ≤ R ≤ **N**), and only the L-th through R-th types of swords (inclusive) will be available for the fight.

Different types of sword are used in different ways, and being good with one type of sword does not necessarily mean you are good with another! Charles and Delila have skill levels of **Ci** and **Di**, respectively, with the i-th type of sword. Each of them will look at the types of sword you have made available for this fight, and then each will choose a type with which they are most skilled. If there are multiple available types with which a fighter is equally skilled, and that skill level exceeds the fighter's skill level in all other available types, then the fighter will make one of those equally good choices at random. Notice that it is possible for Charles and Delila to choose the same type of sword, which is fine — there are multiple copies of each type of sword available.

The fight is *fair* if the absolute difference between Charles's skill level with his chosen sword type and Delila's skill level with her chosen sword type is at most **K**. To keep the fight exciting, you'd like to know how many different pairs (L, R) you can choose that will result in a fair fight.

### Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with a line containing the two integers **N** and **K**, as described above. Then, two more lines follow. The first of these lines contains **N** integers **Ci**, giving Charles' skill levels for each type of sword, as described above. Similarly, the second line contains **N** integers **Di**, giving Delila's skill levels.

### 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 choices you can make that result in a fair fight, as described above.

### Limits

1 ≤ **T** ≤ 100.  
0 ≤ **K** ≤ 105.  
0 ≤ **Ci** ≤ 105, for all i.  
0 ≤ **Di** ≤ 105, for all i.  
Time limit: 30 seconds per test set.  
Memory limit: 1GB.

#### Test set 1 (Visible)

1 ≤ **N** ≤ 100.

#### Test set 2 (Hidden)

**N** = 105, for exactly 8 test cases.  
1 ≤ **N** ≤ 1000, for all but 8 test cases.

### Sample

|  |  |
| --- | --- |
| Input | Output |
| 6  4 0  1 1 1 8  8 8 8 8  3 0  0 1 1  1 1 0  1 0  3  3  5 0  0 8 0 8 0  4 0 4 0 4  3 0  1 0 0  0 1 2  5 2  1 2 3 4 5  5 5 5 5 10 | Case #1: 4  Case #2: 4  Case #3: 1  Case #4: 0  Case #5: 1  Case #6: 7 |

In Sample Case #1, the fight is fair if and only if Charles can use the last type of sword, so the answer is 4.

In Sample Case #2, there are 4 fair fights: (1, 2), (1, 3), (2, 2), and (2, 3). Notice that for pairs like (1, 3), Charles and Delila both have multiple swords they could choose that they are most skilled with; however, each pair only counts as one fair fight.

In Sample Case #3, there is 1 fair fight: (1, 1).

In Sample Case #4, there are no fair fights, so the answer is 0.

In Sample Case #5, remember that the *duelists* are not trying to make the fights fair; they choose the type of sword that they are most skilled with. For example, (1, 3) is not a fair fight, because Charles will choose the first type of sword, and Delila will choose the third type of sword. Delila will not go easy on Charles by choosing a weaker sword!

In Sample Case #6, there are 7 fair fights: (1, 3), (1, 4), (2, 3), (2, 4), (3, 3), (3, 4), and (4, 4).

1

file\_uploadUpload file

Select languagekeyboard\_arrow\_down

Syntax pre-check

Show Test Input