UCF Local Contest — September 3, 2016

Majestic 10

filename: `majestic`
(*Difficulty Level:* `Easy`)

The movie "Magnificent 7" has become a western classic. Well, this year we have 10 coaches training the UCF programming teams and once you meet them, you'll realize why they are called the "Majestic 10"! The number 10 is actually special in many different ways. For example, in basketball, they keep track of various statistics (points scored, rebounds, etc.) and if a player has 10+ (10 or more) in a particular stat, they call it a double.

The Problem:

Given three stats for a basketball player, you are to determine how many doubles the player has, i.e., how many of the stats are greater than or equal to 10.

The Input:

The first input line contains a positive integer, *n*, indicating the number of players. Each of the following *n* input lines contains three integers (separated by a space and each between 0 and 100, inclusive), providing the three stats for a player.

The Output:

Print each input line as it appears in the input. Then, on the following output line, print a message indicating how many stats are greater than or equal to 10:

print `zilch` if none of the three stats is greater than or equal to 10,
print `double` if one of the three stats is greater than or equal to 10,
print `double-double` if two of the three stats are greater than or equal to 10,
print `triple-double` if all three stats are greater than or equal to 10.

Leave a blank line after the output for each player.

Sample Input:

```
4
5 0 8
30 10 50
20 5 20
5 100 6
```

Sample Output:

```
5 0 8
zilch

30 10 50
triple-double

20 5 20
double-double

5 100 6
double
```

UCF Local Contest — September 3, 2016

Phoneme Palindromes

filename: `palind`
(*Difficulty Level:* `Easy`)

A palindrome is a string that reads the same forward and backward, e.g., madam and abba. Since some letters sound the same (e.g., c and k), we define a phoneme palindrome as a string that sounds the same forward and backward, e.g., cak and ckckbbkcck.

The Problem:

Given the letters that sound the same and a string, you are to determine if the string is a phoneme palindrome.

The Input:

The first input line contains a positive integer, *n*, indicating the number of test cases to process. Each test case starts with an integer, p ($1 \leq p \leq 13$), indicating the count for pairs of letters that sound the same. Each of the following p input lines provides two distinct lowercase letters (starting in column 1 and separated by a space) that sound the same. Assume that no letter appears in more than one pair. The next input line for a test case contains an integer, q *(*$1 \leq q \leq 100$*)*, indicating the number of strings to test for phoneme palindrome. Each of the following q input lines provides a string (starting in column 1 and lowercase letters only) of length 1 to 50, inclusive.

The Output:

For each test case, print the header "`Test case #n:`", where n indicates the case number starting with 1. Then print each string for that test case followed by a space, followed by a message (`YES` or `NO`) indicating whether or not the string is a phoneme palindrome. Leave a blank line after the output for each test case.

Sample Input:

```
2
1
c k
6
a
cac
ck
cab
kaak
ckckkcck
2
a z
x s
5
abbbz
asxz
cx
sxxabzxss
ks
```

Sample Output:

```
Test case #1:
a YES
cac YES
ck YES
cab NO
kaak YES
ckckkcck YES

Test case #2:
abbbz YES
asxz YES
cx NO
sxxabzxss YES
ks NO
```

UCF Local Contest — September 3, 2016

Don't Break the Ice

filename: `break`
(*Difficulty Level:* `Medium`)

Mr. Pennybags is enthralled with his new board game which involves knocking out tiny plastic ice blocks from a square grid. Pennybags was never very good at games and would like to practice his breaking strategies without having to reset the board. He has asked Unified Coders For Games (UCF Games) to develop such a tool. Pennybags wants a program to take in the description of a board and a sequence of moves and determine the number of invalid moves.

The Problem:

Given the dimensions (number of rows and columns) of a square game board and a list of moves, determine the number of attempted moves that would knock out an ice block that is no longer in the board. Note that when an ice block is knocked out, other blocks may fall out of the board as well. More specifically, an ice block will fall unless it is in a complete row (the row contains all its ice blocks) or it is in a complete column (the column contains all its ice blocks). Note also that if the fall of block B_1 results in the fall of block B_2, then other blocks may fall as a result of block B_2 falling.

The Input:

The first input line contains a positive integer, *t*, indicating the number of game boards. This is followed by the data for each game. The first input line for each game contains two integers (separated by a space): the dimensions (length and width) of a square game board (between 1 and 50 inclusive) and the number of attempted moves in Pennybags' strategy (between 1 and 100 inclusive). This is followed by the row and column (separated by a space) for each attempted move, one move per line. Assume that the input *values* are valid, e.g., the row/column for an attempted move will always be a cell of the game board.

The Output:

For each game board, output "`Strategy #`*`b`*`: `*`i`*" where *b* is the game board number (starting with 1), and *i* is the number of invalid moves. Leave a blank line after the output for each game board.

Sample Input:

```
3
4 5
1 1
1 2
4 1
4 2
1 1
4 4
1 3
2 4
1 4
4 4
3 3
1 1
2 2
3 3
```

Sample Output:

```
Strategy #1: 2

Strategy #2: 1

Strategy #3: 0
```

Explanation of the Sample Input/Output:

In Game #1, “4 2” and the second “1 1” are invalid.
In Game #2, “1 4” is invalid.

UCF Local Contest — September 3, 2016

Wildest Dreams

filename: `dreams`
(*Difficulty Level:* `Medium`)

Secretly, Arup loves Taylor Swift's music. He covers for his addiction by claiming that Anya, his three-year-old daughter, forces him to play the Taylor Swift CD in the car whenever she's in it. The reality is that Arup also listens to the CD even when Anya isn't in the car.

Both of them have peculiar listening habits. Anya typically obsesses over a single song and will request that Arup play that song on infinite repeat while she is in the car. For example, the current song Anya is obsessed with is track 9, Wildest Dreams. So, when Anya enters the car, Arup immediately starts playing track 9 (from the beginning of the track) and this song is played repeatedly. Arup, however, wants to listen to the whole CD and not just one track repeatedly. So, whenever Anya exits the car, Arup just lets the CD play in its natural ordering (tracks play in order and when the end of the CD is reached, track 1 starts again). If Anya exits the car in the middle of her favorite song playing, Arup just lets it continue to play that track until its end and then advances to the next song. For our example, if Anya exits the car in the middle of track 9, Arup lets track 9 continue until its end and then advance to the next song (track 10 or if track 9 is the end of CD, track 1).

The Problem:

Arup is curious exactly how long he has listened to Anya's favorite song. Given a list of the lengths of each song, when Anya is in the car and the song Anya is obsessed with, calculate the amount of time Arup has to hear the song that Anya is obsessed with. Recall that when Anya enters the car, her song is played from the beginning and the song keeps repeating as long as Anya is in the car. You may assume that if Anya gets out of the car in the middle of Anya's song playing, Arup will continue listening to it instead of forwarding the CD to the next song but when Anya's song is finished the next song will continue. If Anya's song ends exactly when Anya is leaving the car, the next song will continue, i.e., Anya's song does not loop back to the beginning.

Since Anya changes which CD she's obsessed with periodically, write a program that can solve multiple instances of the problem.

The Input:

The first line of input will contain a single positive integer, n ($1 \le n \le 50$), representing the number of CDs Anya has been obsessed with.

The first line of input for each CD will contain two integers: t ($1 \le t \le 20$), the number of tracks on the CD, and k ($1 \le k \le t$), the track that Anya is obsessed with. The second line of input for each CD contains t space separated positive integers representing the lengths of each of the t tracks on the CD, in order, in seconds. No CD will be more than 86,400 seconds in length. The

third line of input for each CD contains a single positive integer, d ($1 \leq d \leq 100$), the number of days to evaluate. The following d lines contain information about each day. Each of these lines will start with an integer, s_i ($1 \leq s_i \leq 20$), the number of segments of driving for day i. This is followed by s_i positive integers, representing the lengths of each of the driving segments, in seconds. Assume that Anya is in the car for the odd-number segments (first, third, fifth, etc.) The sum of the lengths of each driving segment will never exceed 86,400, the number of seconds in a day.

The Output:

At the beginning of each CD, on a single line, output "`CD #c:`" where `c` is the CD number (starting with 1). Then, on the following d lines, where d represents the number of days that CD was played, output the number of seconds Arup listed to Anya's favorite song on the CD, for each day, respectively. Leave a blank line after the output for each CD.

Sample Input:

```
2
13 9
212 231 231 235 193 219 207 211 220 247 250 195 270
4
3 1000 900 1000
3 10000 10000 10000
1 2000
2 500 600
2 2
100 200
5
1 70
5 300 277 131 10000 58
2 200 50
2 201 50
2 199 50
```

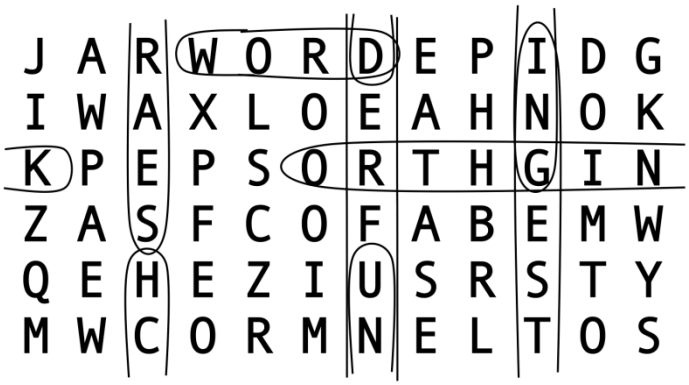
Sample Output:

```
CD #1:
2100
20780
2000
660

CD #2:
70
7335
200
251
200
```

UCF Local Contest — September 3, 2016

Loopy Word Search

filename: `search`
(*Difficulty Level:* `Medium`)



A word search puzzle is a grid of letters where your challenge is to find selected words as formed by consecutive letters in a line along the rows, columns, or diagonals of the grid. Tougher word searches also allow words in the grid to be forwards or backwards in any of those directions. In the "loopy word search", we will also allow words to go off the edge of the grid and continue (along the same line) on the other side, and potentially even reuse letters from that same word. However, in this problem, we won't search for words along diagonal lines, i.e., we only search along the rows and columns. (The UCF programming coaches are sure nice!)

The Problem:

Given a grid of letters and a list of words, identify the location of the first letter of each word in the grid and the direction in which remaining letters of the word can be found in sequence.

The Input:

The first input line contains a positive integer, *n*, indicating the number of word search puzzles. This is followed by the data for these puzzles. The first input line for each puzzle contains two positive integers (separated by a space): *r*, the number of rows in the grid (between 3 and 12 inclusive), and *c*, the number of columns in the grid (between 3 and 20 inclusive). Each of the next *r* input lines for the puzzle contains exactly *c* uppercase letters, with no spaces. The next input line for each puzzle contains a positive integer *s*, the number of words to search for. Each of the next *s* input lines contains a string of uppercase letters (length between 3 and 100 letters, inclusive) which is a word to search for. It is not necessarily a real word in any language.

Each of the *s* words will appear exactly once in the grid, meaning it has exactly one starting location and goes only in one direction. None of the words will be palindromes (same letters backwards and forwards). Assume that the input is valid as described here.

The Output:

For each word search puzzle, output the line "`Word search puzzle #`*p*`:`" where *p* is the puzzle number (counting from 1 in the input). Then, for each word given in that puzzle (and in the order given), output a line of the form "*d r c w*" where *w* is the word, *r* is the row in the grid where the first letter of the word is located (counting from 1), *c* is the column in the grid where the first letter is located (counting from 1), and *d* is the direction where the remaining letters of the word can be found, relative to the first letter, as given below. Output exactly one space after each of *d*, *r*, and *c*. For the direction *d*, use the following 1-letter codes:

Code		**Use for words with letters:**
"R"	➔	in the same row that go to the **right**, into subsequent columns, potentially wrapping to the first column of the same row
"D"	➓	in the same column that go **down**, into subsequent rows, potentially wrapping to the first row of the same column
"L"	➔	in the same row that go **left**, back into previous columns, potentially wrapping to the last column of the same row
"U"	➚	in the same column that go **up**, into previous rows, potentially wrapping to the last row of the same column

Leave a blank line after the output for each puzzle.

(Sample Input/Output on the next page)

Sample Input:

```
2
6 12
JARWORDEPIDG
IWAXLOEAHNOK
KPEPSORTHGIN
ZASFCOFABEMW
QEHEZIUSRSTY
MWCORMNELTOS
5
WORD
SEARCH
KNIGHTRO
UNDERFUND
INGESTING
3 7
UCFAEHT
KNIGHTS
CODETRY
2
AGE
THETHETHETHETH
```

Sample Output:

```
Word search puzzle #1:
R 1 4 WORD
U 4 3 SEARCH
L 3 1 KNIGHTRO
D 5 7 UNDERFUND
D 1 10 INGESTING

Word search puzzle #2:
D 1 4 AGE
U 3 5 THETHETHETHETH
```

UCF Local Contest — September 3, 2016

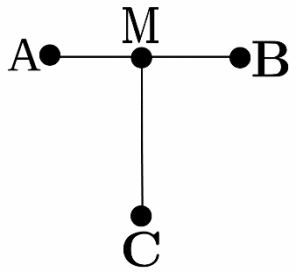
Dot the i's and Cross the T's

filename: `findt`
(*Difficulty Level:* `Medium`)

Mr. T, known for his hair cut and the phrase "you fool" in the old TV series "The A Team", has decided to try out for the UCF Programming Team. Considering the many talented students at UCF, Mr. T's best chance is to become great at geometry. So, he sought help from Euclid, one of the Fathers of Geometry. Considering the teacher and the student, the obvious place to start is with the letter T!

Consider the picture of the letter T to the right. We define four points A, M, B, and C form a T if three conditions holds:

1) M is the midpoint of AB
2) CM = AB, i.e., CM is the same length as AB
3) The angles AMC and BMC are 90 degrees.



The Problem:

Given a set of points, you are to determine how many groups of four points form a T based on the above definition.

The Input:

The first input line contains an integer, n *(1 ≤ n ≤ 100)*, indicating the number of test cases to process. Each test case starts with an integer, p ($4 \le p \le 50$), indicating the number of points. Each of the following p input lines provides two real numbers (each between -1000 and 1000, inclusive), indicating (respectively) the x and y coordinates for a point. Assume all points are distinct.

The Output:

For each set of points, print "`Set #`*n*`: `*m*", where *n* indicates the set number starting with 1 and *m* indicates how many groups of four points form a T. Two groups of four points are considered different if they differ in at least one point. Leave a blank line after the output for each set.

Note/Hint: This problem deals with floating-point numbers and one must be careful about checking for equality of two values. Assume two values are equal if they differ by 10^{-6} or less.

Sample Input:

```
2
5
5.0 2.0
3.0 6.0
5.0 6.0
7.0 6.0
5.0 10.0
4
-3.0 6.0
0.0 6.0
3.0 6.0
0.0 12.0
```

Sample Output:

```
Set #1: 2

Set #2: 1
```

UCF Local Contest — September 3, 2016

Jedi and the Galactic Empire

filename: `jedi`
(*Difficulty Level:* `Medium`)

Jedi Knights are often tasked with protection. Whether protecting shield generators or important diplomats, the Jedi will use their lightsabers to deflect blaster shots and keep their asset safe.

Sometimes a Jedi will go on missions alone or will be accompanied by another Jedi. When protecting an asset, they will stand side by side deflecting shots that would otherwise harm the asset they wish to protect. Sometimes, even together, they cannot block all the blaster shots. That is because Jedi are still limited by their physical reaction time. More specifically, when a Jedi blocks a blaster shot, he has to wait a certain amount of time before he can block another shot. For example, a Jedi that takes t time units between shots can block a shot arriving at time k and a shot arriving at time $k + t$ (or later).

So, Jedi will coordinate which shots they each will block and which shots they will allow to pass through their defense. Either Jedi can independently block each shot as long as there is enough time between the current shot and his last blocked shot. But determining which shots to block and which to let by is no easy task if they want to minimize the number of shots that reach their asset. That is why they seek aid from the other great power in the galaxy, programming.

As the Jedi's knowledge of programming is not as deep as their knowledge of the force, they have asked the programmers of the Universal Computational Federation (UCF) to find better strategies for blocking blaster shots with their lightsabers.

The Problem:

Given the times the blasters reach the Jedi, the number of Jedi, and their reaction time, determine the number of blaster shots that reach the asset they are trying to protect. Note that each Jedi can block a blaster shot at the beginning of the mission but after his first block the Jedi is limited by his reflexes (the time he has to wait before he can block another shot).

The Input:

The first input line contains a positive integer, n, indicating the number of protection missions the Jedi have been assigned. This is followed by the data for each mission. The first input line for each mission contains an integer, b ($1 \leq b \leq 1{,}000$), the number of blaster shots fired at their asset. This is followed by a line containing b numbers, separated by spaces, which are the times the blaster shots will reach the Jedi. These numbers can be in any order but will be positive integers less than 1,000,000. This is followed by an integer j ($1 \leq j \leq 2$) on a line by itself, which is the number of Jedi on the mission. The last input line for a mission contains j space separated integers giving the reaction time of each Jedi, the time it takes a Jedi to prepare to block the next blaster shot. These numbers will be between 1 and 100 inclusive.

The Output:

For each mission, output "`Mission #`*m*`: `*a*" where *m* is the mission number (starting with 1) and *a* is the minimum number of blaster shots that the Jedi are unable to block and will hit their asset. Leave a blank line after the output for each mission.

Sample Input:

```
4
5
10 5 5 10 5
1
100
4
2 4 9 9
2
10 7
5
2 4 8 13 13
2
10 7
5
2 4 6 8 10
1
2
```

Sample Output:

```
Mission #1: 4

Mission #2: 1

Mission #3: 1

Mission #4: 0
```

Explanation of the Sample Input/Output:

In Mission #2, Jedi with speed 7 can block 2 and one 9 and Jedi with speed 10 can block 4, resulting in one shot remaining unblocked.
In Mission #3, Jedi with speed 7 can block 4 and one 13 and Jedi with speed 10 can block 2 and the other 13, resulting in one shot remaining unblocked.

UCF Local Contest — September 3, 2016

Count the Dividing Pairs

filename: `divide`
(*Difficulty Level:* `Medium`)

Number Theory provides many fascinating properties. You have most likely written programs dealing with different groups of numbers such as Prime, Perfect, Amicable, Happy, Powerful, and Untouchable numbers, just to name a few. In this problem, you'll attack yet another fascinating property of numbers, one dealing with pairs of numbers.

An integer D is said to be a proper divisor of an integer N if $D \neq N$ and there exist an integer Q such that $N = Q * D$. For example, 4 is a proper divisor of 8 and 5 is a proper divisor of 15, but 9 is not a proper divisor of 9 and 6 is not a proper divisor of 8. Note that zero is not a proper divisor of any number but all numbers (except zero) are proper divisors of zero.

We will call (D, N) as defined above "proper dividing pairs".

The Problem:

Given a list of integers $A = \{A_1, A_2, \ldots, A_p\}$, you are to determine (count) the number of proper dividing pairs (A_i, A_j), where $1 \leq i, j \leq p$.

The Input:

The first input line contains a positive integer, *n*, indicating the number of test cases to process. Each test case starts with an integer, p ($2 \leq p \leq 10^6$), indicating the number of integers in the list. The following input line will provide p integers, A_i ($0 \leq A_i \leq 10^7$).

The Output:

For each test case, print "`Test case #`*`t`*`: `*`m`*", where *t* indicates the case number starting with 1 and *m* indicates the number of proper dividing pairs. Leave a blank line after the output for each test case.

Note that, as illustrated in Sample Input/Output, duplicate values in the input list are considered as different elements in the list and they each contribute to the total count (proper dividing pairs).

Sample Input:

```
5
3
1 2 3
4
1 2 3 1
2
7 5
3
29 0 17
10
32 16 8 4 2 2 4 8 16 32
```

Sample Output:

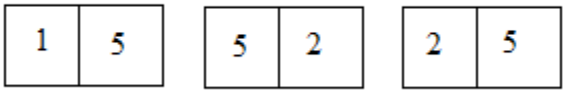
```
Test case #1: 2

Test case #2: 4

Test case #3: 0

Test case #4: 2

Test case #5: 40
```

UCF Local Contest — September 3, 2016

Lineup the Dominoes

filename: `dominoes`
(*Difficulty Level:* `Hard`)

Consider a game of solitary dominoes, where you are given several dominoes with a number of dots on both sides, with the number of dots ranging from 1 to 6, inclusive, and the goal is to line up the dominoes from left to right such that all touching sides between dominoes share the same number of dots. Here is a valid solution with a set of three dominoes:



D_1 D_2 D_3

Given dominoes D_1 through D_n, we consider a solution to be different than another solution if at least one domino is in a different location in the two solutions. Also, we allow dominoes to be flipped so, for example, a domino that reads 5, 2 from left to right can also be placed to read 2, 5 from left to right by flipping it. Note that for two solutions to be different, only the positions of dominoes matter, i.e., the values on the dominoes and the orientation of each domino does not matter. For example, let's assume D_1=[4,4] and D_2=[4,4]; the solution $\{D_1, D_2\}$ is different from the solution $\{D_2, D_1\}$ even though both solutions represent the same pattern: $\{D_1, D_2\}$=[4,4][4,4] and $\{D_2, D_1\}$=[4,4][4,4]. But the solution $\{D_1, D_2\}$ is not different from any other $\{D_1, D_2\}$ even if we flip one or both dominoes.

Using the three dominoes above, multiple solutions exist. One solution is shown above; another solution is D_1, D_3 and D_2, in sequence. We can get the latter to work by flipping both D_2 and D_3 compared to how they are shown above. Note again that these two solutions are different since one solution is $\{D_1, D_2, D_3\}$ and one is $\{D_1, D_3, D_2\}$, i.e., the position of D_2 has changed (and position of D_3 as well).

The Problem:

Given a set of dominoes, count the number of different solutions to the domino puzzle. As previously described, a correct solution will arrange all the dominoes in a line such that all touching sides between dominoes share the same number of dots. Since the number of solutions may be very large, calculate it mod $10^9 + 7$.

The Input:

The first input line contains a single positive integer, m ($1 \leq m \leq 100$), indicating the number of sets of dominoes to evaluate. This is followed by the data for these sets of dominoes.

The first line of input for each set of dominoes will contain an integer, n ($1 \leq n \leq 16$), the number of dominoes for that set. Each of the following n lines will contain two space separated integers, s_i ($1 \leq s_i \leq 6$) and t_i ($1 \leq t_i \leq 6$), representing the number of dots on each side of the i^{th} domino.

The Output:

For each set of dominoes, on a line by itself, output the number of different solutions to the domino puzzle mod $10^9 + 7$.

Sample Input:

```
5
3
1 5
5 2
2 5
2
1 3
4 5
3
1 2
3 4
1 4
4
1 1
1 1
1 1
1 1
4
3 5
3 5
3 5
3 5
```

Sample Output:

```
4
0
2
24
24
```

UCF Local Contest — September 3, 2016

Rising Tides

filename: `tides`
(*Difficulty Level:* `Hard`)

Last May, the UCF Programming Team attended the ACM ICPC World Finals in Phuket, Thailand. Besides the exciting programming contest, Thailand had some great sights to see!

The Phang Nga Bay in Thailand is home to hundreds of islands, some of which (e.g., James Bond Island) are famous from movie scenes. Others have lagoons inside that can only be reached by canoeing through caves on the water. Some caves have ceilings so low that canoers must lean over to make it through.

Besides navigating the tricky passages of the caves, canoers must be aware of the tides. Some caves can only be traversed in low tide. As the tides change, the sea level rises or falls while the explorers are paddling, and they must be careful to choose the correct path through the cave to avoid getting trapped. Of course, if they can make it through, they also want to minimize the amount of energy they spend leaning over in the canoe, i.e., they prefer higher ceiling heights when going through the caves.

In this problem, we assume that canoers start their journey at low tide and the sea level rises by one millimeter each second. Each cave is described by a two-dimensional grid (table) of numbers, where the j^{th} number in the i^{th} row indicates the initial height in millimeters of the cave ceiling at position (i, j) on the surface of the water. Because of the sea level change, the ceiling height (the distance from the sea level to the ceiling) at each cell decreases over time.

The cave can be traversed by starting at the first column of the first row (i.e., northwest corner) and ending at the last column of the last row (i.e., southeast corner). The canoe only moves in one of four directions in the two-dimensional grid (north, south, east, or west), one move at a

time. Each second, the canoe moves to an adjacent cell and the sea level increases by one millimeter, and the height of the cell above sea level must be greater than zero when the canoe enters it. The canoe's move and sea level change happen simultaneously, so the ceiling height may become zero just as the canoe is leaving. You may assume that the height of the cave above the canoe's initial position is greater than zero.

The Problem:

Given the description of a cave, you must find the path with the highest minimum ceiling height, or determine that it is impossible to traverse. Note that the number of cells the path goes through is not important; rather the heights of the cells are important; in particular, you are to find the path with the largest minimum ceiling height.

The Input:

The first line of input contains a single positive integer, *n*, indicating the number of caves to process. This is followed by *n* cave descriptions. Each cave description begins with a line containing two integers, *r* and *c* ($1 \leq r \leq 500$ and $1 \leq c \leq 500$), denoting the number of rows and columns, respectively. The next *r* lines each contain *c* space-separated integers, with the j^{th} number on the i^{th} line representing the height $a_{i,j}$ ($0 \leq a_{i,j} \leq 10^9$; $a_{1,1} > 0$) in millimeters of the cave ceiling above the initial sea level.

The Output:

For each cave, output a line with a single integer k denoting the largest minimum ceiling height, in millimeters, of a path through the cave, or the word `impossible` if the cave can't be traversed.

Sample Input:

```
2
4 5
9 5 4 0 0
9 4 8 9 12
9 6 8 7 12
0 0 9 8 12
3 1
10
1
10
```

Sample Output:

```
3
impossible
```

UCF Local Contest — September 3, 2016

Bouncing Bunnies

filename: `bunnies`
(*Difficulty Level:* `Hard`)

Connie and Ronnie, the bouncing bunnies, enjoy frolicking in the hills. They are both very adventurous and seek extreme weather changes. Connie loves changes in the temperature, so when she bounces from one hill to another, her happiness during that bounce is equal to $|T_A - T_B|$, where T_A is the temperature of the hill she jumped from, in bunny-degrees, and T_B is the temperature of the hill she jumped to, also in bunny-degrees. On the other hand, Ronnie loves changes in humidity, so when she bounces from one hill to another, her happiness during that bounce is equal to $|H_A - H_B|$, where H_A is the humidity of the hill she jumped from, in bunny humidity units, and H_B is the humidity of the hill she jumped to, also in bunny humidity units.

Connie and Ronnie are good friends, and would like to travel together across a field full of hills (starting at their home and ending at their favorite tree), but in order to relate to each other as well as possible, they would like Connie's happiness level to be equal to Ronnie's during every bounce (jump) they make.

The Problem:

Given the weather data for a field of hills, determine the minimum number of jumps needed for Connie and Ronnie to get from their home to their favorite tree. Bunnies are so good at bouncing that they can jump from any hill to any other hill, i.e., any hill is within a single bounce's distance of any other hill.

The Input:

The first input line contains a positive integer, t, indicating the number of fields to process. The description of each field will start (on a new line) with a positive integer, n ($2 \le n \le 500{,}000$), denoting the number of hills in the field. The following input line will contain n positive integers – the i^{th} number on this line, T_i ($1 \le T_i \le 10^9$), will denote the temperature of the i^{th} hill in bunny degrees. The next input line (the last line of each field description) will consist of n positive integers – the i^{th} number on this line, H_i ($1 \le H_i \le 10^9$), will denote the humidity of the i^{th} hill in bunny humidity units. The bunnies' home is located on hill 1, and their favorite tree is located on hill n.

The Output:

For each field, output must consist of a single line of the following form: "`Field #f: b`", where `f` is the field number in the input starting from 1 and `b` is an integer – the minimum number of bounces (jumps) needed for Connie and Ronnie to get from their home to their favorite tree, or the number -1 if such a journey cannot be made by the pair of bunnies. Leave a blank line after the output for each field.

Sample Input:

```
4
3
1 2 1
3 4 5
5
1 2 4 7 11
5 12 14 11 3
4
1 2 3 4
1 2 3 4
3
1 5 2
6 2 2
```

Sample Output:

```
Field #1: 2

Field #2: 4

Field #3: 1

Field #4: -1
```

UCF Local Programming Contest — Sept 3, 2016

Errata

Don't Break the Ice

Use "breakice" for Filename instead of "break" ("break" is a keyword and causes problems with class names in Java).

Dot the i's and Cross the T's

Assume that the x and y coordinates in the input will have at most three digits after the decimal point. This will limit the accumulation of partial errors while performing the intermediate operations.

Jedi and the Galactic Empire

The input section specifies a limit of "less than 1,000,000" for blaster shots reaching the Jedi; this should be "less than or equal to 1,000".