Fourteenth Annual Ohio Wesleyan University Programming Contest

April 6, 2024

Rules:

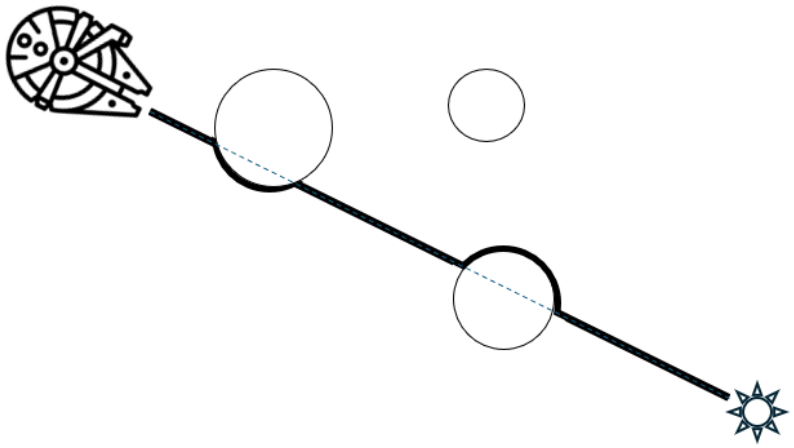
1. There are six questions to be completed in four hours.
2. All questions require you to read the test data from standard input and write results to standard output. **Do not** use files for input or output. **Do not** include any prompts, other debugging information, or any other output except for exactly what is specified by the problem.
3. The allowed programming languages are C++, Python 3, and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed.
6. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
7. All output cases will end in a newline character. So after your program finishes output, the cursor should be on the first column of the line immediately below your last case's output.
8. All communication with the judges will be handled in the PC^2 environment.
9. Judges' decisions are final. No cheating will be tolerated.

Question A: Asteroid Field

In the movie The Empire Strikes Back, Han Solo evades capture from the Empire by entering an asteroid field. This worked out for him, but how much flying did he have to do to avoid being hit?

Suppose that Han starts at a point (x_s, y_s) in the XY plane, and is trying to reach a destination point (x_f,y_f). Normally, he'd travel in a straight line, but the asteroids are in the way. Since all of the asteroids are conveniently circular, Han's path will travel on his straight line until he hits the asteroid, and then move around the circumference of the asteroid until he rejoins the travel line. Han, being a smart and wily pilot, will always take the path around an asteroid that is shorter.

Here is an example of the asteroid field and the path he will take:



The dashed line indicates the path Han would like to take, except for the asteroids in his way. The bold path shows the path he will actually take. Han's a good pilot, and so we can assume he will get as close to the asteroid as possible (so he travels the actual circumference of the asteroid, not further out).

Han's hyperdrive is leaking fuel, and he would like to know just how far he'll be traveling. Can you help him?

Input:

Each input case will begin with an integer n (1 <= n <=10), the number of asteroids. A value of n=0 denotes the end of input cases.

Otherwise, there will follow n lines of 3 integers: x, y, r. (-1000 <= x,y <= 1000, 1 <=r <= 100) This is the (x,y) position of the center of that asteroid, and the radius of that asteroid (all asteroids are perfect circles).
The input concludes with 4 integers: xs, ys, xf, yf (-1200 <= xs,ys,xf,yf <= 1200). Han starts at position (xs,ys) and is trying to reach position (xf,yf).

No two asteroids will overlap or touch, and Han's starting and ending positions will not be inside, or touching, asteroids.

Output:
For each input case i, output:
`Case i: d`
..where d is the distance traveled along the path from (xs,ys) to (xf,yf)

An answer will be judged correct if it is within .01 of the judge's answer.

Sample Input:

5
6 17 2
21 14 6
4 4 3
18 0 1
23 -2 2
1 21 31 4
3
5 4 1
7 -2 3
5 13 4
5 -6 5 8
0

Sample Output:

```
Case 1: 36.5505
Case 2: 15.7159
```

Question B: Eclipse Hunting

As you may know, a total eclipse is coming to the area on Monday. Bubba Bubbikins has never seen an eclipse and would love to see one. He has two problems:

- He does not live in the "totality zone", where the eclipse will be total. So he will have to drive to get inside that zone.
- He has to work on Monday, so has only a limited amount of time in which to drive to reach the totality zone. It's possible that this means Bubba can't reach a place with 100% eclipse coverage in time. If that's the case, Bubba would like to get as high a percentage of a partial eclipse as possible.

Bubba has mapped out locations and roads. Each location has a predicted "eclipse coverage", and distances to some neighboring locations. Bubba would like to get the maximum coverage possible within the time he has before the eclipse. If there are multiple locations he can reach with the same coverage, he'd like the one he can get to the earliest, so he can have time to settle in and get a good spot to watch the eclipse. Can you help him?

Input:

There will be several input instances. Each input instance begins with an integer n (<= 100), the number of locations in the instance, and t (<= 10000), the amount of time Bubba has to travel. A value of n=t=0 denotes the end of the input cases. Otherwise, there will then follow n integers on the next line. These are the coverage percentages of each location, and will be values between 0 and 100 (100 means total coverage). The first location is Bubba's starting point. It may or may not be the case that any location has 100% coverage.

There will then be an integer r, showing the number of roads. There will then be r lines, with three integers on each line separated by spaces: v,w, and d. v and w are numbers between 0 and n-1 and denote locations. d (1 <= d <= 1000) is the time it takes to travel between locations s and t (in either direction.) If a road between two locations is not specified, direct travel is not possible between those locations.

Output:

For each input case I, output:

```
Case i: c e
```

..where c is the maximum eclipse coverage Bubba can reach, and e is the minimum amount of time Bubba can reach that coverage.

Sample Input:

```
8 15
40 10 29 95 100 70 60 50
7
0 1 5
1 2 15
2 3 10
2 4 20
0 5 10
5 6 2
5 7 6
8 50
40 80 29 95 100 70 60 50
7
0 1 5
1 2 15
2 3 10
2 4 20
0 5 10
5 6 2
5 7 6
0 0
```

Sample Output:

```
Case 1: 70 10
Case 2: 100 40
```

Question C: Grid Hopping

Suppose we have a two-dimensional grid we'd like to traverse from some start position to some end position However, one does not just walk across the grid. Instead, each grid holds a number, which is the number of spaces to move in a straight line (up, down, left, or right) to get to the next position.

So, here is a small grid that corresponds to the first sample input:

3	4	2	5	1
1	7	2	4	4
2	2	2	0	3
1	**1**	**2**	4	**1**
1	0	1	1	**0**

If we start in the upper left, and are trying to get to the lower right, we can follow the bold numbers by going down 3 spaces, then right once, then right once more, then right 2 spaces from there. This puts us on the 1 above the end, so moving 1 space down will get us to the ending space in 5 moves.

It's possible that there may be multiple ways to cross the grid, depending on what decisions you make along the way. We're interested in finding the one with the least number of moves. It's also possible that there is no sequence of moves that will lead to the exit. We'd like to know that as well.

Input:
Each input instance will begin with two integers r and c (1 <= r,c <= 20) denoting the number of rows and columns of the grid. A value of r=c=0 denotes the end of the input cases.
Then will follow r lines of c integers. Each integer will be non-negative, and will indicate the step size of the corresponding cell of the grid.
The input instance will conclude with 4 integers: sr,sc, er, ec. (0 <= sr, er <r , 0 <= sc, ec < c).
The starting row and column is denoted by sr and sc, and the end is denoted by er and ec.
The upper left corner is row 0, column 0, and the bottom right corner is row r-1, column c-1.

Output:
For each input instance I, output either:
`Case i: m`
..where m is the minimum number of moves needed to travel from the start to the end.

Or

```
Case i: Impossible
```

..if it is not possible to reach the ending position from the starting position

Sample Input:

```
5 5
3 4 2 5 1
1 7 2 4 4
2 2 2 0 3
1 1 2 4 1
1 0 1 1 0
0 0 4 4
3 3
9 9 9
9 1 9
9 9 9
1 1 2 2
0 0
```

Sample Output:

```
Case 1: 5
Case 2: Impossible
```

Question D: Postmodern Poet

Patty is a poet (and she knows it). But she doesn't just create any poetry. She builds poems out of well-known blocks of text according to a strict process.

Her process is this:

- First, she chooses a *line length*, g, and a block of text to go through.
- Then, she copies words from the text, one at a time, putting a single space between each word. If the word (with or without the space at the end) exactly reaches the end of the line, she'll start a new line.
- If the word she is considering doesn't fit in the remaining space on the line, she throws out the word, and moves on to the next. She will keep doing this until she can place a word, or she runs out of words.

So, suppose her text is from John Lennon's "Imagine":

```
You may say I'm a dreamer, but I'm not the only one. I hope
someday you'll join us. And the world will live as one.
```

If her line length was 10, she would:

- Fit the words "You", and "may" on the first line. With the spaces between words, there are only 2 spaces left on the line. So the words "say" and "I'm" are skipped, and the word "a" fits on the line. The space at the end of "a" brings us to a line length of 10, so we can move on to the next line
- The next word is "dreamer," (Patty does *not* trim any punctuation). That takes 8 letters, plus 1 for the space. We have one space left on the line, so we skip all the way down to the word "I". We do not put a space after the I, because we're exactly at the end of the line, and move on.
- The third letter starts with "hope" (4 letters, plus the space), so the next word that fits on that line is "join". The space after "join" fills the rest of the line, so we move on.
- The fourth line will start with "us.", followed by "And". We have only room for a 2 letter word, so we skip down to "as".
- The final line is just the last word "one.".

The first test case has the way this output should look.

Patty would like your help rapidly generating lots of poems, so she can see which ones look the best, and will be included in her new poetry book "Quotespiration".

Input:

There will be several input cases. Each will begin with two integers, n (1 <= n <= 1000), the number of words in the text, and g (1 <= g <= 100), the line length. A value of n=g=0 denotes the end of the input

There will then follow n strings, on one or more lines of input. Strings will be separated by whitespace (possibly spaces, possibly tabs, possibly newlines). Each string's length will be <= g.

Output:

For each new block of text, I, output:

```
Poem i
```

Followed by a newline.

Then the text of the poem following the rules above. Each line of the poem itself (not the "`Poem i`" line should begin with a line number and a colon.

Sample Input:

25 10
You may say I'm a dreamer, but I'm not the only one. I hope someday you'll join us. And the world will live as one.
7 7
Hey ChatGPT, give me 500 poems please.
0 0

Sample Output:

```
Poem 1
1: You may a
2: dreamer, I
3: hope join
4: us. And as
5: one.
Poem 2
1: Hey me
2: 500
```

Queston E: Real Hard Mode

You're probably familiar with the New York Times game "Wordle", but if not, here are the rules:

The player is trying to guess a secret 5-letter word. Each turn, the player submits their own five letter word. Each letter in the guess is then colored in one of three ways:

- Green means that the letter exists in that position in the secret word. We'll use the letter 'G' to represent a green letter.
- Yellow means that the letter exists in the secret word, but at a different position. We'll use the letter 'Y' to represent a yellow letter.
- Grey (or "blank") means that the letter does not exist in the secret word. We'll use the character '-' to represent a blank.

Letters in the guess are processed by color- greens first, then yellows, then blanks. Ties for the same color are resolved left to right. This matters for duplicate letters. For example, if the secret word was "STALL", and the guess was "SHALT", the answer would be colored "G-GGY" (the L in the guess matches the first L in the secret word exactly). But if the guess was "SALSA", the response would be "GYY--": The first S is colored green, the first A is in the wrong place, the L is in the wrong place, and even though the S and A are also incorrect, we've already "scored" those letters, so they turn blank.

What you may not know is that Wordle lets you play in "hard mode". What "hard mode" is supposed to do is disallow guesses that are inconsistent with what you have already learned. So, after our "SHALT" guess, we would only be allowed to make guesses that have an S in the first letter, an A in the third letter, and an L in the fourth letter.

Specifically, Hard Mode should check (note that the "real" Wordle hard mode does <u>not</u> check all of these, though it should):

1. If a letter is marked green, it needs to show up in that position in the next guess.
2. If a letter is marked yellow, it needs to show up someplace in the next guess, and be in a different position as the position in the previous guess.
3. If a letter is repeated multiple times, all copies that are marked green should show up in the next guess (this comes from rule 1), and all copies that are marked yellow should show up in different positions in the same guess (this comes from rule 2). There should be at least as many copies of this letter in the next guess as there were yellow and green responses to that letter in the previous guess.
4. If a letter is marked blank, it should not show up in the next guess (although a copy that was colored green or yellow should show up, following the above rules)

Your job is to enforce the "real" hard mode for a series of guesses. (The real game also checks to see if the words are in the dictionary- you don't need to do that)

Input
Each input case will begin with a number n, the number of guesses. A value of n=0 denotes end of input.

There will then follow n lines. Each line will consist of two 5-letter strings, separated by spaces. The first string will be the guess. The second string will be the coloring of the guess. (so 5 characters that are all 'G', 'Y', or '-'). The colorings will be consistent with the rules of Wordle, with one unchanged secret word per input case.

Output:
For each input case i, output either:
`Case i: Illegal at word j`
..where "j" is the number of the first guess (starting from 1) that breaks the hard mode rules specified above.

Or

`Case i: Correct!`
..if all guesses are legal

Sample Input:
```
3
SALSA GYY--
SHALT GGGG-
SHALT GGGG-
3
SILLY -GY--
LIPID GG---
LIMBO GGGGG
0
```

Sample Output:
```
Case 1: Illegal at word 3
Case 2: Correct!
```

Question F: Yoga Singing

For your New Year's resolution, you've decided to practice mindfulness and attend a Yoga class once a week. When you get to the class, you see that lots of other people have the same idea. The room is arranged into a grid of "squares" where people place their Yoga mats. The room looks kind of like this:

FRONT

S	X	T
T	T	S
X	T	T

BACK

The 'X' spaces are empty. The 'T' spots are taken. The 'S' spots are not only taken, they're occupied by people who are so moved by their yoga-ing that they actually sing out loud during the class. You're all for people getting into things, but the singing does distract you from <u>your</u> exercise.

Over the course of a few classes, you've noticed a few things about the singers:

- Each signer sings differently, but you have managed to quantify each one's "disruption factor" as an integer.
- Depending on where you are relative to them, the "disruption factor" decreases as they get to you:
    - If they are in front of you (either directly or diagonally), the disruption factor decreases by 3 for each row they are in front of you.
    - If they are in the same row as you, the disruption factor decreases by 2 for each space they are away from you.
    - If they are behind you (either directly or diagonally), since they are projecting towards you, the disruption factor only decreases by 1 for each row they are behind you.
- A class may have <u>several</u> signers, and your total disruption (which you'd like to minimize) at a location is the sum of the disruptions you get from all of the singers at that spot.

So, in the example above, there are two open spots, and two singers. Suppose the singer in the front row has a disruption factor of 20, and the singer in the second row has a disruption factor of 30.

If you used the empty space in the front row, your disruption from the first singer would be 18, and your disruption from the second singer would be 29, for a total of 47. But if you

used the back row, your disruption would only be 14 from the first singer and 28 front the second singer, for a total disruption of just 42.

A disruption factor from a singer may go down to zero, if the singer is quiet enough and/or far enough away. It will never become negative.

When you get to class, you can quickly scan the room for the singers, and would like to calculate the minimum disruption you'd face over all of the open positions in the room.

Input: Each input will begin with 2 integers r and c (1 <= r,c <= 20), the number of rows and columns in the classroom. A value of r=c=0 denotes the end of the input.

Otherwise, there will be an integer e (1 <= e <= r*c), denoting the number of empty spaces. There will then follow e lines contain the row and column of each empty space.

Next will follow an integer s (0 <= s <= r*c-1), denoting the number of singers. There will then follow s lines with three integers: the row, column, and disruption of each singer. No singer will be in a place marked empty. All spaced not marked as empty, or having a singer are considered taken by non-singing people.

All positions will have rows between 1 and r, and columns between 1 and c. Row 1 is the front of the room, column 1 is the leftmost column.

Output: For each case *i*, output the line:

```
Case i: d
```

..where 'd' is the minimum disruption factor you will face out of any of the open spots.

Sample Input:

```
3 3
2
1 2
3 1
2
1 1 20
2 3 30
0 0
```

Sample Output:

```
Case 1: 41
```