Label Insight - Backend Code Test

Instructions

Welcome to the Label Insight Code Test.
Please read the following carefully as your solution may be rejected if you do not follow these instructions.

1. We expect that solutions are submitted with a reference to the tools and technologies that Label Insight uses to help show your competency for our teams.

2. Your solution will be judged based on its correctness, efficiency, design, and clarity (including the use of indentation and comments) as well as industry best practice for code maintainability. No form of plagiarism is permitted – come up with your own solution!

3. If you have questions about the problem set, please use your best judgement and carefully document your assumptions in your code or an accompanying readme file.

4. When you have your final solution to all problems, please reply to this email with your solution as a .zip, .tar, or .tar.gz attachment. Do not include any binaries, only source code. Please also provide a readme on how to run the code, assumptions made.

Thank you for participating in the code test!

Problem 1 - Missing Letters

The sentence "A quick brown fox jumps over the lazy dog" contains every single letter in the alphabet. Such sentences are called pangrams. You are to write a method getMissingLetters, which takes as input a string containing a sentence and returns all the letters not present at all in the sentence (i.e., the letters that prevent it from being a pangram). You should ignore the case of the letters in sentence, and your return should be all lower case letters, in alphabetical order. You should also ignore all non-alphabet characters as well as all non-US-ASCII characters.

Imagine that the method you write will be called many thousands of times in rapid succession on strings with length ranging from 0 to 50. Accordingly, you should try to write code that runs as quickly as possible. Also, imagine the case when the input string is quite large (e.g., tens of megabytes). See if you can develop an algorithm that handles this case efficiently while still running very quickly on smaller inputs.

Examples:
(Note that in the examples below, the double quotes should not be considered part of the input or output strings.)
0) "A quick brown fox jumps over the lazy dog"
Returns: ""
(This sentence contains every letter)
1) "A slow yellow fox crawls under the proactive dog"
Returns: "bjkmqz"
2) "Lions, and tigers, and bears, oh my!"
Returns: "cfjkpquvwxz"
3) ""
Returns: "abcdefghijklmnopqrstuvwxyz"

Problem 2 - Animation

A collection of particles is contained in a linear chamber. They all have the same speed, but some are headed toward the right and others are headed toward the left. These particles can pass through each other without disturbing the motion of the particles, so all the particles will leave the chamber relatively quickly.

You will be given the initial conditions by a string 'init' containing at each position an 'L' for a leftward moving particle, an 'R' for a rightward moving particle, or a '.' for an empty location. 'init' shows all the positions in the chamber. Initially, no location in the chamber contains two particles passing through each other.

We would like an animation of the particles as they move. At each unit of time, we want a string showing occupied locations with an 'X' and unoccupied locations with a '.'. Create a method 'animate' that takes a positive integer 'speed' and a string 'init' giving the initial conditions. The speed is the number of positions each particle moves in one unit of time. The method will return an array of strings in which each successive element shows the occupied locations at each time step. The first element of the return should show the occupied locations at the initial instant (at time = 0) in the 'X', '.' format. The last element in the return should show the empty chamber at the first time that it becomes empty.

Again, imagine that the method you write will be called thousands of times for varying initial conditions with size ranging from 0 to 50, and also imagine the case when init is several hundred thousand locations in size (though with speed > size / 20 or so). Try to handle both of these cases efficiently in your implementation.

Examples:
(Note that in the examples below, the double quotes should not be considered part of the input or output strings.)
0) 2, "..R...."
Returns:

```
  { "..X....",
    "....X..",
    "......X",
    "......." }
```

The single particle starts at the 3rd position, moves to the 5th, then 7th, and then out of the chamber.
1) 3, "RR..LRL"
Returns:

```
  { "XX..XXX",
    ".X.XX..",
    "X.....X",
    "......." }
```

Note that, at the first time step after init, there are actually 4 particles in the chamber, but two are passing through each other at the 4th position
2) 2, "LRLR.LRLR"
Returns:

```
  { "XXXX.XXXX",
    "X..X.X..X",
    ".X.X.X.X.",
    ".X.....X.",
    "........." }
```

At time 0 (init) there are 8 particles. At time 1, there are still 6 particles, but only 4 positions are occupied since particles are passing through each other.
3) 10, "RLRLRLRLRL"

Returns:
```
  { "XXXXXXXXXX",
    ".........." }
```
These particles are moving so fast that they all exit the chamber by time 1.

4) 1, "..."

Returns:
```
  { "..." }
```
5) 1, "LRRL.LR.LRR.R.LRRL."

Returns:
```
  { "XXXX.XX.XXX.X.XXXX.",
    "..XXX..X..XX.X..XX.",
    ".X.XX.X.X..XX.XX.XX",
    "X.X.XX...X.XXXXX..X",
    ".X..XXX...X..XX.X..",
    "X..X..XX.X.XX.XX.X.",
    "..X....XX..XX..XX.X",
    ".X.....XXXX..X..XX.",
    "X.....X..XX...X..XX",
    ".....X..X.XX...X..X",
    "....X..X...XX...X..",
    "...X..X.....XX...X.",
    "..X..X.......XX...X",
    ".X..X.........XX...",
    "X..X...........XX..",
    "..X.............XX.",
    ".X...............XX",
    "X.................X",
   "..................." }
```