HW8: Pretty Printing

Overview

💡 This assignment is adapted from Exercise 6.6 in *Algorithm Design* by Kleinberg and Tardos (2006). A link to the book is available through the Colgate Library reserves. The following introductory text is partly taken from the book.

In a word processor, the goal of "pretty-printing" is to take text with a ragged right margin, like this:

```
Call me Ishmael. Some years ago, never mind how long precisely, having little
or no money in my purse, and nothing particular to interest me on shore, I
thought I would sail about a little and see the watery part of the world.
```

and turn it into text whose right margin is as "even" as possible, like this:

```
Call me Ishmael. Some years ago, never mind how long precisely, having little
or no money in my purse, and nothing particular to interest me on shore, I
thought I would sail about a little and see the watery part of the world.
```

In other words, we take the sequence of words in a paragraph and decide where to put line breaks. In particular, we keep the sequence of words in the same order, but decide where the line breaks should be.

For the purpose of this assignment:

- Let a *word* be a contiguous sequence of non-whitespace characters, for example, `Ishmael.`, `years`, or `ago,`.
- Assume we're doing this in the context of a fixed-width font; so, we're just concerned with the number of characters that end up on each line and we don't need to know how much "space on a page" that a character consumes.
- In the output, we assume words on a line are separated by a single space.
- We are not allowed to separate or hyphenate words.

Optimization Problem

To make this precise, we need a definition of what it means for the margin to be "even."

Slack of a line

Our definition depends on the *slack* of a line, which represents the amount of space between the last word in the line and the right margin. Therefore, an input to the problem is L, the maximum length of a line in the output, given by the maximum number of characters allowed on any line. (For example, in the example above, we could have $L = 40$; no line in the output contains more than 40 characters.)

Suppose we are given an input sequence of n words, and the lengths of those words are given by the sequence $w_0, w_1, w_2, \ldots, w_{n-1}$.

Suppose that we decide to put our first line break after the fifth word. That means that the first line will have the first five words and four single spaces separating those words; so, it contains

$$w_0 + w_1 + w_2 + w_3 + w_4 + 4$$

characters. That means the slack on that line is

$$L - (w_0 + w_1 + w_2 + w_3 + w_4 + 4).$$

In general, if a line in our output contains words i through j, then the slack on that line is

$$L - (w_i + w_{i+1} + \cdots + w_{j-1} + w_j + (j - i))$$

where the last $(j - i)$ term accounts for the number of spaces in between the words.

Optimization goal

We assume we are given some function f to apply to the slacks of the lines. For example, in the Kleinberg-Tardos version of the problem, $f(x) = x^2$, so the slack of each line is squared.

Here is the metric we use to evaluate a solution. If the output solution contains k line breaks, then the output has $k + 1$ lines, and let the slacks of those lines be $s_0, s_1, \ldots, s_k$. We apply the given function f to those slacks and sum the result:

$$M = \sum_{i=0}^{k} f(s_i)$$

The optimization goal is to *minimize* M.

(So, for the Kleinberg-Tardos version where $f(x) = x^2$, we want to minimize the sum of the squares of the slacks.)

Specification

In a file `PrettyPrint.java` containing a `public class PrettyPrint`, implement a function

`static List<Integer> splitWords(int[] lengths, int L, SlackFunctor sf)`

that solves the problem described above.

The input to this function consists of three parameters.

1. The first is an integer array `lengths` of length n; `length[i]` represents the number of characters in the ith word of the input. Note that these numbers don't include any whitespace that may have been in between the original words (as that is irrelevant in our problem). For example, if the input sequence of words is:

   ```
   Call me Ishmael. Some years ago, never mind how long
   ```

   then the input array `lengths = {4, 2, 8, 4, 5, 4, 5, 4, 3, 4}`.
2. The second is an integer `L` that gives the maximum line length of any line in the output.

3. The third is an object of type `SlackFunctor`, which is an interface, declared as follows:

   ```
   public interface SlackFunctor { public double f(int slack); }
   ```

   The object must `implement SlackFunctor` and contain an implementation of a function `f` that we can apply to the line slacks to get our optimization metric. For example, if we want the goal to be based on the sum of the cubes of the slacks, then we would call `splitWords` as follows:

   ```
   public static void main(String[] args) { int[] lengths = {4, 2, 8, 4, 5,
   4, 5, 4, 3, 4}; int[] lastWords = PrettyPrint.splitWords(lengths, 20, new
   SlackFunctor() { public double f(int slack) { return Math.pow(slack,3); }
   }); System.out.println(Arrays.toString(lastWords)); }
   ```

   Your code can use that provided function by invoking `sf.f(slack)`, where `slack` is the slack of the line, and `sf` is a reference to a `SlackFunctor` object.

The output of this function should be a list of indices corresponding to the last word on each line, in order of the lines, in an optimal solution. So, if the optimal pretty-printing for the above sequence is determined to be

```
Call me Ishmael. Some years ago, never mind how long
```

then the list returned should be $2, 5, 9$ because the last word on the first line is the third word of the input sequence (its length was at index 2 in the input, because of zero-based indexing), the last word in the second line is the sixth word of the input sequence, and so on.

If a word in the input is longer than `L`, then the function should return `null`, as no pretty-printing with that margin is possible.

Starter Files

You can download the starter files from Moodle:

- `SlackFunctor.java` contains the interface definition for `SlackFunctor`
- `PrettyPrint.java` contains:
  - the declaration of the function you should implement; and
  - and an example `main` function, which obtains `L` from the command line, reads the input sequence of words from standard input (or a file), and uses the array returned from `splitWords` to print the output with the specified line breaks.

Submission

Upload your `PrettyPrint.java` source code file, containing the `splitWords` function described above, and any helper files, to Gradescope.

You do not need to upload `SlackFunctor.java`; it will be replaced with the original Moodle version before the tests are run.

You are free to change the `main` function of `PrettyPrint.java` however you want or add additional testing functions; Gradescope will make multiple calls to `splitWords` but will not call `main`.