Gold Digger

The Idea

You work for a mining company that has identified a huge mine of gold, the elements of gold, and rock. Your job is to write a program that identifies exactly five (no more no less) non-overlapping "zones" in the mine from which to extract the greatest riches you can. In short, gold chunks are great, gold elements are good, and rocks are a nuisance and drop the value of a zone. It's also much cheaper to excavate a smaller zone than a big zone, so some amount of gold in a small zone may be worth more in the end than the same amount of gold in a larger zone. The input describes the elements of gold and rock locations in the mine.

A short example mine looks like this (the gray numbers below indicate index):

```
lodlgoldlogdlogdlogggggdlog--lgdog---lglglg-----dl
01234567890123456789012345678901234567890123456789
```

Your program must output exactly 10 integers (five pairs), where each pair of integers represents the left and right indexes, respectively, of a zone you will excavate. The mine you are working with has exactly 100,000 items with indexes from 0 through 99,999.

When evaluating the value of a zone, a few things are considered as it is reviewed from left to right:

- The number of 'g', 'o', 'l', and 'd' elements are counted in the zone. Whichever has the smallest count indicates how many "gold" you could put together from those elements. We'll call this the "**ElementCount**". In the pink zone above, the ElementCount is 4 because 'd' shows up the least… 4 times.
- The number of "gold" and "dlog" (it's still gold whether you look from left to right or the other way around) are counted. We'll call this the "**GoldCount**". The whole mine above has 4 GoldCount, and the pink zone has 3. You might notice positions 6--10 have "dlog", but this does not add to the GoldCount because the 'd' was already a part of a "gold" already.
    - The GoldCount will benefit your score more because it counts as both gold and gold elements… 😀
- The number of rocks (indicated with a '-' hyphen), we'll call "**RockCount**". The pink zone has a RockCount of zero, but the mine has a RockCount of 10.

The Goal

Your score for this challenge is based on the sum of the values of each of the five zones your program prints to the console. Precisely, each individual zone value is determined as:

$$ZoneScalar \times ((4 \times GoldCount) + (4 \times ElementCount) - RockCount)$$

where **ZoneScalar** takes into account the size of the zone, **Size**, as:

$$ZoneScalar = 1 + \frac{\left(1200 \times e^{(-Size/20000)}\right)}{100}$$

Yeah… that ZoneScalar looks weird, but those mining companies have funky calculations for things. This is the part that considers the value of the gold, elements and rocks relative to the size of the zone. It's just how this is scored; you don't have to understand it. 😂

Input

Your program should expect input to come into your program as if from a user at the keyboard, you can safely assume no malformed input will be provided. The data will be a stream of exactly 100,000 characters consisting of 'g', 'o', 'l', 'd' and '-' (hyphen).

Output

For the following example excavation zones…

	Left Index	Right index
Zone 1	123	345
Zone 2	1000	2500
Zone 3	3749	20009
Zone 4	57632	72011
Zone 5	81823	97456

…the output for your program would be:

```
123 345 1000 2500 3749 20009 57632 72011 81823 97456
```

Your program will effectively be discarded for a number of reasons, but the score you receive may help you understand why.

- Score of -1:
  - part of your output was not an integer
  - not enough integers
  - or your code crashed
- Score of -2: one of your output indexes was outside the range of the mine [0--99,999]
- Score of -3: one of your left:right pairs was out of order

Evaluation

To evaluate your program, the judges will first compile your program (if your language requires compiling, that is). For the sake of example, let's assume you submit a source file called prog.cpp. The judges will perform the most basic compile and run your program through our judging script...

```
$ g++ -o prog prog.cpp
$ python3 judge.py mine.dat ./prog
```

If your program is in an interpreted language, such as Python, it would be run something similar to…

```
$ python3 judge.py mine.dat python3 prog.py
```

This will provide the test data to your program as if it were being entered by someone at the console. It will also capture the output of your program and provide it to the judging program as if it were being entered by someone at the terminal. For your convenience, the judging program's source and data files used for judging are provided.

Runtime

The judges have decided to limit student programs to no more than one minute of runtime.