Detecting Contractor Fraud

01h : 14m : 16s
to test end

Connection attempt failed.

Exercise Overview (from previous page)

Languages Allowed: C, C++, C++14, C#, Java 7, Java 8, JavaScript (Node.js), Python 2, Python 3

The objective of this exercise is to code for correctness (defined as passing the tests) – we will not be reading or grading code for style. Focus your allotted time on ensuring your solution's correctness.

- Focus on implementing some cases correctly, even if you are unable to implement everything within the time limit. You do not need to pass all of the tests to pass the challenge.
- You **are** allowed to use the Internet or other resources available to you to look up algorithms and language features. You **are not** allowed look up solutions or copy/paste any code you may find; all solutions must be your own.
- You are strongly encouraged to develop and test your code inside HackerRank. Otherwise, you should set aside a portion of the provided time to get your code running inside the HackerRank environment against the provided test cases. Make sure to test your code often in HackerRank using the "Run Code" button.
- Your final score is determined by your **last submitted** solution, it is not the highest score obtained over the course of the test.

Problem Statement

Introduction

Background

A company that relies heavily on contractors suspects that some of those contractors are under-reporting the time taken to complete a job. They caught one manager bribing a contractor to falsify when they started a job in order to increase the efficiency of their department. The company needs help automatically identifying similar cases of fraud.

Billing & Reporting Process

The company has separate systems for job tracking and billing:

- Contractors indicate when they start working on a job by submitting a job start notice to the job tracking system.
- During the job, they enter a single invoice into the billing system. The billing system then gives back a unique, increasing numeric invoice ID.
- Upon completion, contractors submit their invoice IDs to job tracking, which records that the jobs were completed. A contractor must complete and submit all open jobs together.

Objective

fraudul Connection attempt failed.

Input

You will receive a string array of at most 300,000 strings, where each string corresponds to one event. These events are already sorted in terms of time, and take one of two forms:

1. Job start events
   - Take the form `<CONTRACTOR_NAME>;START`
   - The `CONTRACTOR_NAME` is a unique identifier for the contractor, and is guaranteed not to contain a semicolon.
2. Job completion events
   - Take the form `<CONTRACTOR_NAME>;<INVOICE_ID>(,<INVOICE_ID>)*`
   - The `CONTRACTOR_NAME` is the same unique identifier for the contractor as before.
   - `INVOICE_ID` 's are integer values, guaranteed to fit within the value of a signed 64-bit long.
   - If a contractor has multiple jobs started, then they will complete and submit invoice IDs for all started jobs as a single job completion event. These invoice IDs will be comma-delimited, one invoice ID per job start event. These are referred to as "batch job completions".
     - For example, if a contractor has started three jobs, then the next job completion from that contractor is guaranteed to consist of three distinct invoice IDs.
     - We do not know which invoice number corresponds to which job start event.

You may assume that the input will always be well-formed and contain no extra characters or whitespace.

Note that we will represent input and output string arrays as lines of text, where each line refers to one string in the array (in order). For example,

```
Jeremy
Tom
Nick
Leah
```

means the string array [Jeremy, Tom, Nick, Leah] (in that order).

Detecting Fraud

There are two kinds of fraudulent submissions you need to detect: shortened jobs and suspicious batches. It is recommended that you first work on detecting shortened jobs before tackling suspicious batches.

Shortened Jobs

For exa Connection attempt failed.

```
David;START
David;24
Evil;START
Evil;18
```

According to the log, first David starts a job, gets an invoice ID of 24, and then submits a job completion with invoice ID 24. Evil later does the same, with a invoice ID of 18.
However, Evil could not have started his job after David finished his, because Evil's smaller invoice ID implies he must have gotten his invoice ID before David, and invoice IDs cannot be gotten before job start.
Therefore, Evil's job start event on the third line is a shortened job.

Suspicious Batches

These are batch job completions which, given their invoice IDs and their associated start events, must contain at least one shortened job.
If it is clear that a given start job event within a batch must have been reported late (the job was shortened), then it is **both** part of a suspicious batch and is itself a shortened job.
For example, consider this event log:

```
Leah;START
Leah;10
Alice;START
Alice;START
Alice;8,14
```

According to the log, first Leah started and completed a job with an invoice ID of 10.
Then Alice starts 2 jobs, gets invoice IDs 8 and 14 for the two jobs, and finally submits a batch job completion with the invoice IDs.
Except Alice's must have gotten an invoice for one of her jobs before Leah, otherwise Alice couldn't submit to an invoice ID of 8 to job tracking.
Therefore, Alice's batch job completion on the fifth line is a suspicious batch, because at least one of the start events should have been before Leah's job completion.
We can't identify which one of Alice's two job start events was the one that was shortened, so we cannot mark either of them as shortened jobs. If both IDs were below 10, then we would also mark both start events as shortened jobs.

Note that, when checking a given job completion event and its associated job starts for violations, you may assume previous events were submitted correctly, even if those previous

Your alg Connection attempt failed. ystems which indicate

For each suspicious behavior identified, the string array you return should contain a line of the form `<LINE_NUMBER>;<CONTRACTOR_NAME>;<VIOLATION_TYPE>`.

- `LINE_NUMBER` is a (one-indexed) line number from the input.
  - For `SUSPICIOUS_BATCH`, this is the line on which the batch job completion occurred.
  - For `SHORTENED_JOB`, this is the line on which the job start occurred.
- `CONTRACTOR_NAME` is the unique identifier of the offending the contractor.
- `VIOLATION_TYPE` is a string that matches either `SHORTENED_JOB` or `SUSPICIOUS_BATCH`. This indicates which pattern we have identified, following the rules above.

You may present violations in any order.

Tests

The tests are split into three sections, labeled as Easy, Medium and Hard tests.

- Tests 1-11 verify that your solution is able to identify shortened jobs correctly and scaleably.
- Tests 12-15 verify that your solution handles batch submissions and can correctly identify suspicious batches. Identifying shortened jobs within shortened batches is not required.
- Tests 16-22 verify that your solution can correctly identify shortened jobs within suspicious batches, handle complex interactions between fraudulent submissions correctly, and scales.

You do not need to pass all of the tests to pass the challenge.

Sample Test Explanations

There are seven test cases which you will be able to see the input and your output for.

Explanations three of them are given below:

Testcase 1:

Input

```
Alice;START
Bob;START
Bob;1
Carson;START
Alice;15
Carson;6
David;START
David;24
Evil;START
Evil;24
Evil;START
```

Output

Connection attempt failed.

```
11;Evil;SHORTENED_JOB
```

Explanation

- "Bob;1" is certainly valid.
- "Alice;15" is valid.
  - It does not matter here that Alice started first; she started before Bob finished.
- "Carson;6" is also valid.
  - Carson started his job after Bob finished, but his invoice number wasn't smaller than Bob's.
  - Carson did *not* start his job after Alice finished, so we can't identify this as a SHORTENED_JOB on the basis of that.
- "David;24" is valid.
- "Evil;24" is valid.
  - This seems unusual, but is permitted (e.g. if they worked together on the job).
- "Evil;18" is not valid.
  - Evil started its job after David's job which had invoice 24 finished; this is a SHORTENED_JOB.
  - This should be logged as "11;Evil;SHORTENED_JOB", because Evil STARTed this job on line 11 and only submitted one job.
- Fiona had not finished her job (or at least, had not reported it to job tracking) at the time the data-source was read, but that's perfectly legitimate.

Testcase 2:

Input

```
Tom;START
Jeremy;START
Dana;START
Jeremy;4
Dana;2
James;START
Leah;START
James;5
Nick;START
Tom;1
Nick;6
Leah;3
```

Output

Connection attempt failed.

Explana

- "Le
  - When Leah started her job, Jeremy and Dana had finished their jobs, with the highest invoice number being 4. This is higher than the invoice Leah ultimately submitted, so she must have shortened her job.
- All other jobs were valid.

Testcase 12:

Input

```
Nick;START
Jeremy;START
Leah;START
Nick;10
Jeremy;START
Jeremy;START
Leah;15
Jeremy;8,14,9
```

Output

```
8;Jeremy;SUSPICIOUS_BATCH
```

Explanation

- "Nick;10" and "Leah;15" are valid, because no one started any jobs before Nick or Leah started.
- "Jeremy;8,14,9" is not valid.
  - This is a suspicious batch, because Jeremy submitted two jobs after Nick finished the job with invoice number 10. One of these jobs must have had an invoice number of either 8 or 9 (definitely less than 10), and must thus have been shortened.
  - However, because each job *might* have had the valid invoice number of 14, none of the actual jobs can uniquely be identified as shortened.

Running Your Own Tests

If you want to run your own tests in the HackerRank interface, note that you will need to enter your string array line by line, in the format mentioned above.
You will also need to prepend one line indicating the number of lines in your string array.

For example, a simple test could look like this:

```
2
John;START
John;1
```

Detecting Contractor Fraud

01h : 14m : 16s
to test end

silently

Connection attempt failed.

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour. Start tour

Original code | Python 3

```
1  #!/bin/python3
2
3  import sys
4  import os
5
6
7  # Complete the function below.
8
9  def findViolations(datafeed):
10
11
12
13 ▸ if __name__ == "__main__":↔
35
```

Line: 8 Col: 1

Test against custom input

Run Code | Submit code & Continue

(You can submit any number of times)

Download sample test cases *The input/output files have Unix line endings. Do not use Notepad to edit them on windows.*

Detecting Contractor Fraud

01h : 14m : 16s
to test end

Connection attempt failed.