https://www.1point3acres.com/bbs/thread-1103536-1-1.html*
https://www.1point3acres.com/bbs/thread-1032751-1-1.html

https://www.1point3acres.com/bbs/thread-1090103-1-1.html*

https://www.1point3acres.com/bbs/thread-1103030-1-1.html

https://www.1point3acres.com/bbs/thread-1037903-1-1.html
https://www.1point3acres.com/bbs/thread-844359-1-1.html
https://www.1point3acres.com/bbs/thread-831181-1-1.html
https://www.1point3acres.com/bbs/thread-800972-1-1.html
https://www.1point3acres.com/bbs/thread-1103536-1-1.html
https://www.1point3acres.com/bbs/thread-1103030-1-1.html

Pre-onsite: ml integration 和一轮coding。onsite是三轮，coding，ml sys design和ml bug squash
ml integration
hackerrank，进去是个**jupyter notebook**。要求share screen，可以查资料但是不能用llm。
跟地里描述的一样，给一个df，要求做二分类。由于提前准备过所以做的还行，但是前面clarification就花了快半个小时，导致后面时间紧张。你需要的技能：两列的组合是否都是unique（customer_id x merchant_id), 补全missing value，对于sparse feature和numerical feature分别处理。sparse 需要转成one hot。最后我抄地里答案用的random forest，acc 90%+，F1 也90%。简单说说效果。会有一些简单的follow up question (最好还是熟悉你用的model的原理，我当时就因为不太懂randomforest轻微翻车但是没想到也过了）
coding:
Q1 运货，不同国家不同运费，不同货物不同运费，计算运费
Q2: 价格出现梯度，比如0-2个是一个价钱，3个以上是另一个价钱。 不同国家，不同货物，都有相应的价格
Q3: 价格梯度出现fixed/incremental，fixed就是在这个quantity区间内，都是一个价钱。
在hackerrank上，没有要求share screen。可以运行，要求跑test case，结束还要聊聊你会怎么test。
一上来就说不要求time complexity，所以我全都为了快点写完用的brute force
感受就是其实很简单，细心一点不会有问题。我大概35分钟就做完了三个题，最后纯聊天。要对json非常熟悉，比如在python里如何遍历一个dict/list/json。Input是json格式，但是不是file，是dict
VO

1. Coding: 给用户发email，welcome，coming expiry，expired。第二问是 如果有plan update，发邮件 update plan，但是用户之后的所有邮件都要对应的改plan。第三问没做到
2. ML bug squash:
    1. For loop range (`for i in range (num_features-1)` to `for i in range (num_features)`
    2. Model.predict -> model.predict_proba
    3. roc_auc calculation wrong, metrics += xxxx wrong. Should be the small triangle area
3. ML sys design: fraud detection. 有一点ml sys design 和传统sys design结合。

ML Design: Account Takeover

You work for a company that runs a payments API as a web service, and bad actors often try to defraud your service. One common fraud attack vector is when previously “good” merchant accounts are taken

over by bad actors, which we call "account takeovers". Design a system to identify and respond to these account takeovers.
Bad actors might have taken over the account by guessing a weak password or using **passwords found in other leaks on the web**. The bad actor then transfers funds from the merchant's account to their own bank. It's hard to identify these occurrences because the accounts have a history of good, non-suspicious behavior. It's possible to catch these account takeovers because they relate to a change in behavior, e.g. changes in **details (e.g. login IP) in how the** account is being accessed. We need to catch these occurrences as quickly as possible when they occur, because in a matter of minutes or hours a bad actor can steal a lot of money.

Model Design (~20 min)

How would you build a model to make predictions for this system?

- Consider what data sources to use, what models to try, how to prepare your data for those model(s), how to collect and use labeled data
- What problems might you encounter along the way?
- Discuss tradeoffs and assumptions throughout
- Call out any decisions that might require more investigation

Implementation at Scale (~15 min)

How would you build a system to make these predictions in production?

- What system components would you need? How will data flow through them?
- How to make the system scalable, and what challenges would you expect?
- How to make the system reliable, and what challenges would you expect?
- Discuss tradeoffs and assumptions throughout

Business Impact (~10 min)

How would you measure the success of this system?

- Given a model that works offline, how should the rollout be designed to mitigate risk of unexpected breakage online?
- Given a model that's been rolled out, how can we evaluate its impact on the business?

Catch me if you can	
Card Metadata	
Intent	
店面	
server	
Http header	

合并交易记录	
Brazil	
运费	https://www.1point3acres.com/bbs/thread-1090103-1-1.html
运费2	
Coding 转账	

Brazil

https://www.1point3acres.com/bbs/thread-1089011-1-1.html
https://www.1point3acres.com/bbs/thread-1089011-2-1.html

Coding 转账

https://www.1point3acres.com/bbs/thread-1037903-1-1.html
https://www.1point3acres.com/bbs/thread-767545-1-1.html

运费

https://www.1point3acres.com/bbs/thread-1090103-1-1.html

运费2

https://www.1point3acres.com/bbs/thread-1086731-1-1.html
https://www.1point3acres.com/bbs/thread-1096463-1-1.html

HTTP header
https://www.1point3acres.com/bbs/thread-709534-1-1.html

In an HTTP request, the Accept-Language header describes the list of languages that the requester would like content to be returned in. The header takes the form of a comma-separated list of language tags. For example:

Accept-Language: en-US, fr-CA, fr-FR

means that the reader would accept:

1. English as spoken in the United States (most preferred)
2. French as spoken in Canada
3. French as spoken in France (least preferred)

We're writing a server that needs to return content in an acceptable language for the requester, and we want to make use of this header. Our server doesn't support every possible language that might be requested (yet!), but there is a set of languages that we do support. Write a function that receives two arguments: an Accept-Language header value as a string and a set of supported languages, and returns the list of language tags that will work for the request. The language tags should be returned in descending order of preference (the same order as they appeared in the header).
In addition to writing this function, you should use tests to demonstrate that it's correct, either via an existing testing system or one you create.
Examples:

```
parse_accept_language(
  "en-US, fr-CA, fr-FR",  # the client's Accept-Language header, a string
  ["fr-FR", "en-US"]      # the server's supported languages, a set of strings
)
```

returns: ["en-US", "fr-FR"]

```
parse_accept_language("fr-CA, fr-FR", ["en-US", "fr-FR"])
```

returns: ["fr-FR"]

```
parse_accept_language("en-US", ["en-US", "fr-CA"])
```

returns: ["en-US"]

Part 2

Accept-Language headers will often also include a language tag that is not region-specific - for example, a tag of "en" means "any variant of English". Extend your function to support these language tags by letting them match all specific variants of the language.
Examples:

```
parse_accept_language("en", ["en-US", "fr-CA", "fr-FR"])
```

returns: ["en-US"]

```
parse_accept_language("fr", ["en-US", "fr-CA", "fr-FR"])
```

returns: ["fr-CA", "fr-FR"]

```
parse_accept_language("fr-FR, fr", ["en-US", "fr-CA", "fr-FR"])
```

returns: ["fr-FR", "fr-CA"]

Part 3

Accept-Language headers will sometimes include a "wildcard" entry, represented by an asterisk, which means "all other languages". Extend your function to support the wildcard entry.
Examples:

```
parse_accept_language("en-US, *", ["en-US", "fr-CA", "fr-FR"])
```

returns: ["en-US", "fr-CA", "fr-FR"]

```
parse_accept_language("fr-FR, fr, *", ["en-US", "fr-CA", "fr-FR"])
```

returns: ["fr-FR", "fr-CA", "en-US"]

Part 4

Accept-Language headers will sometimes include explicit numeric weights (known as q-factors) for their entries, which are used to designate certain language tags

as specifically undesired. For example:
Accept-Language: fr-FR;q=1, fr;q=0.5, fr-CA;q=0
This means that the reader most prefers French as spoken in France, will take any variant of French after that, but specifically wants French as spoken in Canada only as a last resort. Extend your function to parse and respect q-factors.
Examples:
parse_accept_language("fr-FR;q=1, fr-CA;q=0, fr;q=0.5", ["fr-FR", "fr-CA", "fr-BG"])
returns: ["fr-FR", "fr-BG", "fr-CA"]
parse_accept_language("fr-FR;q=1, fr-CA;q=0, *;q=0.5", ["fr-FR", "fr-CA", "fr-BG", "en-US"])
returns: ["fr-FR", "fr-BG", "en-US", "fr-CA"]
parse_accept_language("fr-FR;q=1, fr-CA;q=0.8, *;q=0.5", ["fr-FR", "fr-CA", "fr-BG", "en-US"])
最后一小问我的想法是记录每个supported tag的q factor，比如一开始大家的q factor都是-1。然后根据用户输入，有*的话先处理*，接下来处理单独的language（比如fr, en）如果有的话，最后处理完整的tag（比如fr-FR, en-US），每次都更新对应的supported tag的q factor，最后选出所有q factor不是-1的supported tag然后根据q factor排序输出。这道题现场没写完，只说了个大概思路。

Card Metadata

1. Card Metadata

Card Range Obfuscation

Payment card numbers are composed of eight to nineteen digits, the leading six of which are referred to as the bank identification number (BIN). Stripe's card metadata API exposes information about different card brands within each BIN range, by returning a mapping of card intervals to brands.

However, a given BIN range may have gaps at the beginning, middle, or end of the range where no valid cards are present. This information can be used by fraudsters to test for valid credit cards with a high degree of probability. To prevent fraudsters from abusing the data gaps, we must fill in missing values by extending the returned intervals to cover the entire range.

A *BIN range* refers to all the 16-digit card numbers starting with a given BIN: for example, a BIN of 424242 has a range of 4242420000000000 to 4242429999999999, inclusive.

An *interval* is a subset of a BIN range, also inclusive.

In this problem, we'll be taking as input a 6-digit BIN and a list of 10-digit intervals within the BIN range corresponding to brands. We'll return a list of sorted intervals covering the entire BIN range (i.e. for a BIN of 424242, covering 4242420000000000 to 4242429999999999, inclusive).

Input format

Your input will consist of one line containing a six-digit BIN, one line containing a positive integer N, and N lines containing intervals and their corresponding brands.

The format of each interval is `start,end,brand`, where `start` and `end` are the 10 digits following the input BIN and `brand` is an alphanumeric string.

Example:

```
777777
2
```

```
2
1000000000,3999999999,VISA
4000000000,5999999999,MASTERCARD
```

Output format

Your output will consist of one or more lines containing the resulting intervals and their corresponding brands. The output intervals should be sorted by start.

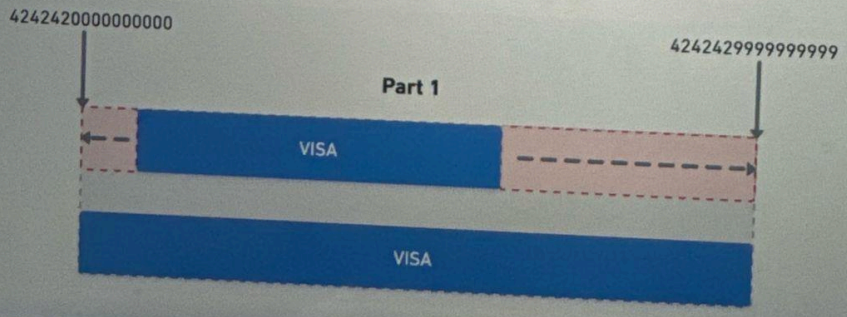
```
7777770000000000,7777773999999999,VISA
7777774000000000,7777779999999999,MASTERCARD
```

Formatting Note

For most languages, Hackerrank will provide you with a code stub and a function to implement.

Part 1

For the first set of testcases, you'll be given a set of non-overlapping intervals with **gaps on the lower end and/or higher end** of the BIN range. You're expected to extend the lowest interval to the lower boundary of the BIN range, and the highest interval to the higher boundary of the BIN range. This will be sufficient to solve the **first 4 test cases**.



Example 1: the VISA interval was extended on the lower and higher ends to cover the BIN range.

Input:

```
424242
1
5000000000,6555555555,VISA
```

Output:

```
4242420000000000,4242429999999999,VISA
```

Example 2: the VISA interval was extended on the lower end and the Mastercard interval was extended on the higher end.

Input:

```
424242
2
4000000000,8999999999,MASTERCARD
1000000000,3999999999,VISA
```

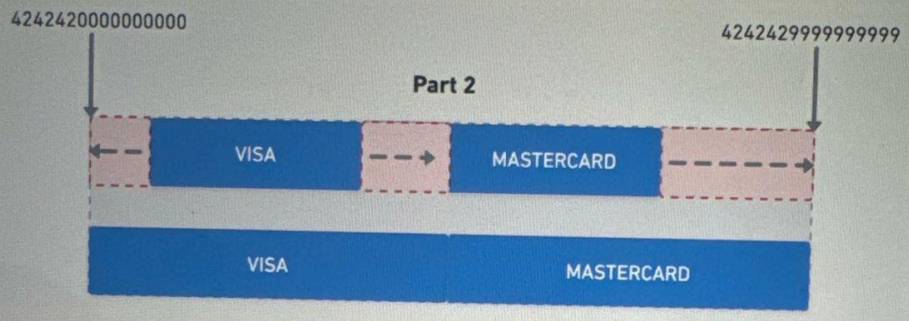
Output:

```
4242420000000000,4242423999999999,VISA
4242424000000000,4242429999999999,MASTERCARD
```

Part 2

Part 2

For the second set of testcases, the set of non-overlapping intervals can also contain **gaps in between known intervals**. In these cases, the interval on the lower end of the gap will be extended to fill the gap. This will be sufficient to solve the **next 4 test cases**.



Example 1: the VISA interval was extended on the higher end to fill the gap, up to the Mastercard interval.

Input:

```
424242
2
0000000000,3700000000,VISA
6100000000,9999999999,MASTERCARD
```

Output:

```
4242420000000000,4242426099999999,VISA
4242426100000000,4242429999999999,MASTERCARD
```

Example 2: the VISA interval is extended on the higher end to fill the gap up to the Mastercard interval. The Mastercard interval is extended on the higher end to fill the gap up to the Amex interval. The VISA interval is extended on the lower end following the requirements in Part 1.

Input:

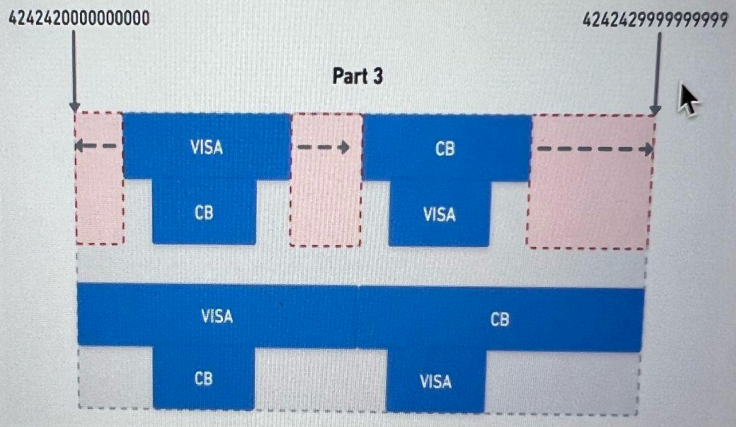
21m left

ALL

1

Part 3

For the third set of testcases, the input may contain intervals that are proper subsets of other intervals. When extending an interval to cover a gap, you should only extend the covering interval. This will be sufficient to solve the **next 2 testcases**.



Example 1: the first Visa interval and second CB interval are extended.

Input:

```
424242
4
1000000000,5000000000,VISA
2000000000,4000000000,CB
6000000000,9000000000,CB
7000000000,8000000000,VISA
```

Output:

Example 1: the first Visa interval and second CB interval are extended.

Input:

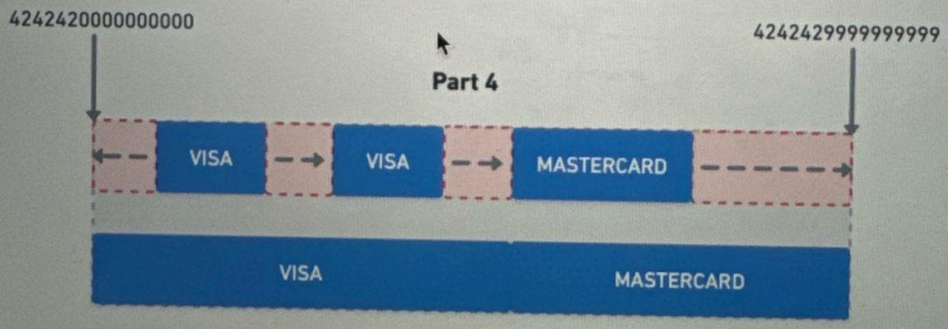
```
424242
4
1000000000,5000000000,VISA
2000000000,4000000000,CB
6000000000,9000000000,CB
7000000000,8000000000,VISA
```

Output:

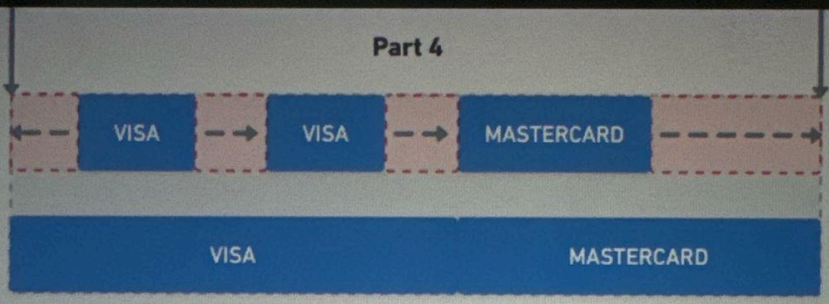
```
4242420000000000,4242425999999999,VISA
4242422000000000,4242424000000000,CB
4242426000000000,4242429999999999,CB
4242427000000000,4242428000000000,VISA
```

Part 4

For the fourth set of testcases, adjacent intervals (i.e. intervals with no gaps in between) with the same brand after extending should be merged into one interval in the output. This will be sufficient to solve the **next 3 testcases**.



Example 1: the two adjacent VISA intervals were merged.



Example 1: the two adjacent VISA intervals were merged.

Input:

```
424242
2
0000000000,5999999999,VISA
6000000000,9999999999,VISA
```

Output:

```
4242420000000000,4242429999999999,VISA
```

Example 2: the VISA intervals were extended to fill the gaps, then merged.

Input:

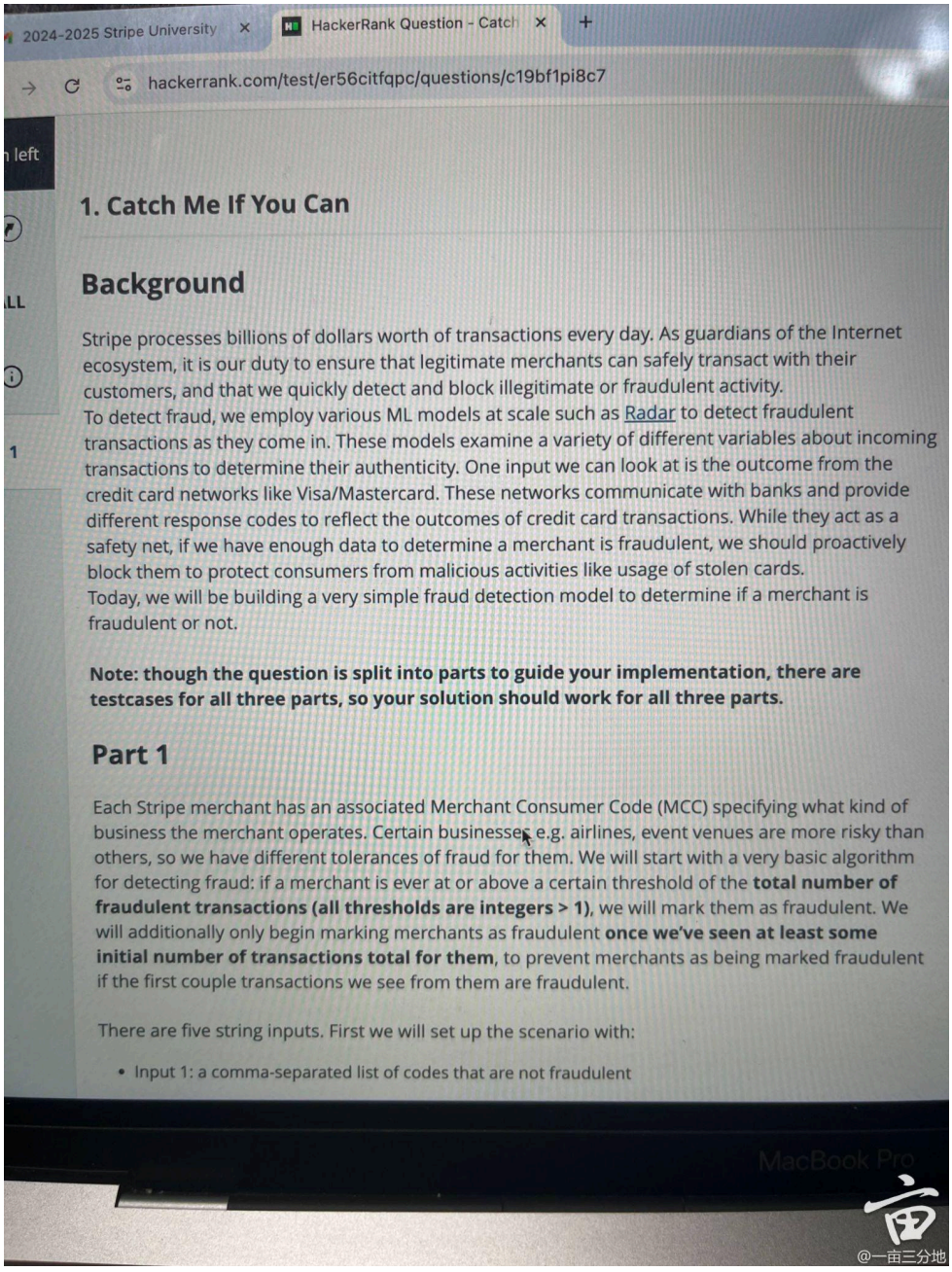
```
424242
3
1000000000,3999999999,VISA
5000000000,6999999999,VISA
8000000000,9999999999,MASTERCARD
```

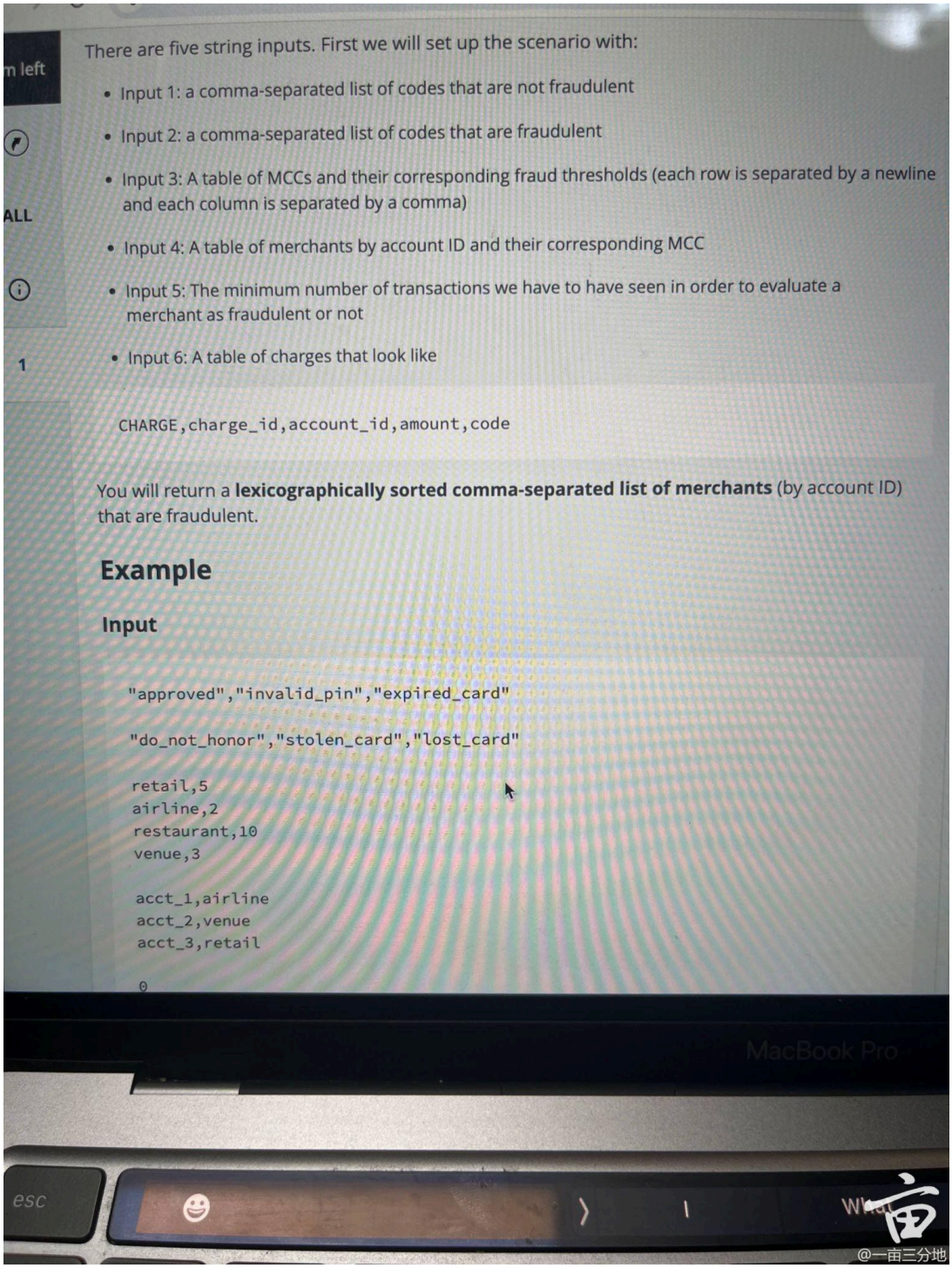
Output:

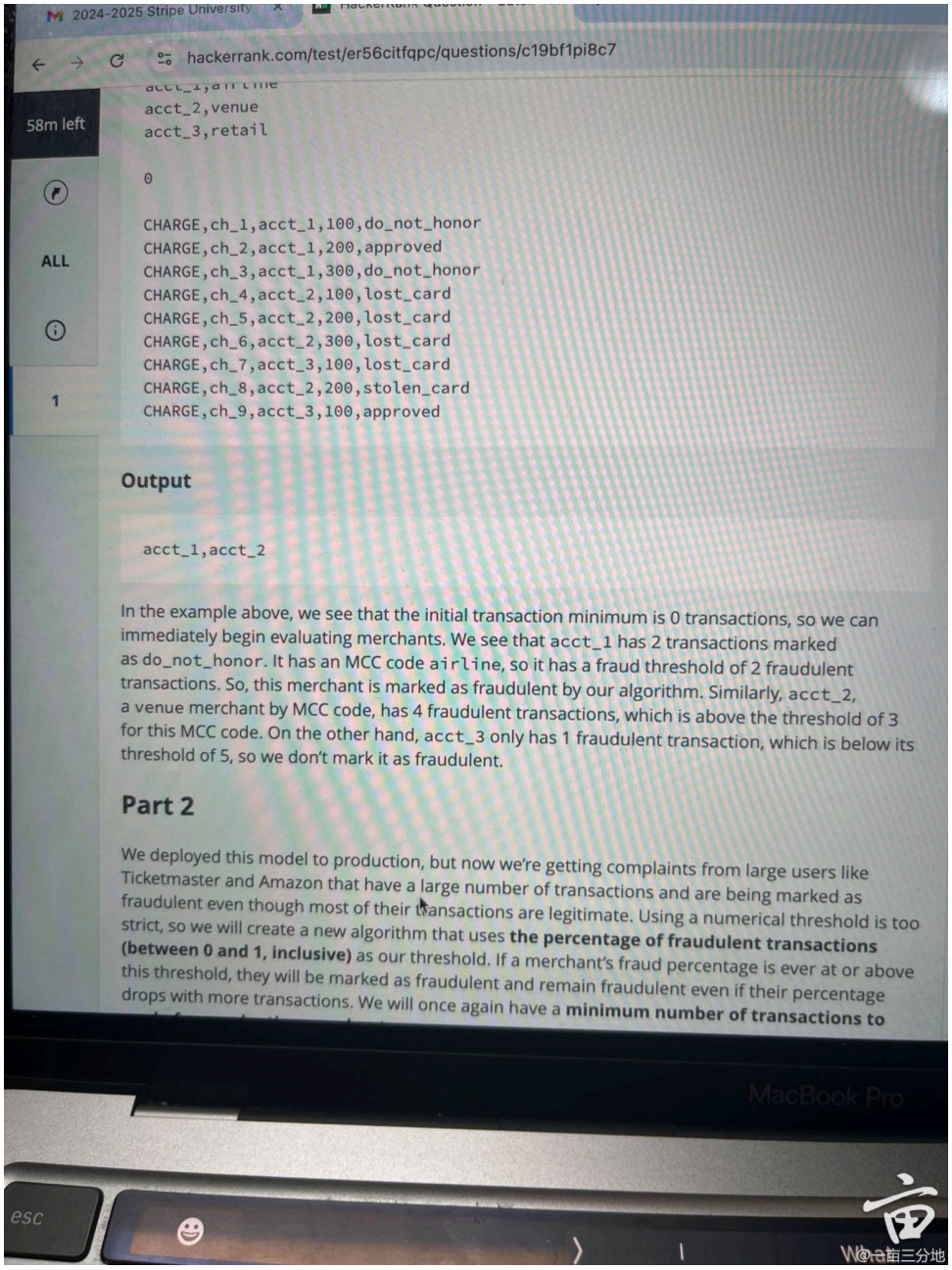
```
4242420000000000,4242427999999999,VISA
4242428000000000,4242429999999999,MASTERCARD
```

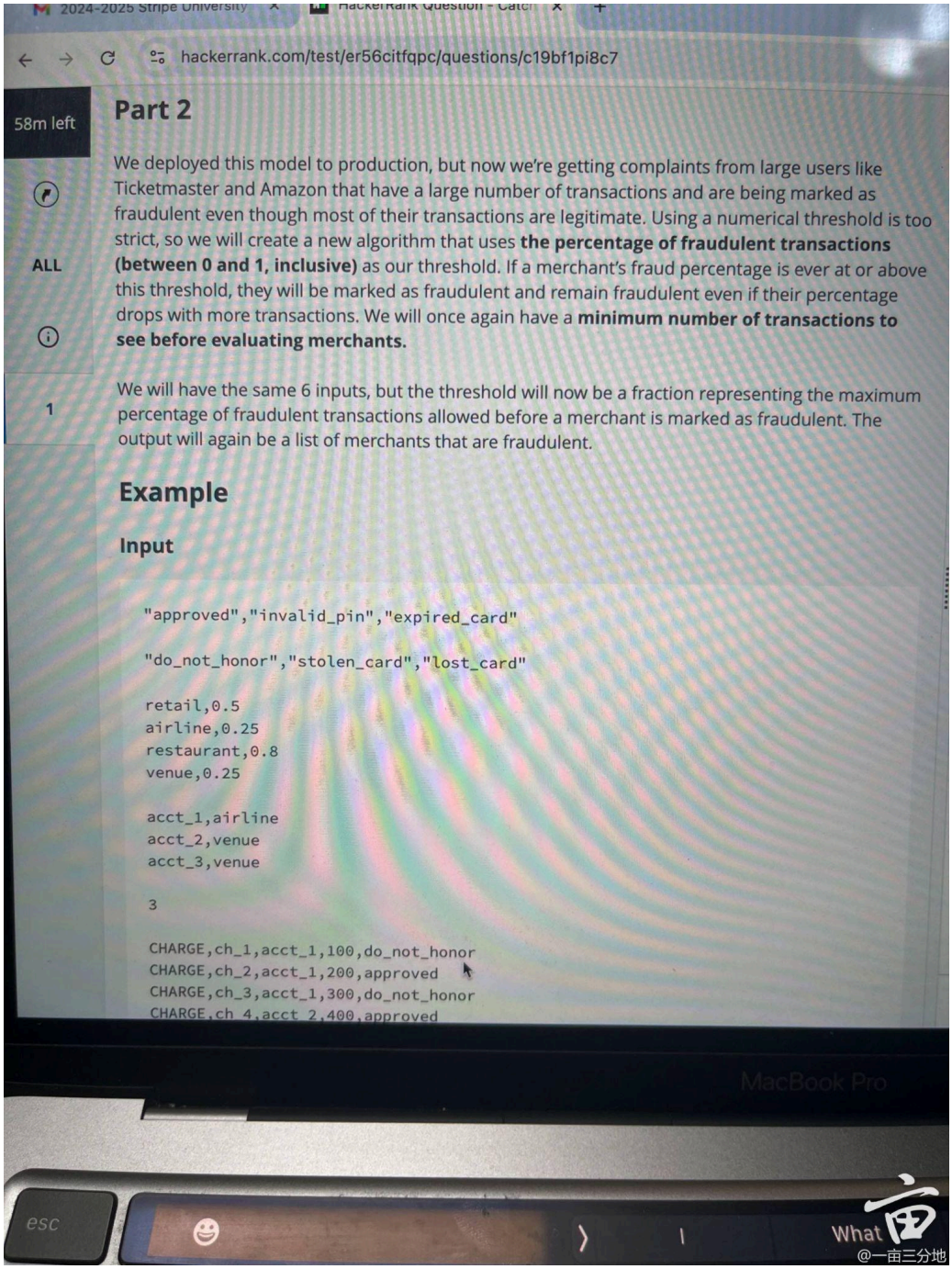
Catch me if you can

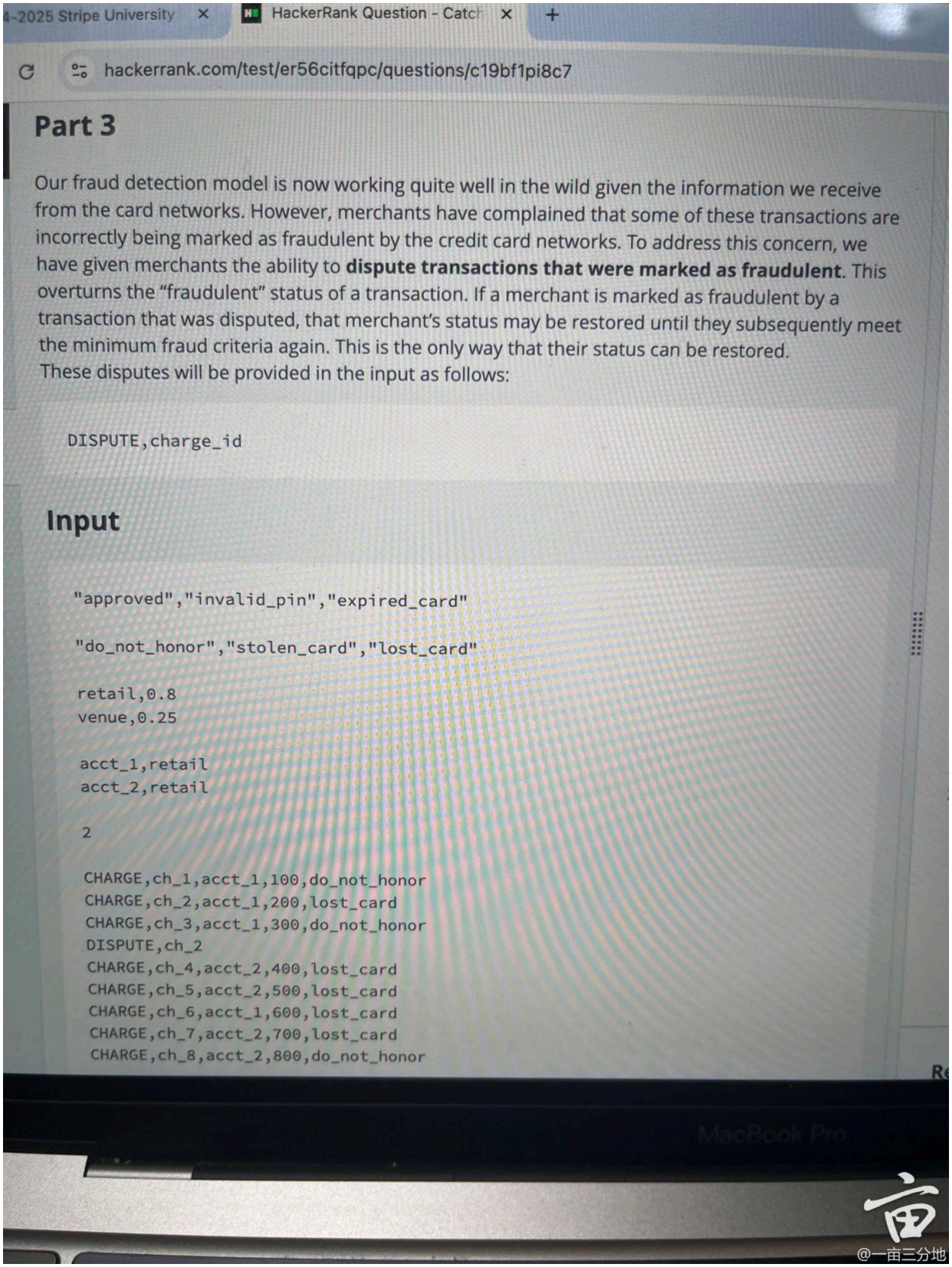
https://www.1point3acres.com/bbs/thread-1086447-1-1.html


2024-2025 Stripe University
HackerRank Question - Catch
hackerrank.com/test/er56citfqpc/questions/c19bf1pi8c7
1. Catch Me If You Can
Background
Stripe processes billions of dollars worth of transactions every day. As guardians of the Internet ecosystem, it is our duty to ensure that legitimate merchants can safely transact with their customers, and that we quickly detect and block illegitimate or fraudulent activity.
To detect fraud, we employ various ML models at scale such as Radar to detect fraudulent transactions as they come in. These models examine a variety of different variables about incoming transactions to determine their authenticity. One input we can look at is the outcome from the credit card networks like Visa/Mastercard. These networks communicate with banks and provide different response codes to reflect the outcomes of credit card transactions. While they act as a safety net, if we have enough data to determine a merchant is fraudulent, we should proactively block them to protect consumers from malicious activities like usage of stolen cards.
Today, we will be building a very simple fraud detection model to determine if a merchant is fraudulent or not.
Note: though the question is split into parts to guide your implementation, there are testcases for all three parts, so your solution should work for all three parts.
Part 1
Each Stripe merchant has an associated Merchant Consumer Code (MCC) specifying what kind of business the merchant operates. Certain businesses e.g. airlines, event venues are more risky than others, so we have different tolerances of fraud for them. We will start with a very basic algorithm for detecting fraud: if a merchant is ever at or above a certain threshold of the total number of fraudulent transactions (all thresholds are integers > 1), we will mark them as fraudulent. We will additionally only begin marking merchants as fraudulent once we've seen at least some initial number of transactions total for them, to prevent merchants as being marked fraudulent if the first couple transactions we see from them are fraudulent.
There are five string inputs. First we will set up the scenario with:
Input 1: a comma-separated list of codes that are not fraudulent
MacBook Pro
@一亩三分地


m left
ALL
1
There are five string inputs. First we will set up the scenario with:
• Input 1: a comma-separated list of codes that are not fraudulent
• Input 2: a comma-separated list of codes that are fraudulent
• Input 3: A table of MCCs and their corresponding fraud thresholds (each row is separated by a newline and each column is separated by a comma)
• Input 4: A table of merchants by account ID and their corresponding MCC
• Input 5: The minimum number of transactions we have to have seen in order to evaluate a merchant as fraudulent or not
• Input 6: A table of charges that look like
CHARGE,charge_id,account_id,amount,code
You will return a lexicographically sorted comma-separated list of merchants (by account ID) that are fraudulent.
Example
Input
"approved","invalid_pin","expired_card"
"do_not_honor","stolen_card","lost_card"
retail,5
airline,2
restaurant,10
venue,3
acct_1,airline
acct_2,venue
acct_3,retail
0
MacBook Pro
esc

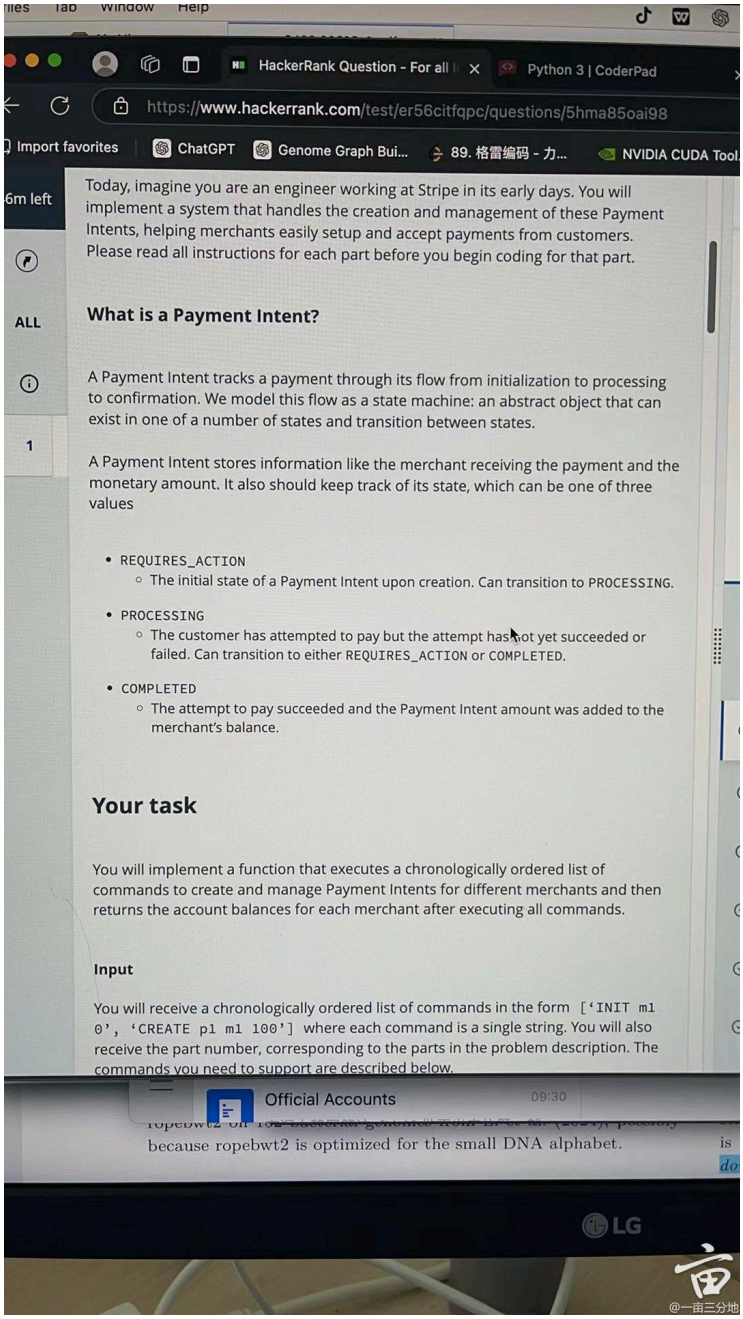

2024-2025 Stripe University
hackerrank.com/test/er56citfqpc/questions/c19bf1pi8c7
58m left
ALL
1
acct_2,venue
acct_3,retail
0
CHARGE,ch_1,acct_1,100,do_not_honor
CHARGE,ch_2,acct_1,200,approved
CHARGE,ch_3,acct_1,300,do_not_honor
CHARGE,ch_4,acct_2,100,lost_card
CHARGE,ch_5,acct_2,200,lost_card
CHARGE,ch_6,acct_2,300,lost_card
CHARGE,ch_7,acct_3,100,lost_card
CHARGE,ch_8,acct_2,200,stolen_card
CHARGE,ch_9,acct_3,100,approved
Output
acct_1,acct_2
In the example above, we see that the initial transaction minimum is 0 transactions, so we can immediately begin evaluating merchants. We see that acct_1 has 2 transactions marked as do_not_honor. It has an MCC code airline, so it has a fraud threshold of 2 fraudulent transactions. So, this merchant is marked as fraudulent by our algorithm. Similarly, acct_2, a venue merchant by MCC code, has 4 fraudulent transactions, which is above the threshold of 3 for this MCC code. On the other hand, acct_3 only has 1 fraudulent transaction, which is below its threshold of 5, so we don't mark it as fraudulent.
Part 2
We deployed this model to production, but now we're getting complaints from large users like Ticketmaster and Amazon that have a large number of transactions and are being marked as fraudulent even though most of their transactions are legitimate. Using a numerical threshold is too strict, so we will create a new algorithm that uses the percentage of fraudulent transactions (between 0 and 1, inclusive) as our threshold. If a merchant's fraud percentage is ever at or above this threshold, they will be marked as fraudulent and remain fraudulent even if their percentage drops with more transactions. We will once again have a minimum number of transactions to
MacBook Pro
esc
@宙三分地


2024-2025 Stripe University
hackerrank.com/test/er56citfqpc/questions/c19bf1pi8c7
58m left
ALL
1
Part 2
We deployed this model to production, but now we're getting complaints from large users like Ticketmaster and Amazon that have a large number of transactions and are being marked as fraudulent even though most of their transactions are legitimate. Using a numerical threshold is too strict, so we will create a new algorithm that uses the percentage of fraudulent transactions (between 0 and 1, inclusive) as our threshold. If a merchant's fraud percentage is ever at or above this threshold, they will be marked as fraudulent and remain fraudulent even if their percentage drops with more transactions. We will once again have a minimum number of transactions to see before evaluating merchants.
We will have the same 6 inputs, but the threshold will now be a fraction representing the maximum percentage of fraudulent transactions allowed before a merchant is marked as fraudulent. The output will again be a list of merchants that are fraudulent.
Example
Input
"approved","invalid_pin","expired_card"
"do_not_honor","stolen_card","lost_card"
retail,0.5
airline,0.25
restaurant,0.8
venue,0.25
acct_1,airline
acct_2,venue
acct_3,venue
3
CHARGE,ch_1,acct_1,100,do_not_honor
CHARGE,ch_2,acct_1,200,approved
CHARGE,ch_3,acct_1,300,do_not_honor
CHARGE,ch_4,acct_2,400,approved
MacBook Pro
esc
What


4-2025 Stripe University
HackerRank Question - Catch
hackerrank.com/test/er56citfqpc/questions/c19bf1pi8c7
Part 3
Our fraud detection model is now working quite well in the wild given the information we receive from the card networks. However, merchants have complained that some of these transactions are incorrectly being marked as fraudulent by the credit card networks. To address this concern, we have given merchants the ability to dispute transactions that were marked as fraudulent. This overturns the "fraudulent" status of a transaction. If a merchant is marked as fraudulent by a transaction that was disputed, that merchant's status may be restored until they subsequently meet the minimum fraud criteria again. This is the only way that their status can be restored.
These disputes will be provided in the input as follows:
DISPUTE,charge_id
Input
"approved","invalid_pin","expired_card"
"do_not_honor","stolen_card","lost_card"
retail,0.8
venue,0.25
acct_1,retail
acct_2,retail
2
CHARGE,ch_1,acct_1,100,do_not_honor
CHARGE,ch_2,acct_1,200,lost_card
CHARGE,ch_3,acct_1,300,do_not_honor
DISPUTE,ch_2
CHARGE,ch_4,acct_2,400,lost_card
CHARGE,ch_5,acct_2,500,lost_card
CHARGE,ch_6,acct_1,600,lost_card
CHARGE,ch_7,acct_2,700,lost_card
CHARGE,ch_8,acct_2,800,do_not_honor
MacBook Pro
@一亩三分地

Intent

https://www.1point3acres.com/bbs/thread-1087358-1-1.html
https://www.1point3acres.com/bbs/thread-1085478-1-1.html
https://www.1point3acres.com/bbs/thread-1090940-1-1.html



Part 1: Good intentions

To build the initial version of our system, we will support a few basic commands for initializing a merchant, creating a Payment Intent, attempting a Payment Intent, and succeeding a Payment Intent.

Commands

For this part, your system should handle the following commands

```
INIT <merchant_id> <starting_balance>
```

- Initializes a merchant with a unique identifier string and starting balance (the amount of money in their account).
- If a merchant with the given identifier has already been created, this command should do nothing.

```
CREATE <payment_intent_id> <merchant_id> <amount>
```

- Creates a Payment Intent for a merchant with a given amount. After creation, the state of the Payment Intent should be `REQUIRES_ACTION`.
- If a Payment Intent with the given identifier already exists, or if a merchant with the given identifier does not exist, or if the amount is negative, this command should do nothing.

```
ATTEMPT <payment_intent_id>
```

Input

```
INIT m1 0
INIT m2 10
CREATE p1 m1 50
ATTEMPT p1
SUCCEED p1
CREATE p2 m2 100
ATTEMPT p2
```

Output

```
m1 50
m2 10
```

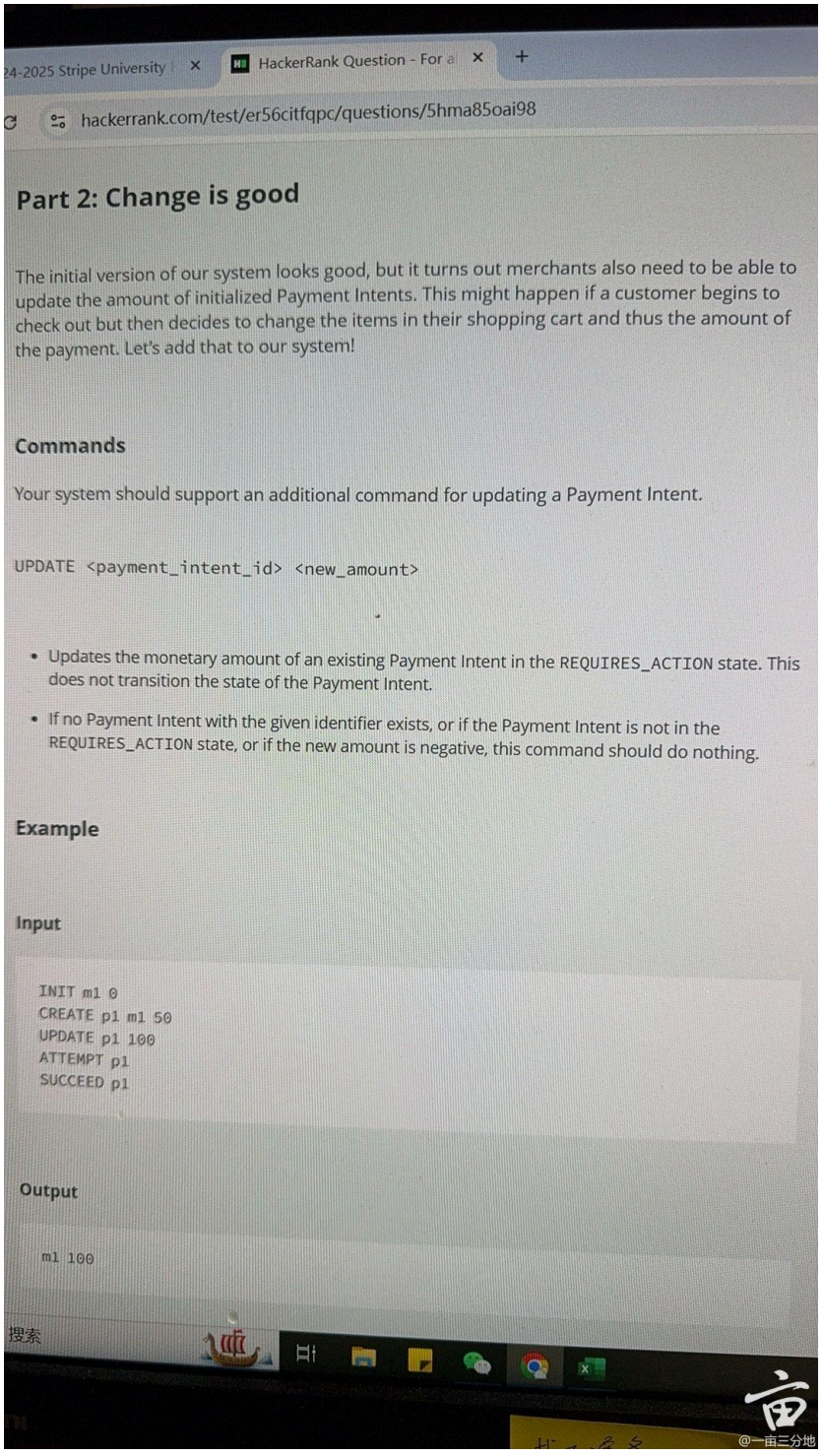
Explanation

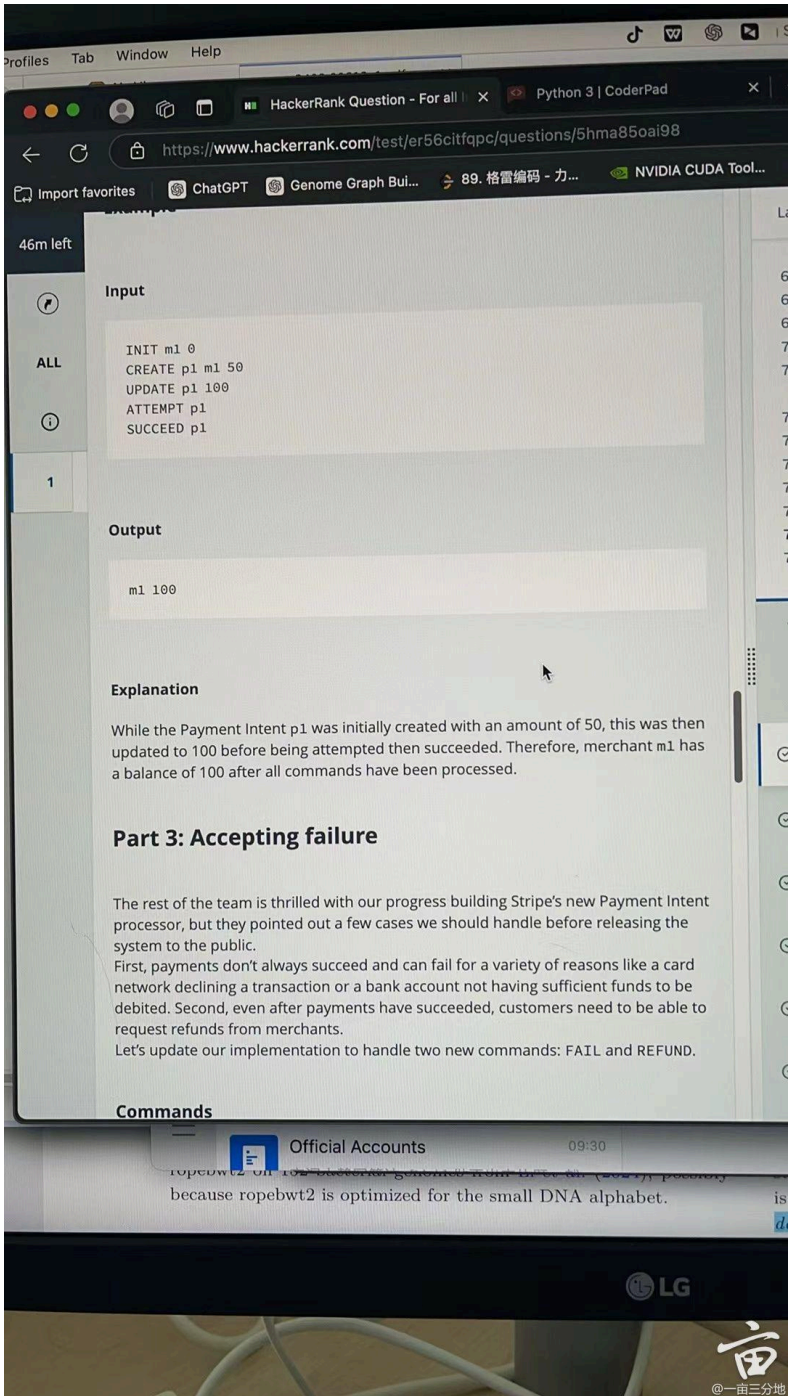
Let's look at what happened for each command

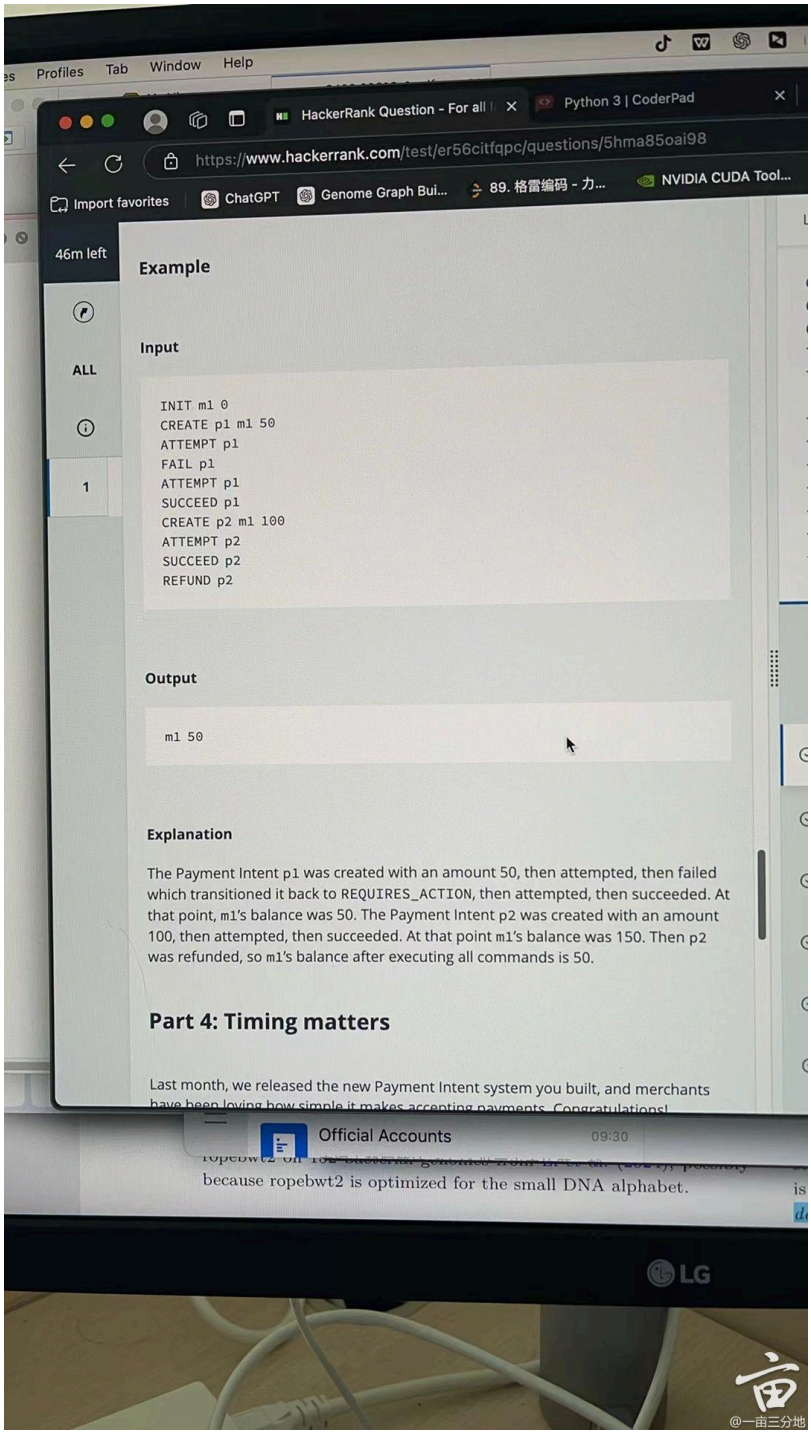
- `INIT m1 0` initializes merchant `m1` with a starting balance of 0
- `INIT m2 10` initializes merchant `m2` with a starting balance of 10
- `CREATE p1 m1 50` creates a Payment Intent `p1` for merchant `m1` with an amount of 50 and initial state of `REQUIRES_ACTION`
- `ATTEMPT p1` transitions `p1` from `REQUIRES_ACTION` to `PROCESSING`
- `SUCCEED p1` transitions `p1` from `PROCESSING` to `COMPLETED` and increments merchant `m1`'s balance by 50
- `CREATE p2 m2 100` creates a Payment Intent `p2` for merchant `m2` with an amount of 100 and initial state of `REQUIRES_ACTION`
- `ATTEMPT p2` transitions `p2` from `REQUIRES_ACTION` to `PROCESSING`

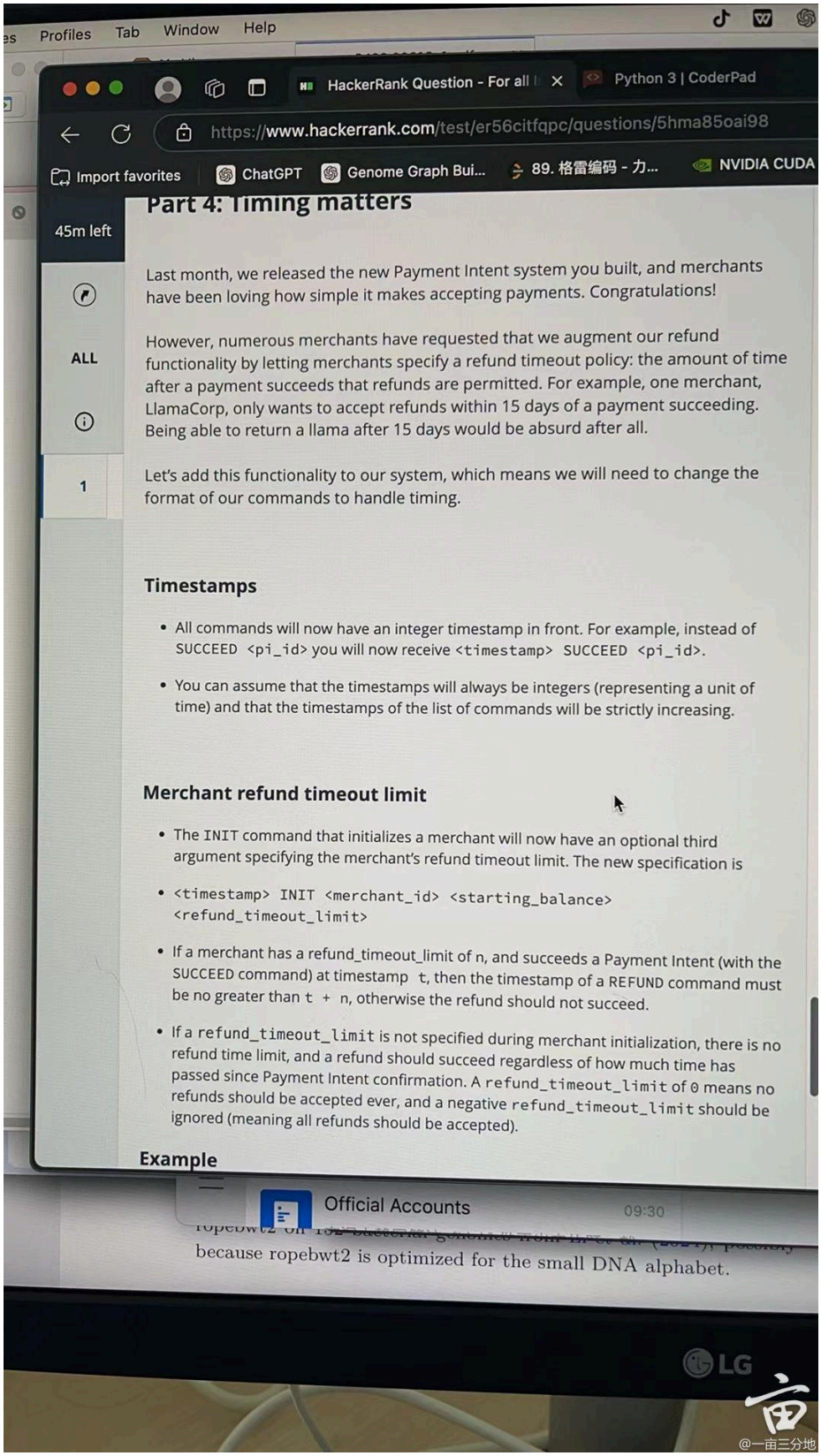
Because only `p1` was successfully completed, at the end of executing all of the commands, `m1` should have a balance of 50 and `m2` should have a balance of 10.

Part 2: Change is good


24-2025 Stripe University
HackerRank Question - For a
hackerrank.com/test/er56citfqpc/questions/5hma85oai98
Part 2: Change is good
The initial version of our system looks good, but it turns out merchants also need to be able to update the amount of initialized Payment Intents. This might happen if a customer begins to check out but then decides to change the items in their shopping cart and thus the amount of the payment. Let's add that to our system!
Commands
Your system should support an additional command for updating a Payment Intent.
UPDATE <payment_intent_id> <new_amount>
Updates the monetary amount of an existing Payment Intent in the REQUIRES_ACTION state. This does not transition the state of the Payment Intent.
If no Payment Intent with the given identifier exists, or if the Payment Intent is not in the REQUIRES_ACTION state, or if the new amount is negative, this command should do nothing.
Example
Input
INIT m1 0
CREATE p1 m1 50
UPDATE p1 100
ATTEMPT p1
SUCCEED p1
Output
m1 100
搜索
@一亩三分地


Profiles Tab Window Help
HackerRank Question - For all
Python 3 | CoderPad
https://www.hackerrank.com/test/er56citfqpc/questions/5hma85oai98
Import favorites
ChatGPT
Genome Graph Bui...
89. 格雷编码 - 力...
NVIDIA CUDA Tool...
46m left
ALL
1
Input
INIT m1 0
CREATE p1 m1 50
UPDATE p1 100
ATTEMPT p1
SUCCEED p1
Output
m1 100
Explanation
While the Payment Intent p1 was initially created with an amount of 50, this was then updated to 100 before being attempted then succeeded. Therefore, merchant m1 has a balance of 100 after all commands have been processed.
Part 3: Accepting failure
The rest of the team is thrilled with our progress building Stripe's new Payment Intent processor, but they pointed out a few cases we should handle before releasing the system to the public.
First, payments don't always succeed and can fail for a variety of reasons like a card network declining a transaction or a bank account not having sufficient funds to be debited. Second, even after payments have succeeded, customers need to be able to request refunds from merchants.
Let's update our implementation to handle two new commands: FAIL and REFUND.
Commands
Official Accounts
09:30
because ropebwt2 is optimized for the small DNA alphabet.
LG
@一亩三分地


Profiles Tab Window Help
HackerRank Question - For all
Python 3 | CoderPad
https://www.hackerrank.com/test/er56citfqpc/questions/5hma85oai98
Import favorites
ChatGPT
Genome Graph Bui...
89. 格雷编码 - 力...
NVIDIA CUDA Tool...
46m left
ALL
1
Example
Input
INIT m1 0
CREATE p1 m1 50
ATTEMPT p1
FAIL p1
ATTEMPT p1
SUCCEED p1
CREATE p2 m1 100
ATTEMPT p2
SUCCEED p2
REFUND p2
Output
m1 50
Explanation
The Payment Intent p1 was created with an amount 50, then attempted, then failed which transitioned it back to REQUIRES_ACTION, then attempted, then succeeded. At that point, m1's balance was 50. The Payment Intent p2 was created with an amount 100, then attempted, then succeeded. At that point m1's balance was 150. Then p2 was refunded, so m1's balance after executing all commands is 50.
Part 4: Timing matters
Last month, we released the new Payment Intent system you built, and merchants
Official Accounts
09:30
because ropebwt2 is optimized for the small DNA alphabet.
LG
@一亩三分地


Profiles Tab Window Help
HackerRank Question - For all
Python 3 | CoderPad
https://www.hackerrank.com/test/er56citfqpc/questions/5hma85oai98
Import favorites
ChatGPT
Genome Graph Bui...
89. 格雷编码 - 力...
NVIDIA CUDA
45m left
ALL
1
Part 4: Timing matters
Last month, we released the new Payment Intent system you built, and merchants have been loving how simple it makes accepting payments. Congratulations!
However, numerous merchants have requested that we augment our refund functionality by letting merchants specify a refund timeout policy: the amount of time after a payment succeeds that refunds are permitted. For example, one merchant, LlamaCorp, only wants to accept refunds within 15 days of a payment succeeding. Being able to return a llama after 15 days would be absurd after all.
Let's add this functionality to our system, which means we will need to change the format of our commands to handle timing.
Timestamps
All commands will now have an integer timestamp in front. For example, instead of SUCCEED <pi_id> you will now receive <timestamp> SUCCEED <pi_id>.
You can assume that the timestamps will always be integers (representing a unit of time) and that the timestamps of the list of commands will be strictly increasing.
Merchant refund timeout limit
The INIT command that initializes a merchant will now have an optional third argument specifying the merchant's refund timeout limit. The new specification is
<timestamp> INIT <merchant_id> <starting_balance> <refund_timeout_limit>
If a merchant has a refund_timeout_limit of n, and succeeds a Payment Intent (with the SUCCEED command) at timestamp t, then the timestamp of a REFUND command must be no greater than t + n, otherwise the refund should not succeed.
If a refund_timeout_limit is not specified during merchant initialization, there is no refund time limit, and a refund should succeed regardless of how much time has passed since Payment Intent confirmation. A refund_timeout_limit of 0 means no refunds should be accepted ever, and a negative refund_timeout_limit should be ignored (meaning all refunds should be accepted).
Example
Official Accounts
09:30
because ropebwt2 is optimized for the small DNA alphabet.
LG
@一亩三分地