



SHOPEE CODE LEAGUE 2020

GUIDELINES TO THE COMPETITION SOLUTIONS

Updated: 6 Aug 2020

Competition #1: Order Brushing [Data Analytics]	1
Competition #2: Product Detection [Data Science]	3
Competition #3: Shopee Programming Contest #1 [Engineering]	5
[NEW] Competition #4: Title Translation [Data Science]	9
Competition #5: Logistics [Data Analytics]	11
[NEW] Competition #6: Sentiment Analysis [Data Science]	13
Competition #7: Shopee Programming Contest #2	16

Competition #1: Order Brushing [Data Analytics]

Task:

1. Identify all shops that are deemed to have conducted order brushing.
2. For each shop that is identified to have conducted order brushing, identify the buyers suspected to have conducted order brushing for that shop.

For the purpose of this question, shops are deemed to have conducted order brushing if their concentrate rate is greater than or equal to 3 at any instance

Concentrate rate = Number of Orders within 1 hour / Number of Unique Buyers within 1 hour

For the purpose of this question, suspicious buyers are deemed as the buyer that contributed the highest proportion of orders to a shop that is deemed to have conducted order brushing.

For calculation of the highest proportion of orders to a shop, only include the orders that occurred in instances when order brushing has been deemed to have taken place.

In the case where multiple users share the same highest proportion of orders for a specific shop, all those users are deemed to be suspicious buyers.

Approach

The basic idea is calculating Concentrate rate for all possible 1 hour windows. Here illustrate the strategy:

- a. Select a 1 hour window and take a slice from order data.

Note when you set the time condition, both ends are included in the 1 hour window. (Hint: For Python, datetime module and timedelta function can be applied)

- b. Find shops which satisfy $CR \geq 3$.

For the slice of order data you select, you can use an aggregation function like groupby to calculate order number and unique user number for each shop in this 1 hour window. And then filter shops which satisfy the $CR \geq 3$ condition. They are brushing shops.

- c. Find the userid who conducts order brushing in brushing shops.

When brushing shops are identified, aggregation function can be applied again to calculate which userid contribute the most orders for each shop. Note if two userid both have the highest order number, then they are both the answer.

- d. Repeat the above procedure for all possible 1 hour window.

For				example:
2019-12-27	00:00:00	~	2019-12-27	01:00:00,
2019-12-27 00:00:01 ~ 2019-12-27 01:00:01, ...				

Note:

When you follow the above steps, you may find **d** step takes a long time since the time period in our data crosses 5 days and the time window is detailed to second. To reduce the time consumption, some advanced strategy might be applied. For example:

- a. Multiprocessing to brutally calculate them all.
- b. Avoid redundant calculation by detecting windows which cover exactly the same orders.

It would be our pleasure if you find your own way to solve it. Build your code and have fun~

Competition #2: Product Detection [Data Science]

Task:

In this competition, a multiple image classification model needs to be built. There are ~100k images within 42 different categories, including essential medical tools like masks, protective suits and thermometers, home & living products like air-conditioner and fashion products like T-shirts, rings, etc. For the data security purpose the category names will be desensitized. The evaluation metrics is top-1 accuracy.

Approach

Preprocessing:

Clean the dataset by some cluster techniques such as p-hash, image histogram and metric learning. Since we know that the most of images within one category are correct, can use such techniques to remove those outliers as much as possible.

For some categories like class No.33 with only ~500 samples, may have less samples than others, you might need to use some augmentation to upsample those categories. Balanced dataset is extremely important for classification, it is always recommended to make dataset balance as possible as you can.

The preprocessing and training strategy may count more than the model itself sometimes.

Data Augmentation:

Some useful techniques for image augmentation are color jittering, jpeg compression, cutout, mixup, label smoothing etc. Those techniques will help to make the model more robust which will let your score more stable for public leaderboard and private leaderboard.

Classification Model:

There are quite a few models that are suitable for this task. ResNet, ResNext, EfficientNet, ResNeSt, VGG, InceptionNet and etc. For such a 40-category classification task, it is recommended to use a relatively larger model to avoid overfitting and also for better performance.

Training strategy:

This part is another important part, but it totally depends on different conditions like different machines, different preprocessing and different optimizers. Basically, if you have a strong machine which can support you to train with a large batch size, then you might use a relatively large learning rate and less epochs. But if your machine cannot support a large batch size training, to get the same level result, you might need to use a smaller learning rate and more epochs. Or you can also use the loss accumulation strategy to simulate larger batch size.

And do note that the training strategy should be decided based on your analysis for validation/public leaderboard result. With some iterations of submissions and analysis, the strategy will be polished to a perfect state.

Competition #3: Shopee Programming Contest #1 [Engineering]

Task: Please refer to the problem statements [here](#).

You may refer to the solution/guideline for each problem statement below.

[3.1 Item Stock](#)

[3.2 Search Engine](#)

[3.3 Judging Servers](#)

[3.4 Lucky Winner](#)

[3.5 Sequences](#)

3.1 Item Stock

We will calculate the final stock for each fixed stock item first since dynamic stock items will always be derived from them.

For each item, store its lowest ancestor which is a fixed stock and the multiplication of the **Qty** to that ancestor. Then, each time we create a fixed stock item we can easily do the stock deduction in **O(1)**. After we finished calculating the fixed stock items, we can just derive the dynamic stock items. The total complexity is **O(N)**.

3.2 Search Engine

This problem is quite straightforward and requires **Trie Tree** style data structure. This problem can also be solved using other techniques. For each input item name, first, hash the words in the name to an integer. For example, “iPhone SE 2” can be converted to a vector of integer [1 2 3] and in the same way, “iPhone 11 Pro” can be converted to [1 4 5].

Then you need to insert all the suffixes of the vector containing the hash values, into the trie tree data structure.

Each node of the Trie tree will store the number of times this node as part of a different item. One tricky part is to eliminate the overcounts. Consider the number of items in the

input is N and each work can have m space. The overall memory complexity is $O(N * m^2)$. And Time Complexity is $O(Q * m)$.

3.3 Judging Servers

This is a classical dynamic programming problem. Only one tricky part of the problem is that you can choose not to take a free server, actually taking a free server is optional. Any point of time you can be in three different types of state and you have to minimize the cost

// 0 => no back

// 1 => took previous one with free now need to buy the current

// 2 => the next one is free or you can buy.

From state 1,

You took the previous server for free, so you pay for the current one -->? move to state 0.

From state 2,

you can take the current server for free → move to state 0

you can buy it → moves to state 2.

You can skip this server → move to state 0

From State 0,

Skip the current server → move to state 0

Take the current server for free → move to state 1

Buy the current server → move to state 2

Time Complexity and memory, $O(N * S) \sim O(N^2)$

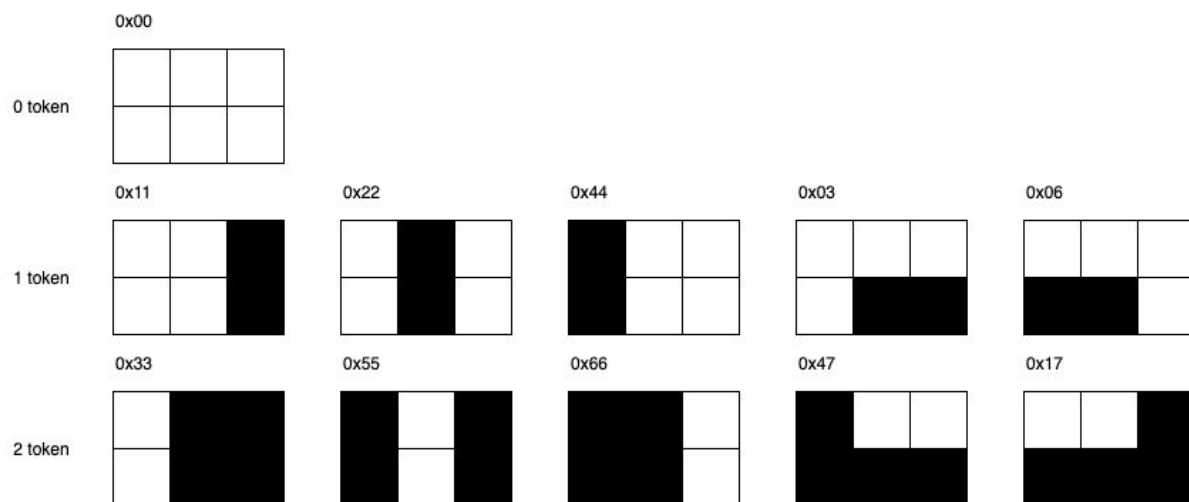
3.4 Lucky Winner

This is a classic dynamic programming problem.

Let $DP[i][j][mask]$ be the maximum worth of items consider that we've used j **tokens** on first i **rows** with **mask** being a bitmask describes the state of **i-th row** (3 bit bitmask, so 8 states in total)

We'll need to calculate this table. Result will be max of $DP[n][k][mask]$ with $0 \leq mask < 8$

The solution for a state can be calculated based on the states of the previous row. There are 11 ways to use the tokens in the last 2 rows. Each of these ways needs some available fields in the last row and will take some fields in the current row. To simplify the implementation, we represent each way of placing tokens in the last rows as a bitmask where the first digit is the state of occupied fields in the previous row and the second is the state of occupied fields in the current row. Please refer to the image below about states and their costs.



The number of states is $N * K * 8 = 8e6$, there are 11 ways of using tokens in the last 2 rows. Overall complexity $O(N * K)$

3.5 Sequences

Let's denote **count(x, y)** as the number of sequences with length **x** which sum modulo **K** equals to **y**. Clearly, the answer we seek will be **count(M, 0)**. Now, observe that **count(a+b, c)** is equal to the summation of **count(a, i) x count(b, (c-i))** for each $0 \leq i < K$. Following this observation, we can calculate the value of **count(x, *)** in similar fashion to exponentiation by squaring. This part can be done in $O(K^2 \log M)$, where K^2 comes from calculating **count(a+b, *)** for any given **a** and **b**.

Now, we need to calculate the base case, which would be **count(1, *)**. Note that this is equal to finding the distribution of elements of each **f(x, y)** from the description in terms of modulo **K**. Observe that for any given sequence, it will be cycled every **K** terms since $K^2 \bmod K = 0$. Hence, using this we can calculate **count(1, *)** in $O(K)$ for each given function, which totals to $O(NK)$ for counting them all.

Combining those, we got a solution with total complexity $O(K^2 \log M + NK)$.

[NEW] Competition #4: Title Translation [Data Science]

At Shopee, we have customers and sellers across Southeast Asia and Taiwan. In order to provide a better shopping experience, we set up the cross border business to improve the variety of products for our customers. In order to help the local sellers to migrate the SKUs to different foreign markets, we hire a group of professional human translators with an e-commerce background to take care of the product translation. The product information translated includes product title, variation and description. However, as Shopee is growing tremendously in recent years, the amount of translation per day is beyond the human translators' capacity. At the same time, with the development of AI technology, machine translation is now deployed in many industrial areas to assist the human translators and they can achieve a near-human level translation quality.

In Shopee, we have an in-house machine translation pipeline which can translate millions of SKUs per week in different languages. The languages include Traditional Chinese, Bahasa, English, Vietnamese, Thai and Portuguese. The challenges in machine translation is usually the lack of labelled data. However, different ways of unsupervised machine translation have been explored and proven to be effective, with little or even no label data. Some techniques are cross-lingual word alignments or pretrained cross-lingual language models.

Task: Given a product title in Traditional Chinese, the candidate is expected to translate the title into English.

Dataset: Candidates are provided with two monolingual product title data (in Traditional Chinese and English). Use of public data is encouraged.

Metrics: Bleu score of the whole test set is used to assess the translation quality.

Approach

Preprocessing:

Tokenization:

Jieba is used as the Chinese tokenizer. Some specific ecommerce terms can be found online and input as a user dictionary to Jieba. This can give a more accurate tokenization result. For English titles, white space tokenizer can be used.

Lowercase:

The words can be lower-cased since the competition evaluates on the lower-cased text.

Subword:

Subword is normally used in machine translation tasks. There are two major advantages: one is to reduce vocabulary size so the model can learn more richful information for the smaller piece. The other advantage is that it can alleviate the problem of out-of-vocabulary during inference (where a word occurs in testing but is not seen in training dataset). Google's sentence piece or Byte-Pair Encoding (BPE) are usually used.

Model Training:

Building dictionary

The first step of doing unsupervised machine translation is always to build an aligned dictionary between source and target language. There are a few ways to build the dictionary. For example, you can collect the dictionary from various online released dataset or existing translation APIs. Also, you can do a supervised/unsupervised word alignment by using Facebook's MUSE library.

Model selection

Transformer is the most recommended model due to its effectiveness in capturing the representations and excellent performance. If you are trying statistical machine translation, phrase-based statistical machine translation is recommended. An initial model can be obtained by using language models and the aligned dictionary. Iterative improvements on this initial model are done with back-translation techniques.

Back-translation

Back-translation is a common strategy in machine translation, where in training a backward model from source to target, you actually build a target-to-source model at the same time. This will give you pairs of (synthetic source, actual target) data for your forward model. It will enhance the translation quality since it sees the actual target sentences.

Competition #5: Logistics [Data Analytics]

Due to the recent COVID-19 pandemic across the globe, many individuals are increasingly turning to online platforms like Shopee to purchase their daily necessities. This surge in online orders has placed a strain onto Shopee and our logistics providers but customer expectations on the timely delivery of their goods remain high. On-time delivery is arguably one of the most important factors of success in the eCommerce industry and now more than ever, we need to ensure the orders reach our buyers on time in order to build our users' confidence in us.

In order to handle the millions of parcels that need to be delivered everyday, we have engaged multiple logistics providers across the region. Only the best logistics providers that are able to meet Shopee's delivery standards are partnered with us.

The performance of these providers is monitored regularly and each provider is held accountable based on the Service Level Agreements (SLA). Late deliveries are flagged out and penalties are imposed on the providers to ensure they perform their utmost. The consistent monitoring and process of holding our logistics providers accountable allows us to maintain our promise of timely deliveries to our buyers.

Task:

Identify all the orders that are considered late depending on the Service Level Agreements (SLA) with our Logistics Provider.

For the purpose of this question, assume that all deliveries are considered successful by the second attempt.

Approach

Here are the main difficulties in this problem statement:

1. Processing of large datasets
2. Handling of epoch time
3. Parsing the address fields to extract the relevant states when calculating SLAs
4. Weekend & Public Holiday exemptions when calculating SLAs
5. 1st attempt deliveries & 2nd attempt deliveries

Perhaps the largest challenge would be the exemptions for Weekend & Public Holiday and applying it to both 1st & 2nd attempts

General approach to solving this:

- 1) Convert all the epoch time format into something that is easy to read and compute.

For python, this can be done easily with “*pd.to_datetime*”. However, do be careful as this will convert the time to GMT timezone. We can change it to GMT+8 by simply adding 8 hours in seconds to the epoch time before applying “*pd.to_datetime*”

- 2) Extract all the names of the states from both buyer and seller addresses.

A simple string match or regex should yield the correct states

- 3) Get # of days between pickup and 1st delivery attempt

We ignore the time and consider only the date. Since we have previously converted our epoch time using “*pd.to_datetime*”, we can continue using Pandas’ datetime functions. We can get the difference between the 2 datetimes and apply “*dt.days*” to get the number of days. Do the same for 1st delivery attempts and 2nd delivery attempts.

- 4) Check if any dates should be exempted from the calculation

One way is to check how many weekends/public holidays exist between the start and end. We’ll have to get the dates of all the Sundays and Public Holidays. If any of those dates are between the start and end, then we minus it from the # of days previously calculated in (3). Note, this step might take quite some time since we’ll have to apply the same calculation to both pickup -> 1st attempt & 1st attempt -> 2nd attempt so do test it on a sample set first

- 5) Compare the # of days with the respective SLAs based on the SLA matrix

If the # of days is more than the SLA, it is deemed late. Both the 1st delivery attempt and 2nd delivery attempt have to be on time, if either is late, then the whole order is considered late.

[NEW] Competition #6: Sentiment Analysis [Data Science]

At Shopee, we always strive to ensure the highest satisfaction of our customers. Whatever is sold on Shopee, we ensure the best user experience starting from product searching to product delivery, including product packaging, and product quality. Once a product is delivered, we always encourage our customers to rate the product and write about their overall experience on the product landing page.

The rating and comments provided for a product by our buyers are most important to us. These product reviews help us to understand our customers needs and quickly adapt our services to provide a much better experience for our customers for the next order. The user's comments for a product ranges from aspects including delivery services, product packaging, product quality, product specifications, payment method, etc. Therefore, it is important for us to build an accurate system to understand these reviews which has a great impact on overall Shopee's user experience. This system is termed: "Shopee Product Review Sentiment Analyser".

Task: In this competition, a multiple product review sentiment classification model needs to be built. There are ~150k product reviews from different categories, including electronics, furniture, home & living products like air-conditioner and fashion products like T-shirts, rings, etc. For data security purposes, the review ids will be desensitized. The evaluation metrics is top-1 accuracy.

Approach

Preprocessing:

Data preprocessing is the most important task for getting good accuracy for this problem. The dataset is noisy and contains tokens from other languages (i.e., Bahasa, Malay). The data is generated by the user, so it's called user generated content (UGC). Following are few examples from datasets and how to handle those samples:

1. 'GOOD GOOD GOOD GOOD GOOD GOOD GOOD GOOD GOOD GOOD'
 - a. Repetition of a particular adjective in the review.
 - b. Repetition can increase confidence being a particular rating.
 - c. When there is repetition of a particular adjective, it has an indication for higher rating.
 - d. Repetition of stop words can be removed but while removing repetition of adjective, or other aspect/opinion have to be extra careful.
2. 'Excellent packaging, the product is **sooooo goooooood**.'
 - a. **sooooo** and **goooooood** are not from any language vocabulary.

Misspelled tokens.

- b. UGC must have grammatical and spelling errors and no proper sentence structures.
- c. word embeddings or n-gram (char level) embeddings can be used to handle these cases as follow

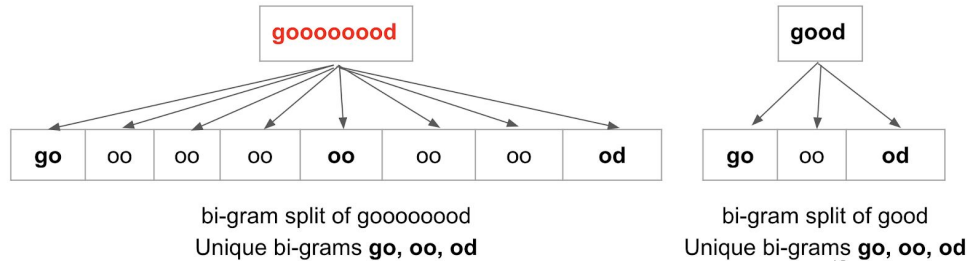


Figure 1. a) bi-gram split of the misspelled token. b) bi-gram split of the actual token.

- d. In the above example “**gooooood**” can not be recognized in any language vocabulary. So if we split the token into bi-gram, (go, oo, od) would be the unique bi-grams which are common for both “**gooooood**” and “good”. This way character level embeddings may help to find the nearest meaning of a misspelled token in the dataset.

3. ‘Like 🍷🍷🍷🍷🍷🍷 funny 🍷🍷🍷🍷🍷🍷🍷🍷 Excellent product quality 🍷🍷🍷🍷’

- a. Lots of emojis in the dataset. Buyers express their sentiment toward a particular product using emojis.
- b. A specific emojis may indicate a particular rating. Also may mislead sometimes.
- c. We can use emojis in prediction but have to be extra careful. If we are very much confident from the dataset that any specific emojis would be a particular class, we can use emojis otherwise can remove it during preprocessing.

4. ‘These shoes are very comfortable and it's wonderful to be able to get them in a 4E. Size wise they are shorter than expected . I typically wear a 9 and based on other reviews I ordered a 9.5’

- a. This is a long review with multiple sentences.
- b. One solution is to split the sentences, make individual predictions and ensemble the output. Sometimes aspect and the opinion related to the aspect can directly help to find the sentiment or rating for that particular product. Following figure shows how to extract aspect and opinion from a

given review -

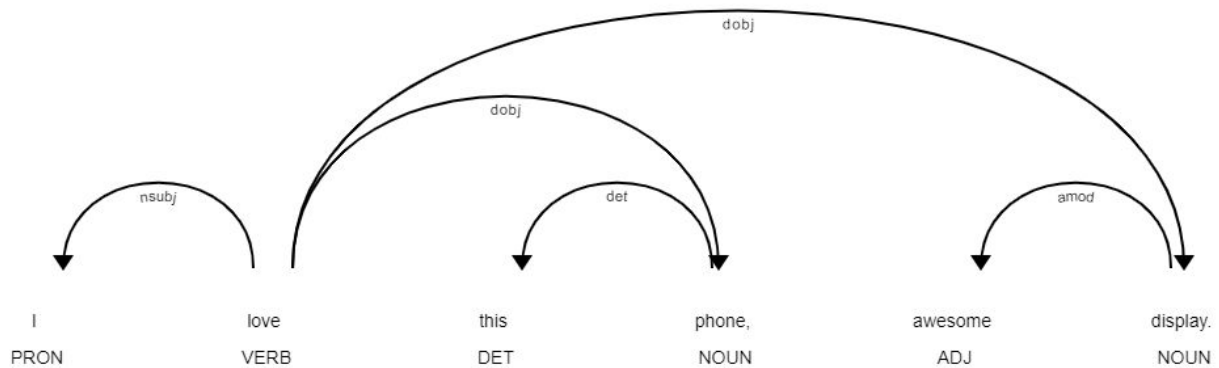


Figure 2. Dependency parsing of a review.

- c. We can use dependency parsing (from spacy or other framework) to find the relationship between tokens in the review. The 'amod' or "adjective modifier" relationship can be used to find the (aspect and opinion) pair from the review. In the above example the 'amod' relationship between "awesome" and "display" clearly indicates that "display" is the aspect and "awesome" is opinion for that aspect.

5. *'Bagussss be that his bingittttt 🥰🥰🥰, the first new order here and satisfied, sellernya also friendly bgt, Thank you.'*

- a. In this review there are tokens from other languages. So while building the model we can use vocabulary from other languages to identify those tokens.

Classification Model:

For training we can use one of the recent language models for classification. BERT is the most recommended model for this type of task. BERT has a lot of different variations. The 'bert-base-uncased' pre-trained model can be used. The reviews are not case sensitive so we can use the uncased model and the base model is good enough to get good accuracy.

Competition #7: Shopee Programming Contest #2

Task: Please refer to the problem statements [here](#).

You may refer to the solution/guideline for each problem statement below.

[7.1 Highest Mountain](#)

[7.2 Shopee Logistics](#)

[7.3 Connectivity](#)

[7.4 Wifi Network](#)

[7.5 Number Tree](#)

7.1 Highest Mountain

For this problem, we can solve it with the following steps:

1. Locate all the number "1"s in the array
2. Starting from the first "1" from left side of the array, try to increase by 1 for both left and right and get the max value between left and right
3. If there is an existing max value, compare it with the result from Step 2.
4. Store the max value and the index with variables.
5. Repeat step 2-4 for all "1" in the array.
6. After we finished with all the number "1"s, the stored value will be the highest height and index.

7.2 Shopee Logistics

The main idea is to find the longest path in the tree (graf). To make it you can use dynamic programming or using two searches:

Run BFS from any node to find the farthest leaf node. Label that node T1. Run another BFS to find the farthest node from T1 (will name this node T2). The path you found in step 2 is the longest path in the tree.

OK! Now we can find the longest path. To Find the second longest path we can remove node T1 from init graf and try to find the longest path (L1). Remove node T2 from init graf and to find the longest path (L2).

The answer will be $\text{Max}(L1, L2)$.

7.3 Connectivity

Here the switches are equivalent to nodes and connections are equivalent to edges in a graph. Also, the PUSH and POP operations here are similar to that of a stack. So we keep track of the edges added or removed to the graph using a stack.

We also maintain a disjoint-set union (dsu) data structure optimized by rank. At a PUSH operation, we add the edge in dsu and also keep track of which node was added to which parent node in this step.

At a POP operation, we get the edge from top of our stack and remove the parent-child connection we made in the previous step.

7.4 Wifi Network

This problem can be solved using a segment tree or BIT data structure.

1. First sort the input coordinates of the engineers based on their distance from the two wifi hubs.
2. For each engineer save the sorted index from AirFi hub.
3. Then sort all the radius tuple in the query based on the area covered by them.
4. Keep a global index(initially 0) for all engineers and traverse the sorted queries.
 - a. If the engineer (sorted based on the distance at step 1) at global index is covered by the current radius of GeoFi
 - i. Set the position of that engineer in the calculated in step 2 in a segment tree or BIT
 - ii. Increment the global index
 - b. For the current AirFi radius find the index in the sorted list of engineers calculated in the 1st step. (Binary Search) \rightarrow airfi_position
 - c. The answer for this query is $= (n - \text{global index} - \text{airfi_position} - 1)$
 - d. Add the over deduction from step 4.c. The final result for the current query is, result at 4.c + $2 * x$. Where x is the number of engineers covered by AirFi from index 0 to airfi_position. The value of x can be calculated using a segment tree or BIT

Time Complexity is $O(N * \log N^2)$.

7.5 Number Tree

For this problem, we can solve it in the following steps:

1. Pick an arbitrary root
2. Do calculation for all possible paths that go through the root
3. Remove the root from the tree, which will split the tree into multiple trees
4. Repeat above steps from the produced trees, until we don't have any node to process

To optimize the 1st step, we can use [centroid decomposition](#), which will ensure that any node will be processed in at most $O(\log N)$ steps, and after each step the newly produced trees will have size at most half of the original tree.

To optimize the 2nd step, we can optimize it by using data structure that can support range sum query, and keep the following data:

- counts of node we have processed from the current tree
- sum of numbers, if we construct it from the root into the node
- sum of numbers, if we construct it from the node into the root
- sum of $10^{(\text{length from node to root})}$ for each node we have processed from the current tree

If we have those data, we can quickly calculate the sum of any pair $(i, \text{current node})$ for each i which are nodes we have processed in the current tree. By using a data structure which can do update and range sum query in $O(\log N)$, the second step can be done in $O(N \log N)$ for a tree with size N .

Combining 1st and 2nd optimization, we can obtain a solution with $O(N \log^2 N)$ complexity.