CocCoc

Java version:

```
openjdk 11.0.9 2020-10-20
OpenJDK Runtime Environment (build 11.0.9+11-Ubuntu-0ubuntu1.18.04.1)
OpenJDK 64-Bit Server VM (build 11.0.9+11-Ubuntu-0ubuntu1.18.04.1,
mixed mode, sharing)
```

class DataStream

Present object has **object_id**, **category_ids** and **category_counts**.

Source code: *src/vn.coccoc/DataProcessing*

I. Data Processing

Idea:

- For each category, save it into a HashMap with key is category_id, value is category_count.
- If category_id is not exists in HashMap, add new. If exists, add category_count to existed value.

1. What is the most popular category for this sample ?

Source code: *src/vn.coccoc/DataProcessing*

Output:

```
Most popular category is 6 with 1899183 counts
```

2. Which category has the largest appeared times ?

As I understand, this question is counting number of appeared times of **category_id** then find the max. And no care about **category_counts**. So to do this task, instead of adding category_count for each category_id, just add one for each appeared time.

Source code: *src/vn.coccoc/DataProcessing*

Output:

```
Largest appeared category is 15276 with 85471 times
```

3. Is there any idea how to represent/visualize the data sample for analysing ?

I have used Kibana or AWS QuickSight for visualizing data. So I highly recommended using them for represent this data.

III. Algorithm

Idea:

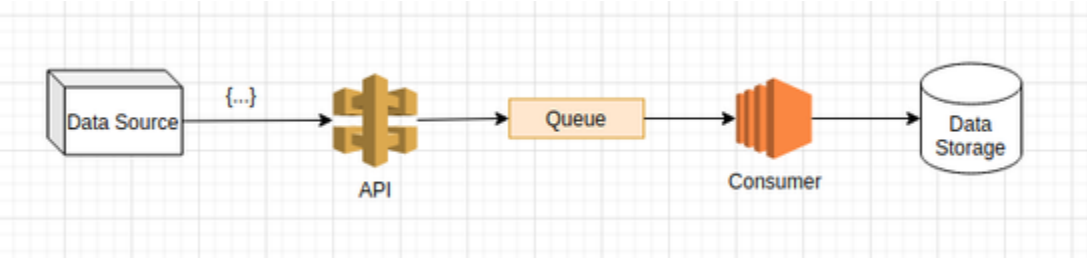
- Split large file into many smaller files.
- Sort each file by **object_id**.
- Combine all sorted files into one file.
    - Create a temporary list contains all first object of each file then sort item in it.
    - Get and remove the first object in above list. Write it to output file.

- Insert the next object from the file of removed object to the above list. If there is no line in this file, ignore it.
- Use **Binary Search** for find the position that new object will be located.
- Loop until there is no object in temporary list.

Source Code: *src/vn.coccoc/Algorithm*

> Another way to solve this problem is using **Trie** (re***Trie***val data structure) to present all **object_id** in a Tree. After that, using DFS to parse sorted object_id.

V. Categories of objects are updated in real-time



Step 1: DataSource send JSON data to API

Step 2: API verify JSON data and forward it to Queue

Step 3: Consumer receive verified data then update it to DataStorage

Pros:

Received data is verified and handle asynchronously.

Cons:

When architect has more than two **Consumer**, update **category_counts** when two **object_id** come at the same time maybe lead to conflicted.