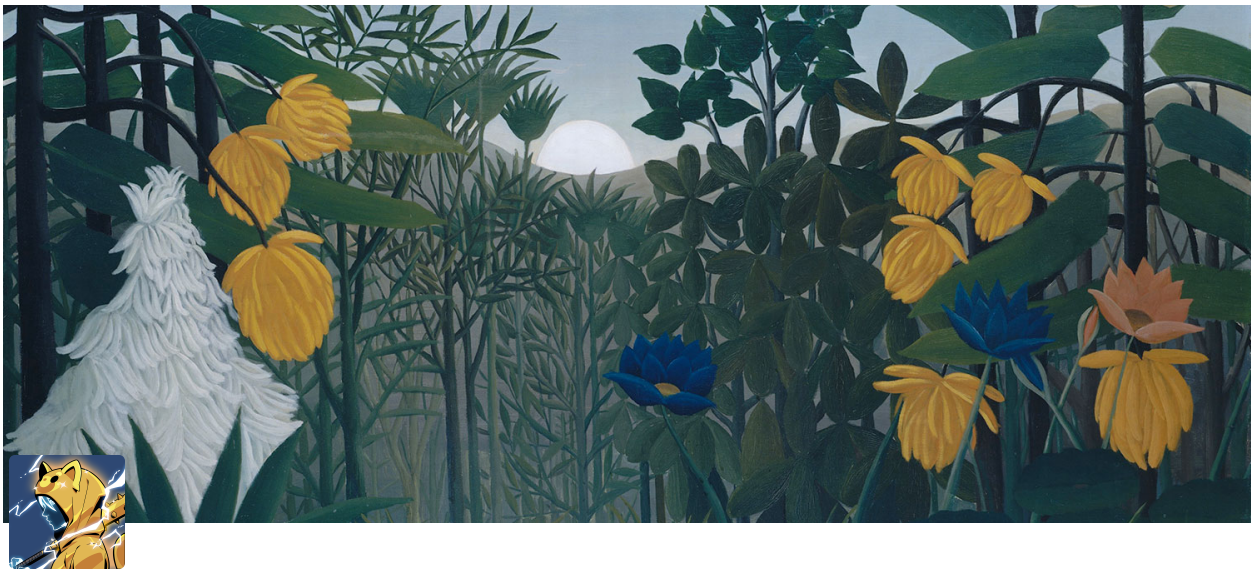


Skip Golang Take home

Prompt

❓ We have noticed a market inefficiency related to NFT releases that we would like to trade on. For many collections, the trait data is accessible on IPFS for collections that are liquid on Opensea before the Opensea frontend updates to display trait data to users.

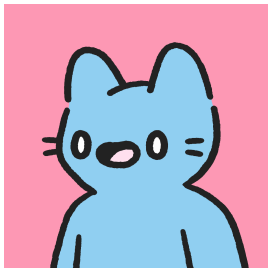
This means users holding rare NFTs list those NFTs at the floor price on Opensea without realizing they're selling potentially valuable NFTs at a discount to the value they will have when the trait data is more easily accessible and it becomes clear that the asset is rare.

Our goal is to build a program that allows us to identify instances of rare NFTs listed at floor on Opensea, purchase them before Opensea updates with the trait/rarity information, relist them at a higher price, and sell them, capturing the difference as profit.

You will only need to implement a portion of this to complete the assignment. Please read on for details.

Definitions and Background

Collection: A collection c is a n_c NFTs minted by a particular smart contract where each NFT has a unique visual appearance that consists of programitically and randomly combining a set of **trait values** from a predetermined set of **trait categories.** For example, the NFTs below come from the Cool Cats collection, which contains 9,999 items:



Trait Categories: Categorical variables giving visual features that define the appearance of each NFT in a collection. A collection c has m_c unique trait categories that are identified by strings. For example, the Cool Kat NFTs above include “face”, “shirt”, “hat”, “background”, and “earring” as trait categories.

Trait values: The actual string values values that a trait may take on. For each NFT, each trait category i, each token t may only take on at most 1 trait value $v_{t,i}$ from a set of o_i discrete possibilities. For example, the “hat” trait in the Cool Kat collection may take on the trait values of “barrett green” or “mohawk green” among others.

The order of operations is important to understand:

1. At time t_0, a collection attached to a specific smart contract address begins minting, then users begin minting NFTs from the collection
2. After minting, some paper hands traders turn around and immediately list their NFT at the mint price on Opensea.
3. At time $t_1 > t_0$, the rarity data becomes available on via IPFS
4. At time $t_2 > t_1$, the rarity data becomes available on Opensea, and users who have listed their rare NFTs at floor price realize their mistake and raise the prices

Rarity Definition

Opensea and other rarity calculators use the following metric to define the rarity of a token t for a collection c:

$$\text{rarity}_{c,t} = \sum_{i=0}^{m_c} \frac{1}{\sum_{j=1}^{n_c} 1(v_{j,i} == v_{t,i}) \cdot o_i}$$

In plain English that means for a particular token, summing over each trait category the reciprocal of the number of NFTs with that particular trait value times the number of trait values for that category.

In simplified pseudocode it is:

```
rarity = 0
for all categories:
  rarity += 1 / (count_with_value * num_values_in_category)

## example
rarity =
(1/ (50 * 20)) + (1/(1 * 2))
```

The higher the value, the more rare a token is.

As an example, consider a token

```
{
"hat": "green beret",
"earring": "gold"
}
```

In a collection where there are:

- 20 different values for “hat” (e.g. “red beret”, “red baseball cap”, and so on …)
- 50 tokens with the “green beret” for “hat”
- 2 different kinds of earring

- 1 token with “gold” for “earring”

This token’s rarity would be:

```
(1/ (50 * 20)) + (1/(1 * 2))
```

Assignment

Goal: Write a program to download the trait metadata for a collection, compute the rarity scores for all tokens, sort the list by rarity, and output the rarity scores of the top 5 tokens in golang.

Stub program: The provided stub program includes data structures to represent tokens and the rarity scores for tokens, as well as a helper method for retrieving a token’s metadata from our server.

Hints:

1. You should leverage concurrency and golang’s built in concurrency primitives.
2. If you do use concurrency, you should make the maximum number of threads configurable
3. As a sanity check, the most rare token is 6088 with a rarity score of 0.00856

Questions: Please reach out to us if you have any questions or problems.