Daily Coding Problem #287

Problem

This problem was asked by Quora.

You are given a list of `(website, user)` pairs that represent users visiting websites. Come up with a program that identifies the top k pairs of websites with the greatest similarity.

For example, suppose `k = 1`, and the list of tuples is:

```
[('a', 1), ('a', 3), ('a', 5),
 ('b', 2), ('b', 6),
 ('c', 1), ('c', 2), ('c', 3), ('c', 4), ('c', 5)
 ('d', 4), ('d', 5), ('d', 6), ('d', 7),
 ('e', 1), ('e', 3), ('e': 5), ('e', 6)]
```

Then a reasonable similarity metric would most likely conclude that a and e are the most similar, so your program should return `[('a', 'e')]`.

Solution

The first question we must answer is: what does similarity mean in this context?

While there are several ways to mathematically represent similarity, one of the most straightforward is known as the Jaccard index, and is computed for two sets by dividing the

straightforward is known as the Jaccard index, and is computed for two sets by dividing the size of their intersection by the size of their union.

For example, consider websites a and c in the example above. Since their intersection consists of `{1, 3, 5}` and their union is `{1, 2, 3, 4, 5}`, the Jaccard index for this pair would be `3 / 5`, or `0.6`.

```
def compute_similarity(a, b, visitors):
    return len(visitors[a] & visitors[b]) / len(visitors[a] | visitors[b])
```

This function relies on our ability to quickly find all the visitors for a given website. Therefore, we should first iterate through our input and build a hash table that maps websites to sets of users.

Following this, we can consider each pair of websites and compute its similarity measure. Since we will need to know the top scores, we can store each value and the pair that generated it in a heap. To keep our memory footprint down, we can ensure that at any given time only the k pairs with the highest scores remain in our heap by popping lower-valued pairs.

Finally, we can return the values that are left in our heap after processing each pair of websites.

```
def top_pairs(log, k):
    visitors = defaultdict(set)

    for site, user in log:
        visitors[site].add(user)

    pairs = []
    sites = list(visitors.keys())

    for _ in range(k):
        heapq.heappush(pairs, (0, ('', '')))

    for i in range(len(sites) - 1):
        for j in range(i + 1, len(sites)):
            score = compute_similarity(sites[i], sites[j], visitors)
            heapq.heappushpop(pairs, (score, (sites[i], sites[j])))

    return [pair[1] for pair in pairs]
```

For each pair of websites, we must compute the union of its users. As a result, this part of our algorithm will take $O(N^2 * M)$, where N is number of sites and M is the number of users. Inserting into and deletion from the heap is logarithmic in the size of the heap, so if we assume $k < M$, our heap operations will be dominated by the calculation above. Therefore, our time complexity will be $O(N^2 * M)$.

As for space complexity, our hash table will require N^2 keys, and our heap will have at most k elements. Assuming $k < N^2$, then, the space required by this program will be $O(N^2)$.