Assignment: Document Hash

Qianlang Chen (u1172983)

CS 5140 Spring 2021

Problem 1

```python
num_docs = 4
doc_path = 'data/D%d.txt'

# Create the k-grams interested in this problem.
G1, G2, G3 = [], [], []
for doc in range(1, num_docs + 1):
    contents = open(doc_path % doc, 'r').read().strip()
    words = contents.split()
    # G1: 2-grams based on words
    G1.append({tuple(words[i:i+2]) for i in range(len(words) - 1)})
    # G2: 3-grams based on words
    G2.append({tuple(words[i:i+3]) for i in range(len(words) - 2)})
    # G3: 3-grams based on characters
    G3.append({contents[i:i+3] for i in range(len(contents) - 2)})
```

Part A

```
# Report the answers formatted nicely.
print('\t\tG1\tG2\tG3')
for i in range(num_docs):
    print(f'Document {i + 1}:\t{len(G1[i]):,}\t{len(G2[i]):,}'
          f'\t{len(G3[i]):,}')
```

```
                G1      G2      G3
Document 1:     161     167     482
Document 2:     142     147     443
Document 3:     390     423     977
Document 4:     364     381     770
```

Part B

```
# Computes and returns the Jaccard similarity between two sets `S` and
# `T`.
def jaccard(S, T): return len(S & T) / len(S | T)

# Report the answers formatted nicely.
print('Documents\tG1\tG2\tG3')
for i in range(num_docs - 1):
    for j in range(i + 1, num_docs):
        print(f'{i + 1} vs {j + 1}:\t\t{jaccard(G1[i], G1[j]):.4f}'
              f'\t{jaccard(G2[i], G2[j]):.4f}'
              f'\t{jaccard(G3[i], G3[j]):.4f}')
```

```
Documents       G1      G2      G3
1 vs 2:         0.8704  0.8580  0.9191
1 vs 3:         0.0110  0.0017  0.2449
1 vs 4:         0.0057  0.0000  0.2621
2 vs 3:         0.0095  0.0018  0.2348
2 vs 4:         0.0060  0.0000  0.2557
3 vs 4:         0.0121  0.0012  0.3135
```

Problem 2

Part A

```python
import random
import time

# The size of the hash function `h`'s output space.
m = 2**16

# Hashes an object `x` using some random seed `i`. The random seed is
# used to fake different hash functions using the XOR technique
# described here: https://stackoverflow.com/a/19711615/157605 (thank
# you, Bill Dimm). The output of this hash function will be within
# [0, m).
def h(x, i):
    random.seed(i)
    return hash(x) % m ^ random.randint(0, m - 1)

# Estimates the Jaccard similarlity between two sets using the Fast
# MinHash algorithm.
def fast_min_hash(S, T, num_trials):
    num_matches = 0
    for i in range(num_trials):
        min_hash_s = min(h(x, i) for x in S)
        min_hash_t = min(h(x, i) for x in T)
        if min_hash_s == min_hash_t: num_matches += 1
    return num_matches / num_trials

# Run the experiments and report the estimates of JS.
T = [100, 200, 400, 800, 1600]
print('t\tJS_hat\tRun-time (s)')
for t in T:
    start_time = time.time()
    print(f'{t:,}\t{fast_min_hash(G3[0], G3[1], t):.4f}'
          f'\t{(time.time() - start_time):.3g}')
```

```
t       JS_hat  Run-time (s)
100     0.9300  0.712
200     0.9300  1.37
400     0.9275  2.83
800     0.9275  5.61
1,600   0.9250  11.2
```

Part B

In this particular case, since the documents aren't so big, I'd go with $t = 1600$ since it gave an estimate closer to the true value (calculated in *Problem 1-B*, 0.9191) than the other t-values. However, I can see that this t value wouldn't be practical for even documents just a bit bigger as the estimation would take too long.