

Assignment: Frequent Items

Qianlang Chen (u1172983)

CS 5140 Spring 2021

```
S1 = list(open('data/S1.txt').read().strip())
S2 = list(open('data/S2.txt').read().strip())
print(len(S1), len(S2))
```

4000000 5000000

Problem 1

Part A

```
# Runs the Misra-Gries Algorithm on stream `S` using `(k - 1)
# counters. Returns (the labels, the counts).
def misra_gries(S, k):
    L = [None] * (k - 1)
    C = [0] * (k - 1)
    for x in S:
        if x in L:
            C[L.index(x)] += 1
        else:
            if 0 in C:
                j = C.index(0)
                L[j] = x
                C[j] = 1
```

```

        else:
            for j in range(k - 1): C[j] -= 1
    return L, C

def misra_gries_summarize(name, S, k, must, might):
    m = len(S)
    L, C = misra_gries(S, k)
    print(f'Summary of {name}:\n'
          '    Frequent characters:\n'
          '\tchar\tfreq\tratio')
    for j in range(k - 1): print(f'\t{L[j]}\t{C[j]}\t{C[j] / m:.4f}')
    X_must = {L[j] for j in range(k - 1) if C[j] / m >= must}
    print('\n'
          f'    Characters that must\'ve occurred at least {must:.1%}'
          ' of the time:')
    print('\t' + ', '.join(X_must))
    X_might = {L[j] for j in range(k - 1) if C[j] / m + 1 / k >= might}
    if might <= must: X_might -= X_must
    print('\n'
          f'    Characters that might\'ve occurred at least {might:.1%}'
          ' of the time:')
    print('\t' + ', '.join(X_might))
    print('')

misra_gries_summarize('S1', S1, 8, .3, .2)
misra_gries_summarize('S2', S2, 8, .4, .5)

```

Summary of S1:

Frequent characters:

char	freq	ratio
j	1	0.0000
b	639426	0.1599
a	1840126	0.4600
c	240652	0.0602
o	1	0.0000
i	1	0.0000

v	1	0.0000
---	---	--------

Characters that must've occurred at least 30.0% of the time:

a

Characters that might've occurred at least 20.0% of the time:

b

Summary of S2:

Frequent characters:

char	freq	ratio
j	1	0.0000
b	757426	0.1515
a	2068126	0.4136
c	398652	0.0797
h	1	0.0000
w	1	0.0000
r	1	0.0000

Characters that must've occurred at least 40.0% of the time:

a

Characters that might've occurred at least 50.0% of the time:

a

Part B

```
import random

t = 6
k = 8
salts = [chr(random.randint(0, 127)) for _ in range(t)]

# Hashes an object `x` using the hash function h_`j` with output space
# [0, k).
```

```

def h(x, j): return hash(x + salts[j]) % k

# Runs Count-Min Sketch on stream `S` using `t` hashing algorithms.
# Returns the 2D-array of counters.
def count_min(S):
    C = [[0] * k for _ in range(t)]
    for x in S:
        for j in range(t): C[j][h(x, j)] += 1
    return C

def count_min_freq(C, x): return min(C[j][h(x, j)] for j in range(t))

def count_min_summarize(name, S, lookup_chars, all_chars, must, might):
    m = len(S)
    C = count_min(S)
    print(f'Summary of {name}:\n'
          '    Looked-up characters:\n'
          '\tchar\tfreq\tratio')
    for x in lookup_chars:
        f = count_min_freq(C, x)
        print(f'\t{x}\t{f}\t{f / m:.4}')
    X_must = {x for x in all_chars
               if count_min_freq(C, x) / m - 1 / k >= must}
    print('\n'
          f'    Characters that must\'ve occurred {must:.1%} of the'
          ' time:')
    print('\t' + ', '.join(X_must))
    X_might = {x for x in all_chars
               if count_min_freq(C, x) / m >= might}
    if might <= must: X_might -= X_must
    print('\n'
          f'    Characters that might\'ve occurred {might:.1%} of the'
          ' time:')
    print('\t' + ', '.join(X_might))
    print('')

```

```
count_min_summarize(
    'S1', S1, 'abc', 'abcdefghijklmnopqrstuvwxyz', .2, .2)
count_min_summarize(
    'S2', S2, 'abc', 'abcdefghijklmnopqrstuvwxyz', .2, .2)
```

Summary of S1:

Looked-up characters:

char	freq	ratio
a	2040026	0.51
b	799400	0.1998
c	440315	0.1101

Characters that must've occurred 20.0% of the time:

a

Characters that might've occurred 20.0% of the time:

Summary of S2:

Looked-up characters:

char	freq	ratio
a	2343428	0.4687
b	979400	0.1959
c	673637	0.1347

Characters that must've occurred 20.0% of the time:

a

Characters that might've occurred 20.0% of the time:

Part C

We could still perform the algorithms mostly the same way, except now we'd like to treat the words as case-insensitive (by casting all letters into lowercase) and split up the words (by buffering the characters until receiving one or more whitespace characters). The objects in the stream would now be strings individual characters, which our labels or hash functions would also need to support.

Part D

Count-Min Sketch provides lowerbounds to the frequencies for the queries, which is useful when we don't want to underestimate the frequency of anything or when underestimating frequencies have serious consequences.