Asmt 4: Frequent Items

Yulong Liang (u1143816)

March 14, 2018

1 Streaming Algorithms

A: (20 points) Misra-Gries Algorithm

- For stream S1, the output of the counters is as follows,

$$[1, 104715, 194715, 1, 147715, 1, 1, 0, 1]$$

1. **Three objects** *might* occur more than 20% of the time.
2. **No objects** *must* occur more than 20% of the time.

Reason: According to the formula $f_q - \frac{m}{k} \leq \hat{f}_q \leq f_q$, we can derive $\hat{f}_q \leq f_q \leq \hat{f}_q + \frac{m}{k}$, where $\frac{m}{k} = \frac{1000000}{10} = 100000$.
For element `104715`, we have $104715 \leq f_q \leq 204715$
For element `194715`, we have $194715 \leq f_q \leq 294715$
For element `147715`, we have $147715 \leq f_q \leq 247715$

- For stream S2, the output of the counters is as follows,

$$[1, 121429, 1, 161430, 231429, 0, 0, 0, 0]$$

1. **Two objects** *might* occur more than 20% of the time.
2. **One object** *must* occur more than 20% of the time.

Reason: According to the formula $f_q - \frac{m}{k} \leq \hat{f}_q \leq f_q$, we can derive $\hat{f}_q \leq f_q \leq \hat{f}_q + \frac{m}{k}$, where $\frac{m}{k} = \frac{1000000}{10} = 100000$.
For element `121429`, we have $121429 \leq f_q \leq 221429$
For element `161430`, we have $161430 \leq f_q \leq 261430$
For element `231429`, we have $231429 \leq f_q \leq 331429$

B: (20 points) Count-Min Sketch

- For stream S1, the estimated counts for objects a, b, and c are,

$$\hat{f}_a = 266799 \quad \hat{f}_b = 203000 \quad \hat{f}_c = 193378$$

1. **a & b** *might* occur more than 20% of the time.

Reason: According to the formula $f_q \leq \hat{f}_q \leq f_q + \varepsilon F_1$, we can derive $\hat{f}_q - \varepsilon F_1 \leq f_q \leq \hat{f}_q$, where we have

$$\varepsilon = \frac{2}{k} = \frac{2}{10} = 0.2$$

$$F_1 = \sum_j f_j = 1,000,000$$

For element `'a'`, We have $66799 \leq f_q \leq 266799$, which might occur more than 20% time with probably $1 - \delta = 31/32$.
For element `'b'`, We have $3000 \leq f_q \leq 203000$, which might occur more than 20% time with probably $1 - \delta = 31/32$.
For element `'c'`, We have $-6622 \leq f_q \leq 193378$.

- For stream S2, the estimated counts for objects a, b, and c are,

$$\hat{f}_a = 294848 \quad \hat{f}_b = 170000 \quad \hat{f}_c = 239349$$

1. **a & c** *might* occur more than 20% of the time.

Reason: According to the formula $f_q \leq \hat{f}_q \leq f_q + \varepsilon F_1$, we can derive $\hat{f}_q - \varepsilon F_1 \leq f_q \leq \hat{f}_q$, where we have

$$\varepsilon = \frac{2}{k} = \frac{2}{10} = 0.2$$

$$F_1 = \sum_j f_j = 1,000,000$$

For element `'a'`, we have $94848 \leq f_q \leq 294848$, which might occur more than 20% time with probably $1 - \delta = 31/32$.
For element `'b'`, we have $-30000 \leq f_q \leq 170000$
For element `'c'`, we have $39349 \leq f_q \leq 239349$, which might occur more than 20% time with probably $1 - \delta = 31/32$.

C: (5 points)

- Since the object is a word instead of a character, we read each word at a time from the stream and process it.
- For Misra-Gries Algorithm, store the words as strings in the label array. Since the words can be various, increase k value so that we can have better accuracy.
- For Count-Min Algorithm, create hash functions that map the words into different counters.

D: (5 points) Count-Min Sketch supports parallel counting with multi-core computing or distributed systems while Misra-Gries doesn't. With Count-Min Sketch, each thread can simultaneously read streams and perform the counting process with the same hash family. After all the threads finished their work, we can simply add every counter table up to get a total counter table. This feature makes Count-Min Sketch much more efficient when dealing with large-scale data.

2 Appendix: Codes

```python
def misraGries(s, k):

    c = [0] * (k-1)
    l = [''] * (k-1)
    m = 0

    while True:
        a = s.read(1)
        if not a:
            break

        m += 1
        found = False
        for i in range(k-1):
            if a == l[i]:
                c[i] += 1
                found = True
                break
        if not found:
            hasEmpty = False
            for i in range(k-1):
                if c[i] == 0:
                    l[i] = a
                    c[i] = 1
                    hasEmpty = True
                    break
            if not hasEmpty:
                for i in range(k-1):
                    c[i] -= 1

    return c, l, m

class CountMin:

    def __init__(self, k, t):
        self.k = k
        self.t = t
        self.salt = self.randomsalt(t)

    def randomsalt(self, t):
        salt = []
        for i in range(t):
            salt.append(''.join(random.choices(string.ascii_letters, k=10)))
        return salt
```

```python
    def h(self, i, x):
        hashstr = hashlib.sha1((x+self.salt[i]).encode('utf-8')).hexdigest()
        hashint = int(hashstr, 16) % self.k
        return hashint

    def count(self, s):
        s.seek(0)
        c = []
        for i in range(t):
            c.append([0] * k)
        f1 = 0

        while True:
            a = s.read(1)
            if not a:
                break
            f1 += 1
            for i in range(t):
                j = self.h(i, a)
                c[i][j] = c[i][j] + 1
        self.c = c
        self.f1 = f1
        return c, f1

    def query(self, char):
        minCount = float('inf')
        for i in range(self.t):
            j = self.h(i, char)
            if self.c[i][j] < minCount:
                minCount = self.c[i][j]
        return minCount
```