# SDSC3001 - Assignment 3

## Question 1

In [1]:
```
import random


def reservoir_sampling(k, stream):
    reservoir = []
    for i, item in enumerate(stream):
        if i < k:
            # Fill reservoir until we have k items
            reservoir.append(item)
        else:
            # Randomly decide whether to replace an item
            j = random.randint(0, i)  # Probability of keeping new item: k/(i+1)
            if j < k:
                reservoir[j] = item

    return reservoir
```

In [2]:
```
stream = range(1000)  # Simulate a data stream
sample_size = 5
result = reservoir_sampling(sample_size, stream)
print(f"Random sample of {sample_size} items:", result)
```

```
Random sample of 5 items: [405, 708, 311, 138, 409]
```

### Proof of correctness

Maintaining $k$ uniform samples from a streaming set guarantees at any time point $t \geq k$, the probability of any element already possessed from the sampling set is $\frac{k}{t}$, which can be proven inductively.

When $t = k$, the reservoir is filled with the first $k$ elements and each of these $k$ elements in the reservoir with probability 1.

Assume that after filling first $k$ element and processing $t-1$ elements, each element $x_i$ is in the reservoir with the probability of $\frac{k}{t-1}$. Then, considering the $t$-th element $x_t$, if the probability of $x_t$ being in the reservoir (not replaced by $x_t$, in other words) is $1-\frac{1}{t}$.

Therefore, the probability that $x_i$ is kept as a sample is the product of these two probability. $\frac{k}{t-1} \cdot (1-\frac{1}{t}) = \frac{k}{t}$.

## Question 2

## Part A

When an itemset $I$ has a size of $m$, there are $2^m - 1$ possible subsets. When mining top-$k$ most frequent patterns,

$$\begin{gathered} 2^m - 1 \leq k \\ 2^m \leq k + 1 \\ m \leq \log_2(k+1) \\ \therefore m = \lceil \log_2(k+1) \rceil \end{gathered}$$

## Part B

### b.1

To prove $f_S \geq \hat{f}_S \geq f_S - \frac{L}{C+1}$ for any pattern S, where:

- $f_S$ is the real support of pattern S
- $\hat{f}_S$ is the approximate support from Misra-Gries
- $L$ is total number of subsets across all transactions
- $C$ is number of counters

First, $f_S \geq \hat{f}_S$ is trivial since Misra-Gries algorithm never overestimates the frequency of any item, as it either increments a counter of decrements of all counters.

Next, for $\hat{f}_S \geq f_S - \frac{L}{C+1}$, each decrement operation in Miscra-Gries affects at most C+1 counters, so the total number of decrements cannot exceed $\frac{L}{C+1}$, and maximum error for any patter is bounded by total decrements. Therefore, $f_S - \hat{f}_S \leq \frac{L}{C+1}$, which is equal to $\hat{f}_S \geq f_S - \frac{L}{C+1}$.

### b.2

The inequality $f_{S^k} \geq \hat{f}^k \geq f_{S^k} - \frac{L}{C+1}$ where

- $S^k$ is the real k-th most frequent pattern
- $\hat{f}^k$ is the k-th largest approximate support

can be proven similarly to the previous example. First, $f_{S^k} \geq \hat{f}^k$ is trivial. Next, for $\hat{f}^k \geq f_{S^k} - \frac{L}{C+1}$, we know that $\hat{f}_{S^k} \geq f_{S^k} - \frac{L}{C+1}$ as $\hat{f}^k$ is the k-th largest approximate support and $\hat{f}_{S^k}$ is just one of the approximate supports. Therefore, similar to the previous example, each decrement operation is bounded by $\frac{L}{C+1}$.

### b.3

(1) If $f_S \geq f_{S^k}$, as we know the approximation error $f_S - \hat{f}_S \leq \frac{L}{C+1}$, and $\frac{L}{C+1}$ is a constant, $\hat{f}_S$ is always less than or equal to $f_{S^k}$ (= $\hat{f}_S \leq f_{S^k}, \therefore S \in A$).

(2) $\hat{f}_S \geq t = \hat{f}^k - \frac{L}{C+1}$ for any pattern in $A$. From b.1, we know that $f_S \geq \hat{f}_S$ and we know $\hat{f}^k \geq f_{S^k} - \frac{L}{C+1}$ from b.2.

- $\hat{f}_S \geq \hat{f}^k - \frac{L}{C+1} \ldots 1$
- $f_S \geq \hat{f}_S \ldots 2$
- $\hat{f}^k \geq f_{S^k} - \frac{L}{C+1} \ldots 3$

By combining inequalities 1 and 2, $f_S \geq \hat{f}^k - \frac{L}{C+1}$, and combining this with inequality 3, $f_S \geq f_{S^k} - \frac{2L}{C+1}$.

## b.4

In [3]:
```
import math
import sys
from collections import Counter
from itertools import combinations


class FrequenctPatterns:
    def __init__(self):
        self.transactions = []
        self.patterns = {}

        # def load_data(self):
        with open("trans.txt") as f:
            for line in f:
                transaction = list(map(int, line.split()))
                self.transactions.append(transaction)

        with open("patterns_Apriori.txt") as f:
            for line in f:
                key, value = line.strip().split(":")
                # key = tuple(sorted(map(int, key.split(","))))
                key = frozenset(map(int, key.split(",")))
                self.patterns[key] = int(value)

    def Misra_Gries(self, C, k=500):
        m = math.ceil(math.log2(k + 1))  # maximum size of patterns we need to cons
        L = 0  # number of subsets in transactions processed

        counter = Counter()  # pattern frequency counter

        # Upon receiving a_t, check if there is a counter for a_t
        for transaction in self.transactions:
            subsets = []  # transaction subsets
            for i in range(1, min(m, len(transaction)) + 1):
                # subsets.extend(tuple(sorted(c)) for c in combinations(transaction
```

```python
                subsets.extend(frozenset(c) for c in combinations(transaction, i))
            L += len(subsets)

            for subset in subsets:
                if subset in counter or len(counter) < C:
                    counter[subset] += 1
                else:
                    for key in list(counter.keys()):
                        counter[key] -= 1
                        if counter[key] == 0:
                            del counter[key]
        counter_sorted = counter.most_common()

        # threshold t = \hat{f}^k - \frac{L}{C + 1}
        threshold = counter_sorted[k - 1][1] - L / (C + 1)
        # filtered patterns: \hat{f}_s \ge t
        A = list(filter(lambda x: x[1] >= threshold, counter_sorted))

        min_sup = sys.maxsize
        min_pattern = frozenset()
        for item, _ in A:
            if self.patterns[item] < min_sup:
                min_sup = self.patterns[item]
                min_pattern = item

        return L, min_sup, min_pattern
```

In [4]:

```python
                # if there is one, increment the counter
                # if there isn't one,
                #     and there is at least one counter available, use an available
                #     and there is no available counter, decrement all counters by
                #
                # if subset in counter:
                #     counter[subset] += 1
                # else:
                #     if len(counter) < C:
                #         counter[subset] = 1
                #     else:
                #         for key in list(counter.keys()):
                #             counter[key] -= 1
                #             if counter[key] == 0:
                #                 del counter[key]
```

In [5]:

```python
frequent_patterns = FrequenctPatterns()
```

In [6]:

```python
for count in [500_000, 750_000, 1_000_000]:
    L, min_sup, min_pattern = frequent_patterns.Misra_Gries(count)
    print(f"C = {count}; value of {L = }, {min_sup = }, {min_pattern = }")
```

```
C = 500000; value of L = 59340244, min_sup = 1037, min_pattern = frozenset({829})
C = 750000; value of L = 59340244, min_sup = 1077, min_pattern = frozenset({77, 15
1})
C = 1000000; value of L = 59340244, min_sup = 1098, min_pattern = frozenset({24, 46
8})
```