■ Aim

To implement the Flajolet–Martin algorithm in Python to estimate the number of distinct elements (cardinality) in a data stream using probabilistic counting.

■ Theory

1. Introduction

When dealing with large datasets or data streams, it's inefficient to store and count every distinct element exactly. The Flajolet–Martin algorithm (FM algorithm) provides an approximate but memory-efficient way to estimate the number of distinct elements using hashing and bit-pattern analysis.

It works by:
• Hashing each element to a random binary number.
• Counting trailing zeros in the hash value.
• Keeping the maximum trailing zero count observed.
• Estimating distinct count ≈ 2^R / ϕ, where R = max trailing zeros, and ϕ = 0.77351.

Using multiple hash functions and averaging improves accuracy.

■ Python Code (Used in Experiment)

```
import hashlib
import random
import math

def sha1_hash_int(x, seed=0):
    """Return integer hash using SHA1 with a seed."""
    s = f"{seed}|{x}".encode('utf-8')
    h = hashlib.sha1(s).hexdigest()
    return int(h, 16)

def trailing_zeros(n):
    """Count trailing zeros in binary representation of n."""
    if n == 0:
        return 0
    tz = 0
    while (n & 1) == 0:
        tz += 1
        n >>= 1
    return tz

def flajolet_martin(stream, num_hashes=64, group_size=8):
    """Estimate number of distinct elements in stream using Flajolet–Martin."""
    assert num_hashes % group_size == 0, "num_hashes must be divisible by group_size"
    num_groups = num_hashes // group_size
    max_r = [0] * num_hashes

    for x in stream:
        for i in range(num_hashes):
            h = sha1_hash_int(x, seed=i)
            r = trailing_zeros(h)
            max_r[i] = max(max_r[i], r)
    phi = 0.77351
```

```
    estimates = [(2 ** r) / phi for r in max_r]

    group_estimates = []
    for g in range(num_groups):
        group_slice = estimates[g * group_size : (g + 1) * group_size]
        group_estimates.append(sum(group_slice) / len(group_slice))

    group_estimates.sort()
    mid = len(group_estimates) // 2
    if len(group_estimates) % 2 == 1:
        final_est = group_estimates[mid]
    else:
        final_est = (group_estimates[mid - 1] + group_estimates[mid]) / 2.0

    return int(final_est)

if __name__ == "__main__":
    data = [random.randint(0, 2000) for _ in range(5000)]
    actual = len(set(data))
    estimated = flajolet_martin(data, num_hashes=64, group_size=8)

    print("Flajolet-Martin Algorithm Demo")
    print(f"Stream length: {len(data)}")
    print(f"Actual distinct elements: {actual}")
    print(f"Estimated distinct elements: {estimated}")
```

■ Sample Output

```
Flajolet-Martin Algorithm Demo
Stream length: 5000
Actual distinct elements: 1987
Estimated distinct elements: 2013
```