Online Algorithms

Chiara Iannicelli

November 2023

1 Introduction

Online algorithms, also known as streaming algorithms, are designed to process data in a sequential manner, typically in the form of a data stream.
Unlike traditional batch algorithms that assume the entire dataset is available from the beginning, online algorithms work with data as it arrives, making them suitable for scenarios where the data is too large to be stored in memory or when it is impractical to process the entire dataset at once.

I'll show some key concepts and characteristics of online algorithms for data streams.

- **Sequential Processing:** Online algorithms process data sequentially, one item at a time, and make decisions or updates based on the current item without knowledge of future items.

- **Limited Memory:** Online algorithms are designed to work with limited memory resources. Since the entire dataset is not available, these algorithms maintain a compact representation of the information needed to make decisions.

- **Single Pass:** In most cases, online algorithms perform a single pass over the data stream. Once an item is processed, it is usually discarded to make room for the next incoming item.

- **Approximation and Estimation:** Online algorithms often provide approximate solutions or estimates due to the limited information available at any given time. This is acceptable in scenarios where exact solutions are not critical, and the focus is on efficiency and scalability.

- **Examples of Online Algorithms:**
  - Counting: One simple online algorithm is counting, where you maintain a count of certain events (e.g., frequency of items) as data arrives.
  - Sampling: Random sampling is an online algorithm that selects a subset of items from the data stream to approximate properties of the entire stream.
  - Sketches: Sketch-based algorithms use data structures called sketches to summarize information about the data stream in a compact form.
- **Challenges:** Dealing with the uncertainty introduced by processing incomplete information poses challenges for online algorithms. Balancing accuracy and resource efficiency is a key consideration.
- **Applications:** Online algorithms are widely used in various applications, including network monitoring, financial data analysis, sensor data processing, and real-time analytics.
- **Adaptivity:** Online algorithms often need to be adaptive, adjusting their behavior based on the characteristics of the incoming data. This adaptability is crucial for handling changing patterns and distributions in the stream.
- **Communication Complexity:** In distributed systems, online algorithms also consider communication complexity, as they may need to exchange information between different processing nodes.
- **Competitive Analysis:** Competitive analysis is a theoretical framework used to evaluate the performance of online algorithms. It compares the algorithm's solution to an optimal offline solution and measures the competitive ratio, which quantifies how well the online algorithm performs compared to the offline optimum.

2 Simulation

Let's consider an example of an online algorithm that estimates the frequency moments of a data stream using the Flajolet-Martin algorithm. The frequency moments are statistical measures of the distribution of item frequencies in a data stream.

```python
import random
import math

def flajolet_martin_algorithm(data_stream, num_hash_functions):
    def trailing_zeros(binary_repr):
        return len(binary_repr) - len(binary_repr.rstrip('0'))

    def hash_function(value, hash_num):
        # Simulate a hash function using a simple deterministic hash
        return (hash(value) + hash_num) % (2 ** 32)

    max_trailing_zeros = [0] * num_hash_functions

    for item in data_stream:
        for hash_num in range(num_hash_functions):
            hash_value = hash_function(item, hash_num)
            binary_repr = format(hash_value, '032b')  # 32-bit binary representation
            max_trailing_zeros[hash_num] = max(max_trailing_zeros[hash_num], trailing_zeros(binary_repr))

    estimated_counts = [2 ** trailing_zeros for trailing_zeros in max_trailing_zeros]

    return sum(estimated_counts) / num_hash_functions

# Simulating a data stream with random integers
data_stream_size = 10000
data_stream = [random.randint(1, 1000) for _ in range(data_stream_size)]

# Applying the Flajolet-Martin algorithm with 3 hash functions
num_hash_functions = 3
estimated_distinct_elements = flajolet_martin_algorithm(data_stream, num_hash_functions)

print(f"Estimated number of distinct elements: {int(estimated_distinct_elements)}")

```

Which will give us the following output:

```
Estimated number of distinct elements: 512
```

Figure 1: