"Online" Alghoritms (Data Streams)

Giacomo Babudri

December 2023

1 Introduction

Efficient processing over massive data sets has taken an increased importance in the last few decades due to the growing availability of large volumes of data in a variety of applications in computational sciences. In particular, monitoring huge and rapidly changing streams of data that arrive online has emerged as an important data management problem.

For this reason the advent of online algorithms has introduced a paradigm shift in how we process and analyze data. Online algorithms are designed to handle streaming data—information that is generated continuously and often in real-time. Unlike traditional algorithms that assume access to the entire dataset from the outset, online algorithms dynamically adapt to incoming data, making decisions on the fly and utilizing limited resources judiciously.

In particular typical streaming algorithms use space at most polylogarithmic in the length of the input stream and must have fast update and query times.

1.1 Examples

1. ***Frequency Counting***: Counting the frequency of elements in a data stream is a fundamental task in data stream processing. In scenarios where the data is continuously generated, maintaining an up-to-date count of element occurrences is crucial for various applications.

2. ***Moving Average***: Calculating the moving average of a stream of numerical data is essential for obtaining a smoothed trend and identifying patterns in dynamic data sets. This is particularly useful in scenarios where understanding the overall trend while considering recent data points is crucial.

3. ***Sliding Window Maximum***: Finding the maximum element in a sliding window of fixed size within a data stream is crucial for various applications,

such as real-time monitoring and trend analysis. This problem requires efficiently updating the maximum as new elements arrive and old elements leave the sliding window.

2 Alghoritms

2.1 Frequency Counting

The algorithm employs a data structure, such as a dictionary in this specific case, to keep track of the frequency of each unique element in the stream. As new elements arrive, the algorithm increments the corresponding count, ensuring an accurate representation of the element frequencies over time.

```
using System;
using System.Collections.Generic;

public class FrequencyCounting
{
    static void CountElements(List<int> stream)
    {
        Dictionary<int, int> counter = new Dictionary<int, int>();

        foreach (int element in stream)
        {
            if (!counter.ContainsKey(element))
                counter[element] = 1;
            else
                counter[element]++;

            Console.WriteLine("Element: " + element + ", Count: " + counter[element]);
        }
    }
}

```

Figure 1: Alghoritm for Frequeny Counting Problem

Example:

```
public static void Main()
    {
      List<int> dataStream = new List<int> { 1, 2, 1, 3, 2, 1, 4, 5, 3, 2, 4};
      CountElements(dataStream);
    }
```

```
Element: 1, Count: 1
Element: 2, Count: 1
Element: 1, Count: 2
Element: 3, Count: 1
Element: 2, Count: 2
Element: 1, Count: 3
Element: 4, Count: 1
Element: 5, Count: 1
Element: 3, Count: 2
Element: 2, Count: 3
Element: 4, Count: 2
```

Figure 2: Output

2.2 Moving Average

The moving average algorithm maintains a running window of the last N elements in the stream. As new data points arrive, the algorithm updates the average, providing a dynamic and real-time representation of the data's central tendency. This is useful for identifying trends and anomalies over time.

```
using System;
using System.Collections.Generic;
using System.Linq;

public class MovingAverage
{
    private int windowSize;
    private List<double> data;

    public MovingAverage(int windowSize)
    {
        this.windowSize = windowSize;
        this.data = new List<double>();
    }

    public void AddDataPoint(double value)
    {
        data.Add(value);

        if (data.Count > windowSize)
            data.RemoveAt(0);

        double average = data.Average();
        Console.WriteLine("New Data Point: " + value + ", Moving Average: " + average);
    }
}
```

Figure 3: Alghoritm for Movin Average Problem

Example:

```
public static void Main(){
```

```
    MovingAverage movingAverage = new MovingAverage(3);
    movingAverage.AddDataPoint(5.0);
    movingAverage.AddDataPoint(10.0);
    movingAverage.AddDataPoint(15.0);
    movingAverage.AddDataPoint(20.0);
}
```

```
New Data Point: 5, Moving Average: 5
New Data Point: 10, Moving Average: 7.5
New Data Point: 15, Moving Average: 10
New Data Point: 20, Moving Average: 15
```

Figure 4: Output

2.3 Sliding Window Maximum

The sliding window maximum algorithm utilizes a double-linkedList, in this specific case, to efficiently track indices of elements within the current window. By maintaining the linked list in a way that preserves the order of elements and their relevance to the current window, the algorithm can quickly identify the maximum element in each window as the stream evolves.

These online algorithms provide practical solutions to common challenges encountered in data stream processing, offering efficiency, adaptability, and real-time insights across different application domains.

```
using System;
using System.Collections.Generic;
using System.Linq;

public class SlidingWindowMaximum
{
    static void SlidingWindowMax(List<int> nums, int k)
    {
        List<int> result = new List<int>();
        LinkedList<int> window = new LinkedList<int>();
        for (int i = 0; i < nums.Count; i++)
        {
            while (window.Count > 0 && window.First.Value < i - k + 1)
                window.RemoveFirst();
            while (window.Count > 0 && nums[window.Last.Value] < nums[i])
                window.RemoveLast();

            window.AddLast(i);

            if (i >= k - 1)
                result.Add(nums[window.First.Value]);
        }

        Console.WriteLine("Sliding Window Maximum: " + string.Join(", ", result));
    }
}
```

Figure 5: Alghoritm for Sliding Window Maximum Problem

Example:

```
public static void Main()
    {
        List<int> dataStream = new List<int> { 1, 3, -1, -3, 5, 3, 6, 7 };
        int k = 3;
        SlidingWindowMax(dataStream, k);
    }
```

```
Sliding Window Maximum: 3, 3, 5, 5, 6, 7
```

Figure 6: Output