Th7: Data Stream Algorithms

Federico Detomaso

1 Introduction

In the data stream scenario, input arrives very rapidly and there is limited memory to store the input. Algorithms have to work with one or few passes over the data, space less than linear in the input size or time significantly less than the input size. In the past few years, a new theory has emerged for reasoning about algorithms that work within these constraints on space, time, and number of passes. Some of the methods rely on metric embeddings, pseudo-random computations, sparse approximation theory and communication complexity. The applications for this scenario include IP network traffic analysis, mining text message streams and processing massive data sets in general. Researchers in Theoretical Computer Science, Databases, IP Networking and Computer Systems are working on the data stream challenges.

2 Algorithms

These algorithms are particularly useful when dealing with massive datasets that cannot fit into memory, and the goal is to process the data efficiently without storing it entirely. Here are some ideas and examples for online algorithms in the context of data streams.

2.1 Counting Elements

- Problem: Count the frequency of elements in a data stream.
- Idea: Use a hash table (or a data structure like Count-Min Sketch) to estimate the count of each element.

Algorithm 1 Counting Elements in a Data Stream

Initialize an empty dictionary *counts*
Let *data_stream* be the input stream of elements
for each *element* in *data_stream* **do**
  if *element* exists in *counts* **then**
    $counts[element] \mathrel{+}= 1$
  else
    $counts[element] \leftarrow 1$
  end if
end for
Output: *counts*, containing the count of each element

2.2 Distinct Elements

- Problem: Count the number of distinct elements in a data stream.
- Idea: Use a set to keep track of unique elements.

Algorithm 2 Counting Distinct Elements in a Data Stream

```
Initialize an empty set distinct_elements
Let data_stream be the input stream of elements
for each element in data_stream do
    Add element to distinct_elements
end for
Output: The number of elements in distinct_elements
```

2.3 Sliding Window

- Problem: Maintain statistics (e.g., sum, average) over a fixed-size sliding window.
- Idea: Use a queue to keep track of the elements in the current window.

Algorithm 3 Sliding Window Average

```
Initialize an empty deque window with a maximum size of window_size
Initialize current_sum to 0
Let data_stream be the input stream of elements
for each element in data_stream do
    Append element to window
    current_sum += element
    if size of window is equal to window_size then
        average ← current_sum / window_size
        Print "Current Window Average: average"
    end if
end for
```

2.4 Reservoir Sampling

- Problem: Randomly sample k items from an unknown and large stream of items.
- Idea: Maintain a reservoir of k items and update it as new items arrive.

Algorithm 4 Reservoir Sampling

```
Let data_stream be the input stream of elements
Initialize an empty array reservoir of size k
for each i, element in enumerate data_stream do
    if i < k then
        reservoir[i] ← element
    else
        j ← random integer between 0 and i
        if j < k then
            reservoir[j] ← element
        end if
    end if
end for
Output: reservoir, containing the sampled elements
```

References

[1] https://twiki.di.uniroma1.it/pub/Ing_algo/DiarioLezioni/muthu.pdf

[2] https://en.wikipedia.org/wiki/Streaming_algorithm