Algorithmic Methods of Data Mining - Assignment 2

Maksad Donayorov

October 18, 2018

1. **Problem:** *Bag of words:*

2. **Problem:** $\mathcal{O}(\log w \;\; \log n)$:
   Before analyzing the complexity and responding to the question it could be nice to see an algorithm that finds the *max* number in an array. Such algorithm can be easily written in python, that gives us a clear picture of how the computation is done:

```
def get_max(arr):
  max_num = -1
  for el in arr:
    if max_num < el:
      max_num = el
  return max_num
```

The best case for finding the *max* element is when the first element is bigger than all of the other elements of X, which has complexity of $\mathcal{O}(1)$. And the worst case is when the last element is the largest and all the consecutive numbers proceed from small to large. This case has a complexity of $\mathcal{O}(n)$. The tricky part is the average case, where we have to compute the complexity for *the number of times that max is assigned to an element.*

Assuming that $X[1, 2, 3...m]$ is drawn independently and uniformly at random from the interval $(0, 1)$, the expected value will be:

$$E[x] = \sum_{i=1}^{m} Pr(X_i)$$

Where $Pr(X_i)$ is the probability of the *i-th* element being the *max* element in $X[1, 2, 3...m]$. To have a better intuition of what could be the probability of *i-th* element we can have a look at different size of m:

$$Pr(x_1) = 1$$
$$Pr(x_2) = \frac{1}{2}$$
$$Pr(x_3) = \frac{1}{3}$$
$$...$$
$$Pr(x_m) = \frac{1}{m}$$

Looking at this we can infer that the expected value can be calculated as:

$$E[x] = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{m} = \sum_{i=1}^{m} \frac{1}{i} \approx \log m$$

Consequently, we can conclude that $max = X[i]$ will be executed $\mathcal{O}(\log m)$ times, on expectation.

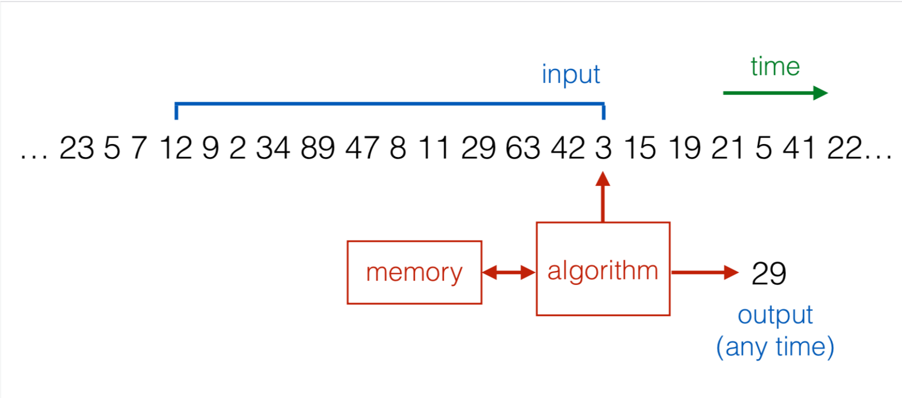

Figure 1: Representation of sliding window: *"CS-E4600 - slide set 7"*.

Using the same logic and intuition we can argue that the *priority sampling for sliding window* will use $\mathcal{O}(\log w \log n)$, where $\log w$ is the length of the sliding window and $\log n$ is the space for numbers to be stored. The arguments for our prove are the criteria that:

- in any given window each item has equal chance to be selected as a random sample
- each removed element has a larger element proceeding it
- at any given point we expect the space efficiency $\mathcal{O}(w)$ with maximal elements
- and finally, maintaining list of maximal elements requires $\mathcal{O}(\log w)$ time

3. **Problem:** *Reservoir algorithm for sampling 1 element in a data stream:*

   3.1. **Question**: *explain k-sample in a data stream is uniform:*
   The meaning of $k-sample$ in a data stream being uniform is: for randomly choosing a sample of k items from a data stream, at any given time each element of the data stream have equal probability of being sampled.

   3.2. **Question**: *value of the probability p:*
   The value of the probability will be the length of $k - sample$ divided by the value of how many elements at current time we have seen, t. Which is $\frac{k}{t}$.

   3.3. **Question**: *prove that value of p gives uniform samples:*
   Let's suppose that our $k - sample$ has 5 elements. When sixth element arrives $i = 6$ and $t = [1, 2...6]$, each element is kept with the probability $\frac{5}{6}$, which is:

$$(1)(\frac{1}{6} + (\frac{5}{6})(\frac{4}{5})) = \frac{1}{6} + \frac{4}{6} = \frac{5}{6}$$

   when the seventh element arrives $i = 7$ and $t = [1, 2...7]$, the seventh element is kept with the probability $5/7$ and each of the previous 6 elements are also kept with the same probability. Following that logic, we can prove by induction that when there are n elements, each one is kept with the probability $5/n$. Consequently, our general notation will be:

$$P(k_{sample}|t_{elements\ seen}) = \frac{k}{t}$$

4. **Problem:** *Resort to sampling of good items:*