

IOANNA PAPAGIANNI

## RESERVOIR ALGORITHM DESCRIPTION

In order to sample  $K$  items from an unknown number of  $N$  elements, it is inefficient to keep all data in memory (space complexity  $O(N)$ ) and then run an algorithm to uniformly choose  $K$  of them with probability  $K/N$ . The  $N$  number might be extremely huge.

The idea, is to incrementally process the data as they come through streams, in our case, reading line by line, so that everytime we calculate its probability of being chosen.

Our algorithm, takes as an input, the number of samples ( $K$ ) we are about to generate and an input file to be examined.

Everytime a new line comes, it calculates the possibility by dividing  $K$  with the current number of lines reached, keeping the  $\min(1, P(K / \text{number of lines}))$  because in early steps of the algorithm, the number of  $K$  outweighs the number of lines and the possibility should not exceed  $[0, 1]$  interval.

Iff the possibility is greater than a random float number between  $[0.0, 1.0)$ , it is added to the list. When the list has reached length  $K$  and the above condition is satisfied, the new item is randomly taking the place of another in the list. Therefore, the possibility of the first ones staying in the list, decreases and all items have  $K/N$  possibility to be in the sampled list.

The space complexity is  $O(K)$  and the time complexity is  $O(N)$ . It scales well even for big data streams.

## PROOF WITH INDUCTION

We have:  $1 \leq i \leq N$  and  $K \geq 1$

**Base case:**  $i = N = 1$  and  $K = 1$  then:

$K/N = 1$  (100% probability of being chosen).

While  $K \geq i$ , the probability is:  $\min(1, K/N)$  (items have 100% probability of being chosen).

**Induction step:**

When  $K < i$ , let's suppose that the probability of choosing any  $i$ -st item is equal to  $K/i$ .

The probability of the  $i+1$  item being picked is  $K / (i+1)$ .

The probability of picking a previous  $i$ -item was  $K/i$ , and the probability of it staying that way is  $i / (i+1)$ .

$(K/i) * (i / (i+1)) = K / (i+1)$ .

We proved that each of the  $i+1$  items now, have  $K / (i+1)$  probability of being selected and consequently, when  $i$  reaches  $N$  the probability is  $K/N$ .

## CODE INSTRUCTIONS, STRUCTURE AND RESULTS

This algorithm is implemented in Python3 and runs with the following command line format in Terminal:

**python3 reservoir.py sample\_number\_K < input\_file.txt**

*sample\_number\_K*: any positive integer number  
*input\_file.txt*: any file.txt

Where bold letters, standard format.

Note that, it is necessary for the files to be **in the same exact folder** and run the above Terminal commands in that exact folder.

The structure of the code is simple.

Importing `sys`, in order to interact with the host system, and `random` to generate some pseudo-random numbers in order to make some comparisons.

`main()` is responsible for basic operations:

- calls `check_input()` function
- opens the input file to read line by line using `__next__` python iterator
- calls `reservoir_sampling()` function
- calls `print_rows()` function

`check_input()` function:

- checks if the format of the command line is correct and if it is not, it terminates the program.
- checks if `K` is zero or negative number and if it is, terminates there (none results obviously).

`reservoir_sampling()` function:

- checks if a random float number in the interval `[0.0, 1.0)` is less than `K` divided by the row number so far, so as to add it in the `selected_rows[]` list.  
 The use of `min()` built-in function, is to be mathematically accurate, possibilities range from 0 to 1. If the list is empty, it inserts the row at the end and if the list has reached the `K` limit, it randomly chooses a position in the interval `[0, K-1]` to place it.
- returns everytime the new selected rows in a list.

`print_rows()` function:

- prints the final sampled rows (following the order which they are kept in the list).

To see how the reservoir works, we used an extra `print()` everytime `reservoir_sampling()` function returned a list (comment in line 56).

We run the same experiment three times, setting all times `K=3` and input file "input.txt" ( the file contains 12 sentences (rows) separated with newline ("space" button) after every dot and no empty lines among them.

Note that, if empty lines existed in the file, it would also work. We did not put empty lines in the file in order to have a better visualization of the results. It also works for empty files – no results are returned.