Data Structure I: Tutorial 8

Bloom Filter

- Given a set $S \subseteq U$ of size n
- We use a binary array T of length m initialized by the value false
- and k hash functions $h_1, \ldots, h_k : U \to M$ where $M = \{0, \ldots, m-1\}$
- An element $x \in S$ is inserted by setting $T[h_i(x)] = true$ for all $i = 1, \ldots, k$

Exercise 1.

- *Can we recognize that an element was inserted into our Bloom filter?*
- *Can we recognize that an element was NOT inserted into our Bloom filter?*
- *When Bloom filters can be useful?*

Approach for the assignment

- You have to implement a Bloom filter
- Use a hash table from the standard library (e.g. Dict, unordered_map) to store candidates for duplicates
- It is necessary to use the data generator twice
- First time, for every element x given by the generator:
  - If it is possible that an element x is stored in the Bloom filter, insert x into the hash table for candidates
  - Insert x into the Bloom filter
- Second time
  - Count the number of occurrences of all candidates in the hash table
  - On the second occurrence of an element, store it in the resulting array

Hints for the assignment:

- You do not have enough memory to store all elements in the hash table, so you have store only candidates given by the Bloom filter
- Properly calculate the size of Bloom filter based on the given memory limits
- Array [False] * 2**20 requires approximately 8 MB since Python stores it as an array of pointers to one value False. Use bytearray instead
- Read carefully the documentation of bytearray and distinguish the terms bit and byte
- In Python, do not import numpy or other libraries consuming more memory to load than available

- It is forbidden to store duplicates in the submitted file

Counting filters:

- Our goal is to modify Bloom filters to be able to delete an element
- We replace the binary array T by an array C of m small counters (4-bits counters are usually sufficient)
- Operation Insert increases by one all counters $C[h_1(x)], \dots, C[h_k(x)]$
- Operation Delete decreases all these counters by one

Definition 2. *Tabular hashing*

- *Assume that* $u = 2^w$ *and* $m = 2^l$ *and* w *is a multiple of an integer* d
- *Binary code of* $x \in U$ *is split to* d *parts* $x^0, \dots, x^{d-1}$ *by* $\frac{w}{d}$ *bits*
- *For every* $i \in [d]$ *generate a totally random hashing function* $T_i : [2^{w/d}] \to M$
- *Hashing function is* $h(x) = T_0(x^0) \oplus \dots \oplus T_{d-1}(x^{d-1})$

$\oplus$ *denotes bit-wise exclusive or (XOR).*

Theorem 3. *Tabular hashing is 3-independent, but it is not 4-independent.*