Probabilistic Data Structures

Big Data Management and Governance

Prof. Giovanni Simonini, Giovanni Malaguti

University of Modena and Reggio Emilia

November 19, 2025

Details of the Lab (1)

- We will implement Bloom Filter, Cuckoo Filter and Count-min Sketch
- Simple implementations, no optimization and we won't cover all the details
- Goal: to obtain new data structures that support creation, insert and some kind of search operations
- At home, attempt to implement other operations (e.g. delete), extend our simple implementations with other known optimizations following original papers (e.g. use more than one bucket for cuckoo filter).

Details of the Lab (2)

- Clone (or update) the repository `https://github.com/Stravanni/bdm.git`
- There are four files in bdm/lab/prob-data-struct: bloom_filter.py, cuckoo_filter.py, count_min_sketch.py and utils.py
- We will implement algorithms and data structures inside the first three files from scratch. In utils.py there are helper functions for visualization and experiments

Details of the Lab (3-*nix/Mac)

Open a shell and move to the current lab folder.

```
$ cd /path/to/the/cloned/repo
```

There create the Python virtual environment. You can use any python environment manager (conda, uv, poetry, ...). Here, for simplicity, we will use the Python venv module:

```
(skiplist-hnsw)$ python -m venv .venv
```

Activate the environment:

```
(skiplist-hnsw)$ source .venv/bin/activate
```

Install the required packages:

```
(skiplist-hnsw)$ pip install -r requirements.txt
```

Details of the Lab (3-Windows)

Open a command-line prompt (Powershell). Then, move to the current lab folder.

```
$ cd path\to\the\cloned\repo
```

There create the Python virtual environment. You can use any python env manager (conda, uv, poetry, ...). Here, for simplicity, we will use the Python venv module:

```
(skiplist-hnsw)$ python -m venv .venv
```

Activate the environment:

```
(skiplist-hnsw)$ .venv/Scripts/activate
```

Install the required packages:

```
(skiplist-hnsw)$ pip install -r requirements.txt
```

Details of the Lab (4)

In the folder bdm/lab/prob-data-struct/data there is the file urls.zip

Extract the CSV from the archive in the same folder. We will use this just for a short demonstration of the data structures at the end of the lab

Bloom Filter

Bloom Filter: Definition

- Data structure for set membership introduced in 1970 [1]
- Each inserted item is hashed with multiple functions; the digests are used to set to 1 the bits of a bitarray
- At query time, an item is hashed with the same functions: if all the required bits are 1, the item *may be* contained. Otherwise, the item is *not* contained.
- More scalable than hash-sets and other classical data structure, both for space and time complexity.

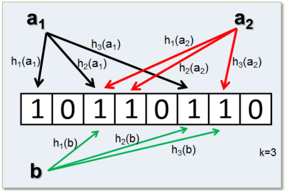

Figure 1: Example of Bloom Filter

Bloom Filter: False-Positive Rate Calibration

When dealing with Bloom Filters, the size of the bitarray m and the number of functions k have to be calibrated with respect to the expected number of items and the desired false positive rate.

$$m = -\left\lceil \frac{n \ln(\epsilon)}{(\ln 2)^2} \right\rceil \tag{1}$$

$$k = \left\lceil \frac{m \ln 2}{n} \right\rceil \tag{2}$$

Bloom Filter: Insert and Check

Algorithm 1 Insert (x)

Require: H set of hash functions, ba bitarray storing the filter, x item to insert
Ensure: item x inserted into the filter

```
for h in H do
    i = h(x) (mod m)
    ba[i] = 1
end for
```

Algorithm 2 Check (x)

Require: H set of hash functions, ba bitarray storing the filter, x query item

```
for h in H do
    i = h(x) (mod m)
    if ba[i] ≠ 1 then
        return False
    end if
end for
return True
```

Cuckoo Filter

Cuckoo Filter: Definition

- Data structure for set membership instroduced in 2014 [3]
- In contrast to Bloom Filters, this data structure supports also deletion
- A Bloom Filter is a list of B buckets, each of them storing M fingerprints (in our implementation, only one)

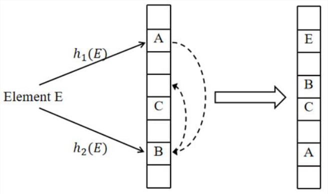

Figure 2: Example of Cuckoo Filter

Cuckoo Filter: Insert

Algorithm 3 Insert

Require: Item x to insert, list of buckets *buckets* of size B
Ensure: Item inserted or failure if filter is full

$f = \text{fingerprint}(x)$
$i_1 = \text{hash}(x) \pmod B$
$i_2 = \text{hash}(f) \oplus \text{hash}(i1) \pmod B$
if $buckets[i_1]$ or $buckets[i_2]$ are empty **then**
  add f to that bucket
  return `Done`
end if
i = randomly pick i_1 or i_2
for $n = 0; n < MaxNumKicks; n++$ **do**
  swap f with the content in bucket i
  $i = i \oplus \text{hash}(f) \pmod B$
  if $buckets[i]$ is empty **then**
    add x to $buckets[i]$
    return `Done`
  end if
end for
return `Failure`

Cuckoo Filter: Contains and Delete

Algorithm 4 Check (x)

$f = \text{fingerprint}(x)$
$i_1 = \text{hash}(x) \pmod{B}$
$i_2 = i_1 \oplus \text{hash}(f) \pmod{B}$
if $buckets[i_1]$ or $buckets[i_2]$ has f **then**
    return `True`
end if
return `False`

Algorithm 5 Delete (x)

$f = \text{fingerprint}(x)$
$i_1 = \text{hash}(x) \pmod{B}$
$i_2 = i_1 \oplus \text{hash}(f) \pmod{B}$
if $buckets[i_1]orbuckets[i_2]hasf$ **then**
    Remove a copy of f from this bucket
    return `True`
end if
return `False`

Count-Min Sketch

Count-Min Sketch: Definition

- Data structure to compute *upper*-bounds of occurences of items in a data stream, introduced in 2005 [2]
- A CM sketch is a table M of depth d and width w, initialized with all zeros.
- An item is hashed with d different functions: the i-th function gives a position in table row i, and the value there is increased.
- At query time, is returned the minimum of the d possible counts given by the set of functions for the query item.

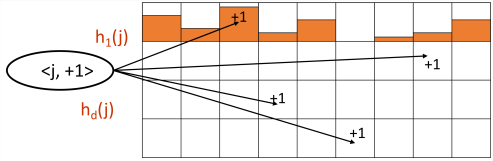

Figure 3: Example of Count-Min Sketch

Count-Min Sketch: Insert and Check

Algorithm 6 Insert (x)

Require: H list of hash functions, M table storing $[dxw]$ counts, x item to insert
Ensure: M counts updated with respect to x hash values

for i in $1..d$ **do**
    $j = H[i](x) \pmod{w}$
    $M[i][j] += 1$
end for

Algorithm 7 Check (x)

Require: x query item
Ensure: An upper-bound of the occurences of x in the inserted data

$values = \emptyset$
for i in $1..d$ **do**
    $j = H[i](x) \pmod{w}$
    $values = values \cup H[i][j]$
end for
return $min(values)$

References

BLOOM, B. H.
Space/time trade-offs in hash coding with allowable errors.
Commun. ACM 13, 7 (July 1970), 422–426.

CORMODE, G., AND MUTHUKRISHNAN, S.
An improved data stream summary: the count-min sketch and its applications.
J. Algorithms 55, 1 (Apr. 2005), 58–75.

FAN, B., ANDERSEN, D. G., KAMINSKY, M., AND MITZENMACHER, M. D.
Cuckoo filter: Practically better than bloom.
In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2014), CoNEXT '14, Association for Computing Machinery, p. 75–88.