PROBABILISTIC DATA STRUCTURES AND ALGORITHMS

FOR BIG DATA APPLICATIONS

ANDRII GAKHOV

gakhov

Probabilistic Data Structures and Algorithms
for Big Data Applications

1st edition, 2019

Bibliographic information published by the Deutsche Nationalbibliothek:

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the Internet at http://dnb.dnb.de.

© 2019 Andrii Gakhov

All rights reserved; no part of this book may be reproduced or transmitted by any means, electronic, mechanical, photocopying or otherwise, without the prior permission of the author. All product names and trademarks referred to are the property of their respective owners.

The paperback edition is printed and published by
BoD — Books on Demand GmbH
22848 Norderstedt
Germany

The publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained in this book.

ISBN (paperback): 978-37-48190-48-6

ASIN (ebook): B07MYKTY8W

To my wife Larysa
and my son Gabriel.

Table of Contents

Preface

Big data is characterized by three fundamental dimensions: *Volume*, *Velocity*, and *Variety*, **The Three V's of Big Data**. The *Volume* expresses the amount of data, *Velocity* describes the speed at which data is arriving and being processed, and *Variety* refers to the number of types of data.

The data could come from anywhere, including social media, various sensors, financial transactions, etc. IBM has stated[1] that people create **2.5 quintillion** bytes of data **every day**, this number is growing constantly and most of it cannot be stored and is usually wasted without being processed. Today, it is not uncommon to process terabyte- or petabyte-sized corpora and gigabit-rate streams.

On the other hand, nowadays every company wants to fully understand the data it has, in order to find value and act on it. This led to the rapid growth in the Big Data Software market. However, the traditional technologies which include data structures and algorithms, become ineffective when dealing with Big Data. Therefore,

[1]What Is Big Data? `https://www.ibm.com/software/data/bigdata/what-is-big-data.html`

many software practitioners, again and again, refer to computer science for the most appropriate solutions and one option is to use probabilistic data structures and algorithms.

Probabilistic data structures is a common name for data structures based mostly on different hashing techniques. Unlike regular (or deterministic) data structures, they always provide approximated answers but with reliable ways to estimate possible errors. Fortunately, the potential losses and errors are fully compensated for by extremely low memory requirements, constant query time, and scaling, the factors that become essential in Big Data applications.

About this book

The purpose of this book is to introduce technology practitioners which includes software architects and developers, as well as technology decision makers to probabilistic data structures and algorithms. Reading this book, you will get a theoretical and practical understanding of probabilistic data structures and learn about their common uses.

This is not a book for scientists, but to gain the most out of it you will need to have basic mathematical knowledge and an understanding of the general theory of data structures and algorithms. If you do not have any "computer science" experience, it is highly recommended you read *Introduction to Algorithms* by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (MIT), which provides a comprehensive introduction to the modern study of computer algorithms.

While it is impossible to cover all the existing amazing solutions, this book is to highlight their common ideas and important areas of application, including membership querying, counting, stream mining, and similarity estimation.

Organization of the book

This book consists of six chapters, each preceded by an introduction and followed by a brief summary and bibliography for further reading relating to that chapter. Every chapter is dedicated to one particular problem in Big Data applications, it starts with an in-depth explanation of the problem and follows by introducing data structures and algorithms that can be used to solve it efficiently.

The first chapter gives a brief overview of popular hash functions and hash tables that are widely used in probabilistic data structures. Chapter 2 is devoted to approximate membership queries, the most well-known use case of such structures. In chapter 3 data structures that help to estimate the number of unique elements are discussed. Chapters 4 and 5 are dedicated to important frequency- and rank-related metrics computations in streaming applications. Chapter 6 consists of data structures and algorithms to solve similarity problems, particularly — the nearest neighbor search.

This book on the Web

You can find errata, examples, and additional information at https://pdsa.gakhov.com. If you have a comment, technical question about the book, would like to report an error you found, or any other issue, send email to pdsa@gakhov.com.

In case you are also interested in Cython implementation that includes many of the data structures and algorithms from this book, please check out our free and open-source Python library called PDSA at https://github.com/gakhov/pdsa. Everybody is welcome to contribute at any time.

About the author

Andrii Gakhov is a mathematician and software engineer holding a Ph.D. in mathematical modeling and numerical methods. He has been a teacher in the School of Computer Science at V. Karazin Kharkiv National University in Ukraine for a number of years and currently works as a software practitioner for ferret go GmbH, the leading community moderation, automation, and analytics company in Germany. His fields of interests include machine learning, stream mining, and data analysis.

The best way to reach the author is via Twitter @gakhov or by visiting his webpage at https://www.gakhov.com.

Acknowledgments

The author would like to thank Asmir Mustafic, Jean Vancoppenolle, and Eugen Martynov for the contribution to reviewing this book and for their useful recommendations. Big gratitude to academia reviewers Dr. Kateryna Nesvit and Dr. Dharavath Ramesh for their invaluable suggestions and remarks.

Special thanks to Ted Dunning, the author of the t-digest algorithm, for a very precise review of the corresponding chapter, the insightful questions, and discussion.

Finally, thanks to all the people who provided feedback and helped make this book possible.

1 Hashing

Hashing plays the central role in probabilistic data structures as they use it for randomization and compact representation of the data. A *hash function* compresses blocks of input data of an arbitrary size by generating an identifier of a smaller (and in most cases fixed) size, called the *hash value* or simply the *hash*.

The choice of hash functions is crucial to avoid bias. Although the selection decision is mostly based on the input and particular use cases, there are certain common properties that a hash function should fulfill in order to be applicable for hash-based selection.

Hash functions compress the input, therefore, cases where they generate the same hash values for two different blocks of data are unavoidable and known as *hash collisions*.

In 1979 J. Lawrence Carter and Mark Wegman proposed the *universal hash functions* whose mathematical properties can guarantee a low expected number of collisions, even if the input data are chosen randomly from the *universe*.

The universal hash functions family H maps elements of the universe to the range $\{0, 1, \ldots, m-1\}$ and guarantees that by randomly picking a hash function from the family the probability of collisions is limited:

$$\Pr\Big(h(x) = h(y)\Big) \leq \frac{1}{m}, \text{ for any } x, y : x \neq y. \tag{1.1}$$

Thus, the random choice of a hash function from the family with property (1.1) is precisely the same as choosing an element uniformly at random.

An important universal hash functions family, designed to hash integers, can be defined as

$$h_{\{k,q\}}(x) = ((k \cdot x + q) \bmod p) \bmod m, \tag{1.2}$$

where k and q are randomly chosen integers modulo p with $k \neq 0$. The value of p should be selected as a prime $p \geq m$, and the common choice is to take one of the known Mersenne prime numbers, e.g., for $m = 10^9$ we choose $p = \mathrm{M}_{31} = 2^{31} - 1 \approx 2 \cdot 10^9$.

Many applications can use the simpler version of the family (1.2):

$$h_{\{k\}}(x) = (k \cdot x \bmod p) \bmod m, \tag{1.3}$$

this is only approximately universal, but still provides a good probability of collisions smaller than $\frac{2}{m}$ in expectation.

However, the above families of hash functions are limited to integers, that is not enough for most practical applications which require to hash variable-sized vectors and are in demand of fast and reliable hash functions with certain guaranteed properties.

There are many classes of hash functions used in practice and the choice mainly depends on their design and particular use. In the current chapter we provide an overview of popular hash functions and simple data structures that are prevalent in various probabilistic data structures.

1.1 Cryptographic hash functions

Practically, *cryptographic hash functions* are defined as fixed mappings from variable input bit strings to fixed length output bit strings.

As stated previously, hash collisions are unavoidable, but a secure hash function is required to be *collision resistant*, meaning that it should be hard to find collisions. Of course, a collision can be found accidentally or computed in advance. This is why such a class of functions always requires mathematical proofs.

Cryptographic hash functions are very important in cryptography and are used in many applications such as digital signatures, authentication schemas, and message integrity.

There are three main requirements that cryptographic hashes are expected to satisfy:

- *Work factor* — to make brute force inversion hard, a cryptographic hash should be computationally expensive.
- *Sticky state* — cryptographic hash should not have a state in which it can stick for a plausible input pattern.
- *Diffusion* — every output bit of a cryptographic hash should be an equally complex function of every input bit.

Theoretically, cryptographic functions can be further divided into *keyed hash functions*, that use a secret key, and *unkeyed hash functions*,