Advanced Data Structures: Final Report

Steil, Patrick ✉
2513240

1 Introduction

In this work, different data structures are implemented that can efficiently answer two types of queries: predecessor data structures and range-minimum queries. For this purpose, the *Elias-Fano coding* and several "Block-Lookup" data structures are implemented and tested. The project is part of the lecture "Advanced Data Structures" at KIT in the summer semester 2023 with Dr. Florian Kurpicz.

2 Algorithms and Data Structures

Predecessor data structures

Predecessor data structures are special data structures that were developed to efficiently solve the problem of predecessor access. The predecessor access problem is to find the predecessor (the element with the highest value that is less than a given element) in a set of elements. Without additional memory consumption, elements can be stored sorted or in a binary search tree, whereby a predecessor query can subsequently be answered with a runtime of $\mathcal{O}(\log(n))$. However, this is too slow in practice, which is why techniques such as *x/y-fast-tries* [**?**] or *Elias-Fano encoding* [**?**] have been developed. The idea of the last technique is to divide the representation of numbers into two parts, on the one hand the *most significants* bits with a length of $\lceil \log(n) \rceil$ and on the other hand the remaining bits. This way, all numbers with the same *most significants* bits are "grouped" during a query, which means that a query only has to look at one "group" of numbers. However, this may result in all numbers having to be scanned if the numbers are not favourable.

Range minimum queries

Range minimum queries (RMQ) refer to a special type of search operation in data structures where the minimum value within a given interval (range) of elements in a set is to be found. The goal is to efficiently find the smallest value among a subset of elements. Analogous to the predecessor data structures, there are trade-offs between additional memory and runtime (the different pre-computation times can be neglected for static data). To be able to answer RMQ quickly, one can use $\Theta(n^2)$ additional memory to calculate all possible queries in advance and thus obtain a constant query time (a lookup). To reduce the impractical quadratic memory, one can now only precompute queries of certain ranges. If the size of this range is cleverly chosen, one obtains constant query time and down to only linear additional memory.

3 Implementation

3.1 Elias-Fano

For an efficient Elias-Fano implementation, one needs a bit-vector data structure that can efficiently answer SELECT query. The actual Elias-Fano data structure then consists of two bit vectors: *upperBits* and *lowerBits*. The former encodes in $2n + 1$ bits the *most significants bits* (MSB) of the numbers, and in *lowerBits* the remaining bits (LSB) of the numbers are stored straightforward. As a note: In the script, the first $\lceil \log(n) \rceil$ bits were defined as MSB, but this definition caused problems with e.g. the following numbers:

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$0, 1, 10, 13, 32, 64, 1926720561250547757, 4354482840745948438, 18446013172173323708$$

Now, looking at the various MSBs of length $\lceil \log(n) \rceil = 4$, we see:

$$(0000)_2, (0001)_2, (0011)_2, (1111)_2 \tag{1}$$

If *upperBits* has a length of $2 \cdot 9 + 1 = 19$, then to decode the last number, you would need to acces the index $(1111)_2 + 8 = 39$, which is out of bounds for *upperBits*. Therefore, I decided to use $\lfloor \log(n) \rfloor$ for the length of the MSB.

Otherwise, my implementation of PREDECESSOR is based on the description of `https://arxiv.org/pdf/2003.11835.pdf`. To identify the predecessor of the number x, use SELECT queries to find the "leftmost" and "rightmost" index of numbers for which the MSB is equal to the MSB of x. Subsequently, it is just a matter of scanning this range to find the predecessor.