Karlsruher Institut für Technologie
Fakultät für Informatik

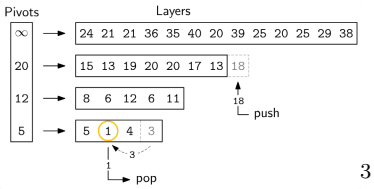
Lehrstuhl Algorithm Engineering
Prof. Dr. Peter Sanders

Master's thesis

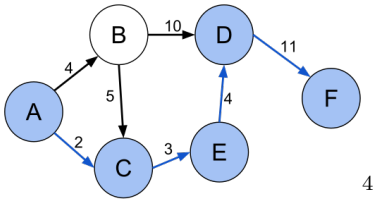
Engineering QuickHeaps

Description

Priority queues are a fundamental data structure used in many algorithms, such as Dijkstra's algorithm or task scheduling. Probably the most popular priority queue implementation is binary heaps. They are straightforward to implement and offer $\mathcal{O}(\log n)$ worst-case running times for both insertions and deletions. Another promising approach to priority queues are quick heaps, first described by Navarro and Paredes[1]. The core idea of quick heaps is, reminiscent of quick sort, to keep the elements in recursively split partitions. In theory, they offer several advantages over binary heaps, including higher cache locality of similar elements and more opportunities for parallelization. A key challenge is to keep the partitions balanced for arbitrary operation sequences to guarantee efficient operations. Randomized quick heaps[2] approach this by repartitioning on each insertion with some probability. Despite their potential, quick heaps received relatively little attention, and a lot of possibilities are still unexplored.



[3]



[4]

Goal of the Thesis

The goal of this thesis is to explore the design space of quick heaps further. We can develop a more specific goal with you according to your strengths and interests. There are many aspects to work on, such as

- **Parallelization.** Devise a design that allows multiple threads to insert and delete elements concurrently.
- **Pivot selection.** A robust pivot selection strategy is key to balanced partitions that are needed for fast operations.
- **Rebalancing.** Even with optimal pivots, the partitions can become skewed over time due to insertions. Devise an algorithm to rebalance the data structure dynamically.
- **SIMD implementation.** Utilize data parallelism to enhance the throughput even further.

There are also interesting open theoretical questions, especially regarding the comparison to binary heaps and the potential speedups achievable with SIMD.

Implementations can be in `C++` or `Rust`.

Requirements

- Solid foundation in (concurrent) algorithms and data structures
- Experience in `C++` or `Rust`
- Experience in parallel programming is a plus

[1] https://www.worldscientific.com/doi/abs/10.1142/S0129054111008507
[2] https://link.springer.com/article/10.1007/s00453-010-9400-6
[3] https://curiouscoding.nl/posts/quickheap
[4] https://commons.wikimedia.org/wiki/File:Shortest_path_with_direct_weights.svg

Contact: Marvin Williams (williams@kit.edu) and Ragnar Groot Koerkamp (ragnar.grootkoerkamp@gmail.com)
Informatikgebäude am Fasanengarten, Raum 206