HADES (High end, Advanced, Data driven, Enterprise grade Sock sorting algorithm) - An algorithm for faster sock sorting

CelloClemens[1,2], Henri🦆[1],

November 23, 2022

[1]Department for theoretical laundry science, Karlsruhe institute of suffering and sorrow (KISS), Karlsruhe, Germany
[2]Institute of laundry sorting, Department for socks, Karlsruhe institute of suffering and sorrow (KISS), Karlsruhe, Germany

Abstract Sorting socks can often be a time consuming task. This paper introduces the fastest method known in the scientific community to tackle this challanging task. To be able to implement this new algorithm a new datastructure will be introduced and discussed. Abundant application of this novel algorithm may be able to reduce the time required for sorting socks considerably.

1 Introduction

While sorting algorithms are one of the most discussed algorithms in the computer science community, application of this field to laundry is still quite new. In fact no research is known to the authors connecting the fields of computer science and laundry sorting. A few definitions are required in order to establish a baseline for the algorithm discussed in the following paper.

1.1 Definitions

In this section a few definitions, common in the field of theoretical laundry science shall be introduced. These are required to understand the algorithm and its advantages.

1.1.1 Sock

Let Λ_a be the Set of laundry. The set of socks, $\Sigma \subset \Lambda_a$ is defined as $\Sigma := \{s \in \Lambda_a | \chi(s) = 1\}$[1], where $\chi(s)$ is the Euler characteristic of s. For every sock s there is an equal counterpart s^{-1} giving rise to the identity $s \cong s^{-1}$. The task commonly known as "sock sorting" is in fact the search for this isomorphism η and matching every sock s to its inverse s^{-1}.

1.1.2 Laundry basket

Let $\Lambda \subseteq \Lambda_a$ be a set of laundry items. Then a laundry basket is a triplet $(\Lambda, +, -)$ representing a datastructure that implements the following functions:

[1]Yes, some socks have holes. So what?!

Fig. 1: A pair of blue socks and a single orange sock.

- `get:` $\mathscr{L} \in \Lambda_a$, returns a uniformly random laundry item from the basket or $\mathscr{L}_0$, the Zero element of laundry, iff There are no items left.
- `put(`$\mathscr{L} \in \Lambda_a$`)`, deposits the given laundry item into the basket.

Note that both operations run in $\mathcal{O}(1)$. Because of the nature of a laundry basket finding a unique item requires transferring the content of the whole basket to a new basket thus requiring $\mathcal{O}(n)$ operations, n being the number of items currently inside the basket.

1.2 Ongoing and latest research

To fully appreciate the gravity of HADES it has to first be discussed how most resent research tackles the problem of sock sorting. The following code describes the most recently developed sock sorting algorithm from the paper by **My colleague et al.** Which is the current industry standard. Notice the code has a runtime complexity of $\mathcal{O}(n^2)$.

Algorithm 1 Conventional sock sorting

```
 1:   ▷ initialize a new laundry basket with a given set of laundry
 2: $A \leftarrow \Lambda$   ▷ WLOG assume $\forall \mathscr{L} \in \Lambda | \mathscr{L}$ is a sock
 3: matches ← []
 4: repeat
 5:     $\mathscr{L} \leftarrow$ A.get
 6:     repeat   ▷ find the inverse Sock by checking all other socks
 7:         $\mathscr{L}' \leftarrow$ A.get
 8:     until $\mathscr{L}' = \mathscr{L}^{-1}$
 9:     matches.append(($\mathscr{L}, \mathscr{L}'$))
10: until $\mathscr{L} \neq \mathscr{L}_0$
```

2 Concepts

The basis for every fast algorithm are simple yet equally fast datastructures. To enable the low runtime achieved by HADES , the introduction of a new datastructure, the "laundry rack" is integral.

2.1 Laundry rack

Let $\Lambda \subseteq \Lambda_a$ be a set of laundry. A laundry rack (See figure 2) is a triplet $(\Lambda, +, -)$ representing a datastructure that implements the following methods:

- `get(`$\mathscr{L} \in \Lambda$`)`: $\mathscr{L} \in \Lambda$, gets a specific laundry item from the laundry rack.
- `put(`$\mathscr{L} \in \Lambda$`)`, deposits a laundry item onto the laundry rack.
- `match(`$\mathscr{L} \in \Lambda$`)`: $(\mathscr{P} \in \Lambda \times \Lambda) | \mathscr{L}_0$, returns a tuple $(\mathscr{L}, \mathscr{L}^{-1})$ representing a pair of socks iff $\mathscr{L}^{-1}$ is already on the laundry rack, $\mathscr{L}_0$ otherwise.

All these operations (especially `match`) run in $\mathcal{O}(1)$, making iteration over all n laundry items to find a pair $(\mathscr{L}, \mathscr{L}^{-1})$ obsolete.

3 Algorithm

Making use of the novel advanced features of a "drying rack" we are able to implement the following algorithm in $\mathscr{O}(n)$:

Fig. 2: A blue drying rack, found in many housholds.

Algorithm 2 HADES

```
 1:   ▷ initialize a new laundry basket with a given set of laundry
 2: $A \leftarrow \Lambda$   ▷ WLOG assume $\forall \mathscr{L} \in \Lambda | \mathscr{L}$ is a sock
 3: matches ← []
 4: $\daleth \leftarrow$ []        ▷ Create a new empty drying rack
 5: repeat
 6:     $\mathscr{L} \leftarrow$ A.get
 7:     result ← $\daleth$.match($\mathscr{L}$)
 8:     if result $\neq \mathscr{L}_0$ then
 9:         matches.append(result)
10:     else
11:         $\daleth$.put($\mathscr{L}$)
12:     end if
13: until $\mathscr{L} \neq \mathscr{L}_0$
```

As evident from the algorithm above, only one loop performing operations which are all in $\mathscr{O}(1)$ is required thus putting the algorithm in a $\mathscr{O}(n)$ runtime complexity class. Assuming that $\forall \mathscr{L} \in \Lambda \exists \mathscr{L}^{-1} | \mathscr{L} \cong \mathscr{L}^{-1}$ the algorithm always yields a correct solution for the problem (proof is left as an exercise to the reader).

4 Discussion and Results

To evaluate the algorithms performance it has been executed on different platforms consisting of diverse hardware:

5 Conclusion

6 Acknowledgements