Permutation Inverse

by Sven Nilsen, 2020

In this paper I show how to calculate the inverse of an permutation.

Computing the inverse of a permutation is simple, but can be counter-intuitive.

Here is function in Rust that computes the inverse of a permutation efficiently:

```
/// Computes the inverse of a permutation.
pub fn inverse(a: Vec<usize>) -> Vec<usize> {
    let n = a.len();
    let mut b = vec![0; n];
    for i in 0..n {b[a[i]] = i}
    b
}
```

The permutation inverse can be used to minimize the work to get from one sort `f` to another sort `g`:

$$g <=> h \cdot f$$
$$g \cdot f^{-1} <=> h$$

For example, if you have a list of type `T` which has a total order, then any fixed sort of this list can be stored as a map of indices relative to the “original” list. Each time you change the sort, you can compute the map directly from the current sort to the desired sort in `O(N)` time complexity.

For example, if you have a list of contacts and two ways to sort them:

- Sorted by first name
- Sorted by last name

One can store a list of indices for each sort. When sorting by first name, the list of indices to sort by last name can be transformed automatically such that one can go directly from sorted by first name to sorted by last name, doing only the minimal work necessary.