WIKIPEDIA

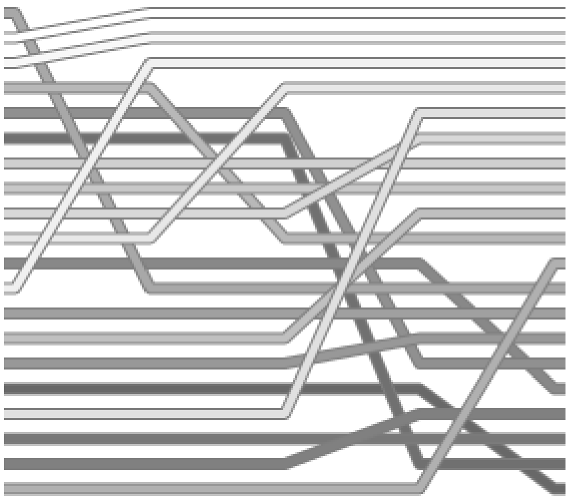
Cycle sort

Cycle sort is an in-place, unstable sorting algorithm, a comparison sort that is theoretically optimal in terms of the total number of writes to the original array, unlike any other in-place sorting algorithm. It is based on the idea that the permutation to be sorted can be factored into cycles, which can individually be rotated to give a sorted result.

Unlike nearly every other sort, items are *never* written elsewhere in the array simply to push them out of the way of the action. Each value is either written zero times, if it's already in its correct position, or written one time to its correct position. This matches the minimal number of overwrites required for a completed in-place sort.

Minimizing the number of writes is useful when making writes to some huge data set is very expensive, such as with EEPROMs like Flash memory where each write reduces the lifespan of the memory.

Cycle sort



Example of cycle sort sorting a list of random numbers.

Class	Sorting algorithm
Data structure	Array
Worst-case performance	$\Theta(n^2)$
Best-case performance	$\Theta(n^2)$
Average performance	$\Theta(n^2)$
Worst-case space complexity	$\Theta(n)$ total, $\Theta(1)$ auxiliary

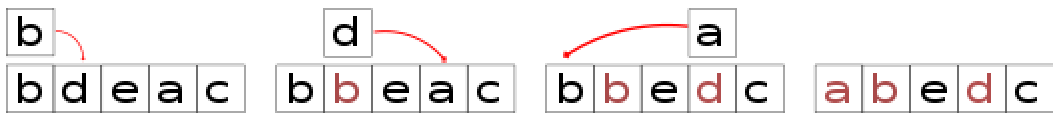
Contents

Algorithm
Implementation
Situation-specific optimizations
References
External links

Algorithm

To illustrate the idea of cycle sort, consider a list with distinct elements. Given an element a, we can find the index at which it will occur in the *sorted list* by simply counting the number of elements in the entire list that are smaller than a. Now

1. If the element is already at the correct position, do nothing.
2. If it is not, we will write it to its intended position. That position is inhabited by a different element b, which we then have to move to *its* correct position. This process of displacing elements to their correct positions continues until an element is moved to the original position of a. This completes a cycle.



Displacement cycle for list "bdeac", when shifting the first letter *b* to its correct position:

Repeating this process for every element sorts the list, with a single writing operation if and only if an element is not already at its correct position. While computing the correct positions takes $O(n)$ time for every single element, thus resulting in a quadratic time algorithm, the number of writing operations is minimized.

Implementation

To create a working implementation from the above outline, two issues need to be addressed:

1. When computing the correct positions, we have to make sure not to double-count the first element of the cycle.
2. If there are duplicate elements present, we could try to move an element a to its correct position, which already happens to be inhabited by an a. Simply swapping these would cause the algorithm to cycle indefinitely. Instead, we have to insert the element *after any of its duplicates*.

The following Python implementation[1] performs cycle sort on an array, counting the number of writes to that array that were needed to sort it.

```
def cycle_sort(array) -> int:
    """Sort an array in place and return the number of writes."""
    writes = 0

    # Loop through the array to find cycles to rotate.
    for cycle_start in range(0, len(array) - 1):
        item = array[cycle_start]

        # Find where to put the item.
        pos = cycle_start
        for i in range(cycle_start + 1, len(array)):
            if array[i] < item:
                pos += 1

        # If the item is already there, this is not a cycle.
        if pos == cycle_start:
            continue

        # Otherwise, put the item there or right after any duplicates.
        while item == array[pos]:
            pos += 1

        array[pos], item = item, array[pos]
        writes += 1

        # Rotate the rest of the cycle.
        while pos != cycle_start:
            # Find where to put the item.
            pos = cycle_start
            for i in range(cycle_start + 1, len(array)):
                if array[i] < item:
                    pos += 1

            # Put the item there or right after any duplicates.
            while item == array[pos]:
                pos += 1
            array[pos], item = item, array[pos]
            writes += 1

    return writes
```

Situation-specific optimizations

When the array contains only duplicates of a relatively small number of items, a constant-time perfect hash function can greatly speed up finding where to put an item[1], turning the sort from $\Theta(n^2)$ time to $\Theta(n + k)$ time, where k is the total number of hashes. The array ends up sorted in the order of the hashes, so choosing a hash function that gives you the right ordering is important.

Before the sort, create a histogram, sorted by hash, counting the number of occurrences of each hash in the array. Then create a table with the cumulative sum of each entry in the histogram. The cumulative sum table will then contain the position in the array of each element. The proper place of elements can then be found by a constant-time hashing and cumulative sum table lookup rather than a linear search.

References

1. sr:Ciklično sortiranje#Algoritam

External links

^ "Cycle-Sort: A Linear Sorting Method", The Computer Journal (1990) 33 (4): 365-367. (http://comjnl.oxfordjournals.org/content/33/4/365.full.pdf+html)

- Original source of unrestricted variant (https://stackoverflow.com/questions/3623509/how-to-sort-an-array-using-minimum-number-of-writes/3852666#3852666)
- Cyclesort - a curious little sorting algorithm (https://corte.si/posts/code/cyclesort/index.html)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Cycle_sort&oldid=928657396"

This page was last edited on 30 November 2019, at 20:15 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.