Sorting Signed Permutations by Reversals in Nearly-Linear Time

Bartłomiej Dudek[1], Paweł Gawrychowski[1], and Tatiana Starikovskaya[2]

[1]University of Wrocław, Poland [2]École normale supérieure, PSL Research University, France

Sorting Signed Permutation by Reversals

Input: Signed permuation π on n elements
Output: Shortest sequence of reversals sorting π

4 5 -2 -6 3 1

4 5 -2 -1 -3 6

1 2 -5 -4 -3 6

1 2 3 4 5 6

History of calculating the shortest sequence of reversals

Early '90s: upper and lower bounds, approximation.

Year	Authors	Runtime
1995	Hannenhalli and Pevzner	$\mathcal{O}(n^4)$
1996	Berman and Hannenhalli	$\mathcal{O}(n^2\alpha(n))$
1999	Kaplan, Shamir, Tarjan	$\mathcal{O}(n^2)$
2001	Bader, Moret, Yan	$\mathcal{O}(n)$ - only the length
2005	Kaplan and Verbin	$\mathcal{O}(n\sqrt{n\log n})$ rand.
2007	Tannier, Bergeron, Sagot	$\mathcal{O}(n\sqrt{n\log n})$
2006	Han	$\mathcal{O}(n\sqrt{n})$
2024	this work	$\mathcal{O}(n\log^2 n/\log\log n)$

Caprara 1997: sorting by unsigned reversals is NP-hard.

Some insights from Hannenhalli-Pevzner theory

Perform oriented reversals that create a new *adjacency*: $i, i+1$ or $-(i+1), -i$:

5 -6 2 -4 3 -1 7 → 5 1 -3 4 -2 **6 7**

5 -6 2 -4 3 -1 7 → -2 6 **-5 -4** 3 1 7

If such a reversal does not exist: run $\mathcal{O}(n)$-time preprocessing and later focus only on finding the oriented reversals [Kaplan, Shamir, Tarjan '99].

Interface of Tannier, Bergeron and Sagot

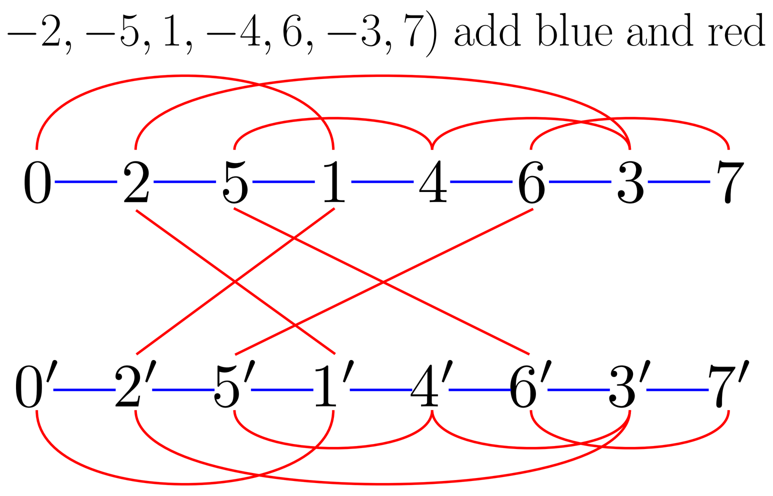
Maintain $\pi \in S_n$ and a set $V \subseteq [n]$ under the following operations:

1. query for π_i or π_i^{-1}, (splay tree+ reverse flag)
2. apply to π a signed reversal of a given interval,
3. find $i \in V$ such that i and $i+1$ have different signs in π,
4. remove an element from V.

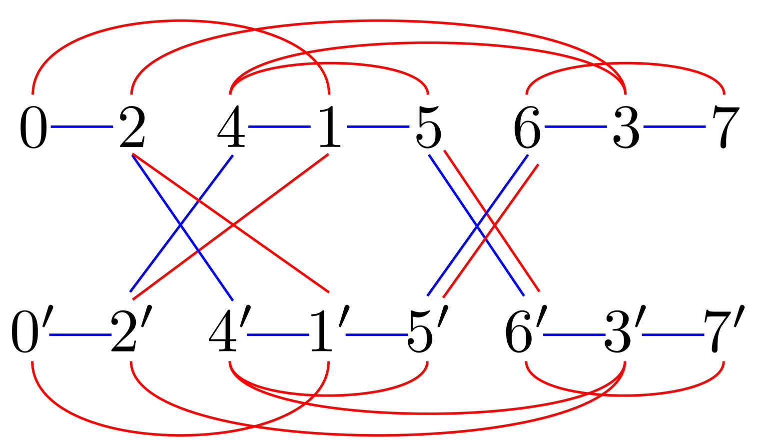
Note: Operations 3 and 4 were previously implemented in $\mathcal{O}(\sqrt{n})$ time.

Red-blue graph

For permutation $\pi = (0, -2, -5, 1, -4, 6, -3, 7)$ add blue and red edges as in the example:



which turns into the following graph after the reversal of interval $[2,4]$:



Properties of the graph

Node i is in the blue component of 0 iff i has the same sign in π as in π_0, so:

Corollary

The endpoints of a red edge $i-(i+1)/(i+1)'$ are in distinct blue components iff i and $i+1$ have different signs in π.

We need to implement the following operations:

Permutation

3. find $i \in V$ s.t. i and $i+1$ have different signs in π,
4. remove an element from V.

Red-blue graph

3. find a red edge connecting distinct blue components,
4. remove a red edge.

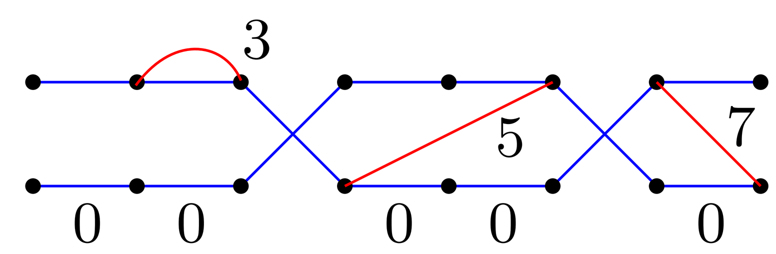
Reformulation of the problem

New goal

Find a red edge connecting distinct blue components under insertions and deletions of blue edges and deletions of red edges.

Put weights on edges:

- 0 on blue edges
- i on i-th red edge



If the graph is connected:

$\implies$ its MST has exactly one red edge

$\implies$ weight of MST gives the index of the red edge.

Use algorithm by Holm, Lichtenberg and Thorup for dynamic MST in amortized $\mathcal{O}(\log^4 n)$ time.

$\implies \mathcal{O}(n\log^4 n)$ total time.

Faster approach

Instead of MST: maintain a spanning forest of the graph on blue and red edges in a link-cut tree in amortized $\mathcal{O}(\log^2 n)$ time using a data structure for fully-dynamic graph connectivity.

How to find a red edge connecting blue components of 0 and 0′?

Recall:

1. Node i is in the blue component of 0 iff i has the same sign in π as in π_0.
2. We maintain π under signed reversals on a splay tree.

Solution: binary search over the path connecting 0 and 0′!

$\implies \mathcal{O}(n\log^2 n)$ total time.

Even faster approach

In the previous approach:

1. $\mathcal{O}(n\log n)$ queries about the sign of i in π
2. $\mathcal{O}(n)$ updates

Idea: maintain shortcuts of length $\varepsilon\log\log n$ up the splay trees

1. $\mathcal{O}(\log n/\log\log n)$ time per query
2. $\mathcal{O}(\log^{1+\varepsilon} n)$ time per update

$\implies \mathcal{O}(n\log^2 n/\log\log n)$ total time for all operations on splay trees.

For the graph: use dynamic connectivity structure by Wulf-Nilsen: $\mathcal{O}(\log^2 n/\log\log n)$ amortized time per each of $\mathcal{O}(n)$ operations.

$\implies \mathcal{O}(n\log^2 n/\log\log n)$ total time.

Cabbage & turnip explained:

Brassica oleracea (cabbage): 1 -5 4 -3 2

Brassica campestris (turnip): 1 2 3 4 5