Sorting Signed Permutations by Reversals in Nearly-Linear Time

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

[1]University of Wrocław, Poland
[2]DIENS, 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

$$4 \quad 5 \quad -2 \quad -6 \quad 3 \quad 1$$

Sorting Signed Permutation by Reversals

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

$$4 \quad 5 \quad -2 \quad -6 \quad 3 \quad 1$$

Sorting Signed Permutation by Reversals

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

$$4 \quad 5 \quad -2 \quad \underline{-6 \quad 3 \quad 1}$$

$$4 \quad 5 \quad -2 \quad -1 \quad -3 \quad 6$$

Sorting Signed Permutation by Reversals

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

$$4 \quad 5 \quad -2 \quad -6 \quad 3 \quad 1$$

$$\underline{4 \quad 5 \quad -2 \quad -1} \quad -3 \quad 6$$

$$1 \quad 2 \quad -5 \quad -4 \quad -3 \quad 6$$

Sorting Signed Permutation by Reversals

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

$$
\begin{array}{cccccc}
4 & 5 & -2 & -6 & 3 & 1 \\
4 & 5 & -2 & -1 & -3 & 6 \\
1 & 2 & \underline{-5} & \underline{-4} & \underline{-3} & 6 \\
1 & 2 & 3 & 4 & 5 & 6
\end{array}
$$

Transforming cabbage into turnip

In the late 80s, Palmer et al. discovered that cabbage and turnip have almost identical gene sequence

Brassica oleracea (cabbage)

Brassica campestris (turnip)

Transforming cabbage into turnip

In the late 80s, Palmer et al. discovered that cabbage and turnip have almost identical gene sequence, but different gene order.

Brassica oleracea (cabbage)

Brassica campestris (turnip)

$\overrightarrow{1}\ \overleftarrow{-5}\ \overrightarrow{4}\ \overleftarrow{-3}\ \overrightarrow{2}$

$\overrightarrow{1}\ \overrightarrow{2}\ \overrightarrow{3}\ \overrightarrow{4}\ \overrightarrow{5}$

Finding 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	$\mathcal{O}(n\sqrt{n \log n})$ empirical
2007	Tannier, Bergeron, Sagot	$\mathcal{O}(n\sqrt{n \log n})$
2006	Han	$\mathcal{O}(n\sqrt{n})$
2024	this work	$\mathcal{O}(n \log^2 n / \log \log n)$

Sorting by unsigned reversals is NP-hard [Caprara 1997]

Finding 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	$\mathcal{O}(n\sqrt{n \log n})$ empirical
2007	Tannier, Bergeron, Sagot	$\mathcal{O}(n\sqrt{n \log n})$
2006	Han	$\mathcal{O}(n\sqrt{n})$
2024	this work	$\mathcal{O}(n \log^2 n / \log \log n)$

Sorting by unsigned reversals is NP-hard [Caprara 1997]

Finding 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	$\mathcal{O}(n\sqrt{n \log n})$ empirical
2007	Tannier, Bergeron, Sagot	$\mathcal{O}(n\sqrt{n \log n})$
2006	Han	$\mathcal{O}(n\sqrt{n})$
2024	this work	$\mathcal{O}(n \log^2 n / \log\log n)$

Sorting by unsigned reversals is NP-hard [Caprara 1997]

Finding 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	$\mathcal{O}(n\sqrt{n \log n})$ empirical
2007	Tannier, Bergeron, Sagot	$\mathcal{O}(n\sqrt{n \log n})$
2006	Han	$\mathcal{O}(n\sqrt{n})$
2024	this work	$\mathcal{O}(n \log^2 n / \log \log n)$

Sorting by unsigned reversals is NP-hard [Caprara 1997]

Finding 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	$\mathcal{O}(n\sqrt{n \log n})$ empirical
2007	Tannier, Bergeron, Sagot	$\mathcal{O}(n\sqrt{n \log n})$
2006	Han	$\mathcal{O}(n\sqrt{n})$
2024	this work	$\mathcal{O}(n \log^2 n / \log \log n)$

Sorting by unsigned reversals is NP-hard [Caprara 1997]

Finding 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	$\mathcal{O}(n\sqrt{n \log n})$ empirical
2007	Tannier, Bergeron, Sagot	$\mathcal{O}(n\sqrt{n \log n})$
2006	Han	$\mathcal{O}(n\sqrt{n})$
2024	this work	$\mathcal{O}(n \log^2 n / \log \log n)$

Sorting by unsigned reversals is NP-hard [Caprara 1997]

Some insights from Hannenhalli-Pevzner

The goal is to perform oriented reversals that create a new *adjacency*: $i, i+1$ or $-(i+1), -i$, e.g.:

5 -6 2 -4 3 -1 7 $\rightarrow$ 5 1 -3 4 -2 **6** **7**

5 -6 2 -4 3 -1 7 $\rightarrow$ -2 6 **-5** **-4** 3 1 7

Sometimes such a reversal does not exist, but we can do an $\mathcal{O}(n)$-time preprocessing and later focus only on finding the oriented reversals [Kaplan, Shamir, Tarjan 1999].

Some insights from Hannenhalli-Pevzner

The goal is to perform oriented reversals that create a new *adjacency*: $i, i+1$ or $-(i+1), -i$, e.g.:

$5 \quad \underline{-6 \quad 2 \quad -4 \quad 3 \quad -1} \quad 7 \quad \rightarrow \quad 5 \quad 1 \quad -3 \quad 4 \quad -2 \quad \mathbf{6} \quad \mathbf{7}$

$\underline{5 \quad -6 \quad 2} \quad -4 \quad 3 \quad -1 \quad 7 \quad \rightarrow \quad -2 \quad 6 \quad \mathbf{-5} \quad \mathbf{-4} \quad 3 \quad 1 \quad 7$

Sometimes such a reversal does not exist, but we can do an $\mathcal{O}(n)$-time preprocessing and later focus only on finding the oriented reversals [Kaplan, Shamir, Tarjan 1999].

Some insights from Hannenhalli-Pevzner

The goal is to perform oriented reversals that create a new *adjacency*: $i, i+1$ or $-(i+1), -i$, e.g.:

$$\underline{5} \quad 5 \quad \underline{-6 \quad 2 \quad -4 \quad 3 \quad -1} \quad 7 \quad \rightarrow \quad 5 \quad 1 \quad -3 \quad 4 \quad -2 \quad \mathbf{6} \quad \mathbf{7}$$

$$\underline{5 \quad -6 \quad 2} \quad -4 \quad 3 \quad -1 \quad 7 \quad \rightarrow \quad -2 \quad 6 \quad \mathbf{-5} \quad \mathbf{-4} \quad 3 \quad 1 \quad 7$$

Sometimes such a reversal does not exist, but we can do an $\mathcal{O}(n)$-time preprocessing and later focus only on finding the oriented reversals [Kaplan, Shamir, Tarjan 1999].

Some insights from Hannenhalli-Pevzner

The goal is to perform oriented reversals that create a new *adjacency*: $i, i+1$ or $-(i+1), -i$, e.g.:

$$5 \quad \underline{-6 \quad 2 \quad -4 \quad 3 \quad -1} \quad 7 \quad \rightarrow \quad 5 \quad 1 \quad -3 \quad 4 \quad -2 \quad \mathbf{6} \quad \mathbf{7}$$

$$\underline{5 \quad -6 \quad 2} \quad -4 \quad 3 \quad -1 \quad 7 \quad \rightarrow \quad -2 \quad 6 \quad \mathbf{-5} \quad \mathbf{-4} \quad 3 \quad 1 \quad 7$$

Sometimes such a reversal does not exist, but we can do an $\mathcal{O}(n)$-time preprocessing and later focus only on finding the oriented reversals [Kaplan, Shamir, Tarjan 1999].

Operations on a black-white graph

Input:	Graph with nodes colored black or white
Output:	The shortest sequence of toggle operations that transform it to an independent set of white nodes

(for black v) toggle(v): negate color of nodes from $N^+(v)$ and edges between them

Operations on a black-white graph

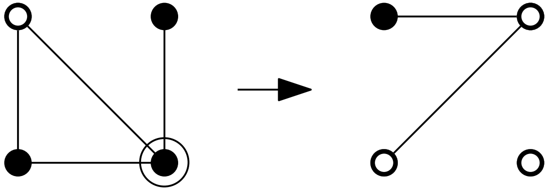
Input:	Graph with nodes colored black or white
Output:	The shortest sequence of toggle operations that transform it to an independent set of white nodes

(for black v) toggle(v): negate color of nodes from $N^+(v)$ and edges between them

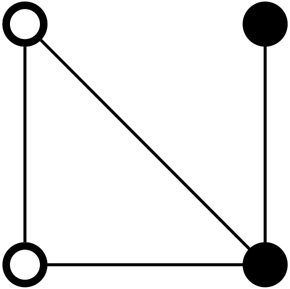
Operations on a black-white graph

Input: Graph with nodes colored black or white
Output: The shortest sequence of toggle operations that transform it to an independent set of white nodes

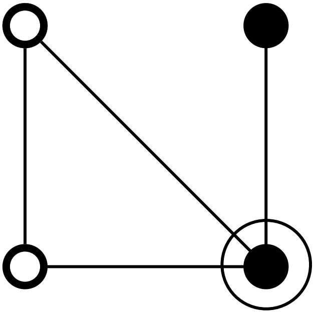
(for black v) toggle(v): negate color of nodes from $N^+(v)$ and edges between them



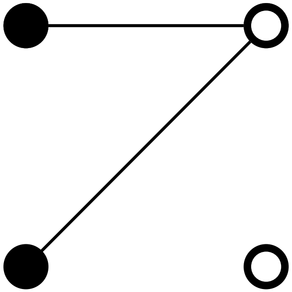
Sometimes it's easy / we're lucky



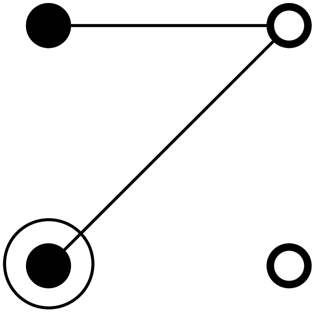
Sometimes it's easy / we're lucky



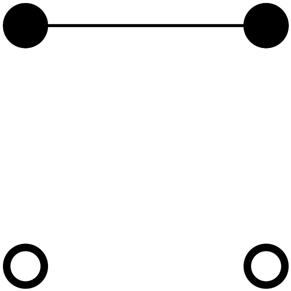
Sometimes it's easy / we're lucky



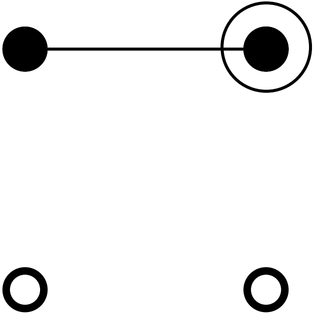
Sometimes it's easy / we're lucky



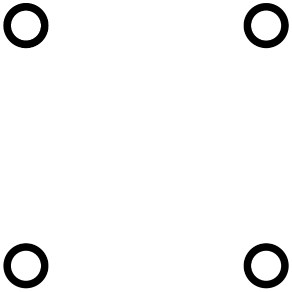
Sometimes it's easy / we're lucky



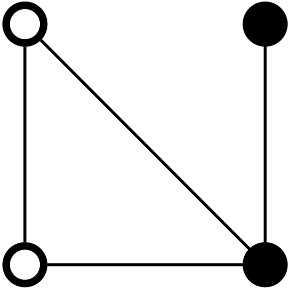
Sometimes it's easy / we're lucky



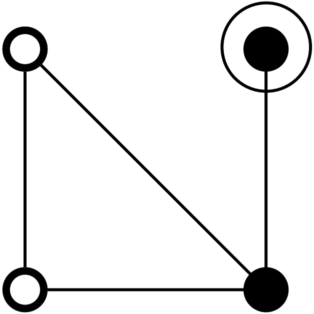
Sometimes it's easy / we're lucky



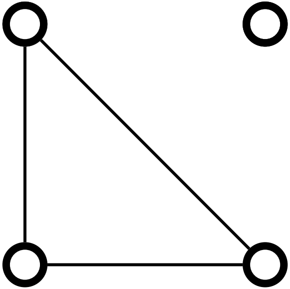
But sometimes...



But sometimes...



But sometimes...



Algorithm of Tannier, Bergeron and Sagot

function PROCESS(graph G with no non-singleton all-white connected components)
    $S := ()$
    $G' := G$
    while there is a black node v in G' **do**
        apply toggle(v) to G'
        $S := S, v$
    while there is a non-singleton all-white connected component in G/S **do**
        $U :=$ set of nodes from non-singleton all-white connected components in G/S
        $S_1, S_2 := S$ for the longest S_1 s.t. G/S_1 has a node of U in a not-all-white component
        $G_1 := G/S_1$
        $S_3 := ()$
        while there is a black node v from U in G_1 **do**
            apply toggle(v) to G_1
            $S_3 := S_3, v$
        if $S_2[1]$ is white in G_1 **then**
            remove the last element w from S_3
            undo toggle(w) in G_1
        $S := S_1, S_3, S_2$
    return S

Interface of Tannier, Bergeron and Sagot

We need to efficiently maintain $\pi \in S_n$ and a set $V \subseteq [n]$ under the following operations:

1. query for π_i or π_i^{-1},
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.

$\mathcal{O}(\log n)$ time: splay tree + reverse flag

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

Interface of Tannier, Bergeron and Sagot

We need to efficiently maintain $\pi \in S_n$ and a set $V \subseteq [n]$ under the following operations:

$\mathcal{O}(\log n)$ time: splay tree + reverse flag

1. query for π_i or π_i^{-1},
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.

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

Interface of Tannier, Bergeron and Sagot

We need to efficiently maintain $\pi \in S_n$ and a set $V \subseteq [n]$ under the following operations:

1. query for π_i or π_i^{-1},
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.

$\mathcal{O}(\log n)$ time: splay tree + reverse flag

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

Interface of Tannier, Bergeron and Sagot

We need to efficiently maintain $\pi \in S_n$ and a set $V \subseteq [n]$ under the following operations:

1. query for π_i or π_i^{-1},

   $\mathcal{O}(\log n)$ time: 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.

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

Interface of Tannier, Bergeron and Sagot

We need to efficiently maintain $\pi \in S_n$ and a set $V \subseteq [n]$ under the following operations:

1. query for π_i or π_i^{-1},
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.

$\mathcal{O}(\log n)$ time: splay tree + reverse flag

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 the following nodes and blue edges:

$$0 - 2 - 5 - 1 - 4 - 6 - 3 - 7$$

$$0' - 2' - 5' - 1' - 4' - 6' - 3' - 7'$$

Add red edges $\{i, (i+1)\}$ and $\{i', (i+1)'\}$ if i and $i+1$ have the same sign in π and $\{i, (i+1)'\}$ and $\{i', (i+1)\}$ otherwise.

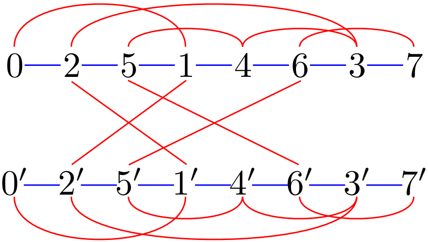
Red-blue graph

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

$$0 — 2 — 5 — 1 — 4 — 6 — 3 — 7$$

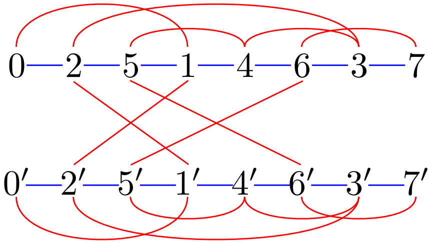
$$0' — 2' — 5' — 1' — 4' — 6' — 3' — 7'$$

Add red edges $\{i, (i+1)\}$ and $\{i', (i+1)'\}$ if i and $i+1$ have the same sign in π and $\{i, (i+1)'\}$ and $\{i', (i+1)\}$ otherwise.



Reversal of an interval

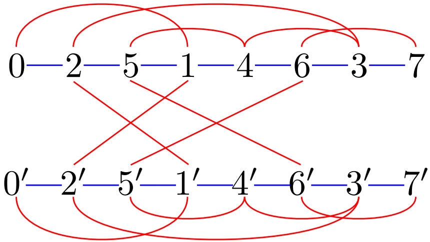
Signed reversal of an interval can be simulated by reattaching 4 blue edges, e.g. consider reversing $[2, 4]$ on $\pi = (0, -2, \underline{-5, 1, -4}, 6, -3, 7)$:



After the reversal we obtain $\pi' = (0, -2, 4, -1, 5, 6, -3, 7)$

Reversal of an interval

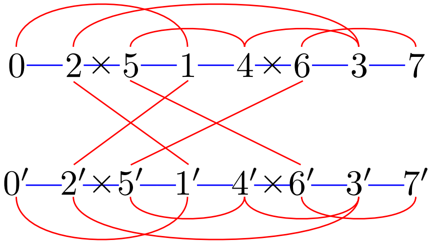
Signed reversal of an interval can be simulated by reattaching 4 blue edges, e.g. consider reversing [2, 4] on $\pi = (0, -2, \underline{-5, 1, -4}, 6, -3, 7)$:



After the reversal we obtain $\pi' = (0, -2, 4, -1, 5, 6, -3, 7)$

Reversal of an interval

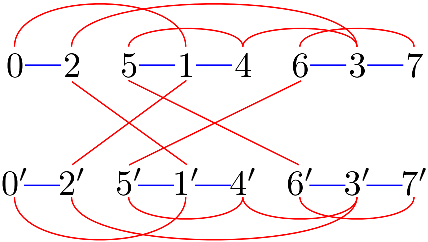
Signed reversal of an interval can be simulated by reattaching 4 blue edges, e.g. consider reversing [2, 4] on $\pi = (0, -2, \underline{-5, 1, -4}, 6, -3, 7)$:



After the reversal we obtain $\pi' = (0, -2, 4, -1, 5, 6, -3, 7)$

Reversal of an interval

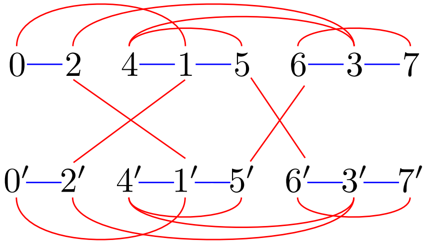
Signed reversal of an interval can be simulated by reattaching 4 blue edges, e.g. consider reversing [2, 4] on $\pi = (0, -2, \underline{-5, 1, -4}, 6, -3, 7)$:



After the reversal we obtain $\pi' = (0, -2, 4, -1, 5, 6, -3, 7)$

Reversal of an interval

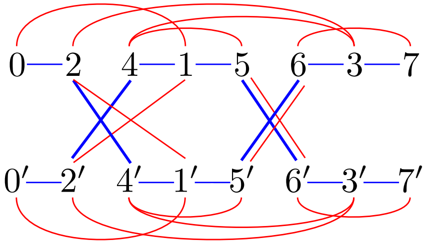
Signed reversal of an interval can be simulated by reattaching 4 blue edges, e.g. consider reversing $[2, 4]$ on $\pi = (0, -2, \underline{-5, 1, -4}, 6, -3, 7)$:



After the reversal we obtain $\pi' = (0, -2, 4, -1, 5, 6, -3, 7)$

Reversal of an interval

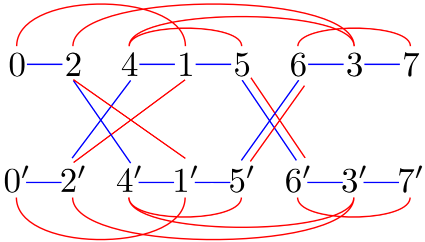
Signed reversal of an interval can be simulated by reattaching 4 blue edges, e.g. consider reversing $[2, 4]$ on $\pi = (0, -2, \underline{-5, 1, -4}, 6, -3, 7)$:



After the reversal we obtain $\pi' = (0, -2, 4, -1, 5, 6, -3, 7)$

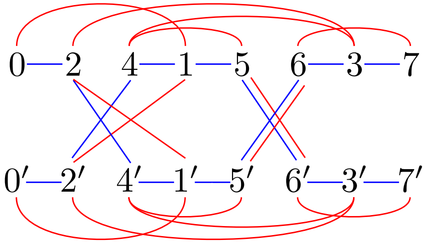
Reversal of an interval

Signed reversal of an interval can be simulated by reattaching 4 blue edges, e.g. consider reversing [2, 4] on $\pi = (0, -2, \underline{-5, 1, -4}, 6, -3, 7)$:



After the reversal we obtain $\pi' = (0, -2, 4, -1, 5, 6, -3, 7)$

Properties of the graph



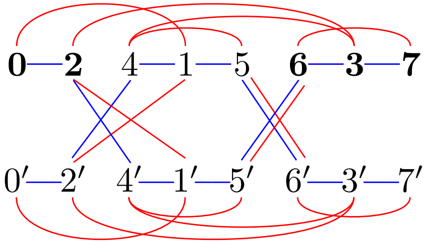
Property

Node *i* is in the blue component of 0 if and only if *i* has the same sign in π as in π_0.

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.

Properties of the graph



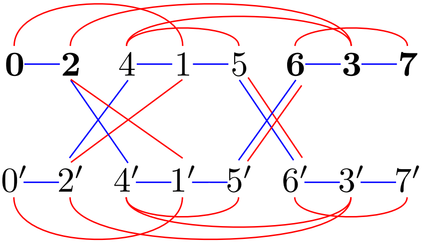
Property

Node i is in the blue component of 0 if and only if i has the same sign in π as in π_0.

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.

Properties of the graph



Property

Node *i* is in the blue component of 0 if and only if *i* has the same sign in π as in π_0.

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.

Recall

We need to implement the following operations:

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

In our red-blue graph they translate to:

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

Recall

We need to implement the following operations:

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

In our red-blue graph they translate to:

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

Recall

We need to implement the following operations:

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

In our red-blue graph they translate to:

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.

Algorithm

Put weights on edges:

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

If the graph is connected:

$\Longrightarrow$ its MST has exactly one red edge

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

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

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

Reformulation of the problem

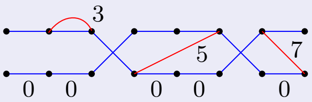
New goal

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

Algorithm

Put weights on edges:

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



If the graph is connected:

$\Longrightarrow$ its MST has exactly one red edge

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

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

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

Reformulation of the problem

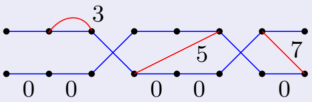
New goal

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

Algorithm

Put weights on edges:

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



If the graph is connected:

$\Longrightarrow$ its MST has exactly one red edge

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

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

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

Reformulation of the problem

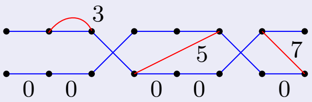
New goal

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

Algorithm

Put weights on edges:

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



If the graph is connected:

$\Longrightarrow$ its MST has exactly one red edge

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

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

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

Reformulation of the problem

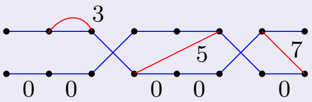
New goal

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

Algorithm

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 the 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.

Reformulation of the problem

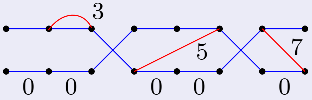
New goal

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

Algorithm

Put weights on edges:

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



If the graph is connected:

$\Longrightarrow$ its MST has exactly one red edge

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

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

$\Longrightarrow$ $\mathcal{O}(n \log^4 n)$ 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 if and only if 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.

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 if and only if 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.

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 if and only if 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.

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 if and only if 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

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

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

$\Longrightarrow$ $\mathcal{O}(n \log^2 n / \log \log n)$ 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 Wulff-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.

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

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

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

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

Open questions

1. Can we improve the time complexity even further?
   – Our solution runs in $\mathcal{O}(n \log^2 n / \log \log n)$ but counting the numer of reversals takes $\mathcal{O}(n)$ time
   – Swenson [arXiv '24]: $\mathcal{O}(n \log n)$
2. Does randomization help?
   – Can we utilize the $\mathcal{O}(n \log n \log^2 \log n)$ time algorithm for dynamic connectivity by Huang, Huang, Kopelowitz, Pettie, and Thorup?

Thank you!

Open questions

1. Can we improve the time complexity even further?
   – Our solution runs in $\mathcal{O}(n \log^2 n / \log\log n)$ but counting the numer of reversals takes $\mathcal{O}(n)$ time
   – Swenson [arXiv '24]: $\mathcal{O}(n \log n)$
2. Does randomization help?
   – Can we utilize the $\mathcal{O}(n \log n \log^2 \log n)$ time algorithm for dynamic connectivity by Huang, Huang, Kopelowitz, Pettie, and Thorup?

Thank you!

Open questions

1. Can we improve the time complexity even further?
   – Our solution runs in $\mathcal{O}(n \log^2 n / \log \log n)$ but counting the numer of reversals takes $\mathcal{O}(n)$ time
   – Swenson [arXiv '24]: $\mathcal{O}(n \log n)$
2. Does randomization help?
   – Can we utilize the $\mathcal{O}(n \log n \log^2 \log n)$ time algorithm for dynamic connectivity by Huang, Huang, Kopelowitz, Pettie, and Thorup?

Thank you!

Open questions

1. Can we improve the time complexity even further?
   – Our solution runs in $\mathcal{O}(n\log^2 n/\log\log n)$ but counting the numer of reversals takes $\mathcal{O}(n)$ time
   – Swenson [arXiv '24]: $\mathcal{O}(n\log n)$
2. Does randomization help?
   – Can we utilize the $\mathcal{O}(n\log n\log^2\log n)$ time algorithm for dynamic connectivity by Huang, Huang, Kopelowitz, Pettie, and Thorup?

Thank you!

Open questions

1. Can we improve the time complexity even further?
   – Our solution runs in $\mathcal{O}(n \log^2 n / \log \log n)$ but counting the numer of reversals takes $\mathcal{O}(n)$ time
   – Swenson [arXiv '24]: $\mathcal{O}(n \log n)$
2. Does randomization help?
   – Can we utilize the $\mathcal{O}(n \log n \log^2 \log n)$ time algorithm for dynamic connectivity by Huang, Huang, Kopelowitz, Pettie, and Thorup?

Thank you!

Open questions

1. Can we improve the time complexity even further?
   – Our solution runs in $\mathcal{O}(n \log^2 n / \log \log n)$ but counting the numer of reversals takes $\mathcal{O}(n)$ time
   – Swenson [arXiv '24]: $\mathcal{O}(n \log n)$
2. Does randomization help?
   – Can we utilize the $\mathcal{O}(n \log n \log^2 \log n)$ time algorithm for dynamic connectivity by Huang, Huang, Kopelowitz, Pettie, and Thorup?

Thank you!