Inverting Permutations In Place

Matthew Robertson

Cheriton School of Computer Science
University of Waterloo

July 8, 2015

The Problem

Given the standard representation of a permutation π in the array `pi` as input, replace the input with the representation of π^{-1}, the inverse of the permutation, quickly and in place.

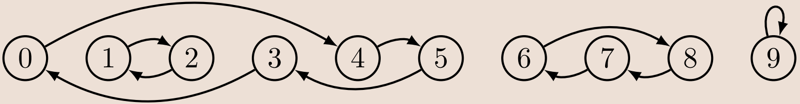
What is a Permutation?

Definition

A **permutation** π is a one-to-one correspondence between $[n]$ and itself, where $[n] = \{0, 1, \ldots, n-1\}$.

Example

i	0	1	2	3	4	5	6	7	8	9
$\pi(i)$	4	2	1	0	5	3	8	6	7	9



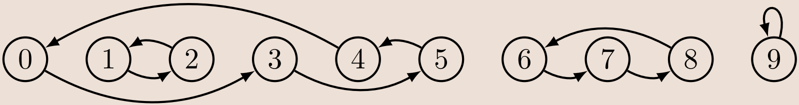
What is the Inverse of a Permutation?

Definition

The **inverse** of the permutation π, π^{-1}, is the permutation such that, for all i, $\pi^{-1} \circ \pi(i) = i$.

Example

$\pi^{-1}(i)$	0	1	2	3	4	5	6	7	8	9
i	4	2	1	0	5	3	8	6	7	9



More Definitions

Definition

An **in place** algorithm is an algorithm that executes by rearranging elements of the input without the use of significant extra space.

Definition

The **standard** representation of the permutation π is an array $\texttt{pi}$ in which $\texttt{pi}[i] = \pi(i)$, where each element of $\texttt{pi}$ is of size $\lceil \lg n \rceil$ bits.

Applications

- Application in data warehousing:
  - Under specific indexing schemes, the permutation corresponding to the rows of a relation sorted by any given key is explicitly stored.
  - To perform certain joins, the inverse of a segment of the permutation is precisely what is needed.
- Application in other fields, such as bioinformatics.

Main Contribution

- A technique that leads to an algorithm to invert the standard representation of a permutation using only:
  - $\mathcal{O}(\log^2 n)$ extra bits of space
  - $\mathcal{O}(n \log n)$ time

  in the worst case[1].
- Previously known algorithms used either quadratic time or a linear number of extra bits of space.

[1] Assume the standard word RAM model for all results.

Structure of Presentation

1. Introduction
2. Background
3. A Naive Solution
4. Breaking the Cycle
5. Conclusion

Background – Outline

1 Introduction

2 Background
- Representations of Permutations
- Permuting In Place
- Cycle Leader Algorithms

3 A Naive Solution

4 Breaking the Cycle

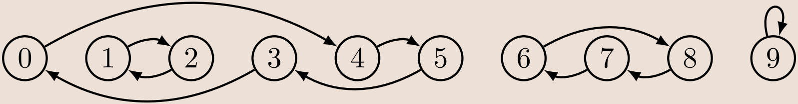
5 Conclusion

The Standard Representation

- Array `pi` of positions 0 to $n-1$ in which $\texttt{pi}[i] = \pi(i)$.
- Uses $n \lceil \lg n \rceil$ bits of space in total.
- Computes $\pi(i)$ in $\mathcal{O}(1)$ time.
- A very natural representation.
- Not trivial to invert in place.
- About $n \lg e$ bits more than optimal.

Example

`pi[] = {4,2,1,0,5,3,8,6,7,9}`

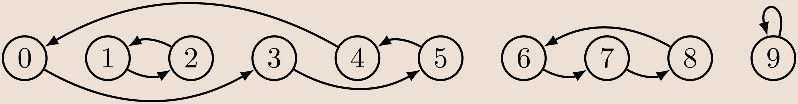


The Standard Representation

- Array `pi` of positions 0 to $n-1$ in which $\texttt{pi}[i] = \pi(i)$.
- Uses $n\lceil \lg n \rceil$ bits of space in total.
- Computes $\pi(i)$ in $\mathcal{O}(1)$ time.
- A very natural representation.
- Not trivial to invert in place.
- About $n \lg e$ bits more than optimal.

Example

`pi[] = {3,2,1,5,0,4,7,8,6,9}`



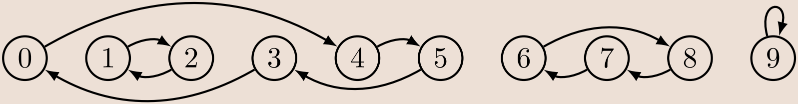
Cycle Form of a Permutation

- Explicitly store each cycle of the permutation.
- Can be represented in $n \lceil \lg n \rceil$ bits of space.
- Trivial to invert by reversing all cycles.
- Cannot compute $\pi(i)$ quickly.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`perm[] = {9,6,8,7,1,2,0,4,5,3}`



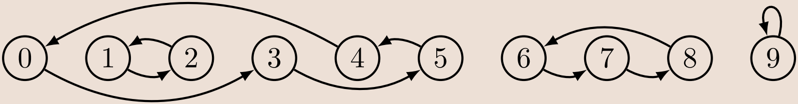
Cycle Form of a Permutation

- Explicitly store each cycle of the permutation.
- Can be represented in $n \lceil \lg n \rceil$ bits of space.
- Trivial to invert by reversing all cycles.
- Cannot compute $\pi(i)$ quickly.

Example

$$\pi = (0\ 3\ 5\ 4)(1\ 2)(6\ 7\ 8)(9)$$

`perm[] = {9,6,7,8,1,2,0,3,5,4}`



Improve the Standard Representation

- The standard representation of a permutation uses $n \lceil \lg n \rceil$ bits.
- Encode n objects of size $\lceil \lg n \rceil$ bits, totalling up to $n \lg n + n$ bits.
- Between about $n \lg e$ and $n(1 + \lg e)$ bits more than optimal.
  - Optimal is $\lceil \lg n! \rceil = n \lg n - n \lg e + \mathcal{O}(\log n)$ bits.
- Consequence of there being no repeated values.
- Reduce space by encoding k consecutive elements into a single object.
- Essentially the k digit, base n number $\texttt{pi}[i]\texttt{pi}[i+1]\ldots\texttt{pi}[i+k-1]$.
- Encode $\frac{n}{k}$ objects of size $\lceil k \lg n \rceil$ bits, totalling up to $n \lg n + \frac{n}{k}$ bits.
- Decode with a constant number of $\lceil k \lg n \rceil$ bit precision operations.

Permuting In Place – Outline

- A related problem is to permute in place.
- Apply a permutation to an array, without much extra space.
- Exploit the cycle structure of a permutation.
- The main inspiration for this research.
- Studied in [Fich et al., 1995].

Permuting – Applying a Permutation

Definition

An array $A[0, n-1]$ is **permuted** according to the permutation π when element $A[i]$ is moved to the position $\pi(i)$ for each $i \in [n]$.

- Not sufficient to simply assign $A[\pi(i)] \leftarrow A(i)$ for each $i \in [n]$.
- Elements in A may have been mutated before being accessed.
- Some very simple ways to deal with this:
  - The use of an auxiliary copy of A — $\mathcal{O}(n \log n)$ bits.
  - Use of an n-bit vector to mark moved elements — $\mathcal{O}(n)$ bits.
  - Destroy the permutation by assigning $\texttt{pi}[i] \leftarrow i$ as it is traversed.
- Not in place, or slow and destroy information.

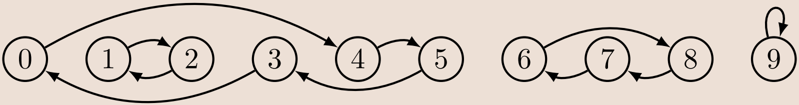
The Cycle Structure of a Permutation

- The cycle structure of a permutation can still be exploited while using the standard representation.
- Cycles are traversed by repeatedly evaluating $i \leftarrow \pi(i)$.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



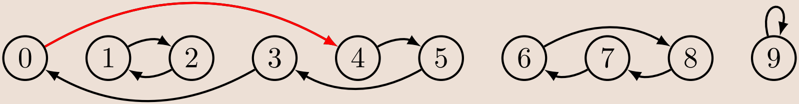
The Cycle Structure of a Permutation

- The cycle structure of a permutation can still be exploited while using the standard representation.
- Cycles are traversed by repeatedly evaluating $i \leftarrow \pi(i)$.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



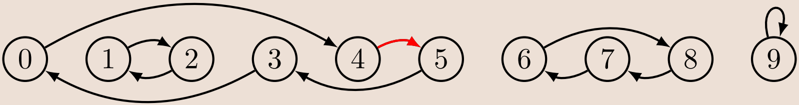
The Cycle Structure of a Permutation

- The cycle structure of a permutation can still be exploited while using the standard representation.
- Cycles are traversed by repeatedly evaluating $i \leftarrow \pi(i)$.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



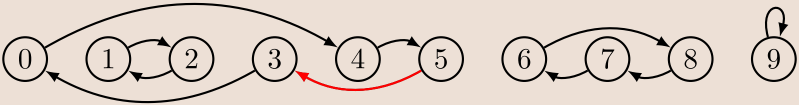
The Cycle Structure of a Permutation

- The cycle structure of a permutation can still be exploited while using the standard representation.
- Cycles are traversed by repeatedly evaluating $i \leftarrow \pi(i)$.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



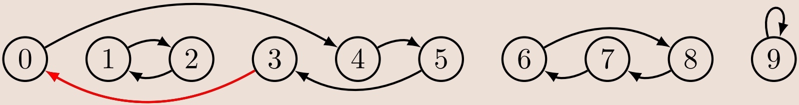
The Cycle Structure of a Permutation

- The cycle structure of a permutation can still be exploited while using the standard representation.
- Cycles are traversed by repeatedly evaluating $i \leftarrow \pi(i)$.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



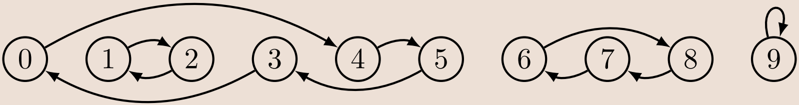
The Cycle Structure of a Permutation

- The cycle structure of a permutation can still be exploited while using the standard representation.
- Cycles are traversed by repeatedly evaluating $i \leftarrow \pi(i)$.
- No obvious way to find the "next" cycle.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

```
pi[] = {4,2,1,0,5,3,8,6,7,9}
```



Rotating a Cycle

- A basic operation in permuting.
- Can be thought of as applying a cycle.
- Cycles are rotated starting from some position, $leader$, within the cycle.
- Takes $\mathcal{O}(\log n)$ bits for pointers and $\mathcal{O}(\ell)$ time, where ℓ is the length of the cycle.
  - Takes $\mathcal{O}(n)$ time to rotate each cycle exactly once.

Algorithm

method `rotate(`A, $leader$`)`
    $i \leftarrow \pi(leader)$
    while $i \neq leader$ **do**
        `swap(`$A[i]$, $A[leader]$`)`
        $i \leftarrow \pi(i)$

Cycle Leader Algorithms – Outline

- Have been studied in the context of permuting.
- Identify cycles while using the standard representation.
- Cycle leader methods:
  - The minimum leader.
  - The Fich et al. leader.
- Known results.

A Cycle Leader

Definition

A **cycle leader** is a protocol by which precisely one position in each cycle is so designated.

- Not trivial because the standard representation of a permutation does not naturally distinguish cycles.
- Cycles are identified by testing each position i in π for cycle leadership.
- An array is permuted by rotating each cycle once from its cycle leader.
- The minimum position of each cycle is an example of a cycle leader.

The Permute Algorithm

Algorithm

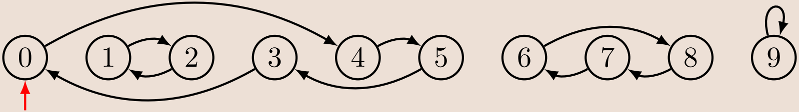
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

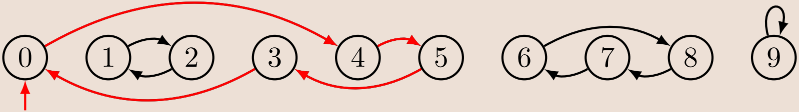
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

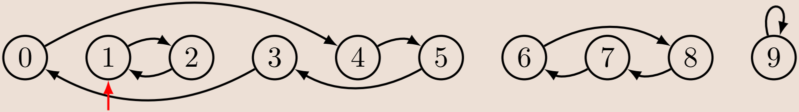
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

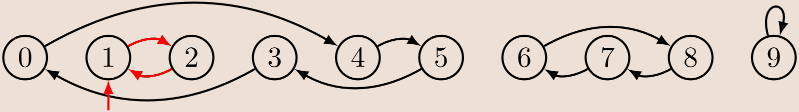
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

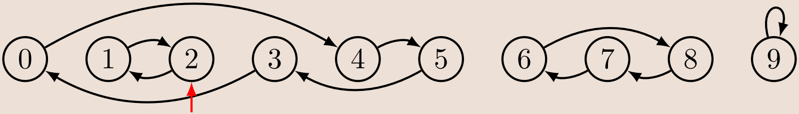
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

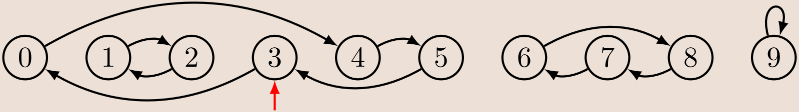
method `permute`(A)
    for $i \leftarrow 0$ **to** $n-1$ **do**
        if `isLeader`(i) **then**
            `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

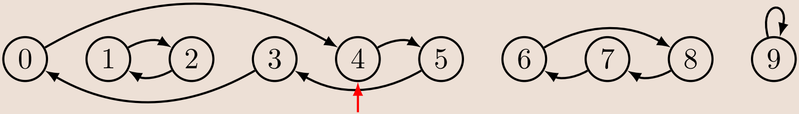
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

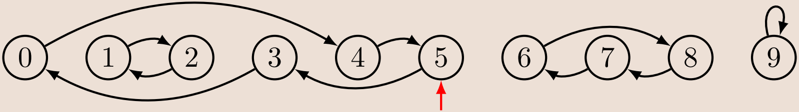
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

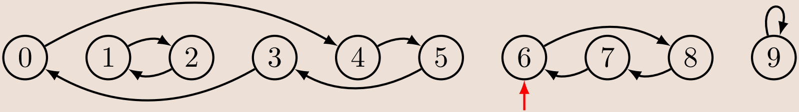
method `permute(`A`)`
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader(`i`)` **then**
      `rotate(`A, i`)`

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

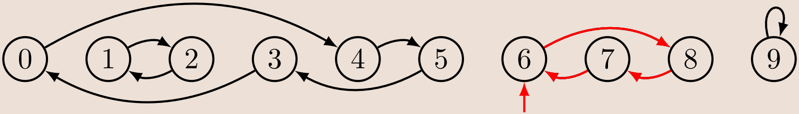
method `permute(`A`)`
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader(`i`)` **then**
      `rotate(`A, i`)`

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

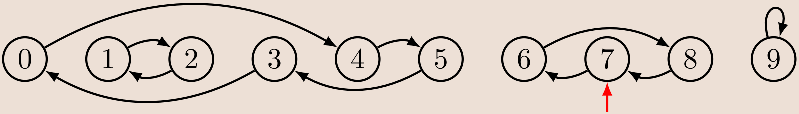
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

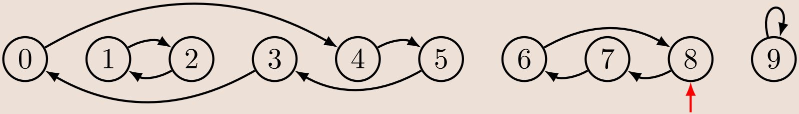
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

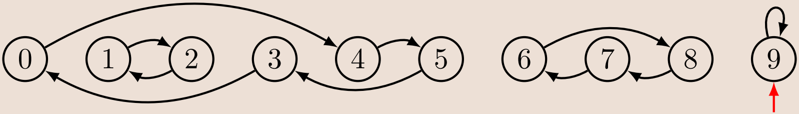
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

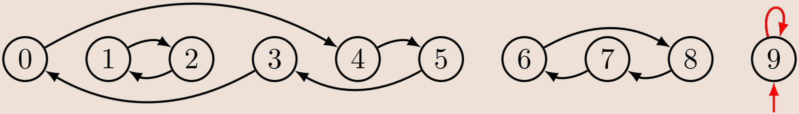
method `permute`(A)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader`(i) **then**
      `rotate`(A, i)

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Permute Algorithm

Algorithm

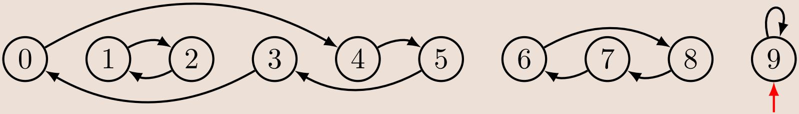
method `permute(`A`)`
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader(`i`)` **then**
      `rotate(`A, i`)`

- Permute A according to π.
- For each cycle, rotate the cycle.
- Could use any leader method.
- This e.x. uses minimum leader.

Example

$$\pi = (0\ 4\ 5\ 3)(1\ 2)(6\ 8\ 7)(9)$$

`pi[] = {4,2,1,0,5,3,8,6,7,9}`



The Minimum Leader Method

- Determine the minimum position in each cycle.
- Traversing only forward along the cycle.
- Permuting using the minimum leader:
  - Takes as few as $2n$ value inspections in the best case.
  - Takes as many as n^2 value inspections in the worst case.
  - Takes about $n \lg n$ for a random cycle of length n.

Algorithm

method `isLeader` ($leader$)

    $i \leftarrow \pi(leader)$

    while $i > leader$ **do**

        $i \leftarrow \pi(i)$

    return $i \stackrel{?}{=} leader$

The Minimum Leader – Theorem

Theorem [Fich et al., 1995]

In the worst case, permuting an array of length n, given the permutation, can be done in $\mathcal{O}(n^2)$ time and $\mathcal{O}(\log n)$ additional bits of storage.

The Fich et al. Leader Method

- The leader method presented in [Fich et al., 1995].
- Designate as leader the position from which the "local minima" phenomenon can be observed.
  - α_0 is a cycle in π and x_0 is the cycle leader.
  - α_r is the cycle of the local minima of α_{r-1}.
  - $x_r = \alpha_{r-1}(x_{r-1})$ is the leader at depth r.
  - Recurse until an α_d is found such that $\alpha_d(x_d) = x_d$.

Example

$$\alpha_0 = (0\ 6\ 1\ \vec{8}\ 3\ 4\ 2\ 9\ 5\ 7)$$

$$\alpha_1 = (0\ 1\ \vec{3}\ 2\ 5)$$

$$\alpha_2 = (0\ \vec{2})$$

$$\alpha_3 = (\vec{0})$$

- $\pi = [6, 8, 9, 4, 2, 7, 1, 0, 3, 5]$.
- The leader is $x_0 = 8$ because...
- $\alpha_0(8) = 3$,
- $\alpha_1(3) = 2$,
- $\alpha_2(2) = 0$, and
- $\alpha_3(0) = 0$ points to itself.

The Fich et al. Leader Method

- The leader method presented in [Fich et al., 1995].
- Designate as leader the position from which the "local minima" phenomenon can be observed.
  - α_0 is a cycle in π and x_0 is the cycle leader.
  - α_r is the cycle of the local minima of α_{r-1}.
  - $x_r = \alpha_{r-1}(x_{r-1})$ is the leader at depth r.
  - Recurse until an α_d is found such that $\alpha_d(x_d) = x_d$.

Example

$$\alpha_0 = (0\ 6\ 1\ \vec{8}\ 3\ 4\ 2\ 9\ 5\ 7)$$

$$\alpha_1 = (0\ 1\ \vec{3}\ 2\ 5)$$

$$\alpha_2 = (0\ \vec{2})$$

$$\alpha_3 = (\vec{0})$$

- $\pi = [6, 8, 9, 4, 2, 7, 1, 0, 3, 5]$.
- The leader is $x_0 = 8$ because...
- $\alpha_0(8) = 3$,
- $\alpha_1(3) = 2$,
- $\alpha_2(2) = 0$, and
- $\alpha_3(0) = 0$ points to itself.

The Fich et al. Leader Method

- The leader method presented in [Fich et al., 1995].
- Designate as leader the position from which the "local minima" phenomenon can be observed.
  - α_0 is a cycle in π and x_0 is the cycle leader.
  - α_r is the cycle of the local minima of α_{r-1}.
  - $x_r = \alpha_{r-1}(x_{r-1})$ is the leader at depth r.
  - Recurse until an α_d is found such that $\alpha_d(x_d) = x_d$.

Example

$$\alpha_0 = (0\ 6\ 1\ \vec{8}\ 3\ 4\ 2\ 9\ 5\ 7)$$

$$\alpha_1 = (0\ 1\ \vec{3}\ 2\ 5)$$

$$\alpha_2 = (0\ \vec{2})$$

$$\alpha_3 = (\vec{0})$$

- $\pi = [6, 8, 9, 4, 2, 7, 1, 0, 3, 5]$.
- The leader is $x_0 = 8$ because. . .
- $\alpha_0(8) = 3$,
- $\alpha_1(3) = 2$,
- $\alpha_2(2) = 0$, and
- $\alpha_3(0) = 0$ points to itself.

The Fich et al. Leader Method

- The leader method presented in [Fich et al., 1995].
- Designate as leader the position from which the “local minima” phenomenon can be observed.
    - α_0 is a cycle in π and x_0 is the cycle leader.
    - α_r is the cycle of the local minima of α_{r-1}.
    - $x_r = \alpha_{r-1}(x_{r-1})$ is the leader at depth r.
    - Recurse until an α_d is found such that $\alpha_d(x_d) = x_d$.

Example

$$\alpha_0 = (0\ 6\ 1\ \vec{8}\ 3\ 4\ 2\ 9\ 5\ 7)$$

$$\alpha_1 = (0\ 1\ \vec{3}\ 2\ 5)$$

$$\alpha_2 = (0\ \vec{2})$$

$$\alpha_3 = (\vec{0})$$

- $\pi = [6, 8, 9, 4, 2, 7, 1, 0, 3, 5]$.
- The leader is $x_0 = 8$ because...
- $\alpha_0(8) = 3$,
- $\alpha_1(3) = 2$,
- $\alpha_2(2) = 0$, and
- $\alpha_3(0) = 0$ points to itself.

The Fich et al. Leader Method

- The leader method presented in [Fich et al., 1995].
- Designate as leader the position from which the "local minima" phenomenon can be observed.
  - α_0 is a cycle in π and x_0 is the cycle leader.
  - α_r is the cycle of the local minima of α_{r-1}.
  - $x_r = \alpha_{r-1}(x_{r-1})$ is the leader at depth r.
  - Recurse until an α_d is found such that $\alpha_d(x_d) = x_d$.

Example

$$\alpha_0 = (0\ 6\ 1\ \vec{8}\ 3\ 4\ 2\ 9\ 5\ 7)$$

$$\alpha_1 = (0\ 1\ \vec{3}\ 2\ 5)$$

$$\alpha_2 = (0\ \vec{2})$$

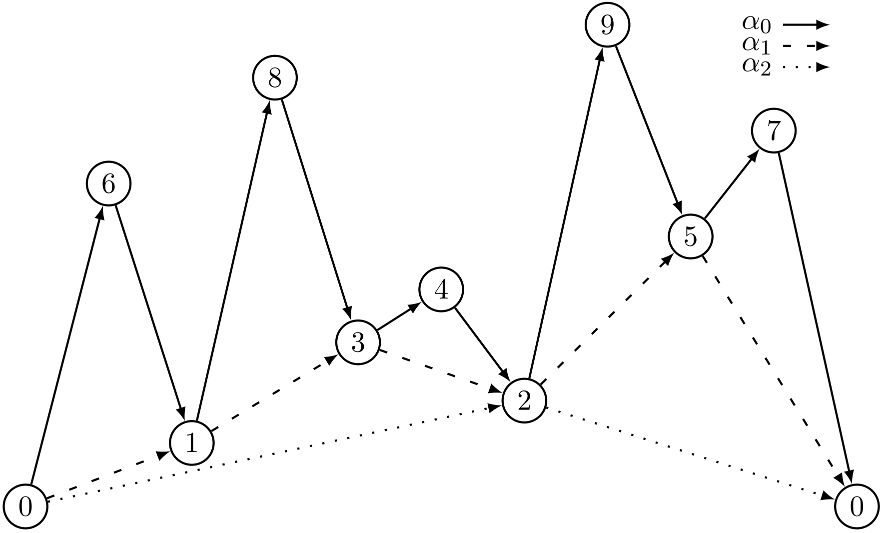
$$\alpha_3 = (\vec{0})$$

- $\pi = [6, 8, 9, 4, 2, 7, 1, 0, 3, 5]$.
- The leader is $x_0 = 8$ because...
- $\alpha_0(8) = 3$,
- $\alpha_1(3) = 2$,
- $\alpha_2(2) = 0$, and
- $\alpha_3(0) = 0$ points to itself.

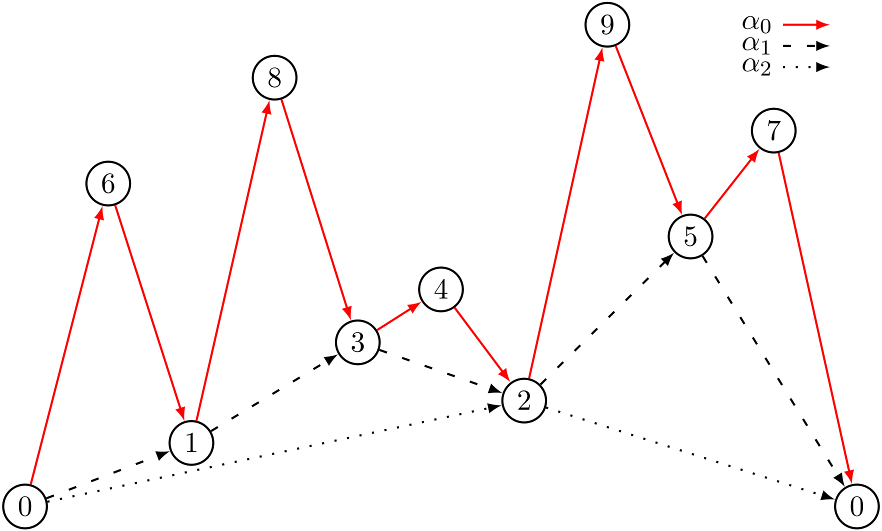
The Fich et al. Leader – Local Minima

- The position i is a local minimum when $\pi^{-1}(i) > i < \pi(i)$.
- Testing from position i:
  - Easy to compute $\pi(i)$ but hard to compute $\pi^{-1}(i)$.
  - Test if $\pi(i)$ is a local minimum, i.e., $i > \pi(i) < \pi \circ \pi(i)$.
- Testing in depth r:
  - A position x_r is in α_r if it is a local minimum of α_{r-1}.
  - At least half of the positions are excluded at each level.
- The maximum depth is $d \leq \lceil \lg \ell \rceil$.
- The leader is the position which leads to the maximum depth.
- In general, the leader is i if $\alpha_d \circ \alpha_{d-1} \circ \ldots \circ \alpha_0(i) = x_d$.
- Graphic example on the following slide.

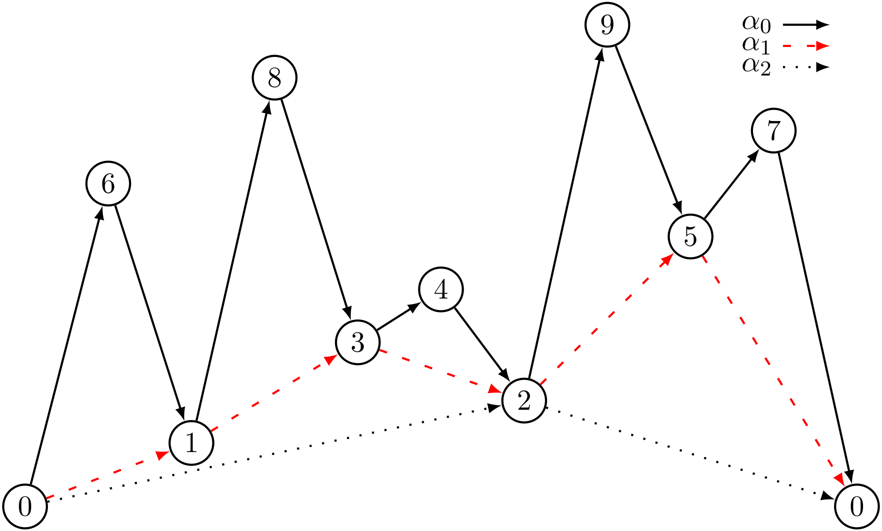
The Fich et al. Leader – Graphic Example



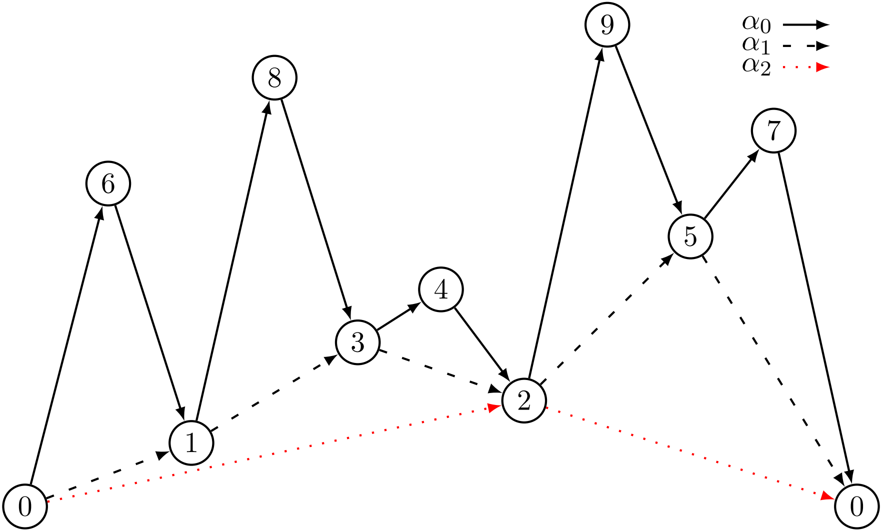
The Fich et al. Leader – Graphic Example



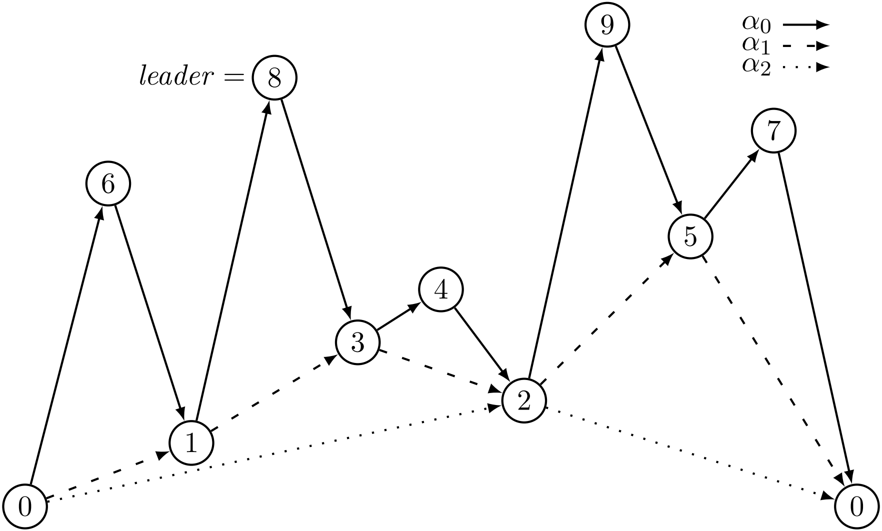
The Fich et al. Leader – Graphic Example



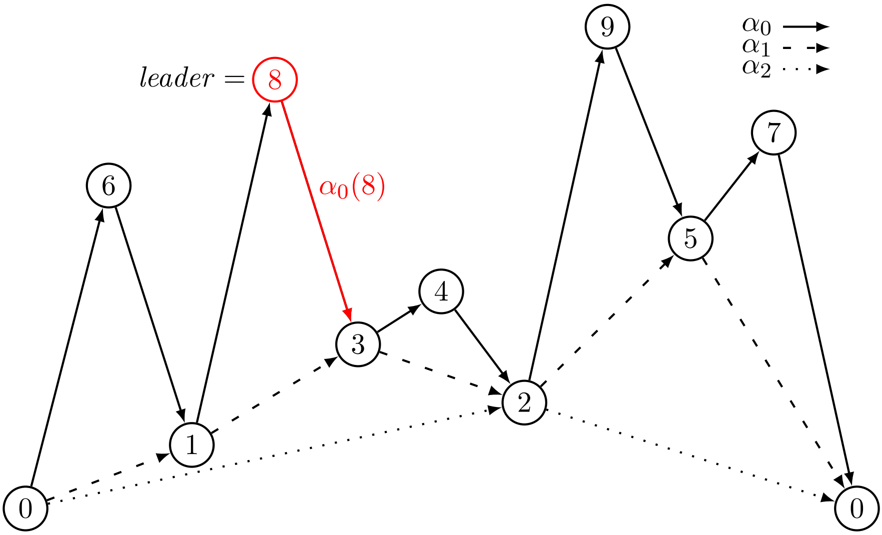
The Fich et al. Leader – Graphic Example



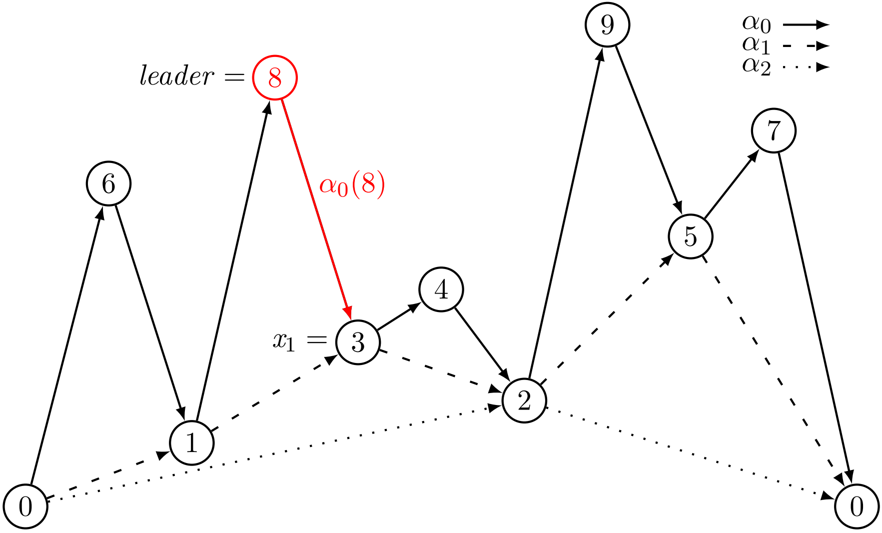
The Fich et al. Leader – Graphic Example



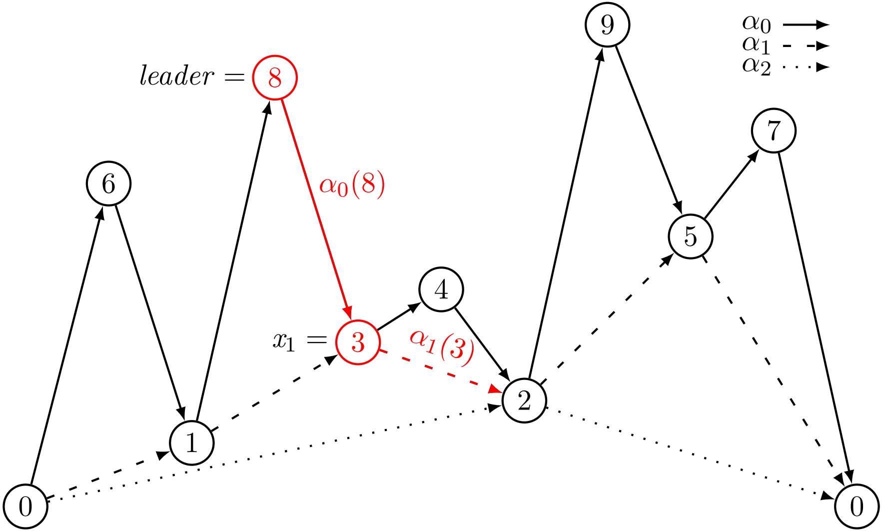
The Fich et al. Leader – Graphic Example



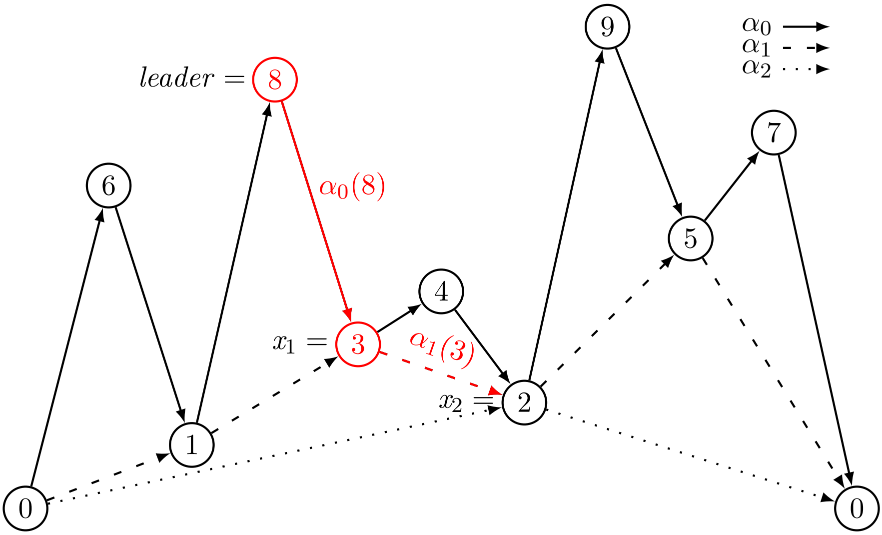
The Fich et al. Leader – Graphic Example



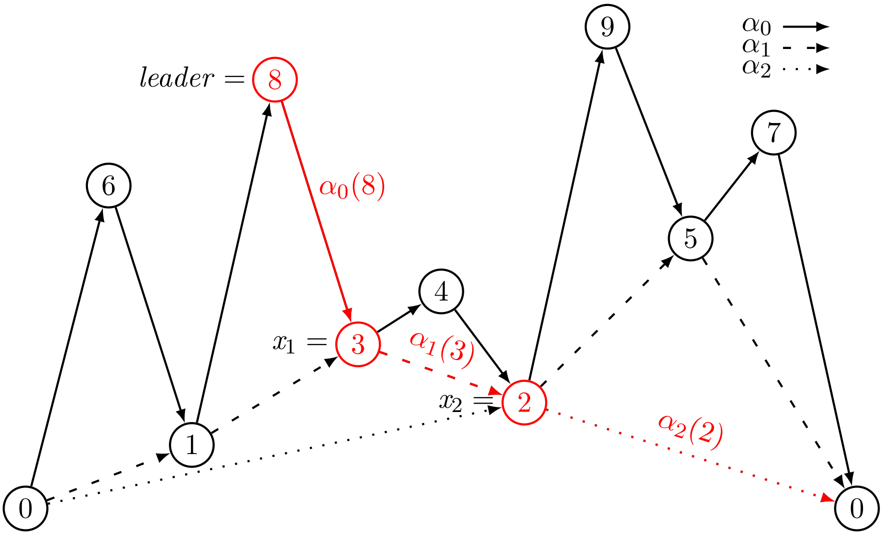
The Fich et al. Leader – Graphic Example



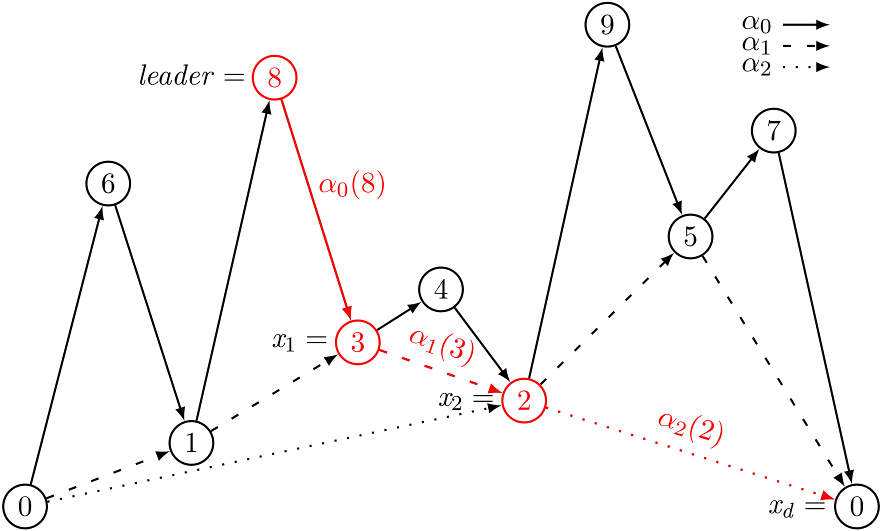
The Fich et al. Leader – Graphic Example



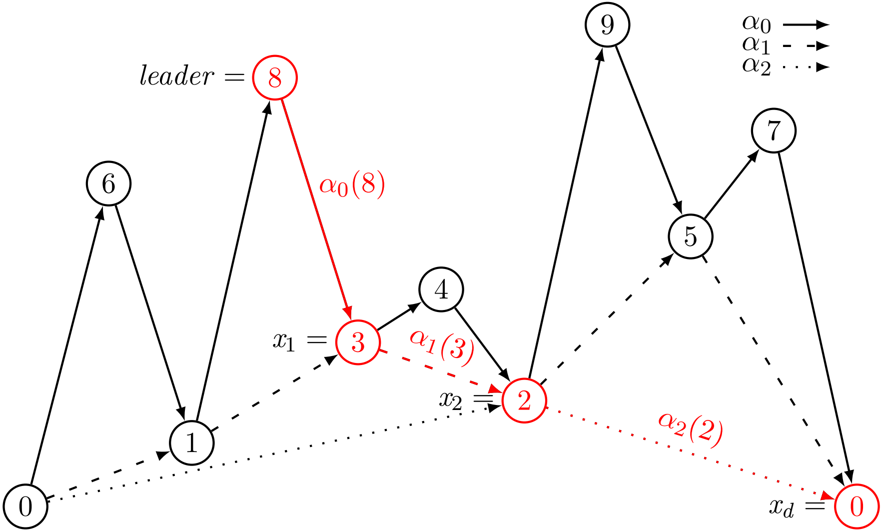
The Fich et al. Leader – Graphic Example



The Fich et al. Leader – Graphic Example



The Fich et al. Leader – Graphic Example



The Fich et al. Leader – Algorithm

Algorithm

method `next ()`
    $elbow[0] \leftarrow \pi(elbow[1])$

method `next (`r`)`
    if $r = 1$ **then**
        `next ()`
    else
        while $elbow[r-1] < elbow[r-2]$ **do**
            $elbow[r-1] \leftarrow \pi(elbow[r-2])$
            `next (`$r-1$`)`
        while $elbow[r-1] > elbow[r-2]$ **do**
            $elbow[r-1] \leftarrow \pi(elbow[r-2])$
            `next (`$r-1$`)`

The Fich et al. Leader – Algorithm

Algorithm

method `isLeader` ($leader$)
    $elbow[1] \leftarrow leader$
    for $i \leftarrow 1$ **to** $\lceil \lg n \rceil$ **do**
        `next` (r)
        if $elbow[r] > elbow[r-1]$ **then**
            $elbow[r] \leftarrow elbow[r-1]$
            `next` (r)
            if $elbow[r] > elbow[r-1]$ **then**
                return false
            $elbow[r+1] \leftarrow elbow[r]$
        else
            return $elbow[r] \stackrel{?}{=} elbow[r-1]$

The Fich et al. Leader – Theorem

Theorem [Fich et al., 1995]

In the worst case, permuting an array of length n, given the permutation, can be done in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(\log^2 n)$ additional bits of storage.

A Naive Solution – Outline

1. Introduction
2. Background
3. A Naive Solution
   - Inverting In Place
   - Minimum Leader
   - Bit Vector
4. Breaking the Cycle
5. Conclusion

A Naive Approach to Inverting Permutations

- Same structure as the `permute` algorithm.
- Reverse cycles instead of rotating them.
- Same time and space complexity as permuting.
- Reversing a cycle mutates `pi` instead of an array.

Algorithm

method `invert()`
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if `isLeader(`i`)` **then**
      `reverse(`i`)`

- Invert the permutation in place.
- May change the cycle leader.
  - Can use the minimum leader.
  - Cannot use Fich et al. leader.

Reversing a Cycle

- A basic operation in inverting permutations.
- Can be thought of as inverting a cycle.
- Takes $\mathcal{O}(\log n)$ bits for pointers and $\mathcal{O}(\ell)$ time.
  - Takes $\mathcal{O}(n)$ time to reverse each cycle exactly once.

Algorithm

method `reverse`($leader$)
    $curr \leftarrow \texttt{pi}[leader]$
    $prev \leftarrow leader$
    while $curr \neq leader$ **do**
        $next \leftarrow \texttt{pi}[curr]$
        $\texttt{pi}[curr] \leftarrow prev$
        $prev \leftarrow curr$
        $curr \leftarrow next$
    $\texttt{pi}[leader] \leftarrow prev$

The Minimum Leader

- Same time and space complexity as permuting using minimum leader.
- Safe, but too slow.

The Minimum Leader – Result

Theorem

In the worst case, the standard representation of a permutation of length n can be replaced with its own inverse in $\mathcal{O}(n^2)$ time using $\mathcal{O}(\log n)$ extra bits of space.

Bit Vector – Outline

- Alternative method to invert a permutation.
- Improve the speed of minimum leader.
- The bit vector can be shrunk.
- A natural compromise between the space and time.

Using an n-bit Vector

- Easy to invert using an n-bit vector to mark moved positions.
- Discover the "next" cycle by finding the next unset bit.
- Invert in $\mathcal{O}(n)$ time using $\mathcal{O}(n)$ bits. – Not in place.
  - Takes $n + \mathcal{O}(\log n)$ bits for the bit vector and pointers.
  - Traverses each cycle exactly one.
  - Traverses the bit vector exactly once.
- Leads to another algorithm using a sublinear number of extra bits.
- Equivalent to using the minimum leader.

Using a smaller b-bit Vector

- Divide the permutation into n/b evenly spaced sections.
- Apply the b-bit vector to one section at a time.
- A trade-off between the extra space required and the worst case time.
- In `invert`, an outer pointer iterates through the permutation.
- As a cycle is being tested for minimum leader, any position found in the current section is marked by setting its bit.
- When a marked bit is found, that position is known to not be the minimum leader.
- When the outer pointer enters a new section, reset the bit vector.

Using a b-bit Vector – Algorithm

Algorithm

method `isLeader` ($leader$)

    if $leader \bmod b = 0$ **then**

        `clear` (v)

    if $v(leader \bmod b) = 1$ **then**

        return false

    $i \leftarrow \texttt{pi}[leader]$

    while $i > leader$ **do**

        if $\lfloor i/b \rfloor = \lfloor leader/b \rfloor$ **then**

            $v(i \bmod b) \leftarrow 1$

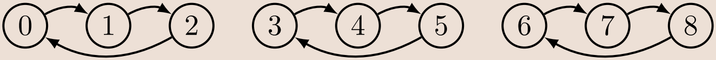
        $i \leftarrow \texttt{pi}[i]$

    return $i \stackrel{?}{=} leader$

Using a b-bit Vector – Analysis

- Takes $b + \mathcal{O}(\log n)$ bits for the b-bit vector and pointers.
- Runs in $\mathcal{O}(n^2/b)$ time in the worst case.
  - Tests each cycle no more than $\lceil n/b \rceil$ times.
- In example, $k = 3$.

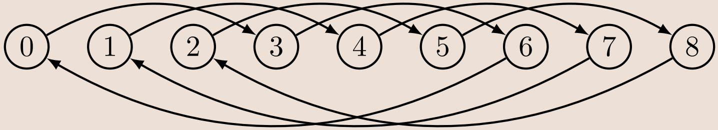
Example



Using a b-bit Vector – Analysis

- Takes $b + \mathcal{O}(\log n)$ bits for the b-bit vector and pointers.
- Runs in $\mathcal{O}(n^2/b)$ time in the worst case.
  - Tests each cycle no more than $\lceil n/b \rceil$ times.
- In example, $k = 3$.

Example



Using a b-bit Vector – Theorem

Theorem

In the worst case, the standard representation of a permutation of length n can be replaced with its own inverse in $\mathcal{O}(n^2/b)$ time using $b + \mathcal{O}(\log n)$ extra bits of space.

A Natural Compromise

- A very natural compromise between the space and time is achieved by setting $b = \sqrt{n}$.

A Natural Compromise – Result

Theorem

In the worst case, the standard representation of a permutation of length n can be replaced with its own inverse in $\mathcal{O}(n\sqrt{n})$ time using $\mathcal{O}(\sqrt{n})$ extra bits of space.

Breaking the Cycle – Outline

1. Introduction
2. Background
3. A Naive Solution
4. Breaking the Cycle
   - The Strategy
   - Bad Cycles
   - Sentinel Value
   - Unique Cycle Lengths
   - The "$\mho$" Structure
5. Conclusion

The Strategy

- The strategy allows using Fich et al. leader to invert a permutation.
- When the leader of a cycle is detected, reverse it normally.
- If the reversed cycle will be detected again, mark it “do not reverse.”
- Mark these badly behaved cycles by breaking them in a way that can be detected and recovered easily.
- When the leader of a broken cycle is detected, restore the cycle.

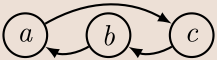
What is a Bad Cycle?

Definition

A **bad cycle** is a cycle with the property that if reversed, has a new cycle leader not yet processed, i.e., larger than the original leader.

- Position b is the Fich et al. leader of the cycle.
- If reversed, c is the leader of the new cycle.
- Since $c > b$, the naive `invert` would reverse the cycle twice.
- A permutation can contain up to $\lfloor n/3 \rfloor$ bad cycles.

Example



$$a < b < c$$

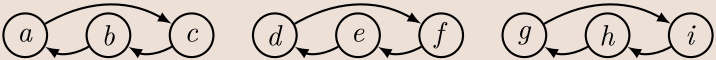
What is a Bad Cycle?

Definition

A **bad cycle** is a cycle with the property that if reversed, has a new cycle leader not yet processed, i.e., larger than the original leader.

- Position b is the Fich et al. leader of the cycle.
- If reversed, c is the leader of the new cycle.
- Since $c > b$, the naive `invert` would reverse the cycle twice.
- A permutation can contain up to $\lfloor n/3 \rfloor$ bad cycles.

Example



$a < b < c$ $\quad$ $d < e < f$ $\quad$ $g < h < i$

Detecting Bad Cycles

- Fich et al. leader naturally reveals the leader of the reversed cycle.
- Avoid re-testing each position of the reversed cycle for leadership.
- $x_d = \alpha_d \circ \alpha_{d-1} \circ \ldots \circ \alpha_0(i)$, because i is the leader of α.
- $j = \alpha_0 \circ \alpha_1 \circ \ldots \circ \alpha_d(x_d)$ is the leader of α^{-1}, because...
- $\alpha_d^{-1} \circ \alpha_{d-1}^{-1} \circ \ldots \circ \alpha_0^{-1}(j) = x_d$
- If position $j > i$, then α is a bad cycle.

Example

$$\alpha_0 = (0\ 6\ 1\ \vec{8}\ 3\ 4\ 2\ 9\ 5\ \overleftarrow{7})$$

$$\alpha_1 = (0\ 1\ \vec{3}\ 2\ \overleftarrow{5})$$

$$\alpha_2 = (0\ \overleftarrow{\vec{2}})$$

$$\alpha_3 = (\overleftarrow{\vec{0}})$$

Breaking Bad Cycles at the Tail

Definition

The **tail** of a cycle is the predecessor of the leader, the position i such that $\pi(i)$ is determined to be the leader of the cycle.

- The choice of where to break the cycle is important.
- Breaking a cycle at the tail allows easy recovery.
- Find the tail by computing $\pi(j)$ before reversing α.
- Simulate that the tail points to the potential leader.
- When the leader of a broken cycle is detected, point the tail to it.

Using a Sentinel Value

- Set the tail of a bad cycle to the sentinel value $\emptyset$.
- When $\texttt{pi}[i] = \emptyset$ is encountered, return $leader$.
- The alphabet needs to be increased by 1.
- In the worst case, the word size needs to be increased by 1.
- Just as bad as using an n-bit vector to mark positions.

Snippet

method $\texttt{get}\,(i)$
    if $\texttt{pi}[i] = \emptyset$ **then**
        return $leader$
    return $\texttt{pi}[i]$

Simulating a Sentinel Value

Instead of storing the sentinel value $\emptyset$, simulate it by:

1. Using 0 and storing the true position of the 0 pointer.
   - When $\texttt{pi}[i] = 0$, check against $zeroPointer$.
   - If the sentinel value and zero pointer are the same, lazy delete it.
2. Setting $\texttt{pi}[i] \leftarrow i$.
   - Finding $i \neq \texttt{pi}[i]$ but $\texttt{pi}[i] = \texttt{pi}[\texttt{pi}[i]]$.

A Problem...

- This strategy does not always work.
- The left shows a cycle being restored correctly.
- The right shows a cycle being restored incorrectly.
  - Position 2 is determined to be a leader before 3 is tested.

Example

- A more intelligent breaking technique is required.

Unique Cycle Lengths – Outline

- The limited number of unique cycle lengths, c, can be exploited in a technique for breaking bad cycles.
- Break bad cycles by pointing their tail to the rank of their length.
- Store the true positions of the first c pointers.
- Detect a broken cycle by encountering a cycle length rank.
- Supports `invert` in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(\sqrt{n} \log n)$ extra bits.

Number of Unique Cycle Lengths

Theorem

A permutation of length n contains $c \leq \left\lfloor \sqrt{2n} \right\rfloor$ unique cycle lengths.

Proof

Consider a permutation of length n with $c \geq \left\lceil \sqrt{2n} \right\rceil$ unique cycle lengths. The total length of the permutation must be at least

$$\sum_{i=1}^{\left\lceil \sqrt{2n} \right\rceil} i \geq \frac{2n + \sqrt{2n}}{2} > n \ ,$$

a contradiction. □

Break Cycles with the Rank of Their Length

- Break a reversed cycle by pointing the tail to the rank of the cycles' length, instead of back to the leader.
- Store the true positions of the first c pointers.

Example

$\pi(i)$	0	1	2
i	4	7	5

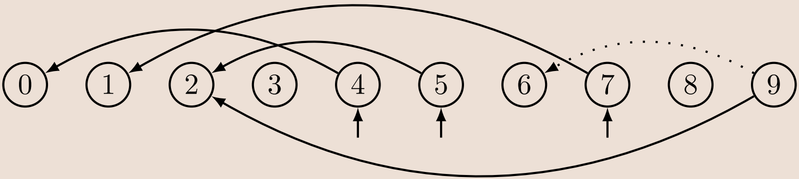


Table of True Positions

- Keep a table of the true positions, T, of the first c pointers.
- The table can be stored in $\mathcal{O}(\sqrt{n}\log n)$ bits.
- Initialize the table before reversing any cycles in $\mathcal{O}(n)$ time.
- Update the table when a cycle is reversed in $\mathcal{O}(\ell)$ time.
  - Support update in $\mathcal{O}(n)$ time in total.
- If the rank and pointer happen to be equal, lazy delete it.
  - Support lazy deletion from the table in exactly c bits.
- The table can be accessed in $\mathcal{O}(1)$ time.

Tree of Cycle Length Ranks

- Keep a balanced search tree of the unique cycle lengths.
- The tree can be stored in $\mathcal{O}(\sqrt{n}\log n)$ bits.
- Build the tree in $\mathcal{O}(n\log n)$ time by inserting lengths when discovered.
- The tree can be searched in $\mathcal{O}(\log n)$ time.
    - No more than one search per position, totalling $\mathcal{O}(n\log n)$ time.
- The rank of a cycle length is its rank in this tree.

Detect Broken Cycles

- A cycle is determined to be broken when a position that points to the rank of a cycle length is encountered.
- A real pointer and a rank can be distinguished in $\mathcal{O}(1)$ time.
  - If $\mathtt{pi}[i] < c$ and $T[\mathtt{pi}[i]] \neq i$, then i must point to a rank.
  - Otherwise, position i is a real pointer.
- When a broken cycle is detected, abort the leader method.
- Broken cycles need to be restored.

Restore Broken Cycles

- When a broken cycle is detected, it needs to be restored.
- It is possible to encounter a broken tail having not started at the leader.
- Search the tree for the length so far, ℓ, in $\mathcal{O}(\log n)$ time.
  - If $\mathrm{rank}(\ell) = \mathtt{pi}[i]$, then point $\mathtt{pi}[i]$ to the leader.
  - Otherwise, do nothing and the real leader will be found later.
- Avoids the problem observed with using a sentinel value.
- Every broken cycle is restored exactly once and correctly.
- When `pi` has been completely iterated, every cycle has been restored.

Unique Cycle Lengths – Result

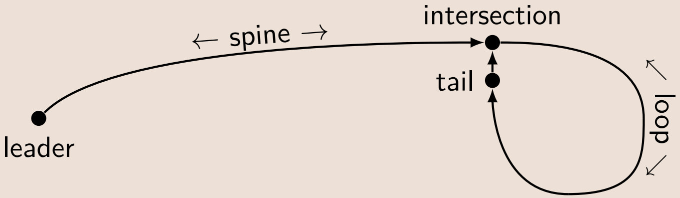
Theorem

In the worst case, the standard representation of a permutation of length n can be replaced with its own inverse in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(\sqrt{n} \log n)$ extra bits of space.

The "$\mho$" Structure – Outline

- Break a bad cycle by pointing the tail to itself.
- This turns the cycle into a structure with a spine and loop at the end.
- Updates to this $\mho$ structure will subsequently be performed.
- Detect a broken cycle by detecting a "cycle loop."
- The tail always points to some intersection within the structure.

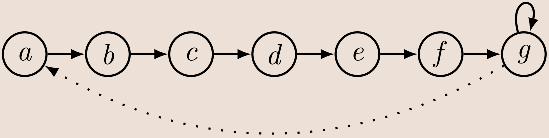
Figure



Break Cycles by Creating Trivial Loops

- When a bad cycle is detected, break it by setting $\texttt{pi}[\mathit{tail}] \leftarrow \mathit{tail}$.
- This creates a trivial cycle loop at the end of the ɒ structure.
- A broken cycle can be detected and "restored" easily:
  - Detect i where $i \neq \texttt{pi}[i]$ but $\texttt{pi}[i] = \texttt{pi}[\texttt{pi}[i]]$.
  - "Restore" by setting $\texttt{pi}[\texttt{pi}[i]] \leftarrow \mathit{leader}$.
- Position a is the leader of the complete cycle.
- Position g is the tail and the intersection.

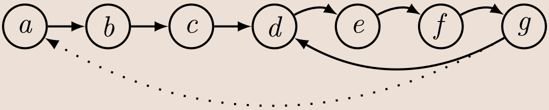
Example



Break Cycles by Creating Trivial Loops

- When a bad cycle is detected, break it by setting $\mathtt{pi}[tail] \leftarrow tail$.
- This creates a trivial cycle loop at the end of the ꝺ structure.
- A broken cycle can be detected and "restored" easily:
  - Detect i where $i \neq \mathtt{pi}[i]$ but $\mathtt{pi}[i] = \mathtt{pi}[\mathtt{pi}[i]]$.
  - "Restore" by setting $\mathtt{pi}[\mathtt{pi}[i]] \leftarrow leader$.
- Position a is the leader of the complete cycle.
- Position g is the tail, both to the leader a and intersection d.
- Position d is the erroneous leader of the cycle.

Example



Detecting Nontrivial Cycle Loops

- A cycle loop is a subcycle of the cycle, excluding the original cycle.
- Cycle loops can be detected using the classic "the tortoise and the hare" [Knuth, 1969] algorithm.
  - The tortoise traverses the cycle at 1 step per call.
  - The hare traverses the cycle at 2 steps per call.
  - If the cycle contains a loop, the tortoise and hare will rendezvous.
  - The intersection where the cycle loops back on itself can be found.
- Run the Fich et al. leader method and loop detection simultaneously.
- The loop detection method needs to find the intersection before the Fich et al. leader method encounters it.
  - So the leader method can simulate that the tail points to the leader.

Interleaving the Scans

These processes can be run simultaneously by interleaving the scans:

$\mathcal{F}$. The Fich et al. leader scan.
  - This scan proceeds the slowest at 1 step per call.

$\mathcal{T}$. The "tortoise" scan.
  - This scan proceeds at 4 steps per call.

$\mathcal{H}$. The "hare" scan.
  - This scan proceeds at 2 steps per call.

The $\mathcal{F}$ Scan

The $\mathcal{F}$ scan will terminate on one of the following 3 cases:

1. $\mathcal{F}$ determines the position is not a leader. Then the entire process of all three scans is aborted.
2. $\mathcal{F}$ determines the position is the leader of a complete cycle. Then the reverse, and perhaps broken, cycle replaces the current cycle.
3. $\mathcal{F}$ determines the position is the leader of a cycle whose tail is a position already determined by $\mathcal{T}$ and $\mathcal{H}$. Then mutate the tail to point to the current position, enlarging the loop of the $\mathfrak{d}$ and potentially eliminating the spine.

The $\mathcal{T}$ and $\mathcal{H}$ Scans

The $\mathcal{T}$ and $\mathcal{H}$ scans have two phases:

1. The first phase is to detect if the cycle is broken.
   - If $\mathcal{H}$ returns to the starting position, the cycle is not broken. $\mathcal{T}$ and $\mathcal{H}$ are aborted and $\mathcal{F}$ continues until one of its first 2 cases happen.
   - Otherwise, $\mathcal{T}$ and $\mathcal{H}$ meet at some position, i.e. there is a position in common in the 4:2 (or possibly 2:1) steps taken in the current call. If this happens, the cycle is broken so advance to phase 2.
2. The second phase is to find the tail of the cycle.
   - Reset $\mathcal{T}$ to the start and decrease the speed of $\mathcal{H}$ to 2 steps per call.
   - When $\mathcal{T}$ and $\mathcal{H}$ meet for a second time, the tail can be identified.
   - $\mathcal{F}$ is left to continue, either to abort from a position that is determined not to be a cycle leader or to increase the size of the loop of the $\mathfrak{o}$.

Tail Detection

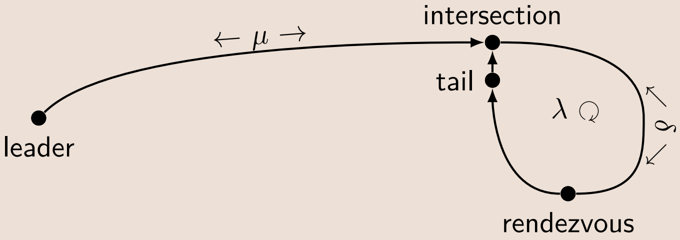
Theorem

The tail of the broken cycle α can be identified in less than 2ℓ iterations, where ℓ is the cycle length of α.

- The first phase will take less than ℓ iterations.
  - The tortoise will either rendezvous with the hare, or return to the start.
- The second phase will take less than ℓ iterations.
  - The length of the spine cannot be longer than the cycle.

Tail Detection – Figure

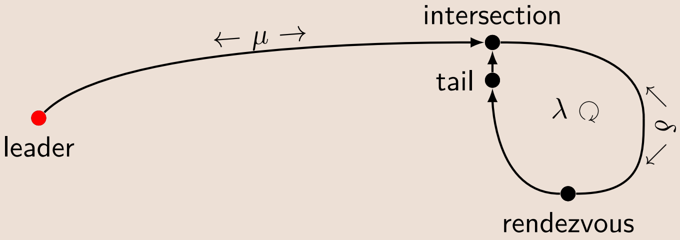
Figure



$$d_t = \frac{1}{2} d_h$$
$$2(\mu + \delta) = \mu + k\lambda + \delta$$
$$\mu = k\lambda - \delta$$

Tail Detection – Figure

Figure



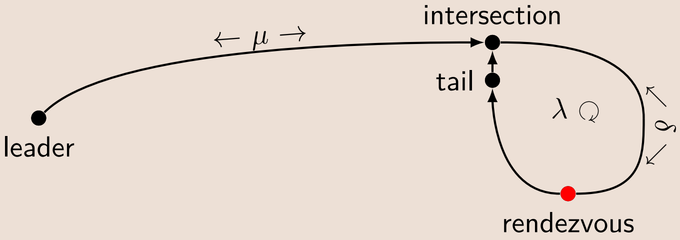
$$d_t = \frac{1}{2} d_h$$

$$2(\mu + \delta) = \mu + k\lambda + \delta$$

$$\mu = k\lambda - \delta$$

Tail Detection – Figure

Figure



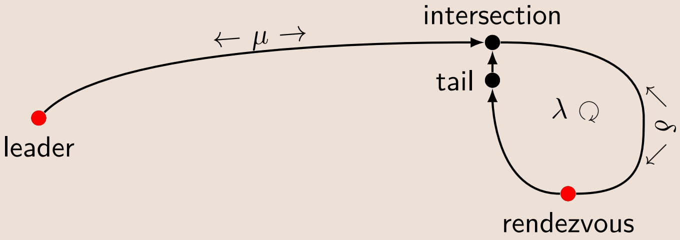
$$d_t = \frac{1}{2} d_h$$

$$2(\mu + \delta) = \mu + k\lambda + \delta$$

$$\mu = k\lambda - \delta$$

Tail Detection – Figure

Figure



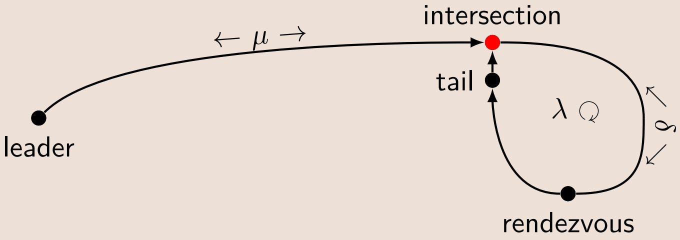
$$d_t = \frac{1}{2} d_h$$

$$2(\mu + \delta) = \mu + k\lambda + \delta$$

$$\mu = k\lambda - \delta$$

Tail Detection – Figure

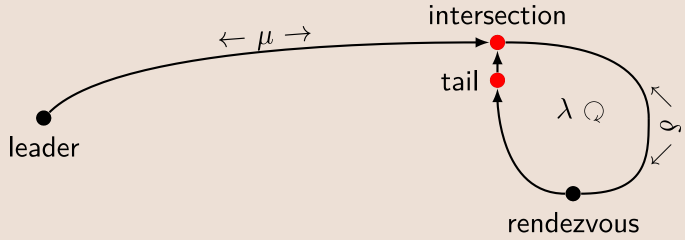
Figure



$$d_t = \frac{1}{2} d_h$$
$$2(\mu + \delta) = \mu + k\lambda + \delta$$
$$\mu = k\lambda - \delta$$

Tail Detection – Figure

Figure



$$d_t = \frac{1}{2} d_h$$
$$2(\mu + \delta) = \mu + k\lambda + \delta$$
$$\mu = k\lambda - \delta$$

Implementation – Loop and Tail Detection

Algorithm

method `detectCycleLoop()`

  switch $state$ **do**

    case SEARCHING_FOR_LOOP

      $t \leftarrow \texttt{pi}[t]$

      $h \leftarrow \texttt{pi}[\texttt{pi}[h]]$

      if $t = h$ **then**

        $state \leftarrow$ LOOP_DETECTED

        $t \leftarrow \texttt{pi}[t]$

      if $h = leader$ **then**

        $state \leftarrow$ CYCLE_NOT_MARKED

    case LOOP_DETECTED

      if $t = \texttt{pi}[h]$ **then**

        $tail \leftarrow h$

        $state \leftarrow$ IDENTIFIED_TAIL

      $t \leftarrow \texttt{pi}[t]$

      $h \leftarrow \texttt{pi}[h]$

Implementation – Next Snippet

- Run two steps of cycle loop detection per one step of Fich et al. leader.
- Replace the `next` method from the original Fich et al. leader method.

Snippet

method `next()`
  `detectCycleLoop()`
  `detectCycleLoop()`
  $elbow[0] \leftarrow \texttt{pi}[elbow[1]]$
  if $state =$ IDENTIFIED_TAIL **and** $elbow[1] = tail$ **then**
    $elbow[0] \leftarrow leader$

Implementation – Invert Algorithm

Algorithm

method `invert()`
    for $leader \leftarrow 0$ **to** $n-1$ **do**
        $state \leftarrow$ SEARCHING_FOR_LOOP
        $t \leftarrow h \leftarrow leader$
        if `isLeader`($leader$) **then**
            $newLeader = elbow[0]$
            switch $state$ **do**
                case IDENTIFIED_TAIL
                    `pi`$[tail] \leftarrow leader$
                case CYCLE_NOT_MARKED
                    if $newLeader > leader$ **then**
                        $newTail \leftarrow$ `pi`$[newLeader]$
                        `reverse`($leader$)
                        `pi`$[newTail] \leftarrow newTail$
                    else
                        `reverse`($leader$)

Main Result

Theorem

In the worst case, the standard representation of a permutation of length n can be replaced with its own inverse in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(\log^2 n)$ extra bits of space.

Conclusion – Results

In the worst case, the standard representation of a permutation of length n can be replaced with its own inverse in:

- $\mathcal{O}(n^2)$ time, using $\mathcal{O}(\log n)$ extra bits.
- $\mathcal{O}(n\sqrt{n})$ time, using $\mathcal{O}(\sqrt{n})$ extra bits.
- $\mathcal{O}(n \log n)$ time, using $\mathcal{O}(\sqrt{n} \log n)$ extra bits.
- $\mathcal{O}(n \log n)$ time, using $\mathcal{O}(\log^2 n)$ extra bits.

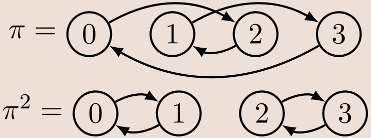
Future Work on a Better Leader Method

- A better leader method will lead to a better `invert` algorithm.
- The $\mathfrak{v}$ technique can be independent of the leader method.
  - Bad cycles were detected by exploiting a property of Fich et al. leader.
  - Instead, traverse the reversed cycle testing for the new leader.
- Overhead added by the $\mathfrak{v}$ technique is $\mathcal{O}(\log n)$ bits and $\mathcal{O}(n)$ time.
- No leader method could use less resources and look at every position.

Future Work on Other Transformations

- Apply an arbitrary power to a permutation in place.
- [Fich et al., 1995] can apply an arbitrary power of permutation.
- Rotate each cycle several steps, similar to transposing an array.
- A cycle may split into several smaller cycles.
- Could be done with maximum leader, but too slow.

Example



Acknowledgments

- Ian Munro
- Jonathan Buss
- Gord Agnew
- Hicham El-Zein

Thank You!

References

- Fich, F. E., Munro, J. I., and Poblete, P. V. (1995).
  Permuting in place.
  SIAM Journal on Computing, 24(2):266–278.
- Knuth, D. E. (1969).
  The Art of Computer Programming, Volume II: Seminumerical Algorithms.
  Addison-Wesley.