EC 5: Cycle and Component Detection in Signed Permutations

Problem Statement

Let π be a signed permutation of length n. We are interested in analyzing the cycle and component structure of π using a cycle graph representation.

(a) Show that the number of cycles in π can be found in $\mathcal{O}(n)$ time.

(b) Show that all components in π can be found in $\mathcal{O}(n)$ time.

Solution

(a) Computing the Number of Cycles in $\mathcal{O}(n)$ Time

We start by transforming the signed permutation π into a form suitable for constructing a cycle graph.

Step 1: Extend and Transform the Permutation

1. Extend π to include sentinel elements:

$$\pi(0) = 0, \quad \pi(2n+1) = 2n+1.$$

2. Convert each signed element of π into an unsigned pair:
   (a) If $\pi_i > 0$, create the pair $(2\pi_i - 1, 2\pi_i)$.
   (b) If $\pi_i < 0$, create the pair $(2|\pi_i|, 2|\pi_i| - 1)$.

   This results in a permutation defined on the set $\{0, 1, 2, \ldots, 2n+1\}$ with $2n+2$ vertices.

Step 2: Construct the Cycle Graph

1. Define **gray edges** between $\pi(2i)$ and $\pi(2i+1)$.
2. Define **black edges** between $2i$ and $2i+1$.
3. The graph is composed of cycles with alternating edge colors.
4. Two cycles overlap when their corresponding intervals intersect, e.g., $(\pi(1), \pi(2))$ and $(\pi(3), \pi(4))$.

Step 3: Detect Overlap Components in $\mathcal{O}(n)$ Time

1. Build the **overlap graph** where each vertex represents a pair and edges indicate overlapping intervals.
2. Construct a forest where each tree corresponds to a connected component (i.e., a cycle).
3. Each edge and vertex is processed once, resulting in linear time complexity.

Conclusion: The number of cycles in π is the number of connected components in the overlap graph, computable in $\mathcal{O}(n)$ time.

(b) Computing All Components in $\mathcal{O}(n)$ Time

We can use a stack-based algorithm to identify all trees in the overlap forest, representing the components of π.

Algorithm

1. Label each position i with interval $C[i] = (A_i, B_i)$ from the unsigned pairs.
2. Initialize an empty stack.
3. For $i \leftarrow 0$ to $2n + 1$:
   (a) If $i = C[i].A$, push $C[i]$ onto the stack.
   (b) Let `extent` $\leftarrow C[i]$.
   (c) While the top of the stack satisfies `top.A > C[i].A`:
      i. Update `extent.A` $\leftarrow$ min(`extent.A`, `top.A`).
      ii. Update `extent.B` $\leftarrow$ min(`extent.B`, `top.B`).
      iii. Pop `top` and set `parent[top.A]` $\leftarrow C[i].A$.
   (d) Update `top.A` and `top.B` with current `extent` values.
   (e) If $i =$ `top.B`, then pop `top`.
4. After the loop, use the `parent[]` array to label each tree (i.e., connected component).

Conclusion: Each vertex and edge is processed at most once, so the total runtime is $\mathcal{O}(n)$. All components in π are detected in linear time.