Combinatorial Mathematics

Mong-Jen Kao (高孟駿)

Monday 18:30 – 21:20

Outline

- The RMQ Problem
- Cartesian Tree for Sequences
  - $O(n)$ Time Construction
  - Binary Encodings
- The Optimal Algorithm for the RMQ problem

The Range Minimum Query (RMQ) Problem

The RMQ Problem

- Given a sequence of numbers $a_1, a_2, \dots, a_n$,
  preprocess the sequence such that
  - For each $1 \leq \ell \leq r \leq n$,
    the minimum within $[a_\ell, \dots, a_r]$ *can be answered quickly*.

- Two factors of concern
  - The time / space it takes to preprocess the sequence
  - The time it takes to answer the query.

Existing Approaches for the RMQ Problem

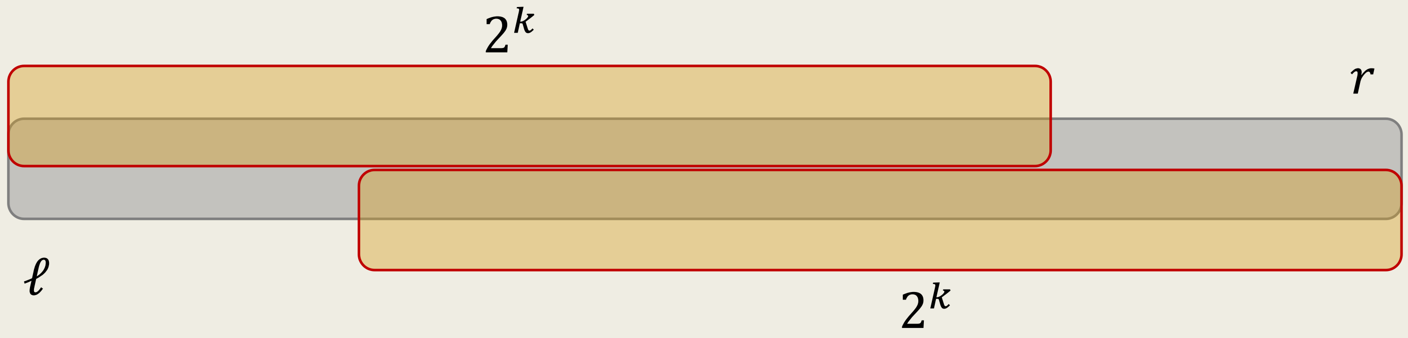
1. ***Precompute*** the answer *for all possible intervals*.
   - $O(n^2)$ for preprocessing, $O(1)$ for query
   - Simple, but not applicable when n is large.

2. ***Segment tree***
   - $O(n)$ for preprocessing, $O(\log n)$ for query
   - Support update in $O(\log n)$ time.
   - Simple to implement

Existing Approaches for the RMQ Problem

3. ***Sparse table***

   - Precompute the answer for $[\, i,\ i + 2^k - 1\,]$ and $[\, i - 2^k + 1,\ i\,]$ for all $1 \leq i \leq n$ and all $0 \leq k \leq \log n$.



   - $O(n \log n)$ time & space for preprocessing, $O(1)$ for query.

Existing Approaches for the RMQ Problem

4. Optimal algorithm
   - The optimal algorithm combines ideas from the above methods.
   - $O(n)$ for preprocessing, $O(1)$ for query.
   - Partition the sequence into groups of <u>*small size*</u>.
     - For each group, encode its structure and precompute the answer if it hasn't been computed before.
     - Precompute the min-value for all groups and apply Sparse table method on it.

Cartesian Tree & Binary Encoding

Cartesian Tree

Let $a_1, a_2, \ldots, a_n$ be a sequence. The *Cartesian Tree* for the sequence is defined as follows.

- The root of the tree is the element a_i that satisfies the property that $a_i < a_j$ for all $1 \leq j < i$ and
  $a_i \leq a_k$ for all $i < k \leq n$.
- The *left child* of a_i is the Cartesian tree for $a_1, \ldots, a_{i-1}$.
- The *right child* of a_i is the Cartesian tree for $a_{i+1}, \ldots, a_n$.

Building the Cartesian Tree in $O(n)$ Time

Let $a_1, a_2, \dots, a_n$ be a sequence.

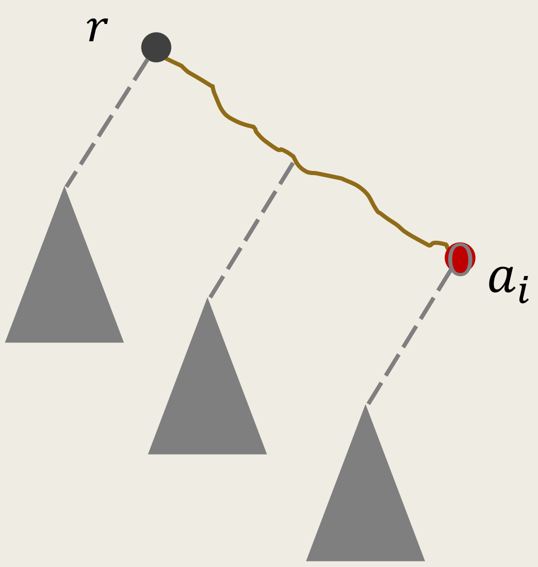
- Consider the elements one by one, e.g., $a_1, a_2, \dots, a_n$, in order.
  - Let T_i denote the Cartesian tree for $a_1, \dots, a_i$.
  - For each a_i considered,

    we will use T_{i-1} to build T_i in *amortized* $O(1)$ time.

Building the Cartesian Tree in $O(n)$ Time

Let $a_1, a_2, \ldots, a_n$ be a sequence.

- Consider the tree T_i for $a_1, \ldots, a_i$.



A key property for T_i is that

a_i must be *at the end* of the *right-most path* from the root.

Let $a_1, a_2, \dots, a_n$ be a sequence.

- Consider the tree T_i for $a_1, \dots, a_i$.

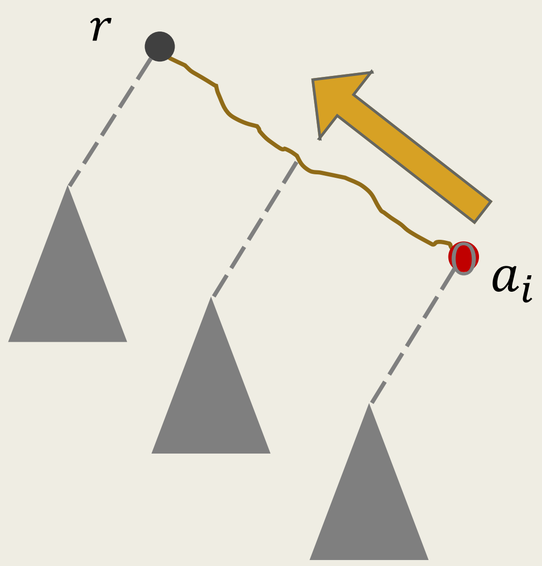
  <u>A key property</u> for T_i is that

  a_i must be *<u>at the end</u>* of the *<u>right-most path</u>* from the root.

- To construct T_{i+1},

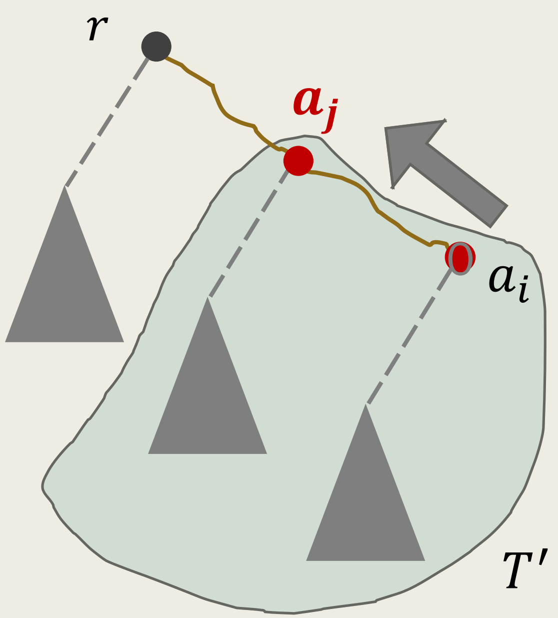
  it suffices to **walk-up the tree from a_i** until

  we reach the place where a_{i+1} belongs in T_{i+1}.

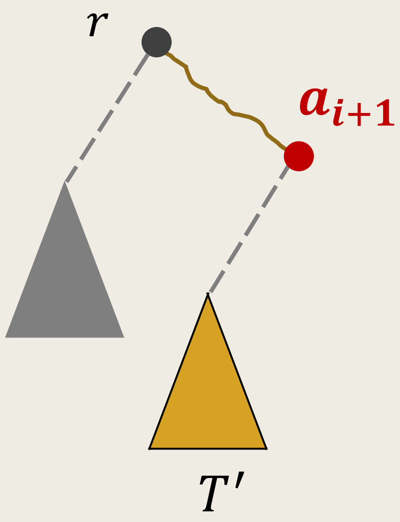


- To construct T_{i+1},

  it suffices to **walk-up the tree from a_i** until we reach the place where a_{i+1} belongs in T_{i+1}.



- Let a_j be the *first node* in T_i with $\boldsymbol{a_j \leq a_{i+1}}$ *when we walk-up from* a_i.
  - Then the subtree rooted at a_j should be the left-subtree of a_{i+1}, and a_{i+1} should be the right-child of $p(a_j)$.



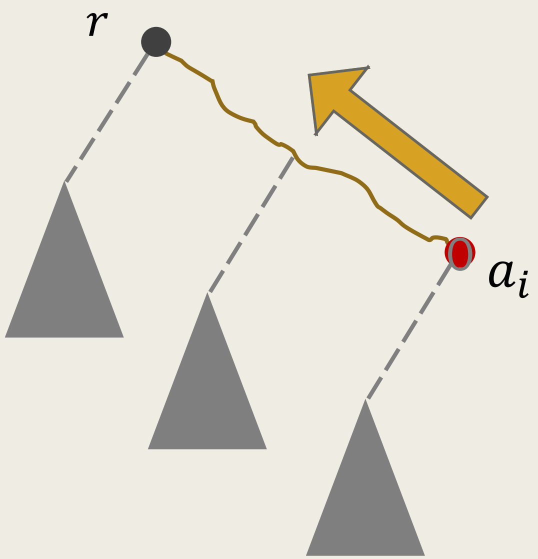
- To construct T_{i+1},

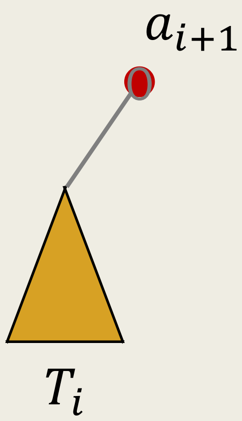
  it suffices to **walk-up the tree from $\boldsymbol{a_i}$** until we reach the place where a_{i+1} belongs in T_{i+1}.

- Let a_j be the <u>*first node*</u> in T_i with $\boldsymbol{a_j \leq a_{i+1}}$ *when we walk-up from a_i*.

  - If there is no such node, i.e., $a_{i+1} < r$,

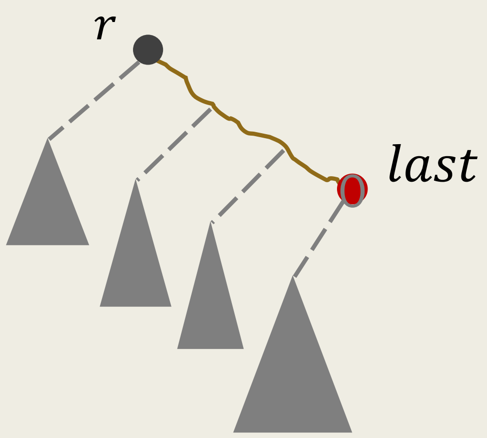
    then a_{i+1} should be the new root.





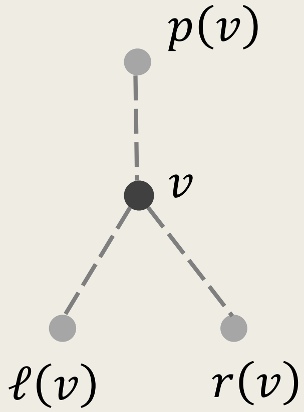
Building the Cartesian Tree in $O(n)$ Time

To describe the algorithm formally,



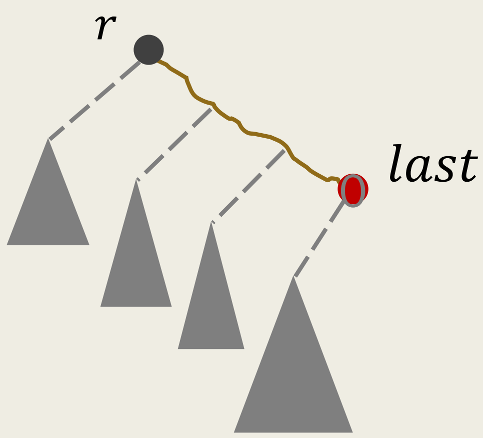
- Let T be the current tree.
  - Let r be the root node of T.
  - Let $last$ be the last node inserted into T.

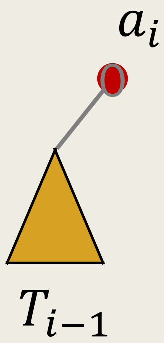
- For any $v \in T$,
  - Let $p(v)$ denote the parent of v.
  - Let $\ell(v)$, $r(v)$ denote the left- and right-child of v.

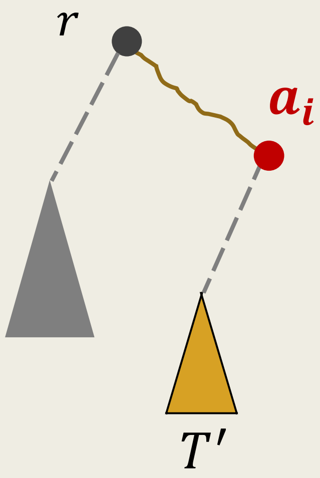


- Initially, $T = r = last = NiL$.
- For $i = 1,2,\ldots,n$ do the followings.
  - Create node v for a_i with $p(v) = \ell(v) = r(v) = NiL$.
  - While $last \neq NiL$ and $\mathrm{val}(last) > a_i$, do the following.
    - $last \leftarrow p(last)$.
  - If $last$ is equal to NiL, then
    - Set $p(r) \leftarrow v$, and $r \leftarrow v$.

    Else,
    - Set $p(v) \leftarrow p(last)$, $p(last) \leftarrow v$, $r(p(last)) \leftarrow v$.
  - Set $\ell(v) \leftarrow last$ and $last \leftarrow v$.





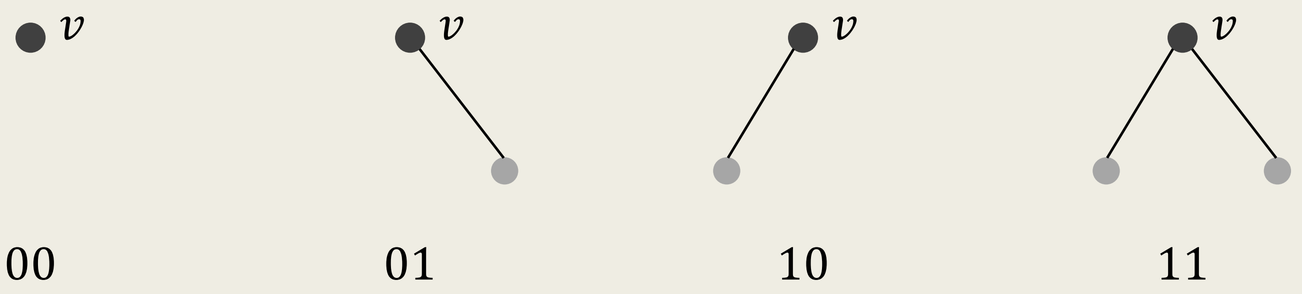


Binary Encoding of Cartesian Trees

- It is not difficult to show that,
  - The number of possible Cartesian trees with k vertices is equal to the k^{th}-Catalan number, which is $\frac{1}{\mathrm{k+1}}\binom{2k}{k} = O(4^k)$.
- Hence, it is possible to encode the Cartesian trees with a binary string of length $2k$.
  - The encodings can be used to uniquely identify a Cartesian tree.

Binary Encoding of Cartesian Trees

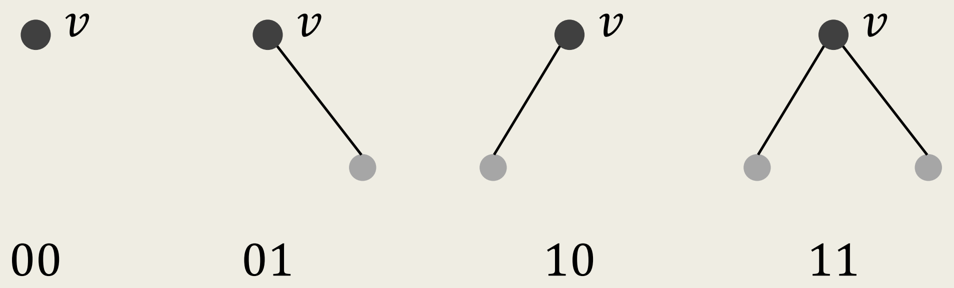
- Encoding a Cartesian tree T is fairly straightforward.
  - For any $v \in T$, distinguish the status of v with $\{0,1\}^2$.



  - Simply dump the status of the nodes in a *fixed* and *consistent* order, e.g., the order given by DFS or BFS traversal.

Binary Encoding by DFS Traversal

- Procedure DFS(v)
  - Print the status of v.
  - If $\ell(v) \neq NiL$, then $\mathrm{DFS}\big(\ell(v)\big)$.
  - If $r(v) \neq NiL$, then $\mathrm{DFS}\big(r(v)\big)$.



- With the above procedure,
  - To the tree, we simply call $\mathrm{DFS}(r)$.

Binary Encoding of Cartesian Trees

- The way of encoding is not unique.

- For example, the following procedure also gives a valid encoding.

  Procedure DFS'(v)

  – If $\ell(v) \neq NiL$, then print '1' and $\mathrm{DFS}\big(\ell(v)\big)$.
    Otherwise, print '0'.

  – If $r(v) \neq NiL$, then print '1' and $\mathrm{DFS}\big(r(v)\big)$.
    Otherwise, print '1'.

Optimal Algorithm for RMQ

Optimal RMQ - Preprocessing

Let $A = a_1, a_2, \ldots, a_n$ be a sequence.

1. Pick $s \approx \frac{\log n}{4}$.
   - W.L.O.G., assume that $n = M \cdot s$ for some integer M.
     (if not, add arbitrary numbers to make it so.)

2. Divide A into M groups,
   i.e., $A_i \coloneqq [\, a_{is}, a_{is+1}, \ldots, a_{is+s-1} \,]$ for all $0 \leq i < M$.

Optimal RMQ - Preprocessing

Let $A = a_1, a_2, \ldots, a_n$ be a sequence.

Pick $s \approx \frac{\log n}{4}$ and divide A into $A_1, A_2, \ldots, A_M$ where $n = M \cdot s$.

3. Let $\mathrm{idx}_i \coloneqq \mathrm{enc}(A_i)$ be the encoding of the Cartesian tree T_i for A_i.

4. Precompute and store the answer for the RMQ query for T_i if it hasn't been computed yet.

5. Let $B = b_1, b_2, \ldots, b_M$ be the minimum value in $A_1, A_2, \ldots, A_M$. Apply sparse table method on B.

Optimal RMQ - Query

Let $[\,\ell, r\,]$ be the query to be answered.

- We have two types of queries.

Use the precomputed table, e.g., $\mathrm{RMQ}(\mathrm{idx}_i, \ell', r')$ to answer this query directly in $O(1)$ time.

Let $[\,\ell, r\,]$ be the query to be answered.

- We have two types of queries.

Use the sparse table precomputed for $b_1, \dots, b_\mathrm{M}$ to find the minimum within this part in $O(1)$ time.

Use the precomputed tables $\mathrm{RMQ}(\mathrm{idx}_i, ?, ?)$ and $\mathrm{RMQ}(\mathrm{idx}_j, ?, ?)$ to find the minimum within these two parts in $O(1)$ time.

The Analysis

Let $A = a_1, a_2, \dots, a_n$ be a sequence and pick $s \approx \frac{\log n}{4}$.

- Time & Space complexity for preprocessing
  - Sparse table

$$O\left(\frac{n}{\log n} \cdot \log \frac{n}{\log n}\right) = O(n - \log\log n) = O(n)\,.$$

  - Solution Table for all Cartesian trees

$$O\left(4^s \cdot s^2\right) = O\left(\sqrt{n} \cdot \log^2 n\right) = O(n)\,.$$