Weekplan:
Nearest Common Ancestors and Range Minimum Queries

Hjalte Wedel Vildhøj

References and Reading

[1] The LCA problem revisited, M. A. Bender, M. Farach-Colton, Latin American Symposium 2000.

[2] Scribe notes from MIT.

[3] Fast Algorithms for Finding Nearest Common Ancestors, D. Harel and R. E. Tarjan, SIAM J. Comput., 13(2), 338–355.

We recommend reading [1] and [2] in detail.

Exercises

1 [w] **Range X Queries** We saw how to support *range minimum queries* on an array A of n elements in linear space and constant time. Try to support the following similar queries on A:

- Range *Maximum* Queries
- Range *Sum* Queries
- Range *Median* Queries

Let S be a set and c be a constant, and consider a function $f : S \mapsto [n^c]$. Formulate a *general and sufficient condition* for supporting *range f queries* in linear space and constant time. Such a query takes indicies $1 \leq i \leq j \leq n$ and returns $f(\{A[i], A[i+1], \ldots, A[j]\})$.

2 **2D Range Emptiness Queries** Let $P = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subset [1, n]^2$ be a set n points of an $n \times n$ grid. Let $1 \leq x_{\min} \leq x_{\max} \leq n$ and $1 \leq y_{\max} \leq n$. A *3-sided range emptiness query* on P is defined as follows:

- Empty$(x_{\min}, x_{\max}, y_{\max})$: Returns YES if $([x_{\min}, x_{\max}] \times [1, y_{\max}]) \cap P = \emptyset$ and NO otherwise.

Give a data structure for P that supports efficient 3-sided range emptiness queries.

3 **Distance Queries in Trees** Let T be a unrooted tree in which each edge has an integer weight. The *distance* between two nodes u and v is the sum of edge weights on the path between u and v. Give a linear-space data structure for T that can report the distance between any pair of nodes in constant time.

4 **Rank Queries** Let $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$ be a bit vector. A *rank query* on x takes an index $1 \leq i \leq n$ and returns the number of 1-bits among first i bits in $\mathbf{x}$, i.e., $\text{rank}(i) = \sum_{j=1}^{i} x_i$.

- [w] Suppose we have a data structure supporting rank queries on $\mathbf{x}$. Provided we care about the *value* and not the *index*, show how to support RMQ on $\mathbf{x}$, using no additional space.
- Assume a word size of $\log n$ bits. Give a data structure for $\mathbf{x}$ that supports constant-time rank queries using
  - $O(n \log n)$ *bits* of space, i.e., $O(n)$ words.
  - $O(n)$ bits of space.
  - (**) $O(n \log \log n / \log n)$ bits of space (assuming a read-only copy of $\mathbf{x}$ is stored on the side).

5 Longest Common Prefixes Let S be a set of strings and $n = \sum_{x \in S} |x|$ be their total length. Give an $O(n)$-space data structure that supports the following query in constant time:

- LCP(i, j): Return the length of the longest common prefix of the two strings $x_i, x_j \in S$.

E.g., if $x_i = \texttt{algorithms}$ and $x_j = \texttt{alcohol}$ then LCP$(i, j) = |\texttt{al}| = 2$.

6 The Longest Common Extension Problem The *Longest Common Extension Problem* is to preprocess a string x of length n to support the following query:

- LCE(i, j): returns the length of the longest common prefix of the suffixes of x starting at positions i and j.

Give a reduction from the RMQ problem on bit strings (where we care about the *index*) to the LCE problem. Try to do it using no more than $O(1)$ additional space.

7 Cartesian Trees Give an efficient algorithm for constructing the Cartesian tree of an array with n elements.