EE 382V: Parallel Algorithms **Summer 2017**

Lecture 22: July 22

Lecturer: Vijay Garg *Scribe: Michael Spagon*

22.1 Introduction

This lecture covers **range minimum queries (RMQ)** and briefly touches the related problem of finding the **lowest common ancestor (LCA)** of a pair of nodes in a segmented tree.

We started to discuss **tree contraction algorithms** but did not finish by the end of the lecture.

22.2 Range Minimum Query (RMQ)

Given a fixed array $A[0 \ldots n\text{-}1]$ of n elements, and two specified indices $i \le j$, a **range minimum query (RMQ)** is used to find the *position* of the minimum element between the two indices.

A range minimum query on array A with the indices 2 and 7 would be denoted as $RMQ_A(2,7)$.

With respect to the array below, $RMQ_A(2,7) = 3$. The smallest element within the subarray $A[2 \ldots 7]$ is 1, and the element is in the third position of the array.

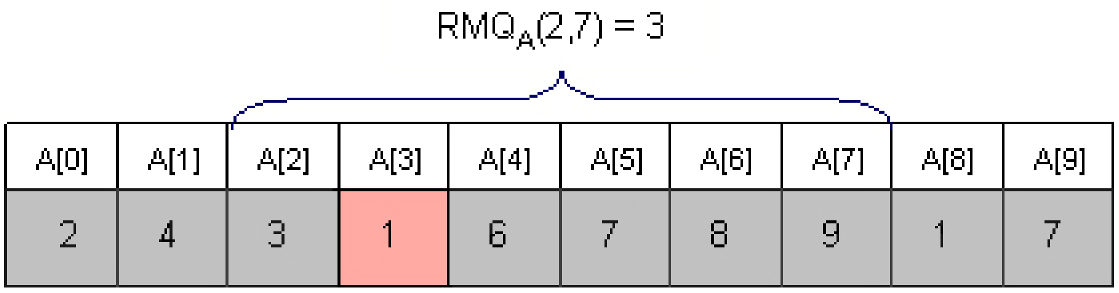
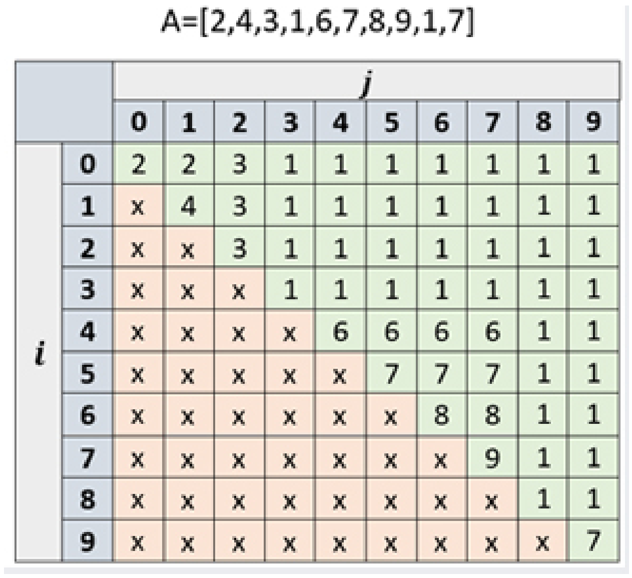


Figure 22.1: Range minimum query on A with index i = 2 and j = 7

22.2.1 A Naïve Solution

What if we precomputed all possible queries and stored the results in a data structure?

By creating a 2D array, B, such that $B[i,j] = min(A[i...j])$, a range min query can be solved in constant time $O(1)$ by array look-up in B. Below is a look-up table for the RMQ's on a 10 element array A.



A=[2,4,3,1,6,7,8,9,1,7]

		j									
		0	1	2	3	4	5	6	7	8	9
	0	2	2	3	1	1	1	1	1	1	1
	1	x	4	3	1	1	1	1	1	1	1
	2	x	x	3	1	1	1	1	1	1	1
	3	x	x	x	1	1	1	1	1	1	1
	4	x	x	x	x	6	6	6	6	1	1
i	5	x	x	x	x	x	7	7	7	1	1
	6	x	x	x	x	x	x	8	8	1	1
	7	x	x	x	x	x	x	x	9	1	1
	8	x	x	x	x	x	x	x	x	1	1
	9	x	x	x	x	x	x	x	x	x	7

Figure 22.2: Look-up table for $RMQ_A(i,j)$

As you can see this wastes a lot of space and is impractical for very large n: There are $O(n^2)$ possible queries for a length-n array and therefore $O(n^2)$ entries for B.

22.2.2 A Better Solution

Intuition: Instead of storing all possible ranges of indices, we should only store ranges that are a power of 2 (*i.e.*, $i = 2^a$ $j = 2^b$ where $a \leq b$.)

In this solution we will be using segmented trees and calculating the least common ancestor of two nodes within the tree, so we must discuss them first.

Segmented Tree

We will use a **segmented tree**, a structure for storing intervals, where:

1. Leaf nodes are the elements of array A.
2. Each internal node is labeled with indices i, j that represent information about the sub**segment** $A[i, j]$. Each internal node stores two additional arrays P and S that represent the prefix and suffix minima, respectively.

Below is a picture of a segmented tree that stores data that will help us perform RMQs on array $A =$ [3,1,5,7,2,8,6,4]. Notice that for this problem the array index starts at `1`.

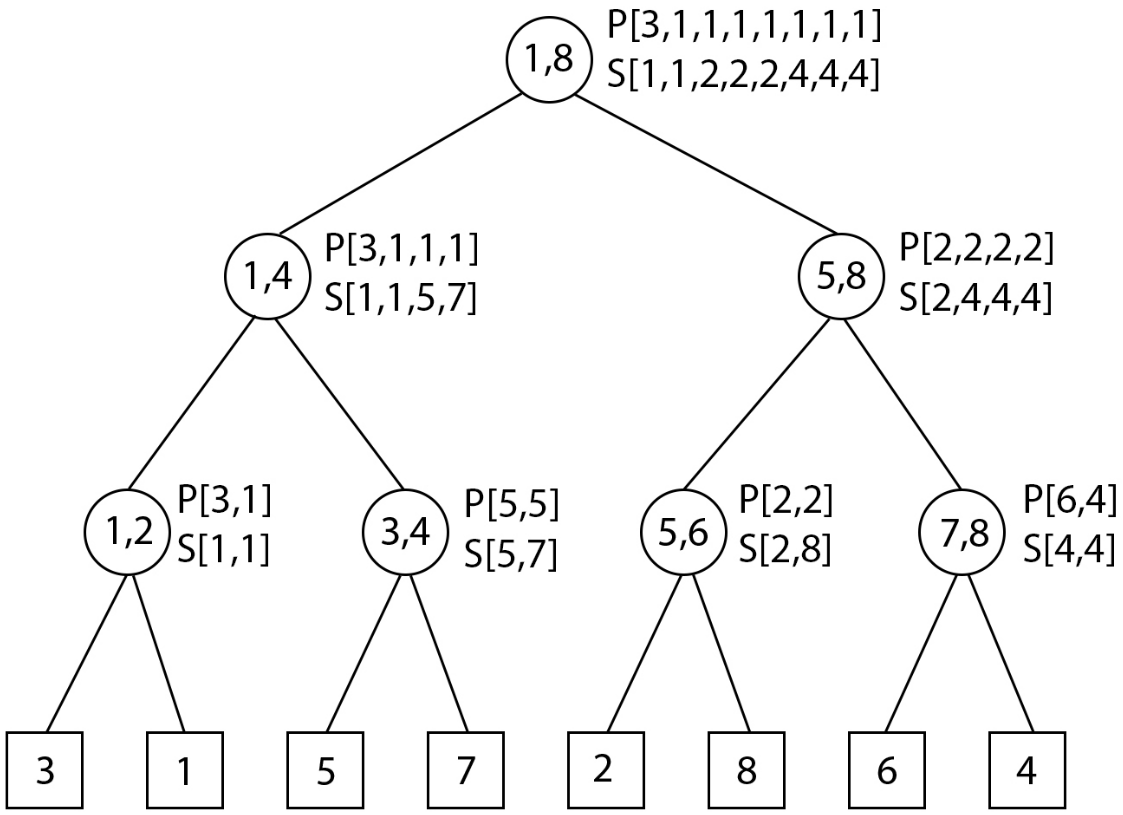


Figure 22.3: A segmented tree used for range minimum query

Prefix Minima Array

A **prefix minima** array P stores the current minimum value seen while traversing the input array from *left* to *right*.

For example, the internal node labeled "1,4" is associated with the subarray with indices $i = 1$ and $j = 4$. Therefore the input to `prefix minima` is $A[1, 4] = [3, 1, 5, 7]$.

- $P[1] = min(A[1, 1]) = min(3) = 3$
- $P[2] = min(A[1, 2]) = min(3, 1) = 1$
- $P[3] = min(A[1, 3]) = min(3, 1, 5) = 1$
- $P[4] = min(A[1, 4]) = min(3, 1, 5, 7) = 1$

The P array contains [3,1,1,1] since the minimum value in $A[1, 1]$ is 3; the minimum value in $A[1, 2]$ is 1; the minimum value from $A[1, 3]$ is 1; the minimum value from $A[1, 4]$ is 1.

Suffix Minima Array

A **suffix minima** array S stores the current minimum value seen while traversing the input array from *right* to *left*.

For example, the internal node labeled "1,4" is associated with the subarray with indices $i = 1$ and $j = 4$. Therefore the input to `suffix minima` is $A[1, 4] = [3, 1, 5, 7]$. Remember this time we calculate the minimum from *right* to *left*.

- $S[1] = min(A[4, 4]) = min(7) = 7$
- $S[2] = min(A[4, 3]) = min(7, 5) = 5$
- $S[3] = min(A[4, 2]) = min(7, 5, 1) = 1$
- $S[4] = min(A[4, 1]) = min(7, 5, 1, 3) = 1$

The S array contains [7,5,1,1] since the minimum value in $A[4, 4]$ is 7; the minimum value in $A[4, 3]$ is 5; the minimum value in $A[4, 2]$ is 1; the minimum value in $A[4, 1]$ is 1.

Least Common Ancestor

The least common ancestor of two nodes k and n in a tree is the lowest node that has both k and n as descendants. For the graphic below, this is node i.

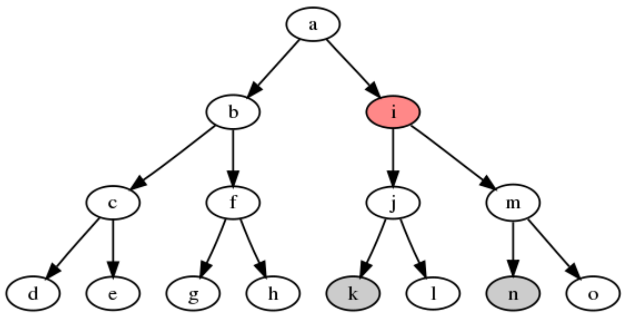


Figure 22.4: Node i is the least common ancestor of k and n.

To calculate the least common ancestor of nodes in our segmented tree we start by labeling each node using an in-order traversal. The labeling is done in red in the figure below.

Then we proceed by comparing the binary representation of the nodes in question.

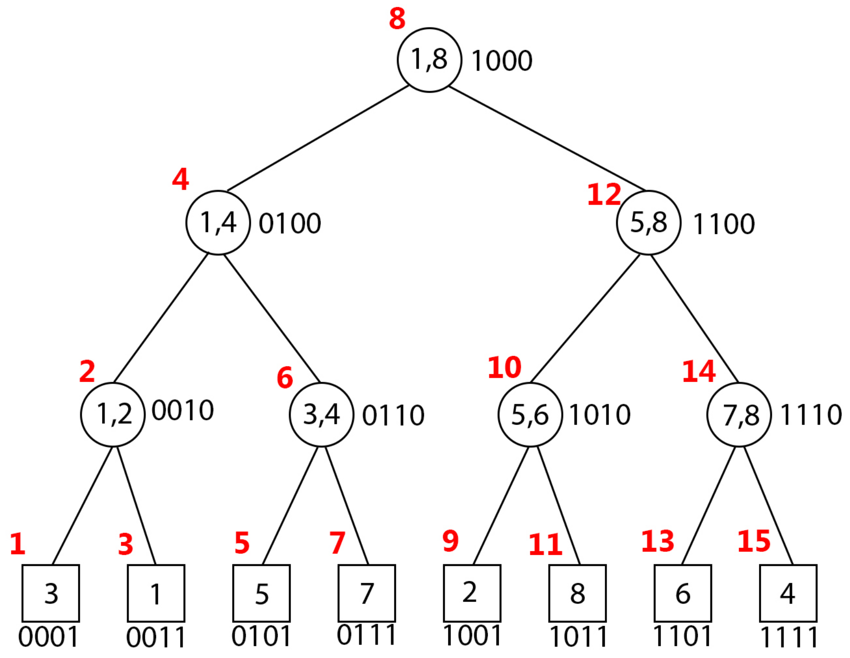


Figure 22.5: Calculating the least common ancestor using binary representation

The least common ancestor of nodes A and B is node C and can be computed by comparing the binary representation of A and B.

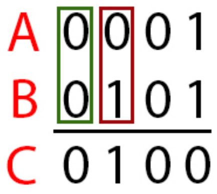
1. Compare A and B in bit-wise fashion starting from the left (most significant bit).
2. For a given bit position, if the bit in A and B match, then C gets that value for that position.

3. Continue until you reach the first position where the bit in A is not equal to the bit in B. Set that position in C to 1.

4. Set all subsequent positions in C to 0.

For example:

The first position in A and B match so the output C gets the same value. The second position differs so set that position in C gets set to 1. All subsequent values in C are set to 0.



The least common ancestor of nodes 1 (0001) and 5 (0101) is 4 (0100).

Putting It All Together: Using Segmented Tree to Calculate RMQ

We have generate a segmented tree, filled it with P and S arrays, and numbered the nodes using in-order traversal. We still need to compute the range minimum query.

The steps to compute the $RMQ_A(i, j)$ are as follows:

1. Find the **least common ancestor** of leaf nodes i and j. For example if $RMQ_A(2, 3)$ then the internal node labeled "1,4" would be the least common ancestor of the leaf nodes 2 and 3.

2. If the answer can be determined from that node we are done. Otherwise, use the suffix minima array from left child and the prefix minima array from the right child to compute the answer.

Using **figure 22.3** as an example we are going to compute $RMQ_A(3, 5)$:

The least common ancestor of the third and fifth leaf node is the root node. But the answer cannot be determined from the root node itself so we must examine its children. Using the S = [1,1,5,7] from the left child and P = [2,2,2,2] from the right child, we calculate $min(S[3], S[4], P[1]) = min(5, 7, 2) = 2$.

$S[3], S[4]$, and $P[1]$ correspond to the indices (3,5) in the range minima query.

22.3 Tree Contraction

An expression can be represented by an expression tree where all leaf nodes are constants and all the internal nodes are operators. Solving the expression tree iteratively can be inefficient for large or unbalanced trees.

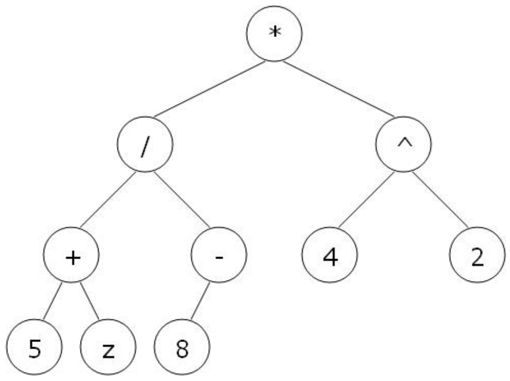


Figure 22.6: An expression tree

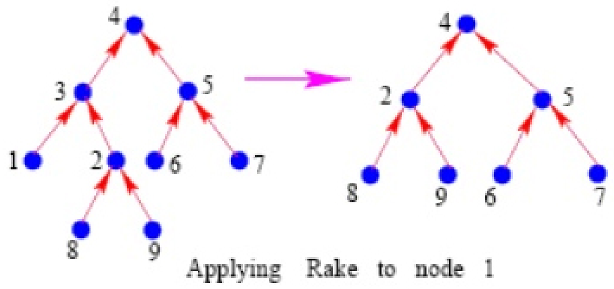
22.3.1 `Rake` operation

In order to solve the tree contraction problem efficiently we need to introduce the `rake` operation.

Let $T = (V, E)$ be a rooted binary tree for each vertex $v, P(v)$ is its parent. $sib(v)$ is the child of $p(v)$.

The `rake` operation does the following:

1. Removes u and $p(u)$ from T, and
2. Connects $sib(u)$ to $p(p(u))$



22.3.2 Algorithm

In our tree contraction algorithm, we must apply the rake operation repeatedly to reduce the size of the binary tree.

We first label the leaves consecutively from left to right. We exclude the leftmost and rightmost leaves. These two leaves will be the two children of the root when the tree is contracted to a three-node tree.

`Tree Contraction Algorithm`

`INPUT:` Rooted binary tree T.

`OUTPUT:` 3-node binary tree

1. We first label the leaves consecutively from left to right. We exclude the leftmost and rightmost leaves. These two leaves will be the two children of the root when the tree is contracted to a three-node tree. Store all n leaves in array A. (A_{odd} is the subarray consisting of the odd-indexed elements of A. A_{even} is the subarray consisting of the even-indexed elements of A.)
2. For $i := 1$ to $log(n+1)$ **do** :
   (a) Apply `rake` operation concurrently to each element of A_{odd} that are left children.
   (b) Apply `rake` operation concurrently to the rest of elements in A_{odd}.
   (c) Set $A := A_{even}$.

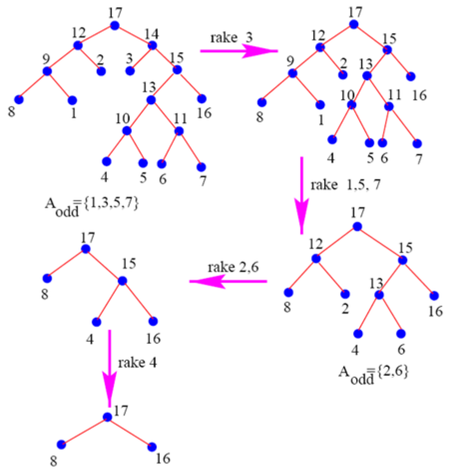


Figure 22.7: Tree contraction using `rake` operator