CS 261: Data Structures

Week 8: Navigating in trees

Lecture 8a: Succinct trees and common ancestors

David Eppstein
University of California, Irvine

Spring Quarter, 2025

This work is licensed under a Creative Commons Attribution 4.0 International License

Euler tours in trees

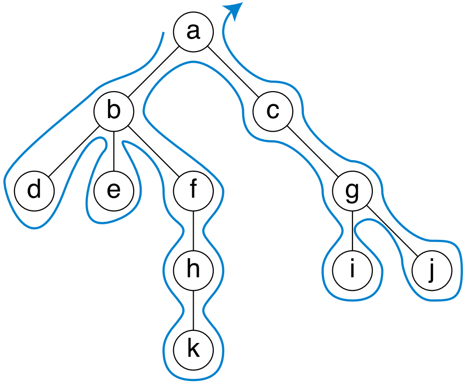
Euler tours (intuition)

Draw a tree in the plane, from the root down

Trace your finger around the boundary of the drawing

What order does it reach each vertex (multiple times)?

Each edge traced twice $\Rightarrow$ # vertices in tour $= 2n - 1$



a, b, d, b, e, b, f, h, k, h, f, b, a, c, g, i, g, j, g, c, a

Euler tours (pseudocode)

To produce an Euler tour of a subtree rooted at x:

- Output x
- For each child y of x:
    - Recursively produce an Euler tour for the subtree rooted at y
    - Output x again

Like

preorder (root comes before children),
inorder (root comes between children), and
postorder (root comes after children),

all merged into one

Representing trees succinctly

Basic tree navigation operations

If we represent trees with an object for each node, we can handle:

- Find the root
- Check if a node is the root, or is a leaf
- Go to the parent of a node
- Go to the first child of a node
- Go to the next child of the same parent

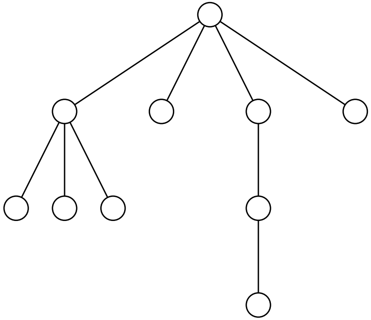
. . . by including a pointer to each of these things in each node

Space is $O(n)$ words of memory . . . but we can do better!

Arbitrary trees with ordered children

Write nested parentheses for each node,
surrounding string representing the subtree below it

Convert to binary: “(” = 0, “)” = 1



0 00101011 01 000111 01 1

$$(\ \underbrace{(\,(\,)\,(\,)\,(\,)\,)}_{\text{left child and its three grandchildren}}\ (\,)\ (\,(\,(\,)\,)\,)\ (\,)\ )$$

n-node tree $\Rightarrow$ $2n$-bit binary number

Open parenthesis: parent-to-child edge in Euler tour

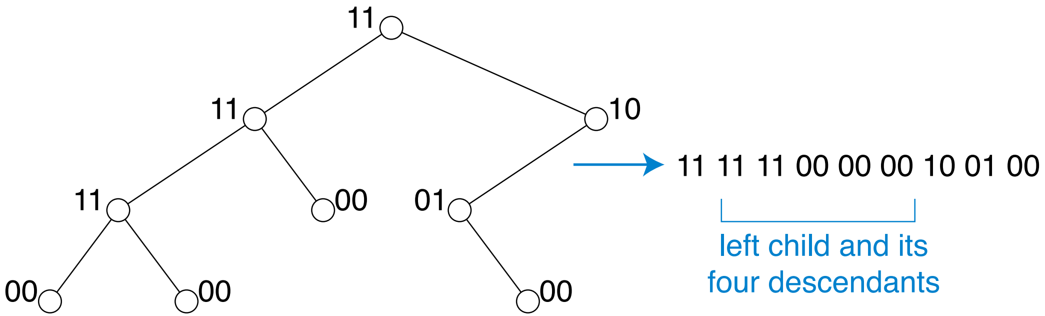
Close parenthesis: child-to-parent edge

Binary trees

Parentheses don't work: can't tell if single child is left or right

Traverse tree in preorder (root first, then children)

For each node, write down two bits:
Does it have a left child (0 for no, 1 for yes)?
Does it have a right child (0 for no, 1 for yes)?



n-node tree $\Rightarrow$ $2n$-bit binary number

Both codes are near-optimal

In each case, # trees is a Catalan number

$$C_n = \binom{2n}{n} \Big/ n + 1 \approx \frac{4^n}{\sqrt{n^3 \pi}}$$

(C_n for n-node binary trees, C_{n-1} for ordered trees)

So in order to have a different code for each tree, # bits must be

$$\geq \log_2 C_n = 2n - O(\log n)$$

Blocking

One long sequence of parentheses is concise but difficult to navigate

Break it up into blocks of $\frac{1}{2} \log n$ parens (represented as a sequence of $\frac{1}{2} \log n$ bits)

There are $\sqrt{n}$ different sequences of $\frac{1}{2} \log n$ bits $\Rightarrow$ we can build a table indexed by these sequences that lets us navigate within a block in constant time per step (tables take much less than n space to store)

There are $O(n/\log n)$ blocks $\Rightarrow$ we can use a less space-efficient structure to navigate from block to block and the small number of blocks will make it more efficient

We will see this same blocking strategy repeatedly all week!

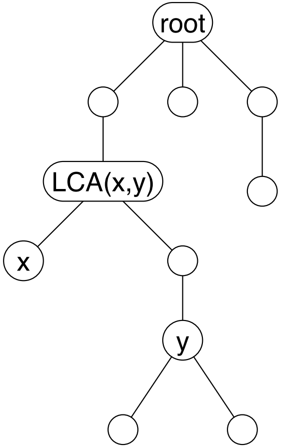
Log-shaving principle

Using blocks can reduce memory requirement
from $S(n)$ to 2n bits + $S(n/\log n)$

Sometimes repeatable for $O(1)$ levels,
saving $\log n$ in space each time

Common ancestors and their applications

Lowest common ancestor



Lowest common ancestor of two nodes x and y in a rooted tree is:

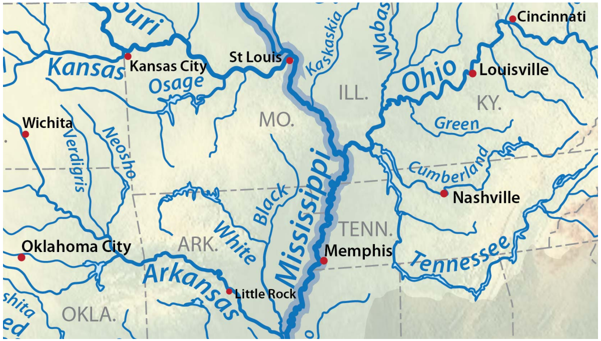
- An ancestor of x (on path to root)
- An ancestor of y
- Lower (farther from root) than any other common ancestor

It may be x or y itself, if one is an ancestor of the other

Shortest paths in undirected trees

Data: An undirected tree with lengths on edges

Goal: Process so we can quickly look up distances between pairs of vertices, or find shortest path between vertices



E.g. what are the river-distances between these nine cities?

Cropped from File:Mississippiriver-new-01.png by Shannon1 on Wikimedia commons

Shortest path solution

Choose a root for the tree (the mouth of the river)

Build a data structure for finding lowest common ancestors
(the point where the streams from any two cities meet)

Store the distance to the root at each node

Then:

- $\text{distance}(x, y) = \text{distance}(x, \text{root}) + \text{distance}(y, \text{root}) - 2 \cdot \text{distance}(\text{ancestor}, \text{root})$
- Path from x to y can be found by following parent links from x to common ancestor, and from y to common ancestor,
  and then gluing together these two paths to the ancestor

Bandwidth in computer networks

Network: Graph of computers and routers with communication links (edges) between them

Different communication links have different bandwidths (number of bits/second you can transfer across the link)

Bandwidth of a path = minimum bandwidth of any link

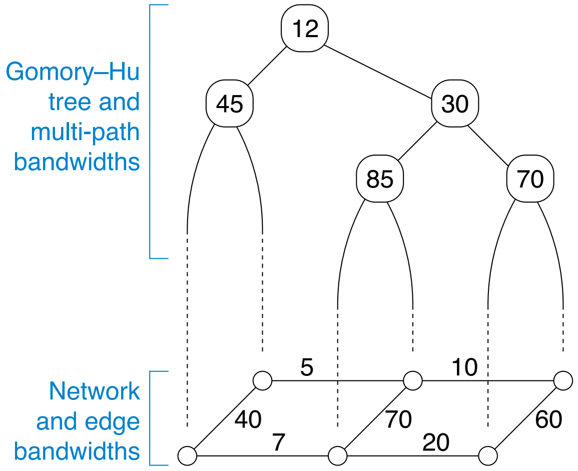
Can also route messages along multiple paths as long as total traffic on any edge is at most its bandwidth

Gomory–Hu tree

Hierarchical clustering of vertices of a graph, with clusters labeled by numbers

Bandwidth between any two nodes is the label of their ancestral cluster

Can be defined for either single or multi-path routing


Gomory–Hu tree and multi-path bandwidths
12
45
30
85
70
Network and edge bandwidths
5
10
40
70
60
7
20

The remaining question

Both shortest paths and bandwidth motivate the question:

If we are given as data a rooted tree,

How can we design and build a data structure

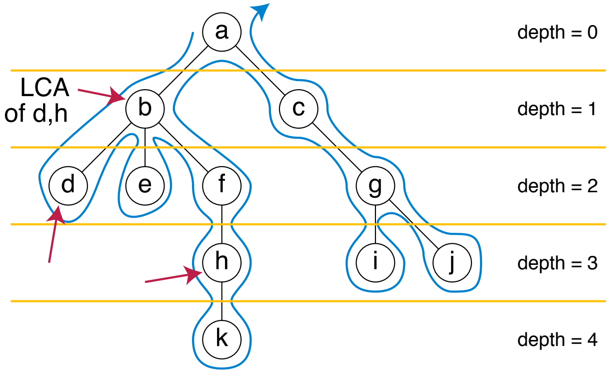
For quickly answering lowest common ancestor queries?

From ancestors to range minima

Ancestors and Euler tours

Number tree vertices by depth (distance from root)

LCA(x, y) = smallest depth node between first occurrence of x and first occurrence of y in the Euler tour



a:0, b:1, d:2, b:1, e:2, b:1, f:2, h:3, k:4, h:3, f:2, ...

range between 1st occurrences of d, h

From ancestors to range minima

Given a tree, to be processed for ancestor queries:

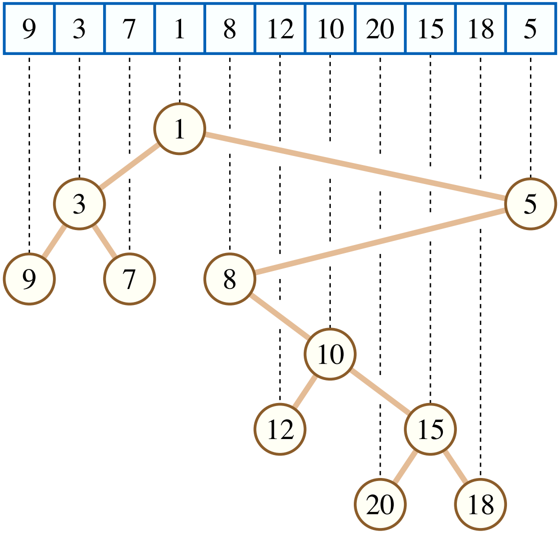
- Number vertices by depth (distance from root)
- Make array E of $2n-1$ vertices, where $E[i] =$ ith vertex of Euler tour
- Make array D of $2n-1$ numbers, $D[i] = \mathrm{depth}(E[i])$
- Store for each vertex v the position $P[v]$ of its first occurrence in the tour
- Build a data structure for range minimum queries in D

Then $\mathrm{LCA}(x, y) = E[\mathrm{rangemin}(P[x], P[y])]$

So how quickly can we answer range minimum queries?

Cartesian trees

Example



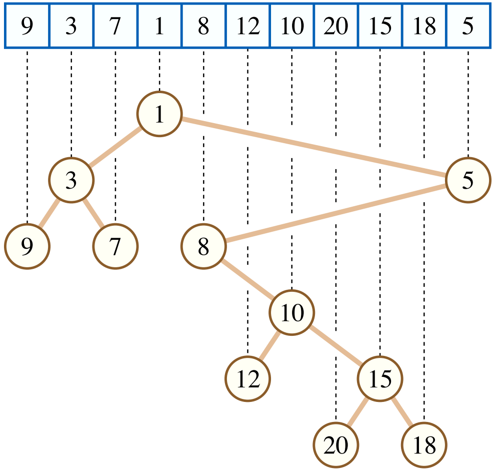
Recursive definition

Defined from an array of numbers

Root is the minimum number
(leftmost copy if there are ties)

Left subtree defined recursively
from subarray to left of root

Right subtree defined recursively
from subarray to right of root



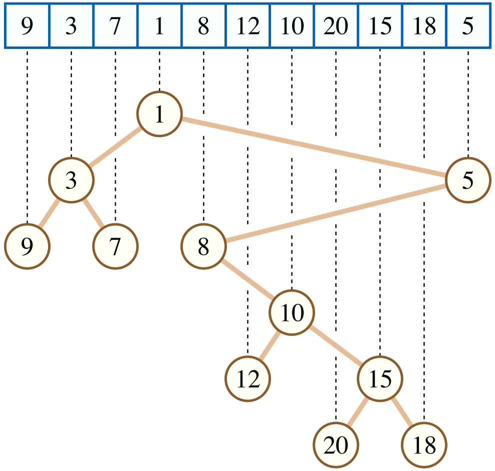
Axiomatic definition

Binary tree whose nodes are the numbers in a given array

(but not a binary search tree)

Inorder traversal order of the tree = array order

Heap-ordered



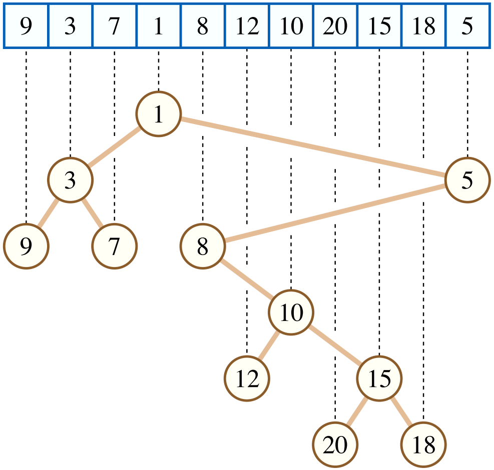
Linear time construction

Add numbers to tree in left-to-right order

For each number $A[i]$

- Follow parent links from $A[i-1]$ until finding ancestor that is $\leq A[i]$
- Add x as right child
- Add previous right child as left child of x

Amortized time for adding $A[i] = O(1)$ with potential function $\Phi =$ length of path from root to rightmost node



E.g. when adding 5 to this tree, path of parent links = 18—15—10—8—1, ancestor = 1, and previous right child = 8

From range minima to ancestors

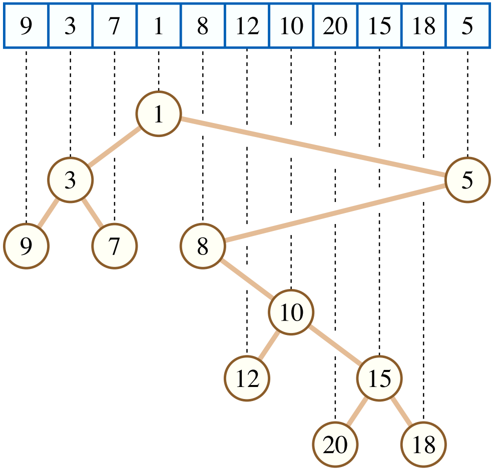
For any array A of numbers

Minimum of A in range $[i, j]$
$=$ LCA of $A[i]$ and $A[j]$ in the
Cartesian tree of A

Explanation: The recursive definition chooses roots of subtrees in sorted order

$A[i]$ and $A[j]$ stay together in the same subtree until we choose a root between them

That root must be the smallest number in the range



Circular logic

By constructing an Euler tour, we can convert ancestor problems into range minimum problems

tree with n nodes $\Rightarrow$ array with $2n-1$ numbers

By constructing a Cartesian tree, we can convert range minimum problems into ancestor problems

array with n numbers $\Rightarrow$ tree with n nodes

How is this helpful?

Ancestor and range minimum solutions

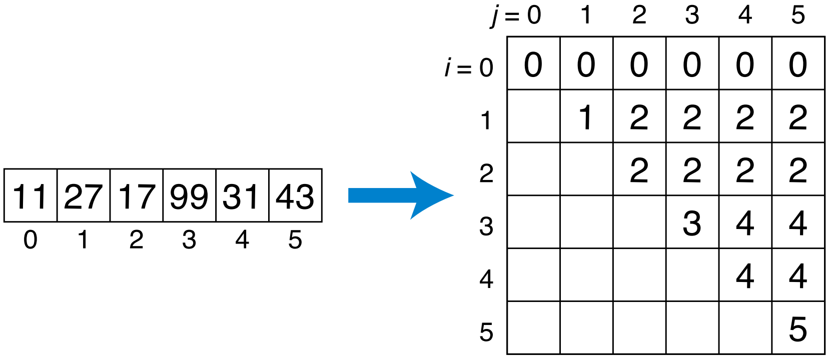
History

- Aho et al. [1973]: Define ancestor problem
- Harel and Tarjan [1984]: $O(n)$ space, $O(1)$ query time, but complicated
- Schieber and Vishkin [1988]: use bitwise operations
- Gabow et al. [1984]: convert range-minimum to LCA using Cartesian tree
- Berkman and Vishkin [1993]: convert LCA to range-min by Euler tour; solve by table lookup + common substructures
- Bender and Farach-Colton [2000], Alstrup et al. [2004], Fischer and Heun [2006]: additional simplifications

Table lookup

Easy but space-inefficient solution for range minimum:

Construct a two-dimensional array $M[i,j]$
where $M[i,j]$ = the index of the minimum in range $[i,j]$



11	27	17	99	31	43
0	1	2	3	4	5

	$j=0$	1	2	3	4	5
$i=0$	0	0	0	0	0	0
1		1	2	2	2	2
2			2	2	2	2
3				3	4	4
4					4	4
5						5

Query = look up array value

Table lookup construction

$M[i,j]$ = the index of the minimum in range $[i,j]$
To construct M, loop over each pair (i,j) with $i \leq j$

- If $i = j$, range has only one element, so $M[i,j] = i$
- Otherwise, compare $A[M[i,j-1]]$
  (minimum element of subrange $[i,j-1]$) to $A[j]$
- If $A[M[i,j-1]] \leq A[j]$, whole range has same minimum as subrange, so set
  $M[i,j] = M[i,j-1]$
- If not, set $M[i,j] = j$

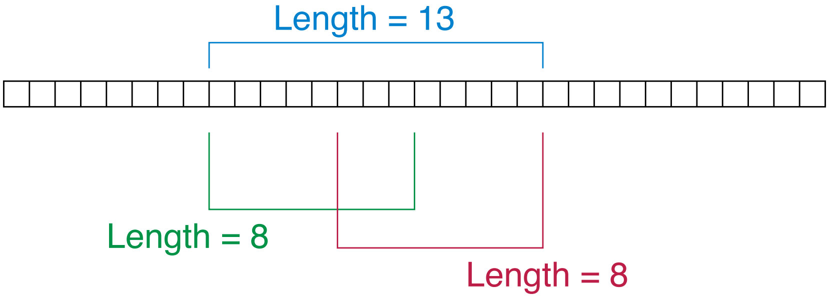
Query time: $O(1)$

Space: $O(n^2)$

Construction time: $O(n^2)$

Decomposition into power-of-two ranges

Key insight: Any interval $[i, j]$ is the union of two overlapping intervals whose length is a power of two



Answer queries in overlapping ranges and combine results

Few ranges have power-of-two lengths $\Rightarrow$ smaller lookup table

Formulas for power-of-two ranges

The largest power of two that will fit inside range $[i, j]$:

$$k = \lfloor \log_2(j - i + 1) \rfloor$$

Decomposition of $[i, j]$ into two length-2^k ranges:

$$[i, j] = [i, i + 2^k - 1] \cup [j - 2^k + 1, j]$$

Make table of only length-2^k ranges, for $k = 0 \ldots \log_2 n$

$$P[i, k] = M[i, i + 2^k - 1]$$

(this is just definition of P; we'll see faster calculation next slide)

Answer queries in overlapping ranges and combine results

$$\text{rangemin}(i, j) = \min(A[P[i, k]], A[P[j - 2^k + 1, k]])$$

Table construction for power-of-two ranges

To compute $P[i,k]$, the index of the minimum of the length-2^k range starting at i, simply divide it into two length-2^{k-1} ranges and compare their minima

For $k = 0, 1, \dots \lfloor \log_2 n \rfloor$:

- If $A[P[i,k-1]] \leq A[P[i+2^{k-1}, k-1]]$, set $P[i,k] = P[i,k-1]$
- Otherwise, set $P[i,k] = P[i+2^{k-1}, k-1]$

Query time: $O(1)$

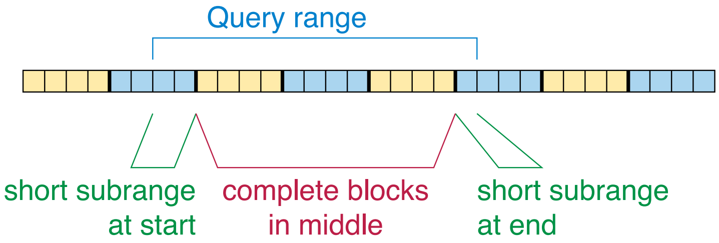
Space: $O(n \log n)$

Construction time: $O(n \log n)$

Blocking for linear space

Idea: Partition input array into blocks of some length b

Query range $\Rightarrow$ two short ranges at ends
+ one longer range of complete blocks in the middle



If blocks are short enough, many have similar structures
$\Rightarrow$ re-use same lookup table for short ranges of multiple blocks

Longer middle range only uses the minimum value in each block
$\Rightarrow$ less space because shorter sequence, length $\approx n/b$

Blocking details

Choose block size $b \approx \frac{1}{3} \log_2 n$

Label each block by succinct representation t of its Cartesian tree

- $2b$ bits per label
- Blocks with same label behave the same

Precompute tables $M_t[i, j]$ for queries inside blocks with label t

- Number of tables $= 2^{2b} = n^{2/3}$
- Time and space per table $= b^2 = O((\log n)^2)$
- Total $= n^{2/3}(\log n)^2 = o(n)$

Blocking analysis

Any ancestor or range minimum data structure with space $S(n)$ and query time $O(1)$ + blocking $\Rightarrow$ a data structure with space:

- $2n$ bits for the block labels
- $O(n^{2/3} \log^2 n)$ for the lookup tables
- $S(O(n/\log n))$ for the ranges of complete blocks

Power-of-two solution $S(n) = O(n \log n)$ words $\Rightarrow$ $O(n)$ words

Second level of blocking $\Rightarrow$ $O(n)$ bits

Third level of blocking $\Rightarrow$ $2n + O(n/\log n)$ bits

References and image credits, I

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. On finding lowest common ancestors in trees. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1973, Austin, Texas, USA*, pages 253–265. Association for Computing Machinery, 1973. doi: 10.1145/800125.804056.

Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. Nearest common ancestors: A survey and a new algorithm for a distributed environment. *Theory of Computing Systems*, 37(3):441–456, 2004. doi: 10.1007/s00224-004-1155-5.

Michael A. Bender and Martin Farach-Colton. The LCA Problem Revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10–14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi: 10.1007/10719839_9.

Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 22(2):221–242, 1993. doi: 10.1137/0222017.

References and image credits, II

Johannes Fischer and Volker Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5–7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006. doi: 10.1007/11780441_5.

Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1984, Washington, DC, USA*, pages 135–143. Association for Computing Machinery, 1984. doi: 10.1145/800057.808675.

Dov Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi: 10.1137/0213024.

liempdma. Cutting lumber with a swingblade sawmill. CC-BY-SA image, September 4 2018. URL `https://commons.wikimedia.org/wiki/File:Cutting_lumber_with_a_swingblade_sawmill.jpg`.

Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988. doi: 10.1137/0217079.