6.851: Advanced Data Structures Spring 2007

Lecture 16 — April 11, 2005

Oren Weimann *Scribe: Alison Cichowlas*

1 Overview

In the last lecture we found even lower minimum bounds for the static predecessor problem, using message elimination.

In this lecture we move on to examining how to find the minimum index in a range of numbers or the least common ancestor of two nodes of a tree in constant time with linear preprocessing. This problem was first solved in 1984 [1], but a variety of different proofs of varying degrees of elegance have been published since.

2 Main Section

Instead of the original solution by Harel and Tarjan in 1984 [1], this lecture follows the simplified analysis presented in Bender and Farach-Calton's 2000 paper [2]. Other examinations of this problem are briefly overviewed in section 2.4.
We will examine the solution on an array A of size n. Our example array will be
$(A = 17, 0, 36, 16, 23, 15, 42, 18, 20)$.
The naive solution to this problem requires $O(n^2)$ space in order to return an answer in $O(1)$ time, by a simple lookup table.

2.1 Reducing Range Minimum Queries to Least Common Ancestor Queries

We can turn a range minimum query into a least common ancestor query by using Cartesian trees. Cartesian trees are constructed in the following manner: say the minimum of the array A is the element $A[i]$. This element is the root node of the Cartesian tree. Now for the left child, recurse on both remaining portions of the array: the left subtree is constructed from the elements $A[1]...A[i]$; the right subtree is constructed from the elements $A[i+1]...A[n]$. As we will learn in the next lecture, we can construct this tree in linear time.
The range minimum query returns the smallest element with an array index greater than i and less than j. In the Cartesian tree, this element will be the least common ancestor of nodes $A[i]$ and $A[j]$. So, we can reduce an RMQ problem to an LCA problem with linear prepreprocessing.

2.2 Least Common Ancestor Problems

In LCA problems, we examine a static tree of n nodes. (Dynamic trees have been considered, but so far $O(1)$ solutions have only been found for the operations of inserting and deleting at the leaves, and subdividing or merging edges.) [3]

In a complete binary tree, you can find the LCA by this process: express paths to both nodes as binary numbers, XOR the two paths, and find the most significant bit of the result.
We will reduce the least common ancestor query back to another, less complicated range minimum query. First we write the Euler tour of the tree in question. In this case, the tour goes
$(0, 17, 0, 15, 16, 36, 16, 23, 16, 15, 18, 42, 18, 20, 18, 15, 0)$. We can also track and remember just the tree depth of each element in the tour; in this case the array produced would be
$(0, 1, 0, 1, 2, 3, 2, 3, 2, 1, 2, 3, 2, 3, 2, 1, 0)$.
This array has the useful property that each adjacent element differs from its neighbors by exactly one.

2.3 Solving RMQ +/- 1

We are now back to a range minimum query on this list of tree depths. (We want to find the minimum depth between our two indices i and j.) In order to solve it, we will split it up into many smaller arrays, find the minimum in each of the smaller arrays within our bounds, and then find the minimum among those minimums.
We split the array into arrays of size $(1/2 * lg(n))$. There will be $(2 * n/lg(n))$ of these groups. Any range minimum query will have to access the end of a block (the one containing i), a section of several full blocks, and the beginning of a block (the one containing j). The returned value of the range minimum query should be

$$min \left\{ \begin{array}{l} RMQ(i, \infty) in i' s group; \\ RMQ(-\infty, j) in j' s group; \\ RMQ of all groups > i' s group and < j' s group in summary \end{array} \right\}$$

The questions we must now answer are: how can we find the RMQ for each small group? and how can we find the RMQ over the minimums of the relevant part of the summary structure?

2.3.1 RMQ for Each Group

We will solve this using a lookup table, storing the RMQ for all values of i, j within this particular sub-group.
Because we have $\sqrt(n)$ distinct groups, $1/2 * lg(n)$ possible queries, and $lglg(n)$ query output size, the total size of this lookup table will be $\sqrt(n) * 1/2lg(n) * lglgn = o(n)$.

2.3.2 RMQ on Summary

In the summary, we cannot rely on adjacent elements being +/- of each other. We know we must do better than the trivial solution of creating a lookup table here, because it will require $(2 * n/lg(n)^2)$ in both time and space, which (although better than the initial solution) is still worse than linear. However, since the summary reduces the input size from the original problem, we can use up to $O(nlgn)$ preparation time to achieve our goal of an O(1) query. We can achieve this by storing, for every start point in the array, the RMQ answer for all end points that are exactly a power of 2 away from the start point. This will take $O(nlgn)$ space.
We can now retrieve the answer of a query over two indices separated by a power of two in constant

time by retrieving its answer from the lookup table. Any other index separation can be viewed as two overlapping sections of a size that is a power of two – for instance, an interval of length 6 can be covered by two overlapping intervals of size 4, and the minimum of these two overlapping queries is the RMQ of the desired query. By using this strategy on the results from the many small groups we produced and found answers for in 2.3.1, we can find the answer to our initial query.

2.4 Other methods

In 1993, Berkman and Vishkin [5] examined this problem, taking a very similar angle on the problem, but without the factor of 1/2 Bender's method uses in splitting up the array into sub-groups. This means that the size of the groups is too large to cope with in our desired time order of growth, so they are forced to use another layer of indirection before applying the same methods to the final solution, splitting each group of initial size lgn into groups of size $lglgn$.
In 2006, Fischer and Heum [4] showed a method whose goal was to avoid the expensive building of the initial Cartesian tree. They skipped the reduction of the initial RMQ into an LCA and then back into an RMQ with adjacent terms differing by one, instead building small cartesian trees on each of the possible sub-groups of the initial problem with $(1/4 * lg(n))$ elements. Since these trees are of much smaller size than the initial tree, they are much less costly to build.

References

[1] Dov Harel, Robert Endre Tarjan, *Fast Algorithms for Finding Nearest Common Ancestors*, SIAM J. Comput. 13(2): 338-355 (1984)

[2] Michael A. Bender, Martin Farach-Colton, *The LCA Problem Revisited*, LATIN 2000: 88-94

[3] Richard Cole, Ramesh Hariharan: *Dynamic LCA Queries on Trees*, SODA 1999: 235-244

[4] Johannes Fischer, Volker Heun, *Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE*, CPM 2006: 36-48

[5] Omer Berkman, Uzi Vishkin: *Recursive Star-Tree Parallel Data Structure*, SIAM J. Comput. 22(2): 221-242 (1993)