Advanced Data Structures

Lecture 12

Lecture by Dr. Shay Mozes
Typeset by Steven Karas

2017-02-02
Last edited 20:48:37 2017-02-02

Today we will finish up RMQ, and hopefully start covering Suffix Trees.

1 RMQ

We track the minimums of all the groups, and the minimums for each suffix/prefix. Preprocessing of $O(n \log n)$. Query time of 2.

We want to reduce the preprocessing time to be linear.

By reusing the previous solution as a black box, we can reduce the preprocessing time to $O(n \log^* n)$ with a query time of 4.

Note that we chose to use blocks of size $\frac{1}{4} \log n$, such that $\frac{n}{x} \log \left(\frac{n}{x}\right) = O(n)$ when $x = \log n$.

If we use the previous solution, we can reduce the preprocessing time even further. In the end, we get the inverse Ackermann function: $O(n\alpha(n))$ with a query time of $2\alpha(n)$.

1.1 Linear preprocessing solution

Reduce from RMQ to LCA back to RMQ. Each block can be reduced further to a vector of ± 1 elements. Note that there are $2^{1/4 \log n} = n^{1/4}$ possible block configurations. This gives us $n^{\frac{1}{4}} \cdot \left(\frac{1}{4} \log n\right)^2 = O(n)$ to build the prefix/suffix tree. We can then track the type of the blocks in a lookup table. This gives us a query time of 4.

Note that in our solution, we can just as easily use a different block size smaller than $\log n$ by any ε.

1.2 Practical concerns

More practically, this means running DFS, which is inefficient in practical terms (doesn't parallelize, cache-misses). Examine the Cartesian trees that we build as part of the reduction to LCA. The shapes of the trees are largely reused, so we can reduce the shape of the trees to binary numbers. The bound on the number of such shapes is $2^{2n} = 4^n$, although theoretically a stricter bound is possible using Catalan numbers. This gives us a bound on the number of shared block shapes: $4^{1/4 \log n} = \sqrt{n}$.

2 String/Pattern Matching

One shot solution of $O(|T| + |P|)$[1].

We want to study data structures that preprocesses the text in $O(|T|)$ and answers queries in $O(|P|)$.

3 Trie

Dictionary data structure implemented as a rooted tree. Each node has up to $|\Sigma| + 1$ children. Root to leaf path gives a word from the dictionary. Each word ends with a $ symbol to distinguish prefixes. Left to right traversal of the leaves gives us a sorted list of the words.

Space $O(n)$ nodes, where n is the sum of the word lengths. Each node stores $|\Sigma| + 1$ entries. Total space: $O(n|\Sigma|)$.

We can reduce the space further by using BSTs instead of arrays for each node.

Compressed Trie By combining paths keyed by the first character, we can save a lot of space.

4 Suffix Tree

Space $O(|T| \cdot |\Sigma|)$, but Σ is constant, so $O(|T|)$.

Construction $O(|T| + \mathit{sort}(\Sigma))$ construction time. $O(|T|)$ when $|T| \gg |\Sigma|$.

Query $O(|P|)$

4.1 Structure

Compressed trie of all $|T| + 1$ suffixes "T$". $|T| + 1$ leaves. Edge label is the substring of T, so we can store pointers to within T.

A full proof is left as an exercise to the reader.

4.2 Query

Find all the appearances of a substring in the text: Find/Count the leaves in the subtree of the prefix.

Generalized multi document search Concatenate multiple documents $D1\#D2\#...\#Dk$. The separator character $\# \notin \Sigma$.

How many documents does P appear in?

Find the largest substring of T in T'.

[1] [Knuth, Morris, Pratt - 1977]; [Boyer, Moore - 1977]; [Karp, Rabin - 1987]

5 Suffix Array

Suffix tree leaves sorted left to right. A lexicographical ordering of the suffixes.

For example, for the text "banana":

b	a	n	a	n	a	$
6	5	3	1	0	4	2

We can perform a binary search for query strings in this array: $O(|P| \log |T|)$ to find P in T.

Goes together with a Longest-Common-Prefix array: $\text{LCP}[i]$ = the length of the longest common prefix of the i-th and i+1-th suffix in order.

Note: $\text{LCP}[i] = \text{RMQ}(\text{LCP}[i], ..., \text{LCP}[j])$.

All together now! $\text{RMQ} + \text{LCP} + \text{SA} \Rightarrow O(|P| + \log |T|)$ to search for P in T.

The binary search can be aided by using the LCP as a comparison key for the binary search.

5.1 Kangaroo method

For "fuzzy" searching with k-mismatches.

Create a suffix tree/array for $s = P\#T$

Check P at each location i of T by kangarooing.

We compare, and are willing to skip over k LCP mismatches.

5.2 Reduction to Suffix Tree

Reduction from Suffix Tree to Suffix Array is trivial: in-order traversal, writing the leaves/LCP.

The full transformation is in the slides, but we did not cover it.

6 Exam

The course is based on similar courses at MIT and Haifa. Sample exams can be found there. True/False questions are a good measure of our knowledge of the material, and we should expect this on the exam. Open questions are less interesting, but there will be similar questions. The solutions from Haifa are only the most basic solution.

If needed, additional office hours can be set by appointment.