15-451/651 Algorithm Design & Analysis, Fall 2025

Recitation #3

Objectives

- Understand the technique of *fingerprinting* and apply it to solve string problems
- Understand the SegTree data structure and its applications
- See how to define and use custom operators to make SegTrees solve even more problems

Recitation Problems

1. **(A String Matching Oracle)** In this recitation we generalize the fingerprinting method described in lecture. Let $T = t_0, t_1, \ldots, t_{n-1}$, be a string over some alphabet $\Sigma = \{0, 1, \ldots, z-1\}$. Let $T_{i,j}$ denote the substring $t_i, t_{i+1}, \ldots, t_{j-1}$. This string is of length $j-i$. We want to preprocess T such that the following comparison of two substrings of T of length ℓ can be answered (with a low probability of a false positive) in constant time:

   $$\text{Test if } T_{i,i+\ell} = T_{j,j+\ell}$$

   First of all let's define the fingerprinting function. Let p be a prime, along with a base b (larger than the alphabet size). The Karp-Rabin fingerprint of T is

   $$h(T) = (t_0 b^{n-1} + t_1 b^{n-2} + \cdots + t_{n-1} b^0) \bmod p$$

   We are essentially interpreting the characters as digits in an integer in base b instead of base 2 like in lecture. From now on we will omit the mod p from these expressions.

   Now, to preprocess the string T, we will compute the following arrays for $0 \le i \le n$:
   (*Don't forget we are omitting the* mod*s!*)

   $$\begin{aligned} r[i] &= b^i \\ a[i] &= t_0 b^{i-1} + t_1 b^{i-2} + \cdots + t_{i-1} b^0 \end{aligned}$$

   (a) Give algorithms for computing these in time $O(n)$:

   (b) Find an expression for $h(T_{i,j})$.

   (c) Explain how to select p such that for $T_{i,i+\ell} \neq T_{j,j+\ell}$, we have

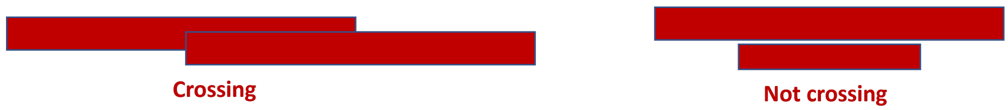
   $$\Pr[h(T_{i,i+\ell}) = h(T_{j,j+\ell})] \le 1/n.$$

   (d) Argue that this choice of p allows for constant-time modular arithmetic under the word-RAM model.

   So the end result is that we can test if $T_{i,i+\ell} = T_{j,j+\ell}$ by comparing $h(T_{i,i+\ell})$ with $h(T_{j,j+\ell})$. The probability of a false positive can be made as small as desired by picking a sufficiently large random prime p.

2. **(Abby's Favorite Problem)** Suppose we start with some array of integers A. Given a sequence of query intervals in the form $[l_1, r_1), [l_2, r_2), \ldots, [l_m, r_m)$, return the maximum element and how many times it appears in $A[l_i], \ldots, A[r_i - 1]$ in $O(\log n)$ time for each query $[l_i, r_i)$. You can use $O(n)$ time for preprocessing.

3. **(Crossing intervals)** Suppose we have a list of n intervals $I_i = [a_i, b_i]$ where $0 \leq a_{,} b_i < 2n$, such that the endpoints of all of the intervals are distinct (no two intervals ever share an endpoint at either end). A pair of intervals I_i and I_j are *crossing* if they overlap but one does not strictly contain the other. We want to devise an algorithm to count the number of pairs of crossing intervals.



   (a) Give a simple $O(n^2)$ algorithm for the problem

   (b) Come up with a more efficient $O(n \log n)$ algorithm by making use of a SegTree