CS 240 — Lecture 19

Bartosz Antczak *Instructor: Eric Schost* *March 16, 2017*

19.1 Rabin-Karp Fingerprint Algorithm

The main idea behind this algorithm is to use hashing. We compute the hash function for each text position.

19.1.1 Overview

We define a fixed value n for our hash "table". We then define our search pattern and our search text. With that, we define our hash function $h(x) = x \mod n$. We will use this hash function on every substring of the same size as our pattern. Once we arrive at a substring whose hash function matches the hash function of the pattern, we verify that the strings match: if they do, we found it; otherwise, we keep searching.

19.1.2 A Problem Occurs

Note that the runtime of the defined algorithm is $\Theta(mn)$, which is no better than brute force.

19.1.3 The Solution Arrives

Two brave men, Rabin and Karp, discovered a way to update the hashes of every substring in constant time! *(P = NP confirmed)*. The idea is use the hash from the previous substring to compute the next one! The runtime is $O(1)$ per hash, except the first one. The way we compute it is as follows: It is recommended to

- Previous hash: 41592 mod 97 = 76
- Next hash: 15926 mod 97 = ??

Observation:

$$\begin{aligned} 15926 \bmod 97 &= (41592 - (4 * 10000)) * 10 + 6 \\ &= (76 - (4 * 9)) * 10 + 6 \\ &= 406 \\ &= 18 \end{aligned}$$

7

choose a random large prime number as your hash table size.

19.2 Suffix Tries and Suffix Trees

What if we want to search for many patterns P within the same fixed text T? Here, we can preprocess the text T rather the the pattern P. From this, we make an observation:

$$P \textit{ is a substring of } T \iff P \textit{ is a prefix of some suffix of } T$$

19.2.1 Definition — Suffix Trie and Suffix Tree

A **suffix trie** is a trie that stores all suffixes of a text T. A **suffix tree** is the compressed suffix trie of T.