Tutorial 9

Carter-Wegman's hashing, number theoretic algorithms,
more problems on hashing and tries

CS 240E W23
University of Waterloo
Monday, March 20

Number theoretic algorithms

1. For increasing the table size in hashing, we wanted to find a prime of a certain size. Specifically, given an integer $M > 1$, we would like to find a prime of size at least $2M$.

   Our approach is based on Bertrand's postulate:

   > For all $n > 1$, there is a prime p such that
   > $$n < p < 2n.$$

   We now have a naive approach to finding this next prime: iterate over $2M \leq x \leq 4M$, and break as soon as we find that x is prime. We can check if a number x is prime by iterating over its divisors (that are at most $\sqrt{x}$) in $O(\sqrt{x})$ time. This gives a runtime of $\Theta(M\sqrt{M})$.

   If we could improve the time that it takes to check if x is a prime, we would drastically improve the runtime. We will therefore try to efficiently precompute an array $\textit{is\_prime}[x]$ that stores a boolean indicating whether x is prime or not.

   We will precompute primes with the *Sieve of Eratosthenes*, a very well-known and practical algorithm for computing primes in range $[0, n]$ in $O(n \log \log n)$ time. The idea is to write down all the numbers between 2 and n, to initialize $\textit{is\_prime}[x]$ to true for all of them, and to iterate over them in increasing order.

   Every time we arrive at a number that has not been "crossed out" (ie. $\textit{is\_prime}[x]$ is true), we "cross out" all the multiples of x starting with $x \cdot x$.

```
is_prime[0..n] = {true, ..., true}
is_prime[0] = is_prime[1] = false
for i = 2..n:
  if is_prime[i]:
    for j = i*i..n:
      is_prime[j] = false
```

   When implementing the Sieve, we can store a dynamic array of all primes, by simply inserting i after the check $\textit{is\_prime}[i]$. We should also be careful because `i*i` will likely overflow. The proof of runtime of this algorithm uses techniques that go beyond the scope of CS240E (we may discuss it in consulting hours if you are interested).

Carter-Wegman universal hashing

2. The material we discussed is [Biedl, line 4700]. The notes also contain the example that we did not complete in class (specifically, in Figure 7.13).

More problems on tries

3. **Prefix search.** Let $w_1, \ldots, w_k$ be words, where $n = |w_1| + \cdots + |w_k|$. Give an algorithm to find the longest word w such that w is the prefix of at least two words from $w_1, \ldots, w_k$.

4. **MSD-radix sort as a trie.** Consider the following base-4 numbers:

$$300, 211, 112, 230, 1, 0, 12, 101, 233, 110.$$

   (a) Draw the recursion tree that results from sorting these numbers with MSD-radix. Note that it is a 4-way pruned trie.

   (b) Show that the expected time to insert a base-4 number into a 4-way pruned trie is at most $\log_4 n + O(1)$, assuming all numbers have been uniformly chosen. You may assume the numbers have been padded with 0s so that all numbers begin with the same place value.

5. Suppose we have n English words (26-letter alphabet), where the combined length of all words is l. Give an algorithm to sort the words (obtain lexicographical ordering) in $O(l)$ time.

More problems on hashing

6. **Universal hash functions.** Recall: a family $\mathcal{H}$ of hash-functions is *universal* if

$$P(h(k) = h(k')) \leq \frac{1}{M} \quad \text{for all keys } k \neq k'.$$

$\mathcal{H}$ has *uniform hash-values* if

$$P(h(k) = i) = \frac{1}{M},$$

for all keys k and all slots i. The probability is taken over the random uniform choice of h among $\mathcal{H}$.

   (a) Consider the family $\mathcal{H}$ on slide 4 of module07e:

$$U = \mathbb{Z}_5, M = 2$$
$$h_b(k) = ((k + b) \mod 5) \mod 2$$
$$\mathcal{H} = \{h_b : b \in \mathbb{Z}_5\}$$

   Choose $b \in \mathbb{Z}_5$ randomly to get hash-function. Does this have uniform hash-values? Is this universal?

(b) Assume that $\mathcal{H}$ has uniform hash-values. Prove or disprove: The expected time for an unsuccessful search in hashing with chaining is $O(\alpha)$.

(c) Assume that $\mathcal{H}$ has uniform hash-values. Prove or disprove: The expected time for a successful search in hashing with chaining is $O(1 + \alpha)$.