References for Today’s Lecture

• Required reading
  – CLRS Chapter 32
String Matching

- The **string-matching problem** is to determine whether a given string occurs as part of a larger string.
- The smaller string is usually called the **search pattern**.
- This problem is also referred to as **pattern matching**.
- String matching has numerous important applications.
  - Finding words or phrases in large documents (text editors, search engines).
  - Finding the occurrence of a given genetic sequence in a large genome.
Terminology

- The set of allowable characters will be denoted $\Sigma$ (usually the ASCII characters).
- The set of strings composed of these characters is denoted $\Sigma^*$. 
- For convenience, we will view a given string as an array of members of $\Sigma$. 
- As in previous settings, we may also convert strings to integers in order to work with them more efficiently. 
- Let $T$ be a given string of length $n$ and $P$ a search pattern of length $m$. 
- We say $P$ occurs with shift $s$ in $T$ if $T[s..s+m-1] = P[0..m-1]$. 
- If $P$ occurs with shift $s$ in $T$, then we call $s$ a valid shift. 
- We can restate the string-matching problem as that of locating all valid shifts in $T$ with respect to $P$. 
The Naive Approach

• The most obvious approach to string matching is to simply compare $T[s..s+m−1]$ to $P[0..m−1]$ for each $0 \leq s \leq n − m$.

• What is the running time of this algorithm?
The Rabin-Karp Algorithm

- To apply the Rabin-Karp Algorithm, we first convert $P$ to its integer equivalent $p$.
- Recall that this can be done in $\Theta(m)$ steps using Horner’s rule.
- Let $t_s$ be the integer value of the substring $T[s..s + m - 1]$.
- Note that $s$ is a valid shift if and only if $t_s = p$.
- If we can calculate $t_s$ for all $0 \leq s \leq n - m$, then we can find all valid shifts in $\Theta(n - m)$ comparisons in theory.
- **Problem**: As before, the integers we are talking about can be HUGE!
Implementing the Rabin-Karp Algorithm

• We can calculate $t_0$ using Horner’s rule, as we did with $p$.
• Let $K = |\Sigma|$ be the number of characters in our alphabet.
• Give $t_s$, we can calculate $t_{s+1}$ by the formula

$$t_{s+1} = K(t_s - K^{m-1}T[s]) + T[s + m]$$

• Hence, we can calculate $t_{s+1}$ from $t_s$ in constant time, in theory.
• This means that in theory, we can calculate $t_s$ for all $0 \leq s \leq n - m$ in $\Theta(n - m)$ steps.
• Again, however, the numbers we are dealing with can be very large.
Modifying the Rabin-Karp Algorithm

- To deal with the very large numbers that can occur, we apply a simple hash function.
- This approach is similar to that for generating compact digital signatures.
- Our simple hash function for a string $S$ will be $S \mod q$, where $q$ is usually the size of the maximum integer that fits into a computer word.
- Using modular arithmetic, we can calculate $t_s \mod q$ for all $0 \leq s \leq n - m$ in $\Theta(n - m)$ steps.
- We then compare $t_s \mod q$ to $p \mod q$ for all $0 \leq s \leq n - m$.
- If $t_s \not\equiv p \mod q$, then $t_s \not\equiv p$.
- Otherwise, we need to calculate $t_s$ and compare it to $p$.
- What is the running time of this modification in the worst case?
Average Case Analysis for Rabin-Karp

- In this case, we can do a simplistic average case analysis.
- We need to know the number of shifts $s$ for which $t_s \equiv p \mod q$.
- As usual, we will assume a random string is equally likely to hash to any value between 0 and q-1.
- In this case, an invalid shift will produce a *spurious hit* with probability $1/q$.
- The overall running time is then

$O(n + m(v + n/q)),$

where $v$ is the number of valid shifts).
- If $v$ is “small” (constant) and $q \geq m$, then this simplifies to $O(n)$. 