Advanced Algorithm Design and Analysis (Lecture 13)

SW5 fall 2007
Simonas Šaltenis
3.2.12
simas@cs.aau.dk
Text-search Algorithms

- Goals of the lecture:
  - *Naive text-search algorithm and its analysis*;
  - *Rabin-Karp* algorithm and its analysis;
  - *Knuth-Morris-Pratt* algorithm ideas
  - *Boyer-Moore-Horspool* algorithm
Text-Search Problem

- **Input:**
  - *Text* $T$ = “at the thought of”
    - $n = \text{length}(T) = 17$
  - *Pattern* $P$ = “the”
    - $m = \text{length}(P) = 3$

- **Output:**
  - *Shift* $s$ – the smallest integer $(0 \leq s \leq n - m)$ such that $T[s \ldots s+m-1] = P[0 \ldots m-1]$. Returns $-1$, if no such $s$ exists.

  \[
  \begin{array}{cccccccccccccccc}
  0 & 1 & 2 & 3 & \ldots & & & & n-1 \\
  \text{at the thought of} \\
  \text{the} \\
  \end{array}
  \]
Naïve Text Search

- Idea: Brute force
  - Check all values of s from 0 to n – m

\[\text{Naive-Search}(T,P)\]

\[
01 \text{ for } s \leftarrow 0 \text{ to } n - m \\
02 \quad j \leftarrow 0 \\
03 \quad \text{// check if } T[s..s+m-1] = P[0..m-1] \\
04 \quad \text{while } T[s+j] = P[j] \text{ do} \\
05 \quad \quad j \leftarrow j + 1 \\
06 \quad \quad \text{if } j = m \text{ return } s \\
07 \text{ return } -1
\]

- Let \( T = "\text{at the thought of}" \) and \( P = "\text{though}" \)
  - What is the number of character comparisons?
Analysis of Naïve Text Search

- **Worst-case:**
  - Outer loop: \( n - m + 1 \)
  - Inner loop: \( m \)
  - Total \((n-m+1)m = O(nm)\)
  - *What is the input that gives this worst-case behavior?*

- **Best-case:** \( n - m + 1 \)
  - *When?*

- **Completely random text and pattern:**
  - \( O(n-m) \)
Fingerprint idea

- Assume:
  - We can compute a fingerprint $f(P)$ of $P$ in $O(m)$ time.
  - If $f(P) \neq f(T[s..s+m-1])$, then $P \neq T[s..s+m-1]$
  - We can compare fingerprints in $O(1)$
  - We can compute $f' = f(T[s+1..s+m])$ from $f(T[s..s+m-1])$, in $O(1)$
Algorithm with Fingerprints

- Let the alphabet $\Sigma=\{0,1,2,3,4,5,6,7,8,9\}$
- Let fingerprint to be just a decimal number, i.e., $f("1045") = 1\times10^3 + 0\times10^2 + 4\times10^1 + 5 = 1045$

**Fingerprint-Search**($T,P$)

01 fp $\leftarrow$ compute $f(P)$
02 $f \leftarrow$ compute $f(T[0..m-1])$
03 for $s \leftarrow 0$ to $n - m$ do
04 \hspace{1em} if $fp = f$ return $s$
05 \hspace{1em} $f \leftarrow (f - T[s]\times10^{m-1})\times10 + T[s+m]$
06 return $-1$

- Running time $2O(m) + O(n-m) = O(n)$!
- *Where is the catch?*
Using a Hash Function

- **Problem:**
  - We can not assume we can do arithmetics with $m$-digits-long numbers in $O(1)$ time!

- **Solution:** Use a hash function $h = f \mod q$
  - For example, if $q = 7$, $h(“52”) = 52 \mod 7 = 3$
  - $h(S_1) \neq h(S_2) \Rightarrow S_1 \neq S_2$
  - But $h(S_1) = h(S_2)$ does not imply $S_1 = S_2$!
    - For example, if $q = 7$, $h(“73”) = 3$, but “73” ≠ “52”

- **Basic “mod q” arithmetics:**
  - $(a+b) \mod q = (a \mod q + b \mod q) \mod q$
  - $(a\times b) \mod q = (a \mod q)*(b \mod q) \mod q$
Preprocessing and Stepping

- **Preprocessing:**
  - \( fp = P[m-1] + 10*(P[m-2] + 10*(P[m-3]+ ... ...
    ... + 10*(P[1] + 10*P[0]))...)) \mod q \)
  - In the same way compute \( ft \) from \( T[0..m-1] \)
  - Example: \( P = "2531" \), \( q = 7 \), what is \( fp \)?

- **Stepping:**
  - \( ft = (ft - T[s]*10^{m-1} \mod q)*10 + T[s+m]) \mod q \)
  - \( 10^{m-1} \mod q \) can be computed once in the preprocessing
  - Example: Let \( T[...] = "5319" \), \( q = 7 \), what is the corresponding \( ft \)?

![Diagram](image-url)
Rabin-Karp Algorithm

Rabin-Karp-Search(T, P)
01 q $\leftarrow$ a prime larger than m
02 c $\leftarrow$ $10^{m-1}$ mod q // run a loop multiplying by 10 mod q
03 fp $\leftarrow$ 0; ft $\leftarrow$ 0
04 for i $\leftarrow$ 0 to m-1 // preprocessing
05 fp $\leftarrow$ (10*fp + P[i]) mod q
06 ft $\leftarrow$ (10*ft + T[i]) mod q
07 for s $\leftarrow$ 0 to n – m // matching
08 if fp = ft then // run a loop to compare strings
09 if P[0..m-1] = T[s..s+m-1] return s
10 ft $\leftarrow$ ((ft - T[s]*c)*10 + T[s+m]) mod q
11 return -1

- How many character comparisons are done if 
  $T = \"2531978\"$ and $P = \"1978\"$ (and $q = 7$)?
Analysis

- If $q$ is a prime, the hash function distributes $m$-digit strings evenly among the $q$ values
  - Thus, only every $q$-th value of shift $s$ will result in matching fingerprints (which will require comparing strings with $O(m)$ comparisons)

- Expected running time (if $q > m$):
  - Preprocessing: $O(m)$
  - Outer loop: $O(n-m)$
  - All inner loops: $\frac{n-m}{q} \cdot m = O(n-m)$
  - Total time: $O(n-m)$

- Worst-case running time: $O(nm)$
Rabin-Karp in Practice

- If the alphabet has \( d \) characters, interpret characters as radix-\( d \) digits (replace 10 with \( d \) in the algorithm).

- Choosing prime \( q > m \) can be done with randomized algorithms in \( O(m) \), or \( q \) can be fixed to be the largest prime so that \( 10^*q \) fits in a computer word.

- Rabin-Karp is simple and can be easily extended to two-dimensional pattern matching.
Searching in $n$ comparisons

- The goal: each character of the text is compared only once!
- Problem with the naïve algorithm:
  - Forgets what was learned from a partial match!
  - Examples:
    - $T = "\text{Tweedledee and Tweedledum}"$ and $P = "\text{Tweedledum}"$
    - $T = "\text{pappappappar}"$ and $P = "\text{pappar}"$
General situation

- **State of the algorithm:**
  - Checking shift $s$,
  - $q$ characters of $P$ are matched,
  - we see a non-matching character $\alpha$ in $T$.

- **Need to find:**
  - Largest prefix "$P-$" such that it is a suffix of $P[0..q-1]\alpha$:
    - New $q' = \max\{k \leq q \mid P[0..k-1] = P[q-k+1..q-1]\alpha\}$
Finite automaton search

Algorithm:
- **Preprocess:**
  - For each $q$ ($0 \leq q \leq m-1$) and each $\alpha \in \Sigma$ pre-compute a new value of $q$. Let’s call it $\sigma(q,\alpha)$
  - Fills a table of a size $m|\Sigma|
- **Run through the text**
  - Whenever a mismatch is found ($P[q] \neq T[s+q]$):
    - Set $s = s + q - \sigma(q,\alpha) + 1$ and $q = \sigma(q,\alpha)$

Analysis:
- ☺️ Matching phase in $O(n)$
- ☹️ Too much memory: $O(m|\Sigma|)$, too much preprocessing: at best $O(m|\Sigma|)$. 
Prefix function

- **Idea:** forget unmatched character ($\alpha$)!

- **State of the algorithm:**
  - Checking shift $s$,
  - $q$ characters of $P$ are matched,
  - we see a non-matching character $\alpha$ in $T$.

- **Need to find:**
  - Largest prefix "$P-$" such that it is a suffix of $P[0..q-1]$:
    - New $q' = \pi[q] = \max\{k < q \mid P[0..k-1] = P[q-k..q-1]\}$
Prefix table

- We can pre-compute a prefix table of size $m$ to store values of $\pi[q]$ ($0 \leq q \leq m$)

<table>
<thead>
<tr>
<th>$P$</th>
<th>$p$</th>
<th>$a$</th>
<th>$p$</th>
<th>$p$</th>
<th>$a$</th>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$\pi[q]$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

- Compute a prefix table for: $P = \text{"dadadu"}$
Knuth-Morris-Pratt Algorithm

**KMP-Search** $(T, P)$

01 $\pi \leftarrow \text{Compute-Prefix-Table}(P)$
02 $q \leftarrow 0$    // number of characters matched
03 for $i \leftarrow 0$ to $n-1$  // scan the text from left to right
    04 while $q > 0$ and $P[q] \neq T[i]$ do
    05      $q \leftarrow \pi[q]$
    06    if $P[q] = T[i]$ then $q \leftarrow q + 1$
    07     if $q = m$ then return $i - m + 1$
08 return $-1$

- **Compute-Prefix-Table** is essentially the same KMP search algorithm performed on $P$.

- **What is the running time?**
Analysis of KMP

- Worst-case running time: $O(n+m)$
  - Main algorithm: $O(n)$
  - Compute-Prefix-Table: $O(m)$
- Space usage: $O(m)$