String Matching (Chapter 32)

- Text is an array $T[1..n]$, pattern is an array $P[1..m]$, and $m \leq n$.
- Elements of $P$ and $T$ are characters from alphabet $\Sigma$.
- We want to find where pattern $P$ occurs in text $T$.
- Find all valid shifts with which a pattern $P$ occurs in text $T$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preprocessing Time</th>
<th>Matching Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Rabin-Karp</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Finite automaton</td>
<td>$O(m\cdot</td>
<td>\Sigma</td>
</tr>
<tr>
<td>Knuth-Morris-Pratt</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Differing Approaches:

• Brute-force algorithm
•

Running Time
Prefix and Suffix

Prefix:
For any strings \( x \) and \( y \) and any character \( a \), if \( x \sqsubseteq y \) then \( xa \sqsubseteq y \).

Suffix:
For any strings \( x \) and \( y \) and any character \( a \), if \( x \sqsubseteq y \) then \( ya \sqsubseteq x \).

Examples:
- \( ab \, \sqsubseteq \, abcca \)
- \( abcca \, \sqsubseteq \, abcba \)
- \( cca \, \sqsubseteq \, abcca \)
- \( \epsilon \, \sqsubseteq \, abcba \)

• For any strings \( x \) and \( y \) and any character \( a \), if \( x \, \sqsubseteq \, y \) then \( xa \, \sqsubseteq \, ya \).

• Reflexive?
• Symmetric?
• Transitive?

Notation and Terminology

Strings:
- \( \Sigma \) set of symbols/characters
- \( \Sigma^* \) set of all finite length strings formed from characters in \( \Sigma \)
- Zero-length string \( \epsilon \) is in \( \Sigma^* \)
- Length of string \( x \) denoted by \( |x| \)
- Set of all finite-length strings formed from characters in \( \Sigma \)
- Zero-length string is in \( \Sigma^* \)
- Set of symbols/characters
Comparing Strings

Comparing two equal-length strings
- Might write this as $x = y$, but does not take constant time
- Say that $z$ is longest prefix shared between $x$ and $y$ ($z ≡ x$ and $z ≡ y$)
- Recall where this as $x = y$, but does not take constant time

Lemma 32.1: Overlapping-suffix lemma

Suppose there are $x$, $y$, and $z$ are strings such that $x ≡ z$ and $y ≡ z$.

Lemma 32.1: Overlapping-suffix lemma
Naive String-Matching Algorithm

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- Worst-case: text of length \( n \), pattern of length \( m \), must do \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.

- Naive: entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \). This entirely ignores information gained about the text for one value of \( s \) when it considers other values of \( s \).

- Running time is \( O(nm) \) operations.

- No preprocessing, so running time is its matching time.

- Running time \( O(nm) \) operations.
Overview

1. Naive String Matching
2. Rabin-Karp Algorithm
3. String Matching with Finite Automata

Questions from Textbook

32.1-2 Suppose that all characters in the pattern \( P \) are different. Show how to accelerate Naive-String-Matcher to run in time \( O(n) \), and analyze the running time of your algorithm.

32.1-4 Suppose we allow the pattern \( P \) to contain occurrences of a gap character ⋄ that can match any string of characters (even one of zero length). For example, ⋄ba ⋄ c occurs in ab ⋄ ba ⋄ cbacabacabas. Give a polynomial-time algorithm to determine whether such a pattern occurs in a given text \( T \), and analyze the running time of your algorithm.
Computing and $d$'s

\begin{align*}
[p]\cdot L + 01((p - [p]L) + \cdots + 01([p]_L + 01([p]_L)\cdots)) = 0 & \cdot \\
\text{Similarly, we can compute } 01 & \cdot \\
[p]\cdot d + 01((p - [p]d) + \cdots + 01([p]_d + 01([p]_d)\cdots)) = d & \cdot \\
\text{Can compute in time } d & \cdot \\
\end{align*}

Rabin-Karp Algorithm

- Makes use of elementary number theoretic notions
- Based on certain assumptions: average-case is better
- Matching time: worst-case $\Theta(m)$ preprocessor time

\begin{align*}
\text{Let } I & \cdot \\
\text{For a text } T & \cdot \\
\text{Given a pattern } P & \cdot \\
\text{Can view a length string as a } d \cdot \\
\text{Assume each character is a digit in base } & \cdot \\
\text{Assume } n \cdot \\
\text{Assume each character is a decimal digit } & \cdot \\
\text{Use } q & \cdot \\
\text{Compute } & \cdot \\
\text{Multiply } & \cdot \\
\text{Repeat for } & \cdot \\
\text{subject to } & \cdot \\
\text{Based on certain assumptions: average-case is better } & \cdot \\
\text{Matching time: worst-case } & \cdot \\
\text{Use } (w(1 + w - w)) & \cdot \\
\text{Preprocessor time } & \cdot \\
\end{align*}
Doing Comparison in Constant Time

• Compute \( p \) and \( t_s \) in mod \( q \)

• Old way of computing \( t_s \):
  \[
  t_s + 1 = 10(t_s - 10m - 1) + T(s + m + 1) \mod q
  \]
  Takes time \( O(m) \) due to multiplication of \( 10^{m-1} \) and \( T(s + m + 1) \)

• Facts about mod:
  \[
  (x + y) \mod q = (x \mod q + y \mod q) \mod q
  \]
  \[
  xy \mod q = (x + \cdots + x) \mod q = (x \mod q) \cdot (y \mod q)
  \]

• New way:
  \[
  t_s' + 1 = 10(t_s' - hT(s + 1)) + T(s + m + 1) \mod q
  \]
  \( h = 10^{m-1} \mod q \)
  Now computation of \( t_s \) done in size \( q \cdot 10 \) not \( m \)

• Pick \( q \) so that \( q \cdot 10 \) fits into a computer word and \( q \) is prime
  \( q \) prime: helps make \( b + p \) terms make \( b \) depend on whole substring

Computation

\( p \mod [1 + u + s]L + ([1 + s]L - s)01 = 1 + s \)

\( b \mod (x + \cdots + x) = b \mod (b + x) \)

Old way of computing:

• \( t_s \) can be arbitrarily long (size \( m \) in time \( O(m) \))

Preprocessing: compute \( d \mod q \)

But \( s \) can be arbitrarily long (size \( m \) in time \( O(m) \))

(\( m \mod q \))
\[
|z| = p \text{ where } p
\]

\[
b \mod (\lceil \frac{1 + w + s}{L} \rceil + \lceil \frac{|1 + s|}{L} \rceil \cdot p) = \frac{t + \gamma}{w - u > s}
\]

\[
s \text{ start a match with shift,}\]

\[
\lceil \frac{1}{w + s} \rceil + \lceil \frac{1}{w + s} \rceil = \lceil \frac{|1 + s|}{d} \rceil \mod q = \gamma
\]

\[
\text{ matching} \quad \frac{w - u}{\delta} = s \quad \text{get} \quad b \mod (\lceil \frac{1}{d} \rceil + \gamma p) = \eta
\]

\[
b \mod (\lceil \frac{1}{d} \rceil d + dp) = d
\]

\[
\text{ preprocessing} \quad \frac{w}{\delta} = i \quad \text{get} \quad 0 = \phi
\]

\[
0 = \delta
\]

\[
b \mod (\lceil \frac{1}{d} \rceil + \phi p) = \psi
\]

\[
\lceil \frac{1}{d} \rceil + \gamma i \mod d = u
\]

\[
\frac{b \cdot p \cdot d \cdot \phi}{\eta \cdot \psi} \rceil
\]

\section*{Code}

\section*{Spurious Hits}

- Hopefully spurious hits do not happen too often.
- The number of checks will take time.
- \( \gamma = d \) if there need to check. If \( \gamma = d \) anymore, then need to check all possibilities.
- But not all the way around.
- \( b \mod \gamma = b \mod d \iff \gamma = d \) - Spurious Hits.
String Matching with Finite Automata

- Faster yet: constant time per text character

Overview
- Naive String-Matching
- Rabin-Karp Algorithm

Finite Automaton
- \( Q \) is a finite set of states
- \( q_0 \in Q \) is the start state
- \( A \subseteq Q \) is a set of accepting states
- \( \Sigma \) is a finite input alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function
String-Matching Automata

- For any pattern $P$, it is possible to construct a FA.
  - FA is based on the pattern.
  - Transition from any state to its successor one.
  - FA monitors how much of pattern seen in processing text up to that point.
  - FA is in an accepting state whenever the pattern is fully seen (at end of input).
  - Other states correspond to all pattern characters seen so far.
  - A state for each successive character in pattern seen so far.
  - Start point: seen nothing of pattern.
  - Short point: seen nothing of pattern.
  - For any pattern $P$, if possible to construct a FA.

Accepting Strings

- FA starts in state $q_0$.
- Reads characters of input string one at a time.
- If FA is in state $b$ and reads character $a$,
  - Moves from state $b$ to state $g(a,b)$ (transition).
- FA is based on the input.
- $\phi$ induces an initial-state function $\phi : \Sigma \to \Sigma^*$,
  - For $a \in \Sigma$ and $P \in \Sigma^*$,
    - $\phi(\epsilon) = q_0$.
    - $\phi(a) = (\phi(a) \phi)$.

- Accidental Stings
  - $\phi$ induces a final-state function $\phi \in \Sigma^*$,
  - Otherwise if has rejected the string read so far.

- Whenever FA is in $b \in \Sigma^*$, it has accepted string read so far.

- Reads characters of input string one at a time.
  - FA starts in state $q_0$.
Some More Notation

Find all shifts \( s \) in range \( 0 \leq s \leq s \) such that

- String matching problem can be written as:

Similarly for text \( T \), denote the first \( k \) characters by \( T_k \).

- So \( P = \epsilon \) and \( P^m = P \).

- For prefix \( P^{m-1} \), denote the first \( k \) characters as \( P_k \).

Example: \( P = ababaca \)

```
 0 1 2 3 4 5 6 7
a b a b a c a
```

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
<td>8</td>
</tr>
</tbody>
</table>

```

Example: \( P = ababaca \)

```
 0 1 2 3 4 5 6 7 8 9 10 11
a b a b a b a c a b a
```

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>b</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>b</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>a</td>
<td>12</td>
</tr>
</tbody>
</table>

Example: \( P = ababaca \)
Defining the String Matching Automaton

- **State set** $Q$ is \{0, 1, ..., m\}.
- **Start state** $q_0$ is 0. State $m$ is only accepting state.
- **Transition function** $\delta$ is $\delta(q, a) = \sigma(Pq^a)$.
- Makes sense!
- We will be in the state that corresponds to the longest match of the prefix of $P$ with the amount of string we have processed so far.
- But does not tell us how to build the transition table.

Remember $\phi(w)$? Final state function.
- Outputs state that FA is in after processing $w$.
- Our definition of $\delta$ gives us $\phi(T_k) = \sigma(T_k)$.
- Just saying that the state that FA is in will match how much we are matching.

• Suffix Function

- FA must track prefixes of $P$ that are a suffix of the text so far.
- If longest suffix does not work, remember next longest one.
- If longest suffix is $\epsilon$, then it is there if $x\sqsubseteq y$.

**Suffix Function**
- $\max y = (x)\omega$.
- Implications of $\omega$.
- Example: $P = ab\omega c a a c a a$. Well defined since empty string $\epsilon$ is a suffix of every string.
- $\{x \sqsubseteq y : y\max = (x)\omega\}$.
- Suffix function corresponding to $P$ that is also a suffix of $x$.
- Length of the longest prefix of $P$ that is also a suffix of $x$.
- $\{m \sqsubseteq \}$.
Reading the Next Character

- If FA is in state $q$ at $i$th character of $T$ - $P_q$ is longest prefix of $P$ that is suffix of $T_i$
- FA then reads the next character $T[i + 1]$

Why is $\sigma(T[i]) = \sigma(P_q)$ important?

- $(v^b_L) \phi = (v^b_L) \phi$
- $v^b_P$ is longest suffix of $v^b_P$
- Everything that is important about $L$ (in terms of matching)
- $v^b_P$ is the longest prefix of $v^b_P$ that is a suffix of $L$
- $v^b_P$ is the longest prefix of $v^b_P$ that is a suffix of $L$
- $\sigma = [1 + i]_L$
- $P_A$ is in state $b$ if the character of $L$
**Code: Preprocessing Step**

\[
\text{COMPUTE-TRANSITION-FUNCTION}(P, \Sigma)
\]

1. \( \gamma = (p \cdot b)q \)
2. \( \forall \delta \in \Sigma \)
3. \( 1 - \gamma = \gamma \)
4. \( \exists \) for each character \( a \in \Sigma \)
5. \( 0 \) for \( m = 1 \)
6. \( \text{return} \)
7. \( \text{match occurs with shift } l - \text{m} \)

**Running time:** \( O(|\Sigma|^m) \)

- Can actually do this in \( O(|\Sigma|\text{m}) \)

**Code: Matcher**

\[
\text{FINITE-AUTOMATON-MATCHER}(T, Q, \Sigma, \delta, q_0, F)
\]

1. \( n = T: \) length \( \Sigma \)
2. \( q = 0 \) for \( i = 1 \) to \( n \)
3. \( q = \text{match occurs with shift } l \)
4. \( \text{return} \) if \( q = m \)
5. \( \text{print} \) "Pattern occurs with shift \\
\text{with}\\
\text{shift } l - \text{m} \)

**Matcher runs in \( \Theta(n) \) time**

- Assuming \( \delta \) is just a table lookup