Graduate Algorithms
CS673-2016F-20
String Matching

David Galles

Department of Computer Science
University of San Francisco
String Matching

- Given a source text, and a string to match, where does the string appear in the text?
- Example: ababbbabababa and abbab

```
  a b a b b a b b a b a a
    x x x x x
      x x x x x x
```
NAIVE-STRING-MATCHER\((T, P)\)

\[ n \leftarrow \text{length}[T] \]
\[ m \leftarrow \text{length}[P] \]

for \( s \leftarrow 0 \) to \( n - m \) do

  match \( \leftarrow \) true

  for \( j \leftarrow 1 \) to \( m \) do

    if \( T[i + j] \neq T[j] \) then

      match \( \leftarrow \) false

  if match then

    Print “Pattern occurs with shift” \( s \)
String Matching

- Time for naive string matching:
  - $O(n \times m)$
- What if we could compare strings in unit time?
20-3: String Matching

- Strings are over \{0 \ldots 9\}
- Example: Match 512 in 13512631842
  - We can consider strings/substrings to be integers
  - Do a comparison in a single instruction
Rabin-Karp

- Strings are over \{0 \ldots 9\}
- Example: Match 512 in 13512631842
  - Compare 512 to 135
  - Compare 512 to 351
  - Compare 512 to 512
  - \ldots etc
Example: Match 512 in 13512631842
• (relatively) easy to create 135
• How do we modify 135 to get 351?
Example: Match 512 in 13512631842
- (relatively) easy to create 135
- How do we modify 135 to get 351?
  - Remove first digit, shift remaining digits to left, append the next digit in the input
  - Subtract $1 \times 10^3$, multiply by 10, add 1
  - $t_{s+1} = 10 \times (t_s - 10^{m-1}T[s+1]) + T[s+m+1]$
20-7: Rabin-Karp

- This works great for matching numbers – what about strings of letters?
• This works great for matching numbers – what about strings of letters?
  • Strings of letters are just numbers in base 26
  • (ASCII strings are just numbers in base 256)
• Problems with this method?
This works great for matching numbers – what about strings of letters?

- Strings of letters are just numbers in base 26
- (ASCII strings are just numbers in base 256)

Problems with this method?

- Numbers get big fast – won’t fit in a single integer
- What can we do?
This works great for matching numbers – what about strings of letters?

- Strings of letters are just numbers in base 26
- (ASCII strings are just numbers in base 256)

Problems with this method?

- Numbers get big fast – won’t fit in a single integer
- – Use modular arithmetic
First, using base-$d$ numbers instead of base-10 numbers:

\[ t_{s+1} = d \times (t_s - T[s + 1] \times h) + T[s + m + 1] \]

- \( h = d^{m-1} \), computed once

Compare input number to \( t_1, t_2, \ldots t_{n-m} \)

Problem occurs when \( t_k \) could be too large to fit in an integer ...
Next, use modular arithmetic

\[ t_{s+1} = (d \times (t_s - T[s + 1] \times h) + T[s + m + 1]) \mod q \]

- \[ h = d^{m-1} \mod q \], computed once

Compare input number to \( t_1, t_2, \ldots t_{n-m} \)

Problems?
Next, use modular arithmetic

\[ t_{s+1} = (d \cdot (t_s - T[s + 1] \cdot h) + T[s + m + 1]) \mod q \]

- \( h = d^{m-1} \mod q \), computed once

Compare input number to \( t_1, t_2, \ldots, t_{n-m} \)

Problems?

- Spurious hits (could have \( t_k = \) input number, even if the strings are not the same
Source String 2359023141526739921

Matching 31415, \( q = 13 \)

\( 31415 \mod 13 = 7 \)

\( h = 10^4 \mod 13 = 3 \)

\( t_1 = 23590 \mod 13 = 8 \)

\[
t_2 = (d \ast (t_1 - T[1] \ast h) + T[1 + (m + 1)]) \mod q
\]

\[
= (10 \ast (8 - 2 \ast 3) + 2) \mod 13
\]

\[
= 9
\]
Source String 2359023141526739921

Matching 31415, $q = 13$

- $31415 \mod 13 = 7$
- $h = 10^4 \mod 13 = 3$
- $t_1 = 23590 \mod 13 = 8$

$$t_3 = (d \times (t_2 - T[2] \times h) + T[2 + (m + 1)]) \mod q$$

$$= (10 \times (9 - 3 \times 3) + 3) \mod 13$$

$$= 3$$
**20-16: Rabin-Karp**

- matching 31415
  - 31415 mod 13 = 7

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

- We get hits on:
  - 31415
  - 67399
Dealing with spurious hits
  • Every time we get a potential hit, check the actual strings

Running time:
  • $O(n)$ to go through list
  • $O(m)$ to verify each actual match
  • $O(m)$ to check each spurious hit
Running time:
- $O(n)$ to go through list
- $O(m)$ to verify each actual match
- $O(m)$ to check each spurious hit

$O(n + (v + s) \times m)$, were $v =$ # of actual hits, $m =$ # of spurious hits.

Expected running time: $O(n) + O(m(v + n/q))$
- Assuming expected # of spurious hits $= n/q$
20-19: **DFA**

- Set of states $Q$
- $q_0 \in Q$ start state
- $A \in Q$ accepting states
- $\Sigma$ input alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ Transition function
20-20: DFA

- Start in the initial state
- Go through the string, one character at a time, until the string is exhausted
- Determine if we are in a final state at the end of the string
  - If so, string is accepted
  - If not, string is rejected
20-21: DFA

- All strings over \{0,1\} that end in 111
All strings over \{0,1\} that end in 1001
You can use the DFA for all strings that end in 1001 to find all occurrences of the substring 1001 in a larger string:

- Start at the beginning if the larger string, in state $q_0$
- Go through the string one symbol at a time, moving through the DFA
- Every time we enter a final state, that’s a match
Creating transition function $\delta$:

- Create a new concept: $\sigma_P(x)$
  - Length of the longest prefix of $P$ that is a suffix of $x$
  - $P = aba$
  - $\sigma_P(cba) = 1, \sigma_P(abc) = 0, \sigma_P(cab) = 2, \sigma_P(caba) = 3$

- $P_k = \text{first } k \text{ symbols of } P$
Creating the states of the DFA

If the input pattern is \( P[1 \ldots m] \):

- DFA has \( m + 1 \) states, \( q_0 \ldots q_m \)
- State \( k \) “means” last \( k \) elements in the string so far match first \( k \) elements in \( P \)
20-26: DFA

- Pattern: 1001
- State \( k \) “means” last \( k \) elements in the string so far match first \( k \) elements in \( P \)
To find $\delta(q, a)$:

- Start with string $P_q$: first $q$ characters of $P$
- Append $a$, to get $P_qa$
- Find the longest prefix of $P$ that is a suffix of $P_qa$. 

**DFA**

$$\delta(q, a) = \sigma(P_qa)$$
• Building $\delta$:

$m \leftarrow \text{length}[P]$

for $q \leftarrow 0$ to $m$ do

for each character $a \in \Sigma$ do

$k \leftarrow \min(m + 1, q + 2)$

do

$k \leftarrow k - 1$

until $P_k = ] P_q a$

$\delta(q, a) \leftarrow k$
Example:

\[ P = \text{ababca}, \quad \text{String} = \text{cbababcababc} \]
20-30: DFA

- \( P = ababca \)

- \( P_0 \):
  - \( P_0a = a: q_1 \) \( \delta(q_0, a) = q_1 \)
  - \( P_0b = b: q_0 \) \( \delta(q_0, b) = q_0 \)
  - \( P_0c = c: q_0 \) \( \delta(q_0, c) = q_0 \)
20-31: **DFA**

- \( P = ababca \)
- \( P_1 : a \)
  - \( P_1a = aa: q_1 \) \( \delta(q_1, a) = q_1 \)
  - \( P_1b = ab: q_2 \) \( \delta(q_1, b) = q_2 \)
  - \( P_1c = ac: q_0 \) \( \delta(q_1, c) = q_0 \)
• $P = ababca$

• $P_2 : ab$
  • $P_2a = aba$: $q_3 \quad \delta(q_2, a) = q_3$
  • $P_2b = abb$: $q_0 \quad \delta(q_2, b) = q_0$
  • $P_2c = abc$: $q_0 \quad \delta(q_2, c) = q_0$
• \( P = ababca \)

• \( P_3 : aba \)
  • \( P_3a = abaa: q_1 \) \( \delta(q_3, a) = q_1 \)
  • \( P_3b = abab: q_4 \) \( \delta(q_3, b) = q_4 \)
  • \( P_3c = abac: q_0 \) \( \delta(q_3, c) = q_0 \)
20-34: **DFA**

- \( P = ababca \)
- \( P_4 : abab \)
  - \( P_4a = ababa: q_3 \) \[ \delta(q_4, a) = q_3 \]
  - \( P_4b = ababb: q_0 \) \[ \delta(q_4, b) = q_0 \]
  - \( P_4c = ababc: q_5 \) \[ \delta(q_4, c) = q_5 \]
• $P = ababca$
• $P_5 : ababc$
  • $P_5a = ababca: q_6$ $\delta(q_5, a) = q_6$
  • $P_5b = ababcb: q_0$ $\delta(q_5, b) = q_0$
  • $P_5c = ababcc: q_0$ $\delta(q_5, c) = q_0$
$20-36$: **DFA**

- $P = ababca$
- $P_6: ababca$
  - $P_6a = ababcaa: q_1 \quad \delta(q_6, a) = q_1$
  - $P_6b = ababcab: q_2 \quad \delta(q_6, b) = q_2$
  - $P_6c = ababcac: q_0 \quad \delta(q_6, c) = q_0$
20-37: DFA

The diagram represents a Deterministic Finite Automaton (DFA) with states q0 to q6. The transitions are labeled with symbols a, b, and c. For example, state q0 transitions on a to q1, on b to q2, and on c to q1.
• Running time:
  • Time to build DFA: $O(m^3 \times |\Sigma|)$
    • (Can be improved to $O(m \times |\Sigma|)$
  • Time to run string through DFA: $O(n)$
• Total: $O(m^3 \times |\Sigma| + n)$
20-39: Knuth-Morris-Pratt

- New algorithm: Knuth-Morris-Pratt
- Same $O(n)$ matching time through the string as DFA
- Smaller preprocessing time $O(m)$, amortized
20-40: Knuth-Morris-Pratt

- Maximum overlap array
- How much can the string overlap with itself at each position?

a b a b a a b b a
0 0 1 2 3 1 2 0 1
• Maximum overlap array
• How much can the string overlap with itself at each position?

```
  a b a b a a b b a
  0 0 1 2 3 1 2 0 1
  a b | a b a a b b a
  | a b a b a a b b a
  | a b a b a a b b a
```
Knuth-Morris-Pratt

- Maximum overlap array
- How much can the string overlap with itself at each position?

```
  a b a b a a b b a a b b a
  0 0 1 2 3 1 2 0 1
  a b a b a a b b a a b b a
  a b a b a a b b a a b b a
```
20-43: Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

```
abababaabbaba
001231201
abab|a|abbbab
ab|abaabbba
```
20-44: Knuth-Morris-Pratt

- Maximum overlap array
- How much can the string overlap with itself at each position?

```
  a b a b a a a b b a a
  0 0 1 2 3 1 2 0 1
  a b a b a a | b | b a
  a | b | a b a a a b b a
```
20-45: Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

a b a b a a b b b a
0 0 1 2 3 1 2 0 1
a b a b a a b | b | a
  | a b a b a a b b b a
Knuth-Morris-Pratt

- Maximum overlap array
- How much can the string overlap with itself at each position?

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>
```

```
Knuth-Morris-Pratt

Prefix Function $\pi$:
- $\pi[q] = \max\{k : k < q \land P_k = P_q\}$
- $\pi[q]$ is the length of the longest prefix of $P$ that is a proper suffix of $P_q$
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

|   | a | b | a | b | a | b | b | a | b | b | a | b | b | a | b | a | b | a | b | b |
| a | b | a | b | a | b | b | a | b | b | a | b | b | a | b | a | b | a | b | b |
| a | b | a | b | b |

$\pi$: 

| a | b | a | b | b | 0 | 0 | 1 | 2 | 0 |
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

| a | b | a | b | a | b | b | a | b | b | a | b | b | a | b | a | b | a | b | a | b | b |
| a | b | a | b | a | b | b | a | b | b | a | b | b | a | b | a | b | a | b | a | b | b | a | b | b | a | b | b |
20-50: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

```
  a b | a | b | a | b | b | a | b | b | a | b | b | a | b | a | b | a | b | b | a | b | b | a | b | b | a | b | b | a | b | b
```

$\pi:$

```
  a | b | a | b | b | 0 | 0 | 1 | 2 | 0
```
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

```
abab
abab
```

$\pi$:

```
 a  b  a  b  b
0  0  1  2  0
```
Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:
\[
\begin{array}{ccccccccccccc}
  a & b & a & b & a & b & a & b & b & b & a & b & a & b & a & b & b & b \\
\end{array}
\]

Letter Mismatch
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: $ababb$

Input String:

```
  abab
  a  b  a  b 
  a b a b 
  a  b  a  b
```

$\pi$: 

```
  a  b  a  b  b 
  0  0  1  2  0 
```
Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

\[
\begin{align*}
  &\text{abab} | \text{ab} | \text{babbaababbab} \\
  &\text{abab} | \text{bab} \\
\end{align*}
\]
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

Complete Match!
Try to match pattern to input
When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

<table>
<thead>
<tr>
<th>$\pi$:</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Input String:

| a | b | a | b | a | b | b | a | b | b | a | b | b | a | b | a | b | a | b | b | a | b | b | a | b | b |
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

```
  a b a b a b b a b b b a b a b a b b b
  a b a b b b
```

$\pi$: 

```
  a b a b b b
  0 0 1 2 0
```
Try to match pattern to input
When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:
\[
\begin{array}{cccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} & \text{a} \\
\end{array}
\begin{array}{cccccccc}
\text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} & \text{a} & \text{b} \\
\text{a} & \text{b} & \text{b} & \text{b} & \text{a} & \text{b} & \text{b} & \\
\end{array}
\]

Letter Mismatch
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

```
  a b a b a b b a b b a b b
  b a b a b a b a b b
  a b a b b
```

$\pi$:

```
  a b a b b b
  0 0 1 2 0
```
20-60: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

\[\pi:\]

\[
\begin{array}{cccccc}
a & b & a & b & b & b \\
0 & 0 & 1 & 2 & 0 & 0
\end{array}
\]

Input String:

\[
\begin{array}{ccccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} & \text{a} & \text{b} & \text{b} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{b} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{b}
\end{array}
\]
Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

$\pi$: 

| a | b | b | a | b |
|---|---|---|---|
| 0 | 0 | 1 | 2 | 0 |
Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

**Pattern:** ababb

**Input String:**
```
  ababb a b a b b a b b a b b a b b a b a b b a b a b b
```

$\pi$:
```
  a b a b b
  0 0 1 2 0
```
• Try to match pattern to input
• When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:
\[
\begin{array}{ccccccccc}
  a & b & a & b & a & b & b & a & b & b & a & b & a & b & b & b & a & b & b \\
\end{array}
\]

$\pi$:
\[
\begin{array}{cccccc}
  a & b & a & b & b & b \\
  0 & 0 & 1 & 2 & 0 \\
\end{array}
\]
• Try to match pattern to input
• When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:
\[ a \ b \ a \ b \ a \ b \ b \ b \ a \ b \ b \ a \ b \ a \ b \ a \ b \ b \]

Letter Mismatch
Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

```
  a b a b a b b a b b a b b a b b
  a b a b b
  a b a b b
  a b a b b
```

$\pi$:

```
  a b a b b
  0 0 1 2 0
```
Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:
\[
\begin{array}{cccccccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} & \text{a} & \text{b} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} \\
\end{array}
\]

\[
\pi : \begin{array}{cccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{b} \\
\text{0} & \text{0} & \text{1} & \text{2} & \text{0} \\
\end{array}
\]
Try to match pattern to input
When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: ababb

Input String:

| a | b | a | b | a | b | b | a | b | b | a | b | b | a | b | a | b | a | b | b | a | b | a | b | a | b |

Complete Match
Try to match pattern to input

When a mismatch occurs, the \( \pi \) array tells us how far to shift the pattern forward

Pattern: abab

\[
\begin{array}{cccc}
\pi: & a & b & a & b \\
0 & 0 & 1 & 2 \\
\end{array}
\]

Input String: a b a b a b a b a b a b a b
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: abab

Input String: a b a b a b a b a b a b a b

$\pi$: 

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: abab

Input String: a b a b a b a b a b

$$\pi:\begin{array}{cccc}
a & b & a & b \\
0 & 0 & 1 & 2 \\
\end{array}$$
• Try to match pattern to input
• When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: abab

$\pi$:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Input String:

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>
```

Complete Match
Try to match pattern to input

When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: abab

Input String: a b a b a b a b a b

$\pi$: 

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
• Try to match pattern to input

• When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>π</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Input String:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complete Match
• Try to match pattern to input
• When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: abab

$\pi$: 

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Input String: 

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

20-74: Knuth-Morris-Pratt
Try to match pattern to input
When a mismatch occurs, the $\pi$ array tells us how far to shift the pattern forward

Pattern: abab

$\pi$:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Input String:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complete Match
• Creating $\pi$ array

$m \leftarrow \text{length}[P]$
$\pi[1] \leftarrow 0$
$k \leftarrow 0$

for $q \leftarrow 2$ to $m$ do
    while $k > 0$ and $P[k + 1] \neq P[q]$
        $k \leftarrow \pi[k]$
    if $P[k + 1] = P[q]$
        $k \leftarrow k + 1$
    $\pi[q] \leftarrow k$
KMP-Matching($T, P$)

\[\begin{align*}
    m & \leftarrow \text{length}[P] \\
    n & \leftarrow \text{length}[T] \\
    \pi & \leftarrow \text{ComputePI}(P) \\
    q & \leftarrow 0 \\
    \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
    \quad \text{while } q > 0 \text{ and } P[q + 1] \neq T[i] \text{ do} \\
    \quad \quad q & \leftarrow \pi[q] \\
    \quad \text{if } P[q + 1] = T[i] \text{ then} \\
    \quad \quad q & \leftarrow q + 1 \\
    \quad \text{if } q = m \text{ then} \\
    \quad \quad \text{Print “Match found at” } i - m \\
    \quad q & \leftarrow \pi[q]
\end{align*}\]
Running time:

- Preprocessing time: $\Theta(m)$
  - Using amortized analysis (aggregate)
- Running time: $\Theta(n)$
  - Using amortized analysis (aggregate)