5.3 Substring Search

- introduction
- brute force
- Knuth–Morris–Pratt
5.3 Substring Search

- introduction
- brute force
- Knuth–Morris–Pratt
Substring search

Goal. Find pattern of length $M$ in a text of length $N$.

typically $N >> M$
Substring search applications

**Goal.** Find pattern of length $M$ in a text of length $N$.

- **pattern** → NEEDLE
- **text** → INAHAYSTACK NEEDLE INA

*match*

typically $N \gg M$
Substring search applications

**Goal.** Find pattern of length $M$ in a text of length $N$.

Typically $N \gg M$

Substrings:

- **Pattern** → NEEDLE
- **Text** → INAHAYSTACK NEEDLE INA

**Computer forensics.** Search memory or disk for signatures, e.g., all URLs or RSA keys that the user has entered.

http://citp.princeton.edu/memory
Substring search applications

**Goal.** Find pattern of length $M$ in a text of length $N$. Typically $N >> M$

\[
\begin{align*}
\text{pattern} & \rightarrow \text{NEEDELLE} \\
\text{text} & \rightarrow \text{INAHAYSTACK} \ \underline{\text{NEEDELLE}} \ \text{INA}
\end{align*}
\]

Identify patterns indicative of spam.

- PROFITS
- LOSE WEIGHT
- herbal Viagra
- There is no catch.
- This is a one-time mailing.
- This message is sent in compliance with spam regulations.
Substring search applications

Electronic surveillance.

Need to monitor all internet traffic. (security)

No way! (privacy)

Well, we’re mainly interested in “ATTACK AT DAWN”

OK. Build a machine that just looks for that.

“ATTACK AT DAWN” substring search machine found
Substring search applications

Screen scraping. Extract relevant data from web page.

Ex. Find string delimited by `<b>` and `</b>` after first occurrence of pattern Last Trade:

```
...<tr><td class="yfnc_tablehead1" width="48%">
Last Trade:
</td></tr>
<td class="yfnc_tabledata1">
582.93</td></big></td>
</tr>
```

http://finance.yahoo.com/q?s=goog
Screen scraping: Java implementation

**Java library.** The `indexOf()` method in Java's `String` data type returns the index of the first occurrence of a given string, starting at a given offset.

```java
public class StockQuote {
    public static void main(String[] args) {
        String name = "http://finance.yahoo.com/q?s=";
        In in = new In(name + args[0]);
        String text = in.readAll();
        int start = text.indexOf("Last Trade:", 0);
        int from = text.indexOf("<b>", start);
        int to = text.indexOf("</b>", from);
        String price = text.substring(from + 3, to);
        StdOut.println(price);
    }
}
```

% java StockQuote goog
582.93

**Caveat.** Must update program if Yahoo format changes.
5.3 Substring Search

- introduction
- brute force
- Knuth–Morris–Pratt
- Boyer–Moore
- Rabin–Karp
Brute-force substring search

Check for pattern starting at each text position.

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>i+j</th>
<th>txt</th>
<th>pat</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>A</td>
<td>R A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A B R A</td>
<td>entries in red are mismatches</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>A B R A</td>
<td>entries in gray are for reference only</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>A B R A</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td>A B R A</td>
<td>entries in black match the text</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>A B R A</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>10</td>
<td>A B R A</td>
<td>match</td>
</tr>
</tbody>
</table>

return i when j is M
Brute-force substring search: Java implementation

Check for pattern starting at each text position.

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>i+j</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

```
public static int search(String pat, String txt) {
    int M = pat.length();
    int N = txt.length();
    for (int i = 0; i <= N - M; i++)
    {
        int j;
        for (j = 0; j < M; j++)
            if (txt.charAt(i+j) != pat.charAt(j))
                break;
        if (j == M) return i; // index in text where pattern starts
    }
    return N; // not found
}
```
Backup

In many applications, we want to avoid backup in text stream.
- Treat input as stream of data.
- Abstract model: standard input.

Brute-force algorithm needs backup for every mismatch.

```
A A A A A A A A A
A A A A A A A A A B
A A A A A A A B
---
A A A A A A A A A A A A A
A A A A A A A A A A A B
```

**Approach 1.** Maintain buffer of last $M$ characters.

**Approach 2.** Stay tuned.
Brute-force substring search: alternate implementation

Same sequence of character compares as previous implementation.
- $i$ points to end of sequence of already-matched characters in text.
- $j$ stores # of already-matched characters (end of sequence in pattern).

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$j$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$A$</td>
<td>$C$</td>
<td>$A$</td>
<td>$D$</td>
<td>$A$</td>
<td>$B$</td>
<td>$R$</td>
<td>$A$</td>
<td>$C$</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>$A$</td>
<td>$D$</td>
<td>$A$</td>
<td>$C$</td>
<td>$R$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>$A$</td>
<td>$D$</td>
<td>$A$</td>
<td>$C$</td>
<td>$R$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```java
public static int search(String pat, String txt)
{
    int i, N = txt.length();
    int j, M = pat.length();
    for (i = 0, j = 0; i < N && j < M; i++)
    {
        if (txt.charAt(i) == pat.charAt(j)) j++;
        else { i -= j; j = 0; }
    }
    if (j == M) return i - M;
    else return N;
}
```
Algorithmic challenges in substring search

Brute-force is not always good enough.

**Theoretical challenge.** Linear-time guarantee.  
**Practical challenge.** Avoid backup in text stream.
5.3 Substring Search

- introduction
- brute force
- Knuth–Morris–Pratt
Knuth–Morris–Pratt substring search

Intuition. Suppose we are searching in text for pattern BAAAAA.
- Suppose we match 5 chars in pattern, with mismatch on $6^{th}$ char.
- We know previous 6 chars in text are BAAAAB.
- Don't need to back up text pointer!

Knuth–Morris–Pratt algorithm. Clever method to always avoid backup!
Deterministic finite state automaton (DFA)

DFA is abstract string-searching machine.
- Finite number of states (including start and halt).
- Exactly one state transition for each char in alphabet.
- Accept if sequence of state transitions leads to halt state.

**internal representation**

<table>
<thead>
<tr>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>pat.charAt(j)</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>dfa[][j]</td>
<td></td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

If in state j reading char C:
- if j is 6 halt and accept
- else move to state dfa[c][j]

**graphical representation**
Knuth–Morris–Pratt demo: DFA simulation

A A B A C A A B A B A C A A A

<table>
<thead>
<tr>
<th>pat.charAt(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>dfa[][][j]</td>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A

B, C

A

A

A

A

A

B

B

B, C

C

B, C

B, C

B

B

B

B

B

B

B

B

B

B
Knuth–Morris–Pratt demo: DFA simulation

\[
\begin{array}{cccccc}
A & A & B & A & C & A \\
\end{array}
\]

\[
\begin{array}{ccccccc}
\text{pat.charAt}(j) & 0 & 1 & 2 & 3 & 4 & 5 \\
A & B & A & B & A & C \\
A & 1 & 1 & 3 & 1 & 5 & 1 \\
B & 0 & 2 & 0 & 4 & 0 & 4 \\
C & 0 & 0 & 0 & 0 & 0 & 6 \\
\end{array}
\]
Knuth–Morris–Pratt demo: DFA simulation

```
A A B A C A A B A B A C A A A
```

```
pat.charAt(j) 0 1 2 3 4 5
A B A B A C
A 1 1 3 1 5 1
B 0 2 0 4 0 4
C 0 0 0 0 0 6
```
Knuth–Morris–Pratt demo: DFA simulation

A  A  B  A  C  A  A  B  A  B  A  C  A  A

<table>
<thead>
<tr>
<th>pat.charAt(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>dfa[][j]</td>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

Diagram showing the DFA simulation with states 0 to 6 and transitions labeled with symbols A, B, and C.
Knuth–Morris–Pratt demo: DFA simulation

A A B A C A A B A B A C A A A

pat.charAt(j) 0 1 2 3 4 5
A B A B A C
A 1 1 3 1 5 1
B 0 2 0 4 0 4
C 0 0 0 0 0 6

 DFA simulation diagram
Knuth–Morris–Pratt demo: DFA simulation

A A B A C A A B A B A C A A

\[
\begin{array}{c|ccccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\hline
A & B & A & B & A & C \\
A & 1 & 1 & 3 & 1 & 5 & 1 \\
B & 0 & 2 & 0 & 4 & 0 & 4 \\
C & 0 & 0 & 0 & 0 & 0 & 6 \\
\end{array}
\]
Knuth–Morris–Pratt demo: DFA simulation

\[ \text{A A B A C A A B A B A C A A} \]

<table>
<thead>
<tr>
<th>( \text{pat.charAt(j)} )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

The DFA transitions are shown in the diagram below.
Knuth–Morris–Pratt demo: DFA simulation

A A B A C A A B A B A C A A

pat.charAt(j)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

\[
\text{dfa[][]} = \begin{bmatrix}
A & B & A & B & A & C \\
A & 1 & 3 & 1 & 5 & 1 \\
B & 0 & 2 & 0 & 4 & 0 \\
C & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\text{pat.charAt(j)} = \begin{bmatrix}
A & B \\
A & 1 \\
B & 0 \\
C & 0 \\
\end{bmatrix}
\]
Knuth–Morris–Pratt demo: DFA simulation

\[
\begin{array}{cccccccc}
\end{array}
\]

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
A & B & A & B & A & C \\
A & 1 & 1 & 3 & 1 & 5 & 1 \\
B & 0 & 2 & 0 & 4 & 0 & 4 \\
C & 0 & 0 & 0 & 0 & 0 & 6 \\
\end{array}
\]
Knuth–Morris–Pratt demo: DFA simulation

A A B A C A A B A B A C A A

pat.charAt(j) | 0 1 2 3 4 5
A B A B A C
A 1 1 3 1 5 1
B 0 2 0 4 0 4
C 0 0 0 0 0 6

```java
pat.charAt(j) {
    if (pat.charAt(j) == 'A') {
        // Transition to state 2
    } else if (pat.charAt(j) == 'B') {
        // Transition to state 3
    } else if (pat.charAt(j) == 'C') {
        // Transition to state 4
    }
}
```
Knuth–Morris–Pratt demo: DFA simulation

A A B A C A A B A B A C A A

```
pat.charAt(j)   0 1 2 3 4 5
A   B   A   B   A   C
A   1   1   3   1   5   1
B   0   2   0   4   0   4
C   0   0   0   0   0   6
```
Knuth–Morris–Pratt demo: DFA simulation

\[
\begin{array}{cccccc}
A & A & B & A & C & A \\
A & B & A & B & A & C \\
A & A & A & & & \\
\end{array}
\]

\[
\begin{array}{cccccc}
pat\:\text{charAt}(j) & 0 & 1 & 2 & 3 & 4 & 5 \\
A & B & A & B & A & C \\
A & 1 & 1 & 3 & 1 & 5 & 1 \\
B & 0 & 2 & 0 & 4 & 0 & 4 \\
C & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Knuth–Morris–Pratt demo: DFA simulation

<table>
<thead>
<tr>
<th>pat.charAt(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

Knuth–Morris–Pratt demo: DFA simulation

A A B A C A A B A B A C A A

<table>
<thead>
<tr>
<th>pat.charAt(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>dfa[[j]]</td>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

substring found
Interpretation of Knuth–Morris–Pratt DFA

Q. What is interpretation of DFA state after reading in $\text{txt}[i]$?
A. State = number of characters in pattern that have been matched.

Ex. DFA is in state 3 after reading in $\text{txt}[0..6]$.
Knuth–Morris–Pratt substring search: Java implementation

Key differences from brute-force implementation.
• Need to precompute `dfa[][][]` from pattern.
• Text pointer `i` never decrements.

```
public int search(String txt)
{
    int i, j, N = txt.length();
    for (i = 0, j = 0; i < N && j < M; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == M) return i - M;
    else return N;
}
```

Running time.
• Simulate DFA on text: at most $N$ character accesses.
• Build DFA: how to do efficiently? [warning: tricky algorithm ahead]
Knuth–Morris–Pratt substring search: Java implementation

Key differences from brute-force implementation.
- Need to precompute $\text{dfa}[][]$ from pattern.
- Text pointer $i$ never decrements.
- Could use input stream.

```java
public int search(In in) {
    int i, j;
    for (i = 0, j = 0; !in.isEmpty() && j < M; i++)
        j = dfa[in.readChar()][j];
    if (j == M) return i - M;
    else return NOT_FOUND;
}
```

No backup

Constructing the DFA for KMP substring search for $A \ B \ A \ B \ A \ C$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

$\text{dfa}[][]$
Knuth–Morris–Pratt demo: DFA construction

Constructing the DFA for KMP substring search for A B A B A C
Knuth–Morris–Pratt demo: DFA construction

Include one state for each character in pattern (plus accept state).

<table>
<thead>
<tr>
<th>pat.charAt(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dfa[][j]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

Constructing the DFA for KMP substring search for A B A B A C
**Knuth–Morris–Pratt demo: DFA construction**

**Match transition.** If in state \( j \) and next char \( c \) == \( \text{pat.charAt}(j) \), go to \( j+1 \).

- First \( j \) characters of pattern have already been matched
- Next char matches
- Now first \( j+1 \) characters of pattern have been matched

<table>
<thead>
<tr>
<th>( \text{pat.charAt}(j) )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constructing the DFA for KMP substring search for \( \text{A B A B A C} \)

\[ 0 \xrightarrow{A} 1 \xrightarrow{B} 2 \xrightarrow{A} 3 \xrightarrow{B} 4 \xrightarrow{A} 5 \xrightarrow{C} 6 \]
Mismatch transition: back up if c != pat.charAt(j).

Constructing the DFA for KMP substring search for A B A B A C
Knuth–Morris–Pratt demo: DFA construction

Mismatch transition: back up if \( c \neq \text{pat.charAt}(j) \).

<table>
<thead>
<tr>
<th>( \text{pat.charAt}(j) )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td></td>
<td>A</td>
<td>B</td>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>( B )</td>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>( B )</td>
<td></td>
<td>0</td>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>( C )</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

Constructing the DFA for KMP substring search for \( A B A B A C \)
Mismatch transition: back up if $c \neq \text{pat.charAt}(j)$.

Constructing the DFA for KMP substring search for A B A B A C
Knuth–Morris–Pratt demo: DFA construction

Mismatch transition: back up if $c \neq \text{pat}.\text{charAt}(j)$.

Constructing the DFA for KMP substring search for $A B A B A C$
Knuth–Morris–Pratt demo: DFA construction

Mismatch transition: back up if $c \neq \text{pat.charAt}(j)$.

Constructing the DFA for KMP substring search for A B A B A C

<table>
<thead>
<tr>
<th>pat.charAt(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Knuth–Morris–Pratt demo: DFA construction

Mismatch transition: back up if \( c \neq \text{pat.charAt}(j) \).

Constructing the DFA for KMP substring search for \( A B A B A C \)
Knuth–Morris–Pratt demo: DFA construction

Constructing the DFA for KMP substring search for A B A B A C

```
pat.charAt(j)   0  1  2  3  4  5
A     B     A     B     A     C
A     1     1     3     1     5     1
B     0     2     0     4     0     4
C     0     0     0     0     0     6
```
How to build DFA from pattern?

Include one state for each character in pattern (plus accept state).
How to build DFA from pattern?

**Match transition.** If in state \( j \) and next char \( c = \text{pat}.\text{charAt}(j) \), go to \( j+1 \).

- First \( j \) characters of pattern have already been matched
- Next char matches
- Now first \( j + 1 \) characters of pattern have been matched

<table>
<thead>
<tr>
<th>pat.charAt(( j ))</th>
<th>( 0 )</th>
<th>( 1 )</th>
<th>( 2 )</th>
<th>( 3 )</th>
<th>( 4 )</th>
<th>( 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>( B )</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>( C )</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{dfa}[j][j] \]
How to build DFA from pattern?

Mismatch transition. If in state \( j \) and next char \( c \neq \text{pat.charAt}(j) \), then the last \( j-1 \) characters of input are \( \text{pat}[1..j-1] \), followed by \( c \).

To compute \( \text{dfa}[c][j] \): Simulate \( \text{pat}[1..j-1] \) on DFA and take transition \( c \).

Running time. Seems to require \( j \) steps.

Ex. \( \text{dfa}[\text{A}][5] = 1 \) \( \text{dfa}[\text{B}][5] = 4 \)

simulate \( \text{BABAA} \) simulate \( \text{BABAB} \)
How to build DFA from pattern?

Mismatch transition. If in state \( j \) and next char \( c \neq \text{pat.charAt}(j) \), then the last \( j-1 \) characters of input are \( \text{pat}[1..j-1] \), followed by \( c \).

To compute \( \text{dfa}[c][j] \): Simulate \( \text{pat}[1..j-1] \) on DFA and take transition \( c \).

Running time. Takes only constant time if we maintain state \( x \).

Ex. \( \text{dfa['A'][5]} = 1 \)  \( \text{dfa['B'][5]} = 4 \)  \( x' = 0 \)

from state \( x \),

take transition 'A'

equal to \( \text{dfa['A'][x]} \)

from state \( x \),

take transition 'B'

equal to \( \text{dfa['B'][x]} \)

from state \( x \),

take transition 'C'

equal to \( \text{dfa['C'][x]} \)
Knuth–Morris–Pratt demo: DFA construction in linear time

Constructing the DFA for KMP substring search for A B A B A C

```
dfa[[j]]  0  1  2  3  4  5
pat.charAt(j) A B A B A C
A 1 1 3 1 5 1
B 0 2 0 4 0 4
C 0 0 0 0 0 6
```
Knuth–Morris–Pratt demo: DFA construction in linear time

Include one state for each character in pattern (plus accept state).

Constructing the DFA for KMP substring search for A B A B A C

<table>
<thead>
<tr>
<th>dfa[j][j]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
Knuth–Morris–Pratt demo: DFA construction in linear time

**Match transition.** For each state $j$, $\text{dfa}[\text{pat.charAt(j)}][j] = j+1$. 

- First $j$ characters of pattern have already been matched.
- Now first $j+1$ characters of pattern have been matched.

<table>
<thead>
<tr>
<th>$\text{pat.charAt(j)}$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{A}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>$\text{B}$</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{A}$</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{B}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{C}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constructing the DFA for KMP substring search for $\text{A B A B A C}$

Diagram: 

- State 0
- Transition on $\text{A}$ to State 1
- Transition on $\text{B}$ to State 2
- Transition on $\text{A}$ to State 3
- Transition on $\text{B}$ to State 4
- Transition on $\text{A}$ to State 5
- Transition on $\text{C}$ to State 6

State 0

State 1

State 2

State 3

State 4

State 5

State 6
Mismatch transition. For state 0 and char c != pat.charAt(j), set dfa[c][0] = 0.

Constructing the DFA for KMP substring search for A B A B A C
Mismatch transition. For each state $j$ and char $c \neq \text{pat.charAt}(j)$, set 
\[ \text{dfa}[c][j] = \text{dfa}[c][x]; \] then update $x = \text{dfa}[,\text{pat.charAt}(j)][x]$.

<table>
<thead>
<tr>
<th>$\text{pat.charAt}(j)$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td></td>
<td>3</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>0</td>
<td>2</td>
<td>4</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

Constructing the DFA for KMP substring search for A B A B A C
Knuth–Morris–Pratt demo: DFA construction in linear time

Mismatch transition. For each state $j$ and char $c \neq \text{pat.charAt}(j)$, set $\text{dfa}[c][j] = \text{dfa}[c][x]$; then update $x = \text{dfa[pat.charAt(j)]}[x]$.

Constructing the DFA for KMP substring search for A B A B A C
Mismatch transition. For each state $j$ and char $c != \text{pat.charAt}(j)$, set $\text{dfa}[c][j] = \text{dfa}[c][x]$; then update $x = \text{dfa}[\text{pat.charAt}(j)][x]$.

Constructing the DFA for KMP substring search for $A B A B A C$

<table>
<thead>
<tr>
<th>$\text{pat.charAt}(j)$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

$x =$ simulation of $B A$

0 $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ 4 $\rightarrow$ 5 $\rightarrow$ 6
**Mismatch transition.** For each state $j$ and char $c \neq \text{pat.charAt}(j)$, set $\text{dfa}[c][j] = \text{dfa}[c][x]$; then update $x = \text{dfa[pat.charAt(j)][x]}$. 

\[
\begin{array}{cccccc}
\text{pat.charAt}(j) & 0 & 1 & 2 & 3 & 4 & 5 \\
A & B & A & B & A & C \\
A & 1 & 1 & 3 & 1 & 5 \\
B & 0 & 2 & 0 & 4 \\
C & 0 & 0 & 0 & 0 & 6 \\
\end{array}
\]

Constructing the DFA for KMP substring search for $A\ B\ A\ B\ A\ C$

**Knuth–Morris–Pratt demo:** DFA construction in linear time
Mismatch transition. For each state \( j \) and char \( c \neq \text{pat.charAt}(j) \), set \( \text{dfa}[c][j] = \text{dfa}[c][x] \); then update \( x = \text{dfa}[\text{pat.charAt}(j)][x] \).

Constructing the DFA for KMP substring search for \( A B A B A C \)

<table>
<thead>
<tr>
<th>( \text{pat.charAt}(j) )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

\( x = \) simulation of \( B A B A C \)

<table>
<thead>
<tr>
<th>( \text{dfa}[c][j] )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

Diagram of the DFA:

- States: 0, 1, 2, 3, 4, 5, 6
- Transitions:
  - From 0: A to 1, B, C to 0
  - From 1: A to 2, B, C to 1
  - From 2: A to 3, B, C to 2
  - From 3: B to 4, A to 3
  - From 4: A to 5, B, C to 5
  - From 5: C to 6
  - From 6: Back to 0
- Initial state: 0
Mismatch transition. For each state $j$ and char $c \neq \text{pat}.\text{charAt}(j)$, set $\text{dfa}[c][j] = \text{dfa}[c][x]$; then update $x = \text{dfa}[\text{pat}.\text{charAt}(j)][x]$.

Constructing the DFA for KMP substring search for A B A B A C

<table>
<thead>
<tr>
<th>pat.charAt(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

$x = \text{simulation of B A B A C}$
Knuth–Morris–Pratt demo: DFA construction in linear time

Constructing the DFA for KMP substring search for $A B A B A C$

```
pat.charAt(j)  0  1  2  3  4  5
A   B   A   B   A   C
A   1   1   3   1   5   1
B   0   2   0   4   0   4
C   0   0   0   0   0   6
```
Constructing the DFA for KMP substring search: Java implementation

For each state j:

- Copy dfa[] [x] to dfa[] [j] for mismatch case.
- Set dfa[pat.charAt(j)][j] to j+1 for match case.
- Update x.

```java
public KMP(String pat) {
    this.pat = pat;
    M = pat.length();
    dfa = new int[R][M];
    dfa[pat.charAt(0)][0] = 1;
    for (int x = 0, j = 1; j < M; j++) {
        for (int c = 0; c < R; c++)
            dfa[c][j] = dfa[c][x];
        dfa[pat.charAt(j)][j] = j+1;
        x = dfa[pat.charAt(j)][x];
    }
}
```

Running time. $M$ character accesses (but space/time proportional to $RM$).
Knuth–Morris–Pratt: brief history

- Independently discovered by two theoreticians and a hacker.
  - Knuth: inspired by esoteric theorem, discovered linear algorithm
  - Pratt: made running time independent of alphabet size
  - Morris: built a text editor for the CDC 6400 computer
- Theory meets practice.

*SIAM J. COMPUT.*
Vol. 6, No. 2, June 1977

**FAST PATTERN MATCHING IN STRINGS**

DONALD E. KNUTH†, JAMES H. MORRIS, JR.‡ AND VAUGHAN R. PRATT¶

**Abstract.** An algorithm is presented which finds all occurrences of one given string within another, in running time proportional to the sum of the lengths of the strings. The constant of proportionality is low enough to make this algorithm of practical use, and the procedure can also be extended to deal with some more general pattern-matching problems. A theoretical application of the algorithm shows that the set of concatenations of even palindromes, i.e., the language \(\{\alpha R\}^*\), can be recognized in linear time. Other algorithms which run even faster on the average are also considered.