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 $N$. Typically $N >> M$

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 $N >> M$

Identify patterns indicative of spam.

- PROFITS
- LOSE WEIGHT
- 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:.

Screen scraping: Java implementation

Java library. The `indexOf()` method in Java’s string library 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
% java StockQuote msft
24.84
Substring Search

- 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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>10</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;
    }
    return N;  // not found
}
```

Worst case. ~ MN char compares.
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.

Approach 1. Maintain buffer of last $M$ characters.
Approach 2. Stay tuned.

Brute-force substring search: alternate implementation

Same sequence of char compares as previous implementation.

• $i$ points to end of sequence of already-matched chars in text.
• $j$ stores number of already-matched chars (end of sequence in pattern).

```java
public static int search(String pat, String txt) {
    int i, N = txt.length();
    int j, M = pat.length();
    for (; 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.

Now is the time for all people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many good people to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party.
Knuth-Morris-Pratt substring search

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

Knuth-Morris-Pratt algorithm. Clever method to always avoid backup. (!)

**DFA simulation**

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

DFA simulation

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

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

C
A
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
dfa[][][j] B 0 2 0 4 0 4
C 0 0 0 0 0 6

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

C
A
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
dfa[][][j] B 0 2 0 4 0 4
C 0 0 0 0 0 6

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

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

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
dfa[][][j] B 0 2 0 4 0 4
C 0 0 0 0 0 6

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

C
A
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
dfa[][][j] B 0 2 0 4 0 4
C 0 0 0 0 0 6

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

C
A
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
dfa[][][j] B 0 2 0 4 0 4
C 0 0 0 0 0 6

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

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

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
dfa[][][j] B 0 2 0 4 0 4
C 0 0 0 0 0 6

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

C
A
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
dfa[][][j] B 0 2 0 4 0 4
C 0 0 0 0 0 6

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

C
A
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
dfa[][][j] B 0 2 0 4 0 4
C 0 0 0 0 0 6

B, C A
A
B
A
A
C
B, C
B, C
A
A
B, C

C
A
A
A B A C
A
A B A B A C A A
**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 $\text{txt}[0..6]$.

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

<table>
<thead>
<tr>
<th>pat</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>

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.

```
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 NOT_FOUND;
}
```

Running time.

- Simulate DFA on text: at most $N$ character accesses.
- Build DFA: how to do efficiently? [warning: tricky algorithm ahead]

**Knuth-Morris-Pratt construction**

Include one state for each character in pattern (plus accept state).
Knuth-Morris-Pratt construction

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

- \( \text{first } j \text{ characters of pattern have already been matched} \)
- \( \text{next char matches} \)
- \( \text{now first } j+1 \text{ characters of pattern have been matched} \)

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

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

**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>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>3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constructing the DFA for KMP substring search for \( \text{A B A B A C} \)
Knuth-Morris-Pratt construction

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

Constructing the DFA for KMP substring search for \textcolor{blue}{A B A B A C}\n
\[
\begin{array}{ccccccc}
& 0 & 1 & 2 & 3 & 4 & 5 \\
pat.charAt(j) & A & B & A & B & A & C \\
\text{A} & 1 & 1 & 3 & 1 & 5 \\
dfa[[]][] & B & 0 & 2 & 0 & 4 & 0 \\
\text{C} & 0 & 0 & 0 & 0 & 6 \\
\end{array}
\]

Knuth-Morris-Pratt construction

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

Constructing the DFA for KMP substring search for \textcolor{blue}{A B A B A C}\n
\[
\begin{array}{ccccccc}
& 0 & 1 & 2 & 3 & 4 & 5 \\
pat.charAt(j) & A & B & A & B & A & C \\
\text{A} & 1 & 1 & 3 & 1 & 5 \\
dfa[[]][] & B & 0 & 2 & 0 & 4 & 0 \\
\text{C} & 0 & 0 & 0 & 0 & 6 \\
\end{array}
\]
How to build DFA from pattern?

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

<table>
<thead>
<tr>
<th>state</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>
</tbody>
</table>

pat.charAt(j) = 0 1 2 3 4 5

dfa[j][j] = A B A B A C

A

1

2

3

4

5

6

How to build DFA from pattern?

Match transition. If in state \( j \) and next char \( c \) == 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>state</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>
</tbody>
</table>

pat.charAt(j) = 0 1 2 3 4 5

dfa[j][j] = A B A B A C

<table>
<thead>
<tr>
<th>state</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>
</tbody>
</table>

0

A

B

2

B

A

4

A

5

C

6

How to build DFA from pattern?

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

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

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

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

simulated BABA; take transition 'A' = dfa['A'][3]

pat.charAt(j) = 0 1 2 3 4 5

dfa[j][j] = A B A B A C

state X

<table>
<thead>
<tr>
<th>state</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>
</tbody>
</table>

0

A

B

2

B

A

4

A

5

C

6

How to build DFA from pattern?

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

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

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

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

from state X, take transition 'A' = dfa['A'][3]

from state X, take transition 'B' = dfa['B'][X]

pat.charAt(j) = 0 1 2 3 4 5

dfa[j][j] = A B A B A C

state X

<table>
<thead>
<tr>
<th>state</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>
</tbody>
</table>

0

A

B

2

B

A

4

A

5

C

6

———
Knuth-Morris-Pratt 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

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

Mismatch transition. For state 0 and char $c \neq \text{pat.charAt}(j)$, set $\text{dfa}([c])[0] = 0$.

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 $X = \text{dfa}([\text{pat.charAt}(j)])[X]$.

X = simulation of empty string

Constructing the DFA for KMP substring search for A B A B A C
Mismatch transition. For each state $j$ and character $c \neq \text{pat}.\text{charAt}(j)$, 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**

**Knuth-Morris-Pratt construction (in linear time)**
Knuth-Morris-Pratt 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 \( X = \text{dfa[pat.charAt(j)][X]} \).

KMP substring search analysis

**Proposition.** KMP substring search accesses no more than \( M + N \) chars to search for a pattern of length \( M \) in a text of length \( N \).

**Pf.** Each pattern char accessed once when constructing the DFA; each text char accessed once (in the worst case) when simulating the DFA.

**Proposition.** KMP constructs \( \text{dfa}[][] \) in time and space proportional to \( R M \).

**Larger alphabets.** Improved version of KMP constructs \( \text{nfa}[] \) in time and space proportional to \( M \).
Knuth-Morris-Pratt: brief history

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

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

Substring Search

- Brute force
- Knuth-Morris-Pratt
- Boyer-Moore
- Rabin-Karp

Boyer-Moore: mismatched character heuristic

Intuition.
- Scan characters in pattern from right to left.
- Can skip as many as $M$ text chars when finding one not in the pattern.

Case 1. Mismatch character not in pattern.

Before

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

After

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>5</td>
</tr>
</tbody>
</table>

Mismatch character 'T' not in pattern: increment i one character beyond 'T'

Before: `text`...

| T | L |

After: `text`...

| T | L |

Mismatch character 'T' not in pattern: increment i one character beyond 'T'
Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 2a. Mismatch character in pattern.

before

\[
\begin{array}{c}
\text{txt} \quad \ldots \ldots \quad N \quad L \quad E \\
\text{pat} \quad N \quad E \quad E \quad D \quad L \quad E
\end{array}
\]

after

\[
\begin{array}{c}
\text{txt} \quad \ldots \ldots \quad N \quad L \quad E \\
\text{pat} \quad N \quad E \quad E \quad D \quad L \quad E
\end{array}
\]

mismatch character ‘N’ in pattern: align text ‘N’ with rightmost pattern ‘N’

Case 2b. Mismatch character in pattern (but heuristic no help).

before

\[
\begin{array}{c}
\text{txt} \quad \ldots \ldots \quad E \quad L \quad E \\
\text{pat} \quad N \quad E \quad E \quad D \quad L \quad E
\end{array}
\]

after

\[
\begin{array}{c}
\text{txt} \quad \ldots \ldots \quad E \quad L \quad E \\
\text{pat} \quad N \quad E \quad E \quad D \quad L \quad E
\end{array}
\]

mismatch character ‘E’ in pattern: increment i by 1

Boyer-Moore skip table computation

right = new int[R];
for (int c = 0; c < R; c++)
  right[c] = -1;
for (int j = 0; j < M; j++)
  right[pat.charAt(j)] = j;

A. Precompute index of rightmost occurrence of character \( c \) in pattern (-1 if character not in pattern).
public int search(String txt) {
    int N = txt.length();
    int M = pat.length();
    int skip;
    for (int i = 0; i <= N-M; i += skip) {
        skip = 0;
        for (int j = M-1; j >= 0; j--) {
            if (pat.charAt(j) != txt.charAt(i+j)) {
                skip = Math.max(1, j - right[txt.charAt(i+j)]);
                break;
            }
        }
        if (skip == 0) return i;
    }
    return N;
}
Efficiently computing the hash function

Modular hash function. Using the notation $i$, for `txt.charAt(i)`, we wish to compute

$$h_i = R^i R^M + h_{i-1} R^{M-1} + \ldots + h_{i-M+1} R^1 + h_0 \pmod{Q}$$

Intuition. $M$-digit, base-$R$ integer, modulo $Q$.

Horner’s method. Linear-time method to evaluate degree-$M$ polynomial.

```
// Compute hash for M-digit key
private long hash(String key, int M) {
    long h = 0;
    for (int j = 0; j < M; j++)
        h = (R * h + key.charAt(j)) % Q;
    return h;
}
```

Rabin-Karp substring search example

```
public class RabinKarp {
    public long patHash; // pattern hash value
    private int M; // pattern length
    private long Q; // modulus
    private int R; // radix
    private long RM; // R^(M-1) % Q

    public RabinKarp(String pat) {
        M = pat.length();
        R = 256;
        Q = longRandomPrime();
        RM = 1;
        for (int i = 1; i <= M; i++)
            RM = (R * RM) % Q;
        patHash = hash(pat, M);
    }

    private long hash(String key, int M) {
        long h = 0;
        // as before */
        return h;
    }

    public int search(String txt) {
        /* see next slide */
    }
}
```

Rabin-Karp: Java implementation
Rabin-Karp: Java implementation (continued)

Monte Carlo version. Return match if hash match.

```java
public int search(String txt)
{
    int N = txt.length();
    int txtHash = hash(txt, M);
    if (patHash == txtHash) return 0;
    for (int i = M; i < N; i++)
    {
        txtHash = (txtHash*R + txt.charAt(i)) % Q;
        if (patHash == txtHash) return i - M + 1;
    }
    return N;
}
```

Las Vegas version. Check for substring match if hash match; continue search if false collision.

Rabin-Karp fingerprint search

Advantages.
- Extends to 2d patterns.
- Extends to finding multiple patterns.

Disadvantages.
- Arithmetic ops slower than char compares.
- Las Vegas version requires backup.
- Poor worst-case guarantee.

Rabin-Karp analysis

Theory. If Q is a sufficiently large random prime (about $M N^2$), then the probability of a false collision is about $1 / N$.

Practice. Choose Q to be a large prime (but not so large as to cause overflow). Under reasonable assumptions, probability of a collision is about $1 / Q$.

Monte Carlo version.
- Always runs in linear time.
- Extremely likely to return correct answer (but not always!).

Las Vegas version.
- Always returns correct answer.
- Extremely likely to run in linear time (but worst case is $M N$).

Substring search cost summary

Cost of searching for an $M$-character pattern in an $N$-character text.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>version</th>
<th>operation count</th>
<th>backup</th>
<th>correct?</th>
<th>extra space</th>
</tr>
</thead>
<tbody>
<tr>
<td>brute force</td>
<td>—</td>
<td>$MN$</td>
<td>yes</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>Knuth-Morris-Pratt</td>
<td>full DFA (Algorithm 5.6)</td>
<td>$2N$</td>
<td>no</td>
<td>yes</td>
<td>MR</td>
</tr>
<tr>
<td></td>
<td>mismatch transitions only</td>
<td>$3N$</td>
<td>no</td>
<td>yes</td>
<td>M</td>
</tr>
<tr>
<td>Boyer-Moore</td>
<td>full algorithm</td>
<td>$3N$</td>
<td>yes</td>
<td>yes</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>mismatched char heuristic only (Algorithm 5.7)</td>
<td>$MN$</td>
<td>yes</td>
<td>yes</td>
<td>R</td>
</tr>
<tr>
<td>Rabin-Karp†</td>
<td>Monte Carlo (Algorithm 5.8)</td>
<td>$7N$</td>
<td>no</td>
<td>yes†</td>
<td>1</td>
</tr>
<tr>
<td>Las Vegas</td>
<td></td>
<td>$7N$</td>
<td>yes</td>
<td>yes</td>
<td>1</td>
</tr>
</tbody>
</table>

† probabilistic guarantee, with uniform hash function