5.3 Substring Search

- introduction
- brute force
- Knuth–Morris–Pratt
- Boyer–Moore
- Rabin–Karp
5.3 Substring Search

- introduction
- brute force
- Knuth–Morris–Pratt
- Boyer–Moore
- Rabin–Karp
**Substring search**

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

Typically $N \gg M$.

- **pattern**: NEEDLE
- **text**: INA HAY STACK NEEDLE INA

**match**
Substring search applications

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

![Substring search example](image)

Pattern: NEEDLE

Text: INAHAYSACK NEEDLE INA

Match:

- Typically $N \gg M$
Substring search applications

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

Typically $N \gg M$

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 $N$.

 typica$N \gg M$

\[
\begin{align*}
\text{pattern} & \rightarrow \text{NEEDELE} \\
\text{text} & \rightarrow \text{INAHAYSTACK NEEDLEINA}
\end{align*}
\]

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

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>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>txt</td>
<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>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>A</td>
<td>B</td>
<td>R</td>
<td>A</td>
<td>entry in red are mismatches</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A</td>
<td>B</td>
<td>R</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>A</td>
<td>B</td>
<td>R</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>A</td>
<td>B</td>
<td>R</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td>A</td>
<td>B</td>
<td>R</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>A</td>
<td>B</td>
<td>R</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>10</td>
<td>A</td>
<td>B</td>
<td>R</td>
<td>A</td>
<td>return i when j is M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

entries in gray are for reference only
Brute-force substring search: Java implementation

Check for pattern starting at each text position.

\[
\begin{array}{cccccccccc}
  i & j & i+j & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
  \hline
  4 & 3 & 7 & A & D & A & C & R \\
  5 & 0 & 5 & A & D & A & C & R \\
\end{array}
\]

```java
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
}
```
Substring search quiz 1

What is the worst-case running time of brute-force substring search as a function of the number of characters in the pattern $M$ and text $N$?

A. $M + N$
B. $M^2$
C. $MN$
D. $N^2$
E. I don't know.
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 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).

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<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>
<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>
<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;
}
```

explicit backup
Algorithmic challenges in substring search

Brute-force is not always good enough.

Theoretical challenge. Linear-time guarantee. ← fundamental algorithmic problem

Practical challenge. Avoid backup in text stream. ← often no space (or time) to save text

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 people to come to the aid of their party. Now is the time for a lot of good people to come to the aid of their party. Now is the time for all of the good people to come to the aid of their party. Now is the time for many of the good people to come to the aid of their party. Now is the time for each good person 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 all good Republicans 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 all good Democrats 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 all good Republicans 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 or 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 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 all good Democrats to come to the aid of their party.
5.3 Substring Search

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

**Intuition.** Suppose we are searching in text for pattern \( \text{BAAAA} \).

- Suppose we match 5 chars in pattern, with mismatch on 6\(^{th}\) char.
- We know previous 6 chars in text are \( \text{BAAAAB} \).
- 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>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</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

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

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\text{txt} & B & C & B & A & A & B & A & C
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\text{pat} & A & B & A & B & A & C
\end{array}
\]

length of longest prefix of \( \text{pat}[] \) that is a suffix of \( \text{txt}[0..i] \)

prefix of \( \text{pat}[] \)
Substring search quiz 2

Which state is the DFA in after processing the following input?


A. 0
B. 1
C. 3
D. 4
E. I don't know.
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.

```java
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 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;
}
```

Constructing the DFA for KMP substring search for A B A B A C 0 1 2 3 4 5

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

dfa[j]

X

pat.charAt(j) j

no backup
Knuth–Morris–Pratt demo: DFA construction

Constructing the DFA for KMP substring search for A B A B A C
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 $c \equiv \text{pat.charAt}(j)$, 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>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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

```
0  A  1  B  2  A  3  B  4  A  5  C  6
```
How to build DFA from pattern?

Mismatch transition. If in state \( j \) and next char \( c \) \(!=\ pat\text{.charAt}(j)\), 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. \( \text{dfa[}'A'\text{'][5] = 1} \quad \text{dfa[}'B'\text{'][5] = 4} \)

simulate BABAA

simulate BABAB

<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>( \text{pat\text{.charAt}(j)} )</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

still under construction (!)
How to build DFA from pattern?

Mismatch transition. If in state $j$ and next char $c \neq \text{pat}.\text{charAt}(j)$, 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.** $\text{dfa}['A'][5] = 1$

- from state $x$, take transition 'A'
- $= \text{dfa}['A'][x]$

$\text{dfa}['B'][5] = 4$

- from state $x$, take transition 'B'
- $= \text{dfa}['B'][x]$

$x' = 0$

- from state $x$, take transition 'C'
- $= \text{dfa}['C'][x]$

<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>
</tbody>
</table>

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.** $\text{dfa}['A'][5] = 1$

- from state $x$, take transition 'A'
- $= \text{dfa}['A'][x]$

$\text{dfa}['B'][5] = 4$

- from state $x$, take transition 'B'
- $= \text{dfa}['B'][x]$

$x' = 0$

- from state $x$, take transition 'C'
- $= \text{dfa}['C'][x]$
Constructing the DFA for KMP substring search for A B A B A C
Constructing the DFA for KMP substring search: Java implementation

For each state \( j \):

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

```
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 \( R M \)).
**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 character accessed once when constructing the DFA; each text character accessed once (in the worst case) when simulating the DFA.

**Proposition.** KMP constructs $\text{dfa}[][]$ in time and space proportional to $RM$.

**Larger alphabets.** Improved version of KMP constructs $\text{nfa}[]$ in time and space proportional to $M$. 

![KMP NFA for ABABAC](image)
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 \( \{a^n b^n\}^* \), can be recognized in linear time. Other algorithms which run even faster on the average are also considered.

Don Knuth  Jim Morris  Vaughan Pratt
A string $s$ is a **cyclic rotation** of $t$ if $s$ and $t$ have the same length and $s$ is a suffix of $t$ followed by a prefix of $t$.

**Problem.** Given two binary strings $s$ and $t$, design a linear-time algorithm to determine if $s$ is a cyclic rotation of $t$. 

<table>
<thead>
<tr>
<th>yes</th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROTATED STRING</td>
<td>A B A B B A B B B A B A</td>
<td>ROTATED STRING</td>
</tr>
<tr>
<td>STRING ROTATED</td>
<td>B A B B A B B A B A</td>
<td>GNIRTSDETATOR</td>
</tr>
</tbody>
</table>


5.3 Substring Search

- introduction
- 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.

```
<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>
```

\[ \text{return } i = 15 \]
Boyer–Moore: mismatched character heuristic

Q. How much to skip?

Case 1. Mismatch character not in pattern.

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

\[
\begin{align*}
\text{txt} & \quad \ldots \quad \ldots \quad \ldots \quad N \quad L \quad E \quad \ldots \quad \ldots \\
\text{pat} & \quad N \quad E \quad E \quad D \quad L \quad E
\end{align*}
\]

after

\[
\begin{align*}
\text{txt} & \quad \ldots \quad \ldots \quad \ldots \quad N \quad L \quad E \quad \ldots \quad \ldots \\
\text{pat} & \quad N \quad E \quad E \quad D \quad L \quad E
\end{align*}
\]

mismatch character 'N' in pattern: align text 'N' with rightmost pattern 'N'
Boyer–Moore: mismatched character heuristic

Q. How much to skip?

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

Mismatch character 'E' in pattern: align text 'E' with rightmost pattern 'E'?
Boyer–Moore: mismatched character heuristic

Q. How much to skip?

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

Mismatch character 'E' in pattern: increment i by 1
Boyer–Moore: mismatched character heuristic

Q. How much to skip?

A. Precompute index of rightmost occurrence of character \( c \) in pattern.
   (-1 if character not in pattern)

```java
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;
```

<table>
<thead>
<tr>
<th>c</th>
<th>N</th>
<th>E</th>
<th>E</th>
<th>D</th>
<th>L</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>B</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>D</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>M</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>N</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Boyer-Moore skip table computation
```java
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;
}
```
Boyer–Moore: analysis

**Property.** Substring search with the Boyer–Moore mismatched character heuristic takes about $\sim N / M$ character compares to search for a pattern of length $M$ in a text of length $N$.

**Worst-case.** Can be as bad as $\sim M N$.

<table>
<thead>
<tr>
<th>i</th>
<th>skip</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>txt</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

**Boyer–Moore variant.** Can improve worst case to $\sim 3 N$ character compares by adding a KMP-like rule to guard against repetitive patterns.
5.3 Substring Search

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

Michael Rabin
Dick Karp
Rabin–Karp fingerprint search

Basic idea = modular hashing.
- Compute a hash of \( \text{pat}[0..M-1] \).
- For each \( i \), compute a hash of \( \text{txt}[i..M+i-1] \).
- If pattern hash = text substring hash, check for a match.

\[
\begin{array}{cccc}
\text{pat.charAt}(i) & 0 & 1 & 2 & 3 & 4 \\
\hline
i & 2 & 6 & 5 & 3 & 5
\end{array}
\quad \% \ 997 = 613
\]

\[
\begin{array}{cccccccccccccccc}
\text{txt.charAt}(i) & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
i & 3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 & 5 & 3 & 5 & 8 & 9 & 7 & 9 & 3 \\
0 & 3 & 1 & 4 & 1 & 5 & \% \ 997 = 508 \\
1 & 1 & 4 & 1 & 5 & 9 & \% \ 997 = 201 \\
2 & 4 & 1 & 5 & 9 & 2 & \% \ 997 = 715 \\
3 & 1 & 5 & 9 & 2 & 6 & \% \ 997 = 971 \\
4 & 5 & 9 & 2 & 6 & 5 & \% \ 997 = 442 \\
5 & 9 & 2 & 6 & 5 & 3 & \% \ 997 = 929 \\
6 \leftarrow \text{return } i = 6 & 2 & 6 & 5 & 3 & 5 & \% \ 997 = 613
\end{array}
\]

modular hashing with \( R = 10 \) and \( \text{hash}(s) = s \mod 997 \)
**Modular arithmetic**

**Math trick.** To keep numbers small, take intermediate results modulo $Q$.

Ex. \[
(10000 + 535) \times 1000 \pmod{997} \\
= (30 + 535) \times 3 \pmod{997} \\
= 1695 \pmod{997} \\
= 698 \pmod{997}
\]

$10000 \equiv 30 \pmod{997}$

\[
\begin{align*}
(a + b) \pmod{Q} & = ((a \pmod{Q}) + (b \pmod{Q})) \pmod{Q} \\
(a \times b) \pmod{Q} & = ((a \pmod{Q}) \times (b \pmod{Q})) \pmod{Q}
\end{align*}
\]

two useful modular arithmetic identities
Efficiently computing the hash function

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

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \ldots + t_{i+M-1} R^0 \pmod{Q}$$

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

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

\[
\begin{array}{ccccccc}
\text{i} & 0 & 1 & 2 & 3 & 4 \\
\text{pat.charAt()} & 2 & 6 & 5 & 3 & 5 \\
\hline
0 & 2 & 6 & 5 & 3 & 5 & 5 \\
1 & 2 & 6 & 5 & 3 & 5 & 997 \\
2 & 2 & 6 & 5 & 3 & 5 & 997 \\
3 & 2 & 6 & 5 & 3 & 5 & 997 \\
4 & 2 & 6 & 5 & 3 & 5 & 997 \\
\end{array}
\]

\[
\begin{align*}
26535 &= 2 \times 10000 + 6 \times 1000 + 5 \times 100 + 3 \times 10 + 5 \\
&= (((2 \times 10 + 6) \times 10 + 5) \times 10 + 3) \times 10 + 5
\end{align*}
\]

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

**Challenge.** How to efficiently compute $x_{i+1}$ given that we know $x_i$.

\[
x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \ldots + t_{i+M-1} R^0
\]

\[
x_{i+1} = t_{i+1} R^{M-1} + t_{i+2} R^{M-2} + \ldots + t_{i+M} R^0
\]

**Key property.** Can update "rolling" hash function in constant time!

\[
x_{i+1} = (x_i - t_i R^{M-1}) R + t_{i+M}
\]

> current value  subtract leading digit  multiply by radix  add new trailing digit

(can precompute $R^{M-1}$)

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>current value</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>new value</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>text</td>
</tr>
</tbody>
</table>

\[
4 1 5 9 2 \quad current value
\]

\[
- 4 0 0 0 0 0
\]

\[
1 5 9 2 \quad subtract leading digit
\]

\[
* 1 0 \quad multiply by radix
\]

\[
1 5 9 2 0
\]

\[
+ 6 \quad add new trailing digit
\]

\[
1 5 9 2 6 \quad new value
\]
Rabin–Karp substring search example

First R entries: Use Horner's rule.

Remaining entries: Use rolling hash (and \% to avoid overflow).

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>% 997 = 3[Q]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>% 997 = (3*10 + 1) % 997 = 31[RM]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>% 997 = (31*10 + 4) % 997 = 314[R]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>% 997 = (314*10 + 1) % 997 = 150[RM]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>% 997 = (150*10 + 5) % 997 = 508[R]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>% 997 = ((508 + 3*(997 - 30))*10 + 9) % 997 = 201[RM]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>% 997 = (((201 + 1*(997 - 30))*10 + 2) % 997 = 715[R]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>% 997 = (((715 + 4*(997 - 30))*10 + 6) % 997 = 971[RM]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>% 997 = (((971 + 1*(997 - 30))*10 + 5) % 997 = 442[R]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>% 997 = (((442 + 5*(997 - 30))*10 + 3) % 997 = 929[RM]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>return i-M+1 = 6</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>% 997 = (((929 + 9*(997 - 30))*10 + 5) % 997 = 613[R]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-30 (mod 997) = 997 – 30
10000 (mod 997) = 30
public class RabinKarp
{
    private long patHash; // pattern hash value
    private int M; // pattern length
    private long Q; // modulus
    private int R; // radix
    private long RM1; // R^(M-1) % Q

    public RabinKarp(String pat) {
        M = pat.length();
        R = 256;
        Q = longRandomPrime();

        RM1 = 1;
        for (int i = 1; i <= M-1; i++)
            RM1 = (R * RM1) % Q;
        patHash = hash(pat, M);
    }

    private long hash(String key, int M) {
        // as before
    }

    public int search(String txt) {
        // see next slide
    }
}
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 + Q - RM*txt.charAt(i-M) % Q) % Q;
        txtHash = (txtHash*R + txt.charAt(i)) % Q;
        if (patHash == txtHash) return i - M + 1;
    }
    return N;
}
```

Las Vegas version. Modify code to check for substring match if hash match; continue search if false collision.
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 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 $M N$).
Rabin–Karp fingerprint search

Advantages.
- Extends to two-dimensional patterns.
- Extends to finding multiple patterns.

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

Q. How would you extend Rabin–Karp to efficiently search for any one of \( P \) possible patterns in a text of length \( N \)?
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 in input?</th>
<th>Correct?</th>
<th>Extra space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>operation count</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>guarantee</td>
<td>typical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>brute force</td>
<td>—</td>
<td>( MN )</td>
<td>( 1.1N )</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Knuth-Morris-Pratt</td>
<td>full DFA (Algorithm 5.6)</td>
<td>( 2N )</td>
<td>( 1.1N )</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>mismatch transitions only</td>
<td>( 3N )</td>
<td>( 1.1N )</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Boyer-Moore</td>
<td>full algorithm</td>
<td>( 3N )</td>
<td>( N/M )</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>mismatched char heuristic only (Algorithm 5.7)</td>
<td>( MN )</td>
<td>( N/M )</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Rabin-Karp( ^\dagger )</td>
<td>Monte Carlo (Algorithm 5.8)</td>
<td>( 7N )</td>
<td>( 7N )</td>
<td>no</td>
<td>yes( ^\dagger )</td>
</tr>
<tr>
<td></td>
<td>Las Vegas</td>
<td>( 7N )</td>
<td>( 7N )</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

\( ^\dagger \) probabilistic guarantee, with uniform hash function