5.3 Substring Search

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

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

Goal. Find pattern of length $m$ in a text of length $n.$

typically $n \gg m$

(pattern) $\rightarrow$ NEEDLE

text $\rightarrow$ INAHAYSACK NEEDLE INA

match
**Substring search applications**

**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 $n$.

Typically $n \gg m$

Pattern $\rightarrow$ NEEDLE

Text $\rightarrow$ INA HAY STACK NEEDLE INA

Match

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 $n \gg m$

- **Pattern:** NEEDLE
- **Text:** INA HAYSTACK NEEDLE INA

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

Web scraping. Extract relevant data from web page.

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

as rendered by browser

raw HTML

```html
...  
<tr>
    <td class="yfnc_tablehead1" width="48%">
        Last Trade:
    </td>
    <td class="yfnc_tabledata1">
        582.93
    </td>
    <td class="yfnc_tablehead1" width="48%">
        Trade Time:
    </td>
    <td class="yfnc_tabledata1">
        Nov 29 10:00 PM EST
    </td>
</tr>
```

http://finance.yahoo.com/q?s=goog
Web 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);
    }
}
```

Caveat. Must update program whenever Yahoo format changes.
5.3 Substring Search

- introduction
- brute force
- Knuth–Morris–Pratt
- Boyer–Moore
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 B R A</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A B R A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>A B R A</td>
<td></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></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></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>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

A B A C A D A B R A C

```
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; // number of characters that match
        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
}
```
What is the worst-case running time of brute-force substring search as a function of both the pattern length $m$ and text length $n$?

A. $m + n$
B. $m^2$
C. $mn$
D. $n^2$
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.

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>R</td>
<td>6</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
</tr>
<tr>
<td>R</td>
<td>6</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>0</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 substring search 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 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 all 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 many or 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. 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 all of the 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 each 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 many or 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
Knuth–Morris–Pratt substring search

**Intuition.** Suppose we are searching in text for pattern B A A A A A A A A A A A A A A A.
- Suppose we match 5 chars in pattern, with mismatch on 6th char.
Knuth–Morris–Pratt substring search

**Intuition.** Suppose we are searching in text for pattern 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 A A A A A A A A A A.

- Suppose we match 5 chars in pattern, with mismatch on 6th char.
- We know previous 6 chars in text must be B A A A A A.
- Don’t need to back up text pointer!

![Diagram showing text and pattern matching](image)

**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>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</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>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>

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

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C

A
B
C
B, C
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>
<th>7</th>
<th>8</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>
<td>C</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>

Diagram of DFA states and transitions.
Substring search: quiz 2

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

B A A B A B A B A

A. 0
B. 1
C. 3
D. 4
Substring search: quiz 3

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


A. 0
B. 1
C. 3
D. 4
E. 5
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;
}
```
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 \( c = \text{pat.charAt}(j) \), 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></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></td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td></td>
<td></td>
<td>4</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 \( c \) \( != \) \( \text{pat.charAt}(j) \), 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. \( \text{dfa['A'][5]} = 1 \) \( \text{dfa['B'][5]} = 4 \)

simulate BABAA

simulate BABAB

![DFA Diagram]

<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.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 \(!=\) pat.charAt(j), then the last j-1 characters of input are pat[1..j-1], followed by c.

To compute dfa[c][j]: Simulate pat[1..j-1] on DFA and take transition c.

Running time. Takes only constant time if we maintain state x.

Ex. dfa['A'][5] = 1
    from state x,
    take transition 'A'
    = dfa['A'][x]

dfa['B'][5] = 4
    from state x,
    take transition 'B'
    = dfa['B'][x]

x' = 0
    from state x,
    take transition 'C'
    = 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

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

dfa[][][j]  
A: [0, 1, 2, 3, 4, 5]  
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 \( R m \)).
**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 $m$. 
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.

**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 \( \{ \alpha \alpha^R \}^* \), can be recognized in linear time. Other algorithms which run even faster on the average are also considered.
Cyclic Rotation

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 $t$.

<table>
<thead>
<tr>
<th>yes</th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ROTATEDSTRING</strong></td>
<td><strong>ABABABBABBA</strong></td>
<td><strong>ROTATEDSTRING</strong></td>
</tr>
<tr>
<td><strong>STRINGROTATED</strong></td>
<td><strong>BABBABBAABA</strong></td>
<td><strong>GNIRTSDETATOR</strong></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>yes</th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ROTATED STRING</strong></td>
<td><strong>ABABBABBABABABA</strong></td>
<td><strong>ROTATED STRING</strong></td>
</tr>
<tr>
<td><strong>STRING ROTATED</strong></td>
<td><strong>BABBABBABAABAA</strong></td>
<td><strong>GNIRTSDETAOR</strong></td>
</tr>
</tbody>
</table>

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

**Solution.**

- Check that $s$ and $t$ are the same length.
- Search for $s$ in $t + t$ using Knuth–Morris–Pratt.
5.3 Substring Search

- introduction
- brute force
- Knuth–Morris–Pratt
- Boyer–Moore
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.

```
i  j  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
---+-----------------------------------------------
   |     text        | FINDINAHYSTACKNEDLEINA     |
   |  0 | NEEDLE → pattern |
   |  5 | NEEDLE          |
   | 11 |               no S in pattern               |
   | 15 | align N in text with  |
       | N in pattern         |
   |     return i = 15     |
```

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

Q. How much to skip?

Case 1. Mismatch character not in pattern.

before

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

i

after

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

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.

Mismatch character N in pattern: align text N with rightmost (why?) pattern N
Boyer–Moore: mismatched character heuristic

Q. How much to skip?

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

![Diagram showing text and pattern with mismatch character E. The text is aligned with the rightmost character of the pattern to find the position for the next comparison.](image)

**mismatch character E in pattern: align text E with rightmost pattern E?**
Q. How much to skip?

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

Mismatch character E in pattern: increment i by 1
Which text character is compared with the E next in Boyer–Moore?

A. R (index 5)
B. O (index 6)
C. O (index 12)
D. O (index 13)
Which text character is compared with the E next in Boyer–Moore?

A. O
B. R
C. E
D. J

Substring search: quiz 5

- **text**: BOOYEROBERTMOOREJS
- **pattern**: MOORE

<table>
<thead>
<tr>
<th>text</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>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>O</td>
<td>O</td>
<td>Y</td>
<td>E</td>
<td>R</td>
<td>O</td>
<td>B</td>
<td>E</td>
<td>R</td>
<td>T</td>
<td>M</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>E</td>
<td>J</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>E</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></td>
<td>M</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>E</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></td>
<td>M</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>E</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>
</tbody>
</table>
Q. How much to skip?

A. Precompute index of rightmost occurrence of character \( c \) in pattern.

\((-1\) if character not in pattern\)

\[
\begin{array}{cccccc}
\text{c} & \text{N} & \text{E} & \text{E} & \text{D} & \text{L} & \text{E} \\
\hline
\text{A} & -1 & -1 & -1 & -1 & -1 & -1 \\
\text{B} & -1 & -1 & -1 & -1 & -1 & -1 \\
\text{C} & -1 & -1 & -1 & -1 & -1 & -1 \\
\text{D} & -1 & -1 & -1 & -1 & \textbf{3} & \textbf{3} \\
\text{E} & -1 & -1 & \textbf{1} & \textbf{2} & 2 & \textbf{2} \\
\text{L} & -1 & -1 & -1 & -1 & \textbf{4} & 4 \\
\text{M} & -1 & -1 & -1 & -1 & -1 & -1 \\
\text{N} & -1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Rightmost occurrence:

- A: \(-1\)
- B: \(-1\)
- C: \(-1\)
- D: \(3\)
- E: \(5\)
- L: \(4\)
- M: \(-1\)
- N: \(-1\)

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;
```
Boyer–Moore: Java implementation

```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)]);  // compute skip value
                break;
            }
        }
        if (skip == 0) return i;  // match
    }
    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 $n$. **sublinear!**

**Worst-case.** Can be as bad as $\sim m n$.

<table>
<thead>
<tr>
<th>i 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>
</tr>
<tr>
<td>0</td>
<td></td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</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.