5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Review: substring search

- Knuth-Morris-Pratt (deterministic finite automaton)
- Boyer-Moore (skip-ahead heuristic)
- Rabin-Karp (modular hashing)

**Deterministic Finite Automaton**

- Abstract string-matching machine
- Represented by state-transition matrix
- Reaches accept state $\Rightarrow$ substring found

$$
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\hline
A & B & A & B & A & C \\
A & 1 & 1 & 3 & 1 & 5 & 1 \\
B & 0 & 2 & 0 & 4 & 0 & 4 \\
C & 0 & 0 & 0 & 0 & 0 & 6 \\
\end{array}
$$

![Deterministic Finite Automaton Diagram]
Trick question

Which search pattern does this DFA correspond to?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Either an A or a B followed by a C.

Every string corresponds to a DFA,
but not every DFA corresponds to a string

Every DFA corresponds to a pattern called a regular expression
(strings are a simple type of regular expression)
5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Finding interesting words

$ egrep '^[a-j]{8,}$' /usr/share/dict/words
acidified
beachhead
beheaded
headache

$ egrep '^[qwertyuiop]{10,}$' /usr/share/dict/words
perpetuity
proprietor
repertoire
typewriter

Subtle differences in syntax
XKCD t-shirt

/Everybody stand back/
I know regular expressions
Google allows a limited form of regular expression search

"it was the * of despair"

A Tale of Two Cities - Wikiquote
https://en.wikiquotes.org/wiki/A_Tale_of_Two_Cities ▾ Wikiquotes ▾
... of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair...

A Tale of Two Cities - Wikipedia, the free encyclopedia
A Tale of Two Cities (1859) is a novel by Charles Dickens, set in London and Paris before and ... it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we ...
Genomics

- Fragile X syndrome is a common cause of mental retardation.
- A human's genome is a string.
- It contains triplet repeats of CGG or AGG, bracketed by CGG at the beginning and CTG at the end.
- Number of repeats is variable and is correlated to syndrome.

pattern  \text{GCG(CGG|AGG) \ast CTG}

text  \text{GCGGCGTGTGTGCGAGAGAGTGGGTTAAAGCTG\underline{GCGCGGAGGC\underline{GGCTG}}GCGGCGGAGGGCTG}
/* Compilation:  javac NFA.java  
* Execution:  java NFA regexp text  
* Dependencies: Stack.java Bag.java Digraph.java DirectedDFS.java  
*  
*  
*  
*  */  

public class NFA  
{
  private Digraph G;  // digraph of epsilon transitions  
  private String regexp;  // regular expression  
  private int M;  // number of characters in regular expression  

  // Create the NFA for the given RE  
  public NFA(String regexp)  
  {
    this.regexp = regexp;  
    M = regexp.length();  
    Stack<Integer> ops = new Stack<Integer>();  
    G = new Digraph(M+1);  
    ...
  
  GNU source-highlight 3.1.4

  // ...  

  */  

  */
Google code search

Search public source code

Search via regular expression, e.g. ^java/.*\java$

Search Options

Package

Language

File Path

Class

Function

License

Case Sensitive

In Search Box

package:linux-2.6

lang:c++

file:(code|[^or]g)search

class:HashMap

function:toString

license:mozilla

case:yes

http://code.google.com/p/chromium/source/search
Prosite (computational biochemistry)

Database of protein domains, families and functional sites

PROSITE consists of documentation entries describing protein domains, families and functional sites as well as associated patterns and profiles to identify them [More... / References / Commercial users]. PROSITE is complemented by ProRule, a collection of rules based on profiles and patterns, which increases the discriminatory power of profiles and patterns by providing additional information about functionally and/or structurally critical amino acids [More...].


http://prosite.expasy.org
Even more applications

Test if a string matches some pattern.
- Scan for virus signatures.
- Process natural language.
- Specify a programming language.
- Access information in digital libraries.
- Search genome using PROSITE patterns.
- Filter text (spam, NetNanny, Carnivore, malware).
- Validate data-entry fields (dates, email, URL, credit card).
  ...

Parse text files.
- Compile a Java program.
- Crawl and index the Web.
- Read in data stored in ad hoc input file format.
- Create Java documentation from Javadoc comments.
  ...
# Regular expressions

A **regular expression** is a notation to specify a set of strings.

![possibly infinite]

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>concatenation</strong></td>
<td>AABAAB</td>
<td>AABAAB</td>
<td>every other string</td>
</tr>
<tr>
<td><strong>or</strong></td>
<td>AA</td>
<td>BAAB</td>
<td>AA BAAB</td>
</tr>
<tr>
<td><strong>star</strong></td>
<td>AB* A</td>
<td>AA ABBBBBBBBBA</td>
<td>AB ABABA</td>
</tr>
<tr>
<td>(aka closure)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>parentheses</strong></td>
<td>A(A</td>
<td>B) AAB</td>
<td>AAAAB ABAAB</td>
</tr>
<tr>
<td></td>
<td>(AB)* A</td>
<td>A ABABABABABABA</td>
<td>AA ABBA</td>
</tr>
</tbody>
</table>
Regular expressions: operator precedence

- Star applies only to immediately preceding char or parenthetical group
  \(AB^*A\)
- | has the lowest priority
  \(AA | BA^*B(AB)^*\)

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>concatenation</td>
<td>AABAAB</td>
<td>AABAAB</td>
<td>every other string</td>
</tr>
</tbody>
</table>
| or                         | AA | BAAB                    | AA
                               |                  | BAAB                    | every other string      |
| star (aka closure)         | AB^*A             | AA
                               |                  | ABBBBBBBBBA            | AB
                               |                  |                          | ABABABA                 |
| parentheses                | A(A | B)AAB         | AAAAB
                               |                  | ABAAB                   | every other string      |
|                            | (AB)^*A           | A                         | ABABABABABABA           |
|                            |                  | AA                         | ABBA                    |
Which one of the following strings is not matched by the regular expression \((A \, B \mid C \, * \, D) \, *\)?

A. A B A B A B
B. C D C C D D D D
C. A B C C D A B
D. A B D A B C A B D
E. I don't know.
Regular expression shortcuts

Additional operations further extend the utility of REs.

<table>
<thead>
<tr>
<th>operation</th>
<th>example RE</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>wildcard</td>
<td>.U.U.U.</td>
<td>CUMULUS JUGULUM</td>
<td>SUCCUBUS TUMULTUOUS</td>
</tr>
<tr>
<td>character class</td>
<td>[A-Za-z][a-z]*</td>
<td>wordCapitalized</td>
<td>camelCase 4illegal</td>
</tr>
<tr>
<td>one or more</td>
<td>A(BC)+DE</td>
<td>ABCDE ABCBCDE</td>
<td>ADE BCDE</td>
</tr>
<tr>
<td>exactly k</td>
<td>[0-9]{5} - [0-9]{4}</td>
<td>08540-1321 19072-5541</td>
<td>1111111111 166-54-111</td>
</tr>
</tbody>
</table>

**Note.** These operations are useful but not essential.

**Ex.** [A-E]+ is shorthand for (A|B|C|D|E)(A|B|C|D|E)*
Exercise

Simplify the following regular expression over the alphabet \{A, B\}:

\((B \mid A^*B^* \mid BAA^*)^*\)
Exercise

Simplify the following regular expression over the alphabet \{A, B\}:

\[(B \mid A^*B^* \mid BAA^*)^*\]

\[
\equiv (B \mid A)^* \\
\equiv .^*
\]
## Regular expression examples

RE notation is surprisingly expressive.

<table>
<thead>
<tr>
<th>regular expression</th>
<th>matches</th>
<th>does not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>.<em>SPB.</em></td>
<td>RASPBERRY CRISPBREAD</td>
<td>SUBSPACE SUBSPECIES</td>
</tr>
<tr>
<td></td>
<td>(substring search)</td>
<td></td>
</tr>
<tr>
<td>[0-9]{3}-[0-9]{2}-[0-9]{4}</td>
<td>166-11-4433 166-45-1111</td>
<td>11-55555555 8675309</td>
</tr>
<tr>
<td></td>
<td>(U. S. Social Security numbers)</td>
<td></td>
</tr>
<tr>
<td>[a-z]+@[a-z]+.(edu</td>
<td>com)</td>
<td><a href="mailto:wayne@princeton.edu">wayne@princeton.edu</a> <a href="mailto:rs@princeton.edu">rs@princeton.edu</a></td>
</tr>
<tr>
<td></td>
<td>(simplified email addresses)</td>
<td></td>
</tr>
<tr>
<td>$_A-Za-z[$_A-Za-z0-9]*</td>
<td>ident3 PatternMatcher</td>
<td>3a ident#3</td>
</tr>
<tr>
<td></td>
<td>(Java identifiers)</td>
<td></td>
</tr>
</tbody>
</table>

REs play a well-understood role in the theory of computation.
Exercise

Write a regular expression that matches strings of even length that start with an ‘A’ and contain a ‘B’.
Write a regular expression that matches strings of even length that start with an ‘A’ and contain a ‘B’.

Case 1: A and B are separated by an even number of characters
A (\.)* B (\.)*

Case 2: A and B are separated by an odd number of characters
A (\.)* . B . (\.)*

Put it together:
A(\.)*B(\.)* | A(\.)*.B.(\.)*

Optionally simplify:
A (\.)* (B | .B.) (\.*)
You can go crazy with regular expressions

Perl RE for valid RFC822 email addresses

http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html
Regular expression caveat

Writing a RE is like writing a program.
- Need to understand programming model.
- Can be easier to write than read.
- Can be difficult to debug.

“Some people, when confronted with a problem, think ‘I know I'll use regular expressions.’ Now they have two problems.”
— Jamie Zawinski

Bottom line. REs are amazingly powerful and expressive, but using them in applications can be amazingly complex and error-prone.
5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Duality between REs and DFAs

**RE.** Concise way to describe a set of strings.

**DFA.** Machine to recognize whether a given string is in a given set.

**Kleene's theorem.**

- For any DFA, there exists a RE that describes the same set of strings.
- For any RE, there exists a DFA that recognizes the same set of strings.

**Example:**

**RE**

\[0^* \mid (0^*10^*10^*10^*)^*\]

**DFA**

Number of 1's is a multiple of 3

Number of 1's is a multiple of 3

Stephen Kleene
Princeton Ph.D. 1934
Pattern matching implementation: basic plan (first attempt)

Overview is the same as for KMP.

- No backup in text input stream.
- Linear-time guarantee.

Underlying abstraction. Deterministic finite state automata (DFA).

Basic plan. [apply Kleene’s theorem]

- Build DFA from RE.
- Simulate DFA with text as input.

Bad news. Basic plan is infeasible (DFA may have exponential # of states).
Pattern matching implementation: basic plan (revised)

Overview is similar to KMP.

- No backup in text input stream.
- Quadratic-time guarantee (linear-time typical).

Underlying abstraction. Non-deterministic finite state automata (NFA).

Basic plan. [apply Kleene’s theorem]

- Build NFA from RE.
- Simulate NFA with text as input.

Q. What is an NFA?
Nondeterministic finite-state automata

Regular-expression-matching NFA.

- We assume RE enclosed in parentheses.
- One state per RE character (start = 0, accept = M).
- Match transition (change state and scan to next text char).
- Dashed $\varepsilon$-transition (change state, but don't scan text).
- Accept if any sequence of transitions ends in accept state.

Nondeterminism.

- One view: machine can guess the proper sequence of state transitions.
- Another view: sequence is a proof that the machine accepts the text.

NFA corresponding to the pattern \(( ( A \ast B | A C ) D )\)
**Non-deterministic finite-state automata**

**Q.** Is A A A A B D matched by NFA?

**A.** Yes, because some sequence of legal transitions ends in state 11.

---

**Diagram Description:**

- **Pattern:** \( ( ( A \ast B \mid A C ) D ) \)
- **Transition Explanation:**
  - **Match Transition:** Scan to next input character and change state.
  - **Epsilon-Transition:** Change state with no match.
  - **Accept State Reached:** Pattern found.

---

**NFA Diagram:**

- States: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
- Transitions:
  - \((0, A, 1)\)
  - \((1, *, 2)\)
  - \((2, B, 3)\)
  - \((3, |, 4)\)
  - \((4, A, 5)\)
  - \((5, C, 6)\)
  - \((6, D, 7)\)
  - \((7, )\)
  - \((8, 9, 10)\)
  - \((10, 11)\)

---

**NFA Corresponding to Pattern:**

\( ( ( A \ast B \mid A C ) D ) \)
Q. Is A A A A B D matched by NFA?
A. Yes, because some sequence of legal transitions ends in state 11.
   [ even though some sequences end in wrong state or get stuck ]

NFA corresponding to the pattern ( ( A * B | A C ) D )

Wrong guess if input is: A A A A A B D

No way out of state 7
Q. Is A A A C matched by NFA?
A. No, because no sequence of legal transitions ends in state 11. [ but need to argue about all possible sequences ]

Nondeterministic finite-state automata

NFA corresponding to the pattern ( ( A * B | A C ) D )
Nondeterminism

Q. How to determine whether a string is matched by an automaton?

DFA. Deterministic ⇒ easy (only one applicable transition at each step).

NFA. Nondeterministic ⇒ hard (can be several applicable transitions at each step; need to select the "right" ones!)

Q. How to simulate NFA?
A. Systematically consider all possible transition sequences. [stay tuned]
**NFA vs. quantum computers**

How are nondeterministic finite automata different from quantum computers?

Quantum computers are *actually, physically* nondeterministic.

With NFAs, we’re just pretending.
We can simulate them efficiently with regular computers (Turing machines).
We can’t do that with quantum computers (as far as we know).
5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
**NFA representation**

**State names.** Integers from 0 to $M$. 

**Match-transitions.** Keep regular expression in array `re[]`.

<table>
<thead>
<tr>
<th><code>re[]</code></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>(</td>
<td>A</td>
<td>*</td>
<td>B</td>
<td></td>
<td>A</td>
<td>C</td>
<td>)</td>
<td>D</td>
<td>)</td>
</tr>
</tbody>
</table>

**ε-transitions.** Store in a digraph $G$.

$0 \rightarrow 1$, $1 \rightarrow 2$, $1 \rightarrow 6$, $2 \rightarrow 3$, $3 \rightarrow 2$, $3 \rightarrow 4$, $5 \rightarrow 8$, $8 \rightarrow 9$, $10 \rightarrow 11$ 

NFA corresponding to the pattern `( ( A * B | A C ) D )`
NFA simulation

Q. How to efficiently simulate an NFA?
A. Maintain set of **all** possible states that NFA could be in after reading in the first $i$ text characters.

one step in simulating an NFA

Q. How to perform reachability?
A. DFS with multiple source vertices.
**Goal.** Check whether input matches pattern.

NFA corresponding to the pattern `( ( A * B | A C ) D )`
NFA simulation demo

Before reading any input characters:
- Find states reachable by $\varepsilon$-transitions from start state

input: A A B D

Diagram shows the states and transitions in the NFA.
Before reading any input characters:
- Find states reachable by $\epsilon$-transitions from start state

```
input
A A B D
```

```
set of states reachable via $\epsilon$-transitions from start
```

---

NFA simulation demo
NFA simulation demo

Before reading any input characters:
- Find states reachable by $\varepsilon$-transitions from start state

Input: A A B D

Set of states reachable via $\varepsilon$-transitions from start: \{0, 1, 2, 3, 4, 6\}
Before reading any input characters:

- Find states reachable by $\epsilon$-transitions from start state

set of states reachable via $\epsilon$-transitions from start : { 0, 1, 2, 3, 4, 6 }
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by \( \varepsilon \)-transitions

![Diagram of NFA simulation]

set of states reachable after matching A
Read next input character.

- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable after matching $A$ : \{ 3, 7 \}
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable via $\varepsilon$–transitions after matching A
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable via $\varepsilon$–transitions after matching A : \{ 2, 3, 4, 7 \}
### NFA simulation demo

**Read next input character.**
- Find states reachable by match transitions.
- Find states reachable by \( \epsilon \)-transitions

**Input**

```
A A B D
```

**Diagram**

```
0 1 2 3 4 5 6 7 8 9 10 11
( ( A * B | A C ) ) D )
```

**Set of states reachable via \( \epsilon \)-transitions after matching A:** \( \{ 2, 3, 4, 7 \} \)
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable after matching A A
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by \( \epsilon \)-transitions

set of states reachable after matching A A : \{ 3 \}
**NFA simulation demo**

**Read next input character.**

- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

**set of states reachable via $\varepsilon$–transitions after matching A A**
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable via $\varepsilon$–transitions after matching A A : \{ 2, 3, 4 \}
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by $\epsilon$-transitions

set of states reachable via $\epsilon$-transitions after matching A A : { 2, 3, 4 }
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable after matching A A B
NFA simulation demo

Read next input character.
  - Find states reachable by match transitions.
  - Find states reachable by $\varepsilon$-transitions

set of states reachable after matching A A B : \{ 5 \}
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable via $\varepsilon$–transitions after matching A A B
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by \( \epsilon \)-transitions

set of states reachable via \( \epsilon \)-transitions after matching A A B: \{ 5, 8, 9 \}
Read next input character.

- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

**set of states reachable via $\varepsilon$–transitions after matching A A B: \{ 5, 8, 9 \}**
NFA simulation demo

Read next input character.

- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable after matching A A B D
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable after matching A A B D : { 10 }
Read next input character.

- Find states reachable by match transitions.
- Find states reachable by $\epsilon$-transitions

NFA simulation demo

set of states reachable via $\epsilon$-transitions after matching A A B D
NFA simulation demo

Read next input character.
- Find states reachable by match transitions.
- Find states reachable by $\varepsilon$-transitions

set of states reachable via $\varepsilon$–transitions after matching $A \ A \ B \ D$ : { 10, 11 }
When no more input characters:

- Accept if any state reachable is an accept state.
- Reject otherwise.

set of states reachable: \{ 10, 11 \}
NFA simulation: analysis

**Proposition.** Determining whether an $N$-character text is recognized by the NFA corresponding to an $M$-character pattern takes time proportional to $M$ in the worst case.

**Pf.** For each of the $N$ text characters, we iterate through a set of states of size no more than $M$ and run DFS on the graph of $\varepsilon$-transitions. [The NFA construction we will consider ensures the number of edges $\leq 3M$.]
5.4 **Regular Expressions**

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Building an NFA corresponding to an RE

**States.** Include a state for each symbol in the RE, plus an accept state.

NFA corresponding to the pattern \(( ( A \ast B \mid A \ C ) \ D )\)
Building an NFA corresponding to an RE

**Concatenation.** Add match-transition edge from state corresponding to characters in the alphabet to next state.

**Alphabet.** A B C D

**Metacharacters.** ( ) . * |
Building an NFA corresponding to an RE

Parentheses. Add $\varepsilon$-transition edge from parentheses to next state.

NFA corresponding to the pattern $( ( A * B \mid A C ) D )$
Building an NFA corresponding to an RE

**Closure.** Add three $\varepsilon$-transition edges for each $*$ operator.

NFA corresponding to the pattern \(( ( A \ast B \mid A \ C ) \ D )\)
Building an NFA corresponding to an RE

2-way or. Add two \( \varepsilon \)-transition edges for each | operator.

2-way or expression

NFA corresponding to the pattern ( ( A \* B | A C ) D )
Building an NFA corresponding to an RE

**States.** Include a state for each symbol in the RE, plus an accept state.

**Concatenation.** Add match-transition edge from state corresponding to characters in the alphabet to next state.

**Parentheses.** Add \( \varepsilon \)-transition edge from parentheses to next state.

**Closure.** Add three \( \varepsilon \)-transition edges for each \(*\) operator.

**2-way or.** Add two \( \varepsilon \)-transition edges for each \(|\) operator.

NFA corresponding to the pattern \(( ( A \ast B | A \ C ) \ D )\)
How would you modify the NFA below to match \(((ABC^*)^+)\)?

A. Remove ε-transition edge 1\(\rightarrow\)7.
B. Remove ε-transition edge 7\(\rightarrow\)1.
C. Remove ε-transition edges 1\(\rightarrow\)7 and 7\(\rightarrow\)1.
D. *I don't know.*

NFA corresponding to the pattern \(((A\ B\ C\ *\ )\ *\ )\)
5.4 Regular Expressions

- regular expressions
- REs and NFAs
- NFA simulation
- NFA construction
- applications
Industrial-strength grep implementation

To complete the implementation:

- Add multiway or.
- Handle metacharacters.
- Support character classes.
- Add capturing capabilities.
- Extend the closure operator.
- Error checking and recovery.
- Greedy vs. reluctant matching.

Ex. Which substring(s) should be matched by the RE `<blink>.*</blink>`?

```plaintext
reluctant
<blink>text</blink> some text<blink>more text</blink>

reluctant

greedy
```
Regular expressions in the wild

Broadly applicable programmer's tool.

- Originated in Unix in the 1970s.
- Built in to many tools: grep, egrep, emacs, ....

```bash
% grep 'NEWLINE' */*.java  # print all lines containing NEWLINE which occurs in any file with a .java extension

% egrep '^[qwertyuiop]*[zxcvbnm]*$' words.txt | egrep '..........'

typewritten
```

- Built in to many languages: awk, Perl, PHP, Python, JavaScript, ....

```bash
% perl -p -i -e 's/from/to/g' input.txt  # replace all occurrences of from with to in the file input.txt

% perl -n -e 'print if /^[A-Z][A-Za-z]*$/' words.txt  # print all words that start with uppercase letter
```

do for each line
Regular expressions in Java

Validity checking. Does the input match the re?
Java string library. Use `input.matches(re)` for basic RE matching.

```java
public class Validate {
    public static void main(String[] args) {
        String regexp = args[0];
        String input = args[1];
        StdOut.println(input.matches(regexp));
    }
}
```

% java Validate "[_A-Za-z]$_A-Za-z0-9]*" ident123
true

% java Validate "[a-z]+@[a-z]+\.(edu|com)" rs@cs.princeton.edu
true

% java Validate "[0-9]{3}-[0-9]{2}-[0-9]{4}" 166-11-4433
true

legal Java identifier
valid email address (simplified)
Social Security number
Harvesting information

Goal. Print all substrings of input that match a RE.

% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgcgcggcggcgcggcggcgcggcgcggctg
 gcgctg
 gcgctg
 gcgctg
 gcgcgcggcggcggcggcggcgcggcgcggctg

harvest patterns from DNA

% java Harvester "http://(\w+/.)*(\w+)" http://www.cs.princeton.edu
http://www.w3.org
http://www.cs.princeton.edu
http://drupal.org
http://csguide.cs.princeton.edu
http://www.cs.princeton.edu
http://www.princeton.edu

harvest links from website
Harvesting information

RE pattern matching is implemented in Java's java.util.regex.Pattern and java.util.regex.Matcher classes.

```java
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class Harvester {
    public static void main(String[] args) {
        String regexp = args[0];
        In in = new In(args[1]);
        String input = in.readAll();
        Pattern pattern = Pattern.compile(regexp);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find()) {
            StdOut.println(matcher.group());
        }
    }
}
```

compile() creates a Pattern (NFA) from RE

matcher() creates a Matcher (NFA simulator) from NFA and text

find() looks for the next match

group() returns the substring most recently found by find()
Algorithmic complexity attacks

**Warning.** Typical implementations do not guarantee performance!

Unix grep, Java, Perl, Python

```java
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 1.6 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 3.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 9.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 23.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 62.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 161.6 seconds
```

SpamAssassin regular expression.

```java
% java RE "[a-z]+@[a-z]+([a-z\./]+\.[a-z]+) spammer@x ....................
```

- Takes exponential time on pathological email addresses.
- Attacker can use such addresses to DOS a mail server.
Not-so-regular expressions

Back-references.

- \$1 notation matches subexpression that was matched earlier.
- Supported by typical RE implementations.

\[
(\.+)(11\?)(1+) \quad // \text{beriberi couscous}
\]
\[
1?\^\((11?)\)1+ \quad // \ 1111 \ 11111 \ 11111111
\]

Some non-regular languages.

- Strings of the form \(ww\) for some string \(w\): beriberi.
- Unary strings with a composite number of 1s: 111111.
- Bitstrings with an equal number of 0s and 1s: 01110100.
- Watson-Crick complemented palindromes: atttcggaat.

Remark. Pattern matching with back-references is intractable.
Harvesting information in Java

RE pattern matching is implemented in Java's java.util.regex.Pattern and java.util.regex.Matcher classes.

```java
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class Harvester
{
    public static void main(String[] args)
    {
        String regexp = args[0];
        In in = new In(args[1]);
        String input = in.readAll();
        Pattern pattern = Pattern.compile(regexp);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find())
        {
            StdOut.println(matcher.group());
        }
    }
}
```

- **compile()** creates a Pattern (NFA) from RE
- **matcher()** creates a Matcher (NFA simulator) from NFA and text
- **find()** looks for the next match
- **group()** returns the substring most recently found by **find()**
Regular expressions in context

Regexes are powerful, but far less powerful than Java programs.

**Compiler.** A program that translates a program to machine code.
- KMP  string $\Rightarrow$ DFA.
- grep  RE $\Rightarrow$ NFA.
- javac Java language $\Rightarrow$ Java byte code.

<table>
<thead>
<tr>
<th></th>
<th>KMP</th>
<th>grep</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>string</td>
<td>RE</td>
<td>program</td>
</tr>
<tr>
<td>parser</td>
<td>unnecessary</td>
<td>check if legal</td>
<td>check if legal</td>
</tr>
<tr>
<td>compiler output</td>
<td>DFA</td>
<td>NFA</td>
<td>byte code</td>
</tr>
<tr>
<td>simulator</td>
<td>DFA simulator</td>
<td>NFA simulator</td>
<td>JVM</td>
</tr>
</tbody>
</table>
Algorithmic complexity attacks

**Warning.** Typical implementations do not guarantee performance!

Unix grep, Java, Perl, Python

<table>
<thead>
<tr>
<th>Command</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>`% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaaaaaac`</td>
</tr>
<tr>
<td>`% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbccc`</td>
</tr>
<tr>
<td>`% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaacccccccccc`</td>
</tr>
<tr>
<td>`% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaccdddddddde`</td>
</tr>
<tr>
<td>`% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaacccccccc`</td>
</tr>
<tr>
<td>`% java Validate &quot;(a</td>
<td>aa)*b&quot; aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaacccccccccc`</td>
</tr>
</tbody>
</table>

SpamAssassin regular expression.

% java RE "[a-z]+@[a-z]+([a-z\.]\+[a-z]*)" spammer@x..........................

- Takes exponential time on pathological email addresses.
- Attacker can use such addresses to DOS a mail server.
Summary of pattern-matching algorithms

**Programmer.**
- Implement substring search via DFA simulation.
- Implement RE pattern matching via NFA simulation.

**Theoretician.**
- RE is a compact description of a set of strings.
- NFA is an abstract machine equivalent in power to RE.
- DFAs, NFAs, and REs have limitations.

**You.**
- Core CS principles provide useful tools that you can exploit now.
- REs and NFAs provide introduction to theoretical CS.

**Example of essential paradigm in computer science.**
- Build the right intermediate abstractions.
- Solve important practical problems.