Pattern Matching

Regular Expressions

Test if a string matches some pattern.
- Process natural language.
- Scan for virus signatures.
- Search for information using Google.
- Access information in digital libraries.
- Retrieve information from Lexis/Nexis.
- Search-and-replace in a word processors.
- Filter text (spam, NetNanny, Carnivore, malware).
- Validate data-entry fields (dates, email, URL, credit card).
- Search for markers in human genome using PROSITE patterns.

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

String search. Search for given string in a large text file.

Regular expression.
- Natural and compact way to express multiple text patterns.
- Quintessential programmer’s tool.

Ex. Fragile X syndrome is a common cause of mental retardation.
- Human genome contains triplet repeats of CGG or AGG, starting with CGG and ending with CTG.
- Number of repeats is variable, and correlated with syndrome.
- Use regular expression to specify pattern: CGG (CGG|AGG) +CTG.

Regular Expressions: Basic Operations

Regular expression. Notation to specify a set of strings.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Regular Expression</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenation</td>
<td>aabaab</td>
<td>aabaab</td>
<td>every other string</td>
</tr>
<tr>
<td>Wildcard</td>
<td>.u.u.u.</td>
<td>cumulus</td>
<td>succubus tumultuous</td>
</tr>
<tr>
<td>Union</td>
<td>aa</td>
<td>baab</td>
<td>aa baab</td>
</tr>
<tr>
<td>Closure</td>
<td>ab*a</td>
<td>aa abba</td>
<td>ab ababa</td>
</tr>
<tr>
<td>Parentheses</td>
<td>a(a</td>
<td>b)aab</td>
<td>aaaab</td>
</tr>
<tr>
<td></td>
<td>(ab)*a</td>
<td>a ababababa</td>
<td>aa abba</td>
</tr>
</tbody>
</table>
Regular Expressions: Examples

Regular expression. Notation is surprisingly expressive.

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>.<em>spb.</em></td>
<td>raspberry, crispsbread</td>
<td>subspace, subspecies</td>
</tr>
<tr>
<td>a*</td>
<td>(a<em>ba</em>ba*)*</td>
<td>multiple of three b's</td>
</tr>
<tr>
<td>.[^0-9]....</td>
<td>fifth to last digit is 0</td>
<td>100234, 98701234</td>
</tr>
<tr>
<td>gcg{cg</td>
<td>gg}ctg</td>
<td>gcgcctg, gcgccgctg</td>
</tr>
<tr>
<td>fragile X syndrome indicator</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generalized Regular Expressions

- Additional operations typically added for convenience.
- Ex: \([a-e]+\) is shorthand for \((a|b|c|d|e)(a|b|c|d|e)^*\).

<table>
<thead>
<tr>
<th>Operation</th>
<th>Regular Expression</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>One or more</td>
<td>a(bc)+de</td>
<td>abode, abcdode</td>
<td>ade, bcde</td>
</tr>
<tr>
<td>Character classes</td>
<td>[A-Za-z][a-z]*</td>
<td>word, Capitalized</td>
<td>camelCase, 4illegal</td>
</tr>
<tr>
<td>Exactly k</td>
<td>[0-9][0-9]*-[0-9][4]</td>
<td>08540-1221, 19072-5541</td>
<td>11111111, 166-54-111</td>
</tr>
<tr>
<td>Negations</td>
<td>[^aeiou]{}</td>
<td>rhythm, decade</td>
<td></td>
</tr>
</tbody>
</table>

Regular Expressions in Java

Validity checking. Is input in the set described by the re?

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

Generalized Regular Expressions in Other Languages

- Broadly applicable programmer’s tool.
  - Many languages support extended regular expressions.
  - Built into grep, awk, emacs, Perl, PHP, Python, JavaScript.

**PERL.** Practical Extraction and Report Language.

- `grep NEELINE /*.java` print all lines containing NEELINE which occurs in any file with a .java extension
- `egrep '^[qwertuyiop][zxcvbnm]*$' dict.txt` egrep '.............'
- `perl -p -l -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-Za-z][a-z]/' dict.txt` do for each line
Regular Expression Caveats

Writing a RE is like writing a program.
- Need to learn syntax.
- Can be easier to write than read.

“Sometimes you have a programming problem and it seems like the best solution is to use regular expressions; now you have two problems.”

Engineering Grep

Generalized regular expression print.
- First implemented in 1973 by Ken Thompson for text-to-speech.
- Quintessential programmer’s tool.

Approach to develop grep algorithm.
- Define class of abstract machines.
- Write simulator for machine.
- Write translator from REs to machines.

Example of essential paradigm in computer science.
- Build intermediate abstractions.
- Pick the right ones!
- Solve important practical problem.

Perl RE for Valid RFC822 Email Addresses

Reference: http://www.ex-parrot.com/~rolf/Mail-RFC822-Address.html

Deterministic Finite State Automata

DFA review.

```java
int pc = 0;
while (!tape.isEmpty()) {
    boolean bit = tape.read();
    if (pc == 0) { if (bit) pc = 0; else pc = 1; }
    else if (pc == 1) { if (bit) pc = 1; else pc = 2; }
    else if (pc == 2) { if (bit) pc = 2; else pc = 0; }
}
if (pc == 0) System.out.println("accepted");
else System.out.println("rejected");
```
**Duality**

**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.

**Good news.** To match RE, build DFA and simulate DFA on input string.

**Bad news.** The DFA can be exponentially large.

**Consequence.** Need more efficient abstract machine.

---

**Nondeterministic Finite State Automata**

**NFA.**
- Finite state automata.
- May have 0, 1, or more transitions for each input symbol.
- May have ε-transitions.
- Accept if any sequence of transitions leads to accept state.

---

**Simulating an NFA**

**How to simulate an NFA?** Maintain set of all possible states that NFA could be in after reading in the first i symbols.

---

**NFA Simulation**

An NFA trace.
NFA Representation

NFA representation. Maintain several digraphs, one for each symbol in the alphabet, plus one for ε.

NFA Simulation

How to simulate an NFA?

- Maintain a set of all possible states that NFA could be in after reading in the first i symbols.
- Use Digraph adjacency and reachability ops to update.

NFA: Java Implementation

```java
public class NFA {
    private int START = 0; // start state
    private int ACCEPT = 1; // accept state
    private int N = 2; // number of states
    private String ALPHABET = "01";
    private int EPS = ALPHABET.length(); // symbols in alphabet
    private Digraph G[];

    public NFA(String re) {
        G = new Digraph(EPS + 1);
        for (int i = 0; i <= EPS; i++)
            G[i] = new Digraph();
        build(0, 1, re);
    }

    private void build(int from, int to, String re) {
    }

    public boolean simulate(Tape tape) {
        SET<Integer> pc = G[EPS].reachable(START);

        while (!tape.isEmpty()) {
            char c = tape.read();
            int i = ALPHABET.indexOf(c);
            SET<Integer> next = G[i].neighbors(pc);
            pc = G[EPS].reachable(next);
        }
        return (state == ACCEPT);
    }
}
```
NFA Simulation Running Time

**Input:** Text with \( N \) characters, NFA with \( M \) transitions.

**Running time:** \( O(MN) \)
- Bottleneck = \( 1 \) graph reachability per input character.
- Can be substantially faster in practice if few \( \epsilon \)-transitions.

**Note:** Easy to extend graph search to handle multiple sources.

**Implicit assumption:** Alphabet size is a small constant.

Extended NFA

Some extended NFAs.

Converting from an RE to an NFA: Basic Transformations

**Ex.** Create NFA for \( ab^* | a^*b \).
Converting from an RE to an NFA

**Ex.** Create NFA for \(ab^*|a^*b\).

Input.  Text with \(N\) characters, RE with \(M\) characters.

### Claim.
The number of edges in the NFA is at most \(2M\).

- Single character: consumes 1 symbol, creates 1 edge.
- Wildcard character: consumes 1 symbol, creates 2 edges.
- Concatenation: consumes 1 symbols, creates 0 edges.
- Union: consumes 1 symbol, creates 1 edges.
- Closure: consumes one symbol, creates 2 edges.

**NFA simulation.** \(O(MN)\) since NFA has \(2M\) transitions.

**NFA construction.** Ours is \(O(M^2)\) but not hard to make \(O(M)\).

Surprising bottom line. Worst case cost for grep is the same as for elementary string match!

---

NFA Construction: Java Implementation

```java
private void build(int from, int to, String re) {
    int or = re.indexOf('|');
    if (or == -1) {
        char c = re.charAt(0);
        for (int i = 0; i < EPSILON; i++)
            if (c == ALPHABET.charAt(i) | c == '.')
                G[i].addEdge(from, to);
    } else if (or == 1) {
        build(from, to, re.substring(0, or));
        build(from, to, re.substring(or + 1));
    } else if (re.charAt(1) == '*') {
        closure
            G[EPSILON].addEdge(from, N);
            build(N, N, re.substring(0, 1));
            build(N++, to, re.substring(1));
    } else {
        build(from, N, re.substring(0, 1));
        build(N++, to, re.substring(1));
    }
}
```

To complete grep implementation.

- Parentheses.
- Documentation.
- Extend the alphabet.
- Add character classes.
- Add capturing capabilities.
- Deal with meta characters.
- Extend the closure operator.
- Error checking and recovery.
- Greedy vs. reluctant matching.

Industrial Strength Grep Implementation

```java
String regexp = "(?i)<blink>(.+)<\/blink>";
String update = s.replaceAll(regexp, "$2")
remove all blink tags from web page
```
Converting from an RE to an NFA

Transformations for parsing parentheses.

Application: Harvester

Harvesting info. Print all occurrences of regexp from text file or URL.

- Harvest patterns from DNA.

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

- Harvest links from website.

```java
% java Harvester "http://(\w+\.)*http://(www.)*http://(www.)*http://(www.)*
http://www.princeton.edu
http://www.google.com
```

Application: Parsing a Data File

Parsing input files. NCBI genome file, ...

```java
LOCUS AC146046 128142 bp DNA linear STG 13–NOV-2003
DEFINITION Ornithorhynchus anatinus clone CLB-39B8, ACCESION AC146046
KEYWORDS STG, STG_PHASE2, STG_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
1 tgcattctc ttgctttctc tcgtcttctc tcgtcttctc tcgtcttctc tcgtcttctc tcgtcttctc
21 gttgcttctc ttgctttctc tcgtcttctc tcgtcttctc tcgtcttctc tcgtcttctc tcgtcttctc
12R101 ggcacgtcg ccacgtcg ccacgtcg ccacgtcg ccacgtcg ccacgtcg ccacgtcg ccacgtcg
```

```java
String regexp = "[\d]*[0-9]+([actg]*).*";
Pattern pattern = Pattern.compile(regexp);
In in = new In(filename);
while (!in.isEmpty()) {
    String line = in.readLine();
    Matcher matcher = pattern.matcher(line);
    if (matcher.find()) {
        String s = matcher.group().replaceAll(",", "");
        // do something with s
        replace this RE with this string
    }
}
```

Matcher. Simulate the NFA.

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

```java
public class Harvester {
    public static void main(String[] args) {
        String regexp = args[0];
        In in = new In(args[1]);
        String text = in.readAll();
        Pattern pattern = Pattern.compile(regexp);
        Matcher matcher = pattern.matcher(text);
        while (matcher.find()) {
            System.out.println("find next subsequence of input that matches the pattern");
            the input subsequence matched by most recent call to find
        }
    }
}
```
Algorithmic Complexity Attacks

Warning. Most implementations do not guarantee performance!
```
grep, Java, Perl
```

| java Validate "[a-z]"  | substitute                        | 1.6 seconds |
| java Validate "[a-z]"  | substitute                        | 3.7 seconds |
| java Validate "[a-z]"  | substitute                        | 9.7 seconds |
| java Validate "[a-z]"  | substitute                        | 23.2 seconds |
| java Validate "[a-z]"  | substitute                        | 62.2 seconds |
| java Validate "[a-z]"  | substitute                        | 161.6 seconds |

SpamAssassin regular expression.
```
java RE "^[a-z]([a-z]+\{[a-z]\}\+[a-z]+)" spammer@x.................
```

- Takes exponential time.
- Spammer can use a pathological email addresses to DOS a mail server.

Not-So-Regular Expressions

Back-references.
- \1 notation matches sub-expression that was matched earlier.
- Supported by most RE implementations.
```
java Harvester "\([^\]*\)\1" dictionary.txt
beriberi
berberi
couscus
cousus
```

Some non-regular languages.
- All strings of the form $ww$ for some string w: beriberi,
- All bitstring with an equal number of Os and Is: 01110100,
- All Watson-Crick complemented palindromes: atttcggaaat.

Remark. Pattern matching with back-references is intractable.

Context

Abstract machines, languages, and nondeterminism.
- Basis of the theory of computation.
- Intensively studied since the 1930s.

Compiler. A program that translates from one language to another.
- grep  $RE \Rightarrow NFA.$
- javac $Java\text{ language} \Rightarrow Java$ byte code.

<table>
<thead>
<tr>
<th>Abstraction</th>
<th>RE</th>
<th>Java program</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>parser</td>
<td>check if legal RE</td>
<td>check if legal Java program</td>
</tr>
<tr>
<td>compiler</td>
<td>output NFA</td>
<td>output Java byte code</td>
</tr>
<tr>
<td>simulator</td>
<td>use NFA to find match</td>
<td>execute byte code on JVM</td>
</tr>
</tbody>
</table>

Summary

Programmer.
- REs are a powerful pattern matching tool.
- Implement regular expressions with NFAs.

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

You. Practical application of core CS principles.