Pattern Matching and Grep

Grep
Regular expressions
Nondeterministic finite state automata


Pattern Matching

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 GCG and ending with CTG.
- Number of repeats is variable, and correlated with syndrome.
- Use regular expression to specify pattern: GCG(CGG|AGG)*CTG.

More Pattern Matching Applications

Test if a string matches some pattern.
- Process natural language.
- Scan for virus signatures.
- Search for information using Google.
- Access information in digital libraries.
- Find Java file containing certain string.
- Retrieve information from Lexis/Nexis.
- Search-and-replace in a word processors.
- Filter text (SpamAssassin, NetNanny, Carnivore, AdAware).
- 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 some file format.
- Automatically create Java documentation from Javadoc comments.

Regular Expressions: Basic Operations

Regular expression.
- Compact notation to specify a set of strings.
- Core operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>RE</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>.abba..</td>
<td>babbaaa</td>
<td>abba abbabb</td>
</tr>
<tr>
<td>Union</td>
<td>aa</td>
<td>baab</td>
<td>every other string</td>
</tr>
<tr>
<td>Closure</td>
<td>ab*a</td>
<td>aa</td>
<td>ababa ababa ababa</td>
</tr>
<tr>
<td>Grouping</td>
<td>a(a</td>
<td>b)aab</td>
<td>aaaab abaab</td>
</tr>
<tr>
<td></td>
<td>(ab)*a</td>
<td>a</td>
<td>ababa aa</td>
</tr>
</tbody>
</table>
Regular Expressions: Examples

Regular expression examples.
- Notation is surprisingly expressive.

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>a*</td>
<td>e</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>bbb</td>
<td>bb</td>
</tr>
<tr>
<td></td>
<td>abbaababb</td>
<td>baabbbaa</td>
</tr>
<tr>
<td>a</td>
<td>a.*a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>aba</td>
<td>ab</td>
</tr>
<tr>
<td></td>
<td>abbaabba</td>
<td>ba</td>
</tr>
<tr>
<td>.<em>abba.</em></td>
<td>abba</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>bbabbabb</td>
<td>ab</td>
</tr>
<tr>
<td></td>
<td>abbaabba</td>
<td>ba</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>a*</td>
<td>(a<em>ba</em>ba*)*</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>abbaababb</td>
</tr>
<tr>
<td>a</td>
<td>a.*a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>aba</td>
<td>ab</td>
</tr>
<tr>
<td></td>
<td>abbaabba</td>
<td>ba</td>
</tr>
<tr>
<td>.<em>abba.</em></td>
<td>abba</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>bbabbabb</td>
<td>ab</td>
</tr>
<tr>
<td></td>
<td>abbaabba</td>
<td>ba</td>
</tr>
</tbody>
</table>

Using Regular Expressions

Additional operations typically added for convenience.
- Ex: [a-z]+ 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>Any single character</td>
<td>..oo..oo.</td>
<td>spoonfood</td>
<td>choochoo</td>
</tr>
<tr>
<td>One or more</td>
<td>a(bc)+de</td>
<td>abcbede</td>
<td>ade</td>
</tr>
<tr>
<td>Character classes</td>
<td>[a-z]+</td>
<td>decade</td>
<td>Upper45</td>
</tr>
<tr>
<td>Exactly N times</td>
<td>[a-z]{6}</td>
<td>decade</td>
<td>ade</td>
</tr>
<tr>
<td>Negations</td>
<td>[^aeiou]{6}</td>
<td>rhythm</td>
<td>decade</td>
</tr>
</tbody>
</table>

Validity checking. Is text in the set described by the pattern?

```
public class Validator {
    public static void main(String[] args) {
        String pattern = args[0];
        String text    = args[1];
        System.out.println(text.matches(pattern));
    }
}
```

% java Validator "0*10*10*" 010000011
true

% java Validator "^[A-Za-z]+[A-Za-z0-9]*" ident123
true

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

% java Validator "[0-9]{3}-[0-9]{2}-[0-9]{4}" 166-11-5655
true

Regular Expressions in Java

Validity checking. Is text in the set described by the pattern?

```
public class Validator {
    public static void main(String[] args) {
        String pattern = args[0];
        String text    = args[1];
        System.out.println(text.matches(pattern));
    }
}
```

```
% java Validator "0*10*10*" 010000011
true

% java Validator "^[A-Za-z]+[A-Za-z0-9]*" ident123
true

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

% java Validator "[0-9]{3}-[0-9]{2}-[0-9]{4}" 166-11-5655
true
```

Using Regular Expressions

Additional operations typically added for convenience.
- Ex: [a-z]+ 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>Any single character</td>
<td>..oo..oo.</td>
<td>spoonfood</td>
<td>choochoo</td>
</tr>
<tr>
<td>One or more</td>
<td>a(bc)+de</td>
<td>abcbede</td>
<td>ade</td>
</tr>
<tr>
<td>Character classes</td>
<td>[a-z]+</td>
<td>decade</td>
<td>Upper45</td>
</tr>
<tr>
<td>Exactly N times</td>
<td>[a-z]{6}</td>
<td>decade</td>
<td>ade</td>
</tr>
<tr>
<td>Negations</td>
<td>[^aeiou]{6}</td>
<td>rhythm</td>
<td>decade</td>
</tr>
</tbody>
</table>

Regular Expressions in Java

Validity checking. Is text in the set described by the pattern?

```
public class Validator {
    public static void main(String[] args) {
        String pattern = args[0];
        String text    = args[1];
        System.out.println(text.matches(pattern));
    }
}
```

```
% java Validator "0*10*10*" 010000011
true

% java Validator "^[A-Za-z]+[A-Za-z0-9]*" ident123
true

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

% java Validator "[0-9]{3}-[0-9]{2}-[0-9]{4}" 166-11-5655
true
```

Regular Expressions in Java

Validity checking. Is text in the set described by the pattern?

```
public class Validator {
    public static void main(String[] args) {
        String pattern = args[0];
        String text    = args[1];
        System.out.println(text.matches(pattern));
    }
}
```

```
% java Validator "0*10*10*" 010000011
true

% java Validator "^[A-Za-z]+[A-Za-z0-9]*" ident123
true

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

% java Validator "[0-9]{3}-[0-9]{2}-[0-9]{4}" 166-11-5655
true
```
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.”

Perl RE for Valid RFC822 Email Addresses

Google supports * for full word wildcard and | for union.

TiVo WishList has very limited pattern matching.

Pattern Matching in Google

Pattern Matching in TiVo

Reference: page 76, Hughes DirectTV TiVo manual
**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.

---

**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 (1956):** 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 list of all possible states that NFA could be in after reading in the first i symbols.

NFA Simulation

NFA Representation

NFA representation. Maintain several graphs, one for each symbol in the alphabet, plus one for \( \varepsilon \).

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"; // RE alphabet
    private int EPS = ALPHABET.length(); // symbols in alphabet
    private Graph G[];

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

    private void build(int from, int to, String re) {
        // Build NFA graph
    }

    public boolean simulate(Tape tape) {
        // Simulate NFA
    }
}
```
NFA Simulation

How to simulate an NFA?
- Maintain list of all possible states that NFA could be in after reading in the first i symbols.
- Use graph adjacency and reachability ops to update.

```java
public boolean simulate(Tape tape) {
    List pc = G[EPS].reachable(START);
    // simulate NFA using input from tape
    while (!tape.isEmpty()) {
        char c = tape.read();
        int i = ALPHABET.indexOf(c);
        List next = G[i].neighbors(pc);
        pc = G[EPS].reachable(next);
    }
    while (!pc.isEmpty())
        if (pc.remove() == ACCEPT) return true;
    return false;
}
```

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 $\varepsilon$-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

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

NFA Construction: Java Implementation

```java
private void build(int from, int to, String re) {
    if (re.length() == 0) G[EPSILON].insert(from, to);
    else if (re.length() == 1) {
        char c = re.charAt(0);
        for (int i = 0; i < EPSILON; i++)
            if (c == ALPHABET.charAt(i) || c == '.')
                G[i].insert(from, to);
    }
    int or = re.indexOf(' | ');
    else if (or > 0) {
        build(from, to, re.substring(0, or));
        build(from, to, re.substring(or + 1));
    }
    else if (re.charAt(0) == '*') {
        G[EPSILON].insert(from, N);
        build(N, N, re.substring(1));
        build(N++, to, re.substring(2));
    }
    else {
        build(from, N, re.substring(0, 1));
        build(N++, to, re.substring(1));
    }
}
```

Grep Running Time

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 2 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!
Industrial Strength Grep Implementation

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.
- Greedily match longest possible match.

Converting from an RE to an NFA

Transformations for parsing parentheses.

```
from ( R ) to ( R ) S
from ( R* ) S to S
from ( R ) S to ( R ) S
```

grouping  closure  concatenation

Application: Harvester

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

**Pattern.** Compiles RE to an NFA.

**Matcher.** Simulates the NFA.

```
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 text = in.readAll();
    Pattern pattern = Pattern.compile(regexp);
    Matcher matcher = pattern.matcher(text);
    while (matcher.find())
      System.out.println(matcher.group());
  }
}
```

Application: Harvester

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

- Word puzzles.

```
java Harvester ".*hh.*" dictionary.txt
beachhead
highhanded
withheld
```

- Harvest email addresses for spam campaign.

```
mdudik@cs.princeton.edu
nallon@cs.princeton.edu
wayne@cs.princeton.edu
```

simple email validator
Application: Data File Parser

Parsing input files: Internet movie database, NCBI genome file, ...

LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhynchus anatinus clone CL1-39089,
ACCESSION AC146846
VERSION AC146846.2 G1:38504214
KEYWORDS NCBI: STG:0E4: HTG: DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
1 tgtatttcat ttgaccgtgc tgttttttcc cggtttttca gtacggtgtt agggagccac
61 gtgattctgt ttgttttatg ctgccgaata gctgctcgat gaatctctgc atagacagct // a comment
121 ggcggcaggga gaaatgacca gtttgtgatg acaaaatgta ggaaagctgt ttcttcataa...
128101 ggaaatgcga cccccacgct aatgtacagc ttctttagat tg

Application: Web Crawler

Crawling the Web: Find all web page reachable from site s.

Queue q = new Queue(); // queue of sites to crawl
HashSet visited = new HashSet(); // ST of visited websites
q.enqueue(s); // start crawl from site s
visited.add(s);
while (!q.isEmpty()) {
    String v = (String) q.dequeue();
    System.out.println(v);
    String input = in.readAll();
    String regexp = "http://(\w+\.)*(\w+)";
    Pattern pattern = Pattern.compile(regexp);
    Matcher matcher = pattern.matcher(input);
    while (matcher.find()) {
        String w = matcher.group();
        if (!visited.contains(w)) {
            visited.add(w);
            q.enqueue(w);
        }
    }
}

Algorithmic Complexity Attacks

Warning: most everyday implementations (Unix grep, Java, Perl) do not guarantee performance!

java Validator "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaac             1.6 seconds
java Validator "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac           3.7 seconds
java Validator "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac         9.7 seconds
java Validator "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaac      23.2 seconds
java Validator "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 161.6 seconds

SpamAssassin regular expression.

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

- Takes exponential time.
- Spammer could use a pathological return email addresses to DOS a mail server running SpamAssassin.

Not-So-Regular Expressions

Back-references.
- \1 notation matches subexpression that was matched earlier.
- Supported by most RE implementations.

java Harvester "^(.*)\1$" dictionary.txt
beriberi
couscous

Some non-regular languages.
- All strings of the form ww for some string w: couscous, beriberi.
- All bitstring with an equal number of 0s and 1s: 10, 01110100.
- All Watson-Crick complemented palindromes: atttcggaat.
  * 

Remark. Pattern matching with back-references is NP-hard.
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 language \Rightarrow Java byte code.

<table>
<thead>
<tr>
<th>Abstract Machine</th>
<th>NFA</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>Word in CFL</td>
<td>Word in CFL</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 machine executable</td>
</tr>
<tr>
<td>Simulator</td>
<td>Find match</td>
<td>Run program in hardware</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.