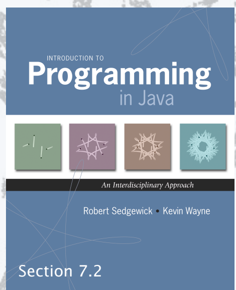


http://introcs.cs.princeton.edu

COMPUTER SCIENCE
SEDGEWICK / WAYNE

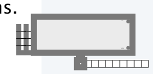
17. Introduction to Theoretical CS

Introduction to theoretical computer science

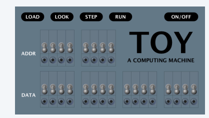
Fundamental questions
- What can a computer do?
- What can a computer do with limited resources?

General approach
- Don't talk about specific machines or problems.
- Consider minimal abstract machines.
- Consider general classes of problems.







Surprising outcome. Sweeping and relevant statements about *all* computers.

Why study theory?

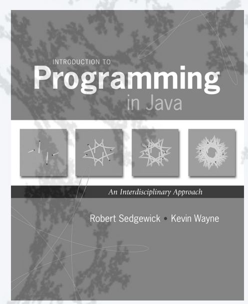
In theory...
- Deeper understanding of computation.
- Foundation of all modern computers.
- Pure science.
- Philosophical implications.

In practice...
- Web search: theory of pattern matching.
- Sequential circuits: theory of finite state automata.
- Compilers: theory of context free grammars.
- Cryptography: theory of computational complexity.
- Data compression: theory of information.
- ...

"In theory there is no difference between theory and practice. In practice there is."

— Yogi Berra

COMPUTER SCIENCE
SEDGEWICK / WAYNE



http://introcs.cs.princeton.edu

17. Introduction to Theoreticaal CS

- **Regular expressions**
- DFAs
- Applications
- Limitations

Pattern matching

Pattern matching problem. Is a given string a member of a given set of strings?

Example 1 (from genomics)

A nucleic acid is represented by one of the letters a, c, t, or g.

A genome is a string of nucleic acids.

A Fragile X Syndrome pattern is a genome having an occurrence of gcg, followed by any number of cgg or agg triplets, followed by ctg.

Note. The number of triplets correlates with Fragile X Syndrome, a common cause of mental retardation.

Q. Does this genome contain a such a pattern?

gcggcgtgtgtgcgagagagtgggtttaaagctggcgcggaggcggctggcgcggaggctg

A. Yes. gcgcggaggcggctg

gcg start mark

sequence of cgg and agg triplets

ctg end mark

Pattern matching

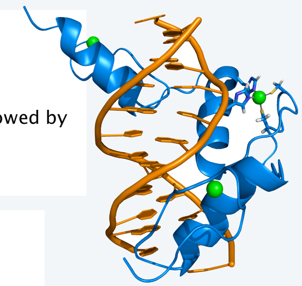
Example 2 (from computational biochemistry)

An amino acid is represented by one of the characters CAVLIMCRKHDENQSTYFWP.

A protein is a string of amino acids.

A C_2H_2-type zinc finger domain signature is

- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by
- H.



Q. Is this protein in the C_2H_2-type zinc finger domain?

CAASCGGPYACGGWAGYHAGWH

A. Yes.

C 3 C 3 Y 8 H 3 H

Pattern matching

Example 3 (from commercial computing)

An e-mail address is

- A sequence of letters, followed by
- the character "@", followed by
- the character "." , followed by a sequence of letters, followed by
- [any number of occurences of the previous pattern]
- "edu" or "com" (others omitted for brevity).

Q. Which of the following are e-mail addresses?

	A.
rs@cs.princeton.edu	✓
not an e-mail address	✗
wayne@cs.princeton.edu	✓
eve@airport	✗
rs123@princeton.edu	✗

Ooops, need to fix description → rs123@princeton.edu

Challenge. Develop a precise description of the set of strings that are legal e-mail addresses.

Regular expressions

A regular expression (RE) is a notation for specifying sets of strings.

An RE is

- A sequence of letters or "."
- The *union* of two REs
- The *closure* of an RE
- May be delimited by ().

operation	*example RE*	*matches (IN the set)*	*does not match (NOT in the set)*
concatenation	aabaab	aabaab	*every other string*
wildcard	.u.u.u.	cumulus jugulum	succubus tumultuous
union	aa | baab	aa baab	*every other string*
closure	ab*a	aa abbba	ab ababa
parentheses	a(a|b)aab	aaaab abaab	*every other string*
	(ab)*a	a ababababa	aa abbba

More examples of regular expressions

The notation is surprisingly expressive.

regular expression	*matches*	*does not match*
`.*spb.*` *contains the trigraph* spb	raspberry crispbread	subspace subspecies
`a* | (a*ba*ba*ba*)*` *multiple of three* b's	bbb aaa bbbaababbaa	b bb baabbbaa
`.*0....` *fifth to last digit is* 0	1000234 98701234	111111111 403982772
`gcg(cgg|agg)*ctg` *fragile X syndrome pattern*	gcgctg gcgcggctg gcgcggaggctg	gcgcgg cggcggcggctg gcgcaggctg

Generalized regular expressions

Additional operations futher extend the utility of REs.

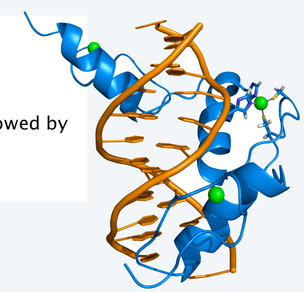
operation	*example RE*	*matches*	*does not match*
one or more	`a(bc)+de`	abcde abcbcde	ade bcde
character class	`[A-Za-z][a-z]*`	lowercase Capitalized	camelCase 4illegal
exactly k	`[0-9]{5}-[0-9]{4}`	08540-1321 19072-5541	111111111 166-54-1111
negation	`[^aeiou]{6}`	rhythm	decade
white space	`\s`	*any whitespace char (space, tab, newline...)*	*every other character*

Note. These operations are all *shorthand*.
They are very useful but not essential.

RE: `(a|b|c|d|e)(a|b|c|d|e)*`
shorthand: `(a-e)+`

Example of describing a pattern with a generalized RE

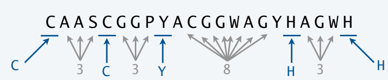
A C_2H_2-type zinc finger domain signature is
- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by
- H.



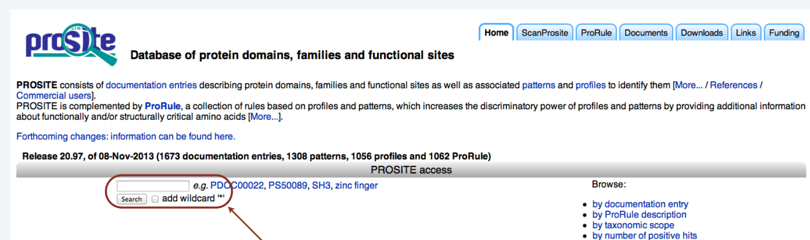
Q. Give a generalized RE for all such signatures.

A. `C.{2,4}C...[LIVMFYWCX].{8}H.{3,5}H`

"Wildcard" matches any of the letters
CAVLIMCRKHDENQSTYFWP



Example of a real-world RE application: PROSITE



Type an RE here

Another example of describing a pattern with a generalized RE

An e-mail address is
- A sequence of letters, followed by
- the character "@", followed by
- the character "." , followed by a sequence of letters, followed by
- [any number of occurences of the previous pattern]
- "edu" or "com" (others omitted for brevity).

Q. Give a generalized RE for e-mail addresses.

A. `[a-z]+@([a-z]+\.)+(edu|com)`

Exercise. Extend to handle `rs123@princeton.edu`, more suffixes such as `.org`, and any other extensions you can think of.

Next. Determining whether a given string matches a given RE.

TEQ 1 on REs

Q. Which of the following strings match the RE `a*bb(ab|ba)*` ?

(match the RE ↑ is in the set it describes)

1. abb
2. aaba
3. abba
4. bbbaab
5. cbb
6. bbababbab

TEQ 1 on REs

Q. Which of the following strings match the RE `a*bb(ab|ba)*` ?

(match the RE ↑ is in the set it describes)

	A.	
1. abb	✓	
2. aaba	✗	Must have bb.
3. abba	✗	After bb, no way to match a at the end.
4. bbbaab	✓	
5. cbb	✗	No c.
6. bbababbab	✗	Need even number of characters after bb.

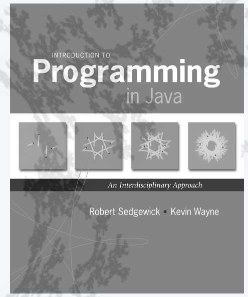
TEQ 2 on REs

Q. Give an RE for *genes*
- Characters are a, c, `t` or g.
- Starts with `atg` (a *start codon*).
- Length is a multiple of 3.
- Ends with `tag`, `taa`, or `ttg` (a *stop codon*).

TEQ 2 on REs

Q. Give an RE for *genes*
- Characters are a, c, t or g.
- Starts with atg (a *start codon*).
- Length is a multiple of 3.
- Ends with tag, taa, or ttg (a *stop codon*).

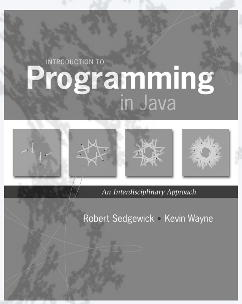
A. atg((a|c|t|g)(a|c|t|g)(a|c|t|g))*(tag|taa|ttg)



http://introcs.cs.princeton.edu

17. Introduction to Theoreticaal CS

- **Regular expressions**
- DFAs
- Applications
- Limitations



http://introcs.cs.princeton.edu

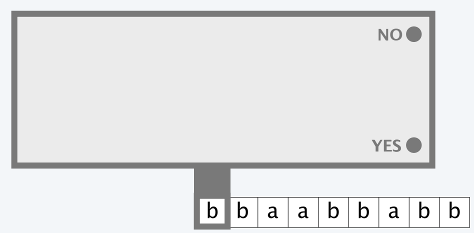
17. Introduction to Theoreticaal CS

- Regular expressions
- **DFAs**
- Applications
- Limitations

Deterministic finite state automata (DFA)

A DFA is an abstract machine that solves a pattern matching problem.
- A string is specified on an input tape (no limit on its length).
- The DFA reads each character on input tape once, moving left to right.
- The DFA lights "YES" if it *recognizes* the string, "NO" otherwise.

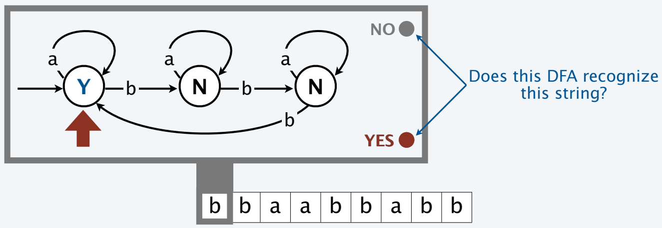
Each DFA defines a *set* of strings (all the strings that it recognizes).



Deterministic finite state automata details and example

A DFA is an abstract machine with a finite number *states*, each labeled Y or N, and *transitions* between states, each labelled with a symbol. One state is the *start* state.

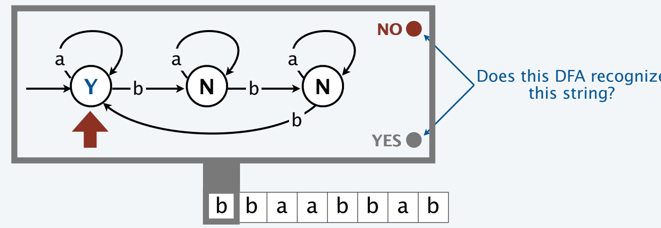
- Begin in the *start* state (denoted by an arrow from nowhere).
- Read an input symbol and move to the indicated state.
- Repeat until the last input symbol has been read.
- Turn on the "YES" or "NO" light according to the label on the current state.



Deterministic finite state automata details and example

A DFA is an abstract machine with a finite number *states*, each labeled Y or N and *transitions* between states, each labelled with a symbol. One state is the *start* state.

- Begin in the *start* state.
- Read an input symbol and move to the indicated state.
- Repeat until the last input symbol has been read.
- Turn on the "YES" or "NO" light according to the label on the current state.

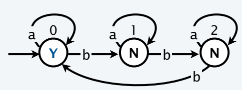


Simulating the operation of a DFA

```
public class DFA
{
   private int state;
   private int start;
   private String[] action;
   private ST<Character, Integer>[] next;
   public DFA(In in)
   {  /* Fill in data structures */  }
   public String simulate(String input)
   {
      state = start;
      for (int i = 0; i < input.length(); i++)
         state = next[state].get(input.charAt(i));
      return action[state];
   }
   public static void main(String[] args)
   {
      DFA dfa = new DFA(new In(args[0]));
      while (!StdIn.isEmpty())
      {
         input = StdIn.readString();
         StdOut.println(dfa.simulate(input));
      }
   }
}
```

symbol table to map chars a, b, ... to next state 0, 1, ...

action[]		next[]		
			a	b
0	Yes	0	0	1
1	No	1	1	2
2	No	2	2	0

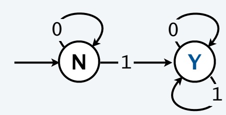


states → alphabet → start state →

```
% more b3.txt
3
ab
0
Yes 0 1
No  1 2
No  2 0
% java DFA b3.txt
bababa
Yes
bb
No
abbabbababbbabaaa
Yes
abbabbababbba
No
```


TEQ 1 on DFAs

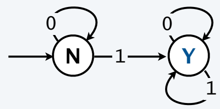
Q. Which of the following strings does this DFA accept?



1. Bitstrings that end in 1
2. Bitstrings with an equal number of occurrences of 01 and 10
3. Bitstrings with more 1s than 0s
4. Bitstrings with an equal number of occurrences of 0 and 1
5. Bitstrings with at least one 1

TEQ 1 on DFAs

Q. Which of the following strings does this DFA accept?

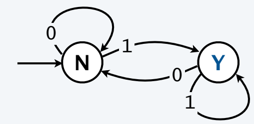


A.

1. Bitstrings that end in 1	✗	0110
2. Bitstrings with an equal number of occurrences of 01 and 10	✗	011010
3. Bitstrings with more 1s than 0s	✗	0001
4. Bitstrings with an equal number of occurrences of 0 and 1	✗	0001
5. Bitstrings with at least one 1	✓	

TEQ 2 on DFAs

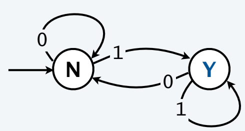
Q. Which of the following strings does this DFA accept?



1. Bitstrings with at least one 1
2. Bitstrings with an equal number of occurrences of 01 and 10
3. Bitstrings with more 1s than 0s
4. Bitstrings with an equal number of occurrences of 0 and 1
5. Bitstrings that end in 1

TEQ 2 on DFAs

Q. Which of the following strings does this DFA accept?



A.

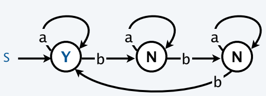
1. Bitstrings with at least one 1	✗	0100
2. Bitstrings with an equal number of occurrences of 01 and 10	✗	0110
3. Bitstrings with more 1s than 0s	✗	010000001
4. Bitstrings with an equal number of occurrences of 0 and 1	✗	01111111
5. Bitstrings that end in 1	✓	

Kleene's theorem

Two ways to define a set of strings
- Regular expressions (REs).
- Deterministic finite automata (DFAs).

S = the set of ab strings where the number of occurrences of b is a multiple of 3

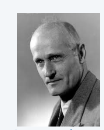
DFA for S



RE for S a* | (a*ba*ba*ba*)*

Remarkable fact. DFAs and REs are *equivalent*.

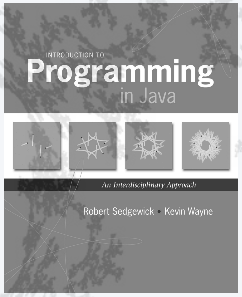
Equivalence theorem (Kleene)
Given any RE, there exists a DFA that accepts the same set of strings.
Given any DFA, there exists an RE that matches the same set of strings.



Steven Kleene
1909–1994

Consequence: A way to solve the RE pattern matching problem
- Build the DFA corresponding to the given RE.
- Simulate the operation of the DFA.

COMPUTER SCIENCE
S E D G E W I C K / W A Y N E

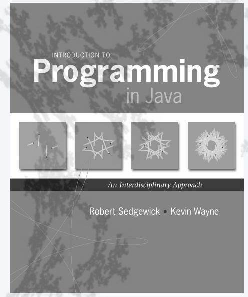


http://introcs.cs.princeton.edu

17. Introduction to Theoreticaal CS

- Regular expressions
- **DFAs**
- Applications
- Limitations

COMPUTER SCIENCE
S E D G E W I C K / W A Y N E



http://introcs.cs.princeton.edu

17. Introduction to Theoreticaal CS

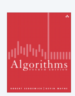
- Regular expressions
- DFAs
- **Applications**
- Limitations

GREP: a solution to the RE pattern matching problem

An algorithm for the RE pattern matching problem?
- Build the DFA corresponding to the given RE.
- Simulate the operation of the DFA.

Practical difficulty: The DFA might have *exponentially* many states.

A more efficient algorithm: use Nondeterministic Finite Automata (NFA)
- Build the NFA corresponding to the given RE.
- Simulate the operation of the NFA.



Interested in details? Take a course in algorithms.

"GREP" (Generalized Regular Expression Pattern matcher).
- Developed by Ken Thompson, who designed and implemented Unix.
- Indispensible programming tool for decades.
- Found in most development environments, including Java.

Ken Thompson
1983 Turing Award

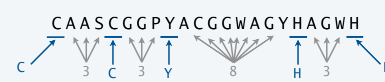
REs in Java

Java's `String` class implements GREP.

public class String	
...	
boolean matches(String re)	*does this string match the given RE?*
...	

```
String re = "C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H";
String zincFinger = "CAASCGGPYACGGAAGYHAGAH";
boolean test = zincFinger.matches(re);
```

true!



Java RE client example: Validation

```
public class Validate
{
   public static void main(String[] args)
   {
      String re = args[0];
      while (!StdIn.isEmpty())
      {
         String input = StdIn.readString();
         StdOut.println(input.matches(re));
      }
   }
}
```

Does a given string match a given RE?

- Take RE from command line.
- Take strings from StdIn.

need quotes to "escape" the shell

```
% java Validate "C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H"
CAASCGGPYACGGAAGYHAGAH
true
CAASCGGPYACGGAAGYHGAH
false

% java Validate "[$_A-Za-z][$_A-Za-z0-9]*"
ident123
true
123ident
false

% java Validate "[a-z]+@([a-z]+\.)+(edu|com)"
wayne@cs.princeton.edu
true
eve@airport
false
```

C_2H_2 type zinc finger domain

legal Java identifier

valid email address (simplified)

Applications

- Scientific research.
- Compilers and interpreters.
- Internet commerce.
- ...

Beyond matching

Java's `String` class contains other useful RE-related methods.

- RE search and replace
- RE delimited parsing

public class String	
...	
String replaceAll(String re, String to)	*replace all occurrences of substrings matching RE with* `to`
String[] split(String re)	*split the string around matches of the given RE*
...	

Tricky notation (typical in string processing): \ signals "special character" so "\\" means "\" and "\\s" means "\s"

Examples using the RE "\\s+" (matches one or more whitespace characters).

Replace each sequence of at least one whitespace character with a single space.

```
String s = StdIn.readAll();
s = s.replaceAll("\\s+", " ");
```

Create an array of the words in StdIn (basis for `StdIn.readAllStrings()` method)

```
String s = StdIn.readAll();
String[] words = s.split("\\s+");
```


Way beyond matching

Java's `Pattern` and `Matcher` classes give fine control over the GREP implementation.

public class Pattern	
...	
static Pattern compile(String re)	*parse the* `re` *to construct a* `Pattern`
Matcher matcher(String input)	*create a* `Matcher` *that can find substrings matching the pattern in the given input string*
...	

Why not a constructor? Good question.

public class Matcher	
...	
boolean find()	*set internal variable* `match` *to the next substring that matches the RE in the input. If none, return* `false`*, else return* `true`
String group()	*return* `match`
String group(int k)	*return the kth group (identified by parens within RE) in* `match`
...	

[A sophisticated interface designed for pros, but very useful for everyone.]

Java pattern matcher client example: Harvester

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

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

   }
}
```

Harvest information from input stream

- Take RE from command line.
- Take input from file or web page.
- Print all substrings matching RE.

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

% java Harvester "[a-z]+@([a-z]+\.)+(edu|com)" http://www.cs.princeton.edu/people/faculty
...
rs@cs.princeton.edu
...
wayne@cs.princeton.edu
...
```

harvest patterns from DNA

harvest email addresses from web for spam campaign.

Java pattern matcher real-world example: Parsing a data file

A typical situation

- An institution publishes data on the web to be shared by all.
- The data is published in human-readable form.
- You want to strip out everything but the raw data in order to process it.

Example: National Center for Biotechnology Information genome data

```
LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhynchus anatinus clone CLM1-393H9,
ACCESSION AC146846
VERSION AC146846.2 GI:38304214
KEYWORDS HTG; HTGS_PHASE2; HTGS_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
      1 tgtatttcat ttgaccgtgc tgttttttcc cggtttttca gtacggtgtt agggagccac
     61 gtgattctgt ttgttttatg ctgccgaata gctgctcgat gaatctctgc atagacagct // a comment
    121 gccgcaggga gaaatgacca gtttgtgatg acaaaatgta ggaaagctgt ttcttcataa
    ...
 128101 ggaaatgcga cccccacgct aatgtacagc ttctttagat tg
//
```

header information

spaces

line numbers

don't want this "a"

Java pattern matcher real-world example: Parsing a data file

Key challenge: Develop an appropriate RE.

```
[ ]*[0-9]+([actg ]*).*
```

Parens identify a *group* that includes only the data (a, c, t, g, or spaces).

Slight glitch: Need to remove spaces afterwards.

Extract data after spaces followed by a line number.

Ignore everything else

```
LOCUS AC146846 128142 bp DNA linear HTG 13-NOV-2003
DEFINITION Ornithorhynchus anatinus clone CLM1-393H9,
ACCESSION AC146846
VERSION AC146846.2 GI:38304214
KEYWORDS HTG; HTGS_PHASE2; HTGS_DRAFT.
SOURCE Ornithorhynchus anatinus (platypus)
ORIGIN
      1 tgtatttcat ttgaccgtgc tgttttttcc cggtttttca gtacggtgtt agggagccac
     61 gtgattctgt ttgttttatg ctgccgaata gctgctcgat gaatctctgc atagacagct // a comment
    121 gccgcaggga gaaatgacca gtttgtgatg acaaaatgta ggaaagctgt ttcttcataa
    ...
 128101 ggaaatgcga cccccacgct aatgtacagc ttctttagat tg
//
```

1st match

first (only) group in 2nd match

Java pattern matcher real-world example: Parsing a data file

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

public class ParseNCBI
{
   public static void main(String[] args)
   {
      String re = "[ ]*[0-9]+([actg ]*).*";
      Pattern pattern = Pattern.compile(re);
      In in = new In(args[0]);
      while (in.hasNext Line())
      {
         String line = in.readLine();
         Matcher matcher = pattern.matcher(line);
         if (matcher.find())
            StdOut.print(matcher.group(1).replaceAll(" ", ""));
      }
      StdOut.println();
   }
}
```

remove the spaces

```
% java ParseNCBI platypus.txt
tgtatttcatttgaccgtgctgttttttcccgg
tttttcagtacggtgttagggagccacgtgatt
ctgtttgttttatgctgccgaatagctgctcga
tgaatctctgcatagacagctgccgcagggaga
aatgaccagtttgtgatgacaaaatgtaggaaa
gctgtttcttcataa...
```

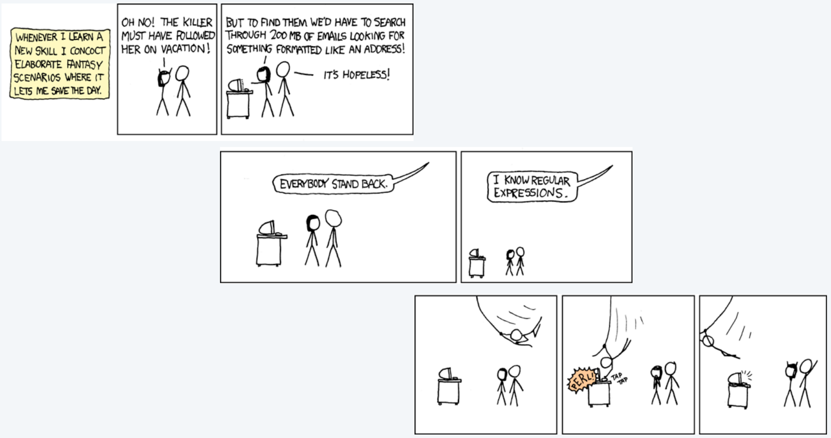

Applications of REs

Pattern matching and beyond.

- Compile a Java program.
- Scan for virus signatures.
- Crawl and index the Web.
- Process natural language.
- Access information in digital libraries.
- Search-and-replace in a word processors.
- Process NCBI and other scientific data files.
- Filter text (spam, NetNanny, ads, Carnivore, malware).
- Validate data-entry fields (dates, email, URL, credit card).
- Search for markers in human genome using PROSITE patterns.
- Automatically create Java documentation from Javadoc comments.

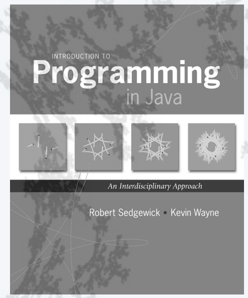
GREP and related facilities are built in to Java, Unix shell, PERL, Python ...

virtually *every* computing environment



http://xkcd.com/208/

COMPUTER SCIENCE
SEDGEWICK / WAYNE

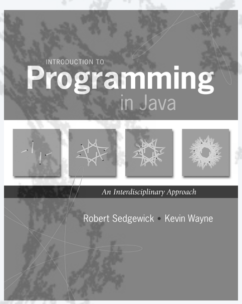


http://introcs.cs.princeton.edu

17. Introduction to Theoreticaal CS

- Regular expressions
- DFAs
- **Applications**
- Limitations

COMPUTER SCIENCE
SEDGEWICK / WAYNE



http://introcs.cs.princeton.edu

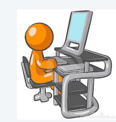
17. Introduction to Theoreticaal CS

- Regular expressions
- DFAs
- Applications
- **Limitations**

Summary

Programmers

- Regular expressions are a powerful pattern matching tool.
- Equivalent DFA/NFA paradigm facilitates implementation.
- Combination greatly facilitates real-world string data processing.

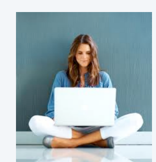


Theoreticians

- REs provide compact descriptions of sets of strings.
- DFAs are abstract machines with equivalent descriptive power.
- Are there languages and machines with more descriptive power?

You

- CS core principles provide useful tools that you can exploit now.
- REs and DFAs provide an introduction to theoretical CS.



Basic questions

Q. Are there sets of strings that cannot be described by *any* RE?
A. Yes.
- Bitstrings with equal number of 0s and 1s.
- Strings that represent legal REs.
- Decimal strings that represent prime numbers.
- DNA strings that are Watson-Crick complemented palindromes.
- ...

The *same* question, by Kleene's theorem

Q. Are there sets of strings that cannot be described by *any* DFA?
A. Yes.
- Bit strings with equal number of 0s and 1s.
- Strings that represent legal REs.
- Decimal strings that represent prime numbers.
- DNA strings that are Watson-Crick complemented palindromes.
- ...

A limit on the power of REs and DFAs

Proposition. There exists a set of strings that cannot be described by any RE or DFA.

Proof sketch. No DFA can recignize the set of bitstrings with equal number of 0s and 1s.
- *Assume that you have such a DFA*, with N states.
- It recognizes the string with $N + 1$ 0s followed by $N + 1$ 1s.
- Some state is revisited when recognizing that string.
- Delete the substring between visits.
- DFA recognizes that string, too.
- It does not have equal number of 0s and 1s.
- *Proof by contradiction*: the assumption that such a DFA exists must be false.

Ex. $N = 10$

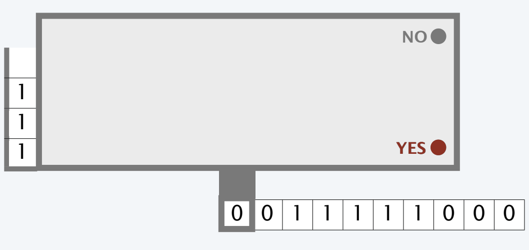
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	3	5	9	8	7	5	.	.	.												

0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	3	5	.	.	.												

Another basic question

Q. Are there abstract machines that are more powerful than DFAs? (can recognize more sets of strings)
A. Yes. A 1-stack DFA can recognize
- Bitstrings with equal number of 0s and 1s.
- Strings that represent legal REs.

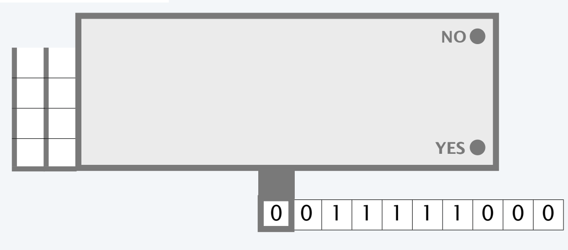
Proof. [details omitted]



Yet another basic question

Q. Are there abstract machines that are more powerful than a 1-stack DFA?
A. Yes. A 2-stack DFA can recognize
- Decimal strings that represent prime numbers.
- Strings that represent legal Java programs.
- ...

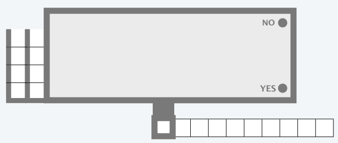
[stay tuned for next lecture]

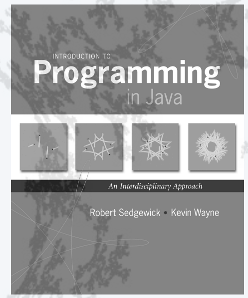


One last basic question

Q. Are there machines that are more powerful than a 2-stack DFA?
A. No! Not even a roomful of supercomputers (!!!)

[stay tuned for next lecture]

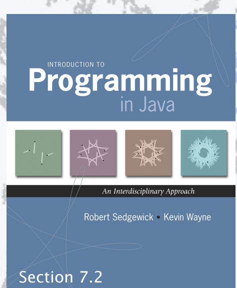




http://introcs.cs.princeton.edu

17. Introduction to Theoreticaal CS

- Regular expressions
- DFAs
- Applications
- **Limitations**



http://introcs.cs.princeton.edu

17. Introduction to Theoretical CS