Pattern Matching

- exact pattern matching
- Knuth-Morris-Pratt
- RE pattern matching
- grep

References:
Algorithms in C (2nd edition), Chapter 19 (pdf online)
`http://www.cs.princeton.edu/algs4/63long`
`http://www.cs.princeton.edu/algs4/72regular`

- **exact pattern matching**
- Knuth-Morris-Pratt
- RE pattern matching
- grep

Exact pattern matching

Goal. Find pattern of length M in a text stream of length N.

typically N >> M

pattern `n e e d l e`

text `i n a h a y s t a c k a n e e d l e i n a`

Computer forensics. Search memory or disk for signatures, e.g., all URLs or RSA keys that the user has entered.

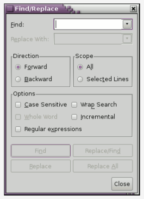
`http://citp.princeton.edu/memory`

Applications

- Parsers.
- Spam filters.
- Digital libraries.
- Screen scrapers.
- Word processors.
- Web search engines.
- Natural language processing.
- Computational molecular biology.
- FBIs Digital Collection System 3000.
- Feature detection in digitized images.
- ...

SpamAssassin

LexisNexis™ It's how you know™



DCS 3000

Spam filtering

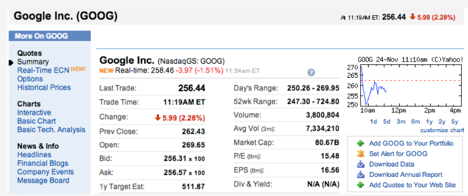
Identify patterns indicative of spam.

- PROFITS
- AMAZING
- GUARANTEE
- L0SE WE1GHT
- herbal Viagra
- There is no catch.
- LOW M0RTGAGE RATES
- This is a one-time mailing.
- This message is sent in compliance with spam regulations.
- You're getting this message because you registered with one of our marketing partners.

Screen scraping

Goal. Extract relevant data from web page.

Ex. Find string delimited by and after first occurrence of pattern **Last Trade:**.



http://finance.yahoo.com/q?s=goog

```
...
<tr>
<td class= "yfnc_tablehead1"
width= "48%">
Last Trade:
</td>
<td class= "yfnc_tabledata1">
<big><b>452.92</b></big>
</td></tr>
<td class= "yfnc_tablehead1"
width= "48%">
Trade Time:
</td>
<td class= "yfnc_tabledata1">
...
```


Exact pattern matching in Java

The method `s.indexOf(pattern, offset)` in Java's `String` library returns the index of the first occurrence of `pattern` in string `s`, starting at given `offset`.

```
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 input = in.readAll();
      int start    = input.indexOf("Last Trade:", 0);
      int from     = input.indexOf("<b>",  start);
      int to       = input.indexOf("</b>", from);
      String price = input.substring(from + 3, to);
      StdOut.println(price);
   }
}
```

```
% java StockQuote goog
256.44

% java StockQuote msft
19.68
```


Brute-force exact pattern match

Check for pattern starting at each text position.

h	a	y	n	e	e	d	s	a	n	n	e	e	d	l	e	x
n	e	e	d	l	e											
	n	e	e	d	l	e										
		n	e	e	d	l	e									
			n	e	e	d	l	e								
				n	e	e	d	l	e							
					n	e	e	d	l	e						
						n	e	e	d	l	e					
							n	e	e	d	l	e				
								n	e	e	d	l	e			
									n	e	e	d	l	e		
										n	e	e	d	l	e	

Brute-force exact pattern match: Java implementation

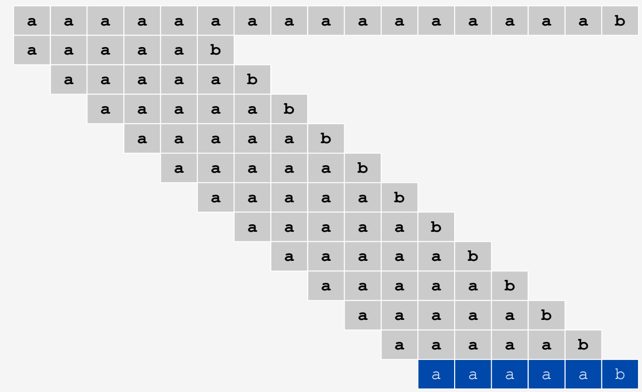
Check for pattern starting at each text position.

```
public static int search(String pattern, String text)
{
   int M = pattern.length();
   int N = text.length();

   for (int i = 0; i < N - M; i++)
   {
      int j;
      for (j = 0; j < M; j++)
         if (text.charAt(i+j) != pattern.charAt(j))
            break;
      if (j == M) return i;   ← index in text where pattern starts
   }
   return -1;   ← not found
}
```


Brute-force exact pattern match: worst case

Brute-force algorithm can be slow if text and pattern are repetitive.



Worst case. ~ MN char compares.

Algorithmic challenges in pattern matching

Brute-force is not good enough for all applications.

Theoretical challenge. Linear-time guarantee. ← fundamental algorithmic problem

Practical challenge. Avoid backup in text stream. ← often no room or time to save text

```
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 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 attack at dawn 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.
```


- exact pattern matching
- **Knuth-Morris-Pratt**
- RE pattern matching
- grep

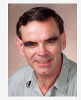
Knuth-Morris-Pratt exact pattern-matching algorithm

KMP. Classic algorithm that meets both challenges.

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

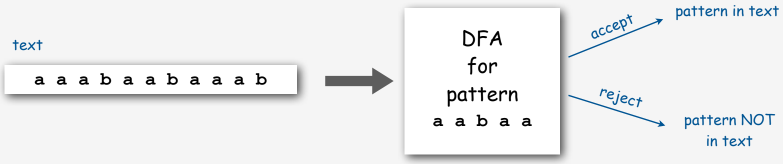

Don Knuth


Jim Morris


Vaughan Pratt

Basic plan (for binary alphabet).

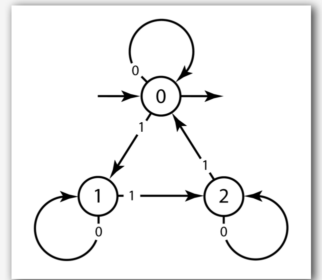
- Build DFA from pattern.
- Simulate DFA with text as input.

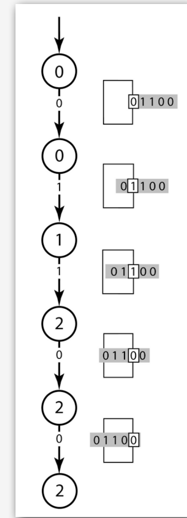


Deterministic finite-state automata

DFA review.

- Finite number of states (including start and accept).
- Exactly one transition for each input symbol.
- Accept if sequence of transitions leads to accept state.



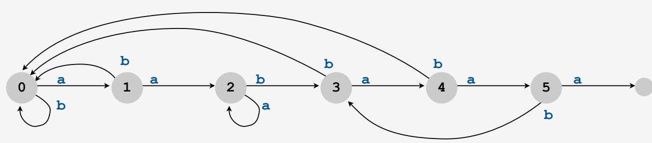


Q. Which bitstrings does this DFA accept?

Knuth-Morris-Pratt DFA example

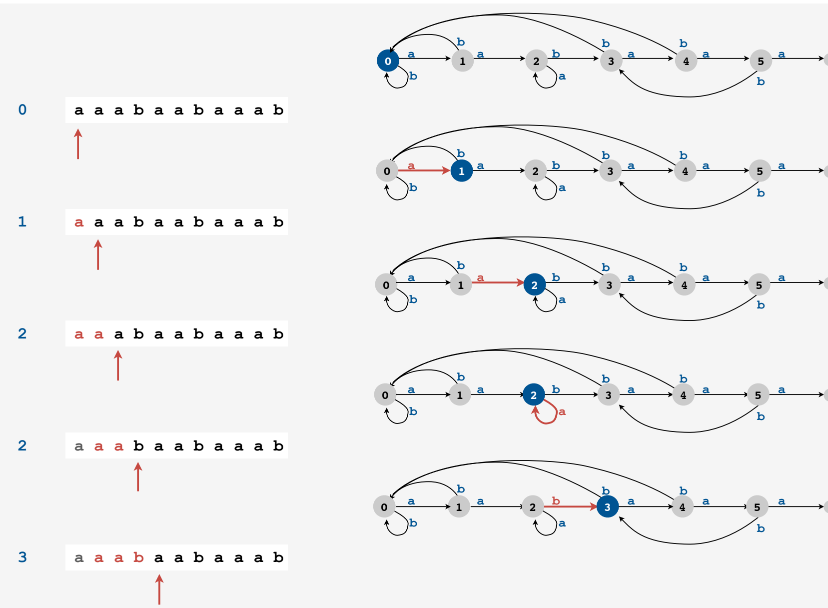
One state for each pattern character.

- Match input character: move from i to i+1.
- Mismatch: move to previous state.

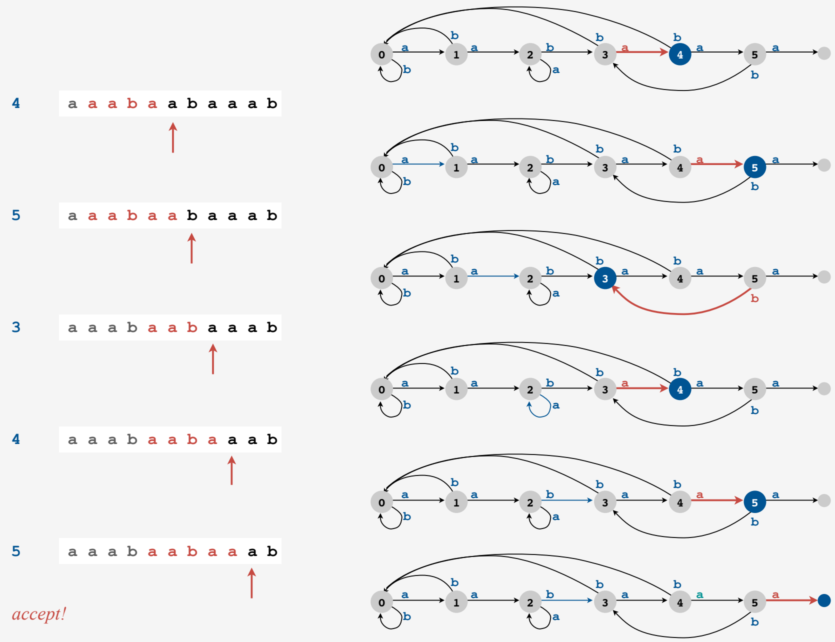


DFA for pattern **aabaaa**

Knuth-Morris-Pratt DFA simulation



Knuth-Morris-Pratt DFA simulation



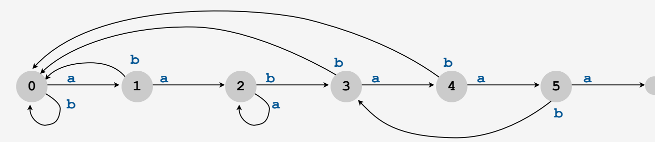
Knuth-Morris-Pratt DFA simulation

When in state i. Matches in i previous input chars (and is longest such match).

Ex. End in state 4 iff text ends in **aaba**.

Ex. End in state 2 iff text ends in **aa** (but not **aabaa** or **aabaaa**).

0	a	a	a	b	a	a	b	a	a	a	b
1	a	a	a	b	a	a	b	a	a	a	b
2	a	a	a	b	a	a	b	a	a	a	b
2	a	a	a	b	a	a	b	a	a	a	b
3	a	a	a	b	a	a	b	a	a	a	b
4	a	a	a	b	a	a	b	a	a	a	b
5	a	a	a	b	a	a	b	a	a	a	b
3	a	a	a	b	a	a	b	a	a	a	b
4	a	a	a	b	a	a	b	a	a	a	b
5	a	a	a	b	a	a	b	a	a	a	b
	a	a	a	b	a	a	b	a	a	a	b



Knuth-Morris-Pratt implementation

DFA representation. A single state-indexed array **next[]**.

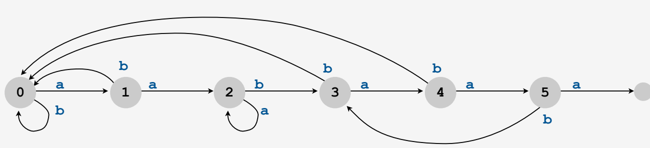
- Upon character match in state **j**, go forward to state **j+1**.
- Upon character mismatch in state **j**, go back to state **next[j]**.

	0	1	2	3	4	5
a	1	2	2	4	5	6
b	0	0	3	0	0	3

	0	1	2	3	4	5
next	0	0	2	0	0	3

← only need to store mismatches

DFA for pattern **aabaaa**



Knuth-Morris-Pratt: Java implementation

Two key differences from brute-force implementation.

- Text pointer **i** never decrements.
- Need to precompute **next[]** table (DFA) from pattern.

Simulation of KMP DFA

```
int j = 0;
for (int i = 0; i < N; i++)
{
   if (text.charAt(i) == pattern.charAt(j))
      j++;                          // char matches
   else
      j = next[j];                  // char mismatch
   if (j == M) return i - M + 1;    // found pattern
}
return -1;                          // not found
```

Knuth-Morris-Pratt: incremental DFA construction

Key idea. DFA for first `i` states contains info needed to build state `i+1`.

Ex. Given DFA for pattern `aabaaa`, to compute DFA for pattern `aabaaab`:

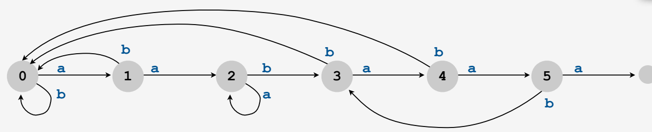
- On mismatch at 7th char, need to simulate 6-char backup.
- Previous 6 chars are known (`abaaaa` in example).
- 6-state DFA (known) determines next state!

6-char backup

0	a	b	a	a	a	a
1	a	b	a	a	a	a
0	a	b	a	a	a	a
1	a	b	a	a	a	a
2	a	b	a	a	a	a
2	a	b	a	a	a	a
2	a	b	a	a	a	a

Q. How to do efficiently?

A. Keep track of DFA state for pattern, starting at 2nd char.



DFA for pattern `aabaaa`

Knuth-Morris-Pratt DFA construction: two cases

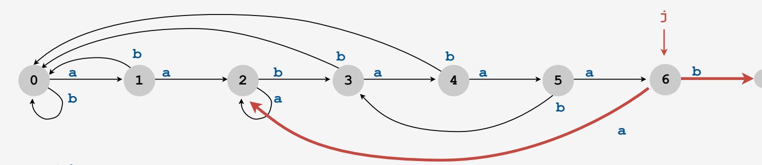
Let `x` be the next state in the simulation and `j` the next state to build.

Case 1. If `p[x]` and `p[j]` match, copy and increment.

- `next[j] = next[X];`
- `X = X + 1;`

state for ~~a~~abaaa *state for ~~a~~abaaab*

	0	1	2	3	4	5	6
p[]	a	a	b	a	a	a	b
next[]	0	0	2	0	0	3	2



DFA for pattern `aabaaab`

Knuth-Morris-Pratt DFA construction: two cases

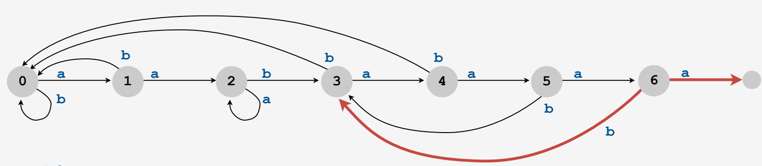
Let `x` be the next state in the simulation and `j` the next state to build.

Case 2. If `p[x]` and `p[j]` mismatch, do the opposite.

- `next[j] = X + 1;`
- `X = next[j];`

state for ~~a~~abaaa *state for ~~a~~abaaaa*

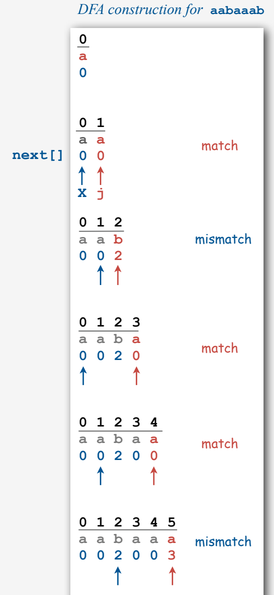
	0	1	2	3	4	5	6
p[]	a	a	b	a	a	a	a
next[]	0	0	2	0	0	3	3

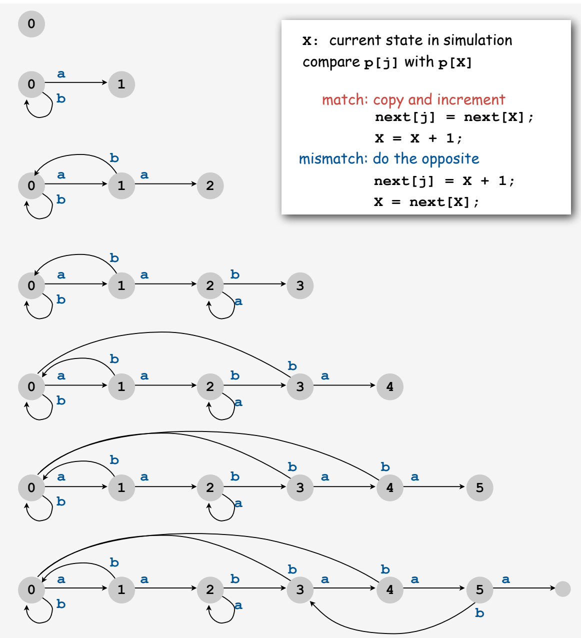


DFA for pattern `aabaaaa`

Knuth-Morris-Pratt DFA construction

DFA construction for `aabaaab`





`X`: current state in simulation
compare `p[j]` with `p[X]`

match: copy and increment
`next[j] = next[X];`
`X = X + 1;`

mismatch: do the opposite
`next[j] = X + 1;`
`X = next[X];`

DFA construction for KMP: Java implementation

```
int X = 0;
int[] next = new int[M];
for (int j = 1; j < M; j++)
{
   if (pattern.charAt(X) == pattern.charAt(j))
   {  // match
      next[j] = next[X];
      X = X + 1;
   }
   else
   {  // mismatch
      next[j] = X + 1;
      X = next[X];
   }
}
```

DFA Construction for KMP (assumes binary alphabet)

Analysis. Takes time and space proportional to pattern length.

Optimized KMP implementation

Ultimate search program for any given pattern.

- One statement comparing each pattern character to next.
- Match: proceed to next statement.
- Mismatch: go back as dictated by DFA.
- Translates to machine language (three instructions per pattern char).

```
int kmpsearch(char text[])
{
   int i = 0;
   s0: if (text[i++] != 'a') goto s0;
   s1: if (text[i++] != 'a') goto s0;
   s2: if (text[i++] != 'b') goto s2;
   s3: if (text[i++] != 'a') goto s0;
   s4: if (text[i++] != 'a') goto s0;
   s5: if (text[i++] != 'a') goto s3;
   s6: if (text[i++] != 'b') goto s2;
   s7: if (text[i++] != 'b') goto s4;
   return i - 8;
}
```

pattern[] next[]

assumes pattern is in text
(o/w use sentinel)

Knuth-Morris-Pratt summary

General alphabet.

- More difficult.
- Easy with `next[i][c]` indexed by mismatch position `i`, character `c`. (but too much space and time)
- KMP paper has ingenious solution that uses a single 1D `next[]` array.
  [build NFA, then prove that it finishes in 2N steps]

Bottom line. Linear-time pattern matching is possible (and practical).

Short history.

- Inspired by esoteric theorem of Cook.
- Discovered in 1976 independently by two theoreticians and a hacker.
  - Knuth: discovered linear time algorithm
  - Pratt: made running time independent of alphabet
  - Morris: trying to build a text editor
- Theory meets practice.

Exact pattern matching: other approaches

Rabin-Karp: make a digital signature of the pattern.

- Hashing without the table.
- Linear-time probabilistic guarantee.
- Plus: extends to 2D patterns.
- Minus: arithmetic ops slower than char comparisons.

Boyer-Moore: scan from right to left in pattern.

- Main idea: can skip M text chars when finding one not in the pattern.
- Needs additional KMP-like heuristic.
- Plus: possibility of sublinear-time performance (~ N/M).
- Used in Unix, emacs.

```
pattern  s y z y g y
text     a a a b b a a b a b a a a b b a g y
         s y z y g y
                     s y z y g y
                                 s y z y g y
```

Exact pattern match cost summary

Cost of searching for an M-character pattern in an N-character text.

algorithm	operations	typical	worst-case
brute-force	char compares	1.1 N †	M N
KMP	char compares	1.1 N †	2N
Karp-Rabin	arithmetic ops	3N	3N ‡
Boyer-Moore	char compares	N/M †	3N

† assumes appropriate model
‡ randomized

- exact pattern matching
- Knuth-Morris-Pratt
- **RE pattern matching**
- grep

Regular-expression pattern matching

Exact pattern matching. Find occurrences of a single pattern in text.
RE pattern matching. Find occurrences of one of multiple patterns in text.

Ex. [genomics]
- Fragile X syndrome is a common cause of mental retardation.
- Human genome contains triplet repeats of `cgg` or `agg`, bracketed by `gcg` at the beginning and `ctg` at the end.
- Number of repeats is variable, and correlated with syndrome.
- Use RE to specify pattern: `gcg(cgg|agg)*ctg`. Do RE pattern match on person's genome to detect Fragile X.

pattern (RE) `gcg(cgg|agg)*ctg`

text `gcggcgtgtgtgcgagagagtgggtttaaagctggcgcggaggcggctggcgcggaggctg`

RE 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.
- 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.

Regular expression examples

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

operation	example RE	in set	not in 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 abbbbbbbba	ab ababa
parentheses	a(a|b)aab	aaaab abaab	every other string
	(ab)*a	a ababababa	aa abba

Regular expression examples (continued)

Notation is surprisingly expressive

regular expression	in set	not in set
.*spb.* *(contains the trigraph spb)*	raspberry crispbread	subspace subspecies
a* | (a*ba*ba*ba*)* *(number of b's is a multiple of 3)*	bbb aaa bbbaababbaa	b bb baabbbaa
.*0.... *(fifth to last digit is 0)*	1000234 98701234	111111111 403982772
gcg(cgg|agg)*ctg *(fragile X syndrome)*	gcgctg gcgcggctg gcgcggaggctg	gcgcgg cggcggcggctg gcgcaggctg

and plays a well-understood role in the theory of computation.

Generalized regular expressions

Additional operations are often added for convenience.

Ex. `[a-e]+` is shorthand for `(a|b|c|d|e)(a|b|c|d|e)*`

operation	example RE	in set	not in set
one or more	a(bc)+de	abcde abcbcde	ade bcde
character classes	[A-Za-z][a-z]*	word Capitalized	camelCase 4illegal
exactly k	[0-9]{5}-[0-9]{4}	08540-1321 19072-5541	111111111 166-54-111
negations	[^aeiou]{6}	rhythm	decade

Caveat. Need to be alert for non-regular additions, e.g., back reference.

Regular expressions in Java

Validity checking. Is `input` in the set described by the `re`?
Java string library. Use `input.matches(re)` for basic RE matching.

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

```
% java Validate "..oo..oo." bloodroot
true

% 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
```

need help solving crosswords?

legal Java identifier

valid email address (simplified)

Social Security number

Regular expressions in other languages

Broadly applicable programmer's tool.

- Originated in Unix in the 1970s
- Many languages support extended regular expressions.
- Built into grep, awk, emacs, Perl, PHP, Python, JavaScript.

```
% grep NEWLINE */*.java
```

print all lines containing `NEWLINE` which occurs in any file with a `.java` extension

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

PERL. Practical Extraction and Report Language.

```
% 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-Za-z][a-z]*$/' dict.txt
```

do for each line

print all uppercase words

Regular expressions to the rescue

http://xkcd.com/208/

Regular expression caveat

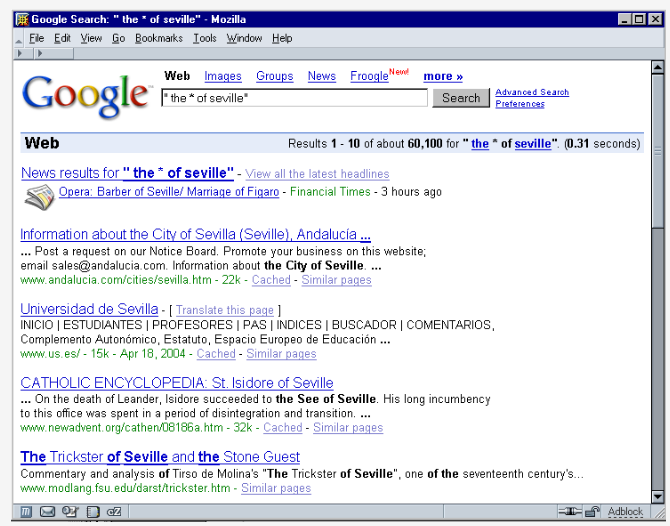
Writing a RE is like writing a program.

- Need to understand programming model.
- Can be easier to write than read.
- Can be difficult to debug.

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

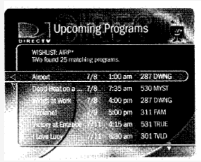
Can the average web surfer learn to use REs?

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



Can the average TV viewer learn to use REs?

TiVo. WishList has very limited pattern matching.



Using * in WishList Searches. To search for similar words in Keyword and Title WishList searches, use the asterisk (*) as a special symbol that replaces the endings of words. For example, the keyword *AIRP** would find shows containing "airport," "airplane," "airplanes," as well as the movie "Airplane!" To enter an asterisk, press the SLOW (▶) button as you are spelling out your keyword or title.

The asterisk can be helpful when you're looking for a range of similar words, as in the example above, or if you're just not sure how something is spelled. Pop quiz: is it "irresistible" or "irresistable?" Use the keyword *IRRESIST** and don't worry about it! Two things to note about using the asterisk:

- It can only be used at a word's end; it cannot be used to omit letters at the beginning or in the middle of a word. (For example, *AIR*NE* or **PLANE* would not work.)

Reference: page 76, Hughes DirectTV TiVo manual

Can the average programmer learn to use REs?

Perl RE for valid RFC822 email addresses

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

- exact pattern matching
- Knuth-Morris-Pratt
- RE pattern matching
- **grep**

GREP implementation: basic plan (first attempt)

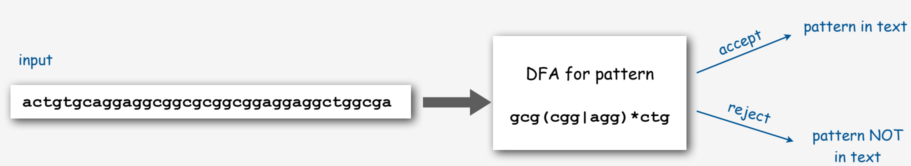
Overview is the same as for KMP !

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

Ken Thompson

Basic plan for grep (generalized regular expression print).

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



Duality

RE. Concise way to describe a set of strings.

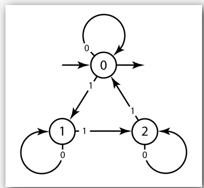
DFA. Machine to recognize whether a given string is in a given set.

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.

RE `0* | (0*10*10*10*)*`

number of 1's is a multiple of 3

DFA



number of 1's is a multiple of 3

Good news. Basic plan works.

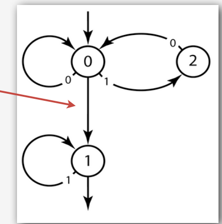
Bad news. The DFA can be exponentially large.

Consequence. Need better abstract machine.

Nondeterministic finite-state automata

NFA.
- May have 0, 1, or more transitions for each input symbol.
- May have ε-transitions (move to another state without reading input).
- Accept if any sequence of transitions leads to accept state.

convention: unlabeled arrows are ε-transitions



bitstrings that do not contain 110

in set: `111, 00011, 101001011`

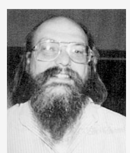
not in set: `110, 00011011, 00110`

linear — exponential blowup possible

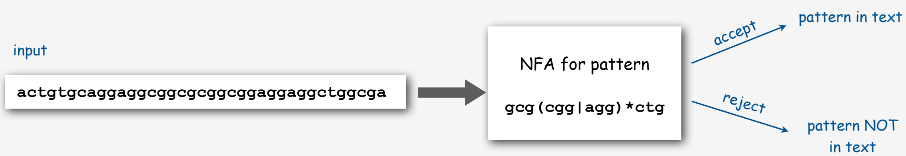
Proof of Kleene's theorem. RE ⇒ NFA ⇒ DFA ⇒ RE.

GREP implementation: basic plan (revised)

Basic plan for GREP.
- build NFA from RE.
- Simulate NFA with text as input.
- Give up on linear-time guarantee (but establish quadratic-time guarantee).



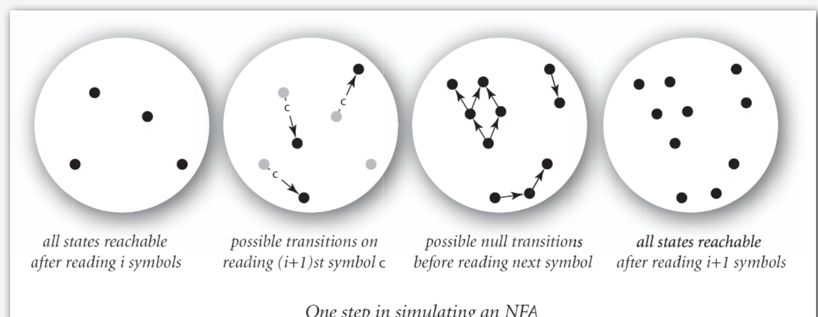
Ken Thompson



Simulating an NFA

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 symbols.

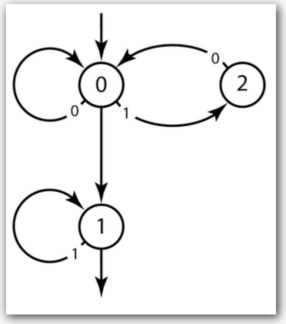


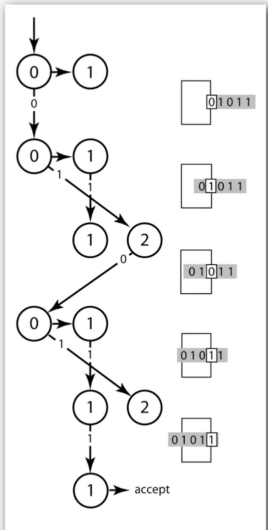
One step in simulating an NFA

Q. How to perform reachability?

A. Graph reachability in a digraph. (!)

NFA simulation



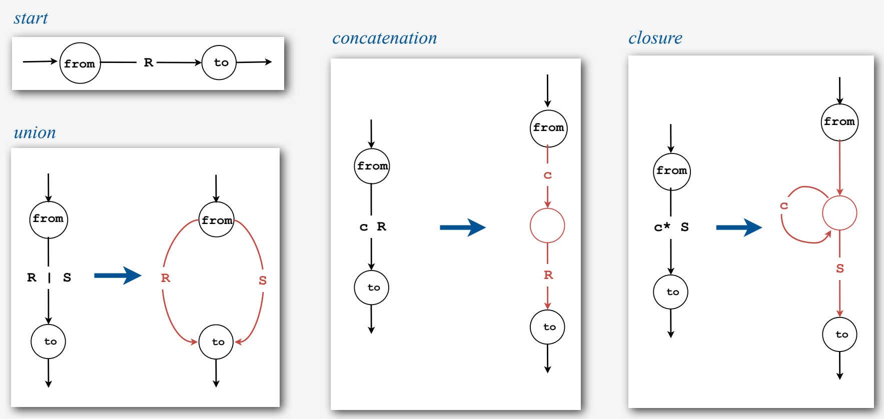


An NFA trace

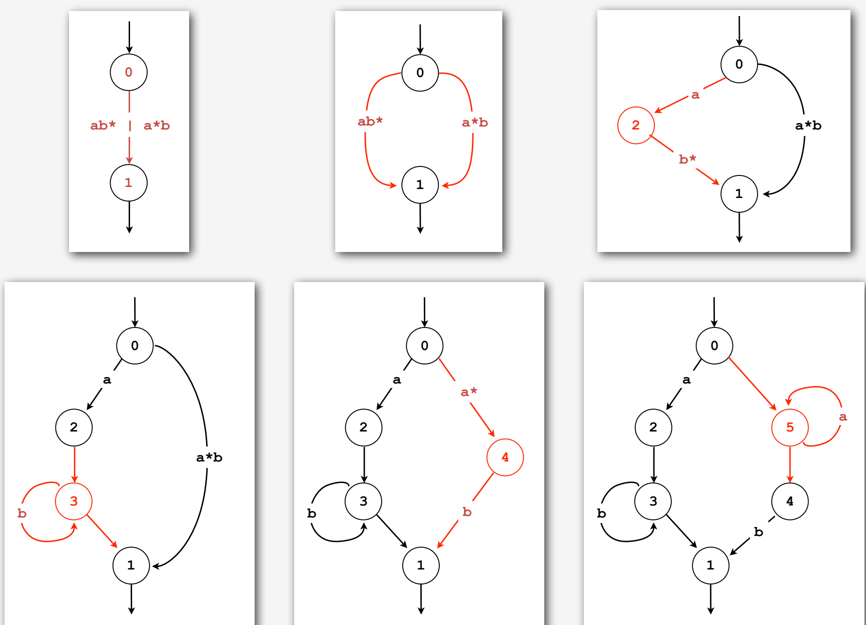
Converting from an RE to an NFA: basic transformations

Use generalized NFA with full RE on transitions.

- Start with one transition having given RE.
- Remove operators with transformations given below.
- Goal: standard NFA (all single-character or ε-transitions).



Converting from an RE to an NFA example: `ab* | a*b`



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 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

- Bottleneck: 1 graph reachability per input character.
- Can be substantially faster in practice if few ε-transitions.

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

Industrial-strength grep implementation

To complete the implementation,

- Deal with parentheses.
- 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.

Harvesting information

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

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

harvest patterns from DNA

harvest links from website

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


Regular expressions in Java (revisited)

RE pattern matching is implemented in Java's `Pattern` and `Matcher` classes.

```
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());
   }
}
```

`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!

grep, Java, Perl

```
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaac              1.6 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac            3.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac          9.7 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac       23.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac     62.2 seconds
% java Validate "(a|aa)*b" aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac 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 address to DOS a mail server.

Not-so-regular expressions

Back-references.

- \1 notation matches sub-expression that was matched earlier.
- Supported by typical RE implementations.

```
% java Harvester "\b(.+)\1\b"  dictionary.txt
beriberi
couscous
```

word boundary

Some non-regular languages.

- Set of strings of the form ww for some string w: beriberi.
- Set of bitstrings with an equal number of 0s and 1s: 01110100.
- Set of 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
- basis of programming languages

Compiler. A program that translates a program to machine code.

- KMP string ⇒ DFA.
- grep RE ⇒ NFA.
- javac Java language ⇒ Java byte code.

	KMP	grep	Java
pattern	string	RE	program
parser	unnecessary	check if legal	check if legal
compiler output	DFA	NFA	byte code
simulator	DFA simulator	NFA simulator	JVM

Summary of pattern-matching algorithms

Programmer.

- Implement exact pattern matching via DFA simulation (KMP).
- Implement RE pattern matching via NFA simulation (grep).

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.

Example of essential paradigm in computer science.

- Build intermediate abstractions.
- Pick the right ones!
- Solve important practical problems.