Regular Expressions

Linguistics 409 · Computational Linguistics

Rice University

January 14, 2013

Strings

- Text, when stored by a computer, is generally known as a **string**.

Strings

- Text, when stored by a computer, is generally known as a **string**.
- A string is a finite set of symbols (letters, numbers, space, punctuation, etc.)

Strings

- Text, when stored by a computer, is generally known as a **string**.
- A string is a finite set of symbols (letters, numbers, space, punctuation, etc.)
- `woodchuck` is an example of a string.

Strings

- Text, when stored by a computer, is generally known as a **string**.
- A string is a finite set of symbols (letters, numbers, space, punctuation, etc.)
- `woodchuck` is an example of a string.
- `is an example of a string` is an example of a string too.

Strings

- Text, when stored by a computer, is generally known as a **string**.
- A string is a finite set of symbols (letters, numbers, space, punctuation, etc.)
- `woodchuck` is an example of a string.
- `is an example of a string` is an example of a string too.
- `is an example of a string is an example of a string too` is an example of a string too.

Alphabets

- To be precise, a string is a sequence of zero or more symbols drawn from a particular *alphabet* Σ.
- We will refer to the special case of a null string, a string with zero symbols, with lowercase epsilon: ϵ
- An alphabet can contain any unordered collection of unique symbols (a **set**).
- For example:
  - $\Sigma_i = a, b, c, d, e$
  - $\Sigma_j = foo, bar, baz, qux$
  - ...

Formal Languages

- A **formal language** is a set of strings that can be generated or recognized using the alphabet Σ
- We can refer to the special case of *all* of the possible strings of a language with the notation: Σ^*
- That asterisk (*) is called a Kleene star

Patterns and Corpora

- We are going to call a collection of strings a **corpus**
- So let's say you want to find a particular **pattern** within the strings of a corpus.
- For this we will use **regular expressions**
- A regular expression is a description of a particular subset of strings that can be generated or recognized using the alphabet Σ
- Every regular expression, therefore, defines a **regular language** (more on this next time).

Basic Patterns

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/Claire␣says,/	" "Dagmar, my gift please," Claire says,"
/DOROTHY/	"SURRENDER DOROTHY"
/!/	"You've left the burglar behind again!" said Nori

J&M Figure 2.0

Find all occurrences of the string `the`

The North Wind and the Sun were disputing which was the stronger, when a traveller came along wrapped in a warm cloak. They agreed that the one who first succeeded in making the traveller take his cloak off should be considered stronger than the other. Then the North Wind blew as hard as he could, but the more he blew the more closely did the traveller fold his cloak around him; and at last the North Wind gave up the attempt. Then the Sun shined out warmly, and immediately the traveller took his cloak off. And so the North Wind was obliged to confess that the Sun was the stronger of the two.

Accuracy and Coverage

```
grep 'the' northwind.txt
```

- this search matched many appropriate strings, but also returned a number of **false positives**.
- It also missed a number of occurrences of *the*. We'll call these **false negatives**
- This is an important pattern we will see repeatedly in this class.
- The response will often be an iterative process to improve **accuracy** by decreasing false positives and improve **coverage** by eliminating false negatives.

Disjunction

RE	Match	Example Patterns
`/[wW]oodchuck/`	Woodchuck or woodchuck	"Woodchuck"
`/[abc]/`	'a', 'b', *or* 'c'	"In uomini, in soldati"
`/[1234567890]/`	any digit	"plenty of 7 to 5"

J&M Figure 2.1

Anchors

- ^ = start of line
- $ = end of line
- \b word boundary
- \B non word boundary

Ranges

RE	Match	Example Patterns Matched
/[A-Z]/	an upper case letter	"we should call it 'Drenched Blossoms' "
/[a-z]/	a lower case letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

J&M Figure 2.2

Two more uses of the caret

RE	Match (single characters)	Example Patterns Matched
`[^A-Z]`	not an upper case letter	"Oyfn pripetchik"
`[^Ss]`	neither 'S' nor 's'	"I have no exquisite reason for't"
`[^\.]`	not a period	"our resident Djinn"
`[e^]`	either 'e' or '^'	"look up ^ now"
`a^b`	the pattern 'a^b'	"look up a^ b now"

Two of the uses of caret in regular expressions (J&M Figure 2.3)

Optionality

RE	Match	Example Patterns Matched
`woodchucks?`	woodchuck or woodchucks	"woodchuck"
`colou?r`	color or colour	"colour"

- ? = one or none of the preceding symbol
- * = none or more of the preceding symbol
- + = one or more of the preceding symbol (/x+/ is equivalent to /xx*/)
- . = any single symbol *except* carriage return

RE	Match	Example Patterns
`/beg.n/`	any character between *beg* and *n*	begin, beg'n, begun

Operator Precedence

Parenthesis	`( )`
Counters	`* + ? {}`
Sequences and anchors	`the ^my end$`
Disjunction	`|`

J&M operator precedence

Again: Find all occurrences of the string `the`

The North Wind and the Sun were disputing which was the stronger, when a traveller came along wrapped in a warm cloak. They agreed that the one who first succeeded in making the traveller take his cloak off should be considered stronger than the other. Then the North Wind blew as hard as he could, but the more he blew the more closely did the traveller fold his cloak around him; and at last the North Wind gave up the attempt. Then the Sun shined out warmly, and immediately the traveller took his cloak off. And so the North Wind was obliged to confess that the Sun was the stronger of the two.

Common Aliases

RE	Expansion	Match	Examples
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

J&M aliases for common sets of characters

Counting

RE	Match
`*`	zero or more occurrences of the previous char or expression
`+`	one or more occurrences of the previous char or expression
`?`	exactly zero or one occurrence of the previous char or expression
`{n}`	*n* occurrences of the previous char or expression
`{n,m}`	from *n* to *m* occurrences of the previous char or expression
`{n,}`	at least *n* occurrences of the previous char or expression

J&M counting operators

Escaping

RE	Match	Example Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Why don't they come and lend a hand?"
\n	a newline	
\t	a tab	

J&M special characters that need to be escaped.

For next time:

For next time:

J&M Chapter 2 pp 26 – 44
Finite State Automata

“Imagine that you have become a passionate fan of woodchucks...”

Substitutions and Memory

- Regular expressions can be used in many text processing tools to replace one string with another, e.g.:

  s/color/colour/

- Some extended regexp implementations (e.g. perl) support the use of numbered **registers** that allow you to memoize and reproduce matches, e.g.:

  s/My name is (\w*)/Hello, \1./

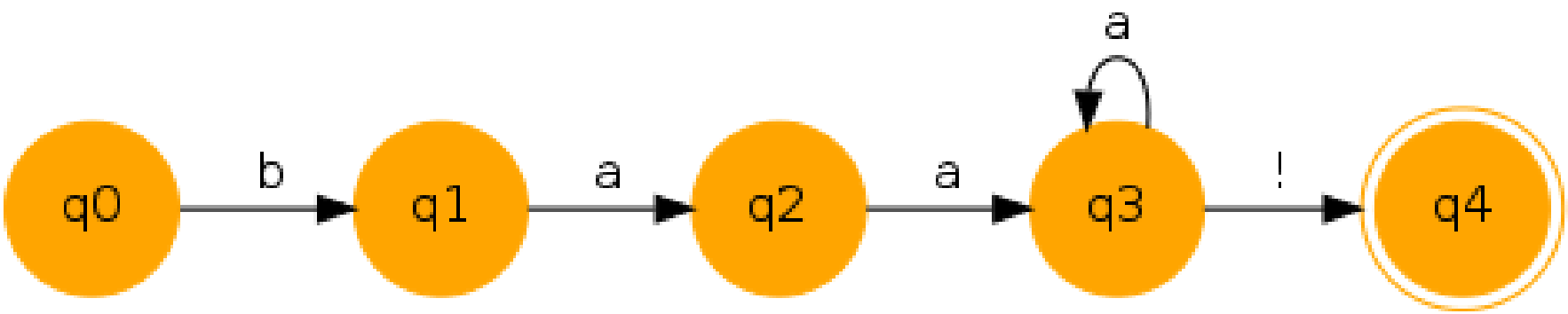

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Finite State Automata

- Remember, every regular expression defines a regular language.
- A regular language can also be represented graphically with a **finite state automaton** or **FSA**.
- FSA's are at the core of much of what we'll do this semester.

FSA's as directed graphs

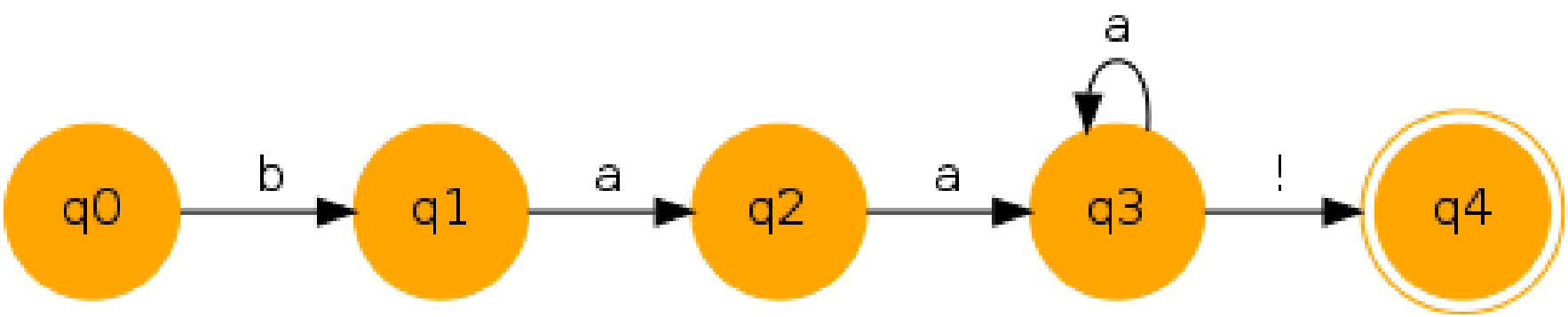
- Let's start with the sheep language
- what strings can this regex/FSA generate or recognize?
- /^baa+!$/



FSA's as state transition matrices

/^baa+!$/

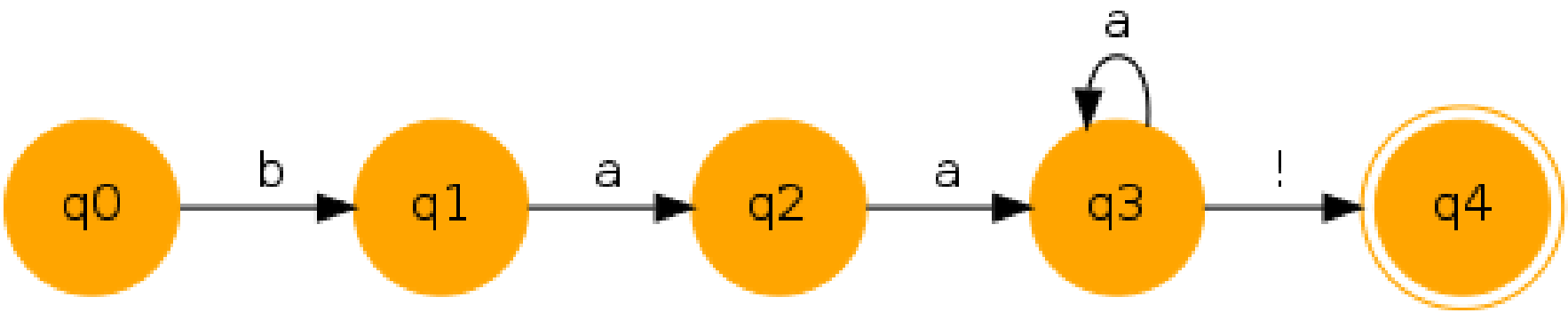
	input		
state	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅



More formally

You can specify an FSA with the following:

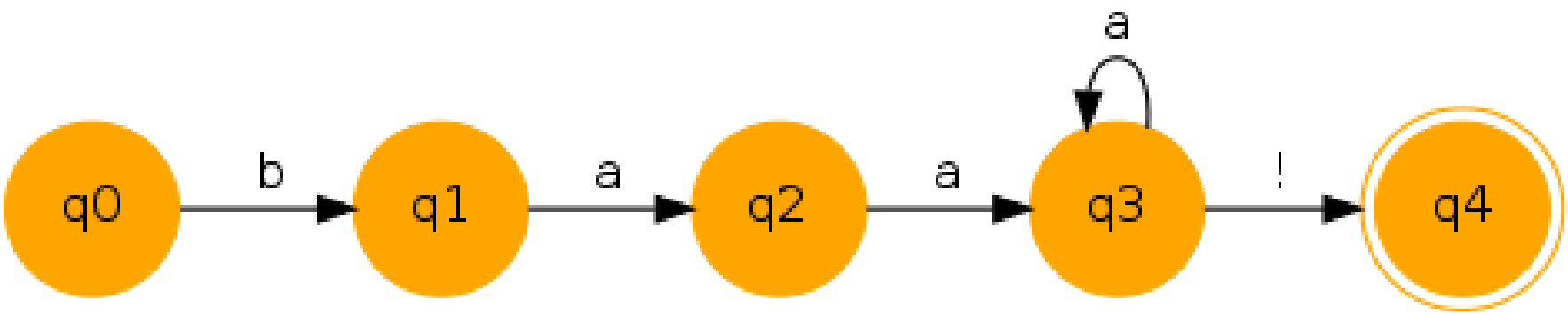
- Σ : the finite alphabet
- Q : the finite set of N states
- q_0 : A start state
- F : A set of accept/final states
- $\delta(q, i)$: A transition function mapping Q x Σ to Q



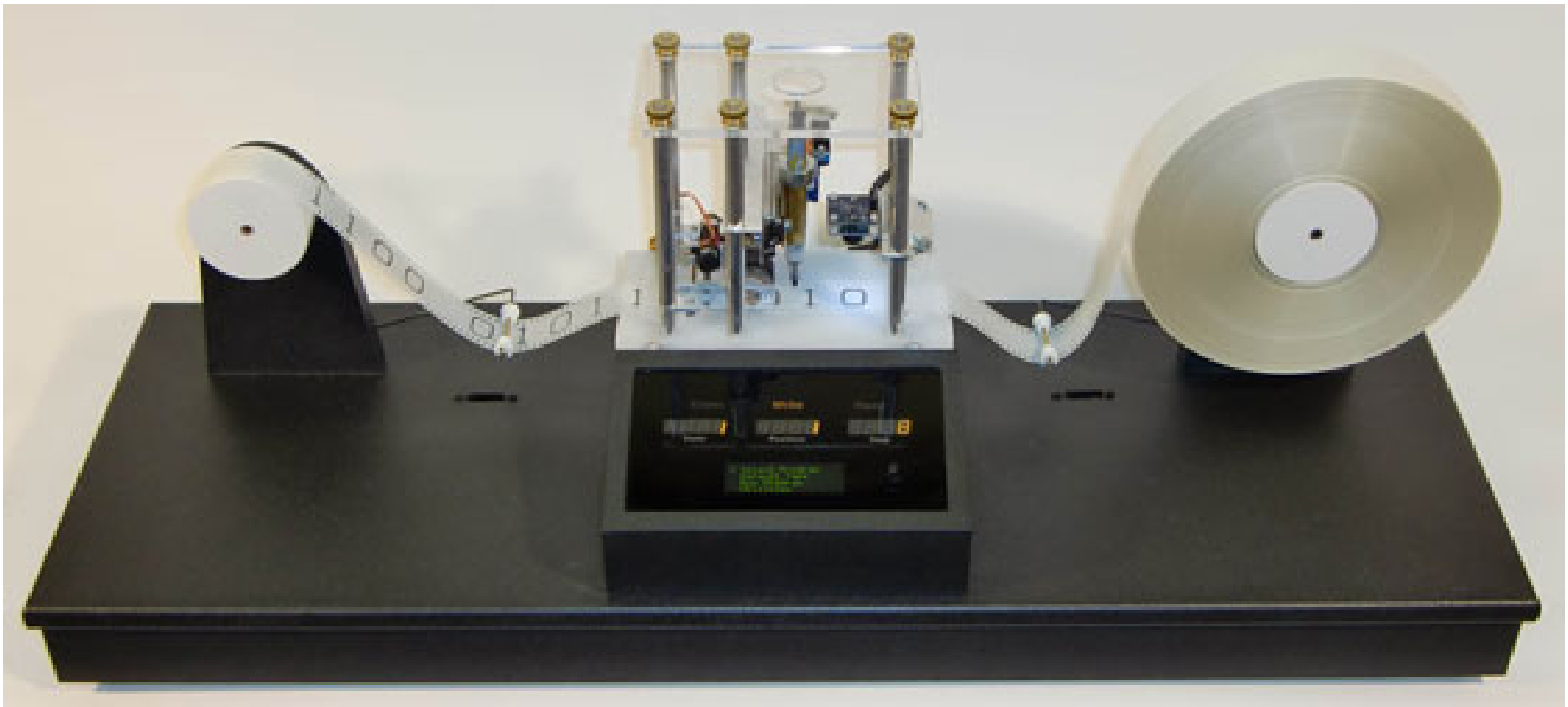
Sheep FSA

what can we say about this FSA?

- what is its alphabet? (minimally)
- how many states does it have?
- ˆwhich nodes? (is|are) start states?\?$
- ˆwhich nodes? (is|are) accept states?\?$
- how many transitions (edges) does it have?



Reading an infinite tape...

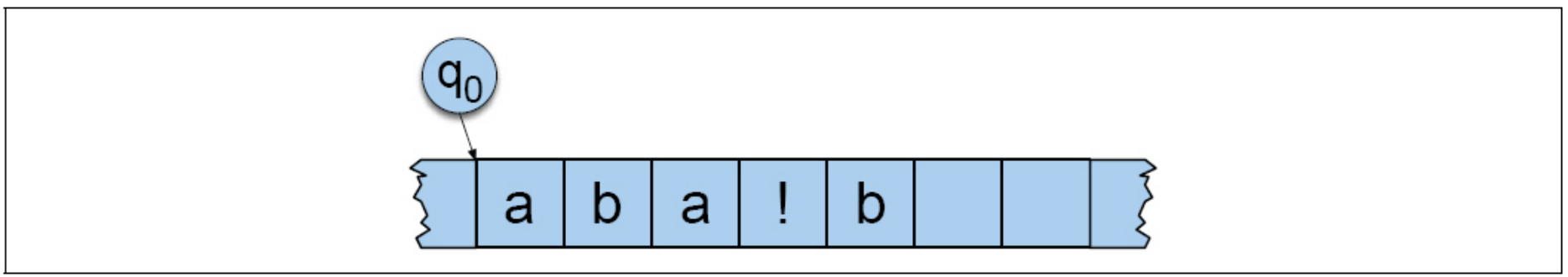


imagine a machine that can read (and write) cells on an infinitely long tape...

A tape with cells

What happens here?

	input		
state	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅



J&M Figure 2.11

D-Recognize

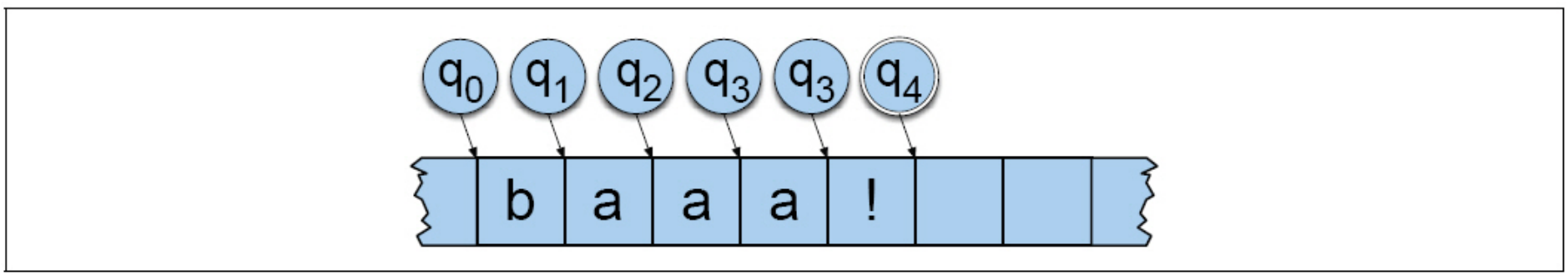
```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
   if End of input has been reached then
    if current-state is an accept state then
      return accept
    else
      return reject
   elsif transition-table[current-state,tape[index]] is empty then
     return reject
   else
     current-state ← transition-table[current-state,tape[index]]
     index ← index + 1
  end
```

J&M Figure 2.12

Tracing a run of FSA 1

What happens here?

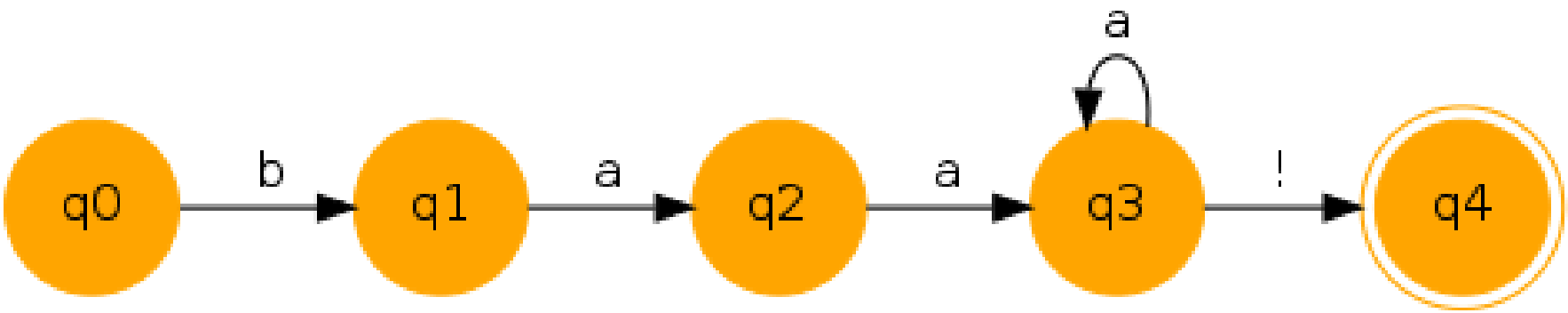
	input		
state	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

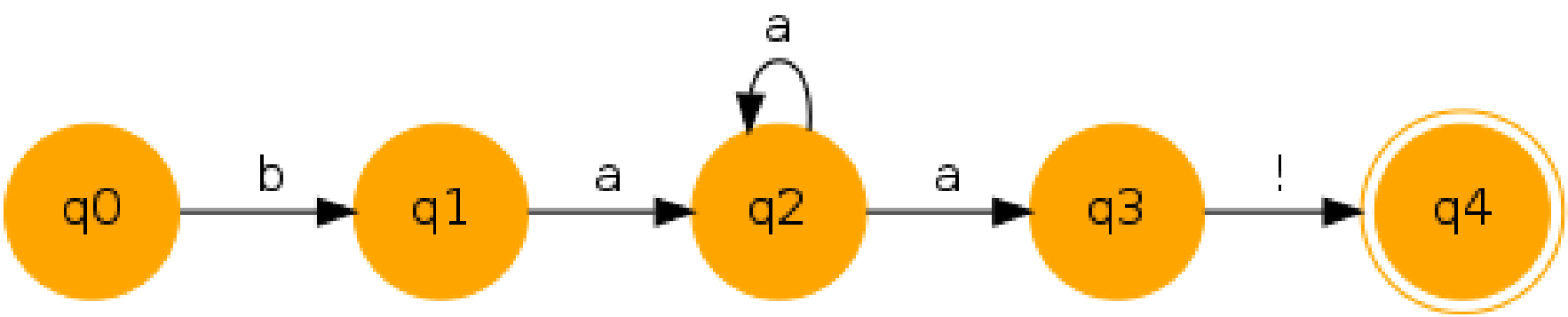


J&M Figure 2.13

Sheep FSA (but notice...)

Different machines can define the same regular language:





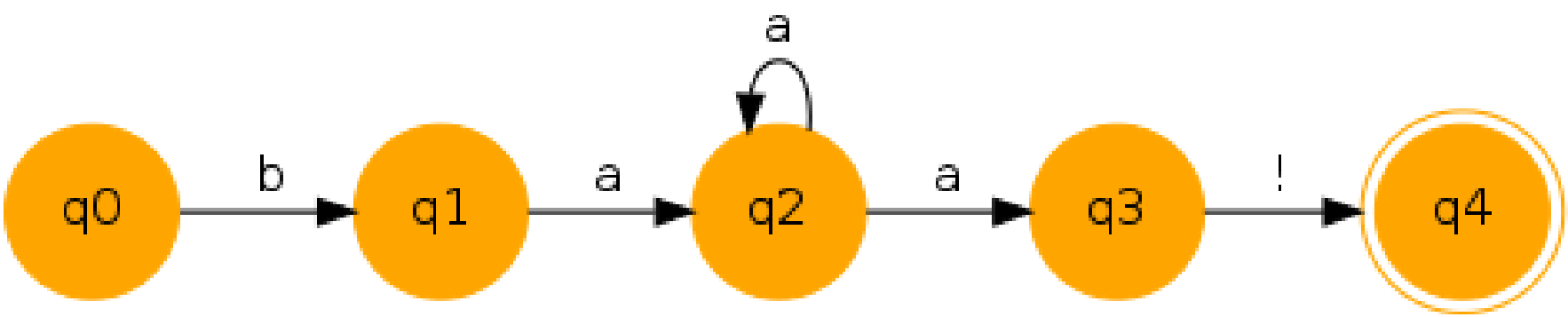
On FSAs, DFSAs & NFSAs

- So far, the behavior of our FSAs has been uniquely determined by the pairing of state and input $\delta(q, i)$
- An FSA of this type is **deterministic**, a **DFSA**.
- But the equivalent FSA for baa! on the previous slide has a decision point when q = 2 and i = a.
- This type of FSA is **non-deterministic** and we will call this important class of automaton an **NFSA**.

Non-deterministic version of baa!

/^ba+a!$/

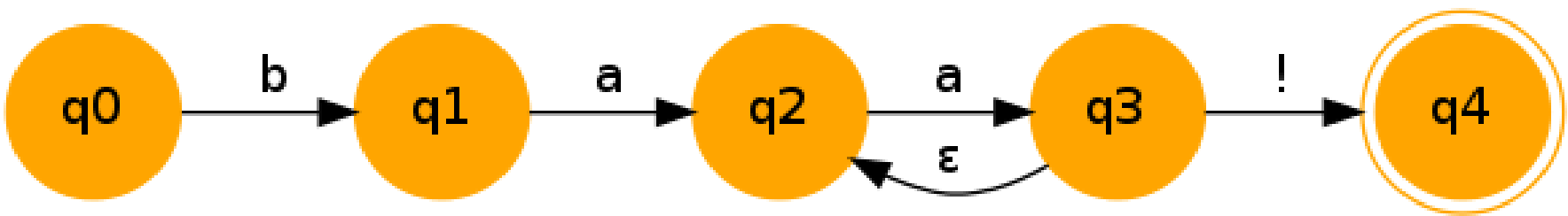
	input		
state	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	2,3	∅
3	∅	∅	4
4:	∅	∅	∅



Another Non-deterministic version of baa!

Adding an ϵ transition from q3 $\rightarrow$ q2 creates an NFSA equivalent in output to our FSA1 (which had a self loop on q3).

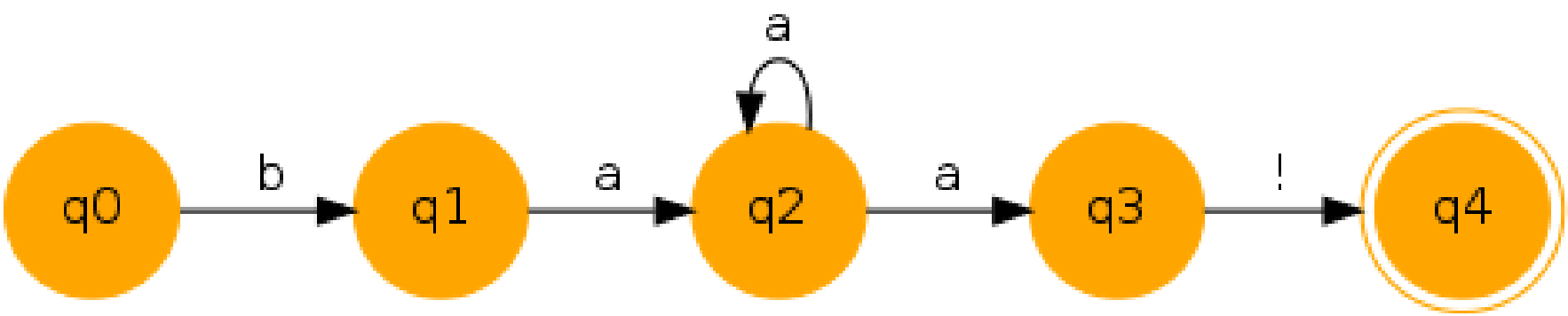
	input			
state	b	a	!	ϵ
0	1	$\varnothing$	$\varnothing$	$\varnothing$
1	$\varnothing$	2	$\varnothing$	$\varnothing$
2	$\varnothing$	3	$\varnothing$	$\varnothing$
3	$\varnothing$	$\varnothing$	4	2
4:	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$



NFSAs can make mistakes

Input: `baaaa!`

- Suddenly we have the ability to make an incorrect decision!
- Let's say we decide to transition to q3 from $\delta(2, a)$
- What's wrong with that decision?
- What are some ways we might recover from it (or avoid the error in the first place)?



Solutions to non-determinism:

- **Backup**: Store the **search state** (current node and input) at each decision point, return to the previous decision point and try again if you fail.
- **Look-ahead**: Read further along the input to make an informed decision.
- **Parallelism**: Just take all of the paths in parallel.

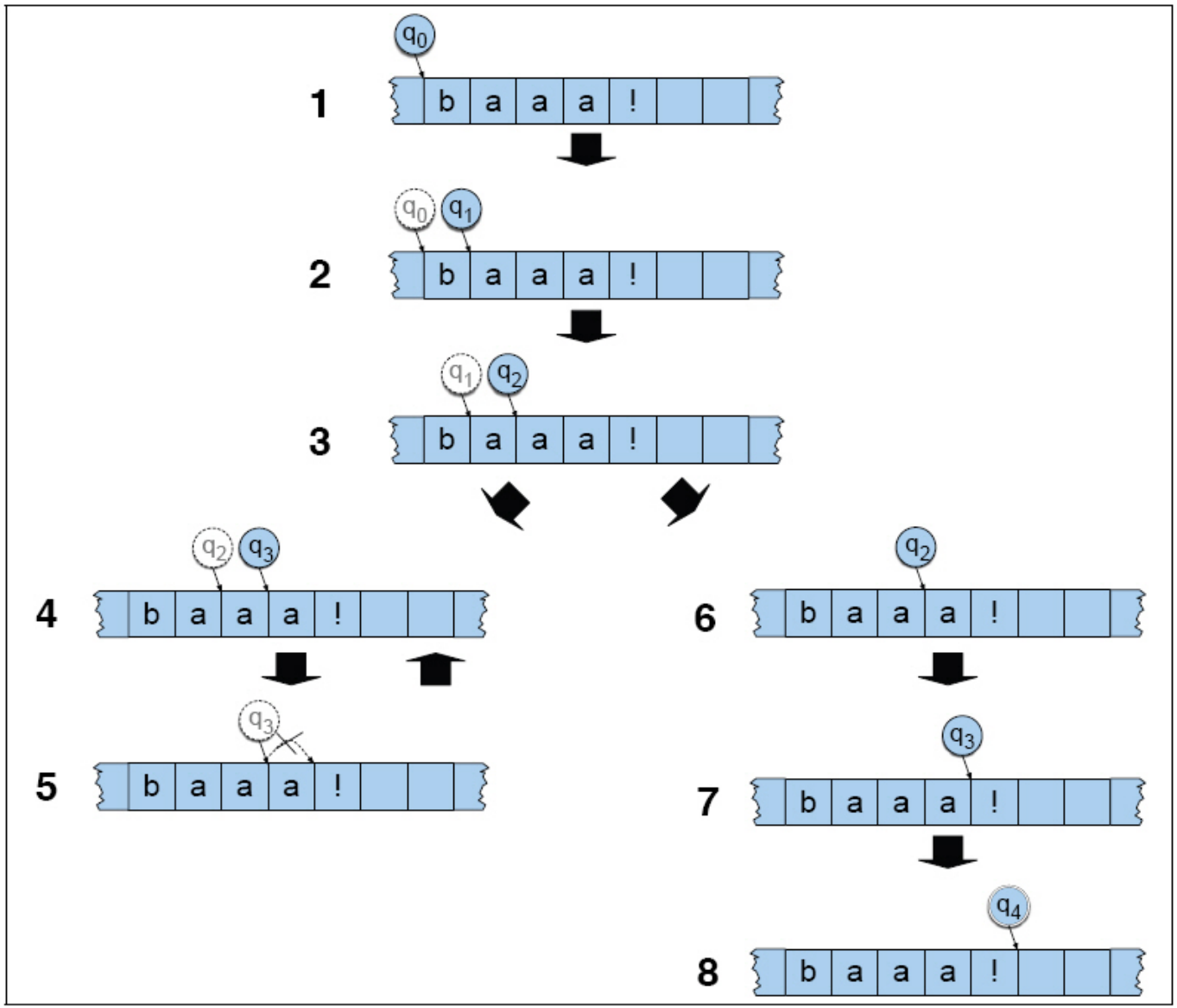
Recognition as Search

- The book gives pseudocode for the algorithm ND-RECOGNIZE.
- This algorithm recognizes strings using an NFSA and a backup strategy that results in an exhaustive **state-space search**.
- State space search generates a list of possible solutions and then exhaustively explores them.
- However, the order in which this exploration happens can be extremely important in determining how fast and efficient the search is.

Depth First vs Breadth First Search

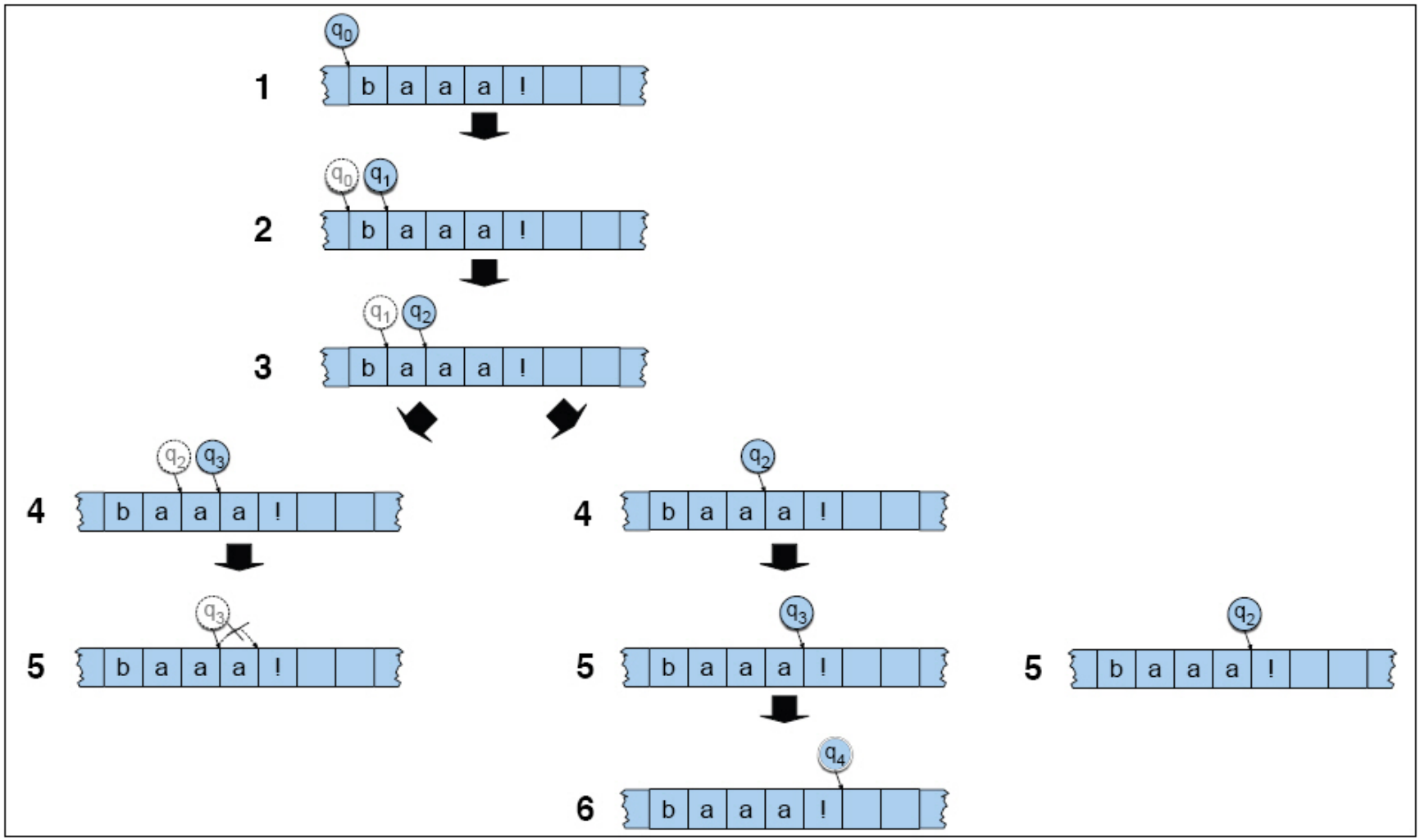
- Two possible ways of ordering search alternatives are **depth first** and **breadth first** search.
- In a depth first search, alternatives are pushed onto a **stack** and evaluated in a last in, first out (LIFO) order.
- In a breadth first search, alternatives are added to a **queue** and evaluated in a first in, last out (FIFO) order.

Depth First (LIFO)



J&M Figure 2.20

Breadth First (FIFO)



J&M Figure 2.21

For next time:

For next time:

1. Assignment 1 will be posted tonight.
2. There will be a short UNIX reading on OwlSpace for Wednesday.
3. Please bring a computer if you have one!
4. Wednesday: **Wrap up FSAs & Start UNIX**