Graduate Algorithms

CS673-2016F-20

String Matching

David Galles

Department of Computer Science
University of San Francisco

20-0: String Matching

T P (pattern)

- Given a source text, and a string to match, where does the string appear in the text? – returns shift i.e. how many time I shift to get match
  - Also can return T/F
- Example: ababbabbaba and abbab

↳ shift = 2

a	b	a	b	b	a	b	b	a	b	a
		x	x	x	x	x				
					x	x	x	x	x	

20-1: String Matching

Returns all shifts

NAIVE-STRING-MATCHER(T, P)

$n \leftarrow$ length[T]

$m \leftarrow$ length[P]

Run $O(n-m)$ if $n >> m$ $O(n)$

for $s \leftarrow 0$ to $n - m$ do ← Potential shift < from 0 to n-m >

match ← true

$O(m)$ → for $j \leftarrow 1$ to m do

Thus, $\boxed{O(nm)}$

if $T[i + j] \neq T[j]$ then

match ← false

if match then

Print "Pattern occurs with shift" s

20-2: String Matching

- Time for naive string matching:
  - $O(n * m)$
- What if we could compare strings in unit time?

20-3: String Matching

- Strings are over $\{0 \ldots 9\}$
- Example: Match 512 in 13512631842
  - We can consider strings/substrings to be integers
  - Do a comparison in a single instruction

20-4: Rabin-Karp

(Benefits from fact that a everything in computer is stored as numbers)

- Strings are over $\{0 \ldots 9\}$

Thus 'inner' 'for' loop runs O(1) time

- Example: Match 512 in 13512631842
  - Compare 512 to 135 (compare as number) i.e '512 == 135'
  - Compare 512 to 351
  - Compare 512 to 512
  - . . . etc

20-5: Rabin-Karp

- Example: Match 512 in 13512631842 (0 1 2)
  - (relatively) easy to create 135
  - How do we modify 135 to get 351?

- 135 → 35 → 350 → 350+1 = 351
- Subtract MSD → x 10 → Add LSD
- $-100 \times \underbrace{T[S]}_{\text{Digit that needs to be subtracted}} \to \times 10 \to \# + T[S+m]$ S → start!

i.e. 135−100 → 35 x10 → 350+1 = 351   0 + 3

jump to 20-8

20-6: Rabin-Karp

- Example: Match 512 in 13512631842
  - (relatively) easy to create 135
  - How do we modify 135 to get 351?
    - Remove first digit, shift remaining digits to left, append the next digit in the input
    - Subtract $1 * 10^3$, multiply by 10, add 1
    - $t_{s+1} = 10 * (t_s - 10^{m-1} T[s+1]) + T[s+m+1]$

20-7: Rabin-Karp

- This works great for matching numbers – what about strings of letters?

20-8: Rabin-Karp

- This works great for matching numbers – what about strings of letters?
  - Strings of letters are just numbers in base 26
  - (ASCII strings are just numbers in base 256)
- Problems with this method?

p: ABC <PAT>

52 53 54 → to big for 'int' '-65536' - '65535'

$\boxed{525354}$

Sol^n Divide by 'q'

jump <20-14>

20-9: Rabin-Karp

- This works great for matching numbers – what about strings of letters?
  - Strings of letters are just numbers in base 26
  - (ASCII strings are just numbers in base 256)
- Problems with this method?
  - Numbers get big fast – won't fit in a single integer
  - What can we do?

20-10: Rabin-Karp

- This works great for matching numbers – what about strings of letters?
  - Strings of letters are just numbers in base 26
  - (ASCII strings are just numbers in base 256)
- Problems with this method?
  - Numbers get big fast – won't fit in a single integer
  - – Use modular arithmetic

20-11: Rabin-Karp

- First, using base-d numbers instead of base-10 numbers:
  - $t_{s+1} = d * (t_s - T[s+1] * h) + T[s+m+1]$
    - $h = d^{m-1}$, computed once
  - Compare input number to $t_1, t_2, \ldots t_{n-m}$
  - Problem occurs when t_k could be too large to fit in an integer ...

20-12: Rabin-Karp

- Next, use modular arithmetic
- $t_{s+1} = (d * (t_s - T[s+1] * h) + T[s+m+1]) \mod q$
  - $h = d^{m-1} \mod q$, computed once
- Compare input number to $t_1, t_2, \ldots t_{n-m}$
- Problems?

20-13: Rabin-Karp

- Next, use modular arithmetic
- $t_{s+1} = (d * (t_s - T[s+1] * h) + T[s+m+1]) \mod q$
  - $h = d^{m-1} \mod q$, computed once
- Compare input number to $t_1, t_2, \ldots t_{n-m}$
- Problems?
  - Spurious hits (could have $t_k =$ input number, even if the strings are not the same

20-14: Rabin-Karp

- Source String 2359023141526739921
- Matching 31415, $q = 13$ Divide by q understand '20-16'
  - $31415 \mod 13 = 7$
  - $h = 10^4 \mod 13 = 3$
  - $t_1 = 23590 \mod 13 = 8$

$$\begin{aligned} t_2 &= (d * (t_1 - T[1] * h) + T[1 + (m + 1)]) \mod q \\ &= (10 * (8 - 2 * 3) + 2) \mod 13 \\ &= 9 \end{aligned}$$

20-15: Rabin-Karp

- Source String $235902314152673992 1$
- Matching 31415, $q = 13$
  - $31415 \mod 13 = 7$
  - $h = 10^4 \mod 13 = 3$
  - $t_1 = 23590 \mod 13 = 8$

$$
\begin{aligned}
t_3 &= (d * (t_2 - T[2] * h) + T[2 + (m + 1)]) \mod q \\
&= (10 * (9 - 3 * 3) + 3) \mod 13 \\
&= 3
\end{aligned}
$$

20-16: Rabin-Karp

- matching 31415
  - $31415 \mod 13 = 7$ – Remainder – 1

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1	String
				8	9	3	11	0	1	7	8	4	5	10	11	7	9	11	t_k

→ Divide each subset of 5 digit with 'q' & check for same 'n-m+1' remainder as 1

- We get hits on:
  - 31415
  - 67399

e.g. False +ve

Problem :
False +ve i.e. another # may also have same remainder!

20-17: Rabin-Karp

- Dealing with spurious hits
  - Every time we get a potential hit, check the actual strings
- Running time:
  1. • $O(n)$ to go through list 'cal all remainders'
  2. • $O(m)$ to verify each actual match ← check if same True +ve
  3. • $O(m)$ to check each spurious hit ← false +ve

How many times?
$q = 13$, Given #. what is the Probability $X \% q = 7$? '$1/q$'
– No. we are checking in substring $n-m+1$ i.e. '$\frac{n-m+1}{q}$'

20-18: Rabin-Karp

- Running time:
  - $O(n)$ to go through list
  - $O(m)$ to verify each actual match
  - $O(m)$ to check each spurious hit

$\exp[s] = \frac{n-m+1}{q}$

- $O(n + (v + s) * m)$, were v = # of actual hits, s = # of spurious hits.
- Expected running time: $O(n) + O(m(v + n/q))$
  - Assuming expected # of spurious hits = n/q

$O(n) + O(mv + \frac{nm}{q})$

Check for q -
- $q-1 <$ Max'int'
- q fit in "word" i.e. to fit single register!

q is prime & largest q satisfying

DFA

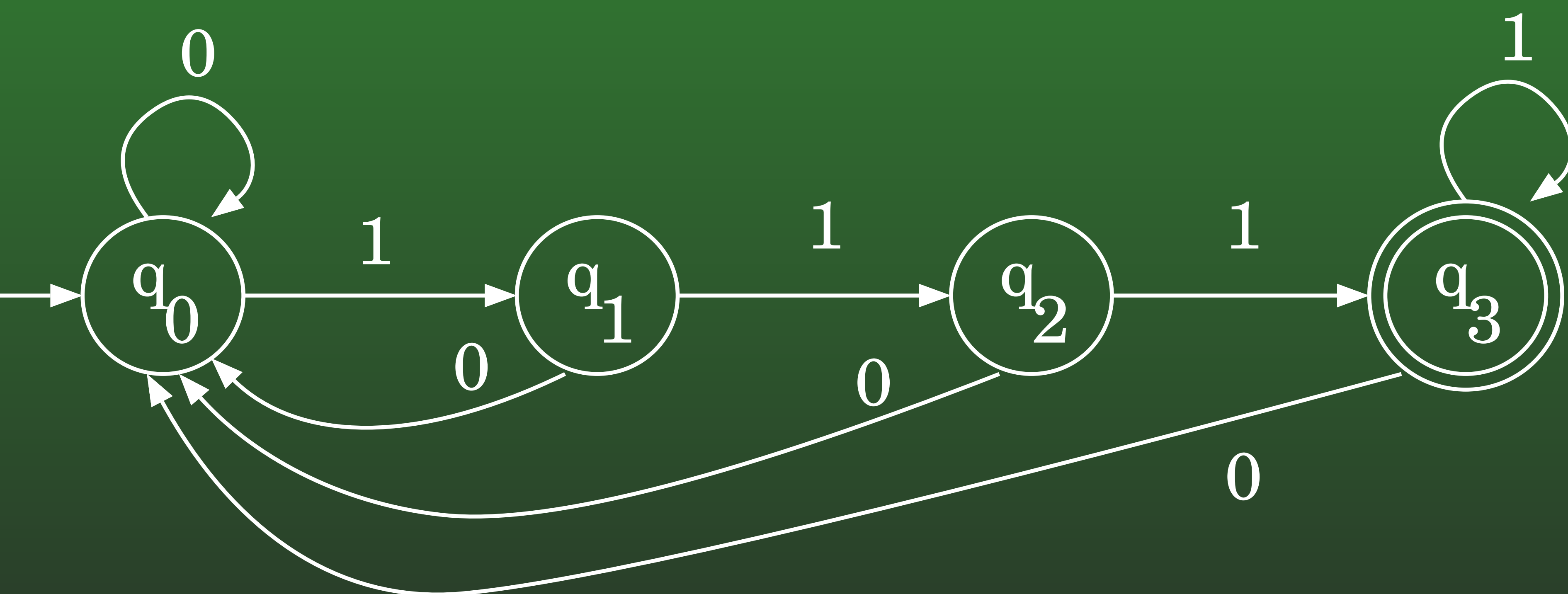
- Set of states Q
- $q_0 \in Q$ start state
- $A \in Q$ accepting states
- Σ input alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ Transition function

DFA

- Start in the initial state
- Go through the string, one character at a time, until the string is exhausted
- Determine if we are in a final state at the end of the string
  - If so, string is accepted
  - If not, string is rejected

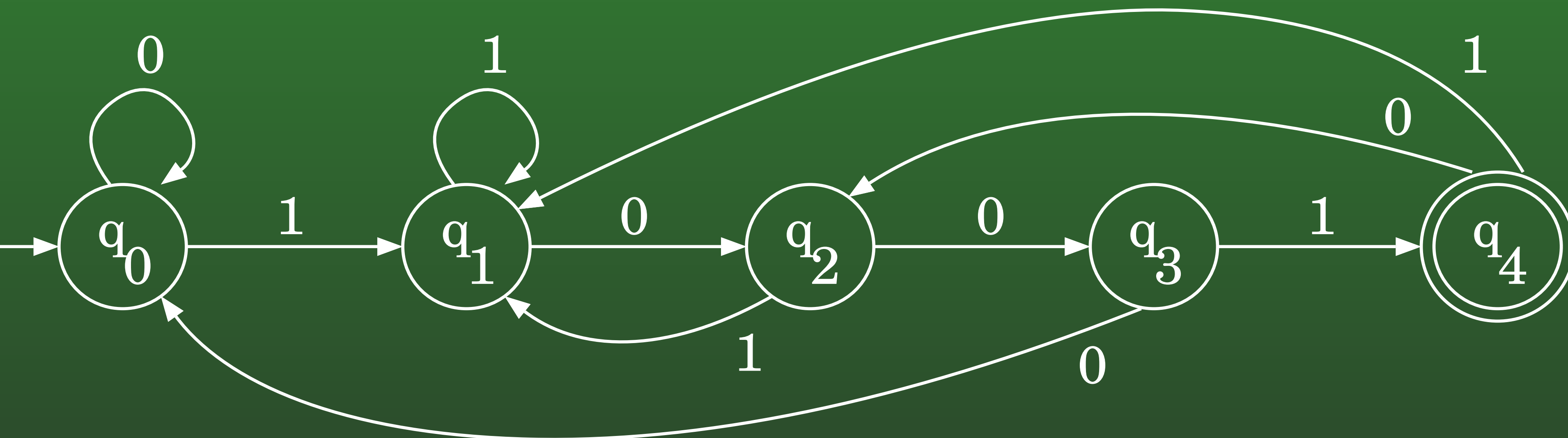
DFA

- All strings over {0,1} that end in 111



DFA

- All strings over {0,1} that end in 1001



20-23: DFA

- You can use the DFA for all strings that end in 1001 to find all occurrences of the substring 1001 in a larger string
  - Start at the beginning if the larger string, in state q_0
  - Go through the string one symbol at a time, moving through the DFA
  - Every time we enter a final state, that's a match

20-24: DFA

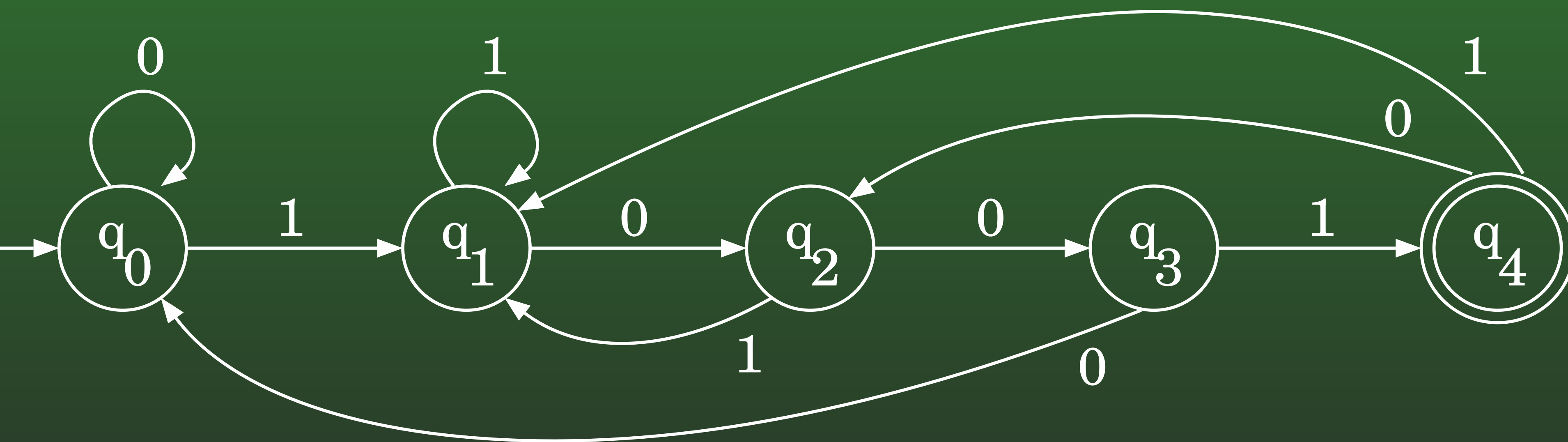
- Creating transition function δ:
- Create a new concept: $\sigma_P(x)$
  - Length of the longest prefix of P that is a suffix of x
  - $P = aba$
  - $\sigma_P(cba) = 1, \sigma_P(abc) = 0, \sigma_P(cab) = 2, \sigma_P(caba) = 3$
- P_k = first k symbols of P

DFA

- Creating the states of the DFA
- If the input pattern is $P[1 \ldots m]$:
  - DFA has $m+1$ states, $q_0 \ldots q_m$
  - State k “means” last k elements in the string so far match first k elements in P

20-26: DFA

- Pattern: 1001
- State k “means” last k elements in the string so far match first k elements in P



20-27: DFA

- $\delta(q, a) = \sigma(P_q a)$
- To find $\delta(q, a)$:
  - Start with string P_q: first q characters of P
  - Append a, to get $P_q a$
  - Find the longest prefix of P that is a suffix of $P_q a$.

20-28: DFA

- Building δ:

$m \leftarrow$ length[P]
for $q \leftarrow 0$ to m do
    for each character $a \in \Sigma$ do
        $k \leftarrow \min(m+1, q+2)$
        do
            $k \leftarrow k - 1$
        until P_k =] $P_q a$
        $\delta(q, a) \leftarrow k$

DFA

- Example:
  - $P = ababca$, String = $cbababcababc$

20-30: DFA

- $P = ababca$
- P_0 :
  - $P_0a = a$: q_1 $\delta(q_0, a) = q_1$
  - $P_0b = b$: q_0 $\delta(q_0, b) = q_0$
  - $P_0c = c$: q_0 $\delta(q_0, c) = q_0$

20-31: DFA

- $P = ababca$
- $P_1 : a$
  - $P_1 a = aa$: q_1 $\delta(q_1, a) = q_1$
  - $P_1 b = ab$: q_2 $\delta(q_1, b) = q_2$
  - $P_1 c = ac$: q_0 $\delta(q_1, c) = q_0$

20-32: DFA

- $P = ababca$
- $P_2 : ab$
  - $P_2a = aba$: q_3 $\qquad \delta(q_2, a) = q_3$
  - $P_2b = abb$: q_0 $\qquad \delta(q_2, b) = q_0$
  - $P_2c = abc$: q_0 $\qquad \delta(q_2, c) = q_0$

20-33: DFA

- $P = ababca$
- $P_3 : aba$
  - $P_3a = abaa$: q_1 $\delta(q_3, a) = q_1$
  - $P_3b = abab$: q_4 $\delta(q_3, b) = q_4$
  - $P_3c = abac$: q_0 $\delta(q_3, c) = q_0$

20-34: DFA

- $P = ababca$
- $P_4 : abab$
  - $P_4a = ababa$: q_3 $\delta(q_4, a) = q_3$
  - $P_4b = ababb$: q_0 $\delta(q_4, b) = q_0$
  - $P_4c = ababc$: q_5 $\delta(q_4, c) = q_5$

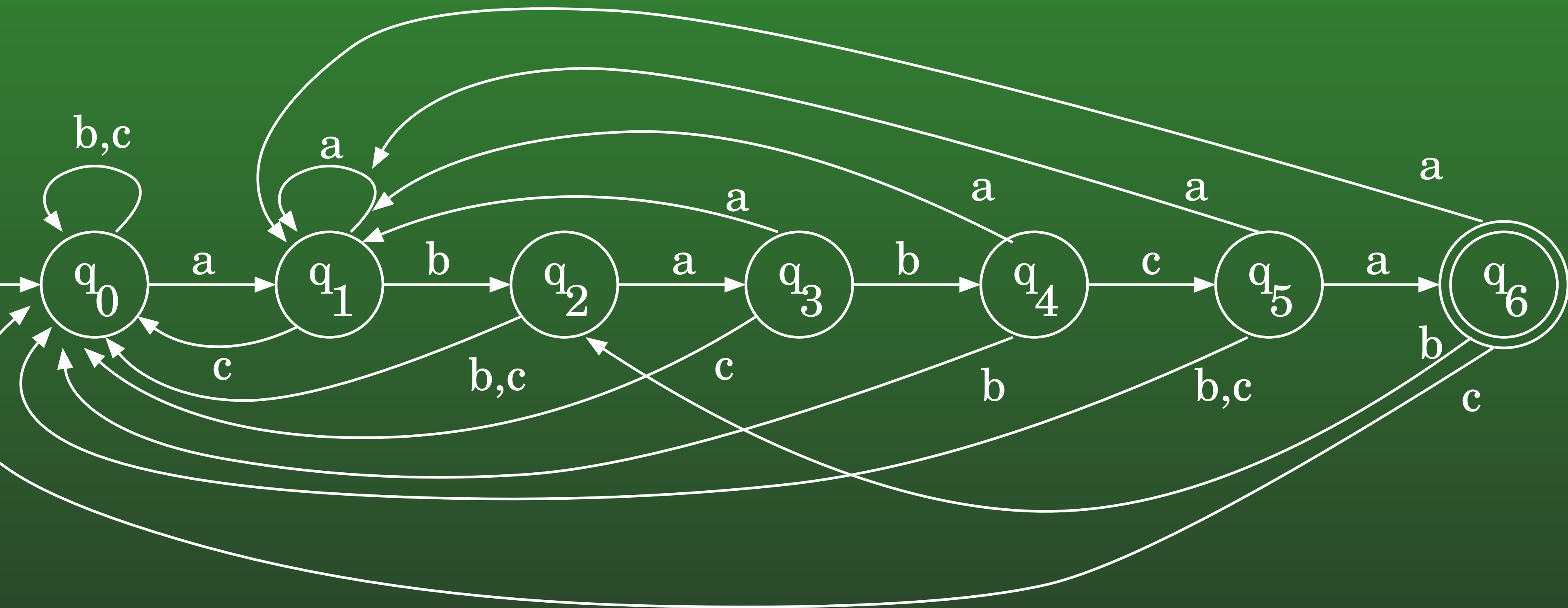
20-35: DFA

- $P = ababca$
- $P_5 : ababc$
  - $P_5a = ababca$: q_6 $\delta(q_5, a) = q_6$
  - $P_5b = ababcb$: q_0 $\delta(q_5, b) = q_0$
  - $P_5c = ababcc$: q_0 $\delta(q_5, c) = q_0$

20-36: DFA

- $P = ababca$
- $P_6 : ababca$
    - $P_6a = ababcaa$: q_1 $\delta(q_6, a) = q_1$
    - $P_6b = ababcab$: q_2 $\delta(q_6, b) = q_2$
    - $P_6c = ababcac$: q_0 $\delta(q_6, c) = q_0$

20-37: DFA



20-38: DFA

- Running time:
  - Time to build DFA: $O(m^3 * |\Sigma|)$
    - (Can be improved to $O(m * |\Sigma|)$
  - Time to run string through DFA: $O(n)$
- Total: $O(m^3 * |\Sigma| + n)$

Knuth-Morris-Pratt

- New algorithm: Knuth-Morris-Pratt
- Same $O(n)$ matching time through the string as DFA
- Smaller preprocessing time $O(m)$, amortized

Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

a	b	a	b	a	a	b	b	a
0	0	1	2	3	1	2	0	1

20-41: Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

a	b	a	b	a	a	b	b	a			
0	0	1	2	3	1	2	0	1			
a	b	a	b	a	a	b	b	a			
		a	b	a	b	a	a	b	b	a	

20-42: Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

```
a   b   a   b   a   a   b   b   a
0   0   1   2   3   1   2   0   1

a   b   a | b | a   a   b   b   a
        a | b | a   b   a   a   b   b   a
```

Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

a	b	a	b	a	a	b	b	a				
0	0	1	2	3	1	2	0	1				

```
a  b  a  b | a | a  b  b  a
      a  b | a | b  a  a  b  b  a
```

Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

a	b	a	b	a	a	b	b	a
0	0	1	2	3	1	2	0	1

a b a b a a b b a

a b a b a a b b a

Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

a	b	a	b	a	a	b	b	a								
0	0	1	2	3	1	2	0	1								
a	b	a	b	a	a	b	b	a								
								a	b	a	b	a	a	b	b	a

Knuth-Morris-Pratt

- Maximum overlap array
  - How much can the string overlap with itself at each position?

a	b	a	b	a	a	b	b	a								
0	0	1	2	3	1	2	0	1								
a	b	a	b	a	a	b	b	a								
								a	b	a	b	a	a	b	b	a

20-47: Knuth-Morris-Pratt

- Prefix Function π:
  - $\pi[q] = max\{k : k < q \&\& P_k$ =] $P_q\}$
  - $\pi[q]$ is the length of the longest prefix of P that is a proper suffix of P_q

20-48: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
a	b	a	b	b												

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
a	b	a	b	b												

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
a	b	a	b	b												

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
a	b	a	b	b												

20-52: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
a	b	a	b	b												

Letter Mismatch

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
		a	b	a	b	b										

20-54: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a | b | b a b b a b a b a b b

a b a | b | b

20-55: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
		a	b	a	b	b										

Complete Match!

20-56: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
							a	b	a	b	b					

20-57: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a b b a | b | b a b a b a b b

a | b | a b b

20-58: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a b b a b | b | a b a b a b b

a b | a | b b

Letter Mismatch

20-59: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a b b a b | b | a b a b a b b

a b a b b

20-60: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a b b a b b | a | b a b a b b

a | b a b b

20-61: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
										a	b	a	b	b		

20-62: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a b b a b b a b a b a b b

a b a b b

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a b b a b b a b a | b | a b b

a b a | b | b

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
										a	b	a	b	b		

Letter Mismatch

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a b b a b b a b a b | a | b b

a b | a | b b

20-66: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a b a b a b b a b b a b a b a | b | b

a b a | b | b

20-67: Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: ababb

π:

a	b	a	b	b
0	0	1	2	0

Input String:

a	b	a	b	a	b	b	a	b	b	a	b	a	b	a	b	b
												a	b	a	b	b

Complete Match

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

π:

a	b	a	b
0	0	1	2

Input String:

a	b	a	b	a	b	a	b
a	b	a	b				

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

π:

a	b	a	b
0	0	1	2

Input String:

a	b	a	b	a	b	a	b
a	b	a	b				

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

π:

a	b	a	b
0	0	1	2

Input String:

a	b	a	b	a	b	a	b
a	b	a	b				

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

π:

a	b	a	b
0	0	1	2

Input String:

a	b	a	b	a	b	a	b
a	b	a	b				

Complete Match

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

π:

a	b	a	b
0	0	1	2

Input String:

a	b	a	b	a	b	a	b
		a	b	a	b		

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

π:

a	b	a	b
0	0	1	2

Input String:

a	b	a	b	a	b	a	b
		a	b	a	b		

Complete Match

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

π:

a	b	a	b
0	0	1	2

Input String:

a	b	a	b	a	b	a	b
				a	b	a	b

Knuth-Morris-Pratt

- Try to match pattern to input
- When a mismatch occurs, the π array tells us how far to shift the pattern forward

Pattern: abab

π:

a	b	a	b
0	0	1	2

Input String:

a	b	a	b	a	b	a	b
				a	b	a	b

Complete Match

20-76: Knuth-Morris-Pratt

- Creating π array

$m \leftarrow$ length[P]
$\pi[1] \leftarrow 0$
$k \leftarrow 0$
for $q \leftarrow 2$ to m do
    while $k > 0$ and $P[k+1] \neq P[q]$
        $k \leftarrow \pi[k]$
    if $P[k+1] = P[q]$
        $k \leftarrow k+1$
    $\pi[q] \leftarrow k$

Knuth-Morris-Pratt

KMP-Matching(T, P)
    $m \leftarrow$ length[P]
    $n \leftarrow$ length[T]
    $\pi \leftarrow$ ComputePI(P)
    $q \leftarrow 0$
    for $i \leftarrow 1$ to n do
        while $q > 0$ and $P[q+1] \neq T[i]$
            $q \leftarrow \pi[q]$
        if $P[q+1] = T[i]$
            $q \leftarrow q + 1$
        if $q = m$
            Print "Match found at" $i - m$
            $q \leftarrow \pi[q]$

20-78: Knuth-Morris-Pratt

- Running time:
  - Preprocessing time: $\Theta(m)$
    - Using amortized analysis (aggregate)
  - Running time: $\Theta(n)$
    - Using amortized analysis (aggregate)