String Matching

Inge Li Gørtz

CLRS 32

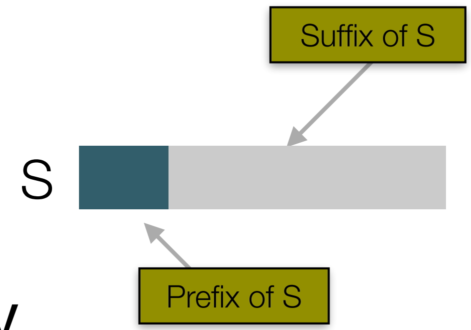
String Matching

- String matching problem:
  - string T (text) and string P (pattern) over an alphabet Σ.
  - $|T| = n$, $|P| = m$.
  - Report all starting positions of occurrences of P in T.

P = a b a b a c a

T = b a c b a b a b a b a b a c a b

Strings



- ε: empty string
- prefix/suffix: v=xy:
  - x *prefix* of v, if y ≠ ε x is a *proper prefix* of v
  - y *suffix* of v, if y ≠ ε x is a *proper suffix* of v.
- Example: S = `aabca`
  - The suffixes of S are: `aabca`, `abca`, `bca`, `ca` and `a`.
  - The strings `abca`, `bca`, `ca` and `a` are proper suffixes of S.

String Matching

- Knuth-Morris-Pratt (KMP)
- Finite automaton

A naive string matching algorithm

b	a	c	b	a	b	a	b	a	b	a	b	a	c	a	b
a	b	a	b	a	c	a									
	a	b	a	b	a	c	a								
		a	b	a	b	a	c	a							
			a	b	a	b	a	c	a						
				a	b	a	b	a	c	a					
					a	b	a	b	a	c	a				
						a	b	a	b	a	c	a			
							a	b	a	b	a	c	a		
								a	b	a	b	a	c	a	
									a	b	a	b	a	c	a

Improving the naive algorithm

P = a a a b a b a

T =	a	a	a	b	a	a	a	b	a	b	a	b	a	c	a	b	b
	a	a	a	b	a	b	a										

Improving the naive algorithm

P = a a a b a b a

T = a a a b a [redacted]

a a a b a b a

a a a a a b b a b a

Improving the naive algorithm

P = a a a b a b a

T = a a a b a a a [redacted]

a a a b a b a

a a a b a b a

a a a b a b a

Improving the naive algorithm

P = a a a b a b a

T = a a a b a a a a b a b a [redacted]

a a a b a b a

a a a b a b a

a a a b a b a

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

Improving the naive algorithm

P = a a a b a b a

T = a a a b a a a a b a b a [redacted]

a a a b a b a

a a a b a b a

a a a b a b a

a a a b a b a

If we matched 5 characters from P and then fail: compare failed character to 2nd character in P

If we matched 3 characters from P and then fail: compare failed character to 3nd character in P

If we matched all characters from P: compare next character to 2nd character in P

Improving the naive algorithm

P = a a a b a b a

matched		a	a	a	b	a	b	a
#matched	0	1	2	3	4	5	6	7
if fail compare to				3		2		2

If we matched 5 characters from P and then fail: compare failed character to 2nd character in P

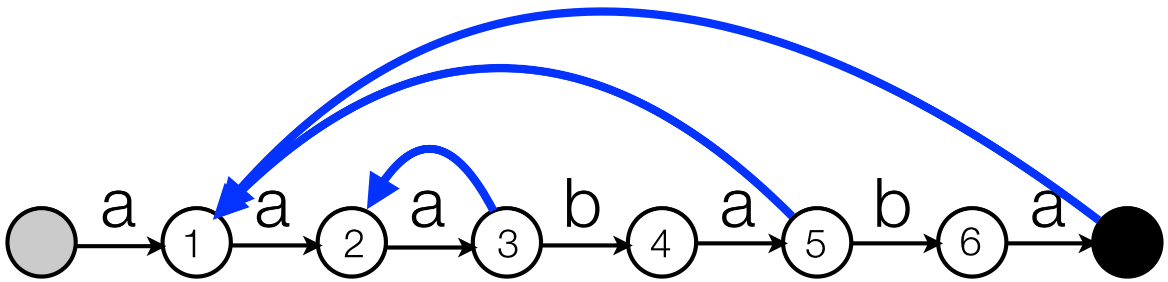
If we matched 3 characters from P and then fail: compare failed character to 3nd character in P

If we matched all characters from P: compare next character to 2nd character in P

Improving the naive algorithm

P = a a a b a b a

matched		a	a	a	b	a	b	a
#matched	0	1	2	3	4	5	6	7
if fail compare to				3		2		2



If we matched 5 characters from T and then fail: compare failed character to 2nd character in P

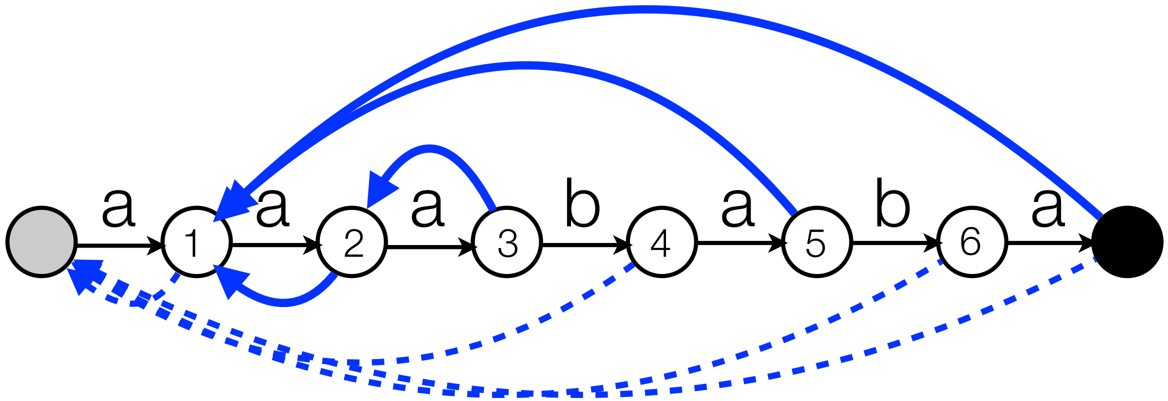
If we matched 3 characters from T and then fail: compare failed character to 3nd character in P

If we matched all characters from T: compare next character to 2nd character in P

Improving the naive algorithm

P = a a a b a b a

matched		a	a	a	b	a	b	a
#matched	0	1	2	3	4	5	6	7
if fail compare to	1	1	2	3	1	2	1	2

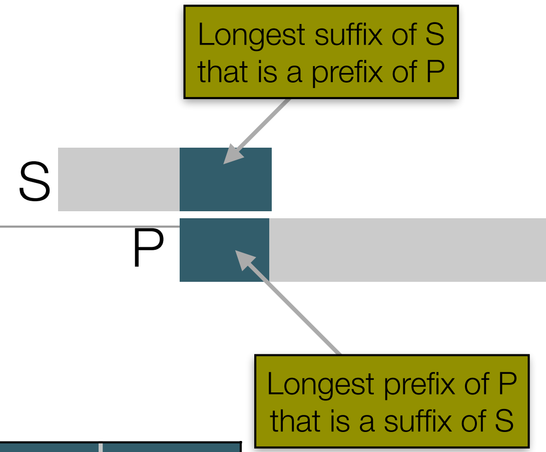


If we matched 5 characters from T and then fail: compare failed character to 2nd character in P

If we matched 3 characters from T and then fail: compare failed character to 3nd character in P

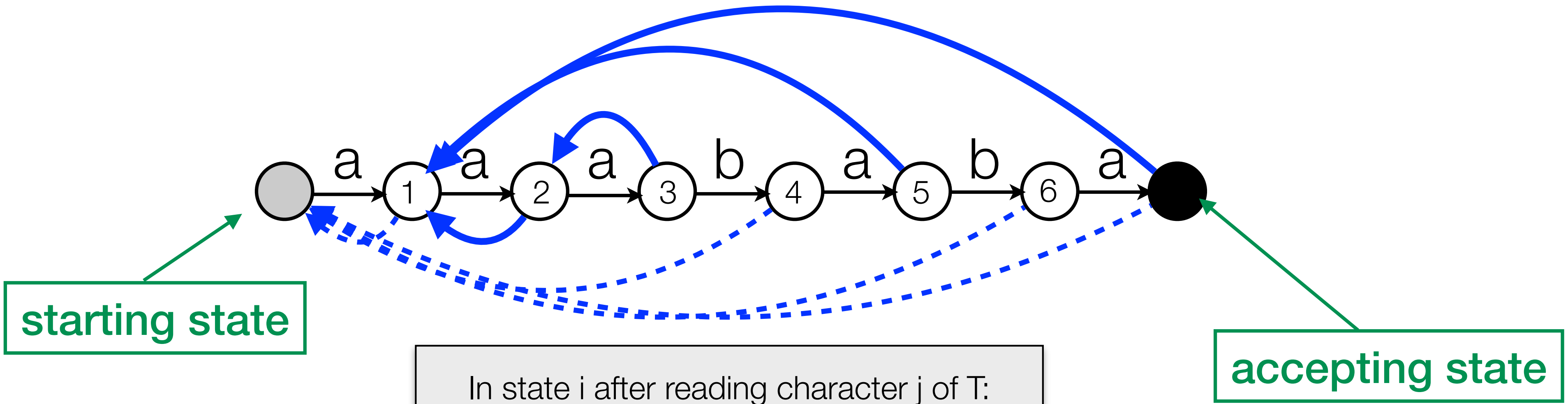
If we matched all characters from T: compare next character to 2nd character in P

Improving the naive algorithm



- KMP: P = aaababa.

matched		a	a	a	b	a	b	a
#matched	0	1	2	3	4	5	6	7
if fail *go to*	0	0	1	2	0	1	0	1

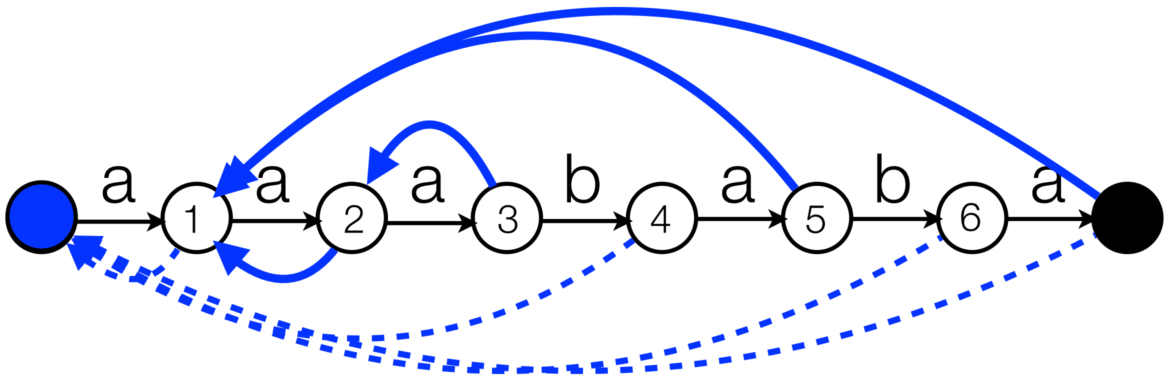


In state i after reading character j of T:
P[1…i] is the longest prefix of P that is a suffix of T[1..j]

Improving the naive algorithm

- KMP: P = aaababa.

matched		a	a	a	b	a	b	a
#matched	0	1	2	3	4	5	6	7
if fail *go to*	0	0	1	2	0	1	0	1

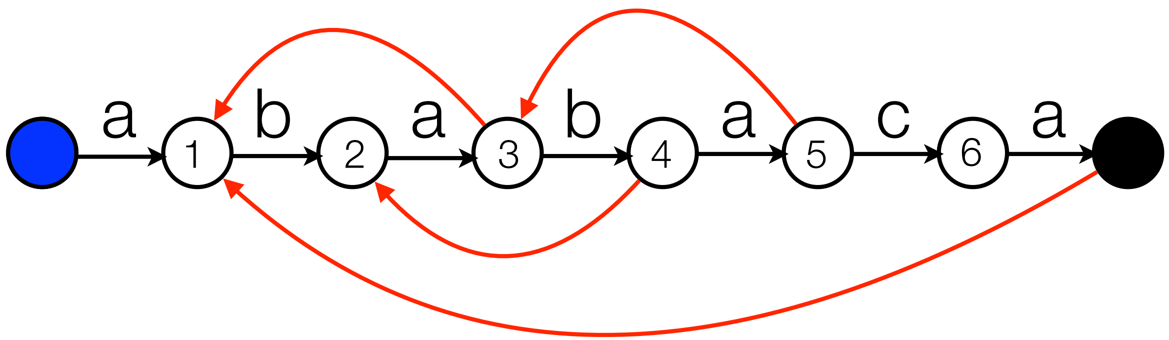


- Matching:

T = a a a b a a a b a b a a

KMP

- KMP: Can be seen as finite automaton with *failure links*:
  - Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.
  - In state i after reading T[j]: P[1..i] is the longest prefix of P that is a suffix of T[1…j].
  - Can follow several failure links when matching one character:



T = a b a b a a

KMP Analysis

- **Analysis.** $|T| = n$, $|P| = m$.
  - How many times can we follow a forward edge?
  - How many backward edges can we follow (compare to forward edges)?
  - Total number of edges we follow?
  - What else do we use time for?

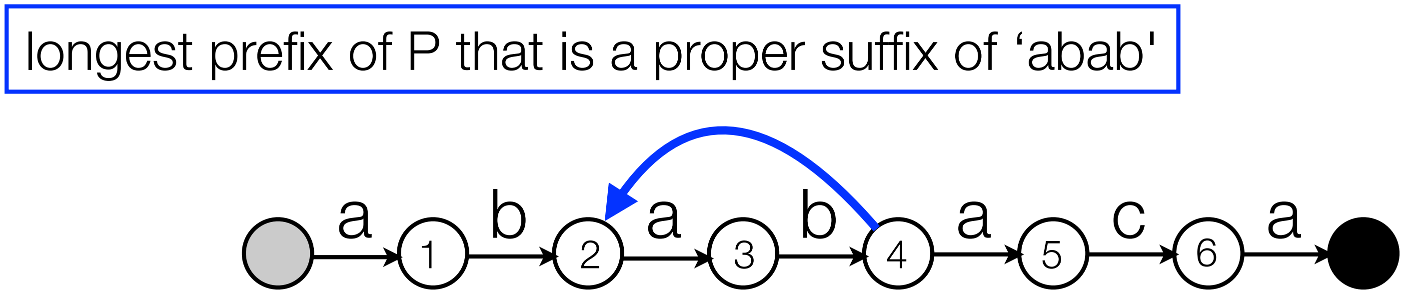
KMP Analysis

- Lemma. The running time of KMP matching is O(n).
  - Each time we follow a forward edge we read a new character of T.
  - #backward edges followed ≤ #forward edges followed ≤ n.
  - If in the start state and the character read in T does not match the forward edge, we stay there.
  - Total time = #non-matched characters in start state + #forward edges followed + #backward edges followed ≤ 2n.

Computation of failure links

- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.

- Computing failure links: Use KMP matching algorithm.

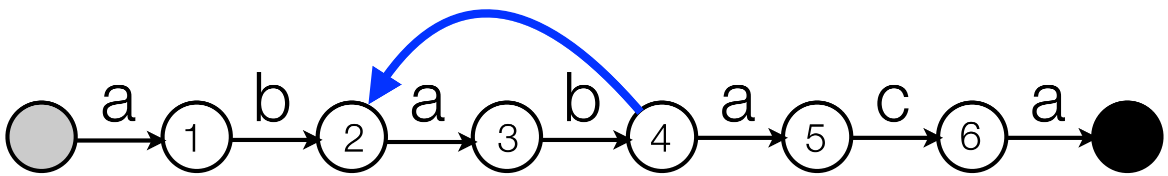


Computation of failure links

- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.

- Computing failure links: Use KMP matching algorithm.

longest prefix of P that is a suffix of ‘bab'

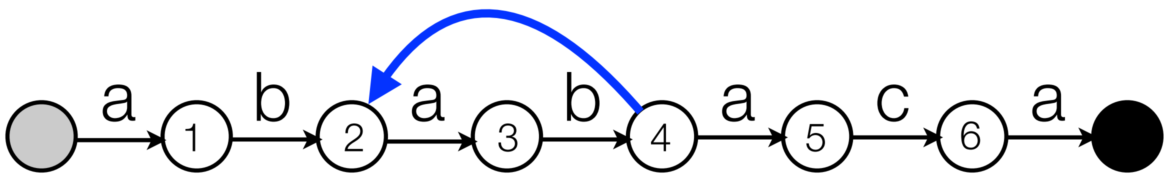


Computation of failure links

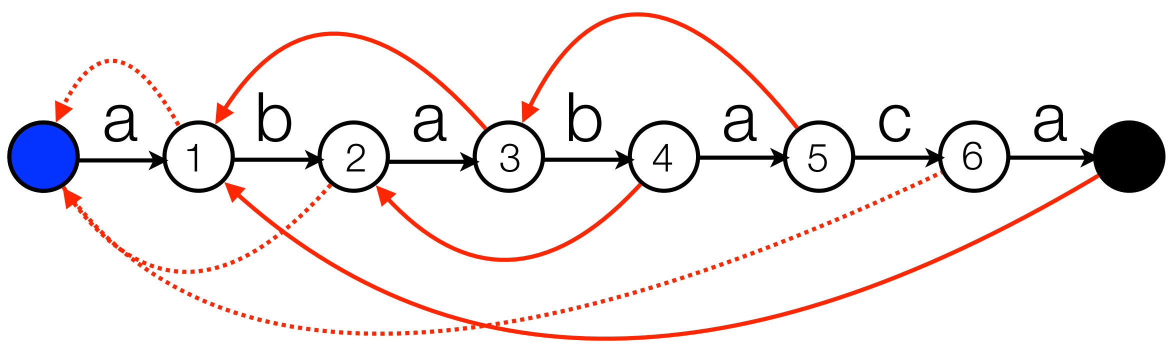
- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.

- Computing failure links: Use KMP matching algorithm.

longest prefix of P that is a suffix of ‘bab'

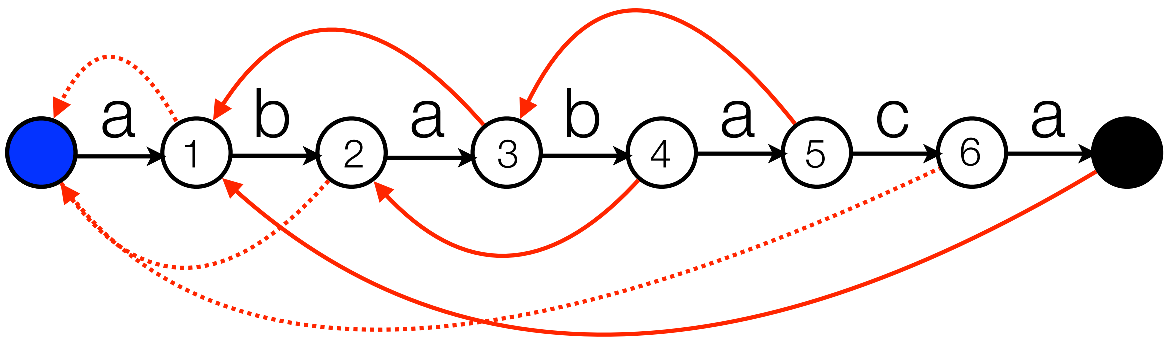


Can be found by using KMP to match ‘bab'



Computation of failure links

- Computing failure links: As KMP matching algorithm (only need failure links that are already computed).
- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.



	1	2	3	4	5	6	7
P =	$\boxed{a}$	b	a	b	a	c	a

Need to match: a, ab, aba, abab, ababa, ababac, ababaca

KMP

- Computing π: As KMP matching algorithm (only need π values that are already computed).
- Running time: O(n + m):
  - Lemma. Total number of comparisons of characters in KMP is at most 2n.
  - Corollary. Total number of comparisons of characters in the preprocessing of KMP is at most 2m.

KMP

- Computing π: As KMP matching algorithm (only need π values that are already computed).

- Running time: O(n + m):

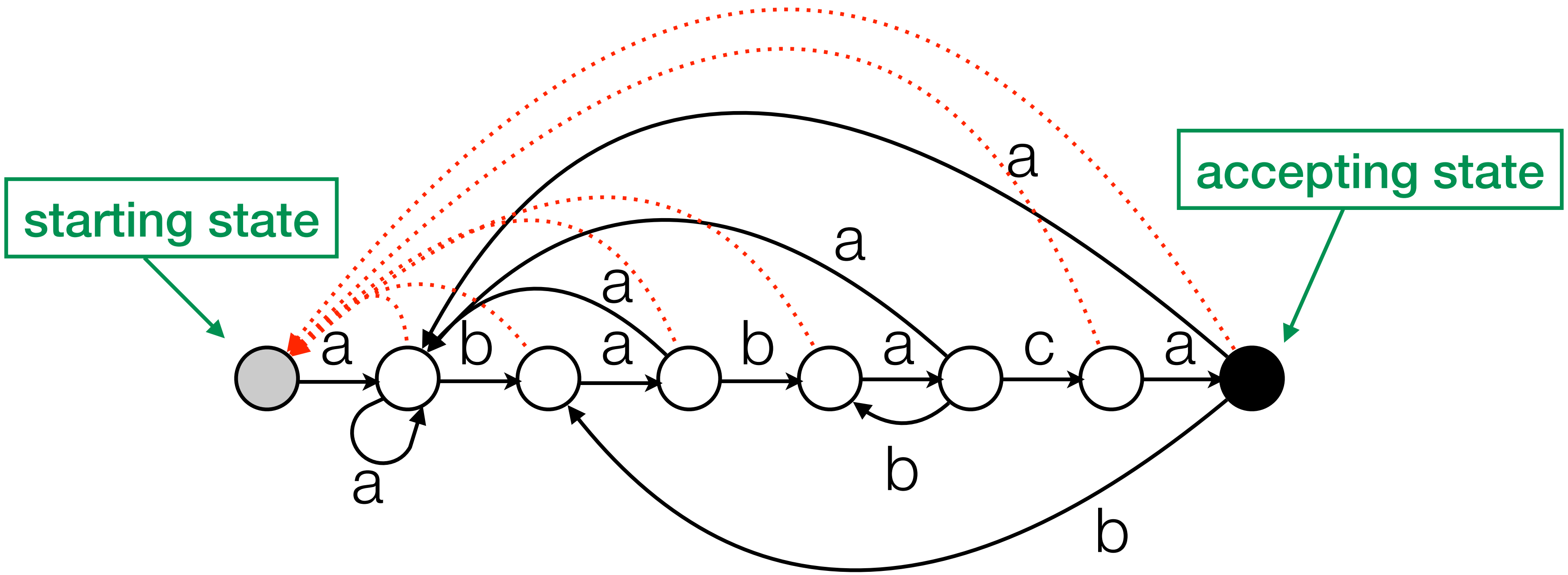
  - Lemma. Total number of comparisons of characters in KMP is at most 2n.

  - Corollary. Total number of comparisons of characters in the preprocessing of KMP is at most 2m.

Finite Automaton

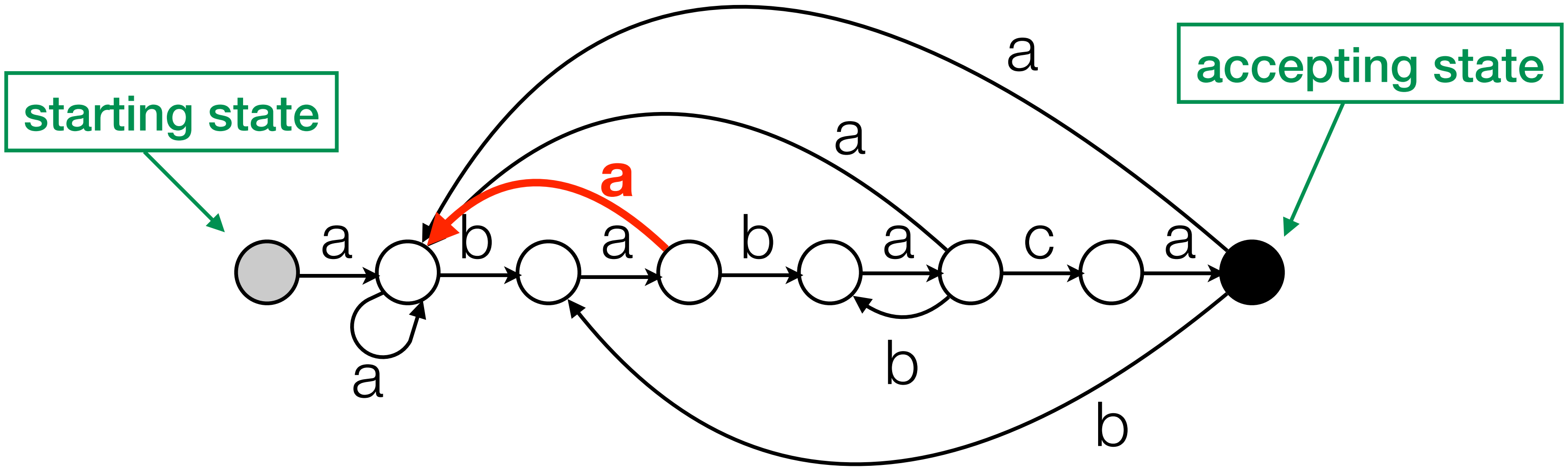
Finite Automaton

- Finite automaton: alphabet $\Sigma = \{a,b,c\}$. P = ababaca.



Finite Automaton

- Finite automaton: alphabet Σ = {a,b,c}. P= ababaca.



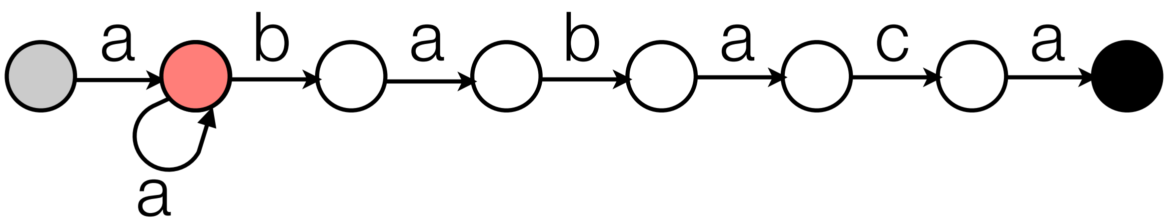
longest prefix of P that is a proper suffix of ‘abaa'

Matched until now: a b a a

P: a b a b a c a

Finite Automaton

- Finite automaton: alphabet $\Sigma = \{a,b,c\}$. P= ababaca.



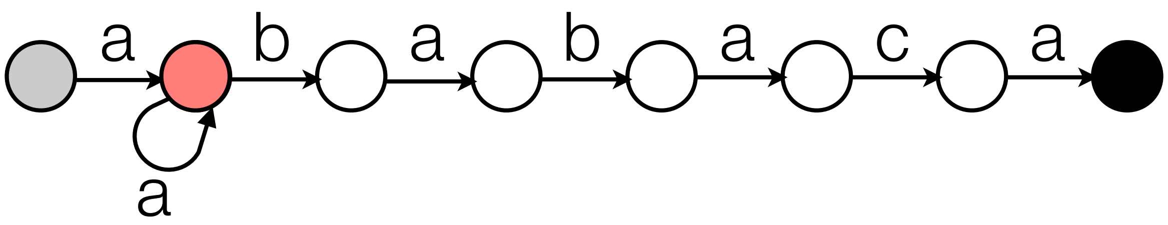
read 'a'?

longest prefix of P that is a proper suffix of 'aa' = 'a'

Matched until now:	a	a					
P:	a	b	a	b	a	c	a

Finite Automaton

- Finite automaton: alphabet Σ = {a,b,c}. P= ababaca.



read ‘c’?

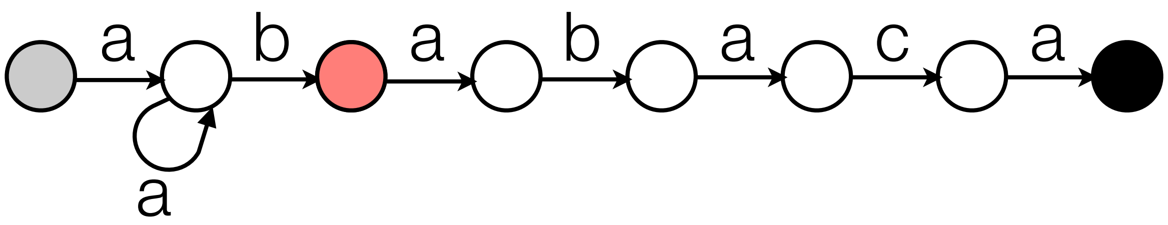
longest prefix of P that is a proper suffix of ‘ac’ = ‘ ’

Matched until now: a c

P: a b a b a c a

Finite Automaton

- Finite automaton: alphabet Σ = {a,b,c}. P= ababaca.



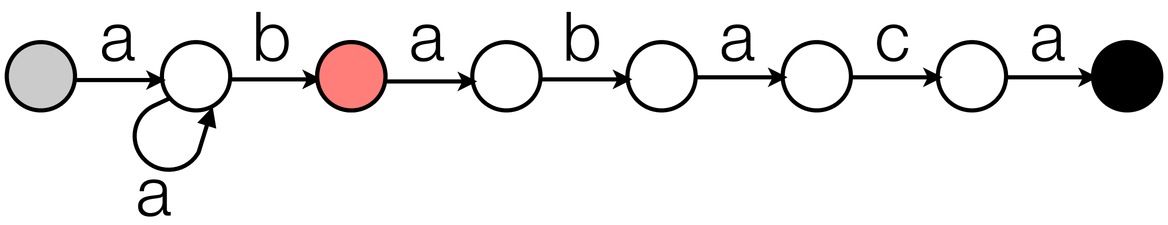
read ‘b’? **longest prefix of P that is a proper suffix of ‘abb’ = ‘ ’**

Matched until now: a b b

P: a b a b a c a

Finite Automaton

- Finite automaton: alphabet Σ = {a,b,c}. P= ababaca.



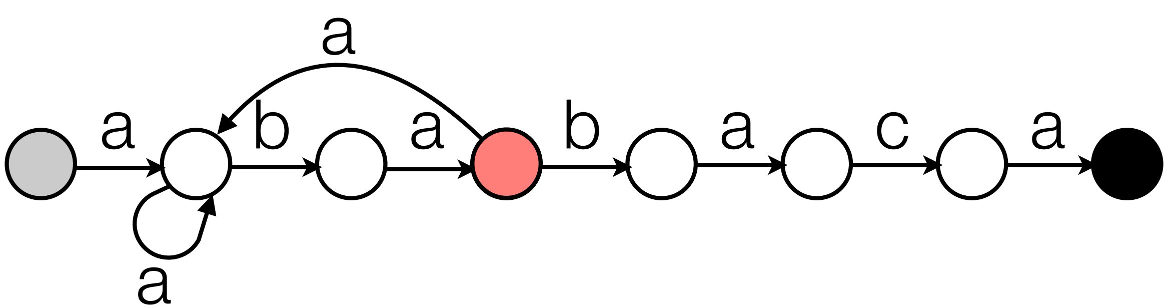
read ‘c’?

longest prefix of P that is a proper suffix of ‘abc’ = ‘ ’

Matched until now:	a	b	c				
P:	a	b	a	b	a	c	a

Finite Automaton

- Finite automaton: alphabet Σ = {a,b,c}. P= ababaca.



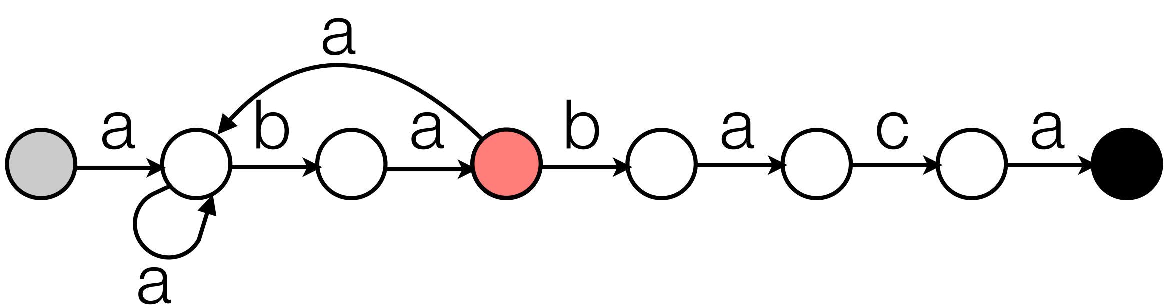
read ‘a’? **longest prefix of P that is a proper suffix of ‘abaa’ = ‘a’**

Matched until now: a b a a

P: a b a b a c a

Finite Automaton

- Finite automaton: alphabet Σ = {a,b,c}. P= abababaca.



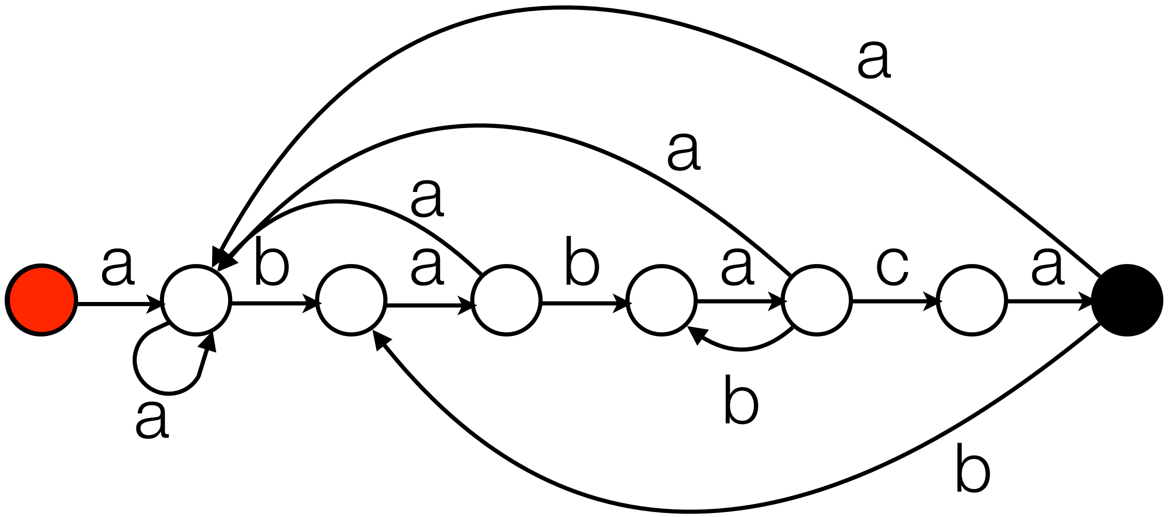
read ‘c’? **longest prefix of P that is a proper suffix of ‘abac’ = ‘ ’**

Matched until now: a b a c

P: a b a b a c a

Finite Automaton

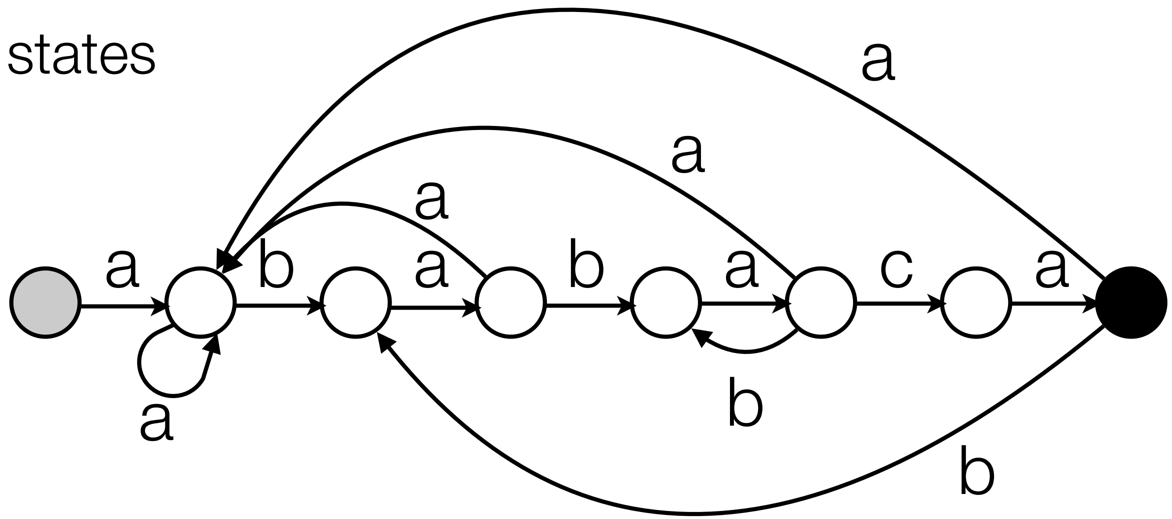
- Finite automaton: alphabet Σ = {a,b,c}. P = ababaca.



T = b a c b a b a b a b a b a c a b

Finite Automaton

- Finite automaton:
  - Q: finite set of states
  - $q_0 \in Q$: start state
  - $A \subseteq Q$: set of accepting states
  - Σ: finite input alphabet
  - δ: transition function



- Matching time: O(n)
- Preprocessing time: $O(m^3|\Sigma|)$. Can be done in $O(m|\Sigma|)$ using KMP.
- Total time: $O(n + m|\Sigma|)$