String Matching

Inge Li Gørtz

CLRS 32

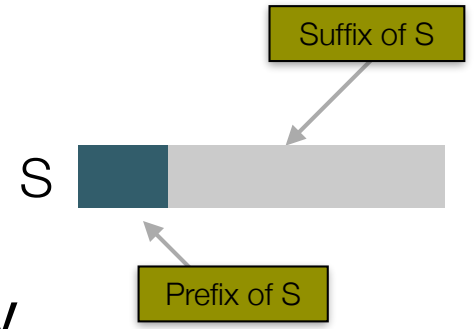
String Matching

- 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 c a b

Strings



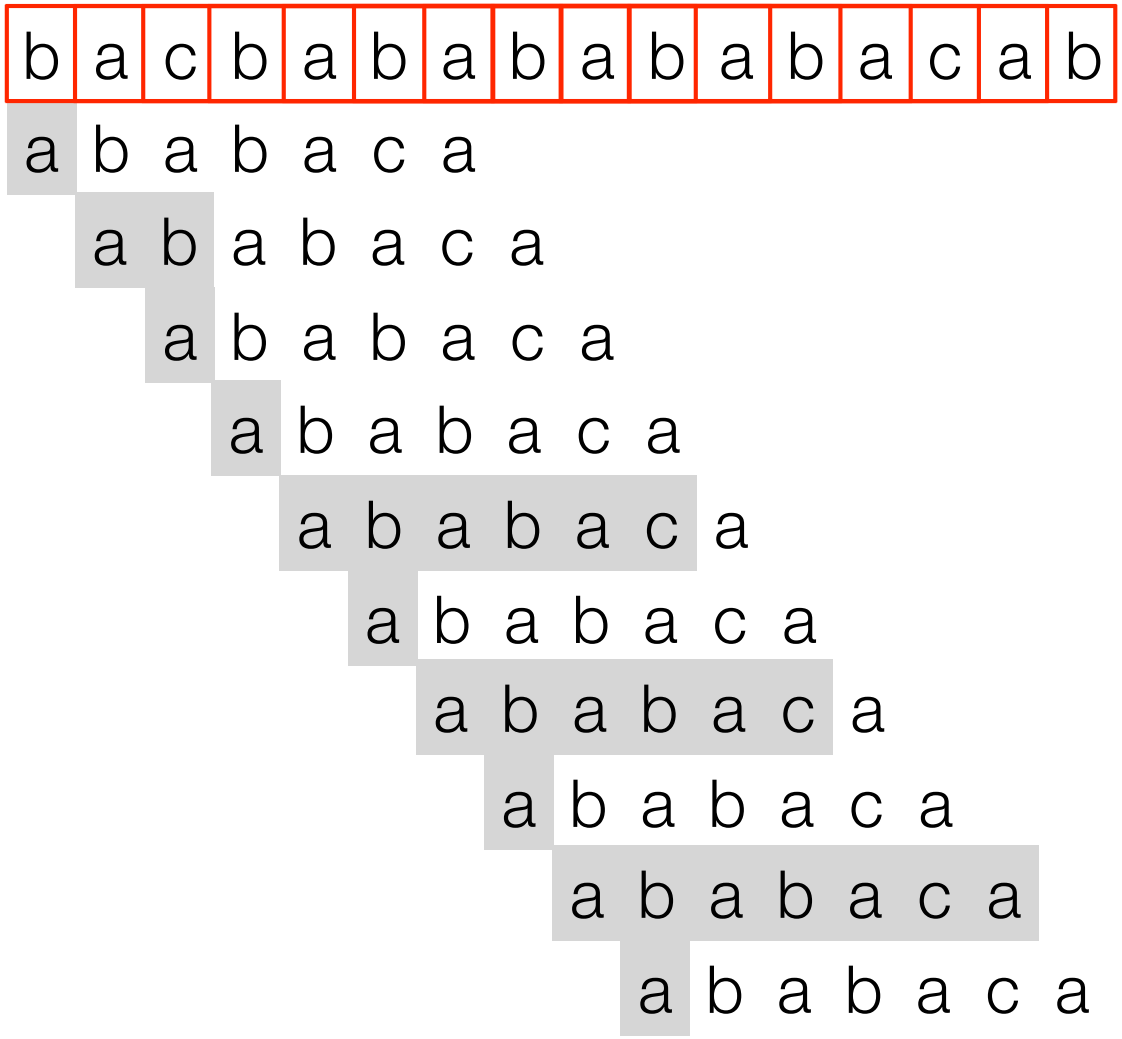
- ε: empty string
- prefix/suffix: v=xy:
  - x *prefix* of v, if y ≠ ε x is a *proper prefix* of v
  - y *suffix* of v, if x ≠ ε y is a *proper suffix* of v.
- Example: S = aabca
  - The suffixes of S are: aabca, abca, bca, ca and a.
  - The strings abca, bca, ca and a are proper suffixes of S.

String Matching

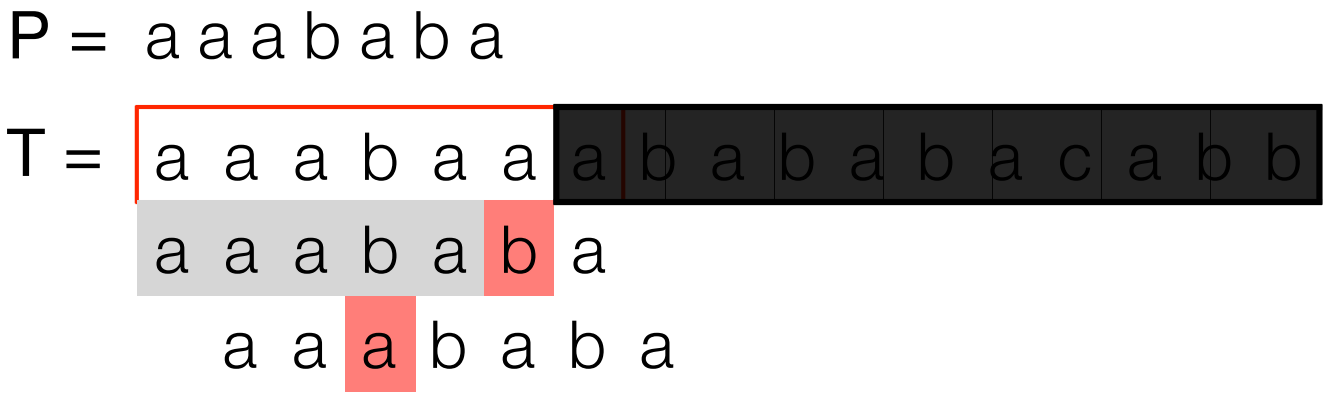
- Knuth-Morris-Pratt (KMP)

- Finite automaton

A naive string matching algorithm



Improving the naive algorithm



Exploiting what we know from pattern

P = a b a b a c a

T = a b a b a a

a b a b a c a

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

How much should we shift the pattern? 5

How much can we “reuse”? 1

T = a b a b a b

a b a b a c a

a b a b a c a

How much should we shift the pattern? 2

How much can we “reuse”? 4

T = a b a b a c

a b a b a c a

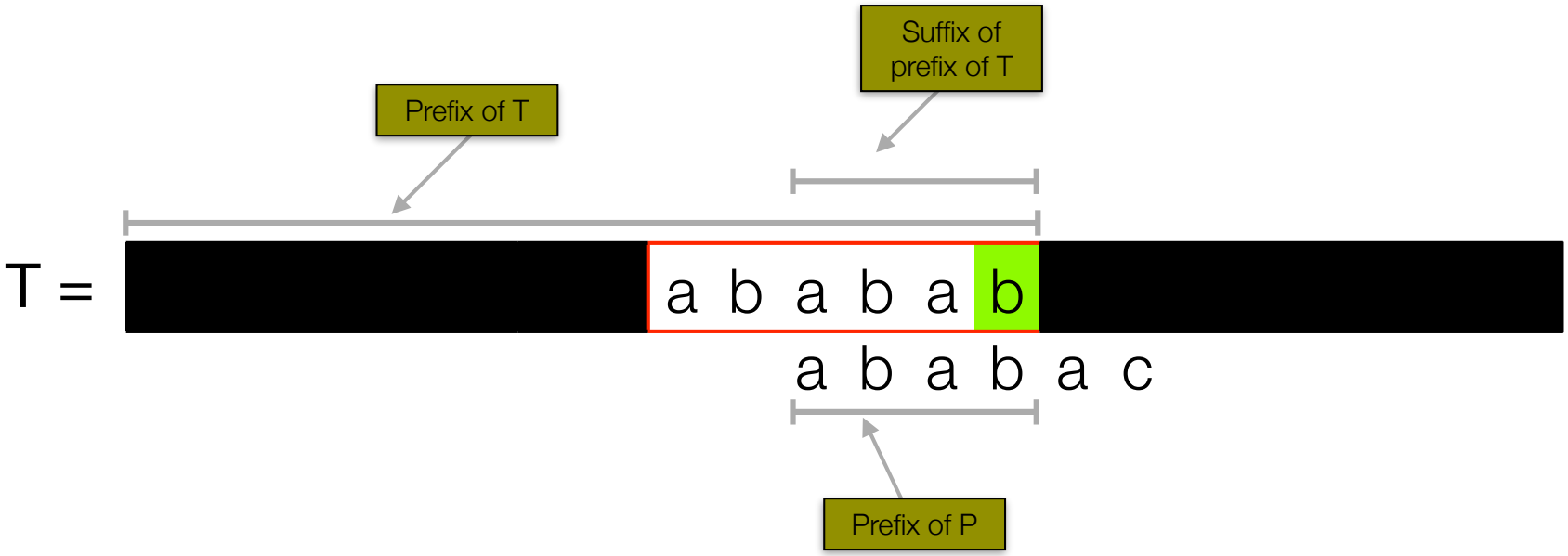
a b a b a c a

How much should we shift the pattern? 0

How much can we “reuse”? 6

Exploiting what we know from the pattern

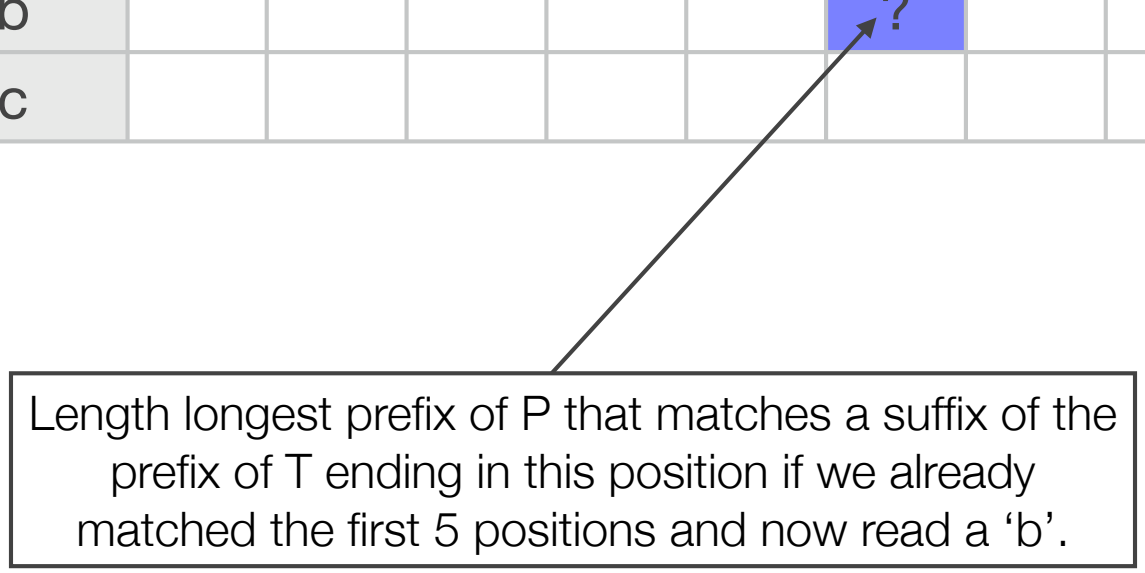
- Assume we matched the first 5 characters of our pattern and now read a ‘b’:
  - What is the longest prefix of P that matches a suffix of the prefix of T ending in this position? *(how much of the already matched part can we “reuse” in a new possible match).*



Fill the table

P = a b a b a c a

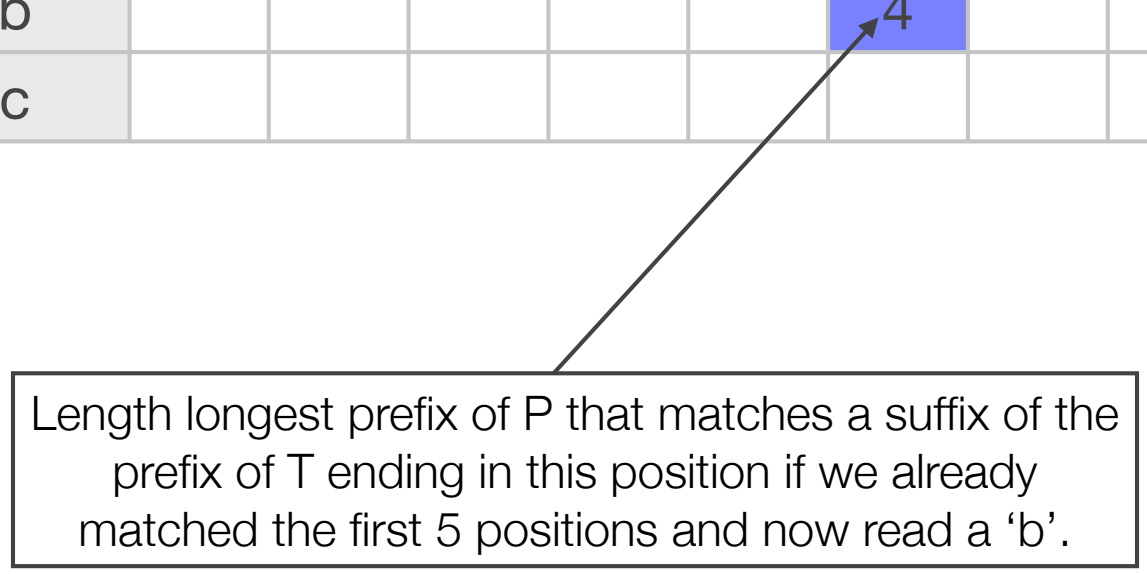
matched		a	b	a	b	a	c	a
#matched	0	1	2	3	4	5	6	7
a								
b						?		
c								



Fill the table

P = a b a b a c a

matched		a	b	a	b	a	c	a
#matched	0	1	2	3	4	5	6	7
a								
b						4		
c								

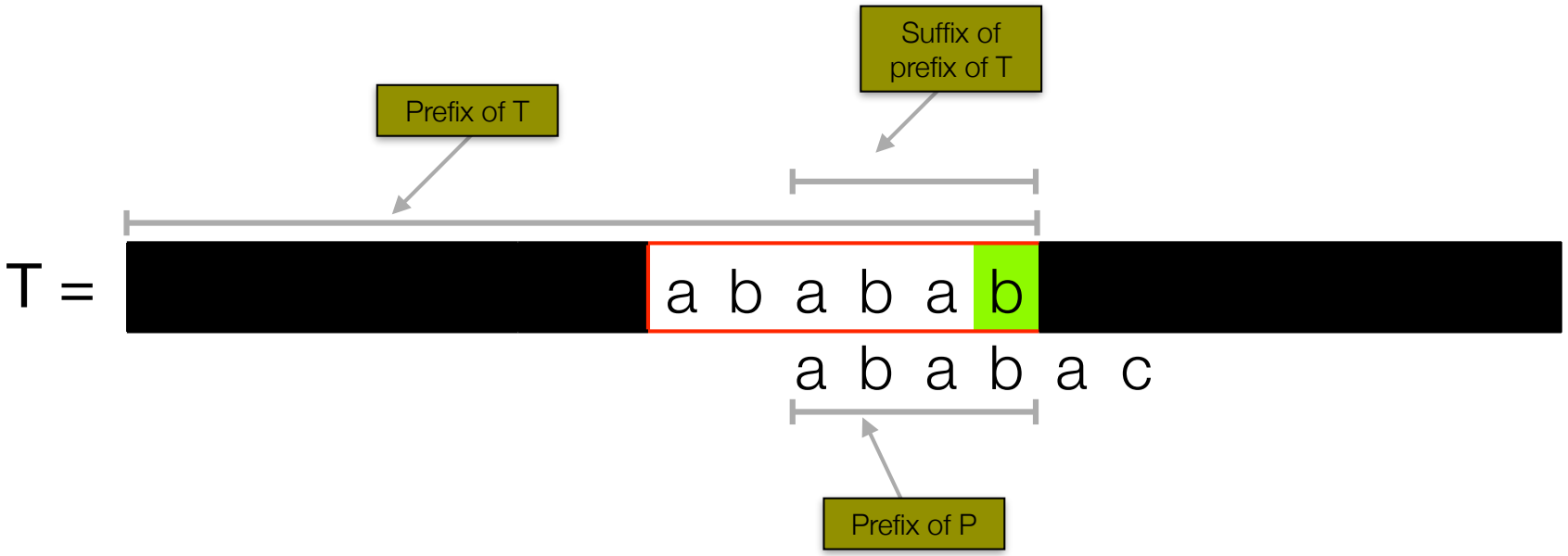


Exploring what we know from the pattern

Length longest prefix of P that matches a suffix of the prefix of T ending in this position if we already matched the first j positions and now read a ‘b’.

=

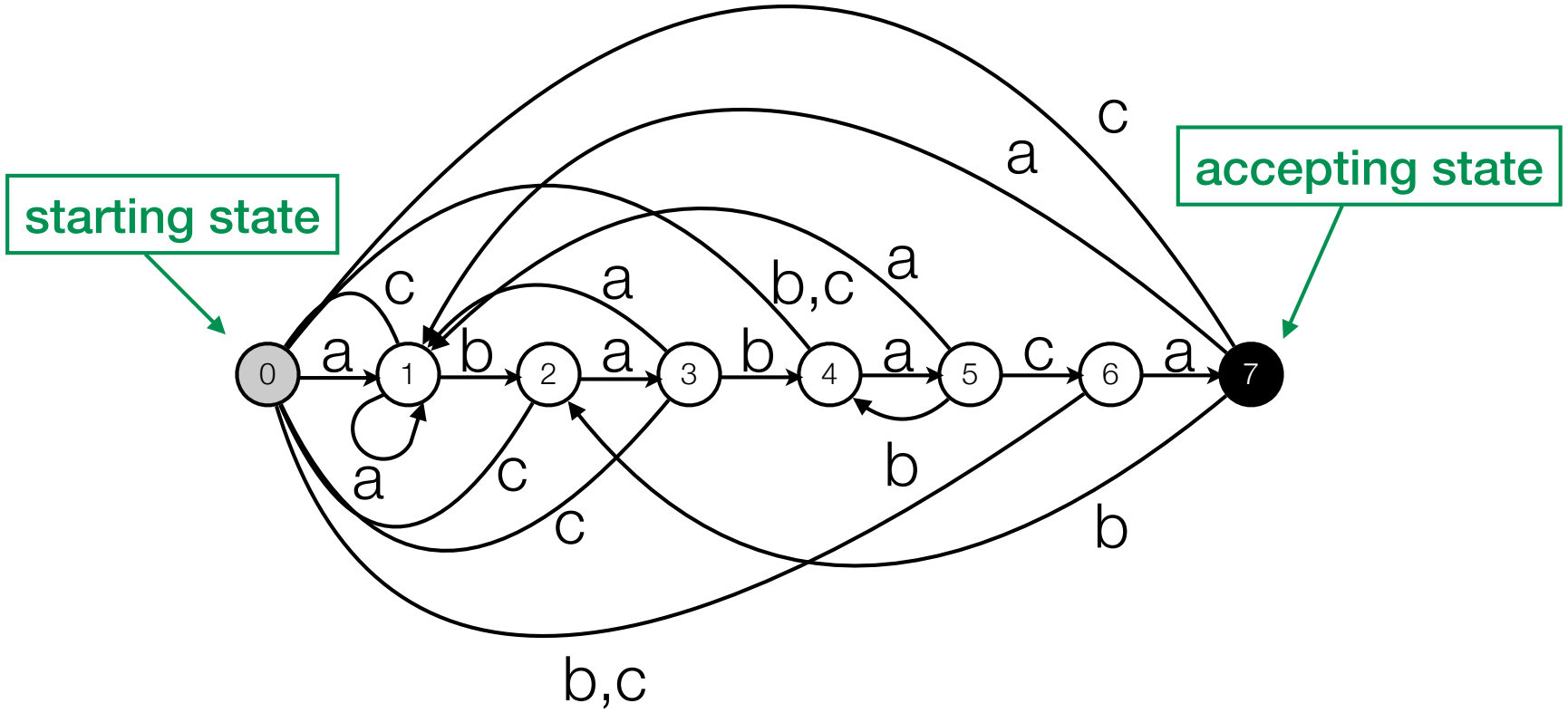
Largest i such that $P[1 \ldots i]$ is the longest prefix of P that is a suffix of $P[1 \ldots j] \cdot b$



Finite Automaton

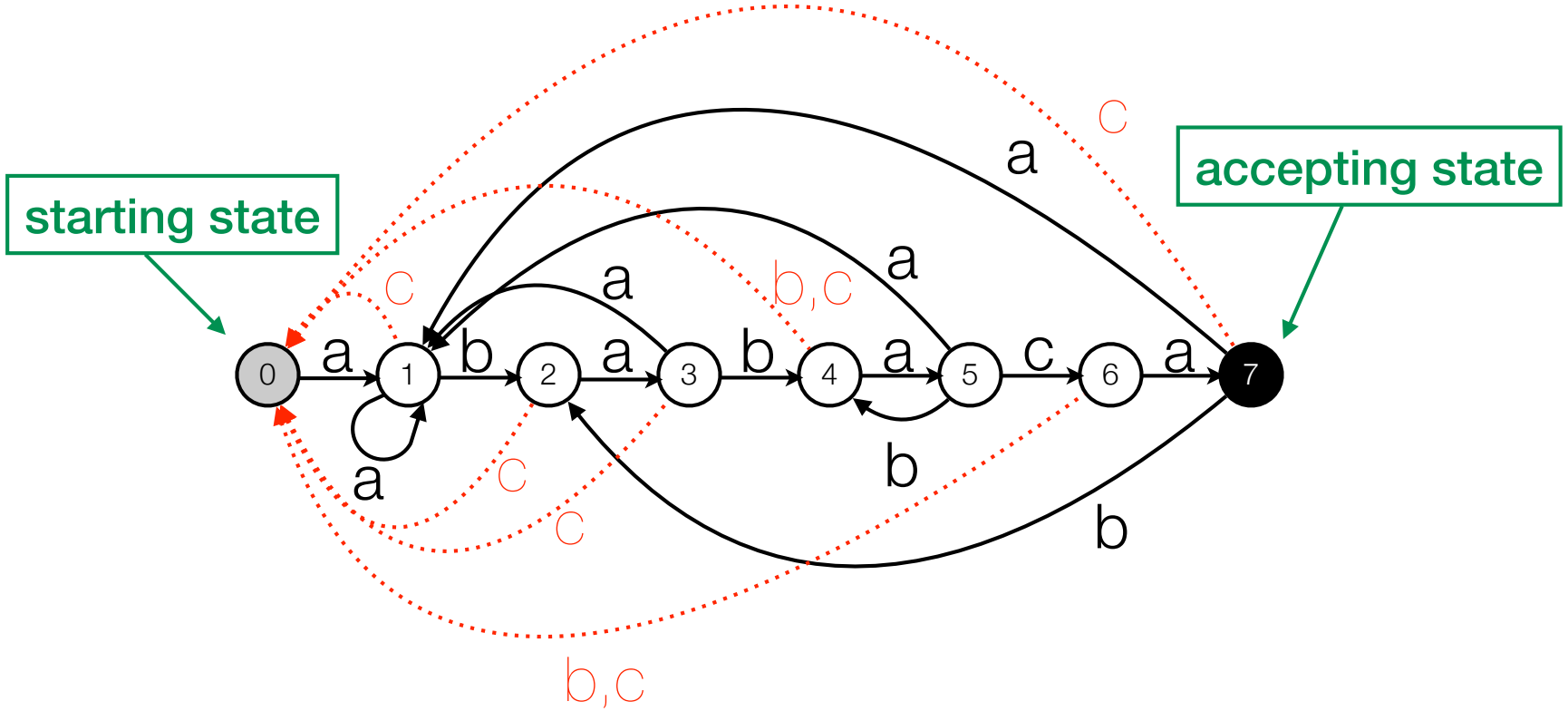
Finite Automaton

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



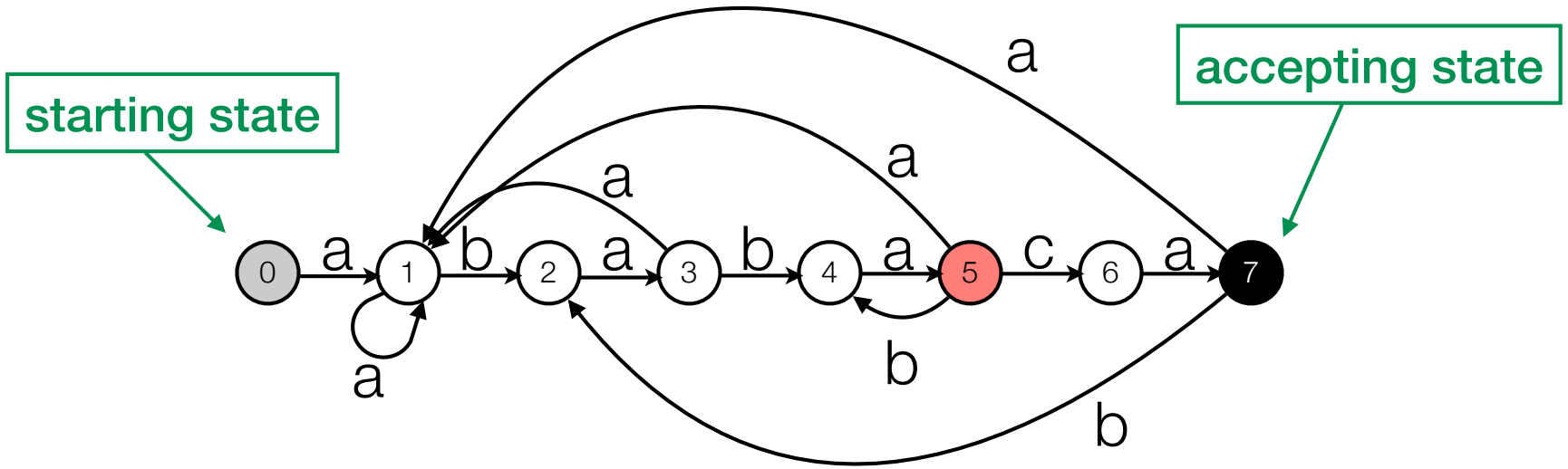
Finite Automaton

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



Finite Automaton

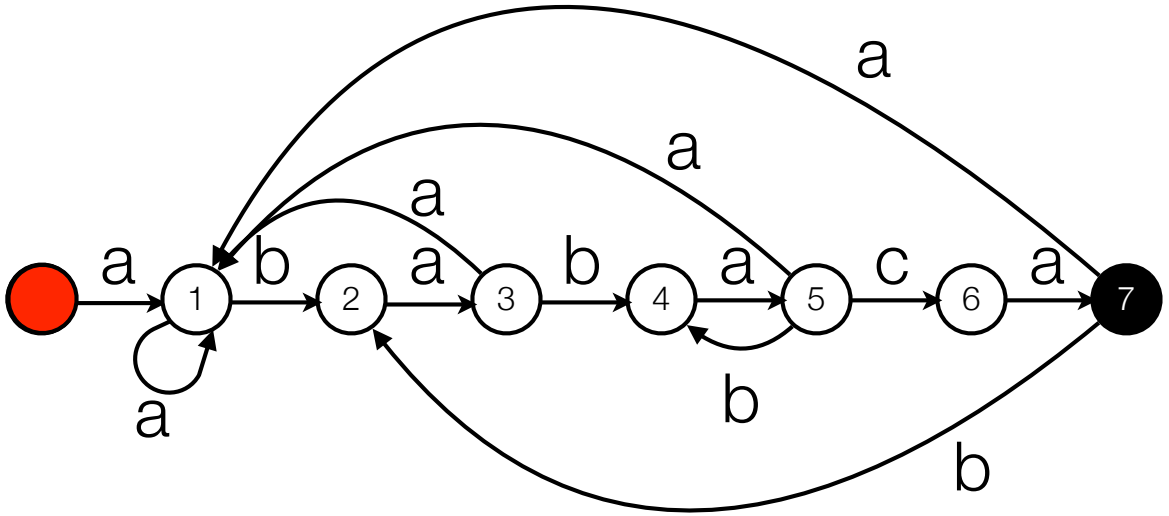
- Finite automaton: alphabet Σ = {a,b,c}. P = ababaca.



- State j: arc with character α goes to state i ≤ j +1 such that $P[1..i]$ is the longest prefix of P that is a suffix of $P[1...j] \cdot \alpha$.

Finite Automaton

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

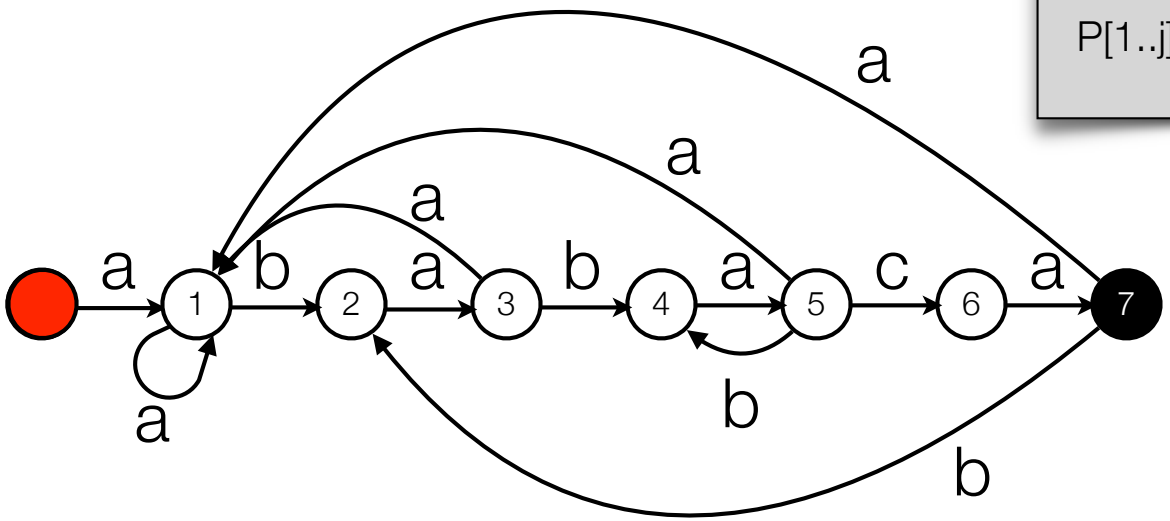


T = b a c b a b a b a b a b a c a b

Finite Automaton

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

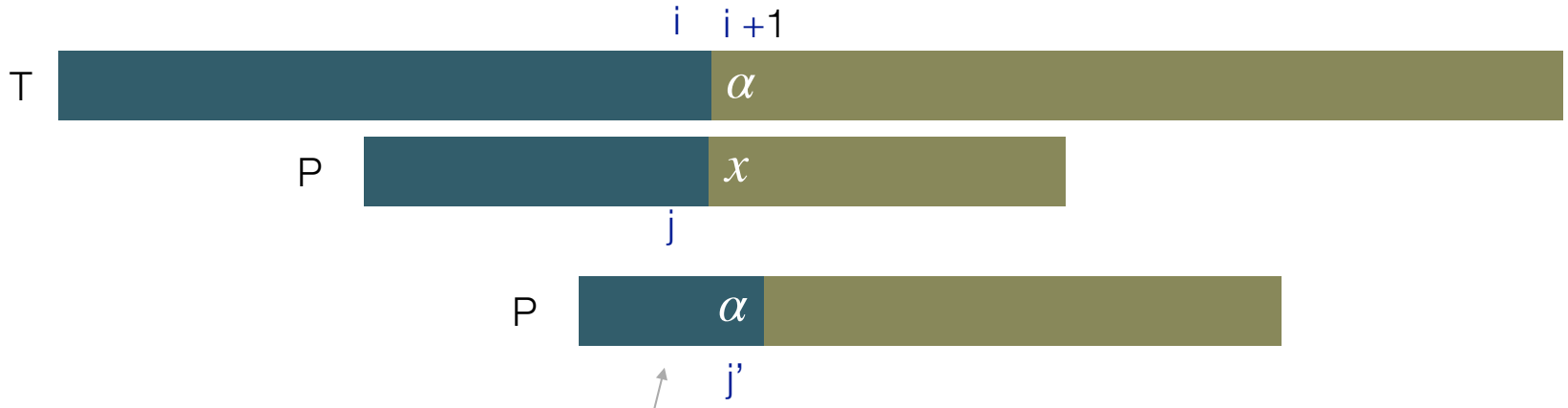
If we are in state j after reading T[1..i], then
P[1..j] is the longest prefix of P that is a suffix of T[1..i].



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

Finite Automaton

If we are in state j after reading T[1..i], then
P[1..j] is the longest prefix of P that is a suffix of T[1..i].



P[1..j'] longest prefix of P that is a suffix of T[1..i+1].

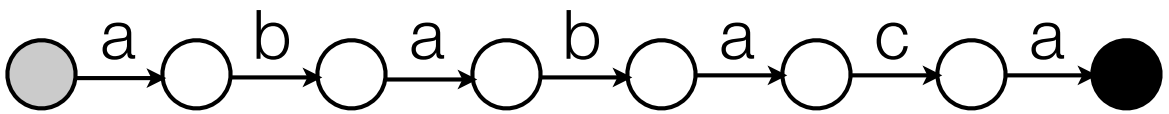
P[1..j'-1] is a suffix of P[1..j].

⇒

P[1..j'] longest prefix of P that is a suffix of P[1..j] · α.

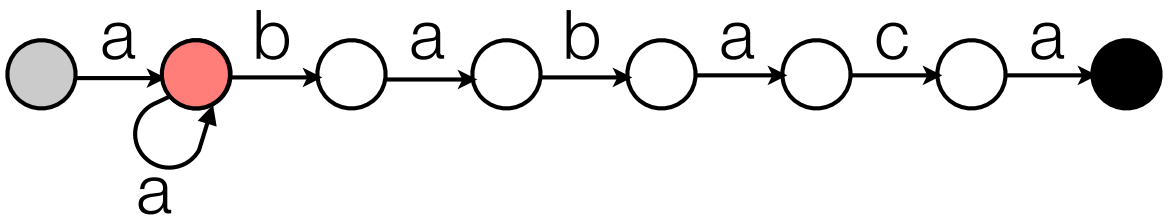
Finite Automaton Construction

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



Finite Automaton Construction

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



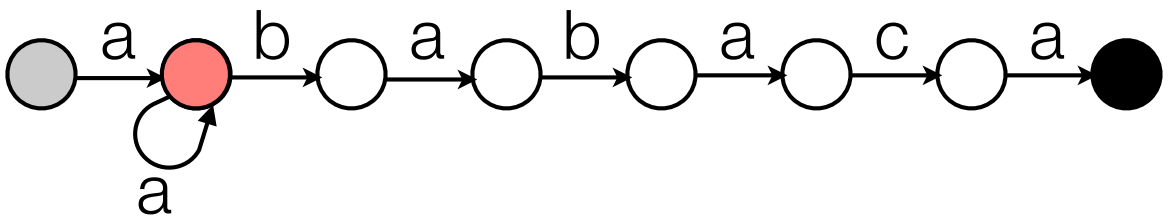
read ‘a’? **longest prefix of P that is a proper suffix of ‘aa’ = ‘a’**

Matched until now: a a

P: a b a b a c a

Finite Automaton Construction

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



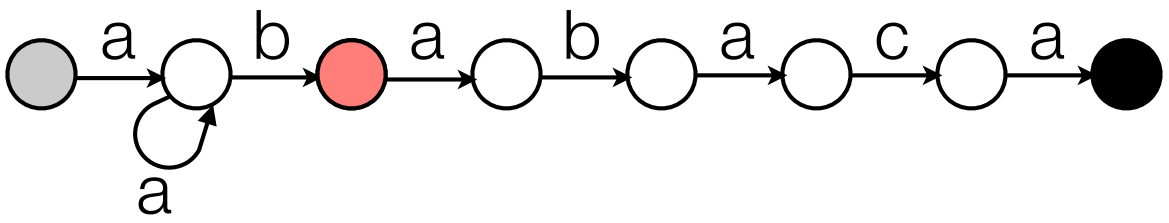
read ‘c’? **longest prefix of P that is a proper suffix of ‘ac’ = ‘ ’**

Matched until now: a c

P: a b a b a c a

Finite Automaton Construction

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



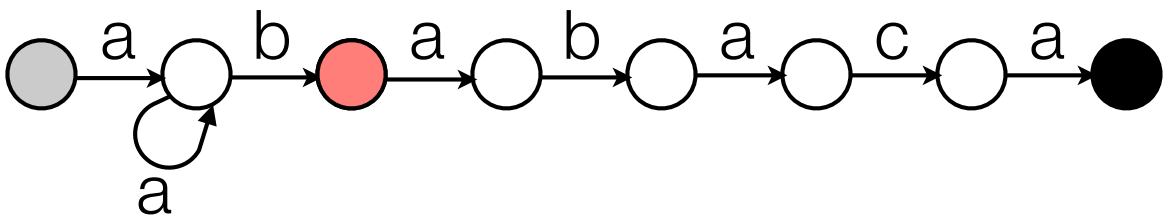
read ‘b’? **longest prefix of P that is a proper suffix of ‘abb’ = ‘ ’**

Matched until now: a b b

P: a b a b a c a

Finite Automaton Construction

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



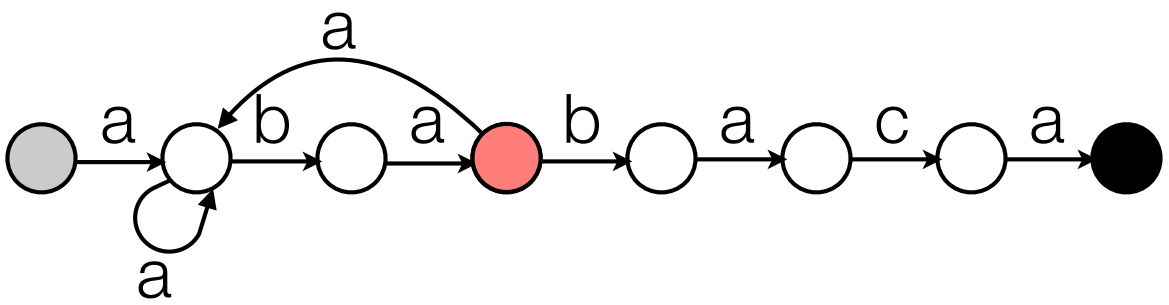
read ‘c’? **longest prefix of P that is a proper suffix of ‘abc’ = ‘ ’**

Matched until now: a b c

P: a b a b a c a

Finite Automaton Construction

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



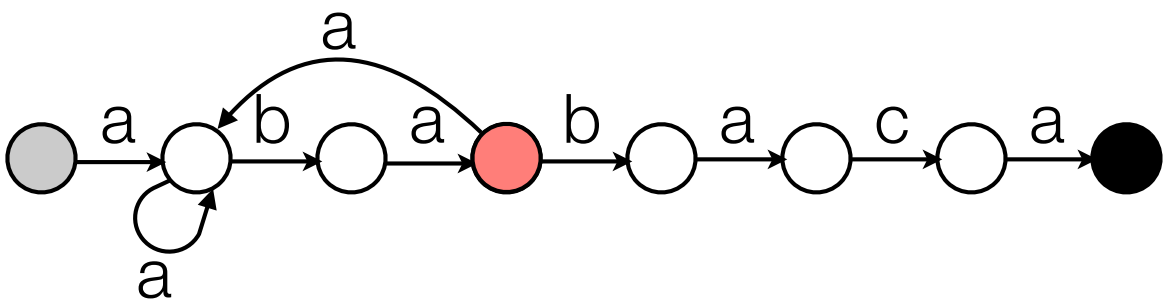
read ‘a’? **longest prefix of P that is a proper suffix of ‘abaa’ = ‘a’**

Matched until now: a b a a

P: a b a b a c a

Finite Automaton Construction

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



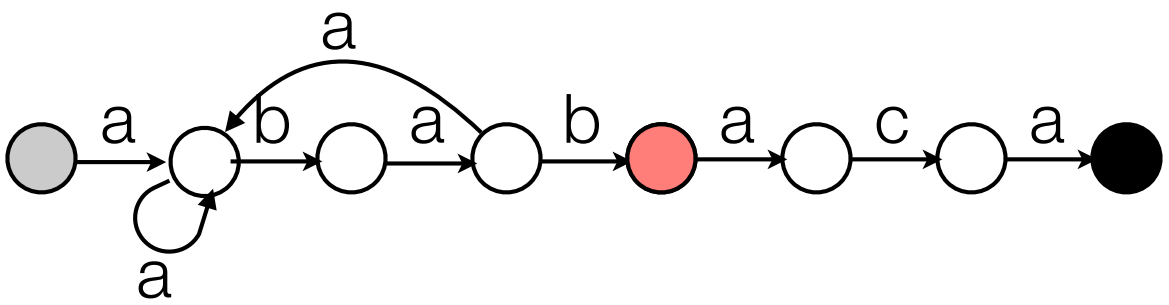
read ‘c’? **longest prefix of P that is a proper suffix of ‘abac’ = ‘ ’**

Matched until now: a b a c

P: a b a b a c a

Finite Automaton Construction

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



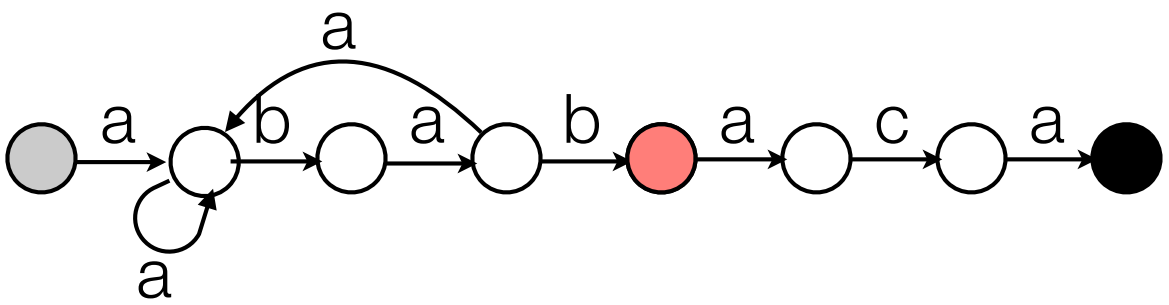
read ‘b’? **longest prefix of P that is a proper suffix of ‘ababb’ = ‘ ’**

Matched until now: a b a b b

P: a b a b a c a

Finite Automaton Construction

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



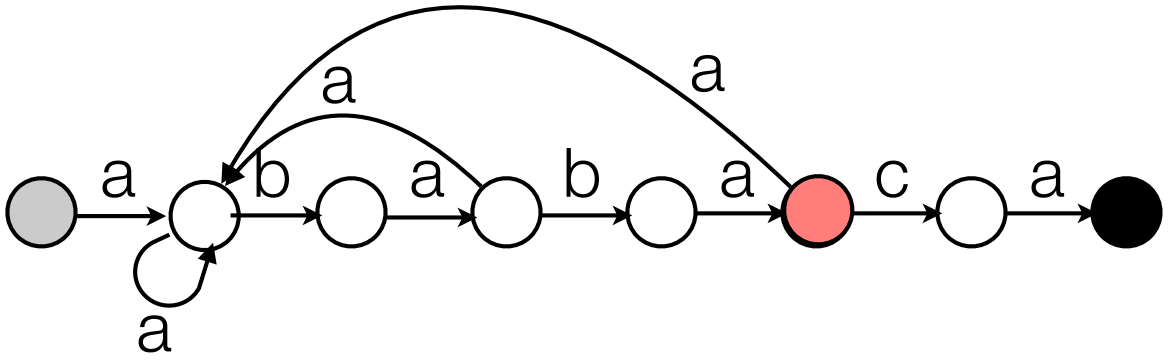
read ‘c’? **longest prefix of P that is a proper suffix of ‘ababc’ = ‘ ’**

Matched until now: a b a b c

P: a b a b a c a

Finite Automaton Construction

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

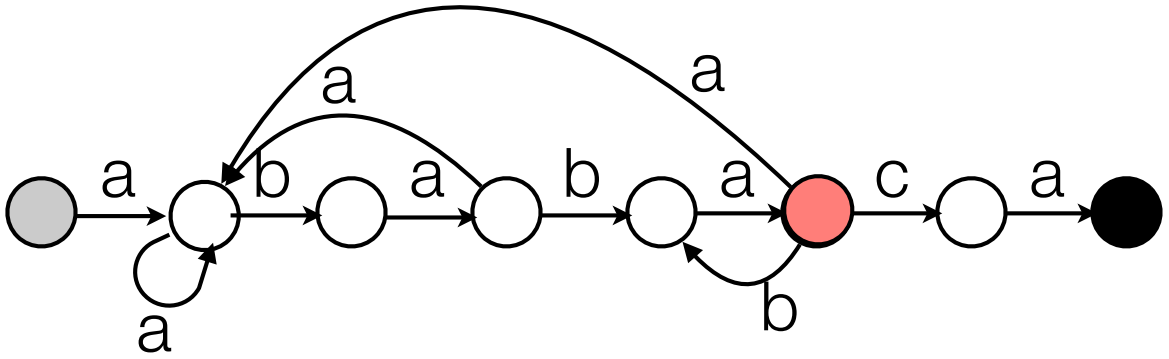


read ‘a’? **longest prefix of P that is a proper suffix of ‘ababaa’ = ‘a’**

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

Finite Automaton Construction

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



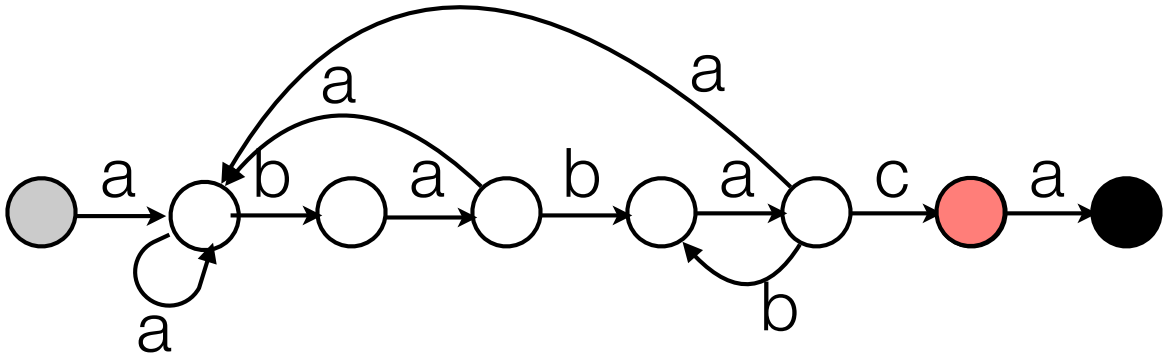
read ‘b’? **longest prefix of P that is a proper suffix of ‘ababaa’ = ‘abab’**

Matched until now: a b a b a b

P: a b a b a c a

Finite Automaton Construction

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

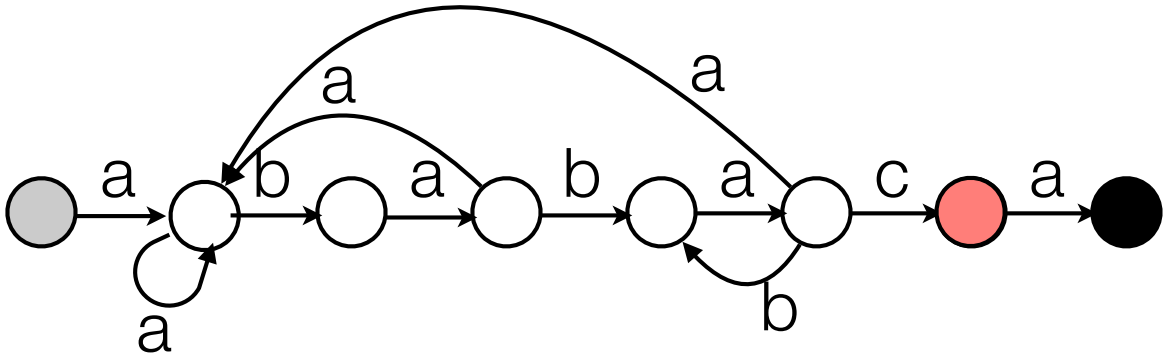


read ‘b’?

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

Finite Automaton Construction

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

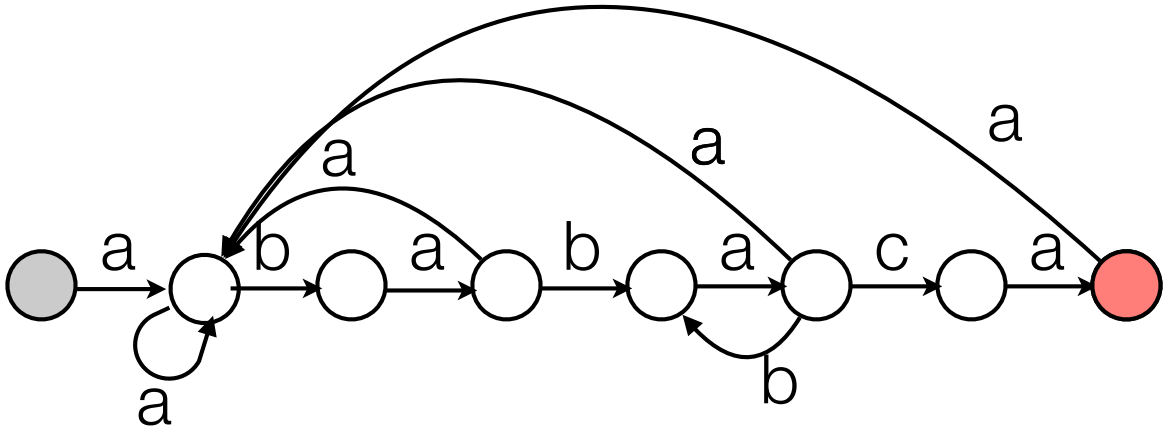


read ‘c’?

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

Finite Automaton Construction

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

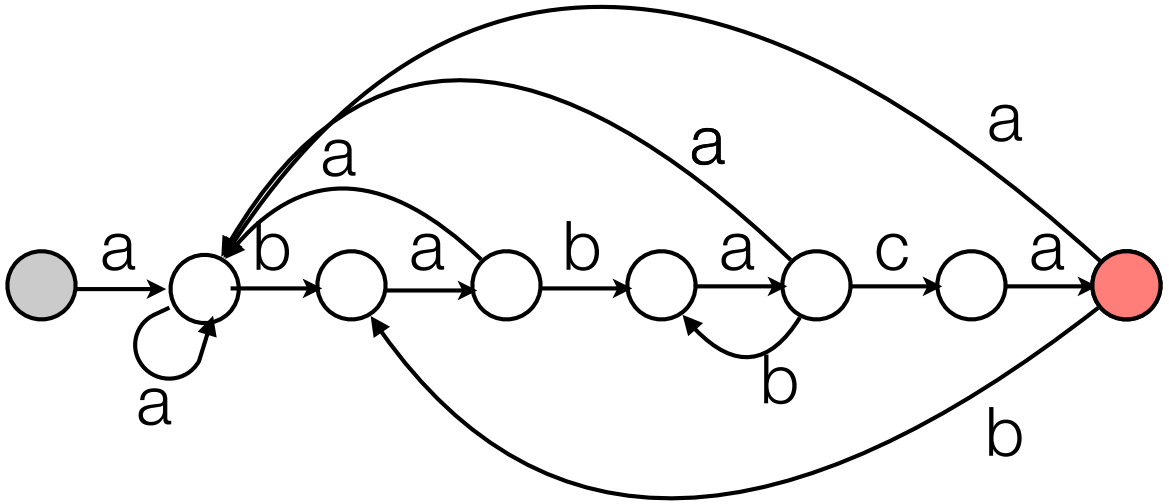


read ‘a’?

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

Finite Automaton Construction

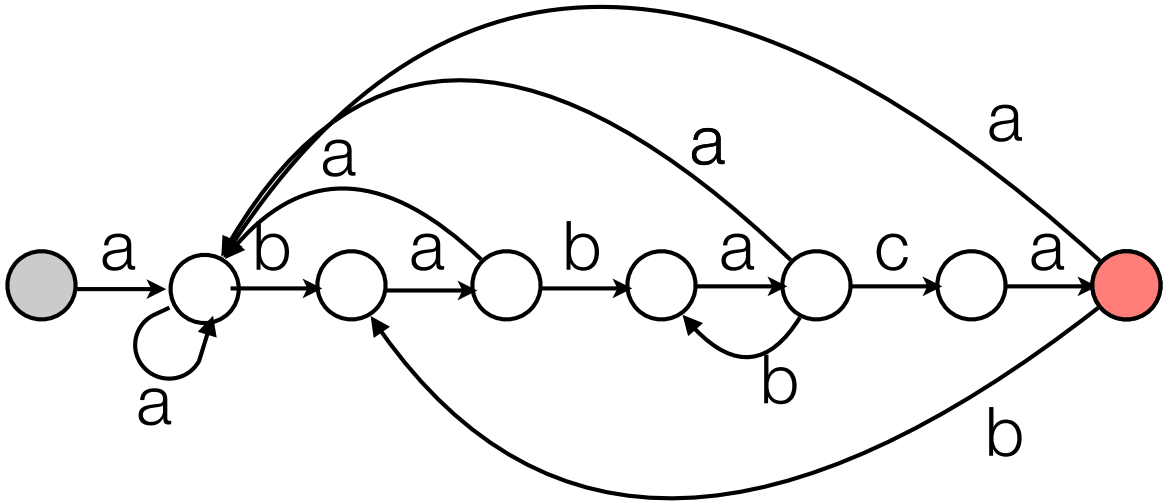
- Finite automaton: alphabet Σ = {a,b,c}. P= abababaca.



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

Finite Automaton Construction

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

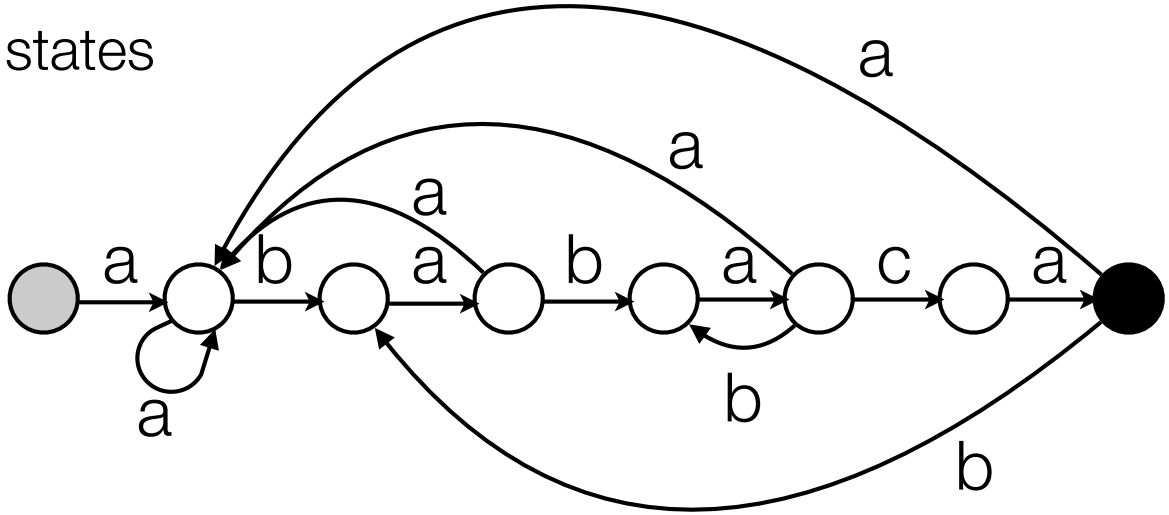


read ‘c’?

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

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

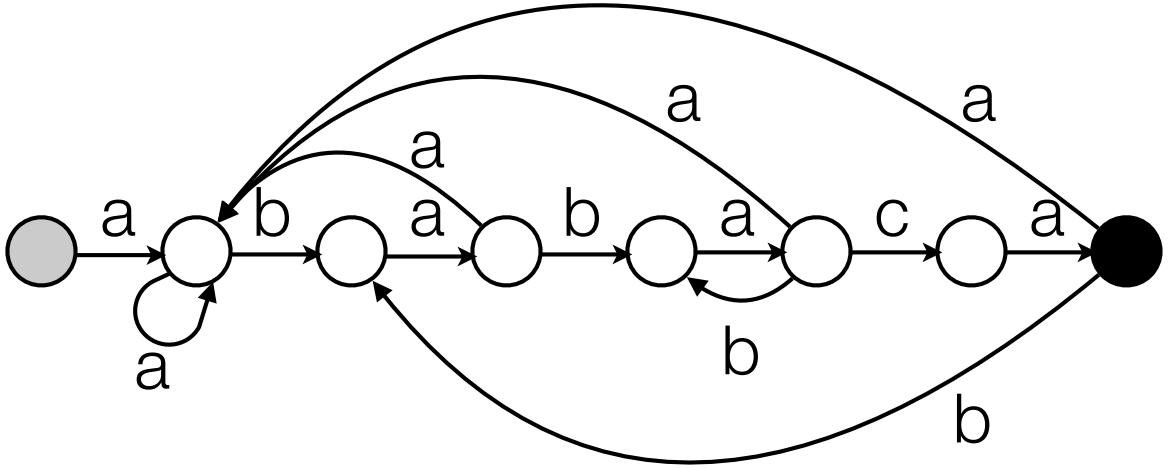


- Matching time: O(n)
- Preprocessing time: $O(m^3|\Sigma|)$. (Can be done in $O(m|\Sigma|)$).
- Total time: $O(n + m|\Sigma|)$

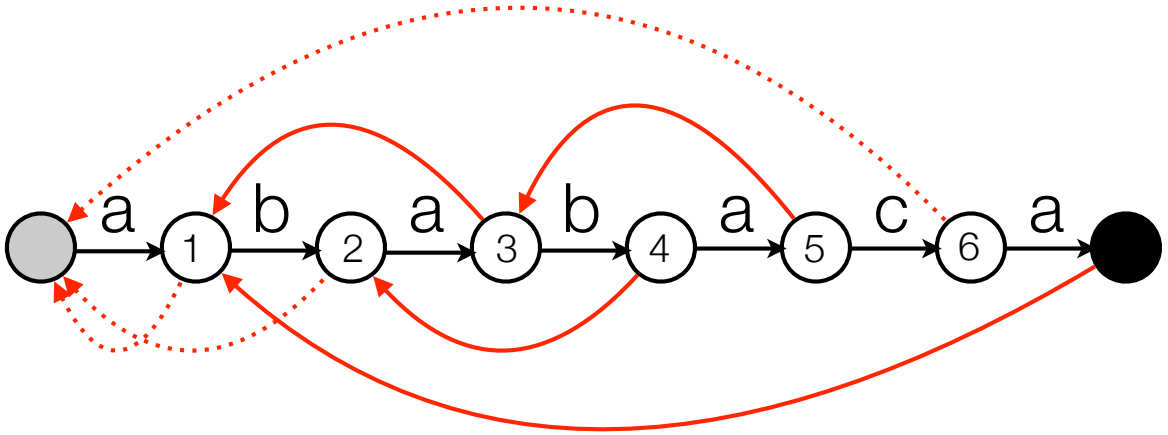
KMP

KMP

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

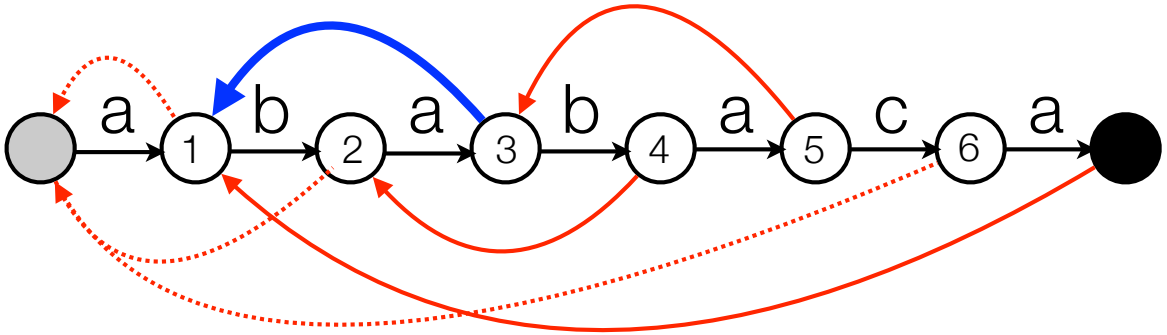


- KMP: Can be seen as finite automaton with *failure links*:



KMP

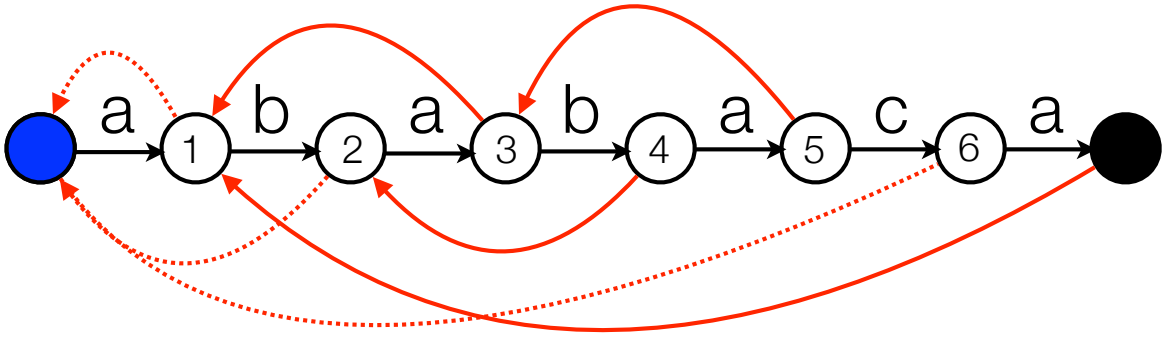
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a suffix of what we have *matched* until now (ignore the mismatched character).



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

KMP matching

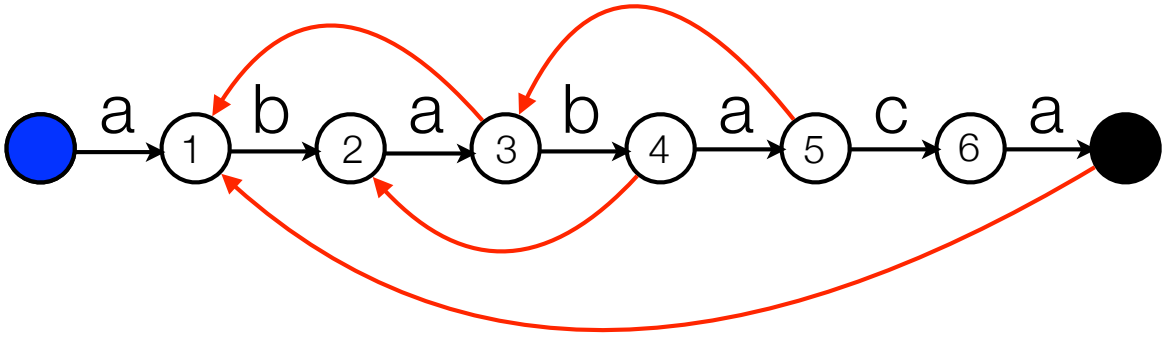
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a suffix of what we have *matched* until now.



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

KMP

- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - 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?

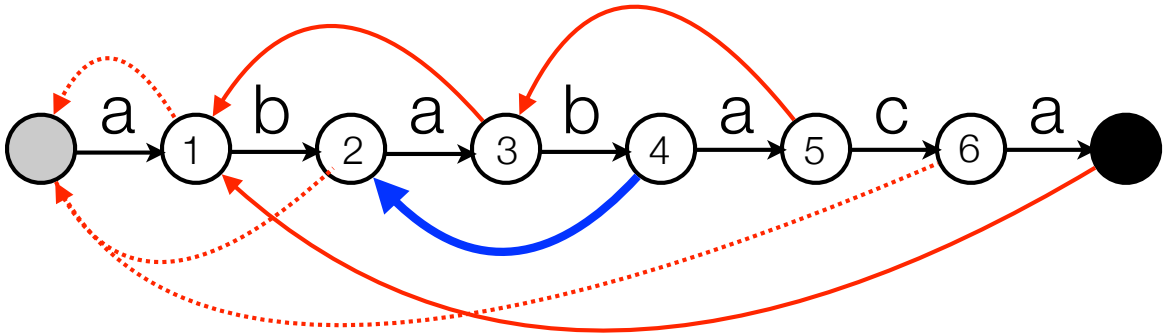
KMP Analysis

- 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

If we are in state j after reading T[1..i], then P[1..j] is the longest prefix of P that is a suffix of T[1..i].

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



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

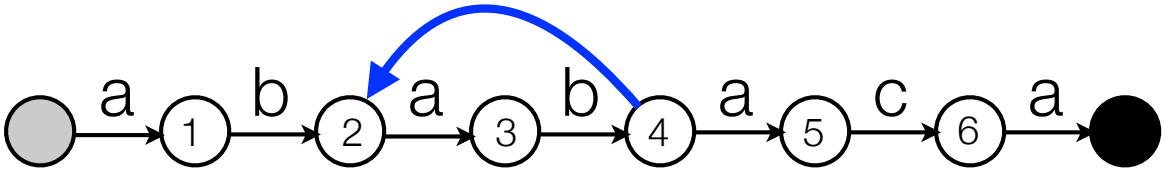
Matched until now: a b a b

a b a b a c a

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 proper suffix of ‘abab'

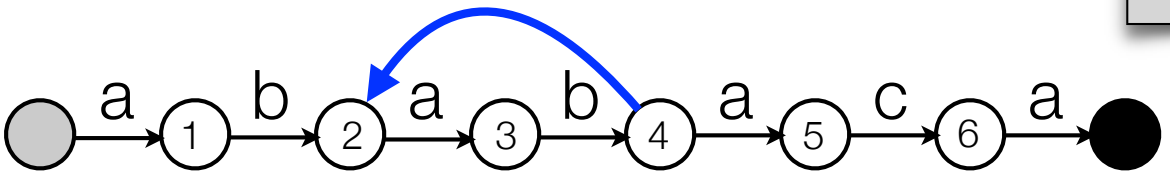


Computation of failure links

- 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.

If we are in state j after reading T[1..i], then P[1..j] is the longest prefix of P that is a suffix of T[1..i].

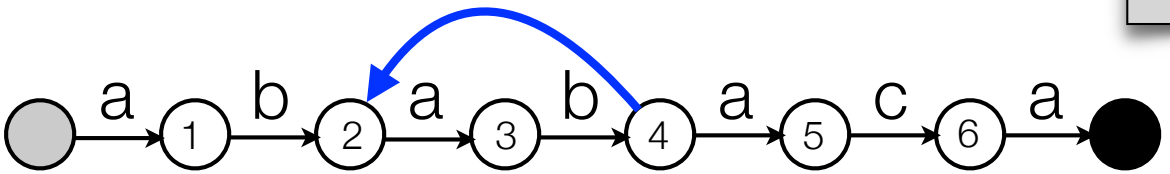
longest prefix of P that is a suffix of ‘bab'



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'

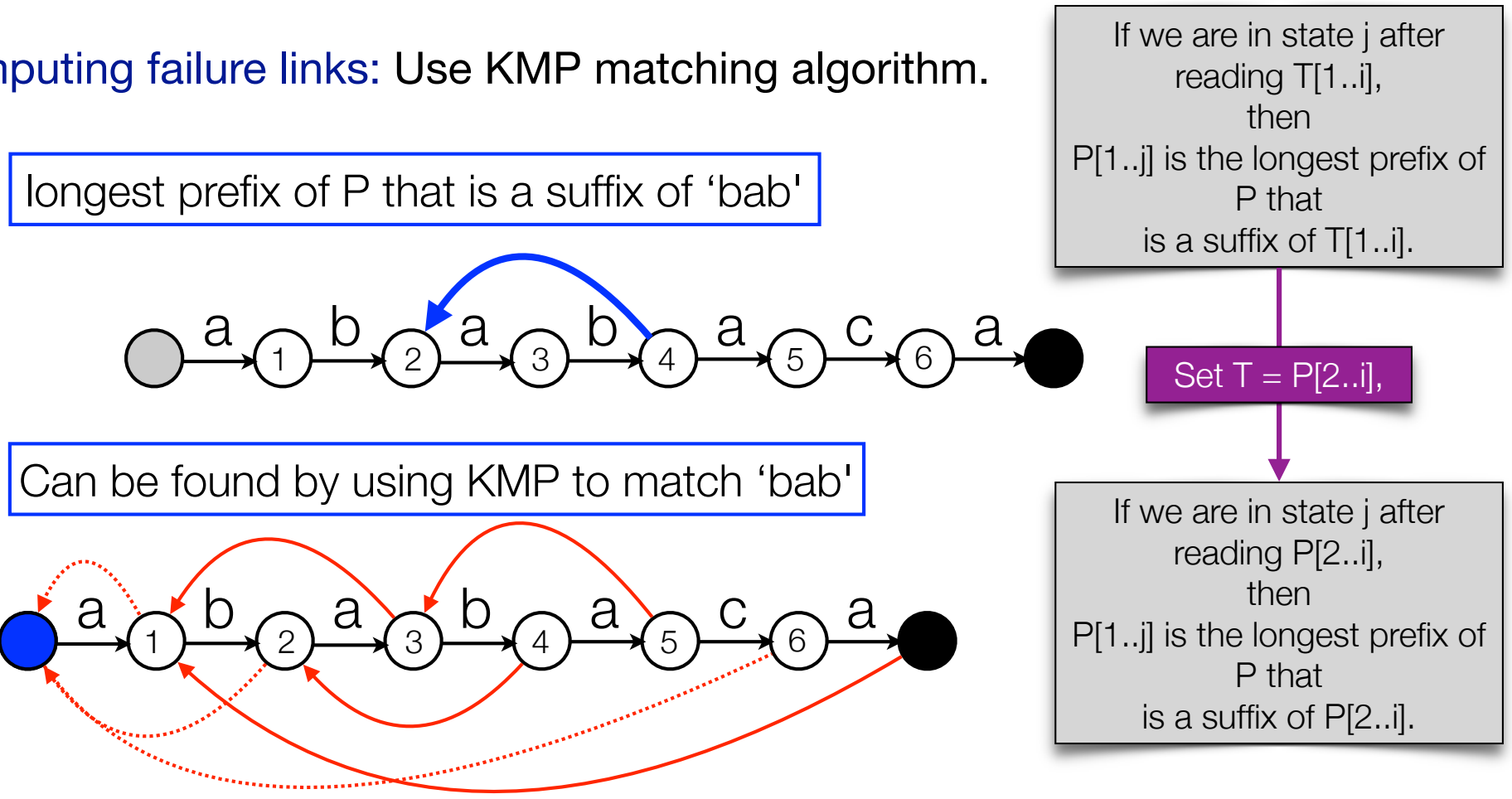
If we are in state j after reading T[1..i], then P[1..j] is the longest prefix of P that is a suffix of T[1..i].

Set T = P[2..i],

If we are in state j after reading P[2..i], then P[1..j] is the longest prefix of P that is a suffix of P[2..i].

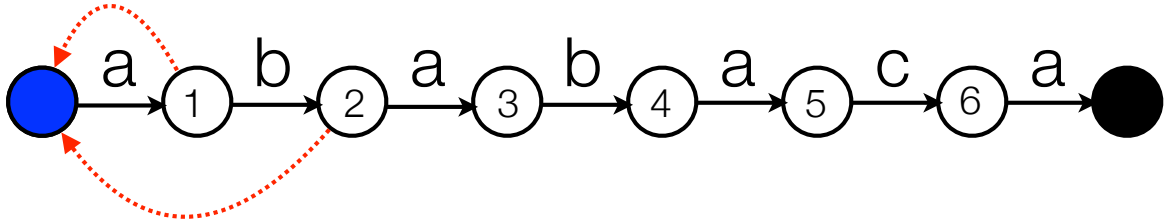
Computation of failure links

- 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

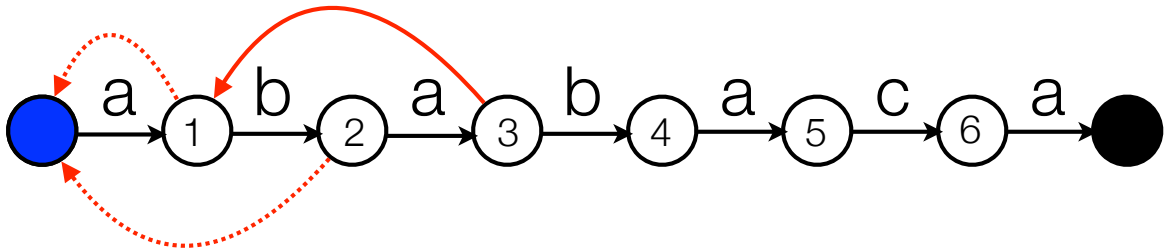
- Computing failure links: As KMP matching algorithm (only need failure links that are already computed).
- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.



T = b

Computation of failure links

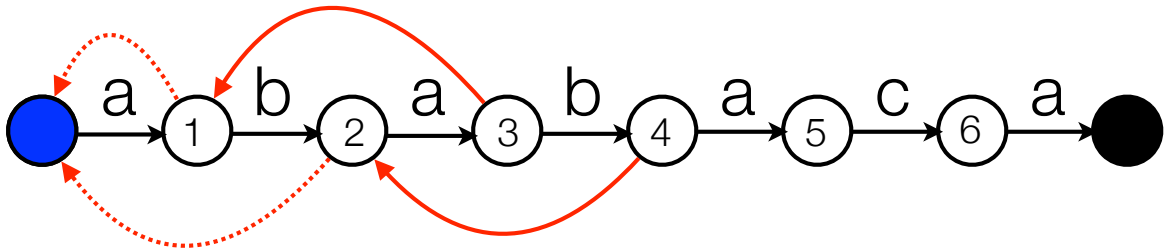
- Computing failure links: As KMP matching algorithm (only need failure links that are already computed).
- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.



T = b a

Computation of failure links

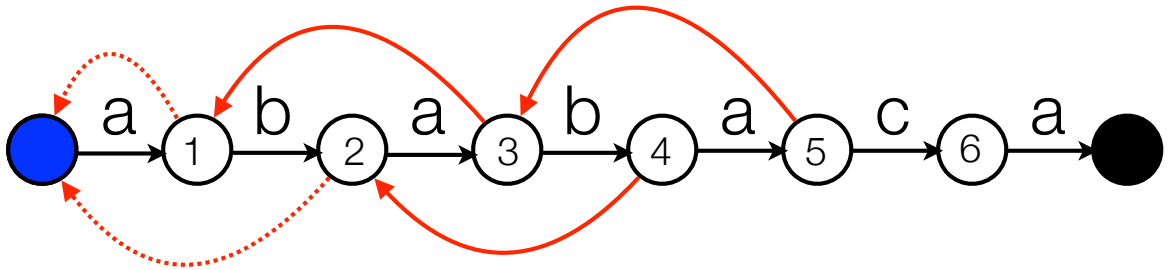
- Computing failure links: As KMP matching algorithm (only need failure links that are already computed).
- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.



T = b a b

Computation of failure links

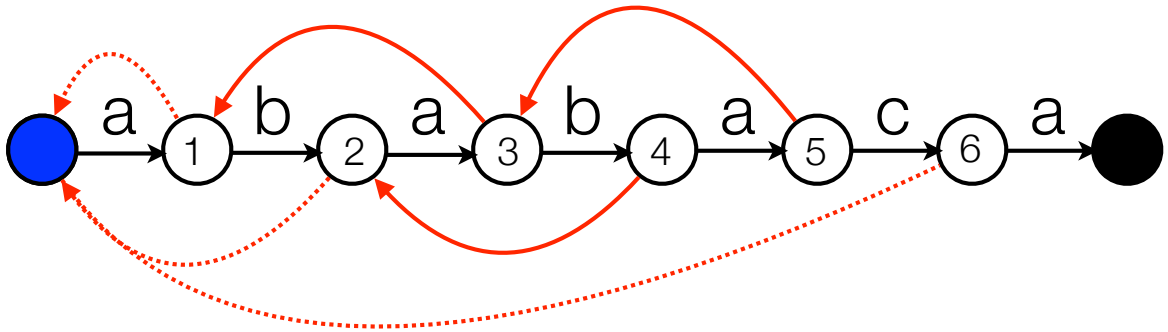
- Computing failure links: As KMP matching algorithm (only need failure links that are already computed).
- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.



T = b a b a

Computation of failure links

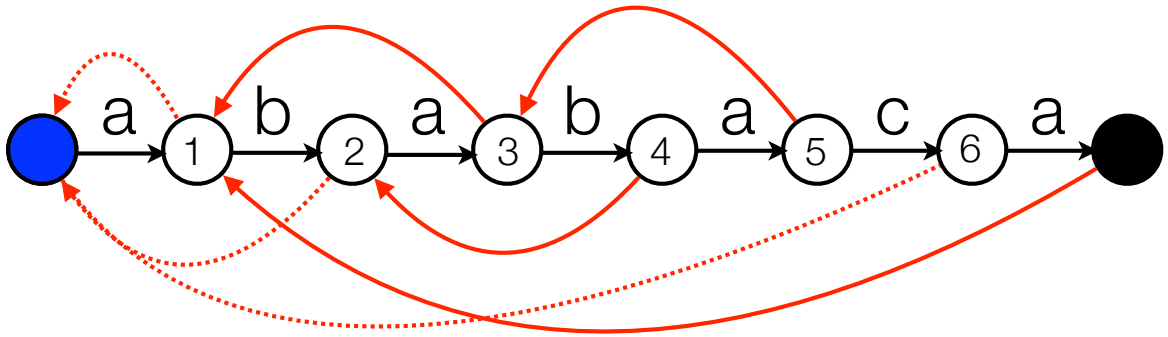
- Computing failure links: As KMP matching algorithm (only need failure links that are already computed).
- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.



T = b a b a c

Computation of failure links

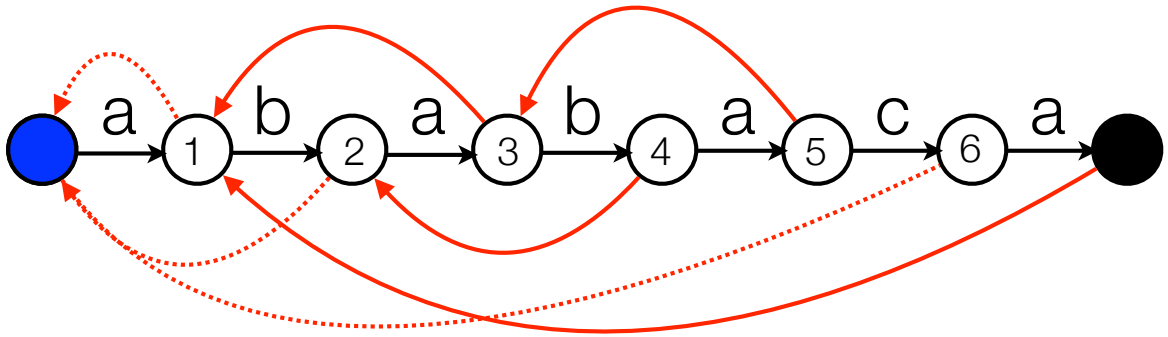
- Computing failure links: As KMP matching algorithm (only need failure links that are already computed).
- Failure link: longest prefix of P that is a proper suffix of what we have *matched* until now.



T = b a b a c a

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.

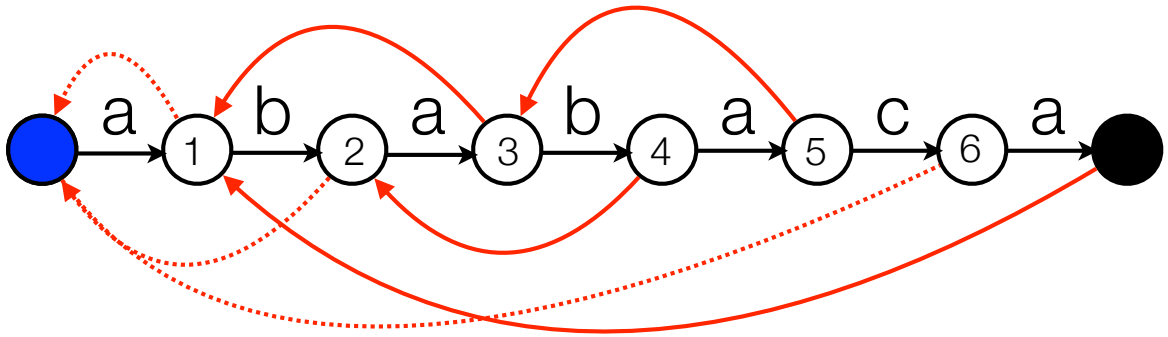


1 2 3 4 5 6 7

P = ■ b a b a c a

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 = ■ b a b a c a

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: the π array

- π array: A representation of the failure links.
- Takes up less space than pointers.

i	1	2	3	4	5	6	7
π[i]	0	0	1	2	3	0	1

