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
  - ε: empty string
  - prefix/suffix: v=xy, x prefix of v, y suffix of v.

String Matching

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

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

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

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

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

a a a b a b a

a a a b a b a

Improving the naive algorithm

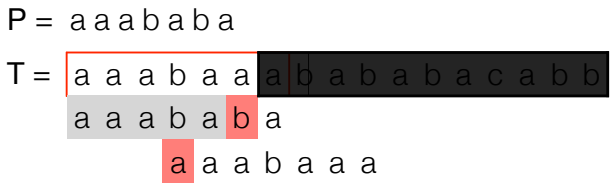
P = a a a b a b a

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

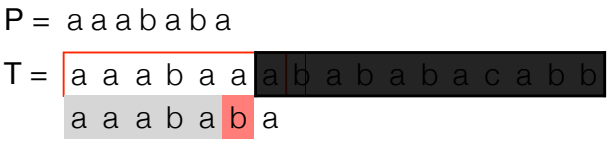
a a a b a b a

a a a b a b a

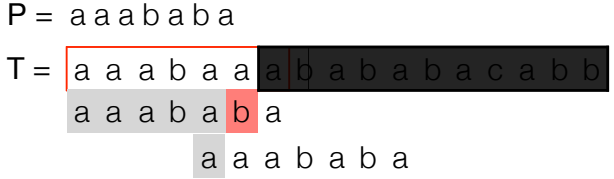
Improving the naive algorithm



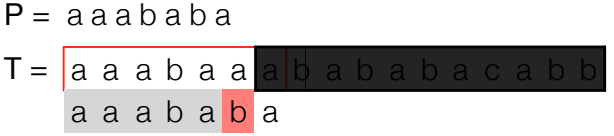
Improving the naive algorithm



Improving the naive 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 What character in the pattern should we check next?

T = a b a b a b

a b a b a c a What character in the pattern should we check next?

T = a b a b a c

a b a b a c a What character in the pattern should we check next?

Fill the table

P = a b a b a c a What character in the pattern should we check next?

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 What character in the pattern should we check next?

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

Fill the table

P = a b a b a c a What character in the pattern should we check next?

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

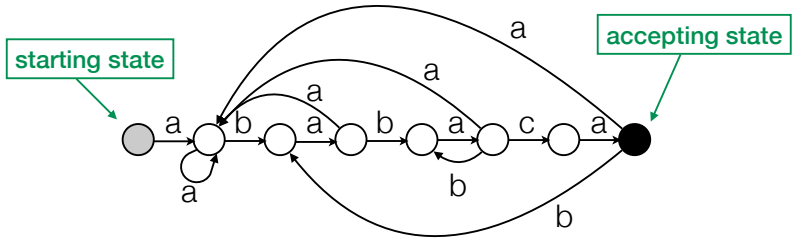
How much of the pattern is matched?

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

Finite Automaton

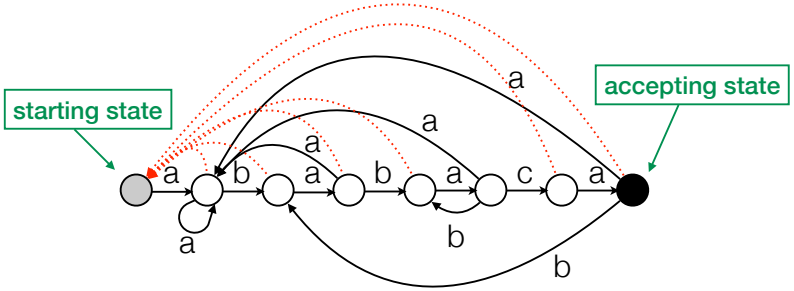
Finite Automaton

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



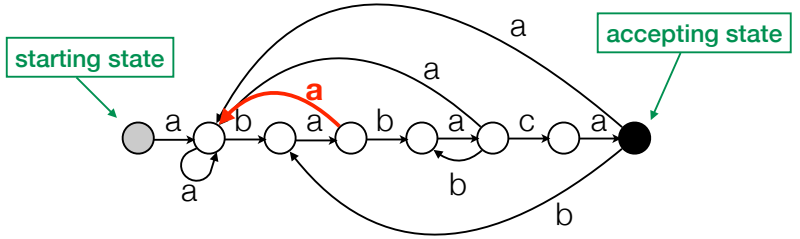
Finite Automaton

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



Finite Automaton

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

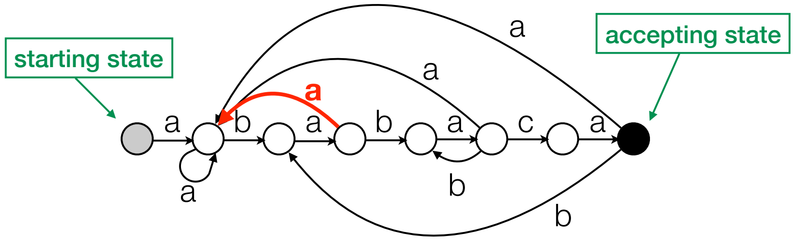


Matched until now: a b a a

P: a b a b a c a

Finite Automaton

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

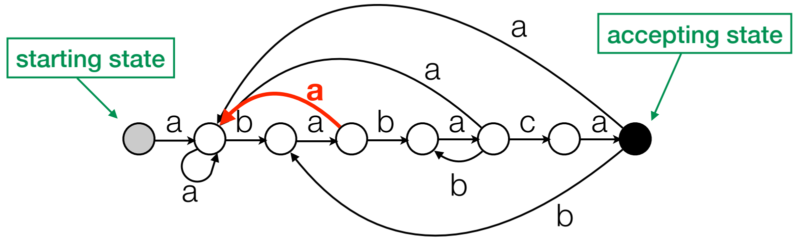


Matched until now: a b a a

P: a b a b a c a

Finite Automaton

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

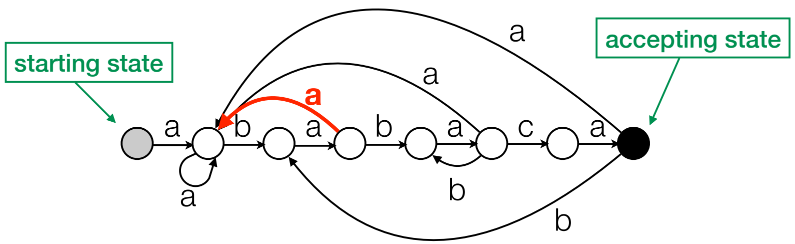


Matched until now: a b a a

P: a b a b a c a

Finite Automaton

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

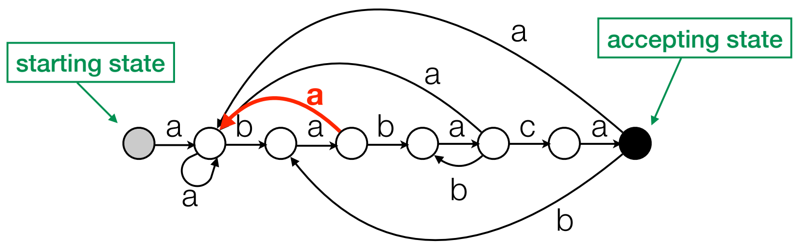


Matched until now: a b a a

P: a b a b a c a

Finite Automaton

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



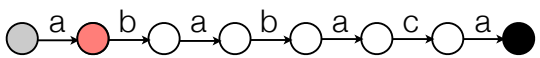
longest prefix of P that is a suffix of 'abaa'

Matched until now: a b a a

P: a b a b a c a

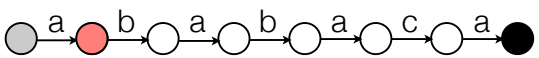
Finite Automaton

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



Finite Automaton

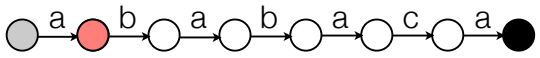
- Finite automaton: alphabet Σ = {a,b,c}. P= abababaca.



read ‘a’?

Finite Automaton

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



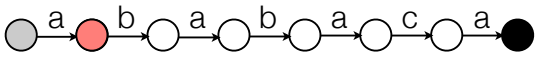
read ‘a’?

Matched until now: a a

P: a b a b a c a

Finite Automaton

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



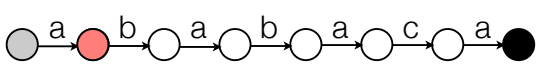
read ‘a’?

Matched until now: a a

P: a b a b a c a

Finite Automaton

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



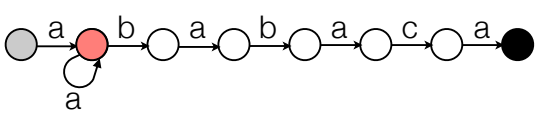
read ‘a’? **longest prefix of P that is a suffix of ‘aa’ = ‘a’**

Matched until now: a a

P: a b a b a c a

Finite Automaton

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



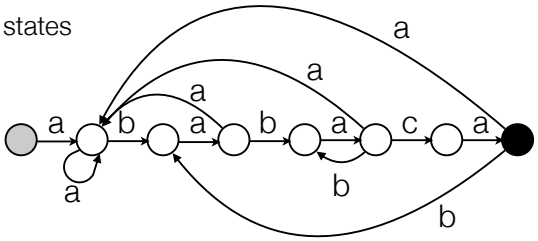
read ‘a’? **longest prefix of P that is a suffix of ‘aa’ = ‘a’**

Matched until now: a a

P: a b a b a c a

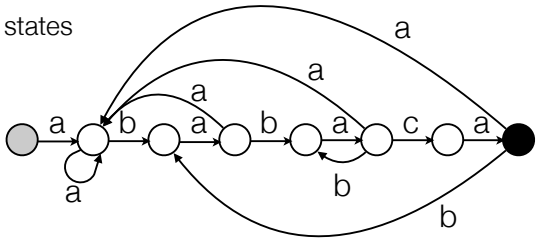
Finite Automaton

- 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



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

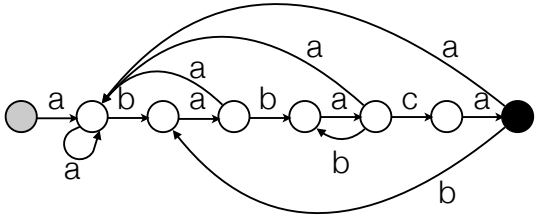


- Matching time: O(n)
- Preprocessing time: $O(m^3|\Sigma|)$. Can be done in $O(m|\Sigma|)$.
- Total time: $O(n + m|\Sigma|)$

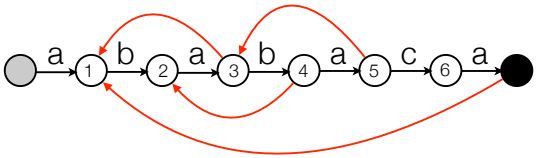
KMP

KMP

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

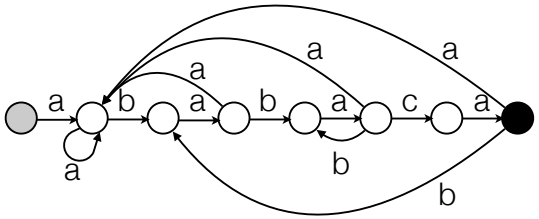


- KMP: Can be seen as finite automaton with *failure links*:

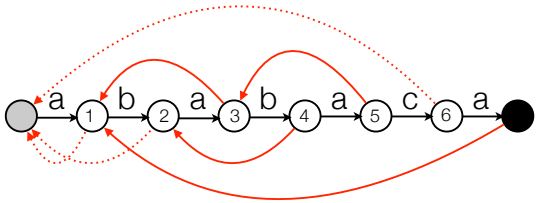


KMP

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

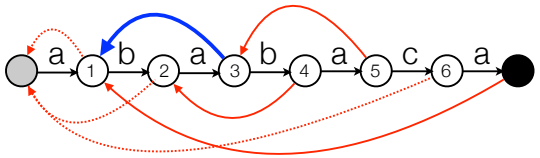


- KMP: Can be seen as finite automaton with *failure links*:



KMP

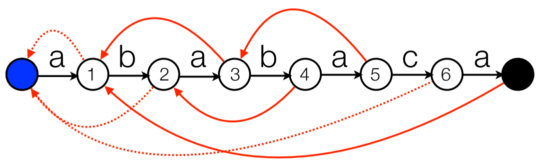
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a suffix of what we have *matched* until now (ignore the mismatched character).



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

KMP matching

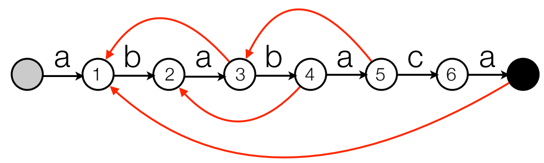
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a suffix of what we have *matched* until now.



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

KMP

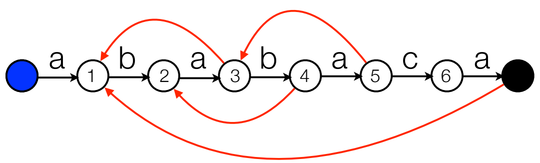
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

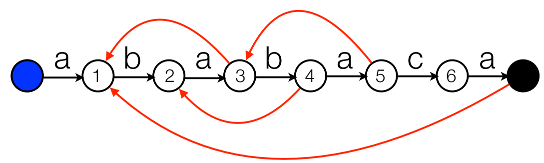
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

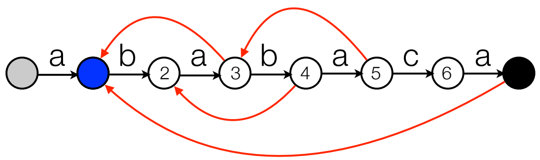
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

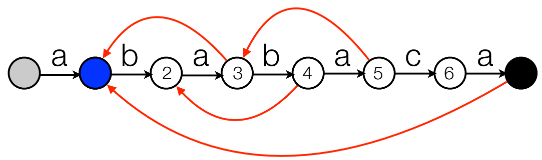
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

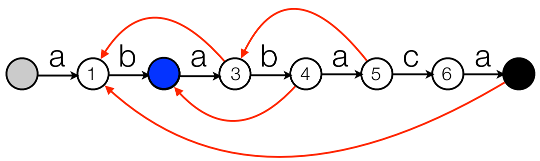
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

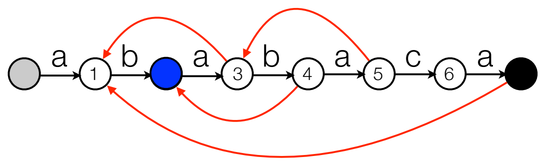
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

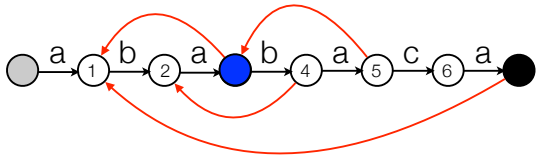
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

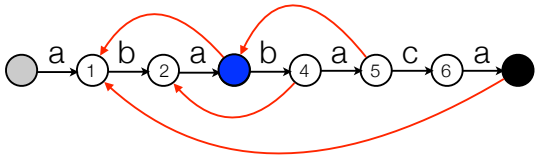
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

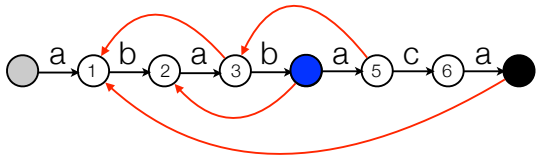
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

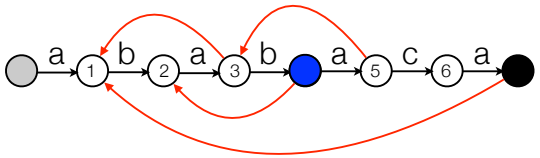
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

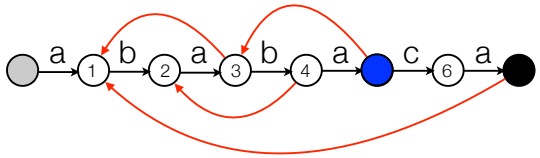
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

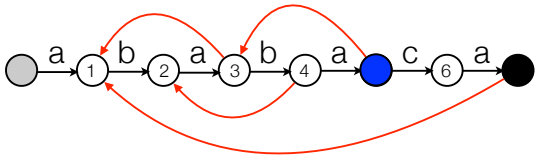
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

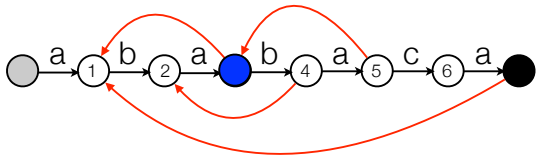
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

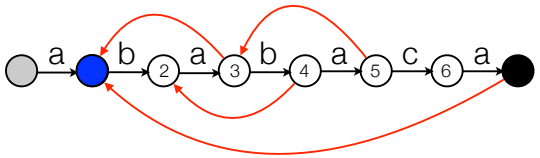
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

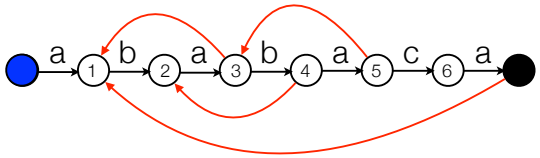
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

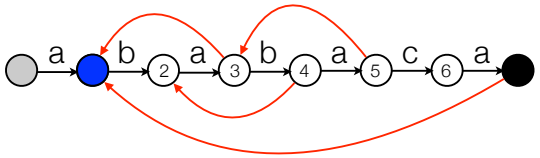
- KMP: Can be seen as finite automaton with *failure links*:
  - longest prefix of P that is a proper suffix of what we have *matched* until now.
  - can follow several failure links when matching one character:



T = a b a b a a

KMP

- 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:



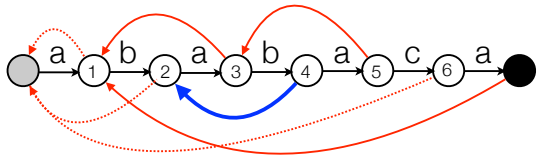
T = a b a b a a

KMP Analysis

- 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.

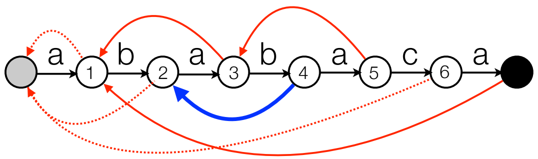


longest prefix of P that is a suffix of 'abab'

Matched until now:

Computation of failure links

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

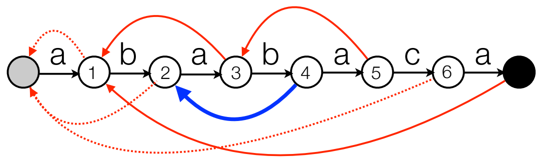


longest prefix of P that is a suffix of 'abab'

Matched until now: a b a b

Computation of failure links

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

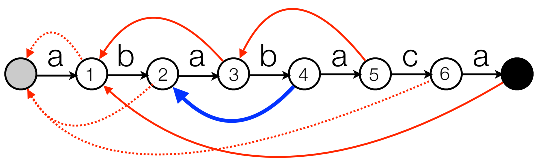


longest prefix of P that is a suffix of 'abab'

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.

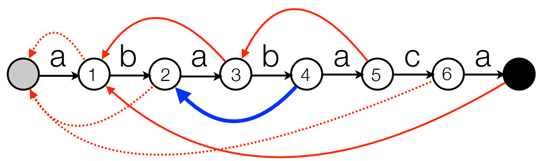


longest prefix of P that is a suffix of 'abab'

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.



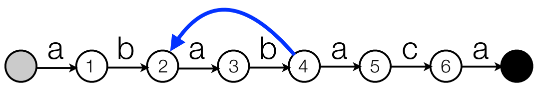
longest prefix of P that is a suffix of 'abab'

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

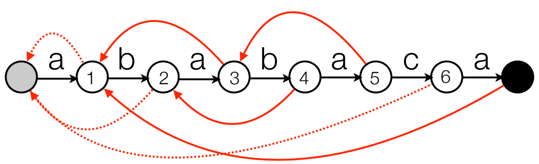
Computation of failure links

- 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 suffix of ‘abab'



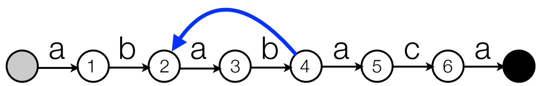
Can be found by using KMP to match ‘bab'



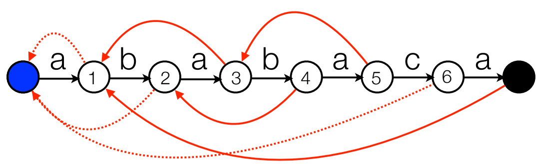
Computation of failure links

- 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 ‘abab'



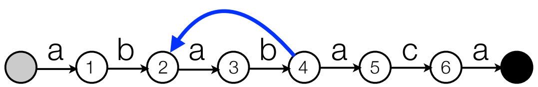
Can be found by using KMP to match ‘bab'



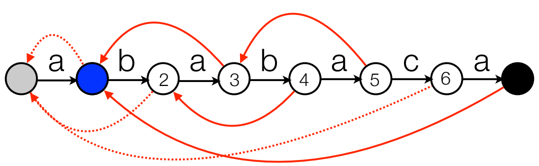
Computation of failure links

- 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 ‘abab'



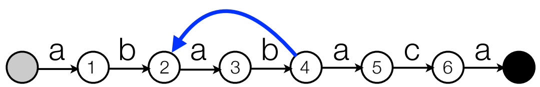
Can be found by using KMP to match ‘bab'



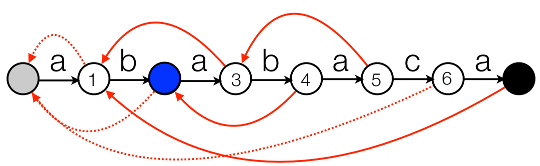
Computation of failure links

- 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 ‘abab'



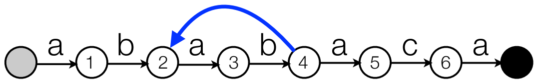
Can be found by using KMP to match ‘bab'



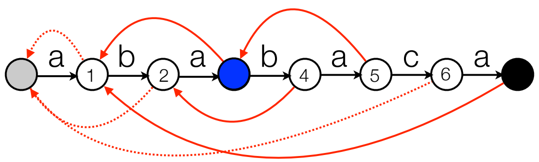
Computation of failure links

- 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 ‘abab'



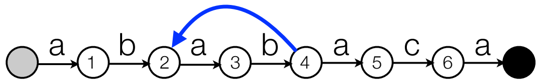
Can be found by using KMP to match ‘bab'



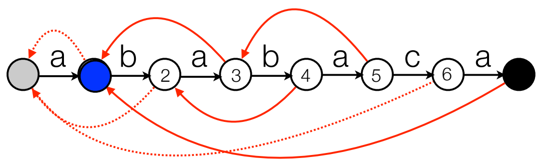
Computation of failure links

- 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 ‘abab'



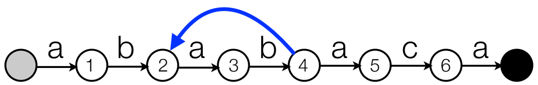
Can be found by using KMP to match ‘bab'



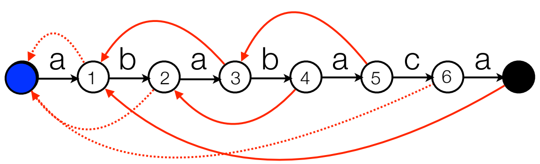
Computation of failure links

- 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 ‘abab'



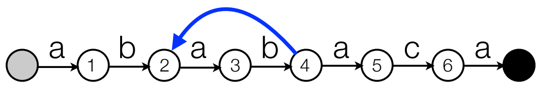
Can be found by using KMP to match ‘bab'



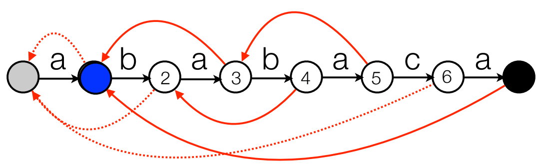
Computation of failure links

- 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 ‘abab'

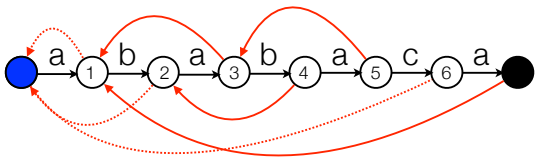


Can be found by using KMP to match ‘bab'



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 =	a	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