# Lecture 5
# String and KMP

Bo Tang @ SUSTech, Spring 2018

# Our Roadmap

- String Concepts
- String Searching Problem
  - Brute Force Solution
  - Finite State Automata (FSA)
  - KMP Algorithm
  - FSA and Prefix Function

# String Definition

- String:
  - Sequence of characters over some alphabet
  - Binary {0,1}: S1 = “10000101010101001010101”
  - DNA {ACGT}: S2 = “ACGTACGTACGTTCGA”
  - English Characters {a…z, A..Z}: S3 = “Hello World”
- Applications
  - Word processors
  - Virus scanning
  - Text retrieval
  - Natural language processing
  - Web search engine

# String Operators

- append: append to string
- assign: assign content to string
- insert: insert to string
- erase: erase characters from string
- replace: replace portion of string
- swap: swap string values
- find: find the specific char in the string
- Give string s="SUSTechCS203", how many sub string it has?

# Our Roadmap

- String Concepts
- String Searching Problem
  - Brute Force Solution
  - Finite State Automata (FSA)
  - KMP Algorithm
  - FSA and Prefix Function

# Why String Searching?

- **Applications in Computational Biology**
  - DNA sequence is a long word (or text) over a 4-letter alphabet
  - GTTTGAGTGGTCAGTCTTTTCGTTTCGACGGAGCCC…..
  - Find a Specific pattern W
- **Finding patterns in documents formed using a large alphabet**
  - Word processing
  - Web searching
  - Desktop search (Google, MSN)
- **Matching strings of bytes containing**
  - Graphical data
  - Machine code
- **grep in unix**
  - grep searches for lines matching a pattern.

# String Searching

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | s | s | u | s | u | s | t | c | s | c |

| Search Pattern | | | | |
|---|---|---|---|---|
| s | u | s | t | c |

| Successful Search | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | s | s | u | s | u | s | t | c | s | c |

- Parameter
  - n: # of characters in text
  - m: # of characters in pattern
  - Typically, n >> m
    - e.g., n = 1 Billion, m = 100

# Brute Force

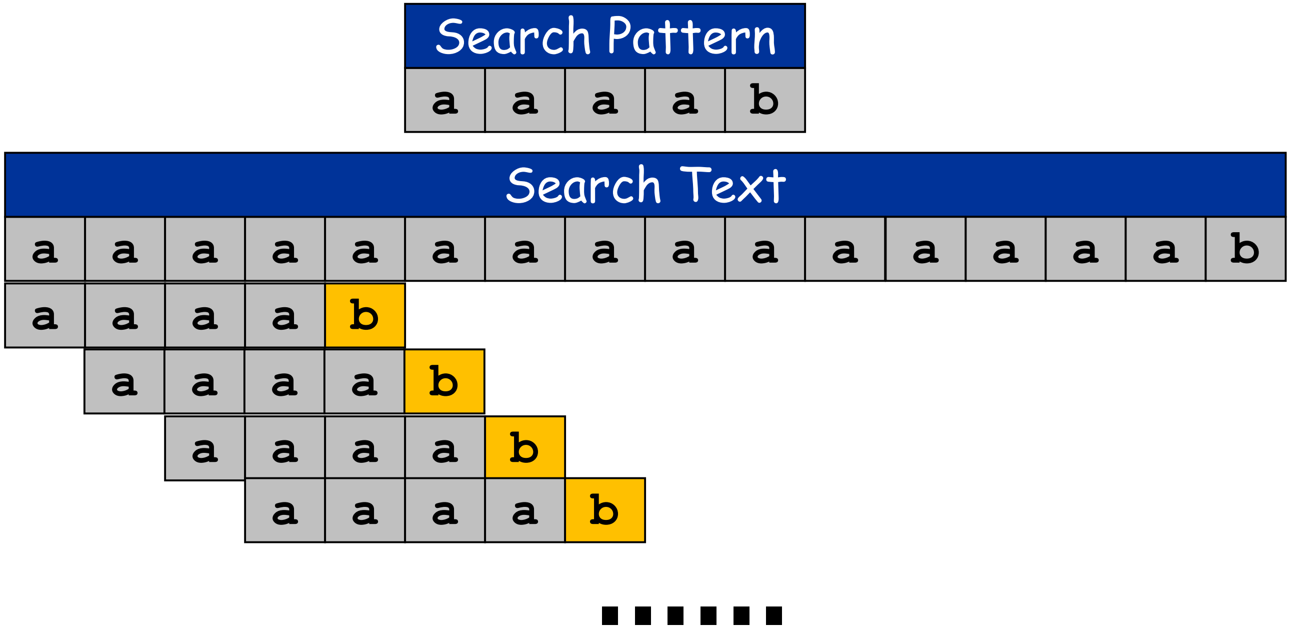
- Brute force
  - Check for pattern starting at every text position
- **Algorithm:** BruteForce(T, P):

```
n ← len(T), m ← len(P)
for i ← 0 to n-1
        for j ← 0 to m-1
                if P[j] != T[i+j] then
                        break;
        if j = m
pattern occurs with shift I
```

- Time complexity?

# Analysis of Brute Froce

- Analysis of brute force
  - Running time depends on pattern and text
  - Can be slow when strings repeat themselves
  - Worst case: mn comparisions
  - Too slow when m and n are large



# Can we do better?

- How to avoid re-computation?
  - Pre-analyze search pattern
  - Example: suppose the first 4 chars of pattern are all a's
    - If t[0..3] matches p[0..3] then t[1..3] matches p[0..2]
    - No need to check i=1, j=0,1,2
    - Saves 3 comparisons
  - Need better ideas in general

| Search Pattern | | | | |
|---|---|---|---|---|
| a | a | a | a | b |

| Search Text | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | a | a | a | a | a | a | a | a | a | a | a | a | b |
| a | a | a | a | b | | | | | | | | | | | |
| | a | a | a | a | b | | | | | | | | | | |

# Our Roadmap

- String Concepts
- String Searching Problem
  - Brute Force Solution
  - Finite State Automata (FSA)
  - KMP Algorithm
  - FSA and Prefix Function

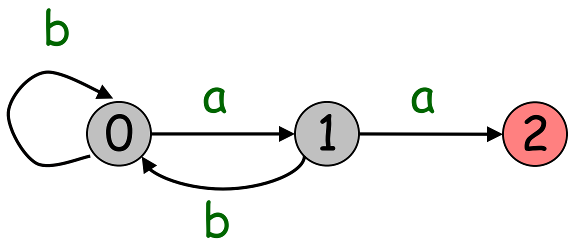
# Finite State Automata (FSA)

- FSA is a computing machine that takes
  - A string as an input
  - Outputs YES/NO answer
    - That is, the machine "accepts" or "rejects" the string

# Finite State Automata

- A finite State automaton is defined by:
  - *Q, a set of states*
  - $q_0 \in Q$, *the start state*
  - $A \subseteq Q$, *the accepting states*
  - $\Sigma$, *the input alphabet*
  - $\delta$, *the transition function, from* $Q \times \Sigma$ *to* $Q$

| | 0 | 1 |
|---|---|---|
| a | 1 | 2 |
| b | 0 | 0 |

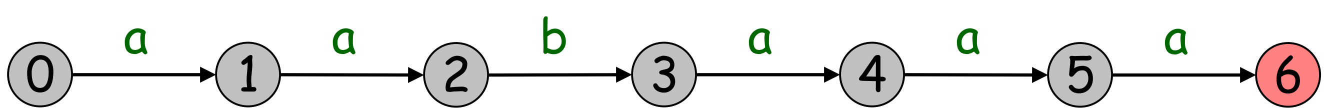


# FSA idea for String Matching

- Start in state $q_0$
- Perform a transition from $q_0$ to $q_1$ if next character of T = P[1]
- State $q_i$ means first $i$ characters of P match.
- Transition from $q_i$ to $q_{i+1}$ if the next character of T = P[i+1]

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

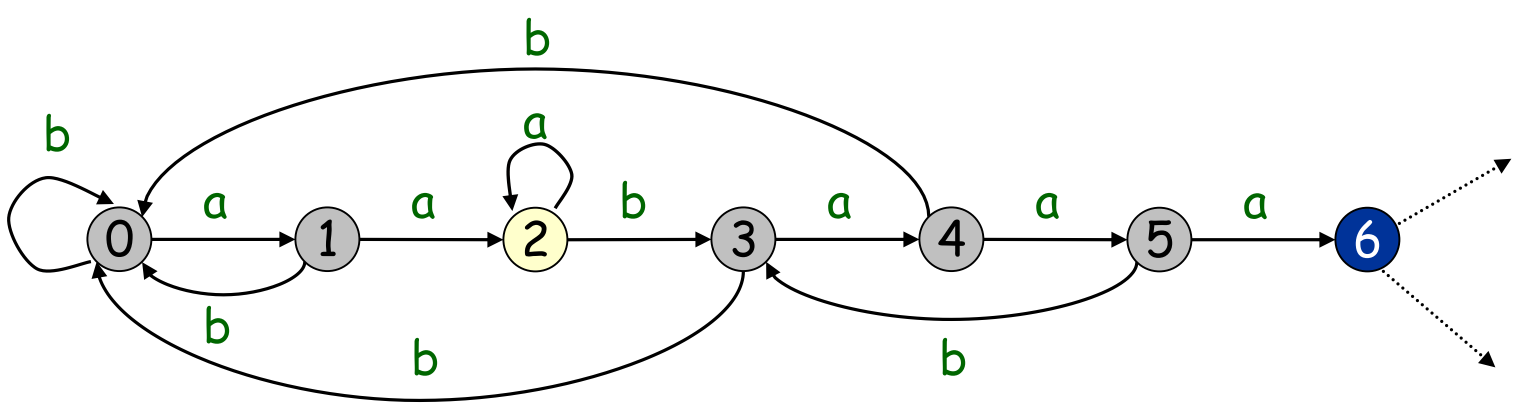
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| a | 1 | 2 | ? | 4 | 5 | 6 |
| b | ? | ? | 3 | ? | ? | ? |




- How to fill these ???
  - Reset to $q_0$? Why not?

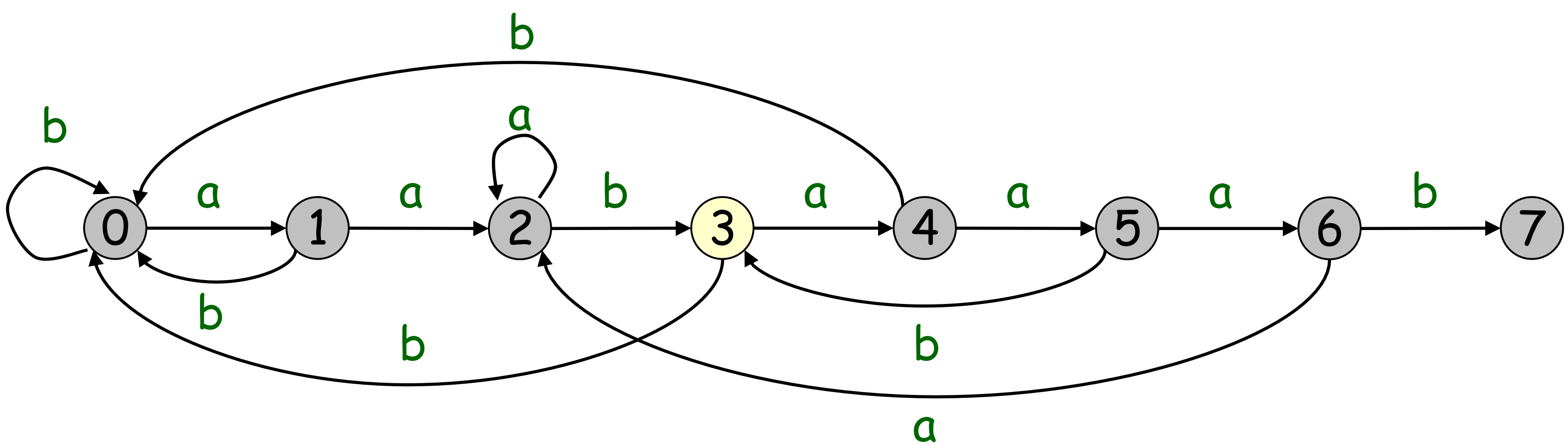
# FSA construction

- FSA construction
  - FSA builds itself
- Example. Build FSA for `aabaaabb`
  - State 6. P[0..5]=aabaaa
  - assume you know state for p[1..5] = abaaa — X = 2
  - if next char is b (match): go forward — 6 + 1 = 7
  - if next char is a (mismatch): go to state for abaaaa — X + 'a' = 2
  - update X to state for p[1..6] = abaaab — X + 'b' = 3



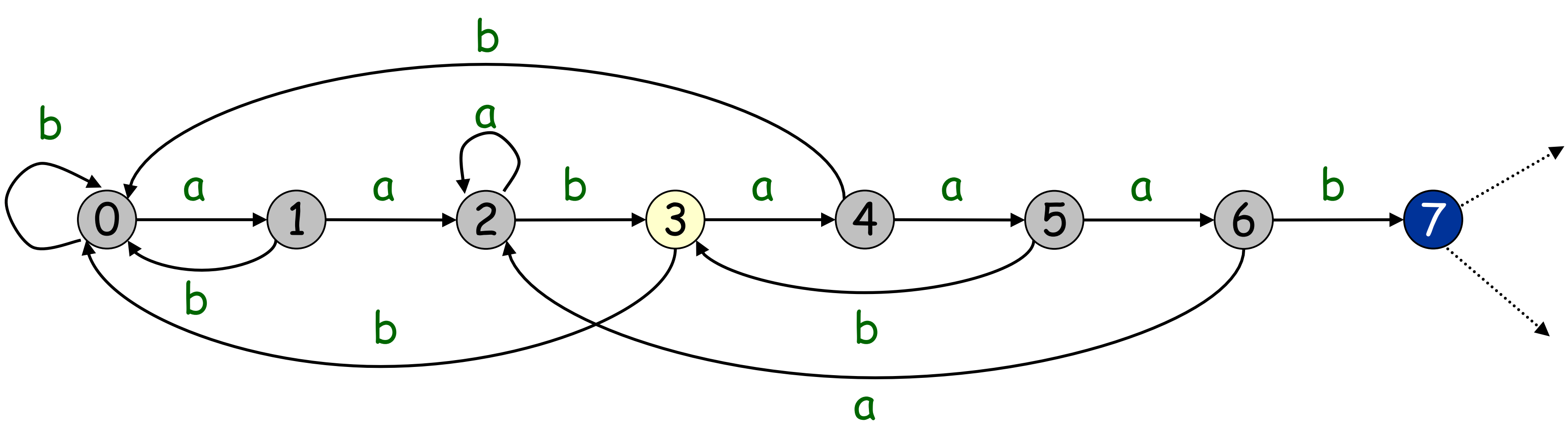
# FSA construction

- FSA construction
  - FSA builds itself
- Example. Build FSA for `aabaaabb`



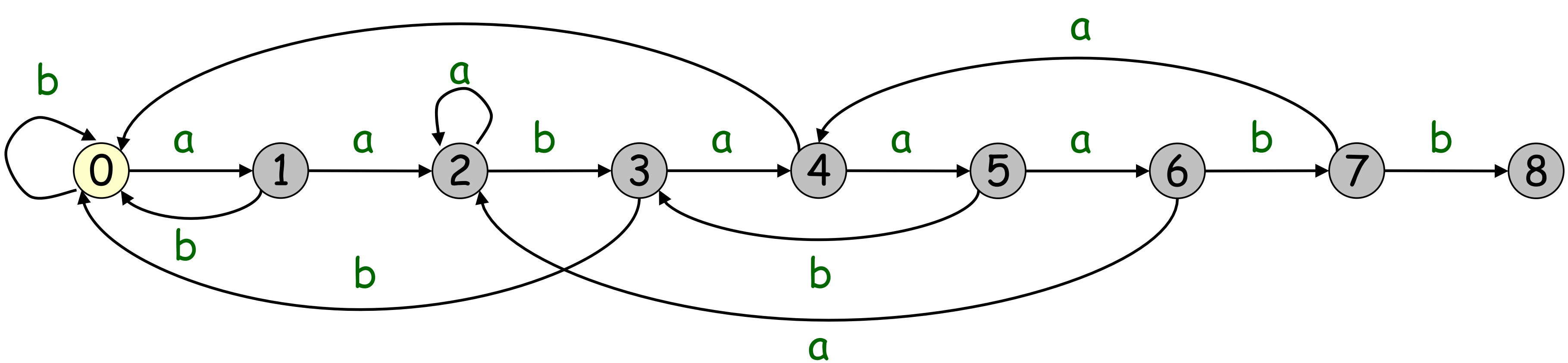
# FSA construction

- FSA construction
  - FSA builds itself
- Example. Build FSA for `aabaaabb`
  - State 7. p[0..6]=aabaaab
  - assume you know state for p[1..6] = abaaab — X = 3
  - if next char is b (match): go forward — 7 + 1 = 8
  - if next char is a (mismatch): go to state for abaaaba — X + 'a' = 4
  - update X to state for p[1..7] = abaaabb — X + 'b' = 0



# FSA construction

- FSA construction
  - FSA builds itself
- Example. Build FSA for `aabaaabb`



# FSA construction

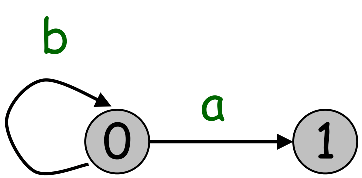
- FSA construction
  - FSA builds itself
- Crucial Insight
  - To compute transitions for state n of FSA, suffices to have:
    - FSA for state 0 to n-1
    - State X that FSA ends up in with input p[1..n-1]
  - To compute state X' that FSA ends up in with input p[1..n], it suffices to have
    - FSA for states 0 to n-1
    - State X that FSA ends up in with input p[1..n-1]

# FSA construction

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| j | pattern[1..j] | X |
|---|---|---|

| |
|---|
| a |
| b |

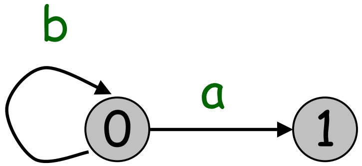


# FSA construction for KMP

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| j | pattern[1..j] | | | | | | | X | next |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 0 | 0 |

| | 0 |
|---|---|
| a | 1 |
| b | 0 |

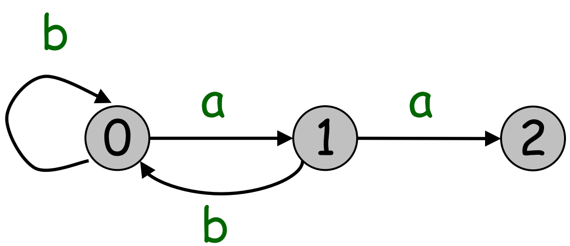


# FSA construction for KMP

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| j | pattern[1..j] | | | | | | | X | next |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 0 | 0 |
| 1 | a | | | | | | | 1 | 0 |

| | 0 | 1 |
|---|---|---|
| a | 1 | 2 |
| b | 0 | 0 |

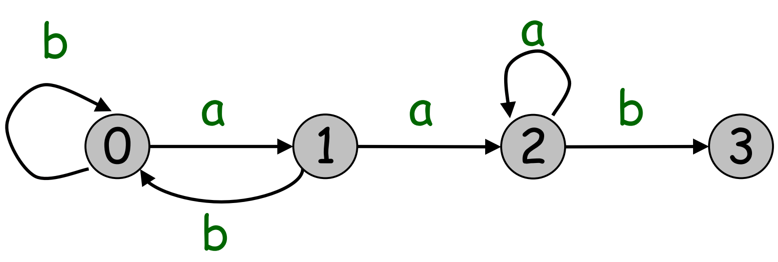


# FSA construction for KMP

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| j | pattern[1..j] | | | | | | | X | next |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 0 | 0 |
| 1 | a | | | | | | | 1 | 0 |
| 2 | a | b | | | | | | 0 | 2 |

| | 0 | 1 | 2 |
|---|---|---|---|
| a | 1 | 2 | 2 |
| b | 0 | 0 | 3 |

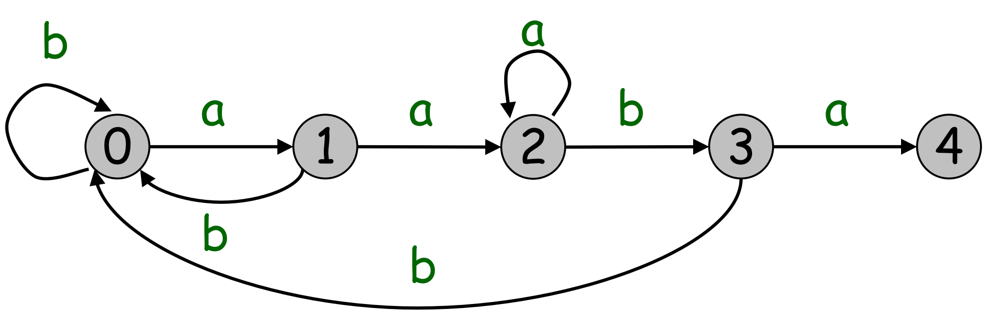


# FSA construction for KMP

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| a | 1 | 2 | 2 | 4 |
| b | 0 | 0 | 3 | 0 |

| j | pattern[1..j] | | | | | | | X | next |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 0 | 0 |
| 1 | a | | | | | | | 1 | 0 |
| 2 | a | b | | | | | | 0 | 2 |
| 3 | a | b | a | | | | | 1 | 0 |

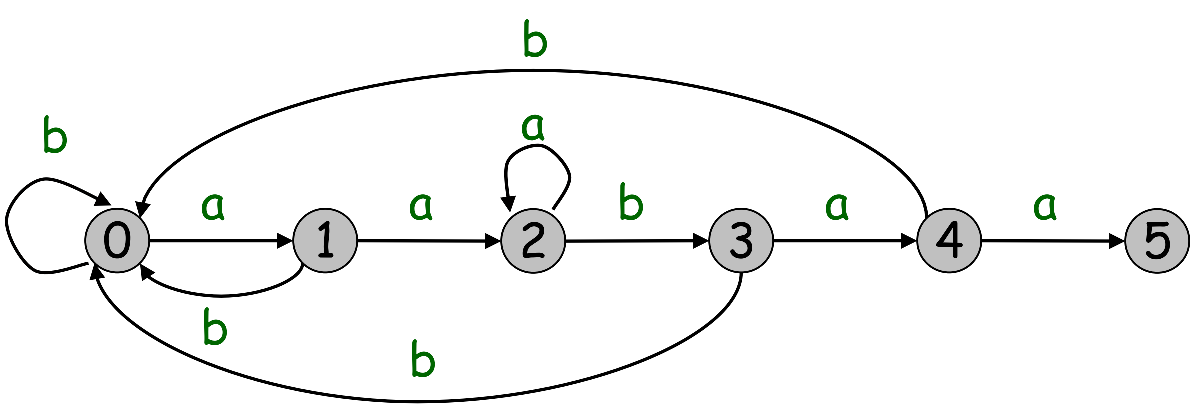


# FSA construction for KMP

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| a | 1 | 2 | 2 | 4 | 5 |
| b | 0 | 0 | 3 | 0 | 0 |

| j | pattern[1..j] | | | | | | | X | next |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 0 | 0 |
| 1 | a | | | | | | | 1 | 0 |
| 2 | a | b | | | | | | 0 | 2 |
| 3 | a | b | a | | | | | 1 | 0 |
| 4 | a | b | a | a | | | | 2 | 0 |

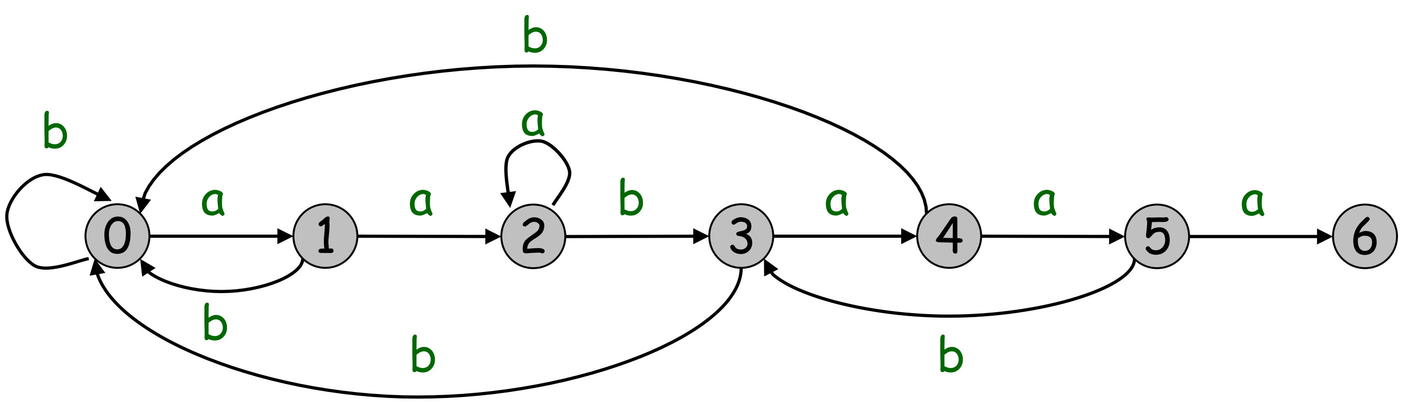


# FSA construction for KMP

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| a | 1 | 2 | 2 | 4 | 5 | 6 |
| b | 0 | 0 | 3 | 0 | 0 | 3 |

| j | pattern[1..j] | | | | | | | X | next |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 0 | 0 |
| 1 | a | | | | | | | 1 | 0 |
| 2 | a | b | | | | | | 0 | 2 |
| 3 | a | b | a | | | | | 1 | 0 |
| 4 | a | b | a | a | | | | 2 | 0 |
| 5 | a | b | a | a | a | | | 2 | 3 |

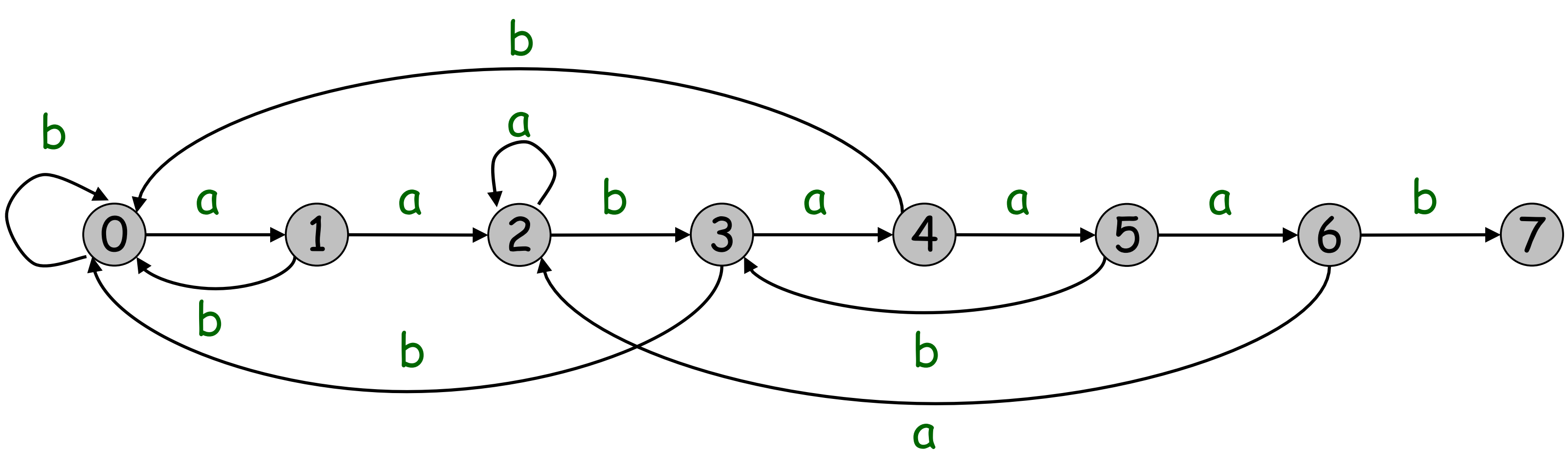


# FSA construction for KMP

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| a | 1 | 2 | 2 | 4 | 5 | 6 | 2 |
| b | 0 | 0 | 3 | 0 | 0 | 3 | 7 |

| j | pattern[1..j] | | | | | | | X | next |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 0 | 0 |
| 1 | a | | | | | | | 1 | 0 |
| 2 | a | b | | | | | | 0 | 2 |
| 3 | a | b | a | | | | | 1 | 0 |
| 4 | a | b | a | a | | | | 2 | 0 |
| 5 | a | b | a | a | a | | | 2 | 3 |
| 6 | a | b | a | a | a | b | | 3 | 2 |

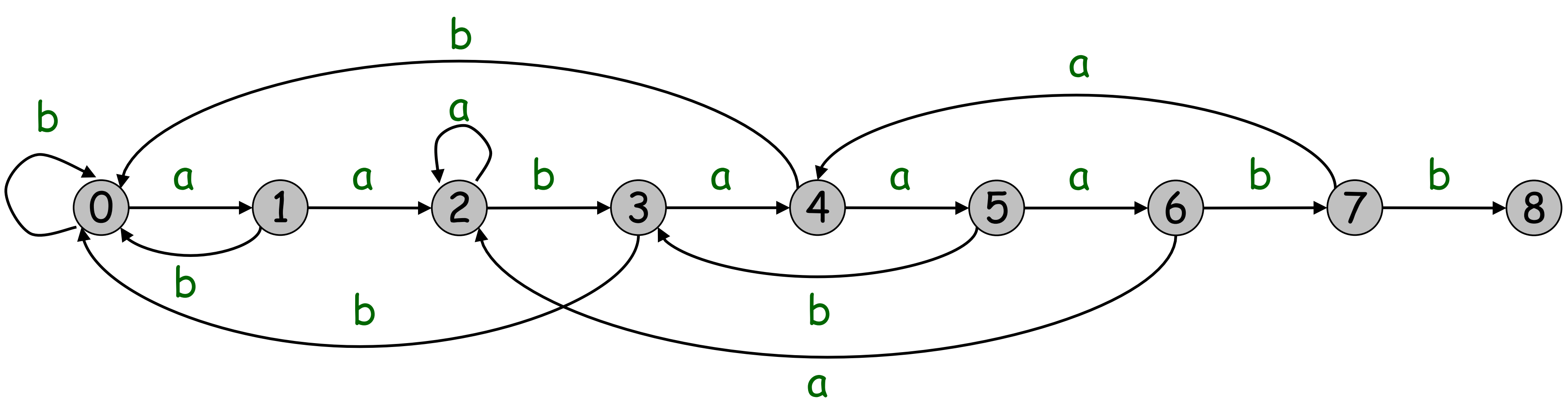


# FSA construction for KMP

| Search Pattern | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | b |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| a | 1 | 2 | 2 | 4 | 5 | 6 | 2 | 4 |
| b | 0 | 0 | 3 | 0 | 0 | 3 | 7 | 8 |

| j | pattern[1..j] | | | | | | | X | next |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 0 | 0 |
| 1 | a | | | | | | | 1 | 0 |
| 2 | a | b | | | | | | 0 | 2 |
| 3 | a | b | a | | | | | 1 | 0 |
| 4 | a | b | a | a | | | | 2 | 0 |
| 5 | a | b | a | a | a | | | 2 | 3 |
| 6 | a | b | a | a | a | b | | 3 | 2 |
| 7 | a | b | a | a | a | b | b | 0 | 4 |



# FSA algorithm

- **Algorithm:** FSA(P):

```
m ← len(P)
next[0] ← 0
X ← 0
for j ← 1 to m - 1
      if P[X] = P[j] then      // char match
            next[j] ← next[X]
            X ← X + 1
      else                     // char mismatch
            next[j] ← X + 1
            X ← next[X]
return next
```

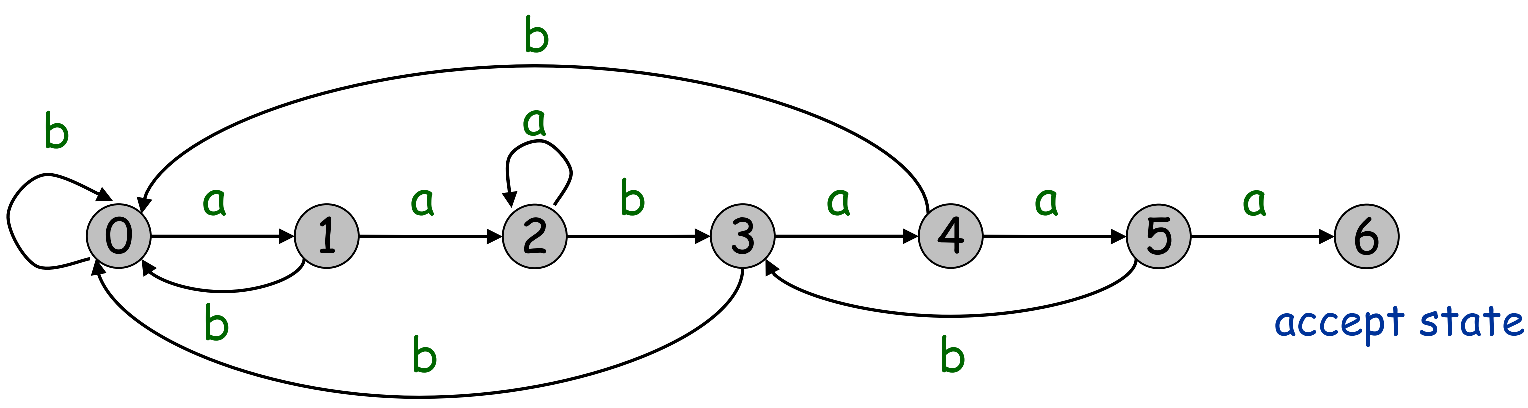
# Our Roadmap

- String Concepts
- String Searching Problem
  - Brute Force Solution
  - Finite State Automata (FSA)
  - KMP Algorithm
  - FSA and Prefix Function

# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

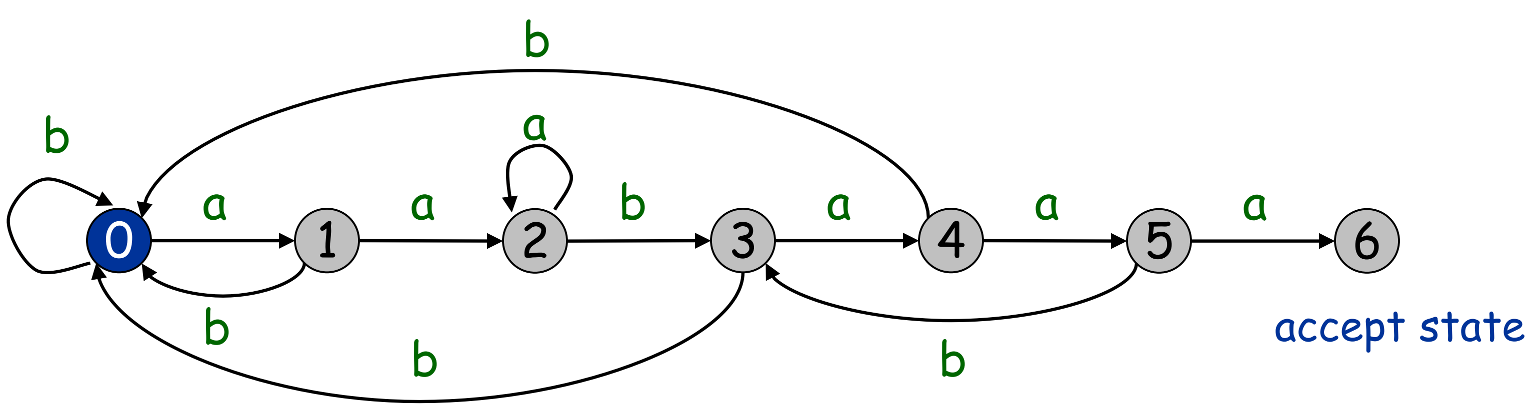


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

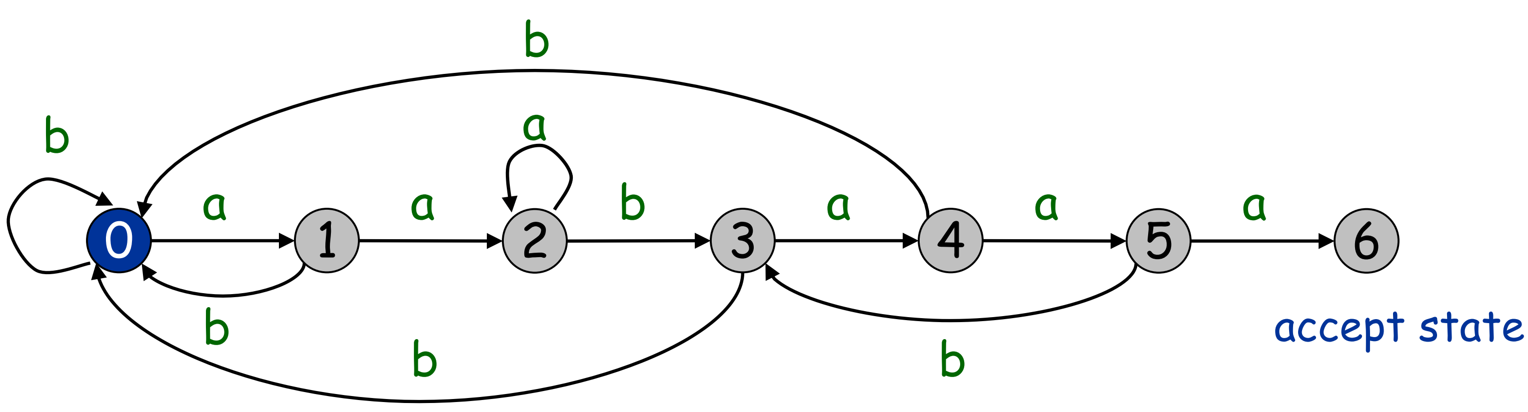


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

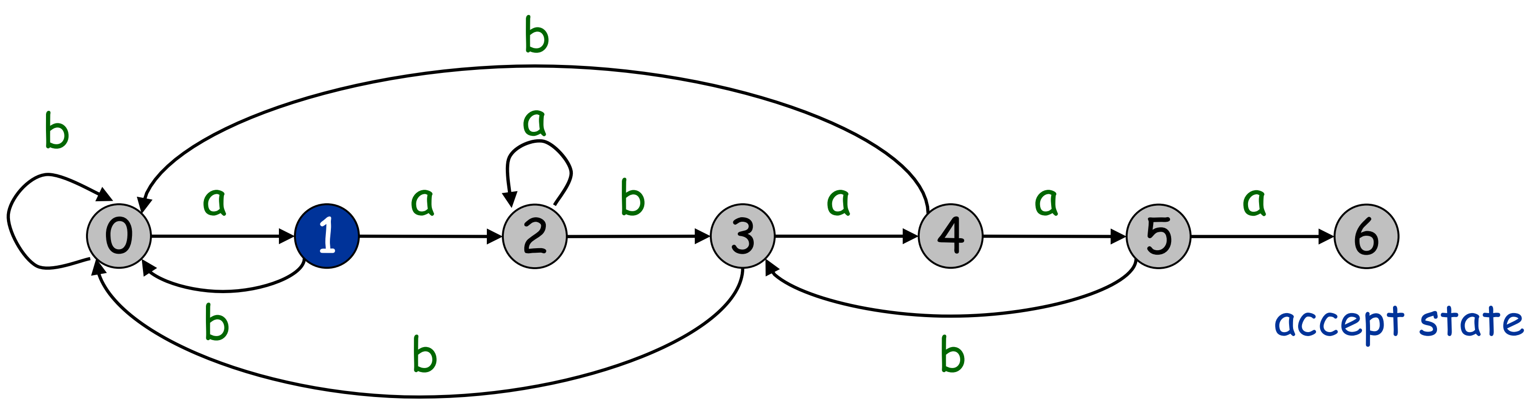


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build Finite State Automata (FSA) from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

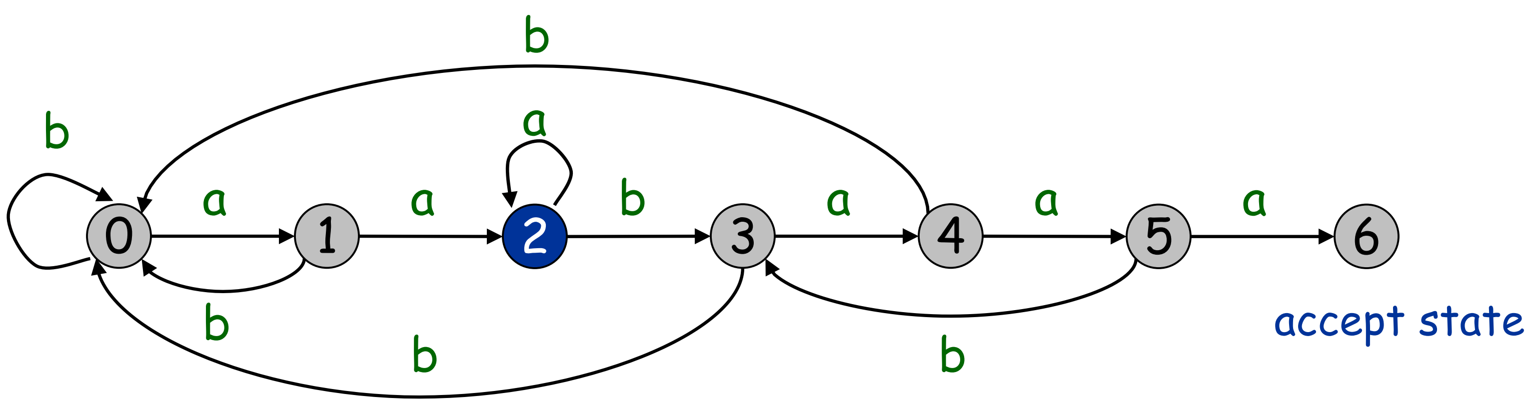


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

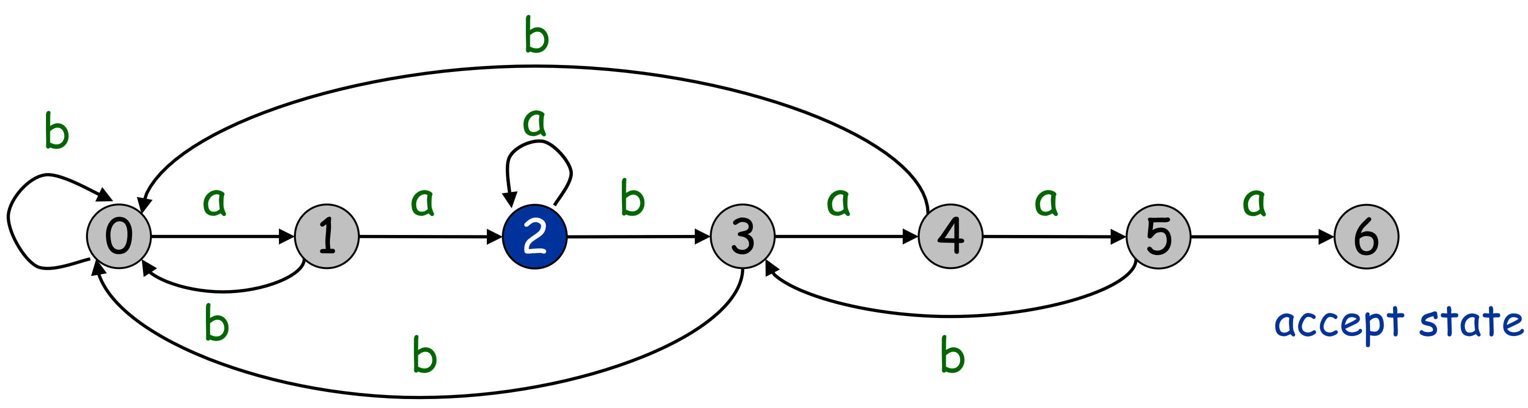


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

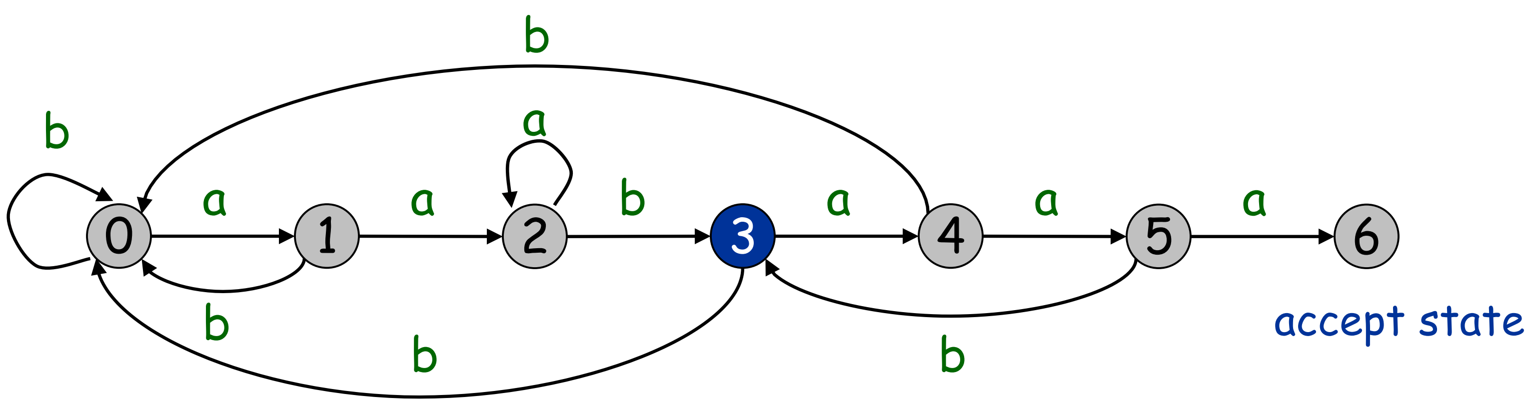


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

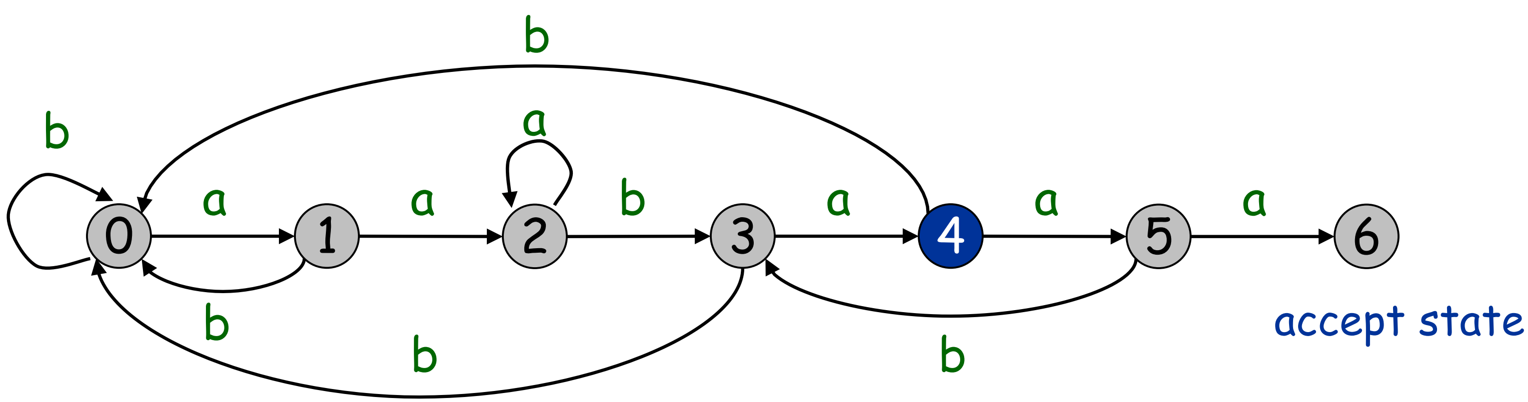


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

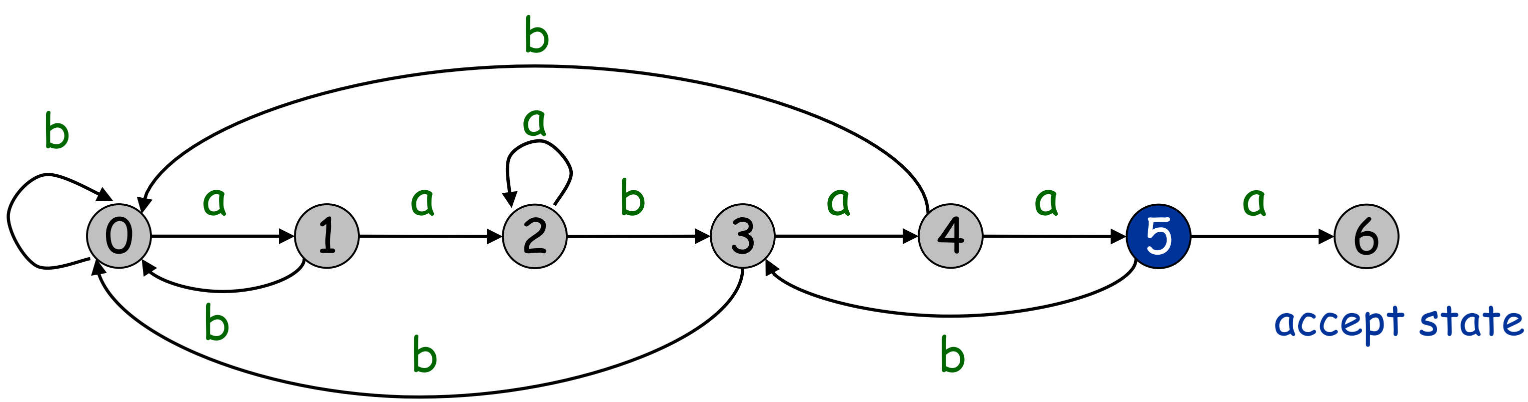


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

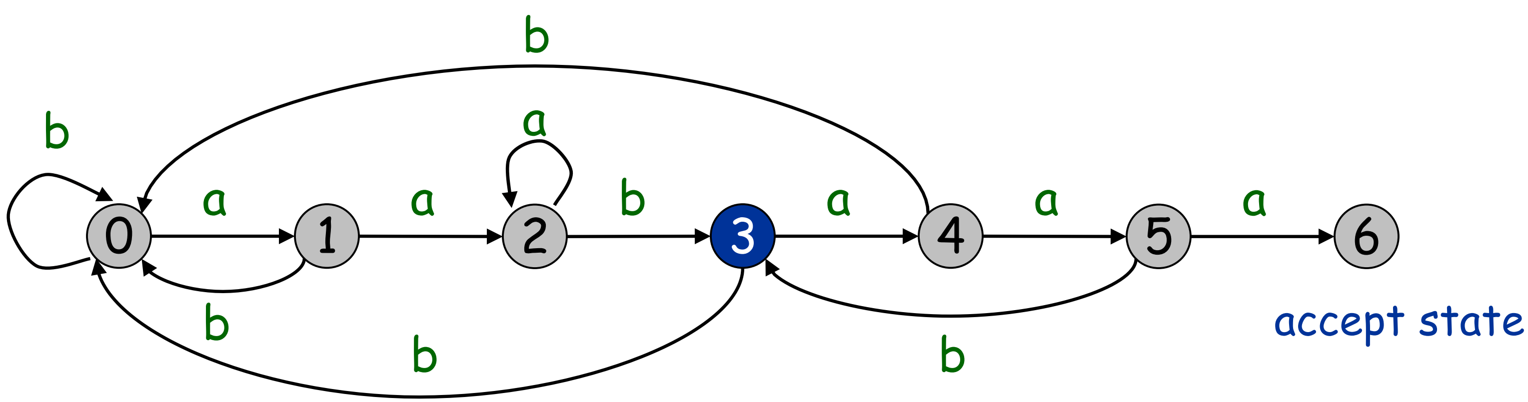


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

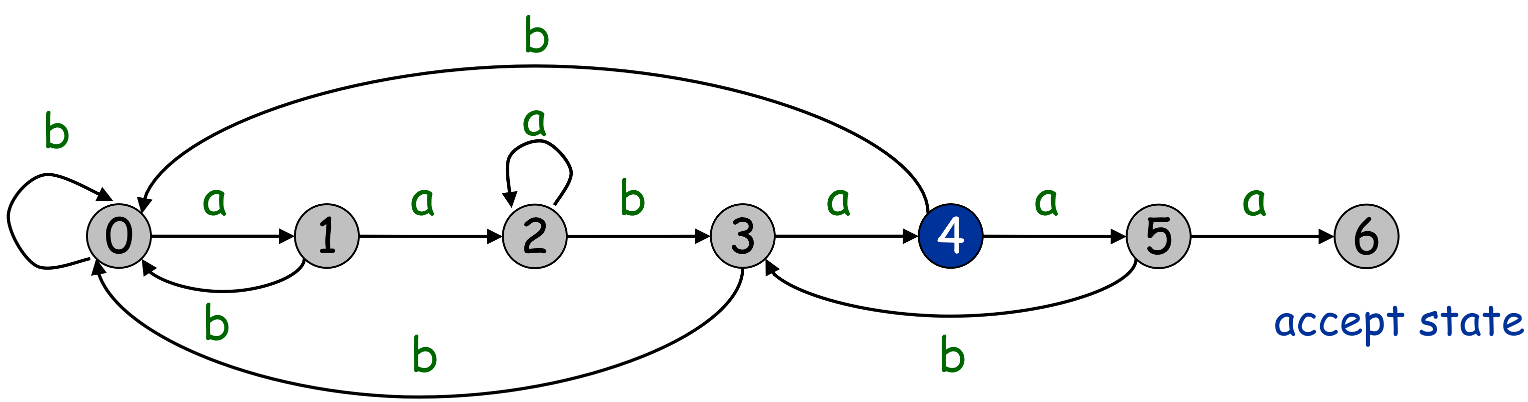


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

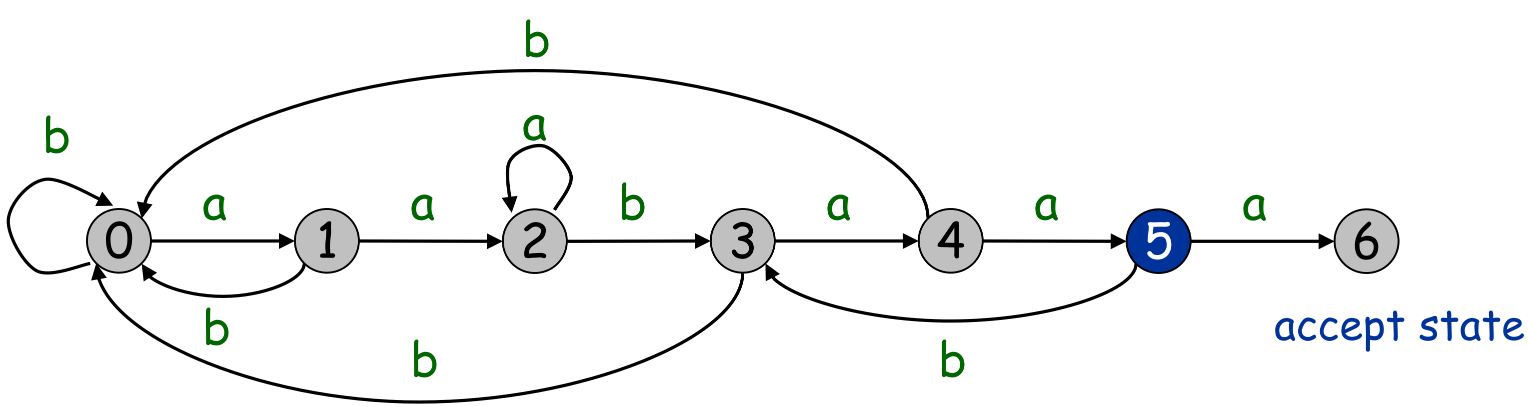


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | a | a | b | a | a | a | b |

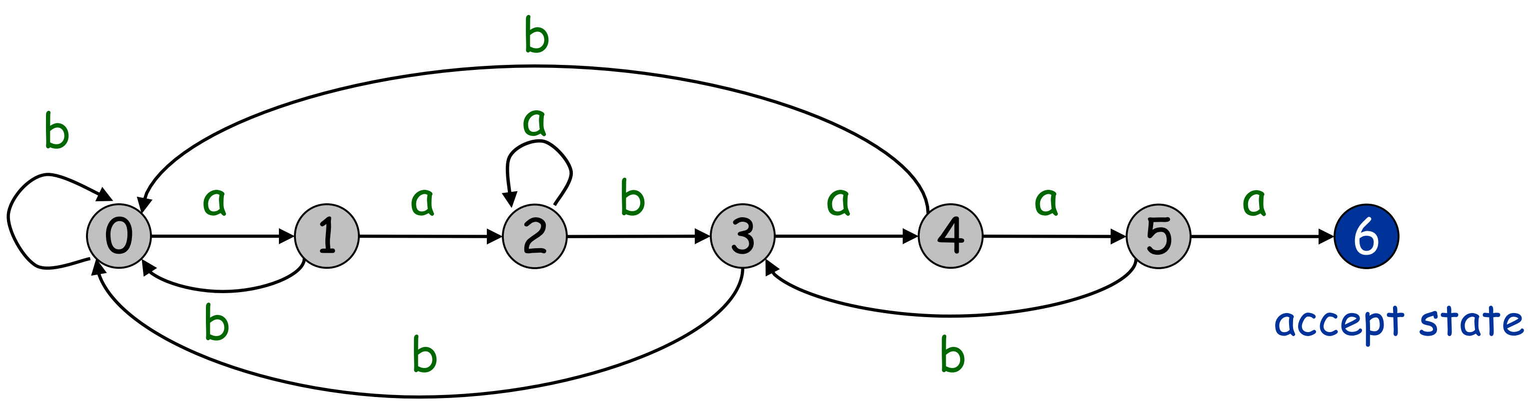


# Knuth-Morris-Pratt (KMP)

- KMP algorithm.
  - Use knowledge of how search pattern repeats itself.
  - Build FSA from pattern.
  - Run FSA on text.

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| Search Text | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| a | a | b | a | a | a | b | a | a | a | b |

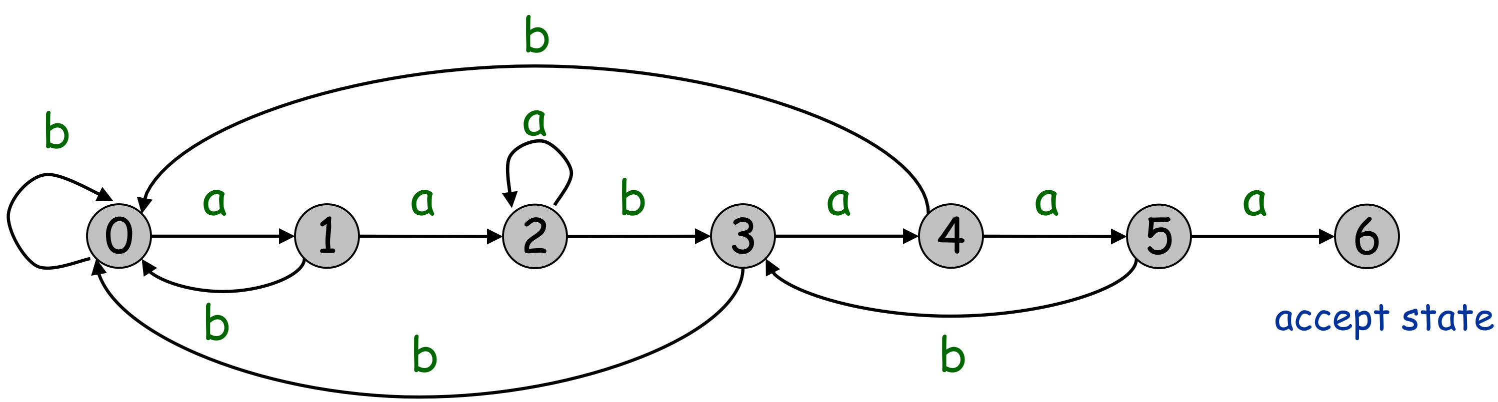


# Finite State Automata (FSA)

- FSA used in KMP has special property
  - If match, go to next state
  - Only need to keep track of where to go upon character mismatch.
    - go to state next[j] if character mismatches in state j

| Search Pattern | | | | | |
|---|---|---|---|---|---|
| a | a | b | a | a | a |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| a | 1 | 2 | 2 | 4 | 5 | 6 |
| b | 0 | 0 | 3 | 0 | 0 | 3 |
| next | 0 | 0 | 2 | 0 | 0 | 3 |



# KMP algorithm

- **Algorithm:** KMP(T, P):

```
n ← len(T), m ← len(P)
next ← FSA(P)
q ← 0 // q is the state of the FSA.
for i ← 0 to n-1
        if P[q] != T[i] then
                q ← next[q]
        else
                q ← q + 1
        if q = m
                pattern occurs with shift i – m
```

# Analysis of KMP

- **Algorithm:** KMP(T, P):

  Cost of Line 1:

  Cost of Line 2:

  Cost of Line 3:

  Cost of Line 4:

  …

  Cost of Line 11:

  Overall Cost:

# Our Roadmap

- String Concepts
- String Searching Problem
  - Brute Force Solution
  - Finite State Automata (FSA)
  - KMP Algorithm
  - FSA and Prefix Function

# History of KMP

- Inspired by the theorem of Cook that says O(m+n) algorithm should be possible
- Discovered in 1976 independently by two groups
- Knuth-Pratt
- Morris was hacker trying to build an editor
- Resolved theoretical and practical problem
  - Surprise when it was discovered
  - In hindsight, seems like right algorithm

# String

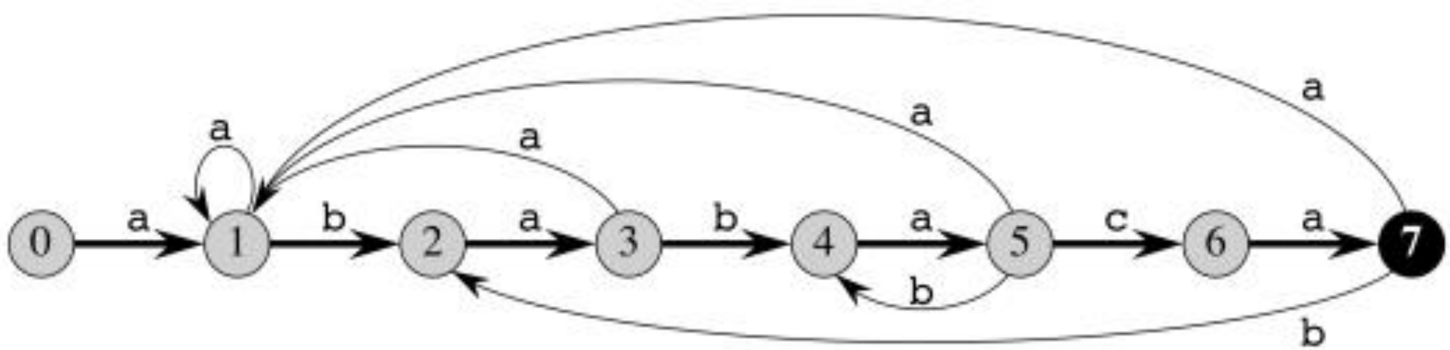
- **String**: “HelloCS203”
- **Substring**: a substring of s string S is a string S’ that occurs in S, e.g., P[1,…,3] = “ell”
- **Prefix (P[0,…])**: a prefix of a string S is a substring of S that occurs at the beginning of S, e.g., P[0,…,0] = “H” (note that P[0]=‘H’), P[0,…,1] = “He”, P[0,…,4] = “Hello”, we denote prefix as: **P[0,…]**
- **Suffix**: a suffix of a string S is a substring of S that occurs at the end of S, e.g., P[9,…,9]=“3”, P[7,…,9]=“203”, P[5,…,9] = “CS203”, we denote suffix as: **P[…,m]**

# Finite State Automata

- P ="ababaca"
- Transition function table

| State | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| a | 1 | 1 | 3 | 1 | 5 | 1 | 7 | 1 |
| b | 0 | 2 | 0 | 4 | 0 | 4 | 0 | 2 |
| c | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 0 |
| P | a | b | a | b | a | c | a | |

- State transition graph



# Finite State Automata

- P ="ababaca" and T ="abababacaba"

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| T | a | b | a | b | a | b | a | c | a | b | a |
| 1 | a | b | a | b | a | c | a | | | | |
| 2 | | | a | b | a | b | a | c | a | | |
| 3 | | | | | | | | | a | b | |

- After **failure**: at i=5, 'c' was expected, but not found in T[5], FSA transition to state $\delta(5,b)=4$, it means pattern prefix P[0..3] ="abab" has matched the text suffix T[2..5] = "abab"

- After **success**, at i=9, a "b" is seen, $\delta(7,b)=2$,
- thus, P[0..1] = T[8..9]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| a | 1 | 1 | 3 | 1 | 5 | 1 | 7 | 1 |
| b | 0 | 2 | 0 | 4 | 0 | 4 | 0 | 2 |
| c | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 0 |

# Finite State Automata

- In general, the FSA is constructed so that the state number tells us how much of a prefix of P has been matched.
- FSA transition function:
  - 1) Find the longest prefix of P is also a suffix of T[...,i], denote as m, i.e., *P[0,…,m]=T[i-m,…,i]*
  - 2) Read the next character at "*m+1*", there are two kinds of transitions:
    - P[m+1] = T[i+1], it is matched, continues.
    - Otherwise, it is mismatched, go to $\delta$(m+1,T[m+1])

# Prefix Function

- Consider the first step of FSA transition function:
  - Find the longest prefix of P is also a suffix of T[...i], note as *m*, i.e., *P[0,...,m]=T[i-m,...,i]*
- Suppose it is mismatched at *"m+1"*, it means:
  - *P[m+1] != T[i+1]*
  - then, we should find the longest prefix of *P[0,...,m]* is also a suffix of *T[i-m+1, ..., i].*
    - *Since P[0,...,m]=T[i-m,...,i], thus, P[1, ..., m] = T[i-m+1, ..., i]*
- **Prefix function (next array in lab),** given $P[0..m]$, the prefix function π for $P$ is π : {1, 2 ..., $m$} -> {0, 1, ..., $m$} such that:

$$\pi[q]=max\{k, k<q \text{ and } P[m-k,...,m] = P[0,..,k-1]\}$$

# Prefix Function

- **Prefix function (next array in lab),** given *P*, the prefix function π for *P* is π : {1, 2 ..., |P|} -> {0, 1, ..., |P|-1} such that:

$$\pi[i]=max\{k,\ k<i\ and\ P[0,...,k-1] = P[...,i]\}$$

- Example: P ="ababaca"

| *I* | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| P[i] | a | b | a | b | a | c | a |
| *π[i]* | -1/0 | 0 | 1 | 2 | 3 | 0 | 1 |

# Prefix Function (next)

- P ="ababaca" and T ="abababacaba"

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| T | a | b | a | b | a | b | a | c | a | b | a |
| 1 | a | b | a | b | a | c | a | | | | |
| 2 | | | a | b | a | b | a | c | a | | |

- After **failure**: at i=5, 'c' was expected, but not found in T[5], then we lookup π[4] = 3

| *i* | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| P[i] | a | b | a | b | a | c | a |
| *π[i]* | -1 | 0 | 1 | 2 | 3 | 0 | 1 |

# Our Roadmap

- String Concepts
- String Searching Problem
  - Brute Force Solution
  - Finite State Automata (FSA)
  - KMP Algorithm
  - FSA and Prefix Function

# Thank You!