Winter 2016
COMP-250: Introduction to Computer Science

Lecture 23, April 5, 2016

Comment about input size…

2)
Write *any* algorithm that runs in time $\Theta(n^2 \log^2 n)$ in worse case.
Explain why this is its running time. I don't care what it does.
I only care about its running time…

```
WhatEver(int m)

FOR i=1 TO m
  FOR j=1 TO m
    x=m; WHILE x>1 DO { x=x/2; y=m;
                        WHILE y>1 DO y=y/2 }
```

n = |m| ~ log m. Therefore running time is $\Theta(m^2 \log^2 m) = \Theta(2^{2n} n^2)$

Comment about input size…

2)
Write *any* algorithm that runs in time $\Theta(n^2 \log^2 n)$ in worse case.
Explain why this is its running time. I don't care what it does.
I only care about its running time…

```
WhatEver(int[] A)

n = A.length;
FOR i=1 TO n
  FOR j=1 TO n
      x=n; WHILE x>1 DO { x=x/2; y=n;
                          WHILE y>1 DO y=y/2 }
```

Public Announcement

Mercury Course Evaluations

Course evaluations matter. Evaluate your courses and instructors!

Default period:
March 21 - May 1

Condensed period:
March 21 - April 17

Click **HERE** to complete your course evaluations.

STRINGS AND PATTERN MATCHING

- Brute Force,Rabin-Karp, Knuth-Morris-Pratt
- Regular Expressions

The Knuth-Morris-Pratt Algorithm

- The Knuth-Morris-Pratt (KMP) string searching algorithm differs from the brute-force algorithm by keeping track of information gained from previous comparisons.
- A failure function (*f*) is computed that indicates how much of the last comparison can be reused if it fails.
- Specifically, *f* is defined to be the longest prefix of the pattern P[0,..,j] that is also a suffix of P[1,..,j]
  - Note: **not** a suffix of P[0,..,j]

The Knuth-Morris-Pratt Algorithm

- Specifically, *f* is defined to be the longest prefix of the pattern P[0,..,j] that is also a suffix of P[1,..,j]
  - Note: **not** a suffix of P[0,..,j]
- Example:
  - value of the KMP failure function:

j	0	1	2	3	4	5
P[*j*]	a	b	a	b	a	c
f(*j*)	0	0	1	2	3	0

- This shows how much of the beginning of the string matches up to the portion immediately preceding a failed comparison.
  - if the comparison fails at (4), we know the a,b in positions 2,3 is identical to positions 0,1

The KMP Algorithm (contd.)

- the KMP string matching algorithm: Pseudo-Code

```
Algorithm KMPMatch(T,P)
  Input: Strings T (text) with n characters and P
    (pattern) with m characters.
  Output: Starting index of the first substring of T
    matching P, or an indication that P is not a
    substring of T.

  f ← KMPFailureFunction(P) {build failure function}
  i ← 0
  j ← 0
  while i < n do
    if P[j] = T[i] then
      if j = m - 1 then
        return i - m - 1 {a match}
      i ← i + 1
      j ← j + 1
    else if j > 0 then {no match, but we have advanced}
      j ← f(j-1) {j indexes just after matching prefix in P}
    else
      i ← i + 1
  return "There is no substring of T matching P"
```

The KMP Algorithm (contd.)

- The KMP failure function: Pseudo-Code

Algorithm KMPFailureFunction(P);

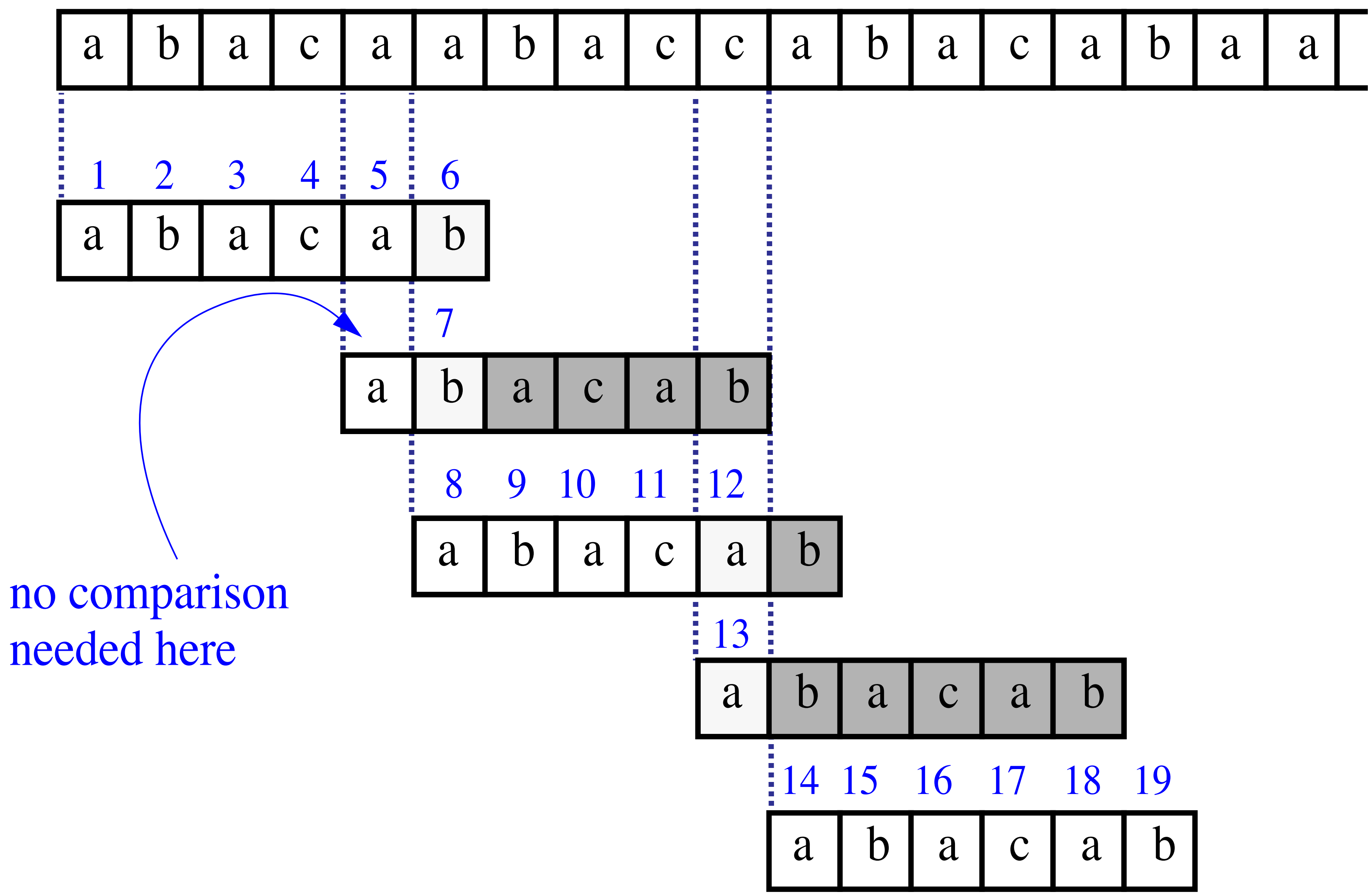
Input: String P (pattern) with m characters

Ouput: The faliure function f for P, which maps j to the length of the longest prefix of P that is a suffix of $P[1,..,j]$

```
i ← 1
j ← 0
while i ≤ m-1 do
   if P[j] = P[i] then
      {we have matched j + 1 characters}
      f(i) ← j + 1
      i ← i + 1
      j ← j + 1
   else if j > 0 then
      {j indexes just after a prefix of P that matches}
      j ← f(j-1)
   else
      {there is no match}
      f(i) ← 0
      i ← i + 1
```

The KMP Algorithm (contd.)

- A graphical representation of the KMP string searching algorithm



The KMP Algorithm (contd.)

- Time Complexity Analysis
- define $k = i - j$
- In every iteration through the while loop, one of three things happens.
  - 1) if $T[i] = P[j]$, then i increases by 1, as does j
    k remains the same.
  - 2) if $T[i] \mathrel{!=} P[j]$ and $j > 0$, then i does not change and k increases by at least 1, since k changes from $i - j$ to $i - f(j-1)$
  - 3) if $T[i] \mathrel{!=} P[j]$ and $j = 0$, then i increases by 1 and k increases by 1 since j remains the same.

The KMP Algorithm (contd.)

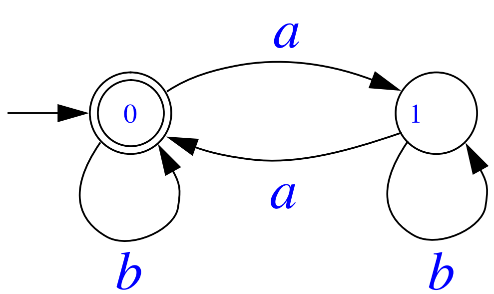
- Thus, each time through the loop, either i or k increases by at least 1, so the greatest possible number of loops is $2n$
- This of course assumes that f has already been computed.
- However, f is computed in much the same manner as KMPMatch so the time complexity argument is analogous. KMPFailureFunction is $\boldsymbol{O}(m)$
- Total Time Complexity: $\boldsymbol{O}(n + m)$

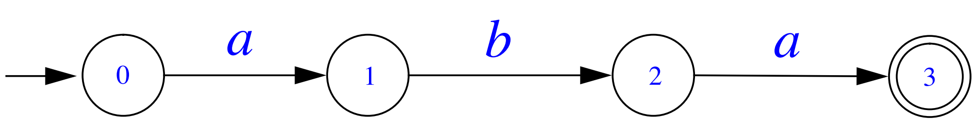
Regular Expressions

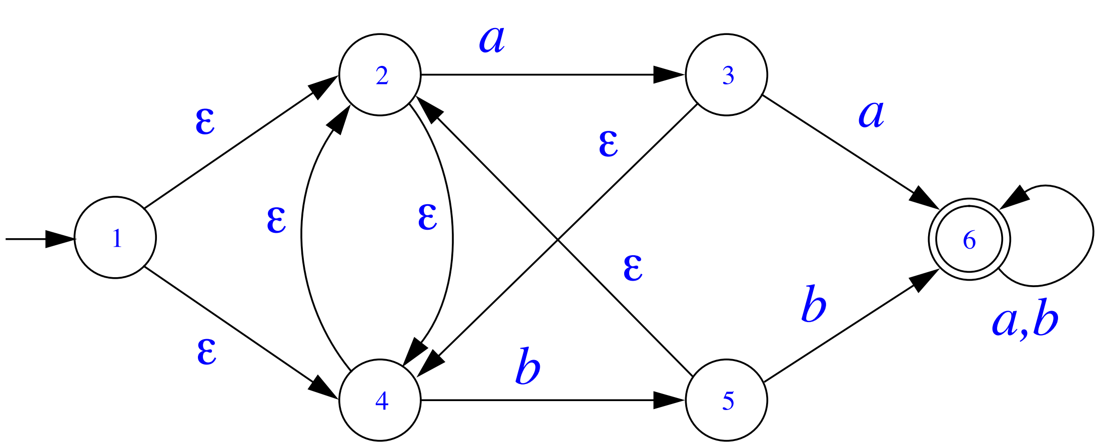
- notation for describing a set of strings, possibly of infinite size
- **ε** denotes the empty string
- **ab + c** denotes the set {ab, c}
- **a*** denotes the set {ε, a, aa, aaa, ...}
- Examples
  - **(a+b)*** all the strings from the alphabet {a,b}
  - **b*(ab*a)*b*** strings with an even number of a's
  - **(a+b)*sun(a+b)*** strings containing the pattern "sun"
  - **(a+b)(a+b)(a+b)a** 4-letter strings ending in a

Finite State Automaton

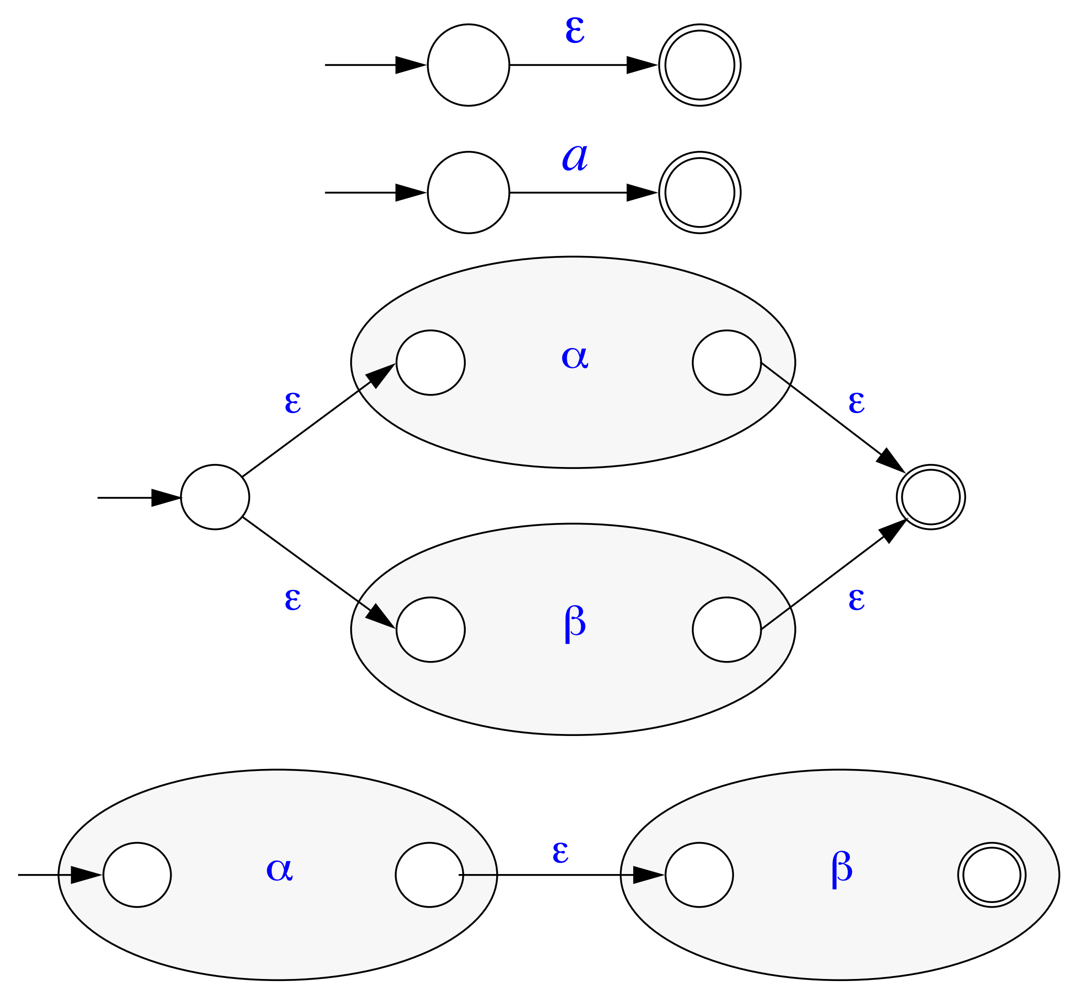
- “machine” for processing strings

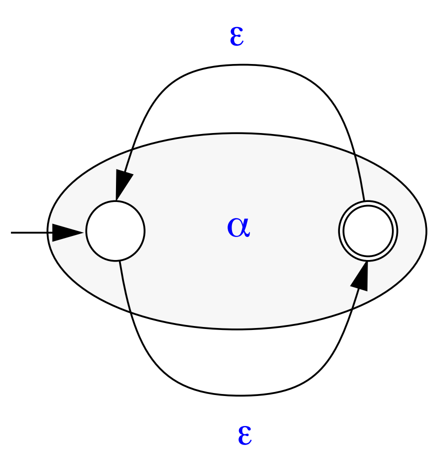






Composition of FSA’s





Winter 2016
COMP-250: Introduction to Computer Science

Lecture 23, April 5, 2016