>>

Boyer-Moore String Matching

∧ >>

❖ String Matching

String matching problem

- given a string T of n chars from alphabet Σ
- given a pattern P of m chars from alphabet Σ, where $m \le n$
- find position in T where P occurs

Example:

```
T = i l i k e p a t t e r n s
P = p a t
```

a match occurs when T and P are aligned as follows

```
T = i l i k e p a t t e r n s
P =           p a t
```

<< ∧ >>

❖ ... String Matching

A naive approach to solving this problem works as follows

```
T = i l i k e p a t t e r n s
P = p a t

T = i l i k e p a t t e r n s
P =   p a t

T = i l i k e p a t t e r n s
P =     p a t

......

T = i l i k e p a t t e r n s
P =           p a t
```

<< ∧ >>

❖ Boyer-Moore Algorithm

The Boyer-Moore string matching algorithm

- aims to do less char comparisons than the naive version
- by moving the pattern more than one position after each fail

It is based on two heuristics:

- Looking-glass heuristic
  - compare P with subsequence of T moving *backwards*
- Character-jump heuristic
  - move forward more than one position at a time
  - depending on where pattern matching failed

❖ ... Boyer-Moore Algorithm

Boyer-Moore algorithm preprocesses pattern *P* and alphabet Σ

- to build a last-occurrence function *L*

L maps Σ to integers such that *L(c)* is defined as

- the largest index *i* such that *P[i]*=*c*, or
- -1 if no such index exists

Example: Σ = {…,**a**,**b**,**c**,**d**,**e**,**f**,…}, *P* = **acab**

c	...	**a**	**b**	**c**	**d**	**e**	**f**	...
L(c)	...	2	3	1	-1	-1	-1	...

L can be represented by an array indexed by the ascii codes of the chars

<< ∧ >>

❖ ... Boyer-Moore Algorithm

The **`lastOccurences`** function to build *L*

```
intArray lastOccurences(P, Σ):
|   Input  pattern string P, alphabet Σ
|   Output array containing last
|             position of each character in pattern
|          characters not in pattern have "position" -1
|
|   L = make array of size |Σ|
|   m = length(P)
|   // set all values in L to -1
|   for each ch ∈ Σ do
|      L[ch] = -1
|   end for
|   for each i = 0 .. m-1 do
|      L[P[i]] = i
|   end for
|   return L
```

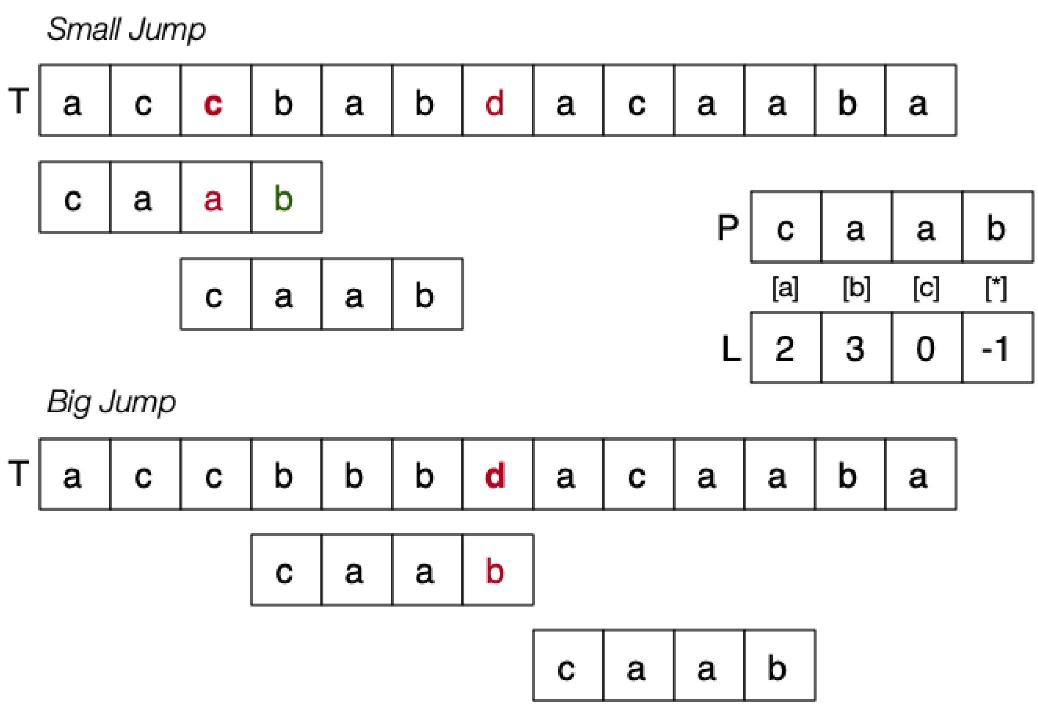
<< ∧ >>

❖ ... Boyer-Moore Algorithm

When a mismatch occurs at *T[i]* = *ch* ...

- if *P* contains *ch* ⇒ shift *P* to align the last occurrence of *ch* in *P* with *T[i]*
- otherwise ⇒ shift *P* to align *P[0]* with *T[i+1]* (a.k.a. "big jump")

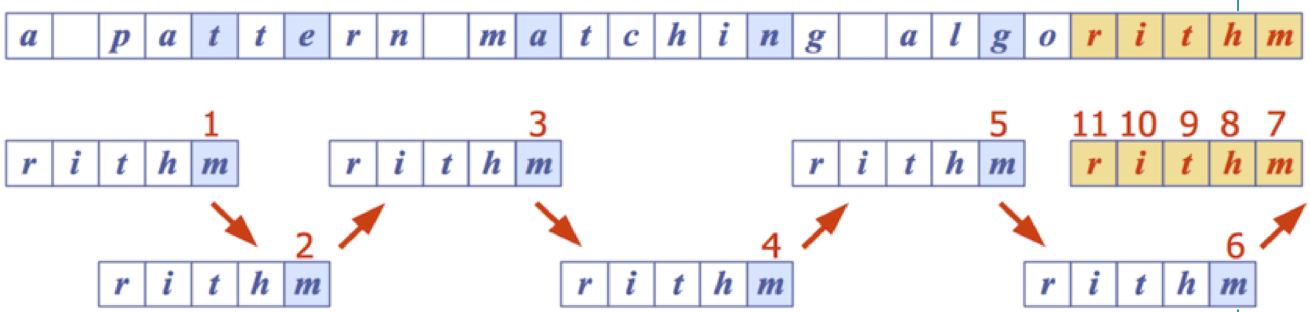
Examples:



<< ∧ >>

❖ ... Boyer-Moore Algorithm

A complete example of matching with multiple "big jumps":



<< ^ >>

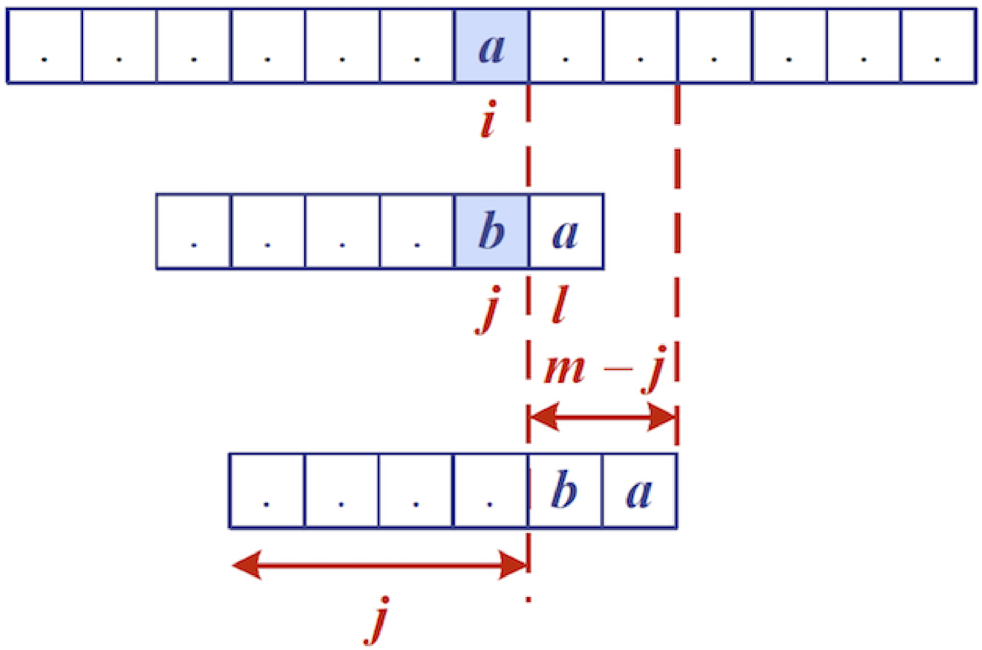
❖ ... Boyer-Moore Algorithm

```
int BoyerMooreMatch(T,P,Σ):
|  Input  text T of length n, pattern P of length m, alphabet Σ
|  Output starting index of a substring of T equal to P
|         -1 if no such substring exists
|
|  L=lastOccurences(P,Σ)
|  i=m-1, j=m-1                  // start at end of pattern
|  repeat
|  |  if T[i]=P[j] then
|  |     if j=0 then
|  |        return i             // match found at i
|  |     else
|  |        i=i-1, j=j-1
|  |     end if
|  |  else                       // character-jump
|  |     i=i+m-min(j,1+L[T[i]])
|  |     j=m-1
|  |  end if
|  until i≥n
|  return -1                     // no match
```

<< ^ >>

❖ ... Boyer-Moore Algorithm

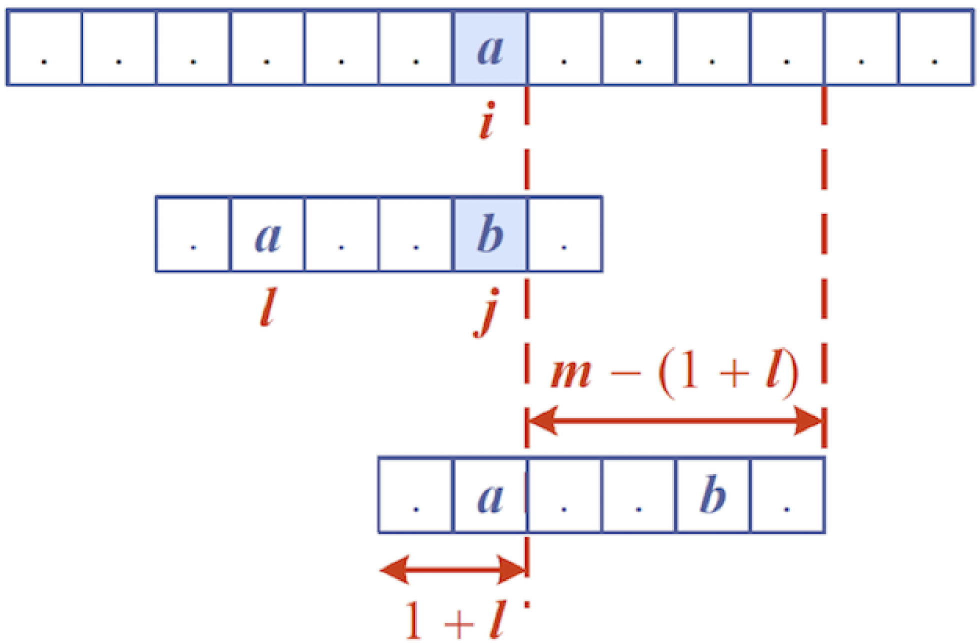
Case 1: $j \le 1+L[c]$



<< ^ >>

❖ ... Boyer-Moore Algorithm

Case 2: *1+L[c] < j*



<< ∧ >>

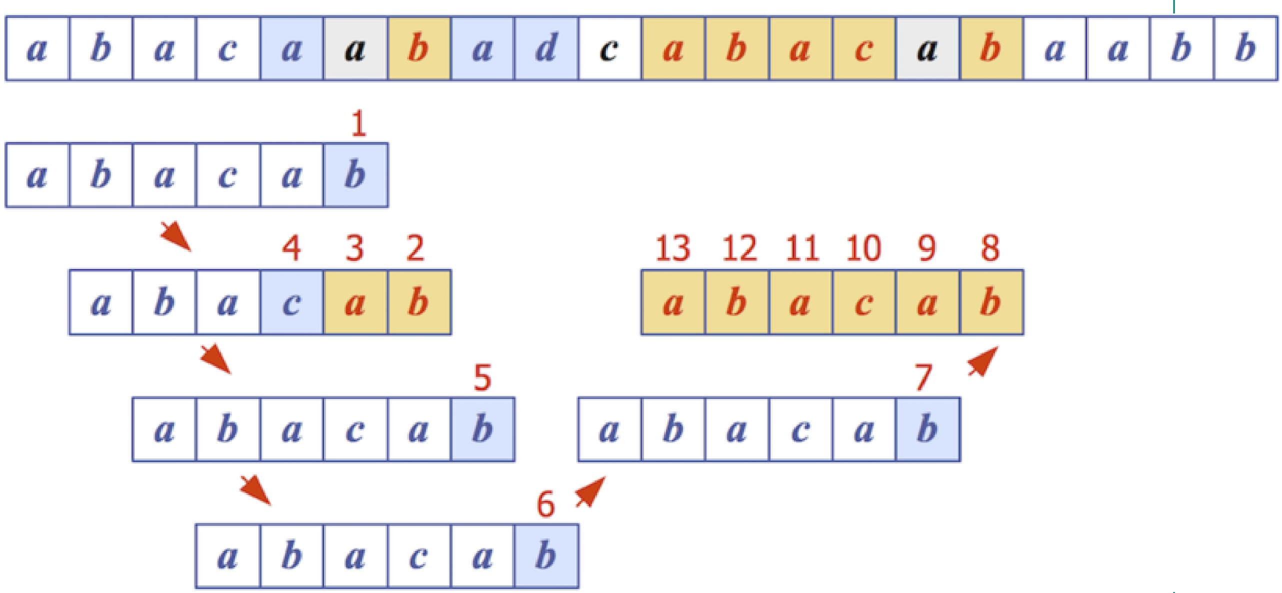
❖ Example Execution

For the alphabet Σ = {**a,b,c,d**} and *P* = **abacab** ...

1. compute the last-occurrence table *L*

c	a	b	c	d
L(c)	4	5	3	-1

2. count comparisons searching for *P* in *T* = **abacaabadcabacabaabb**



❖ Analysis of Algorithm

Reminder:

- *m* ... length of pattern *n* ... length of text *s* ... size of alphabet

Analysis of Boyer-Moore algorithm:

- pre-processing: *L* can be computed in *O(m+s)* time
- matching part: runs in *O(nm)* time

Example of worst case: *T* = **aaa ... a** *P* = **baaa**

Worst case may occur in images or DNA sequences but unlikely in text

Boyer-Moore significantly faster than brute-force on English text

Produced: 8 Aug 2020