

String Matching

- Have a pattern string p of length m
- Have a text string t of length n
- Can we find an index i of string t such that each of the m characters in the substring of t starting at i matches each character in p
 - Example: can we find the pattern "fox" in the text "the quick brown fox jumps over the lazy dog"?
 - Yes! At index 16 of the text string!

Simple approach

- BRUTE FORCE
 - Start at the beginning of both pattern and text
 - Compare characters left to right
 - Mismatch?
 - Start again at the 2nd character of the text and the beginning of the pattern...

Brute force code

```
public static int bf_search(String pat, String txt) {  
    int m = pat.length();  
    int n = txt.length();  
    for (int i = 0; i <= n - m; i++) {  
        int j;  
        for (j = 0; j < m; j++) {  
            if (txt.charAt(i + j) != pat.charAt(j))  
                break;  
        }  
        if (j == m)  
            return i; // found at offset i  
    }  
    return n; // not found  
}
```

Brute force Algorithm

i:	0
text:	A B A B A B A C
pattern:	A B A B A C
j:	0

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:	0	1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0	1						

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		1	2					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:		1	2					

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		2	3					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:		2	3					

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:			3	4				
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:			3	4				

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:					4	5		
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:					4	5		

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:	0					5	
text:	A	B	A	B	A	B	A
pattern:	A	B	A	B	A	C	
j:	0					5	

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:	0	1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		1	2					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		2	3					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0	1						

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:			3	4				
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:		1	2					

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:				4	5		
text:	A	B	A	B	A	B	A
pattern:	A	B	A	B	A	C	
j:			2	3			

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:					5	6	
text:	A	B	A	B	A	B	A
pattern:	A	B	A	B	A	C	
j:				3	4		

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:							6	7
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:					4	5		

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:								7	8
text:	A	B	A	B	A	B	A	C	
pattern:	A	B	A	B	A	C			
j:							5	6	

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:



8

text:

A B A B A B A C

pattern:

A B A B A C

j:

6

```
public static int bf_search(String pat, String txt) {  
    int j, m = pat.length();  
    int i, n = txt.length();  
    for (i = 0, j = 0; i < n && j < m; i++) {  
        if (txt.charAt(i) == pat.charAt(j))  
            j++;  
        else { i -= j; j = 0; }  
    }  
    if (j == m)  
        return i - m; // found at offset i  
    else return n; // not found  
}
```

Brute force analysis

- Runtime?
 - What does the worst case look like?
 - $t = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXY$
 - $p = XXXY$
 - $m(n - m + 1)$
 - $\Theta(nm)$ if $n \gg m$
 - Is the average case runtime any better?
 - Assume we mostly mismatch on the first pattern character
 - $\Theta(n + m)$
 - $\Theta(n)$ if $n \gg m$

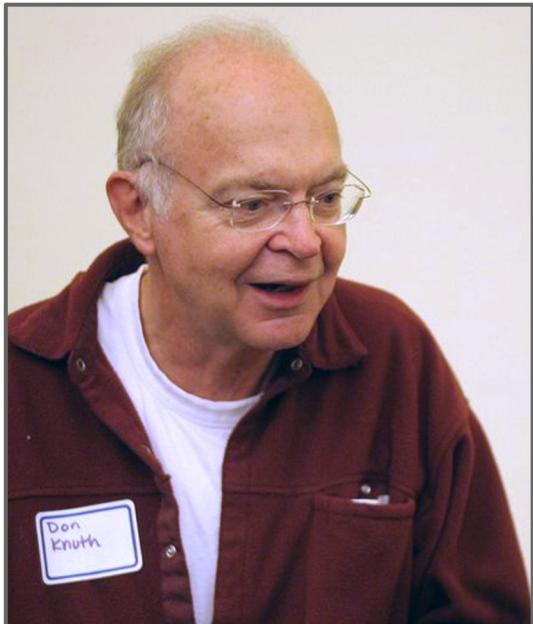
Where do we improve?

- Improve worst case
 - Theoretically very interesting
 - Practically doesn't come up that often for human language
- Improve average case
 - Much more practically helpful
 - Especially if we anticipate searching through large files

First: improving the worst case

Discovered the same algorithm independently

Knuth



Morris



Pratt

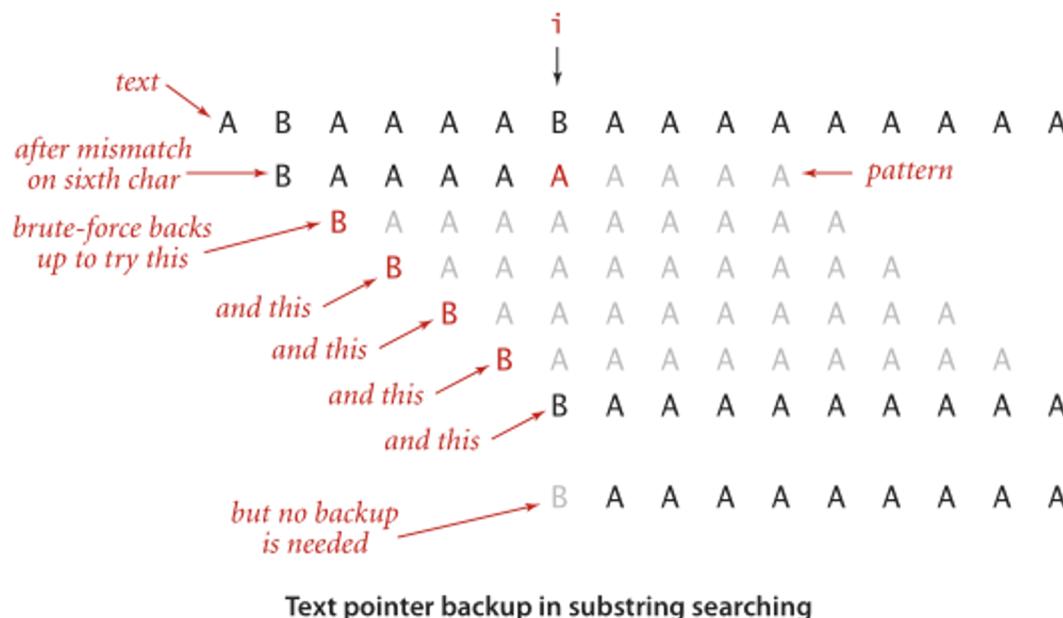


Worked together

Jointly published in 1976

Back to improving the worst case

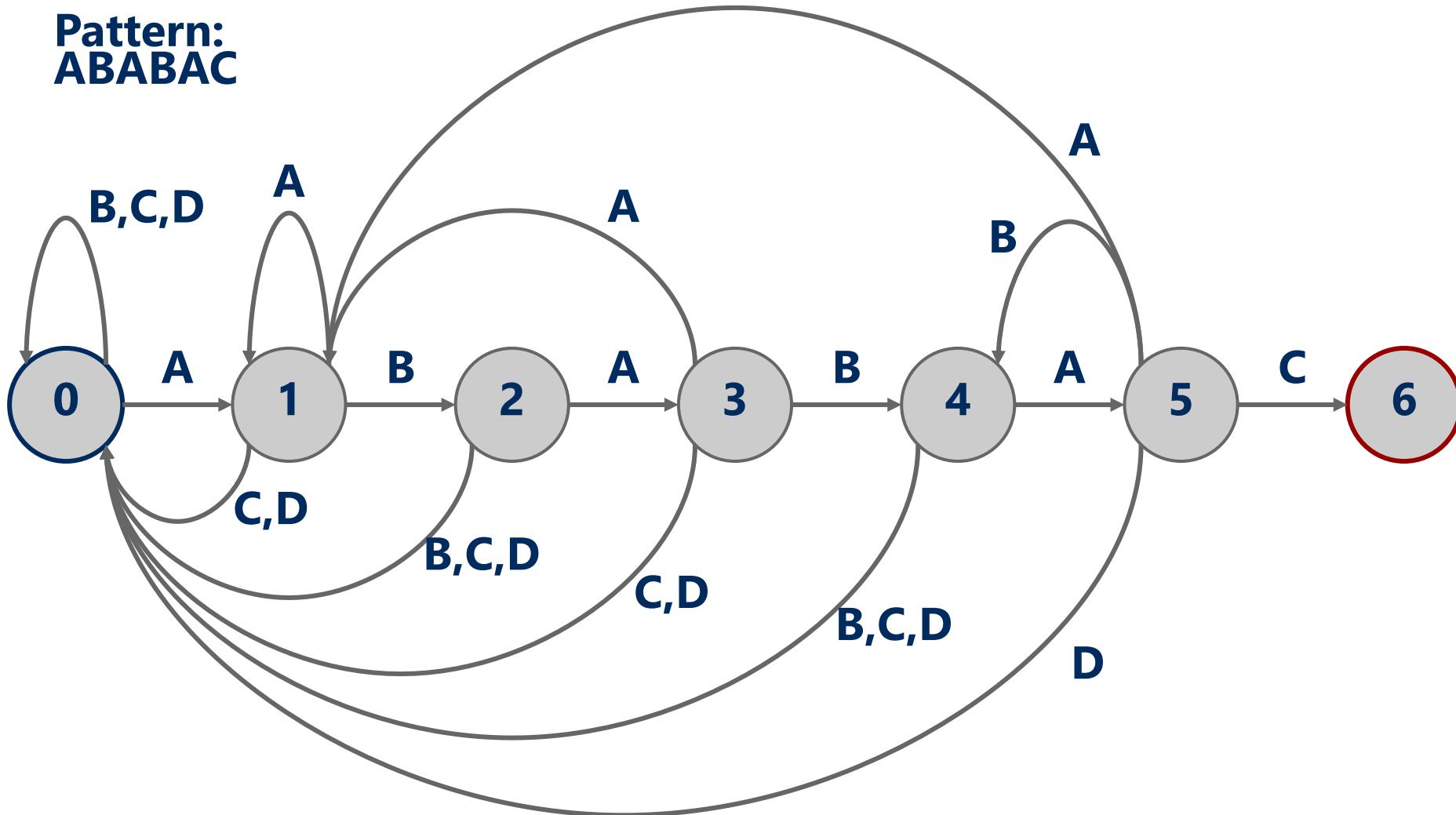
- Knuth Morris Pratt algorithm (KMP)
- Goal: avoid backing up in the text string on a mismatch
- Main idea: In checking the pattern, we learned something about the characters in the text, take advantage of this knowledge to avoid backing up

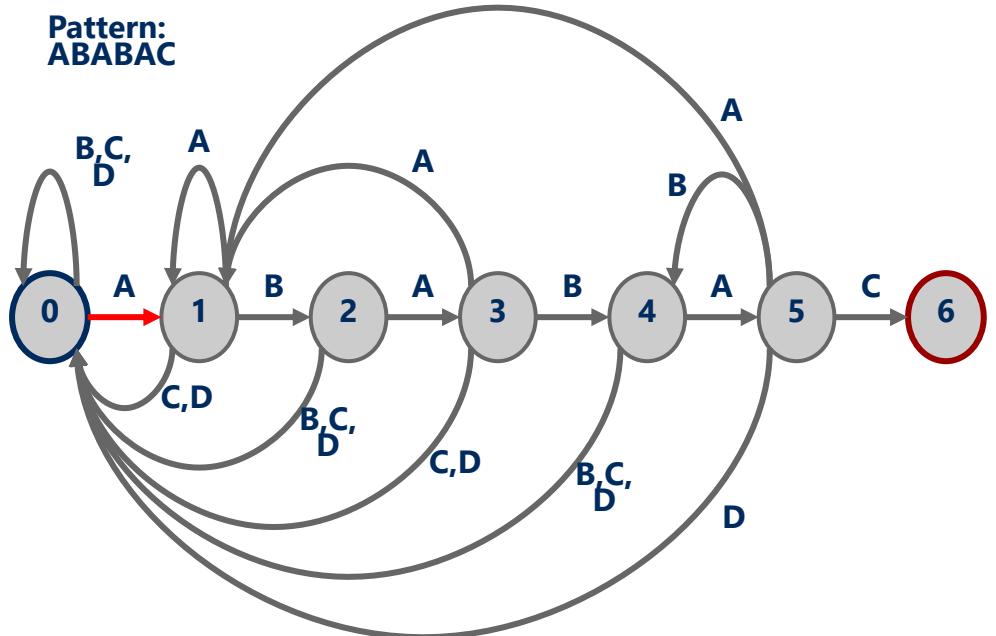


How do we keep track of text processed?

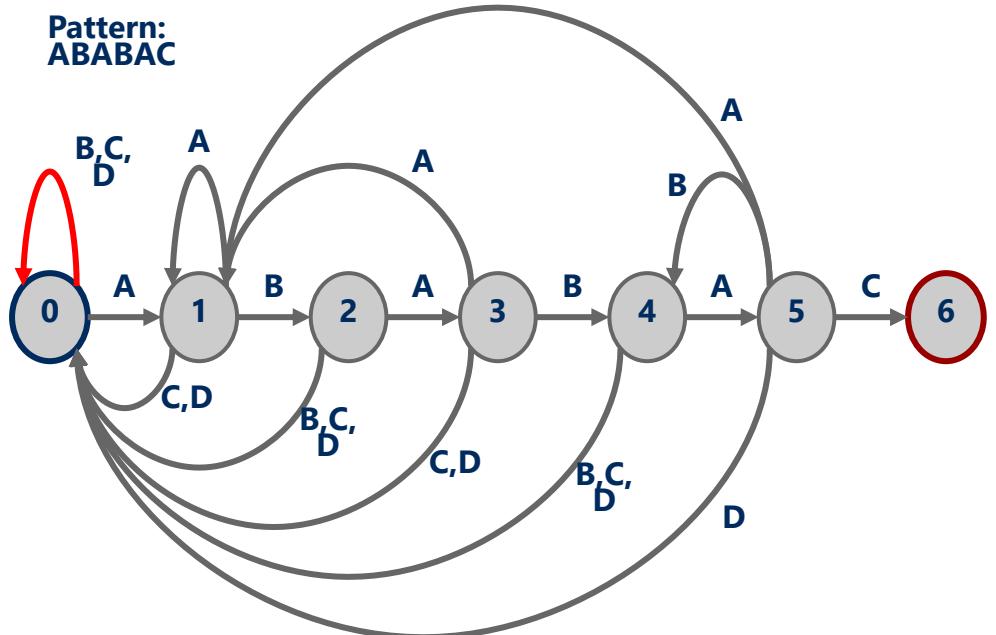
- Actually, build a deterministic finite-state automata (DFA) storing information about the *pattern*
 - From a given state in searching through the pattern, if you encounter a mismatch, how many characters currently match from the beginning of the pattern

DFA example

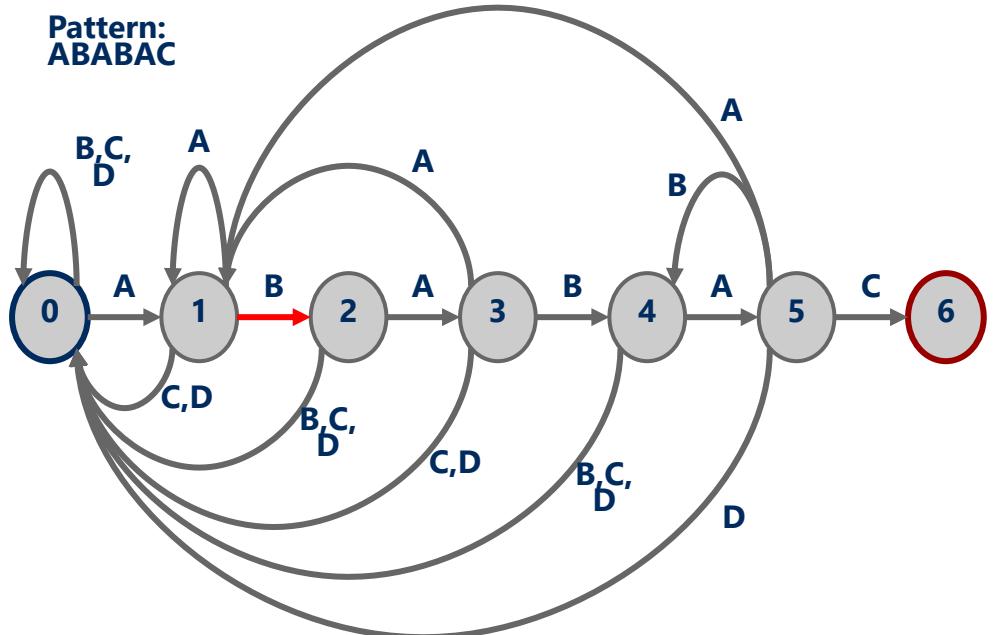




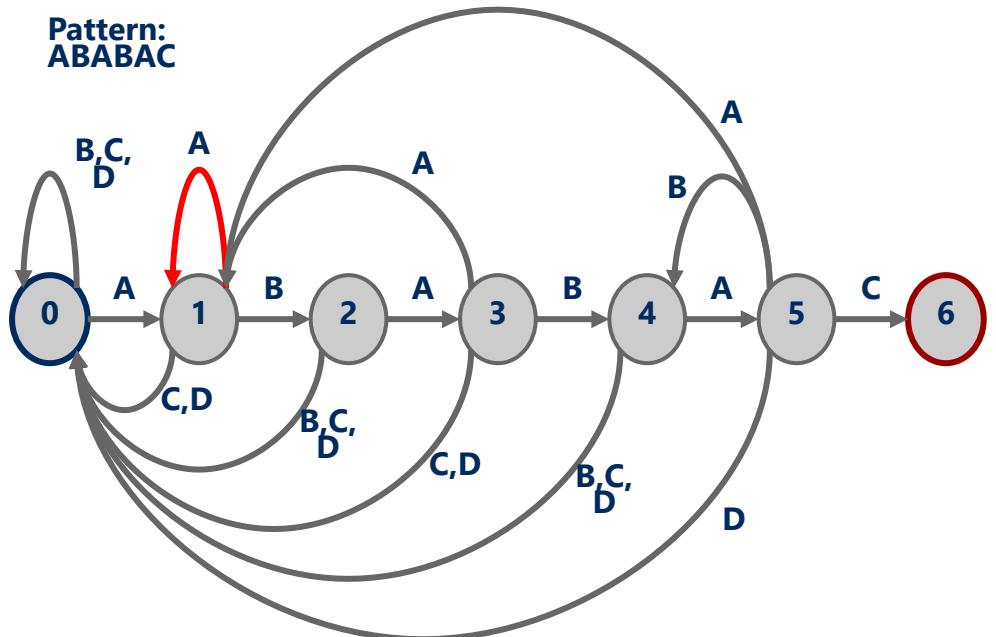
	0	1	2	3	4	5
A	1					
B						
C						
D						



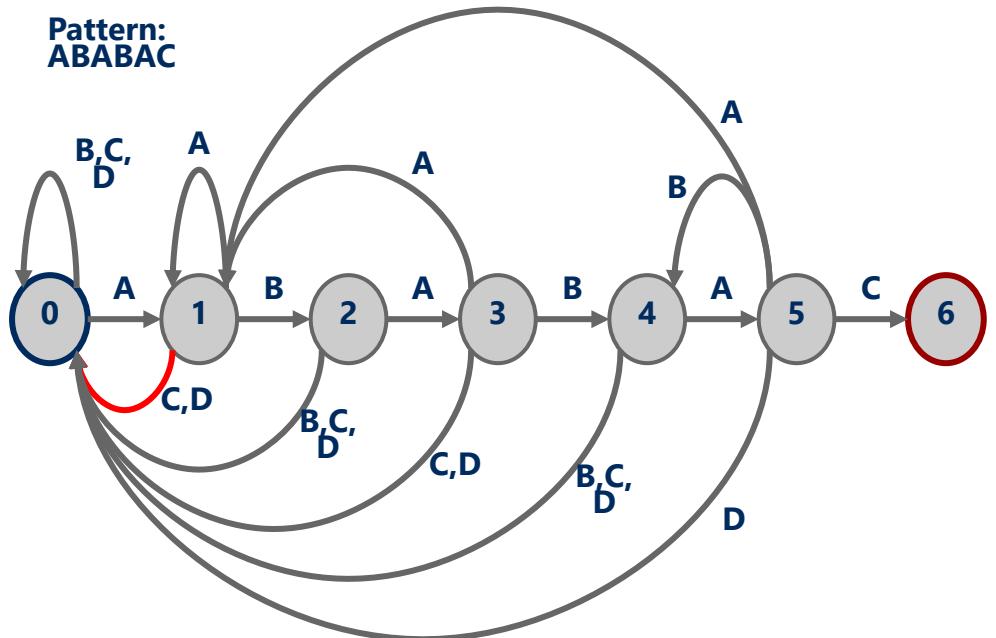
	0	1	2	3	4	5
A	1					
B	0					
C	0					
D	0					



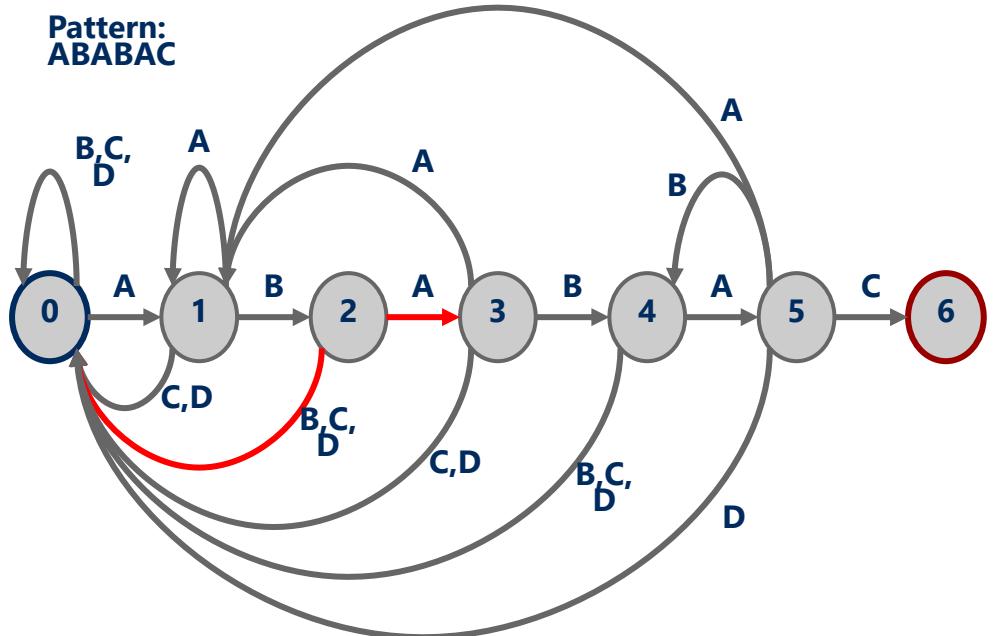
	0	1	2	3	4	5
A	1					
B	0	2				
C	0					
D	0					



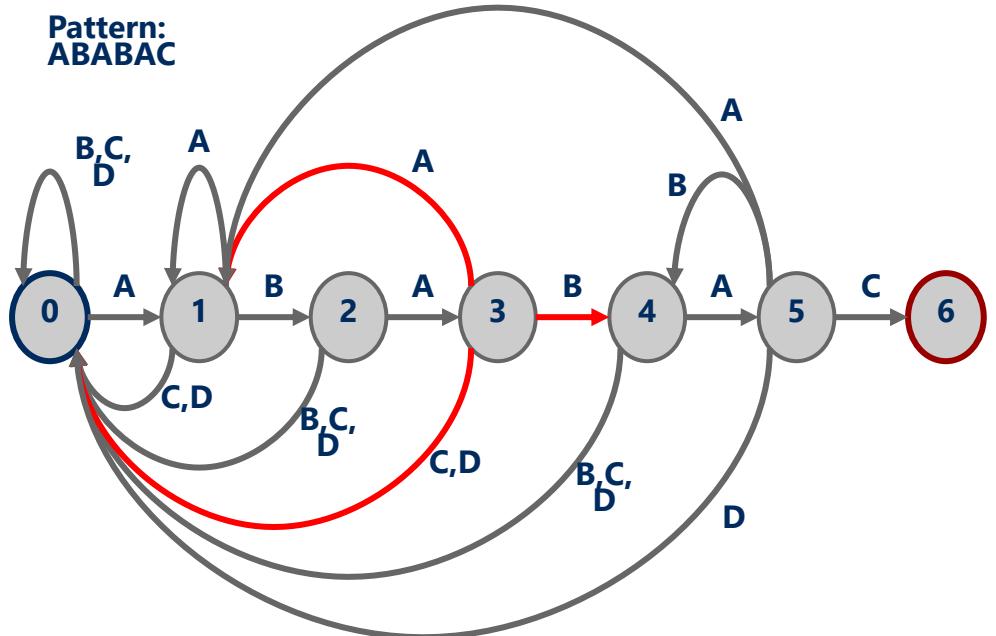
	0	1	2	3	4	5
A	1	1				
B	0	2				
C	0					
D	0					



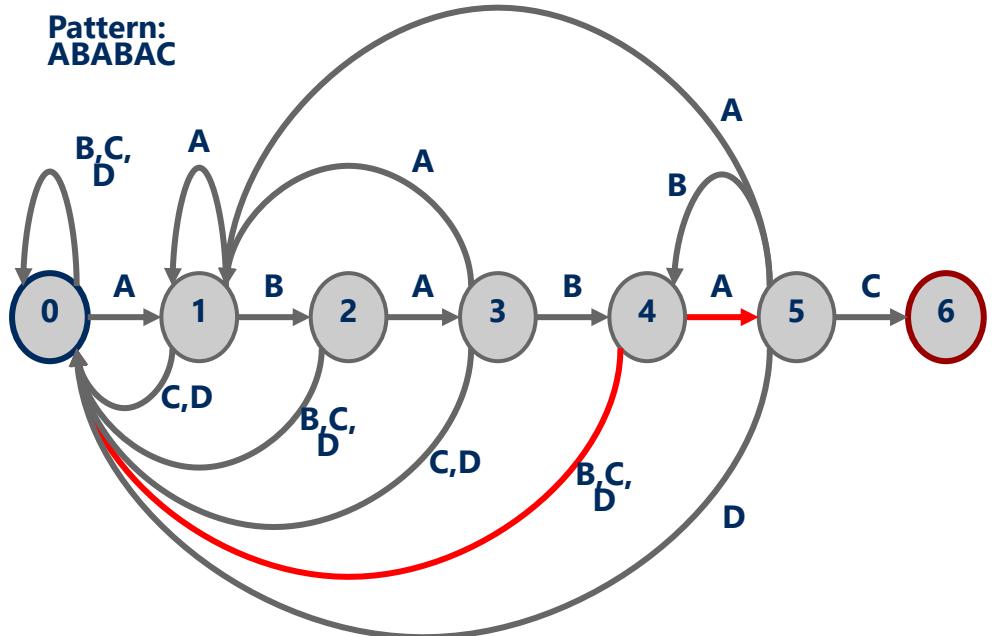
	0	1	2	3	4	5
A	1	1				
B	0	2				
C	0	0				
D	0	0				



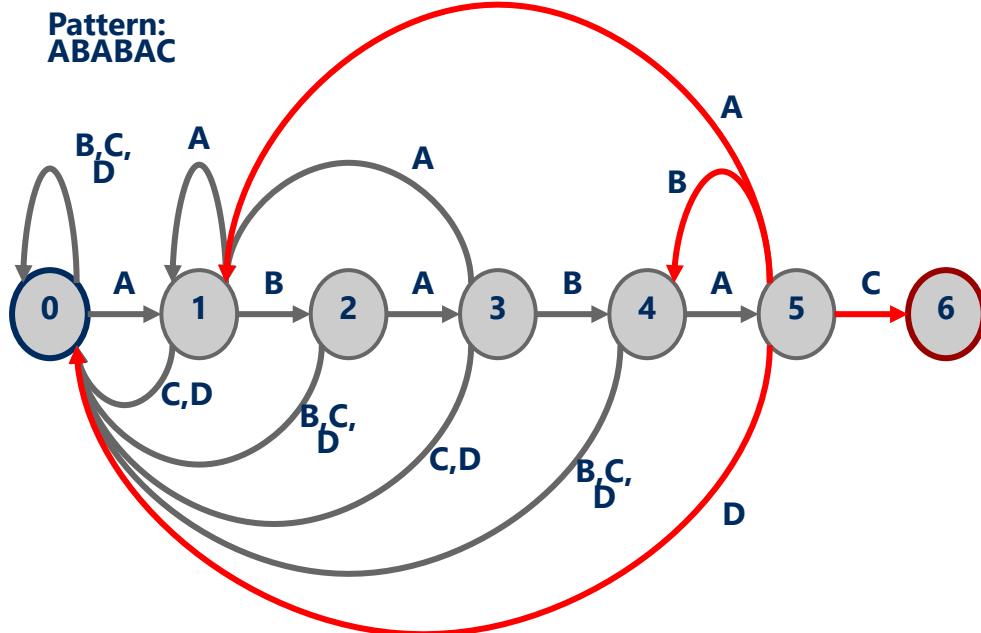
	0	1	2	3	4	5
A	1	1	3			
B	0	2	0			
C	0	0	0			
D	0	0	0			



	0	1	2	3	4	5
A	1	1	3	1		
B	0	2	0	4		
C	0	0	0	0		
D	0	0	0	0		



	0	1	2	3	4	5
A	1	1	3	1	5	
B	0	2	0	4	0	
C	0	0	0	0	0	
D	0	0	0	0	0	



	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

Representing the DFA in code

- DFA can be represented as a 2D array:
 - `dfa[cur_text_char][pattern_counter] = new_pattern_counter`
 - Storage needed?
 - mR

	0	1	2	3	4	5
A						
B						
C						
D						

KMP code

```
public int kmp_search(String pat, String txt) {  
    int m = pat.length();  
    int n = txt.length();  
    int i, j;  
    for (i = 0, j = 0; i < n && j < m; i++)  
        j = dfa[txt.charAt(i)][j];  
    if (j == m) return i - m; // found  
    return n; // not found  
}
```

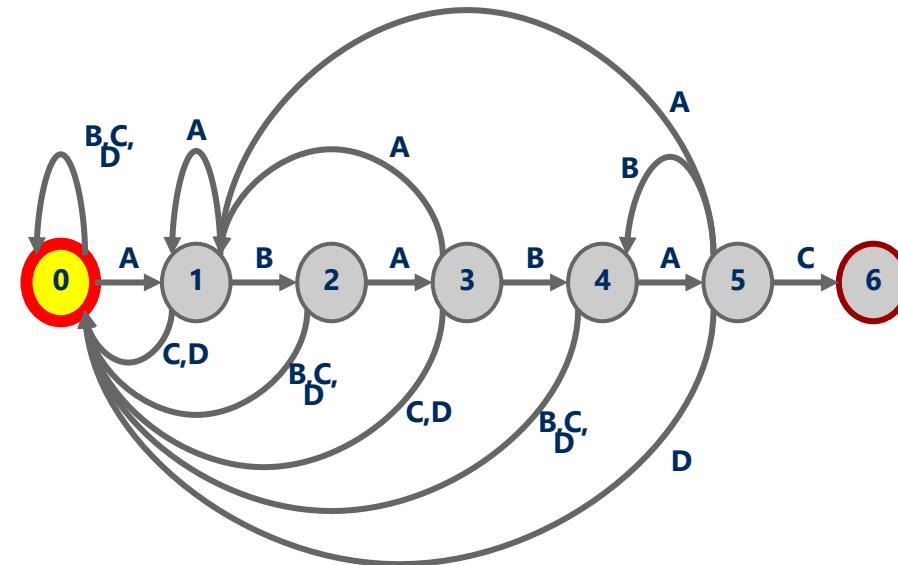
- Runtime?

i:	0
text:	A B A B A B A C
pattern:	A B A B A C
j:	0

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```



dfa[][]

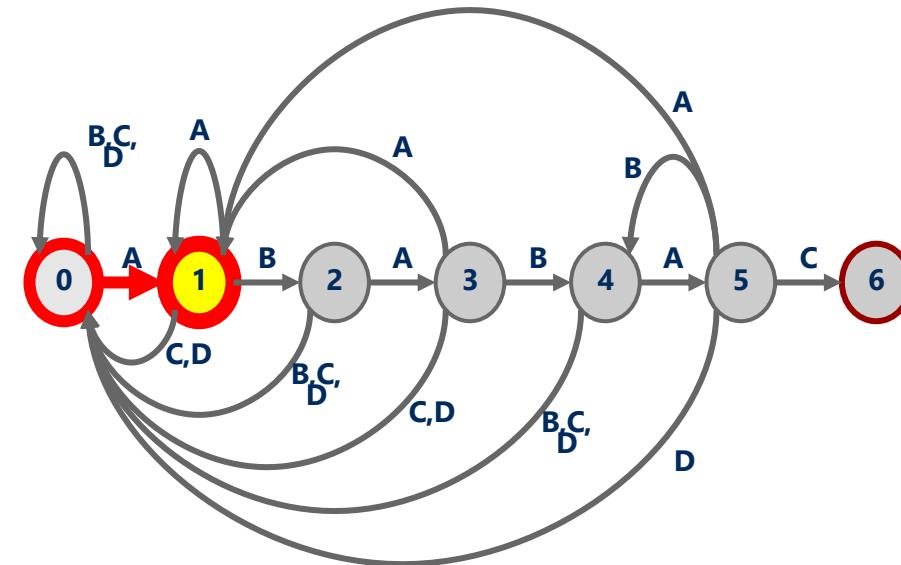
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:	0
text:	A B A B A B A C
pattern:	A B A B A C
j:	0

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```



dfa[][]

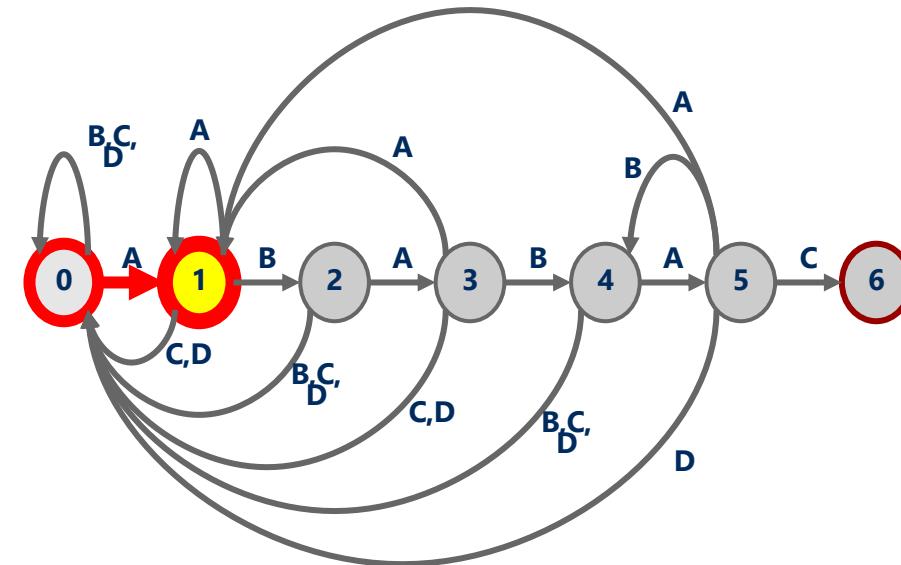
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:	0							
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0	1						

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```

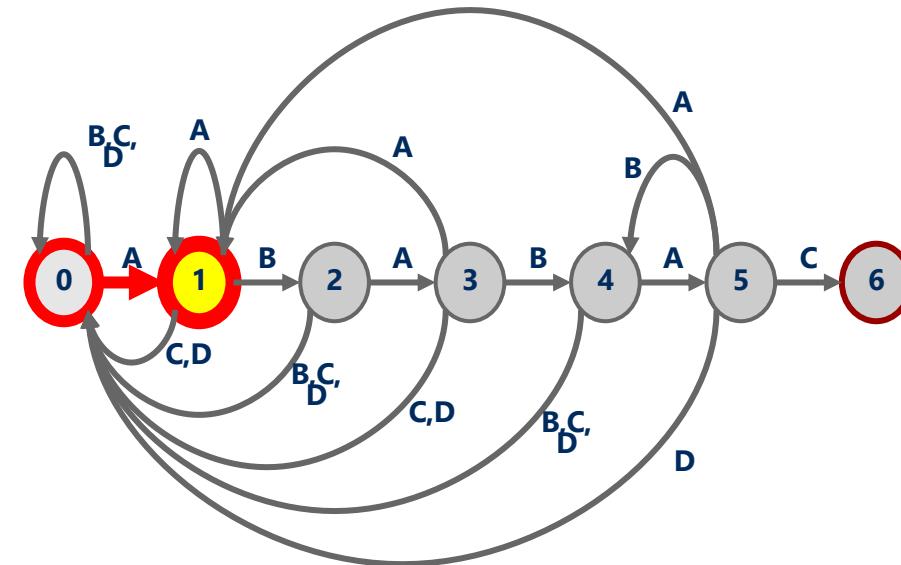


dfa[][]

	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:	0	1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0	1						

```
public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}
```



dfa[][]

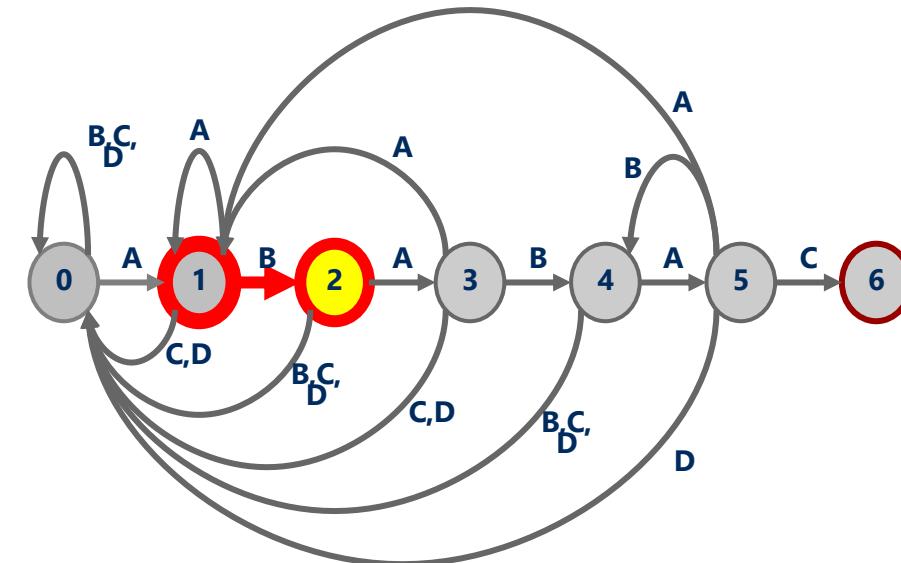
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:	1	2						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	1	2						

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```



dfa[][]

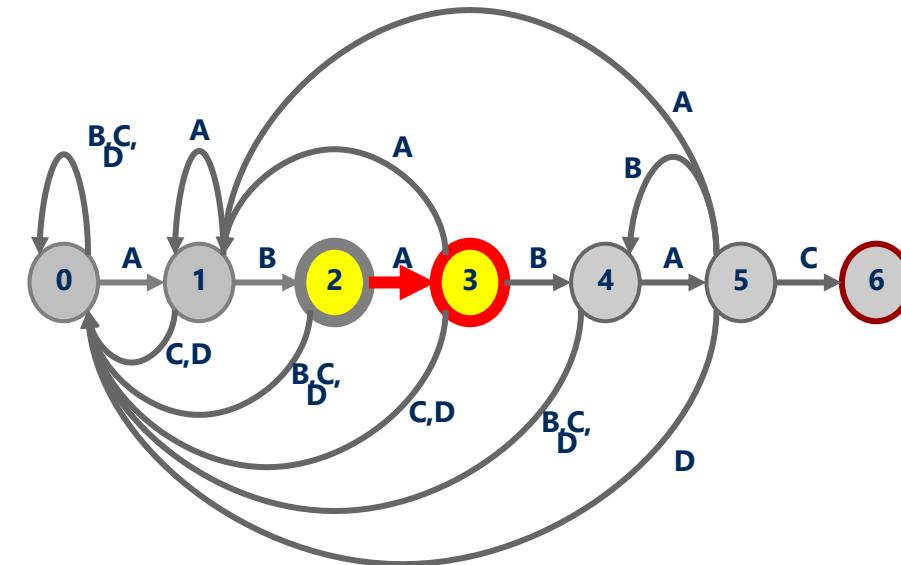
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:		2	3					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:		2	3					

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```



dfa[][]

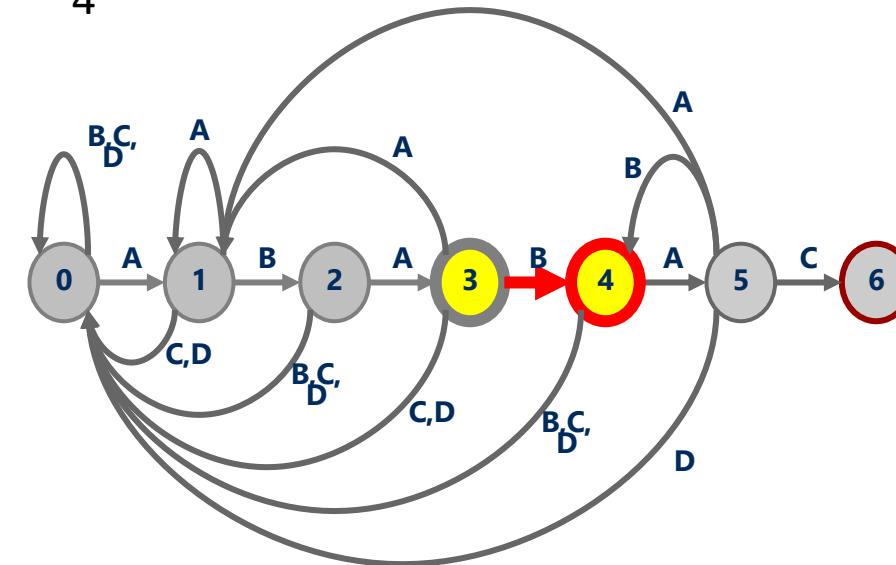
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:		3	4					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:		3	4					

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```



dfa[][]

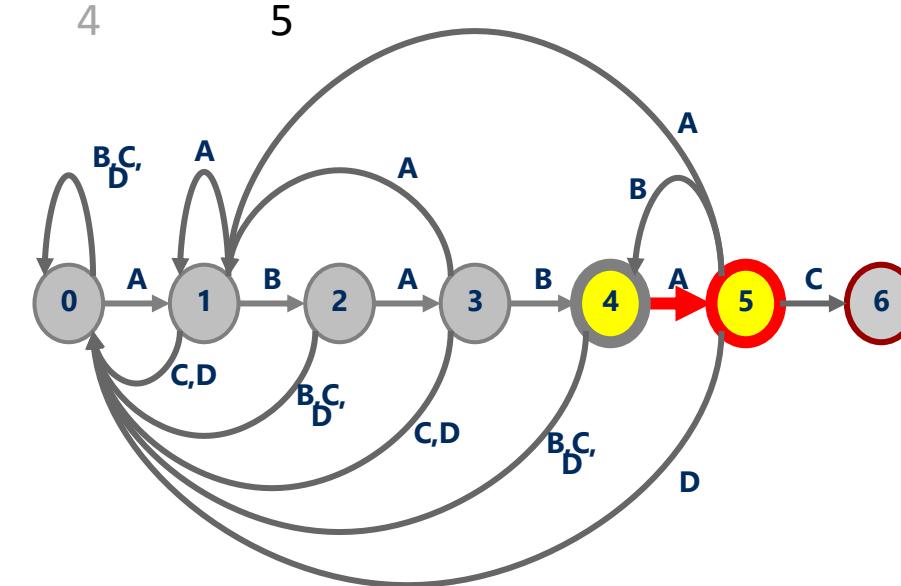
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:		4	5				
text:	A	B	A	B	A	B	A
pattern:	A	B	A	B	A	C	
j:		4	5				

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```



dfa[][]

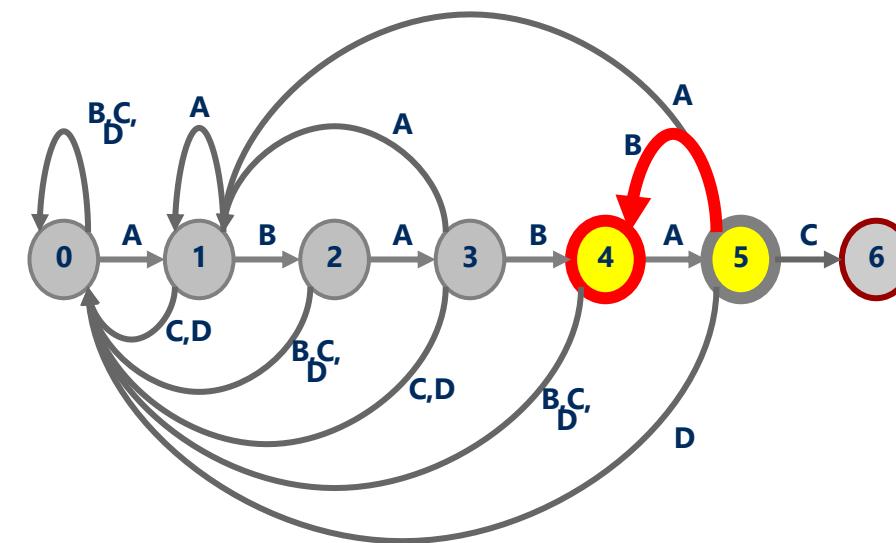
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:						5	6	
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:			4	5				

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```



dfa[][]

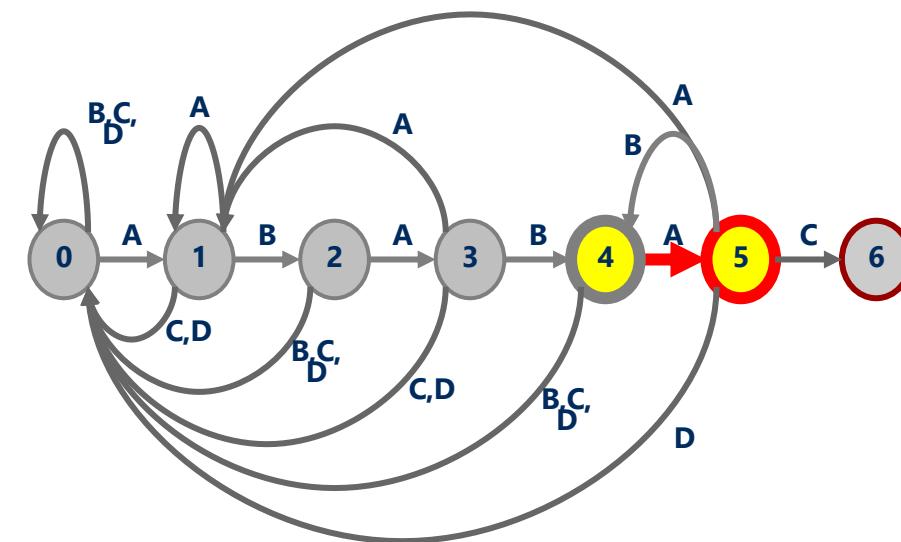
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:							6	7
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:					4	5		

```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

```



dfa[][]

	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:

text: A

B

A

B

A

B

A

7

8

pattern: A

B

A

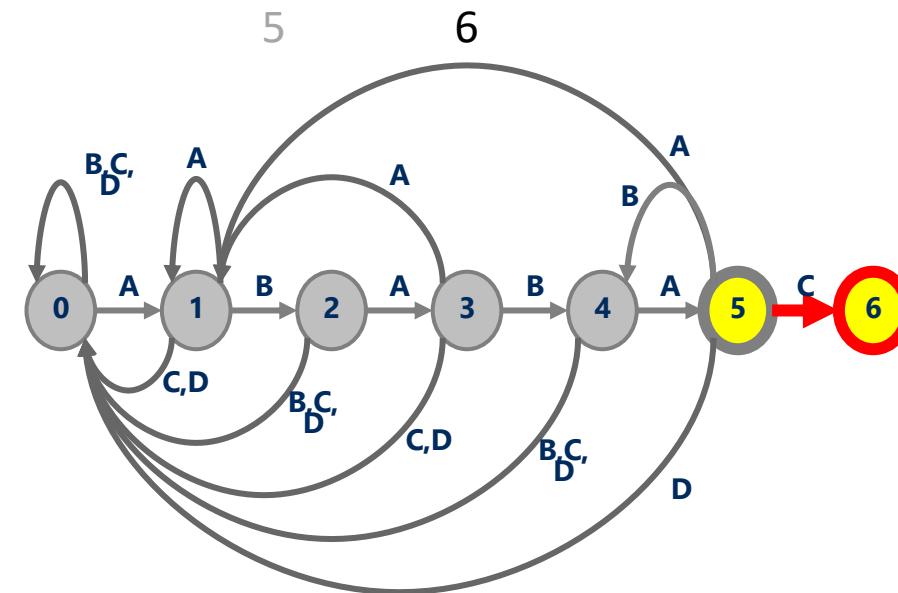
B

A

C

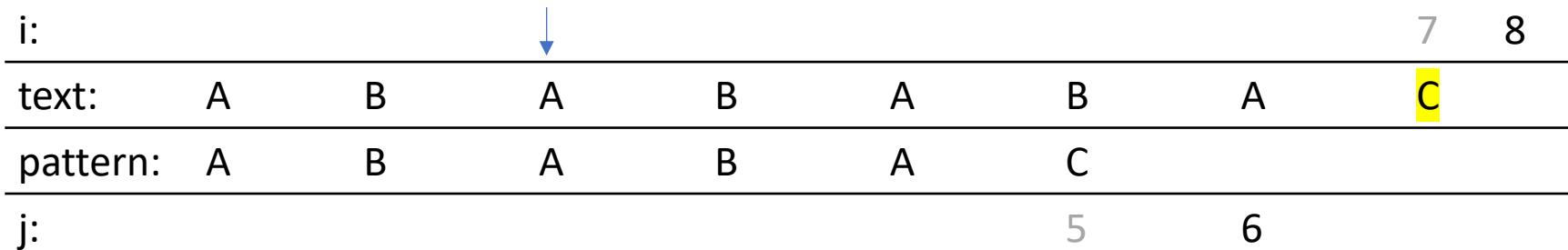
j:

```
public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}
```



dfa[][]

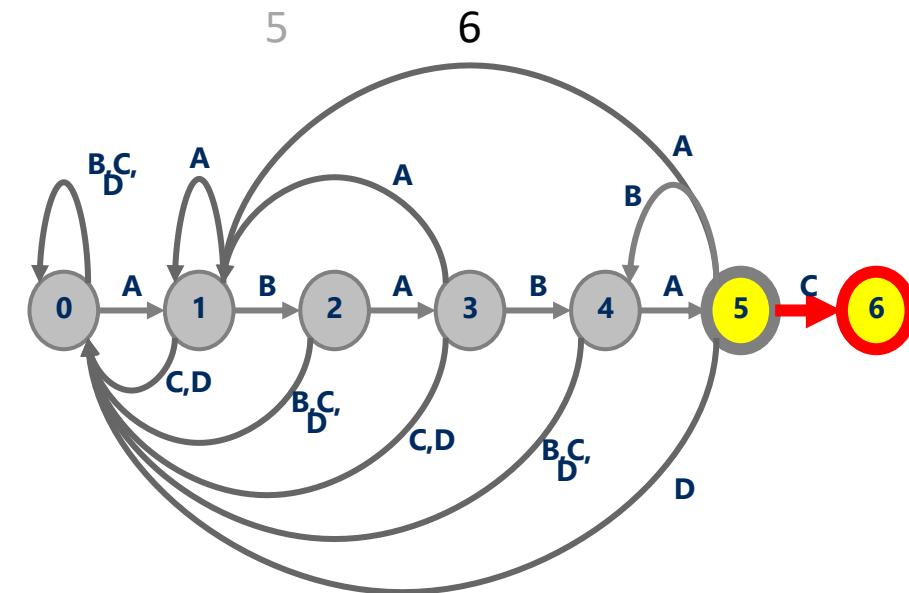
	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0



```

public int kmp_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++)
        j = dfa[txt.charAt(i)][j];
    if (j == m)
        return i - m; // found
    else
        return n; // not found
}

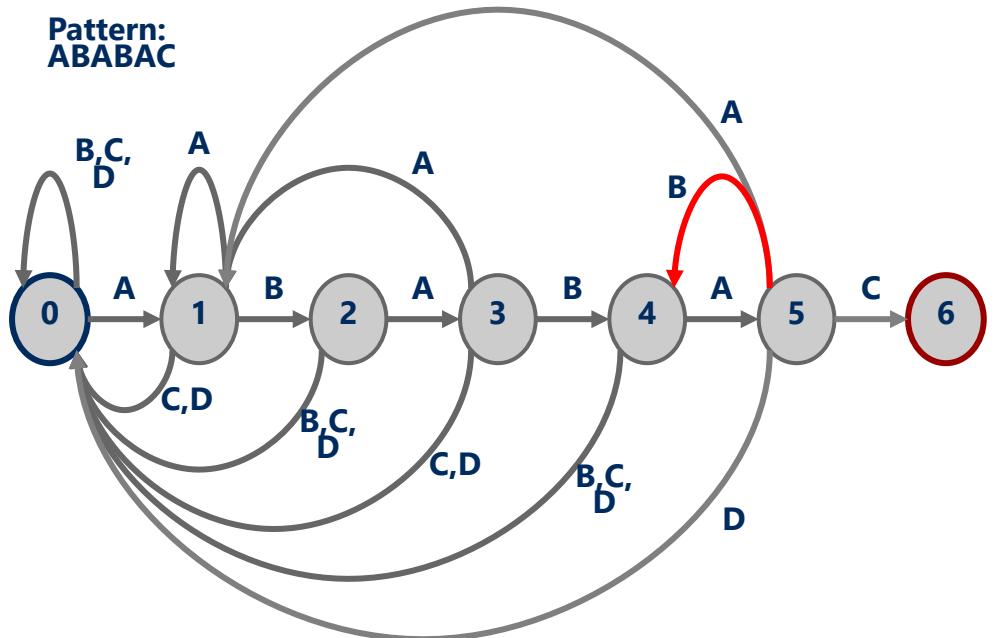
```



dfa[][]

	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

DFA Construction



	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

i:	0					5	
text:	A	B	A	B	A	B	A
pattern:	A	B	A	B	A	C	
j:	0					5	

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:	0	1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		1	2					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		2	3					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0	1						

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:			3	4				
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:		1	2					

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:				4	5		
text:	A	B	A	B	A	B	A
pattern:	A	B	A	B	A	C	
j:			2	3			

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

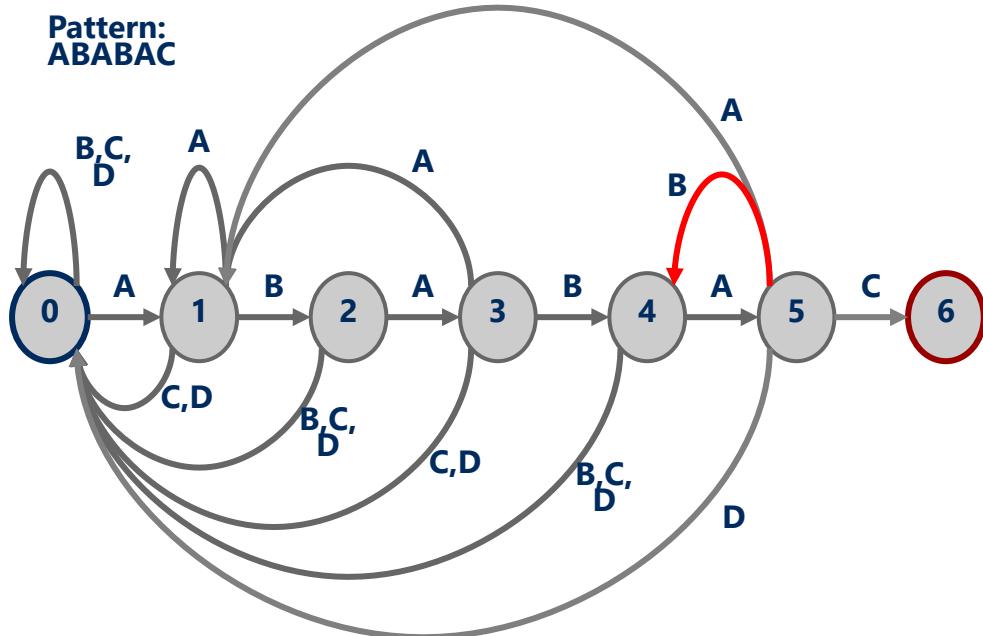
```

i:					5	6	
text:	A	B	A	B	A	B	A
pattern:	A	B	A	B	A	C	
j:				3	4		

```

public static int bf_search(String pat, String txt) {
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```



	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6
D	0	0	0	0	0	0

Another approach: Boyer Moore

- What if we compare starting at the end of the pattern?
 - $t = ABCD\textcolor{red}{V}ABCDWABCDXABCDYABCDZ$
 - $p = ABCDE$
 - V does not match E
 - Further V is nowhere in the pattern...
 - So skip ahead m positions with 1 comparison!
 - Runtime?
 - In the best case, n/m
- When searching through text with a large alphabet, will often come across characters not in the pattern.
 - One of Boyer Moore's heuristics takes advantage of this fact
 - Mismatched character heuristic

Mismatched character heuristic

- How well it works depends on the pattern and text at hand
 - What do we do in the general case after a mismatch?
 - Consider:
 - $t = XYXYZXXXXXXXXXXXXXX$
 - $p = XYZ$
 - If mismatched character *does* appear in p , need to “slide” to the right to the next occurrence of that character in p
 - Requires us to pre-process the pattern
 - Create a right array

```
for (int i = 0; i < R; i++)
    right[i] = -1;
for (int j = 0; j < m; j++)
    right[p.charAt(j)] = j;
```

Mismatched character heuristic example

Runtime for mismatched character

- What does the worst case look like?
 - Runtime:
 - $\Theta(nm)$
 - Same as brute force!
 - This is why mismatched character is only one of Boyer Moore's heuristics
 - Another works similarly to KMP
 - See BoyerMoore.java

Another approach

- Hashing was cool, let's try using that

```
public static int hash_search(String pat, String txt) {  
    int m = pat.length();  
    int n = txt.length();  
    int pat_hash = h(pat);  
    for (int i = 0; i <= n - m; i++) {  
        if (h(txt.substring(i, i + m)) == pat_hash)  
            return i; // found!  
    }  
    return n; // not found  
}
```

Well that was simple

- Is it efficient?
 - Nope! Practically worse than brute force
 - Instead of nm character comparisons, we perform n hashes of m character strings
- Can we make an efficient pattern matching algorithm based on hashing?

Horner's method

- Brought up during the hashing lecture

```
public long horners_hash(String key, int m) {  
    long h = 0;  
    for (int j = 0; j < m; j++)  
        h = (R * h + key.charAt(j)) % Q;  
    return h;  
}
```

- $\text{horners_hash("abcd", 4)} =$
 - $'a' * R^3 + 'b' * R^2 + 'c' * R + 'd' \text{ mod } Q$
- $\text{horners_hash("bcde", 4)} =$
 - $'b' * R^3 + 'c' * R^2 + 'd' * R + 'e' \text{ mod } Q$
- $\text{horners_hash("cdef", 4)} =$
 - $'c' * R^3 + 'd' * R^2 + 'e' * R + 'f' \text{ mod } Q$

Efficient hash-based pattern matching

```
text = "abcdefg"  
pattern = "defg"
```

- This is Rabin-Karp

What about collisions?

- Note that we're not storing any values in a hash table...
 - So increasing Q doesn't affect memory utilization!
 - Make Q really big and the chance of a collision becomes really small!
 - But not 0...
- OK, so do a character by character comparison on a hash match just to be sure
 - Worst case runtime?
 - Back to brute force-esque runtime...

Assorted casinos

- Two options:
 - Do a character by character comparison after hash match
 - Guaranteed correct Las Vegas
 - Probably fast
 - Assume a hash match means a substring match
 - Guaranteed fast Monte Carlo
 - Probably correct