Algorithms and Data Structures 1
CS 0445

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides adapted from slides of Textbook, Dr. Ramirez's CS 0445, and Dr. Farnan's CS 1501)

Announcements

- Upcoming Deadlines
  - Homework 11: This Friday 12/9 @ 11:59 pm
  - Lab 11: next Monday 12/12
  - Lab 12 and Homework 12: Monday 12/19
  - Assignment 5 is now for extra credit ONLY
  - We have 4 programming assignments
    - the lowest is dropped
    - each worth 13.3%
  - Assignment 3: Friday 12/16 @ 11:59 pm
  - Assignment 4: Friday 12/16 @ 11:59 pm

Bonus Opportunities

- Bonus Lab due on 12/19
- Bonus Homework due on 12/19
- Bonus Assignment due on 12/19
- 1 bonus point for entire class when OMETs response rate >= 80%
    - Currently at 23%
    - Deadline is Sunday 12/11

Final Exam

- Same format as midterm
- Non-cumulative
- Date, time and location on PeopleSoft
  - Thursday 12/15 8-9:50 am (coffee served!)
- Same classroom as lectures
- Study guide and practice test to be posted soon

Previous Lecture …

- Hashing!
    - what makes a good hash function
        - Horner’s method + modular hashing
    - Handling collisions
        - Open addressing
            - Linear probing

This Lecture …

- Hashing!
  - Handling collisions
    - Open addressing
      - Double hashing
    - Closed addressing
- String matching

Muddiest Points

- **Q: why do we have iterable interface and iterator interface. As only iterator works here**
- Iterator interface is used to implement iterators
- Iterable interface is used to implement containers that have iterators
  - allows us to use the for-each loop structure

```
IterableLinkedList<Integer> list = new …..
for(Integer x : list){
   //do something with x
}
```

Muddiest Points

- **Q: Can we please get more in class tophat questions? It would be a very helpful way to boost our grades.**
- Sure. Let’s have a couple today and next lecture!

Double hashing

- After a collision, instead of attempting to place the key x in i+1 mod m, look at i+h2(x) mod m
    - h2() is a second, different hash function
        - Should still follow the same general rules as h() to be considered good, but needs to be different from h()
            - h(x) == h(y) AND h2(x) == h2(y) should be very unlikely
                - Hence, it should be unlikely for two keys to use the same increment

Double hashing

- h(x) = x mod 11
- h2(x) = (x mod 7) +1
- Insert 14, 17, 25, 37, 34, 16, 26

0	1	2	3	4	5	6	7	8	9	10
	34		14	37	16	17		25		26

- Why could we not use h2(x) = x mod 7?
  - Try to insert 2401

A few extra rules for h2()

- Second hash function cannot map a value to 0
- You should try all indices once before trying one twice

- Were either of these issues for linear probing?

As $\alpha \to 1$...

- Meaning n approaches m…
- Both linear probing and double hashing degrade to Θ(n)
  - How?
    - Multiple collisions will occur in both schemes
    - Consider inserts and misses…
      - Both continue until an empty index is found
        - With few indices available, close to m probes will need to be performed
          - Θ(m)
        - n is approaching m, so this turns out to be Θ(n)

Horner's method

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

horners_hash("abcd", 4) =

- 'a' * R^3 + 'b' * R^2 + 'c' * R + 'd' % m
- **h = 'a' % m**
- **h = h * R + 'b' % m**
- = ('a' % m) * R + 'b' % m
- **h = h * R + 'c' % m**
- = (('a' % m) * R + 'b' % m) * R + 'c' % m
- **h = h * R + 'd' % m**
- = ((('a' % m) * R + 'b' % m) * R + 'c' % m) * R + 'd'

Open addressing issues

- Must keep a portion of the table empty to maintain respectable performance
  - For linear hashing ½ is a good rule of thumb
    - Can go higher with double hashing
- What do we do when the hash table is more than half full?
  - resizing!
  - How?

Closed addressing

- i.e., if a pigeon’s hole is taken, it lives with a roommate
- Most commonly done with **separate chaining**
  - Create **a linked-list** of keys at each index in the table
  - Similar to Assignment 2!
    - array of linked lists

Closed addressing

- Performance depends on chain length
  - Which is determined by the load factor $\alpha = n/m$ and the quality of the hash function
  - With a good hash function, on average, n/m keys per chain
- In closed addressing, number of keys $n >$ table size m
  - not possible with open addressing

In general...

- Closed-addressing hash tables are fast and efficient for many applications
- Where would open addressing be preferable?
    - Strict memory limits
    - Lack of dynamic memory allocation
        - needed to allocating nodes in the linked lists in separate chaining

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	C
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	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:						5	6	
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:							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 = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXY
    - p = XXXXY
  - m (n - m + 1)
    - Θ(nm) if n >> m
  - Is the average case runtime any better?
    - Assume we mostly mismatch on the first pattern character
    - Θ(n + m)
      - Θ(n) if n >> 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

Another approach: Boyer Moore

- What if we compare starting at the end of the pattern?
  - `t = ABCDVABCDWABCDXABCDYABCDZ`
  - `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 = XYXYXYZXXXXXXXXXXXXXX`
      - `p = XYXYZ`
    - 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

Pattern: A B C D E

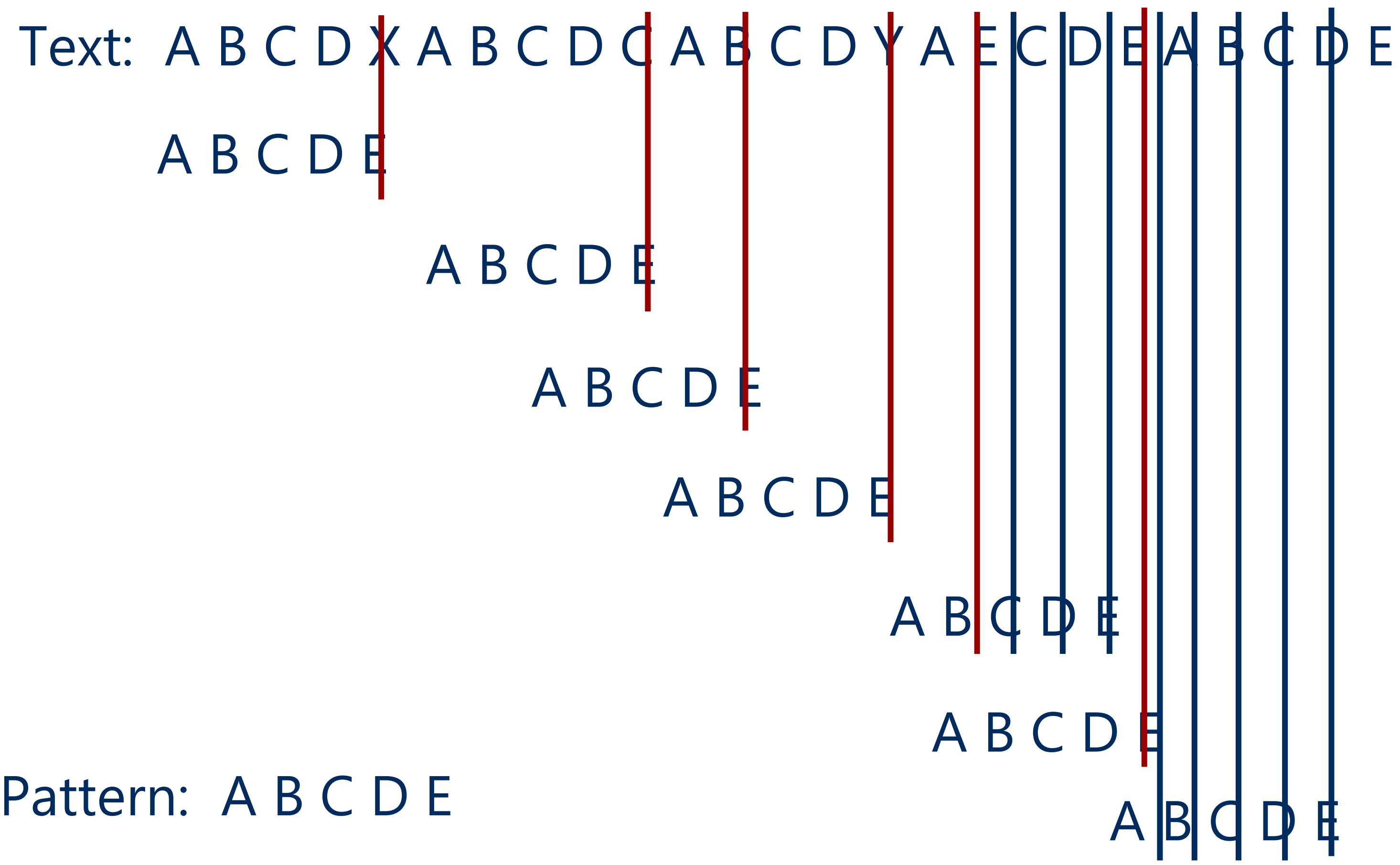
right = [0, 1, 2, 3, 4, -1, -1, …]

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

Mismatched character Procedure

- Let j be the index in the pattern currently under comparison
- At mismatch, slide pattern to the right by
    - j - right[mismatched_text_char] positions
    - If < 1, slide 1

Mismatched character heuristic example



Pattern: A B C D E

right = [0, 1, 2, 3, 4, -1, -1, …]

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;
}
```

- horners_hash("abcd", 4) =
    - 'a' * R^3 + 'b' * R^2 + 'c' * R + 'd' mod Q
- horners_hash("bcde", 4) =
    - 'b' * R^3 + 'c' * R^2 + 'd' * R + 'e' mod Q
- horners_hash("cdef", 4) =
    - 'c' * R^3 + 'd' * R^2 + 'e' * R + 'f' 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
    - Probably fast

Las Vegas

  - Assume a hash match means a substring match
    - Guaranteed fast
    - Probably correct

Monte Carlo

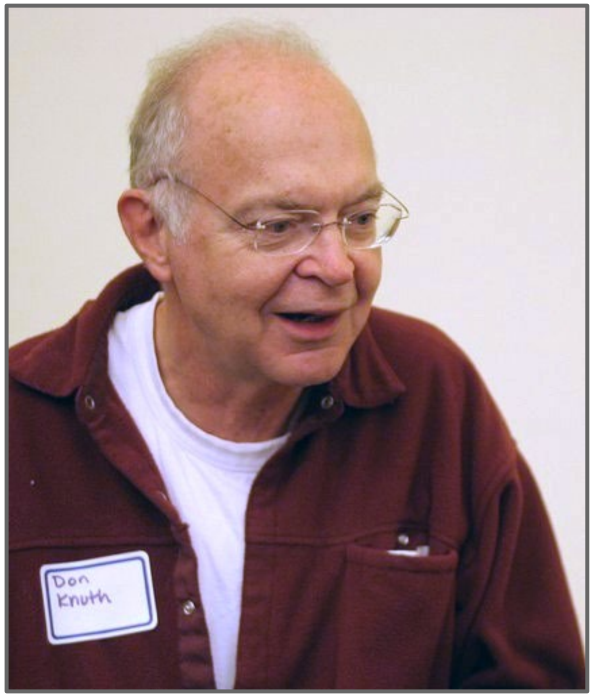
First: improving the worst case

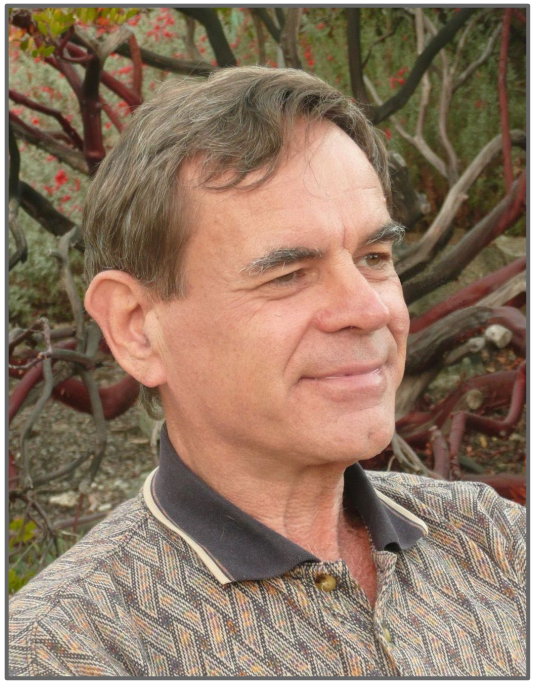
Discovered the same algorithm independently

Knuth

Morris

Pratt



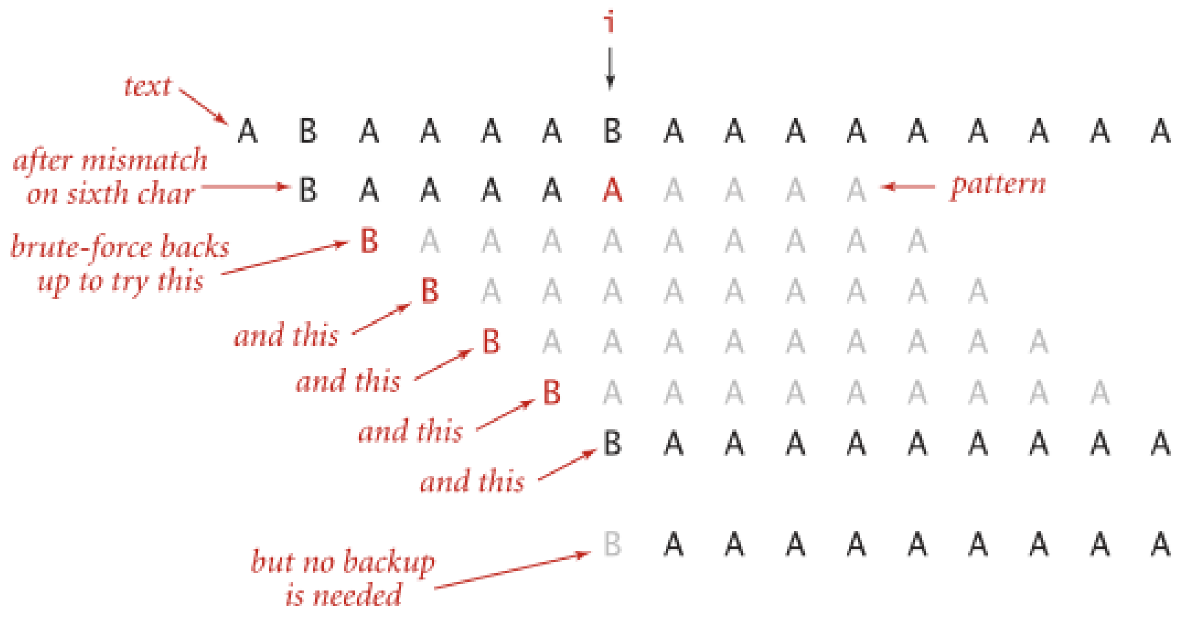


Worked together

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



i
text
A B A A A A B A A A A A A A A A
after mismatch on sixth char
B A A A A A A A A A
pattern
brute-force backs up to try this
and this
and this
and this
and this
but no backup is needed

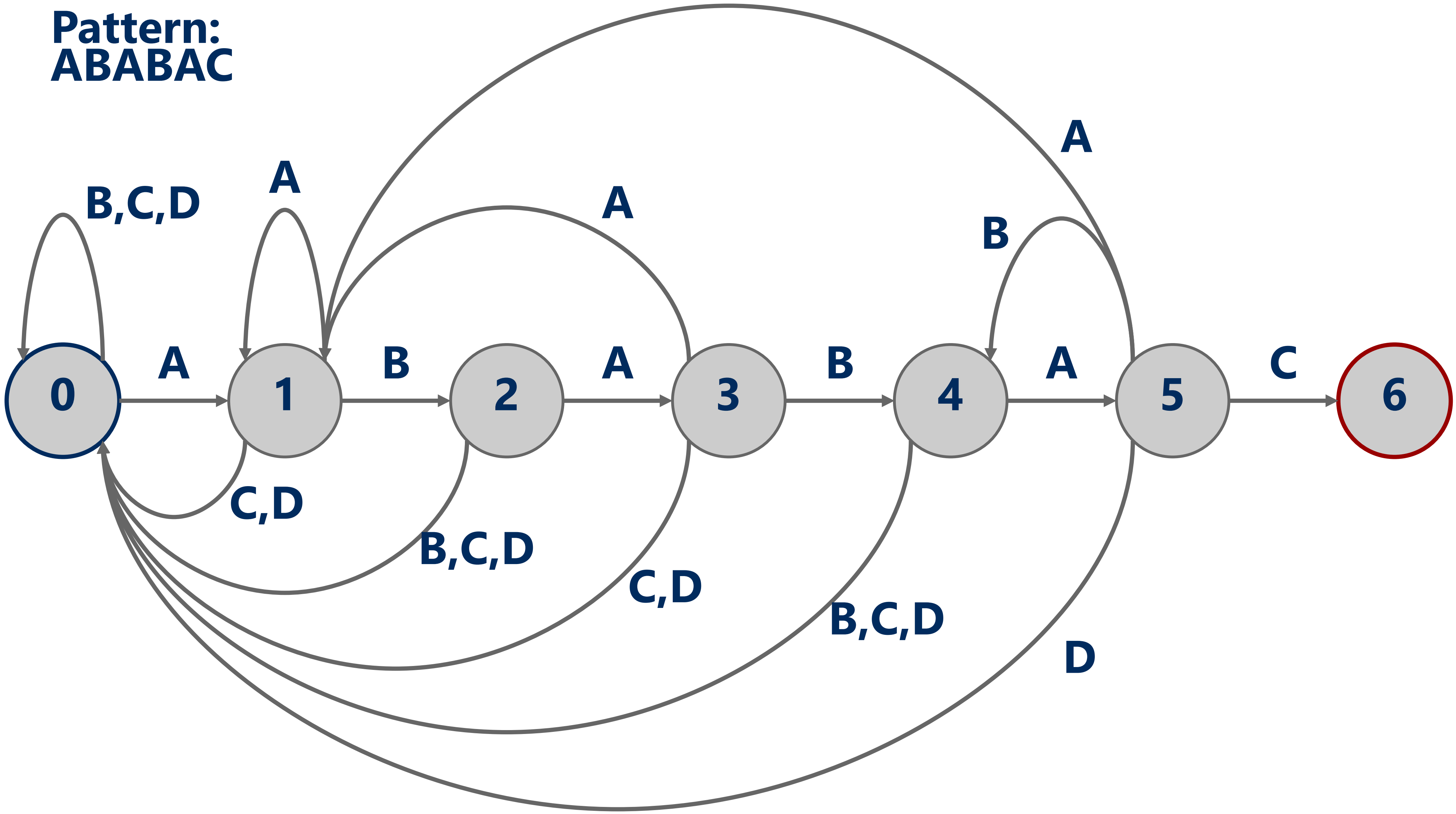
Text pointer backup in substring searching

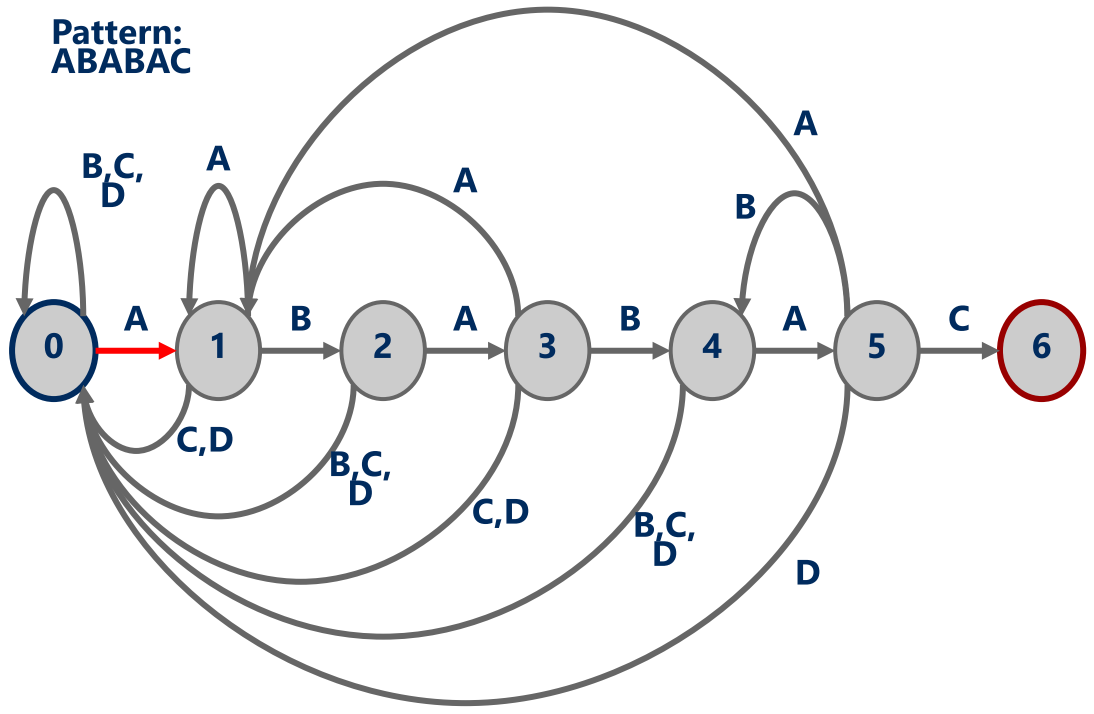
How do we keep track of text processed?

- 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

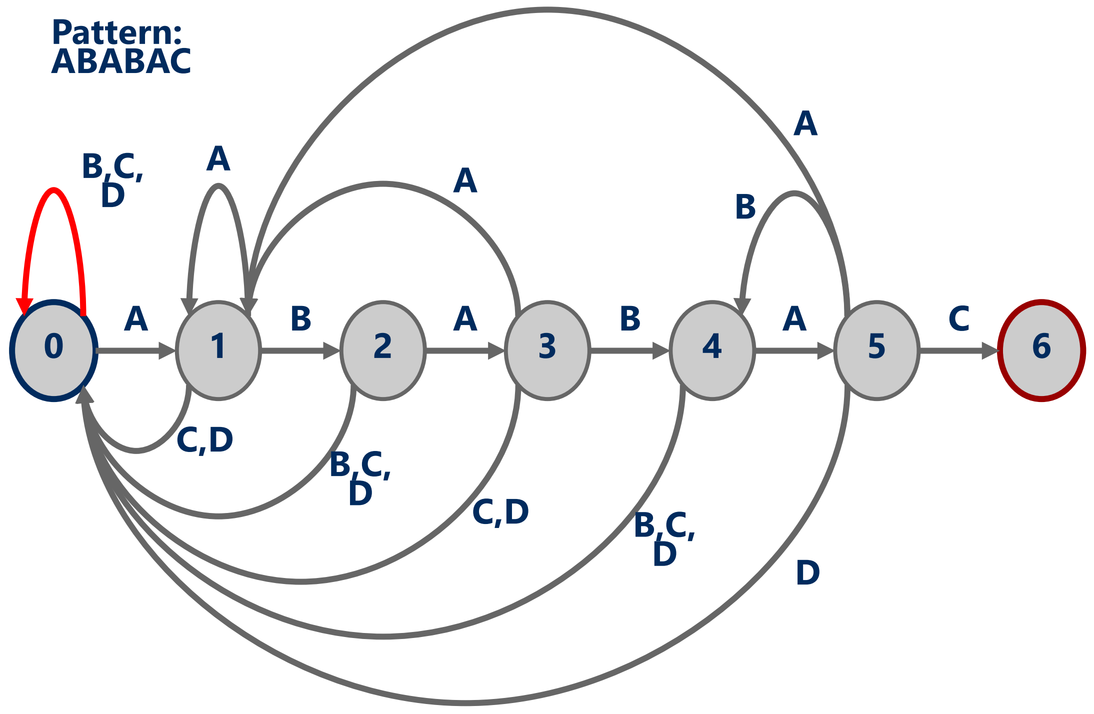
Pattern:
ABABAC





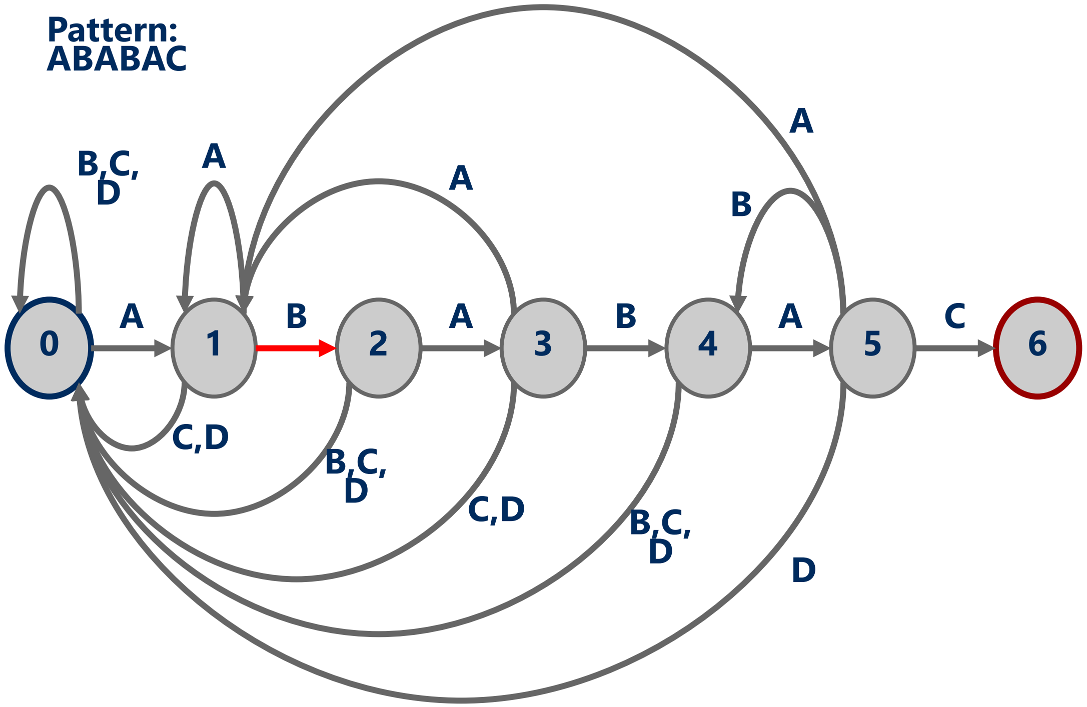
	0	1	2	3	4	5
A	1					
B						
C						
D						

Pattern:
ABABAC

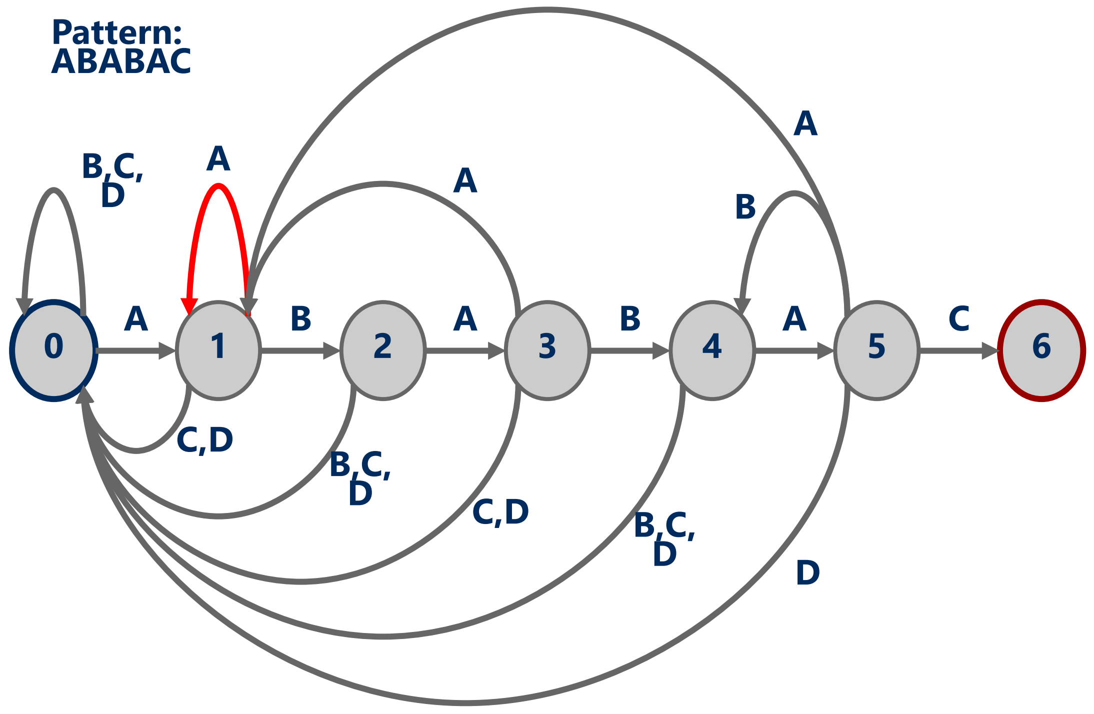


	0	1	2	3	4	5
A	1					
B	0					
C	0					
D	0					

Pattern: ABABAC

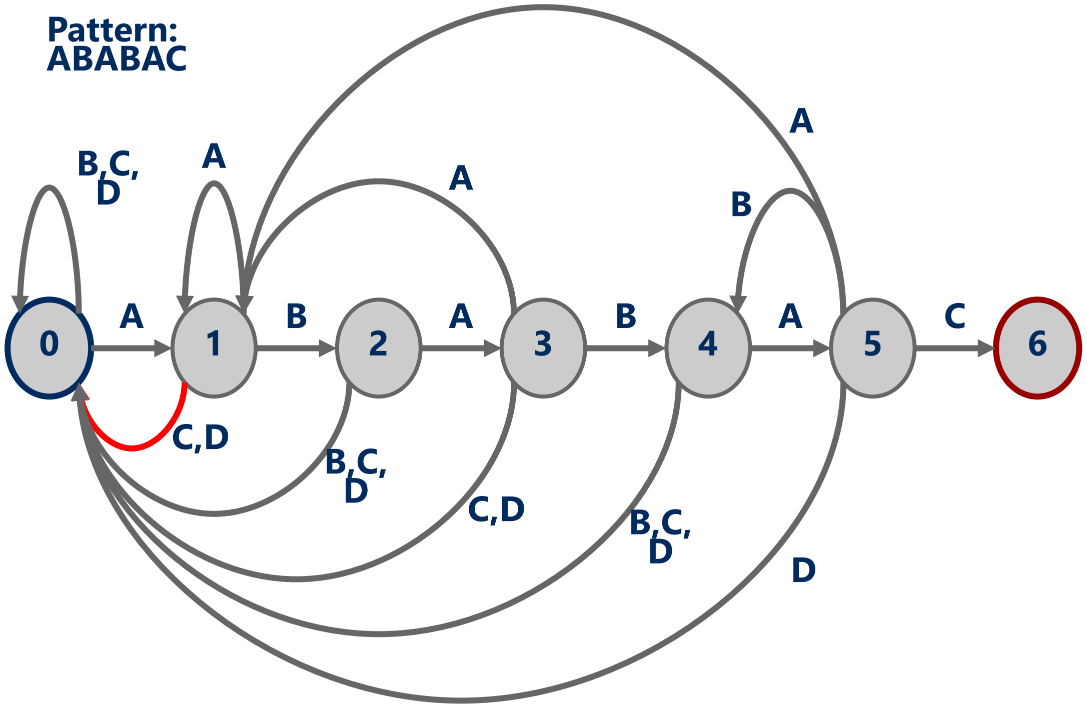


	0	1	2	3	4	5
A	1					
B	0	2				
C	0					
D	0					



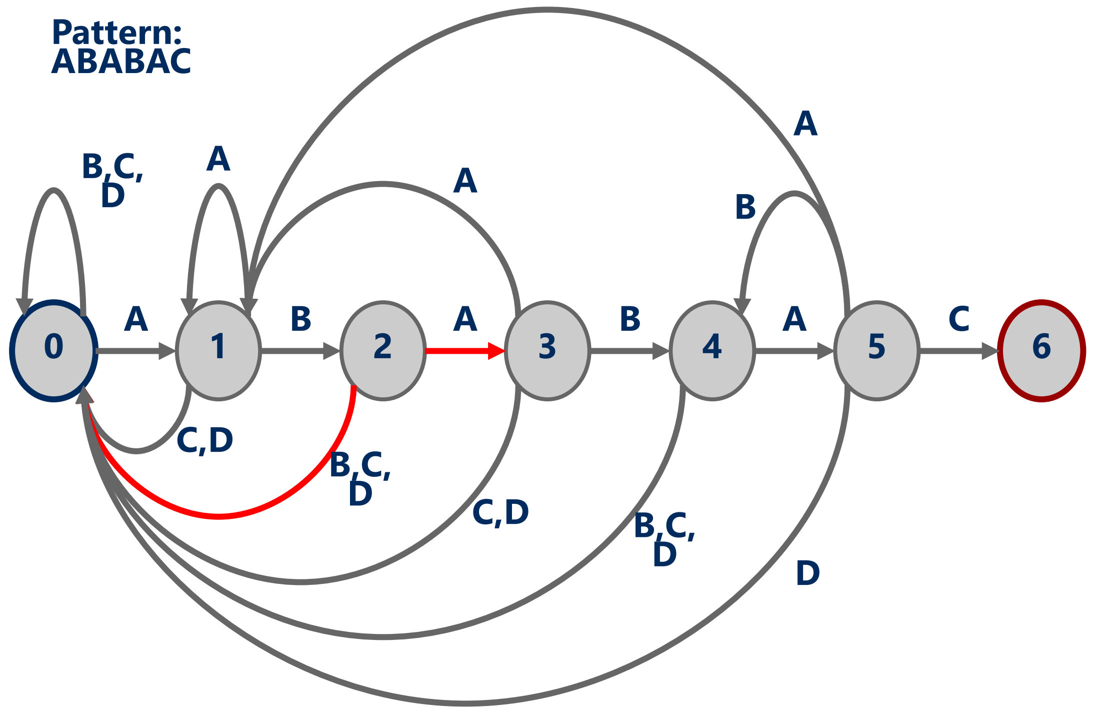
	0	1	2	3	4	5
A	1	1				
B	0	2				
C	0					
D	0					

Pattern:
ABABAC



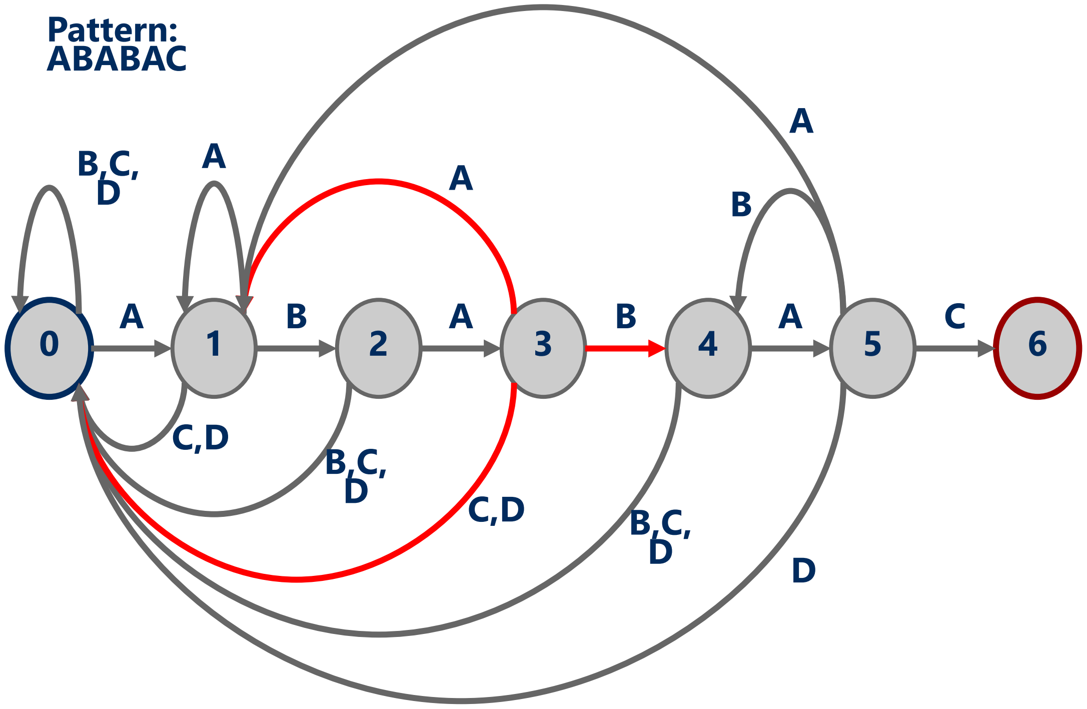
	0	1	2	3	4	5
A	1	1				
B	0	2				
C	0	0				
D	0	0				

Pattern: ABABAC



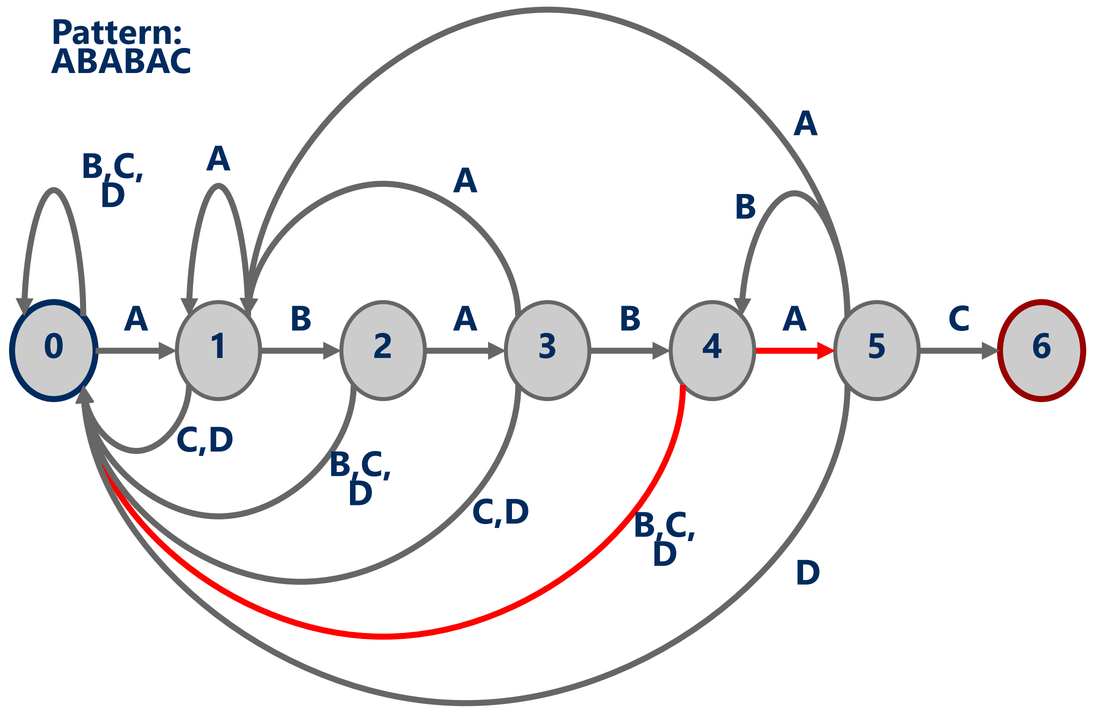
	0	1	2	3	4	5
A	1	1	3			
B	0	2	0			
C	0	0	0			
D	0	0	0			

Pattern: ABABAC



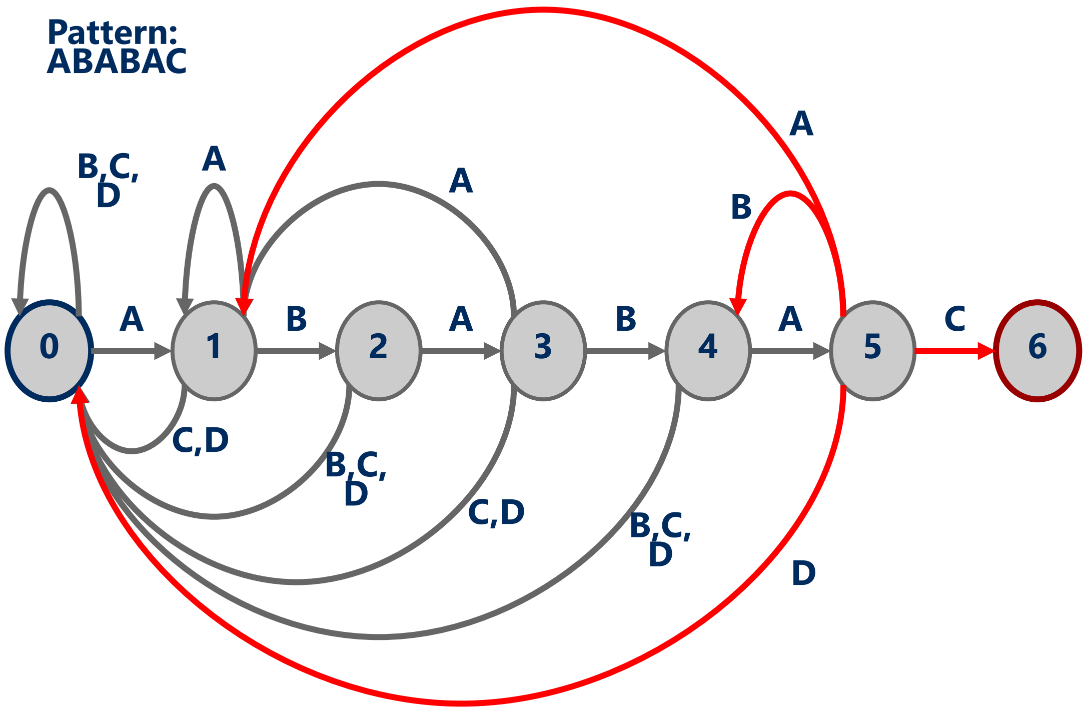
	0	1	2	3	4	5
A	1	1	3	1		
B	0	2	0	4		
C	0	0	0	0		
D	0	0	0	0		

Pattern:
ABABAC



	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	

Pattern:
ABABAC



	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						

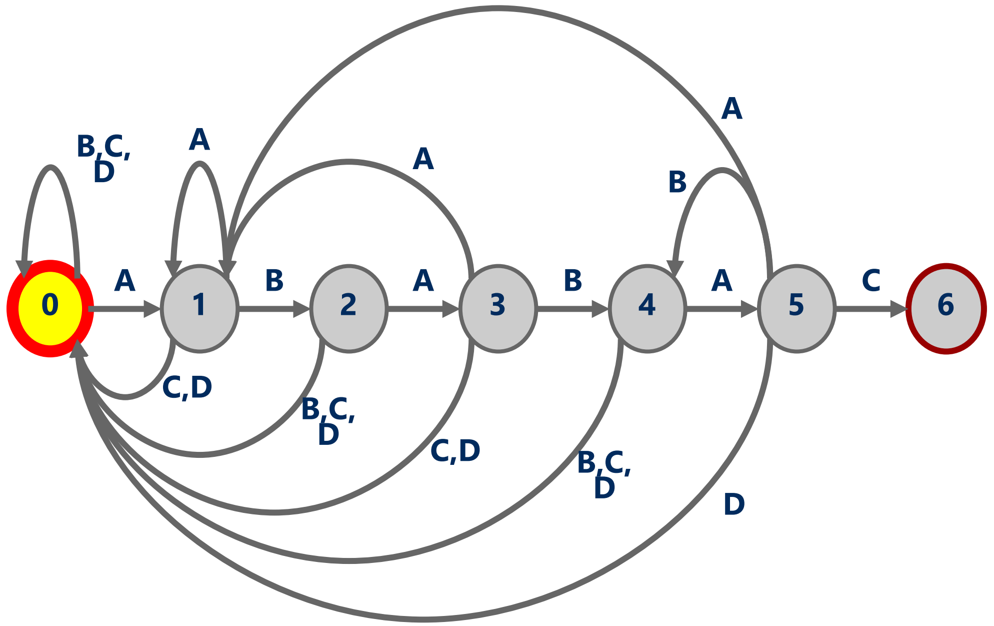
KMP code

```
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
}
```

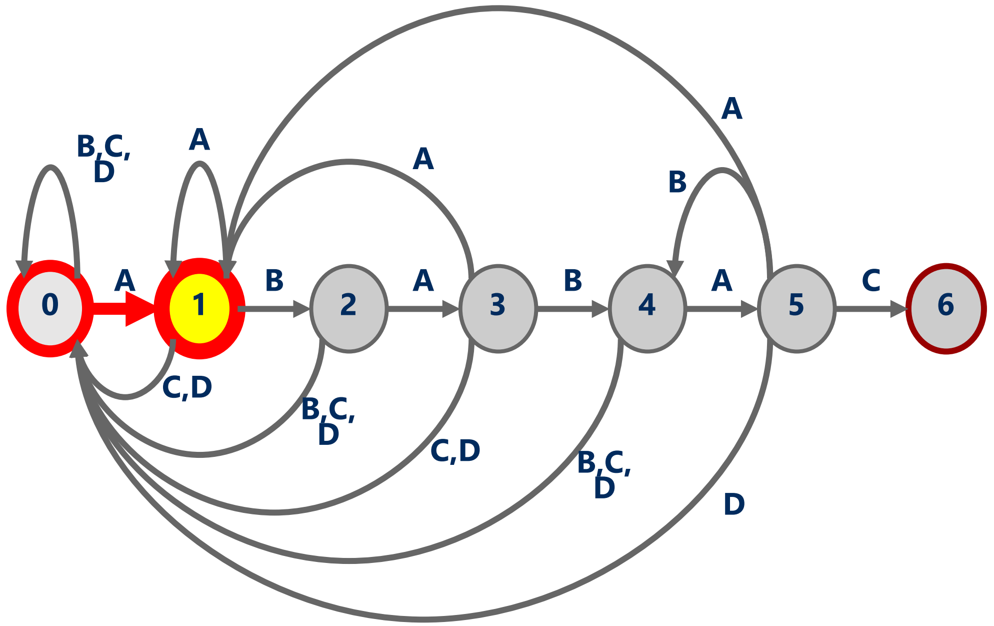


dfa[][]

	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
}
```

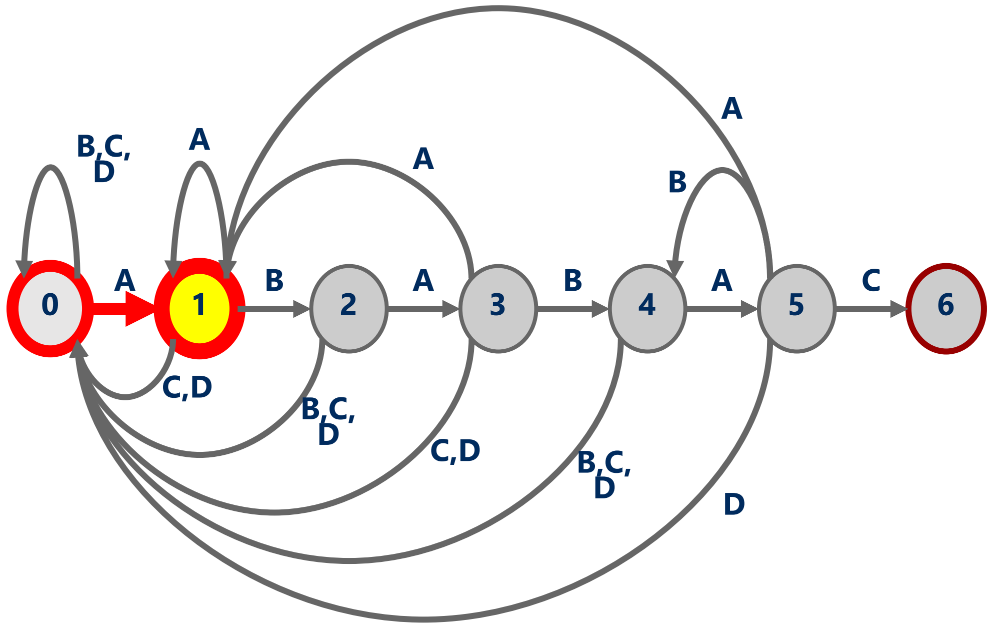


dfa[][]

	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
}
```

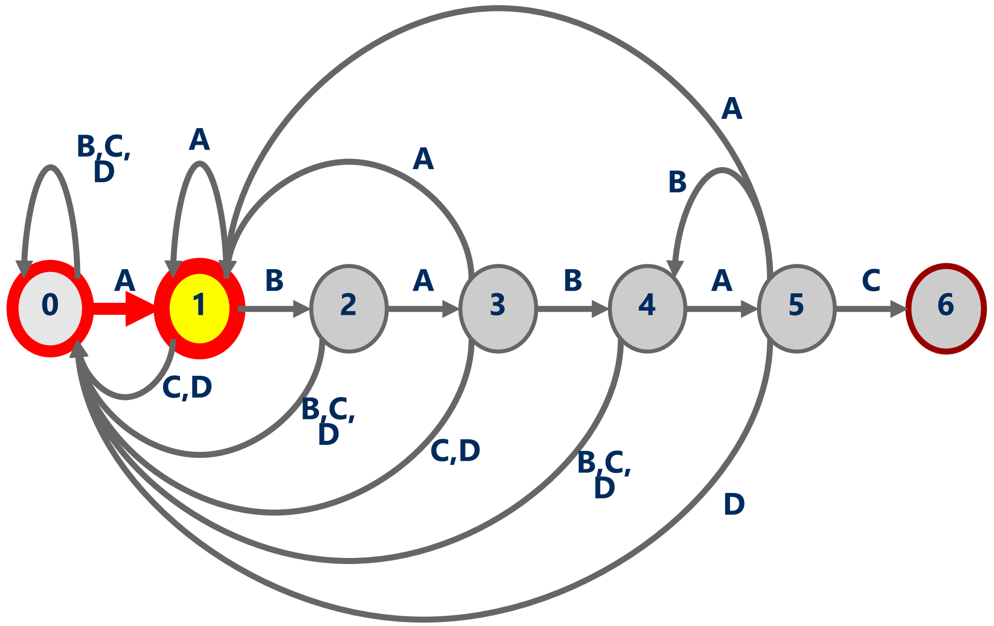


dfa[][]

	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
}
```

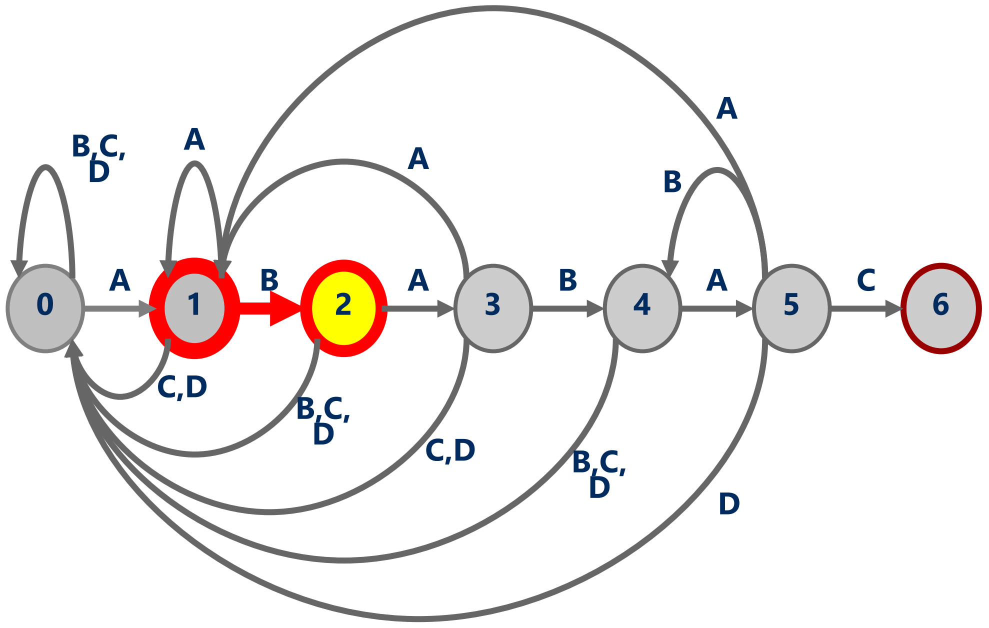


dfa[][]

	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
}
```

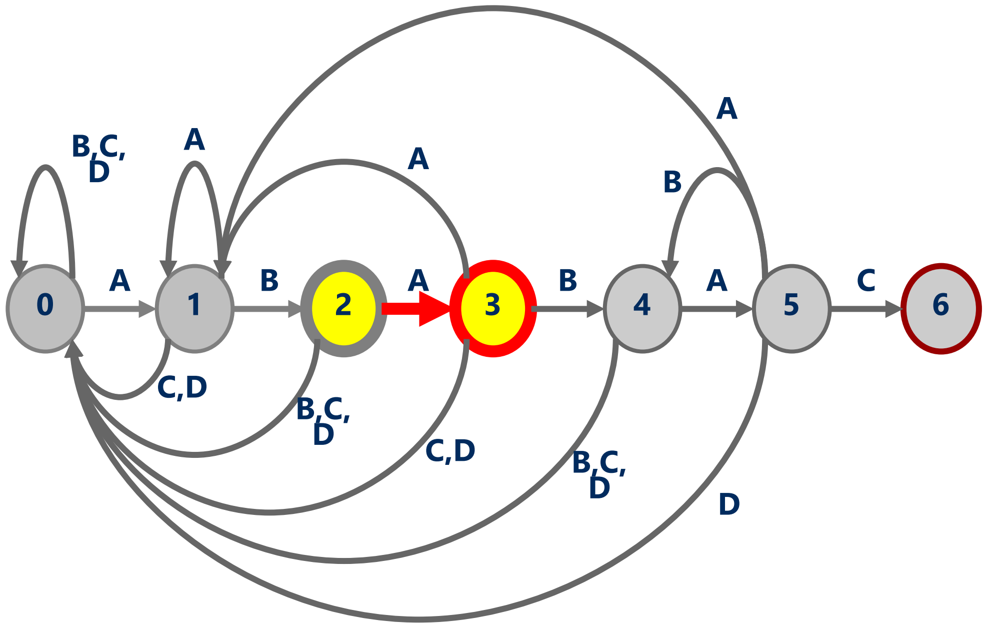


dfa[][]

	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
}
```

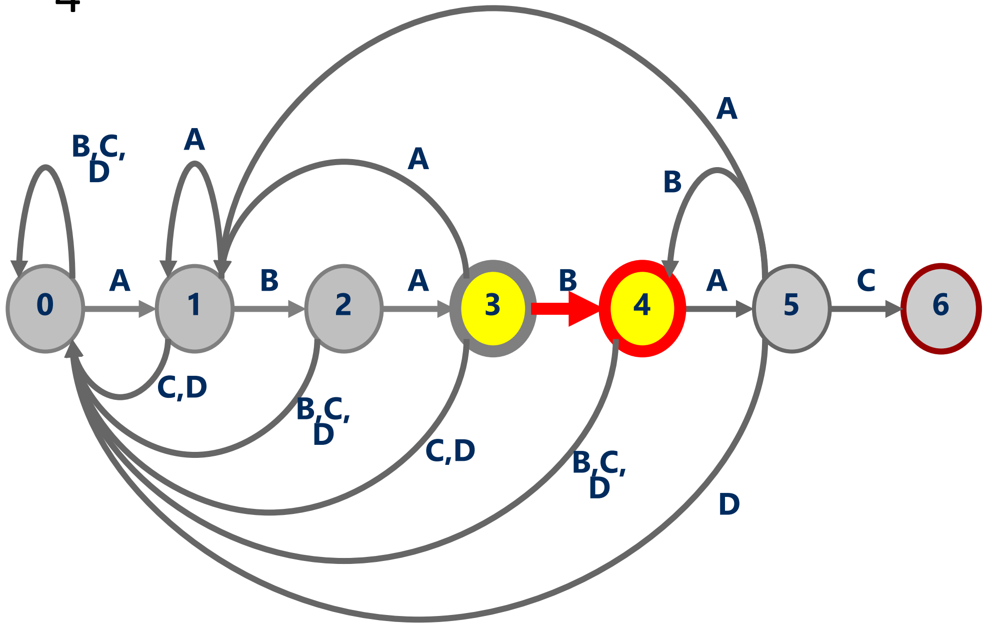


dfa[][]

	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
}
```

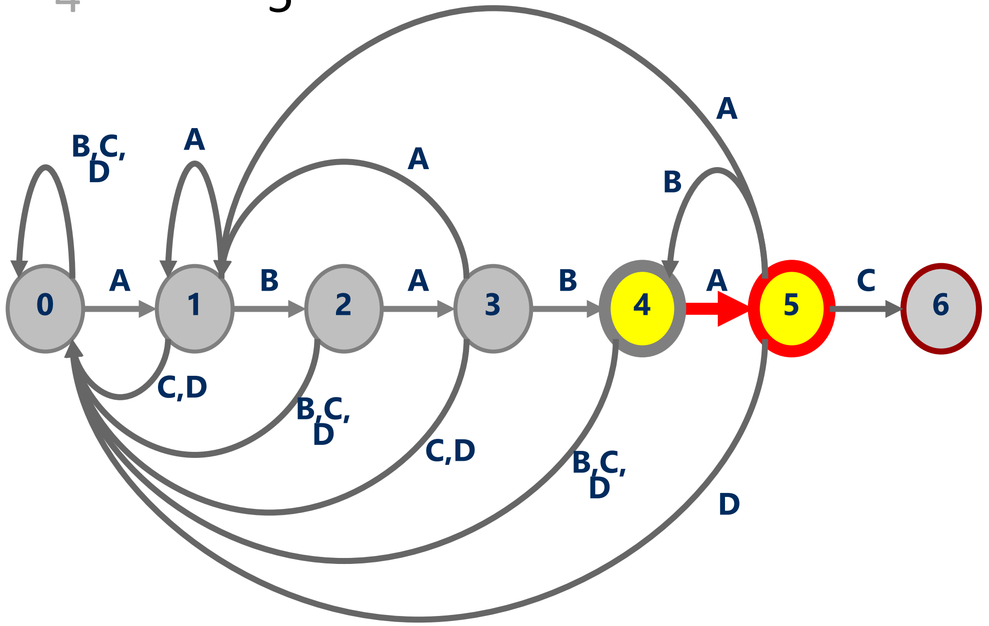


dfa[][]

	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	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
}
```

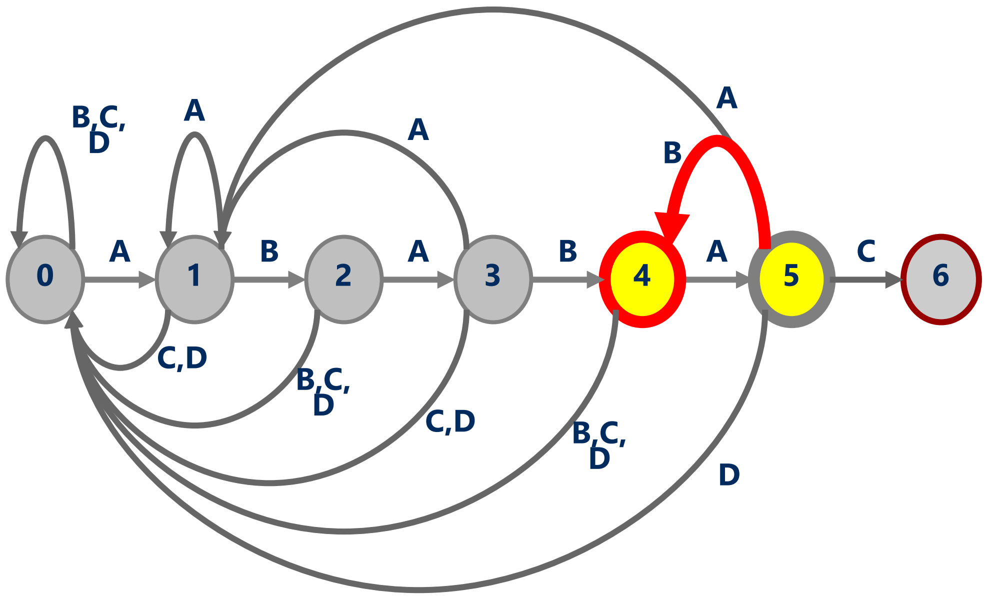


dfa[][]

	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:						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
}
```

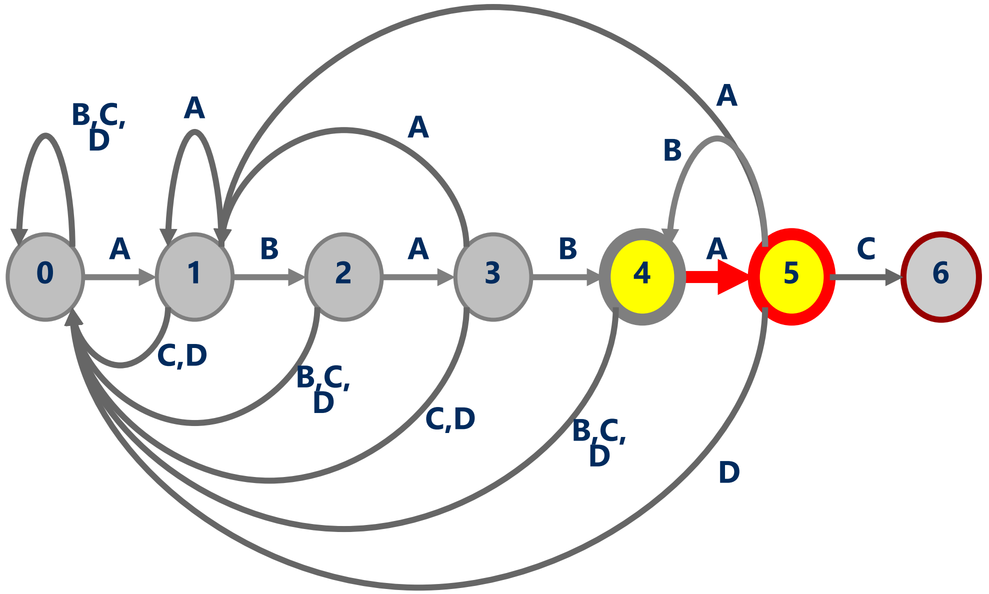


dfa[][]

	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
}
```

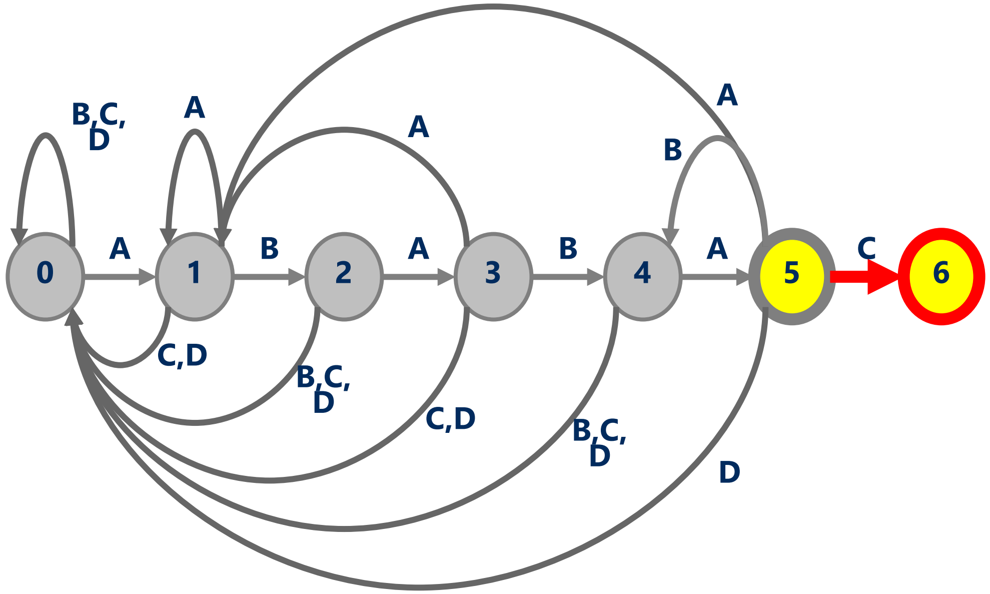


dfa[][]

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

```
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
}
```

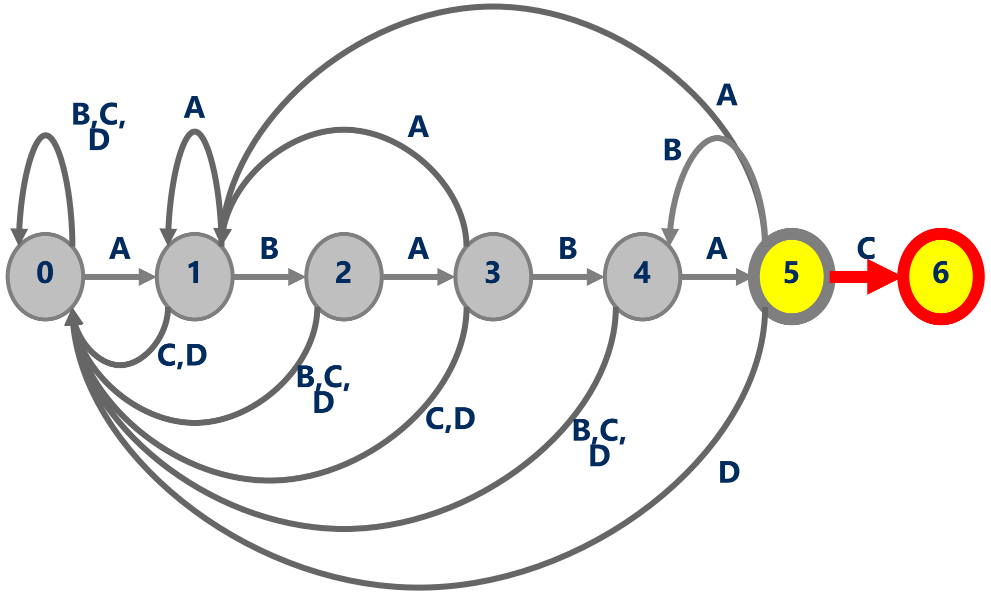


dfa[][]

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

```
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
}
```

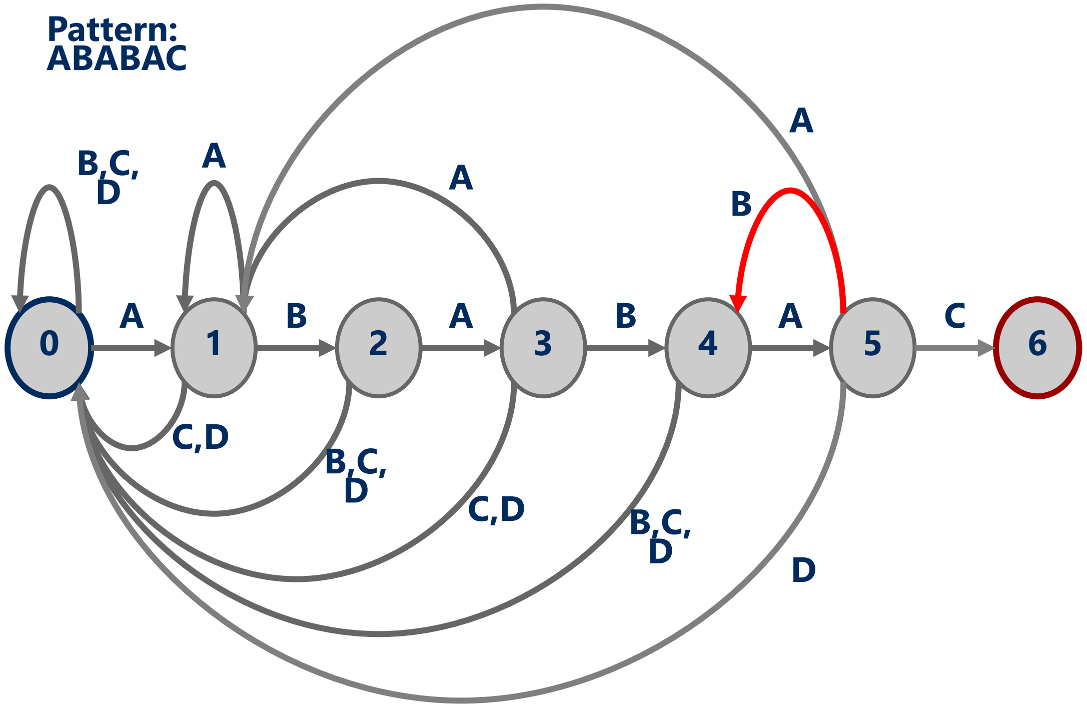


dfa[][]

	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

Pattern: ABABAC



	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	C
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					

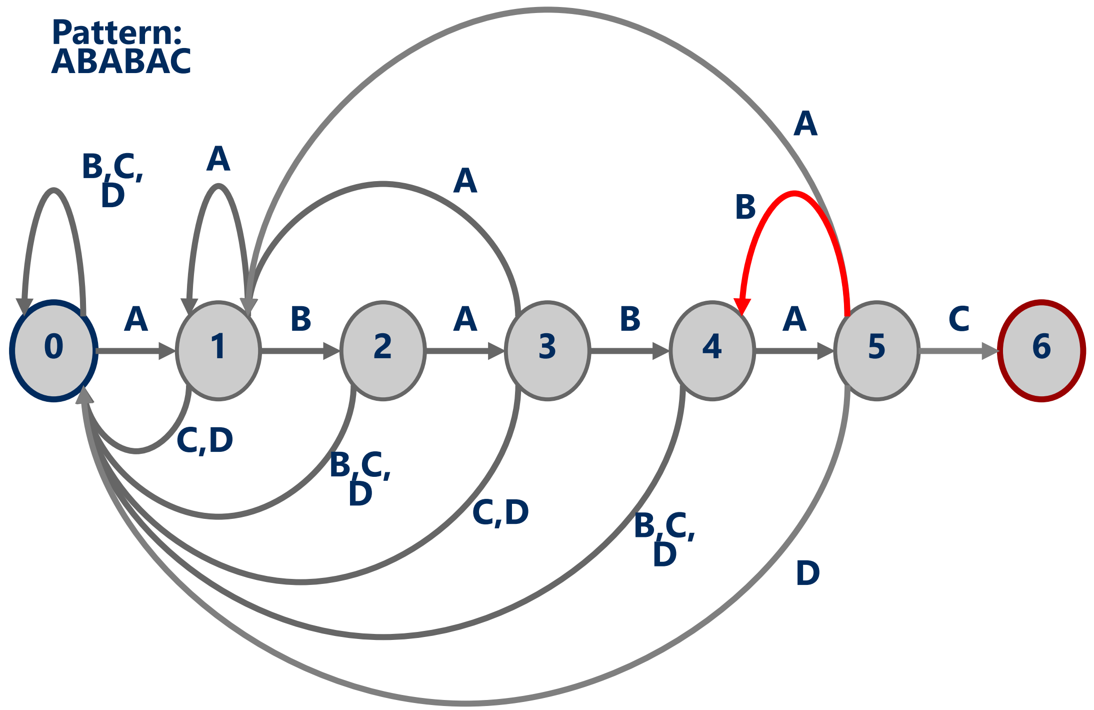
```
public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n – m && j < m; i++) {
       if (txt.charAt(i) == pat.charAt(j))
             j++;
       else { i -= j; j = 0; }
    }
    if (j == m)
           return i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

i:					4	5		
text:	A	B	A	B	A	B	A	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:						5	6	
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
}
```



	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