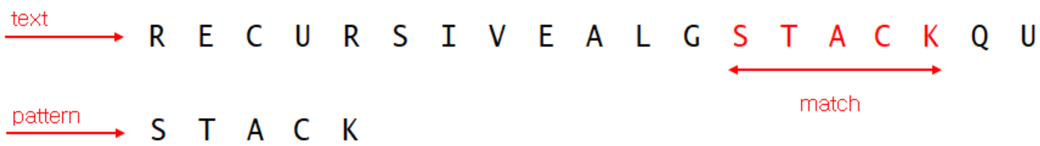
String Matching

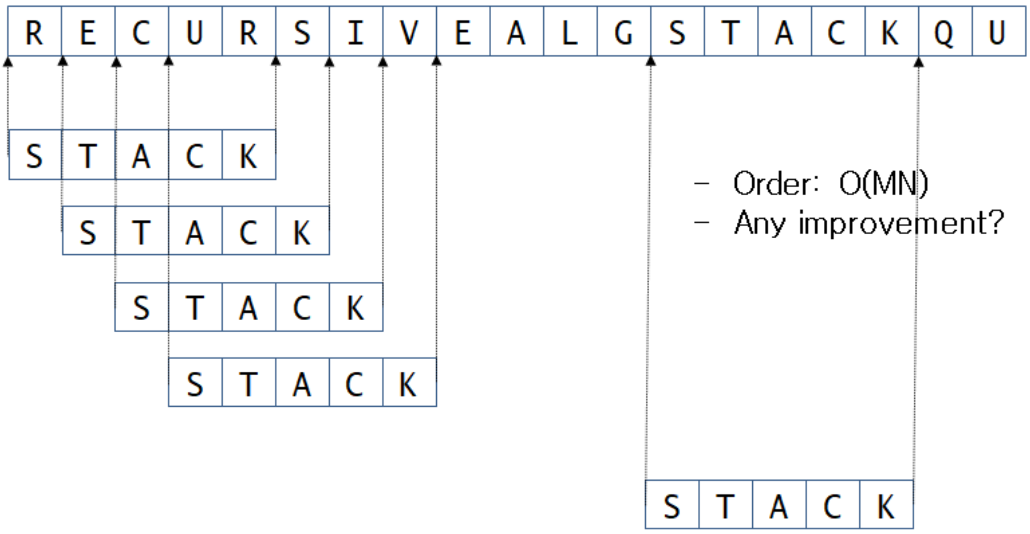
- Substring search
  - Find a pattern of length M in a text of length N
    (typically N >> M)



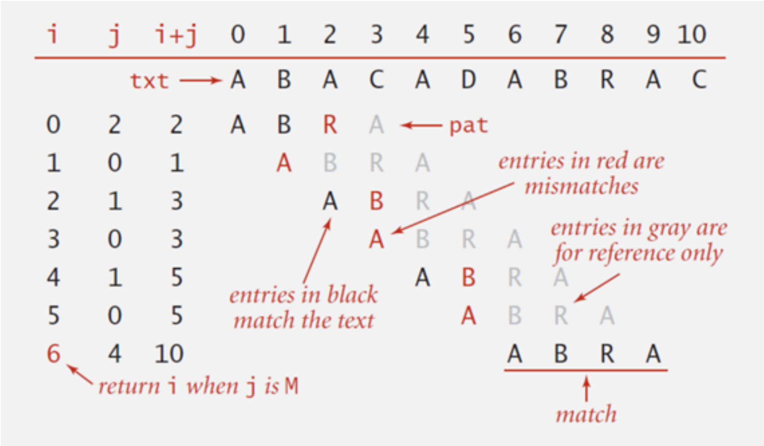
Brute-Force Substring Search

Naive algorithm

- Check for pattern starting at each text position

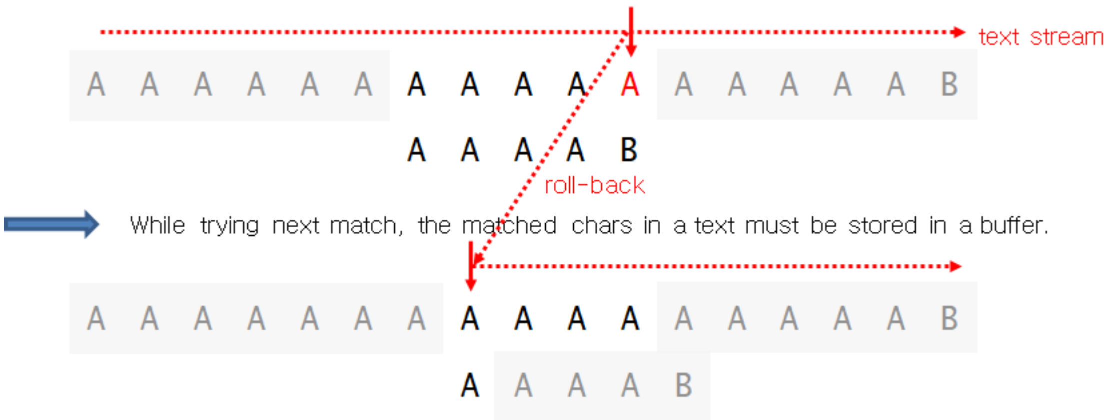



```
int naiveStringMatch(char text[], char pattern[])
{
    int patLength, txtLength;

    patLength = strlen(pattern);
    txtLength = strlen(text);

    for(int i=0; i <= txtLength - patLength; i++)
    {
        for(int j=0; j < patLength; j++)
            if(text[i+j] != pattern[j])
                break;
        if(j == patLength)
            return i;
    }
    return -1;
}
```



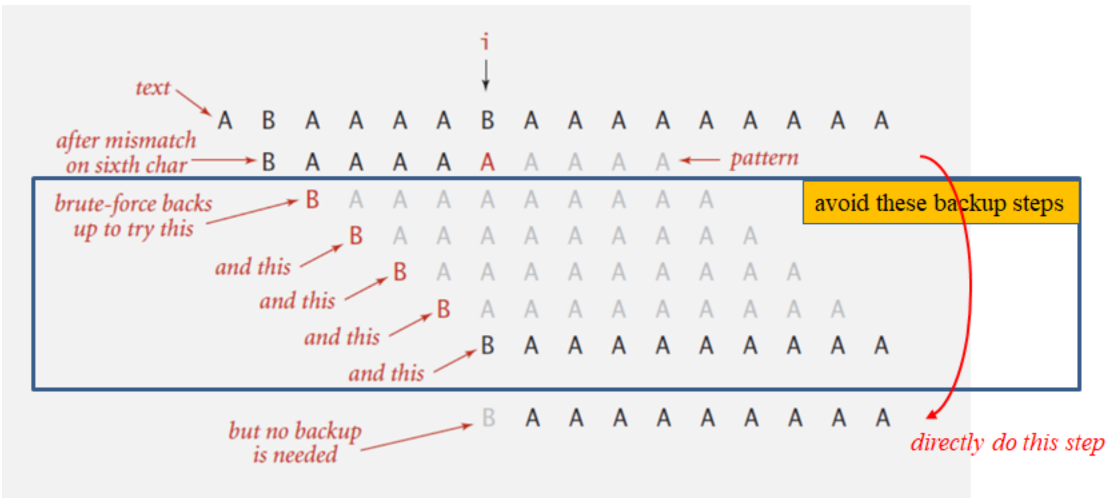
- can be slow if text and pattern are repetitive

- Improvement
    - develop a linear time algorithm
    - avoid **backup**
        - naive algorithm needs backup for every mismatch
        - thus naive algorithm cannot be used when input text is a stream



Knuth-Morris-Pratt(KMP) Algorithm

- Clever method to always avoid **backup** problem

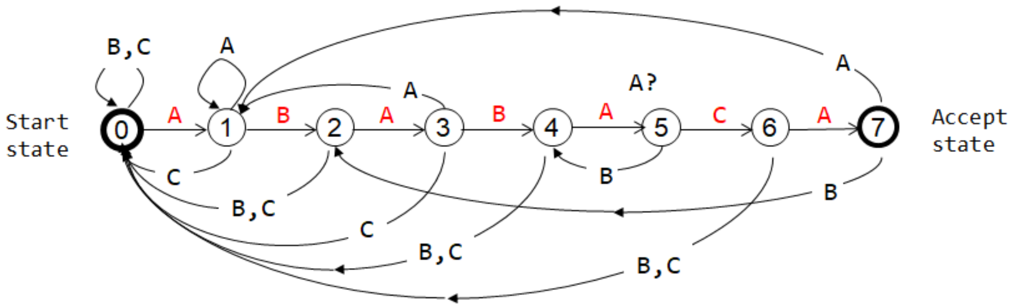


Deterministic Finite Automaton

- DFA
    - Finite number of states (including **start** and **accept states**)
    - Exactly one transition for each char

- Accept if sequence of transitions leads to accept state

DFA for pattern ABABACA

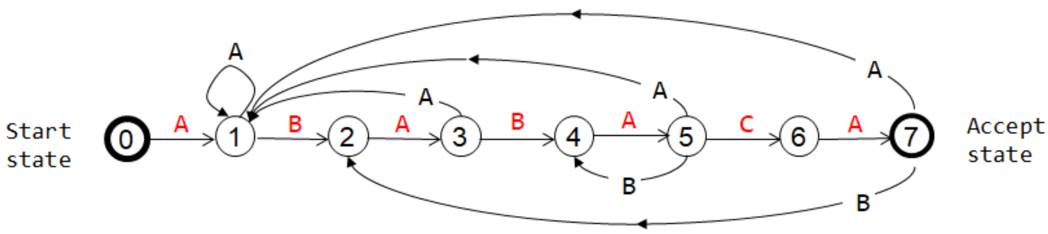


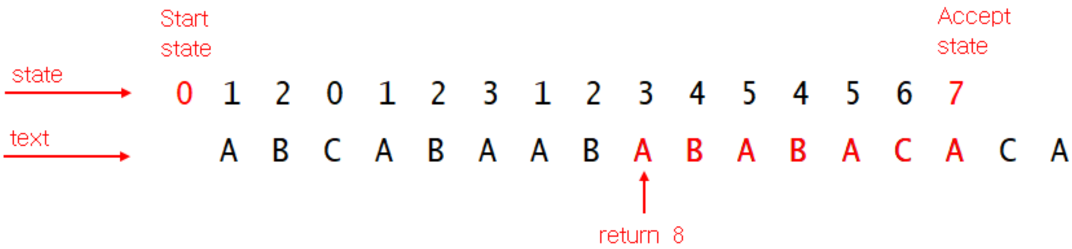
DFA[][]

char \ 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
others	0	0	0	0	0	0	0	0

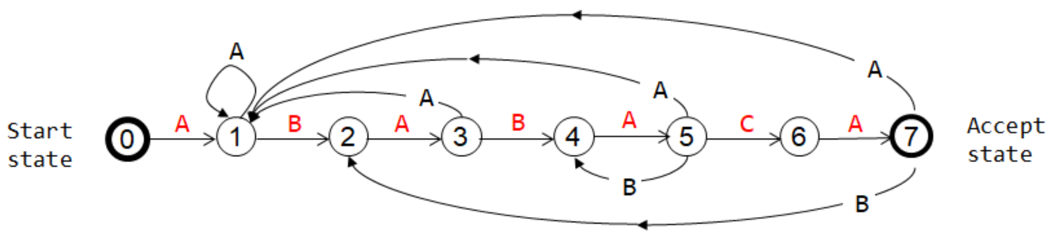
```
if in state j reading char c:
    if j is 7, halt and accept
    else move to state DFA[c][j]

Example:
    text:  ABCABAABABABACACA
    state:012012312345456 7
```

DFA for pattern ABABACA





Simplified Diagram: remove transitions to state 0

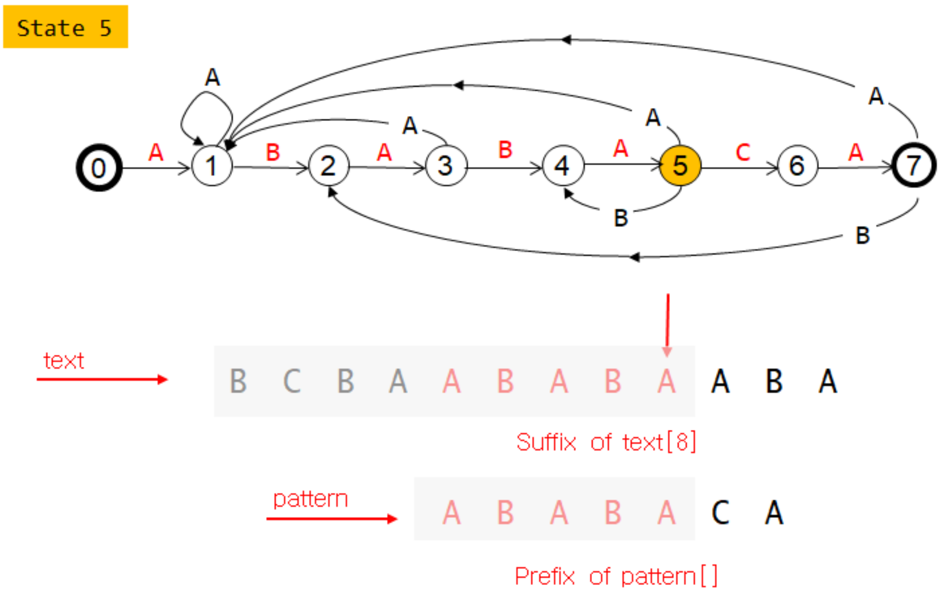


- Difference from naive algorithm
  - precomputation of DFA[][] from pattern
  - text pointer `i` never decrements (**no backup**)

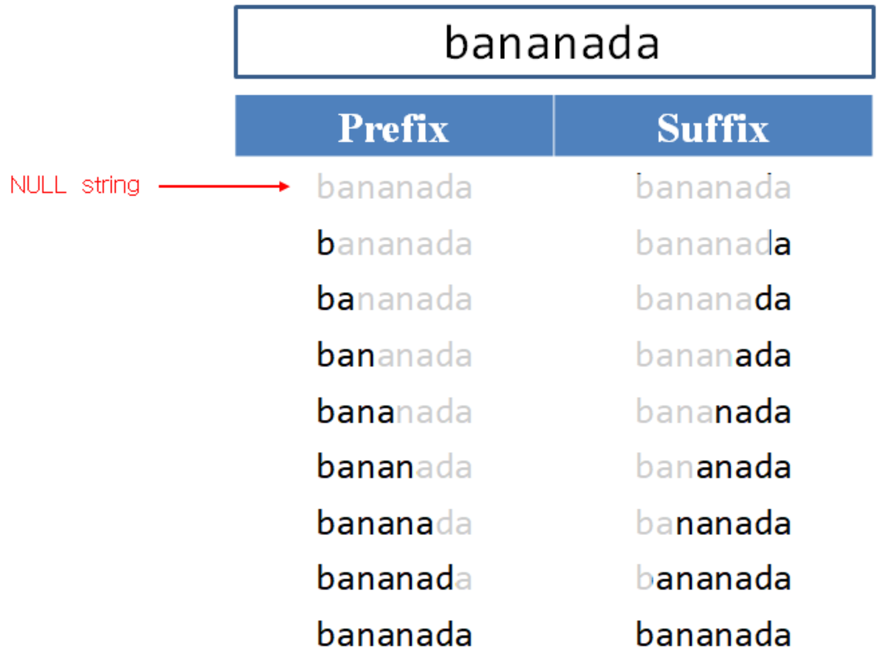
```
// patLength = strlen(pattern);
int DFAmatching(char text[])
{
    int i, j, txtLength;

    txtLength = strlen(text);

    for(i=0, j=0; i <= txtLength && j < patLength; i++)
        j = DFA[text[i]][j];  // text[i] to be modified

    if(j == patLength)
        return i - patLength;
    else
        return -1;
}
```

- Order: O(N)
  simulation of DFA on text with no backup
- How to build DFA efficiently?

- The state of DFA represents
  - the number of characters in pattern that have been matched

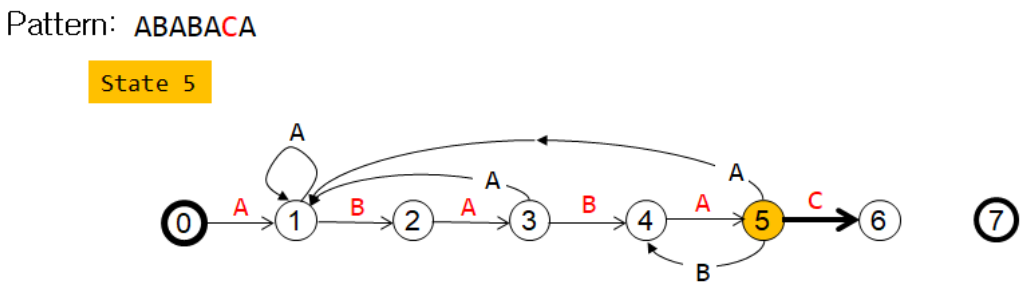


- Prefix / Sufix of a Text

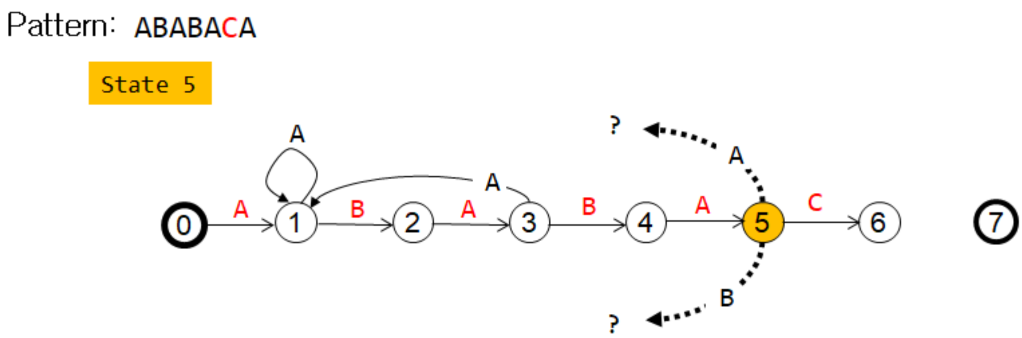


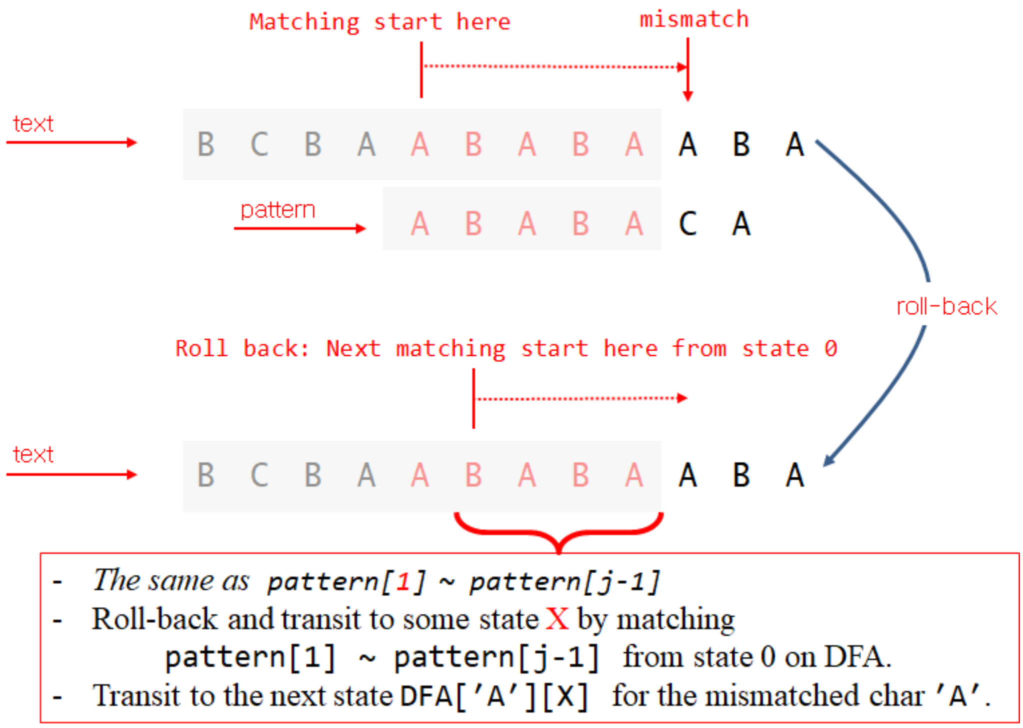
DFA Construction

- Suppose that all transitions from state `0` to stat `j-1` are already computed
- Match transition
    - If in state `j` and next char `char c = pattern[j]`, then transit to state `j+1`



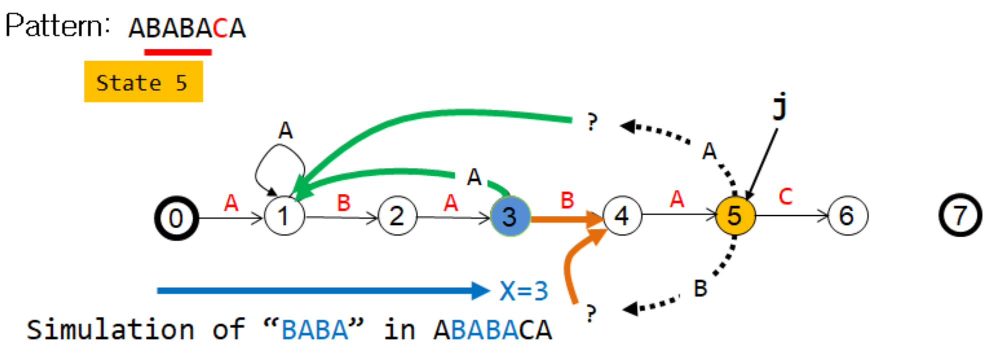
- Mismatch transition
    - If in state `j` and next `char c != pattern[j]`, then which state to transit





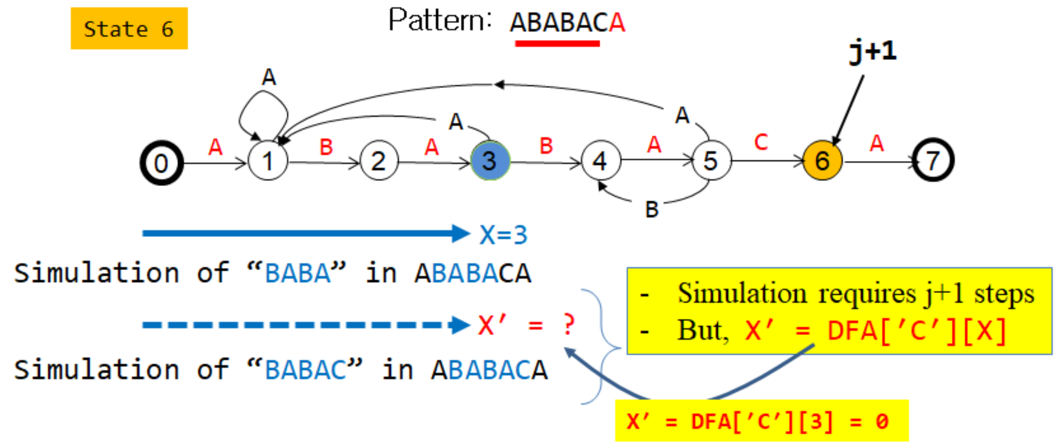
- then the last `j-1` characters of input text are `pattern[1] ~ pattern[j-1]`, followed by `c`
- to compute `DFA[c][j]`:
  - simulate `pattern[1] ~ pattern[j-1]` on DFA (still under construction) and let the current state `X`
  - Then `DFA[c][j] = DFA[c][X]`

```
DFA['A'][5] = DFA['A'][3] = 1
DFA['B'][5] = DFA['B'][3] = 4
```



  - take a transition `c` from state `X`
  - Running time : require `j` steps
  - **But, if we maintain state X, it takes only constant time!**

- Maintaining state `X`:
  - finished computing transitions from state `j`
  - Now, move to next state `j+1`
  - then what the new state(`X'`) of `X` be?



- A Linear Time Algorithm
    - for each state `j`
        - Match case : set `DFA[pattern[j]][j] = j+1`
        - Mismatch case : copy `DFA[][X]` to `DFA[][j]`
        - Update `X`

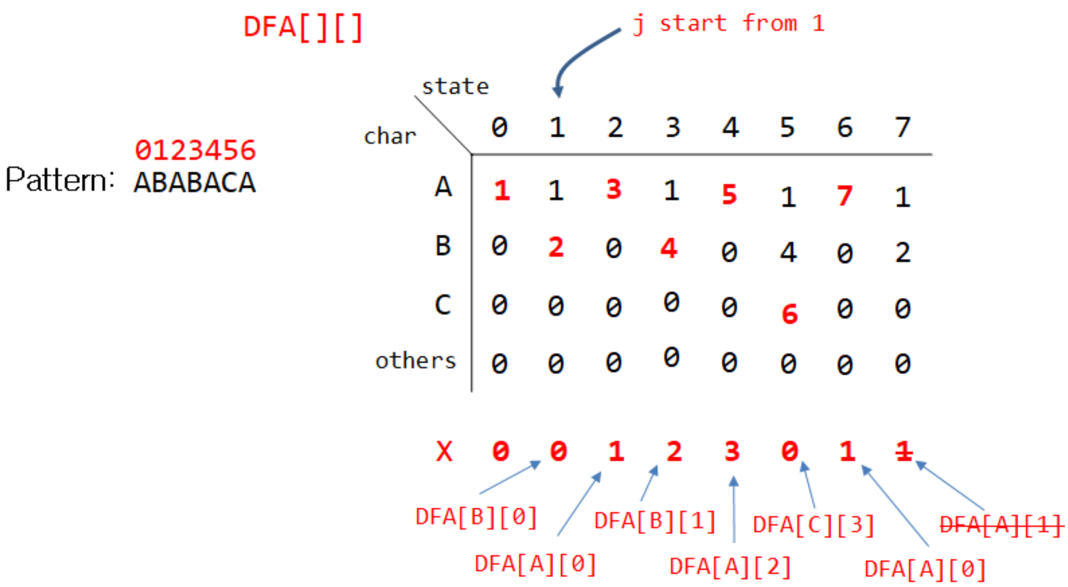
```
int DFA[MAX_SIZE][MAX_SIZE];  /* initially all elements are 0 */
// int R;            /* text character set size */

void constructDFA(char pattern[])
{

    int patLength = strlen(pattern);

    DFA[pattern[0]][0] = 1;

    for(int X=0, j=1; j<patLength; j++)
    {
        for(int c=0; c<R; c++)              // copy mismatch cases
            DFA[c][j] = DFA[c][X];

        DFA[pattern[j]][j] = j+1;           // copy match case
        X = DFA[pattern[j]][X];             // update X
    }
}
```

- Example

DFA[][]

0123456
Pattern: ABABACA

char \ state	0	1	2	3	4	5	6	7
A								
B								
C								
others								




```
int patLength = strlen(pattern);

DFA[pattern[0]][0] = 1;

for(int X=0, j=1; j<patLength; j++)
{
   for(int c=0; c<R; c++)
      DFA[c][j] = DFA[c][X];

   DFA[pattern[j]][j] = j+1;
   X = DFA[pattern[j]][X];
}
```

Algorithm with DFA

- String matching algorithm with DFA accesses no more than M+N chars to search for a pattern of length M in a text of length N
- `DFA[][]` can be constructed in time and space of order `O(RM)`, where `R` is the number of characters used in a text
- Questions : Text에 나타나는 모든 pattern을 찾을 수 있는가?
  - Text : AAAAAAAAA
  - Pattern : AAAAA
  - Solution : 0, 1, 2, 3, 4, 5