Ch. 32 String Matching

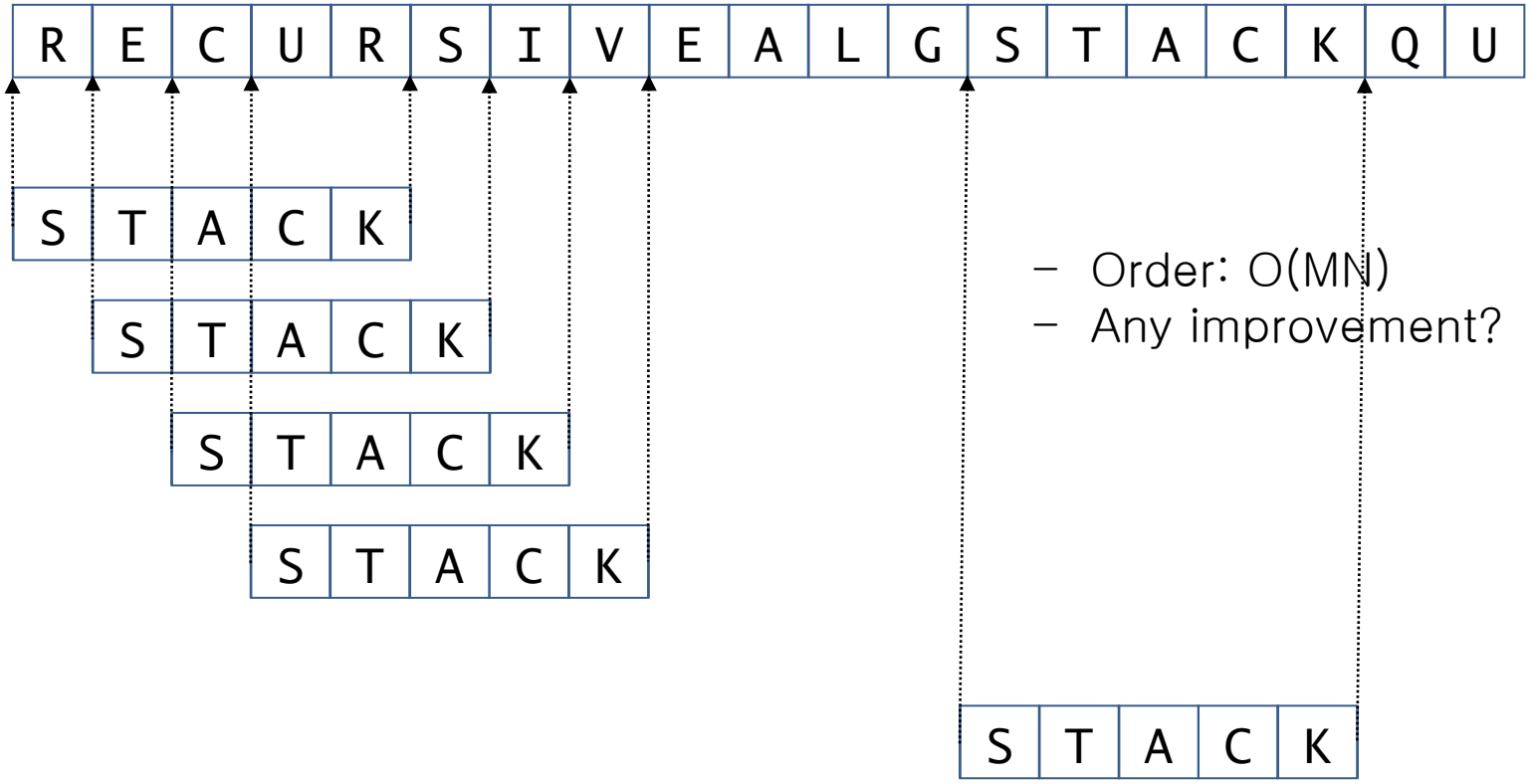
국민대학교 컴퓨터공학부 최준수
Modified by 임은진

String Matching

- Substring search
  - Find a pattern of length M in a text of length N. (typically $N \gg M$)

Brute-Force Substring Search

- Naïve Algorithm
  - Check for pattern starting at each text position



Brute-Force Substring Search

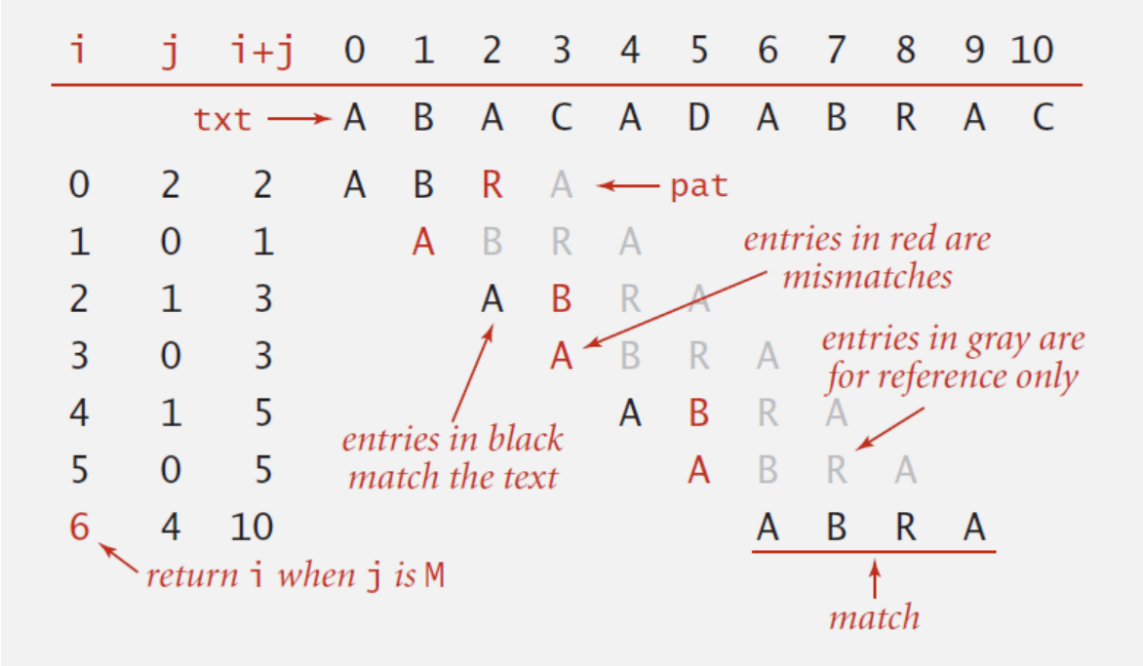
- Naïve Algorithm
  - Check for pattern starting at each text position

```
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;
}
```

Find just only one sample of pattern.
Modify the code to find all samples of the pattern.

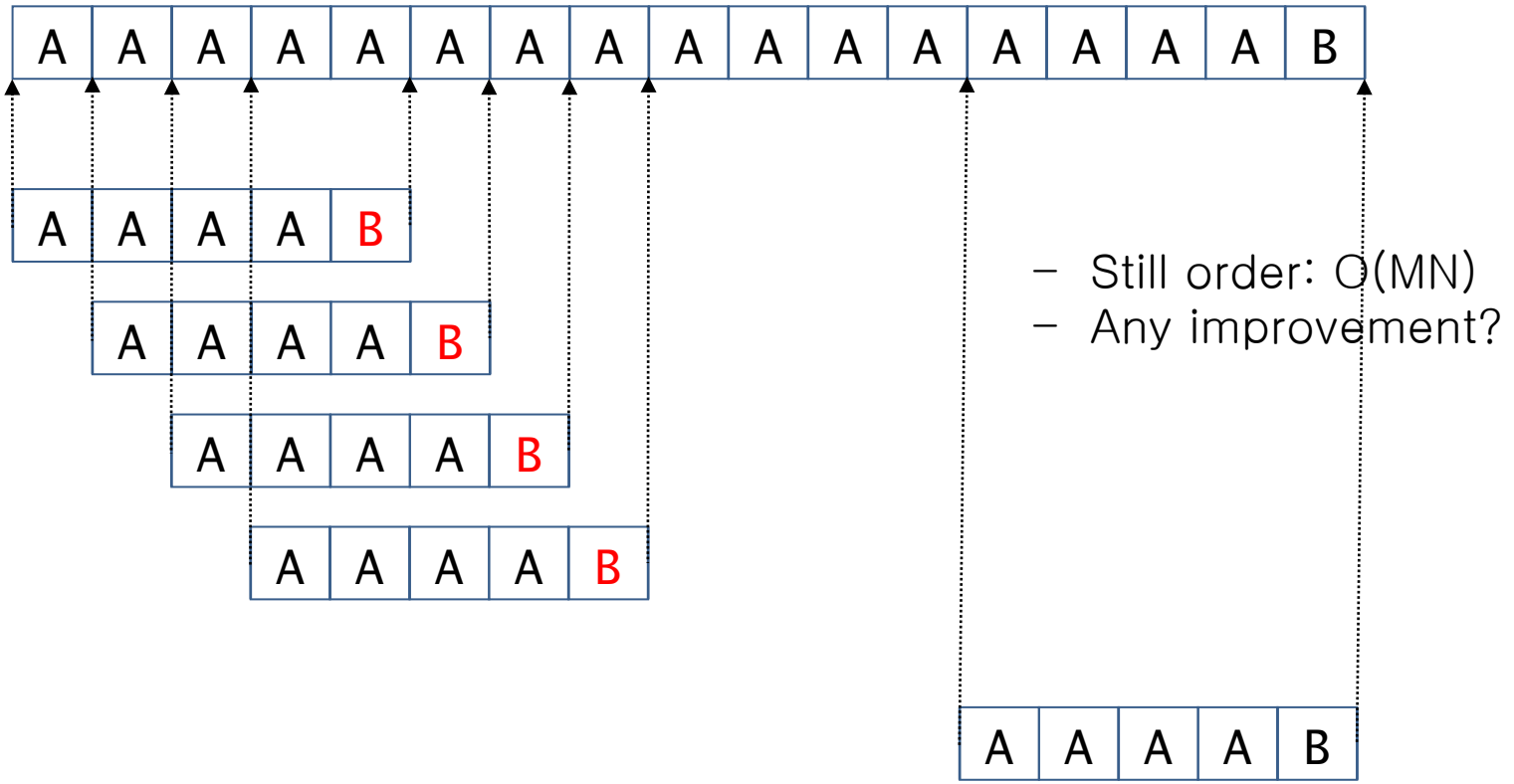
Brute-Force Substring Search

- Naïve Algorithm
  - Check for pattern starting at each text position



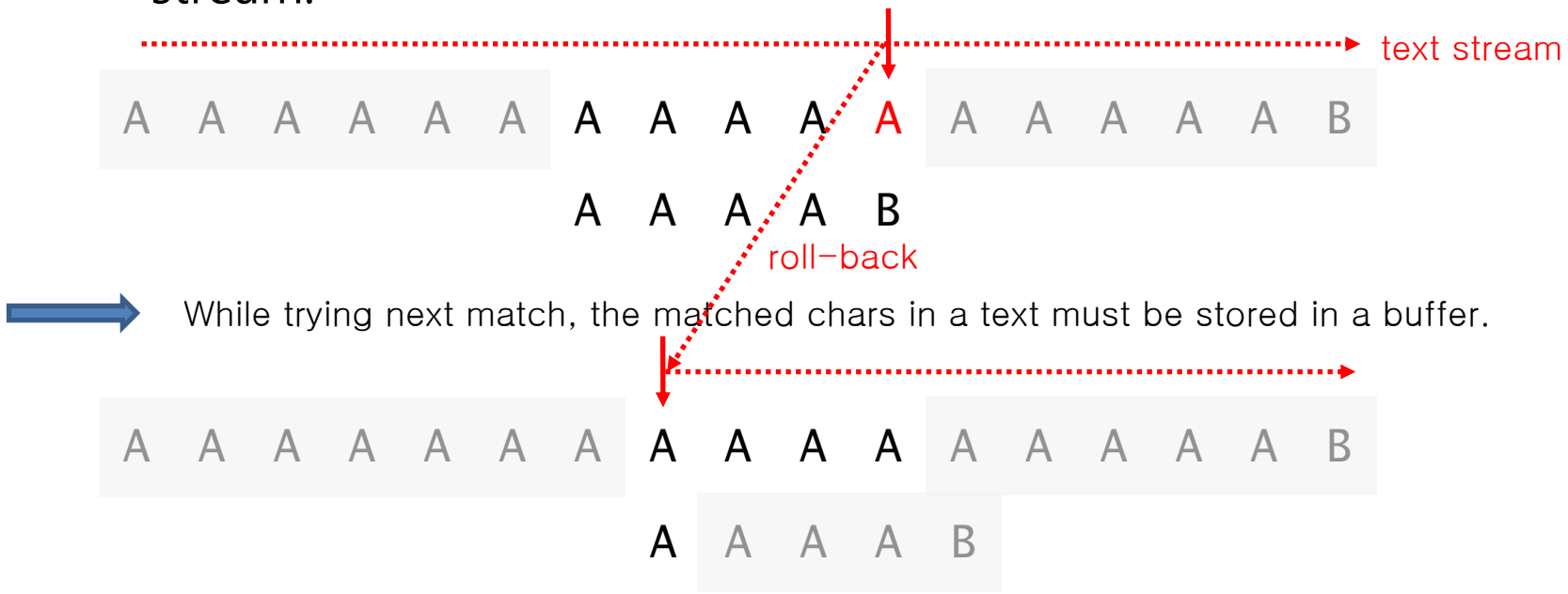
Brute-Force Substring Search

- Naïve Algorithm
  - Naïve algorithm can be slow if text and pattern are repetitive



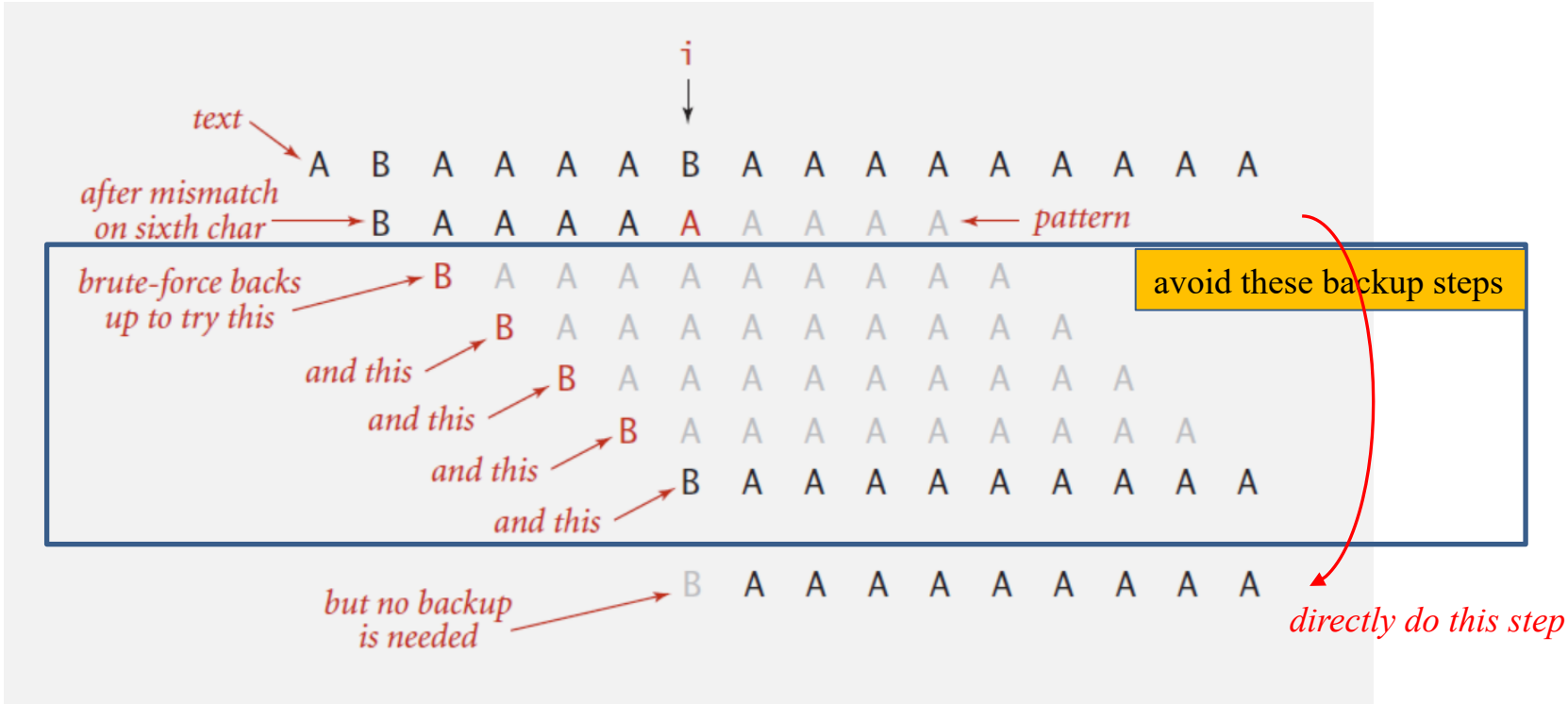
Brute-Force Substring Search

- Improvement
  - Develop a linear time algorithm
  - Avoid backup
    - Naïve algorithm needs backup for every mismatch
    - Thus naïve algorithm cannot be used when input text is a stream.



String Matching without backup

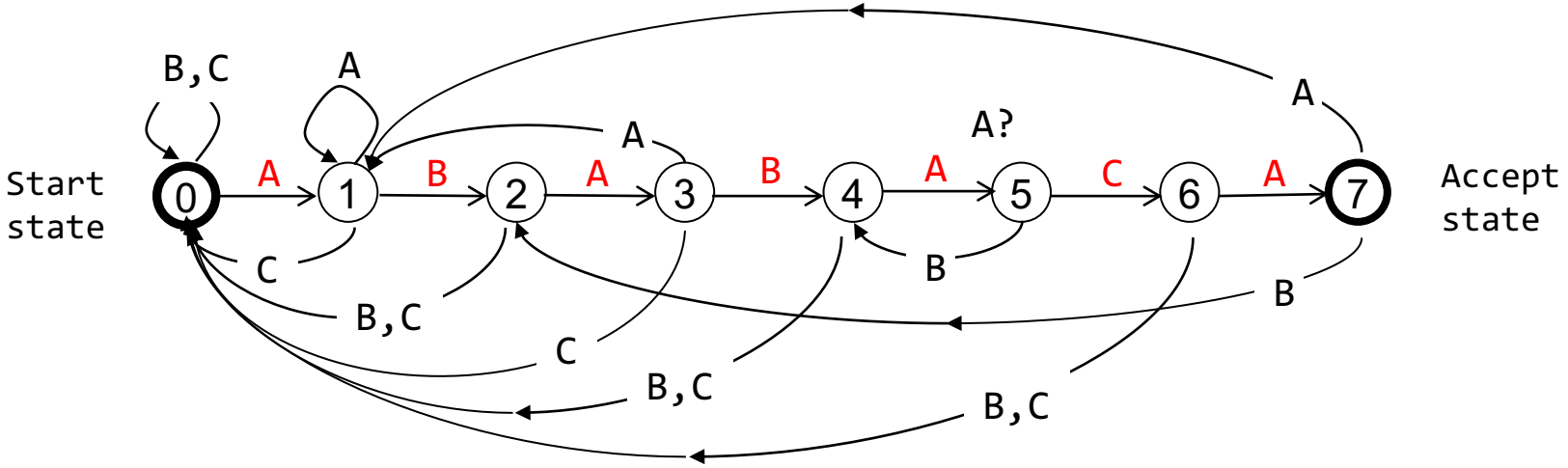
– Clever method to avoid backup problem.



Deterministic Finite Automaton

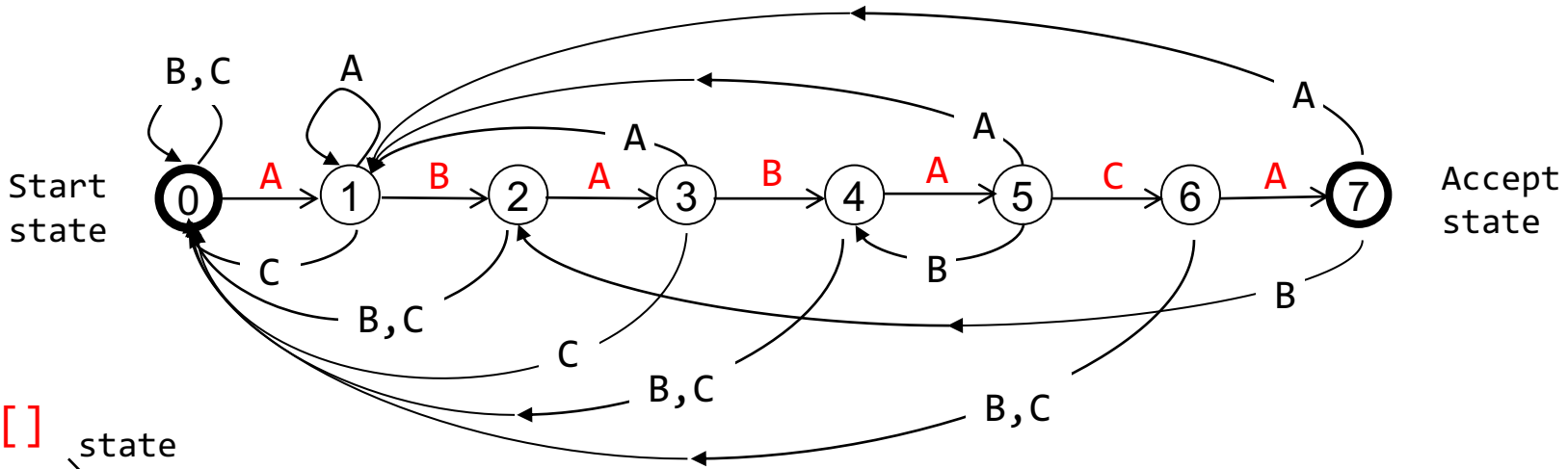
- DFA(Deterministic Finite State Automaton)
  - Finite number of states (including start and accept states)
  - Exactly one transition for each char
  - Accept if sequence of transitions leads to accept state

DFA for pattern ABABACA



representation of DFA

DFA for pattern ABABACA



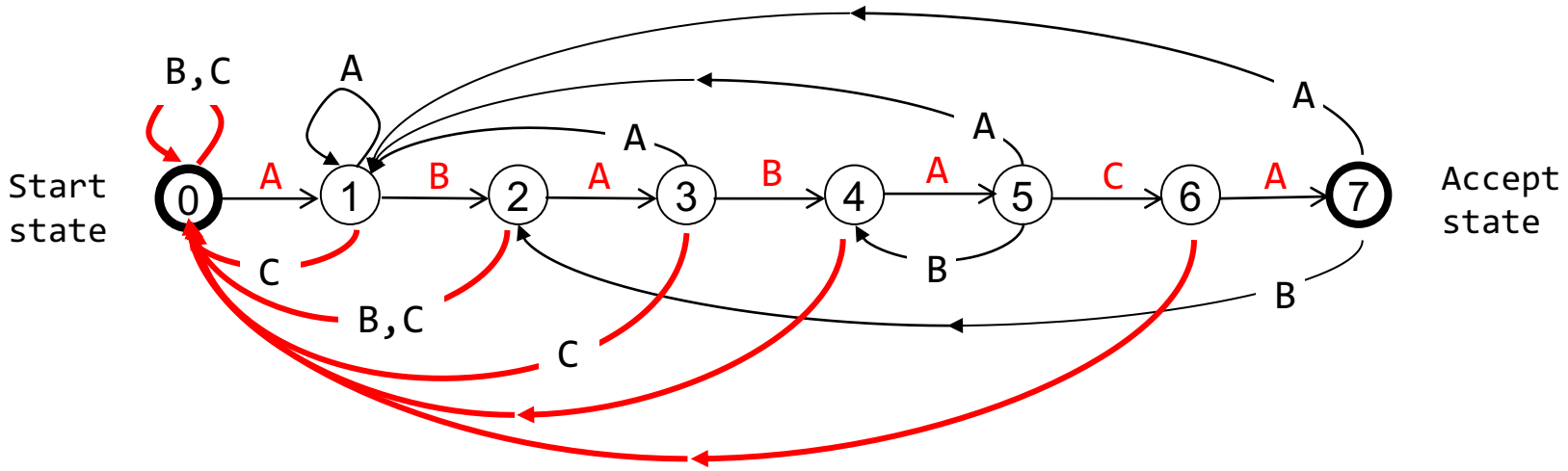
DFA[][]

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

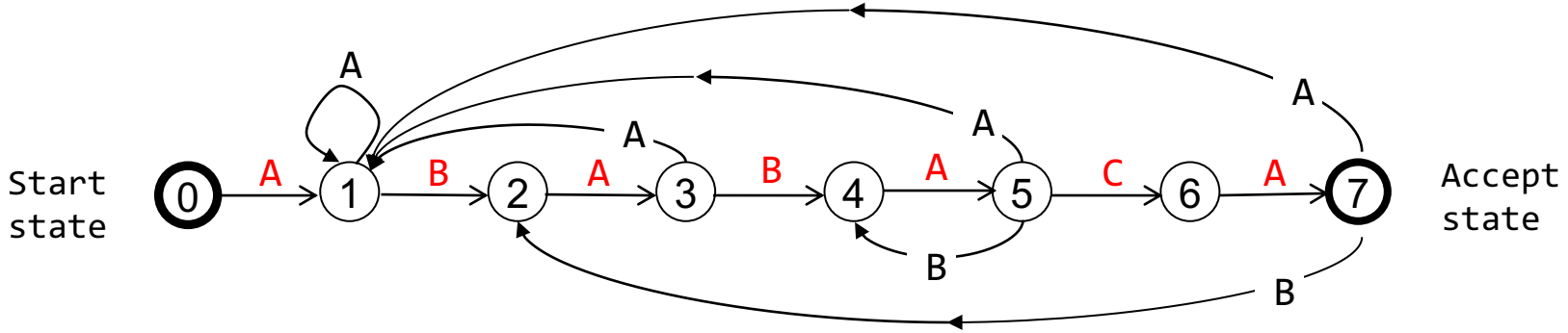
```
if in state j reading char c:
    if j is 7, halt and accept
    else move to state DFA[c][j]

Example:
    text:  ABCABAABABABACACA
    state:0120123123454567
```

simplified diagram of DFA

DFA for pattern ABABACA



Simplified Diagram: remove transitions to state 0



String Matching Algorithm with DFA

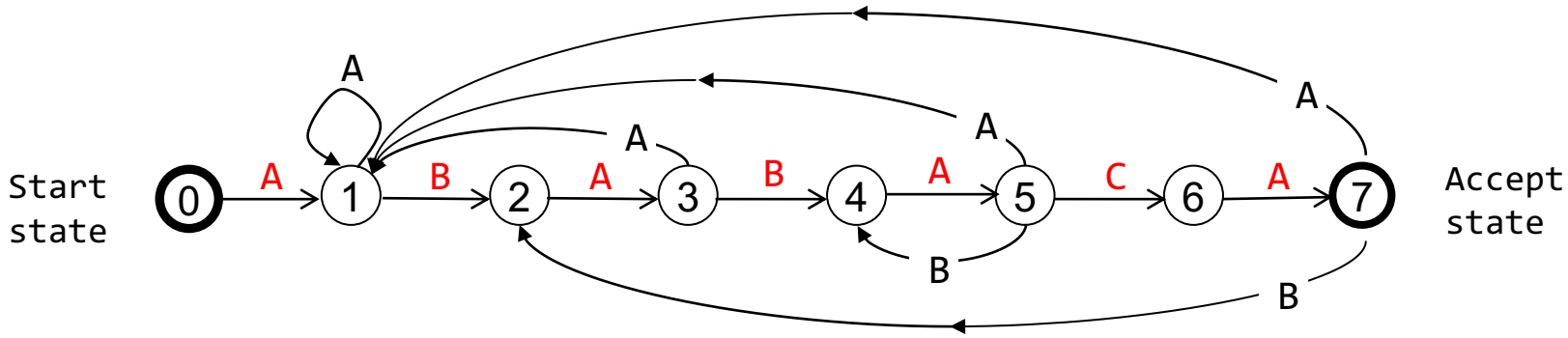
- Difference from naïve algorithm
  - Precomputation of DFA[][] from pattern
  - Text pointer `i` never decrements (no backup)

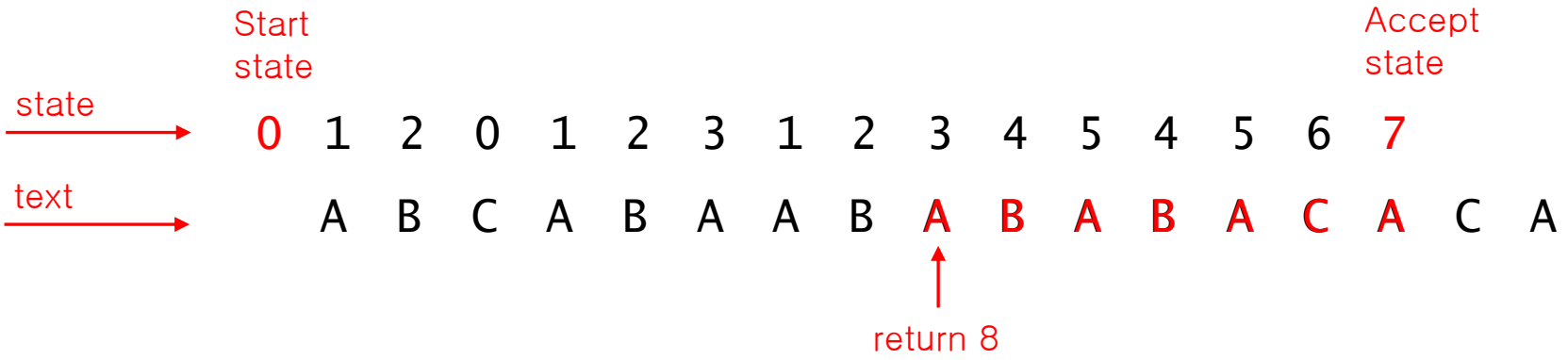
```
// patLength = strlen(pattern);
int DFAmatching(char text[])
{
    int i, j, txtLength;

    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?

Algorithm with DFA

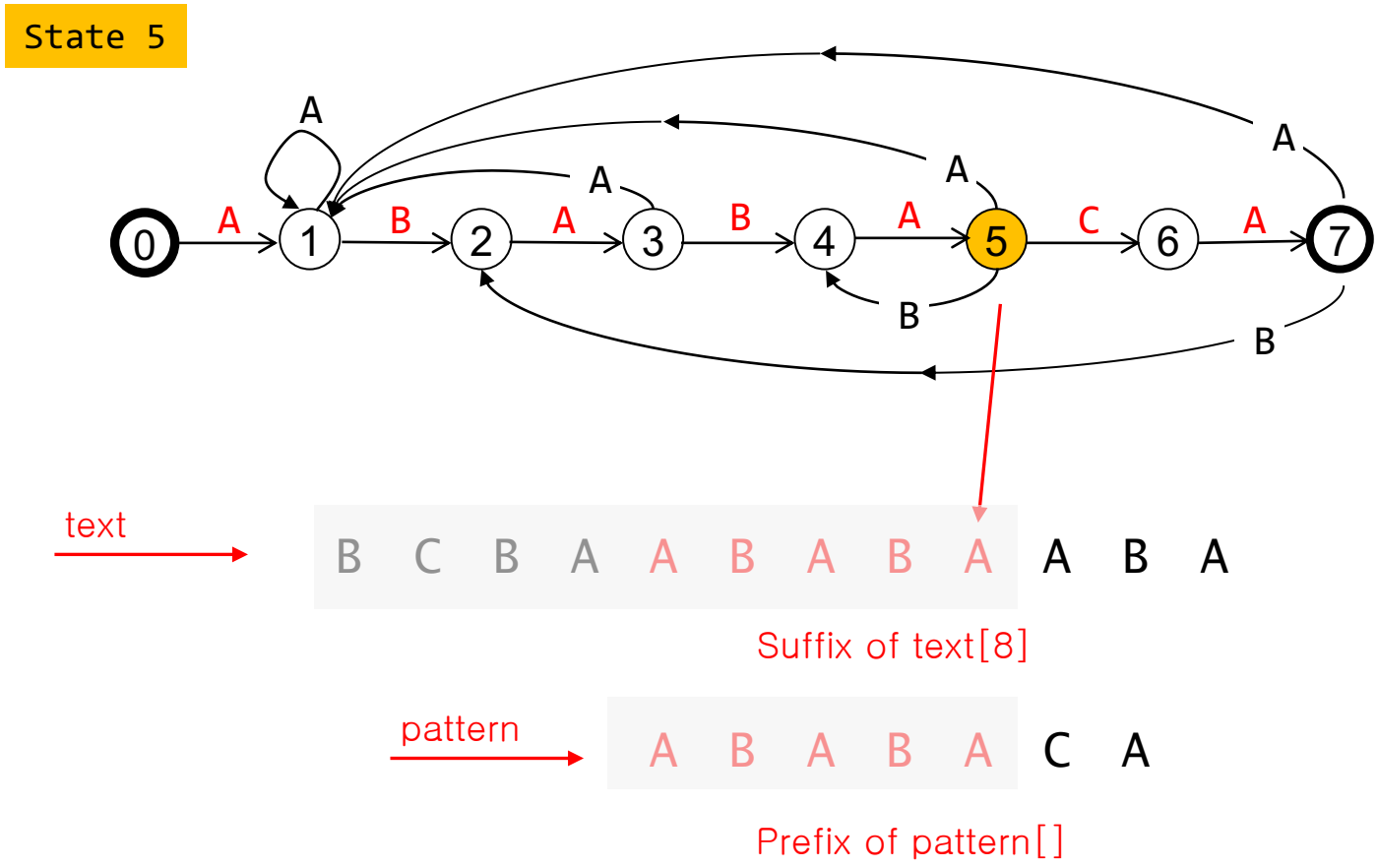
DFA for pattern ABABACA





Interpretation of DFA

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



Prefix/Suffix

- Prefix / Suffix of a Text

bananada

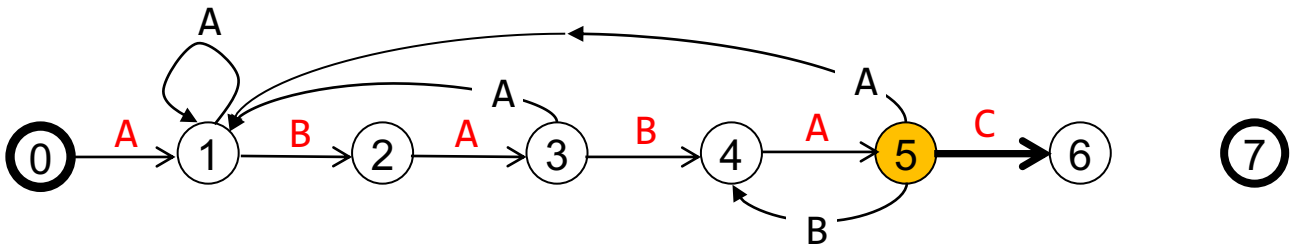
Prefix	Suffix
NULL string → bananada	bananada
bananada	bananada
bananada	bananada
bananada	bananada
bananada	bananada
bananada	bananada
bananada	bananada
bananada	bananada
bananada	bananada

DFA Construction 1

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

Pattern: ABABACA

State 5

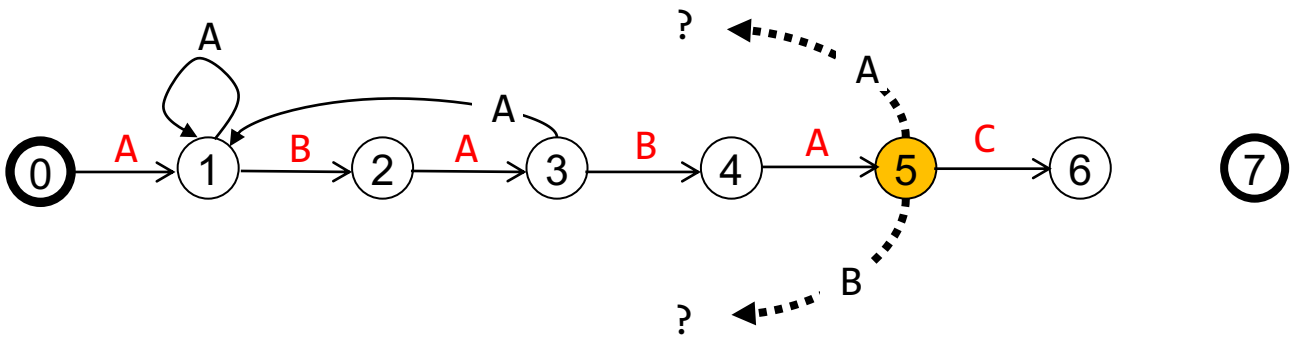


DFA Construction 2

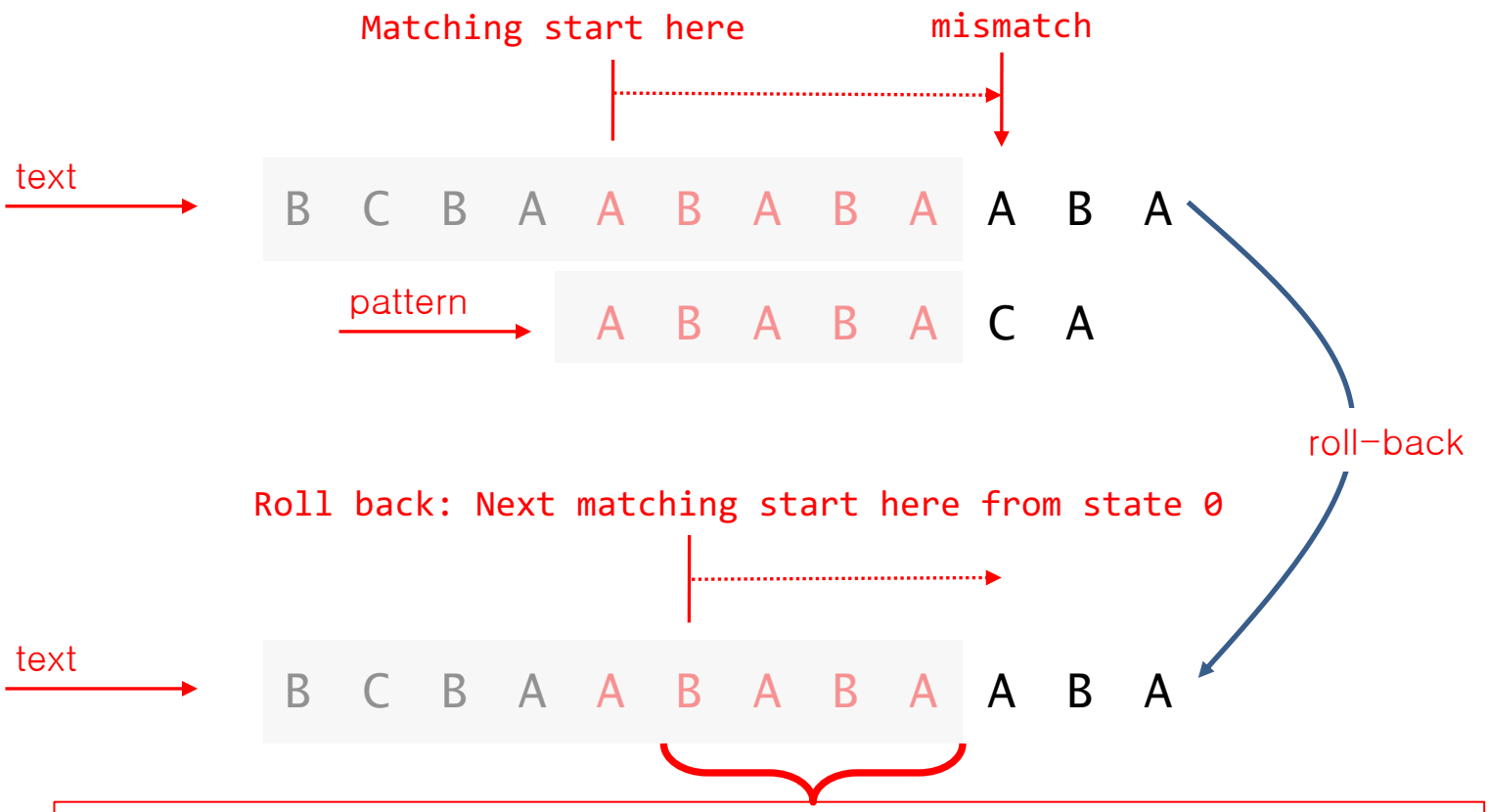
- DFA Construction:
  - Mismatch transition:
    - If in state `j` and next char `c != pattern[j]`, then which state to transit?

Pattern: ABABACA

State 5



DFA Construction 3



- *The same as* `pattern[1] ~ pattern[j-1]`
- Roll-back and transit to some state X by matching
  `pattern[1] ~ pattern[j-1]` from state 0 on DFA.
- Transit to the next state `DFA['A'][X]` for the mismatched char `'A'`.

DFA Construction 4

- DFA Construction:
  - Mismatch transition:
    - If in state `j` and next char `c != pattern[j]`,

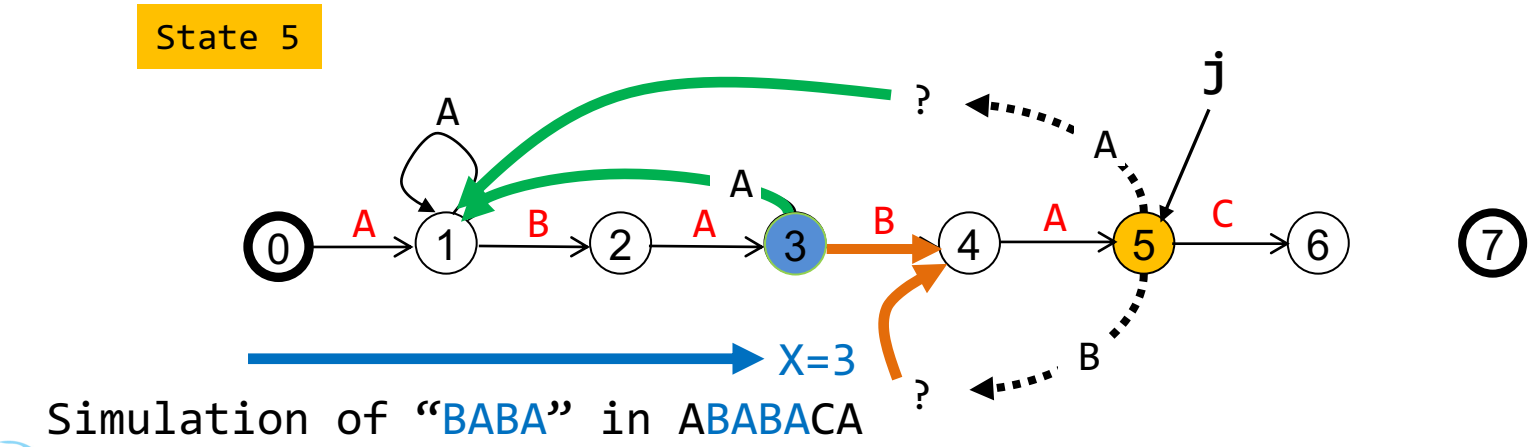
    then the last `j-1` characters of input text are `pattern[1]` ~ `pattern[j-1]`, followed by `c`.
  - Compute `DFA[c][j]`:
    - Simulate `pattern[1]` ~ `pattern[j-1]` on DFA from state 0 and let X be the current state
    - Then `DFA[c][j] = DFA[c][X]`

DFA Construction 5

- DFA Construction:
  - Mismatch transition:
    - `DFA[c][j] = DFA[c][X]`

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

Pattern: ABABACA



Simulation of "BABA" in ABABACA

Analysis of DFA Construction

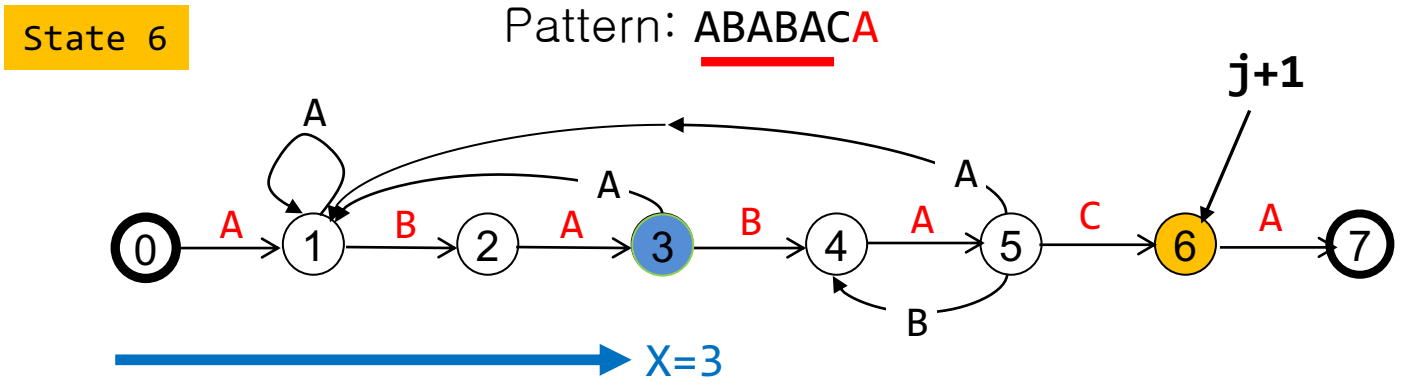
- DFA Construction:
  - Mismatch transition:
    - If in state `j` and next char `c != pattern[j]`,

      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`.
    - take a transition `c` from state `X`.
    - Running time : require `j` steps.

      → $O(M^2)$ *where M is a length of the pattern*
    - But, if we maintain state `X`, it takes only constant time! → $O(M)$

Optimizing DFA Construction

- DFA Construction:
  - Maintaining state X:
    - Finished computing transitions from state `j`.
    - Now, move to the next state `j+1`.
    - Then what the new state(`X'`) of X be?



Linear Time DFA Construction

- 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
    }
}
```

Linear Time DFA Construction

- DFA Construction: Example

DFA[][]

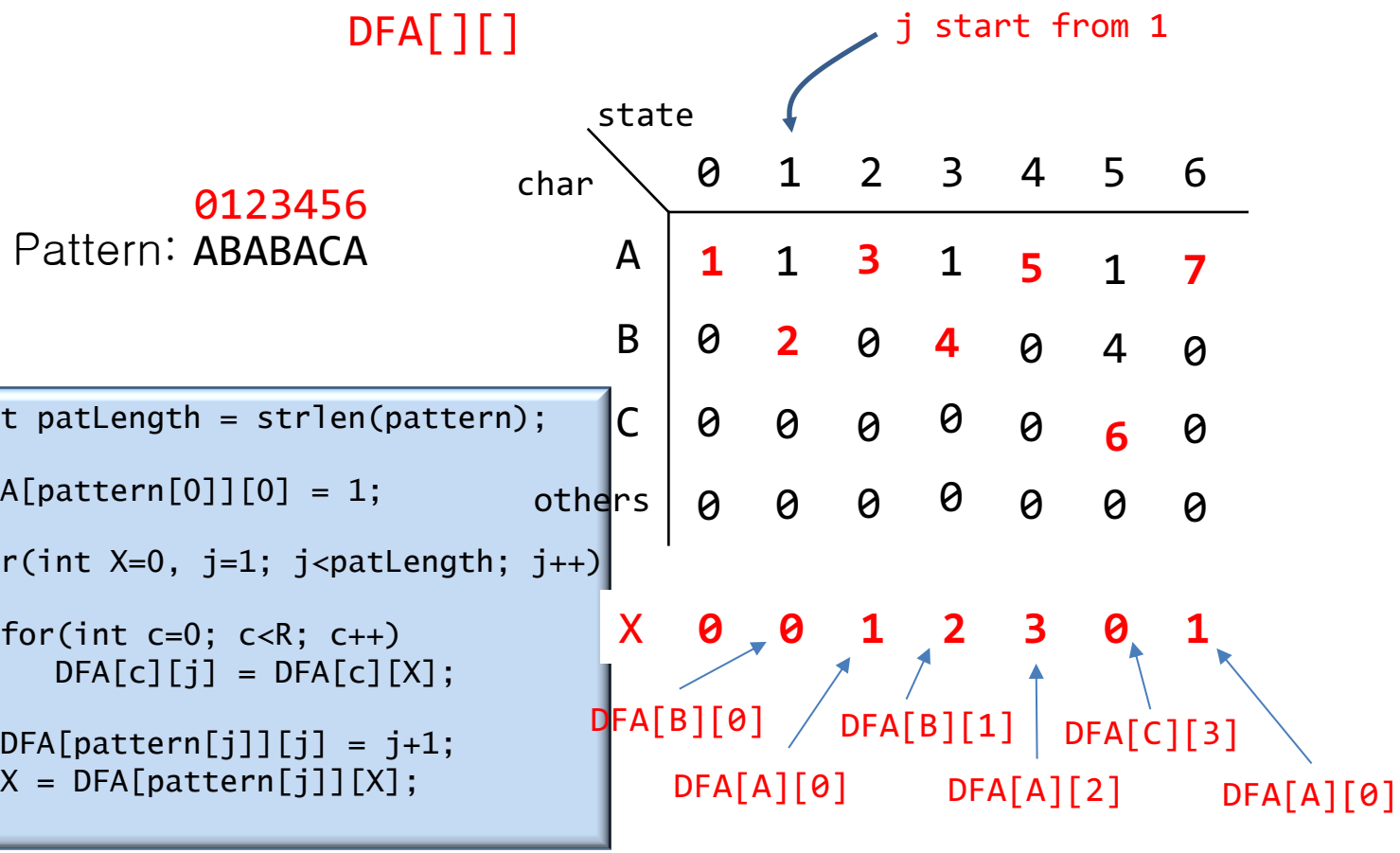
Pattern: ABABACA

0123456

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

DFA Construction

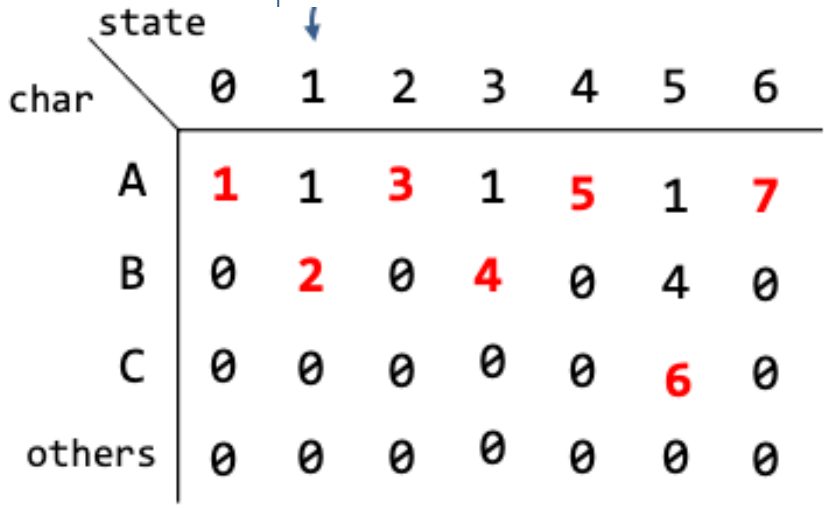
- DFA Construction: Example




```
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];
}
```

String Matching Algorithm with DFA

```
// 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;
}
```



String Matching 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.

Multiple String Matching

- Questions:
  - Text에 나타나는 모든 pattern 을 찾을 수 있는가?
    - Text: AAAAAAAAA
    - Pattern: AAAAA
    - 해답: 0, 1, 2, 3, 4, 5