String Matching

2023-Fall

국민대학교 소프트웨어학부 최준수

String Matching

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

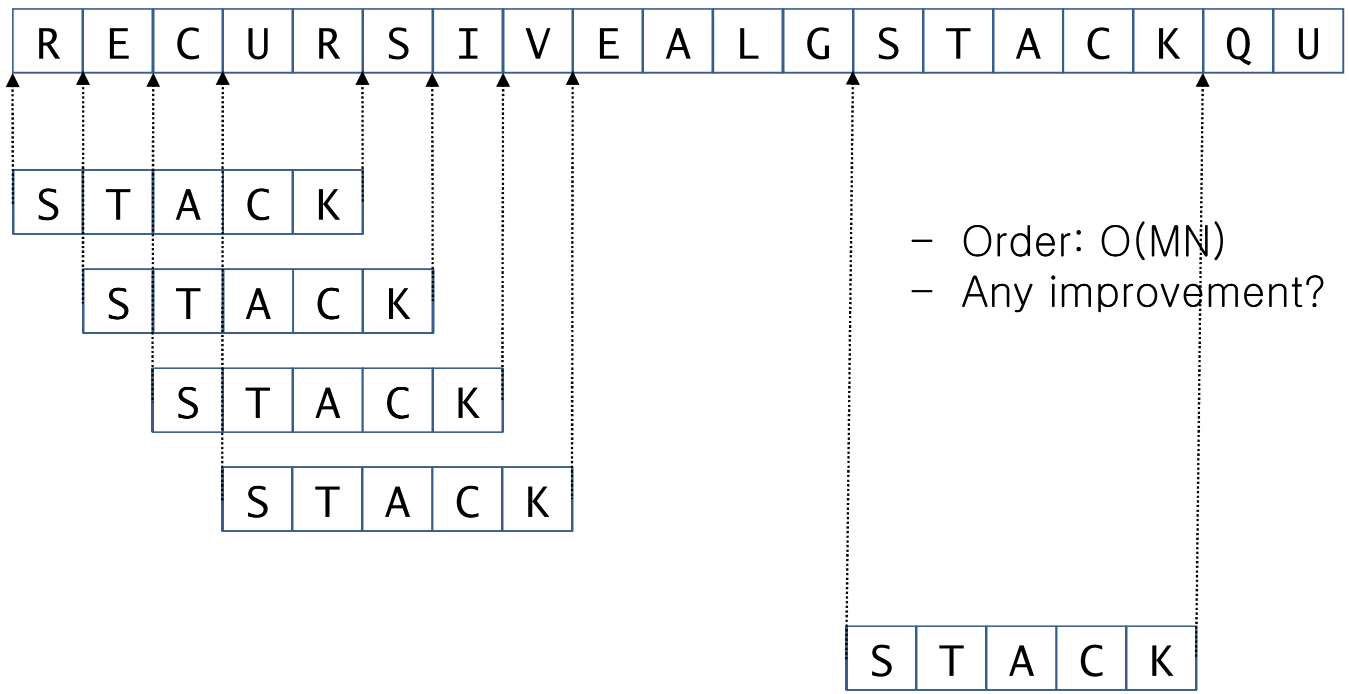
text → R E C U R S I V E A L G S T A C K Q U

match

pattern → S T A C K

Brute-Force Substring Search

- Naïve Algorithm 1
  - Check for pattern starting at each text position



- Order: O(MN)
- Any improvement?

Brute-Force Substring Search

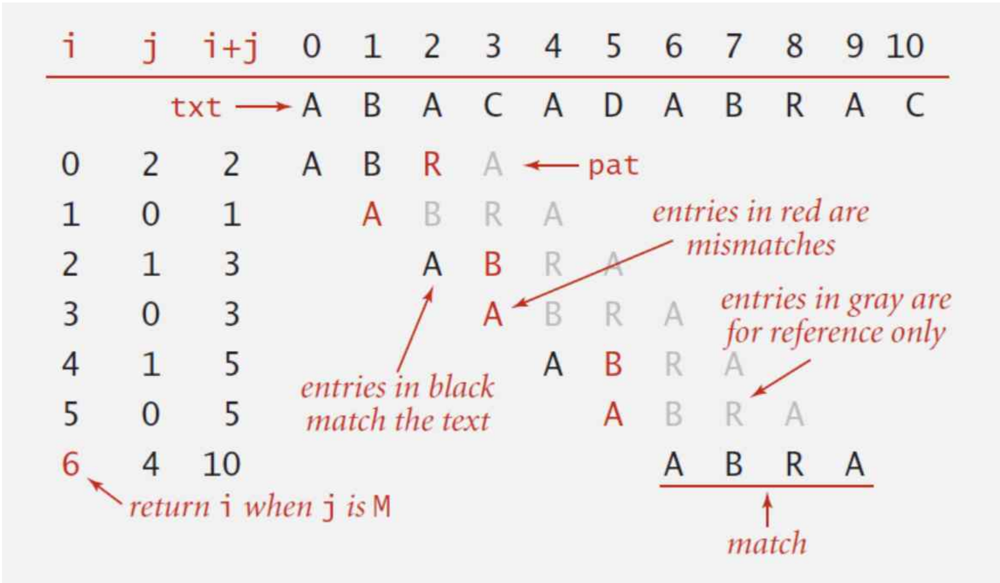
- Naïve Algorithm 2
  - Check for pattern starting at each text position

```
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

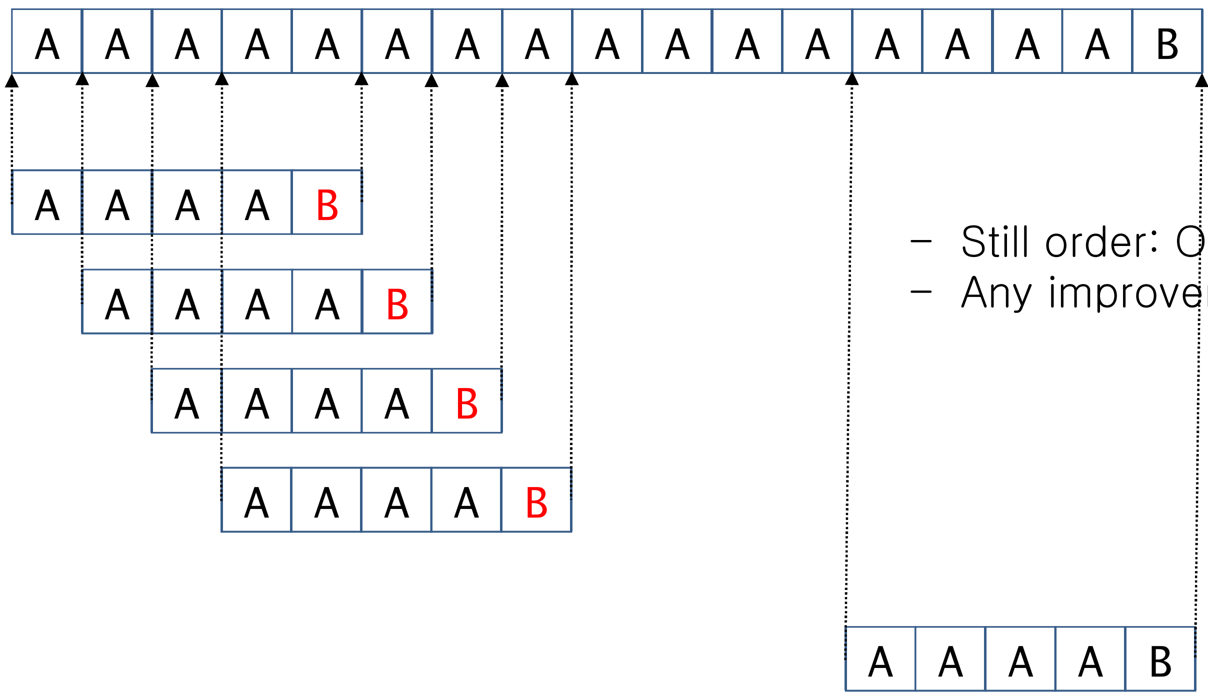
- Naïve Algorithm 2
  - Check for pattern starting at each text position



참고자료: Algorithms, R. Sedgewick

Brute-Force Substring Search

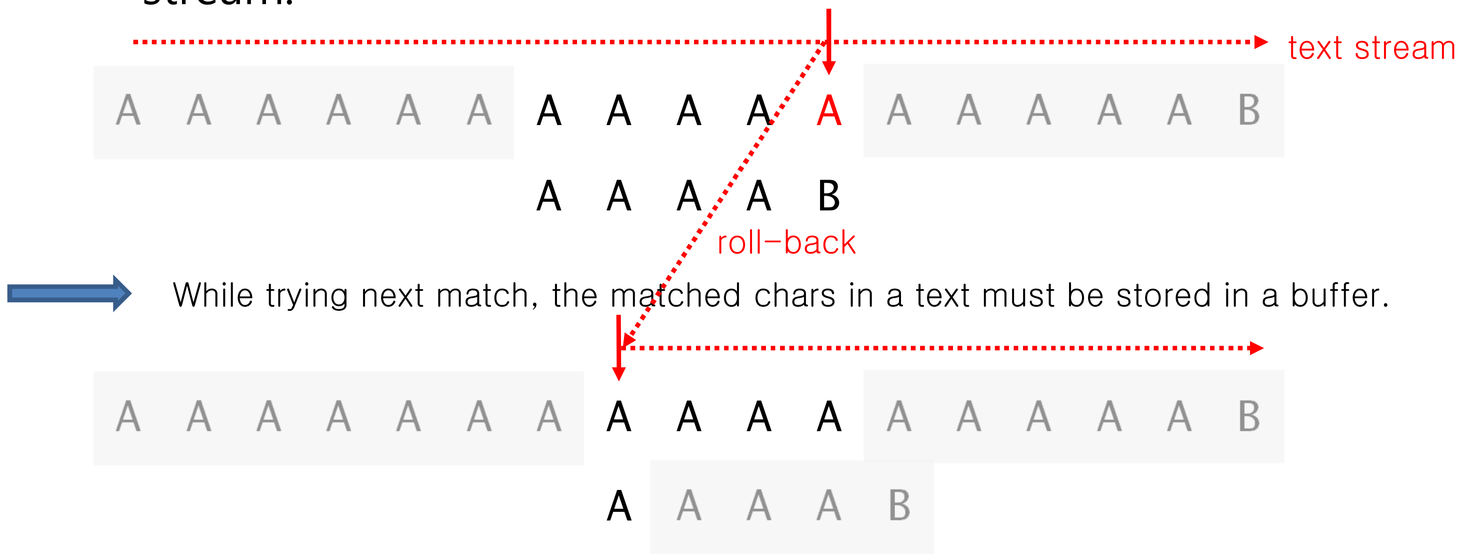
- Naïve Algorithm 2
  – Naïve algorithm can be slow if text and pattern are repetitive



– Still order: O(MN)
– Any improvement?

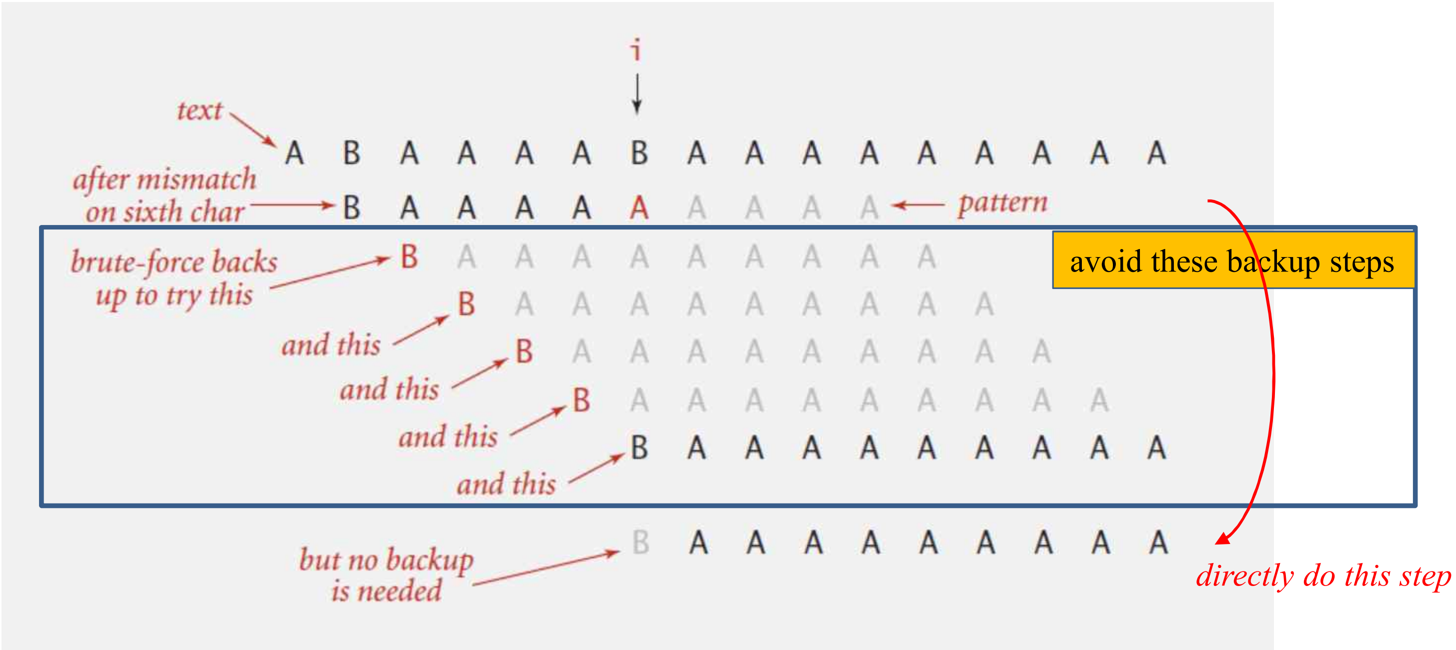
Brute-Force Substring Search

- 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.



Knuth-Morris-Pratt(KMP) Algorithm

- KMP algorithm
  - Clever method to always avoid backup problem.

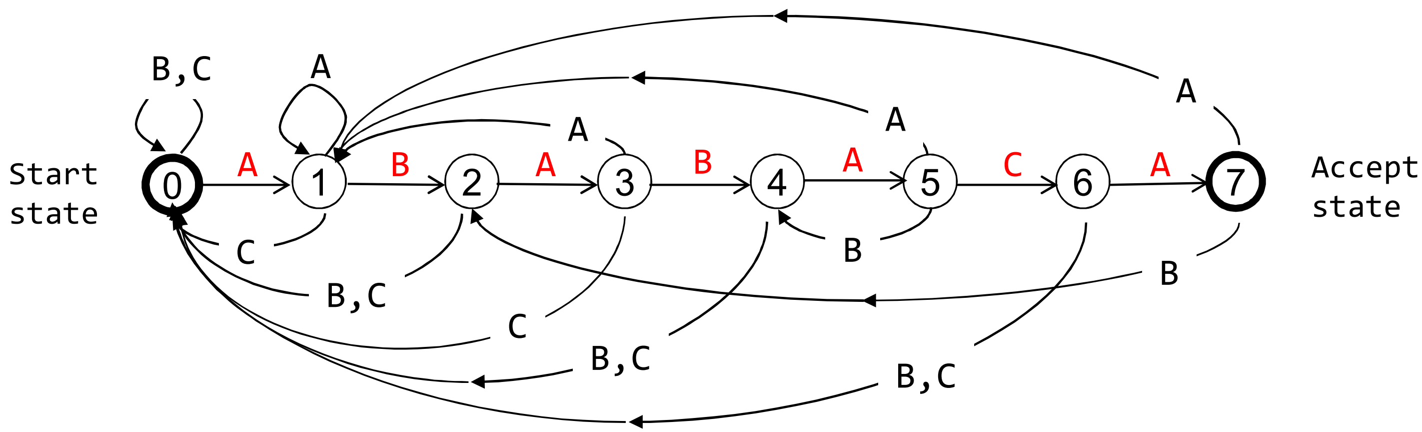


참고자료: Algorithms, R. Sedgewick

Deterministic Finite Automaton

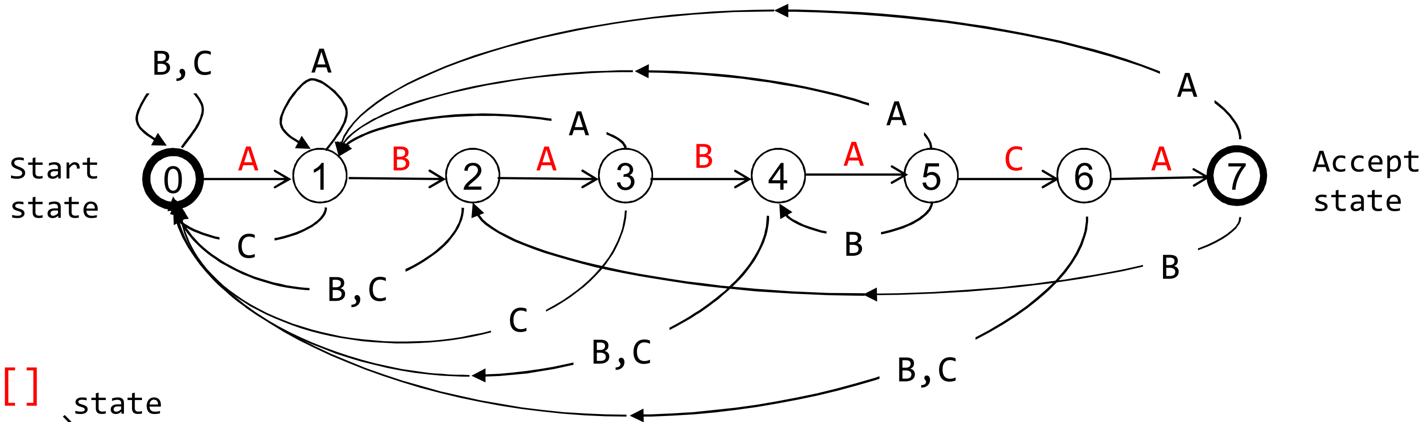
- DFA(Deterministic Finite State Automaton)
  - Finite number of states (including start and accept states)
  - Exactly one transition for each char
  - Accept if sequence of transitions leads to accept state

DFA for pattern ABABACA



DFA

DFA for pattern ABABACA



DFA[][]

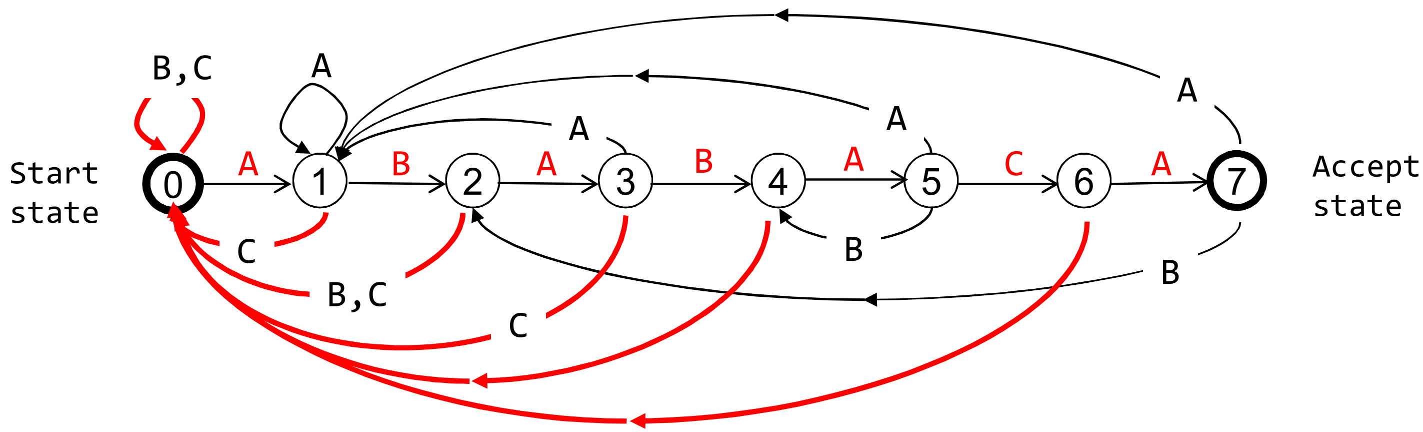
char \ state	0	1	2	3	4	5	6	7
A	1	1	3	1	5	1	7	1
B	0	2	0	4	0	4	0	2
C	0	0	0	0	0	6	0	0
others	0	0	0	0	0	0	0	0

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

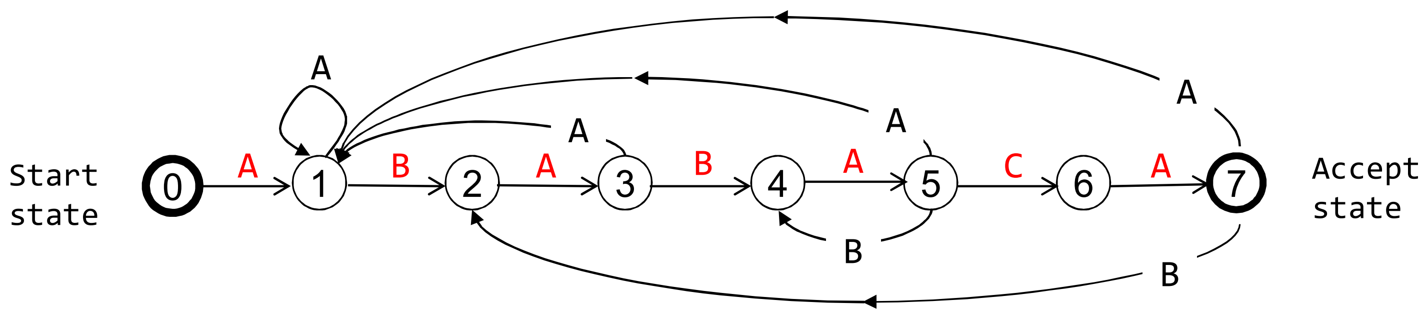
```
Example:
    text:  ABCABAABABABACACA
    state:012012312345456 7
```

DFA

DFA for pattern ABABACA



Simplified Diagram: remove transitions to state 0



Algorithm with DFA

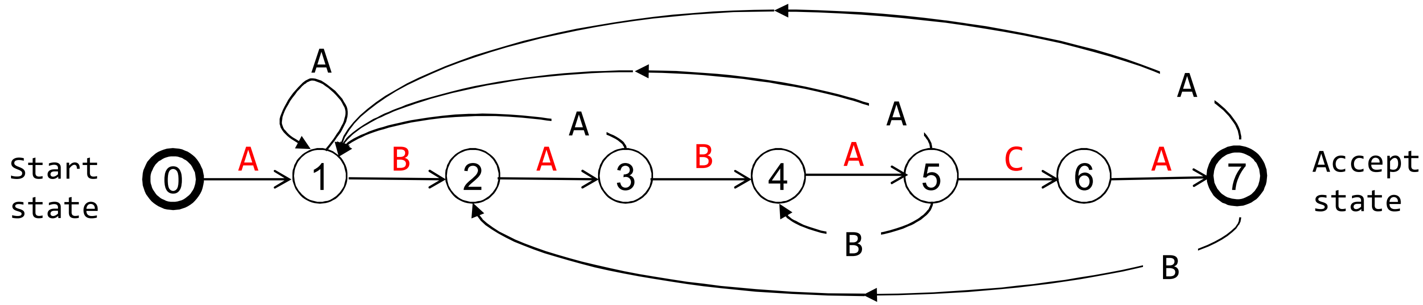
- Difference from naïve algorithm
  - Precomputation of DFA[][] from pattern
  - Text pointer `i` never decrements (no backup)

```
// patLength = strlen(pattern);
int matchingDFA(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



Start state

Accept state

state → 0 1 2 0 1 2 3 1 2 3 4 5 4 5 6 7

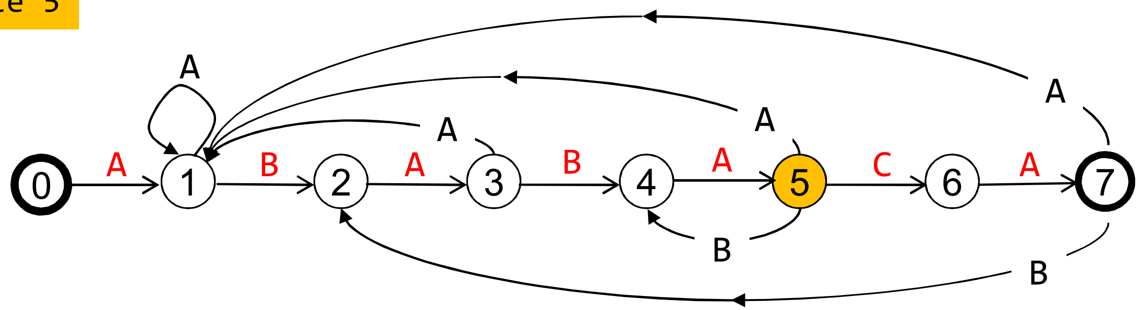
text → A B C A B A A B A B A B A C A C A

return 8

Interpretation of DFA

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

State 5



text → B C B A A B A B A A B A

Suffix of text[8]

pattern → A B A B A C A

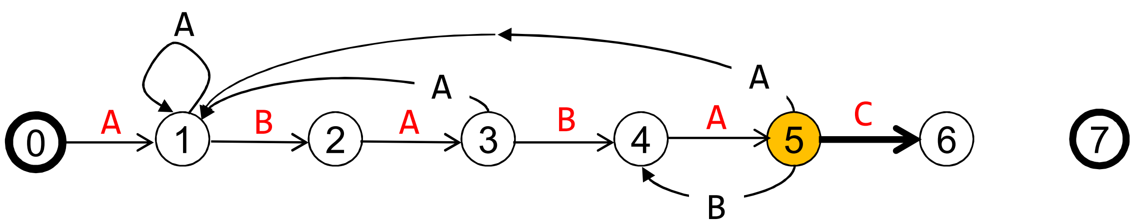
Prefix of pattern[]

DFA Construction

- 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

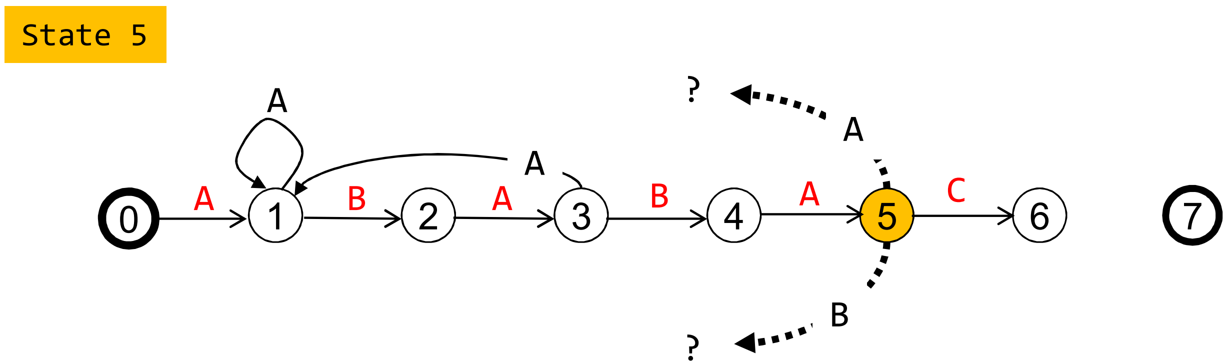
State 5



DFA Construction

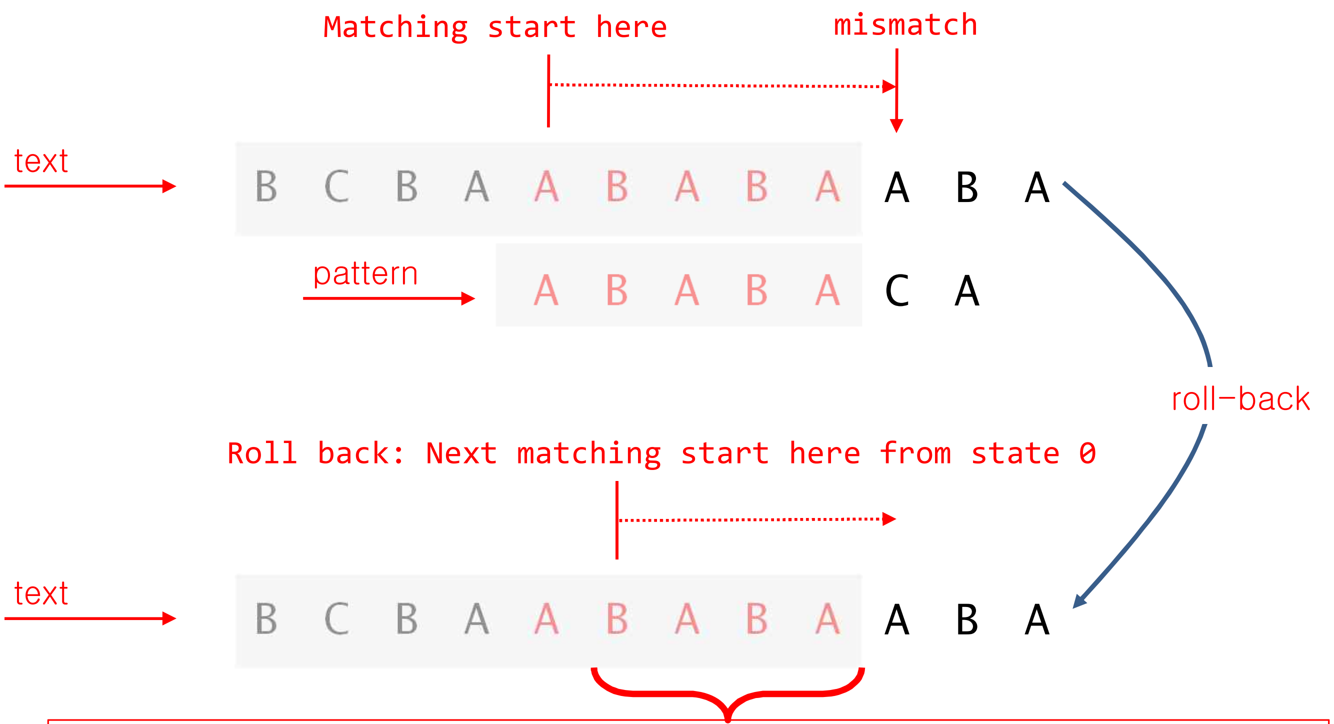
- DFA Construction:
  - Mismatch transition:
    - If in state `j` and next char `c != pattern[j]`, then which state to transit?

Pattern: ABABACA



Algorithmic Engineering Laboratory

DFA Construction



- *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

- 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]`

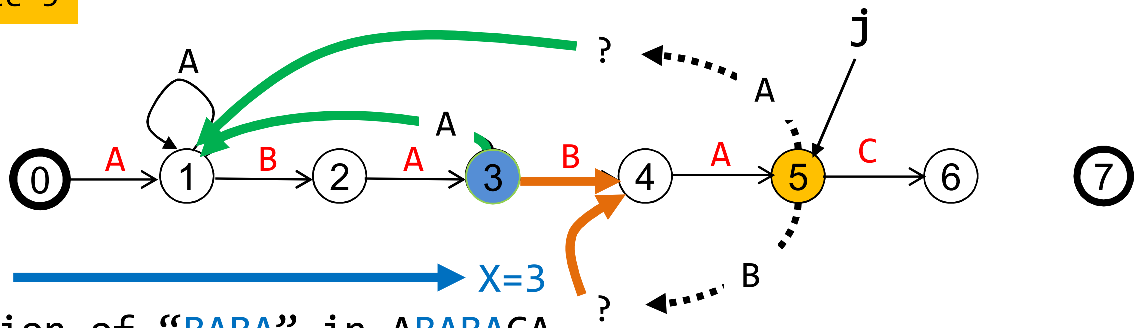
DFA Construction

- 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

State 5



Simulation of “BABA” in ABABACA

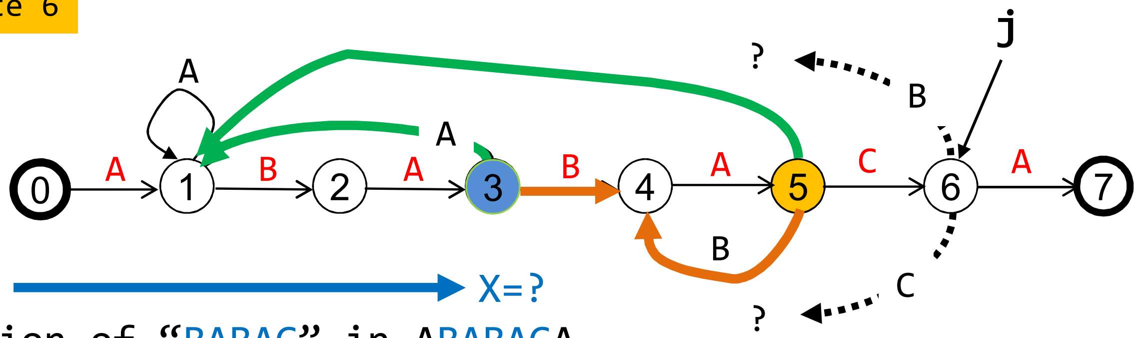
DFA Construction

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

DFA['B'][6] = DFA['B'][?] = ?
DFA['C'][6] = DFA['C'][?] = ?

Pattern: ABABACA

State 6



Simulation of "BABAC" in ABABACA

DFA Construction

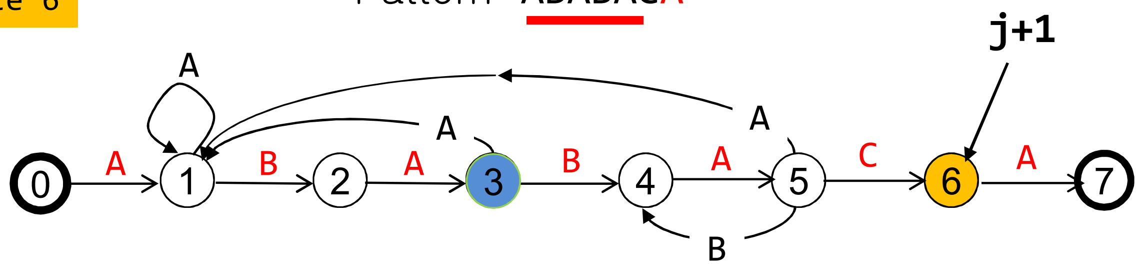
- DFA Construction:
  - Mismatch transition:
    - If in state `j` and next char `c != pattern[j]`, then the last `j-1` characters of input text are `pattern[1]` ~ `pattern[j-1]`, followed by `c`.
  - To compute `DFA[c][j]`:
    - Simulate `pattern[1]` ~ `pattern[j-1]` on DFA (*still under construction*) and let the current state `X`.
    - take a transition `c` from state `X`.
    - Running time : require `j` steps.
    - But, if we maintain state `X`, it takes only constant time!

DFA Construction

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

State 6

Pattern: ABABACA



X=3

Simulation of “BABA” in ABABACA

X’ = ?

Simulation of “BABAC” in ABABACA

- Simulation requires j+1 steps
- But, X’ = DFA[’C’][X]

X’ = DFA[’C’][3] = 0

DFA Construction

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

void constructDFA(char pattern[])
{

    int patLength = strlen(pattern);

    DFA[pattern[0]][0] = 1;  // transition from state 0

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

KMU 국민대학교 KOOKMIN UNIVERSITY

Algorithmic Engineering Laboratory

DFA Construction

- DFA Construction: Example

DFA[][]

0123456
Pattern: ABABACA

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

DFA Construction

- DFA Construction: Example

DFA[][]

j start from 1

0123456
Pattern: ABABACA

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

X **0** **0** **1** **2** **3** **0** **1** ~~**1**~~

DFA[B][0] DFA[A][0] DFA[B][1] DFA[A][2] DFA[C][3] DFA[A][0] ~~DFA[A][1]~~

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.

Algorithm with DFA

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

Prefix/Suffix

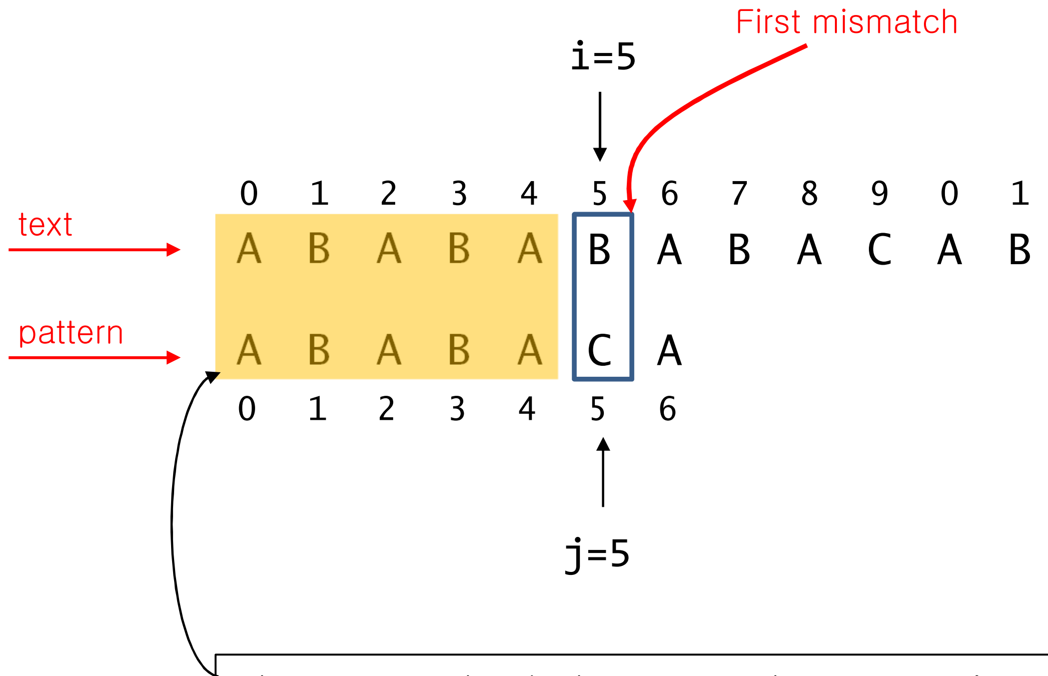
- Prefix / Suffix of a Text

bananada

Prefix	Suffix
NULL string →	
b	a
ba	da
ban	ada
bana	nada
banan	anada
banana	nanada
bananad	ananada
bananada	bananada

KMP Algorithm

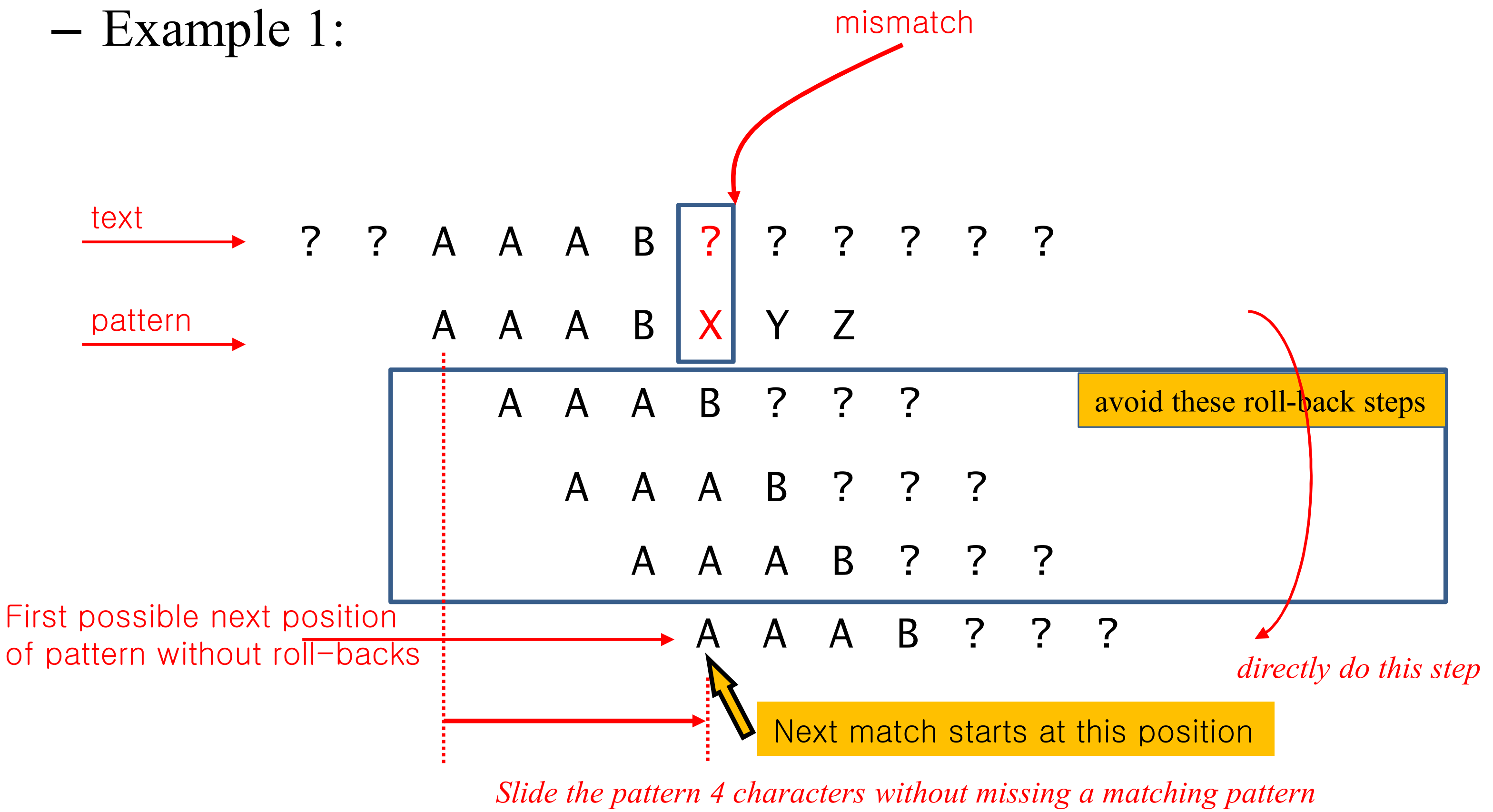
- Naïve algorithm again:



These matched characters have very important information.
Do not waste this information!!!

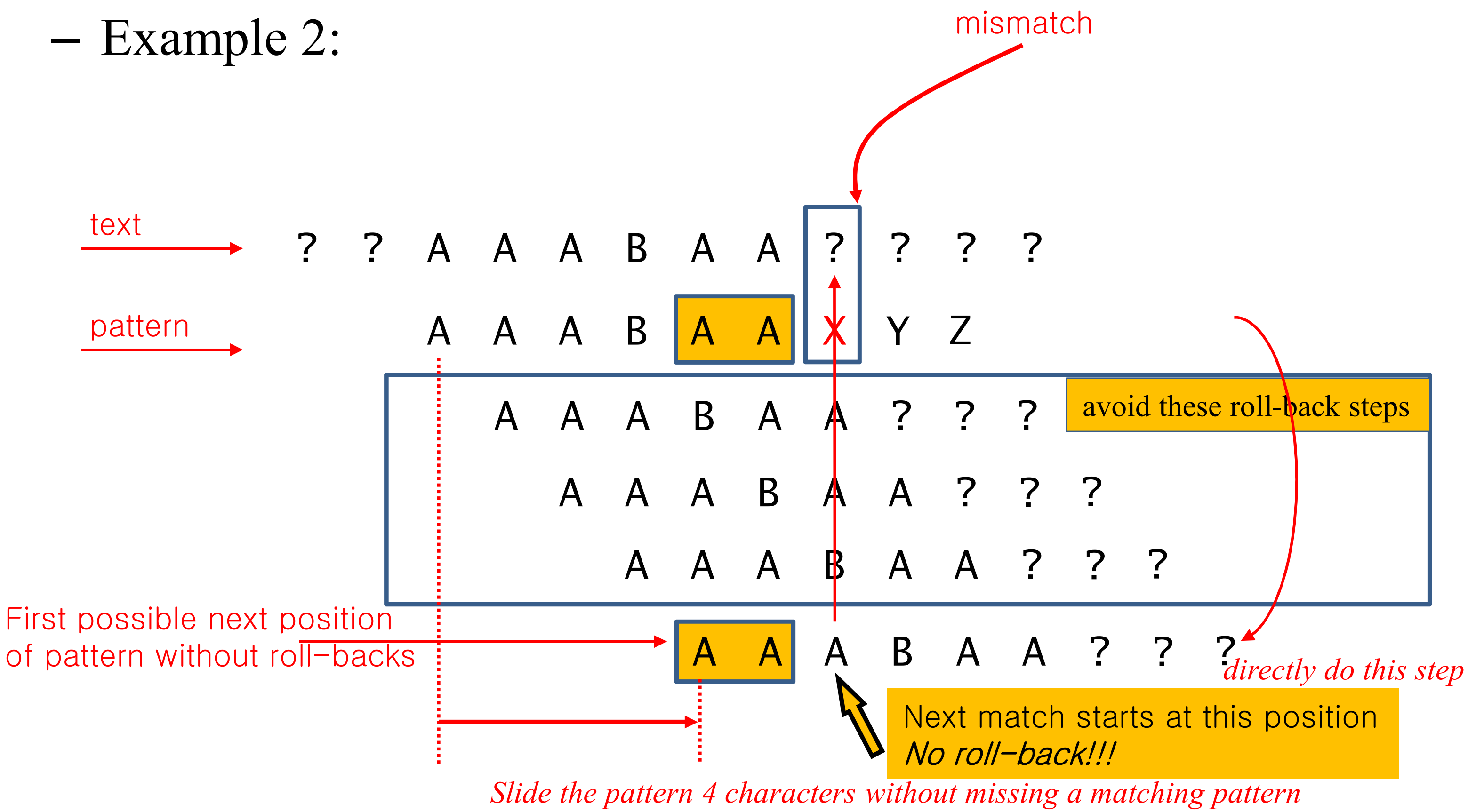
KMP Algorithm

- Avoid roll-backs in naïve algorithm:
  - Example 1:



KMP Algorithm

- Avoid roll-backs in naïve algorithm:
  - Example 2:



KMP Algorithm

- Prefix and Proper Suffix of the Prefix

pattern → A A A B A A X Y Z

$\underbrace{\text{A A A B A A}}_{\text{Prefix}}$ X Y Z

Proper Suffix of the Prefix "AAABAA"

A A A B A A X Y Z

A A A B A A X Y Z

A A A B A A X Y Z

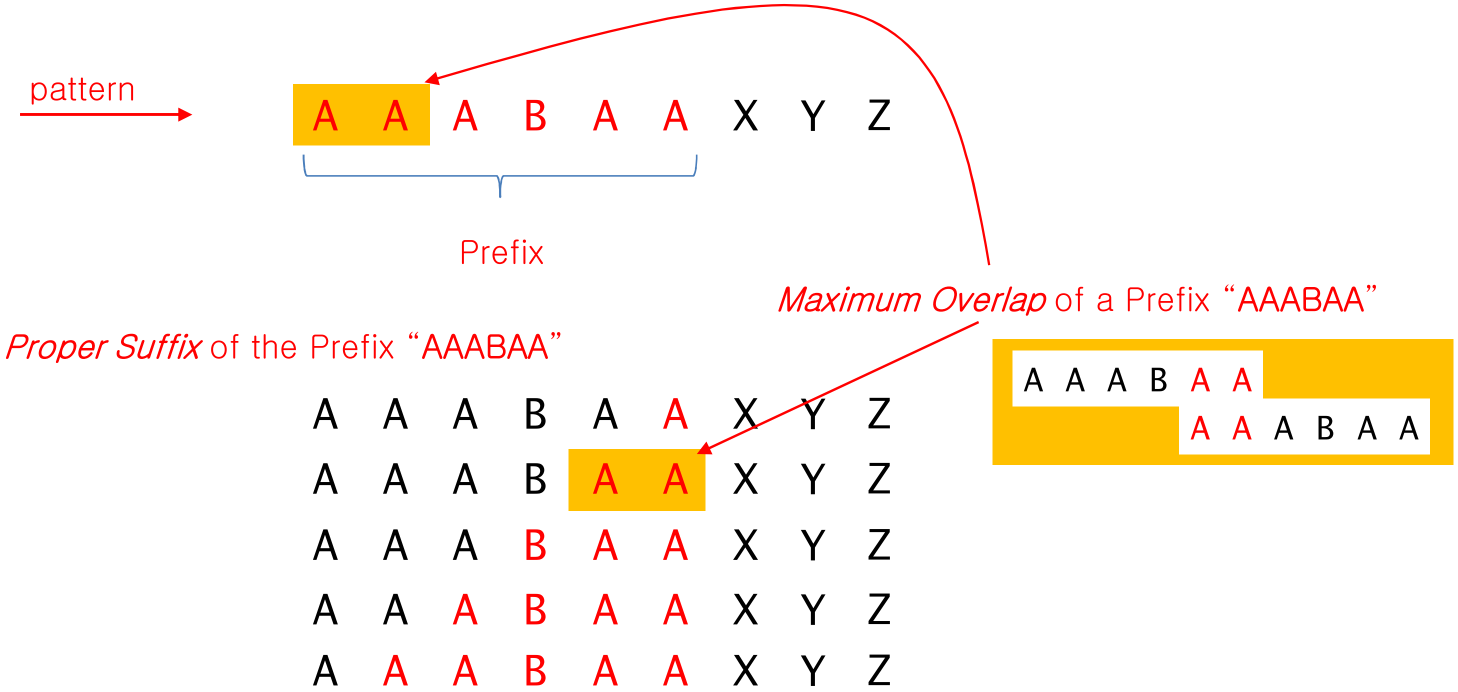
A A A B A A X Y Z

A A A B A A X Y Z

A A A B A A X Y Z ← Not a *proper* suffix of the prefix (the same as the prefix)

KMP Algorithm

- Maximum Overlap of a Prefix
  - the longest proper suffix that is equal to prefix of the prefix



KMP Algorithm

- Maximum Overlap of a Prefix
  - the longest proper suffix that is equal to prefix of the prefix

- Example:

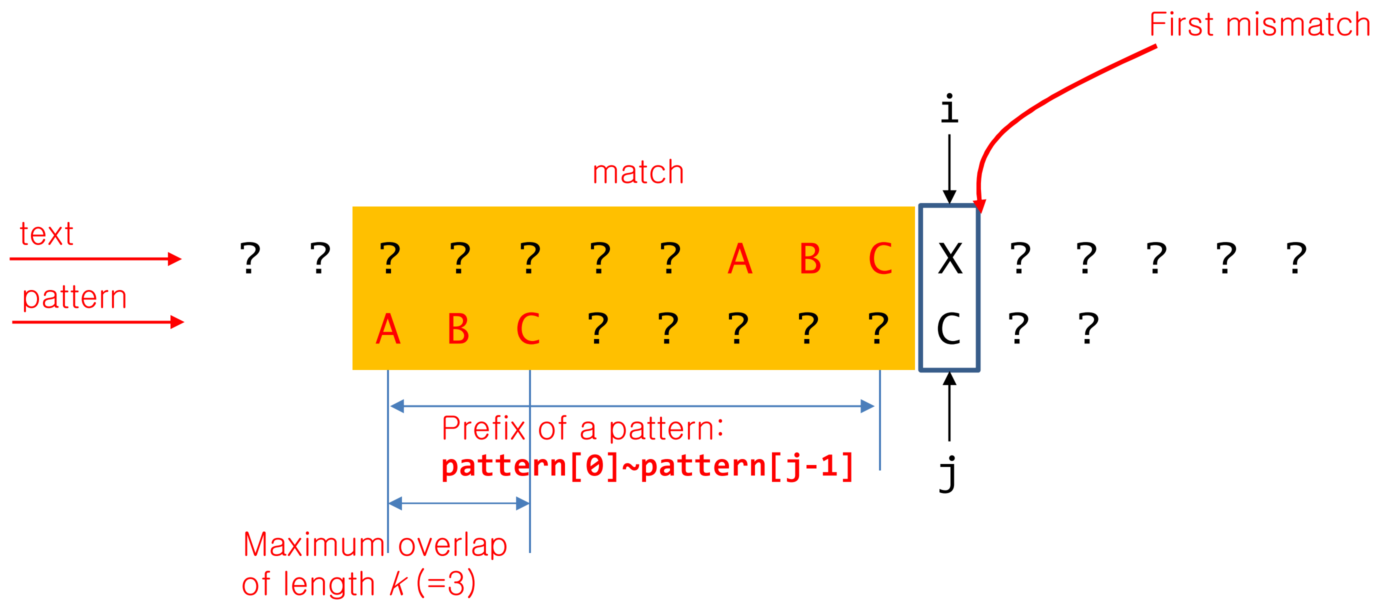
Prefix	Maximum Overlap
AAAAA	AAAA
AABA	A
AAAB	
ABABABAB	ABABAB

not **AAAAA**

NULL String

KMP Algorithm

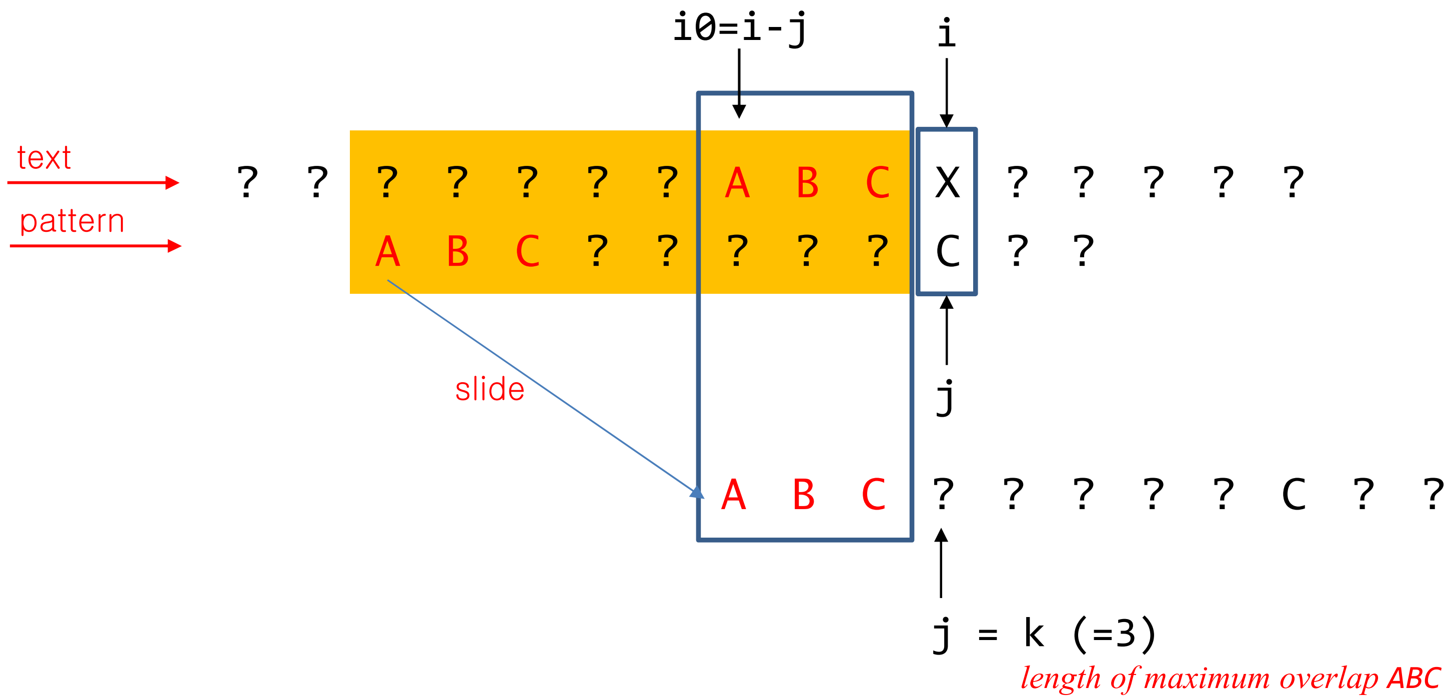
- Reuse of prefix information when there is a mismatch
  - Mismatch at `text[i]` and `pattern[j]`



Note that if the mismatched *location* is `pattern[j]`, then *prefix* is: `pattern[0]~pattern[j-1]`

KMP Algorithm

– Then we can slide the pattern so that the *suffix and prefix aligns without missing out on a match*:



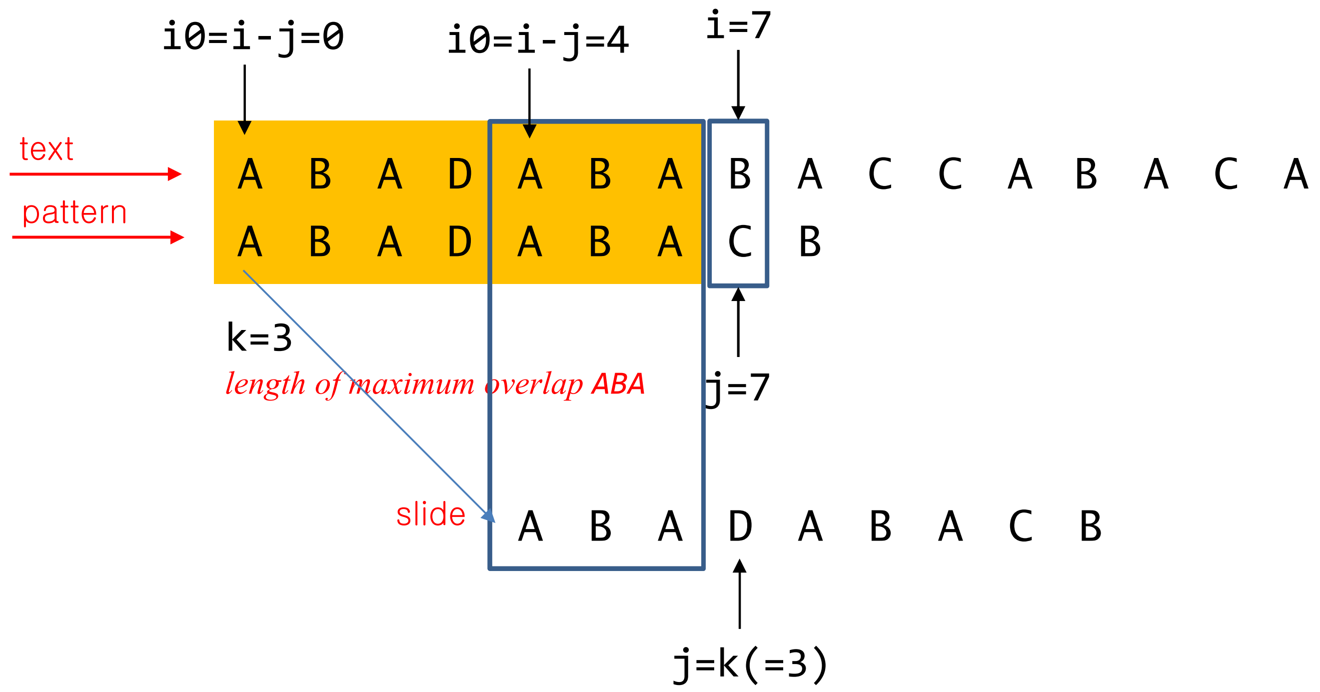
KMP Algorithm

- Fast sliding algorithm:
  - Psuedo program:

```
// mismatch found at text[i], pattern[j]

prefix = pattern[0] ~ pattern[j-1];

k = Length of maximum overlap of prefix;

j = k;
// i is unchanged !

// Matched position i0 in text starts from (i - j);
i0 = i – j;
```

KMP Algorithm

- Fast sliding algorithm:
  - Example:



KMP Algorithm

- Failure function:
  - M : the length of a pattern
  - For $0 < k < M$, the failure function $fail(k)$ is the length of maximum overlap of a prefix `pattern[0]` ~ `pattern[k]`
    - Note that $fail(0) = 0$

banabana

k	prefix	$fail(k)$
0	b	0
1	ba	0
2	ban	0
3	bana	0
4	banab	1
5	banaba	2
6	banaban	3
7	banabana	4

KMP Algorithm

- Knuth-Morris-Pratt(KMP) Algorithm

```
vector<int> kmp(string text, string pattern)
{
    vector<int> ans;
    fail = getFail(pattern);        // failure function
    int n = (int) text.size(), m = (int) pattern.size();
    int j =0;                       // j : index of pattern

    for(int i = 0 ; i < n ; i++)    // i : index of text
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)              // pattern matching is found
            {
                ans.push_back(i-j); // save the matched position
                j = fail[j];
            }
            else
                j++;
    }
    return ans;
}
```

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)     // i : index of text
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)               // pattern matching is found
            {
                ans.push_back(i-j); // save the matched position
                j = fail[j];
            }
            else
                j++;
    }
```

i=0

text → A B A B A B A C A C A B A C A B

pattern → A B A C A B

j=0

text[i] == pattern[j]

➔ i++, j++

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)     // i : index of text
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)                // pattern matching is found
            {
                ans.push_back(i-j); // save the matched position
                j = fail[j];
            }
            else
                j++;
    }
```

i=1

text → A $\boxed{B}$ A B A B A C A C A B A C A B

pattern → A $\boxed{B}$ A C A B

j=1

text[i] == pattern[j]

➔ i++, j++

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)    // i : index of text
   {
       while(j>0 && text[i] != pattern[j])
           j = fail[j-1];
       if(text[i] == pattern[j])
           if(j==m-1)             // pattern matching is found
           {
               ans.push_back(i-j); // save the matched position
               j = fail[j];
           }
           else
               j++;
   }
```

i=2

text → A B A B A B A C A C A B A C A B

pattern → A B A C A B

j=2

text[i] == pattern[j]

➔ i++, j++

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)    // i : index of text
   {
       while(j>0 && text[i] != pattern[j])
           j = fail[j-1];
       if(text[i] == pattern[j])
           if(j==m-1)              // pattern matching is found
           {
               ans.push_back(i-j); // save the matched position
               j = fail[j];
           }
           else
               j++;
   }
```

i=3

text → A B A B A B A C A C A B A C A B

pattern → A B A C A B

j=3

text[i] != pattern[j]

➔ j = fail[j-1]

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)    // i : index of text
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)              // pattern matching is found
            {
                ans.push_back(i-j); // save the matched position
                j = fail[j];
            }
            else
                j++;
    }
```

i=3

text → A B A B A B A C A C A B A C A B

pattern → A B A C A B

j=1

text[i] == pattern[j]

➔ i++, j++

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)    // i : index of text
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)              // pattern matching is found
            {
                ans.push_back(i-j); // save the matched position
                j = fail[j];
            }
            else
                j++;
    }
```

i=4

text → A B A B A B A C A C A B A C A B

pattern → A B A C A B

j=2

text[i] == pattern[j]

➔ i++, j++

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)     // i : index of text
   {
       while(j>0 && text[i] != pattern[j])
           j = fail[j-1];
       if(text[i] == pattern[j])
           if(j==m-1)              // pattern matching is found
           {
               ans.push_back(i-j); // save the matched position
               j = fail[j];
           }
           else
               j++;
   }
```

i=5

text → A B A B A $\boxed{B}$ A C A C A B A C A B

pattern → A B A $\boxed{C}$ A B

j=3

text[i] != pattern[j]

➔ j = fail[j-1]

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)     // i : index of text
   {
       while(j>0 && text[i] != pattern[j])
           j = fail[j-1];
       if(text[i] == pattern[j])
           if(j==m-1)               // pattern matching is found
           {
               ans.push_back(i-j); // save the matched position
               j = fail[j];
           }
           else
               j++;
   }
```

i=5

text: A B A B A B A C A C A B A C A B

pattern: A B A C A B

j=1

text[i] == pattern[j]

➔ i++, j++

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMU 국민대학교 KOOKMIN UNIVERSITY

Algorithmic Engineering Laboratory

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)
            {
                ans.push_back(i-j);
                j = fail[j];
            }
            else
                j++;
    }
```

i=9

text → A B A B A B A C A C A B A C A B

pattern → A B A C A B

j=5

text[i] != pattern[j]

➔ j = fail[j-1]

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)
            {
                ans.push_back(i-j);
                j = fail[j];
            }
            else
                j++;
    }
```

i=9

text → A B A B A B A C A $\boxed{\text{C}}$ A B A C A B

pattern → A $\boxed{\text{B}}$ A C A B

j=1

text[i] != pattern[j]

➔ j = fail[j-1]

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)
   {
       while(j>0 && text[i] != pattern[j])
           j = fail[j-1];
       if(text[i] == pattern[j])
           if(j==m-1)
           {
               ans.push_back(i-j);
               j = fail[j];
           }
           else
               j++;
   }
```

i=9

text → A B A B A B A C A C A B A C A B

pattern → A B A C A B

j=0

j==0 && text[i] != pattern[j]

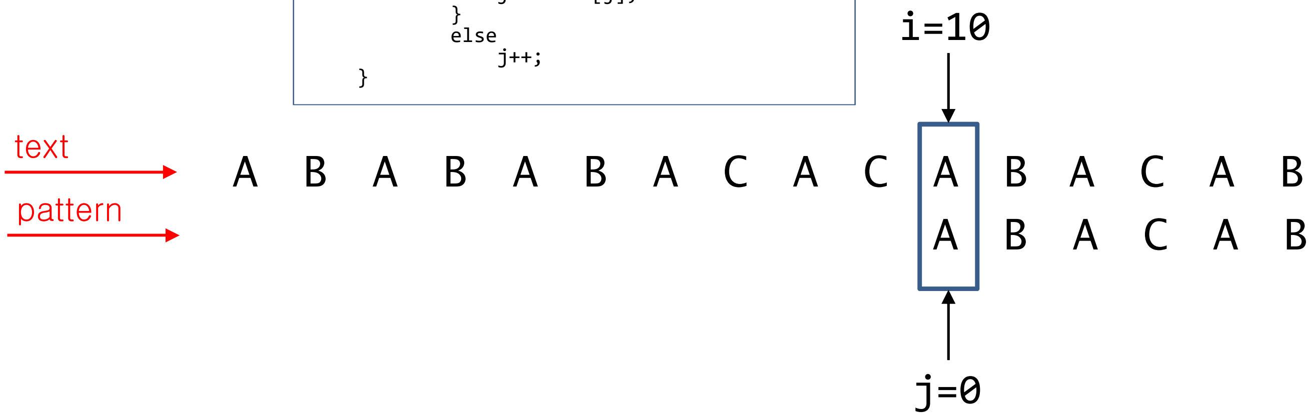
➔ i++

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)
            {
                ans.push_back(i-j);
                j = fail[j];
            }
            else
                j++;
    }
```



text[i] == pattern[j]

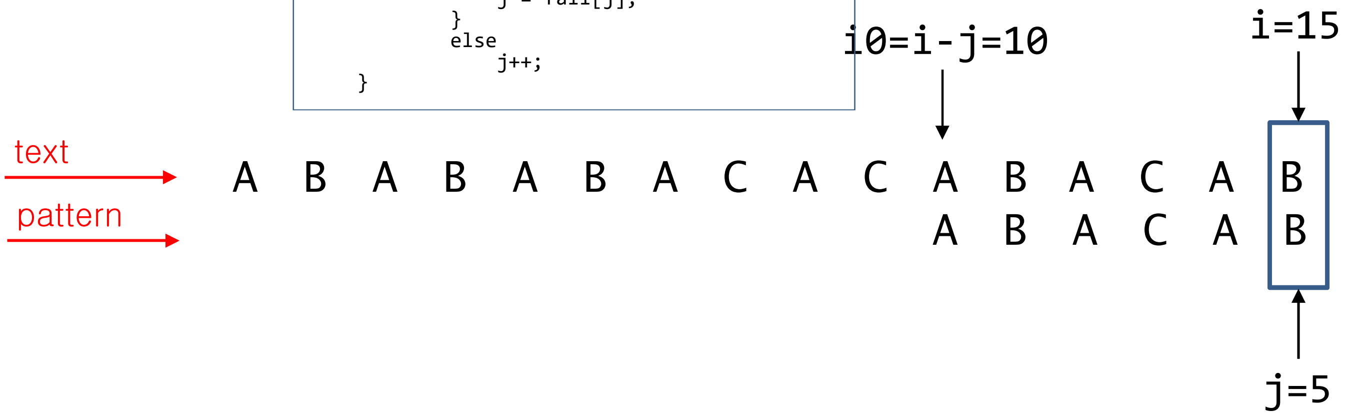
➔ i++, j++

i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

– Example:

```
for(int i = 0 ; i < n ; i++)
    {
        while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
        if(text[i] == pattern[j])
            if(j==m-1)
            {
                ans.push_back(i-j);
                j = fail[j];
            }
            else
                j++;
    }
```



i	0	1	2	3	4	5
fail(i)	0	0	1	0	1	2

KMP Algorithm

- Knuth-Morris-Pratt(KMP) Algorithm
  - Why?

```
while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
```

i=0

text → A B A B A B A C A B A B A B A B

pattern → A B A B A B A B

j=0

`text[i] == pattern[j]`

➔ `i++, j++`

i	0	1	2	3	4	5	6	7
fail(i)	0	0	1	2	3	4	5	6

KMP Algorithm

- Knuth-Morris-Pratt(KMP) Algorithm
  - Why?

```
while(j>0 && text[i] != pattern[j])
        j = fail[j-1];
```

i=7

text → A B A B A B A C A B A B A B A B

pattern → A B A B A B A B

j=7

`text[i] != pattern[j]`

➔ `j = fail[j-1]`

i	0	1	2	3	4	5	6	7
fail(i)	0	0	1	2	3	4	5	6

KMP Algorithm

- Knuth-Morris-Pratt(KMP) Algorithm
  - Why?

```
while(j>0 && text[i] != pattern[j])
        j = fail[j-1];
```

i=7

text → A B A B A B A C A B A B A B A B

pattern → A B A B A B A B

j=5

text[i] != pattern[j]

➔ j = fail[j-1]

i	0	1	2	3	4	5	6	7
fail(i)	0	0	1	2	3	4	5	6

KMP Algorithm

- Knuth-Morris-Pratt(KMP) Algorithm
  - Why?

```
while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
```

i=7

text → A B A B A B A C A B A B A B A B

pattern → A B A B A B A B

j=3

text[i] != pattern[j]

➔ j = fail[j-1]

i	0	1	2	3	4	5	6	7
fail(i)	0	0	1	2	3	4	5	6

KMP Algorithm

- Knuth-Morris-Pratt(KMP) Algorithm
  - Why?

```
while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
```

i=7

text: A B A B A B A C A B A B A B A B

pattern: A B A B A B A B

j=1

text[i] != pattern[j]

➔ j = fail[j-1]

i	0	1	2	3	4	5	6	7
fail(i)	0	0	1	2	3	4	5	6

KMP Algorithm

- Knuth-Morris-Pratt(KMP) Algorithm
  - Why?

```
while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
```

i=7

text → A B A B A B A C A B A B A B A B

pattern → A B A B A B A B

j=0

text[i] != pattern[j]

➔ j = fail[j-1]

i	0	1	2	3	4	5	6	7
fail(i)	0	0	1	2	3	4	5	6

KMP Algorithm

- Knuth-Morris-Pratt(KMP) Algorithm
  - Why?

```
while(j>0 && text[i] != pattern[j])
            j = fail[j-1];
```

i=8

text →	A	B	A	B	A	B	A	C	A	B	A	B	A	B	A	B
pattern →									A	B	A	B	A	B	A	B

j=0

j==0 && text[i] != pattern[j]

➔ i++

i	0	1	2	3	4	5	6	7
fail(i)	0	0	1	2	3	4	5	6

KMP Algorithm

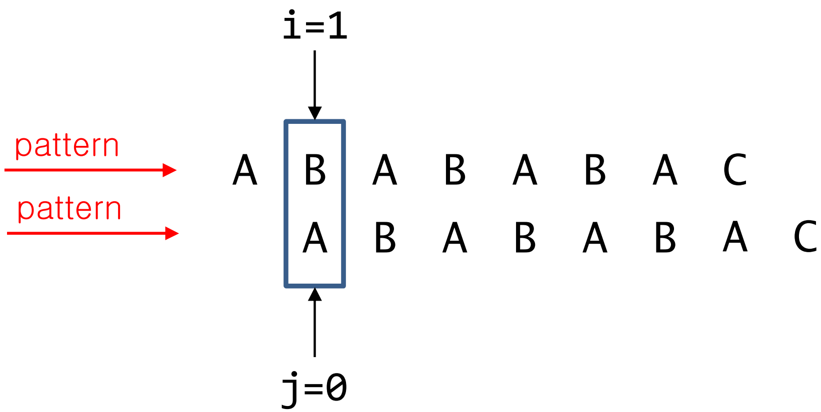
- `getFail()` function
  - Simple `fail[]` computation needs $O(M^3)$ time.
  - $O(M)$ time algorithm: very similar to KMP algorithm itself

```
vector<int> getFail(string pattern)
{
    int m = (int) pattern.size();
    int j=0;
    vector<int> fail(m, 0);

    for(int i = 1; i< m ; i++)
    {
        while(j > 0 && pattern[i] != pattern[j])
            j = fail[j-1];
        if(pattern[i] == pattern[j])
            fail[i] = ++j;
    }
    return fail;
}
```

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0						




```
pattern[i] != pattern[j]
```

```
➔ fail[i] = 0 // initial value
  i++
```

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1					

i=2

```
pattern →  A  B  A  B  A  B  A  C
pattern →        A  B  A  B  A  B  A  C
```

j=0

`pattern[i] == pattern[j]`

➔ `fail[i] = ++j`
`i++`

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2				

```
                    i=3
pattern →  A B A [B] A B A C
pattern →        A [B] A B A B A C
                    j=1
```

`pattern[i] == pattern[j]`

➔ `fail[i] = ++j`
`i++`

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2	3			

i=4

pattern → A B A B A B A C

pattern → A B A B A B A C

j=2

`pattern[i] == pattern[j]`

➔ `fail[i] = ++j`
`i++`

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2	3	4		

```
                          i=5
                           ↓
pattern →   A  B  A  B  A  B  A  C
pattern →         A  B  A  B  A  B  A  C
                           ↑
                          j=3
```

```
pattern[i] == pattern[j]

➔ fail[i] = ++j
  i++
```

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2	3	4	**5**	

```
                              i=6

pattern →   A  B  A  B  A  B  A  C
pattern →         A  B  A  B  A  B  A  C

                              j=4
```

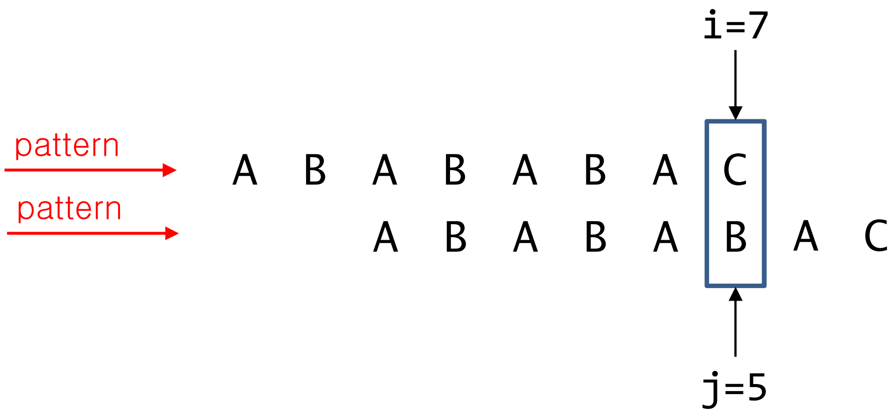
`pattern[i] == pattern[j]`

→ `fail[i] = ++j`
`i++`

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2	3	4	5	




```
pattern[i] != pattern[j]
```

➔ `j = fail[j-1] (=3)`

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2	3	4	5	

```
                        i=7
                         ↓
pattern →  A  B  A  B  A  B  A [C]
pattern →              A  B  A [B] A  B  A  C
                         ↑
                        j=3
```

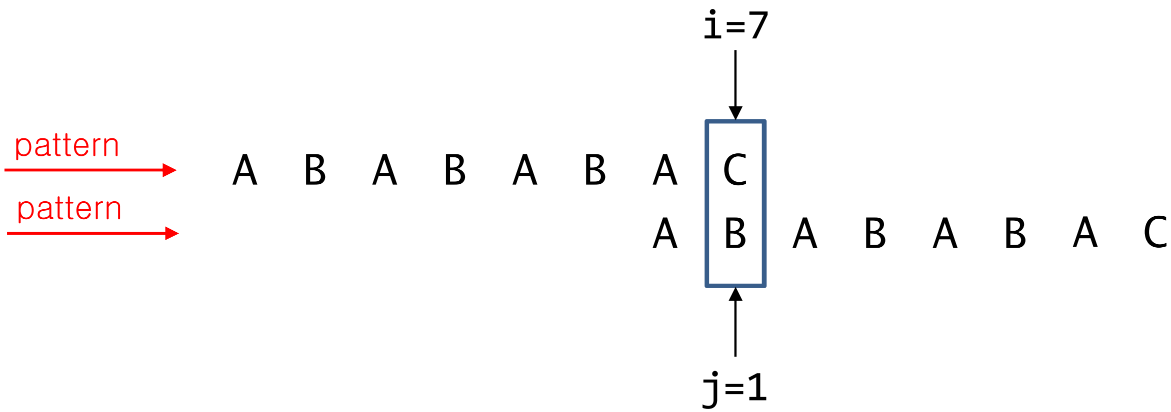
`pattern[i] != pattern[j]`

➔ `j = fail[j-1] (=1)`

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2	3	4	5	



`pattern[i] != pattern[j]`

➔ `j = fail[j-1] (=1)`

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2	3	4	5	**0**

i=7

pattern → A B A B A B A C

pattern → A B A B A B A C

j=0

`pattern[i] != pattern[j]`

➔ `j = fail[j-1] (=0)`

KMP Algorithm

- `getFail()` function
  - Example:

i	0	1	2	3	4	5	6	7
fail[i]	0	0	1	2	3	4	5	0

i=8

pattern → A B A B A B A C

pattern → $\boxed{\text{A}}$ B A B A B A C

j=0

i++

KMP Algorithm

- `getFail()` function
  - Example: `aabaabac`

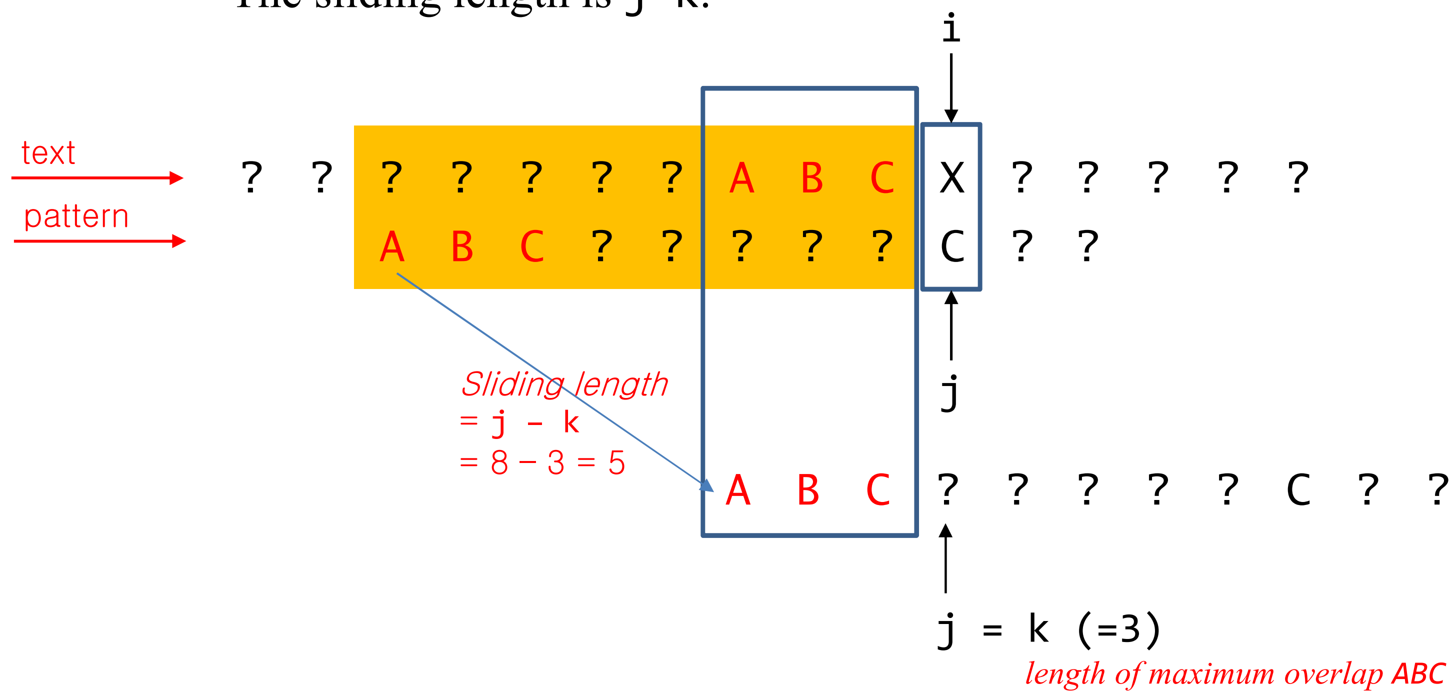
i	0	1	2	3	4	5	6	7
fail[i]	0	1	0	1	2	3	4	0

KMP Algorithm

- `getFail()` function
  - Why $O(M)$?
    - Index `i` increases from 1 to M-1
    - Index `j` *increases* maximally as many as `i` increases
    - Also index `j` *decreases* maximally as many as `j` increases
- KMP algorithm
  - Why $O(N)$ algorithm?
    - Similar logic to get the time complexity of `getFail()` function

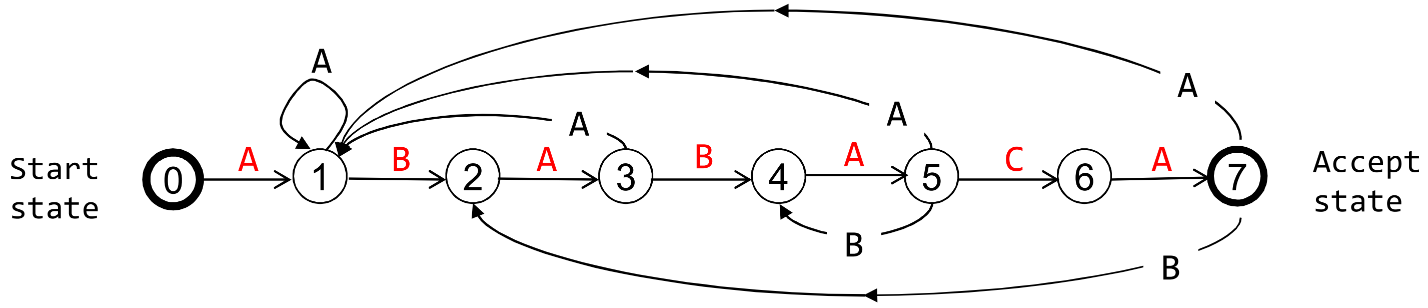
Review

- The meaning of `fail[j-1](=k)`:
  - The length of a maximum Overlap of a Prefix:
    - the longest proper suffix that is equal to prefix of the prefix
    - The sliding length is `j-k`.



학번:________________ 성명:_______________ (Page 1)

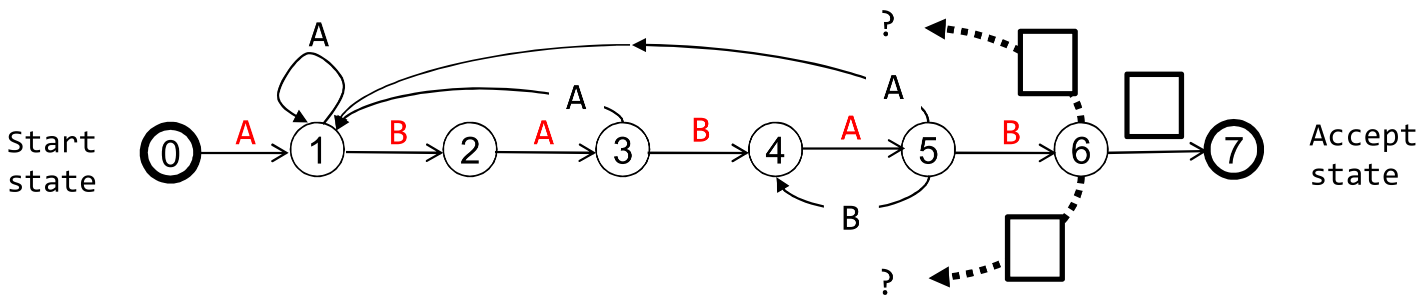
1. 아래와 같은 DFA가 주어지고, 또한 text 가 주어졌을 때, 입력되는 각 text의 문자에 대하여 DFA의 state의 transition이 일어나는데, text의 각 문자를 입력한 후의 DFA state 번호를 나열하시오.



	Start state																	
state →	0																	
text →		A	A	C	B	A	A	B	A	C	A	B	A	B	A	C	A	B

학번:________________ 성명:_______________ (Page 2)

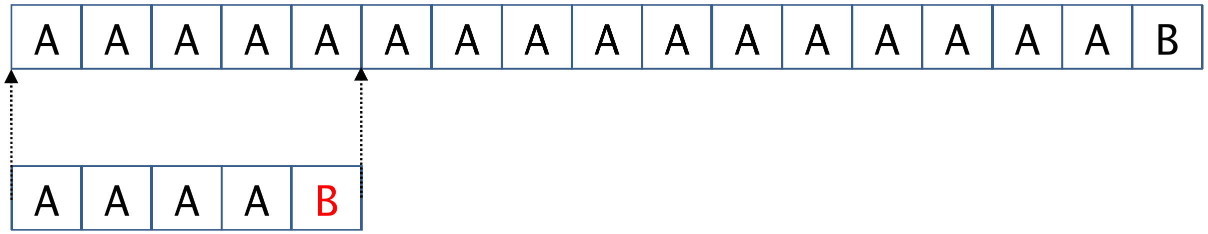
2. Pattern "ABABABC" 에 맞는 DFA를 만들고자 한다. 이 Pattern의 DFA를 0번 state 부터 5번 state 까지의 state transition을 모두 계산한 다음에 6번 state에서 문자 'A', 'B', 'C' 에 대한 state transition을 계산하시오. 즉 6번 state에서 각각 문자 'A', 'B', 'C' 가 입력되었을 때, 몇 번 state로 옮겨가는지를 계산하시오. (네모 빈칸에 각 문자를 넣고, 화살표가 몇 번 state를 향하는지를 표시하시오.




```
DFA['A'][6] = ?
DFA['B'][6] = ?
DFA['C'][6] = ?
```

학번:________________ 성명:_______________ (Page 3)

3. 다음과 같은 Pattern “AAAAB” 에 맞는 DFA를 만들고, 이 DFA 에서 text “AAAAAAAAAAAAAAAAB” 에서 한 문자씩이 입력될 때, DFA에서 변화되는 state를 계산하시오.



학번:________________ 성명:_______________ (Page 4)

4. 다음과 같은 Pattern “ABABABABC” 에 맞는 DFA를 만들고, 이 DFA 에서 text “ABABABABABABABABC” 에서 한 문자씩이 입력될 때, DFA에서 변화되는 state를 계산하시오.

A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	C

A	B	A	B	A	B	A	B	C

(4.1) DFA 알고리즘에서 state ‘X’ 는 어떤 state를 의미하는 지를 설명하시오.