알고스 2주차 알고리즘 스터디

-manacher 알고리즘, kmp 알고리즘-

2016294 이수민

CONTENTS

01 팰린드롬 알고리즘

팰린드롬: 양쪽이 대칭이 되는 문자열 ex) a, aa, abcba

1. Naïve한 방법 - 양쪽 값에서 가운데로 이동하면서, 대칭이 된다면 팰린드롬 수 인 것을 판별.

```
bool is_palindrome(string str) {
    bool flag = true;
    for (int i = 0; i < str.size(); i++) { // 처음부터 끝까지 검사
        if (str[i] != str[str.size()-1-i]) { // 문자열이 다를 경우
            flag = false; // 팰린드롬이 아님
            break;
        }
    }
    return flag;
}
```

01 팰린드롬 알고리즘

2. Modified- 가운데 글자부터 오른쪽에 있는 글자는 이미 확인한 수.

```
bool is_palindrome(string str) {
    bool flag = true;
    for (int i = 0; i < str.size() / 2; i++) { // 처음부터 가운데 전까지 검사
        if (str[i] != str[str.size()-1-i]) { // 문자열이 다를 경우
            flag = false; // 팰린드롬이 아님
            break;
        }
    }
    return flag;
}
```

01 팰린드롬 알고리즘

3. 팰린드롬 + dp배열 메모이제이션

- dp[s][e] : s~e구간 까지 팰린드롬 수인지 아닌지 1 또는 0으로 기록

```
int main(){
  for (int i = 1; i <= N; i++){
    dp[i][i] = 1; // 한자리수는 무조건 펠린드롬

    if (i != 1 && num[i - 1] == num[i]) // 두자리가 같은 경우 펠린드롬
      dp[i - 1][i] = 1;
  }

  for (int i = 2; i <= N - 1; i++){
    for (int j = 1; i + j <= N ; j++){
      if (num[j] == num[i + j] && dp[j + 1][i + j - 1] == 1)
        dp[j][i + j] = 1;
    }
  }
}
```

x+1에서 y-1의 구간이 팰린드롬이라면,
x번째 수와 y번째 수를 비교하는 것만으로도
x에서 y구간의 수가 팰린드롬인지 판별 가능.

O(n^2) 의 복잡도로,
문자열 내 부분 문자열 팰린드롬 확인 가능

3. 팰린드롬 + dp배열 메모이제이션

- dp[s][e] : s~e구간 까지 팰린드롬 수인지 아닌지 1 또는 0으로 기록

```
int main(){
  for (int i = 1; i <= N; i++){
    dp[i][i] = 1; // 한자리수는 무조건 팰린드롬

    if (i != 1 && num[i - 1] == num[i]) // 두자리가 같은 경우 팰린드롬
      dp[i - 1][i] = 1;
  }

  for (int i = 2; i <= N - 1; i++){
    for (int j = 1; i + j <= N ; j++){
      if (num[j] == num[i + j] && dp[j + 1][i + j - 1] == 1)
        dp[j][i + j] = 1;
    }
  }
}
```

더 빠르고 효율적인
O(N)의 시간 복잡도를 가지는
Manacher 알고리즘

x+1에서 y-1의 구간이 팰린드롬이라면,
x번째 수와 y번째 수를 비교하는 것만으로도
x에서 y구간의 수가 팰린드롬인지 판별 가능.

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 알고리즘이란?-

- ✓ 문자열 배열에서 **i번째 문자를 중심**으로 하는 **가장 긴 팰린드롬 길이를 반지름 r**을 저장하여 사용
- ✓ 시간복잡도 **O(N)**

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 알고리즘이란?-

- ✓ **문자열 배열에서 i번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이를 반지름 r을 저장하여 사용**
- ✓ **시간복잡도 O(N)**

문자열	b	c	n	a	n	a	c
배열	0						

배열 A[0] : 0번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 알고리즘이란?-

- ✓ 문자열 배열에서 i번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이를 반지름 r을 저장하여 사용
- ✓ 시간복잡도 O(N)

문자열	b	a	n	a	n	a	c
배열	0	0					

배열 A[1] : 1번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 알고리즘이란?-

- ✓ 문자열 배열에서 i번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이를 반지름 r을 저장하여 사용
- ✓ 시간복잡도 O(N)

문자열	b	a	n	a	n	a	c
배열	0	0	1				

배열 A[2] : 2번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 알고리즘이란?-

- ✓ 문자열 배열에서 i번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이를 반지름 r을 저장하여 사용
- ✓ 시간복잡도 O(N)

문자열	b	a	n	a	n	a	c
배열	0	0	1	2			

배열 A[3] : 3번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이

부분 문자열 S의 3-A[3]에서 3+A[3]까지는 팰린드롬이며, 3-A[3]-1에서 3+A[4]+1까지는 팰린드롬이 아니다.

어떤 문자열 S에서 A[i]가 존재한다면,
i - A[i]에서 i + A[i]까지는 팰린드롬이고, i - A[i] - 1에서 i + A[i] + 1까지는 팰린드롬이 아니라는 것!!!

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 알고리즘이란?-

- ✓ 문자열 배열에서 i번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이를 반지름 r을 저장하여 사용
- ✓ 시간복잡도 O(N)

문자열	b	a	n	a	n	a	c
배열	0	0	1	2	1		

배열 A[4] : 4번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 알고리즘이란?-

- ✓ **문자열 배열에서 i번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이를 반지름 r을 저장하여 사용**
- ✓ **시간복잡도 O(N)**

문자열	b	a	n	a	n	a	c
배열	0	0	1	2	1	0	

배열 A[5] : 5번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 알고리즘이란?-

- ✓ 문자열 배열에서 i번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이를 반지름 r을 저장하여 사용
- ✓ 시간복잡도 O(N)

문자열	b	a	n	a	n	a	c
배열	0	0	1	2	1	0	0

배열 A[6] : 6번째 문자를 중심으로 하는 가장 긴 팰린드롬 길이

02 매내처 알고리즘 (Manacher's algorithm)

-동작방식-

1. i는 1부터 N(문자열의 길이)까지 진행된다.

2. j < i인 모든 j에 대해 r=max(j + A[j])이라 하고, 또한 그러한 j를 p라 하자. 즉, r=p + A[p]

3. i와 r의 대소 관계에 따라 A[i]의 초기값이 결정된다.
   (1) i > r 이라면, A[i]의 초기값은 0이다.
   (2) i ≤ r 이라면, i는 p를 중심으로 한 팰린드롬에 속한다. 따라서 그 회문에서 i의 대칭점을 i'라 하자. 즉, i'=2p - i가 된다.
   그리고 A[i]의 초기값은 min(r - i, A[i'])이다. (즉, r은 중심점 p를 기준으로 하는 p + A[p]를 의미하게 된다.)

4. A[i]의 초기값이 결정되고, S[i - A[i]]와 S[i + A[i]]가 같을 때까지 A[i]를 증가시킨다.

02 매내처 알고리즘 (Manacher's algorithm)

-동작방식-

문자열 S	b	a	n	a	n	a	c
i	0	1	2	3	4	5	6
배열 A	0	0	1	2	1	0	0
r	0	0	4	6	6	6	6
p	0	0	3	4	4	4	4

j < i인 모든 j에 대해 r=max(j + A[j]), 이 때의 r을 p라고 한다.

02 매내처 알고리즘 (Manacher's algorithm)

-매내처 구현-

```
void manachers(string S, int N){
    int r = 0, p = 0;
    for (int i = 0; i < N; i++){
        if (i <= r)
            A[i] = min(A[2 * p - i], r - i);
        else
            A[i] = 0;

        while (i - A[i] - 1 >= 0 && i + A[i] + 1 < N && S[i - A[i] - 1] == S[i + A[i] + 1])
            A[i]++;

        if (r < i + A[i]){
            r = i + A[i];
            p = i;
        }
    }
}
```

03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

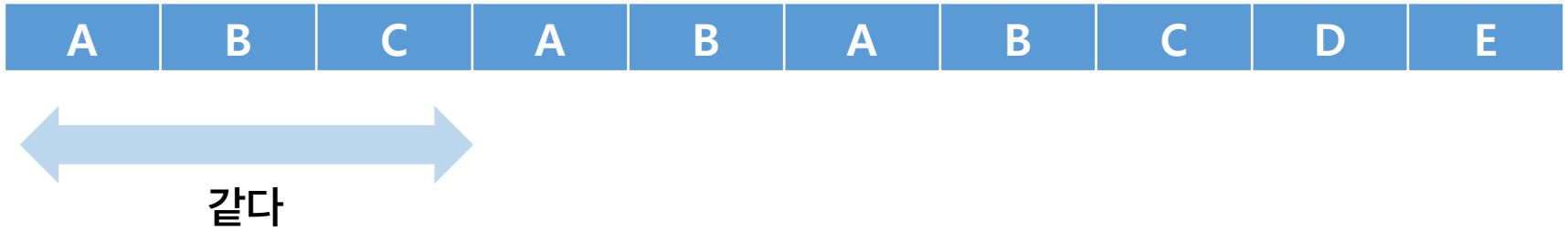
Q) 패턴 “ABC”가 어디서 등장하는지 찾아보자!

A	B	C	A	B	A	B	C	D	E

03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

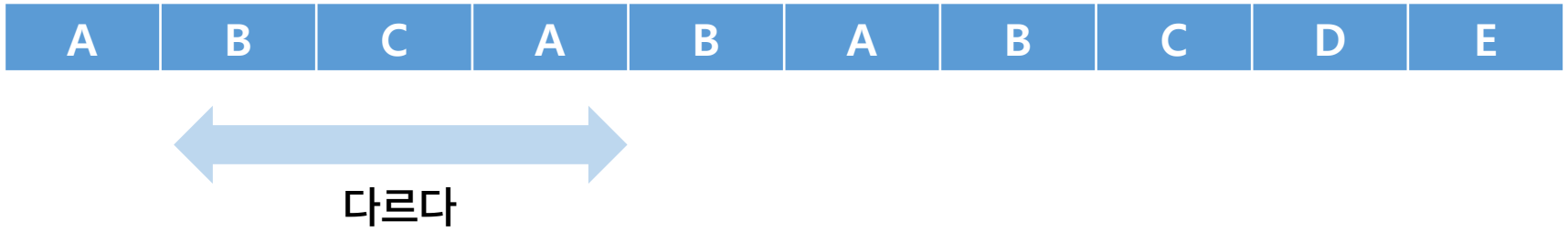
Q) 패턴 "ABC"가 어디서 등장하는지 찾아보자!



03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

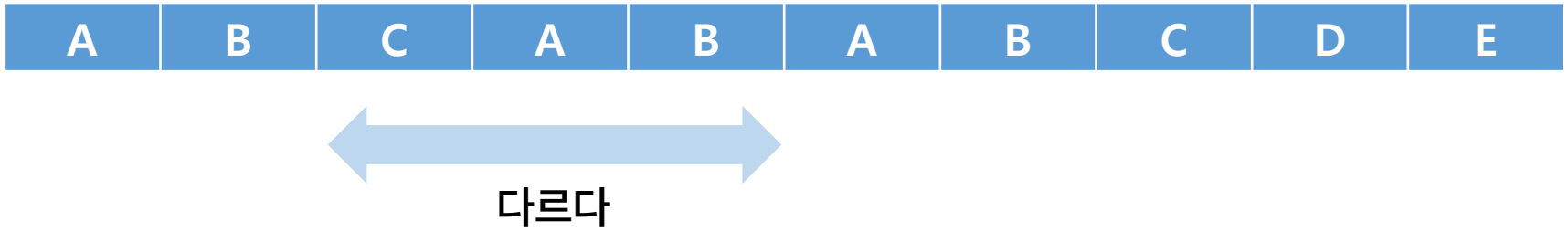
Q) 패턴 "ABC"가 어디서 등장하는지 찾아보자!



03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

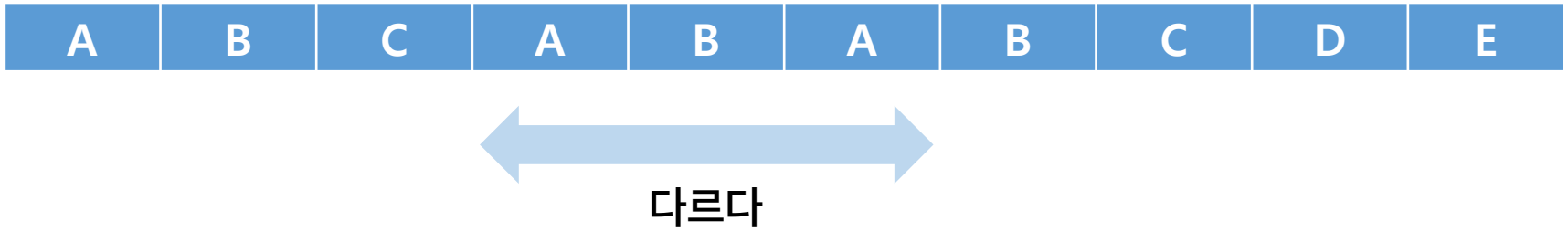
Q) 패턴 “ABC”가 어디서 등장하는지 찾아보자!



03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

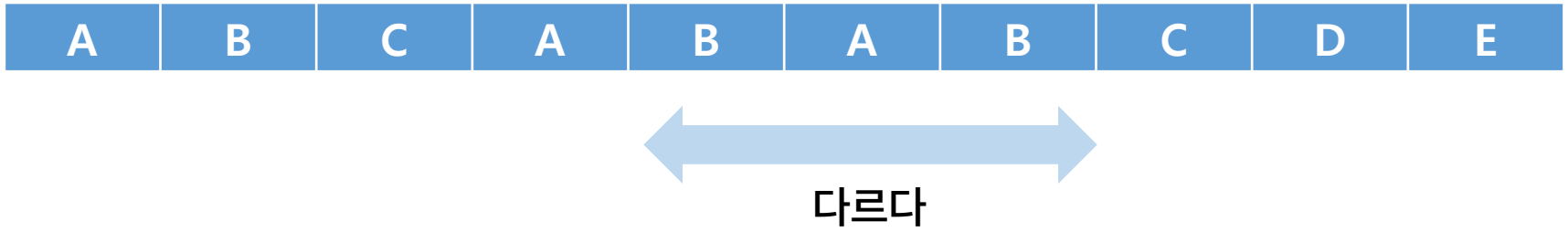
Q) 패턴 "ABC"가 어디서 등장하는지 찾아보자!



03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

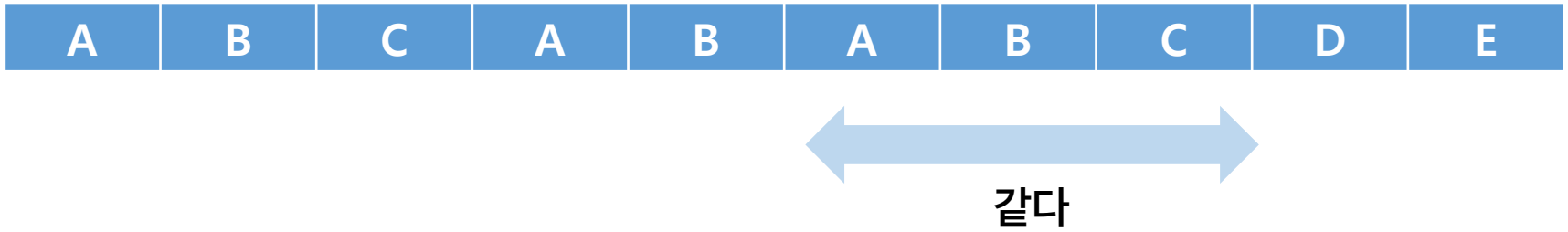
Q) 패턴 “ABC”가 어디서 등장하는지 찾아보자!



03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

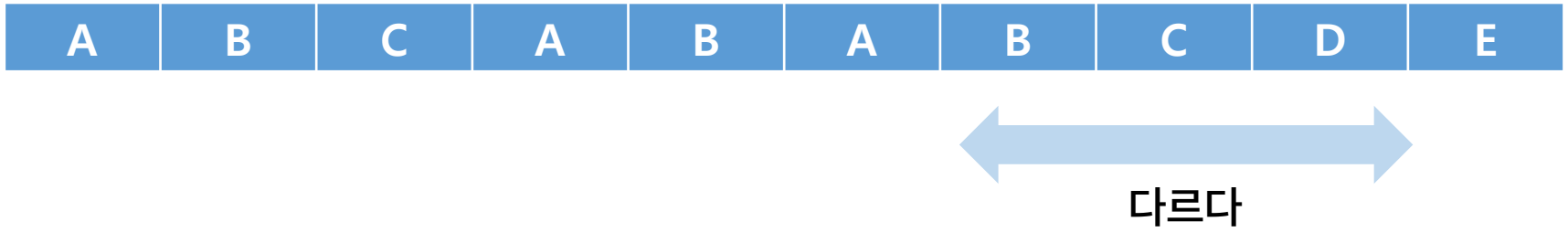
Q) 패턴 “ABC”가 어디서 등장하는지 찾아보자!



03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

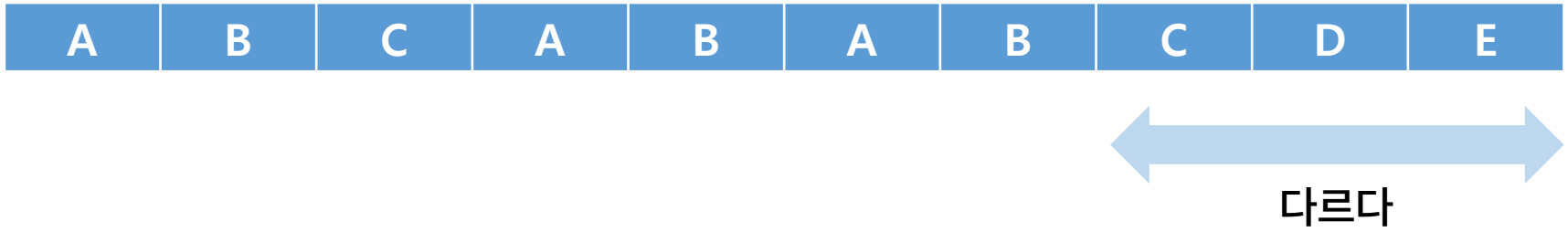
Q) 패턴 "ABC"가 어디서 등장하는지 찾아보자!



03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

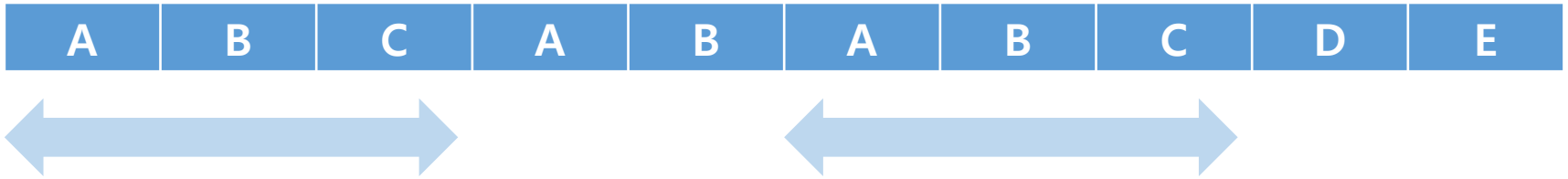
Q) 패턴 "ABC"가 어디서 등장하는지 찾아보자!



03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

Q) 패턴 “ABC”가 어디서 등장하는지 찾아보자!



위와 같은 과정을 진행했을 때,
텍스트 “ABCABABCDE” 에서 패턴 “ABC”는 총 2번 등장한다.

텍스트의 길이: N, 패턴의 길이: M
각 텍스트의 인덱스에 대해 패턴이 일치하는지 비교 -> O(NM)

03 문자열 검색 알고리즘 (KMP algorithm)

-가장 단순한 문자열 문제-

Q) 패턴 “ABC”가 어디서 등장하는지 찾아보자!

위와 같은 과정을 진행했을 때,
텍스트 “ABCABABCDE” 에서 패턴 “ABC”는 총 2번 등장한다.

텍스트의 길이: N, 패턴의 길이: M
각 텍스트의 인덱스에 대해 패턴이 일치하는지 비교 -> O(NM)

더 빠르고 효율적인
KMP 알고리즘을 사용하면
O(N+M)에 문자열 검색!!

03 문자열 검색 알고리즘 (KMP algorithm)

(사전지식)

1. 접두사(prefix)와 접미사(suffix)

banana의 접두사	banana의 접미사
b	a
ba	na
ban	ana
bana	nana
banan	anana
banana	banana

03 문자열 검색 알고리즘 (KMP algorithm)

(사전지식)

1. 접두사(prefix)와 접미사(suffix)

banana의 접두사	banana의 접미사
b	a
ba	na
ban	ana
bana	nana
banan	anana
banana	banana

2. pi 배열

: pi[i]는 주어진 문자열의 0~i까지의 부분 문자열 중에서 prefix == suffix 될 수 있는 부분 문자열 중에서 가장 긴 것의 길이

: ex) “ABAABAB”의 pi배열

i	부분 문자열	Pi[i]
0	A	0
1	AB	0
2	ABA	1
3	ABAA	1
4	ABAAB	2
5	ABAABA	3
6	ABAABAB	2

03 문자열 검색 알고리즘 (KMP algorithm)

-문자열 문제-

Q) 패턴 "ABCDABE"가 어디서 등장하는지 찾아보자!

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
텍스트	A	B	C	D	A	B	C	D	A	B	E	E
패턴	A	B	C	D	A	B	E					

다르다

03 문자열 검색 알고리즘 (KMP algorithm)

-문자열 문제-

Q) 패턴 "ABCDABE"가 어디서 등장하는지 찾아보자!

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
텍스트	A	B	C	D	A	B	C	D	A	B	E	E
패턴	A	B	C	D	A	B	E					

이 부분은 '일치한다'는 정보를 간과하고 있었다!!!

이 정보를 적극 활용해서 검색 속도를 개선하는 것이 KMP 알고리즘!!!

03 문자열 검색 알고리즘 (KMP algorithm)

-문자열 문제-

Q) 패턴 “ABCDABE”가 어디서 등장하는지 찾아보자!

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
텍스트	A	B	C	D	A	B	C(i)	D	A	B	E	E
패턴					A	B	C(j)	D	A	B	E	

일치했다는 사실을 적극 활용하여, 중간 시도를 건너뛰고 바로 이 단계로 넘어갈 수 있다
i: 텍스트의 현재 비교 위치 j: 패턴의 현재 비교위치

이것이 가능한 이유는,
“ABCDAB”에서 접두사 AB와 접미사 AB가 일치 + 이것이 접두사와 접미사가 일치하는 최대길이.
패턴 “ABCDABE”의 pi[5] = 2 값

03 문자열 검색 알고리즘 (KMP algorithm)

-문자열 문제-

Q) 패턴 "ABCDABE"가 어디서 등장하는지 찾아보자!

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
텍스트	A	B	C	D	A	B	C	D	A	B	E	E
패턴		A	B	C	D	A	B	E				

중간단계를 거쳤다면,
이 단계에서 패턴이 일치하려면, 적어도 pi[5]=5

03 문자열 검색 알고리즘 (KMP algorithm)

-문자열 문제-

Q) 패턴 "ABCDABE"가 어디서 등장하는지 찾아보자!

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
텍스트	A	B	C	D	A	B	C	D	A	B	E	E
패턴			A	B	C	D	A	B	E			

중간단계를 거쳤다면,
이 단계에서 패턴이 일치하려면, 적어도 pi[5]=4

03 문자열 검색 알고리즘 (KMP algorithm)

-문자열 문제-

Q) 패턴 "ABCDABE"가 어디서 등장하는지 찾아보자!

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
텍스트	A	B	C	D	A	B	C	D	A	B	E	E
패턴				A	B	C	D	A	B	E		

중간단계를 거쳤다면,
이 단계에서 패턴이 일치하려면, 적어도 pi[5]=3

03 문자열 검색 알고리즘 (KMP algorithm)

-문자열 문제-

Q) 패턴 “ABCDABE”가 어디서 등장하는지 찾아보자!

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
텍스트	A	B	C	D	A(i)	B	C	D	A	B	E	E
패턴					A(j)	B	C	D	A	B	E	

i: 텍스트의 현재 비교 위치　　j: 패턴의 현재 비교위치

i, j 를 여기서(4) 시작할 필요가 없다. “ABCDABE”에서 “AB”는 이미 텍스트와 일치하기 때문.

kmp 알고리즘은 ‘틀렸다’가 아니라, 조금이라도 ‘일치했다’는 정보에 주목하고,
미리 전처리 해둔 pi배열을 이용해 많은 중간 시도를 건너 뛸 수 있다.

03 문자열 검색 알고리즘 (KMP algorithm)

-kmp 구현-

pi배열 구하기

```
#kmp의 원리를 Pi배열에도 적용
#m:패턴의 길이 -> O(m)
vector<int> getPi(string p){
    int m = (int)p.size(); #string 크기
    int j=0;
    vector<int> pi(m, 0); #pi배열(크기=m, 0으로 초기화)

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

kmp 알고리즘

```
vector<int> kmp(string s, string p){
    vector<int> ans;
    vector<int> pi = getPi(p);
    int n = (int)s.size(), m = (int)p.size(), j =0;
    for(int i = 0 ; i < n ; i++){
        #일치했던 정보와 pi배열을 활용해 중간 단계를 뛰어 넘는부분
        #최대한 중간 단계를 뛰어넘고자, while문 사용
        while(j>0 && s[i] != p[j])
            j = pi[j-1];
        if(s[i] == p[j]){
            if(j==m-1){
                ans.push_back(i-m+1);
                j = pi[j];
            }else{
                j++;
            }
        }
    }
    return ans;
}
```

04 문제추천

-팰린드롬 알고리즘-
백준 10942: 팰린드롬?(G4) (https://www.acmicpc.net/problem/10942)

-manacher's 알고리즘-
백준 11046: 팰린드롬??(P5) (https://www.acmicpc.net/problem/11046)
백준 14444: 가장 긴 팰린드롬 부분 문자열(P5) (https://www.acmicpc.net/problem/14444)

-kmp 알고리즘-
백준 1786: 찾기(P5) (https://www.acmicpc.net/problem/1786)
백준 4353: 문자열 제곱(P5) (https://www.acmicpc.net/problem/4354)

THANK YOU