

Algorithm Study



문자열 알고리즘

문자열 알고리즘에는








문자열

단계	문제 번호	제목	정보	정답	제출	정답 비율
1	11654	5 아스키 코드	문자열 다루기 성공 분류	35321	44299	81.436%
아스키 코드에 대해 알아보는 문제						
2	11720	2 숫자의 합	성공 분류	40951	83594	50.839%
정수를 문자열로 입력받는 문제. Python처럼 정수 크기에 제한이 없다면 상관 없으나, 예제 3은 일반적인 정수 자료형에 담기에 너무 크다는 점에 주목합니다.						
3	10809	2 알파벳 찾기	성공 분류	32900	61416	54.672%
한 단어에서 각 알파벳이 처음 등장하는 위치를 찾는 문제						
4	2675	2 문자열 반복	성공 출처 다국어 분류	28301	54731	53.055%
각 문자를 반복하여 출력하는 문제						
5	1157	1 단어 공부	성공 분류	29406	76767	38.683%
주어진 단어에서 가장 많이 사용된 알파벳을 출력하는 문제						
6	1152	2 단어의 개수	성공 분류	37331	138345	27.467%
단어의 개수를 구하는 문제						
7	2908	2 상수	성공 출처 다국어 분류	26151	38457	68.811%
숫자를 뒤집어서 비교하는 문제						
8	5622	2 다이얼	성공 출처 다국어 분류	20580	36428	57.016%
규칙에 따라 문자에 대응하는 수를 출력하는 문제						
9	2941	5 크로아티아 알파벳	성공 출처 다국어 분류	19429	45196	45.449%
크로아티아 알파벳의 개수를 세는 문제						
10	1316	5 그룹 단어 체커	성공 분류	20751	40627	52.153%
조건에 맞는 문자열을 찾는 문제						

문자열 알고리즘

문자열 알고리즘에는

문자열 탐색 1

단계	문제 번호	제목	정보	정답	제출	정답 비율	
1	1786	 찾기	KMP	분류	5685	20143	28.387%
문자열 T에서 문자열 P가 있는지 찾는 알고리즘인 KMP 알고리즘을 배우는 문제							
2	4354	 문자열 제곱	KMP	출처 다국어 분류	926	3797	30.310%
KMP의 failure function을 활용하는 문제 1							
3	1305	 광고	KMP	분류	1365	2926	51.093%
KMP의 failure function을 활용하는 문제 2							
4	10266	 시계 사진들	KMP	출처 다국어 분류	432	1065	42.839%
문자열 찾기 알고리즘을 응용하는 문제							
5	14725	 재미굴	Trie	출처 분류	500	745	71.034%
트라이에 대한 감을 잡는 문제							
6	14425	 문자열 집합	Trie	분류	1590	3046	55.120%
트라이보다 쉽게 풀 수 있는 문제지만, 연습을 위해 트라이로 풀어 보시다.							
7	5670	 휴대폰 자판	Trie	출처 다국어 분류	1084	3329	29.146%
조금 어려운 문제							

문자열 알고리즘

문자열 알고리즘에는

KMP 알고리즘이 문자열 알고리즘의 핵심

접두사와 접미사를 이용한 단순한 최적화 알고리즘

Manacher : 팰린드롬 구하는 최적화 알고리즘

팰린드롬: 접미사와 접두사가 같은 단어

문자열 탐색 2

Z 알고리즘 : KMP 알고리즘의 동작을 뒤집은 것

LCP 배열 : 팰린드롬의 길이를 저장하는 배열

아호-코라식 알고리즘 : KMP를 Trie 를 통해 해결하는 알고리즘

라빈-카프 알고리즘 : 해쉬를 사용하는 $O(N)$ 알고리즘

보이어-무어 알고리즘 : KMP 알고리즘의 연산성 $O(N) \sim O(MN)$ 알고리즘

1	13275	5 가장 긴 팰린드롬 부분 문자열	Manacher	분류	345	1184	44.158%
각 위치를 중심으로 하는 가장 긴 팰린드롬의 길이를 구하는 Manacher 알고리즘에 대해 배워 봅시다.							
2	16163	5 #15164번_제보	Manacher	분류	106	301	34.137%
"M 어찌구" 알고리즘으로 주인공의 원수를 갚는 문제							
3	11046	5 팰린드롬??	Manacher	분류	187	661	32.581%
팰린드롬 판별을 효율적으로 하는 문제							
4	13713	1 문자열과 쿼리	Z	분류	166	446	40.467%
Z 알고리즘에 대해 배워 봅시다. (단, 이 문제에서 구하는 것과 완전히 일치하지는 않습니다.)							
5	16229	4 반복 패턴	Z	출처 분류	65	341	18.696%
Z 알고리즘 응용 문제							
6	9248	3 Suffix Array	접미사와 LCP	류	1185	3125	38.897%
접미사 배열과 LCP 배열에 대해 알아보시다.							
7	1605	4 반복 부분문자열	접미사	분류	555	2001	38.196%
접미사 배열 응용 문제 1							
8	11479	3 서로 다른 부분 문자열의 개수 2	접미사	분류	365	874	42.115%
접미사 배열 응용 문제 2							
9	13322	1 접두사 배열	접미사	분류	258	424	66.013%
접미사 배열은 있는데, 왜 접두사 배열은 없을까요? 이 문제를 풀면 왜 없는지 알 수 있습니다.							
10	9250	2 문자열 집합 판별	아호-코라식	분류	763	2145	33.034%
주어진 단어들을 한꺼번에 찾는 아호-코라식 알고리즘에 대해 알아보시다.							
11	10256	3 돌연변이	아호-코라식	출처 다국어 분류	569	2585	21.045%
아호-코라식 알고리즘 응용 문제							
12	2809	2 아스키 거리	아호-코라식	출처 다국어 분류	137	918	14.990%
접미사 배열로 풀 수도 있고, 아호-코라식으로 풀 수도 있는 문제							

문자열 알고리즘 차후 공부 방향

회의를 해보고;

수업날에 모든 내용을 다룰 수 없으니, 이에 대해 어쩔지?

발표자가 공부 방향과 풀어야할 문제를 제시

1.5 주차 문제 병합과 약간 겹침

↳ 1.5, 2.5 주차 문제 병합시 문제가 겹치는 것을 막기 위해, 사전 공지?

2주차 수업시간에 골드 수준 문제 괜찮은지?

레퍼의 주차별 README에 수업 정리? 작성?

파이썬 사용의 이득

1. 다양한 라이브러리

이미 구현된 다양한 라이브러리

자료구조의 대부분이 이미 구현되어 있다.
c++의 vector는 파이썬의 list

많은 함수들

```
str.upper()  
str.lower()  
str.isupper()  
str.islower()  
str.replace()  
str.find()  
str.index()  
str.reversed()  
.  
.
```

문자열과 리스트를 다루기 쉬운 이유

파이썬 사용의 이득

2. 짧은 코드

1번의 다양한 라이브러리와 자료구조 덕분에 직접 구현하는 코드도 불필요

파이프라인식 코드 구조 때문에 코드의 단축화

```
s = sys.stdin.read().strip()
```

파이썬 사용의 이득

3. 개발 방식의 코딩테스트에서의 이득

카카오 공채 시험에서 실제로 프로그램 개발 방식의 테스트를 실시중
특정 서버와 통신하는 프로그램 개발

C++, JAVA에 비해 필요한 라이브러리와 코드가 훨씬 적음

파이썬 사용의 이득 예시

1152번
단어의 개수
브2

문제

영어 대소문자와 띄어쓰기만으로 이루어진 문자열이 주어진다. 이 문자열에는 몇 개의 단어가 있을까? 이를 구하는 프로그램을 작성하시오. 단, 한 단어가 여러 번 등장하면 등장한 횟수만큼 모두 세어야 한다.

입력

첫 줄에 영어 대소문자와 띄어쓰기로 이루어진 문자열이 주어진다. 이 문자열의 길이는 1,000,000을 넘지 않는다. 단어는 띄어쓰기 한 개로 구분되며, 공백이 연속해서 나오는 경우는 없다. 또한 문자열의 앞과 뒤에는 공백이 있을 수도 있다.

출력

첫째 줄에 단어의 개수를 출력한다.

예제 입력 1 [복사](#)

The Curious Case of Benjamin Button

예제 출력 1 [복사](#)

6

파이썬 사용의 이득 예시

Python

```
import sys
s = sys.stdin.read().strip()
if not s:
    print("0")
else:
    print(len(s.split(" ")))
```

CPP

```
#include<iostream>
#include<string>

using namespace std;

int main()
{
    ios::sync_with_stdio(false);

    string str;
    int ans = 1;
    getline(std::cin, str);
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] == ' ')
        {
            ans++;
        }
    }

    if (str.length() == 1 && str[0] == ' ')
        cout << 0 << endl;
    else
    {
        if (str[0] == ' ')
            ans--;
        if (str[str.length() - 1] == ' ')
            ans--;
        cout << ans << endl;
    }

    return 0;
}
```

문자열 알고리즘 - KMP

문자열 알고리즘에는

문자열 탐색 알고리즘으로는 '라반-카프' '보이어-무어' 'KMP 알고리즘' 등이 있고,

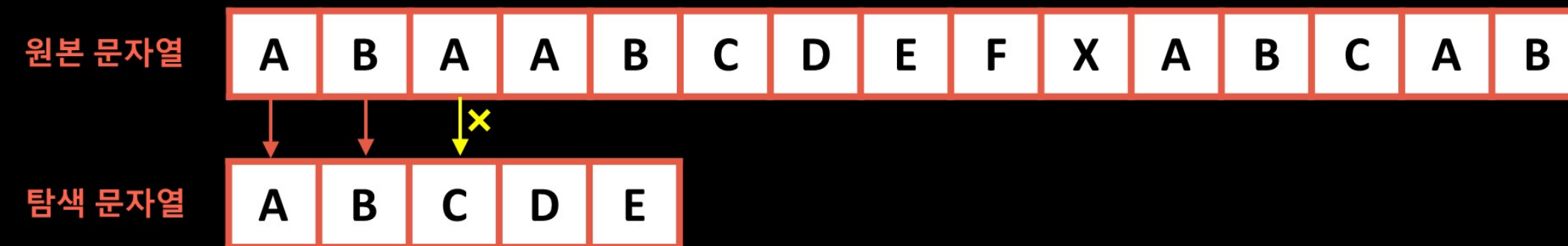
문자열 안에서 여러개의 단어를 동시에 찾는 '아호-코라식 알고리즘'

다양한 문자열 처리 문제에서 유용하게 적용할 수 있는 처리 분야의 '맥가이버 칼'인 접미사 배열을 빠르게 생성하는 '맨버-마이어스의 알고리즘' 등이 개발됨.

문자열 탐색 알고리즘에서 가장 중요한 아이디어를 담고있는
'KMP 문자열 탐색 알고리즘'

문자열 알고리즘 - KMP

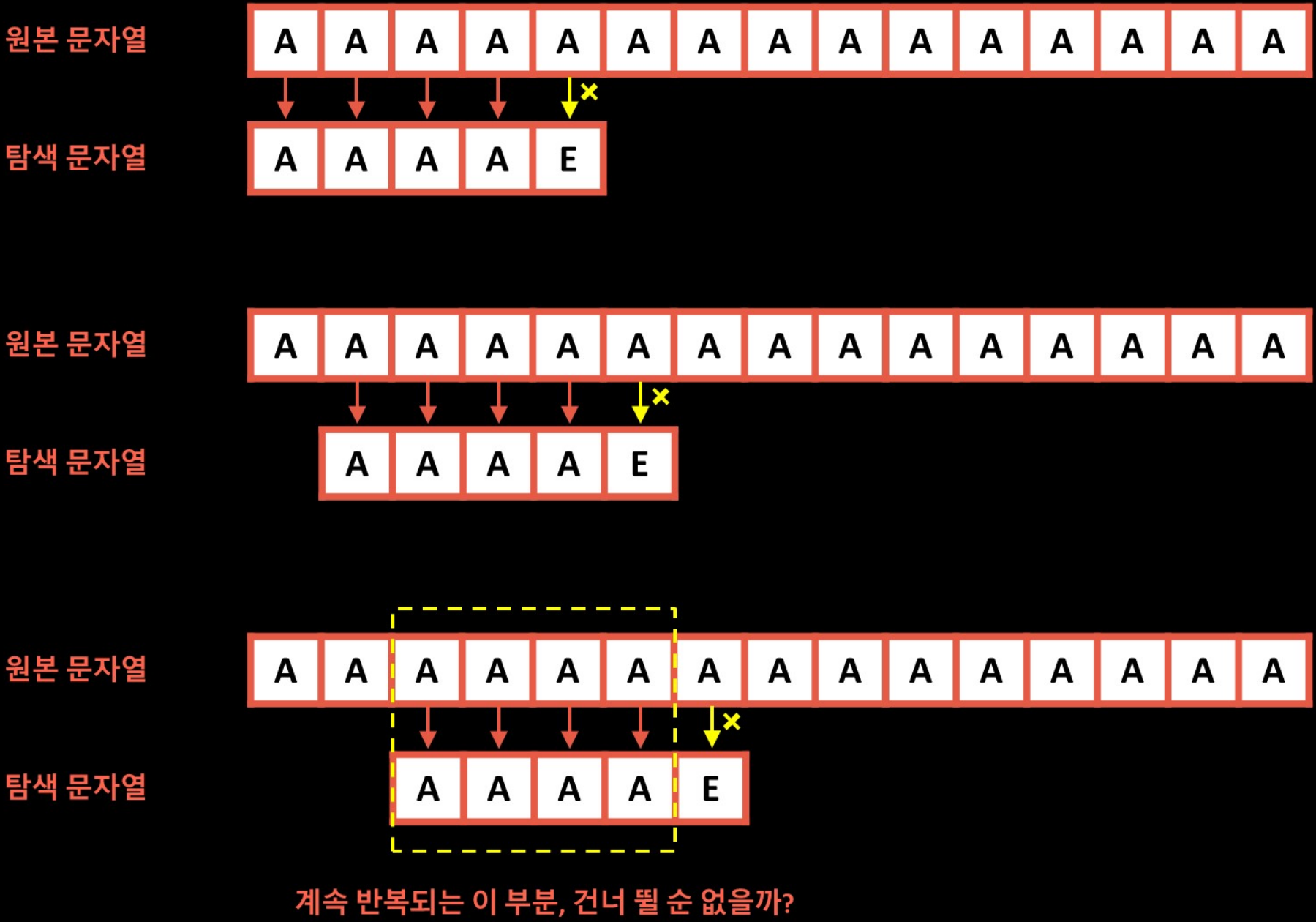
무식하게 탐색



가장 간단하고 쉽게 생각할 수 있는 문자열 탐색 방법은 원본 문자열과 우리가 찾고 싶은
탐색 문자열을 모든 문자에 대해서 하나하나 비교해보는 방법

문자열 알고리즘 - KMP

무식하게 탐색



간단한 문자열 탐색 알고리즘이 매우 느려지는 경우

원본 문자열은 어차피 일치하지 않지만, 끝까지 탐색하기 전까지 이를 알지못하므로
매번 끝까지 탐색해야 한다.

O(NM) 원본 문자열 길이 N, 탐색 문자열 길이 M

문자열 알고리즘 - KMP

무식하게 탐색

원본 문자열 길이 N, 탐색 문자열 길이 M

Brute Force

$O(NM)$

KMP

$O(N + M)$

문자열 알고리즘 - KMP

접두사와 접미사



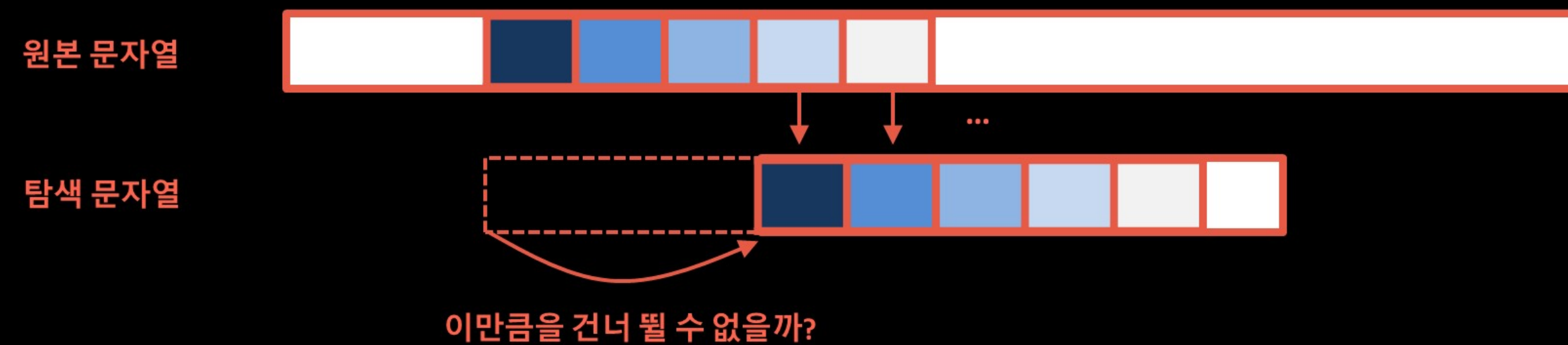
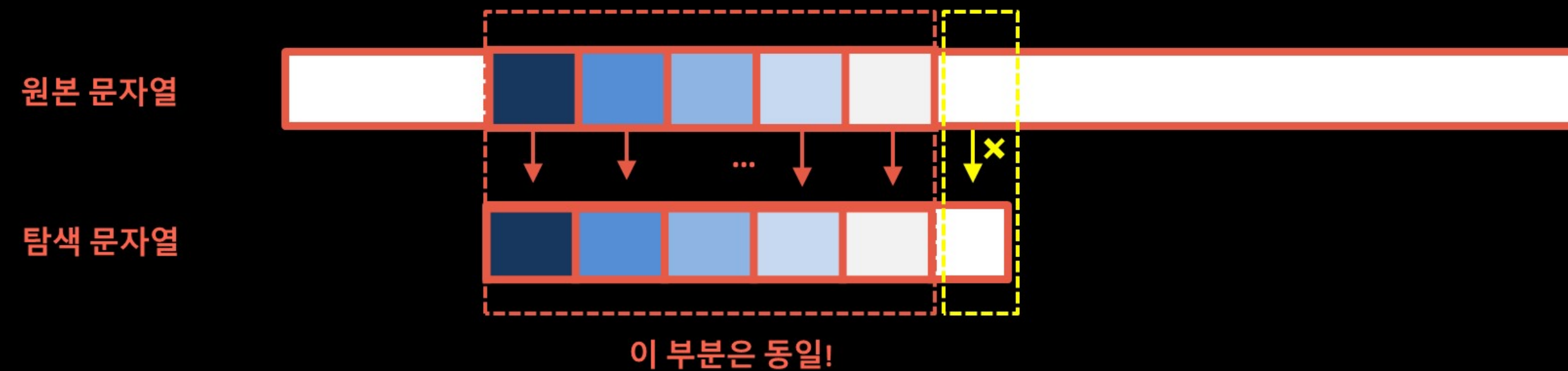
KMP 알고리즘에서 메인 아이디어로 사용되는 접두사와 접미사의 정의

문자열 알고리즘에서의 접두사와 접미사란,
맨 앞과 맨 뒤에서 같은 문자열이 나오는 경우

ABCAB가 대표적인 예

문자열 알고리즘 - KMP

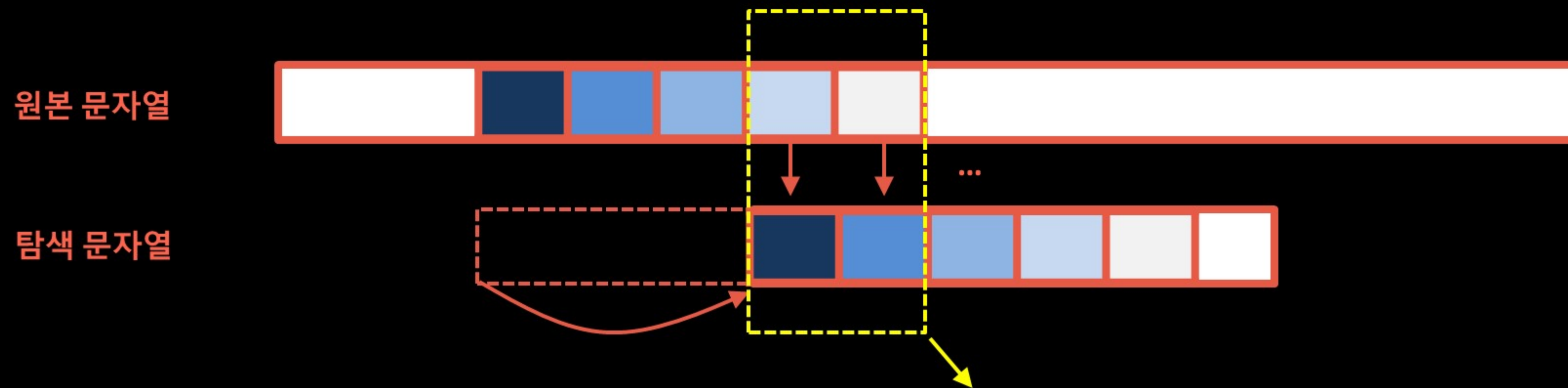
KMP 알고리즘의 아이디어



문자열이 불일치 할 때, 탐색을 시작했던 위치의 다음 문자부터가 아니라
우리가 앞서 탐색했던 정보를 이용하여 "몇 칸 정도는 건너뛴 후 탐색할 수 있지 않을까?"

문자열 알고리즘 - KMP

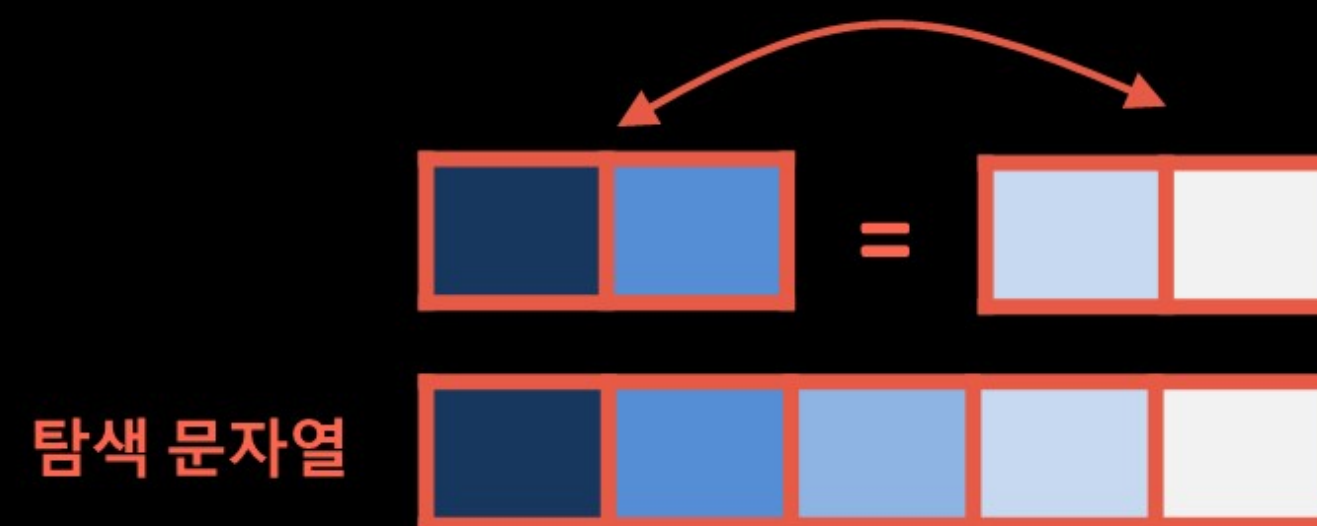
KMP 알고리즘의 아이디어



문자열 탐색을 건너 뛸 수 있다면, 건너 뛴 후 이 부분이 동일해야 함!

탐색 문자열의 앞부분과
원본 문자열의 비교 중인 부분의 뒷부분이
동일한 부분까지는 건너뛸 수 있다

동일해야 하는 부분이, 탐색 문자열의 접두사와 접미사 네!



문자열 알고리즘 - KMP

KMP 알고리즘 코드

문자열 알고리즘 - KMP

KMP 알고리즘 코드 - 탐색 문자열의 접두, 접미사 계산

i begin

↓ ↓

aabaabacd

pi[begin + i] = 1

pi[1 + 0] = 1

```
for (int begin = 1; begin < m; begin++) {  
    for (int i = 0; begin + i < m; i++) {  
        if (search[begin + i] != begin[i]) break;  
        pi[begin + i] = max(pi[begin + i], i + 1);  
    }  
}
```

문자열 알고리즘 - KMP

KMP 알고리즘 코드 - 탐색 문자열의 접두, 접미사 계산

i begin

↓ ↓

aabaabacd

```
for (int begin = 1; begin < m; begin++) {  
    for (int i = 0; begin + i < m; i++) {  
        if (search[begin + i] != begin[i]) break;  
        pi[begin + i] = max(pi[begin + i], i + 1);  
    }  
}
```


문자열 알고리즘 - KMP

KMP 알고리즘 코드 - 탐색 문자열의 접두, 접미사 계산

i begin

↓ ↓

aabaabacd

```
for (int begin = 1; begin < m; begin++) {  
    for (int i = 0; begin + i < m; i++) {  
        if (search[begin + i] != begin[i]) break;  
        pi[begin + i] = max(pi[begin + i], i + 1);  
    }  
}
```

문자열 알고리즘 - KMP

KMP 알고리즘 코드 - 탐색 문자열의 접두, 접미사 계산

i begin

↓ ↓

aabaabacd

pi[begin + i] = 1

pi[3 + 0] = 1

```
for (int begin = 1; begin < m; begin++) {  
    for (int i = 0; begin + i < m; i++) {  
        if (search[begin + i] != begin[i]) break;  
        pi[begin + i] = max(pi[begin + i], i + 1);  
    }  
}
```

문자열 알고리즘 - KMP

KMP 알고리즘 코드 - 탐색 문자열의 접두, 접미사 계산

i begin

↓ ↓

aabaabacd

pi[begin + i] = 2

pi[4 + 1] = 1

```
for (int begin = 1; begin < m; begin++) {  
    for (int i = 0; begin + i < m; i++) {  
        if (search[begin + i] != begin[i]) break;  
        pi[begin + i] = max(pi[begin + i], i + 1);  
    }  
}
```

문자열 알고리즘 - KMP

KMP 알고리즘 코드 - 탐색 문자열의 접두, 접미사 계산

각 위치별로 접두, 접미사의 길이를 계산함.

특정 위치까지 탐색한 상태에서 불일치가 발견될 경우,
불일치가 발생한 위치 직전까지 부분을 기준으로 접미, 접두사를 통해 건너뛰어야 하므로

```
// 현재 불일치가 발생한 위치는 begin + matched  
// 여기서 접두, 접미사의 길이인 pi[matched - 1] 을 빼주면 다음 탐색 시작 위치  
begin += matched - pi[matched - 1];  
matched = pi[matched - 1];
```


문자열 알고리즘 - KMP

KMP 알고리의 시간 복잡도

각 위치별로 접두, 접미사의 길이를 계산함.

특정 위치까지 탐색한 상태에서 불일치가 발견될 경우,
불일치가 발생한 위치 직전까지 부분을 기준으로 접미, 접두사를 통해 건너뛰어야 하므로

```
// 현재 불일치가 발생한 위치는 begin + matched  
// 여기서 접두, 접미사의 길이인 pi[matched - 1] 을 빼주면 다음 탐색 시작 위치  
begin += matched - pi[matched - 1];  
matched = pi[matched - 1];
```

