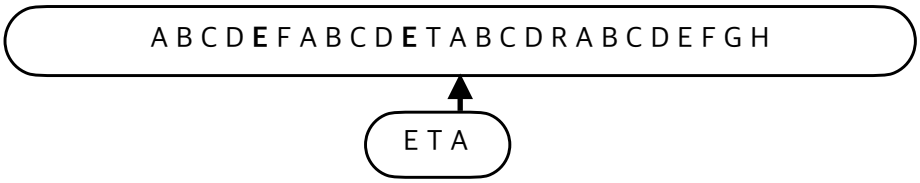
11주. 문자열 탐색과 성능

텍스트 (큰 문서) 에서 패턴 (찾으려 하는 문자열) 이 존재하는지, 존재한다면 문자열의 어느 위치에 있는지 알아내는 프로그램을 작성해 봅니다.
이 때 필요한 개념도 익혀봅니다.

일치하는 문자열을 찾는다는 기본 개념

텍스트 : A B C D E F A B C D E T A B C D R A B C D E F G H
패턴 : E T A



1. 텍스트에서 패턴이 시작되는 위치를 찾는다 (처음 E 가 등장하는 지점)
2. 각 위치에서 패턴의 한 인덱스씩 증가시키며 비교. (E -> T -> A)
3. 패턴의 끝까지 비교 성공하면 문자열을 찾는데 성공
4. 실패한다면 다음 E 를 찾아 1번부터 시작

이 방법을 Brute Force 탐색이라 합니다.
가장 단순하고 가장 높은 시간 복잡도를 가진 알고리즘이기 때문에
다른 문자 탐색 알고리즘에 대한 성능 비교 지표가 됩니다.

물론 시간 복잡도가 높음에도 불구하고, 구현이 매우 간단하며
추가적인 자료구조가 불필요 하므로 문자열이 충분히 짧거나 수행시간이 중요하지 않은 경우에는 사용해볼 만 합니다.

Worst -Case 의 시간 복잡도는 **O**(n*m) 입니다.

패턴이 매우 길며 패턴과 비슷한 문자열이 텍스트 내에 반복적으로 존재하는 경우
(예 : 염기 서열에서 특정 패턴을 찾는 경우 등)
패턴의 종료 지점에서 실패하는 경우가 많이 발생하는 상황에는 적합하지 않음

예를들어,

텍스트).

ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG
ABCDABCDAJABCDABCDAKABCDABCDALABCDABCDAMABCDABCDAB

패턴)
ABCDABCDAB

패턴내의 ABCDABCDA 가 반복적으로 등장하다가 마지막 글자가 다른 경우가 반복되는 비효율적인 경우거나 (염기서열 탐색의 예)

…ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG…
1) ABCDABCDAB
2) ABCDABCDAB
3) ABCDABCDAB
4) ABCDABCDAB
5) ABCDABCDAB
…

텍스트).

ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG
ABCDABCDAJABCDABCDAKABCDABCDALABCDABCDAMABCDABCDAB

패턴)
ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG
ABCDABCDAJABCDABCDAKABCDABCDALABCDABCDAMABCDABCDAZ

본문과 패턴의 길이가 비슷하면서 종단의 문자만 다른 경우
(암호화 해시 검증의 예)

BruteForce 의 경우에 별도의 FallBack 을 하지 않기 때문에 매우 긴 수행시간을 요구하게 됩니다.

KMP Algorithm

Idea.

위의 예시 중.
…ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG…
1) ABCDABCDAB
2) ABCDABCDAB
3) ABCDABCDAB
4) ABCDABCDAB
5) ABCDABCDAB

패턴 내에서 최소한 비슷한 구간은 다시 비교하지 않아도 될것 같다!
= 접두사와 접미사가 같은 부분이 있다면 그 만큼 비교를 건너 뛸 수 있겠다

ABCDABCDAB

문자	A	B	C	D	A	B	C	D	A	B
인덱스	0	1	2	3	4	5	6	7	8	9

LPS 배열 / Longest Proper Prefix which is also a Suffix
= 패턴 내에서 접두사와 접미사가 일치하는 가장 긴 길이

위의 문자열을 계산 하면,
lps[0] = 0, len = 0 , i = 1 로 초기화

1. i = 1 , pattern[1] = B , is not A (pattern[0]) -> lps[1] = 0
2. i = 2 , pattern[2] = C , is not A (pattern[0]) -> lps[2] = 0
3. I = 3 , pattern[3] = D , is not A (pattern[0]) -> lps[3] = 0
4. I = 4 , pattern[4] = A , is A (pattern[len]) -> len++, lps[4] = 1
5. I = 5 , pattern[5] = B , is B (pattern[len]) -> len++, lps[5] = 2

…

문자	A	B	C	D	A	B	C	D	A	B
LPS	0	0	0	0	1	2	3	4	5	6

FallBack 동작의 간단한 예

Step 1
텍스트 : ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG
패턴 : ABCDABCDAB

패턴의 마지막 B 에 대한 LPS 테이블 값은 LPS[9-1] = 5

Step 2. 패턴의 5번째부터 비교
텍스트 :ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG
패턴 : ABCDABCDAB

불일치 B 에 대한 LPS 테이블 값은 LPS[5-1] = 1

Step 3. 패턴의 5번째부터 비교
텍스트 :ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG
패턴 : ABCDABCDAB

불일치 B 에 대한 LPS 테이블 값은 LPS[1-1] = 0

Step 4. 패턴이 일치하지 않으므로 텍스트 인덱스를 증가시키고 패턴 0부터 비교 시작
텍스트 :ABCDABCDACABCDABCDADABCDABCDAZABCDABCDAFABCDABCDAG
패턴 : ABCDABCDAB

KMP알고리즘의 동작 방식

1. 패턴 전처리 (LPS 배열 생성)
2. 문자 일치시 진행
3. 문자 불일치시 LPS 배열의 값을 통해 점프 (텍스트 인덱스는 유지)

KMP 알고리즘의 시간 복잡도는 O (n + m) 입니다

시각화 참고 : YouTube.com/watch?v=pu2aO_3R118