실전 알고리즘 0x0E강
문자열(KMP, 라빈 카프)

BaaaaaaaaaaaaaaaarkingDog

목차

0x00 문자열 기초

- C++에는 String이라는 type이 존재한다. Python보다는 쓰기 불편하지만 그럭저럭 쓸만은 하다. 사실 C++로 하는 것 보다 그냥 Python으로 하는게 낫다.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
  string a = "abcdef";
  string b = "zxcvzxc";
  if(a < b) cout << "a는 사전순으로 b보다 앞이다.\n";
  else cout << "b는 사전순으로 a보다 앞이다.\n";
  string c = a.substr(2,3); // c = "cde", 2번째부터 3글자를 추출하라
  string d = "hello my name is baaaaaarkingdog";
  int idx = d.find("my"); // idx = 6
}
```

0x00 문자열 기초

- S[a:b] : “S[a] S[a+1] ... S[b-1]”
- S = “ABCDEF”일 때 S[2:5] = “CDEF”, S[3:4] = “D”, S[0:2] = “AB”
- 접두사 : 문자열의 첫 문자를 포함하는 연속한 문자열, S[0:x]
- 접미사 : 문자열의 끝 문자를 포함하는 연속한 문자열, S[x:|S|]
- A, AB, ABC, ABCD, ABCDE는 ABCDE의 접두사이다.
- E, DE, CDE, BCDE, ABCDE는 ABCDE의 접미사이다.

0x00 문자열 기초

- 제공되는 함수로 모든 문자열 문제를 비빌 수 있을 줄 알았지만 현실은 그렇지 않다.

Suffix Array란, 문자열 S가 있을 때 그 접미사들을 정렬해 놓은 배열이다. 예를 들어, 문자열 S=banana의 접미사는 아래와 같이 총 6개가 있다.

Suffix	i
banana	1
anana	2
nana	3
ana	4
na	5
a	6

이를 Suffix 순으로 정렬하면 아래와 같다.

Suffix	i
a	6
ana	4
anana	2
banana	1
na	5
nana	3

정렬된 i의 배열 [6,4,2,1,5,3]을 S의 Suffix Array라고 한다.

문자열 S의 LCP Array는 Suffix Array를 구한 다음, 각 Suffix마다 정렬된 상태에서 바로 이전 Suffix와의 LCP (Longest Common Prefix, 최장 공통 접두사)의 길이를 배열에 담은 것이다. 위의 예에서 LCP Array는 [x,1,3,0,0,2]가 된다.

길이가 50만보다 작거나 같은 문자열이 주어졌을 때, Suffix Array와 LCP Array를 구하는 프로그램을 작성하시오.

문제

문자열 S가 주어졌을 때, S의 서로 다른 부분 문자열의 개수를 구하는 프로그램을 작성하시오.

부분 문자열은 S에서 연속된 일부분을 말하며, 길이가 1보다 크거나 같아야 한다.

예를 들어, ababc의 부분 문자열은 a, b, a, b, c, ab, ba, ab, bc, aba, bab, abc, abab, babc, ababc가 있고, 서로 다른것의 개수는 12개이다.

입력

첫째 줄에 문자열 S가 주어진다. S는 알파벳 소문자로만 이루어져 있고, 길이는 1,000 이하이다.

출력

첫째 줄에 S의 서로 다른 부분 문자열의 개수를 출력한다.

문제

집합 S는 크기가 N이고, 원소가 문자열인 집합이다. Q개의 문자열이 주어졌을 때, 각 문자열의 부분 문자열이 집합 S에 있는 판별하는 프로그램을 작성하시오. 문자열의 여러 부분 문자열 중 하나라도 집합 S에 있으면 'YES'를 출력하고, 아무것도 없으면 'NO'를 출력한다.

예를 들어, 집합 S = {"www","woo","jun"} 일 때, "myungwoo"의 부분 문자열인 "woo" 가 집합 S에 있으므로 답은 'YES'이고, "hongjun"의 부분 문자열 "jun"이 집합 S에 있으므로 답은 'YES'이다. 하지만, "dooho"는 모든 부분 문자열이 집합 S에 없기 때문에 답은 'NO'이다.

입력

첫째 줄에 집합 S의 크기 N이 주어진다. (1 ≤ N ≤ 1000)

다음 N개 줄에 집합 S의 원소들이 주어진다. 이 문자열의 길이는 100을 넘지 않는다.

다음 줄에 답을 판별해야 하는 문자열의 개수 Q가 주어진다. (1 ≤ Q ≤ 1000)

다음 Q개 줄에 답을 판별해야 하는 문자열이 주어진다. 이 문자열의 길이는 10000을 넘지 않는다.

입력으로 주어지는 모든 문자열은 알파벳 소문자로만 이루어져 있다.

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?(패턴 매칭 문제)

A	O	R	O	N	D	O	N	T	I	S	S

B	N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A

O	R	O	N	D	O	N	T	I	S	S

B

N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A	O	R	O	N	D	O	N	T	I	S	S

B	N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A

O	R	O	N	D	O	N	T	I	S	S

B

N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A

O	R	O	N	D	O	N	T	I	S	S

B

N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A

O	R	O	N	D	O	N	T	I	S	S

B

N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A

O	R	O	N	D	O	N	T	I	S	S

B

N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A										
O	R	O	N	D	O	N	T	I	S	S

B		
N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A

O	R	O	N	D	O	N	T	I	S	S

B

N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A

O	R	O	N	D	O	N	T	I	S	S

B

N	T	I

0x01 KMP

- 문자열 A 안에 문자열 B가 들어있는지를 어떻게 판단할 수 있을까?

A

O	R	O	N	D	O	N	T	I	S	S

B

N	T	I

A 안에 B가 들어있다.

0x01 KMP

- 구현을 해보자

```
bool find(string &A, string& B){
  if(A.size() < B.size()) return false;
  for(int st = 0; st <= A.size() - B.size(); st++){
    bool isMatch = true;
    for(int i = st; i < st+B.size(); i++){
      if(A[i] != B[i-st]){
        isMatch = false;
        break;
      }
    }
    if(isMatch) return true;
  }
  return false;
}
```

0x01 KMP

- 시간복잡도는 얼마일까?

A	A	A	A	A	A	A	A	A	A	A	A

B	A	A	A	A	A	B

0x01 KMP

- 시간복잡도는 얼마일까?

A	A	A	A	A	A	A	A	A	A
B	A	A	A	A	A	B			

0x01 KMP

- 시간복잡도는 얼마일까?

A	A	A	A	A	A	A	A	A	A
B		A	A	A	A	A	B		

0x01 KMP

- 시간복잡도는 얼마일까?

A

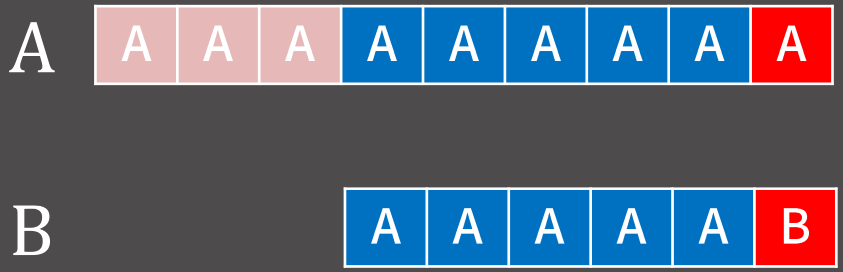
A	A	A	A	A	A	A	A	A

B

A	A	A	A	A	B

0x01 KMP

- 시간복잡도는 얼마일까?



- 최악의 경우 O(|A| × |B|)

0x01 KMP

- KMP : 패턴 매칭 문제를 $|A| + |B|$ 에 해결할 수 있는 기적의 알고리즘
- 뒤돌아서면 헷갈리는 알고리즘ㅠ_ㅠ
- 먼저 KMP에서 쓰이는 “실패함수”를 알면 KMP를 이해하는데 도움이 된다.

0x01 KMP - 실패함수

- 실패함수 F[x] : S[0:k] = S[x+1-k:x+1]을 만족하는 최대 k(단 k는 x 이하)
- 문자열 S[0:x+1]에서 접두사와 접미사가 일치하는 최대 길이

S	A	B	A	B	A	C	A	B	A

F(2) = 1 | A | B | A |

S[0:1] = “A”, S[2:3] = “A”, S[0:1] = S[2:3]

S[0:2] = “AB”, S[1:3] = “AB”, S[0:2] ≠ S[2:3]

F(3) = 2 | A | B | A | B |

S[0:2] = “AB”, S[2:4] = “AB”, S[0:2] = S[2:4]

F(4) = 3 | A | B | A | B | A |

S[0:3] = “ABA”, S[2:5] = “ABA”, S[0:3] = S[2:5]

0x01 KMP - 실패함수

- 실패함수 F[x]를 어떻게 구할 것인가? $k = x$일 때, $k = x\text{-}1$일 때, $k = x\text{-}2$일 때 ⋯ 하나하나 해보면 가능

S

A	B	A	B	A	C	A	B	A

- F(6)을 같이 구해보자

A	B	A	B	A	C

B	A	B	A	C	A

$$F(6) \neq 6$$

0x01 KMP - 실패함수

- 실패함수 F[x]를 어떻게 구할 것인가? k = x일 때, k = x-1일 때, k = x-2일 때 …
  하나하나 해보면 가능

S	A	B	A	B	A	C	A	B	A

- F(6)을 같이 구해보자

A	B	A	B	A

A	B	A	C	A

$F[6] \neq 5$

0x01 KMP - 실패함수

- 실패함수 F[x]를 어떻게 구할 것인가? $k = x$일 때, $k = x\text{-}1$일 때, $k = x\text{-}2$일 때 …
  하나하나 해보면 가능

S

A	B	A	B	A	C	A	B	A

- F(6)을 같이 구해보자

A	B	A	B

B	A	C	A

$$F[6] \neq 4$$

0x01 KMP - 실패함수

- 실패함수 F[x]를 어떻게 구할 것인가? $k = x$일 때, $k = x-1$일 때, $k = x-2$일 때 …
  하나하나 해보면 가능

S	A	B	A	B	A	C	A	B	A

- F(6)을 같이 구해보자

A	B	A

A	C	A

$$F[6] \neq 3$$

0x01 KMP - 실패함수

- 실패함수 F[x]를 어떻게 구할 것인가? $k = x$일 때, $k = x\text{-}1$일 때, $k = x\text{-}2$일 때 ⋯ 하나하나 해보면 가능

S									
	A	B	A	B	A	C	A	B	A

- F(6)을 같이 구해보자

A	B

C	A

$$F[6] \neq 2$$

0x01 KMP - 실패함수

- 실패함수 F[x]를 어떻게 구할 것인가? $k = x$일 때, $k = x\text{-}1$일 때, $k = x\text{-}2$일 때 …
  하나하나 해보면 가능

S	A	B	A	B	A	C	A	B	A

- F(6)을 같이 구해보자

A

A

$F[6] = 1$

0x01 KMP - 실패함수

- 각 F[x]에 대해 최악의 경우 $O(|S|^2)$번의 연산이 필요하므로 총 $O(|S|^3)$.
- 그런데 전체 F를 $O(|S|)$에 구할 수 있는 방법이 있다.

S									
	A	B	A	B	A	C	A	B	A

$F[3] = 2$				
	A	B	A	B

- F[0]부터 F[8]까지 차례로 구한다고 하자. 현재의 F[x]를 구할 때 이전의 F 값들을 이용할 수는 없을까?

0x01 KMP - 실패함수

S	A	B	A	B	A	C	A	B	A

F[3] = 2	A	B	A	B

F[4] = ?	A	B	A	B	A

- F[4]를 구해보자. S[2]와 S[4]가 같다면 F[4] = F[3]+1인 것 같다.
- F[4] ≥ F[3]+1임은 자명한데, 같다는 것은 어떻게 보일 수 있을까?

0x01 KMP - 실패함수

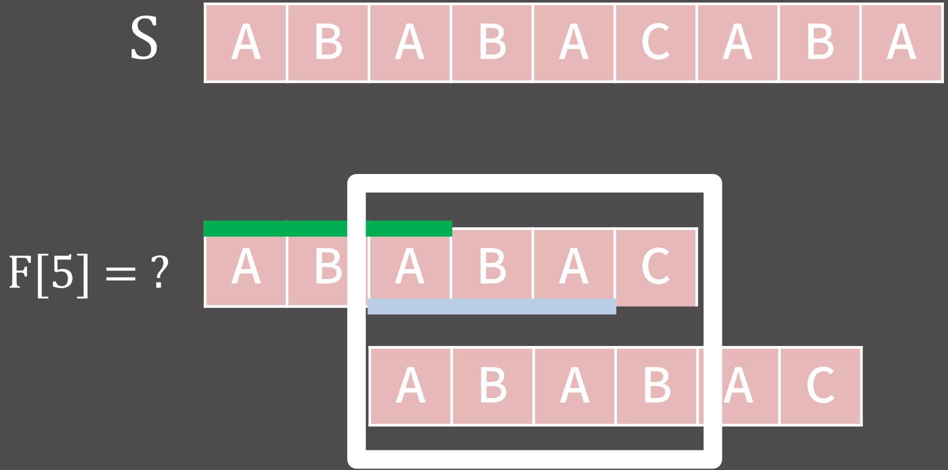
S	A	B	A	B	A	C	A	B	A

F[4] = 3	A	B	A	B	A

F[5] = ?	A	B	A	B	A	C

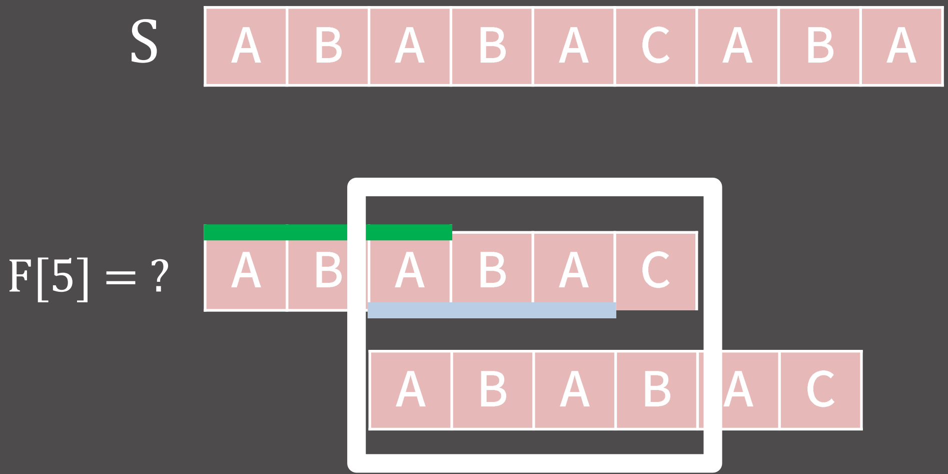
- F[5]를 구해보자. S[3]과 S[5]가 다르다. 어떻게 해야할까?
- F[5]는 4보다 작음이 확실하다.

0x01 KMP - 실패함수



- F[5]가 4인지 확인하는 것은 곧 S[2]부터 시작한 문자열이 S[0]부터 시작한 문자열과 일치하는지 확인하는 것이다.

0x01 KMP - 실패함수



- F[5]가 4가 아님을 알았으니 시작점을 S[2]보다 오른쪽으로 옮겨야 한다. 시작점을 어디로 옮겨야 할까?

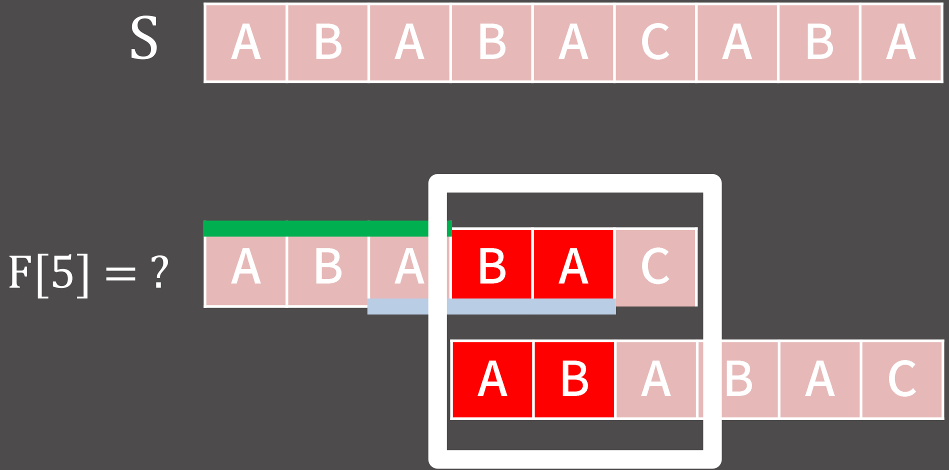
0x01 KMP - 실패함수

S	A	B	A	B	A	C	A	B	A

$F[3] = 1$	A	B	A

- F를 차례로 구한다고 했으니 $F[5]$를 계산하기 전에 이미 $F[3] = 1$임을 알고 있고, 이 말은 곧 $S[0{:}2] \neq S[3{:}5]$임을 의미한다.

0x01 KMP - 실패함수



- 우리는 F[3] = 1임을 알 수 있고, 이 말은 곧 S[0:2] ≠ S[3:5]임을 의미한다.
- 그런데 시작점이 S[3]이 되려면 적어도 S[0:2] = S[3:5]가 성립되어야 한다. 그러므로 시작점은 S[3]일 수 없다.

0x01 KMP - 실패함수

A	B	A	B	A	C	A	B	A

F	0								

0x01 KMP - 실패함수

$i = 1$

A	B	A	B	A	C	A	B	A

$j = 0$

A	B	A	B	A	C	A	B	A

F

0	0							

0x01 KMP - 실패함수

$i = 2$

A	B	A	B	A	C	A	B	A

$j = 0$

A	B	A	B	A	C	A	B	A

F

0	0	1						

0x01 KMP - 실패함수

i = 3

A	B	A	B	A	C	A	B	A

j = 1

A	B	A	B	A	C	A	B	A

F

0	0	1	2					

0x01 KMP - 실패함수

$i = 4$

A	B	A	B	A	C	A	B	A

$j = 2$

A	B	A	B	A	C	A	B	A

F

0	0	1	2	3				

0x01 KMP - 실패함수

i = 5

A	B	A	B	A	C	A	B	A

j = 3

A	B	A	B	A	C	A	B	A

F

0	0	1	2	3	4?			

0x01 KMP - 실패함수

$i = 5$

A	B	A	B	A	C	A	B	A

$j = 1$

A	B	A	B	A	C	A	B	A

($F[2] = 1$임을 이용)

F									
	0	0	1	2	3	2?			

0x01 KMP - 실패함수

$i = 5$

A	B	A	B	A	C	A	B	A

$j = 0$

A	B	A	B	A	C	A	B	A

($F[1] = 0$임을 이용)

F

0	0	1	2	3	0			

0x01 KMP - 실패함수

i = 6

A	B	A	B	A	C	A	B	A

j = 0

A	B	A	B	A	C	A	B	A

F

0	0	1	2	3	0	1		

0x01 KMP - 실패함수

i = 7

A	B	A	B	A	C	A	B	A

j = 1

A	B	A	B	A	C	A	B	A

F

0	0	1	2	3	0	1	2	

0x01 KMP - 실패함수

$i = 8$

A	B	A	B	A	C	A	B	A

$j = 2$

A	B	A	B	A	C	A	B	C

F

0	0	1	2	3	0	1	2	3

0x01 KMP - 실패함수

- 매번 비교가 일어날 때 마다 i가 1 증가하거나 밑의 문자열이 오른쪽으로 이동하므로 시간복잡도는 O(|S|)

```
vector<int> failure(string& S){
  vector<int> f(S.size());
  int j = 0;
  for(int i = 1; i < S.size(); i++){
    while(j > 0 && S[i] != S[j]) j = f[j-1];
    if(S[i] == S[j]) f[i] = ++j;
  }
  return f;
}
```

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

A	A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D
B	A	B	A	B	A	C	A	B	A													

실패 함수 :	0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 0$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 0$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 1$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 1$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 2$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 2$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 3

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 3

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 4$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 4$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 5$	A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 5$	A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

- 여기서 밑의 문자열을 얼마나 밀어야할지(= j를 얼마로 바꿔야할지) 알겠는가?

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 5$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 3$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 6

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 4

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 7$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 5$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 7

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 3

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 8

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 4

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 9$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 5$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 9$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 3$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 10

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 4

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 11

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 5

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 12$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 6$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 13$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 7$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

★

i = 14

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 8

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 15

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 3

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

$i = 20$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 8$

A	B	A	B	A	C	A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

(중략)

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

★ ★

$i = 21$

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

$j = 3$

A	B	A	B	A	C

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 21

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 1

A	B	A	B

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수를 이용해 A 안에 B가 들어있는 모든 위치를 찾는 문제를 |A| + |B| 에 해결해보자.

i = 21

A	B	A	B	A	B	A	B	A	B	A	C	A	B	A	B	A	C	A	B	A	D

j = 0

A	B	A

실패 함수 :

0	0	1	2	3	0	1	2	3

0x01 KMP

- 실패함수에서의 시간복잡도와 비슷하게 매번 비교가 일어날 때 마다 i가 1 증가하거나 밑의 문자열이 오른쪽으로 이동하므로 시간복잡도는 O($|S|$)
- 실패함수를 찾을 때와 코드의 흐름이 거의 동일
- BOJ 1786번 : 찾기 문제를 해결하는 KMP 코드 : http://boj.kr/0f6ccbe50dc14c47a364912fb56eb42e

0x02 라빈 카프

- 라빈 카프는 문자열에서 쓸 수 있는 해쉬 함수이다.
- 적절한 전처리를 통해 부분문자열의 해쉬 값을 바로 알 수 있다는 장점이 있지만 잘 써먹으려면 많은 정수론 지식이 필요하다.

0x02 라빈 카프 – 사전 지식

- $ab \equiv 1 \pmod{m}$ 일 때 b를 a의 곱셈에 대한 역수라고 하고, $b=a^{-1}$ 으로 표시한다.
- 소수 p와 1이상 p-1 이하의 임의의 정수 a에 대해 $a^{p-1} \equiv 1 \pmod{p}$ 이다.
  ex) $3^6 = 729 \equiv 1 \pmod{7}$, $2^{10} = 1024 \equiv 1 \pmod{11}$
- 위의 정리에 따라 mod p에서 1이상 p-1 이하의 임의의 정수 a에 대해 $a^{-1} = a^{p-2}$이다.
- 어떤 1이상 p-1 이하의 임의의 정수 a에 대해 $a^0, a^1, a^2, a^3, \ldots, a^{p-2}$을 p로 나눈 나머지가 모두 다르다면 a를 p의 원시근이라고 한다.
- ex1) $2(=2^1)$와 $16(=2^4)$은 7로 나눈 나머지가 동일하므로 2는 7의 원시근이 아니다.
- ex2) $3^0, 3^1, 3^2, 3^3, 3^4, 3^5$ 는 7로 나눈 나머지가 각각 1, 3, 2, 6, 4, 5로 모두 다르므로 3은 7의 원시근이다.

0x02 라빈 카프 – 사전 지식

- 라빈 카프 알고리즘을 사용하기 위해서는 적당히 큰 소수 p와 $1 < a < p$인 a를 정해야 한다. 이 때 a는 p의 원시근이면서 너무 작지 않은(> 200) 수인 것이 좋다.
- 라빈 카프 알고리즘에서 문자열 S에 대한 해쉬 값은 아래와 같다. ($n = \text{len}(S)$)
- $H = S[0] \times a^{n-1} + S[1] \times a^{n-2} + S[2] \times a^{n-3} + \cdots + S[n-1] \times a^{0} \pmod{p}$
- ex) $a = 2,\ p = 307$일 때 “ABCD”의 해쉬 값 $= 65 \times 2^3 + 66 \times 2^2 + 67 \times 2^1 + 68 \times 2^0$ $= 986 \equiv 65 \pmod{307}$
- 두 문자열 A, B의 해쉬 값이 다르면 애초에 다른 문자열이므로 일단 해쉬 값을 가지고 같은 문자열일 수 있는지 없는지를 걸러낼 수 있다. 더 나아가 만약 p가 굉장히 클 경우, 두 문자열 A, B의 해쉬 값이 동일하다면 굉장히 높은 확률로 $A = B$이다.

0x02 라빈 카프 – 예시

- BOJ 1786번 : 찾기 문제를 라빈 카프 알고리즘으로 해결해보자.

T	B	B	D	B	C	A	B	C	A	B	C	A

P	A	B	C	A

- Step 1) P의 해쉬 값을 구한다.
- Step 2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

0x02 라빈 카프 – 예시

Step 1) P의 해쉬 값을 구한다.$(a = 5, p = 509$로 두자)

T	B	B	D	B	C	A	B	C	A	B	C	A

P	A	B	C	A

- $H(P) = P[0]\times5^3 + P[1]\times5^2 + P[2]\times5^1 + P[3]\times5^0$
  $= 65\times5^3 + 66\times5^2 + 67\times5^1 + 65\times5^0 = 504 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T											
B	B	D	B	C	A	B	C	A	B	C	A

$H(P) = 504$

- $H(\,T[0{:}4]\,) = T[0]\times5^3 + T[1]\times5^2 + T[2]\times5^1 + T[3]\times5^0$

  $= 66\times5^3 + 66\times5^2 + 68\times5^1 + 66\times5^0 = 126 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T	B	B	D	B	C	A	B	C	A	B	C	A

$H(P) = 504$

- $H(\,T[0{:}4]\,) = T[0]\times5^3 + T[1]\times5^2 + T[2]\times5^1 + T[3]\times5^0$
  $= 66\times5^3 + 66\times5^2 + 68\times5^1 + 66\times5^0 = 126 \pmod{509}$
- $H(\,T[1{:}5]\,) = T[1]\times5^3 + T[2]\times5^2 + T[3]\times5^1 + T[4]\times5^0$
  $= 66\times5^3 + 68\times5^2 + 66\times5^1 + 67\times5^0 = 167 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T												
	B	B	D	B	C	A	B	C	A	B	C	A

$H(P) = 504$

- $H(\ T[0:4]\ ) = T[0]\times5^3 + T[1]\times5^2 + T[2]\times5^1 + T[3]\times5^0$
  $= 66\times5^3 + 66\times5^2 + 68\times5^1 + 66\times5^0 = 126 \ (\mathrm{mod}\ 509)$

- $H(\ T[1:5]\ ) = T[1]\times5^3 + T[2]\times5^2 + T[3]\times5^1 + T[4]\times5^0$
  $= 66\times5^3 + 68\times5^2 + 66\times5^1 + 67\times5^0 = 167 \ (\mathrm{mod}\ 509)$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T												
	B	B	D	B	C	A	B	C	A	B	C	A

$H(P) = 504$

- $H(\,T[0:4]\,) = T[0]\times5^3 + T[1]\times5^2 + T[2]\times5^1 + T[3]\times5^0$
  $= 66\times5^3 + 66\times5^2 + 68\times5^1 + 66\times5^0 = 126 \pmod{509}$

- $H(\,T[1:5]\,) = T[1]\times5^3 + T[2]\times5^2 + T[3]\times5^1 + T[4]\times5^0$
  $= 5(T[1]\times5^2 + T[2]\times5^1 + T[3]\times5^0) + T[4]\times5^0$
  $= 5(H(\,T[0:4]\,) - T[0]\times5^3\,) + T[4]\times5^0$
  $= 5(126 - 66\times5^3) + 67\times5^0 = 167 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T	B	B	D	B	C	A	B	C	A	B	C	A

H(P) = 504

- $H(\ T[2:6]\ ) = 5(H(\ T[1:5]\ ) - T[1]\times 5^3\ ) + T[5]\times 5^0$

  $= 5(167 - 66\times 5^3) + 65\times 5^0 = 370 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T	B	B	D	B	C	A	B	C	A	B	C	A

$H(P) = 504$

- $H(\ T[3:7]\ ) = 5(H(\ T[2:6]\ ) - T[2]\times 5^3\ ) + T[6]\times 5^0$

  $= 5(370 - 68\times 5^3) + 66\times 5^0 = 136 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T												
	B	B	D	B	C	A	B	C	A	B	C	A

$H(P) = 504$

- $H(\ T[4:8]\ ) = 5(H(\ T[3:7]\ ) - T[3] \times 5^3\ ) + T[7] \times 5^0$

  $= 5(136 - 66 \times 5^3) + 67 \times 5^0 = 217 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T	B	B	D	B	C	A	B	C	A	B	C	A

$H(P) = 504$

- $H(\ T[5:9]\ ) = 5(H(\ T[4:8]\ ) - T[4]\times5^3\ ) + T[8]\times5^0$

  $= 5(217 - 67\times5^3) + 65\times5^0 = 504 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T											
B	B	D	B	C	A	B	C	A	B	C	A

H(P) = 504

- $H(\ T[6:10]\ ) = 5(H(\ T[5:9]\ ) - T[5]\times5^3\ ) + T[9]\times5^0$

  $= 5(504 - 65\times5^3) + 66\times5^0 = 136 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T

B	B	D	B	C	A	B	C	A	B	C	A

$H(P) = 504$

- $H(\ T[7:11]\ ) = 5(H(\ T[6:10]\ ) - T[5]\times 5^3\ ) + T[9]\times 5^0$

  $= 5(136 - 66\times 5^3) + 67\times 5^0 = 217 \pmod{509}$

0x02 라빈 카프 – 예시

Step2) T[0:4], T[1:5], T[2:6], ..., T[8:12]의 해쉬 값을 구한다. 이 때 해쉬 값이 일치하면 동일한 문자열로 간주한다.

T	B	B	D	B	C	A	B	C	A	B	C	A

H(P) = 504

★ ★

- $H(\ T[8:12]\ ) = 5(H(\ T[7:11]\ ) - T[6] \times 5^3\ ) + T[10] \times 5^0$

  $= 5(136 - 67 \times 5^3) + 65 \times 5^0 = 504 \pmod{509}$

0x02 라빈 카프 – 올바른 사용법

- Q) 해쉬 값이 같지만 실제 문자열은 다른 경우가 있을 수도 있지 않나요?
- A) 맞습니다. 정확한 답을 내기 위해서는 일단 해쉬 값이 일치한다면, 실제로 두 문자열이 동일한지를 비교하는 루틴이 추가되어야 합니다. 그러나 이 문제에서 실제로 두 문자열이 동일한지 비교하는 루틴을 추가하면 시간복잡도가 최악의 경우 $O(500000^2)$가 됩니다. 그러므로 시간 절약을 위해 틀릴 가능성을 감수하더라도 실제로 두 문자열이 동일한지 비교하지 않습니다.

- p가 10^9 정도의 소수일 경우 답이 틀리지 않을 확률 : $(1-10^{-9})^{1000000} ≒ 0.9990005$
- a가 너무 작을 경우 해쉬 충돌 쌍이 쉽게 찾아진다.($a = 2$일 때 H(“AC”) = H(“BA”))

0x02 라빈 카프 – 올바른 사용법

- a가 p의 원시근이 아닐 경우 a, p가 크더라도 충돌쌍이 쉽게 찾아질 수 있다.
- $a = 3002, p = 8191$ 일 때 H(“AAAAAB”) = H(“BAAAAA”) 이다. ($a^5 = 1$이기 때문)
- 그런데 원시근을 빠르게 구할 방법이 없다. 그러니 미리 a, p를 외워가거나 그냥 운에 맡겨야 한다.
- BOJ 1786번 : 찾기 문제를 해결하는 라빈 카프 코드($a = 302, p = 1000000007$)
  http://boj.kr/af68b5012afe40d1b5ea71a880dc419e

0x02 라빈 카프 – 응용

- 지금은 문자열 S에서 길이가 k로 고정된 부분문자열의 해쉬값만 $O(|S|)$에 구할 수 있다.
- $O(|S|)$의 전처리를 거친 후에 임의의 x, y에 대해 $H(S[x:y])$를 $O(1)$에 구할 수 있는 방법이 있다.

0x02 라빈 카프 – 응용

- Prefix Sum 기법을 이용한다.

S	S[0]	S[1]	S[2]	S[3]	S[4]	S[5]

Psum	0						

0x02 라빈 카프 – 응용

- Prefix Sum 기법을 이용한다.

S	S[0]	S[1]	S[2]	S[3]	S[4]	S[5]

Psum	0	$S[0]a^6$					

$Psum[0] + S[0]a^6$

0x02 라빈 카프 – 응용

- Prefix Sum 기법을 이용한다.

S	S[0]	S[1]	S[2]	S[3]	S[4]	S[5]

Psum	0	$S[0]a^6$	$S[0]a^6$ $+S[1]a^5$				

$Psum[1] + S[1]a^5$

0x02 라빈 카프 – 응용

- Prefix Sum 기법을 이용한다.

S	S[0]	S[1]	S[2]	S[3]	S[4]	S[5]

Psum	0	$S[0]a^6$	$S[0]a^6$ $+S[1]a^5$	$S[0]a^6$ $+S[1]a^5$ $+S[2]a^4$			

$$Psum[2] + S[2]a^4$$

0x02 라빈 카프 – 응용

- Prefix Sum 기법을 이용한다.

S	S[0]	S[1]	S[2]	S[3]	S[4]	S[5]	
Psum	0	S[0]a^6	S[0]a^6 +S[1]a^5	S[0]a^6 +S[1]a^5 +S[2]a^4	.	.	S[0]a^6 +S[1]a^5 +S[2]a^5 … +S[5]a^0

Psum[5] + S[5]a^0

0x02 라빈 카프 – 응용

- $H(S[x:y]) = S[x]a^{y-x-1} + S[x+1]a^{y-x-2} + \cdots + S[y-1]a^{0} = a^{-(|S|-y)}(Psum[y]-Psum[x])$
- $a^{-1} = a^{p-2}$ 으로 계산 가능

강의 정리

- KMP, 라빈 카프에 대해 배웠다.
- 응용해서 나올 경우 손절하면 된다ㅠ