

Lecture 12

스트링 알고리즘 (1)

컴퓨터과학과 | 김진욱 교수

학습목차

1 | 기본 개념

2 | 라빈-카프 알고리즘

3 | KMP 알고리즘

01.

기본 개념

▶ 스트링 string?

- 문자가 연속적으로 나열된 문자열
 - ✓ (예) ATATCGCCCCACGTAT
 - ✓ (예) 001011010001110101

▶ 알파벳 alphabet Σ

- 스트링에 사용되는 문자들의 집합
 - ✓ DNA 서열 $\rightarrow \Sigma = \{A, C, G, T\}$
 - ✓ 이진 데이터 $\rightarrow \Sigma = \{0, 1\}$

▶ 스트링에 대한 다양한 문제를 해결하는 알고리즘을 통칭

- 스트링 매칭
- 스트링 압축
- 최장 공통 부분 수열
- 최장 반복 서브스트링
- 최장 회문 서브스트링
- 접미부 트리
- 접미부 배열
- ...

스트링 매칭?

▶ 텍스트에서 패턴이 나타나는 위치를 찾는 것

- 텍스트 $T = t_0 t_1 t_2 \cdots t_{n-1}$ → 긴 스트링, 길이 n
- 패턴 $P = p_0 p_1 p_2 \cdots p_{m-1}$ → 짧은 스트링, 길이 m

$$n \geq m$$

위치 0 1 2 3 4 5 6 7 8

$T =$ a a b a a b a a a

$P =$ a a b a a

► Brute-force algorithm 또는 Naïve algorithm

- 텍스트의 각 위치에서부터 패턴의 길이만큼 문자를 비교하며 매치를 찾는 방법

위치 0 1 2 3 4 5 6 7 8
T = a a b a a b a a a
 || || || || ||
P = a a b a a

► Brute-force algorithm 또는 Naïve algorithm

- 텍스트의 각 위치에서부터 패턴의 길이만큼 문자를 비교하며 매치를 찾는 방법

위치 0 1 2 3 4 5 6 7 8

T = a a b a a b a a a

|| ✗

P = a a b a a

▶ Brute-force algorithm 또는 Naïve algorithm

- 텍스트의 각 위치에서부터 패턴의 길이만큼 문자를 비교하며 매치를 찾는 방법

위치 0 1 2 3 4 5 6 7 8
T = a a b a a b a a a
P = a a b a a

► Brute-force algorithm 또는 Naïve algorithm

- 텍스트의 각 위치에서부터 패턴의 길이만큼 문자를 비교하며 매치를 찾는 방법

위치 0 1 2 3 4 5 6 7 8
T = a a b a a b a a a
 || || || || ||
P = a a b a a

► Brute-force algorithm 또는 Naïve algorithm

- 텍스트의 각 위치에서부터 패턴의 길이만큼 문자를 비교하며 매치를 찾는 방법

위치 0 1 2 3 4 5 6 7 8

T = a a b a a b a a a

 || ✕
P = a a b a a

브루트-포스 스트링 매칭 알고리즘

01 | 기본 개념

```
BruteForce (n, T[ ], m, P[ ])
```

```
{  
  for (i = 0; i <= n-m; i++) { // 텍스트의 가능한 모든 위치에서  
    flag = true;  
    for (j = 0; j < m; j++) // 패턴의 길이만큼  
      if (T[i+j] != P[j]) { // 텍스트와 패턴의 문자를 비교  
        flag = false; // 일치하지 않으면 표시 지움  
        break;  
      }  
    if (flag) 위치 i 출력; // 표시가 남아있으면 매치  
  }  
}
```

$O(n)$

$O(m)$

$O(nm)$

▶ 패턴을 전처리

- 라빈-카프 알고리즘
- KMP 알고리즘
- 보이어-무어 알고리즘

▶ 텍스트를 전처리

- 접미부 트리
- 접미부 배열

02.

라빈-카프 알고리즘

▶ 패턴의 **해시값**으로 매치의 후보를 찾고,
후보에 대해서만 문자별로 비교해서 매치를 찾는 방법

- 해시 함수: 문자열을 위한 가중 합 $h(S) = (\sum_{i=0}^{m-1} s'_i |\Sigma|^{m-1-i}) \bmod M$

✓ (예) $\Sigma = \{a, b, c, d, e, f, g, \dots, x, y, z\}$, $M=101$

P = a a b a a

$h(\text{aabaa})$

$$= (0 \times 26^4 + 0 \times 26^3 + 1 \times 26^2 + 0 \times 26^1 + 0 \times 26^0) \bmod 101$$

$$= 70$$

▶ 패턴의 **해시값**으로 매치의 후보를 찾고,
후보에 대해서만 문자별로 비교해서 매치를 찾는 방법

- 해시 함수: 문자열을 위한 가중 합 $h(S) = (\sum_{i=0}^{m-1} s'_i |\Sigma|^{m-1-i}) \bmod M$

위치 0 1 2 3 4 5 6 7 8

T = a a b a a b a a a

70 3 78 70 2

70

P = a a b a a

▶ 패턴의 **해시값**으로 매치의 후보를 찾고,
후보에 대해서만 문자별로 비교해서 매치를 찾는 방법

- 해시 함수: 문자열을 위한 가중 합 $h(S) = (\sum_{i=0}^{m-1} s'_i |\Sigma|^{m-1-i}) \bmod M$

위치 0 1 2 3 4 5 6 7 8

T = a a b a a b a a a

P = a a b a a

▶ 패턴의 **해시값**으로 매치의 후보를 찾고,
후보에 대해서만 문자별로 비교해서 매치를 찾는 방법

- 해시 함수: 문자열을 위한 가중 합 $h(S) = \left(\sum_{i=0}^{m-1} s'_i |\Sigma|^{m-1-i}\right) \bmod M$

위치 0 1 2 3 4 5 6 7 8

T = a a b a a b a a a

|| || || || ||
P = a a b a a

텍스트 위치별 해시값 계산

02 | 라빈-카프 알고리즘

위치 0 1 2 3 4 5 6 7 8
T = a a b a a b a a a
70 3 78 70 2

▶ 위치 0: O(m)

$$h(aabaa) = (0 \times 26^4 + 0 \times 26^3 + 1 \times 26^2 + 0 \times 26^1 + 0 \times 26^0) \bmod 101 = 70$$

▶ 위치 1~n-m: O(1)

- 해시 함수의 특징을 이용

$$h(abaab) = (0 \times 26^4 + 1 \times 26^3 + 0 \times 26^2 + 0 \times 26^1 + 1 \times 26^0) \bmod 101$$

$$= (26 \times h(aabaa) - 0 \times 26^5 + 1 \times 26^0) \bmod 101$$

$$= (26 \times 70 - 0 + 1) \bmod 101 = 3$$

× 26

라빈-카프 Rabin-Karp 알고리즘

02 | 라빈-카프 알고리즘

```
RabinKarp (n, T[ ], m, P[ ])
{
    int hp = 0, ht = 0;
    int dm = pow(26, m) % M;
    for (j = 0; j < m; j++) {
        hp = (hp * 26 + P[j] - 97) % M;
        ht = (ht * 26 + T[j] - 97) % M;
    }
```

패턴의 해시값

텍스트 위치 0 해시값

```
for (i = 0; i <= n-m; i++) {
    if (hp == ht) { // 해시값 일치하는 위치에 대해
        flag = true;
        for (j = 0; j < m; j++) // 패턴의 길이만큼
            if (T[i+j] != P[j]) { // 텍스트와 패턴의 문자를 비교
                flag = false; // 일치하지 않으면 표시 지움
                break;
            }
        if (flag) 위치 i 출력; // 표시가 남아있으면 매치
    }
    if (i < n-m) // 다음 위치 해시값 계산
        ht = (ht*26 - dm*(T[i]-97) + (T[i+m]-97)) % M;
}
}
```

▶ 텍스트 $T=10011100$ 에서 패턴 $P=0011$ 이 매치되는 모든 위치?

- $\Sigma=\{0, 1\}$, $M=11$

$$h(1001) = (1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) \bmod 11 = 9$$

위치 0 1 2 3 4 5 6 7

$T = 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0$

9 3 7 3 1
|| || || ||

3

$P = 0\ 0\ 1\ 1$

$h(0011)$

$$= (2 \times 9 - 1 \times 2^4 + 1 \times 2^0) \bmod 11 = 3$$

$h(0111)$

$$= (2 \times 3 - 0 \times 2^4 + 1 \times 2^0) \bmod 11 = 7$$

$h(1110)$

$$= (2 \times 7 - 0 \times 2^4 + 0 \times 2^0) \bmod 11 = 3$$

$$h(0011) = (0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \bmod 11 = 3$$

성과와 특징

02 | 라빈-카프 알고리즘

▶ 성능 $\rightarrow O(n+km)$

- 전처리 $\rightarrow O(m)$
- 텍스트에서 해시값 계산 $\rightarrow O(n)$
- 후보 위치는 문자 직접 비교, 매치 개수 $k \rightarrow O(km)$

▶ 매치 개수가 상수 $\rightarrow O(n)$

▶ 모든 위치에서 매치 $\rightarrow O(nm)$

03.

KMP 알고리즘

▶ Knuth-Morris-Pratt 알고리즘

- 패턴 내의 문자들의 관계를 이용하여 매칭 시 중복된 비교를 줄임
- 텍스트의 첫 위치에서 패턴의 앞부분부터 문자 비교

위치 0 1 2 3 4 5 6 7 8

T = a a b a a b a a a

P = a a b a a

a a b a a

a a b a a

a a b a a

a a b a a

▶ Knuth-Morris-Pratt 알고리즘

- 패턴 내의 문자들의 관계를 이용하여 매칭 시 중복된 비교를 줄임
- 텍스트의 첫 위치에서 패턴의 앞부분부터 문자 비교

위치 0 1 2 3 4 5 6 7 8

T = a a b a b b a a a

P = a a b a a

a a b a a

a a b a a

a a b a a

a a b a a

a a b a a

▶ 일치한 서브스트링에 대한 접두부와 접미부의 최대 일치 정보

- $f_i \rightarrow$ 패턴의 서브스트링 $p_0 p_1 \cdots p_i$ 에서 최대 일치인 접두부의 끝 문자 위치
- ✓ 접두부와 접미부의 최대 일치가 없으면 $-1 \rightarrow f_0 = -1$

위치 0 1 2 3 4 $f_4 = 1$

P = a a b a a

a a b a a

1

위치 0 1 2 3 4 $f_2 = -1$

P = a a b a a

a a b a a

-1

위치 0 1 2 3 4 $f_3 = 0$

P = a a b a a

a a b a a

0

PreKMP (m, P[])

{

int F[m], idx = -1;

F[0] = -1;

for (i=1; i<m; i++) {

while (idx >= 0 && P[i] != P[idx+1])

idx = F[idx]; // 최대 접두부 찾기

if (P[i] == P[idx+1])

idx++; // 마지막 문자 일치

F[i] = idx; // i에서의 최대 접두부 설정

}

return (F);

}

패턴의 위치 0부터 m-1까지 차례대로 F[i], 즉 최대 일치 정보 f_i 구함

	idx		i			
	↓		↓			
	-1	0	1	2	3	4
P =		a	a	b	a	a
F		-1	0	-1	0	1

```
KMP (n, T[ ], m, P[ ])
{
    F[0..m-1] = PreKMP (m, P); // 패턴의 전처리
    inf j = -1;
    for (i=0; i < n; i++) {
        while (j >= 0 && T[i] != P[j+1]) j = F[j];
        if (T[i] == P[j+1]) j++;
        if (j == m-1) { // 패턴의 마지막 문자까지 일치하면
            위치 i-j 출력; // 매치 발견
            j = F[j];
        }
    }
}
```

▶ 텍스트 $T = \text{aabaabaaa}$ 에서 패턴 $P = \text{aabaa}$ 가 매치되는 모든 위치?

위치 \downarrow
 0 1 2 3 4 5 6 7 8
 $T =$ **a a b a a b a a a**
 || || || || || || || ||
 $P =$ a a b a a

\downarrow
 0 1 2 3 4
 $P =$ a a b a a
 F -1 0 -1 0 1

▶ 텍스트 $T = \text{ATATATGATATGAA}$ 에서 패턴 $P = \text{ATATGAT}$ 가 매치되는 모든 위치?

↓
위치 0 1 2 3 4 5 6 7 8 9 10 11 12 13
 $T = \text{A T A T A T G A T A T G A A}$
|| || || || ✕ || || || || || || || ✕
 $P = \text{A T A T G A T}$

↓
0 1 2 3 4 5 6
 $P = \text{A T A T G A T}$
F -1 -1 0 1 -1 0 1

성과와 특징

03 | KMP 알고리즘

▶ 전처리 $\rightarrow O(m)$

- idx 값은 최대 $m-1$ 증가
- while 문은 최대 $m-1$ 만큼 수행

```
for (i=1; i<m; i++) {  
    while (idx >= 0 && P[i] != P[idx+1]) idx = F[idx];  
    if (P[i] == P[idx+1]) idx++;  
    F[i] = idx;  
}
```

감소

1 증가

▶ 매칭 $\rightarrow O(n)$

- j 값은 최대 n 증가
- while 문은 최대 n 만큼 수행

```
for (i=0; i < n; i++) {  
    while (j >= 0 && T[i] != P[j+1]) j = F[j];  
    if (T[i] == P[j+1]) j++;  
    if (j == m-1) {  
        위치 i-j 출력;  
        j = F[j];  
    }  
}
```

감소

1 증가

감소

▶ $n \geq m \rightarrow$ 전체 성능 $O(n)$

1. 기본 개념

- 스트링 매칭-텍스트(길이 n)에서 패턴(길이 m)이 나타나는 위치를 찾는 문제
- 브루트-포스 스트링 매칭 알고리즘-성능 $O(nm)$

2. 라빈-카프 알고리즘

- 패턴의 해시값으로 매치의 후보를 찾고, 후보에 대해서만 문자별로 비교
- 성능-매칭 $O(n+km)$ (k : 매치의 개수)
- 매치의 개수에 따라 최선 $O(n)$, 최악 $O(nm)$

3. KMP 알고리즘

- 패턴 내의 문자들의 관계를 이용하여 매칭 시 중복된 비교를 줄임
- 텍스트의 첫 위치에서 패턴의 앞부분부터 문자 비교
- 전처리-패턴의 각 위치별로 접두부와 접미부의 최대 일치 정보를 구함
- 성능 $O(n)$ -전처리 $O(m)$, 매칭 $O(n)$ (단, $n \geq m$)

다음시간에는

Lecture **13**

스트링 알고리즘 (2)

컴퓨터과학과 | 김진욱 교수