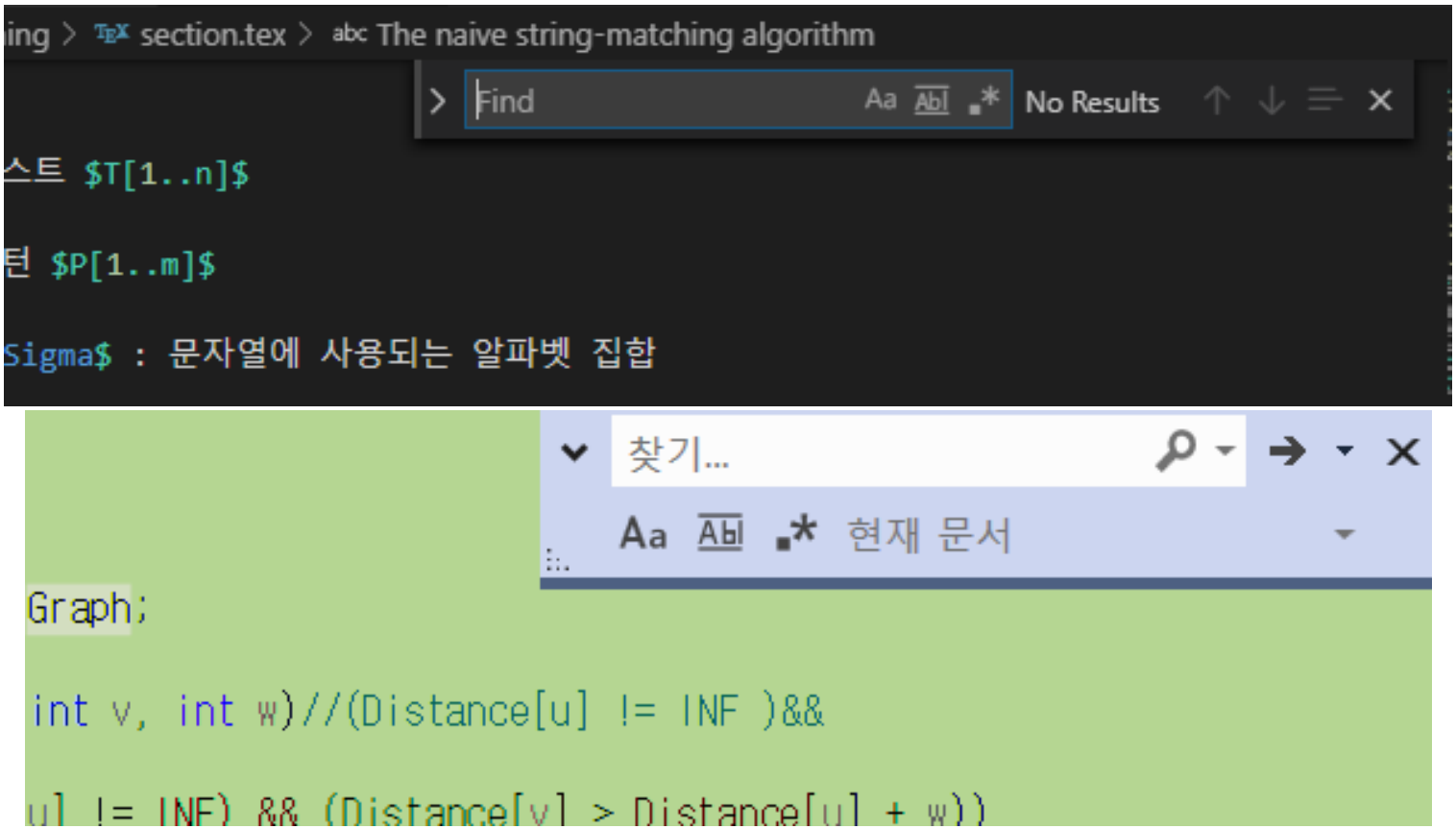
String Matching

EUnS

November 26, 2019

Preface



주먹구구

각 k마다 $T[k]$부터 $T[k+m-1]$까지 하나하나 P와 맞는지 확인하는것이다.

```
1 NAIVE-STRING-MATCHER (T,P)
2     n = T.length
3     m = P.length
4     for s = 0 to n-m
5         if(P[1..m] == T[s+1..s+m])
6             print ``Pattern occurs with shift s"
```

매칭시간 $O((n-m+1)m)$ 좀 더 줄일수 없나?

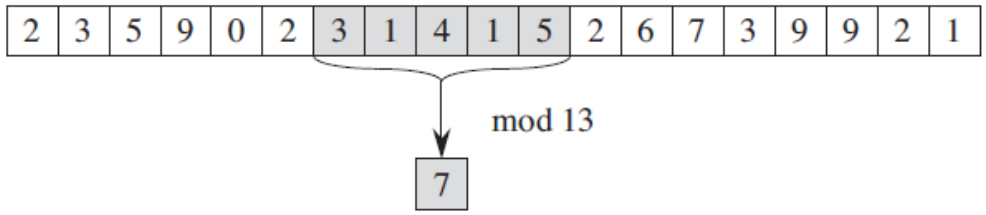
1. The naive string-matching algorithm
2. The Rabin-Karp algorithm
3. String matching with finite automata
4. The KMP algorithm
5. conclusion

약속

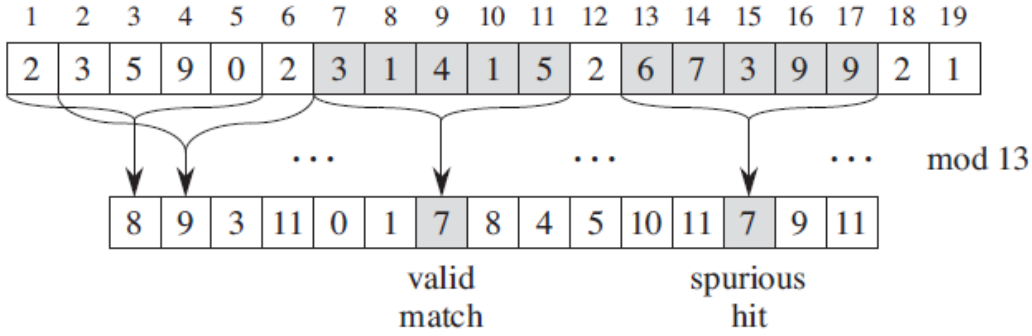
- 텍스트 $T[1..n]$
- 패턴 $P[1..m]$
- Σ : 문자 집합 (영어, 한국어, 중국어..)

idea

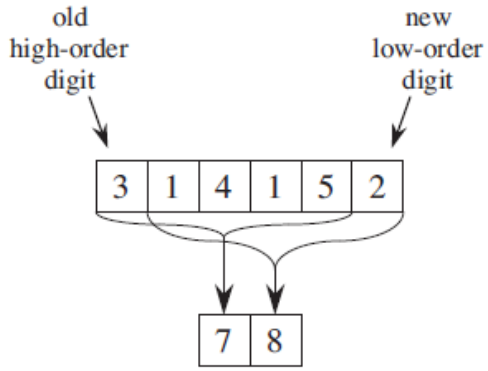
문자를 숫자로 바꾸자!



(a)



(b)



$$14152 \equiv (31415 - 3\cdot 10000)\cdot 10 + 2 \pmod{13}$$
$$\equiv (7 - 3\cdot 3)\cdot 10 + 2 \pmod{13}$$
$$\equiv 8 \pmod{13}$$

(c)

호너의 법칙(Horner's law)

$\Theta(m)$

$$p = P[n]+d(P[n-1]+d(P[n-2]+\cdots+d(P[2]+d(P[1]))\cdots)) \bmod q$$

$$p = (dp + P[i]) \bmod q$$

문자열 $T[s+1..s+m]$에 대한 숫자 t_s의 처리는 매칭과 직후에 계산한다.

$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$$

```
1  RABIN–KARP–MATCHER(T,P,d,q)
2       n = T.length
3       m = P.length
4       h = d^{m–1} mod q
5       p = 0
6       t_0 = 0
7       for( i = 1 to m)
8            p = (dp+P[i])mod q
9            t_0 = (dt_0 + T[i]) mod q
10      for s = 0 to n–m
11           if p == t_s
12                if P[1..m] == T[s+1..s+m]
13                     print ``Pattern occurs with shift s"
14           if s < n – m
15                t_{s+1} = (d(t_s – T[s+1]h) + T[s+m+1]) mod q
```

- 전처리 $\Theta(m)$
- 매칭시간 $O((n-m+1)m)$

아쉬운 점 : 여전히 완벽하게 일치하는지 확인하기 위해 하나하나 비교하는 방법을 사용한다.

정의 (automata)

finite automaton은 다음과 같이 5개의 튜플로 구성된다.

- Q : 유한 상태의 집합
- q_0 :시작 상태 ($q_0 \in Q$)
- A : 받아들이는 상태의 구분된 상태 ($A \subset Q$)
- Σ : 유한 입력 알파벳 집합
- δ : $Q \times \Sigma$에서 Q로 매핑되는 전이 함수 M

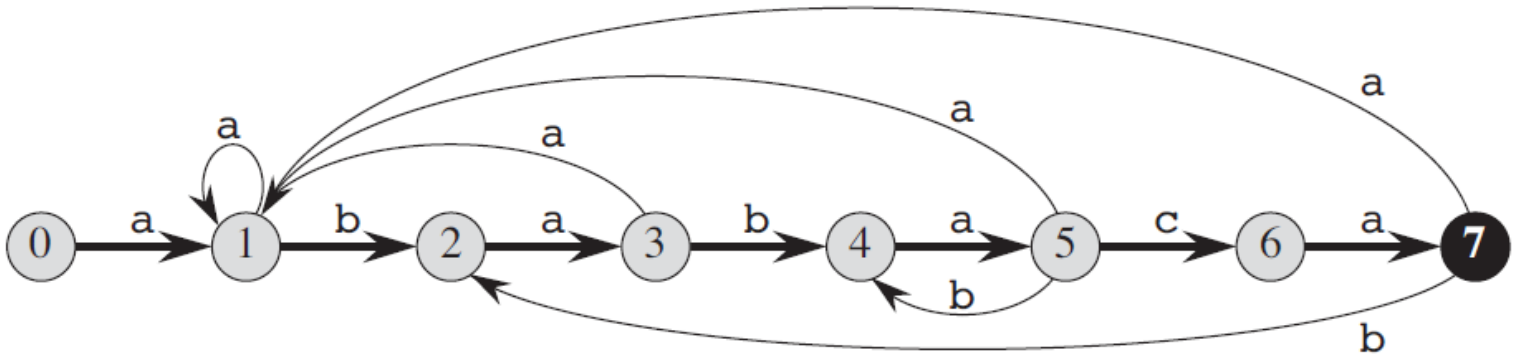
뭔소린지모르겠죠?
저도 그럼
오토마타 내용 통째로 생략

δ 상태표란

- 문자열 T는 각 자리마다 상태를 가진다.
- 초기 시작 상태는 0
- 현재 문자의 상태는 문자의 상태와 현재 문자에 따라 정해진다.
- 상태는 상태표를 보고 정한다.

δ 상태표를 구해라

1. 상태 : 앞에서부터 각각의 문자가 가장 길게 일치하는 갯수
2. $(m+1) \times \Sigma$ 배열
3. 상태가 m에 도달하면 매칭



(a)

state	input a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

(b)

i	—	1	2	3	4	5	6	7	8	9	10
$T[i]$	—	a	b	a	b	a	b	a	c	a	b
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2

(c)

```
1 FINITE_AUTOMATON_MATCHER(T, delta , m)
2 n = T.length
3 q = 0
4 for i = 1 to n
5     q = d(q,T[i])
6     if q == m
7         print ``Pattern occurs with shift i-m"
```

```
1  COMPUTE–TRANSITION–FUNCTION(P, Sigma)
2      m = P.length
3      for q = 0 to m
4          for a in Sigma
5              k = min(m+1,q+2)
6              repeat
7                  k = k - 1
8                  until P_k ] P_q–a
9              d(q,a) = k
10     return d
```

- 전처리의 수행시간 $O(m^3\Sigma)$
- 상태표의 공간 복잡도의 크기가 $\Theta(m|\Sigma|)$
- $\Theta(n)$

- 뒷절의 KMP방식을 차용해서 전처리 시간을 $O(m\Sigma)$로 개선할수있다.
- 공간 복잡도의 크기를 더 줄여보자.

KMP?

- Knuth-Morris-Pratt이 공동 발표한 알고리즘

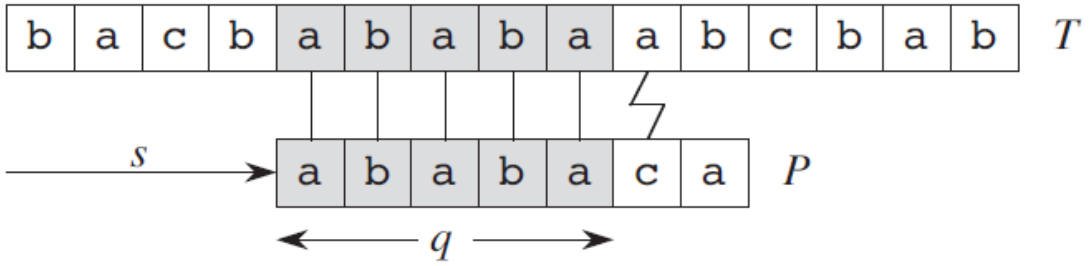
샛길

Donald Knuth

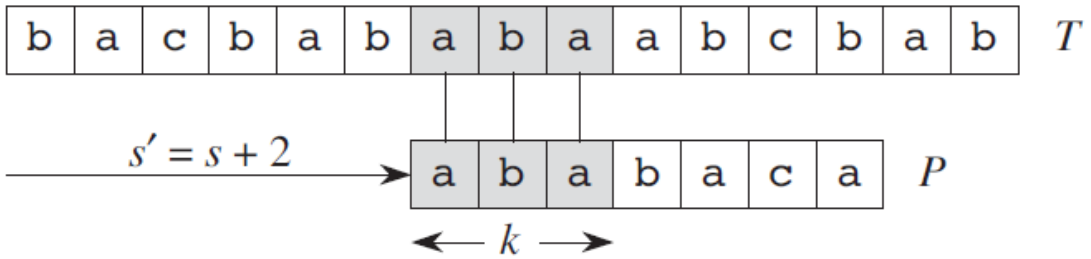
- TAOCP(The art of computer programing)의 저자
  1968에 첫권이나오고 총 7권 계획인데 이제 책이 4권 출시됨
  각종 알고리즘 논문 참고로 빠짐없이 개근상.
- TEX 창시자
- 각종 CS 책에 심심하면 거론되는 인물

δ대신 π

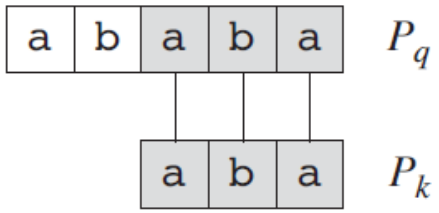
- 크기 m
- 매칭이 실패했을때 사용.
- 앞의 문자열과 가장 근접하게 일치하는 위치를 반환


b a c b a b a b a a b c b a b
T
s
a b a b a c a
P
q

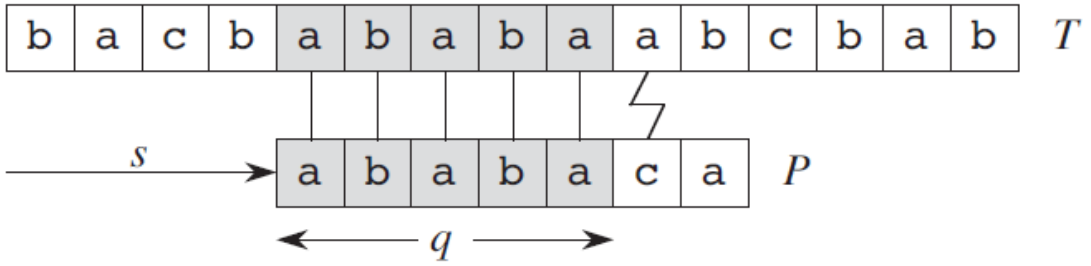
(a)


b a c b a b a b a a b c b a b
T
s′ = s + 2
a b a b a c a
P
k

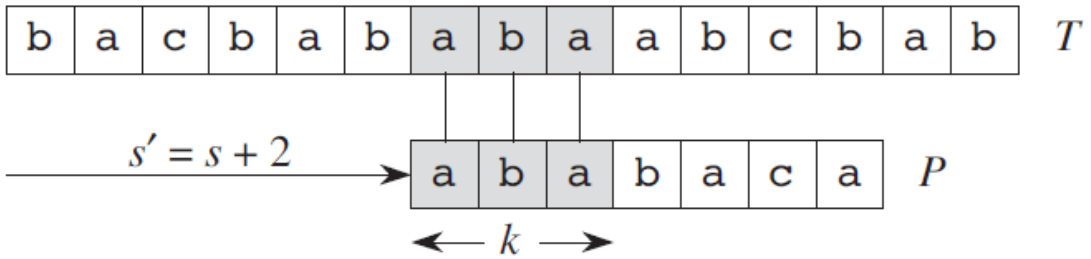
(b)


a b a b a
P_q
a b a
P_k

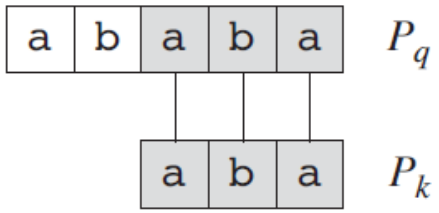
(c)


b a c b a b a b a a b c b a b
T
s
a b a b a c a
P
q

(a)


b a c b a b a b a a b c b a b
T
s′ = s + 2
a b a b a c a
P
k

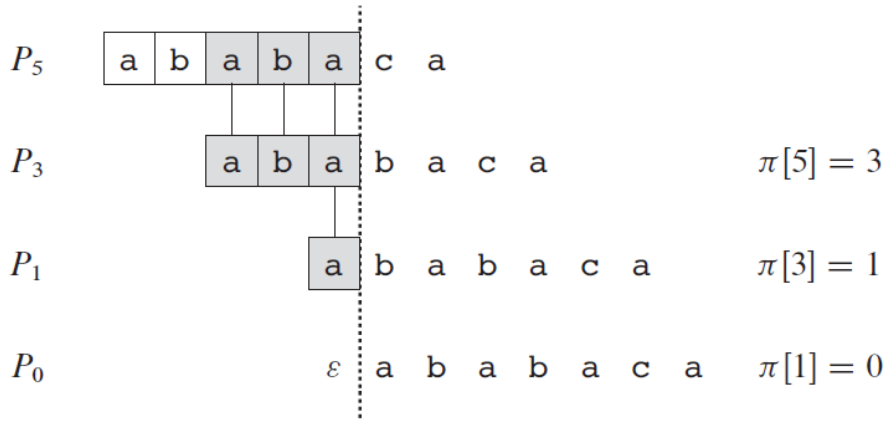
(b)


a b a b a
P_q
a b a
P_k

(c)

i	1	2	3	4	5	6	7
$P[i]$	a	b	a	b	a	c	a
$\pi[i]$	0	0	1	2	3	0	1

(a)



(b)

Figure: (a)는 패턴 P와 전처리한 $\pi[1..7]$ (b)는 $P[5]$에서 다음 문자가 매칭이 틀렸을때 그에 따른 π값과 되돌아가는 순서를 나타낸것이다.

```
1  KMP–MATCHER(T,P)
2       n = T.length
3       m = P:length
4       PI[] = COMPUTE–PREFIX–FUNCTION(P)
5       q = 0 // number of characters matched
6       for i = 1 to n // scan the text from left to right
7            while q>0 and P [q+1] != T[i]
8                 q = PI[q] // next character does not match
9            if P [q+1] == T[i]
10                q = q + 1 // next character matches
11           if q == m // is all of P matched?
12                print ``Pattern occurs with shift" i - m
13                q  = PI[q]
```

```
1  COMPUTE–PREFIX–FUNCTION(P) /
2      m = P.length
3      let PI[1..m] be a new array
4      PI[1] = 0
5      k = 0
6      for q = 2 to m
7          while k>0 and P[k+1] != P[q]
8              k = P[k]
9          if P[k+1] == P[q]
10                 k = k + 1
11         PI[q] = k
12     retunr PI
```

- 전처리 : $O(m)$
- 매칭시간 : $\Theta(n)$

정리

알고리즘	전처리	수행시간	공간복잡도
naive	0	$O((n-m+1)m)$	0
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$	$\Theta(1\|)$
finite automata	$\Theta(m\Sigma)$	$\Theta(n)$	$\Theta(m\|\Sigma\|)$
The KMP algorithm	$\Theta(m)$	$\Theta(n)$	$\Theta(m)$

There's no such thing as a stupid question.