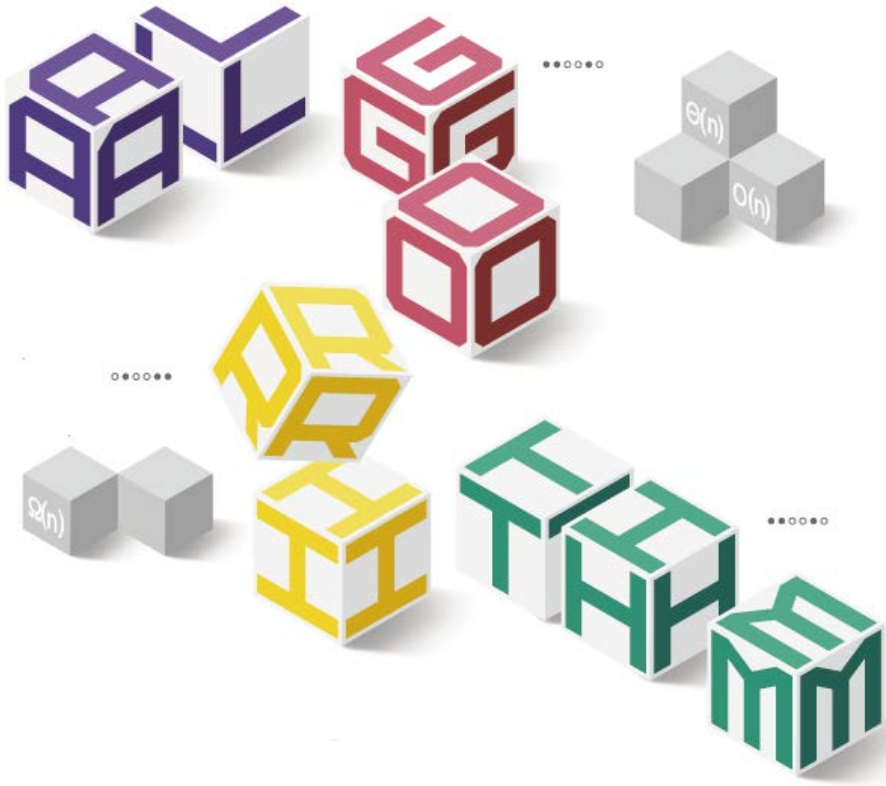


관계 중심의 사고법

쉽게 배우는 알고리즘

12장. 문자열 매칭

12장. 문자열 매칭

새로운 생각이나 새로운 발견이 이루어지는 직관의 도약이 있다.
이렇게 순간적으로 튀어오르는 힘이 유난히 좋은 과학자들이 있다.
특별히 뛰어난 직관력을 가진 과학자들을 보면 마치 종교인들이
말하는 것과 같은 그런 종류의 믿음이 있다.
그들의 마음을 사로잡는
어떤 것이 이끄는 곳으로 흔들림없이
그대로 따라가는 성향은
최정상의 과학자가 흔히 보이는 특성이다.

- 프리초프 카프라

학습목표

- 원시적인 매칭 방법에 깃든 비효율성을 감지할 수 있도록 한다.
- 오토마타를 이용한 매칭 방법을 이해한다.
- 라빈-카프 알고리즘의 수치화 과정을 이해한다.
- KMP 알고리즘을 이해하고, 오토마타를 이용한 방법과 비교해 이점을 이해하도록 한다.
- 보이어-무어 알고리즘의 개요를 이해하고, 다른 매칭 알고리즘들에 비해 어떤 특장점이 있는지 이해한다.

문자열 매칭

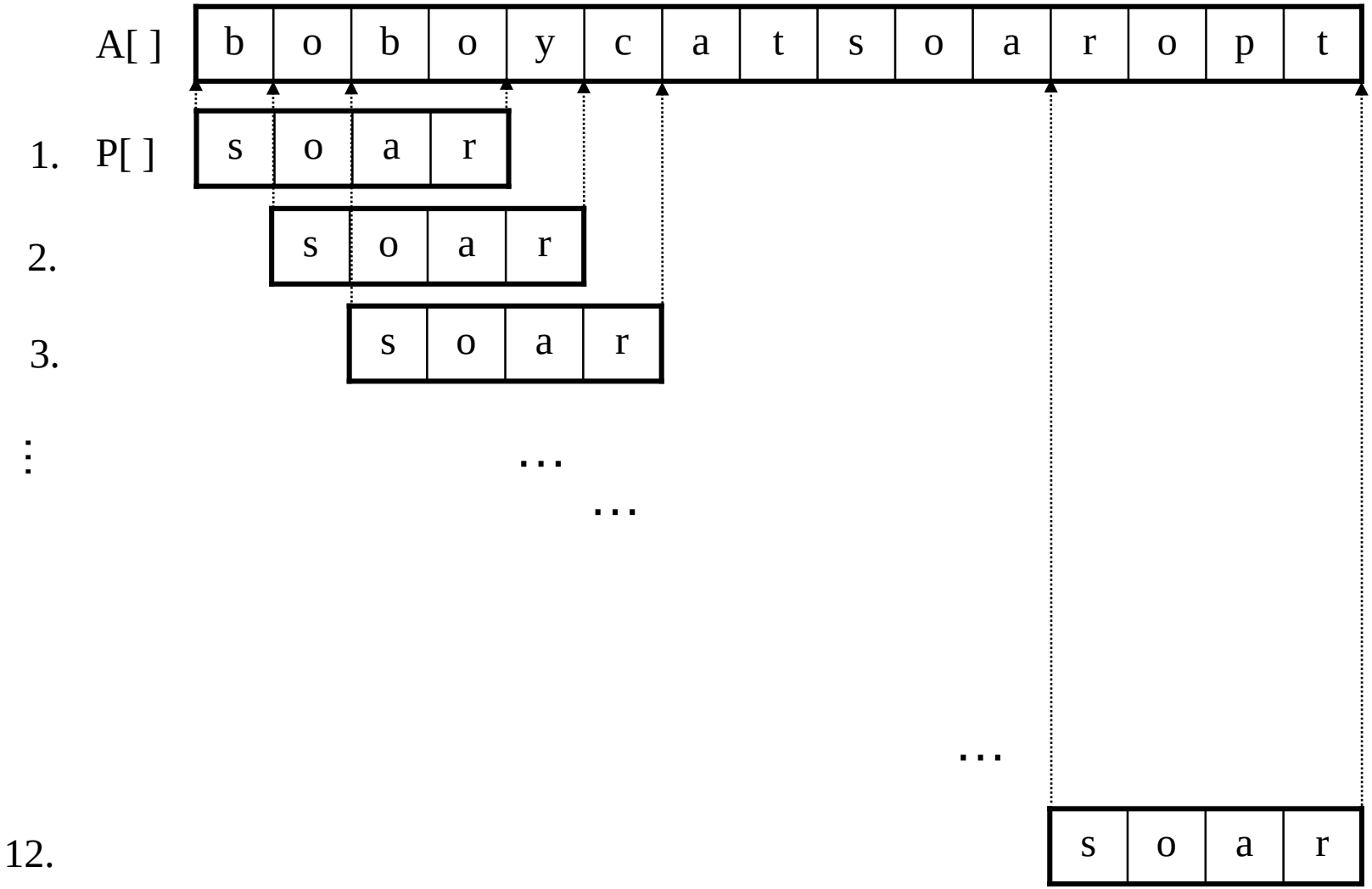
- 입력
  - A[1…n]: 텍스트 문자열
  - P[1…m]: 패턴 문자열
  - $m << n$
- 수행 작업
  - 텍스트 문자열 A[1…n]이 패턴 문자열 P[1…m]을 포함하는지 알아본다

원시적인 매칭

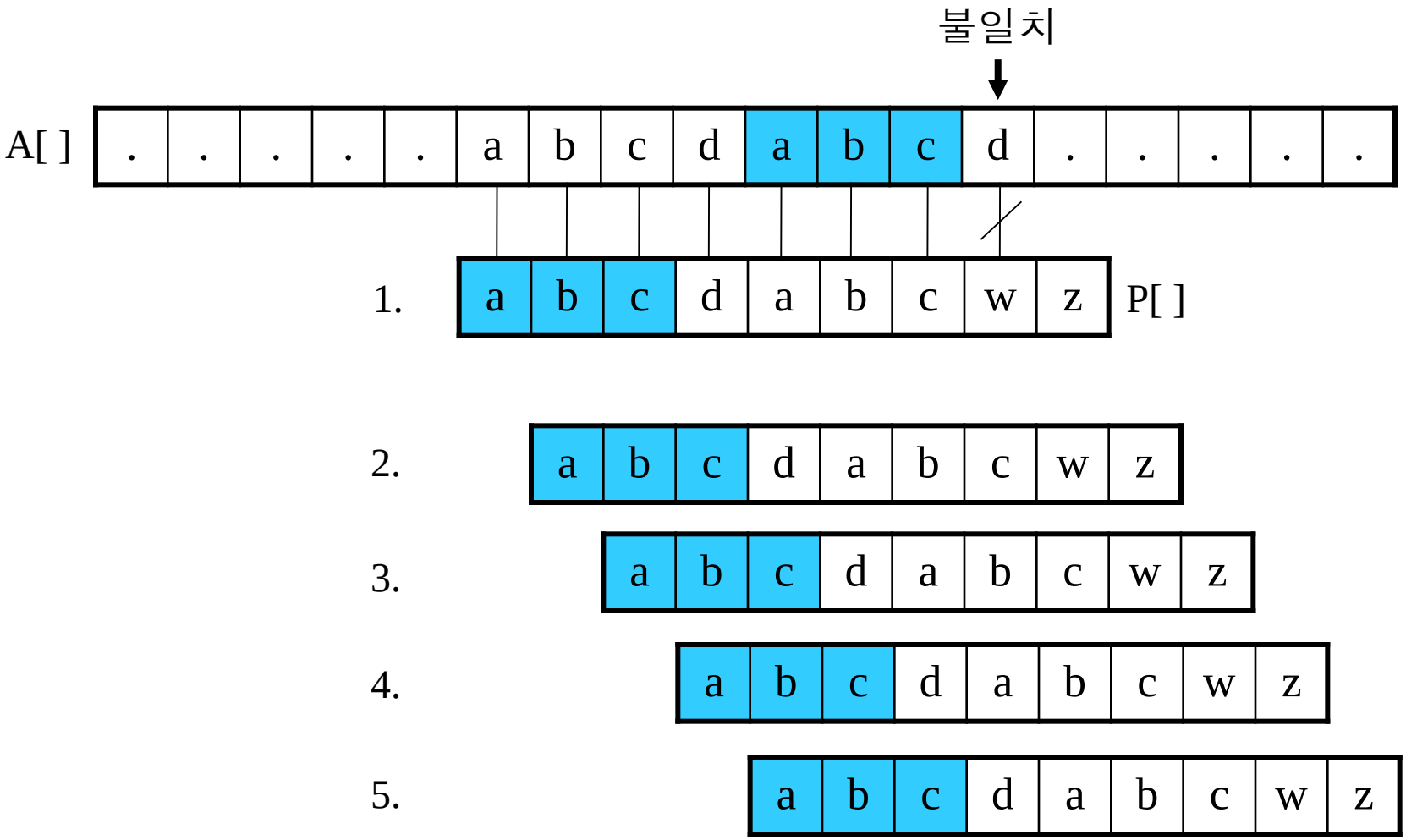
```
naiveMatching(A, P)
{
    ▷ n: 배열 A[ ]의 길이, m: 배열 P[ ]의 길이
    for i ← 1 to n-m+1{
        if (P[1…m] = A[i…i+m-1])
            then  A[i] 자리에서 매칭이 발견되었음을 알린다;
    }
}
```

✓ 수행시간: $O(mn)$

원시적인 매칭의 작동 원리



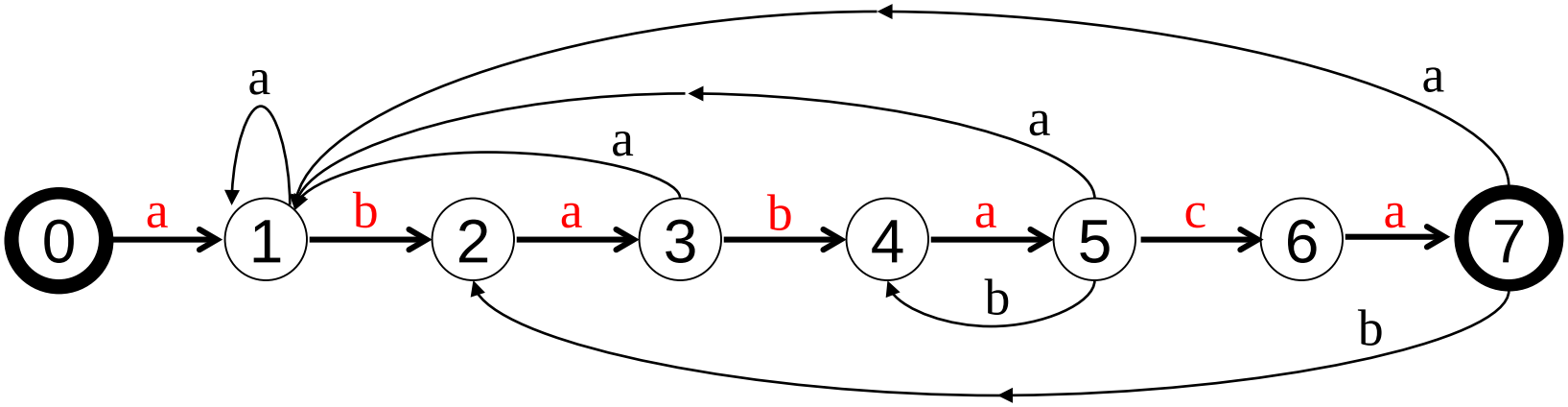
원시적인 매칭이 비효율적인 예



오토마타를 이용한 매칭

- 오토마타
  - 문제 해결 절차를 상태state의 전이로 나타낸 것
  - 구성 요소: $(Q, q_0, A, \sum, \delta)$
    - Q : 상태 집합
    - q_0 : 시작 상태
    - A : 목표 상태들의 집합
    - $\sum$: 입력 알파벳
    - δ : 상태 전이 함수
- 매칭이 진행된 상태들간의 관계를 오토마타로 표현한다

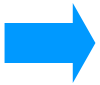
ababaca를 체크하는 오토마타



S: dvganbbactababa**ababaca**b**ababaca**agbk…

오토마타의 S/W 구현

상태 \ 입력문자	a	b	c	d	e	…	z
0	1	0	0	0	0	…	0
1	1	2	0	0	0	…	0
2	3	0	0	0	0	…	0
3	1	4	0	0	0	…	0
4	5	0	0	0	0	…	0
5	1	4	6	0	0	…	0
6	7	0	0	0	0	…	0
7	1	2	0	0	0	…	0



상태 \ 입력문자	a	b	c	기타
0	1	0	0	0
1	1	2	0	0
2	3	0	0	0
3	1	4	0	0
4	5	0	0	0
5	1	4	6	0
6	7	0	0	0
7	1	2	0	0

오토마타를 이용해 매칭을 체크하는 알고리즘

```
FA-Matcher (A, δ , f )
▷ f : 목표 상태
{
    ▷ n: 배열 A[ ]의 길이
    q ← 0;
    for i ← 1 to n {
        q ← δ(q, A[i]);
        if (q = f ) then A[i-m+1]에서 매칭이 발생했음을 알린다;
    }
}
```

✓ 총 수행시간: $\Theta(n + |\sum|m)$

라빈-카프Rabin-Karp 알고리즘

- 문자열 패턴을 수치로 바꾸어 문자열의 비교를 수치 비교로 대신한다
- 수치화
  - 가능한 문자 집합 $\sum$의 크기에 따라 진수가 결정된다
  - 예: $\sum = \{a, b, c, d, e\}$
    - $|\sum| = 5$
    - A, b, c, d, e를 각각 0, 1, 2, 3, 4에 대응시킨다
    - 문자열 “cad”를 수치화하면 $2*5^2+0*5^1+3*5^0 = 28$

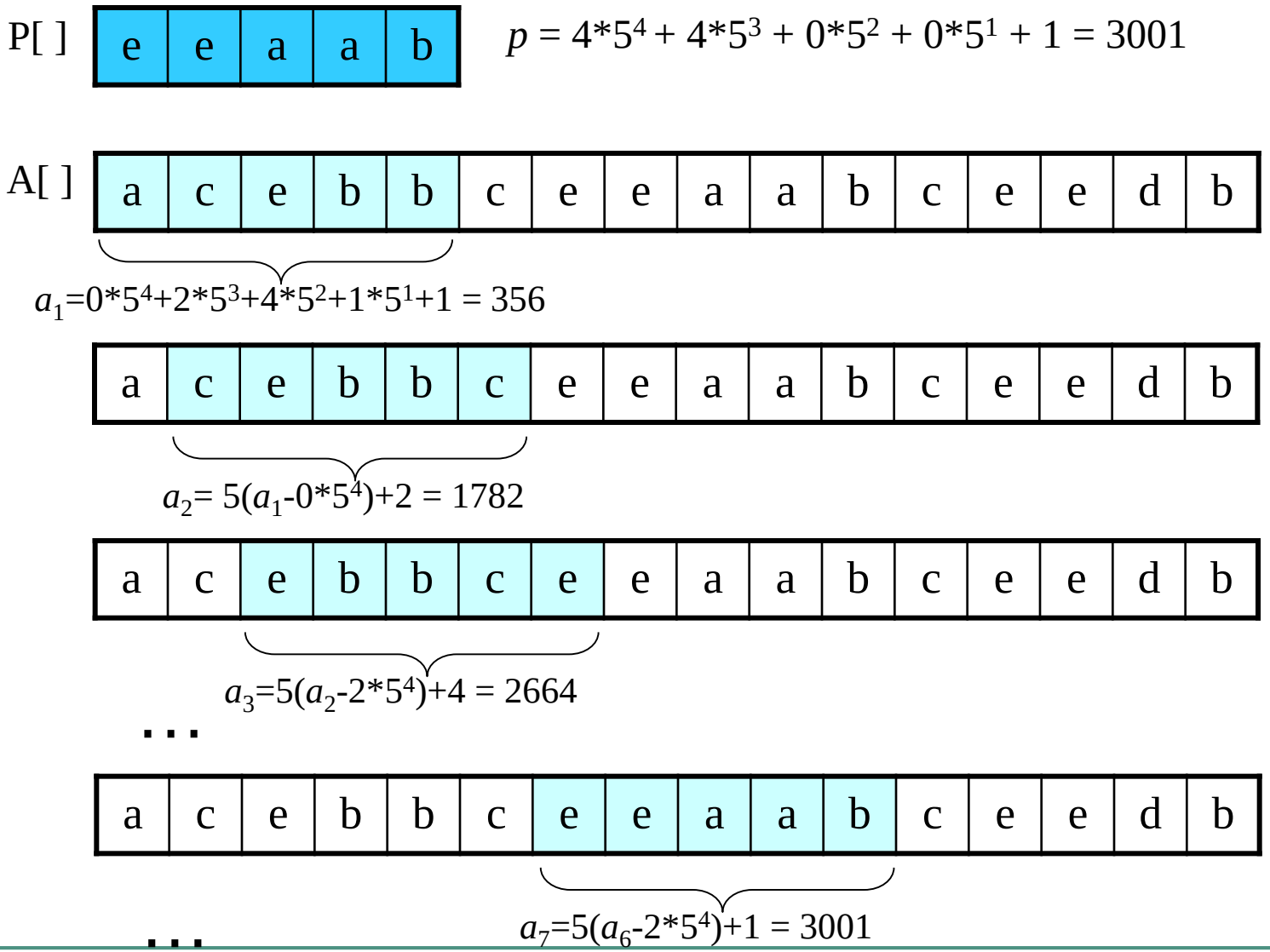
라빈-카프Rabin-Karp 알고리즘

- 문자열 패턴을 수치로 바꾸어 문자열의 비교를 수치 비교로 대신한다
- 수치화
  - 가능한 문자 집합 $\sum$의 크기에 따라 진수가 결정된다
  - 예: $\sum = \{a, b, c, d, e\}$
    - $|\sum| = 5$
    - a, b, c, d, e를 각각 0, 1, 2, 3, 4에 대응시킨다
    - 문자열 “cad”를 수치화하면 $2*5^2+0*5^1+3*5^0 = 28$

수치화 작업의 부담

- A[i…i+m-1]에 대응되는 수치의 계산
  - $a_i = \mathrm{A}[i+m-1] + d(\mathrm{A}[i+m-2] + d(\mathrm{A}[i+m-3] + d(\ldots + d(\mathrm{A}[i]))\ldots)$
  - $\Theta(m)$의 시간이 든다
  - 그러므로 A[1…n] 전체에 대한 비교는 $\Theta(mn)$이 소요된다
  - 원시적인 매칭에 비해 나은 게 없다
- 다행히,
  m의 크기에 상관없이 아래와 같이 계산할 수 있다
  - $a_i = d(a_{i-1} - d^{m-1}\mathrm{A}[i-1]) + \mathrm{A}[i+m-1]$
  - d^{m-1}은 반복 사용되므로 미리 한번만 계산해 두면 된다
  - 곱셈 2회, 덧셈 2회로 충분

수치화를 이용한 매칭의 예



수치화를 이용해 매칭을 체크하는 알고리즘

```
basicRabinKarp(A, P, d, q)
{
    ▷ n : 배열 A[ ]의 길이,  m : 배열 P[ ]의 길이
    p ← 0; a_1 ← 0;
    for i ← 1 to m {            ▷ a_1 계산
        p ← dp + P[i];
        a_1 ← da_1 + A[i];
    }
    for i ← 1 to n-m+1{
        if (i ≠ 1) then a_i ← d(a_{i-1} – d^{m-1}A[i-1]) + A[i+m-1];
        if (p = a_i) then A[i] 자리에서 매칭이 되었음을 알린다;
    }
}
```

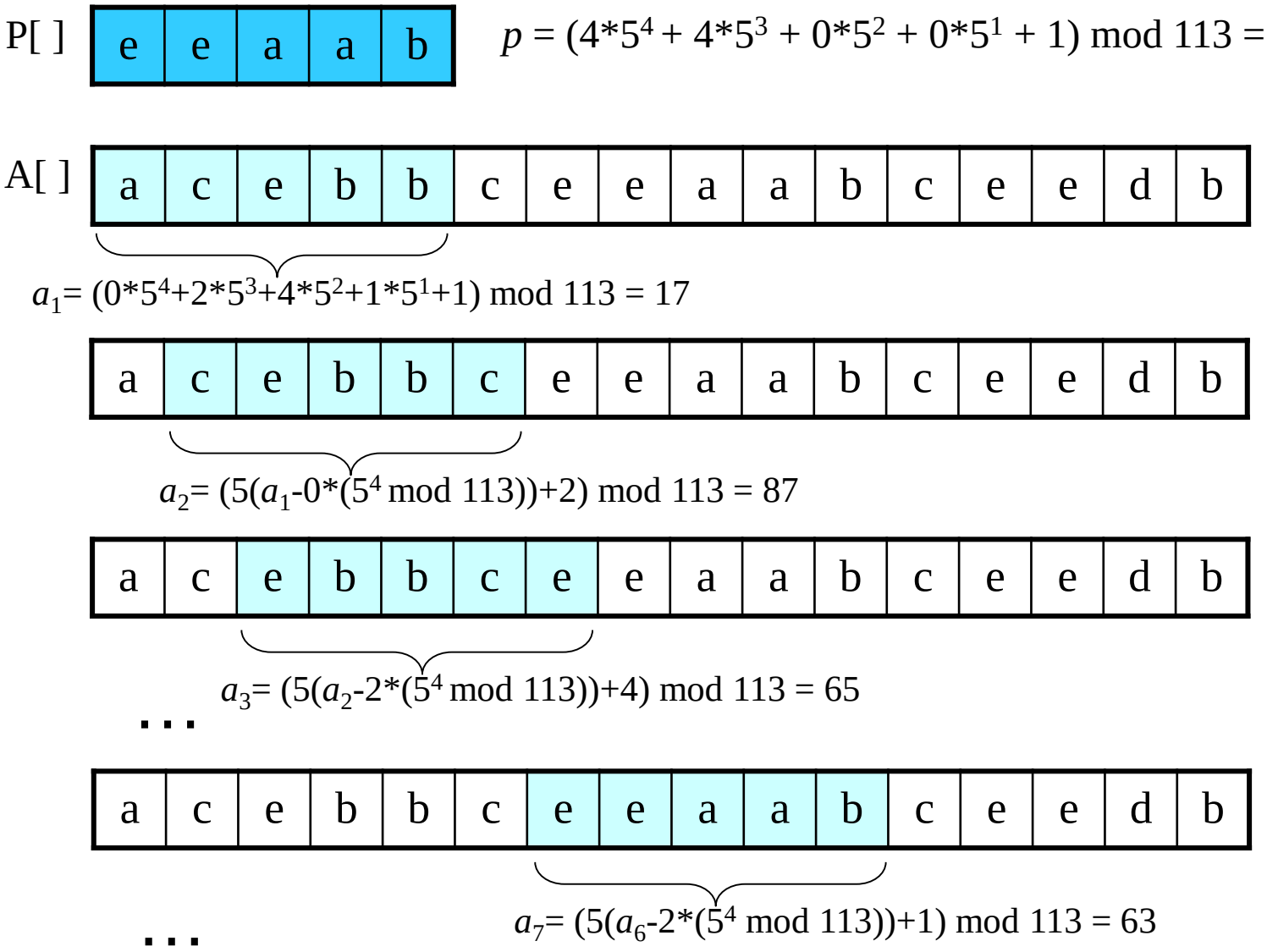
✓ 총 수행시간: $\Theta(n)$

앞의 알고리즘의 문제점

- 문자 집합 Σ와 m의 크기에 따라 a_i가 매우 커질 수 있다
  - 심하면 컴퓨터 레지스터의 용량 초과
  - 오버플로우 발생
- 해결책
  - 나머지 연산modulo을 사용하여 a_i의 크기를 제한한다
  - $a_i = d(a_{i-1} - d^{m-1}\mathrm{A}[i\text{-}1]) + \mathrm{A}[i+m\text{-}1]$ 대신

    $b_i = (d(b_{i-1} - (d^{m-1} \bmod q)\,\mathrm{A}[i\text{-}1]) + \mathrm{A}[i+m\text{-}1]) \bmod q$ 사용
  - q를 충분히 큰 소수로 잡되, dq가 레지스터에 수용될 수 있도록 잡는다

나머지 연산을 이용한 매칭의 예



라빈-카프 알고리즘

```
RabinKarp(A, P, d, q)
{
    ▷ n : 배열 A[ ]의 길이,  m : 배열 P[ ]의 길이
    p ← 0; b1 ← 0;
    for i ← 1 to m {                                        ▷ b1 계산
        p ← (dp + P[i]) mod q;
        b1 ← (db1 + A[i]) mod q;
    }
    h ← d^(m-1) mod q;
    for i ← 1 to n-m+1{
        if (i ≠ 1) then bi ← (d(bi-1 – hA[i-1]) + A[i+m-1]) mod q;
        if (p = bi) then
            if (P[1…m] = A[i…i+m-1]) then
                A[i] 자리에서 매칭이 되었음을 알린다;
    }
}
```

✓ 평균 수행시간: $\Theta(n)$

Thank you