

$$\frac{10 + 20}{\text{"10"} + \text{"20"}} \quad O(1) \quad O(\frac{1}{2}n)$$

$$\frac{10 < 2000000}{\text{"10"} < \text{"2000000"}} \quad O(1)$$

$$S_1 < S_2 \quad O(\frac{1}{2}n)$$

문자열 알고리즘 1

최백준 choi@startlink.io

문자열 S

패턴 P

Ctrl + F
Cmd + F

문자열 매칭 알고리즘

|S|
|

문자열 매칭 알고리즘

3

String Matching Algorithm

- 문자열 S에서 패턴 P를 찾는 알고리즘
- S에서 가장 먼저 나타나는 P를 찾아야 함

$$O(|S| \times |P|)$$

- S = "ABCABDABCABEABC"

- P = "ABCABE"

- S[6]부터 P가 나타남!

S = AAAAAAAAAA --- B

P = A --- AB

문자열 매칭 알고리즘

String Matching Algorithm

- $O(|S| \times |P|)$
- 모든 경우를 다 해보는 알고리즘

문자열 매칭 알고리즘

String Matching Algorithm

5

- 소스: <http://boj.kr/f9b370d632fb46c5b2ede5def53c68d8>

- ① KMP : 문자열 S 에서 패턴 P
- ② Trie : 문자열 집합 A 에서 문자열 S
- ③ Aho-Corasick : 문자열 집합 A 에서 패턴 P

⊕ Suffix Array

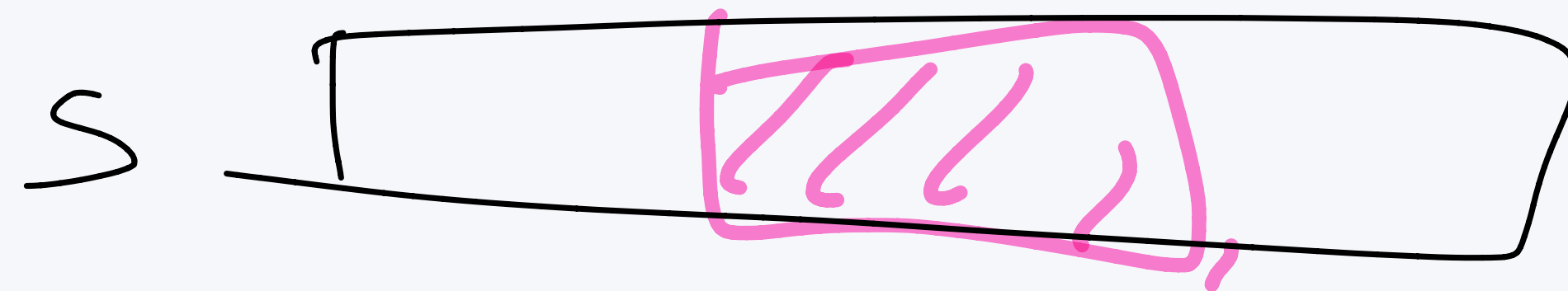
KMP

KMP

String Matching Algorithm

- KMP는 왜 KMP일까?
- <https://www.acmicpc.net/problem/2902>
- 만든 사람의 성이 Knuth, Morris, Prett이어서

2-SATISAl_o.



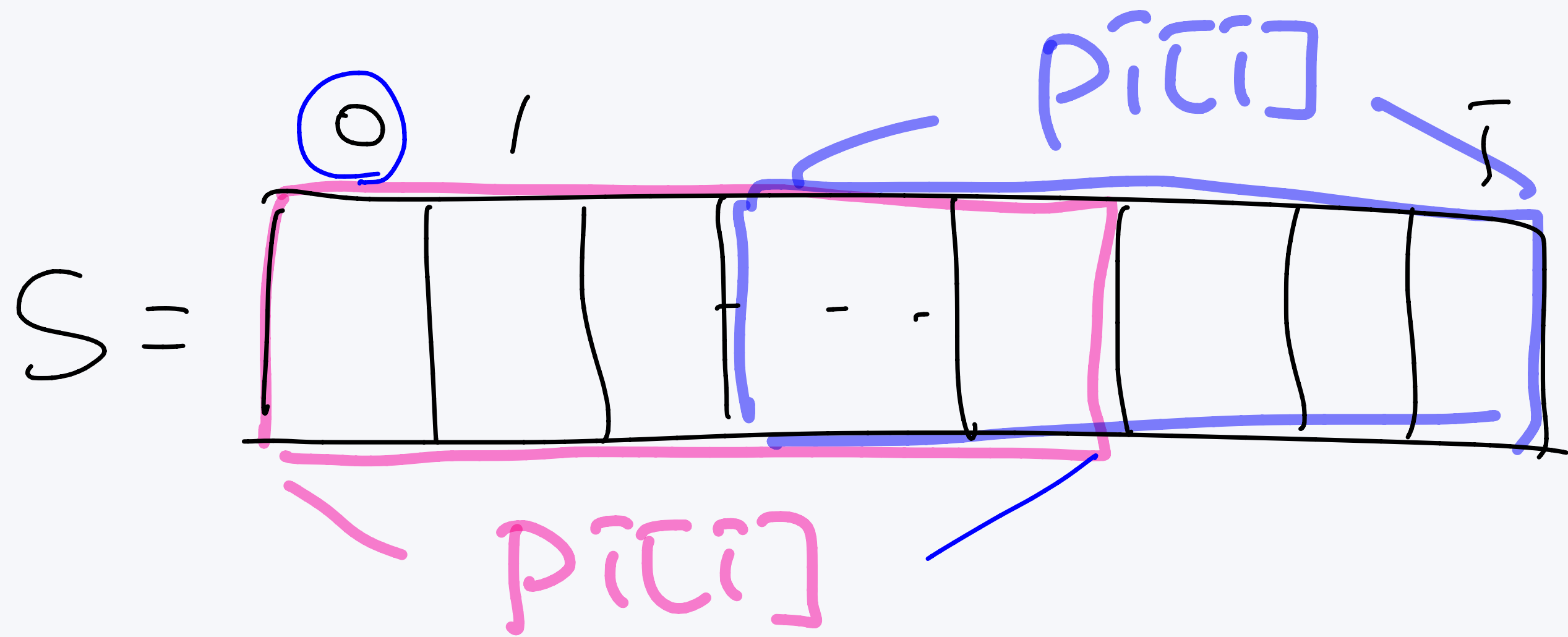
KMP

String Matching Algorithm

- KMP는 배열 pi 를 이용해야 한다

- $pi[i]$


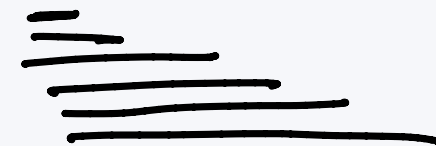
0



- P의 i 까지 부분 문자열에서 $\text{prefix} == \text{suffix}$ 가 될 수 있는 부분 문자열 중에서 가장 긴 것의 길이
- 이 때, prefix 가 i 까지 부분 문자열과 같으면 안된다.

Prefix

String Matching Algorithm

- P = ABCABE 


- Prefix

- A

- AB

- ABC

- ABCA

- ABCAB

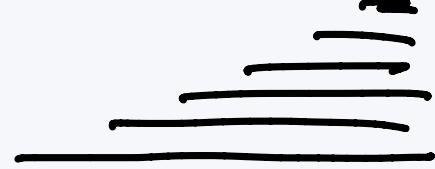
- ABCABE

Suffix

10

String Matching Algorithm

- P = ABCABE



- Suffix
- E
- BE
- ABE
- CABE
- BCABE
- ABCABE

KMP

String Matching Algorithm

- ABCABE 의 pi[]

i	부분 문자열	pi[i]
0	A	0
1	AB	0
2	ABC	0
3	<u>ABCA</u>	1
4	<u>ABCAB</u>	2
5	ABCABE	0

A ≠ C
AB ≠ BC
A A
AB CA
ABC BCA

~~O(N²)~~
O(P)

KMP

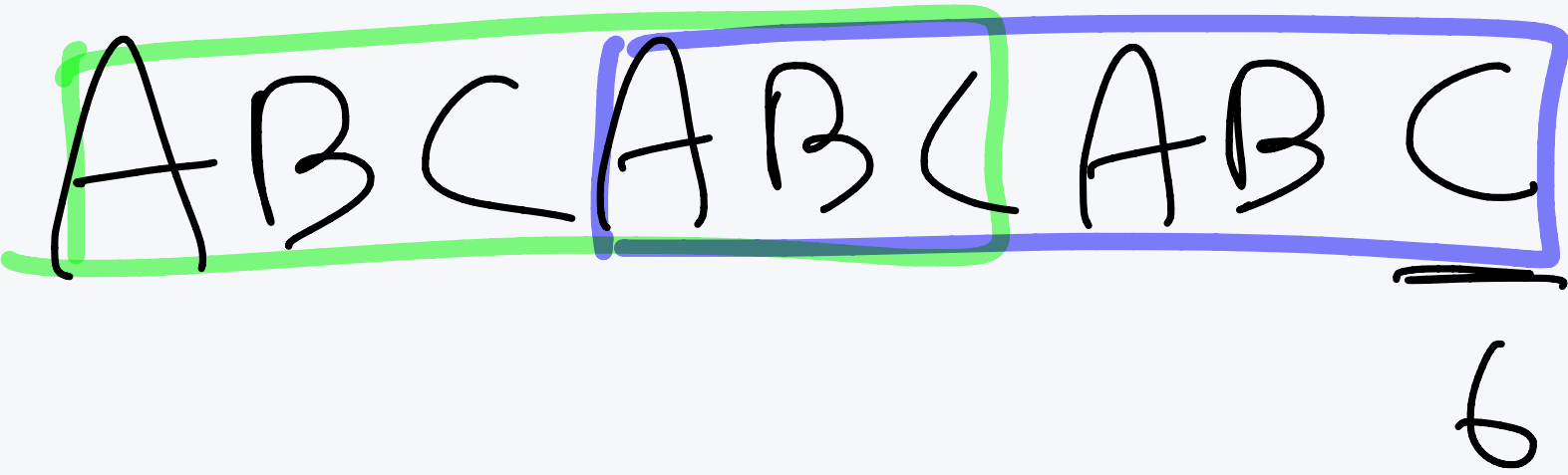
String Matching Algorithm

- ABCABDABCABEABC 의 pi

i	부분 문자열	pi[i]
0	A	0
1	AB	0
2	ABC	0
3	ABCA	1
4	ABCAB	2
5	ABCABD	0
6	ABCABDA	1

KMP

String Matching Algorithm



- ABCABDABCABEABC 의 pi

i	부분 문자열	pi[i]
7	ABCABDAB	2
8	ABCABDABC	3
9	ABCABDABCA	4
10	ABCABDABCAB	5
11	ABCABDABCABE	0
12	ABCABDABCABEA	1
13	ABCABDABCABEAB	2
14	ABCABDABCABEABC	3

KMP

String Matching Algorithm

[illegible]

KMP

String Matching Algorithm

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0									

A

KMP

String Matching Algorithm

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0								

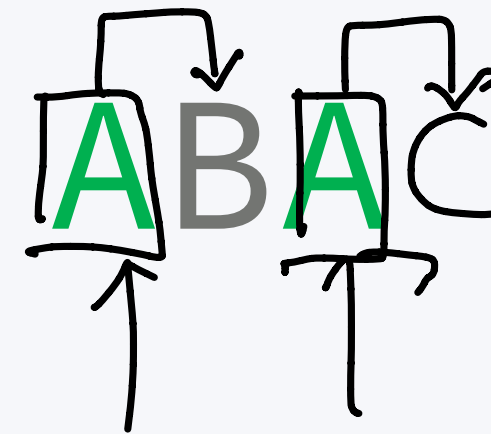
AB

KMP

String Matching Algorithm

17

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1							



KMP

String Matching Algorithm

18

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0						

ABAC

KMP

String Matching Algorithm

19

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0	1	2				

ABACAB

KMP

String Matching Algorithm

20

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0	1	2	3			

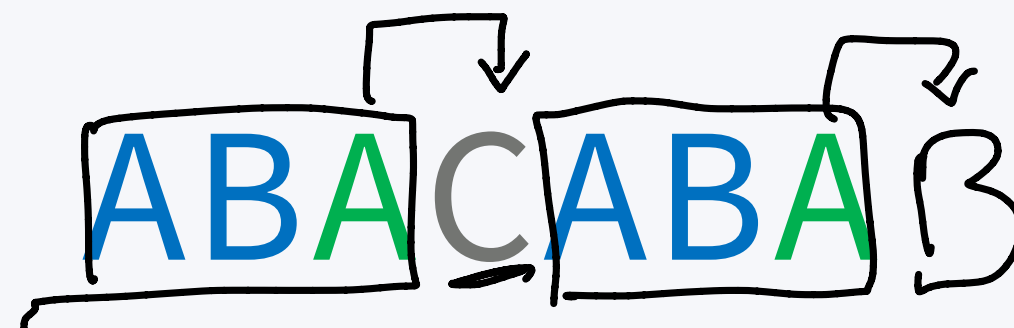


KMP

String Matching Algorithm

21

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0	1	2	3	4		



KMP

String Matching Algorithm

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0	1	2	3			

ABACABAB

KMP

String Matching Algorithm

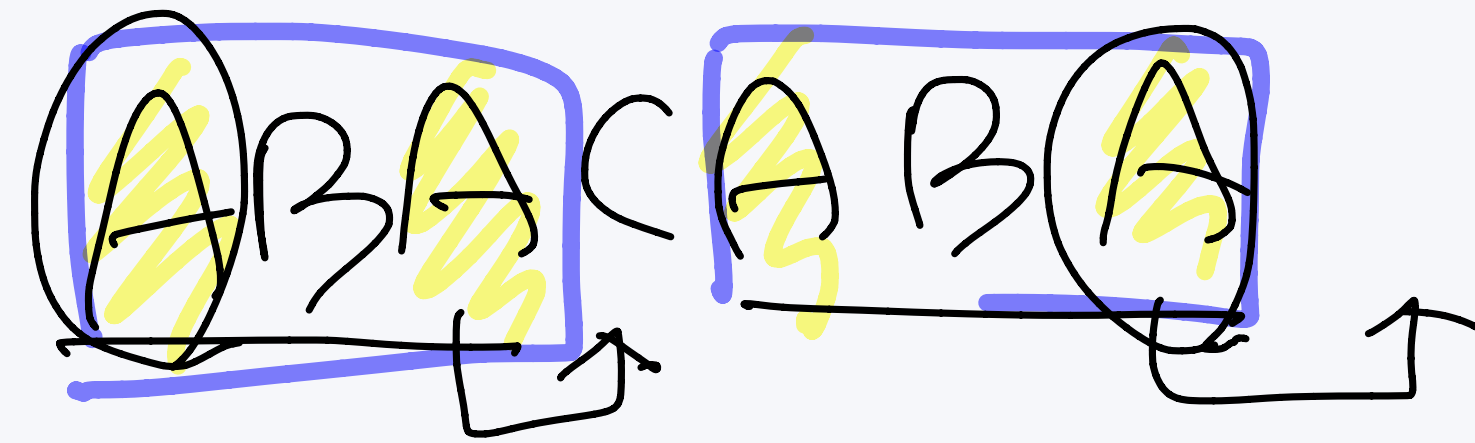
i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1							

ABA

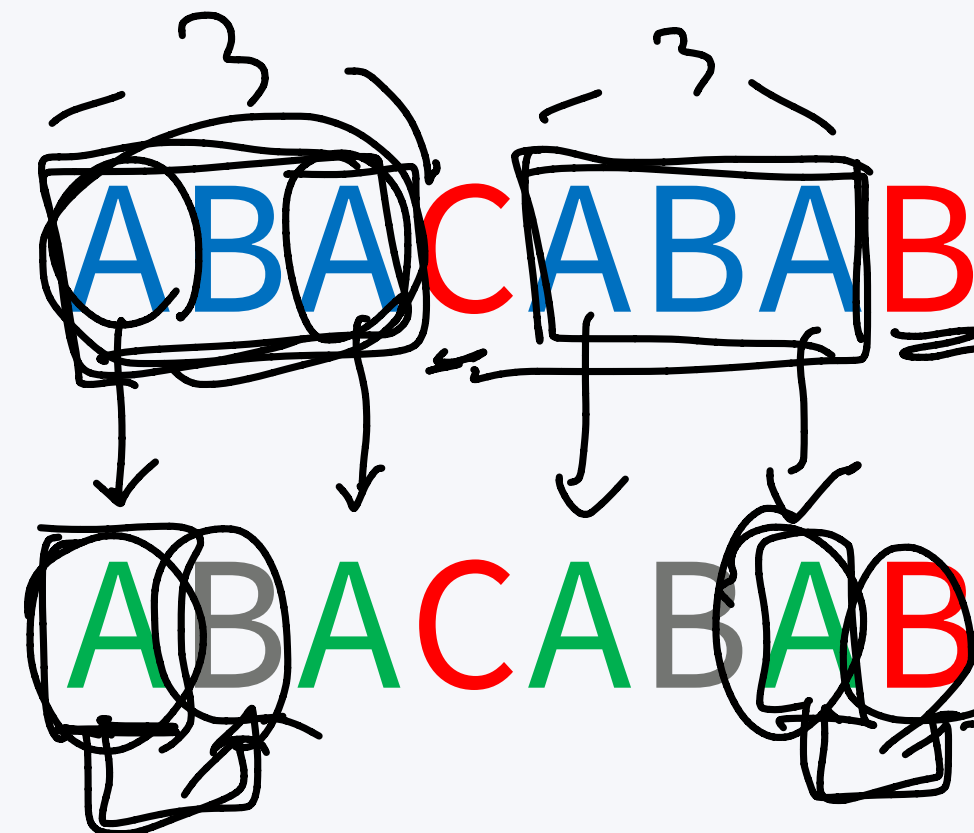
KMP

String Matching Algorithm

24



i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0	1	2	3	2		



KMP

String Matching Algorithm

25

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0	1	2	3	2		

ABACABAB

ABACABAB

ABACABAB

KMP

String Matching Algorithm

i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0	1	2	3	2	3	

ABACABABA

KMP

String Matching Algorithm

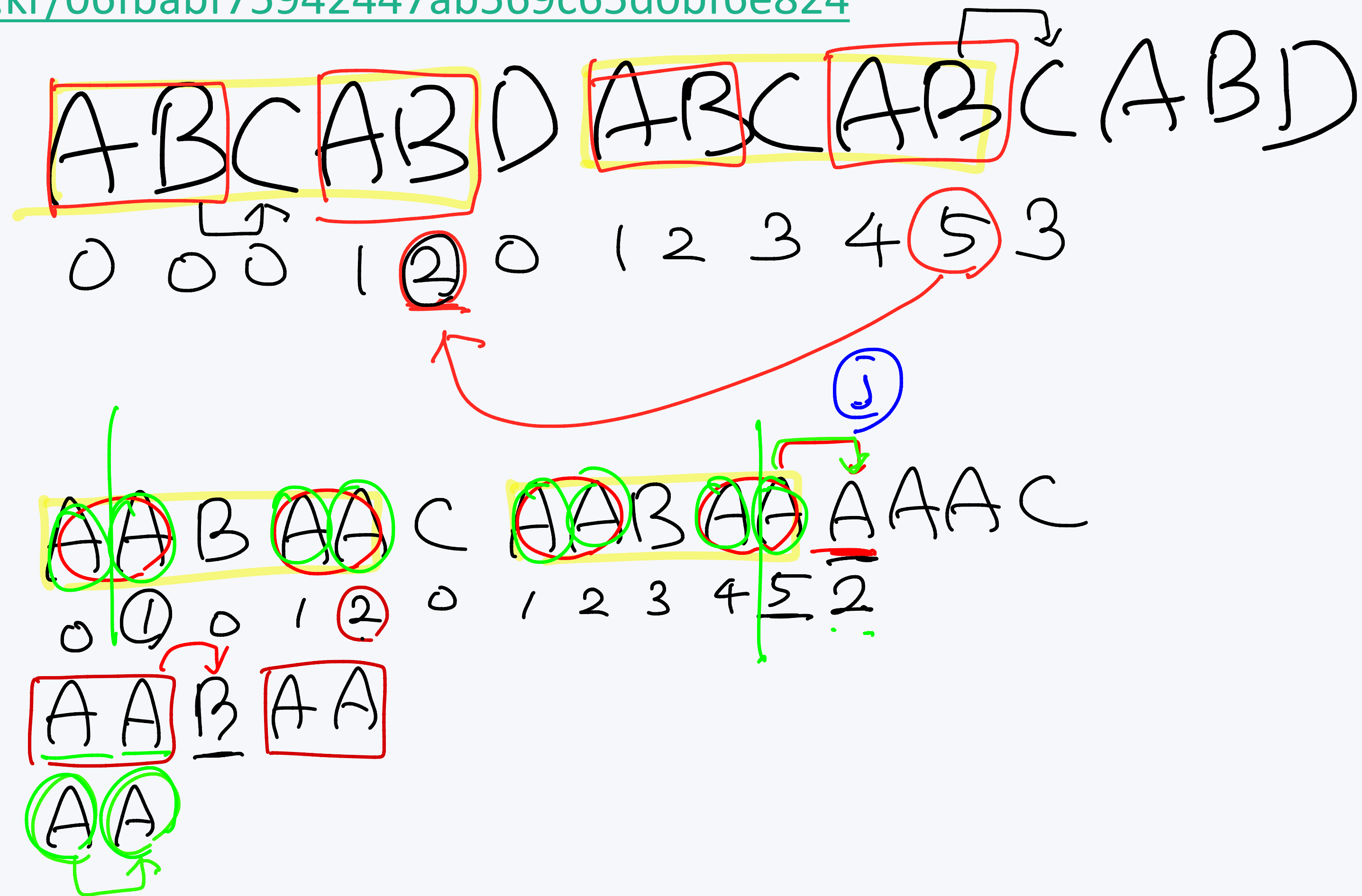
i	0	1	2	3	4	5	6	7	8	9
p[i]	A	B	A	C	A	B	A	B	A	C
pi[i]	0	0	1	0	1	2	3	2	3	4

ABACABABAC

KMP

String Matching Algorithm

- 소스: <http://boj.kr/06fbabf75942447ab569c65d0bf6e824>



KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- pi[4] = 2라는 것은
- $p[0\cdots 1] == p[3\cdots 4]$ 라는 것을 의미
- ABCAB**

KMP

30

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- pi[4] = 2라는 것은
- $p[0 \cdots 1] == p[3 \dots 4]$ 라는 것을 의미
- ABCABCABE에서 ABCABE를 찾을 때
- ABCAB**C**ABE
- ABCAB****E**

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- pi[4] = 2라는 것은
- $p[0 \cdots 1] == p[3 \cdots 4]$ 라는 것을 의미
- ABCABE에서 ABCABE를 찾을 때
- ABCABE
- ABE (앞의 2개를 건너뛰고 비교를 이어가도 된다)
- ABE

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기

- 패턴 ABCABE의 pi

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기

[illegible]

[illegible]

- 패턴 ABCABE의 pi

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 1, j = 1$)

[illegible]

[illegible]

[illegible]

- 패턴 ABCABE의 pi

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 4, j = 4$)

[illegible]

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기
- S[i] == P[j]를 비교하기 (i = 5, j = 5) 다르기 때문에 j = pi[j-1]로 이동한다

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P	A	B	C	A	B	E										

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 5, j = 2$) 다르기 때문에 $j = pi[j-1]$ 로 이동한다

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P				A	B	C	A	B	E							

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 5, j = 0$) 다른데, $j = 0$ 이므로 다음으로 넘어간다

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P						A	B	C	A	B	E					

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCEBF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 6, j = 0$)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P							A	B	C	A	B	E				

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 7, j = 1$)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P							A	B	C	A	B	E				

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCEBF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 8, j = 2$)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P							A	B	C	A	B	E				

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCEBF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 9, j = 3$)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P							A	B	C	A	B	E				

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCEEF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 10, j = 4$)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P							A	B	C	A	B	E				

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCEEF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 11, j = 5$) 다르기 때문에, $j = pi[j-1]$ 로 이동한다

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P							A	B	C	A	B	E				

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCEBF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 11, j = 2$)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P										A	B	C	A	B	E	

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCEBF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 12, j = 3$)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P										A	B	C	A	B	E	

KMP

String Matching Algorithm

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCEBF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 13, j = 4$)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P										A	B	C	A	B	E	

KMP

String Matching Algorithm

51

$$O(|S| + |P|)$$

- 패턴 ABCABE의 pi

i	0	1	2	3	4	5
p[i]	A	B	C	A	B	E
pi[i]	0	0	0	1	2	0

- 문자열 ABCABDABCABCABEF 에서 패턴 ABCABE를 찾아보기
- $S[i] == P[j]$ 를 비교하기 ($i = 14, j = 5$) 찾았다!

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	A	B	C	A	B	D	A	B	C	A	B	C	A	B	E	F
P										A	B	C	A	B	E	

KMP

String Matching Algorithm

- pi 배열 구하는데 걸리는 시간 $O(M)$
- 문자열 매칭하는데 걸리는 시간 $O(N+M)$

찾기

53

<https://www.acmicpc.net/problem/1786>

- 문자열 T가 주어졌을 때 P가 몇 번 등장하는지 구하는 문제

찾기

<https://www.acmicpc.net/problem/1786>

- 소스: <http://boj.kr/8ebc82526b4444fb9964b2ab4bde1400>

광고

<https://www.acmicpc.net/problem/1305>

- 광고판의 크기: L

- 광고 문구의 길이: N

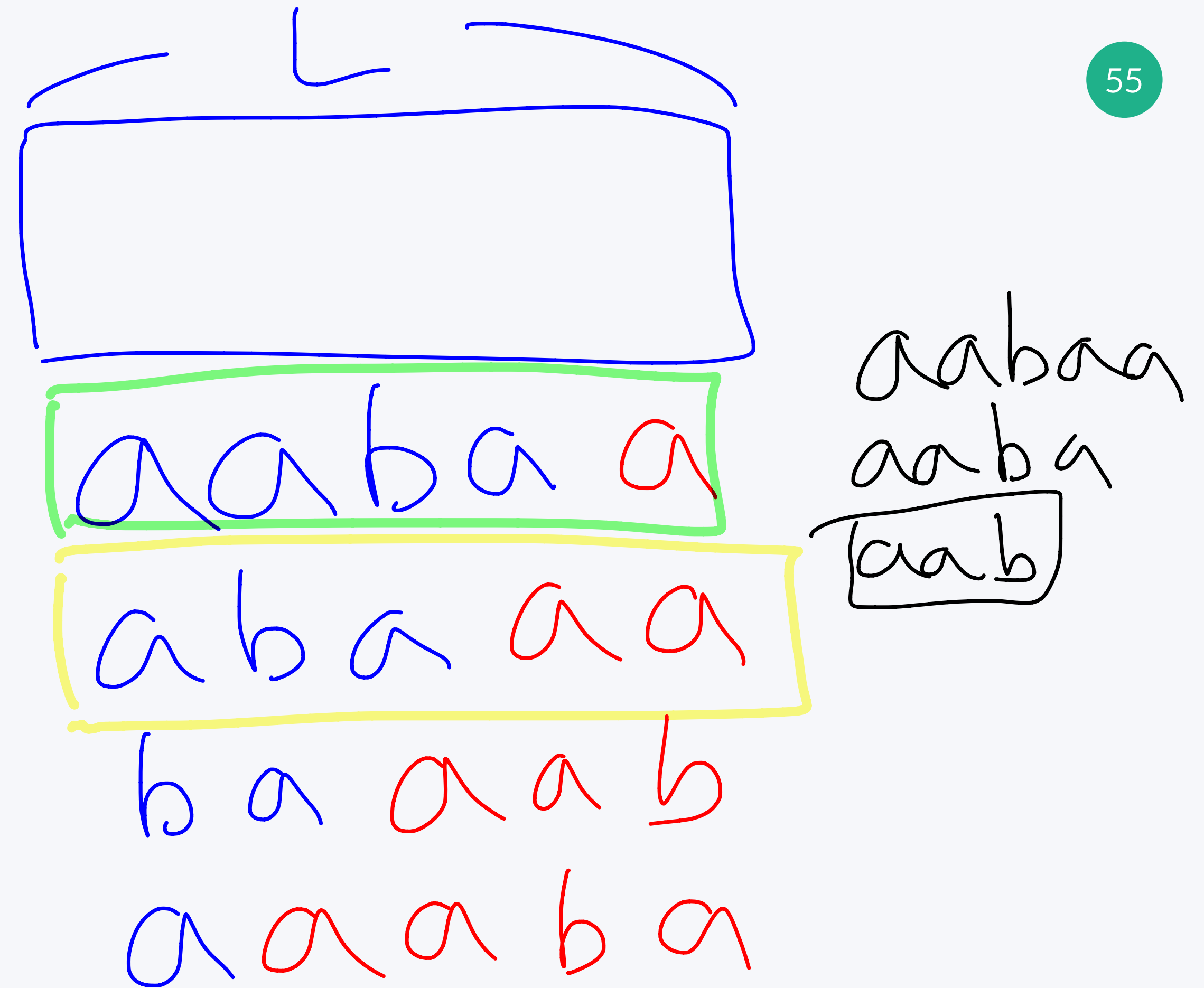
- 광고 문구: aaba

- 광고판의 크기: 5

- 인 경우

정답: $N - P[N]$

- aabaa -> abaaa -> baaab -> aaaba -> ...



광고

<https://www.acmicpc.net/problem/1305>

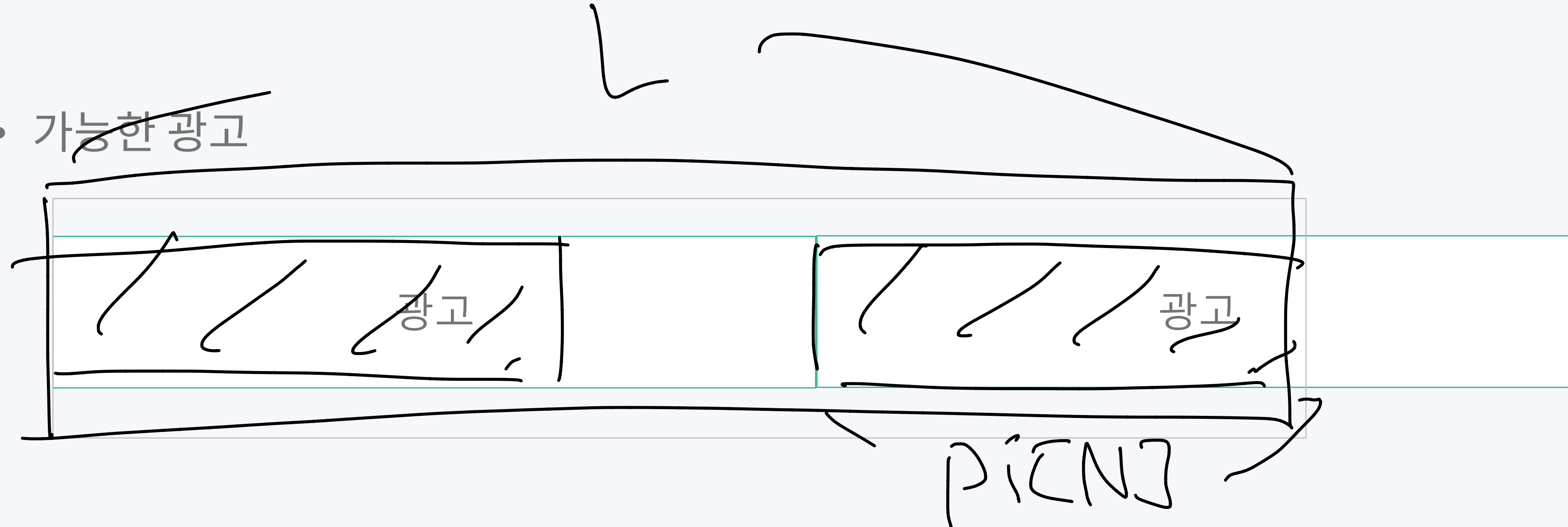
56

abba
0 1 0 1 2

$$5-2=3$$

- 어느 순간 광고 문구가 주어졌을 때
- 가능한 광고의 길이 중 가장 짧은 것 구하기

- 가능한 광고

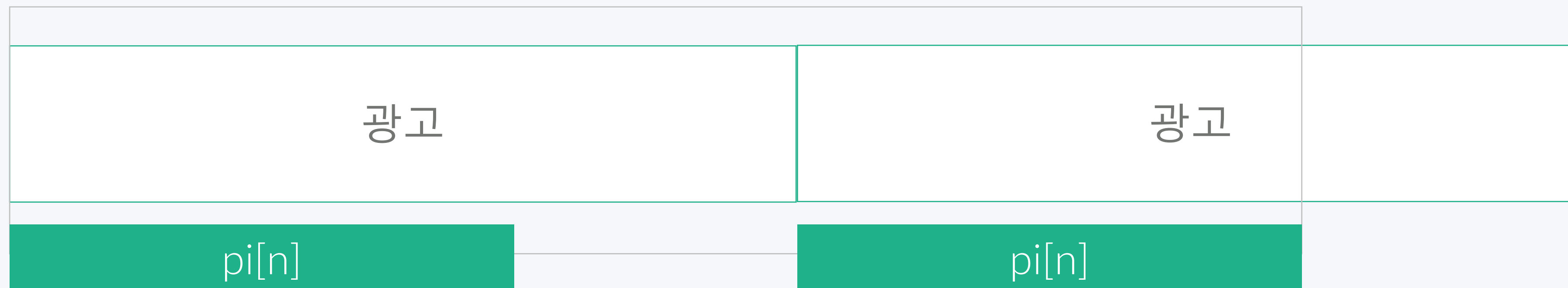


광고

57

<https://www.acmicpc.net/problem/1305>

- 어느 순간 광고 문구가 주어졌을 때
- 가능한 광고의 길이 중 가장 짧은 것 구하기
- 가능한 광고



광고

58

<https://www.acmicpc.net/problem/1305>

- 정답은 $N - \text{pi}[N]$ 이 된다.

광고

<https://www.acmicpc.net/problem/1305>

- 소스: <http://boj.kr/2f6d9086a2104b2c8a8f551155a83a6b>

Cubeditor

<https://www.acmicpc.net/problem/1701>

- 소문자 5,000개 이하로 구성된 문자열 S가 주어진다
- S의 부분 문자열은 연속된 일부분
- 두 번 이상 등장하는 부분 문자열 중에서 가장 긴 것의 길이?

문자열 S에서

두 번 이상 등장하는

부분 문자열

가장 긴 것

검출도 가능

$N \times \Theta(N)$

abcdabcdabb
0000123120
1

abcdabcdabb

정답: 3

Cubeditor

61

<https://www.acmicpc.net/problem/1701>

- 모든 부분 문자열은
- 어딘가에서 시작해서 어딘가에서 끝난다

	부분 문자열	
--	--------	--

Cubeditor

<https://www.acmicpc.net/problem/1701>

- 모든 부분 문자열은
- 어딘가에서 시작해서 어딘가에서 끝난다
- 어떤 suffix의 prefix 이다.
- 원래 문자열에서 KMP를 돌리면, 모든 pi에는
- 가장 처음부터 얼마만큼이 같은지 저장되어 있다

	부분 문자열	
--	--------	--

Cubeditor

63

<https://www.acmicpc.net/problem/1701>

- 따라서, 모든 부분 문자열 $S[i..N]$ 에 대해서 pi 를 구하고, 그 중의 최대값을 구하면 된다.

	부분 문자열	
--	--------	--

Cubeditor

64

<https://www.acmicpc.net/problem/1701>

- 소스: <http://boj.kr/7c40ed7668bf4038b40ba2a8fcf08b62>

순환 순열

<https://www.acmicpc.net/problem/12104>

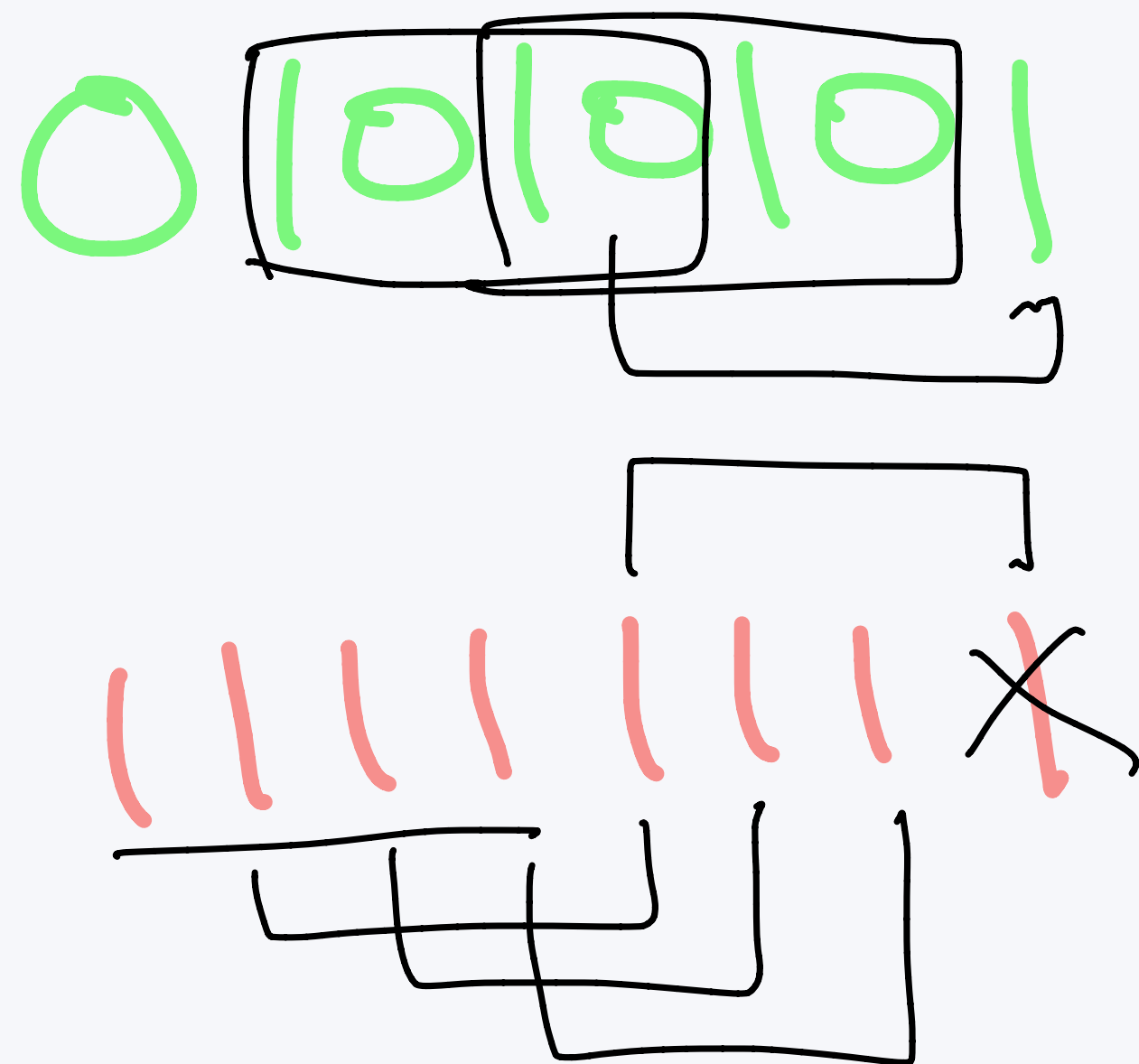
65

03-1

A = 1111
B = 1111

(4)

- 두 바이너리 문자열 A와 B가 주어졌을 때
- A와 XOR 했을 때, 0이 나오는 B의 순환 순열의 개수를 구하는 문제



$B_1 B_2 B_3 B_4$
 $B_2 B_3 B_4 B_1$
 $B_3 B_4 B_1 B_2$
 $B_4 B_1 B_2 B_3$

A = 1010
B = 0101

(2)

B + B 이거나 $A \frac{2}{2}$ $\frac{2}{2}$
 마지막 $\frac{2}{2}$ X

순환 순열

<https://www.acmicpc.net/problem/12104>

- B+B에서 A가 몇 개 나오는지 찾는 문제이다.
- XOR은 같은 수 일 때만 0이 나오기 때문
- B+B에서 마지막 글자는 제외해야 한다.

순환 순열

67

<https://www.acmicpc.net/problem/12104>

- 소스: <http://boj.kr/a2af125c3c76403582c7009463877b9c>

문자열

Binary Search: $O(\lg N)$ 가리키 → 문자열
" : ~~$O(N)$~~ 문자열
 $O(\lg N)$

Trie

문자열 추가
가리키

Trie

Trie

- 숫자 비교는 $O(1)$ 이지만
- 문자열 비교는 최대 $O(\text{길이})$ 가 걸린다.
- 문자열 N 개를 담고있는 BST에서 검색하는데 걸리는 시간은 $O(\lg N)$ 이 아니고 $O(\text{길이} * \lg N)$ 이다.

Trie

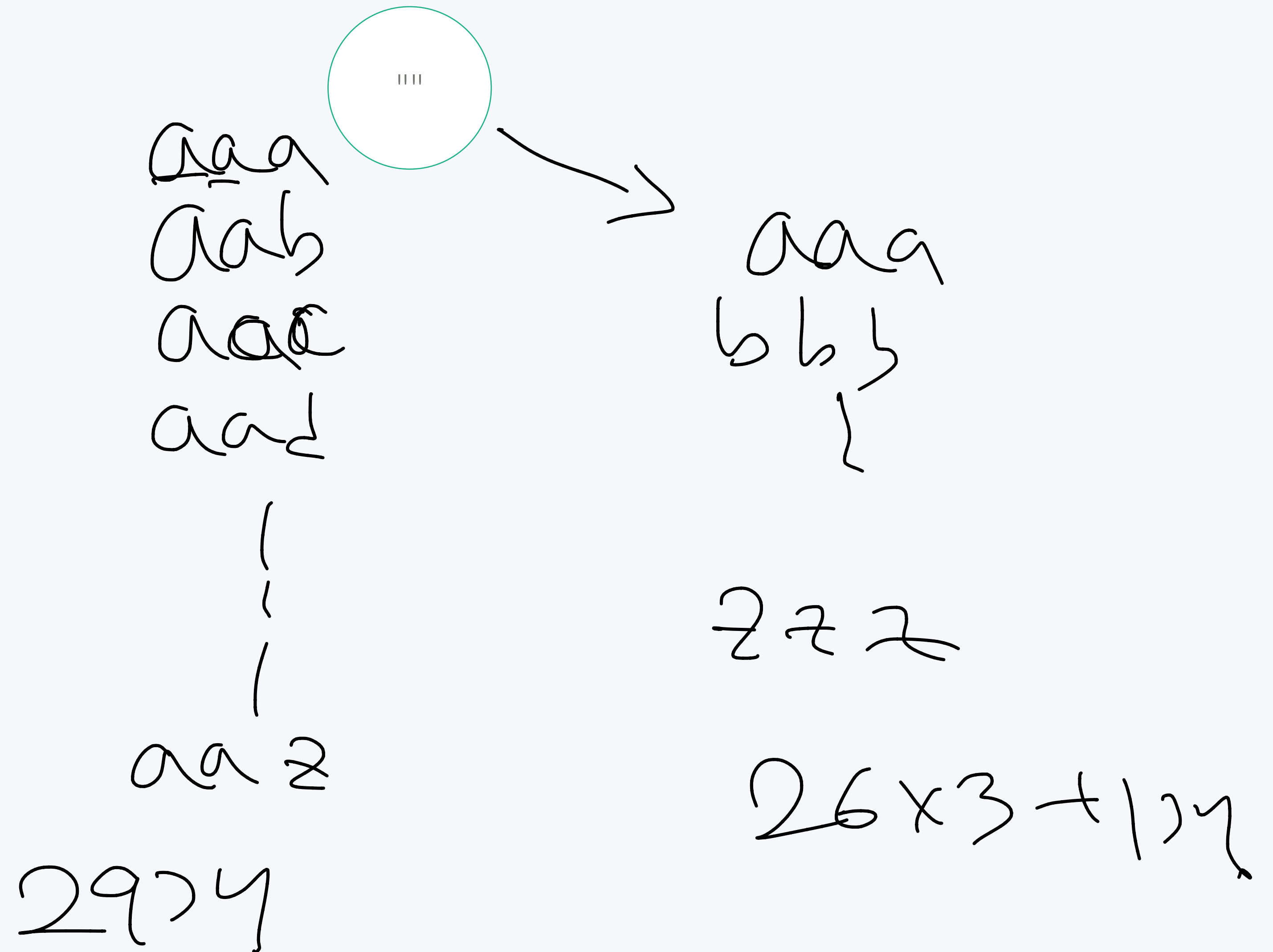
Trie

- 트라이의 한 노드는 어떤 문자열의 접두사를 나타낸다.
- 따라서, Trie는 Prefix Tree 라고도 한다.

Trie

Trie

- 트라이의 초기 상태: 루트만 있다.



Trie

Trie

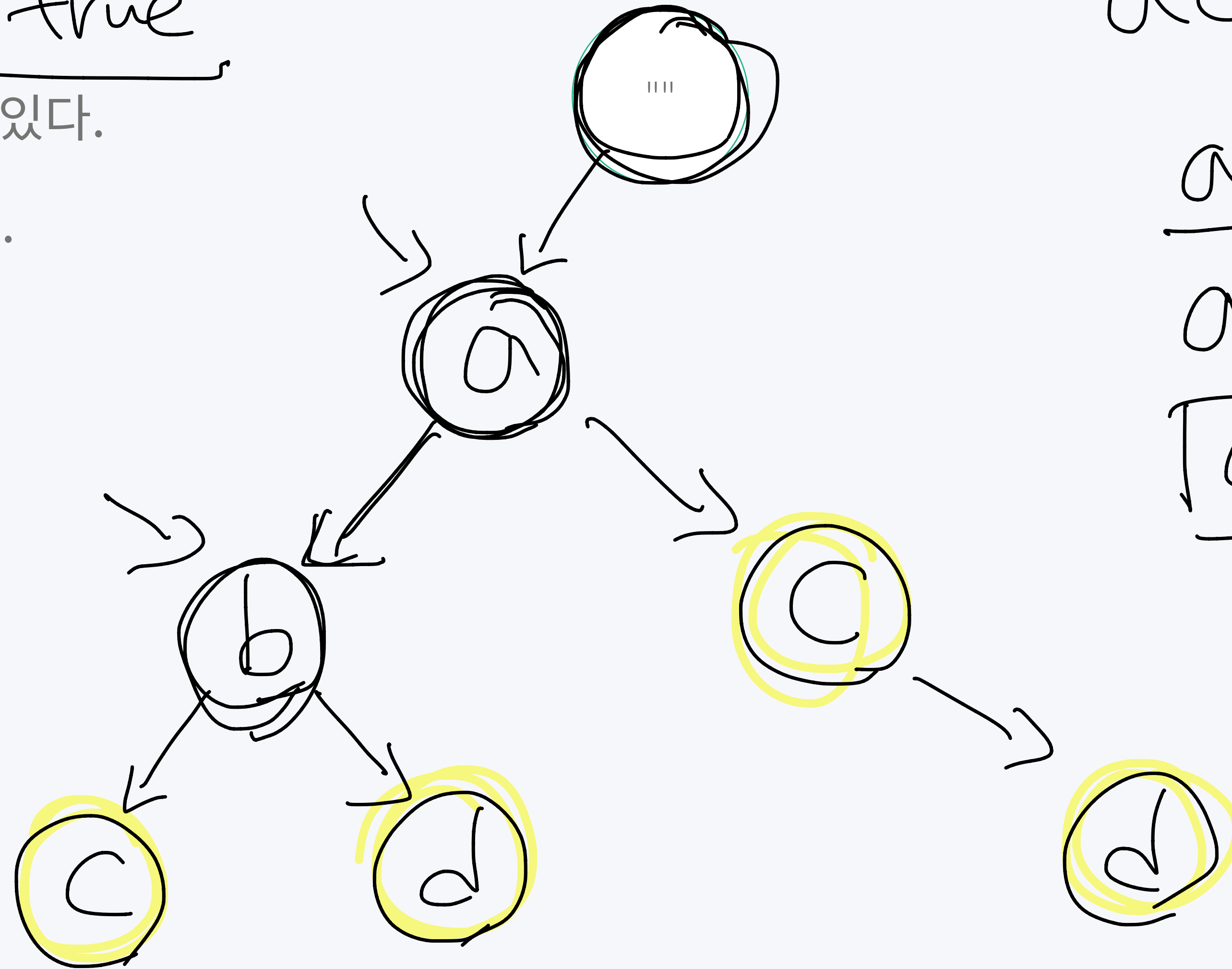
valid = true

- 트라이의 초기 상태: 루트만 있다.
- 여기에 "abc"를 추가해본다.

a ab
abc abd

ab

abc
abd



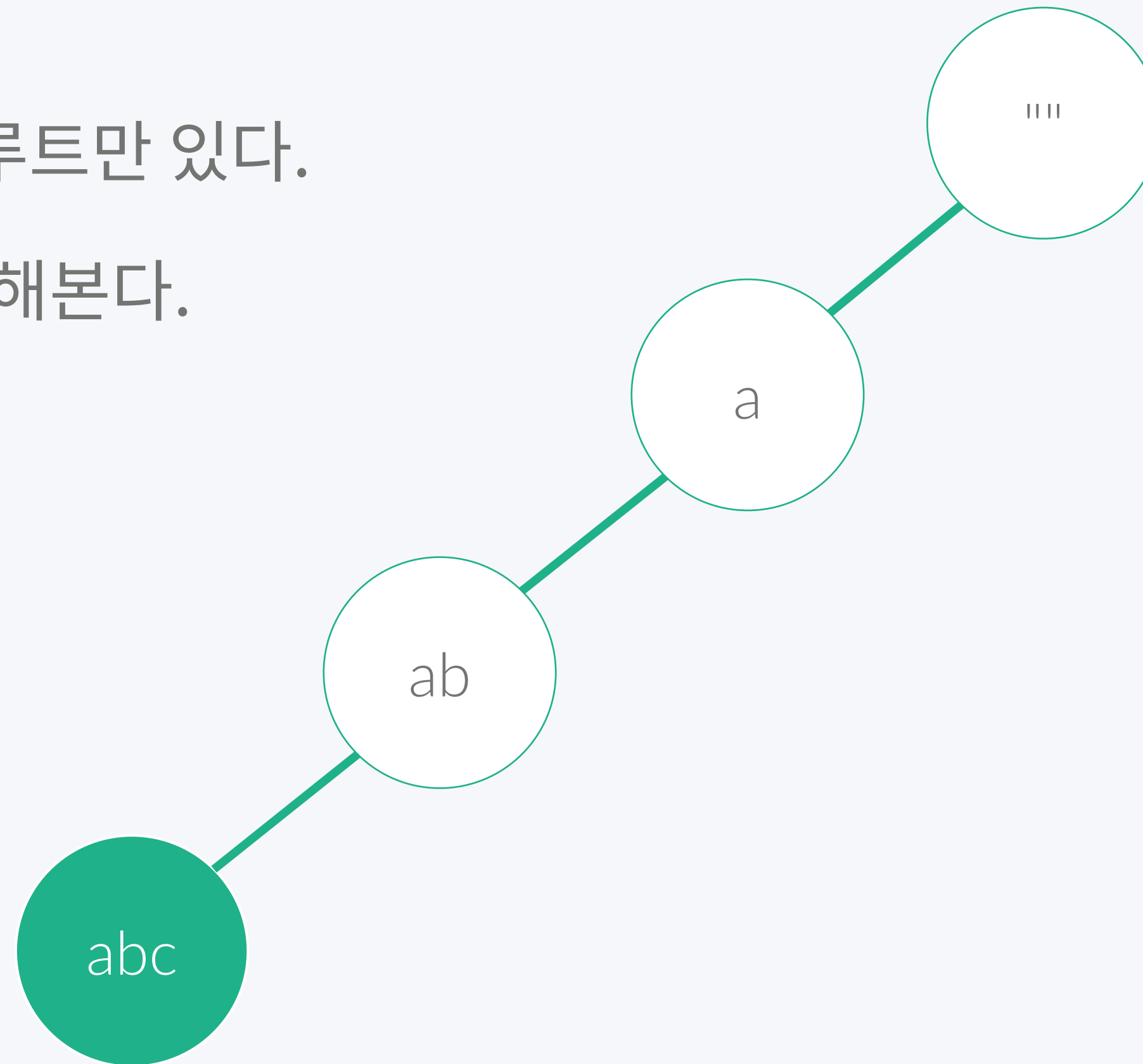
abd
acd

abe
abc
ac

Trie

Trie

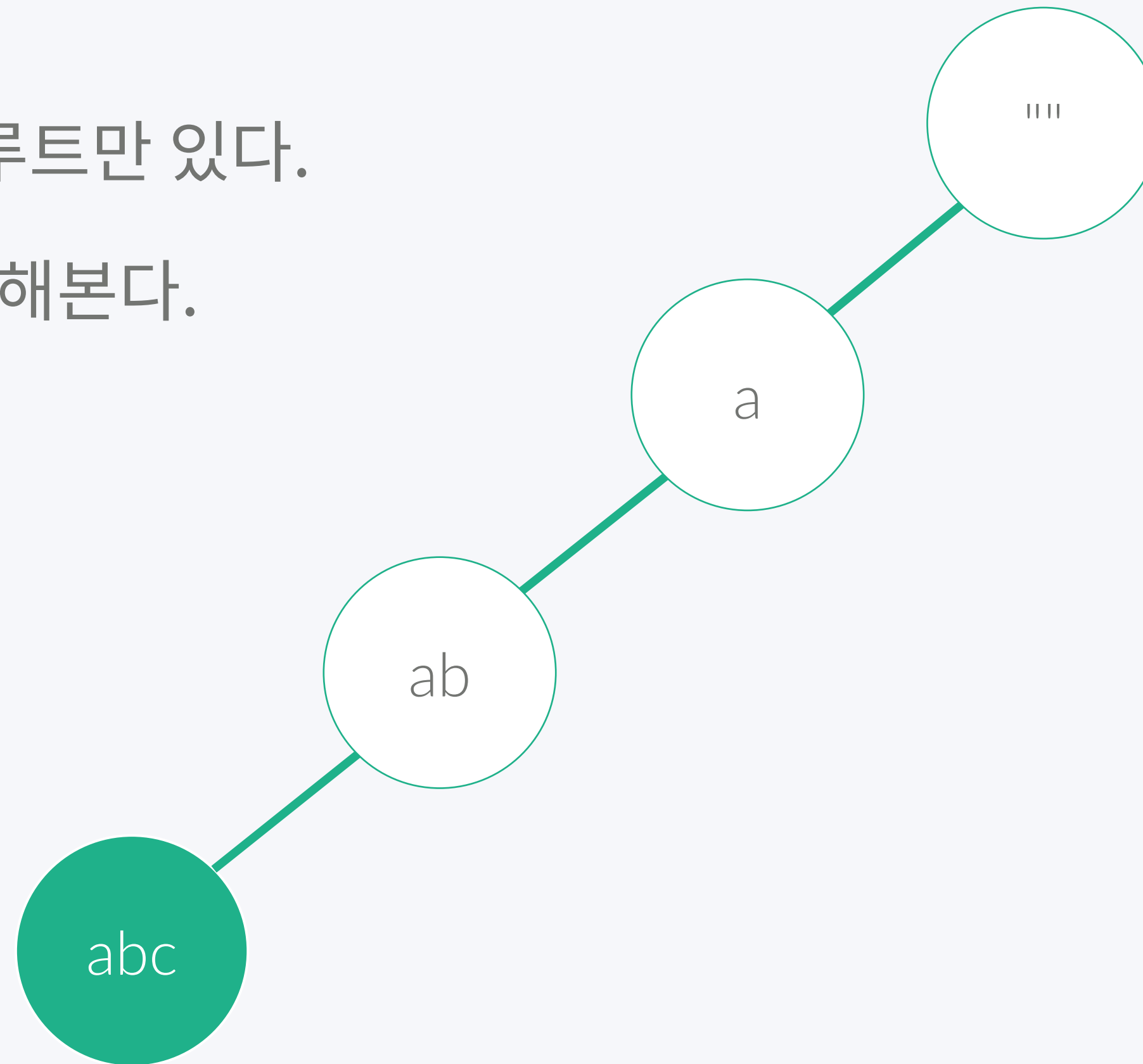
- 트라이의 초기 상태: 루트만 있다.
- 여기에 "abc"를 추가해본다.



Trie

Trie

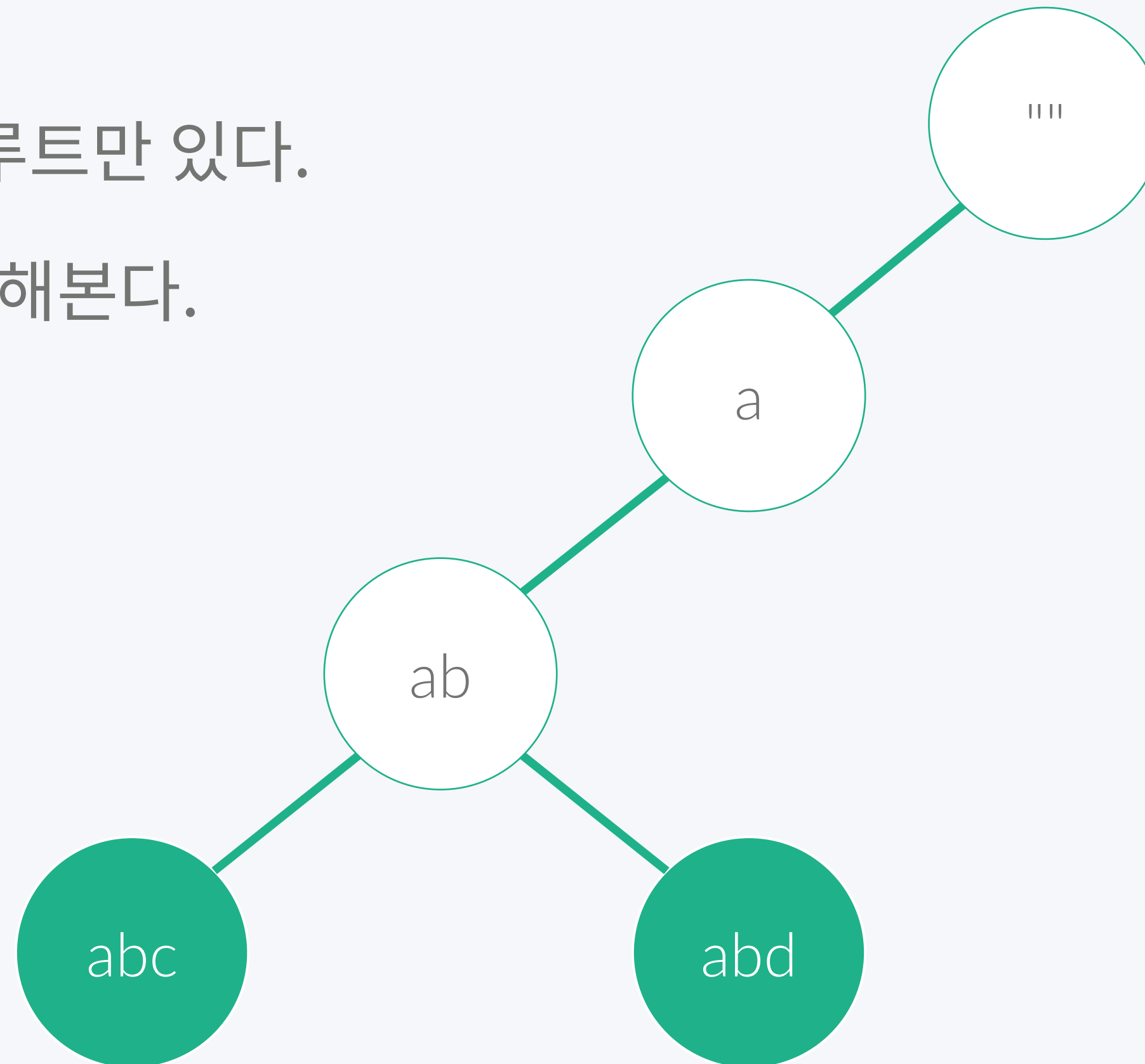
- 트라이의 초기 상태: 루트만 있다.
- 여기에 "abd"를 추가해본다.



Trie

Trie

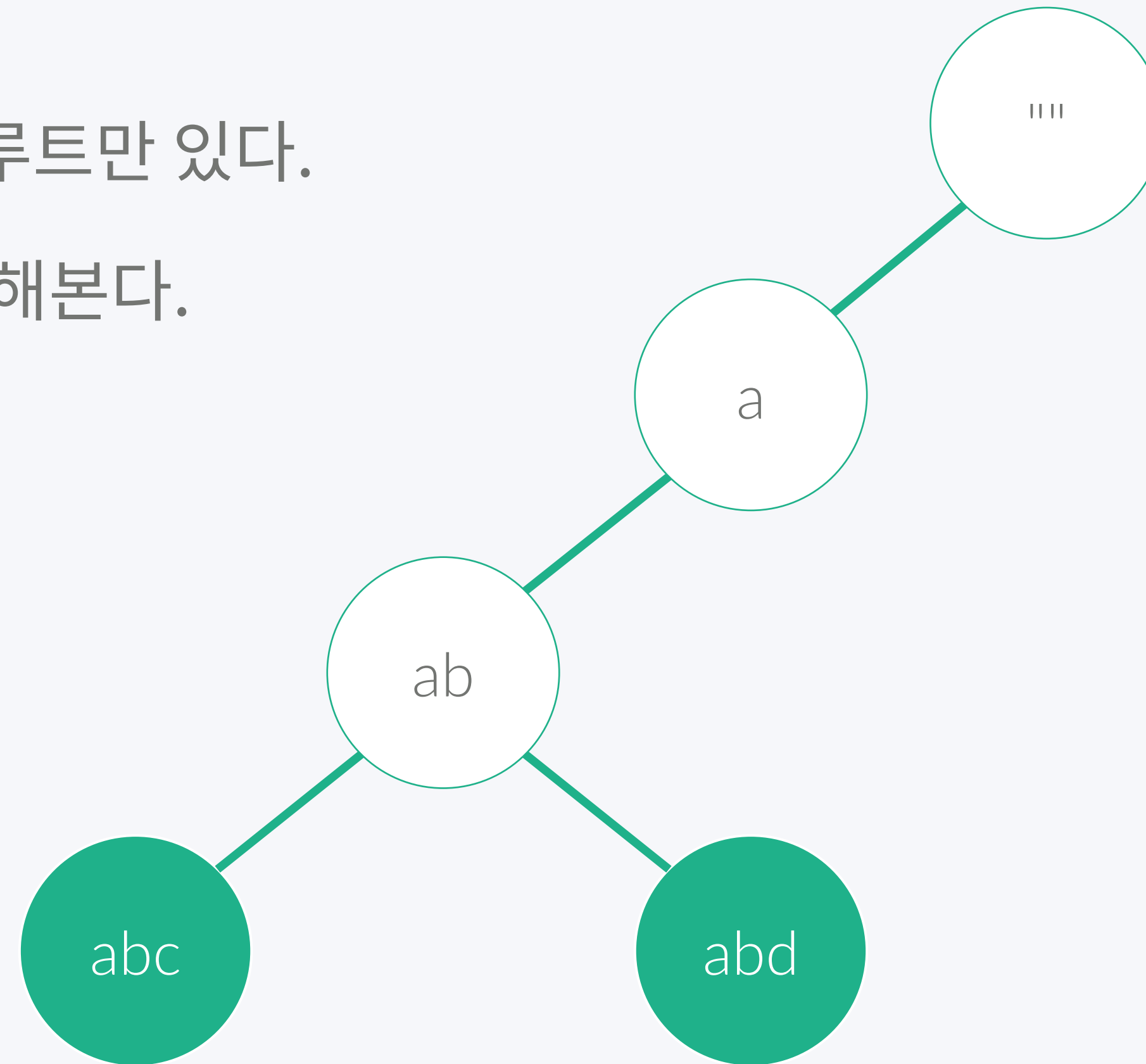
- 트라이의 초기 상태: 루트만 있다.
- 여기에 "abd"를 추가해본다.



Trie

Trie

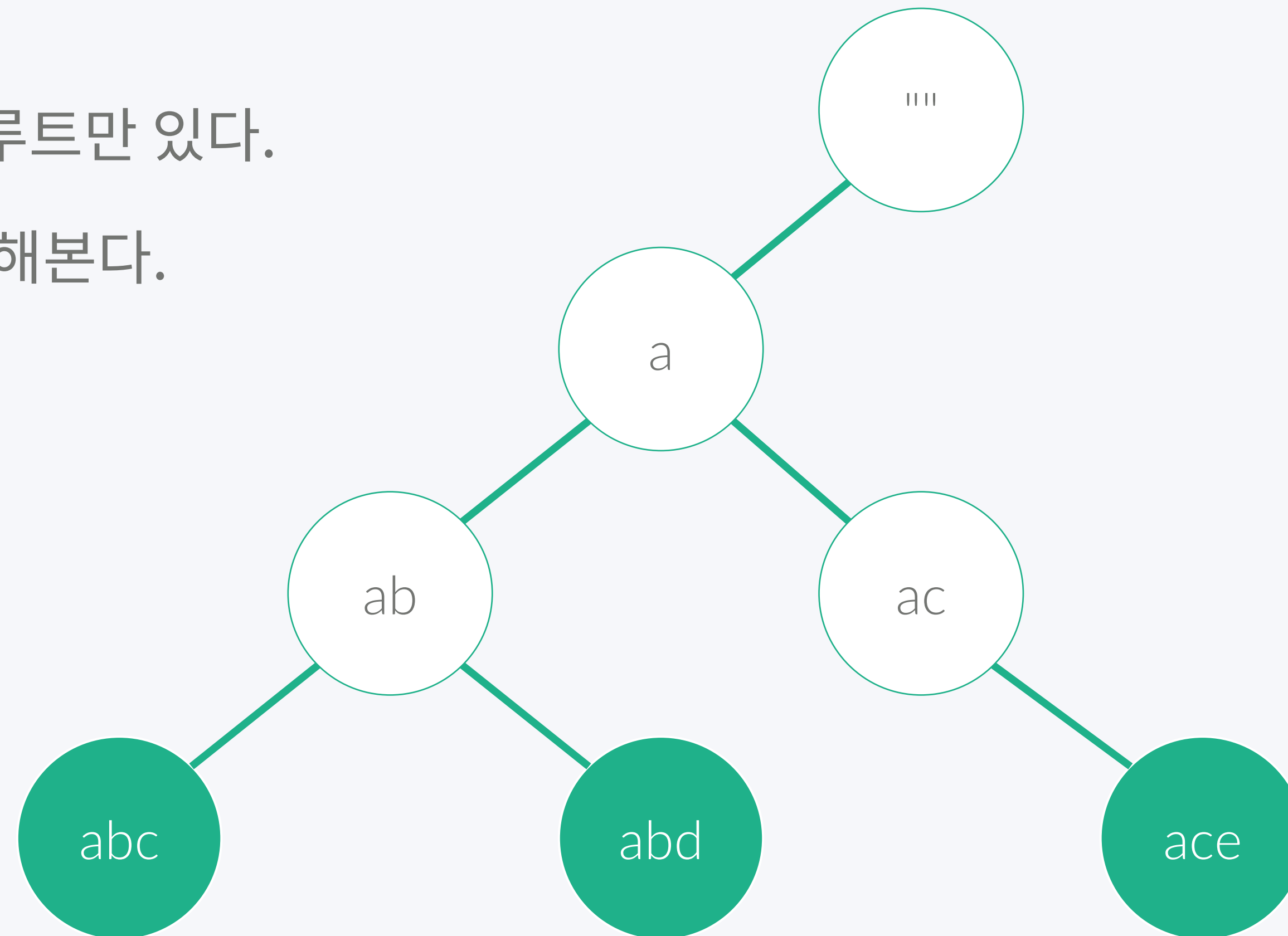
- 트라이의 초기 상태: 루트만 있다.
- 여기에 "ace"를 추가해본다.



Trie

Trie

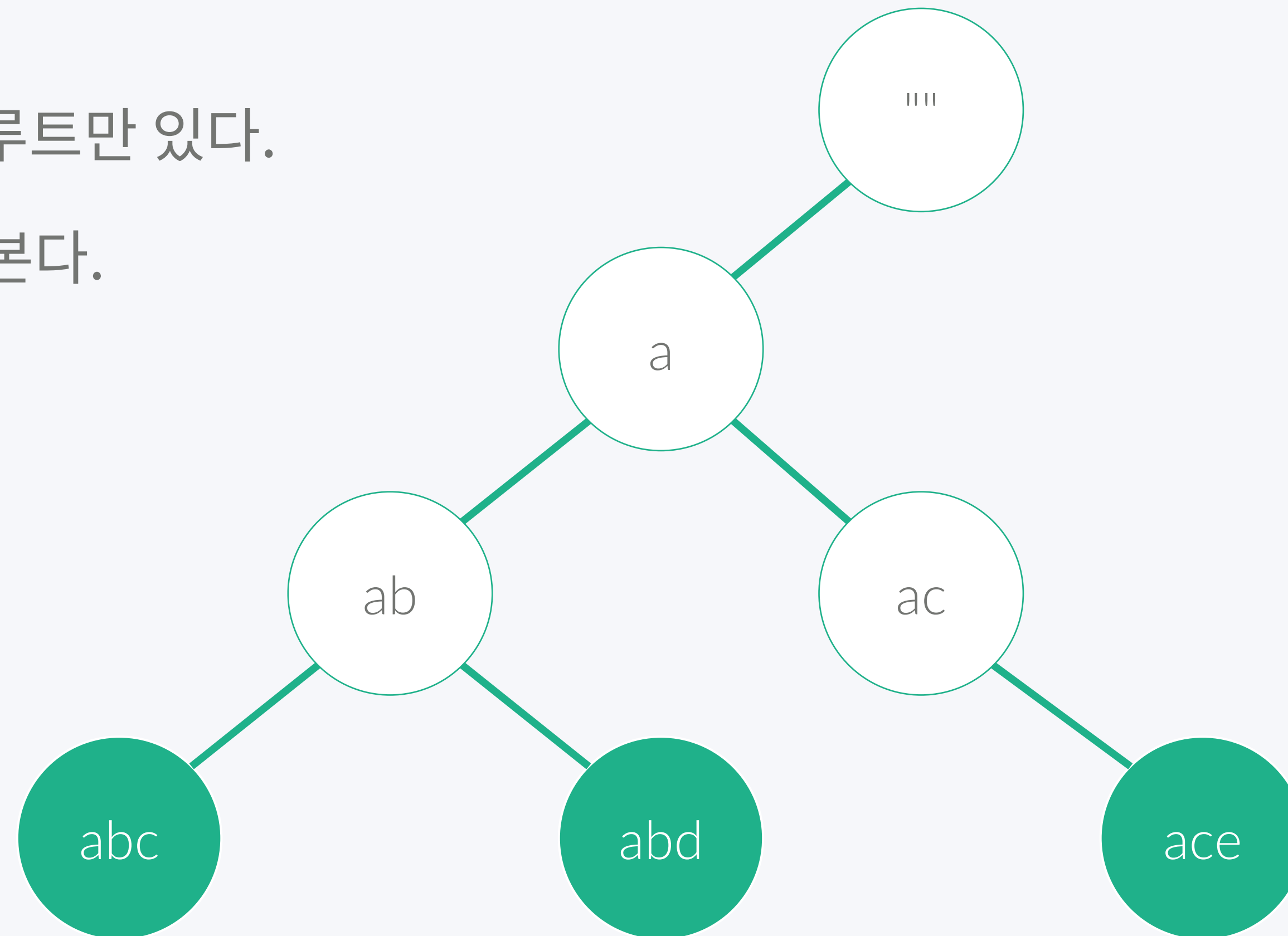
- 트라이의 초기 상태: 루트만 있다.
- 여기에 "ace"를 추가해본다.



Trie

Trie

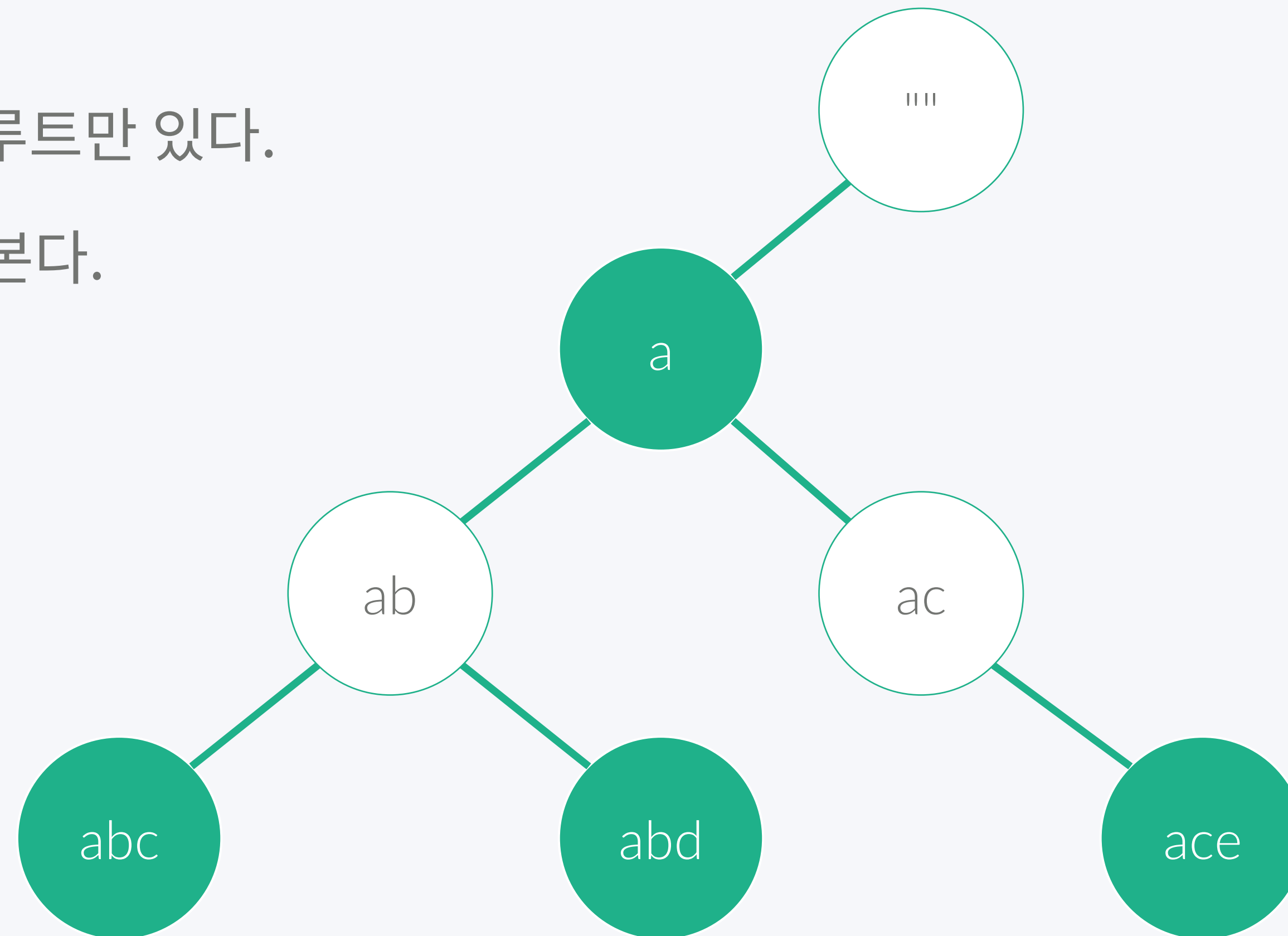
- 트라이의 초기 상태: 루트만 있다.
- 여기에 "a"를 추가해본다.



Trie

Trie

- 트라이의 초기 상태: 루트만 있다.
- 여기에 "a"를 추가해본다.



Trie

Trie

```
struct Node {  
    int children[26];  
    bool valid;  
    Node() {  
        for (int i = 0; i < 26; i++) {  
            children[i] = -1;  
        }  
        valid = false;  
    }  
};  
  
vector<Node> trie;  
int root;
```

Handwritten notes:

- 0 ~ 25까지 (next to children[26])
- true 37L (next to valid)
- false X (next to valid)
- 1 (circled and underlined next to children[i] = -1)
- valid = false; (underlined)

Trie

Trie

```
int init() {  
    Node x;  
    trie.push_back(x);  
    return (int)trie.size() - 1;  
}
```

Trie

Trie

```
void add(int node, string &s, int index) {  
    if (index == s.size()) {  
        trie[node].valid = true;  
        return;  
    }
```

depth
length

```
    int c = s[index] - 'a'; Children Hing
```

```
    if (trie[node].children[c] == -1) {  
        int next = init();
```

```
        trie[node].children[c] = next;
```

```
    }
```

```
    int child = trie[node].children[c];  
    add(child, s, index + 1);
```

```
}
```

Trie

Trie

트라이 네트워크
찾고 싶을 때

83

```
bool search(int node, string &s, int index) {  
    if (node == -1) return false;  
    if (index == s.length()) return trie[node].valid;  
    int c = s[index] - 'a';  
    int child = trie[node].children[c];  
    return search(child, s, index+1);  
}
```

문자열 집합

<https://www.acmicpc.net/problem/14425>

- 총 N개의 문자열로 이루어진 집합 S가 주어진다.
- 입력으로 주어지는 M개의 문자열 중에서 집합 S에 포함되어 있는 것이 총 몇 개인지 구하는 문제

Hash

26진수

abc

⇒

012₂₆

⇒

26 + 2

문자열 집합

<https://www.acmicpc.net/problem/14425>

- 소스: <http://boj.kr/8c4dda57491347c2b65e647fb0a6bc0b>
- 위의 소스에서 Java는 시간 초과가 나기 때문에, 참고만 하세요

접두사 찾기

<https://www.acmicpc.net/problem/14426>

- 총 N 개의 문자열로 이루어진 집합 S 가 주어진다.
- 입력으로 주어지는 M 개의 문자열 중에서 집합 S 에 포함되어 있는 문자열 중 적어도 하나의 접두사인 것의 개수를 구하는 문제

접두사 찾기

87

<https://www.acmicpc.net/problem/14426>

- 소스: <http://boj.kr/eba23e7a559148eab7acb264d5e0e97c>

Boggle

<https://www.acmicpc.net/problem/9202>

- 가능한 모든 단어를 미리 다 만들어보고 단어 사전에 들어있는 단어를 모두 찾아보는 방법
- 가능한 단어의 개수: 673108개

Boggle

<https://www.acmicpc.net/problem/9202>

- 소스: <http://boj.kr/d30815eb980045298b8e32c5ec5124c7>
- 너무 오랜 시간이 걸려서 시간 초과를 받는다.

Boggle

<https://www.acmicpc.net/problem/9202>

- Trie 구현!
- 소스: <http://boj.kr/7b2f198178304c7cb02f9b4c6a90098e>

전화번호 목록

<https://www.acmicpc.net/problem/5052>

- 한 번호가 다른 번호의 접두어이면 안되다

전화번호 목록

<https://www.acmicpc.net/problem/5052>

- 한 번호가 다른 번호의 접두어이면 안되다
- Trie = Prefix Tree

전화번호 목록

<https://www.acmicpc.net/problem/5052>

- 소스: <http://boj.kr/e418fd960c5347cc8a0c71fdf611d5a6>

두 수 XOR

<https://www.acmicpc.net/problem/13505>

- 크기가 N 인 배열이 주어졌을 때
- 두 수를 XOR한 값이 가장 큰 것을 찾는 문제

길이 $N \leq 10^5$

$[A[i] \wedge A[j]]$

최대값

$O(N^2)$



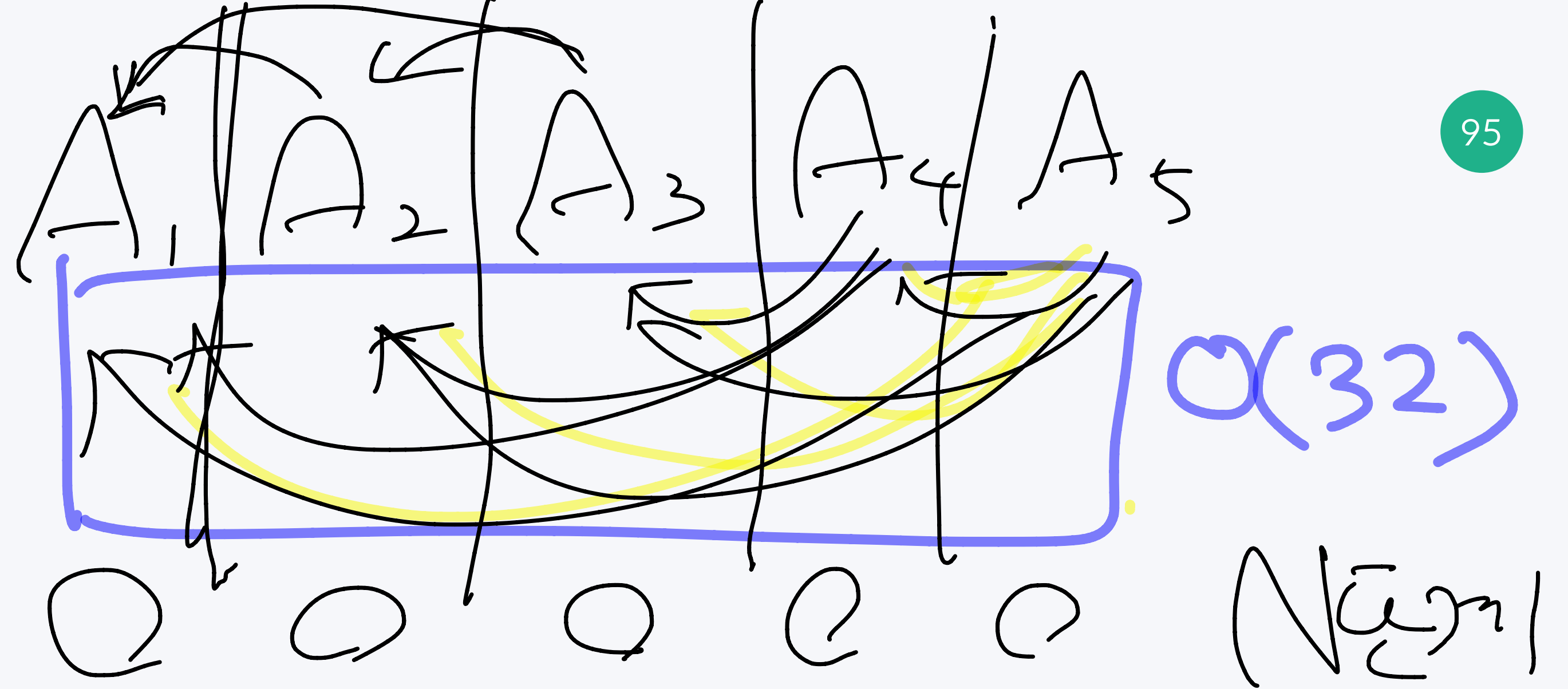
두 수 XOR

<https://www.acmicpc.net/problem/13505>

- Trie를 이용한다
- 두 가지 연산이 필요하다

1. 수 x 를 삽입한다

2. 새로운 수 Y 와 지금까지 삽입한 수와 XOR을 해서 가장 큰 것을 찾는다



두 수 XOR

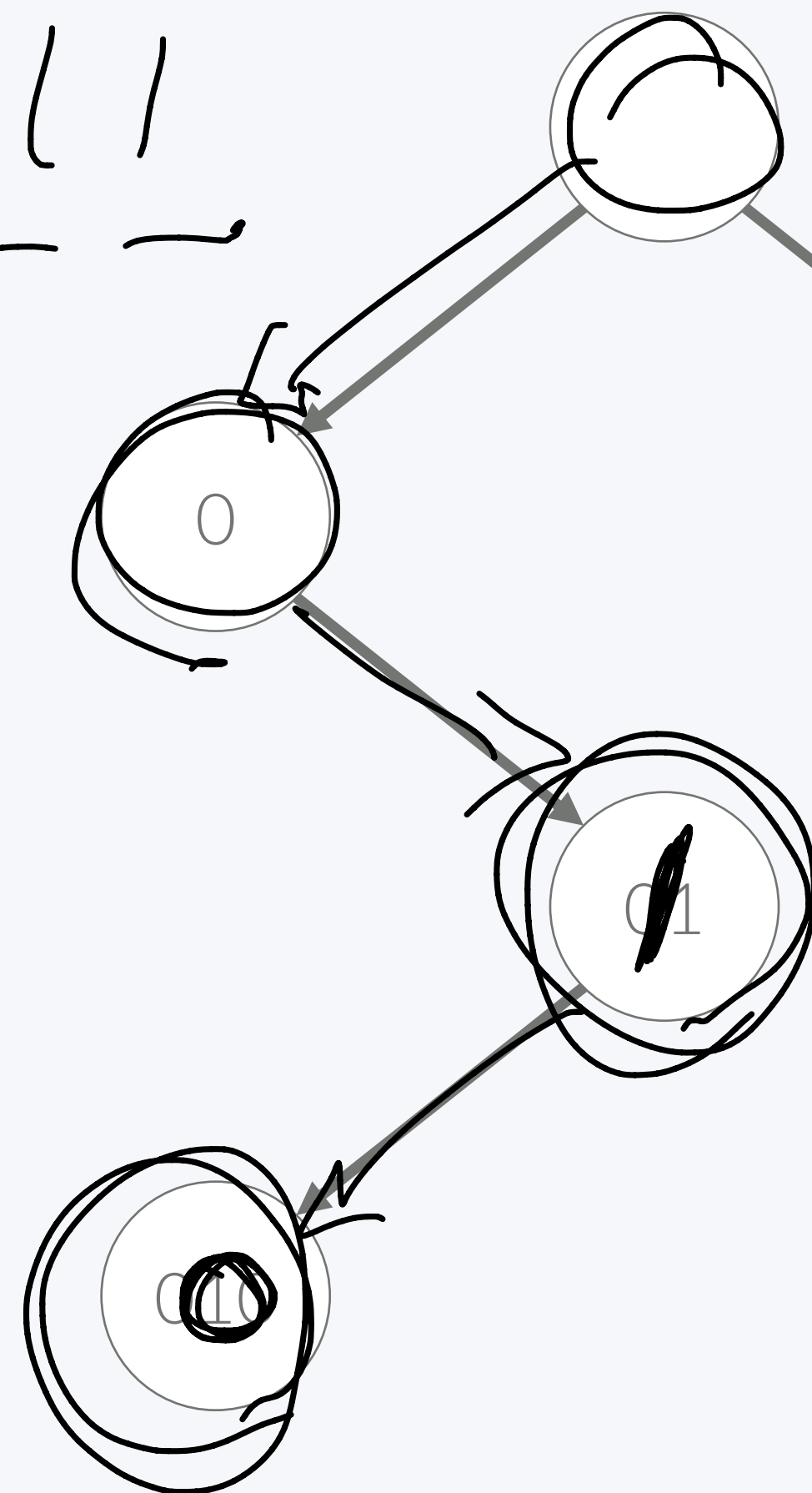
<https://www.acmicpc.net/problem/13505>

96

- 2(010)과 7(111)을 Trie에 넣었다고 가정하자

0 1 0 1 1 1
_ _ _ _ _ _

$O(32N)$



4와 XOR 해서
가장 큰 수

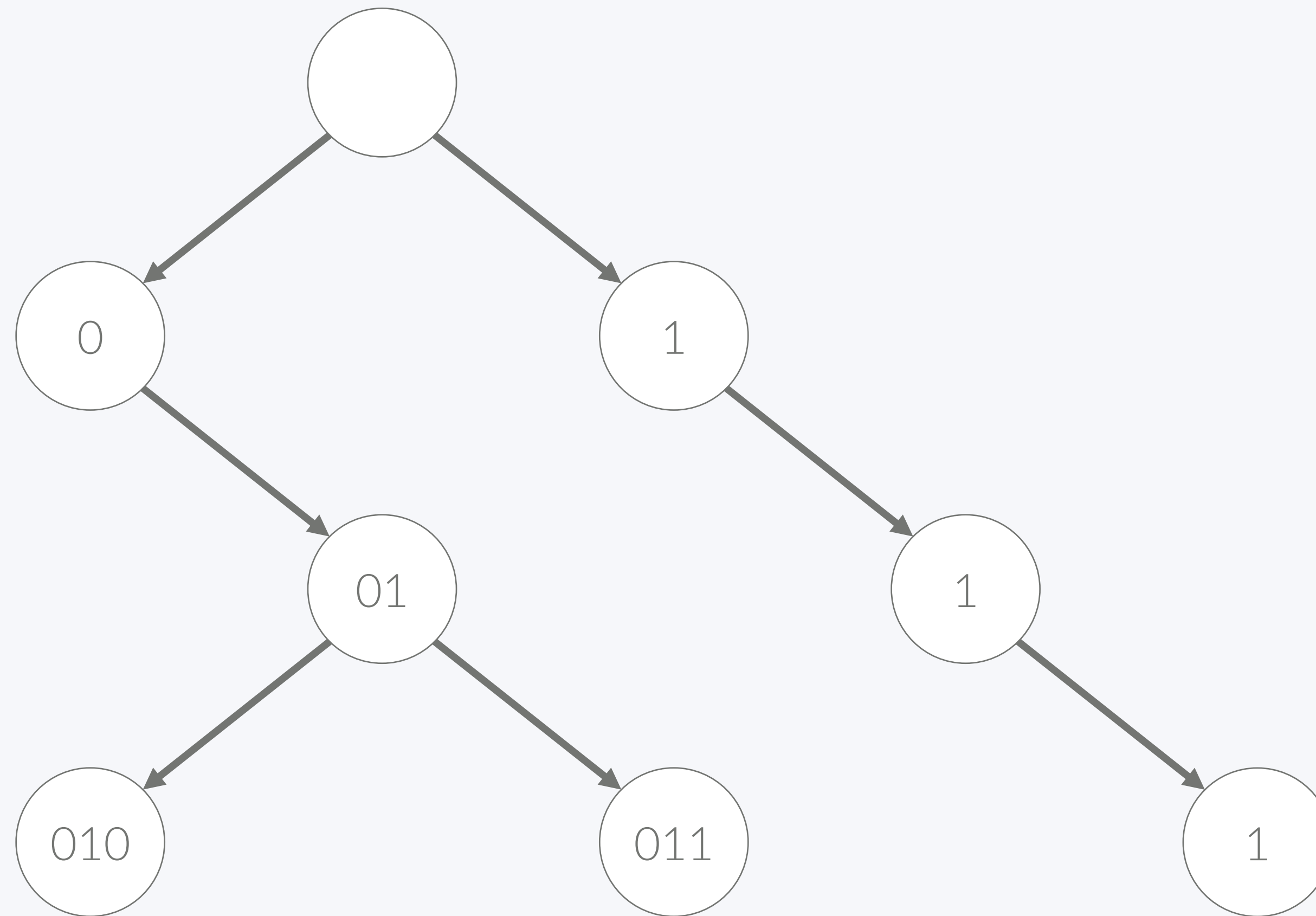
1 0 0
_ _ _
0 1 1

32

두 수 XOR

<https://www.acmicpc.net/problem/13505>

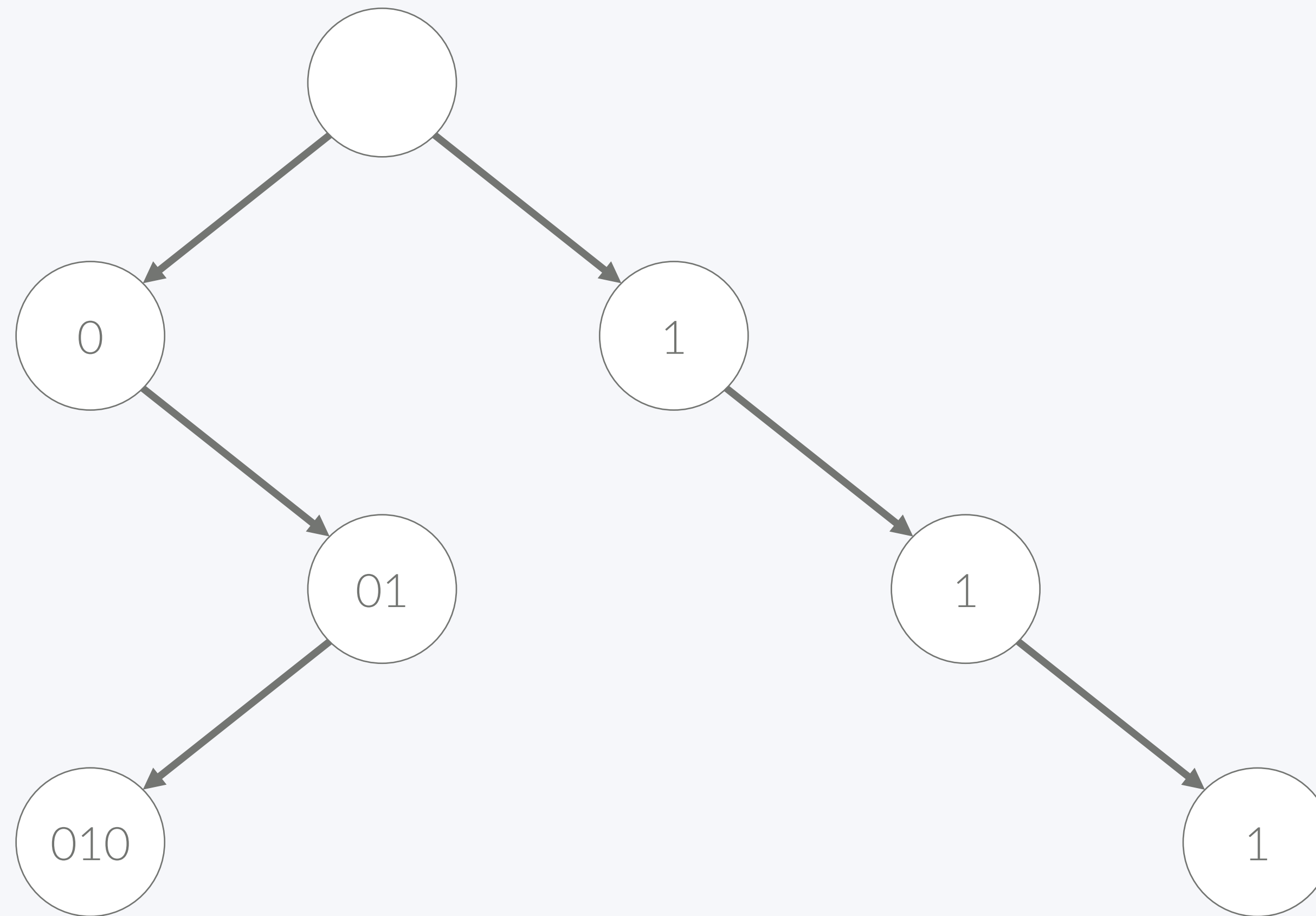
- 여기에 3(011)을 추가하면 트리가 다음과 같이 변한다



두 수 XOR

<https://www.acmicpc.net/problem/13505>

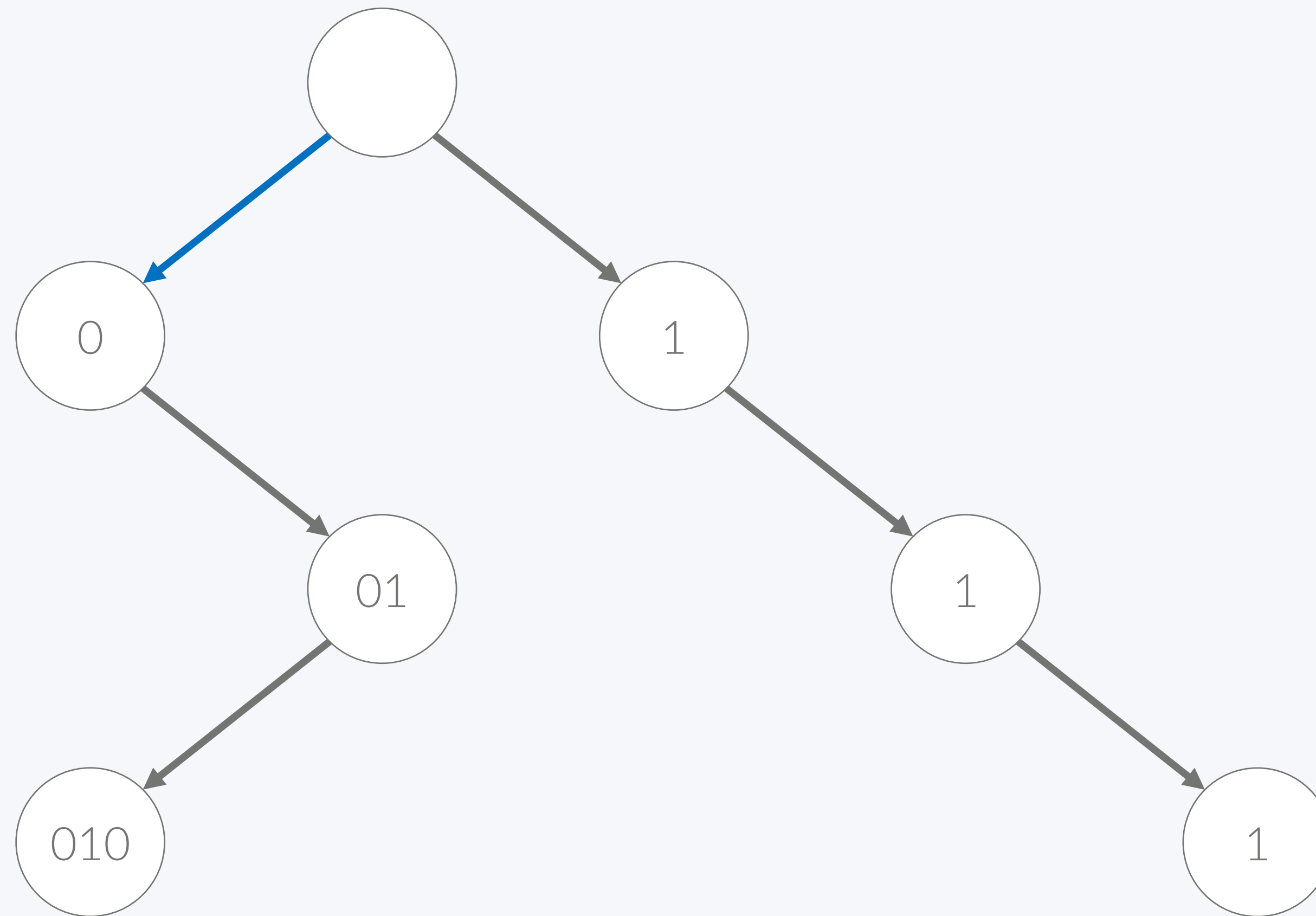
- Trie에 있는 값 중에서 4과 XOR해서 가장 큰 값을 찾으려면, 트리를 탐색하면 된다



두 수 XOR

<https://www.acmicpc.net/problem/13505>

- 4 = 100

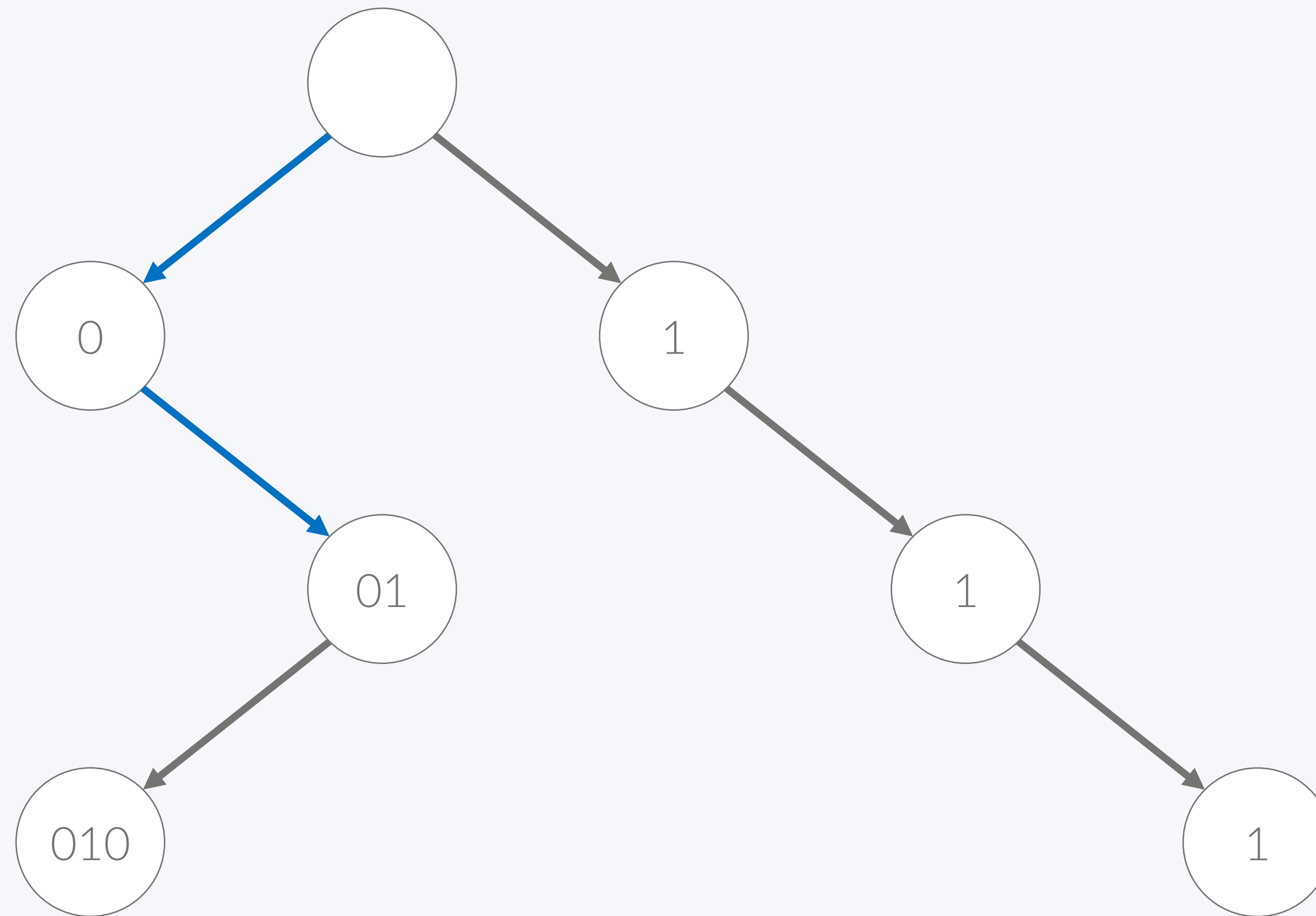


두 수 XOR

100

<https://www.acmicpc.net/problem/13505>

- 4 = 100

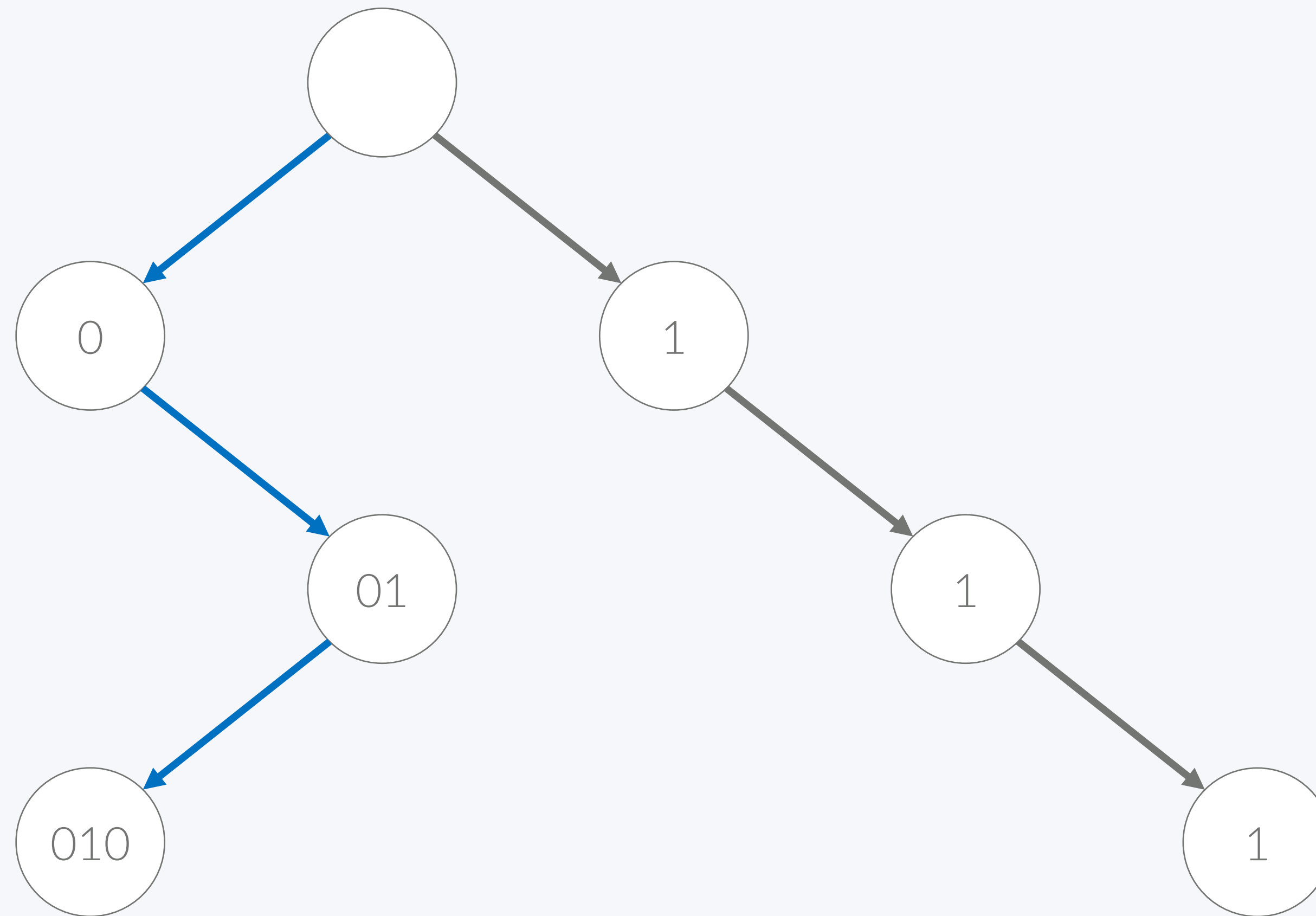


두 수 XOR

101

<https://www.acmicpc.net/problem/13505>

- 4 = 100



두 수 XOR

102

<https://www.acmicpc.net/problem/13505>

- 소스: <http://boj.kr/6c82bd11cc984263af6e29878703c81b>

XOR 합

<https://www.acmicpc.net/problem/13504>

- 수열 A의 연속된 부분 수열 중에서 XOR 합이 가장 큰 것을 찾는 문제
- XOR 합: 수열에 포함된 원소를 모두 XOR한 값

XOR 합

104

<https://www.acmicpc.net/problem/13504>

- 두 수 XOR과 비슷하게 푼다

XOR 합

105

<https://www.acmicpc.net/problem/13504>

- 소스: <http://boj.kr/78a290d07d03459688058d4b05989b16>

부분 수열 XOR

106

<https://www.acmicpc.net/problem/13445>

- 수열 A의 연속된 부분 수열 중에서 XOR 합이 K보다 작은 것의 개수를 세는 문제
- XOR 합: 수열에 포함된 원소를 모두 XOR한 값

부분 수열 XOR

107

<https://www.acmicpc.net/problem/13445>

- XOR 합과 비슷하게 푼다
- Trie의 각 노드에 자식에 몇 개의 노드가 있는지도 기록해 놓는다.

부분 수열 XOR

108

<https://www.acmicpc.net/problem/13445>

```
int query(int node, int num, int k, int bit) {
    if (bit == -1) return 0;
    int c1 = (k >> bit) & 1;
    int c2 = (num >> bit) & 1;
    int ans = 0;
    if (c1 == 1) {
        ans += trie[node].cnt[c2];
        c2 = 1-c2;
    }
    if (trie[node].children[c2] == -1) return ans;
    ans += query(trie[node].children[c2], num, k, bit-1);
    return ans;
}
```

부분 수열 XOR

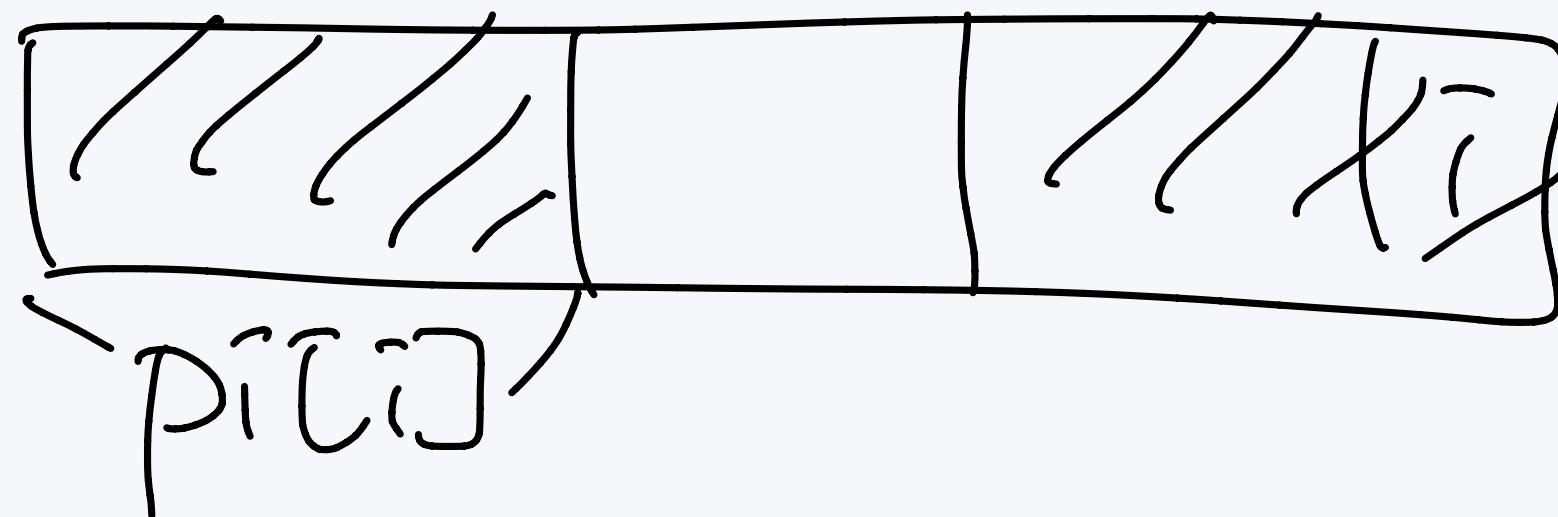
109

<https://www.acmicpc.net/problem/13445>

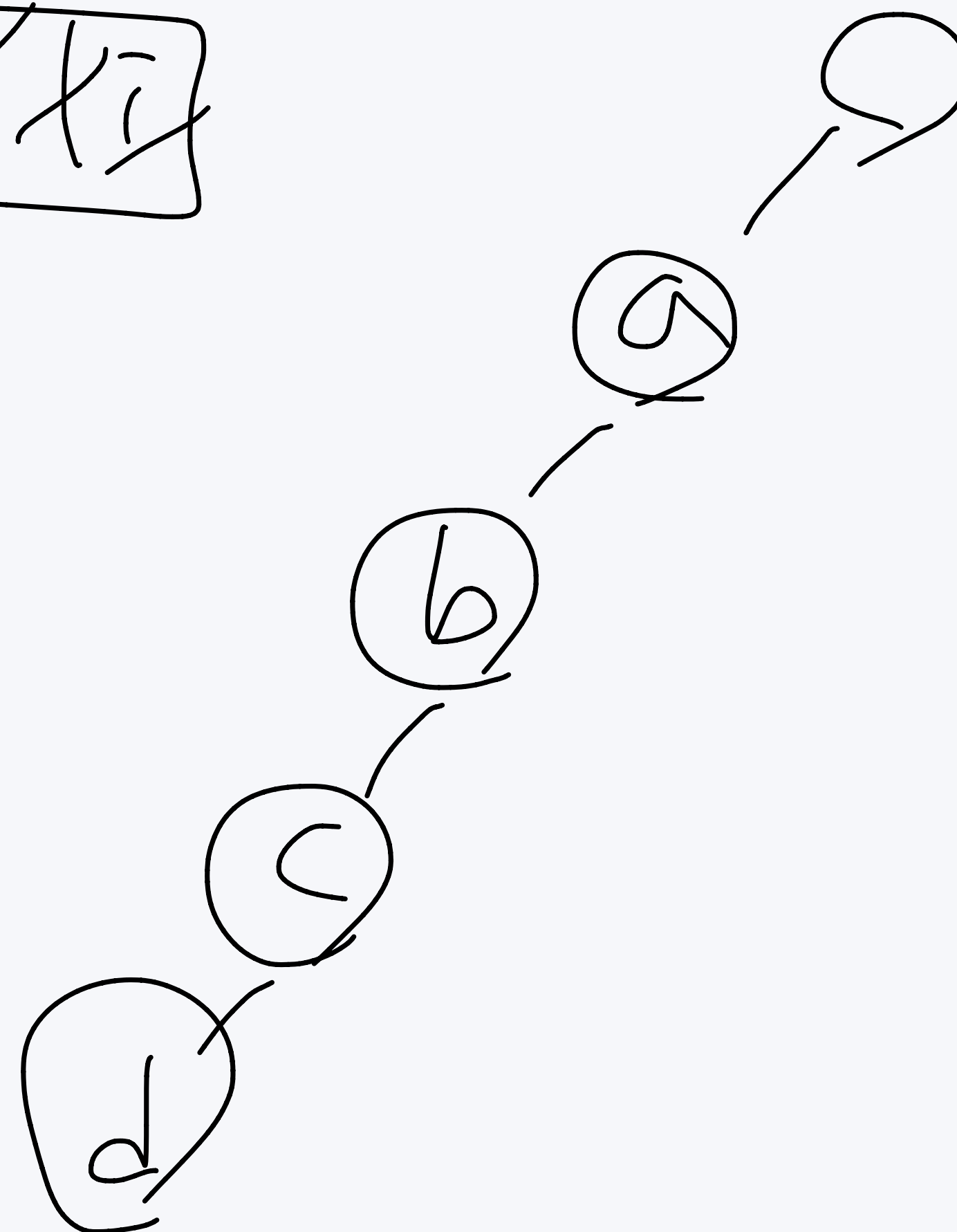
- 소스: <http://boj.kr/dfe1ae321165402e9cbadceb950664dc>

picu-

Trie



Aho-corasick



Aho-corasick

111

Aho-corasick

- KMP에서의 pi를
- Trie에서도 구현하는 것
- $\text{pi}[\text{node}] = \text{node가 나타내는 문자열 } s \text{의 suffix이면서 trie에 포함된 가장 긴 문자열}$

Aho-corasick

Aho-corasick

- S = "ABCDEABABABABCD"
- 에서
- "ABAB", "AD", "ABC", "BCD", "ABABC"가 몇 개 인지 찾는 문제

- KMP를 총 5번 돌린다?

M

$$S \text{ or } P \quad O(|S| + |P|)$$

$$KMP \quad O(M \times (|S| + |P|))$$

$$O(|S| + M \times |P|)$$

Aho-corasick

113

Aho-corasick

- $S = \text{"ABCDEABABABABCD"}$
- 에서
- “ABAB”, “AD”, “ABC”, “BCD”, “ABABC”가 몇 개 인지 찾는 문제
- KMP를 총 5번 돌린다?
- ABAB: 0 0 1 2
- AD: 0 0
- ABC: 0 0 0
- BCD: 0 0 0
- ABABC: 0 0 1 2 0

Aho-corasick

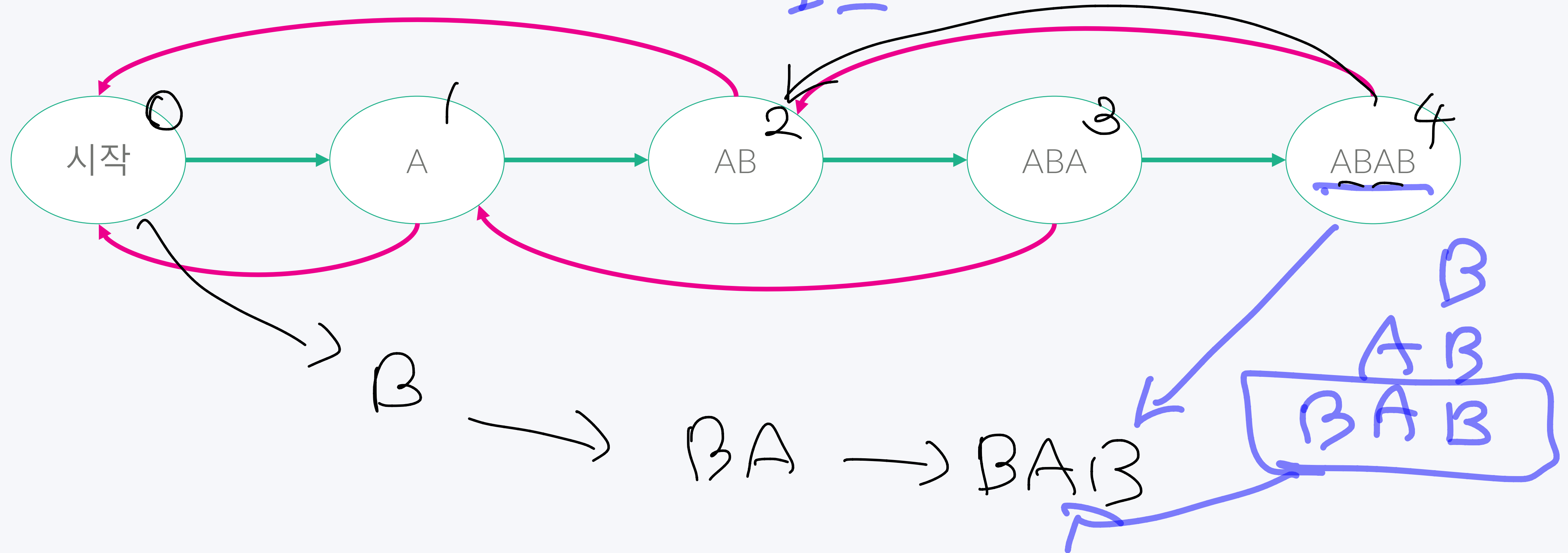
Aho-corasick

Trie: Self prefix

BAAB

ST[i]	A	B	A	B
P[i]	0	0	1	2

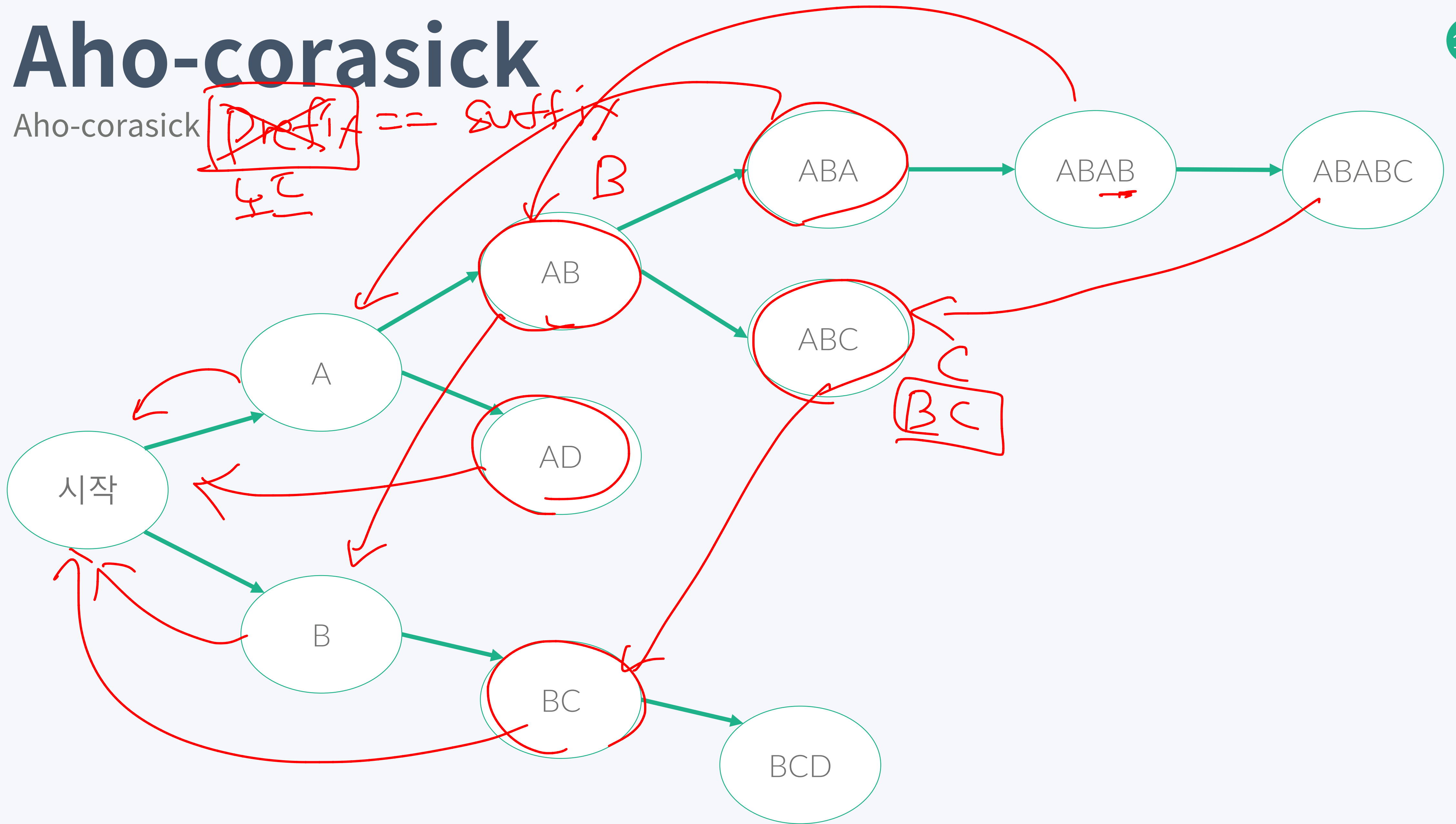
Prefix == Suffix 가랑 같아



Aho-corasick

Aho-corasick

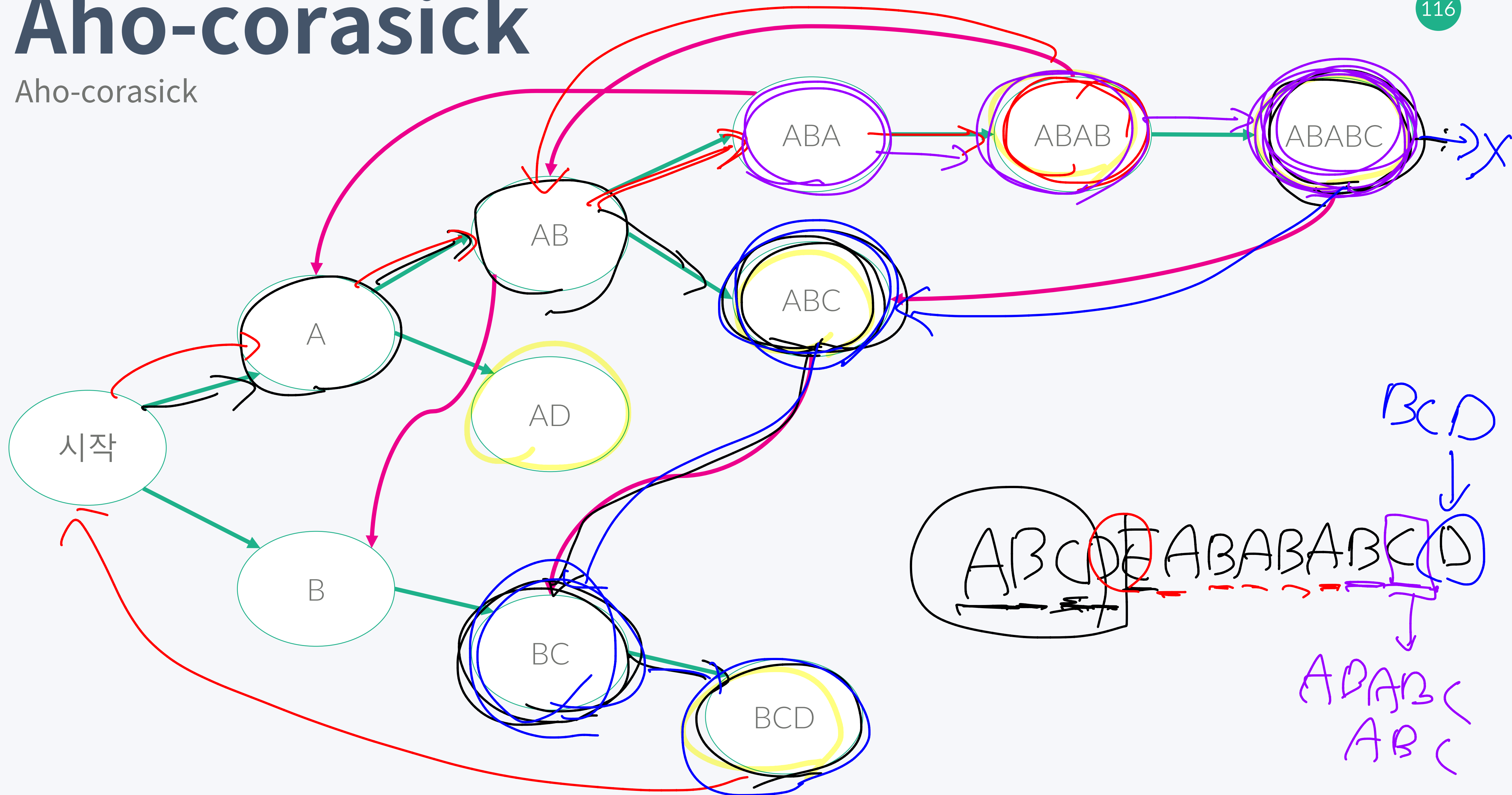
115



Aho-corasick

Aho-corasick

116



Aho-corasick

Aho-corasick

117

- 소스: <http://boj.kr/6ebaaa6d572f41baafdb0afe58209119>

문자열 집합 판별

118

<https://www.acmicpc.net/problem/9250>

- 소스: <http://boj.kr/c8b750b619614d97b73083c0cf51c44c>

돌연변이

119

<https://www.acmicpc.net/problem/10256>

- 소스: <http://boj.kr/2ea4c81768e84e8bbb5957b2ad4a5271>