

# Graph 1 and Dynamic Programming

2016. 12. 20.

신현규



# 지난 시간 요약

- Dynamic Programming
  1. Table을 정의한다
  2. 점화식을 구한다
  3. 어느 순서로 Table을 채울지 확인한다
  4. 정답이 어디에 있는지 찾는다
- 많은 예제를 풀어보는게 가장 좋다



# [활동문제 0] 짜장, 짬뽕, 볶음밥

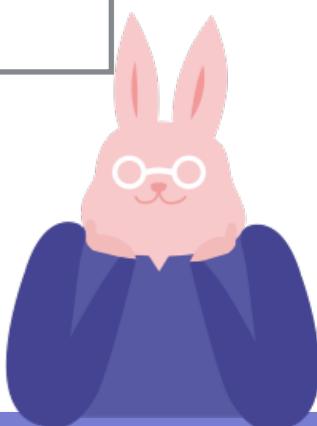
- 매일 짜장, 짬뽕, 볶음밥의 선호도가 다르며, 전날 먹은건 오늘 먹지 않는다. 만족도를 최대화 하라.

입력의 예

```
3  
27 8 35  
18 36 10  
7 22 45
```

출력의 예

```
116
```



# 커리큘럼

1. 재귀호출, 추상화
2. 시간복잡도, 알고리즘 정확성 증명, 자료구조
3. 분할정복법, 탐욕적 기법
4. 동적계획법 1
5. **동적계획법 2**
6. 그래프 이론 1
7. 그래프 이론 2
8. 세계 여러 기업의 입사 인터뷰 문제 도전 (+ NP-Complete)



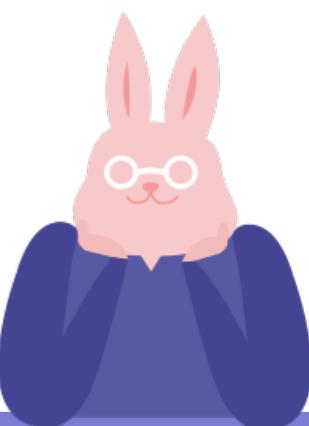
# 동적계획법 (Dynamic Programming)

- 부분문제를 푼 결과를 이용하여 전체문제를 푸는 방법
- 재귀호출 및 분할정복법과 느낌이 비슷합니다



# 동적계획법 문제풀이 순서

1. Table을 정의한다
2. 점화식을 구한다
3. 어느 순서로 Table을 구해야 하는지를 생각한다
4. 답이 어디에 있는지를 찾는다

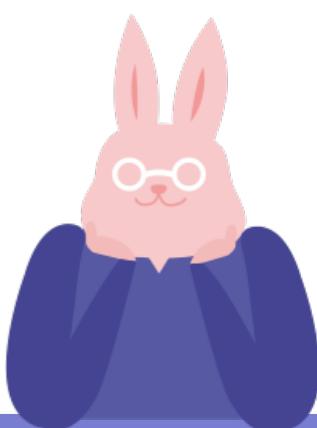


# [연습문제] 최대구간의 합 구하기

1. Table을 정의한다

- $T(i) = \underline{i\text{번째 수를 끝으로 하는}} \text{ 최대 구간의 합}$

data	2	1	-2	5	-10	3	2	5	-3	7	9	-10
T	2	3	1	6	-4	3	5	10	7	14	23	13



# [연습문제] 최대 3 2 의 합 구하기

- ## 1. Table을 정의한다

- $T(i) = i$  번째  $\leftarrow -2$

## 2. 점화식

						2
					3	2
				-10	3	2
			5	-10	3	2
		-2	5	-10	3	2
	1	-2	5	-10	3	2
	2	1	-2	5	-10	3

# 구간의 합

# data

2	1	-2	5	-10	3	2	5	-3	7	9	-10
2	3	1	6	-4	3	5	10	7	14	23	13



# [연습문제] 최대구간의 합 구하기

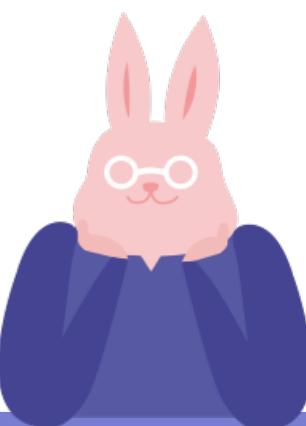
1. Table을 정의한다

- $T(i) = \underline{i\text{번째 수를 끝으로 하는}} \text{ 최대 구간의 합}$

2. 점화식을 구한다

- $T(i) = \max(\text{data}[i], T(i-1) + \text{data}[i])$

data	2	1	-2	5	-10	3	2	5	-3	7	9	-10
T	2	3	1	6	-4	3	5	10	7	14	23	13



# [연습문제] 최대구간의 합 구하기

1. Table을 정의한다
  - $T(i) = \underline{i\text{번째 수를 끝으로 하는}}$  최대 구간의 합
2. 점화식을 구한다
  - $T(i) = \max(\text{data}[i], T(i-1) + \text{data}[i])$
3. 어느 순서로 Table을 구해야 하는지를 생각한다



# [연습문제] 최대구간의 합 구하기

1. Table을 정의한다
  - $T(i) = \underline{i\text{번째 수를 끝으로 하는}}$  최대 구간의 합
2. 점화식을 구한다
  - $T(i) = \max(\text{data}[i], T(i-1) + \text{data}[i])$
3. 어느 순서로 Table을 구해야 하는지를 생각한다
  - $i=0 \rightarrow i=n$



# [연습문제] 최대구간의 합 구하기

4. 답은 어디에 있는지를 찾는다

- $\max(T(i))$

data	2	1	-2	5	-10	3	2	5	-3	7	9	-10
T	2	3	1	6	-4	3	5	10	7	14	23	13



# [연습문제] 최대구간의 합 구하기

- 시간복잡도는?

data	2	1	-2	5	-10	3	2	5	-3	7	9	-10
T	2	3	1	6	-4	3	5	10	7	14	23	13



# [연습문제] 최대구간의 합 구하기

- 시간복잡도는?
  - $O(n)$

data	2	1	-2	5	-10	3	2	5	-3	7	9	-10
T	2	3	1	6	-4	3	5	10	7	14	23	13



# [연습문제] 계단 오르기

- n칸의 계단이 있고, 한 번에 최대 3칸까지 오를 수 있다.  
n칸을 오르는 경우의 수는 ?

입력의 예

3

출력의 예

7

5

13



# [연습문제] 계단 오르기

1. Table을 정의한다

- $T(i)$  = 계단  $i$ 칸을 오를 때의 경우의 수

2. 점화식을 구한다

- 계단  $i$ 칸을 오르는 경우

- 가장 마지막에 1칸을 오르는 경우

- 가장 마지막에 2칸을 오르는 경우

- 가장 마지막에 3칸을 오르는 경우

1 1 1 1 1

1 1 2 1

1 2 1 1

2 1 1 1

1 3 1

3 1 1

4 1

1 1 1 2

1 2 2

2 1 2

3 2

1 1 3

2 3



# [연습문제] 계단 오르기

1. Table을 정의한다

- $T(i)$  = 계단  $i$ 칸을 오를 때의 경우의 수

2. 점화식을 구한다

- $T(i) = T(i-1) + T(i-2) + T(i-3)$



# [연습문제] 계단 오르기

1. Table을 정의한다
  - $T(i)$  = 계단  $i$ 칸을 오를 때의 경우의 수
2. 점화식을 구한다
  - $T(i) = T(i-1) + T(i-2) + T(i-3)$
3. 어느 순서로 Table을 구해야 하는지를 생각한다
  - $i=0 \rightarrow i=n$
4. 답이 어디에 있는지를 찾는다
  - $T(n)$



# [연습문제] 포도주 마시기

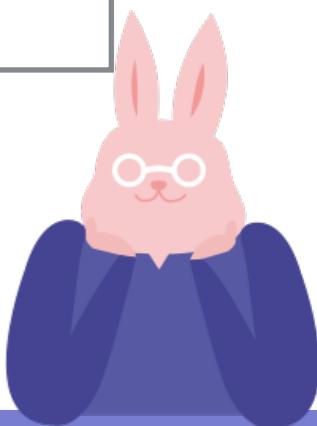
- n잔의 포도주가 있을 때, 마시는 포도주의 양을 최대화 하라  
단, 연속하여 3잔을 모두 마실 수는 없다

입력의 예

```
6  
6 10 13 9 8 1
```

출력의 예

```
33
```



# [연습문제] 포도주 마시기

1. Table을 정의한다



# [연습문제] 포도주 마시기

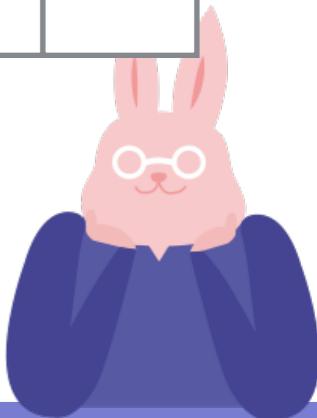
1. Table을 정의한다

- $T(i) = 1 \sim i$  까지의 포도주가 있을 때, 마시는 최댓값

data

T

6	10	13	9	8	1



# [연습문제] 포도주 마시기

1. Table을 정의한다
    - $T(i) = 1 \sim i$  까지의 포도주가 있을 때, 마시는 최댓값

	X					
		X	1			
	X	8	1			
data	6	10	13	9	8	1
T						



# [연습문제] 포도주 마시기

1. Table을 정의한다

- $T(i) = 1 \sim i$  까지의 포도주가 있을 때, 마시는 최댓값

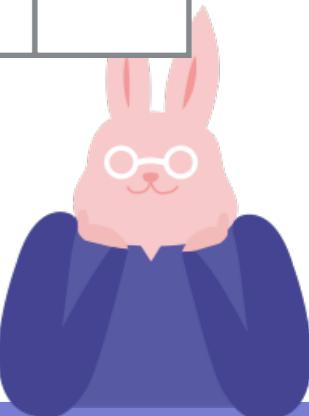
2. 점화식을 구한다

- $T(i) = \max(T(i-1), T(i-2) + \text{data}(i), T(i-3) + \text{data}(i) + \text{data}(i-1))$

data

6	10	13	9	8	1

T



# [연습문제] 특별한 이진수

- 길이가 n인 특별한 이진수의 개수를 구하여라  
특별한 이진수 : 1로 시작하며, 1이 두 번 연속으로 나타나지 않는 이진수

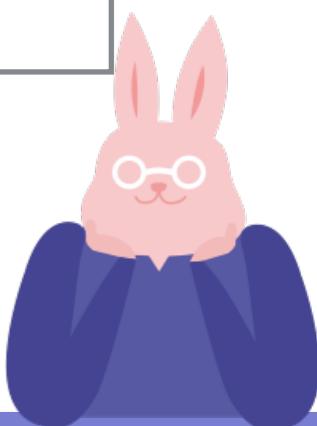
입력의 예

4

1000    1001    1010

출력의 예

3



# [연습문제] 특별한 이진수

1. Table을 정의한다

- $T(i, 0) =$  길이가  $i$ 이고, 맨 끝자리가 0인 특별한 이진수의 개수
- $T(i, 1) =$  길이가  $i$ 이고, 맨 끝자리가 1인 특별한 이진수의 개수

2. 점화식을 구한다

- $T(i, 0) = T(i-1, 0) + T(i-1, 1)$
- $T(i, 1) = T(i-1, 0)$

3. 어느 순서로 구해야 할지를 생각한다

- $i=0 \rightarrow i=n$



# [연습문제] 특별한 이진수

4. 답이 어디에 있는지를 찾는다

- $T(n, 0) + T(n, 1)$

5. 시간복잡도

- $O(n)$



# [연습문제] 특별한 이진수 (다른풀이)

1. Table을 정의한다

- $T(i) =$  길이가  $i$ 인 특별한 이진수의 개수

T	0	1	1	2	



# [연습문제] 특별한 이진수 (다른풀이)

1. Table을 정의한다

- $T(i) =$  길이가  $i$ 인 특별한 이진수의 개수

T	0	1	1	2	
	0	1			



# [연습문제] 특별한 이진수 (다른풀이)

1. Table을 정의한다

- $T(i) =$ 길이가  $i$ 인 특별한 이진수의 개수

2. 점화식을 구한다

- $T(i) = T(i-1) + T(i-2)$

3. 어느 순서로 구해야 할지를 생각한다

- $i=0 \rightarrow i=n$



# [연습문제] R개를 고르는 경우의 수

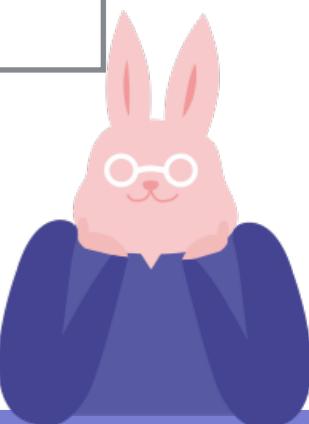
- N개 중에서 R개를 고르는 경우의 수를 구하여라

입력의 예

4 2

출력의 예

6



# [연습문제] R개를 고르는 경우의 수

1. Table을 정의한다

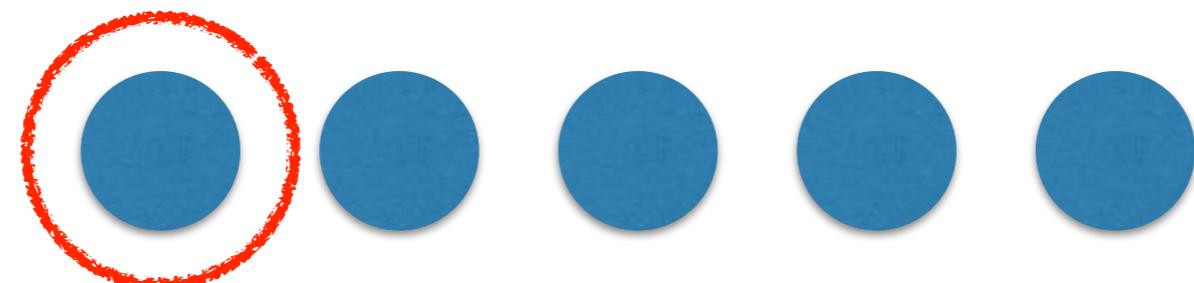
- $T(i, j) = i\text{개 중에서 } j\text{개를 고르는 경우의 수}$



# [연습문제] R개를 고르는 경우의 수

1. Table을 정의한다

- $T(i, j) = i\text{개 중에서 } j\text{개를 고르는 경우의 수}$



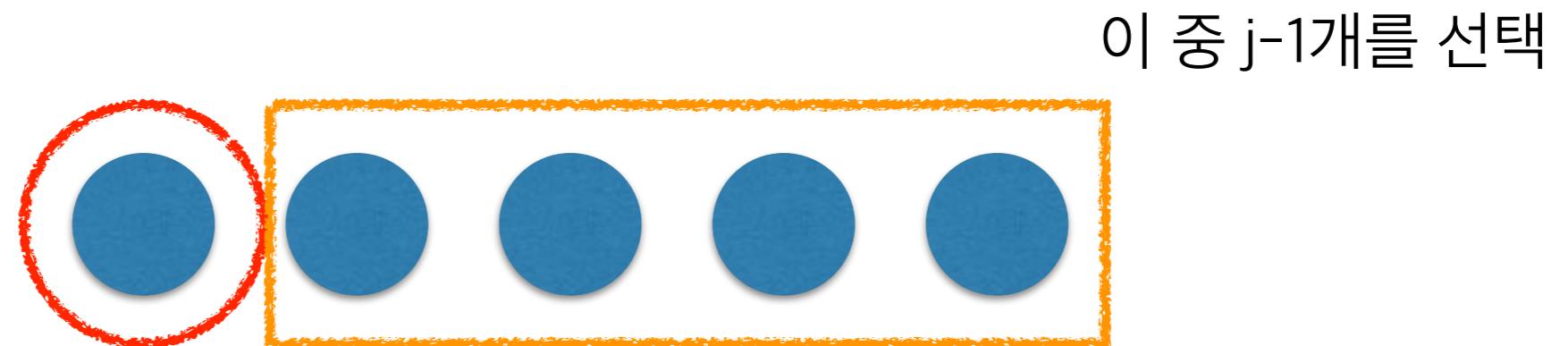
첫 번째 공을 고르는 경우와 고르지 않는 경우가 있다



# [연습문제] R개를 고르는 경우의 수

1. Table을 정의한다

- $T(i, j) = i$ 개 중에서  $j$ 개를 고르는 경우의 수



첫 번째 공을 고르는 경우와 고르지 않는 경우가 있다

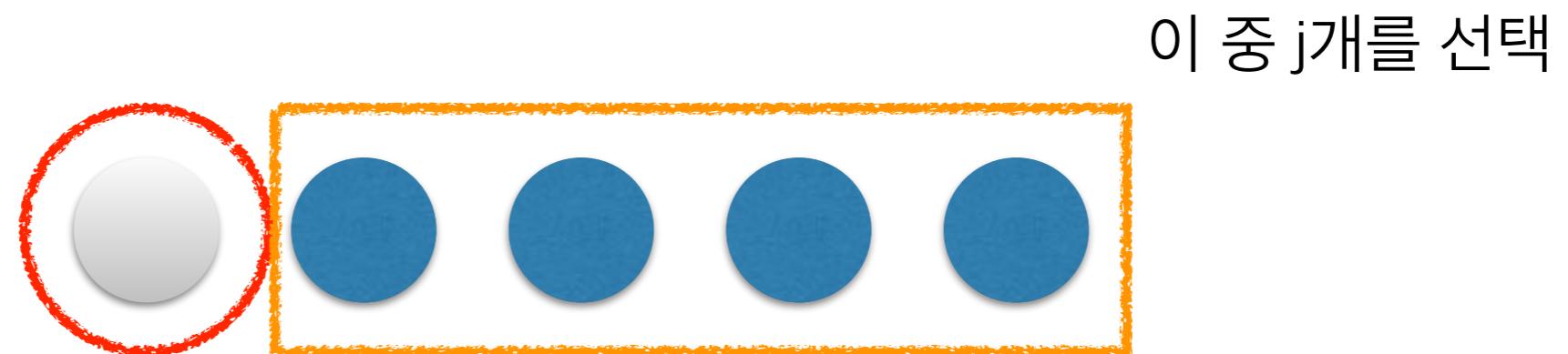
고르는 경우 :  $T(i-1, j-1)$



# [연습문제] R개를 고르는 경우의 수

1. Table을 정의한다

- $T(i, j) = i$ 개 중에서  $j$ 개를 고르는 경우의 수



첫 번째 공을 고르는 경우와 고르지 않는 경우가 있다

고르는 경우 :  $T(i-1, j-1)$

고르지 않는 경우 :  $T(i-1, j)$



# [연습문제] R개를 고르는 경우의 수

1. Table을 정의한다
  - $T(i, j) = i\text{개 중에서 } j\text{개를 고르는 경우의 수}$
2. 점화식을 구한다
  - $T(i, j) = T(i-1, j) + T(i-1, j-1)$
3. 어느 순서로 Table을 구해야 하는지를 생각한다



# [연습문제] R개를 고르는 경우의 수

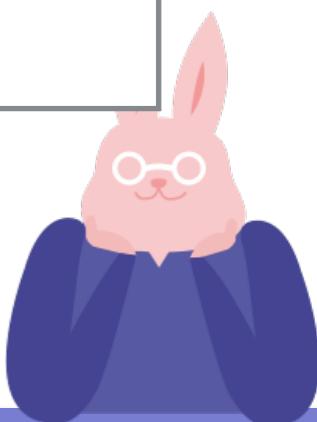
1. Table을 정의한다

- $T(i, j) = i\text{개 중에서 } j\text{개를 고르는 경우}$

2. 점화식을 구한다

- $T(i, j) = T(i-1, j) + T(i-1, j-1)$

3. 어느 순서로 Table을 구해야 하는지를

# [연습문제] R개를 고르는 경우의 수

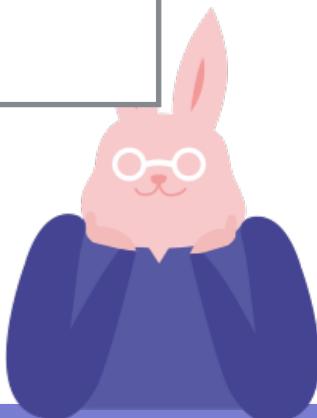
1. Table을 정의한다

- $T(i, j) = i\text{개 중에서 } j\text{개를 고르는 경우}$

2. 점화식을 구한다

- $T(i, j) = T(i-1, j) + T(i-1, j-1)$

3. 어느 순서로 Table을 구해야 하는지를

# [연습문제] R개를 고르는 경우의 수

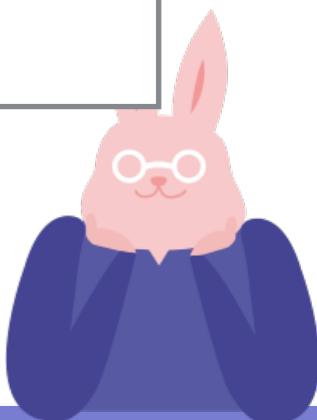
1. Table을 정의한다

- $T(i, j) = i\text{개 중에서 } j\text{개를 고르는 경우}$

2. 점화식을 구한다

- $T(i, j) = T(i-1, j) + T(i-1, j-1)$

3. 어느 순서로 Table을 구해야 하는지를

# [연습문제] R개를 고르는 경우의 수

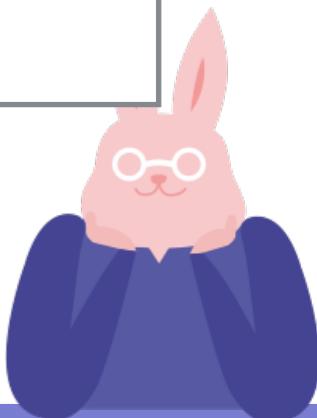
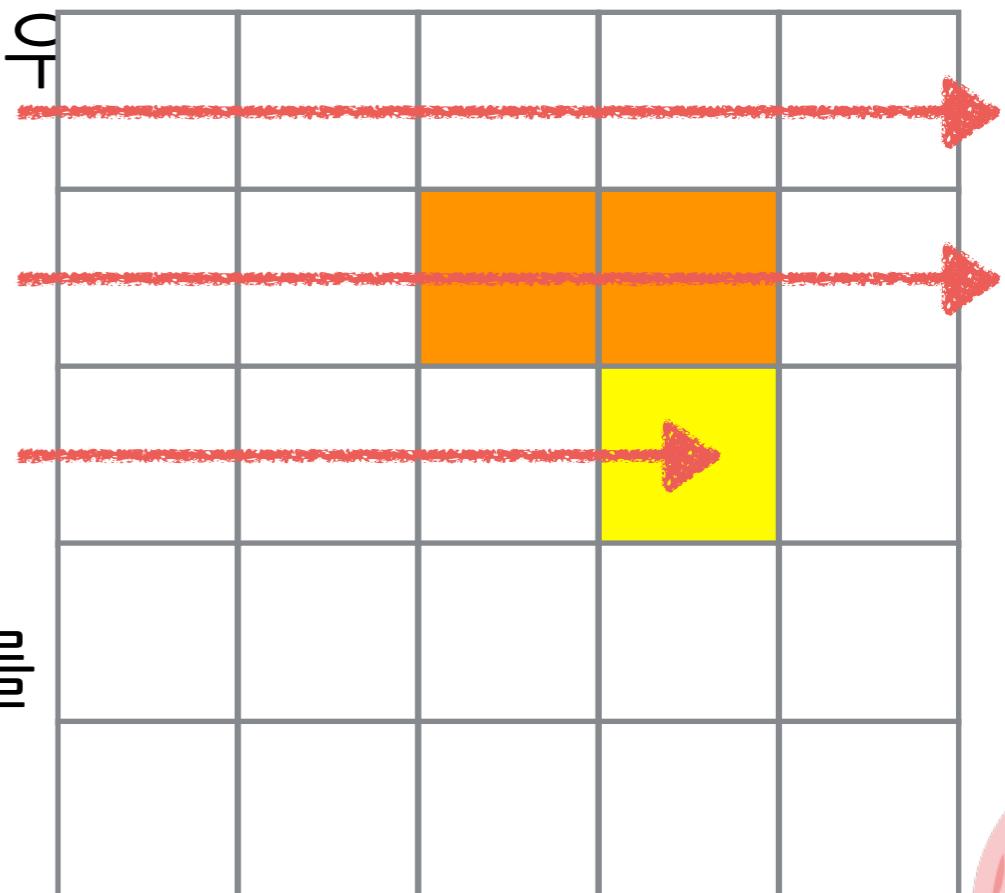
1. Table을 정의한다

- $T(i, j) = i\text{개 중에서 } j\text{개를 고르는 경우}$

2. 점화식을 구한다

- $T(i, j) = T(i-1, j) + T(i-1, j-1)$

3. 어느 순서로 Table을 구해야 하는지를



# [연습문제] R개를 고르는 경우의 수

4. 답이 어디에 있는지를 찾는다

- $T(n, r)$

5. 시간복잡도

- $O(n^2)$



# [연습문제] 짜장, 짬뽕, 볶음밥

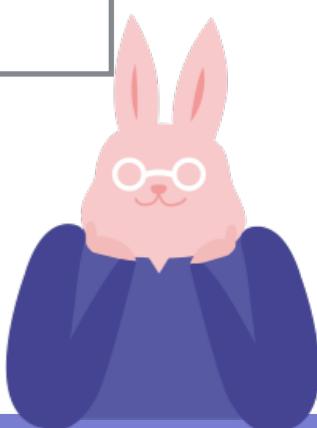
- 매일 짜장, 짬뽕, 볶음밥의 선호도가 다르며, 전날 먹은건 오늘 먹지 않는다. 만족도를 최대화 하라.

입력의 예

```
3  
27 8 35  
18 36 10  
7 22 45
```

출력의 예

```
116
```



# [연습문제] 짜장, 짬뽕, 볶음밥

## 1. Table을 정의한다

- $T(i, 0) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 짜장을 먹을 경우 최대 만족도
- $T(i, 1) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 짬뽕을 먹을 경우 최대 만족도
- $T(i, 2) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 볶음밥을 먹을 경우 최대 만족도



# [연습문제] 짜장, 짬뽕, 볶음밥

## 1. Table을 정의한다

- $T(i, 0) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 짜장을 먹을 경우 최대 만족도
- $T(i, 1) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 짬뽕을 먹을 경우 최대 만족도
- $T(i, 2) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 볶음밥을 먹을 경우 최대 만족도

## 2. 점화식을 구한다



# [연습문제] 짜장, 짬뽕, 볶음밥

## 1. Table을 정의한다

- $T(i, 0) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 짜장을 먹을 경우 최대 만족도
- $T(i, 1) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 짬뽕을 먹을 경우 최대 만족도
- $T(i, 2) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 볶음밥을 먹을 경우 최대 만족도

## 2. 점화식을 구한다

- $T(i, 0) = \max(T(i-1, 1), T(i-1, 2)) + \text{data}(i, 0)$
- $T(i, 1) = \max(T(i-1, 0), T(i-1, 2)) + \text{data}(i, 1)$
- $T(i, 2) = \max(T(i-1, 0), T(i-1, 1)) + \text{data}(i, 2)$



# [연습문제] 짜장, 짬뽕, 볶음밥

3. 어느 순서로 Table을 구해야 하는지를 생각한다

- $i=1 \rightarrow i=n$

4. 답은 어디에 있는지를 찾는다

- $\max(T(i))$

5. 시간복잡도

- $O(n)$



# [활동문제 1] 최장 증가 부분 수열

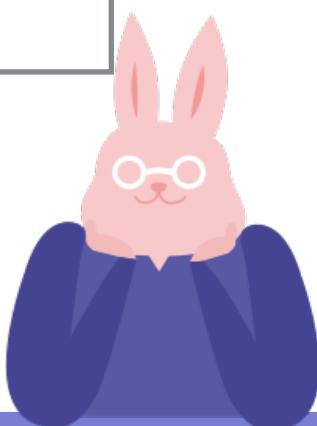
- N개의 숫자 중 최장 증가 부분 수열을 구하여라

입력의 예

```
5  
1 4 2 3 5
```

출력의 예

```
4
```



# [활동문제 1] 최장 증가 부분 수열

1. Table을 정의한다



# [활동문제 1] 최장 증가 부분 수열

1. Table을 정의한다

- $T(i) = i$ 번째 숫자를 끝으로 하는 최장 증가 부분 수열의 길이



# [활동문제 1] 최장 증가 부분 수열

1. Table을 정의한다

- $T(i) = i$ 번째 숫자를 끝으로 하는 최장 증가 부분 수열의 길이

data	5	2	8	6	3	6	9	7
T								



# [활동문제 1] 최장 증가 부분 수열

1. Table을 정의한다

- $T(i) = i$ 번째 숫자를 끝으로 하는 최장 증가 부분 수열의 길이

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4



# [활동문제 1] 최장 증가 부분 수열

1. Table을 정의한다

- $T(i) = i$ 번째 숫자를 끝으로 하는 최장 증가 부분 수열의 길이

2. 점화식을 구한다

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4



# [활동문제 1] 최장 증가 부분 수열

1. Table을 정의한다

- $T(i) = i$ 번째 숫자를 끝으로 하는 최장 증가 부분 수열의 길이

2. 점화식을 구한다

- $T(i) = \max(T(j) + 1) \text{ if } j < i \&& \text{data}[j] < \text{data}[i]$

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4



# [활동문제 1] 최장 증가 부분 수열

3. 어느 순서로 Table을 구해야 하는지를 생각한다

- $i=1 \rightarrow i=n$

4. 답은 어디에 있는지를 찾는다

- $\max(T(i))$

5. 시간복잡도

- $O(n^2)$



# [실습문제 3-1] 최대 공통 부분 수열

- 두 문자열의 최대 공통 부분 수열의 길이를 구하여라

입력의 예

```
aabbaa  
aaaabb
```

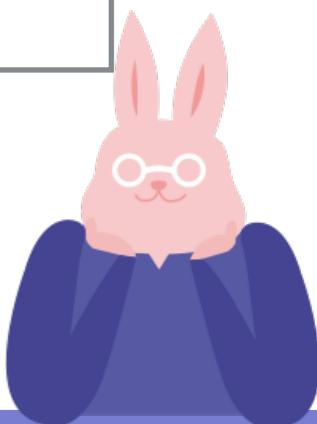
출력의 예

```
4
```

a**abbaa**

**aaaa**bb

aaaa



# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$



# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

	S	N	O	W	Y
S					
U					
N					
N					
Y					



# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

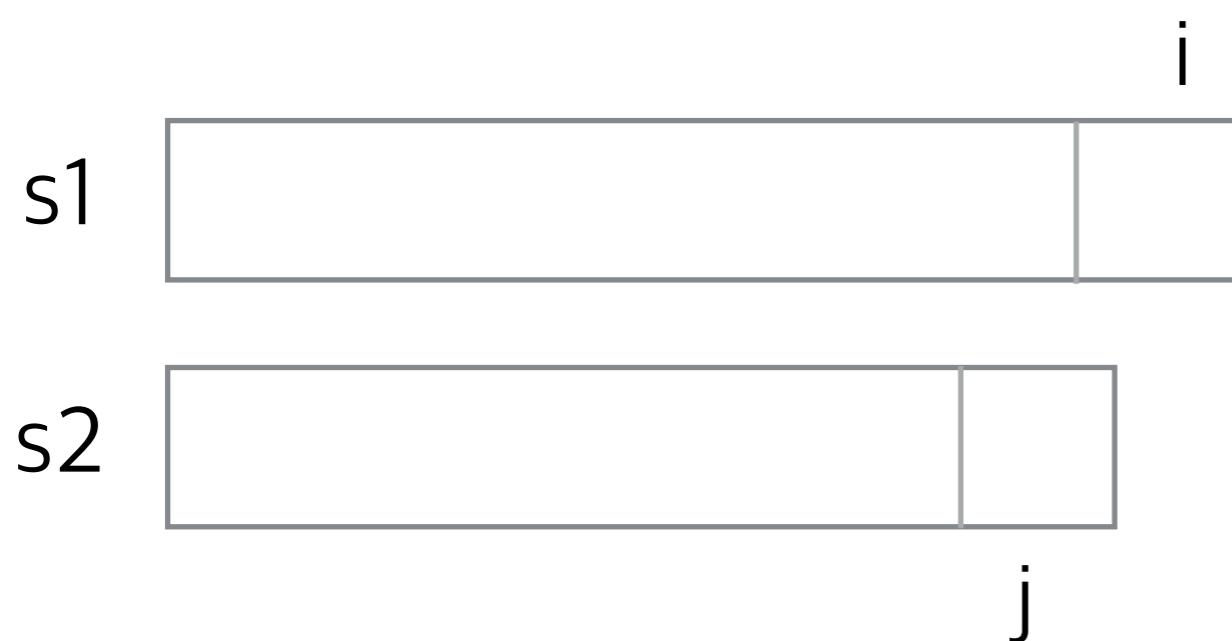
	S	N	O	W	Y
S	1	1	1	1	1
U	1	1	1	1	1
N	1	2	2	2	2
N	1	2	2	2	2
Y	1	2	2	2	3



# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$



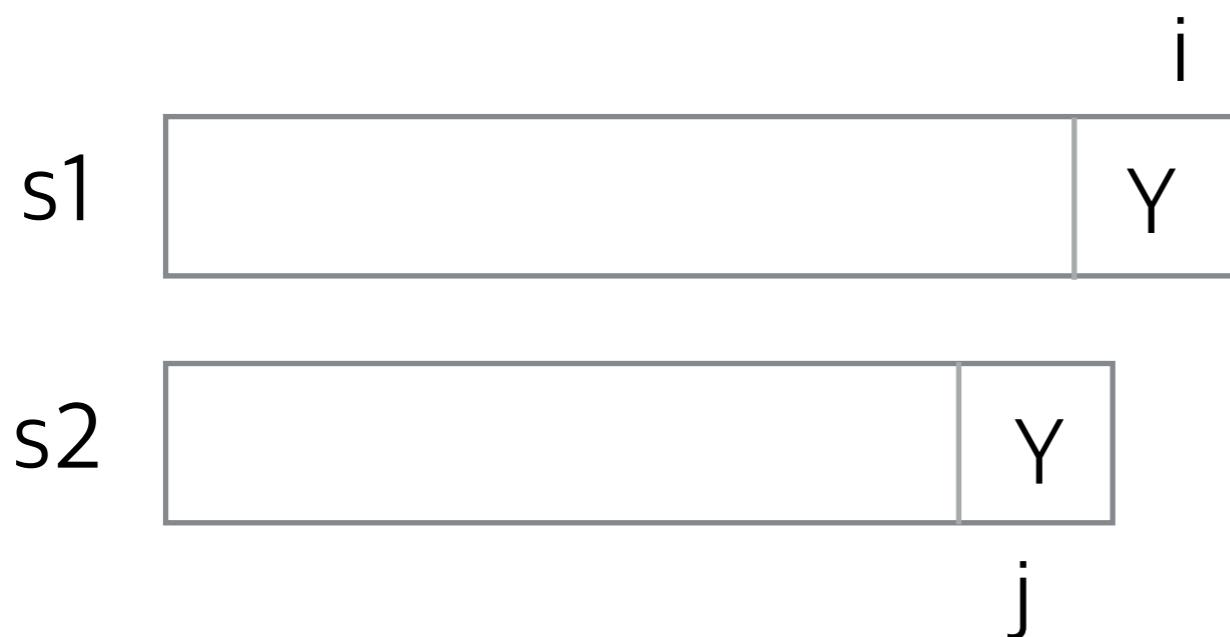
	S	N	O	W	Y
S	1	1	1	1	1
U	1	1	1	1	1
N	1	2	2	2	2
N	1	2	2	2	2
Y	1	2	2	2	3

# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

if  $s1[i] == s2[j]$



	S	N	O	W	Y
S	1	1	1	1	1
U	1	1	1	1	1
N	1	2	2	2	2
N	1	2	2	2	2
Y	1	2	2	2	3

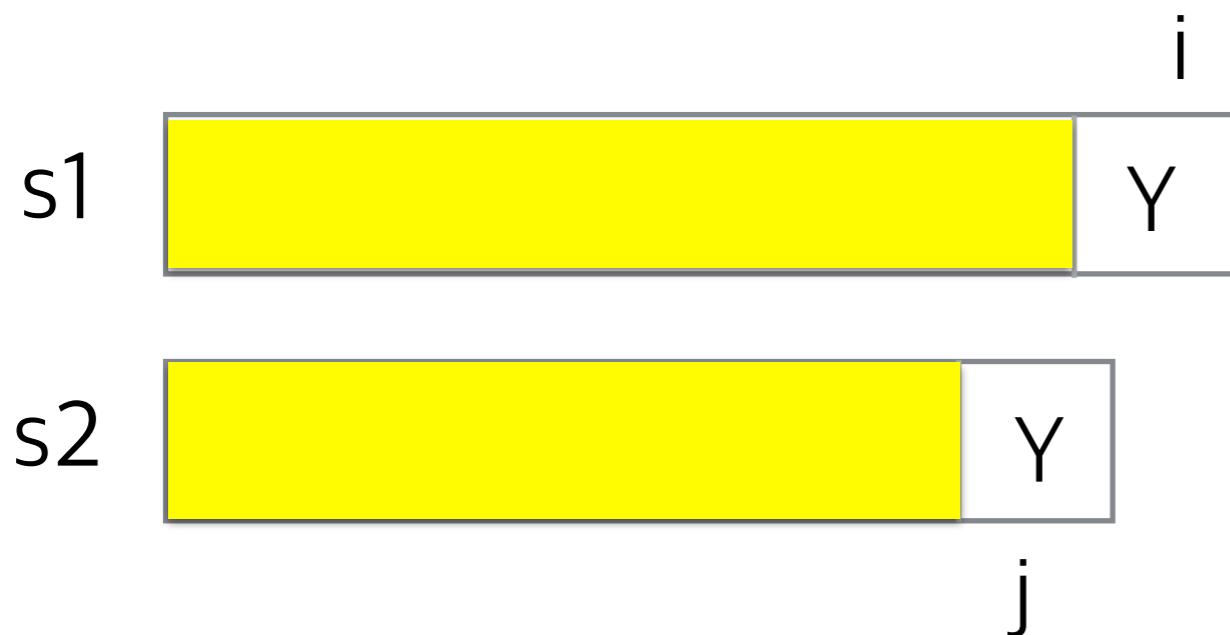
# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

if  $s1[i] == s2[j]$

$$T(i, j) = T(i-1, j-1) + 1$$



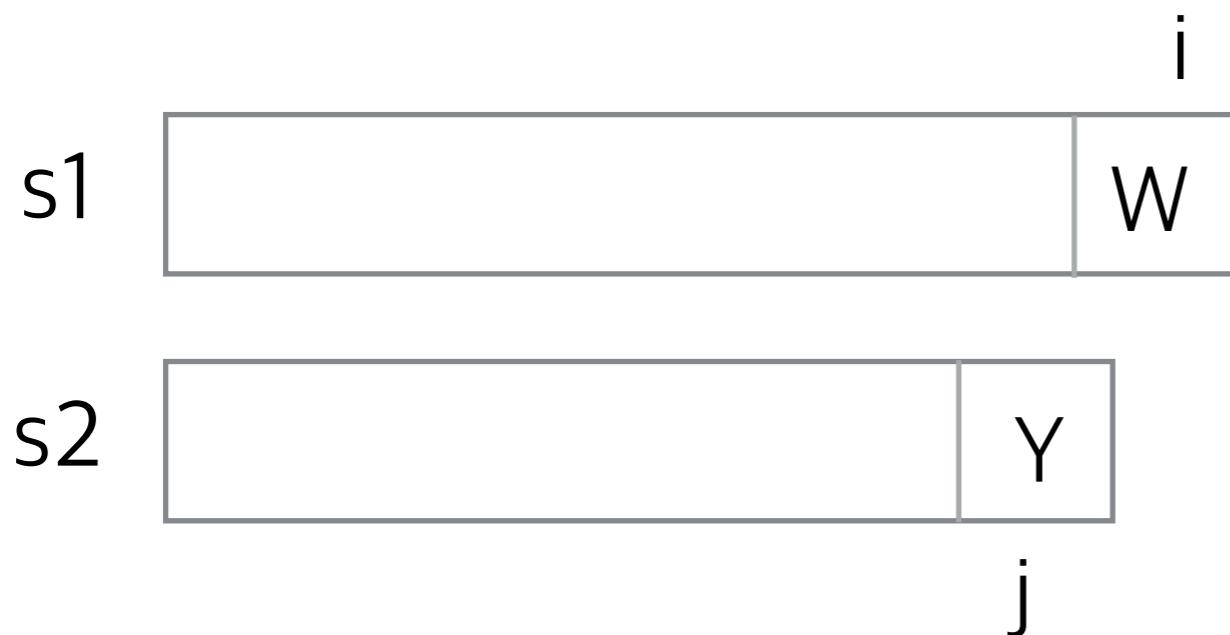
	S	N	O	W	Y
S	1	1	1	1	1
U	1	1	1	1	1
N	1	2	2	2	2
N	1	2	2	2	2
Y	1	2	2	2	3

# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

if  $s1[i] \neq s2[j]$



S	N	O	W	Y
S	1	1	1	1
U	1	1	1	1
N	1	2	2	2
N	1	2	2	2
Y	1	2	2	3

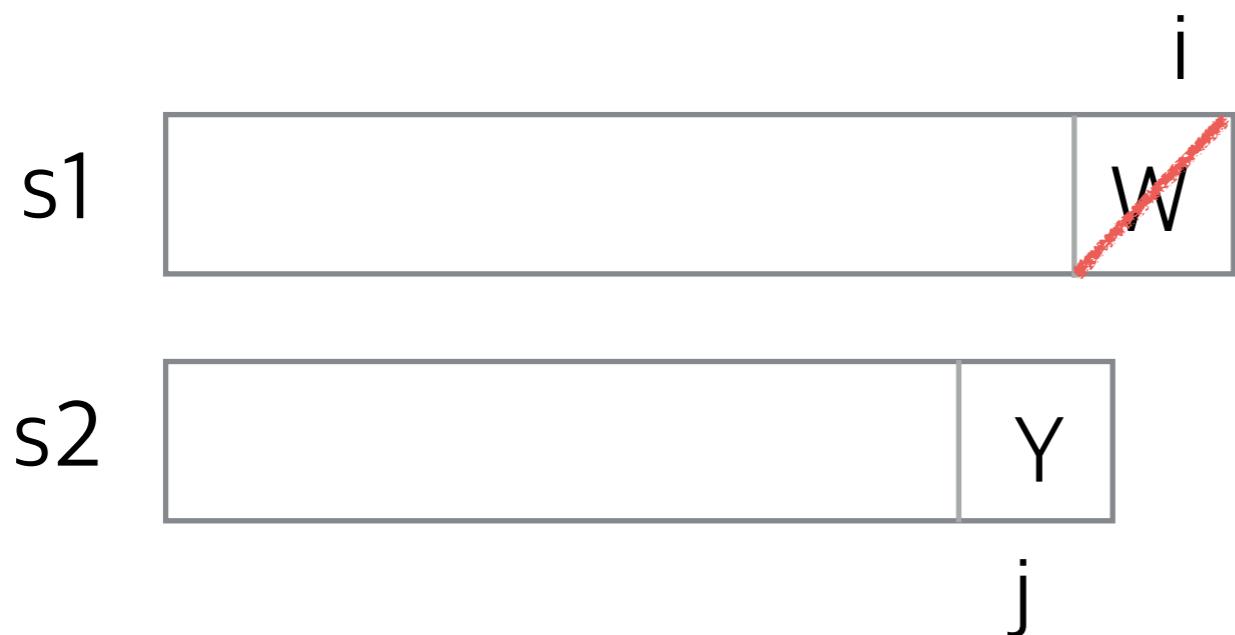
# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

`if s1[i] != s2[j]`

$$T(i, j) = T(i-1, j)$$



S	N	O	W	Y
S	1	1	1	1
U	1	1	1	1
N	1	2	2	2
N	1	2	2	2
Y	1	2	2	3

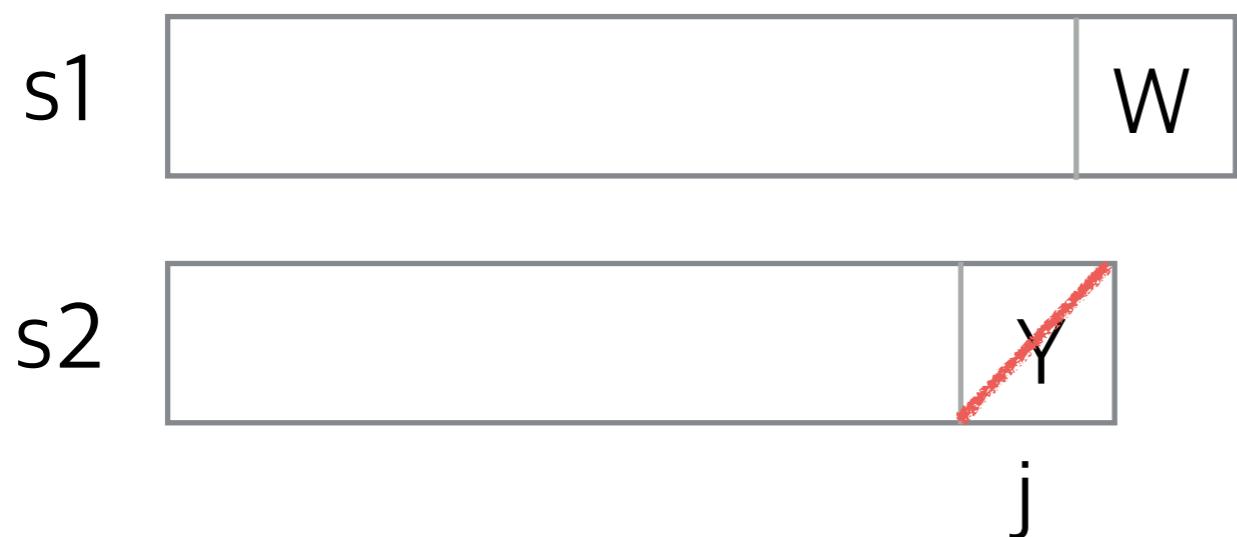
# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

if  $s1[i] \neq s2[j]$

$$T(i, j) = \max(T(i-1, j), T(i, j-1))$$



S	N	O	W	Y
S	1	1	1	1
U	1	1	1	1
N	1	2	2	2
N	1	2	2	2
Y	1	2	2	3

# [실습문제 3-1] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

2. 점화식을 구한다

$$\cdot T(i, j) = \max(T(i-1, j), T(i, j-1)) \quad \text{if } s1[i] \neq s2[j]$$

$$T(i-1, j-1) + 1 \quad \text{otherwise}$$



# [실습문제 3-1] 최대 공통 부분 수열

3. 어느 순서로 Table을 구해야 하는지를 생각한다

- $i=1 \rightarrow i=n$

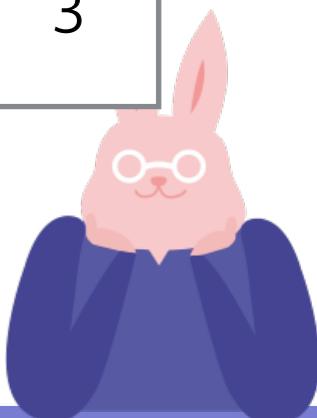
4. 답은 어디에 있는지를 찾는다

- $T(n, m)$
- $n = \text{len(str1)}, m = \text{len(str2)}$

5. 시간복잡도

- $O(nm)$

1	1	1	1	1
1	1	1	1	1
1	2	2	2	2
1	2	2	2	2
1	2	2	2	3



# [예제] 두 문자열의 최단거리

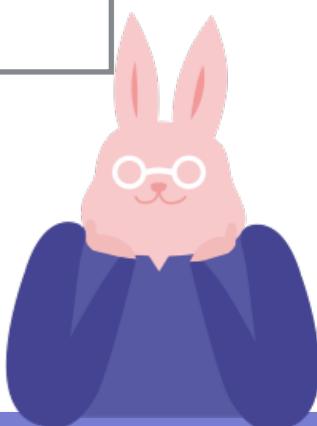
- 문자열 A을 시작으로 문자열 B를 만들기 위한 최소 연산  
연산 : 한 개의 알파벳을 추가 / 제거

입력의 예

```
television  
telephone
```

출력의 예

```
7
```



# [예제] 두 문자열의 최단거리

- $\{ s1 - \text{LCS}(s1, s2) \}$  는 제거하고,  $\{ s2 - \text{LCS}(s1, s2) \}$  를 추가
- $(\text{len}(s1) - \text{LCS}(s1, s2)) + (\text{len}(s2) - \text{LCS}(s1, s2))$
- 증명?



# [예제] 두 문자열의 최단거리

- $\{ s1 - \text{LCS}(s1, s2) \}$  는 제거하고,  $\{ s2 - \text{LCS}(s1, s2) \}$  를 추가
- $(\text{len}(s1) - \text{LCS}(s1, s2)) + (\text{len}(s2) - \text{LCS}(s1, s2))$
- 증명?
  - 연산 순서를 강제해도 괜찮다 : 제거를 먼저 한 후, 추가를 하자
  - $s1 \rightarrow s' \rightarrow s2$  where  $s' \subseteq s1, s' \subseteq s2$
  - $s'$  는 길수록 좋다
- $O(nm)$  where  $n = \text{len}(s1), m = \text{len}(s2)$



# [예제] 옳은 합인지 판단하기

- 두 string을 merging 하여 나올 수 있는 결과인지 판단하라.  
단, merging이란 string을 앞에서부터 빼내어 합치는 것을 말한다.

입력의 예

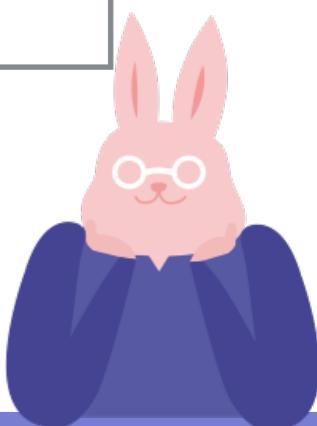
```
abb  
babc  
abbabbc
```

출력의 예

Yes

```
abb  
babc  
babbacb
```

No



# [예제] 옳은 합인지 판단하기

1.  $T(i, j) = \text{str1}(1..i)$ 와  $\text{str2}(1..j)$ 를 merge하여  
 $\text{str3}(1..i+j)$ 를 만들 수 있으면 1, 아니면 0



# [예제] 옳은 합인지 판단하기

1.  $T(i, j) = \text{str1}(1..i)$ 와  $\text{str2}(1..j)$ 를 merge하여  $\text{str3}(1..i+j)$ 를 만들 수 있으면 1, 아니면 0

str1	a	b	b				
str2	b	a	b	c			
str3	a	b	b	a	b	b	c



# [예제] 옳은 합인지 판단하기

1.  $T(i, j) = str1(1..i)$ 와  $str2(1..j)$ 를 merge하여  $str3(1..i+j)$ 를 만들 수 있으면 1, 아니면 0

								$T(1, 2) =$
str1	a	b	b					
str2	b	a	b	c				
str3	a	b	b	a	b	b	c	



# [예제] 옳은 합인지 판단하기

1.  $T(i, j) = str1(1..i)$ 와  $str2(1..j)$ 를 merge하여  $str3(1..i+j)$ 를 만들 수 있으면 1, 아니면 0

str1	a	b	b					$T(1, 2) = 0$
str2	b	a	b	c				
str3	a	b	b	a	b	b	c	



# [예제] 옳은 합인지 판단하기

## 2. 점화식 구하기

str1	a	b	b				
str2	b	a	b	c			
str3	a	b	b	a	b	b	c



# [예제] 옳은 합인지 판단하기

## 2. 점화식 구하기

- $T(i, j) =$

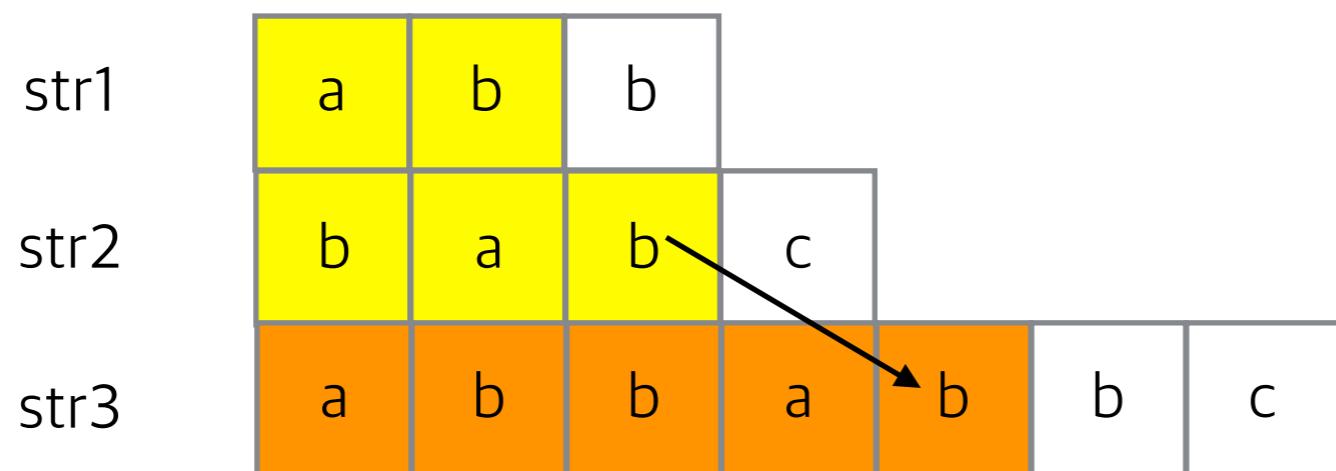
str1	a	b	b				
str2	b	a	b	c			
str3	a	b	b	a	b	b	c



# [예제] 옳은 합인지 판단하기

## 2. 점화식 구하기

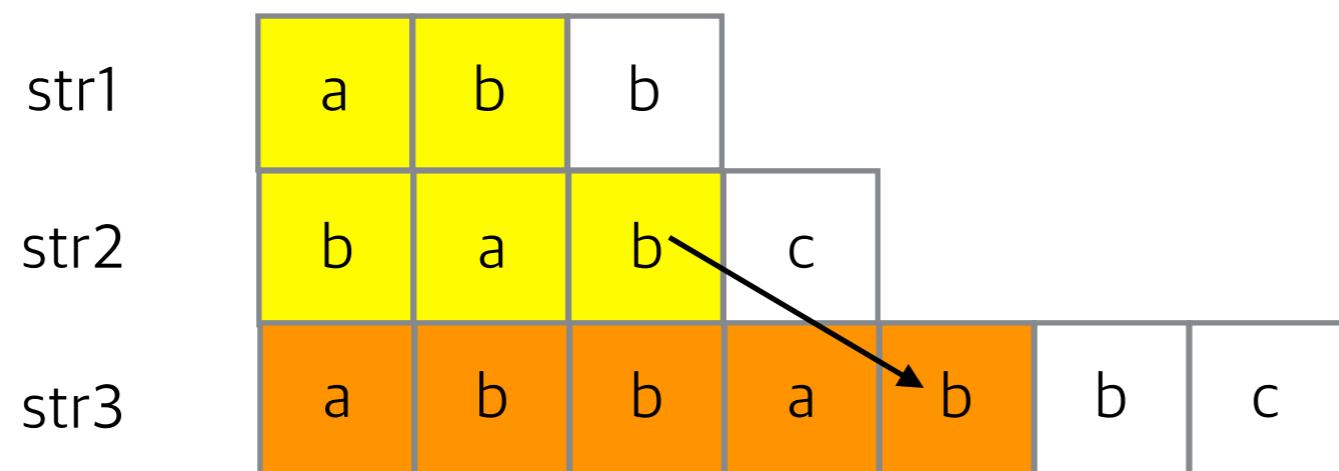
- $T(i, j) =$



# [예제] 옳은 합인지 판단하기

## 2. 점화식 구하기

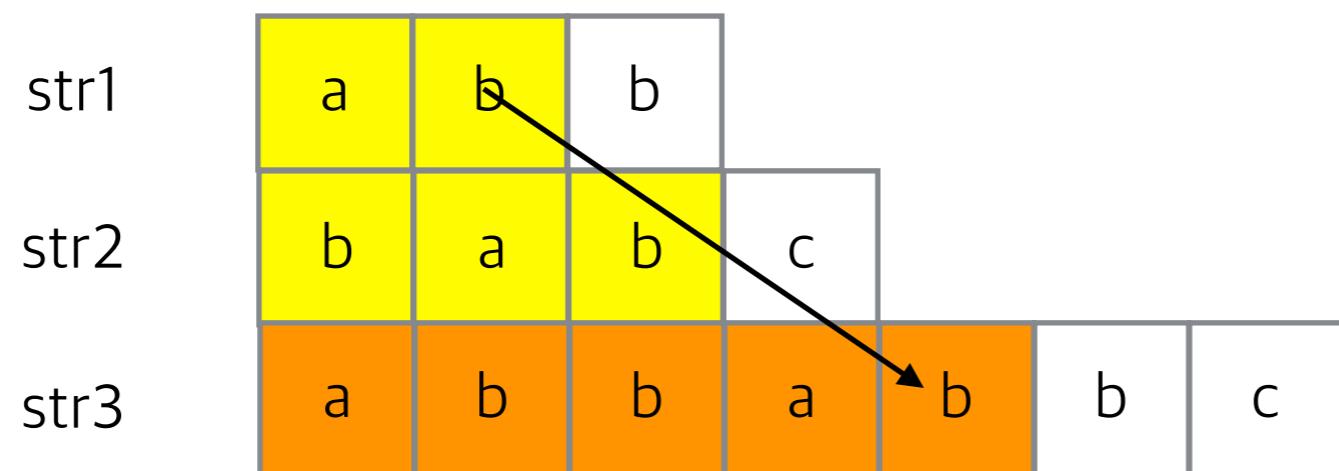
- $T(i, j) = T(i, j-1)$  if  $\text{str2}(j) == \text{str3}(i+j+1)$



# [예제] 옳은 합인지 판단하기

## 2. 점화식 구하기

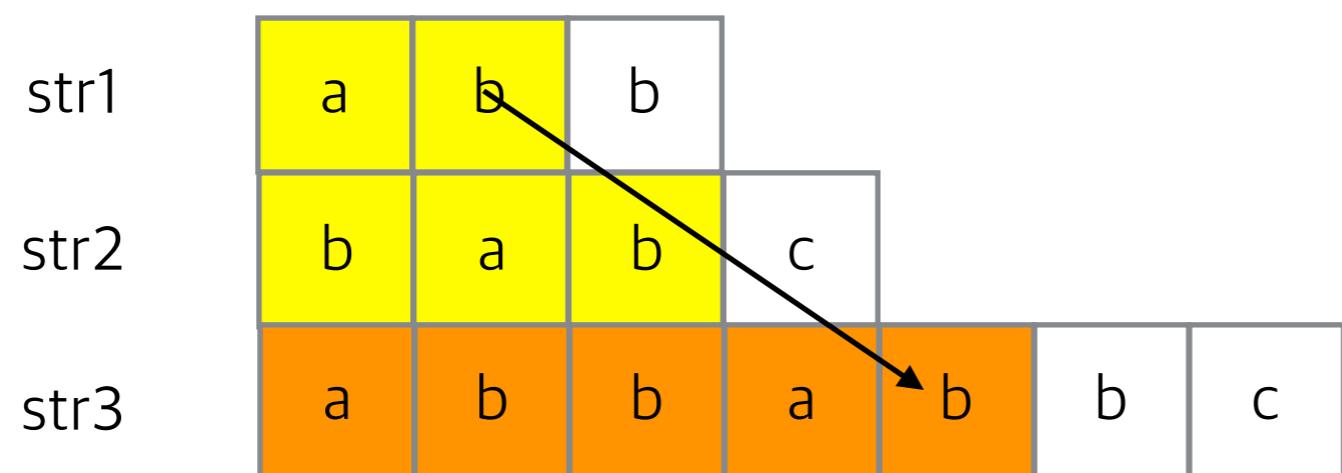
- $T(i, j) = T(i, j-1)$  if  $\text{str2}(j) == \text{str3}(i+j+1)$



# [예제] 옳은 합인지 판단하기

## 2. 점화식 구하기

- $T(i, j) = \begin{cases} T(i, j-1) & \text{if } str2(j) == str3(i+j+1) \\ \text{or } T(i-1, j) & \text{if } str1(i) == str3(i+j+1) \end{cases}$



# [예제] 옳은 합인지 판단하기

## 1. Table 정의

- $T(i, j) = str1(1..i)$ 와  $str2(1..j)$ 를 merge하여  $str3(1..i+j)$ 를 만들 수 있으면 1, 아니면 0

## 2. 점화식 구하기

- $T(i, j) = \begin{cases} T(i+1, j) & \text{if } str1(i) == data(i+j+1) \\ or T(i, j-1) & \text{if } str2(j) == data(i+j+1) \end{cases}$

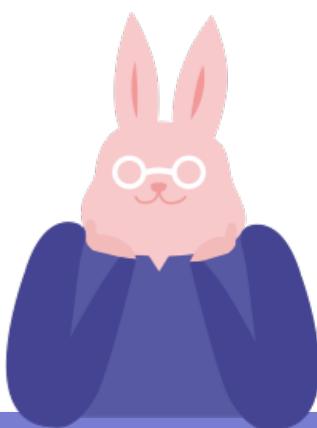


# [예제] 옳은 합인지 판단하기

3. 정답을 구하는 순서

4. 정답의 위치

- T(len1-1, len2-1)



# 경로 출력

- Dynamic Programming에서 실제 경로는 어떻게 얻나 ?
- Table 정의를 잘 생각하면 크게 어렵지 않다!



# [활동문제 0] 최대구간의 합 구하기

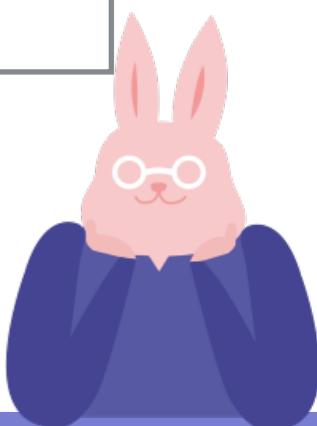
- n개의 숫자 중에서 연속 부분 최대합과 구간을 출력

입력의 예

```
1 2 3 4 -100 1
```

출력의 예

```
10  
0 3
```



# [활동문제 0] 최대구간의 합 구하기

1. Table을 정의한다

- $T(i) = \underline{i\text{번째 수를 끝으로 하는}}$  최대 구간의 합

2. 점화식을 구한다

- $T(i) = \max(\text{data}[i], T(i-1) + \text{data}[i])$

3. 어느 순서로 Table을 구해야 하는지를 생각한다

- $i=0 \rightarrow i=n$



# [활동문제 0] 최대구간의 합 구하기

- 경로 출력을 위해 다른 list L을 정의한다
  - $L(i) = i$ 번째 수를 끝으로 하는 최대 구간의 시작점

data	2	1	-2	5	-10	3	2	5	-3	7	9	-10
T	2	3	1	6	-4	3	5	10	7	14	23	13
L												



# [활동문제 0] 최대구간의 합 구하기

- 경로 출력을 위해 다른 list L을 정의한다
  - $L(i) = i$ 번째 수를 끝으로 하는 최대 구간의 시작점

data	2	1	-2	5	-10	3	2	5	-3	7	9	-10
T	2	3	1	6	-4	3	5	10	7	14	23	13
L	0	0	0	0	0	5	5	5	5	5	5	5



# [활동문제 0] 최대구간의 합 구하기

- 경로 출력을 위해 다른 list L을 정의한다
  - $L(i) = i$ 번째 수를 끝으로 하는 최대 구간의 시작점
- $L(i) = L(i-1)$       if  $T(i-1) + \text{data}(i) > \text{data}(i)$   
 $i$                         otherwise



# [연습문제] 포도주 마시기

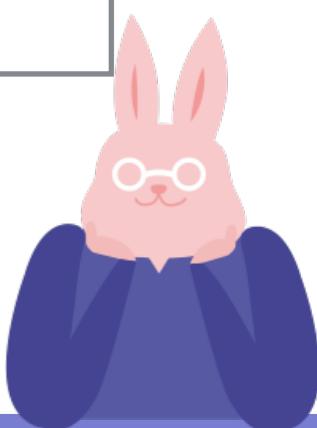
- n잔의 포도주가 있을 때, 마시는 양의 최댓값과, 마시는 포도주들을 출력 단, 연속하여 3잔을 모두 마실 수는 없다

입력의 예

```
6  
6 10 13 9 8 1
```

출력의 예

```
33  
6 10 9 8
```



# [연습문제] 포도주 마시기

1. Table을 정의한다

- $T(i) = 1 \sim i$  까지의 포도주가 있을 때, 마시는 최댓값

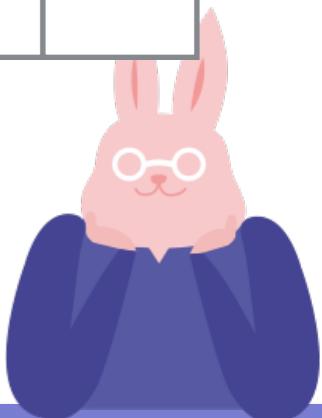
2. 점화식을 구한다

- $T(i) = \max(T(i-1), T(i-2) + \text{data}(i), T(i-3) + \text{data}(i-1) + \text{data}(i-2))$

data

6	10	13	9	8	1

T



# [연습문제] 포도주 마시기

- $L$ 을 정의한다
  - $L(i) = T$ 가 어느 값을 선택했는지를 기록
  - $L(i) = \begin{cases} 1 & \text{if } T(i) = T(i-1) \\ 2 & \text{if } T(i) = T(i-2) + \text{data}(i) \\ 3 & \text{if } T(i) = T(i-3) + \text{data}(i-1) + \text{data}(i-2) \end{cases}$

	data	6	10	13	9	8	1
T	6	10	23	28	33	33	
L							



# [연습문제] 포도주 마시기

- $L$ 을 정의한다
  - $L(i) = T$ 가 어느 값을 선택했는지를 기록
  - $L(i) = \begin{cases} 1 & \text{if } T(i) = T(i-1) \\ 2 & \text{if } T(i) = T(i-2) + \text{data}(i) \\ 3 & \text{if } T(i) = T(i-3) + \text{data}(i-1) + \text{data}(i-2) \end{cases}$

	data	6	10	13	9	8	1
	T	6	10	23	28	33	33
	L	0	0	3	3	3	1



# [연습문제] 포도주 마시기

- $L(n)$ 부터 시작해서 앞으로 따라감
  - $T(n)$ 에 답이 있기 때문

data  
T  
L

6	10	13	9	8	1
6	10	23	28	33	33
0	0	3	3	3	1



# [연습문제] 포도주 마시기

- $L(n)$ 부터 시작해서 앞으로 따라감
  - $T(n)$ 에 답이 있기 때문

data  
T  
L

6	10	13	9	8	1
6	10	23	28	33	33
0	0	3	3	3	1



# [연습문제] 포도주 마시기

- $L(n)$ 부터 시작해서 앞으로 따라감
  - $T(n)$ 에 답이 있기 때문

data  
T  
L

6	10	13	9	8	1
6	10	23	28	33	33
0	0	3	3	3	1



# [연습문제] 포도주 마시기

- $L(n)$ 부터 시작해서 앞으로 따라감
  - $T(n)$ 에 답이 있기 때문

data  
T  
L

6	10	13	9	8	1
6	10	23	28	33	33
0	0	3	3	3	1



# [연습문제] 짜장, 짬뽕, 볶음밥

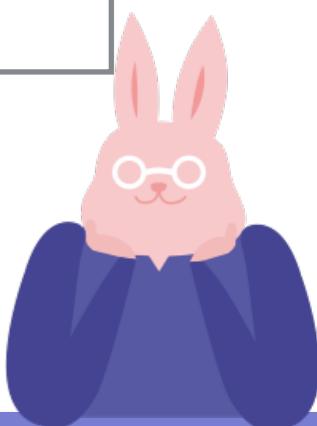
- 매일 짜장, 짬뽕, 볶음밥의 선호도가 다르며, 전날 먹은건 오늘 먹지 않는다. 만족도의 최댓값과, 무엇을 먹어야 하는지를 출력하라.

입력의 예

```
3  
27 8 35  
18 36 10  
7 22 45
```

출력의 예

```
116  
2 1 2
```



# [연습문제] 짜장, 짬뽕, 볶음밥

## 1. Table을 정의한다

- $T(i, 0) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 짜장을 먹을 경우 최대 만족도
- $T(i, 1) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 짬뽕을 먹을 경우 최대 만족도
- $T(i, 2) = i$ 번째 날까지 밥을 먹으며,  $i$ 번째 날에 볶음밥을 먹을 경우 최대 만족도

## 2. 점화식을 구한다

- $T(i, 0) = \max(T(i-1, 1), T(i-1, 2)) + \text{data}(i, 0)$
- $T(i, 1) = \max(T(i-1, 0), T(i-1, 2)) + \text{data}(i, 1)$
- $T(i, 2) = \max(T(i-1, 0), T(i-1, 1)) + \text{data}(i, 2)$



# [연습문제] 짜장, 짬뽕, 볶음밥

- $L$ 을 정의한다
  - $L(i) = i$ 번째 날에 무엇을 먹었는지 기록
  - $L(i) = 1, 2, 3$  중 하나가 들어감
- $L(n)$ 부터 뒤에서 앞으로 진행



# [예제] 최장 증가 부분 수열

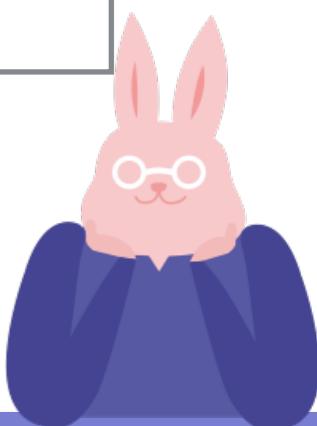
- N개의 숫자 중 최장 증가 부분 수열의 길이와 수열을 출력하라

입력의 예

```
5  
1 4 2 3 5
```

출력의 예

```
4  
1 2 3 5
```



# [예제] 최장 증가 부분 수열

1. Table을 정의한다

- $T(i) = i$ 번째 숫자를 끝으로 하는 최장 증가 부분 수열의 길이

2. 점화식을 구한다

- $T(i) = \max(T(j) + 1) \text{ if } j < i \&& \text{data}[j] < \text{data}[i]$

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4



# [예제] 최장 증가 부분 수열

- $L(i) = i$ 번째 숫자를 끝으로 할 경우, 바로 앞에 오는 숫자의 index
  - $L(i) = j$                   where  $T(i) = T(j) + 1$

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4
L								



# [예제] 최장 증가 부분 수열

- $L(i) = i$ 번째 숫자를 끝으로 할 경우, 바로 앞에 오는 숫자의 index
  - $L(i) = j$                   where  $T(i) = T(j) + 1$

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4
L	-1	-1	0	0	1	4	5	5



# [예제] 최장 증가 부분 수열

- $L(m)$ 에서 시작해서 거꾸로 돌아감. (단,  $T(m)$ 이 최댓값)

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4
L	-1	-1	0	0	1	4	5	5



# [예제] 최장 증가 부분 수열

- $L(m)$ 에서 시작해서 거꾸로 돌아감. (단,  $T(m)$ 이 최댓값)

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4
L	-1	-1	0	0	1	4	5	5



# [예제] 최장 증가 부분 수열

- $L(m)$ 에서 시작해서 거꾸로 돌아감. (단,  $T(m)$ 이 최댓값)

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4
L	-1	-1	0	0	1	4	5	5



# [예제] 최장 증가 부분 수열

- $L(m)$ 에서 시작해서 거꾸로 돌아감. (단,  $T(m)$ 이 최댓값)

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4
L	-1	-1	0	0	1	4	5	5



# [예제] 최장 증가 부분 수열

- $L(m)$ 에서 시작해서 거꾸로 돌아감. (단,  $T(m)$ 이 최댓값)

data	5	2	8	6	3	6	9	7
T	1	1	2	2	2	3	4	4
L	-1	-1	0	0	1	4	5	5



# [실습문제 1-2] 최대 공통 부분 수열

- 두 문자열의 최대 공통 부분 수열의 길이와, 그 수열을 구하여라

입력의 예

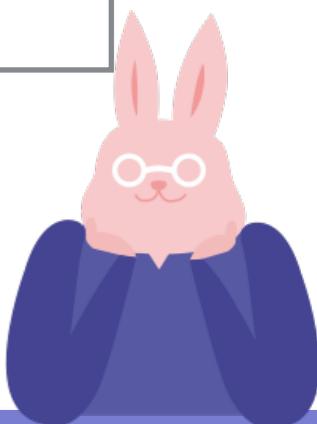
```
aabbaa  
aaaabb
```

출력의 예

```
4  
aaaa
```

a**abbaa**  
**aaaaab**b

aaaa



# [실습문제 1-2] 최대 공통 부분 수열

1. Table을 정의한다

- $T(i, j) = \text{str1의 } 1 \sim i, \text{str2의 } 1 \sim j \text{의 최대공통부분수열 길이}$

2. 점화식을 구한다

$$\cdot T(i, j) = \max(T(i-1, j), T(i, j-1)) \quad \text{if } s1[i] \neq s2[j]$$

$$T(i-1, j-1) + 1 \quad \text{otherwise}$$



# [실습문제 1-2] 최대 공통 부분 수열

- $L$ 을 정의한다
  - $L(i, j) = T(i, j)$ 가 어디에서 왔는지 기록
  - $L(i, j) = \begin{cases} 1 & \text{if } T(i, j) = T(i-1, j-1) + 1 \\ 2 & \text{if } T(i, j) = T(i-1, j) \\ 3 & \text{if } T(i, j) = T(i, j-1) \end{cases}$



# [실습문제 1-2] 최대 공통 부분 수열

- $L$ 을 정의한다
  - $L(i, j) = T(i, j)$ 가 어디에서 왔는지 기록
  - $$\begin{array}{ll} L(i, j) = 1 & \text{if } T(i, j) = T(i-1, j-1) + 1 \\ 2 & \text{if } T(i, j) = T(i-1, j) \\ 3 & \text{if } T(i, j) = T(i, j-1) \end{array}$$
- Recursion으로 구현
  - $\text{printResult}(i, j) = \text{str1}(1..i), \text{str2}(1..j)$ 의 LCS를 return



# [예제] 옳은 합인지 판단하기

- 두 string을 merging 하여 나올 수 있는 결과인지 판단하라.  
단, merging이란 string을 앞에서부터 빼내어 합치는 것을 말한다.

입력의 예

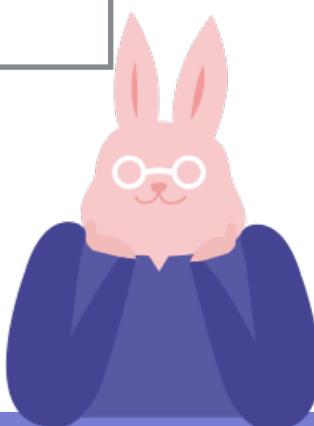
```
abb  
babc  
abbabbc
```

출력의 예

Yes

```
abb  
babc  
babbacb
```

No



# [실습문제 3-1] Palindrome

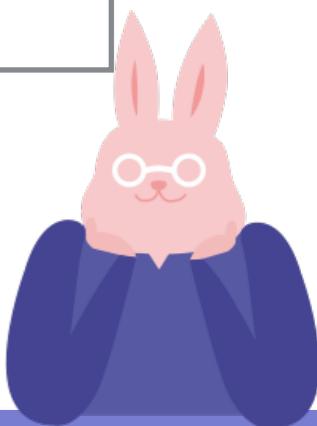
- Palindrome을 만들기 위한 최소 추가 문자 개수를 출력하라  
palindrome이란, 그냥 읽었을 때와 거꾸로 읽었을 때가 같은 문자열이다.

입력의 예

abccdbac

출력의 예

2



# [실습문제 3-1] Palindrome

## 1. Table 정의

- $T(i, j) = i \sim j$  까지 문자열을 palindrome으로 만들기 위하여 추가해야 하는 문자 개수의 최솟값



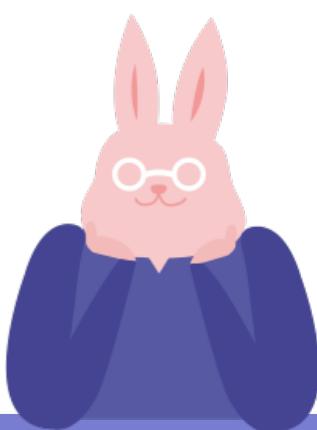
# [실습문제 3-1] Palindrome

1. Table 정의

- $T(i, j) =$

	a	b	c	c	d	b	a	c
a								
b								
c								
c								
d								
b								
a								
c								

를 만들기 위하여



# [실습문제 3-1] Palindrome

1. Table 정의

- $T(i, j) =$

	a	b	c	c	d	b	a	c
a								
b								
c								
c								
d								
b								
a								
c								

로 만들기 위하여



# [실습문제 3-1] Palindrome

1. Table 정의

- $T(i, j) =$

	a	b	c	c	d	b	a	c
a								
b							1	
c								
c								
d								
b								
a								
c								

를 만들기 위하여



# [실습문제 3-1] Palindrome

1. Table 정의

$$\cdot T(i, j) =$$

	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1	2
b		0	1	1	2	1	2	3
c			0	0	1	2	3	3
c				0	1	2	3	2
d					0	1	2	3
b						0	1	2
a							0	1
c								0

• 만들기 위하여



# [실습문제 3-1] Palindrome

1. Table 정의

$$\cdot T(i, j) =$$

	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1	2
b		0	1	1	2	1	2	3
c			0	0	1	2	3	3
c				0	1	2	3	2
d					0	1	2	3
b						0	1	2
a							0	1
c								0

• 만들기 위하여



# [실습문제 3-1] Palindrome

1. Table 정의

- $T(i, j) =$

	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1	2
b		0	1	1	2	1	2	3
c			0	1	2	1	0	2
d				0	1	2	3	

을 만들기 위하여

```
if data(i) == data(j) : T(i, j) = T(i+1, j-1)
```

d					0	1	2	3
b						0	1	2
a							0	1
c								0



# [실습문제 3-1] Palindrome

## 1. Table 정의

- $T(i, j) = i \sim j$  까지 문자열을 palindrome으로 만들기 위하여 추가해야 하는 문자 개수의 최솟값

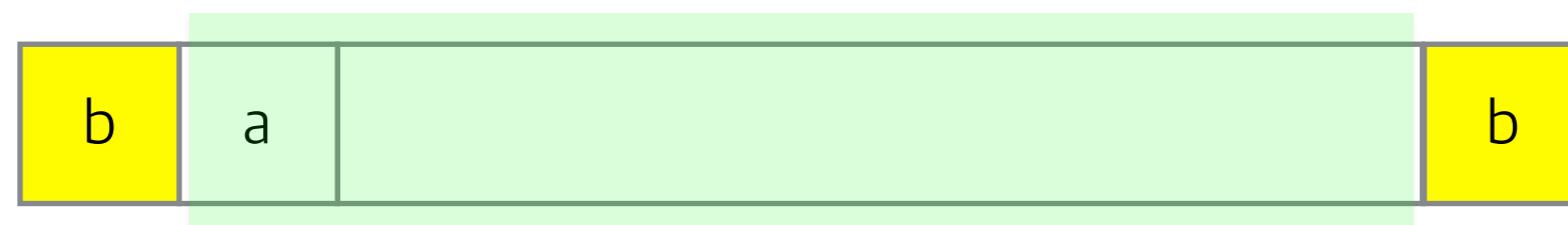
a		b
---	--	---



# [실습문제 3-1] Palindrome

## 1. Table 정의

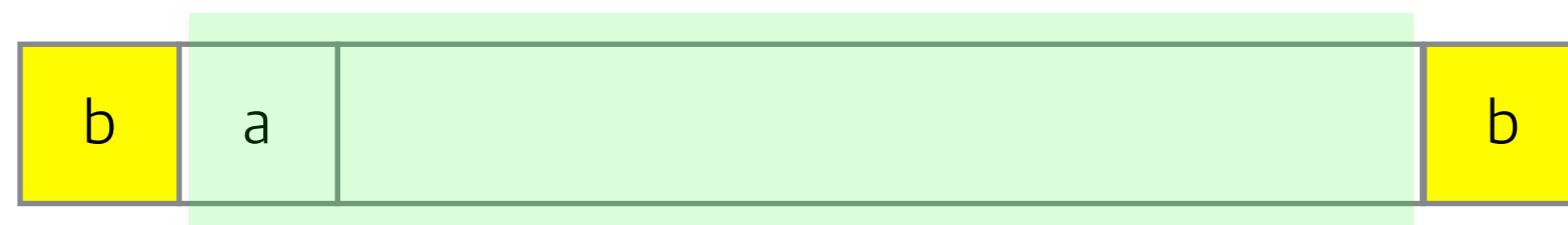
- $T(i, j) = i \sim j$  까지 문자열을 palindrome으로 만들기 위하여 추가해야 하는 문자 개수의 최솟값



# [실습문제 3-1] Palindrome

## 1. Table 정의

- $T(i, j) = i \sim j$  까지 문자열을 palindrome으로 만들기 위하여 추가해야 하는 문자 개수의 최솟값



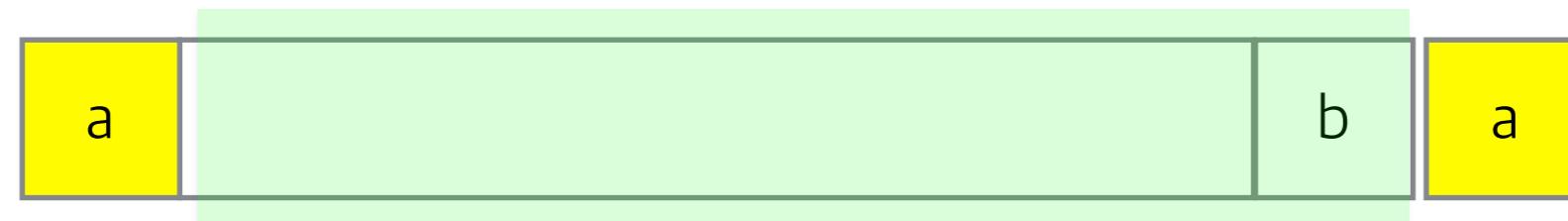
$$T(i, j-1) + 1$$



# [실습문제 3-1] Palindrome

## 1. Table 정의

- $T(i, j) = i \sim j$  까지 문자열을 palindrome으로 만들기 위하여 추가해야 하는 문자 개수의 최솟값



$$T(i, j-1) + 1$$

$$T(i+1, j) + 1$$



# [실습문제 3-1] Palindrome

1. Table 정의

	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1	2
b		0	1	1	2	1	2	3
c			0	1	0	0	0	0
d				1	0	0	0	0

- $T(i, j) = \dots$  만들기 위하여

```
if data(i) == data(j) : T(i, j) = T(i+1, j-1)
else : T(i, j) = min(T(i+1, j), T(i, j-1)) + 1
```

b						0	1	2
a						0	1	
c						0		



# [실습문제 3-1] Palindrome

## 1. Table 정의

- $T(i, j) = i \sim j$  까지 문자열을 palindrome으로 만들기 위하여 추가해야 하는 문자 개수의 최솟값

## 2. 점화식

- $$\begin{aligned} T(i, j) &= T(i+1, j-1) && \text{if } \text{data}(i) == \text{data}(j) \\ &= \min(T(i+1, j), \min(i, j-1) + 1) && \text{otherwise} \end{aligned}$$



# [실습문제 3-1] P

3. Table을 구하는 순서

	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1	2
b		0	1	1	2	1	2	3
c			0	0	1	2	3	3
c				0	1	2	3	2
d					0	1	2	3
b						0	1	2
a							0	1
c								0



# [실습문제 3-1] P

3. Table을 구하는 순서

	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1	2
b		0	1	1	2	1	2	3
c			0	0	1	2	3	3
c				0	1	2	3	2
d					0	1	2	3
b						0	1	2
a							0	1
c								0



# [실습문제 3-1] P

3. Table을 구하는 순서

	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1	2
b	0	1	1	2	1	2		
c	0	0	1	2	3			
c	0	1	2	3				
d	0	1	2					
b	0	1						
a							0	
c								0



# [실습문제 3-1] Palindrome

3. Table을 구하는 순서

4. 정답의 위치

- $T(0, \text{len}-1)$



# [실습문제 3-1] Palindrome

3. Table을 구하는 순서

4. 정답의 위치

- $T(0, \text{len}-1)$

5. 시간복잡도

- $O(n^2)$



# [실습문제 3-1]

- 어느 문자를 추가해야 하나?
  - $L(i, j) = \text{따라온 위치}$

P	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1	2
b		0	1	1	2	1	2	3
c			0	0	1	2	3	3
c				0	1	2	3	2
d					0	1	2	3
b						0	1	2
a							0	1
c								0



# [실습문제 3-1]

- 어느 문자를 추가해야 하나?
  - $L(i, j) =$  따라온 위치

P	a	b	c	c	d	b	a	c
a	0	1	2	2	3	2	1 ← 2	
b		0	1	1	2	1	2	3
c			0	0	1	2	3	3
c				0	1	2	3	2
d					0	1	2	3
b						0	1	2
a							0	1
c								0



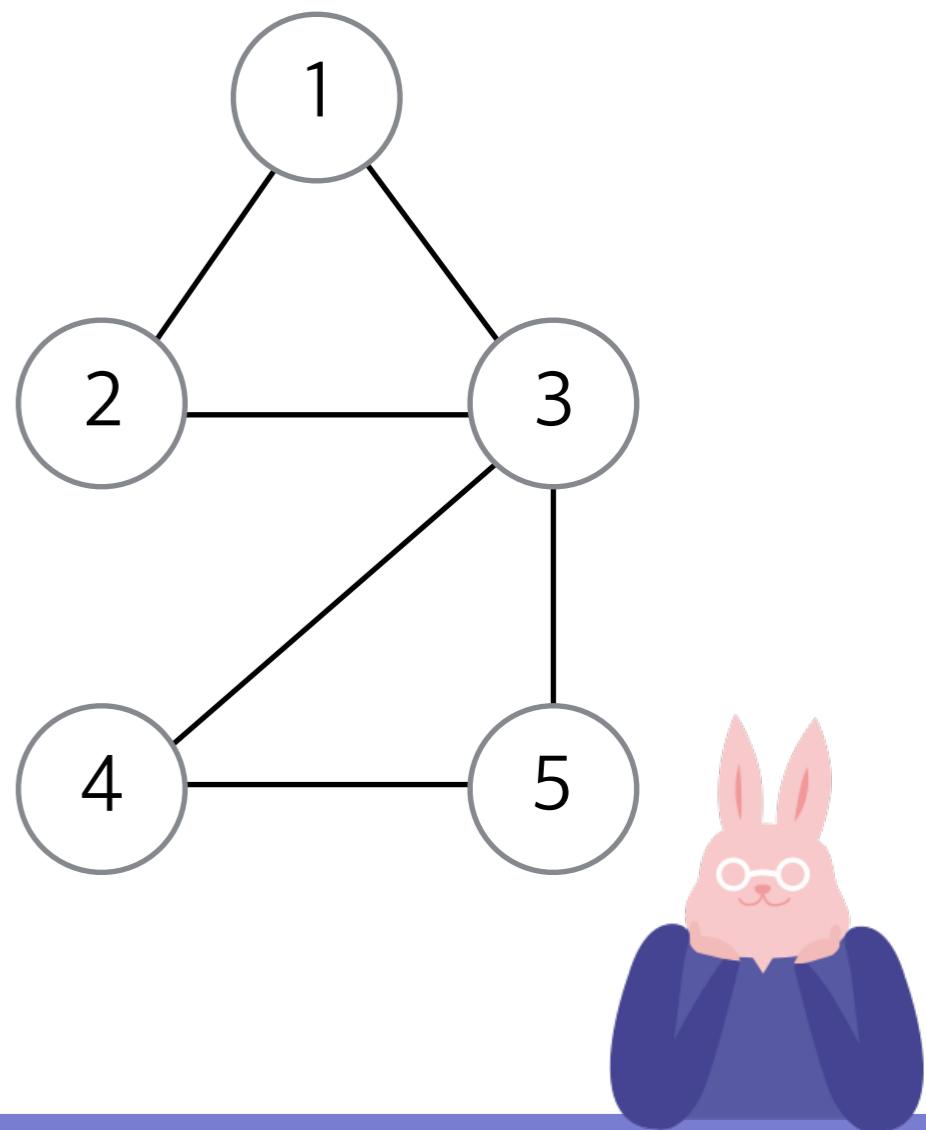
# 요약

- Dynamic Programming이 컴퓨터 공학적 생각의 꽃이다
  - 전체 문제를 부분 문제로 어떻게 나눌 것인가 ?
- 많은 문제를 풀어보는 것이 관건이다
  - 풀다 보면 패턴이 조금씩 보일 수 있음



# 그래프 (Remind)

- Node와 Edge로 이루어져 있는 자료구조
  - Node와 Edge에 정보를 저장한다
- 용어
  - Path : 두 노드 사이의 길
  - Cycle : 자기 자신으로 돌아오는 길



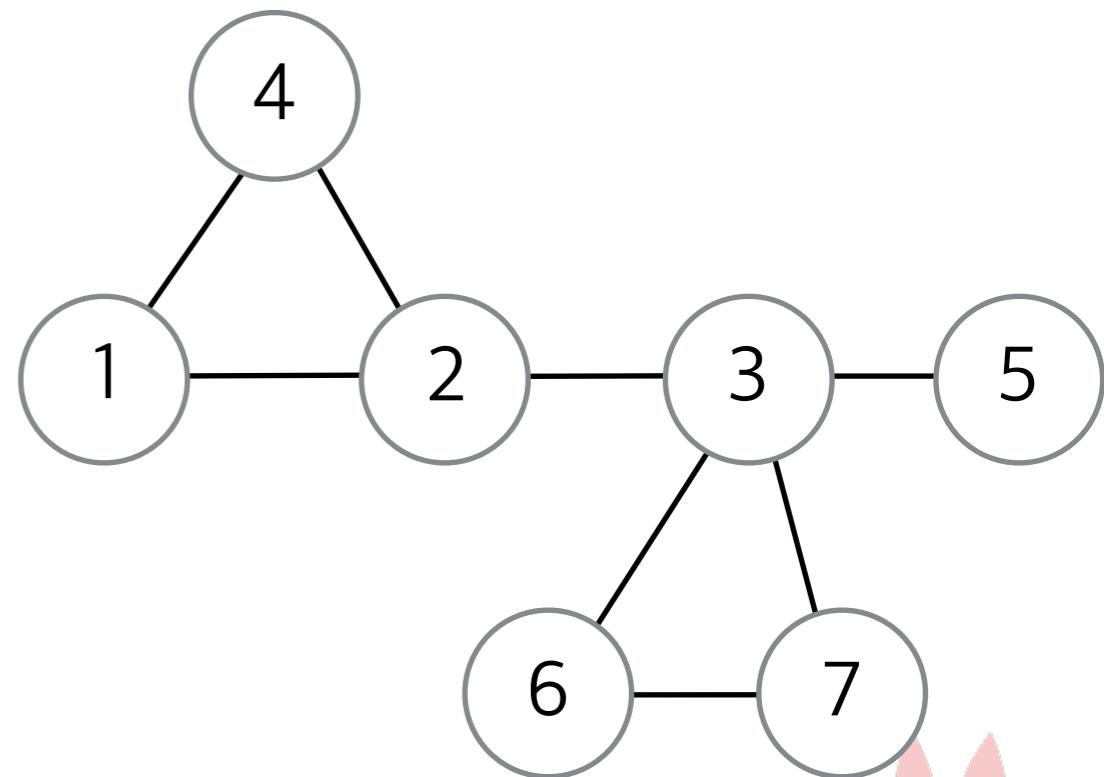
# 그래프 순회

- 깊이 우선 탐색 (DFS)
- 너비 우선 탐색 (BFS)



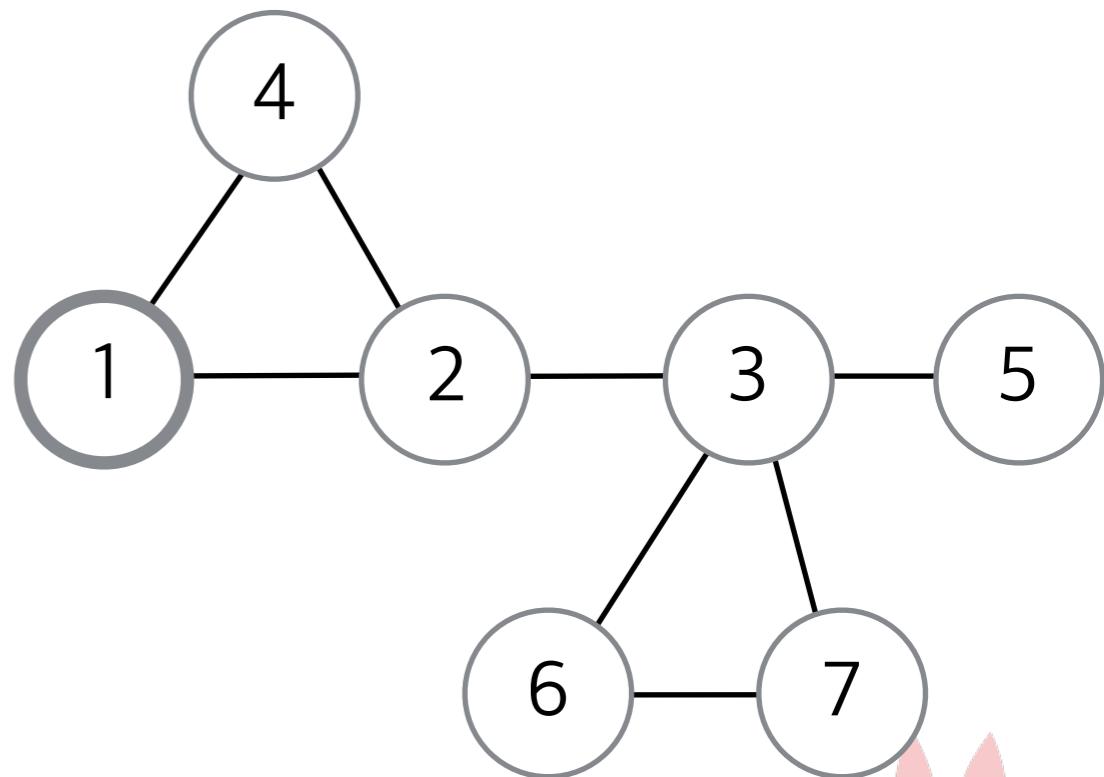
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



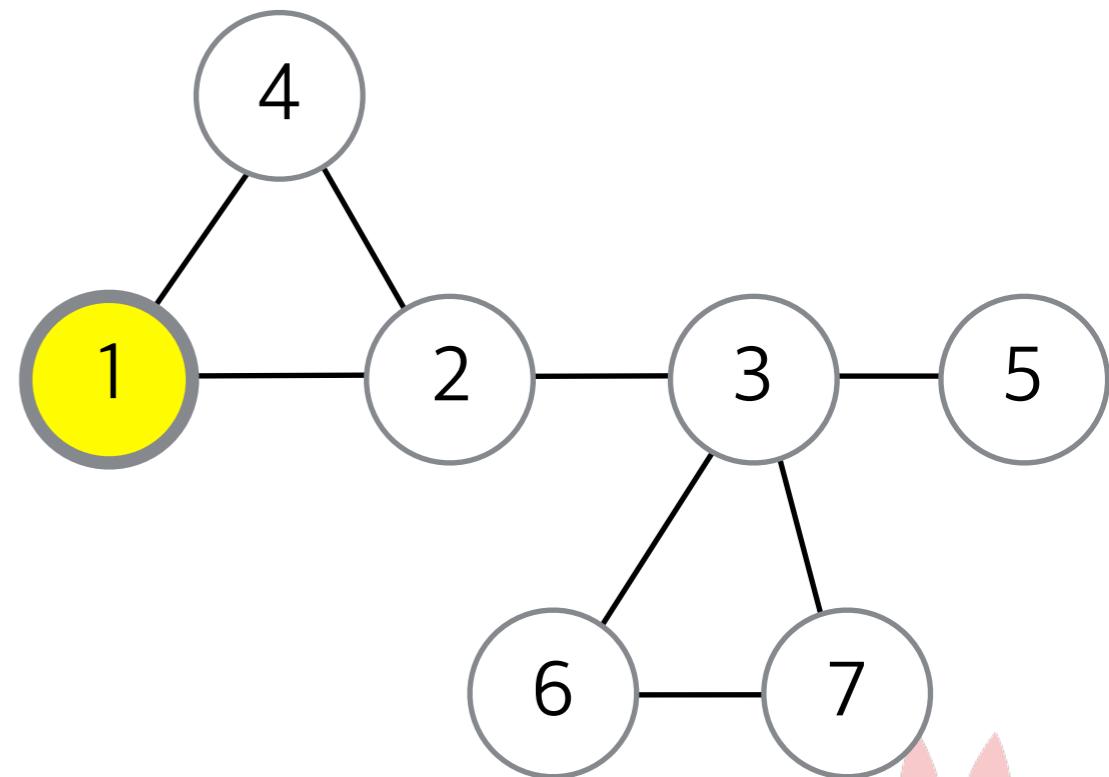
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



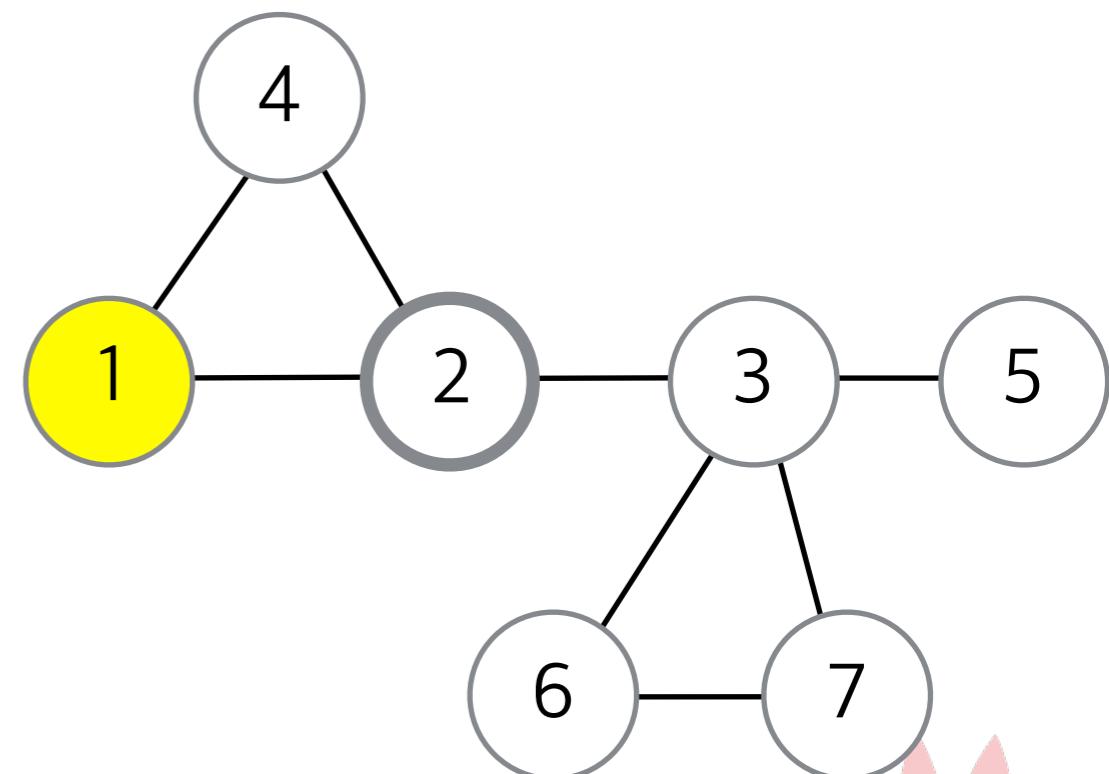
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



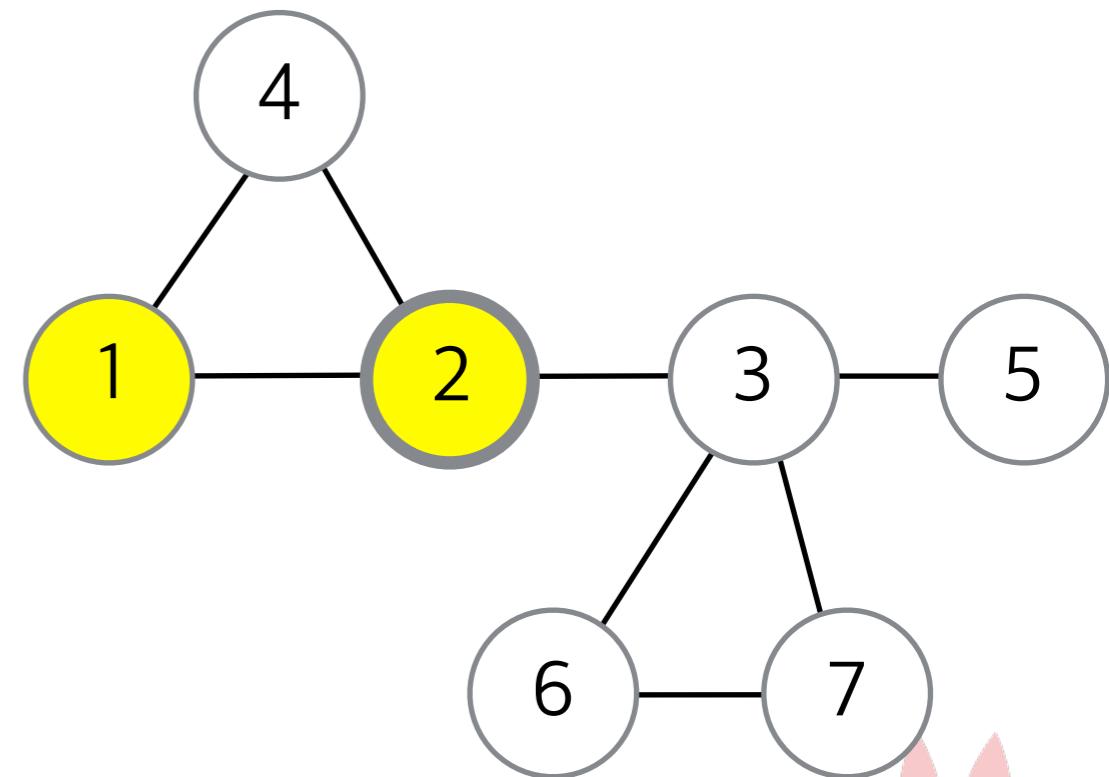
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



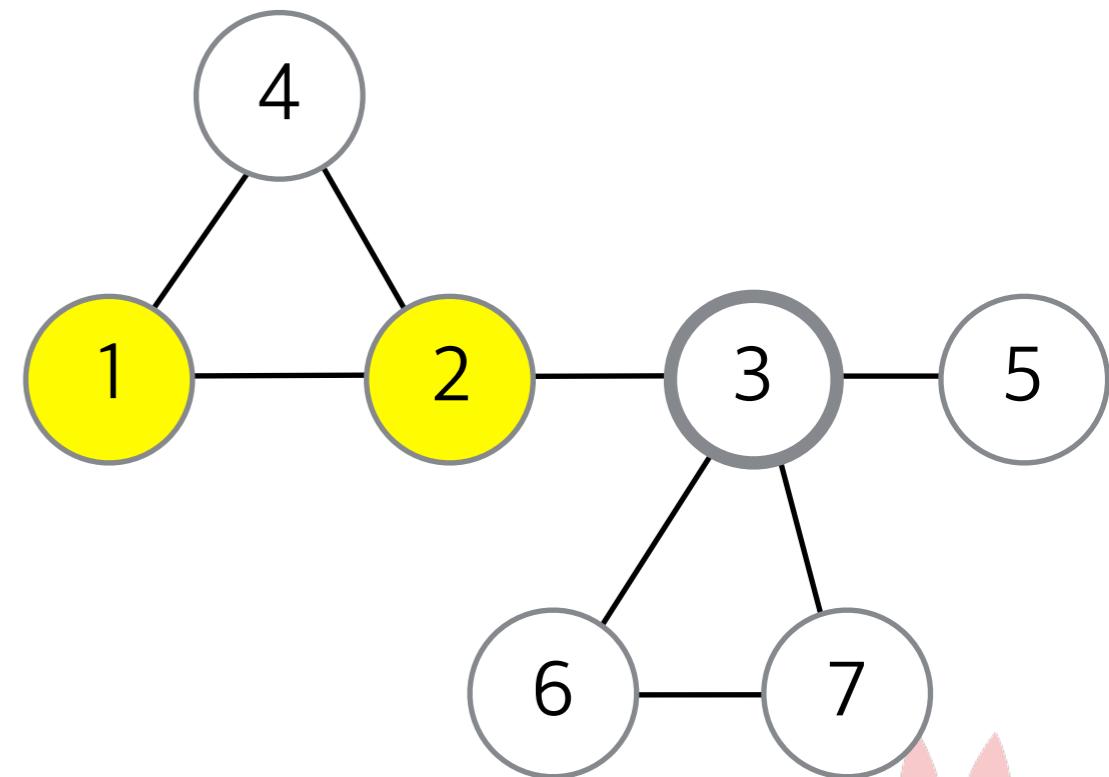
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



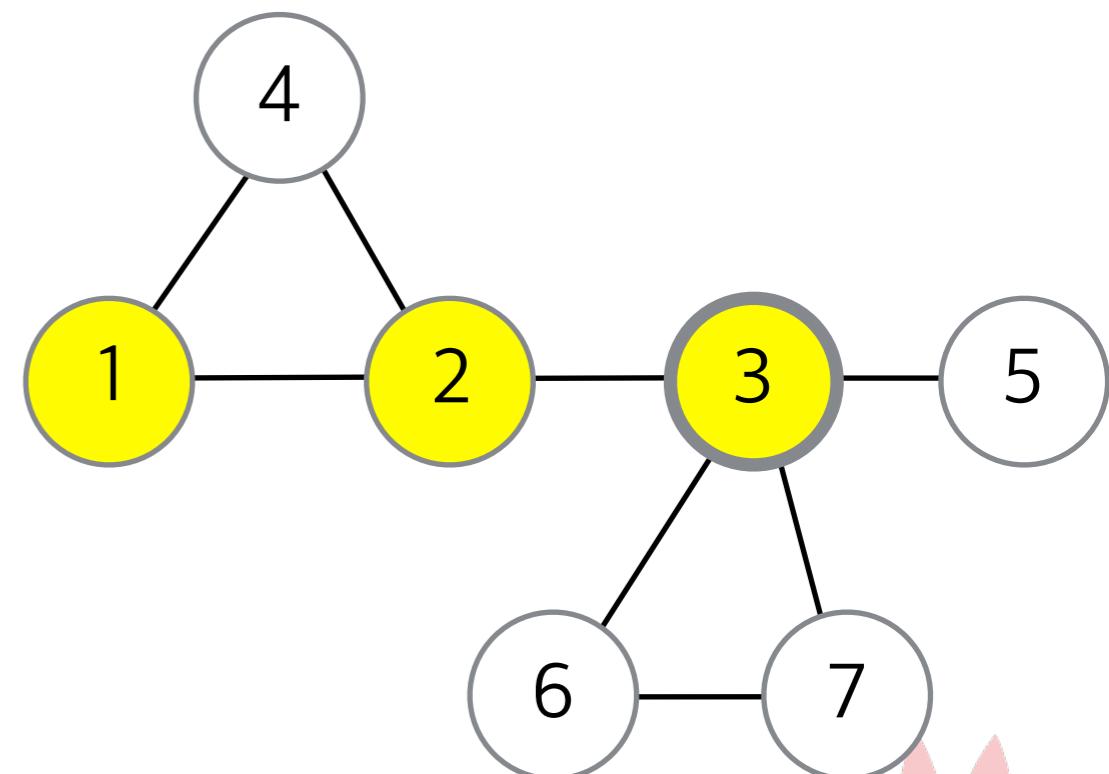
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



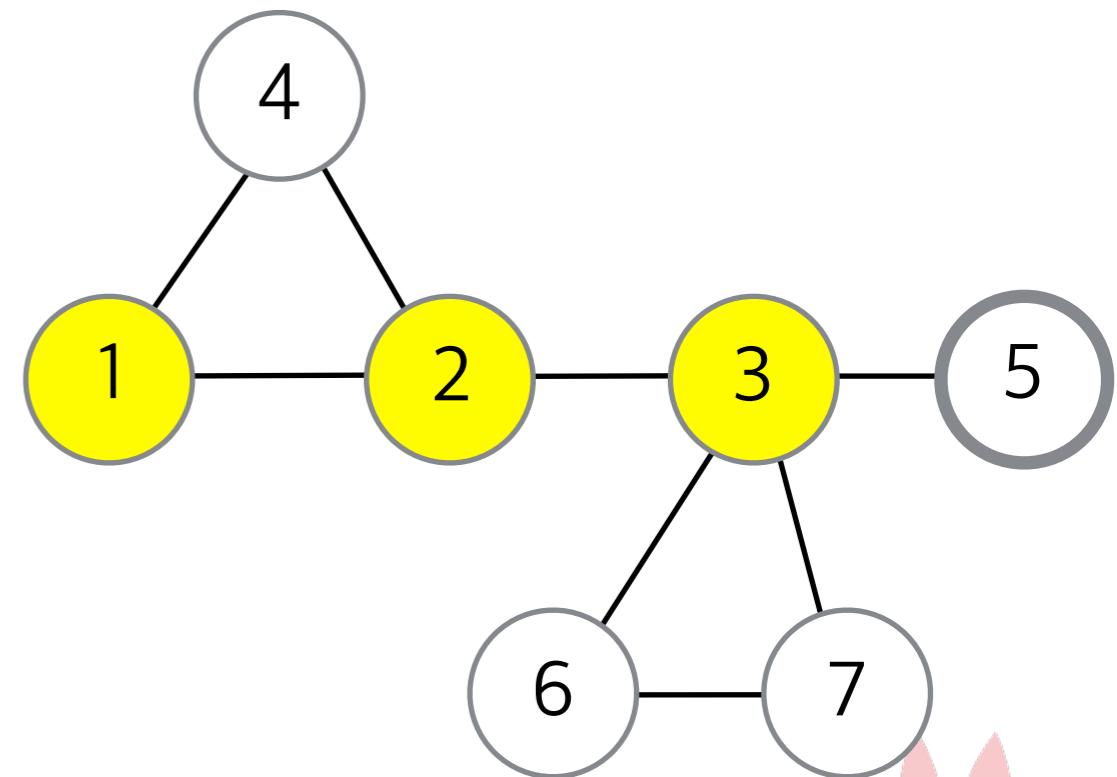
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



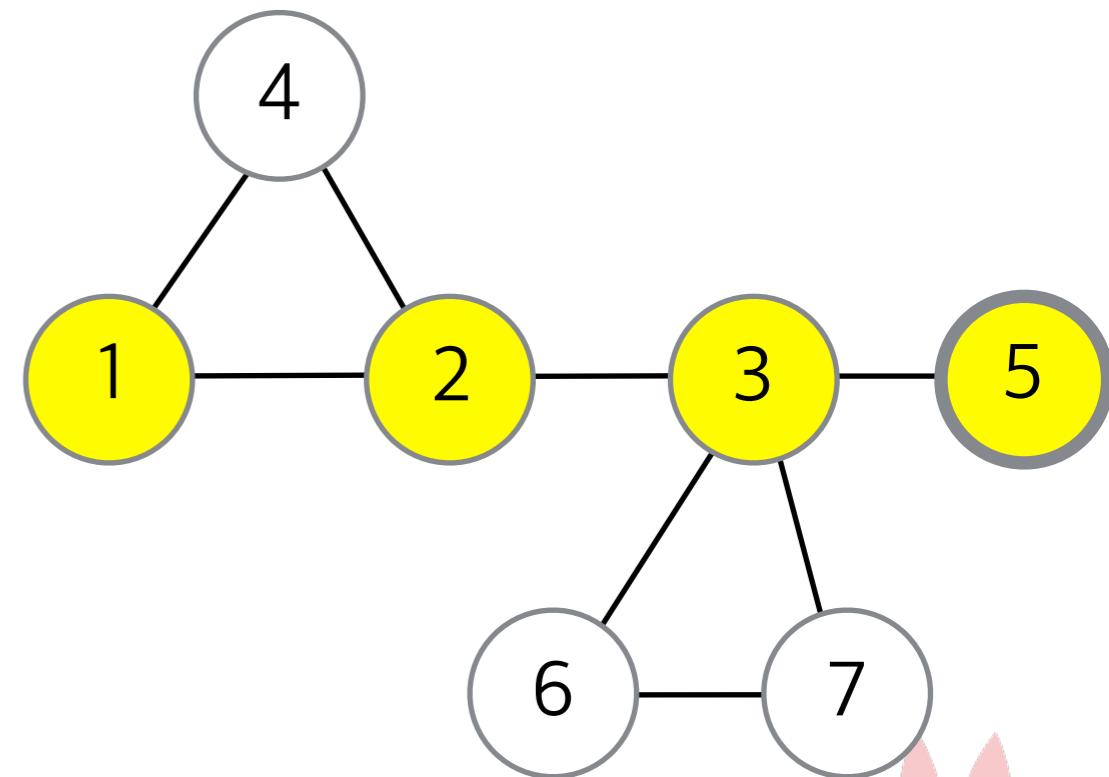
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



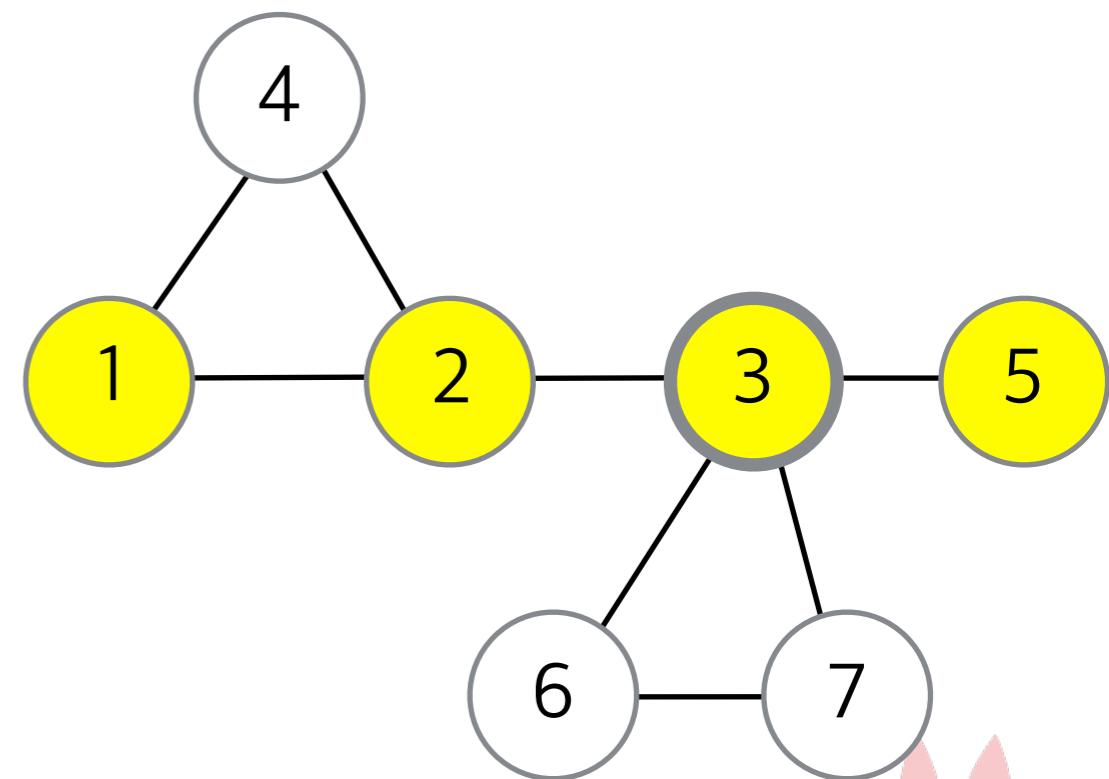
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



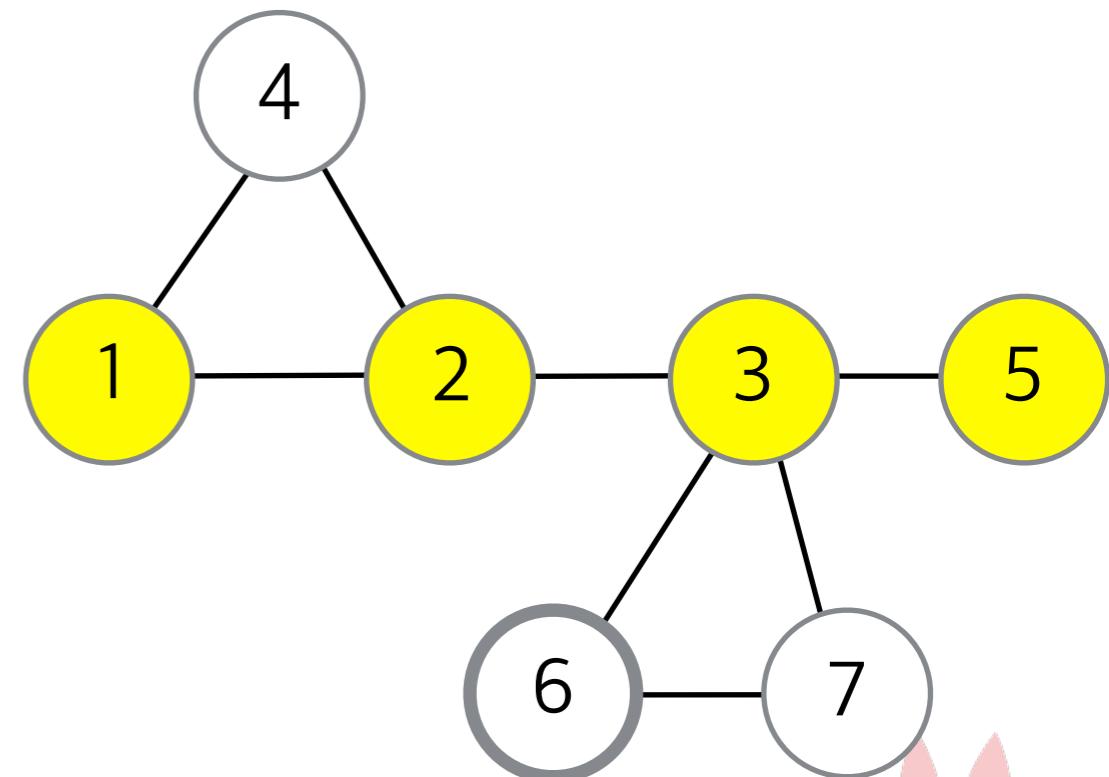
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



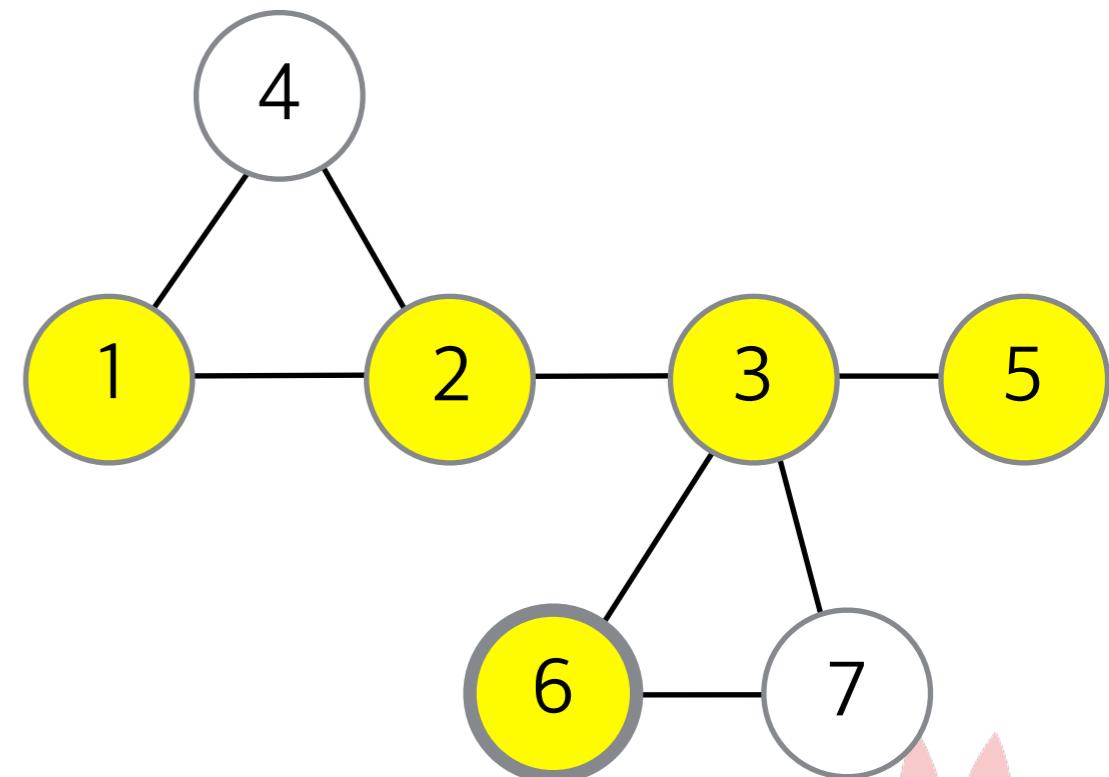
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



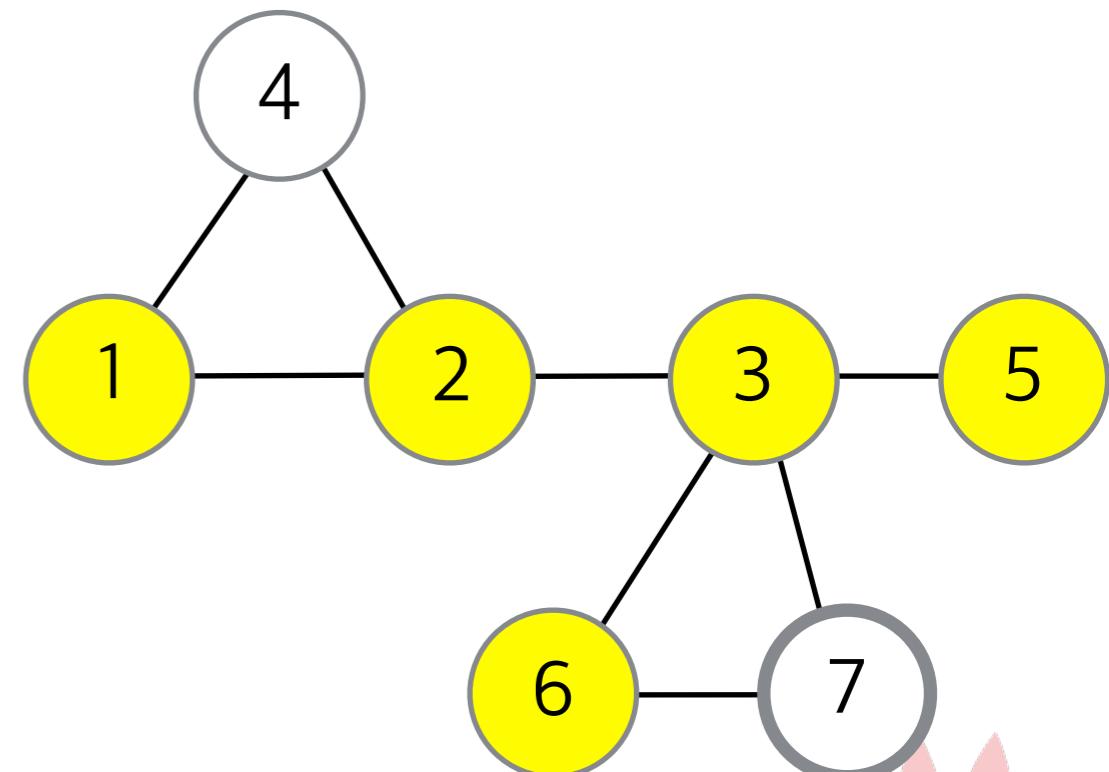
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



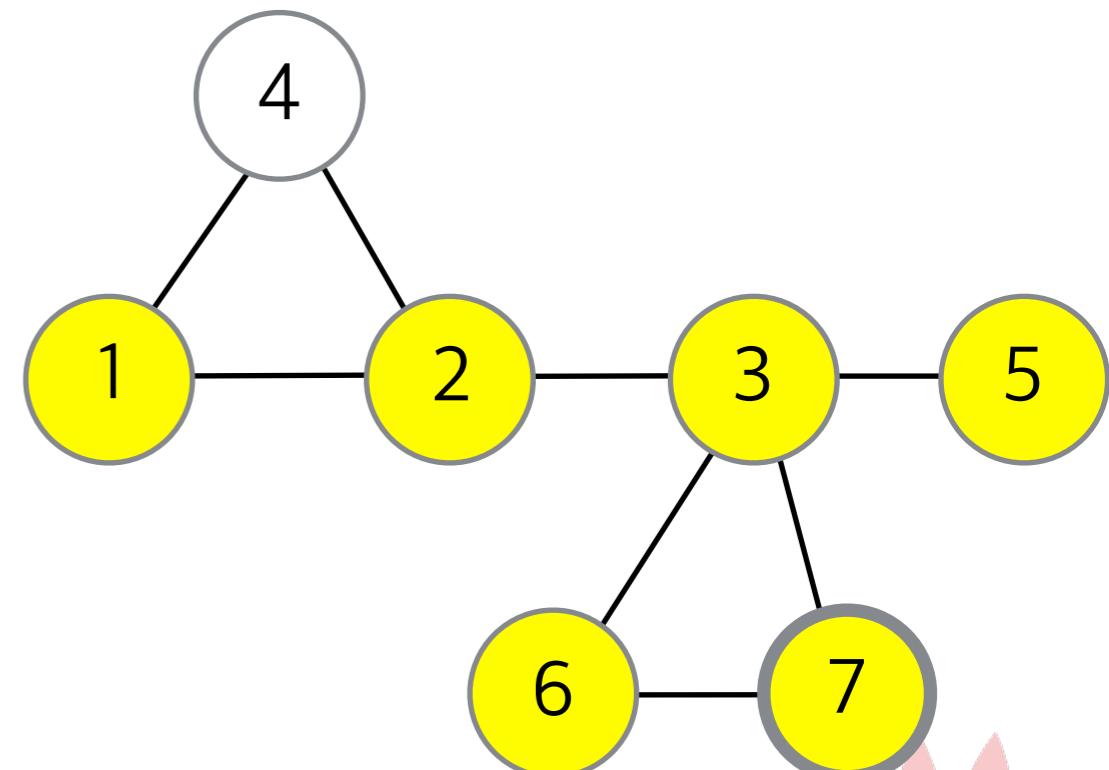
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



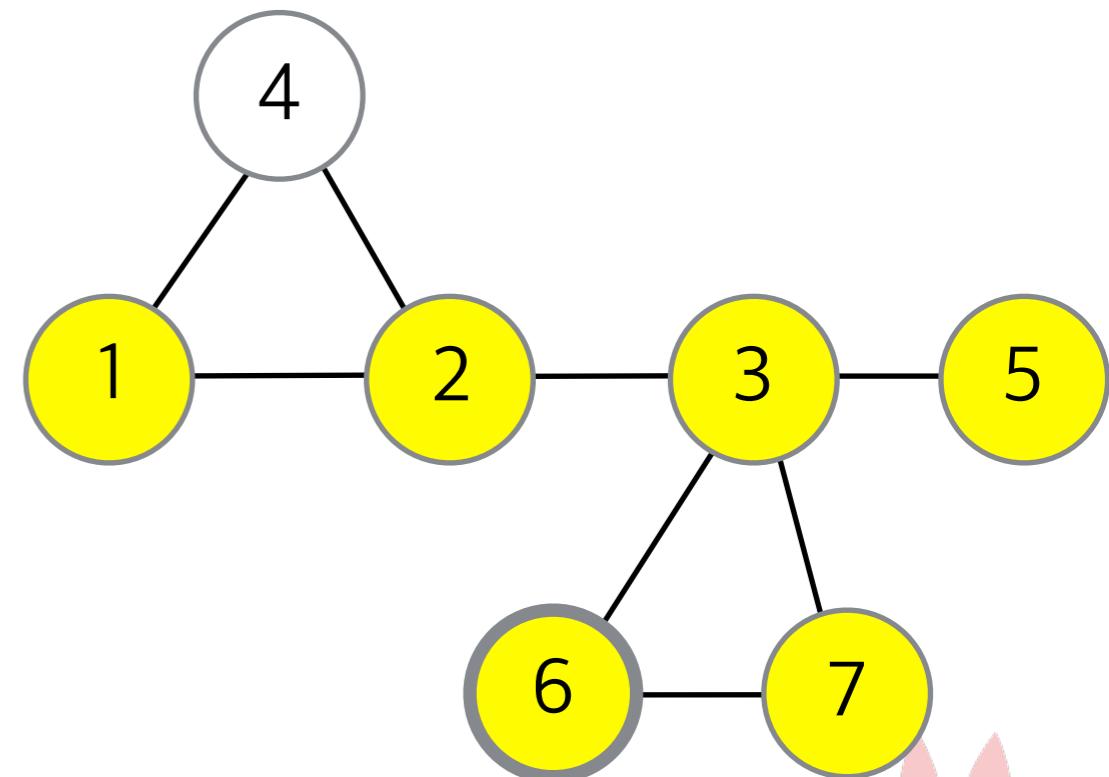
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



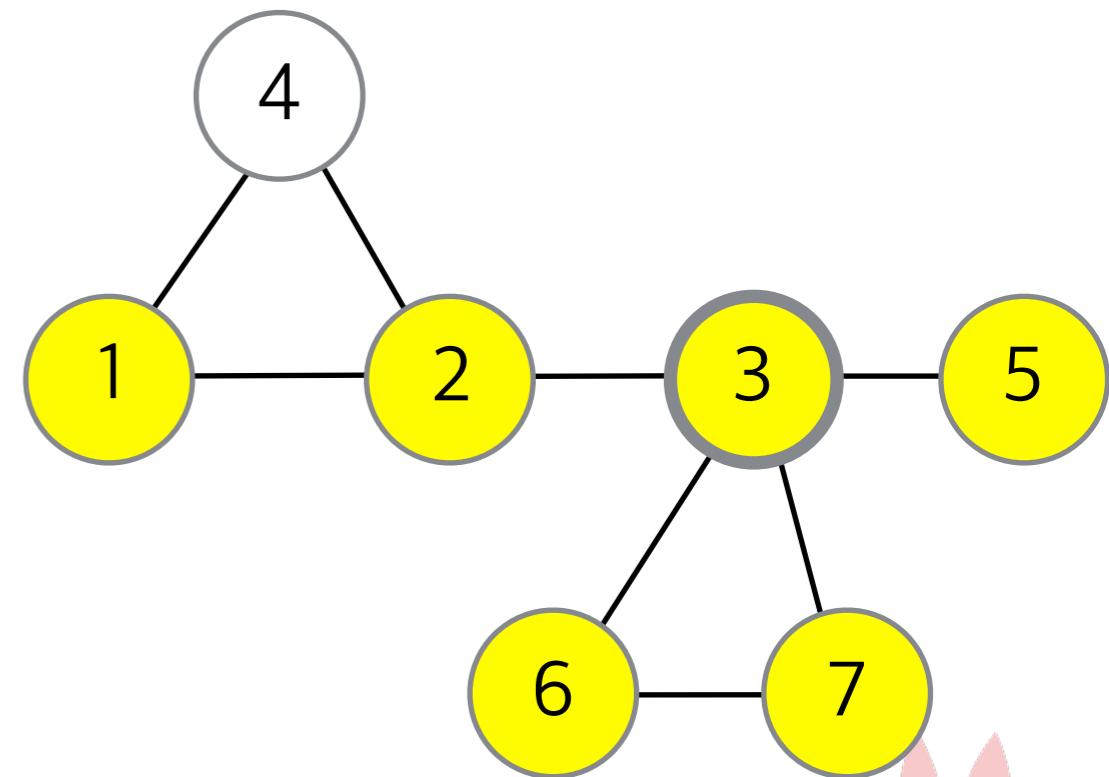
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



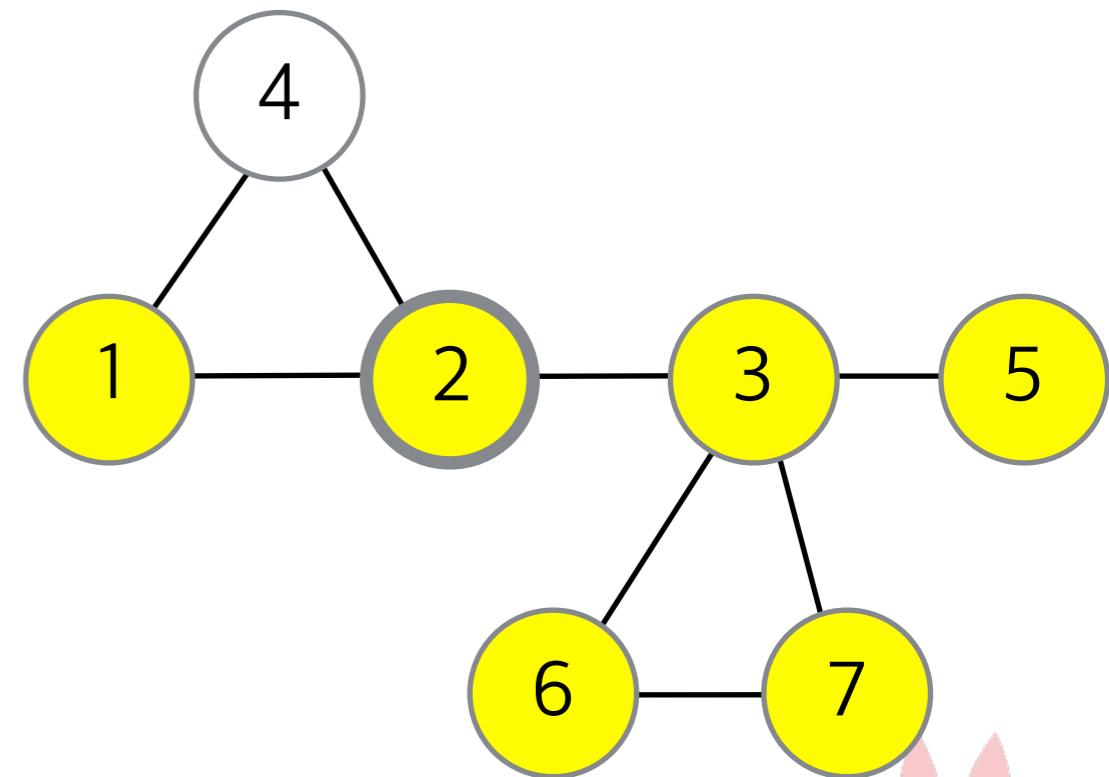
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



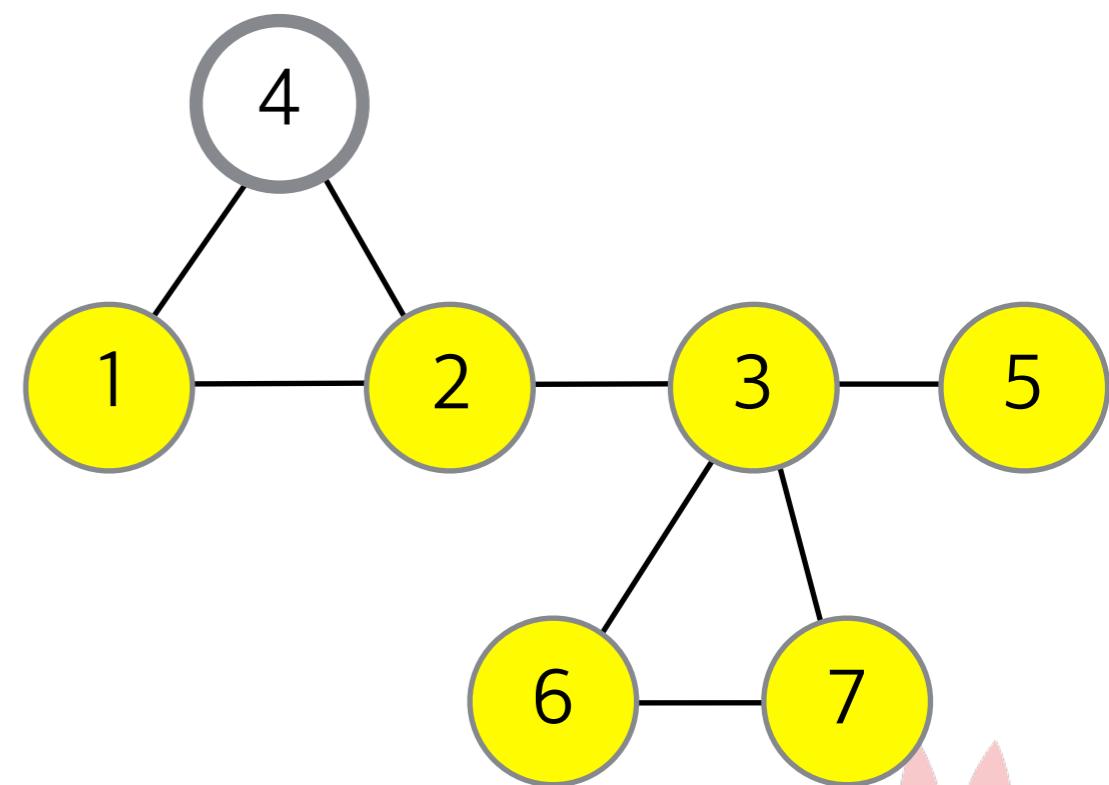
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



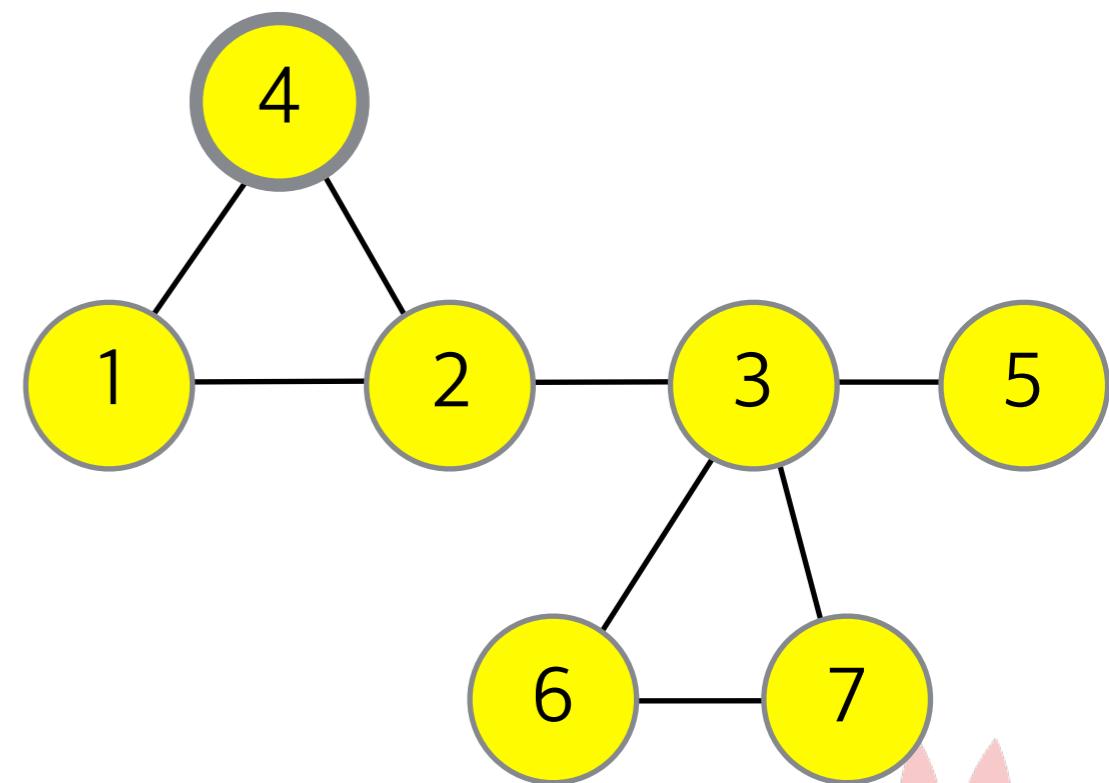
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



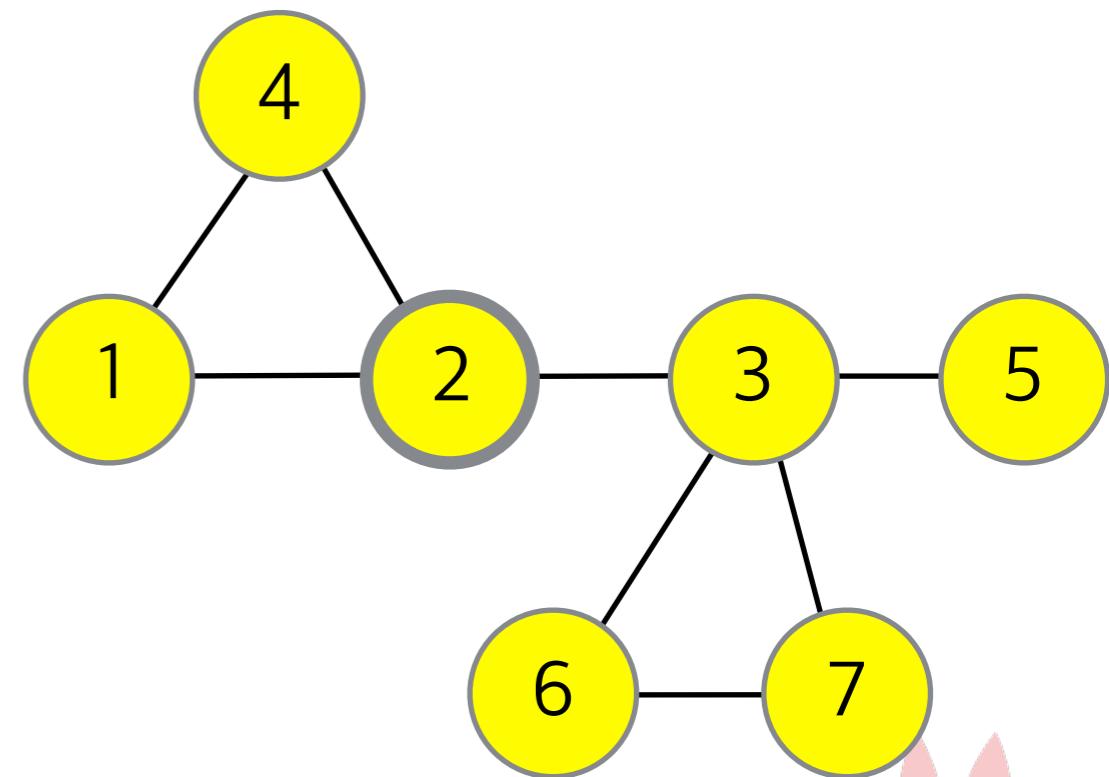
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



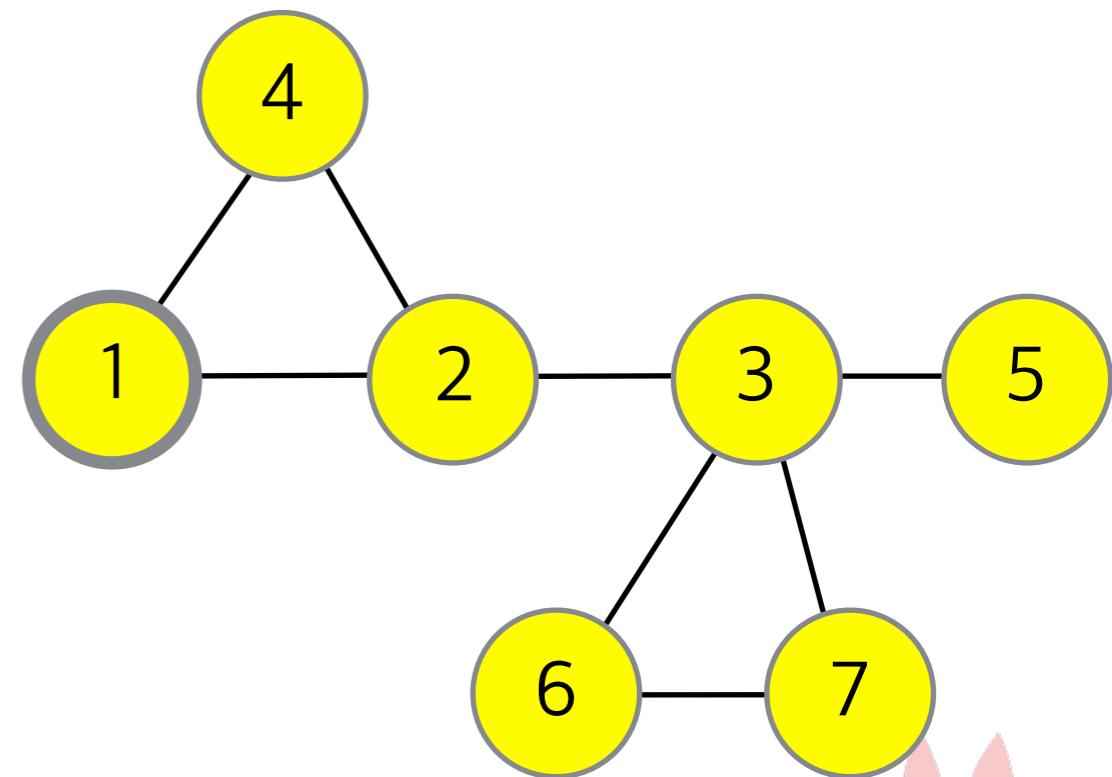
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



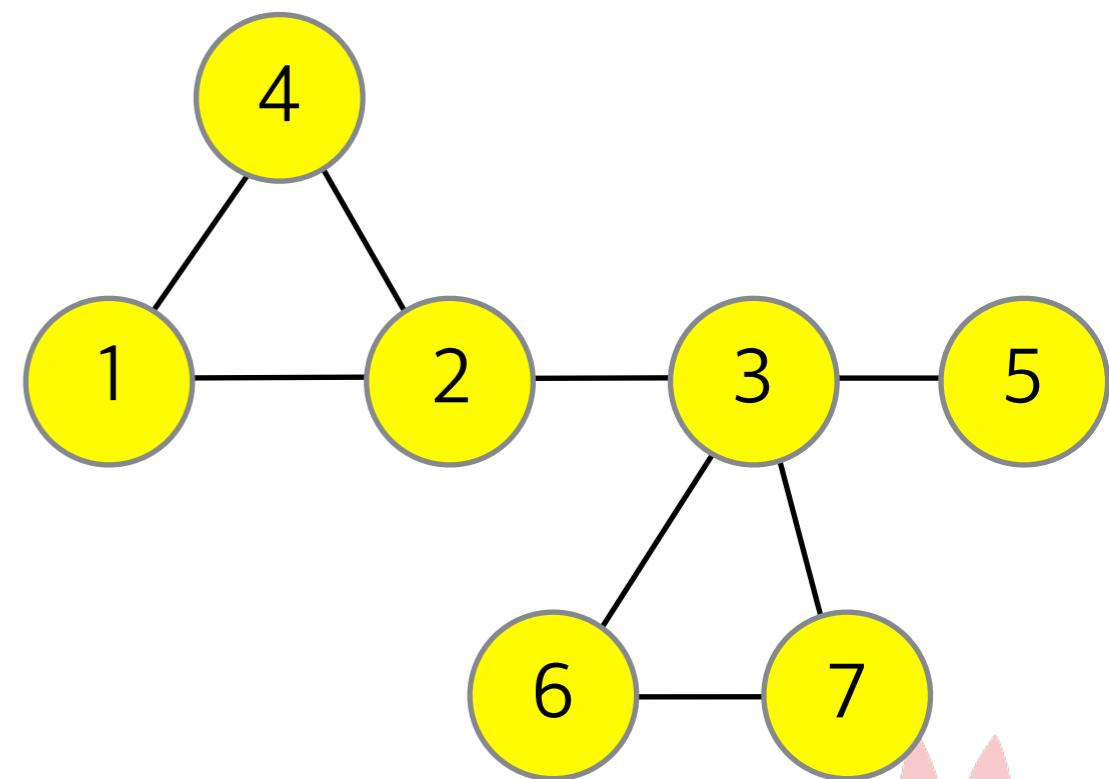
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



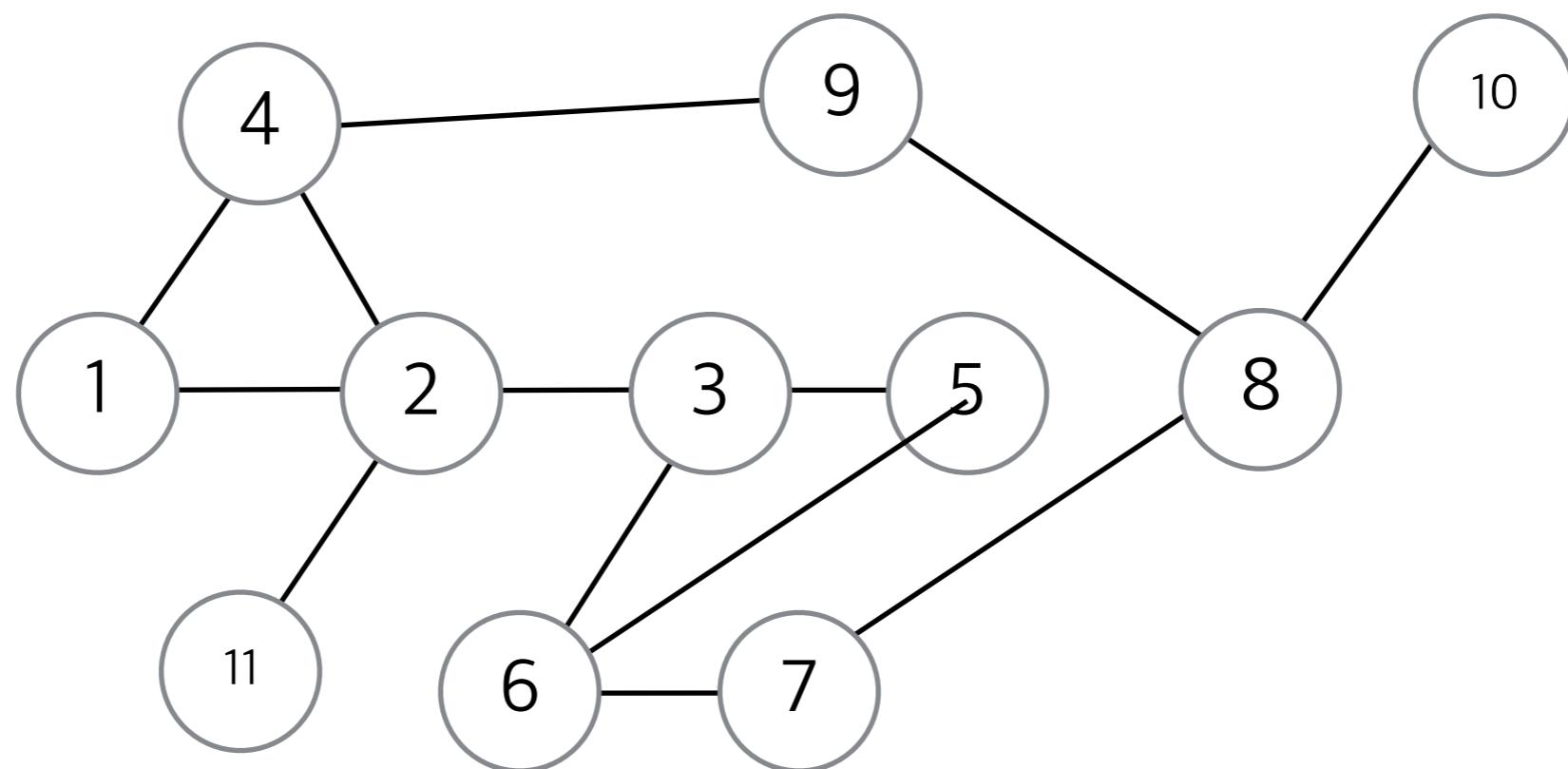
# 깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



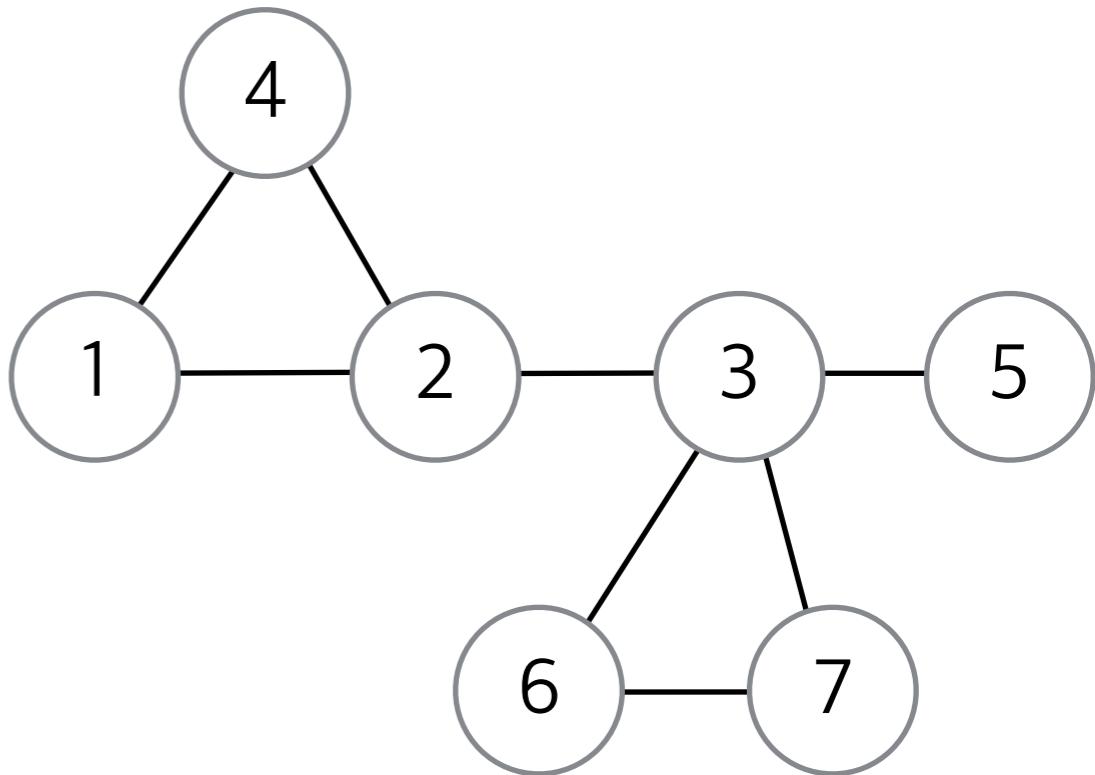
# 깊이 우선 탐색 (DFS)

- 다음 그래프에 대하여 1을 시작으로 DFS한 결과는 ?
  - 단, 인접한 노드가 여러개일 경우 노드 번호가 작은 노드부터 방문



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :  
    visited[x] = True  
    result = [x]  
  
    for v in graph[x] :  
        if visited[v] == False :  
            result = result +  
                DFS(graph, v, visited)  
  
    return result
```

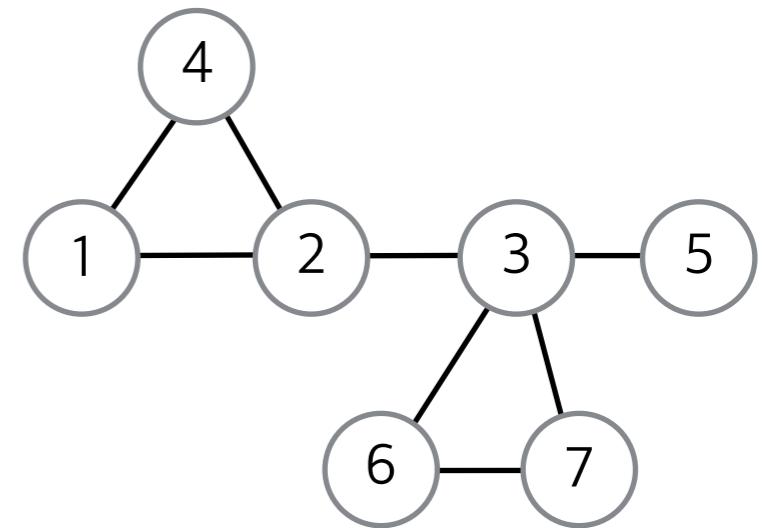


# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]  
graph[2] = [3, 4]  
graph[3] = [2, 5, 6, 7]  
graph[4] = [1, 2]  
graph[5] = [3]  
graph[6] = [3, 7]  
graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

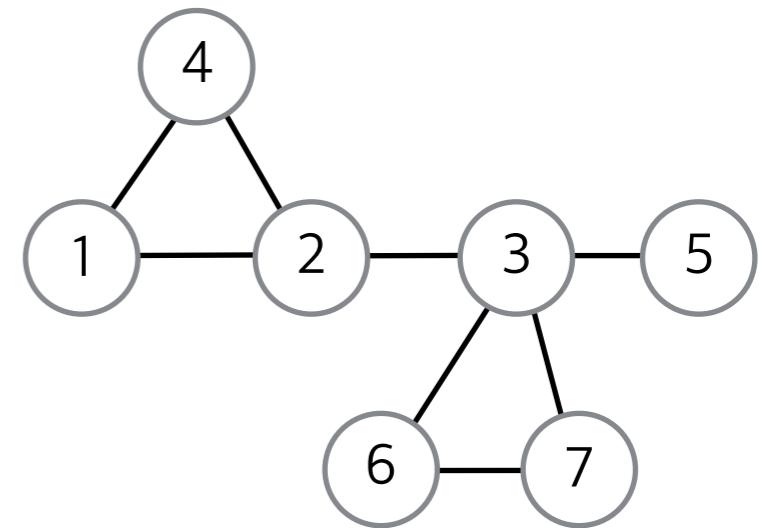
```
→ def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

result = null



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

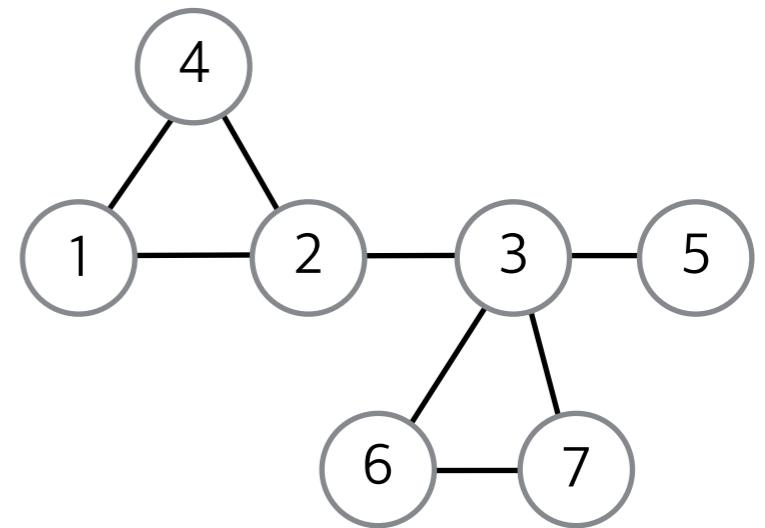
    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

result = null



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

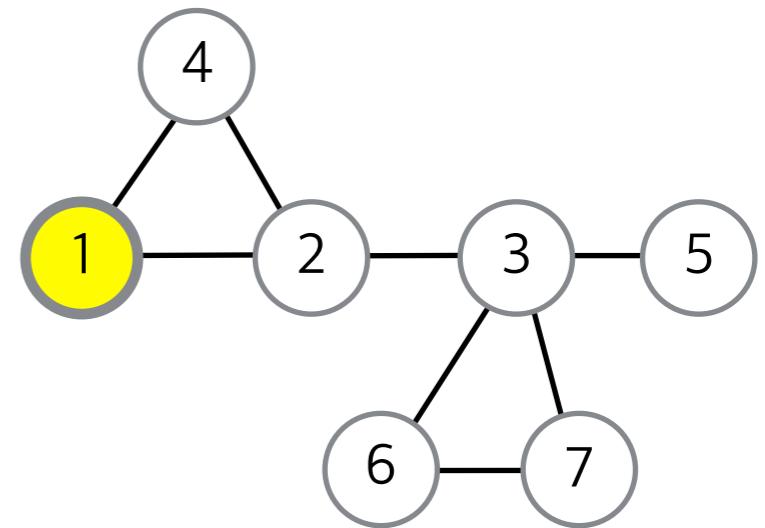
    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

result = null



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] =[2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

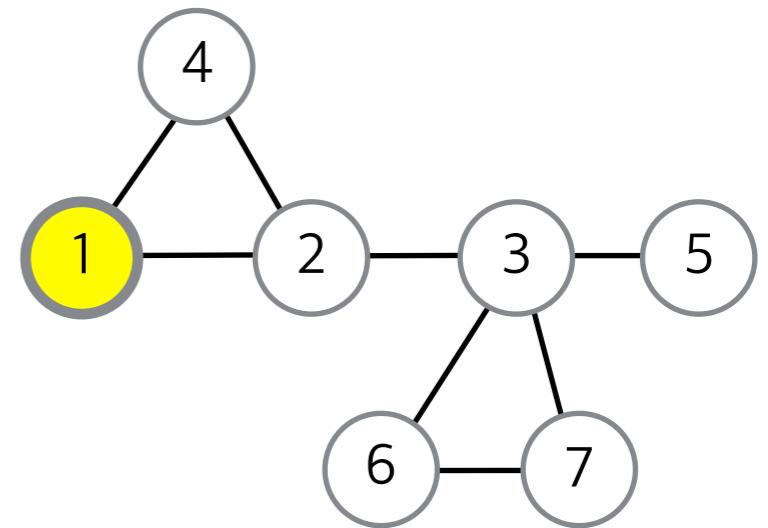
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

result = [1]



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] =[2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

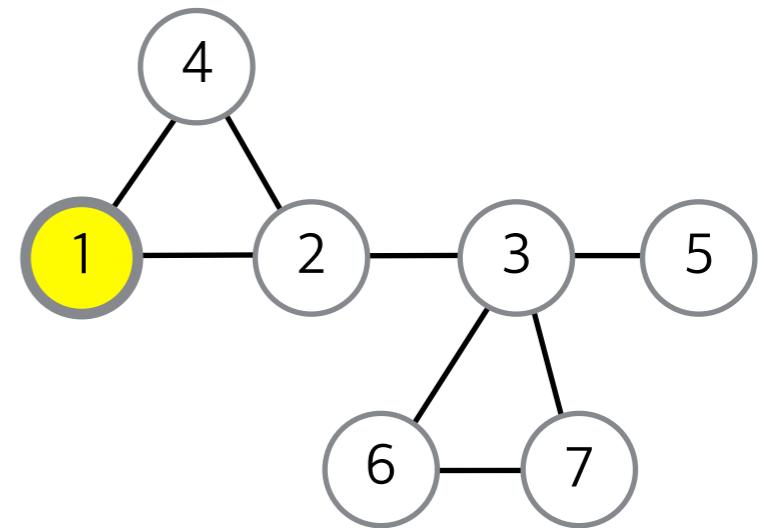
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

result = [1], v = 2



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

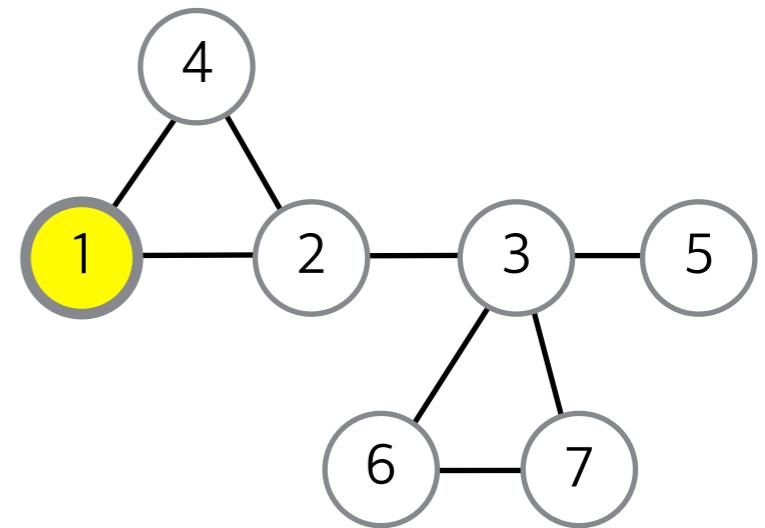
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

→

DFS(graph, 1, visited)

result = [1], v = 2



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
→ def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

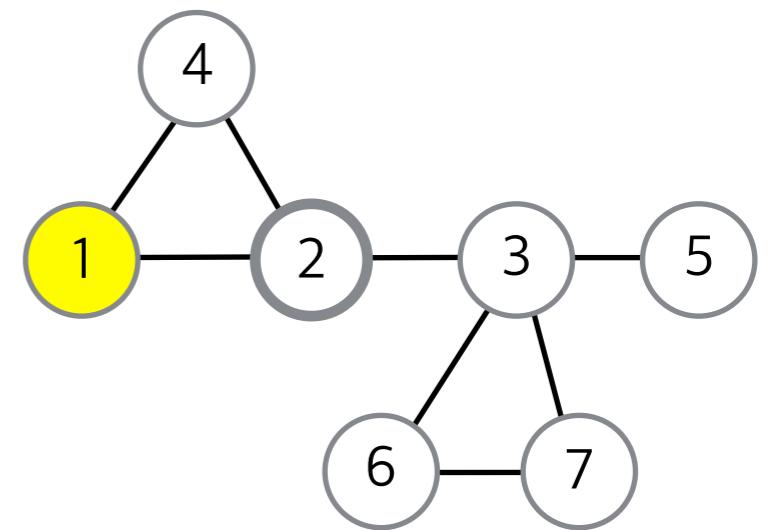
    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = null



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

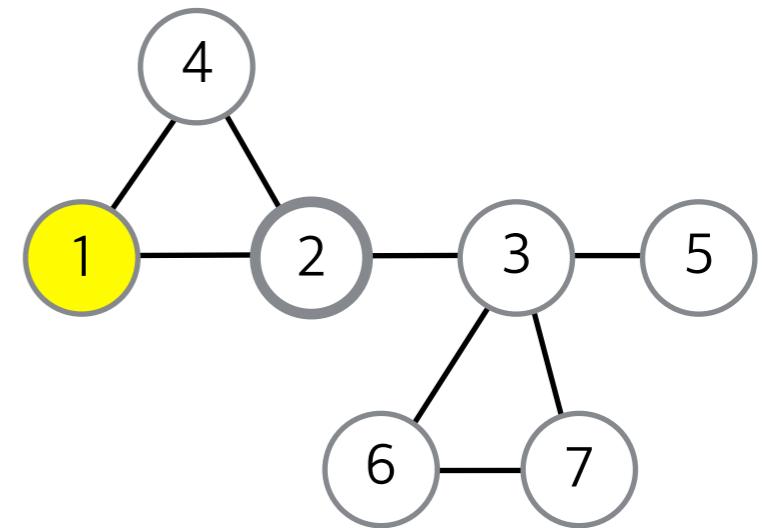
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = null



visited= [F, T, F, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

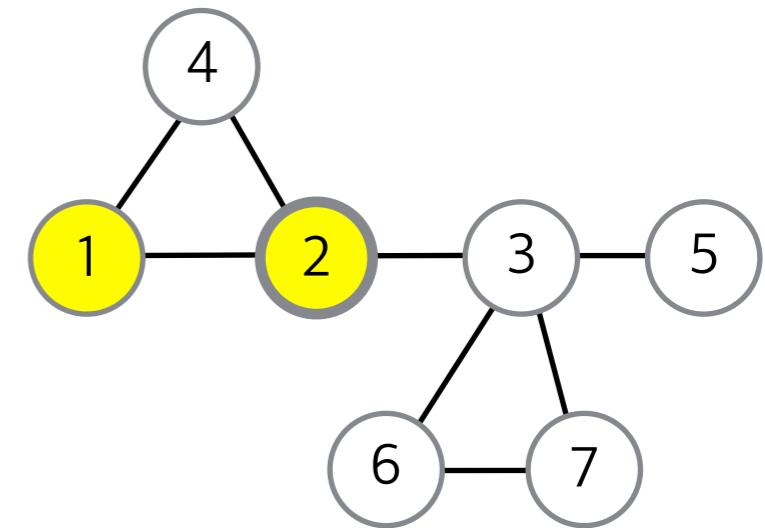
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = null



visited= [F, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

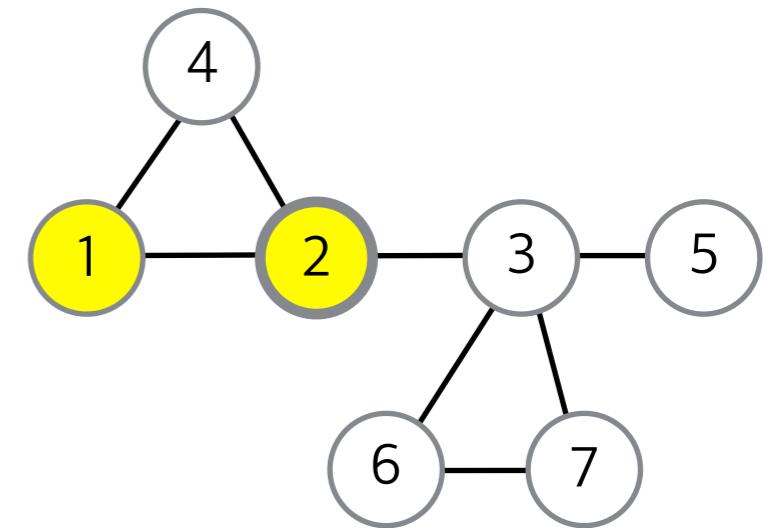
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2]



visited= [F, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

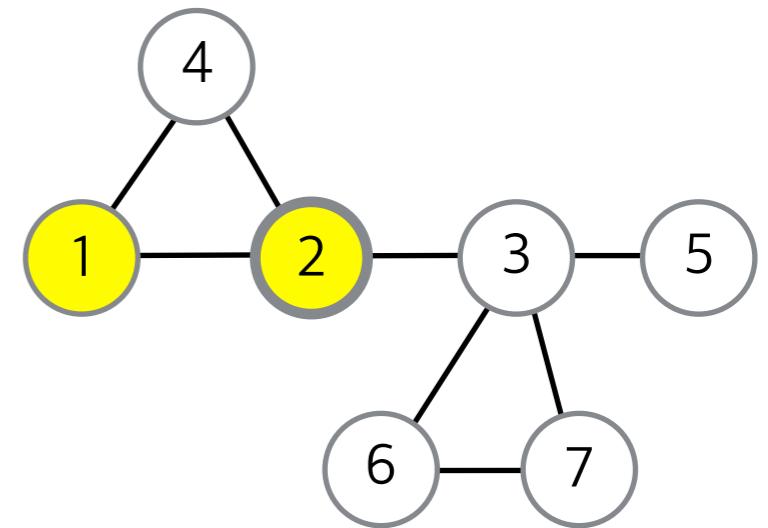
    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3



visited= [F, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

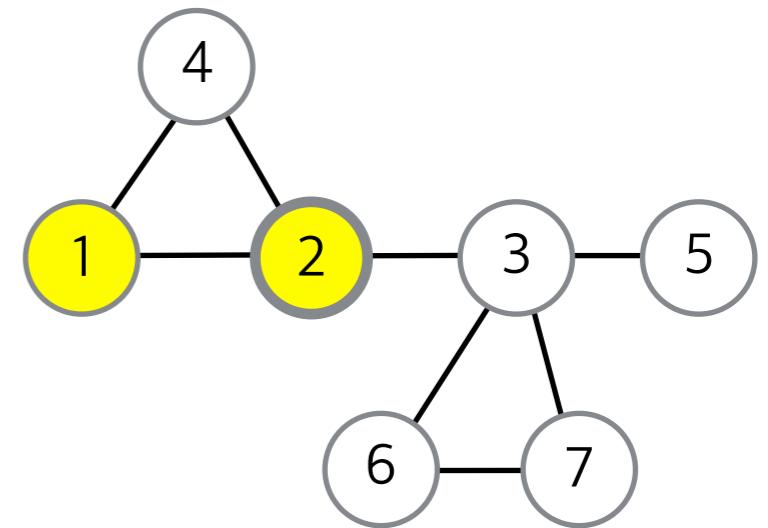
    return result
```

DFS(graph, 1, visited)

result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3



visited= [F, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
→ def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

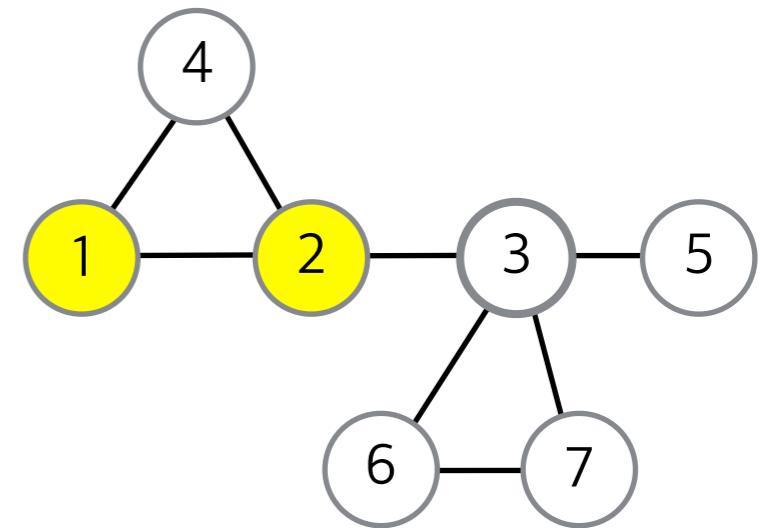
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = null



visited= [F, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    →    visited[x] = True
        result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

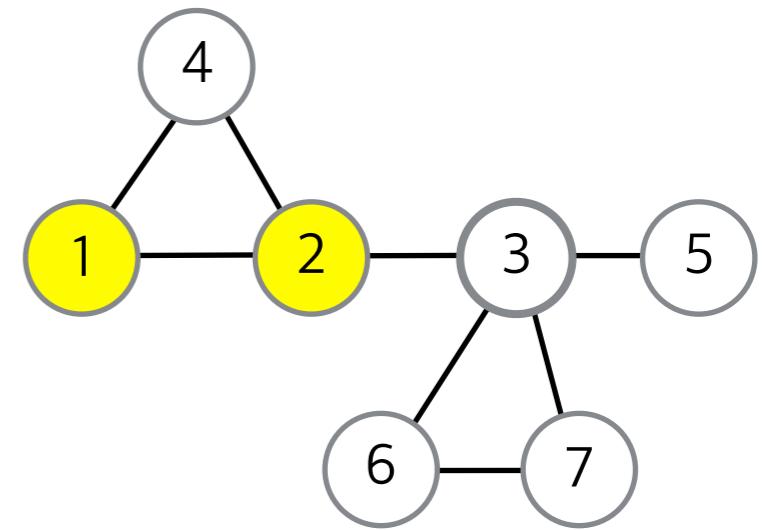
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = null



visited= [F, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

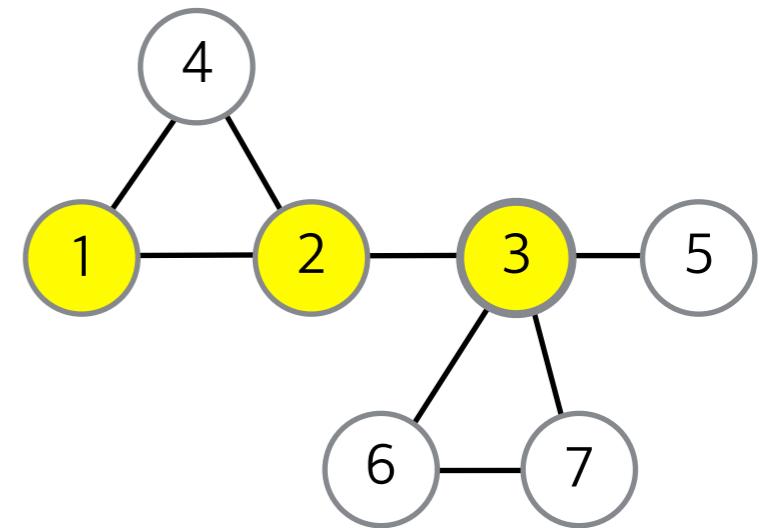
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = null



visited= [F, T, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

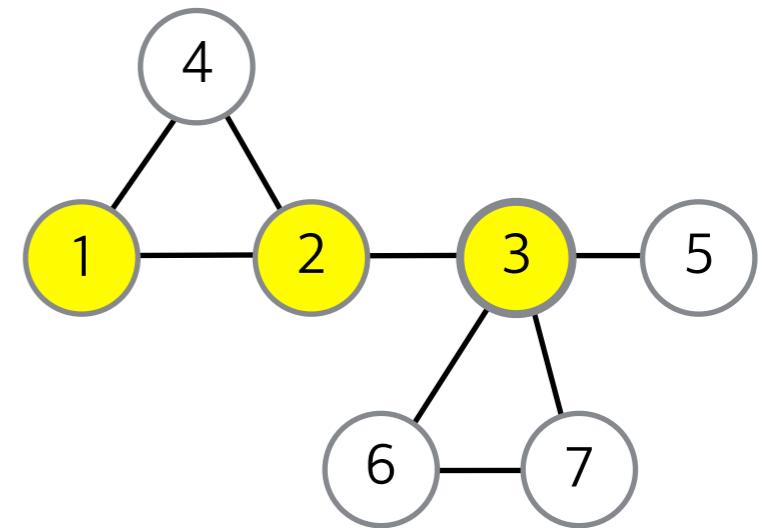
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3]



visited= [F, T, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

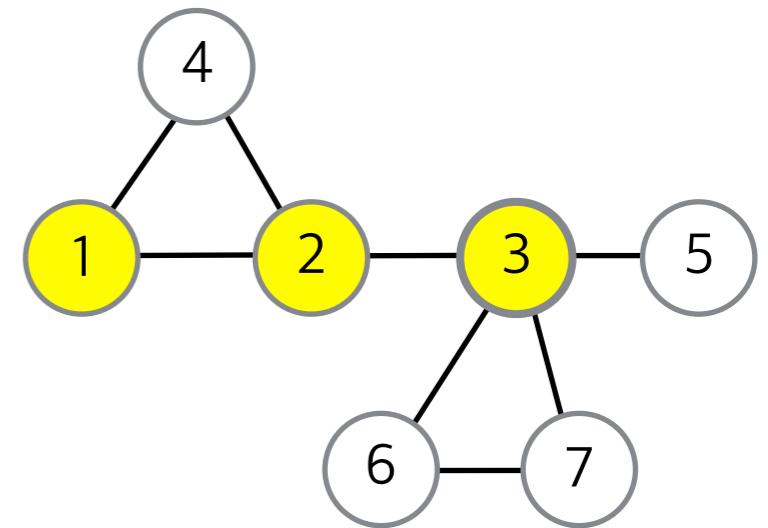
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 2



visited= [F, T, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

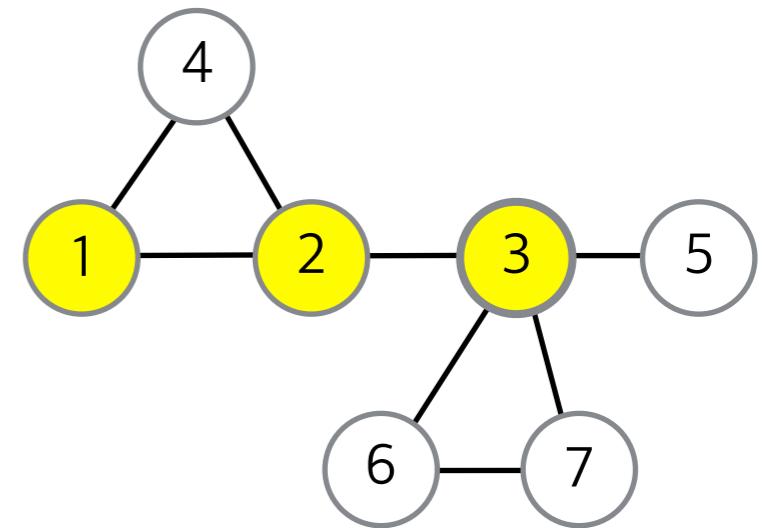
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 2



visited= [F, T, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

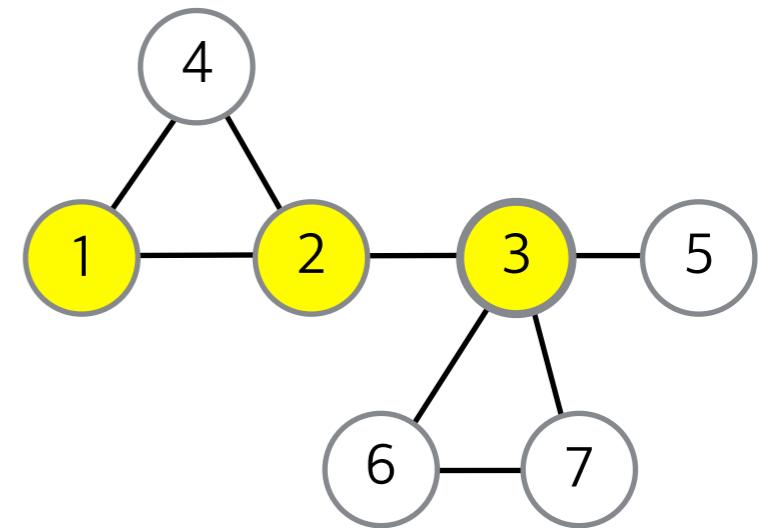
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5



visited= [F, T, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

→

DFS(graph, 1, visited)

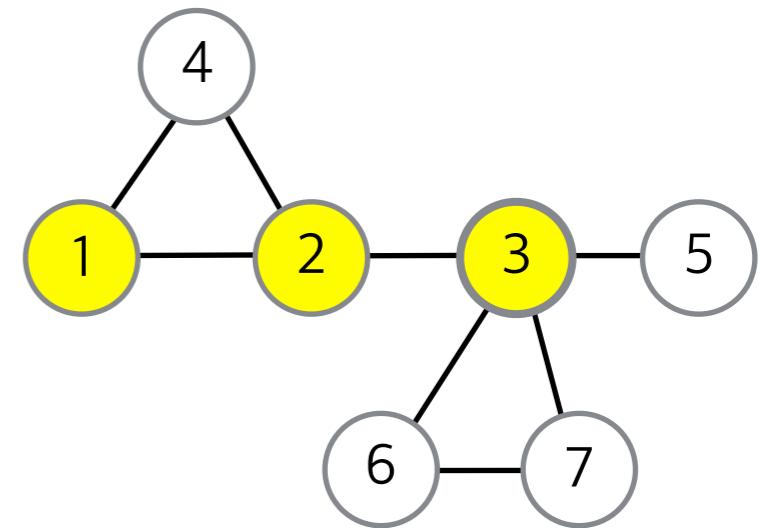
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5



visited= [F, T, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
→ def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

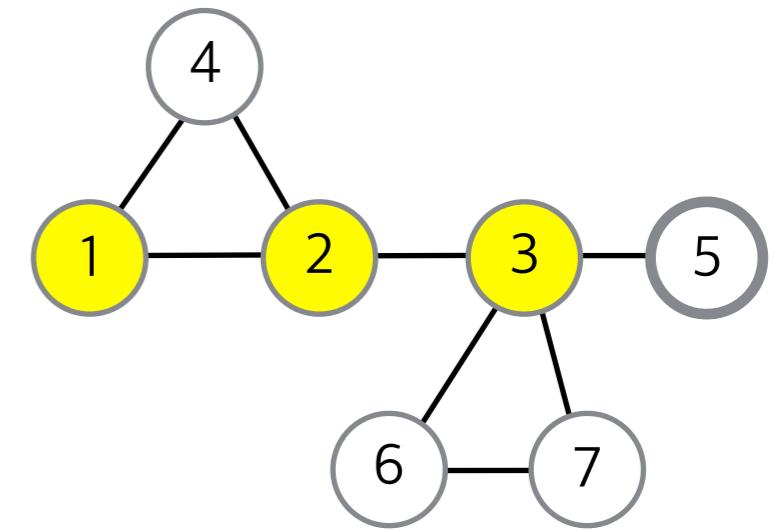
    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5

DFS(graph, 5, visited)

    result = null



visited= [F, T, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

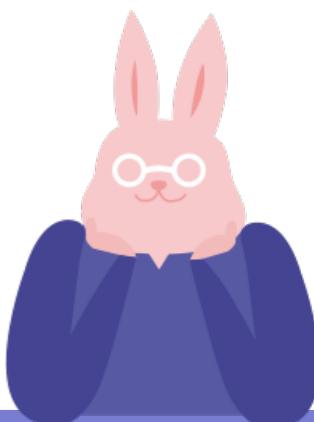
graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    → visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

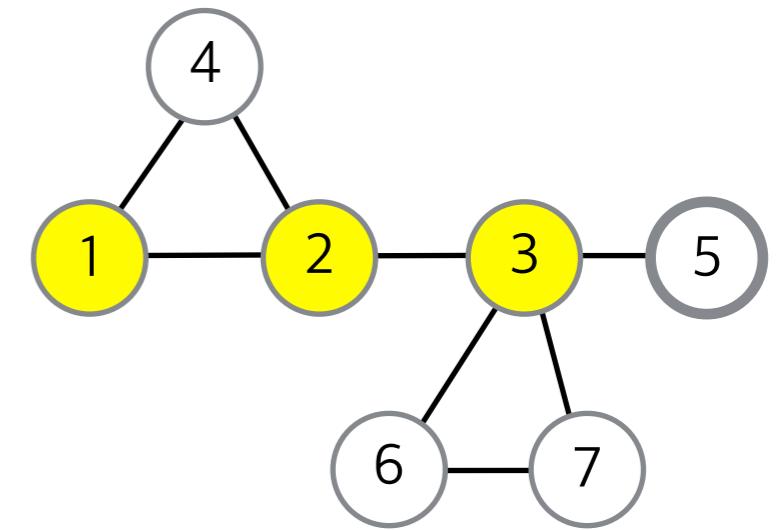
    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5

DFS(graph, 5, visited)

    result = null



visited= [F, T, T, T, F, F, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
  
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

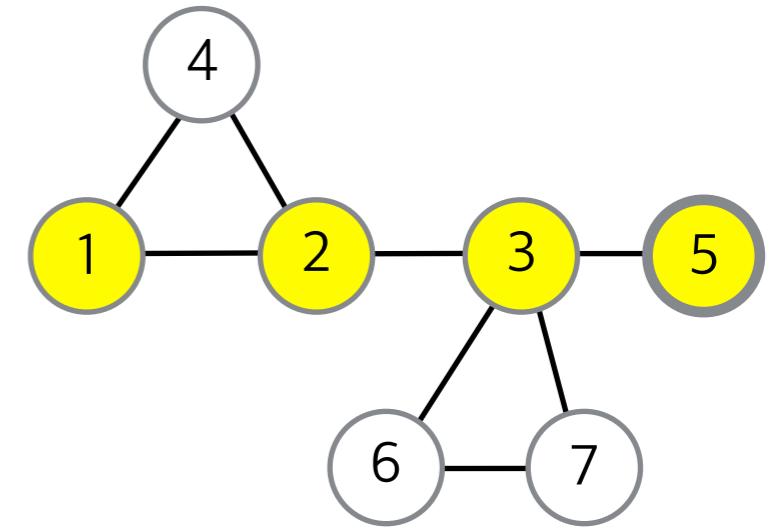
    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5

DFS(graph, 5, visited)

    result = null



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

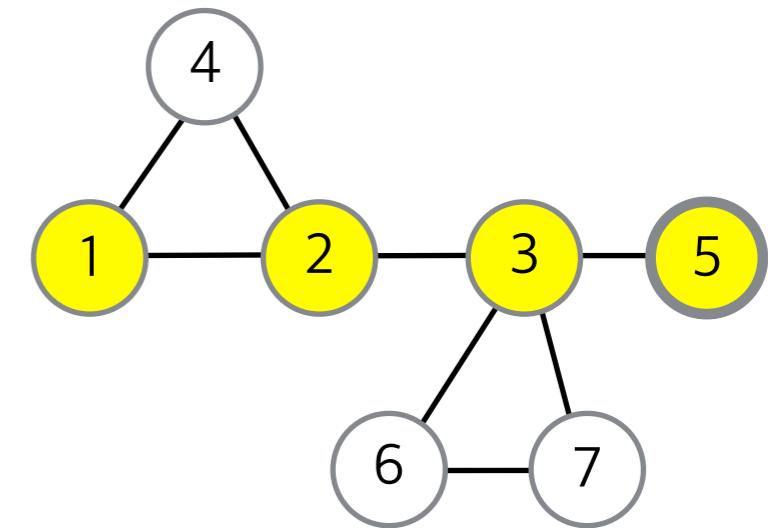
    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5

DFS(graph, 5, visited)

    result = [5]



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

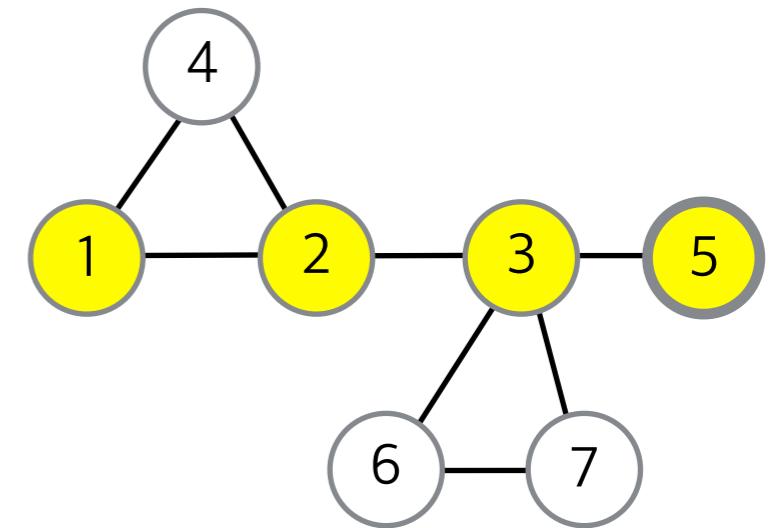
    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5

DFS(graph, 5, visited)

    result = [5], v = 3



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

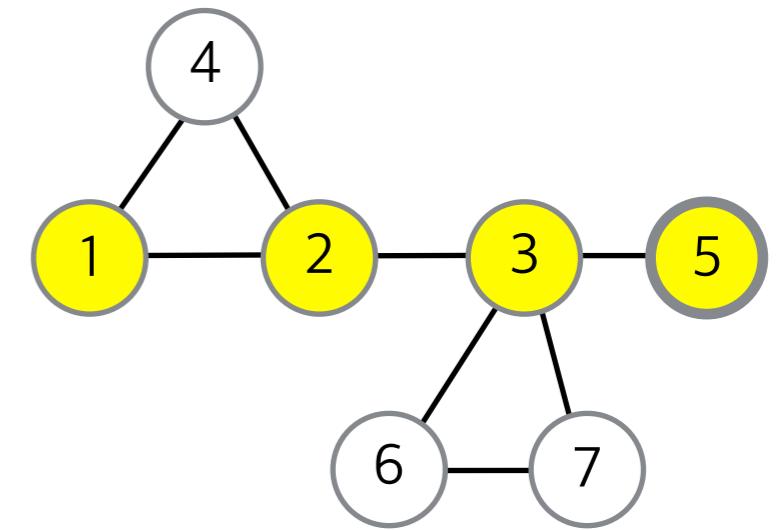
    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5

DFS(graph, 5, visited)

    result = [5], v = 3



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

→ DFS(graph, 1, visited)

result = [1], v = 2

DFS(graph, 2, visited)

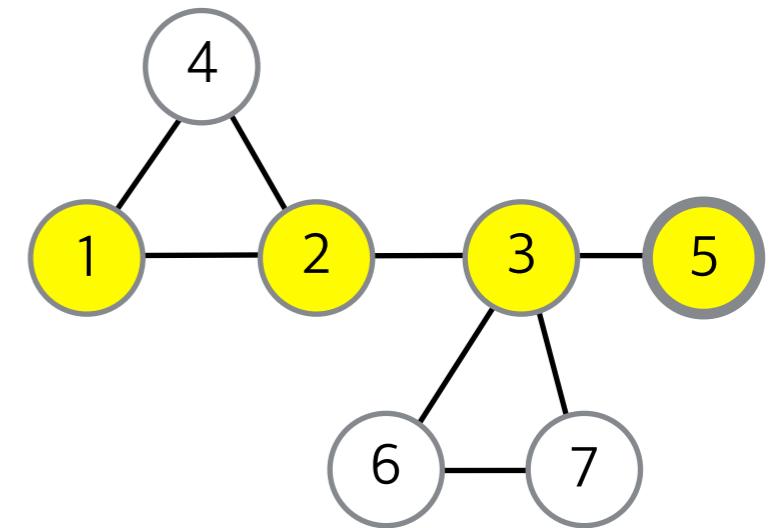
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

DFS(graph, 5, visited)

result = [5], v = 3



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

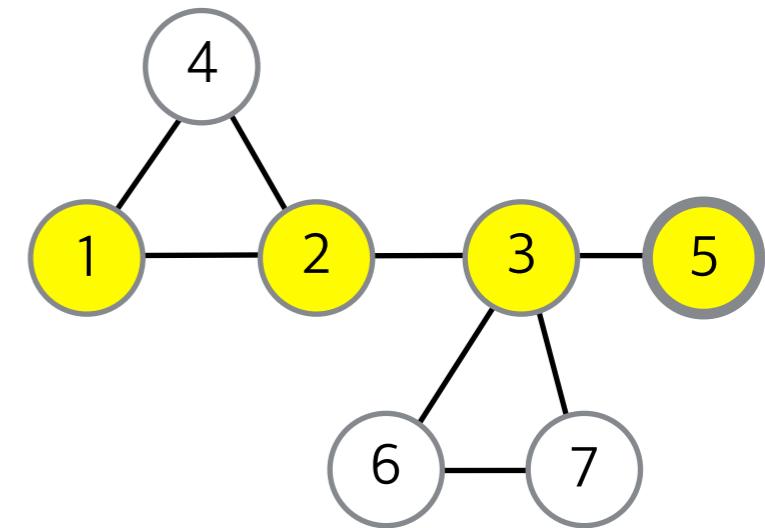
    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5

DFS(graph, 5, visited)

    result = [5], v = 3



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

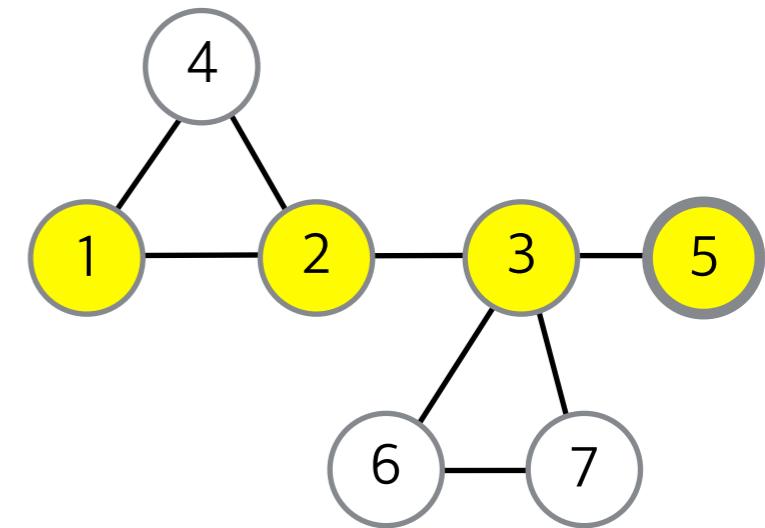
DFS(graph, 3, visited)

    result = [3], v = 5

DFS(graph, 5, visited)

    result = [5], v = 3

[5]



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

DFS(graph, 1, visited)

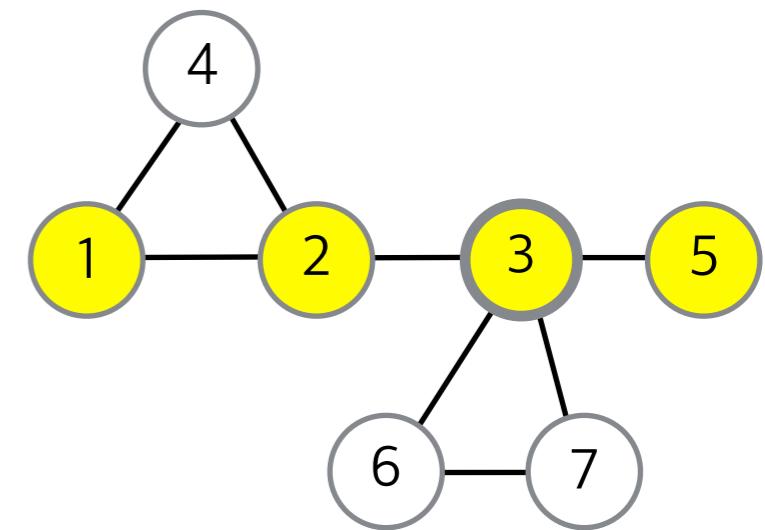
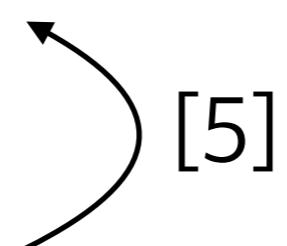
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3], v = 5



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

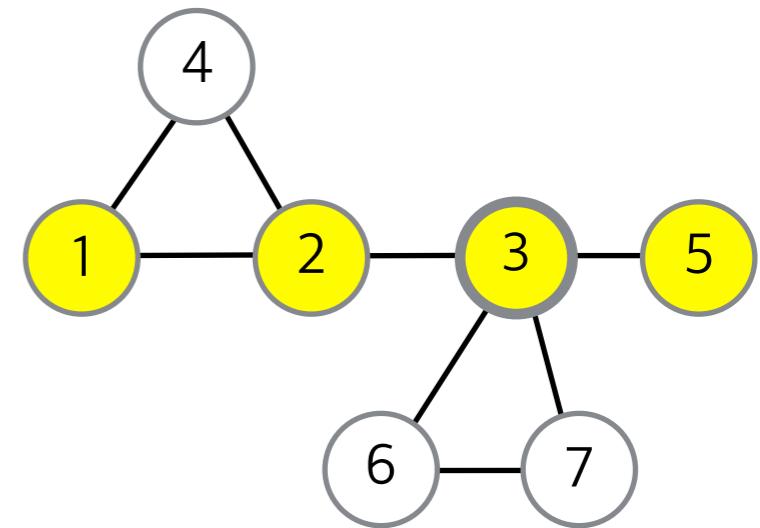
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5], v = 5



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

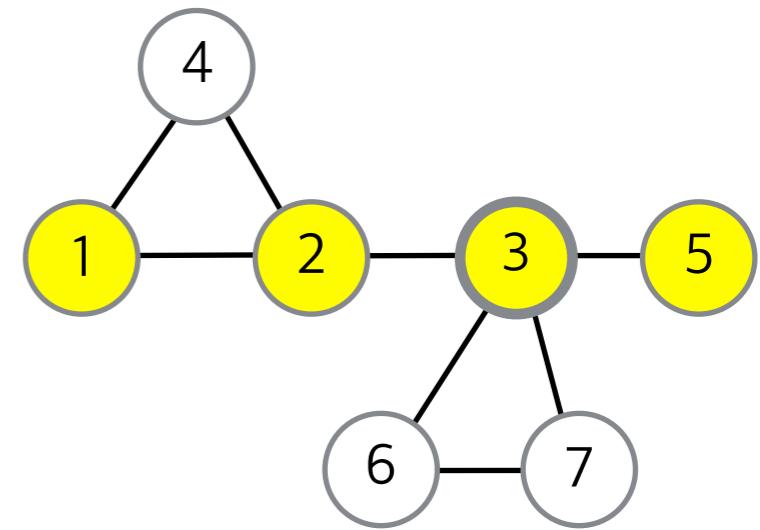
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5], v = 6



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

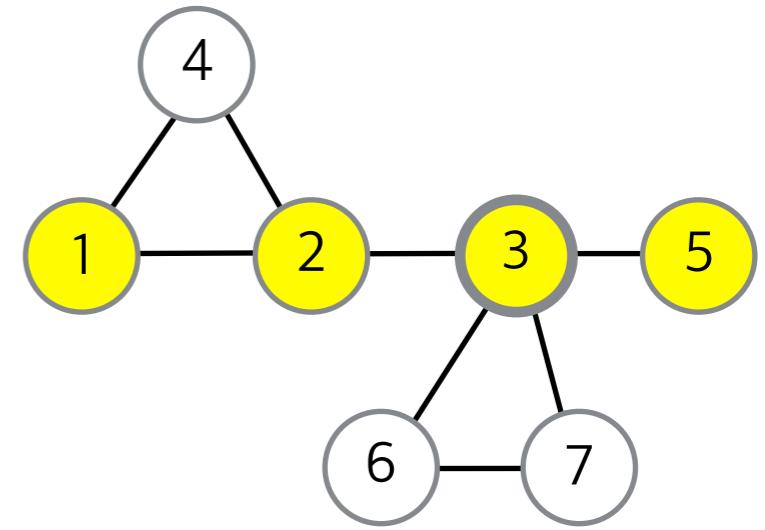
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5], v = 6



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

→

DFS(graph, 1, visited)

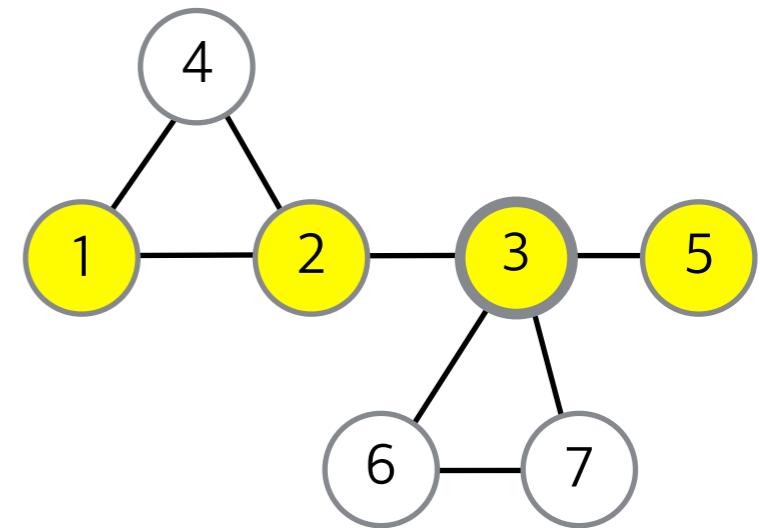
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5], v = 6



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

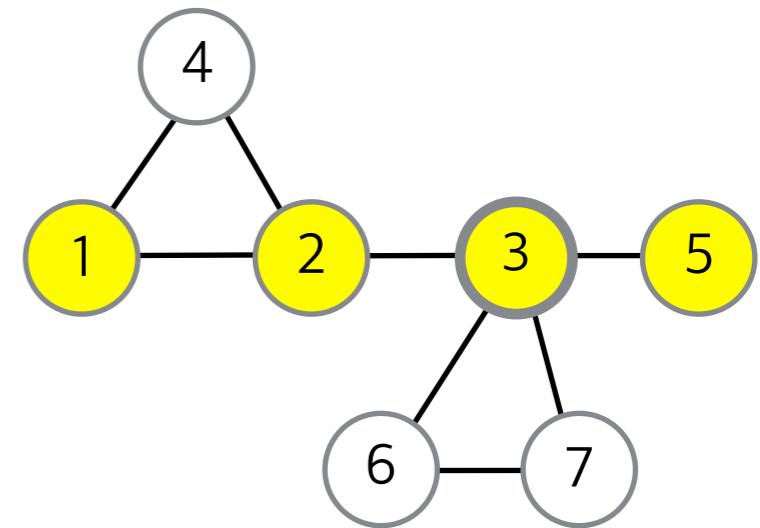
DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5], v = 6

DFS(graph, 6, visited)가 return하는 값은 ?



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

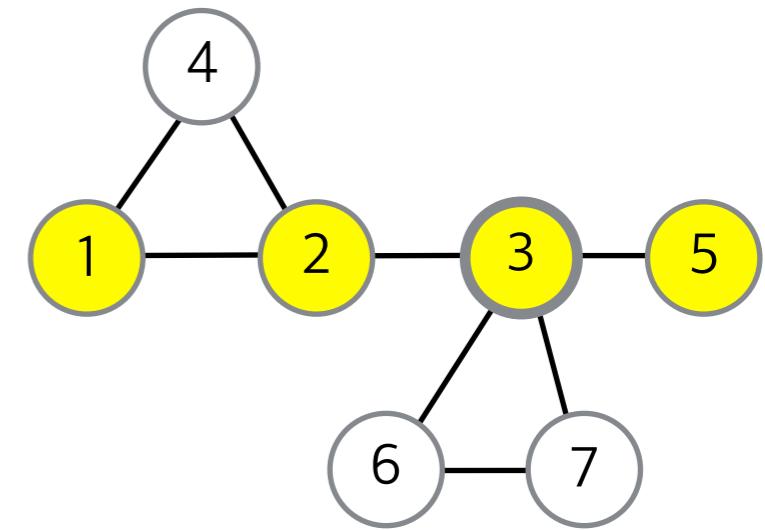
    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5], v = 6

DFS(graph, 6, visited)

[6, 7]



visited= [F, T, T, T, F, T, F, F]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```

def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result

```

DFS(graph, 1, visited)

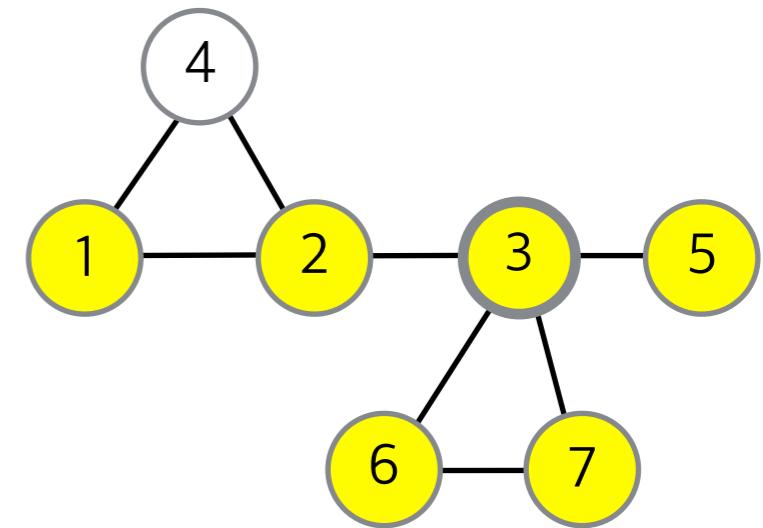
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5, 6, 7], v = 6



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

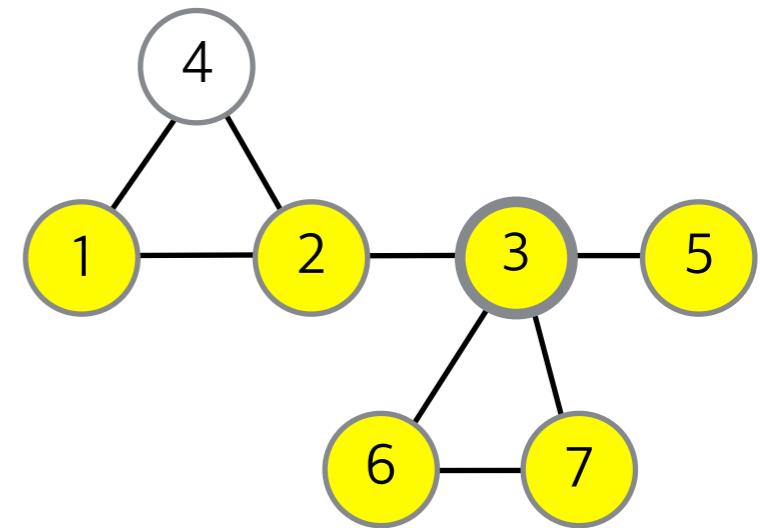
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5, 6, 7], v = 7



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

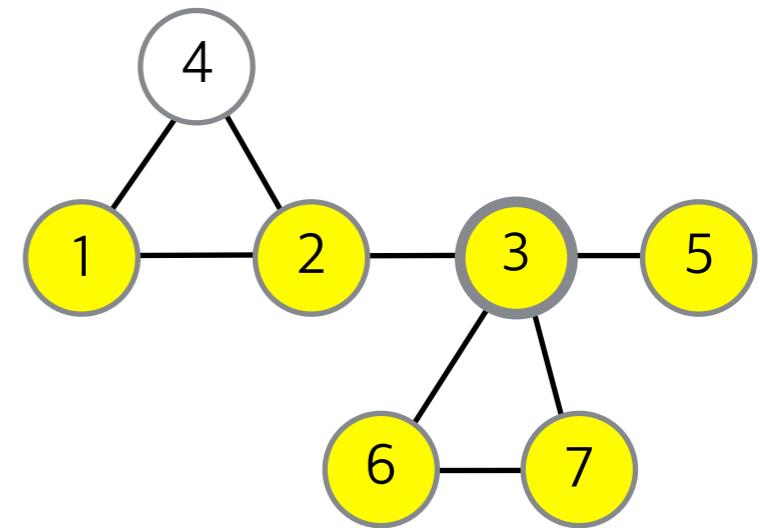
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5, 6, 7], v = 7



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

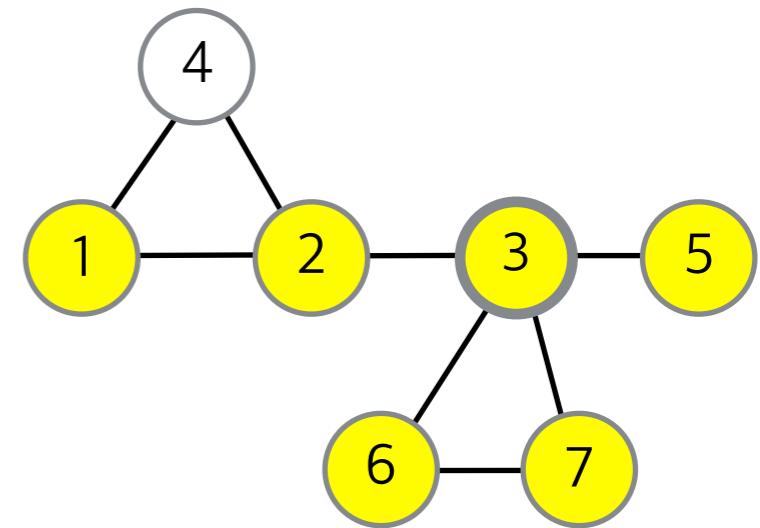
    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5, 6, 7], v = 7



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

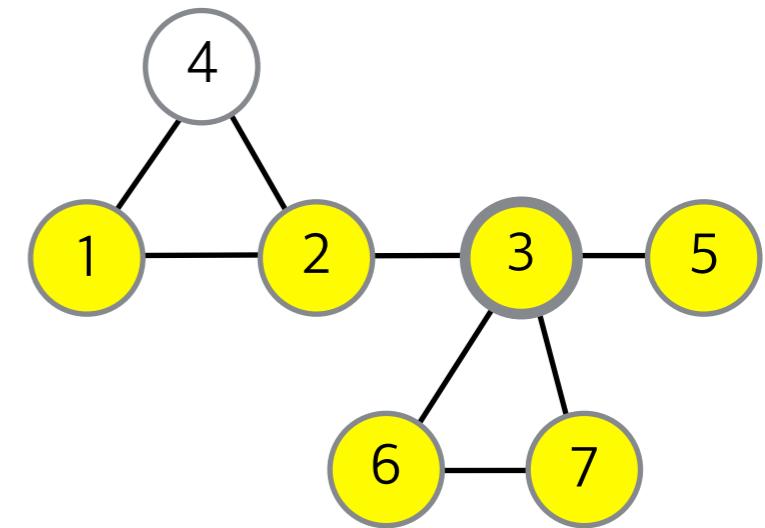
DFS(graph, 2, visited)

    result = [2], v = 3

DFS(graph, 3, visited)

    result = [3, 5, 6, 7], v = 7

[3, 5, 6, 7]



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

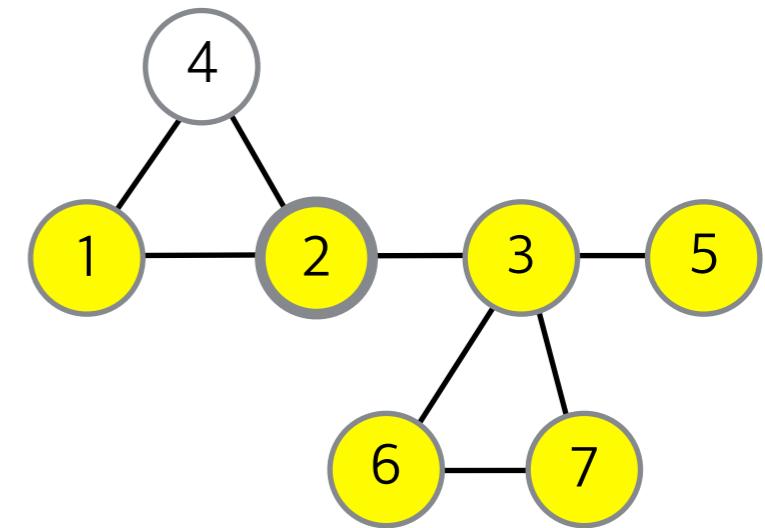
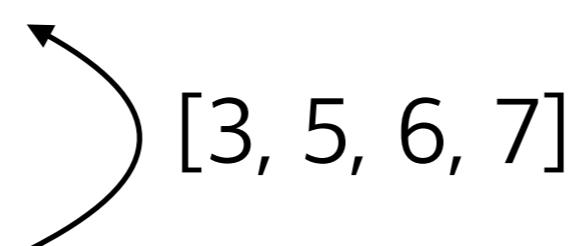
    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2], v = 3



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

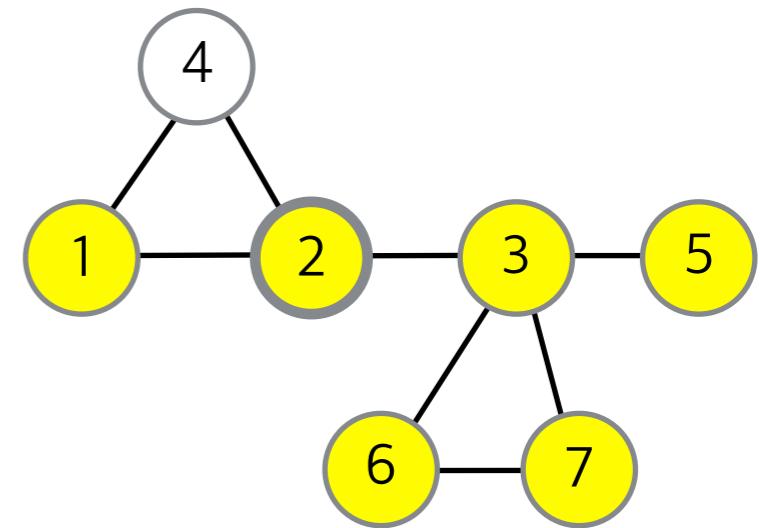
    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2, 3, 5, 6, 7], v = 3



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

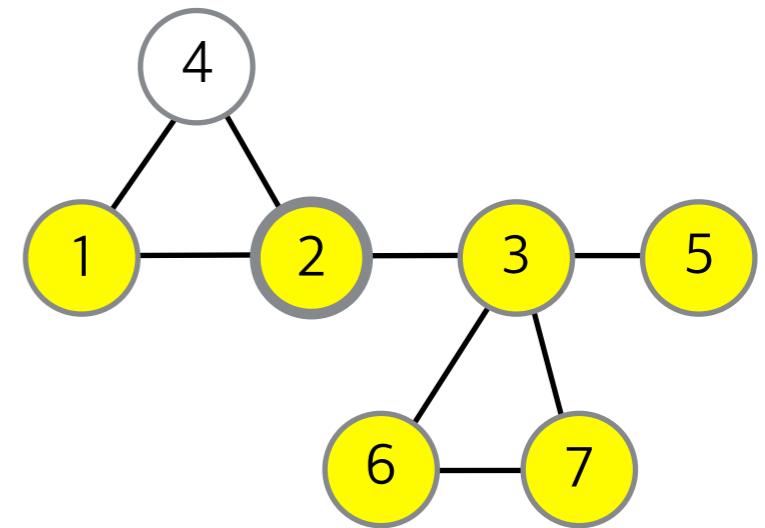
    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2, 3, 5, 6, 7], v = 4



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

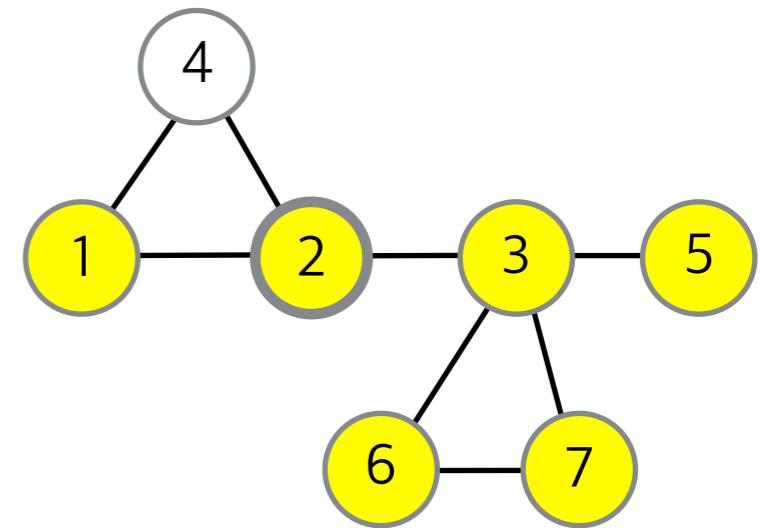
→

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2, 3, 5, 6, 7], v = 4



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

→

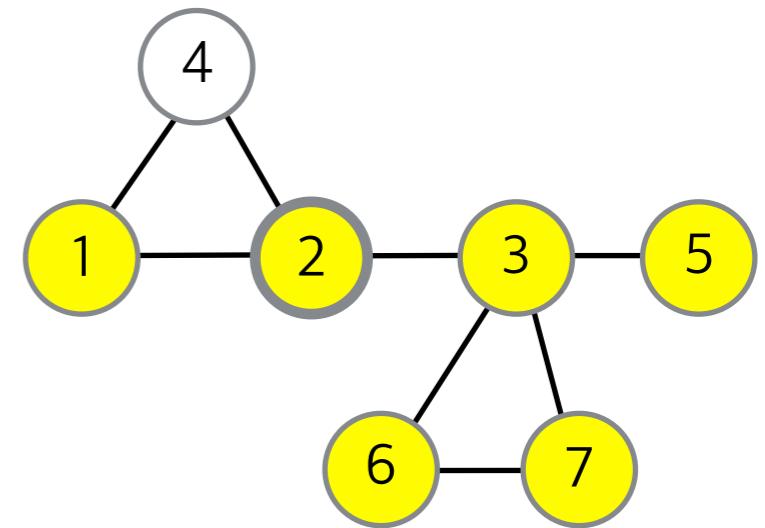
DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2, 3, 5, 6, 7], v = 4

DFS(graph, 4, visited)가 return하는 값은 ?



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

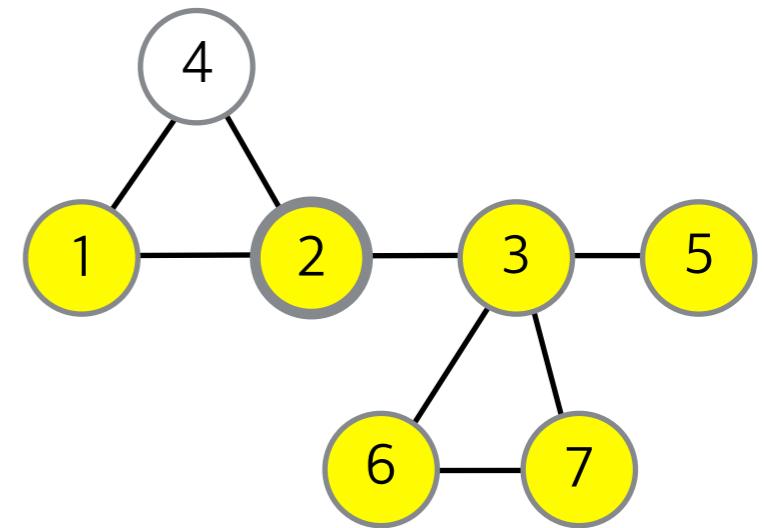
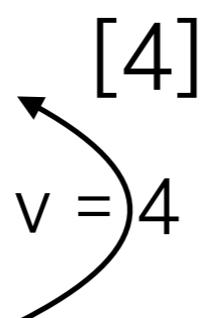
    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)
    return result
```

DFS(graph, 1, visited)

result = [1], v = 2

DFS(graph, 2, visited)

result = [2, 3, 5, 6, 7], v = 4



visited= [F, T, T, T, F, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

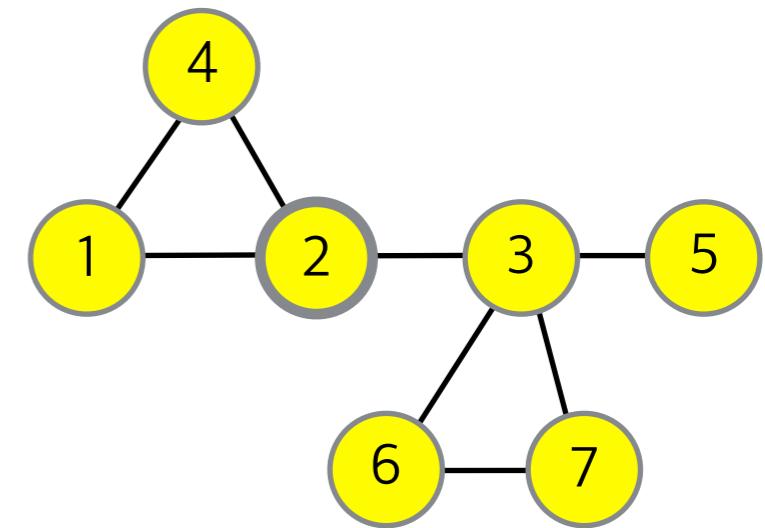
    return result
```

DFS(graph, 1, visited)

    result = [1], v = 2

DFS(graph, 2, visited)

    result = [2, 3, 5, 6, 7, 4], v = 4



visited= [F, T, T, T, **T**, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

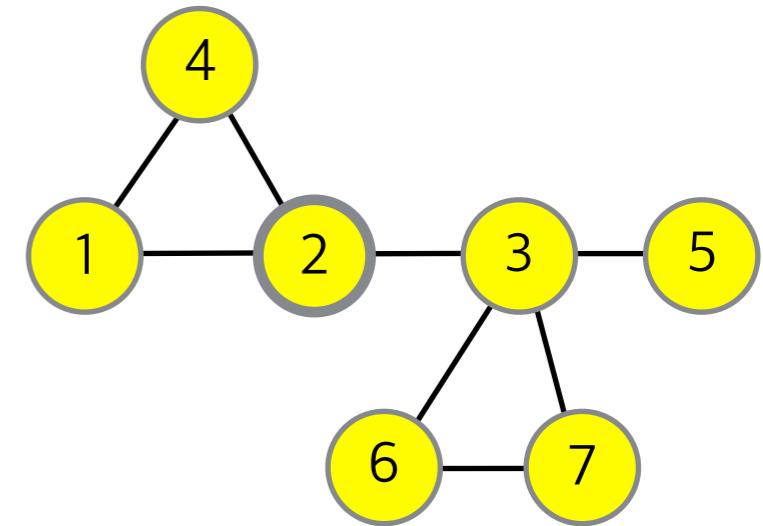
    return result
```

DFS(graph, 1, visited)

result = [1], v = 2

DFS(graph, 2, visited)

result = [2, 3, 5, 6, 7, 4], v = 4



visited= [F, T, T, T, T, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

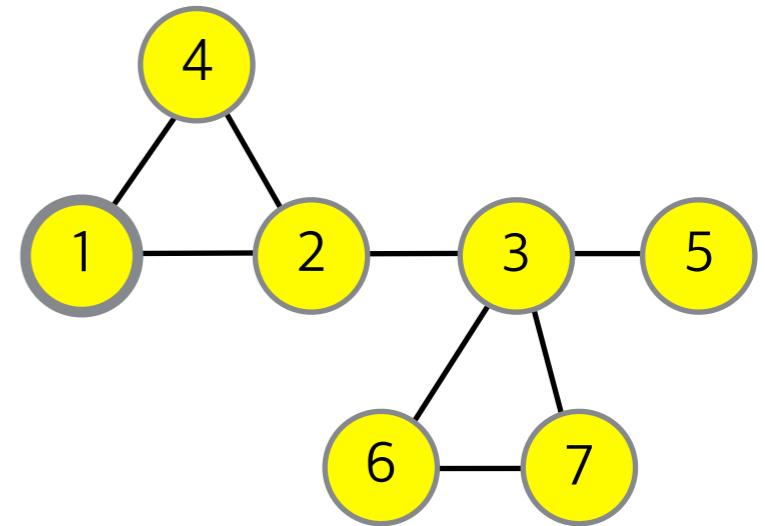
    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

result = [1], v = 2

[2, 3, 5, 6, 7, 4]



visited= [F, T, T, T, T, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

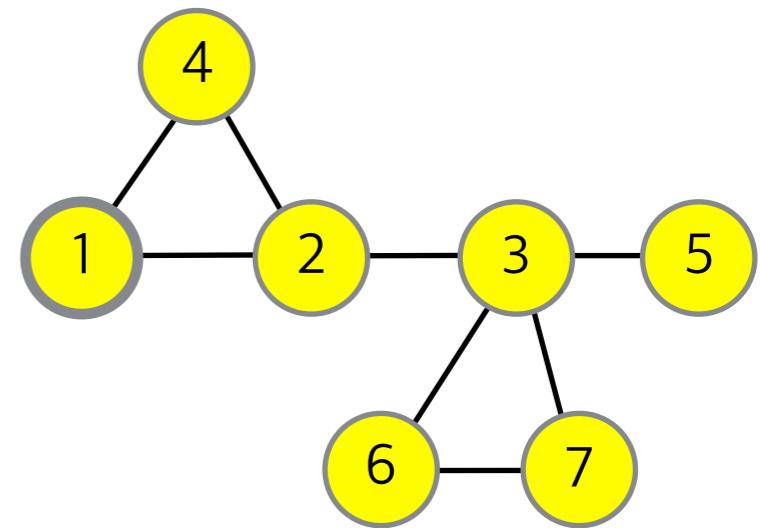
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

result = [1, 2, 3, 5, 6, 7, 4], v = 2



visited= [F, T, T, T, T, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

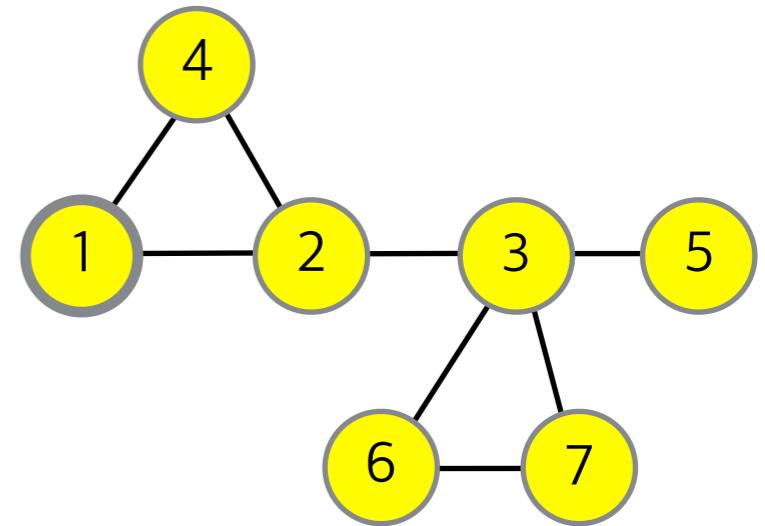
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

result = [1, 2, 3, 5, 6, 7, 4], v = 4



visited= [F, T, T, T, T, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

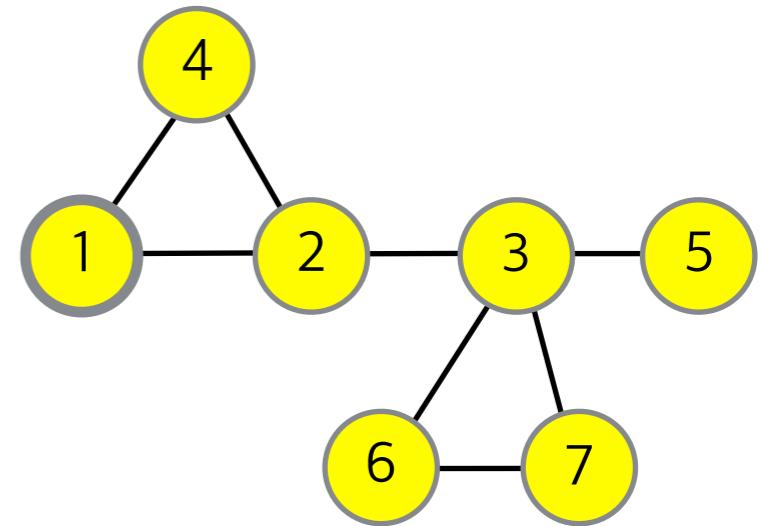
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    → for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

result = [1, 2, 3, 5, 6, 7, 4], v = 4



visited= [F, T, T, T, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



# 깊이 우선 탐색 (DFS) 구현

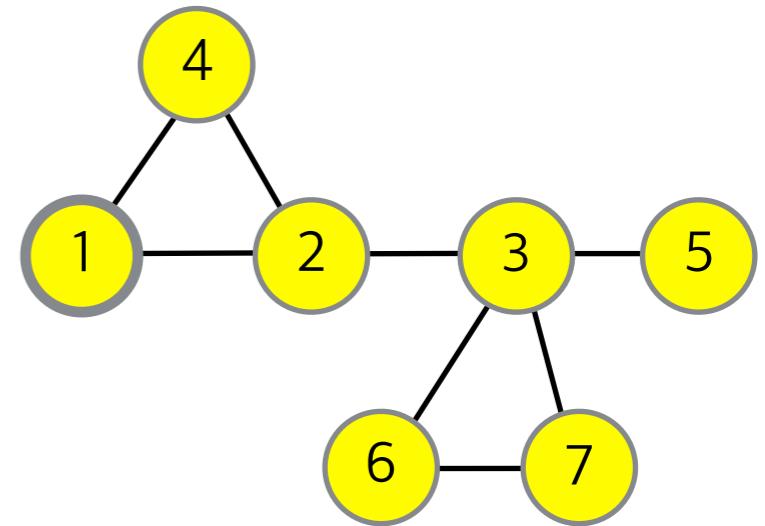
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False :
            result = result +
                DFS(graph, v, visited)

    return result
```

DFS(graph, 1, visited)

result = [1, 2, 3, 5, 6, 7, 4], v = 4



visited= [F, T, T, T, T, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

graph[6] = [3, 7]

graph[7] = [3, 6]



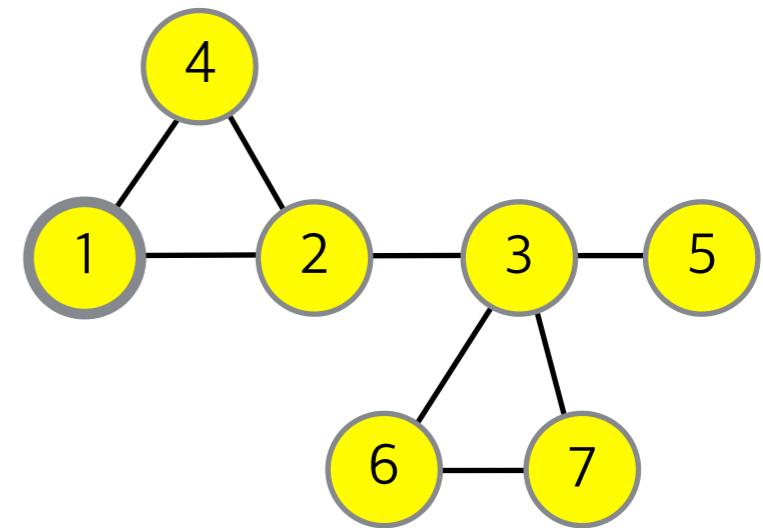
# 깊이 우선 탐색 (DFS) 구현

```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    for v in graph[x] :
        if visited[v] == False
            result += DFS(graph, v, visited)
    return result
```

DFS(graph, 1, visited)

result = [1, 2, 3, 5, 6, 7, 4], v = 4



visited= [F, T, T, T, T, T, T, T]

graph[1] = [2, 4]

graph[2] = [3, 4]

graph[3] = [2, 5, 6, 7]

graph[4] = [1, 2]

graph[5] = [3]

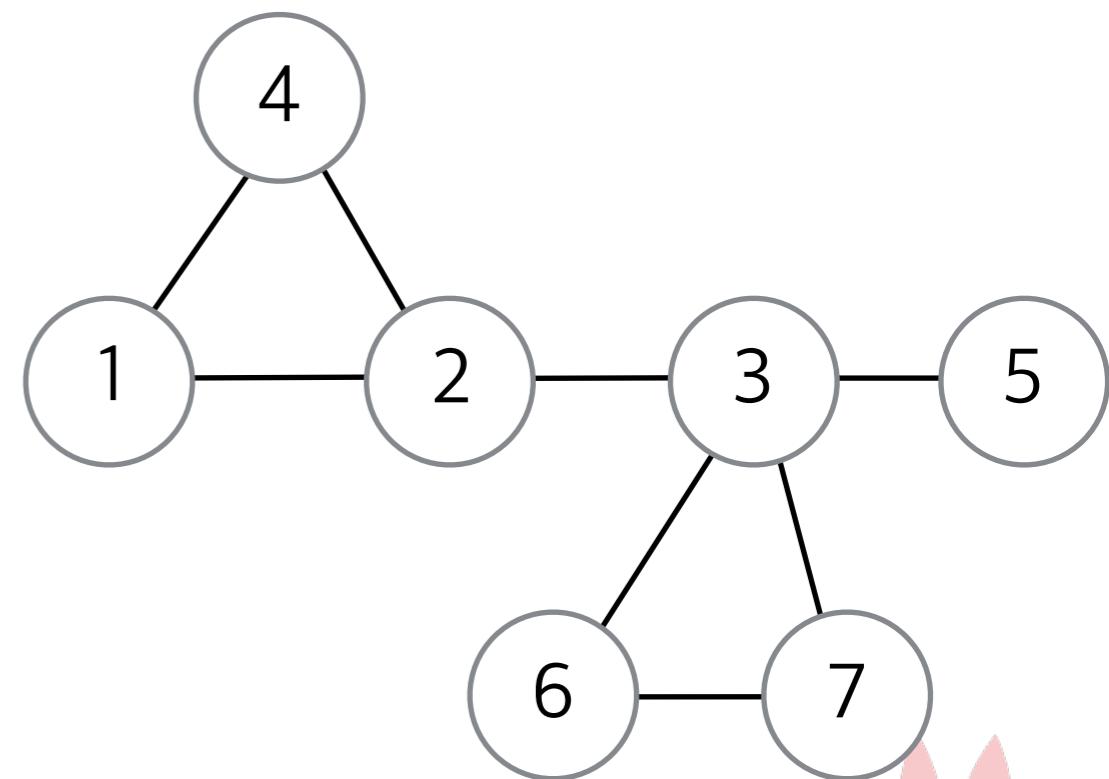
graph[6] = [3, 7]

graph[7] = [3, 6]



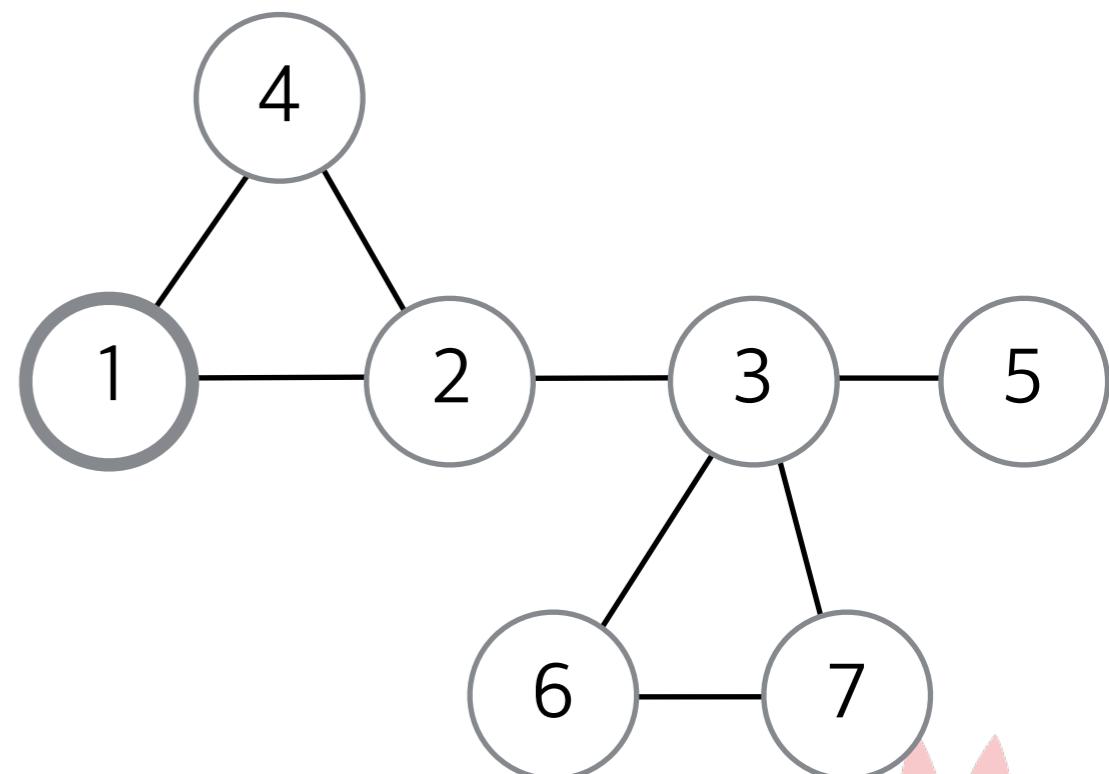
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



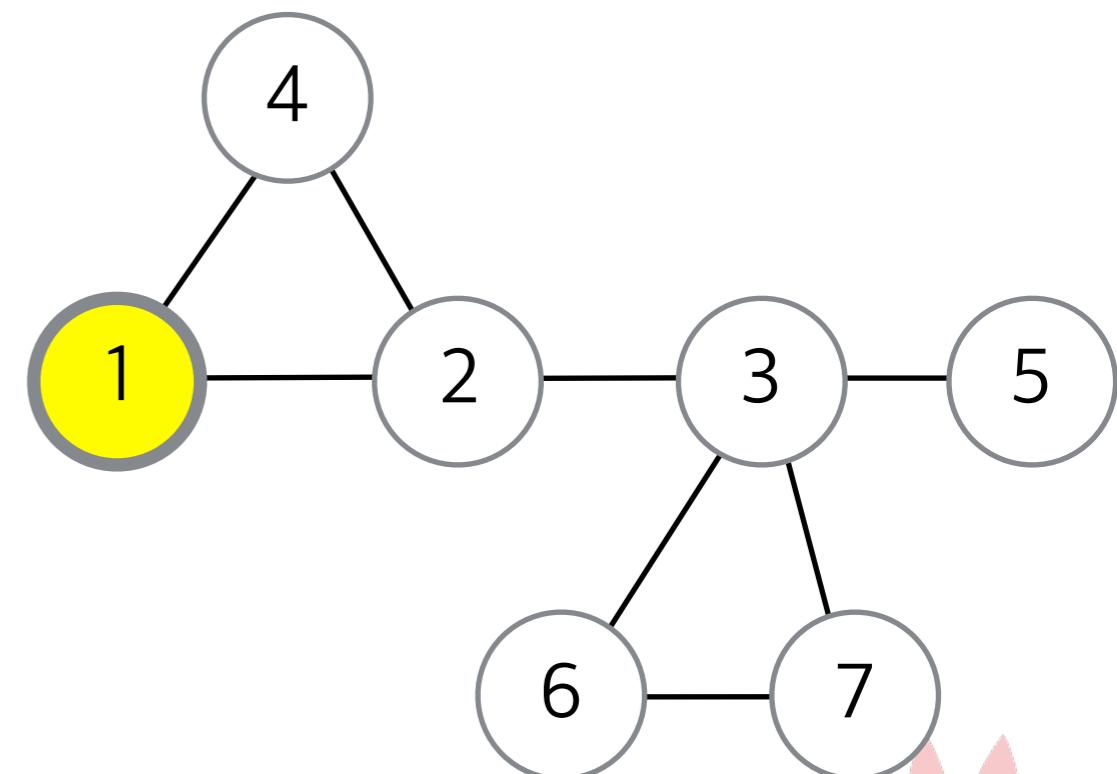
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



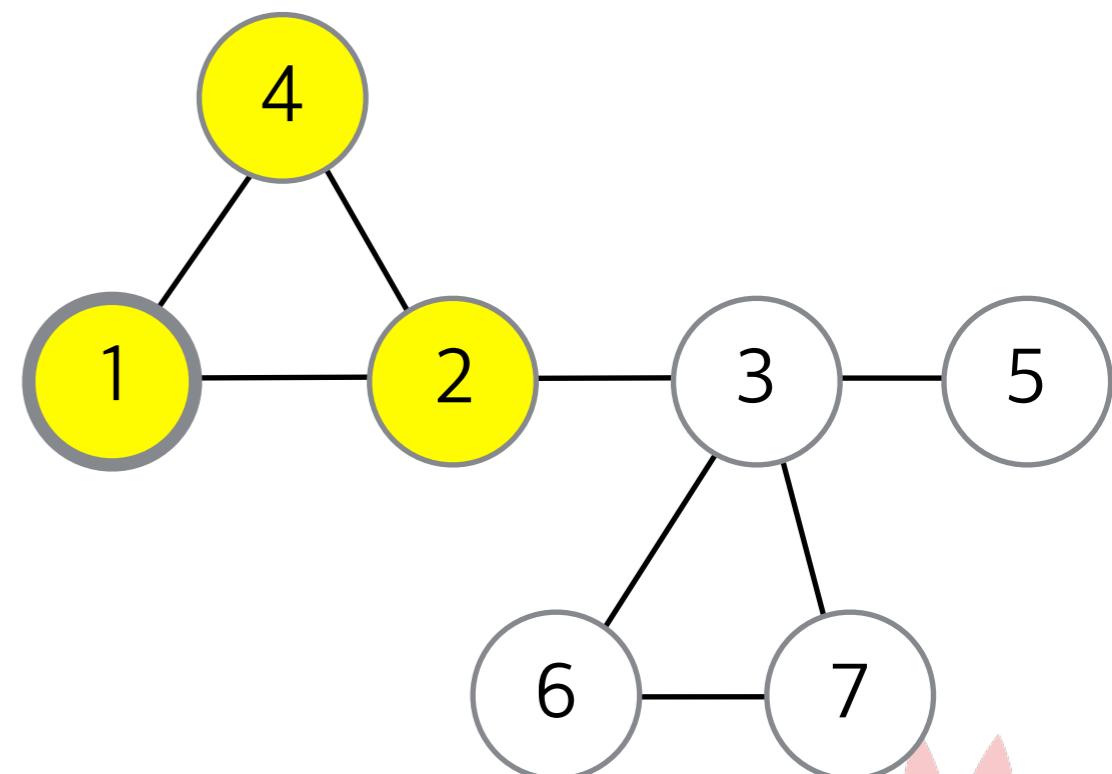
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



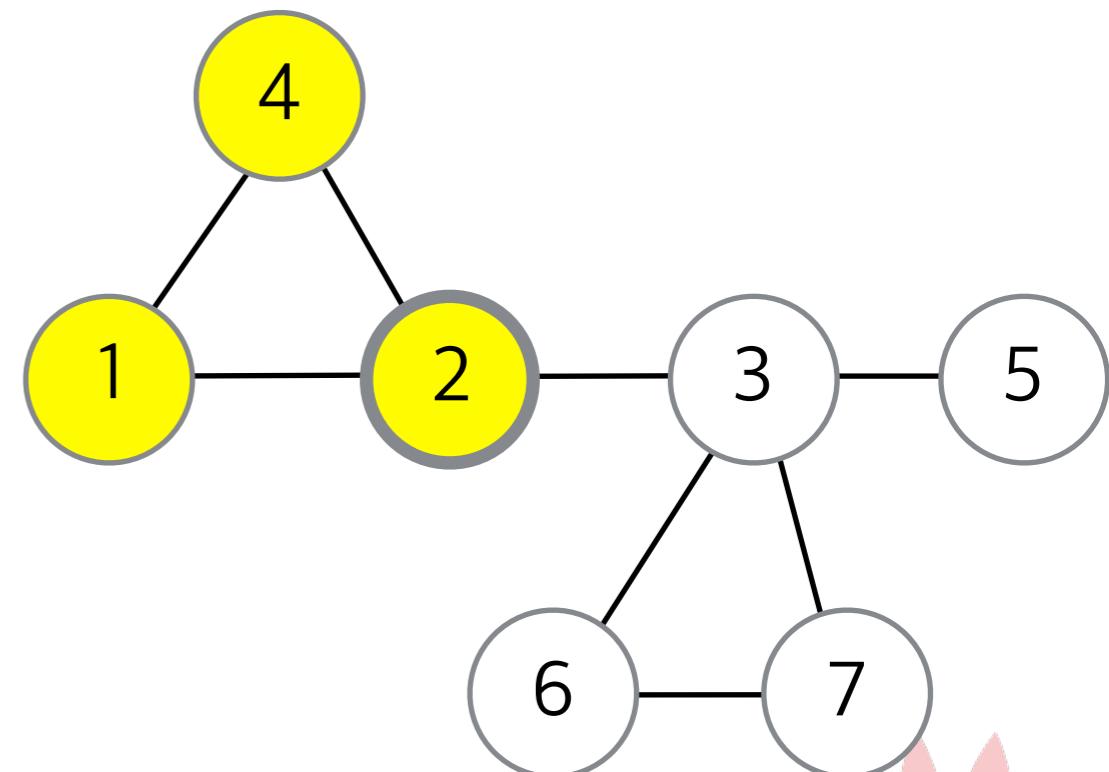
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



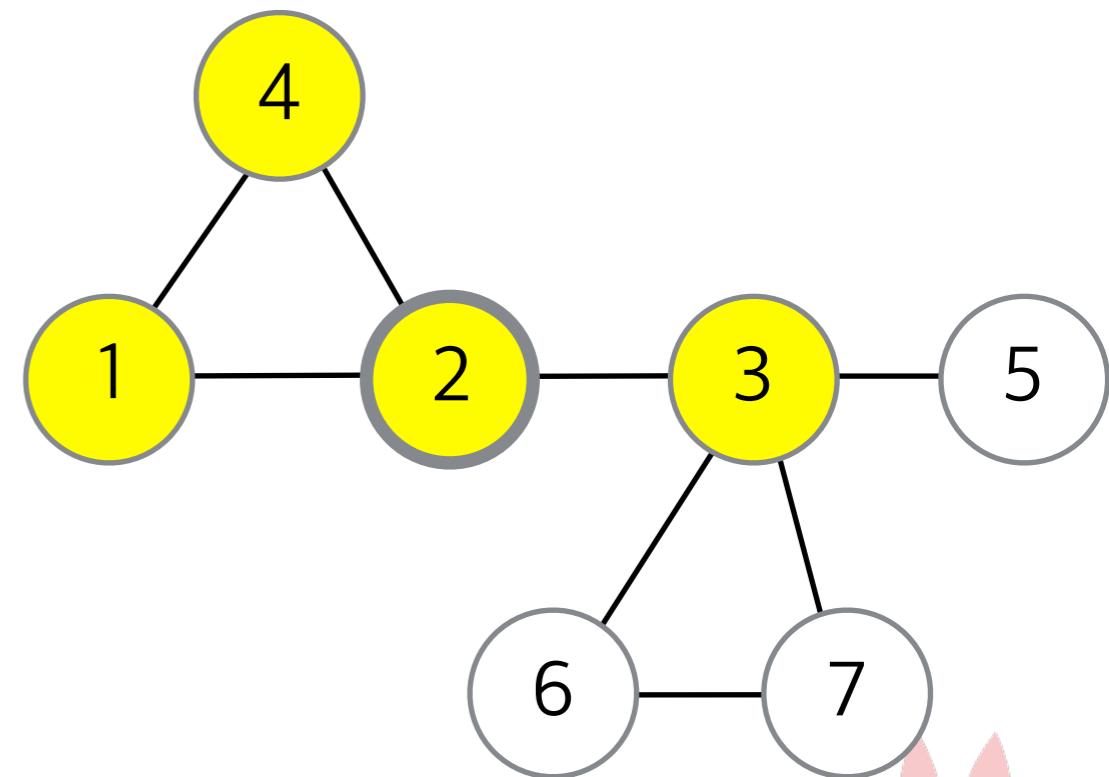
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



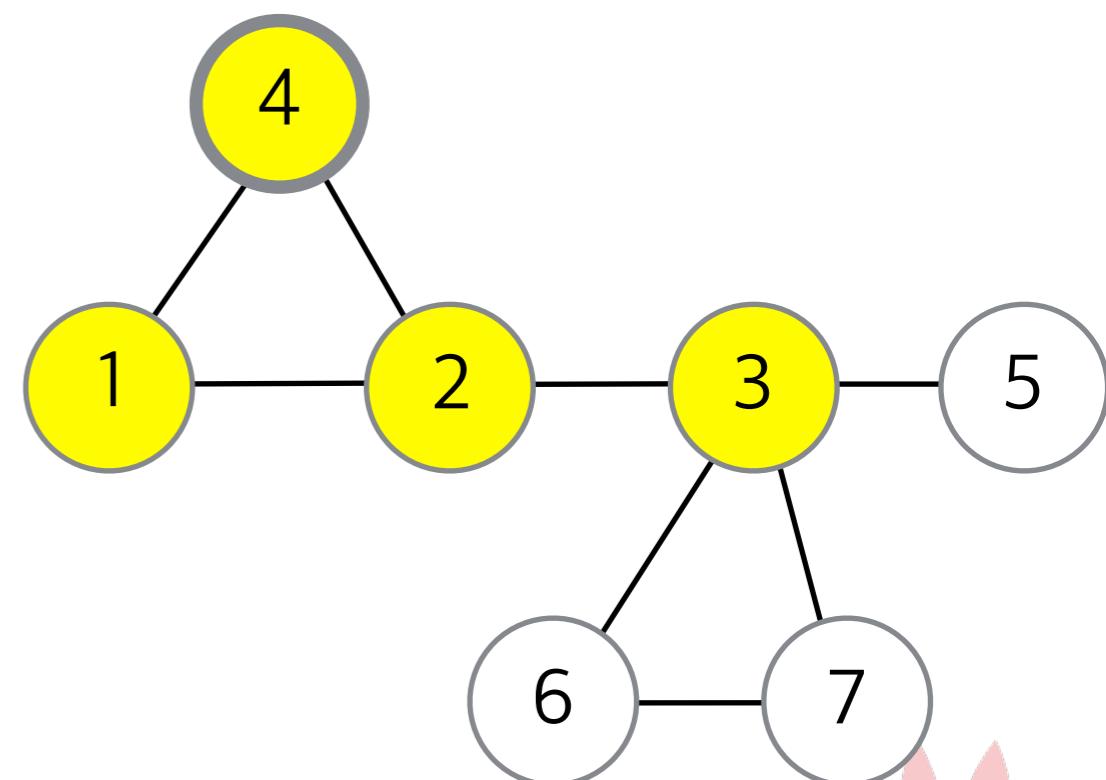
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



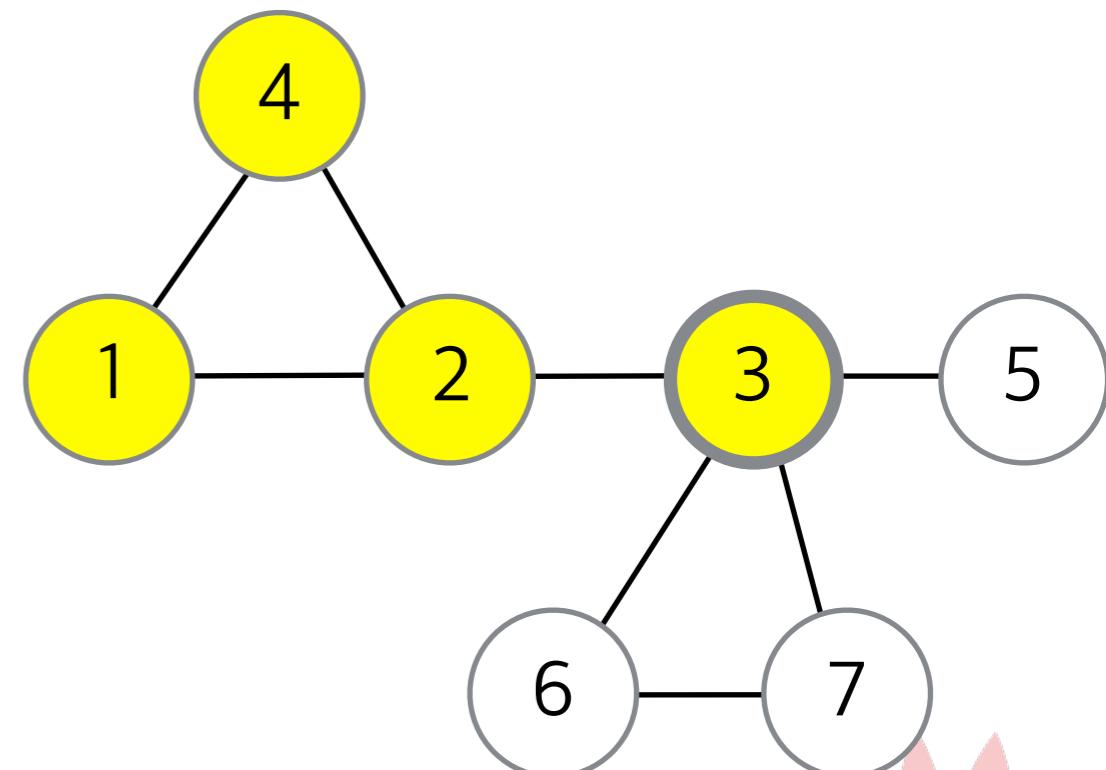
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



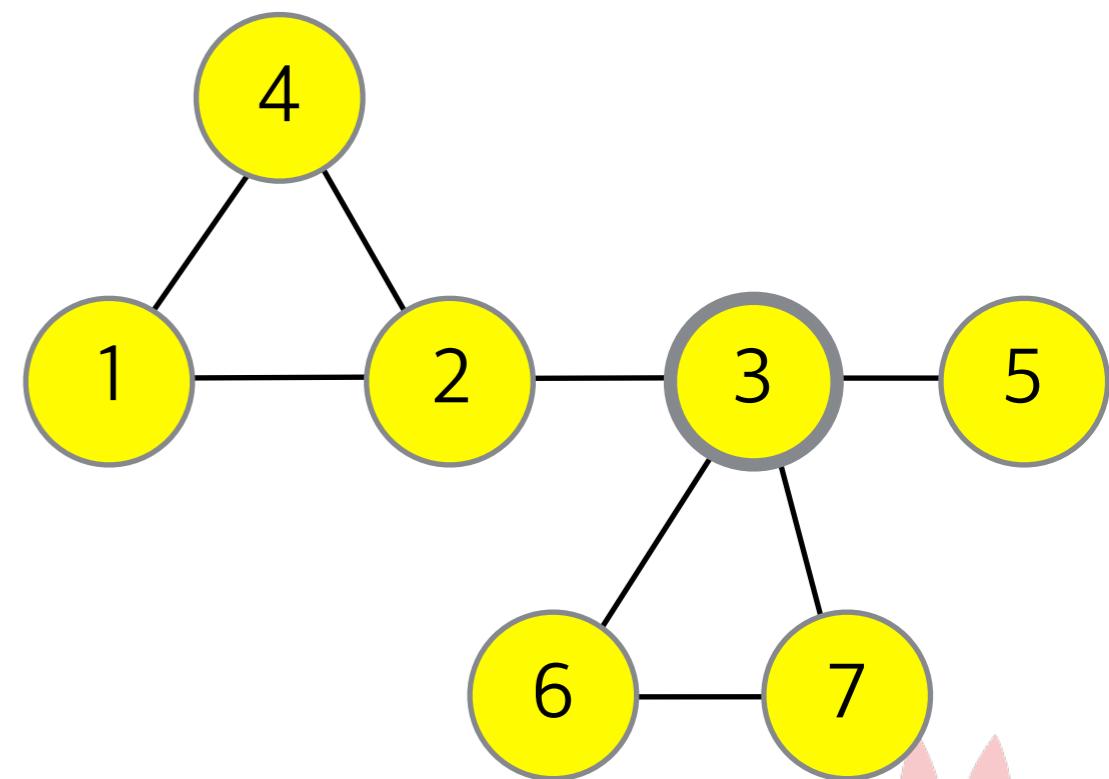
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



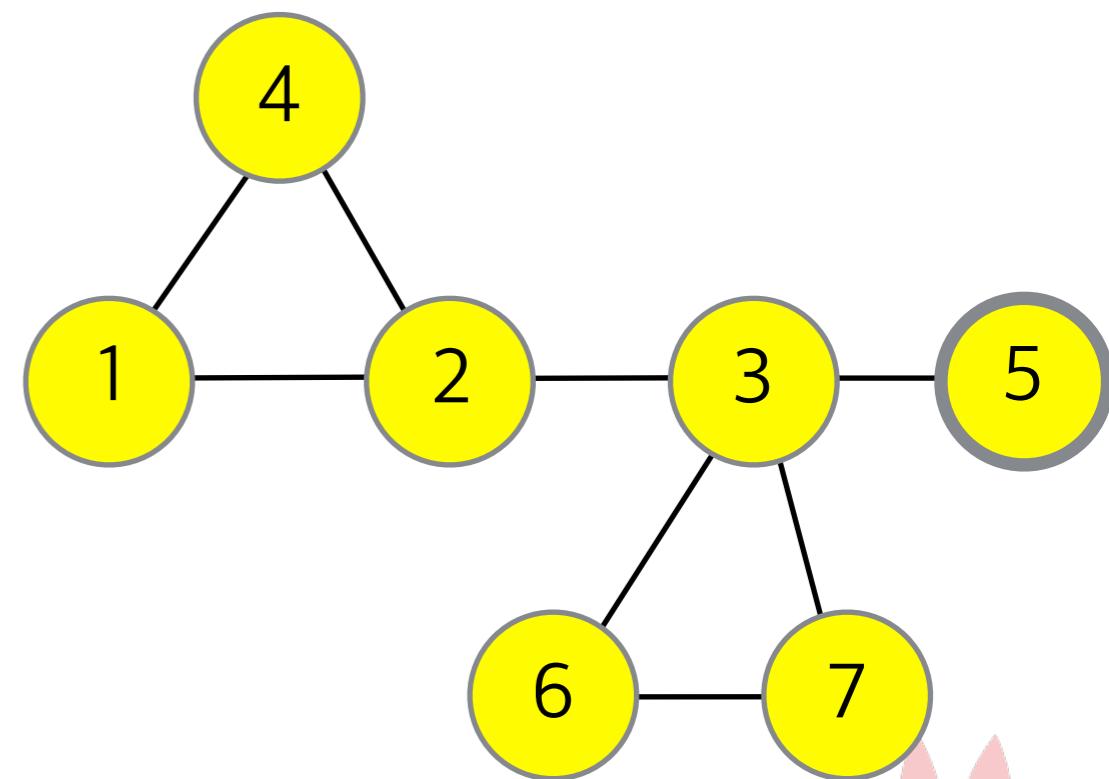
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



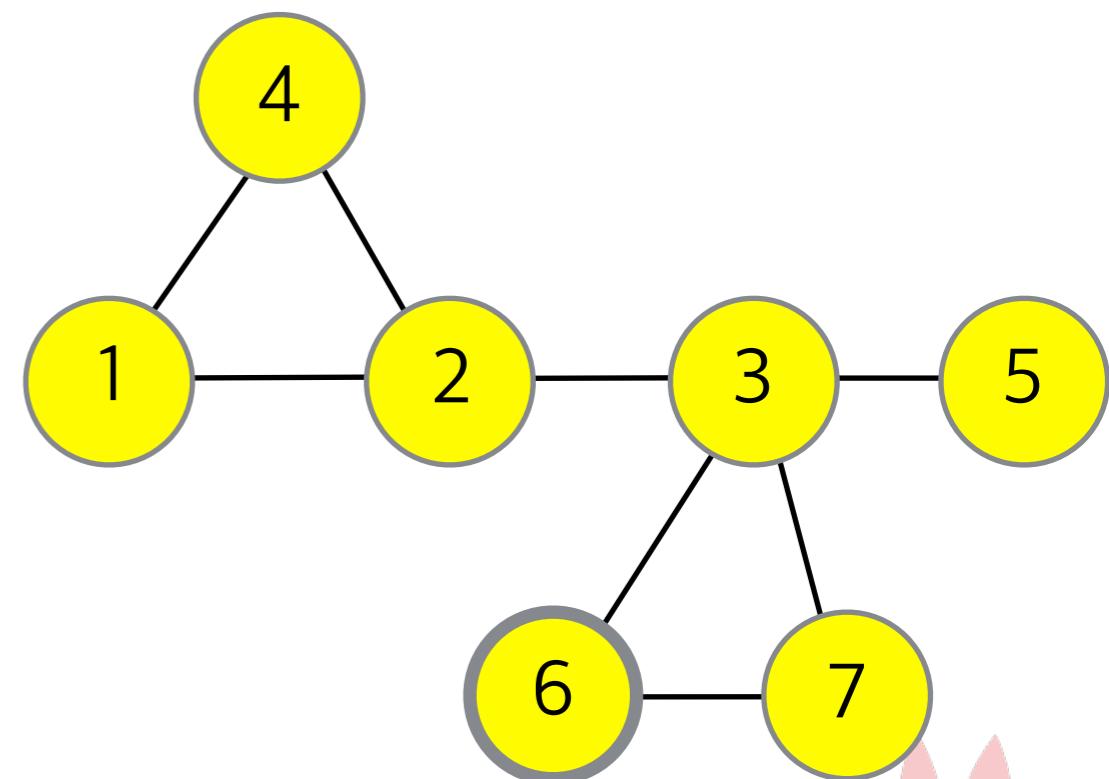
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



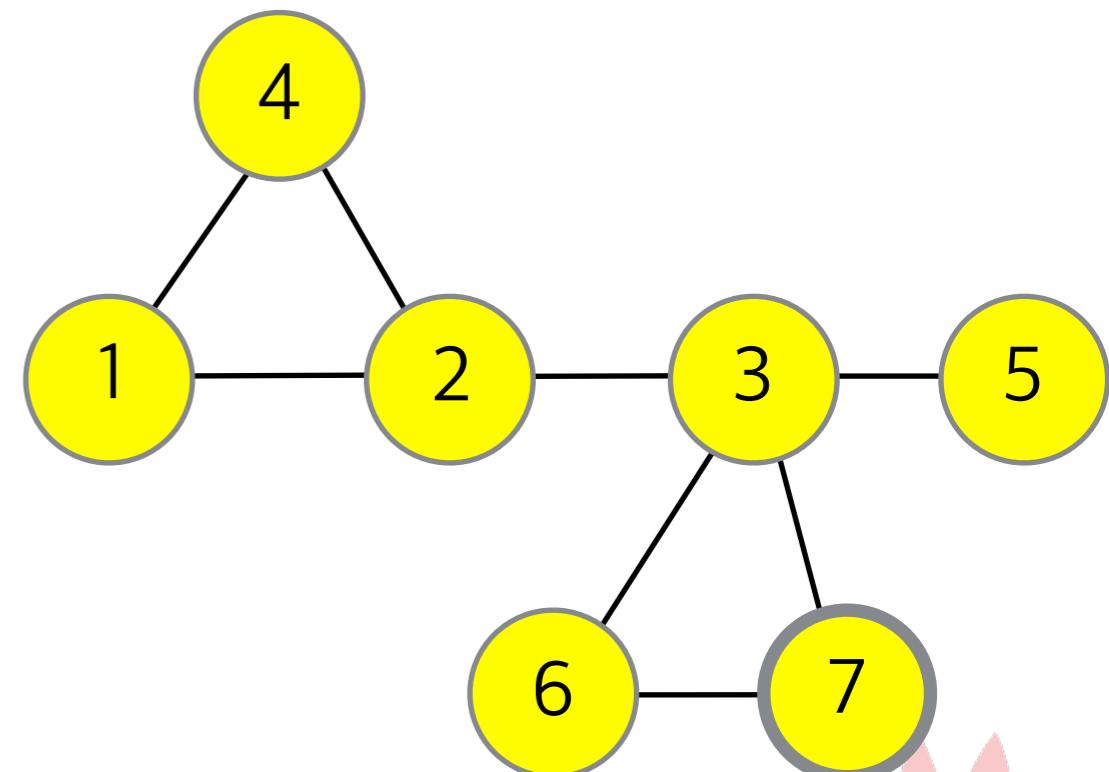
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



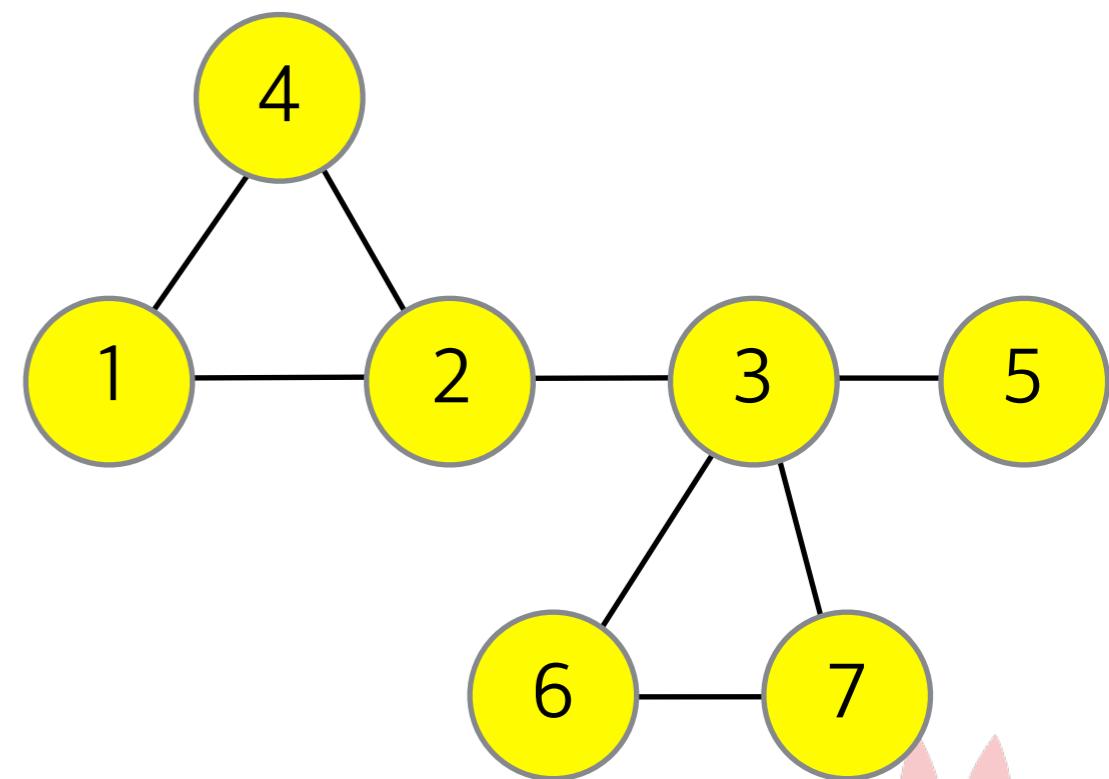
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



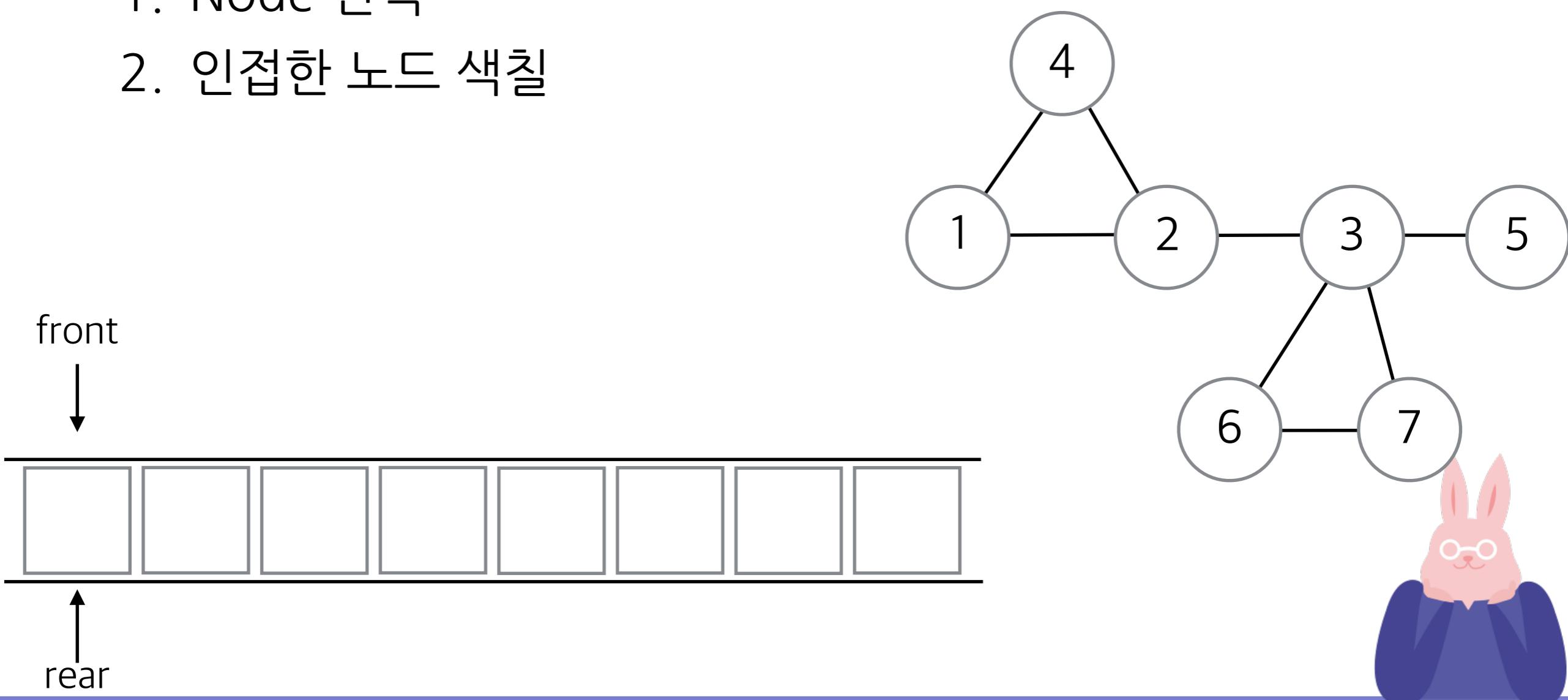
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다



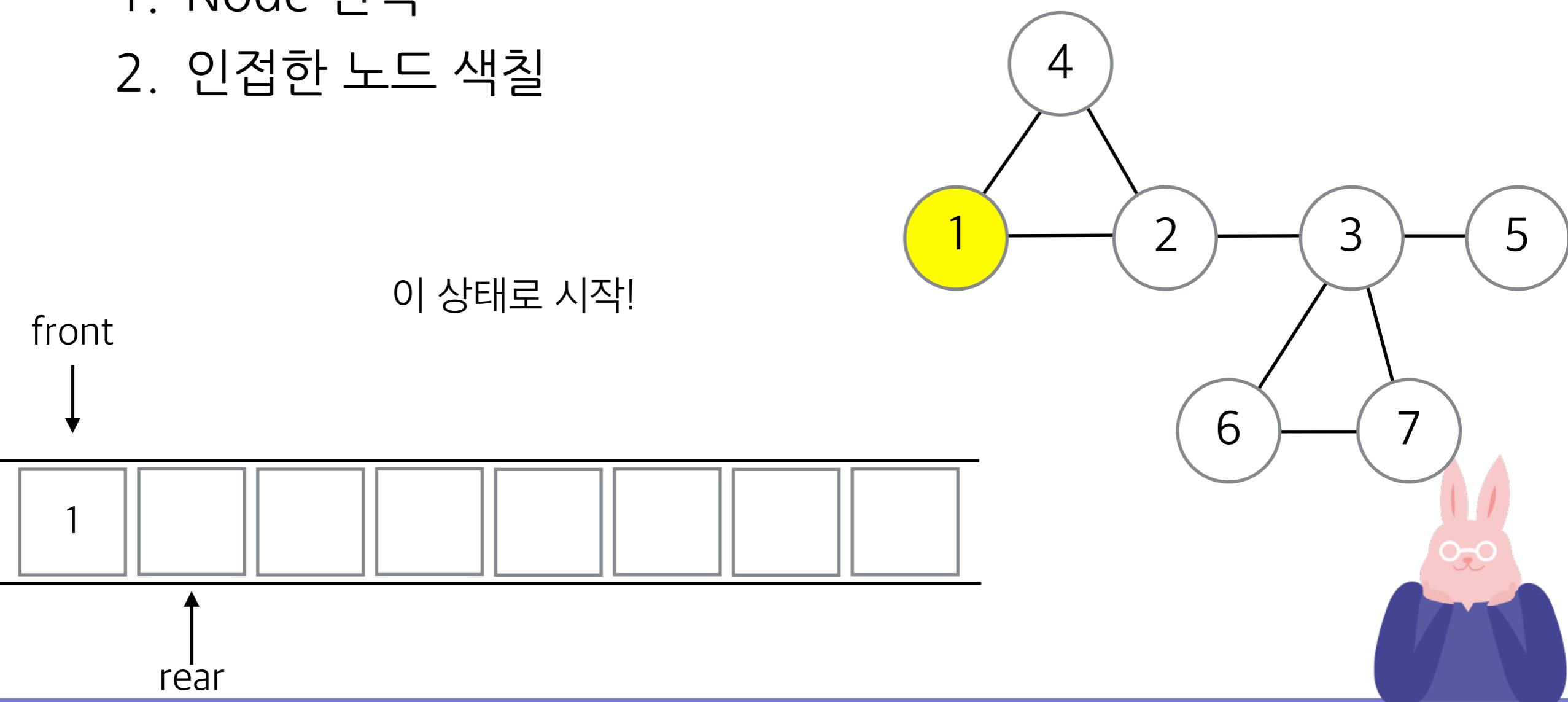
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠



# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠

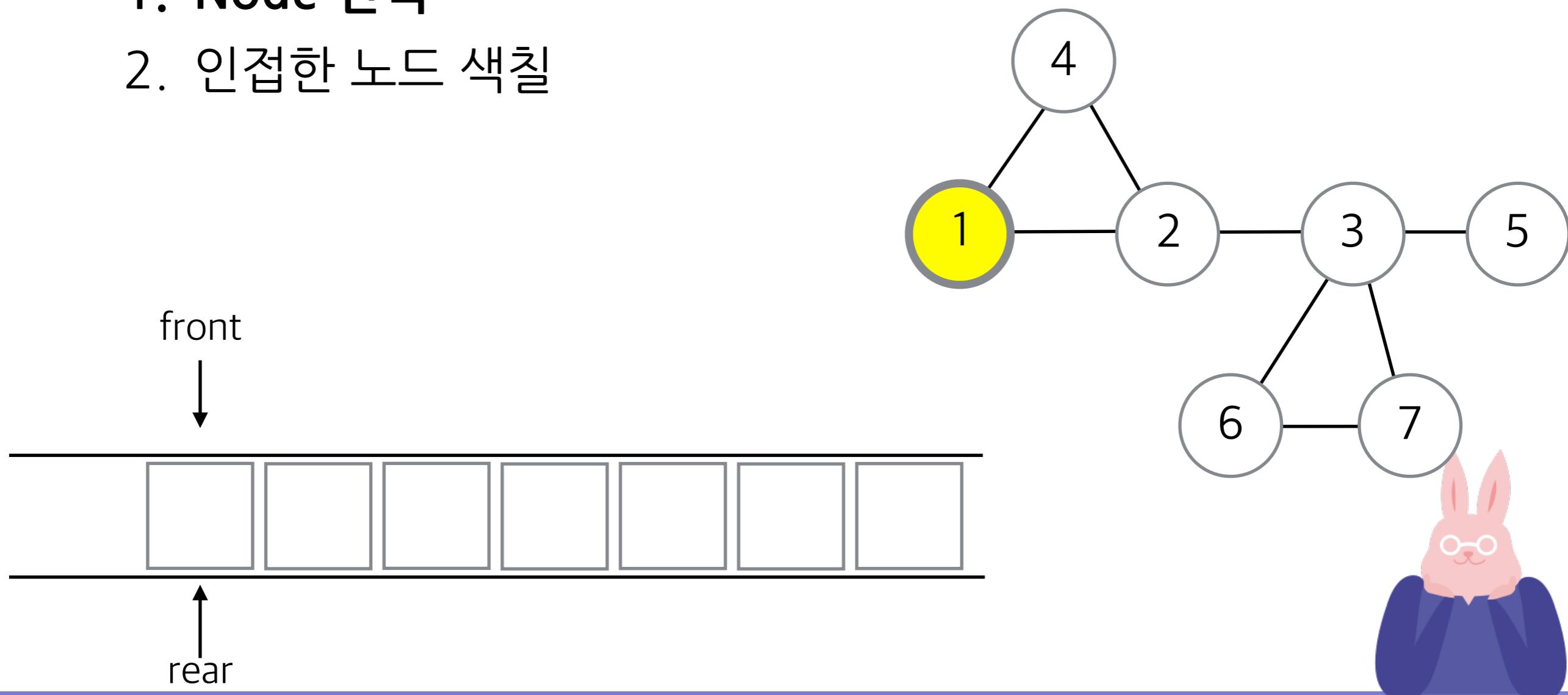


# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다

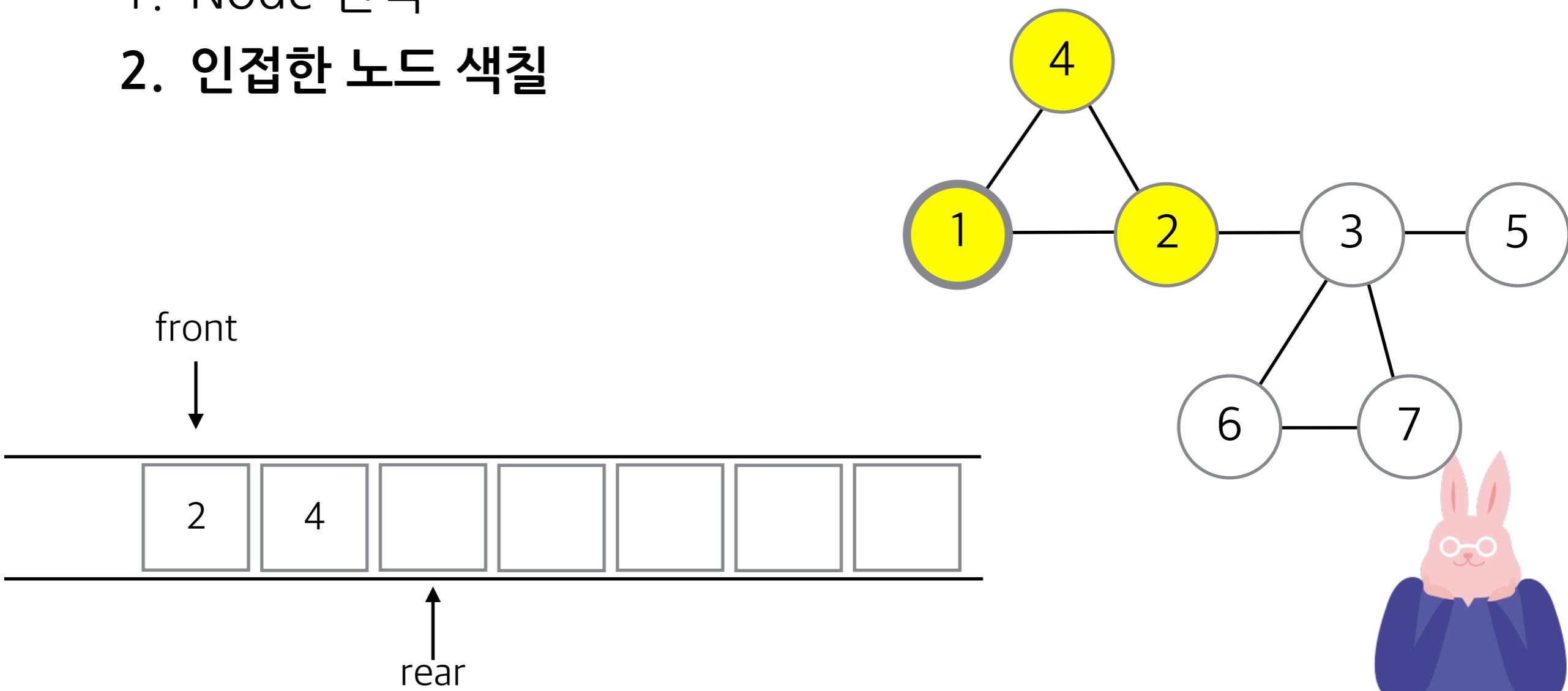
1. Node 선택

2. 인접한 노드 색칠



# 너비 우선 탐색 (BFS)

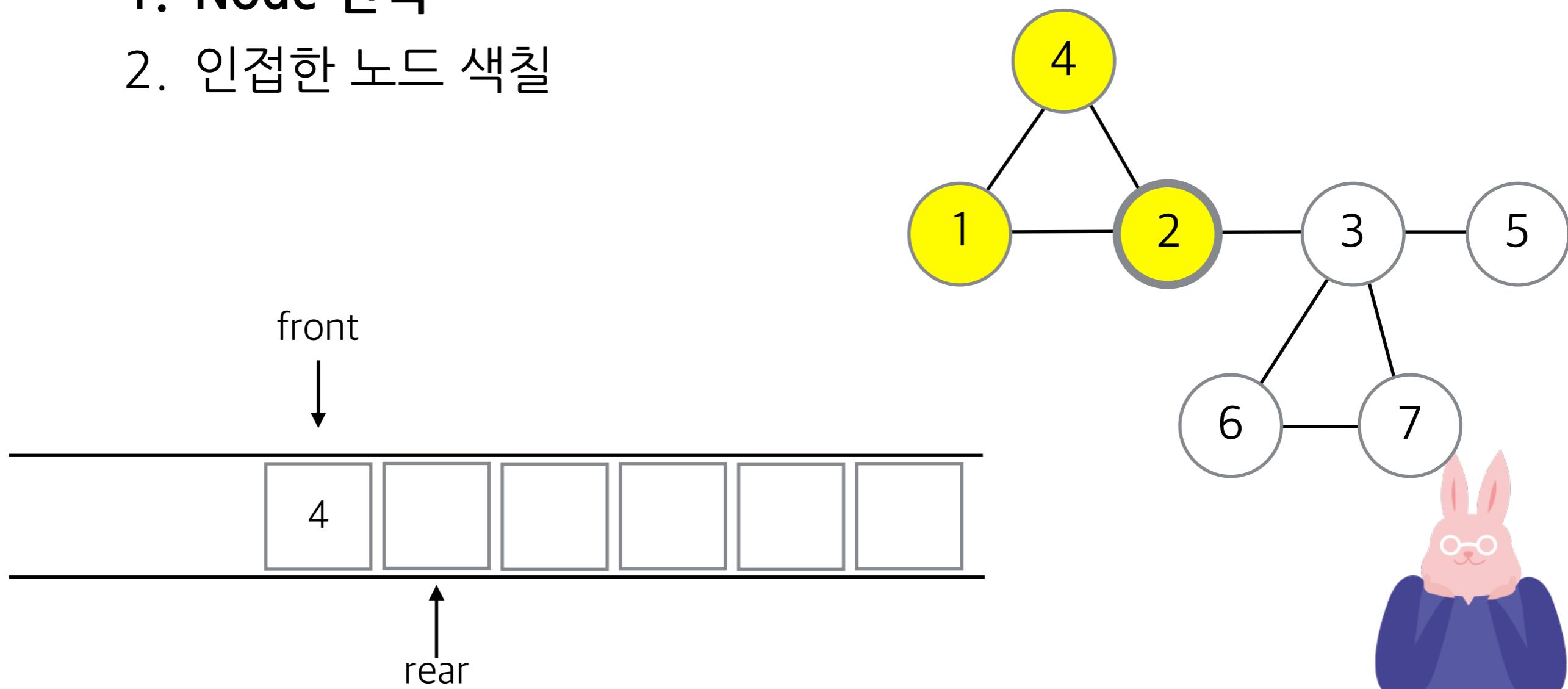
- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠



# 너비 우선 탐색 (BFS)

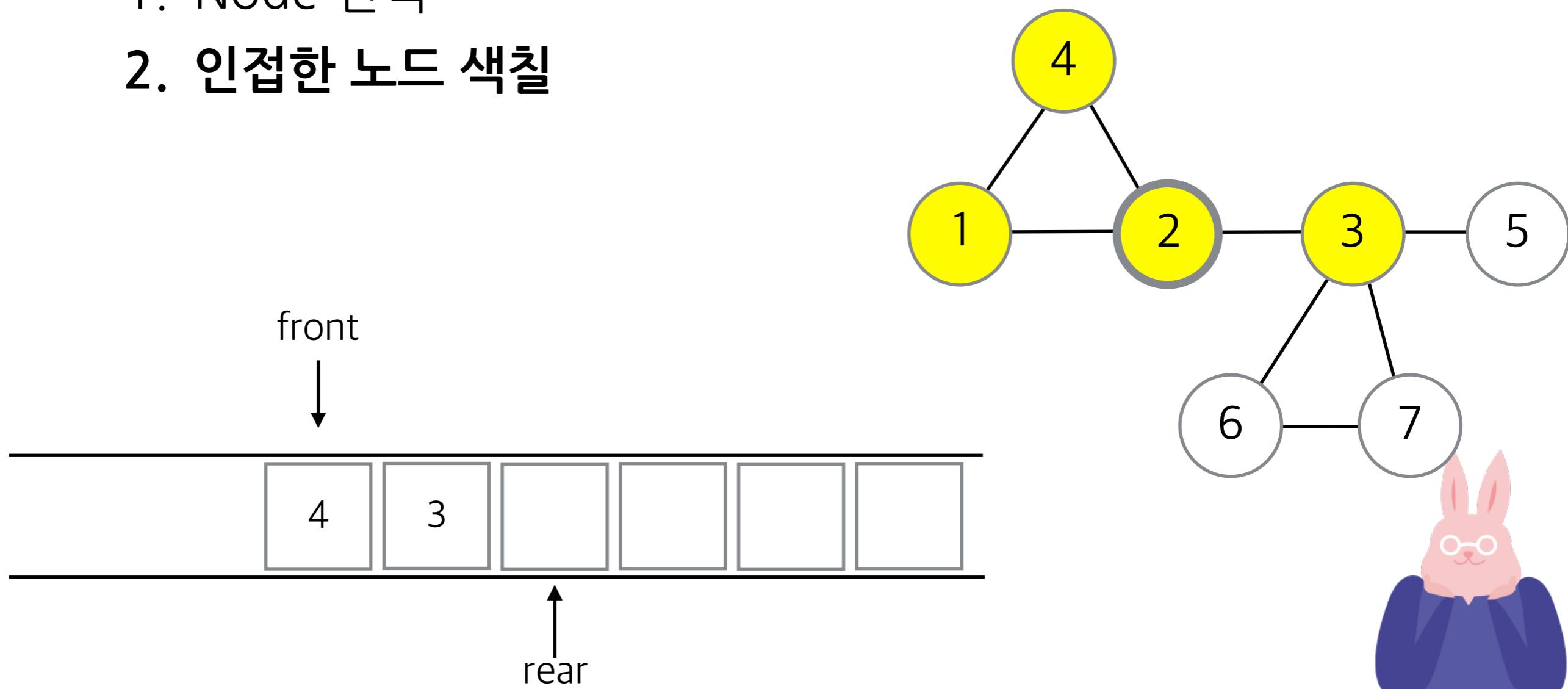
- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다

- Node 선택
- 인접한 노드 색칠



# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠

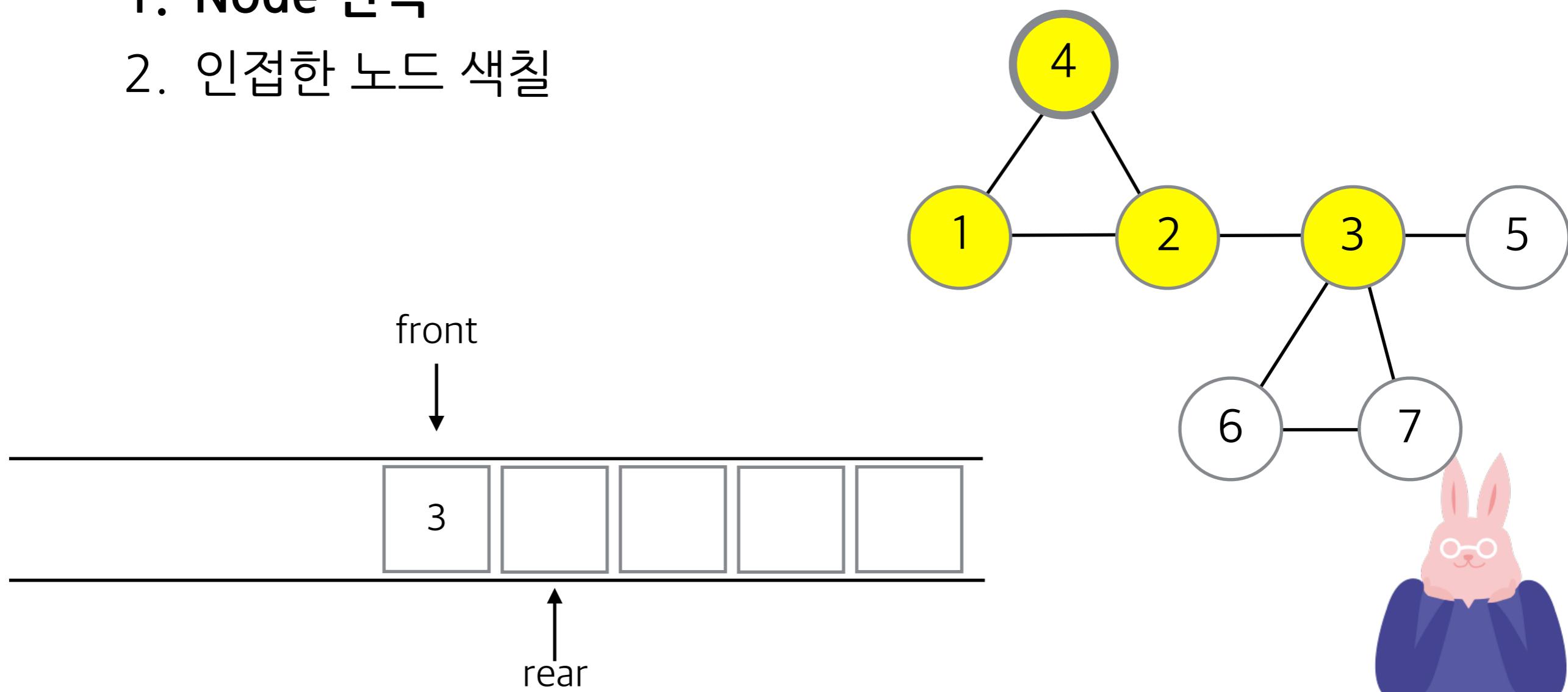


# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다

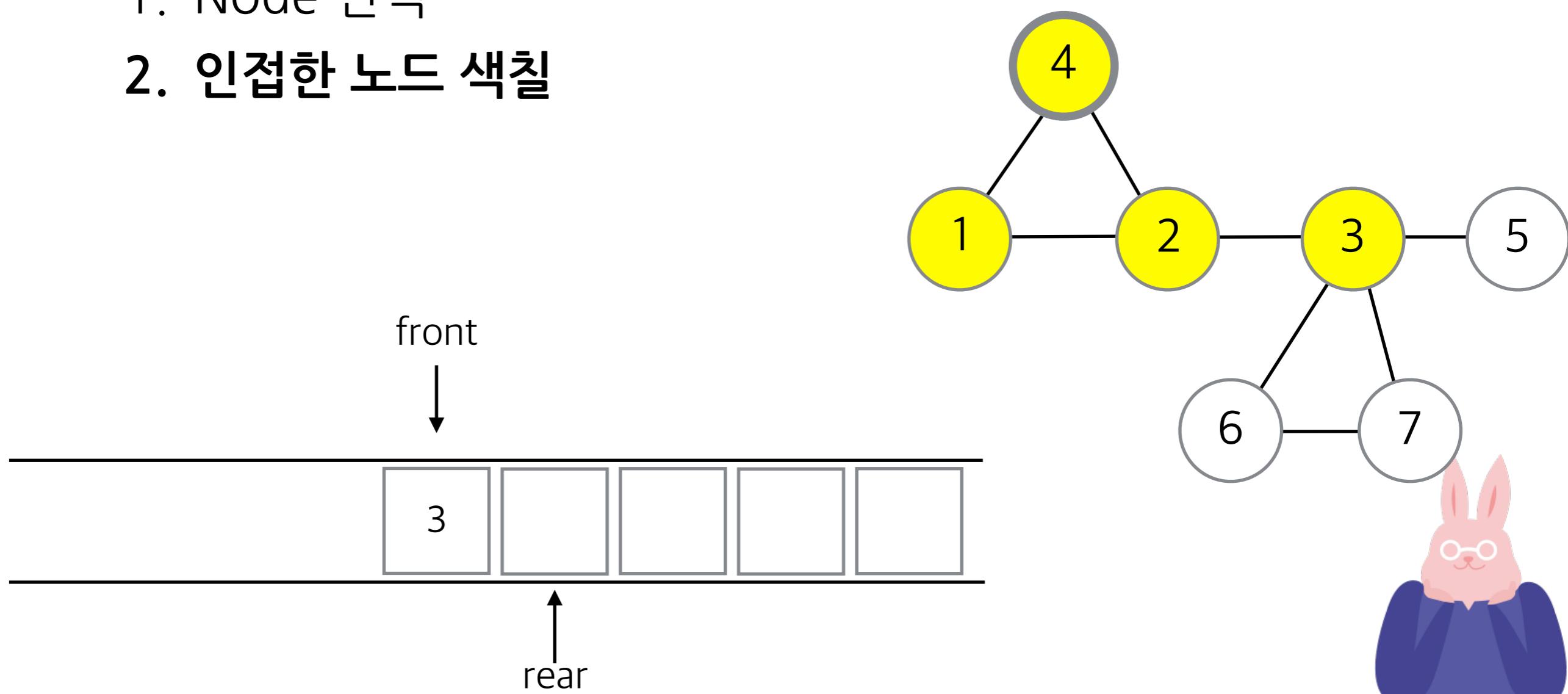
1. Node 선택

2. 인접한 노드 색칠



# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠

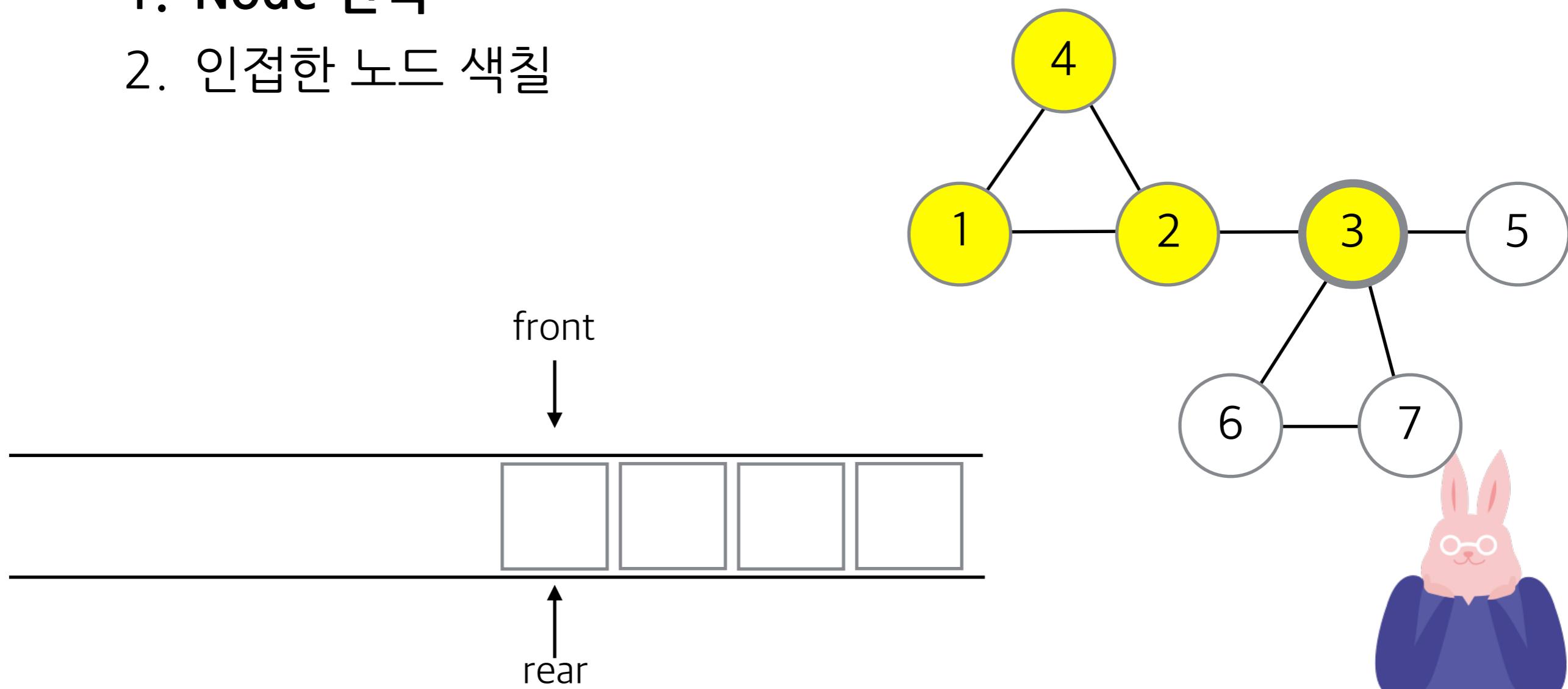


# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다

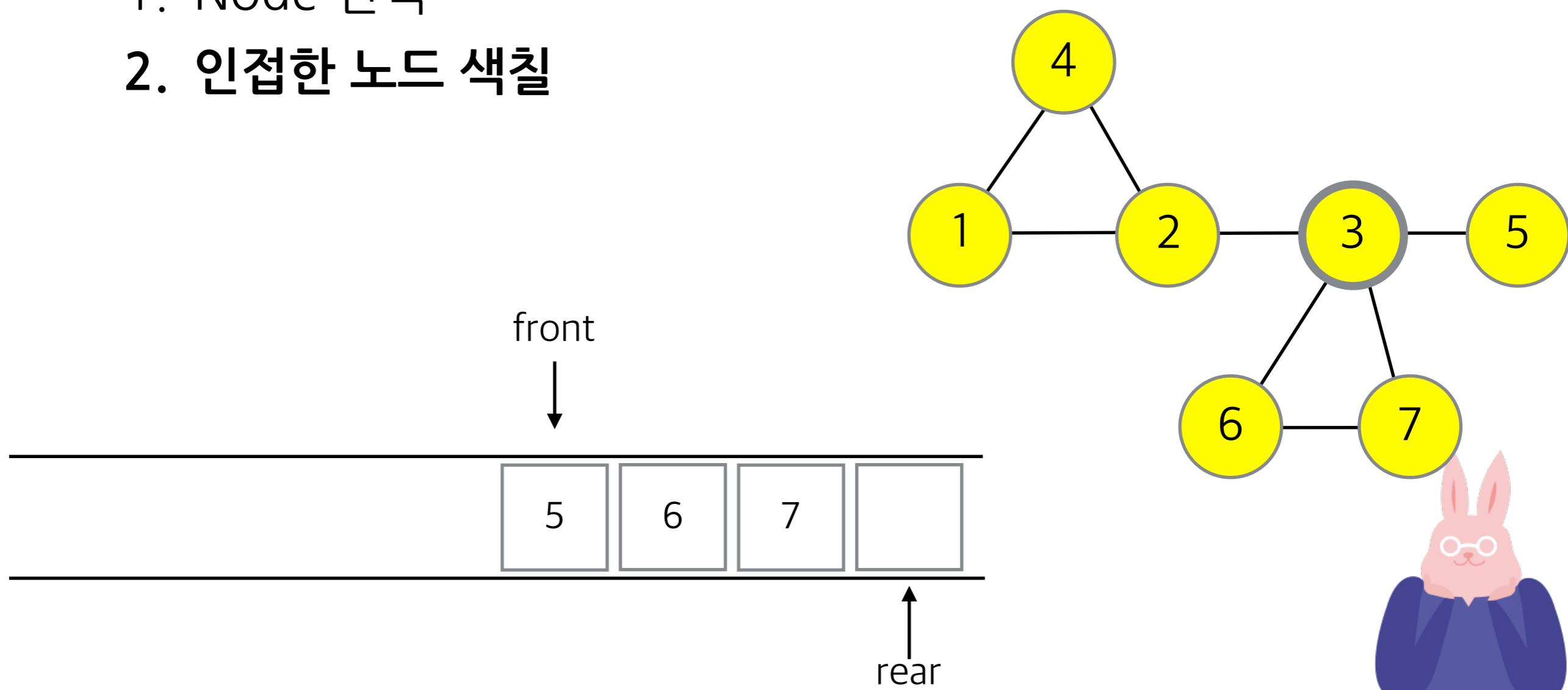
1. Node 선택

2. 인접한 노드 색칠



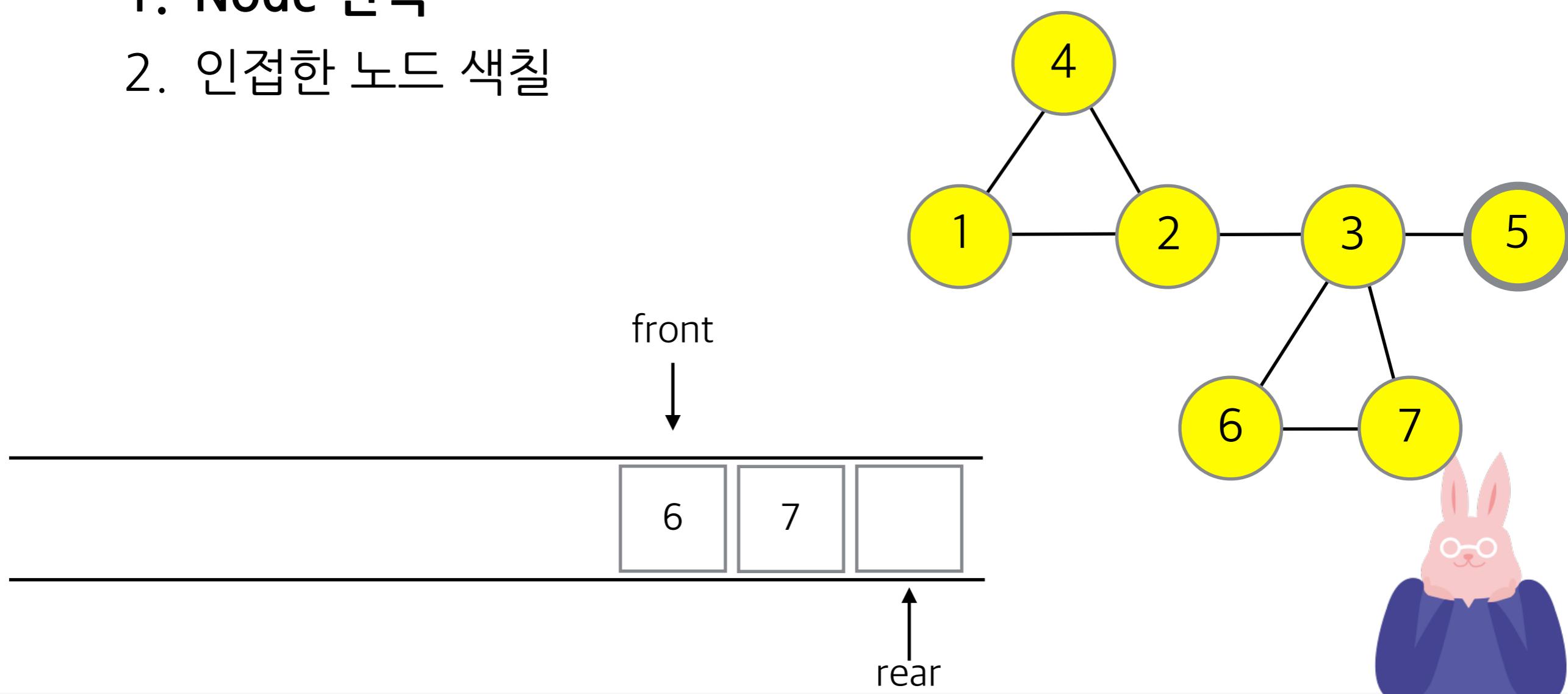
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠



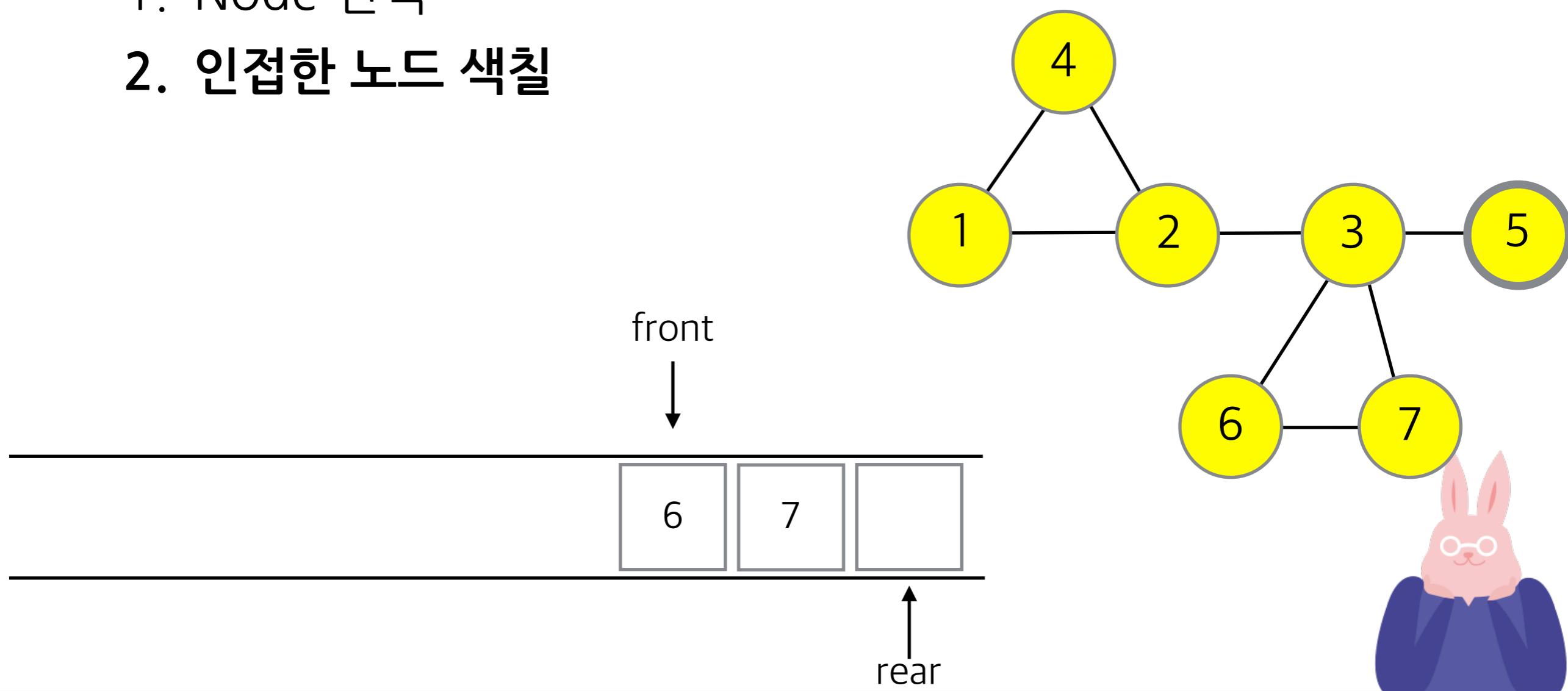
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
1. Node 선택
  2. 인접한 노드 색칠



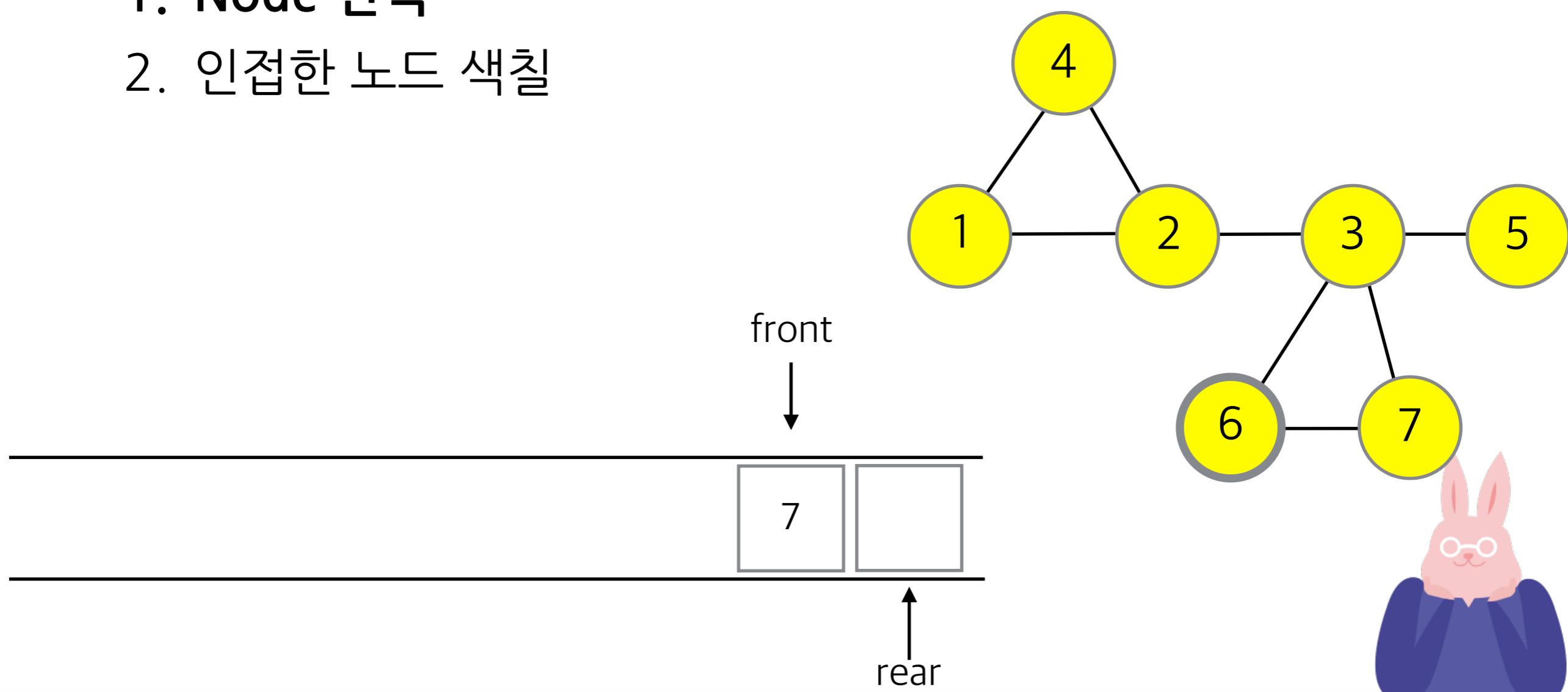
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠



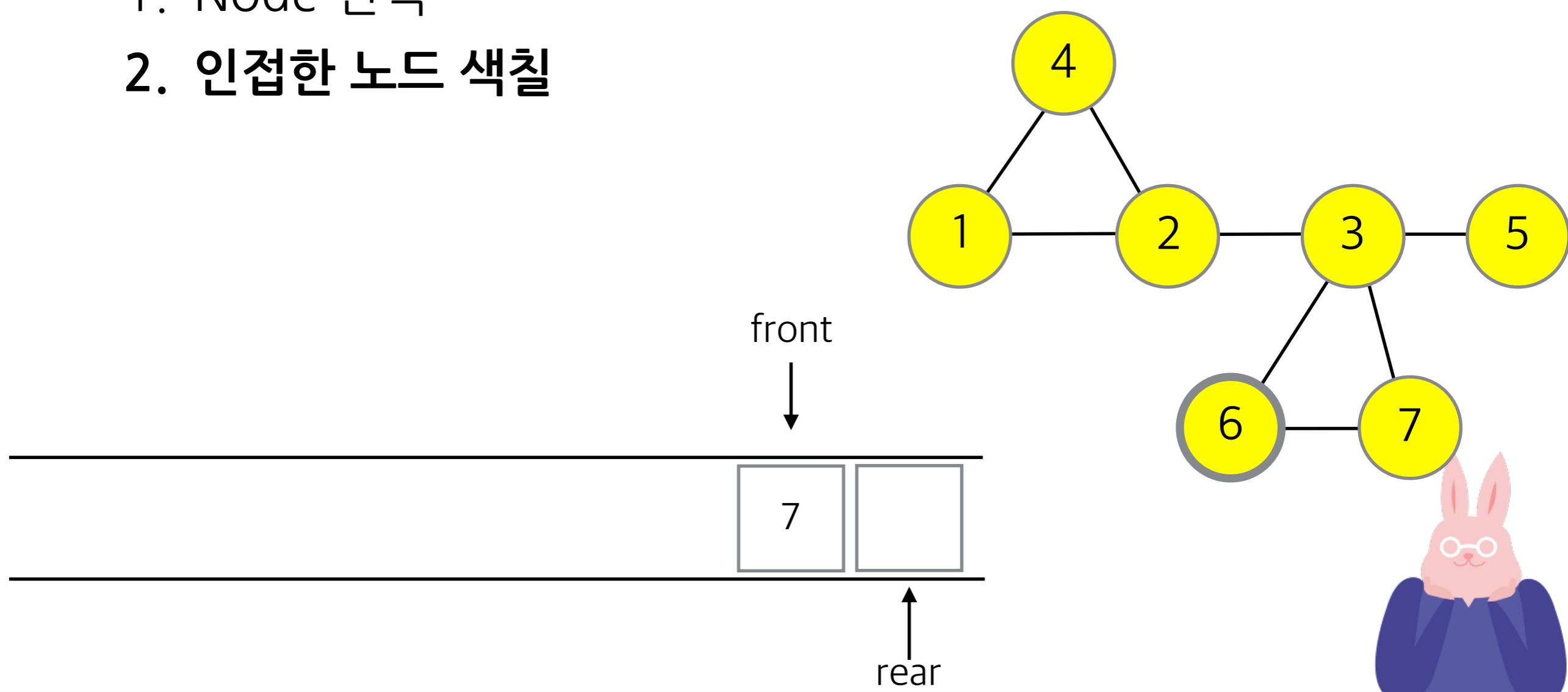
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
1. Node 선택
  2. 인접한 노드 색칠



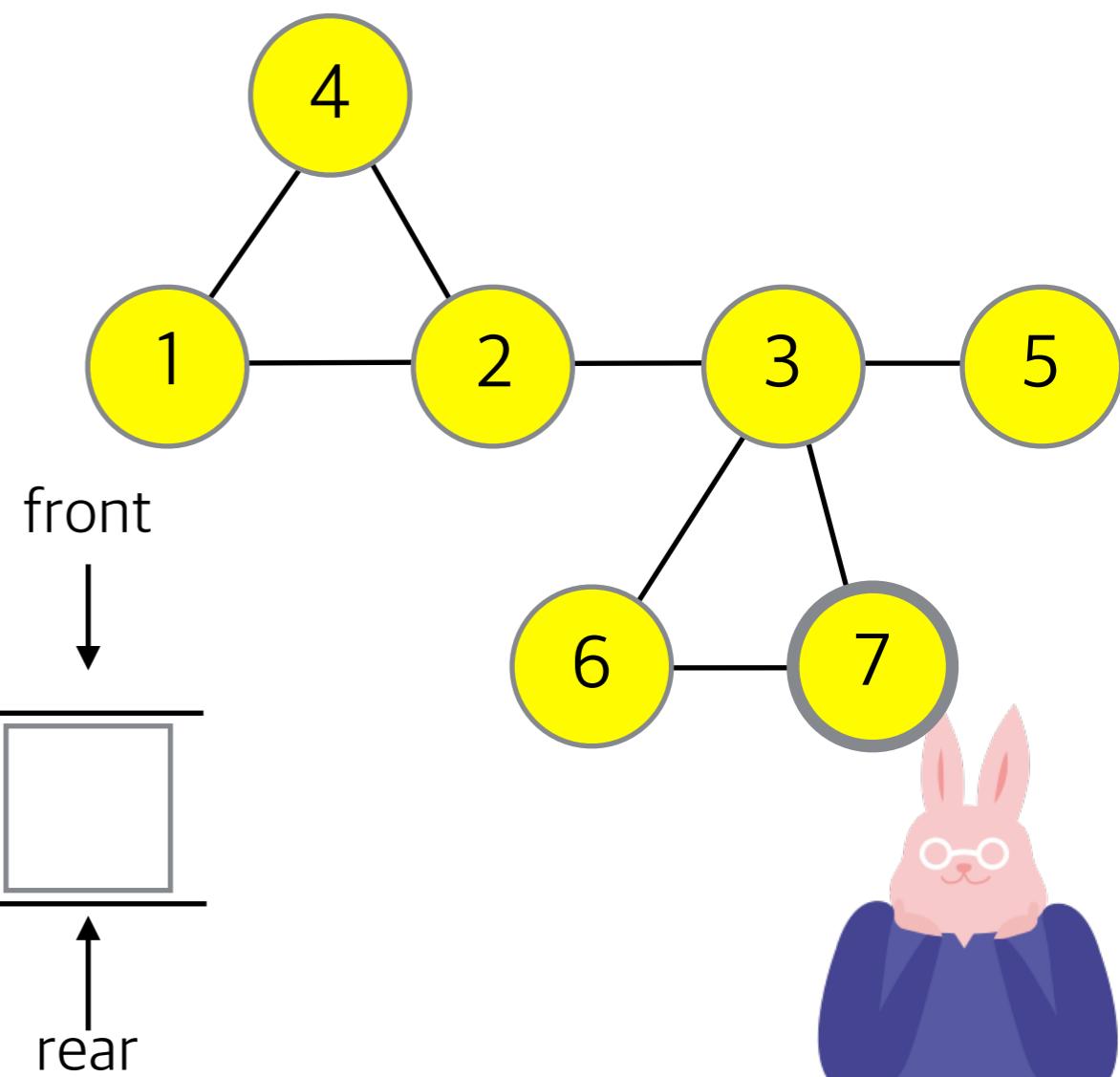
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠



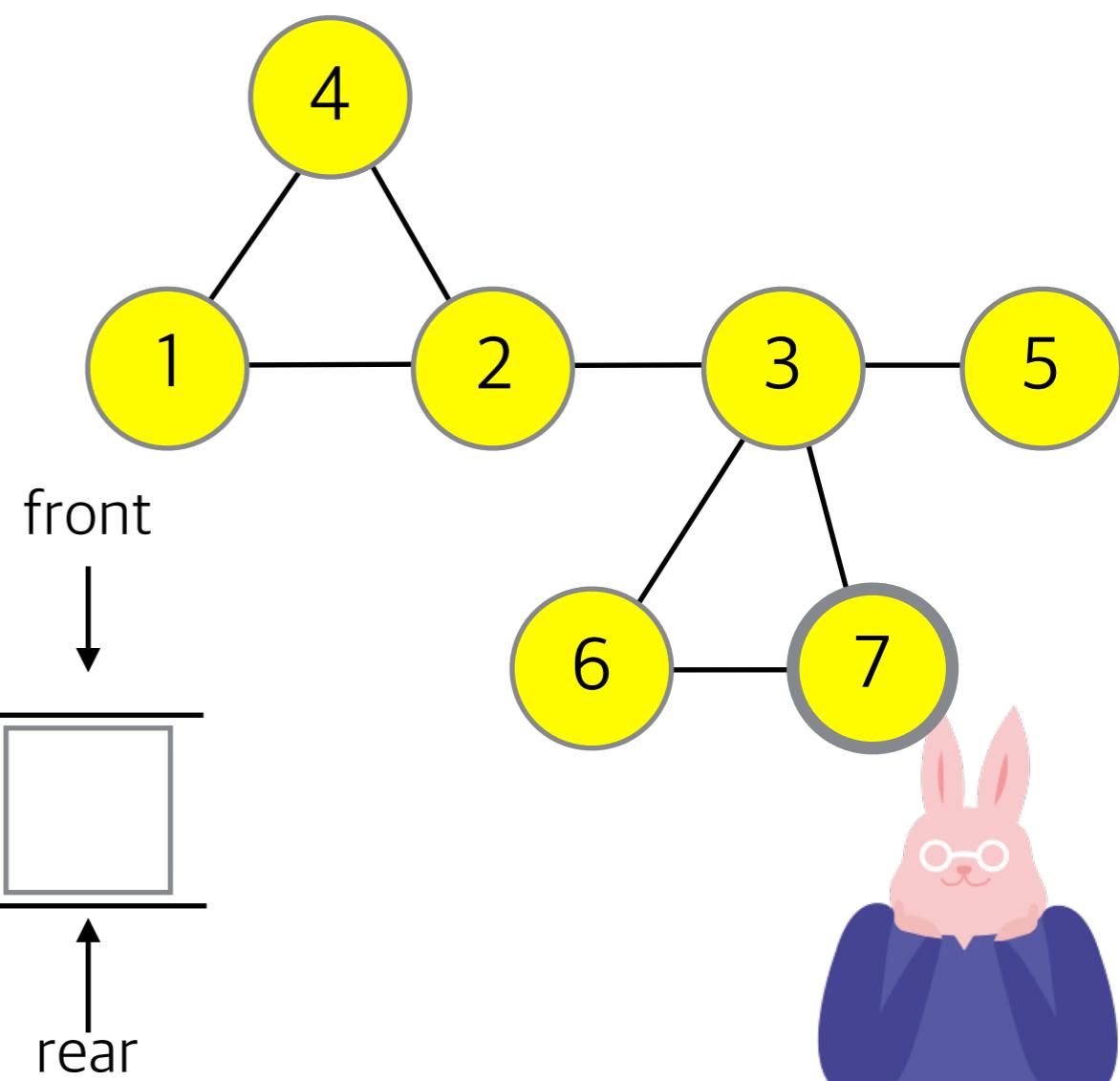
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
1. Node 선택
  2. 인접한 노드 색칠



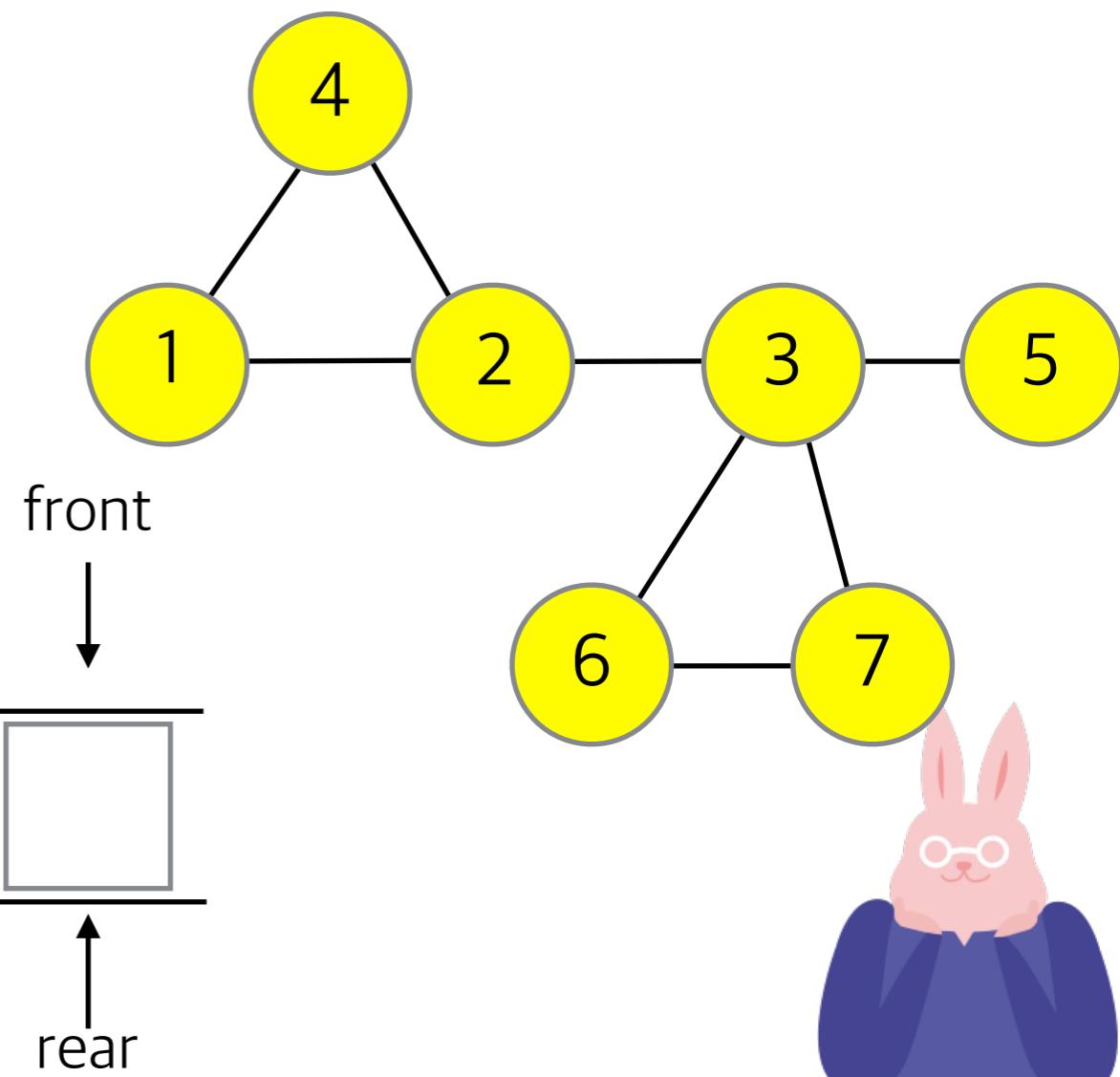
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠



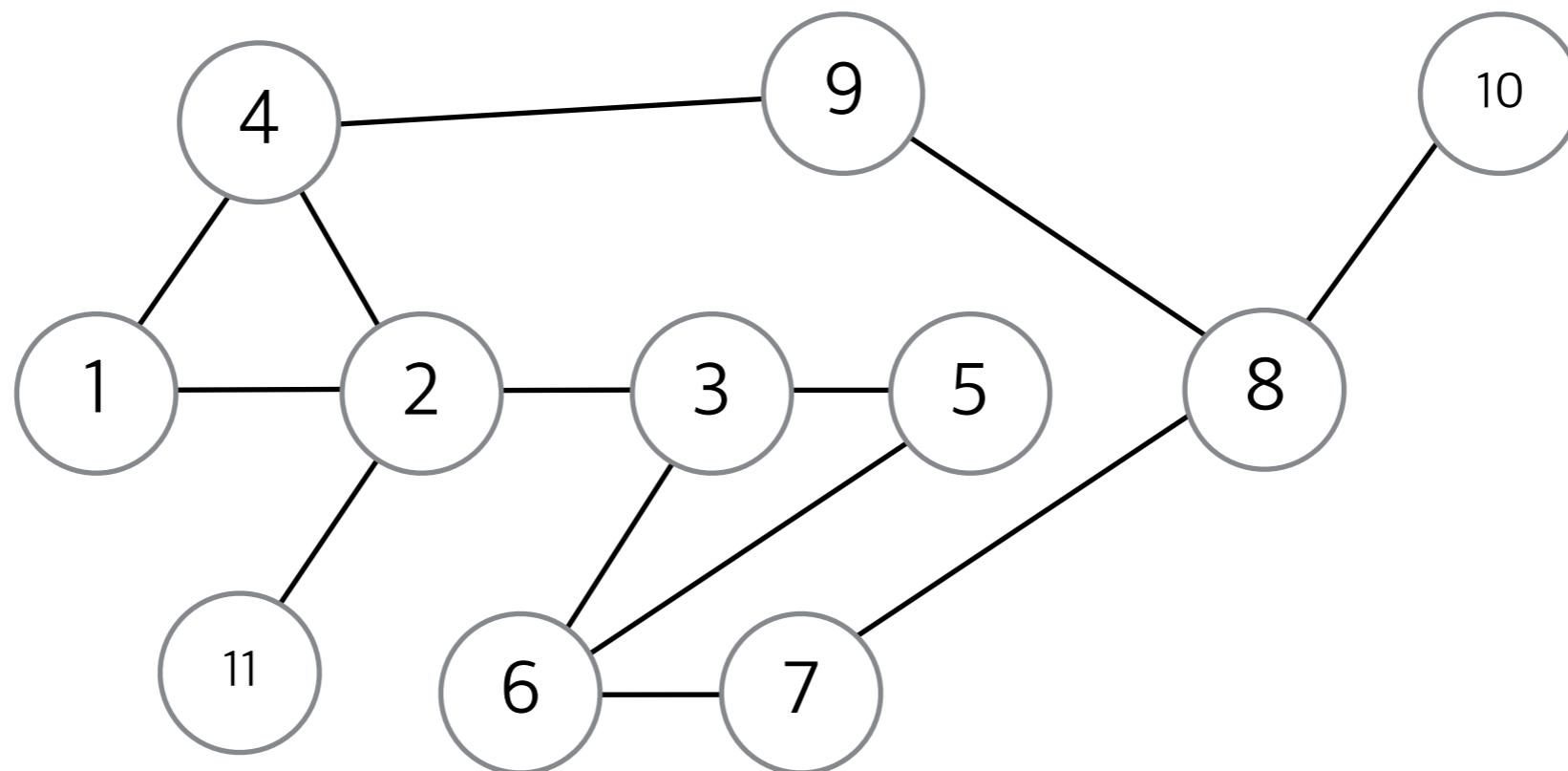
# 너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뺀어나간다
  - Node 선택
  - 인접한 노드 색칠



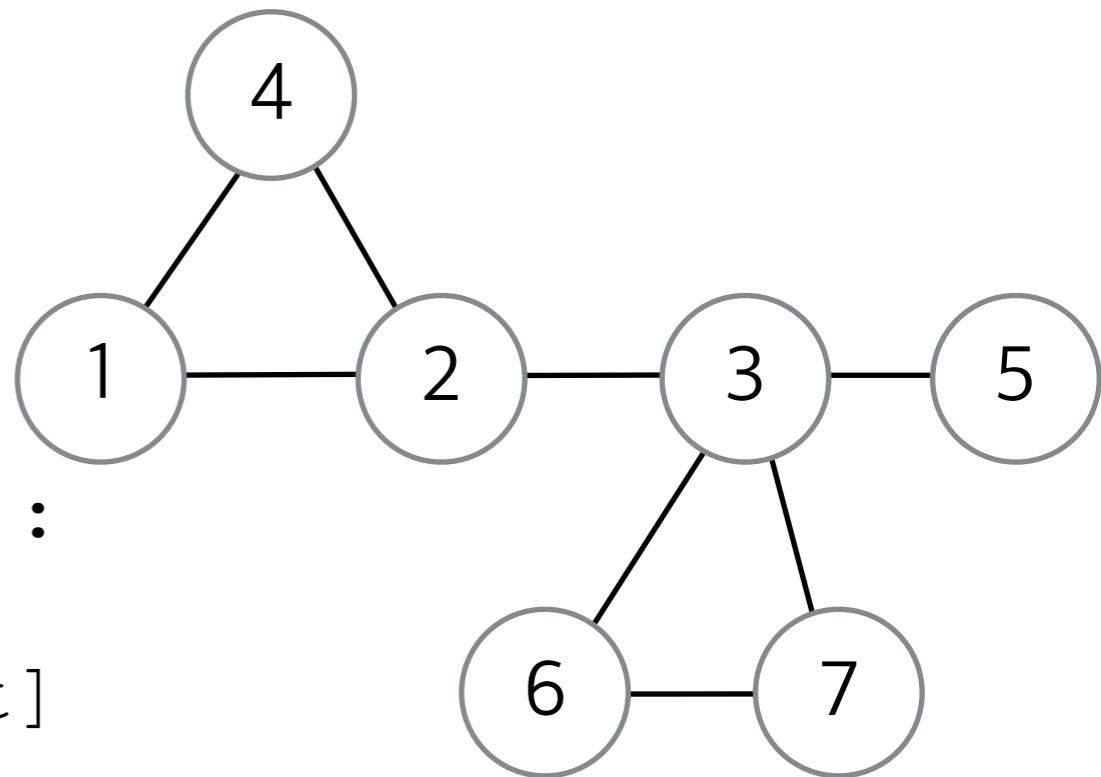
# 너비 우선 탐색 (BFS)

- 다음 그래프에 대하여 1을 시작으로 BFS한 결과는 ?
  - 단, 인접한 노드가 여러개일 경우 노드 번호가 작은 노드부터 방문



# 너비 우선 탐색 (BFS) 구현

```
def BFS(graph, x, visited) :  
    result = []  
    visited[x] = True  
    Queue.put(x)  
  
    while Queue.empty() == False :  
        current = Queue.get()  
        result = result + [current]  
  
        for v in graph[current] :  
            if visited[v] == False :  
                visited[v] = True  
                Queue.put(v)  
  
    return result
```



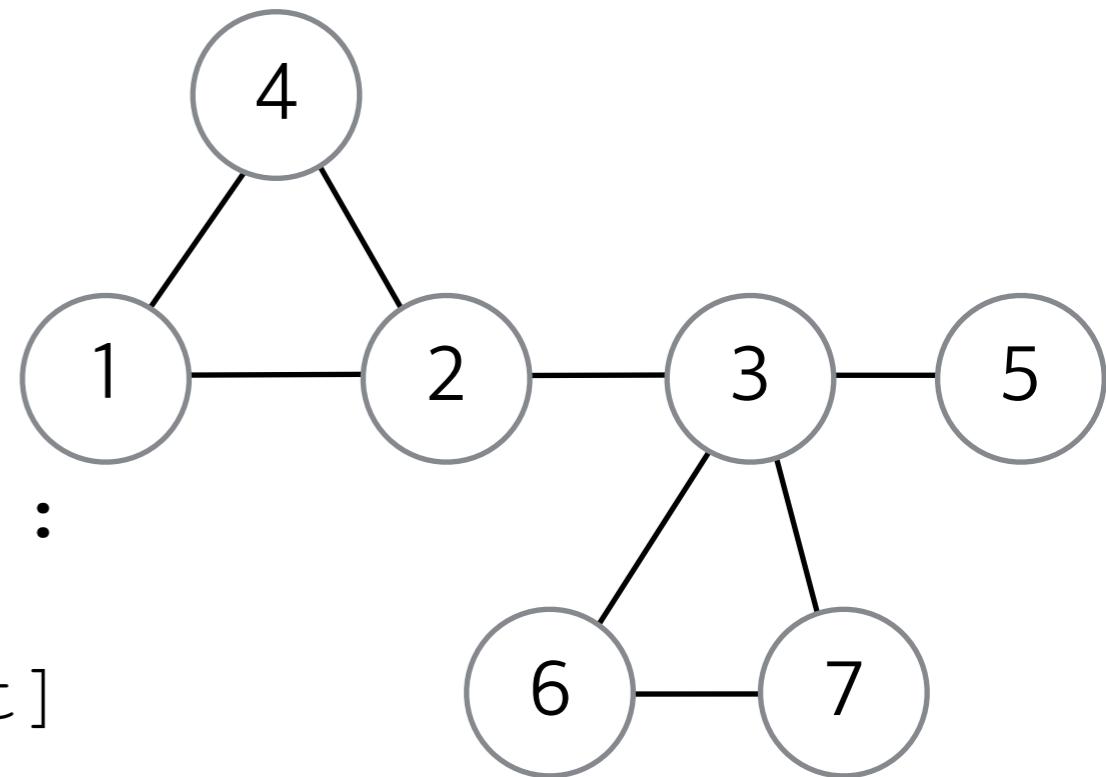
# 너비 우선 탐색 (BFS) 구현

```
def BFS(graph, x, visited) :
    result = []
    visited[x] = True
    Queue.put(x)

    while Queue.empty() == False :
        current = Queue.get()
        result = result + [current]

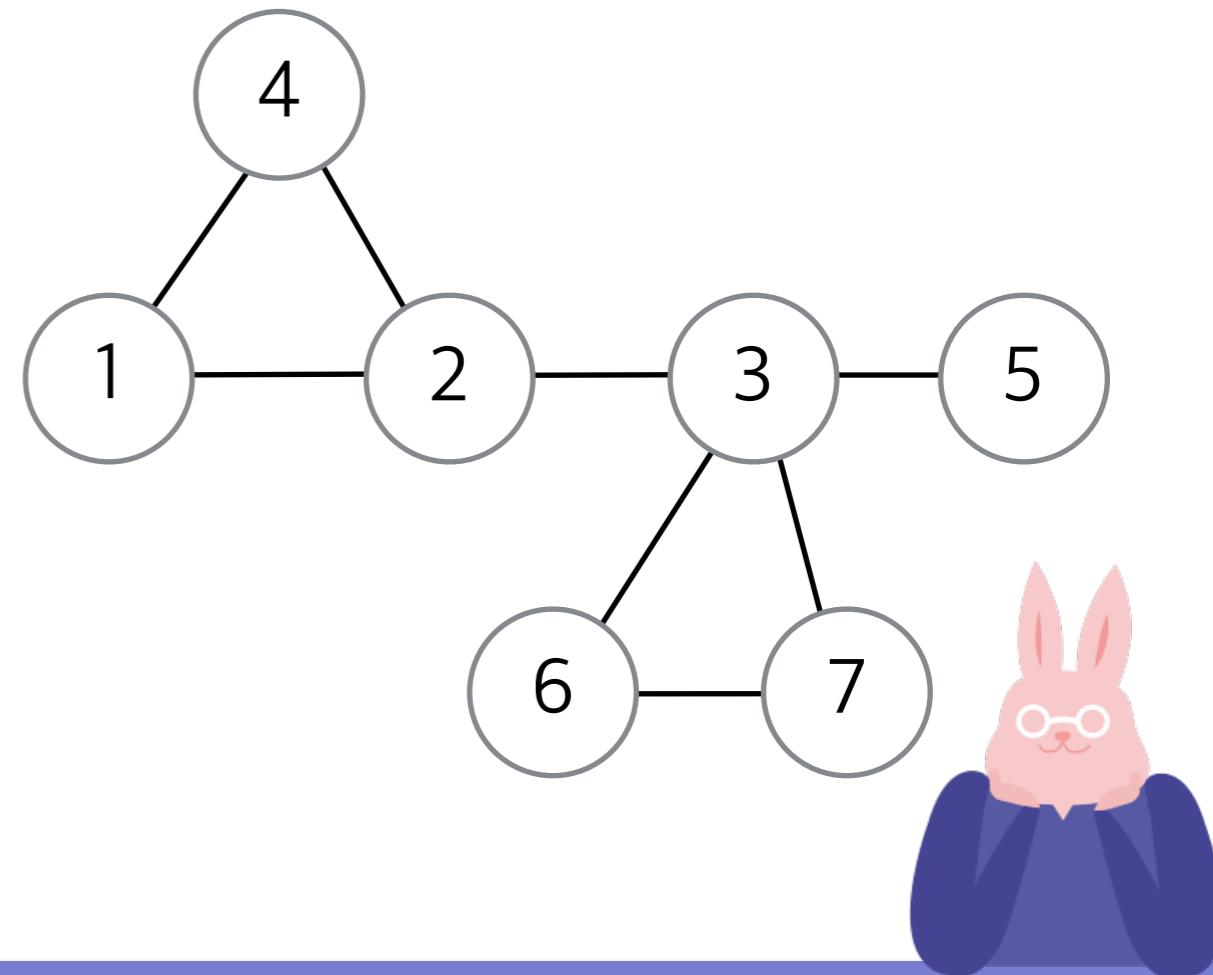
        for v in graph[current] :
            if visited[v] == False :
                visited[v] = True
                Queue.put(v)

    return result
```



# DFS와 BFS의 정확성 증명 및 시간복잡도

- 정확성 증명
  - 질문 : 모든 Node와 Edge를 방문하는가 ?
- 시간복잡도

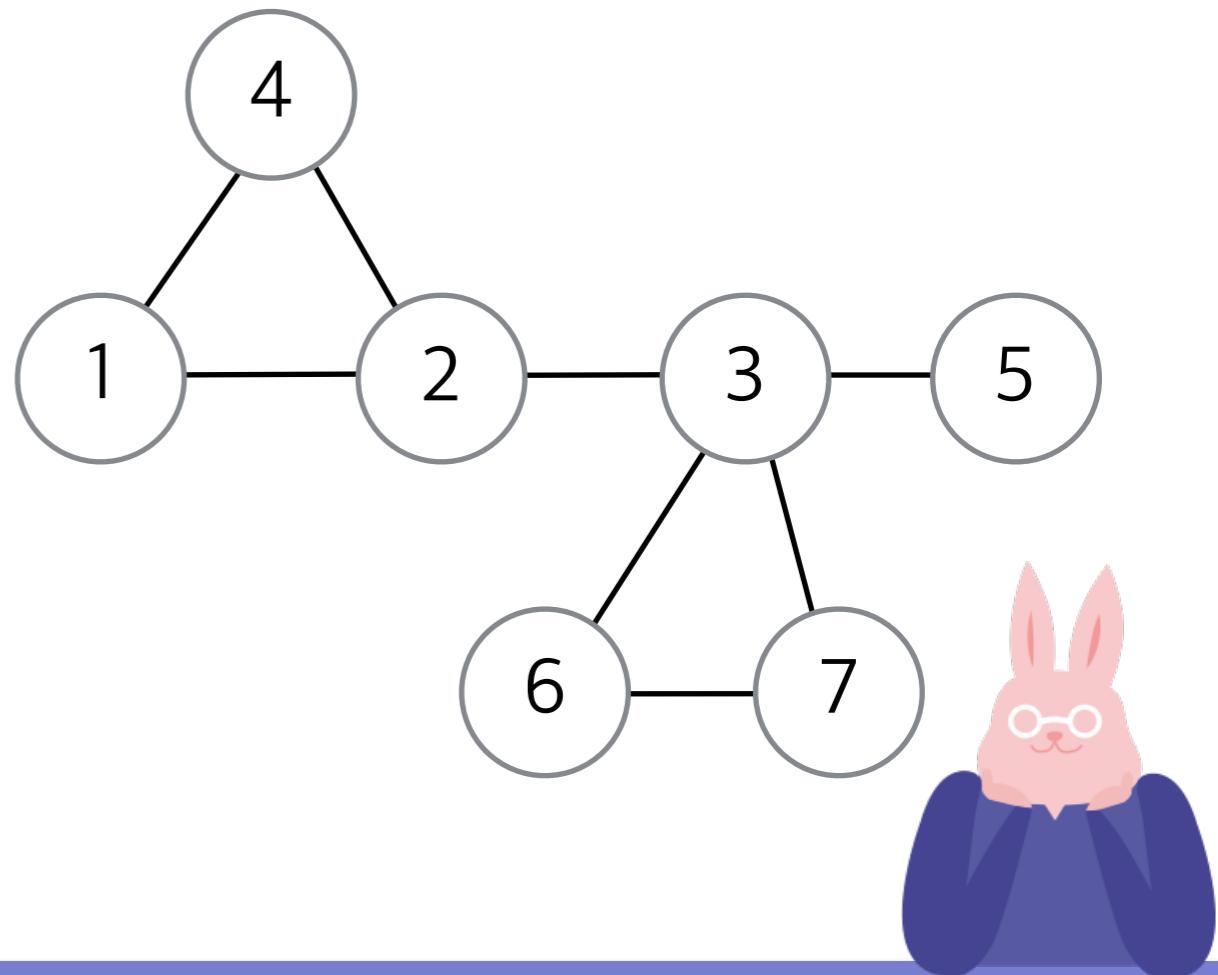


# DFS와 BFS의 정확성 증명 및 시간복잡도

- 정확성 증명
  - 질문 : 모든 Node와 Edge를 방문하는가 ? YES!
- 시간복잡도
  - $O(V+E)$

V : Node의 갯수

E : Edge의 개수



# DFS와 BFS 마무리

- N번 강조해도 지나치지 않음
- 코드를 이해하고, 외우고, 또 외우세요



# [활동문제 1] DFS와 BFS

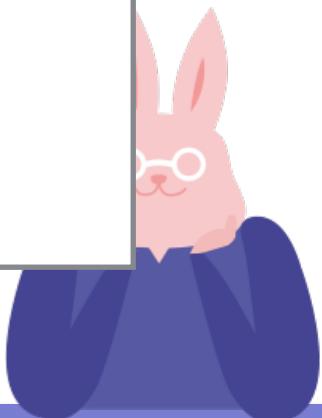
- DFS와 BFS한 결과를 각각 출력

입력의 예

```
7 8  
0 1  
0 3  
1 2  
1 3  
2 4  
2 5  
2 6  
5 6
```

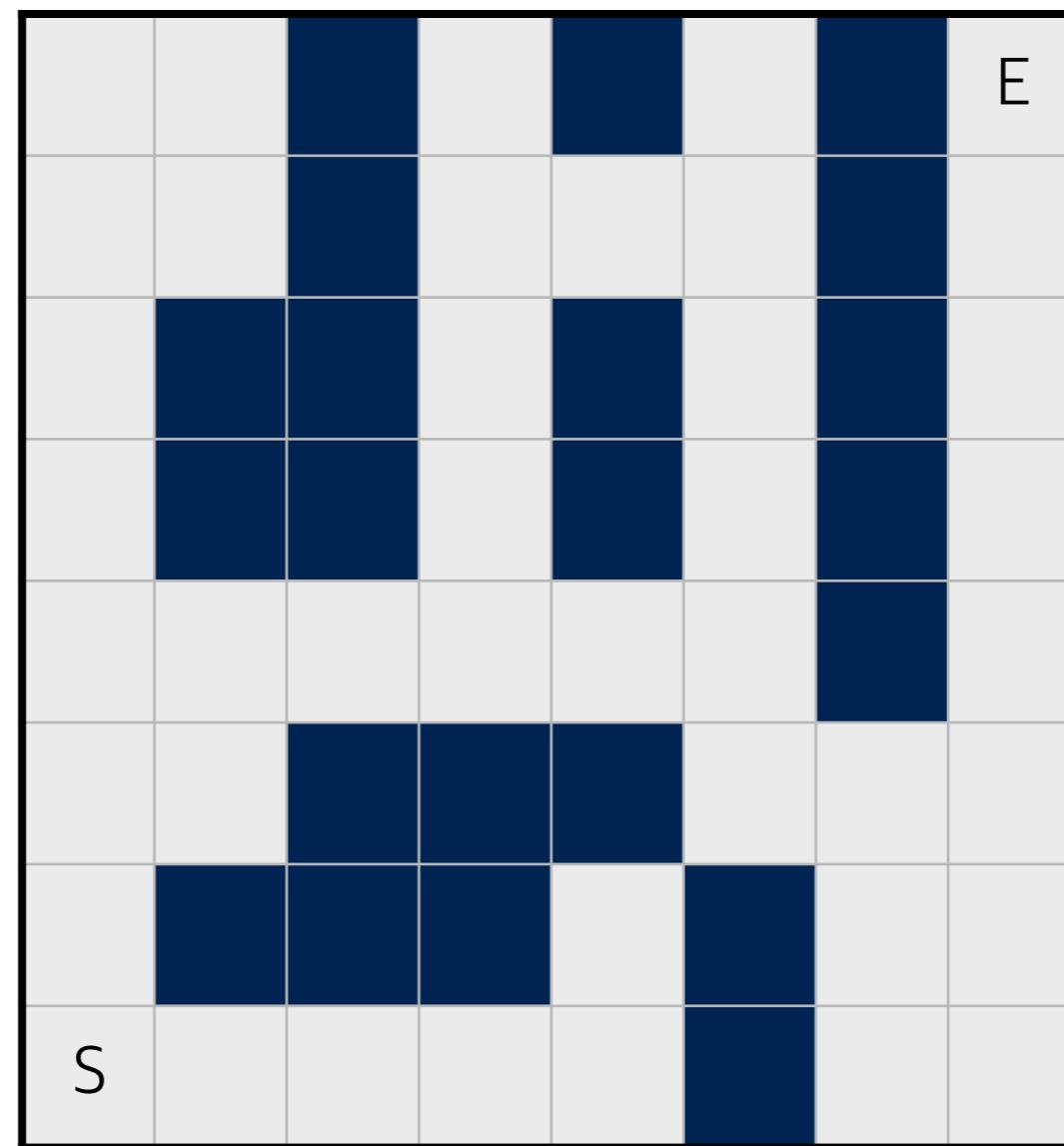
출력의 예

```
0 1 2 4 5 6 3  
0 1 3 2 4 5 6
```



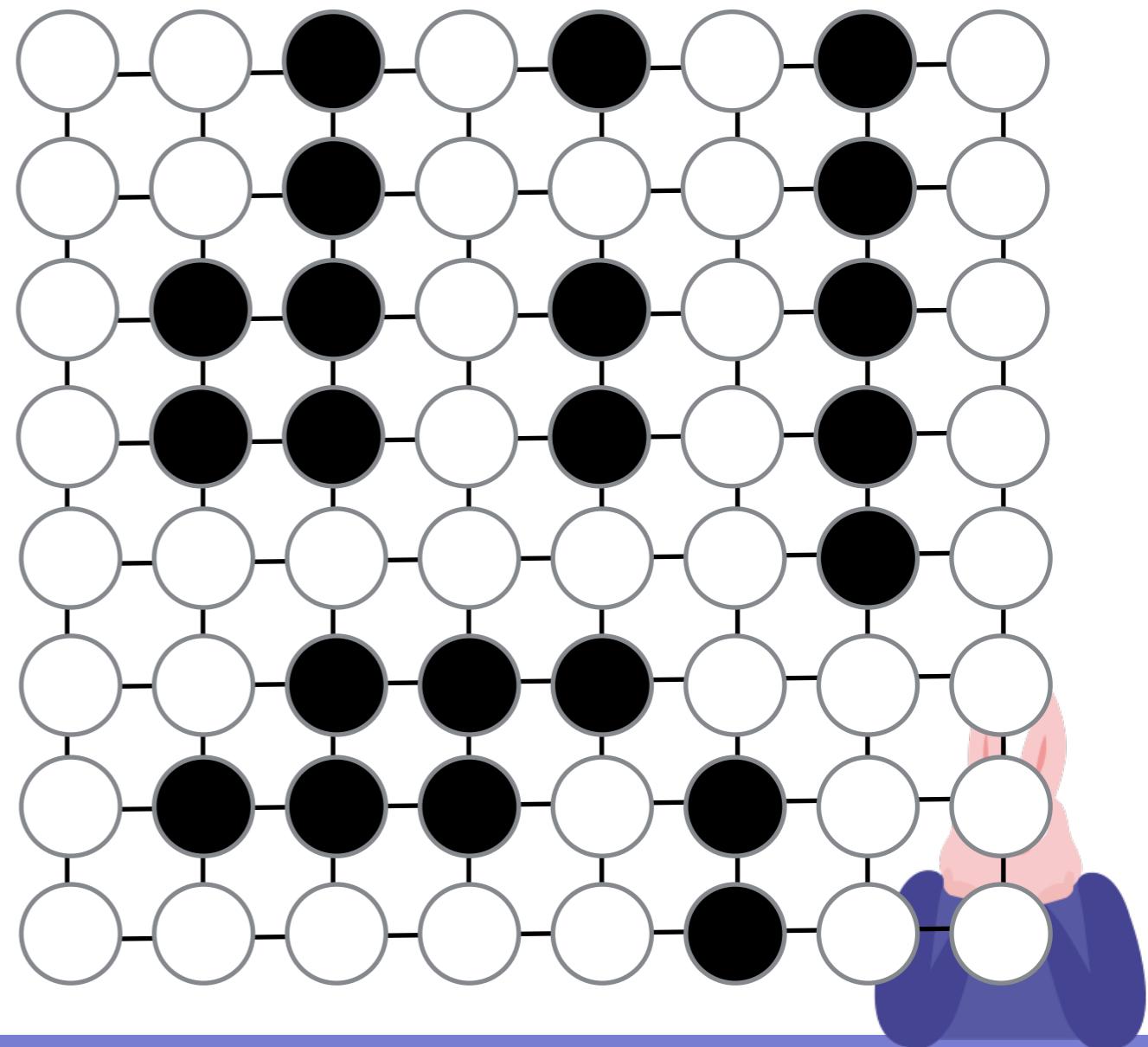
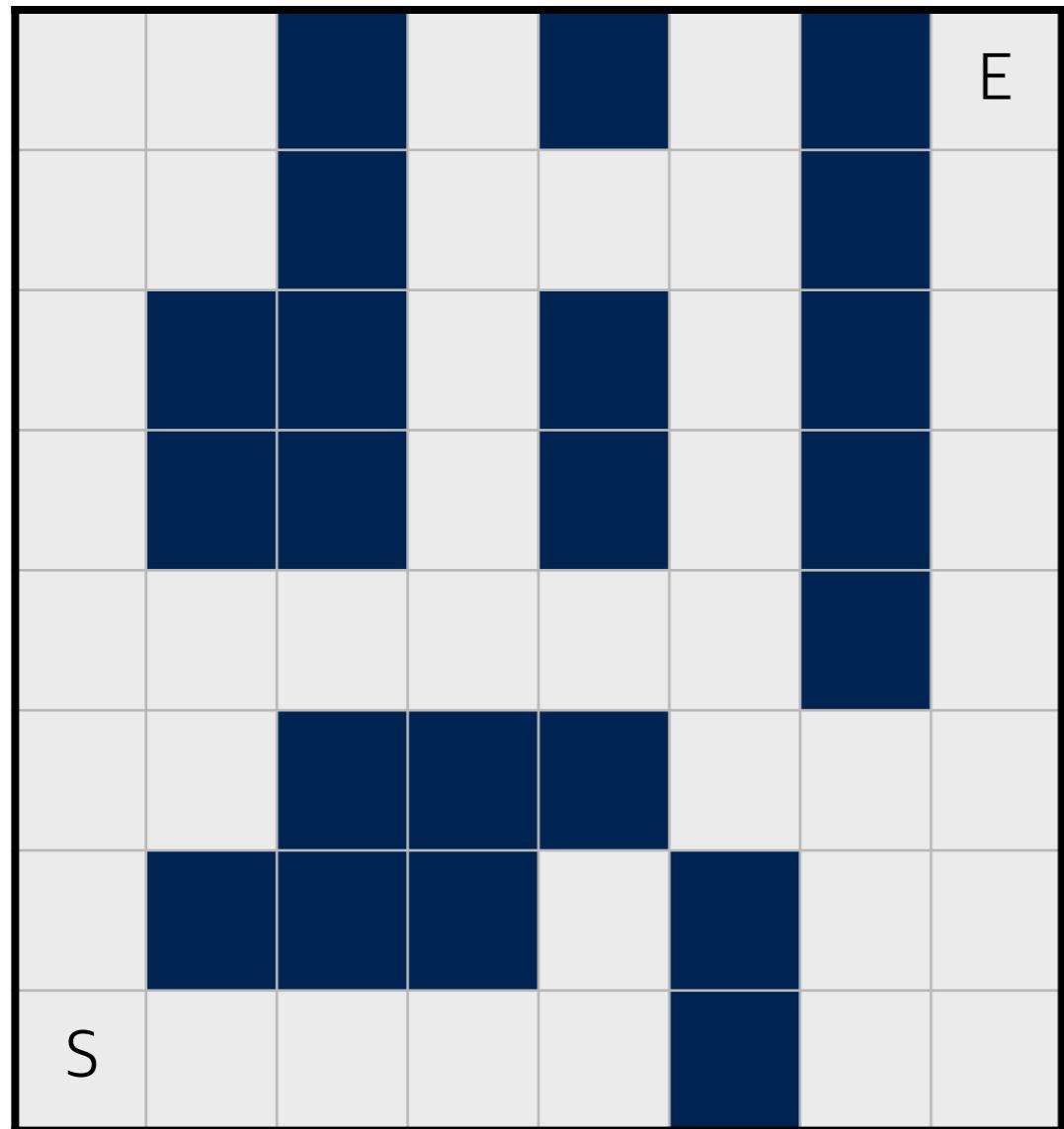
# DFS와 BFS의 응용 : 미로찾기

- 시작점에서 도착점까지 갈 때의 최단거리를 출력하라



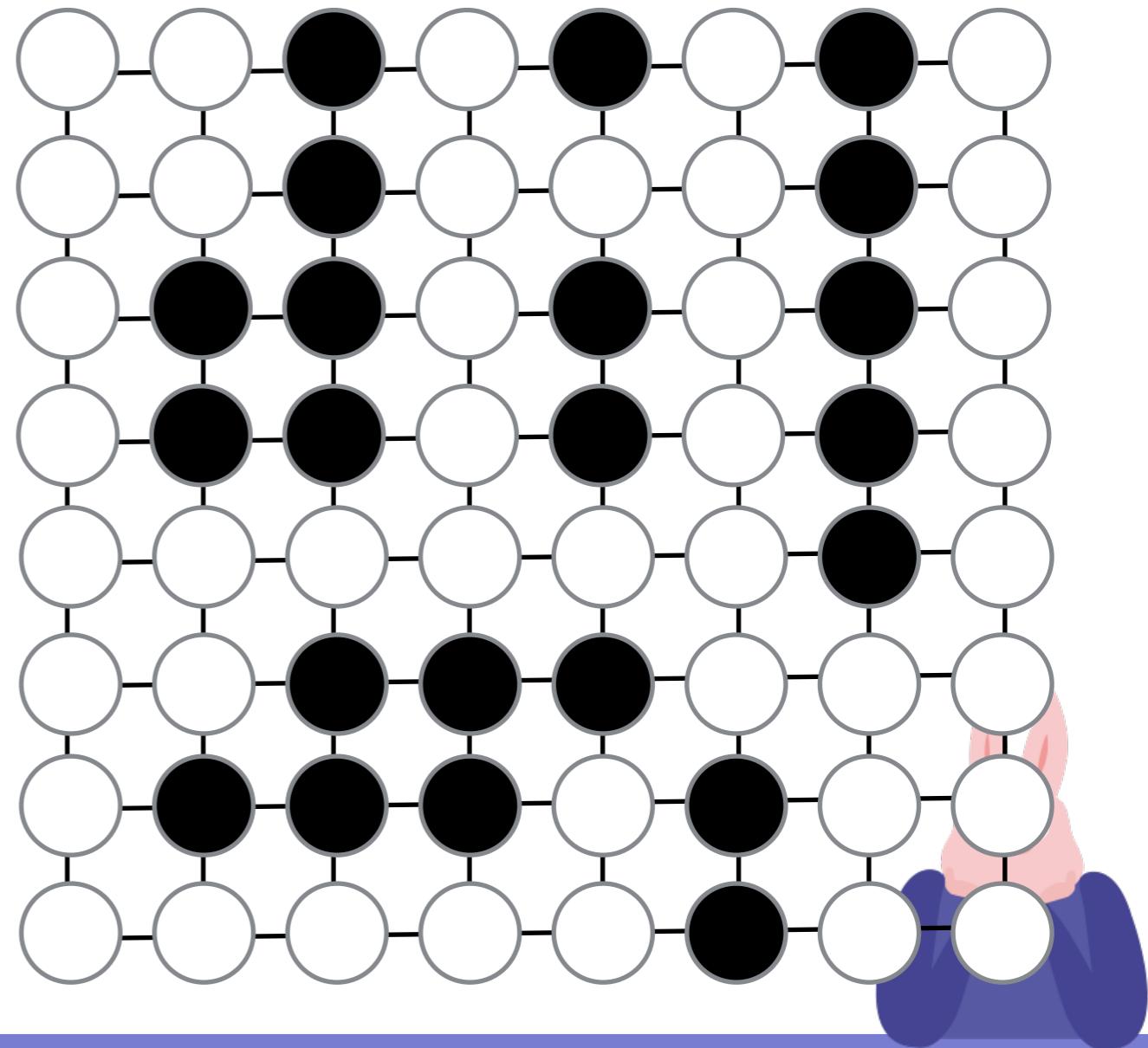
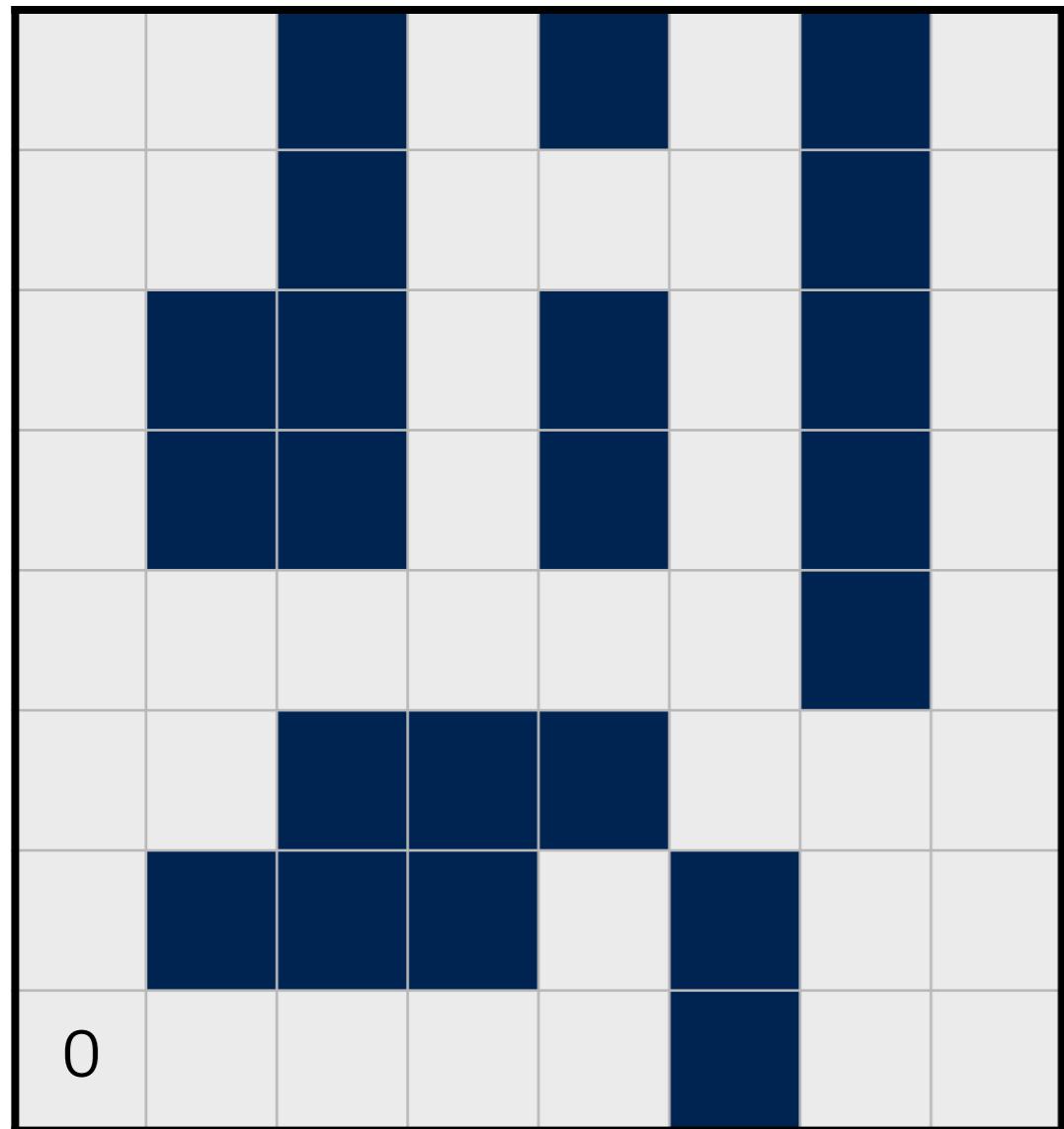
# DFS와 BFS의 응용 : 미로찾기

## 1. Graph로 변환



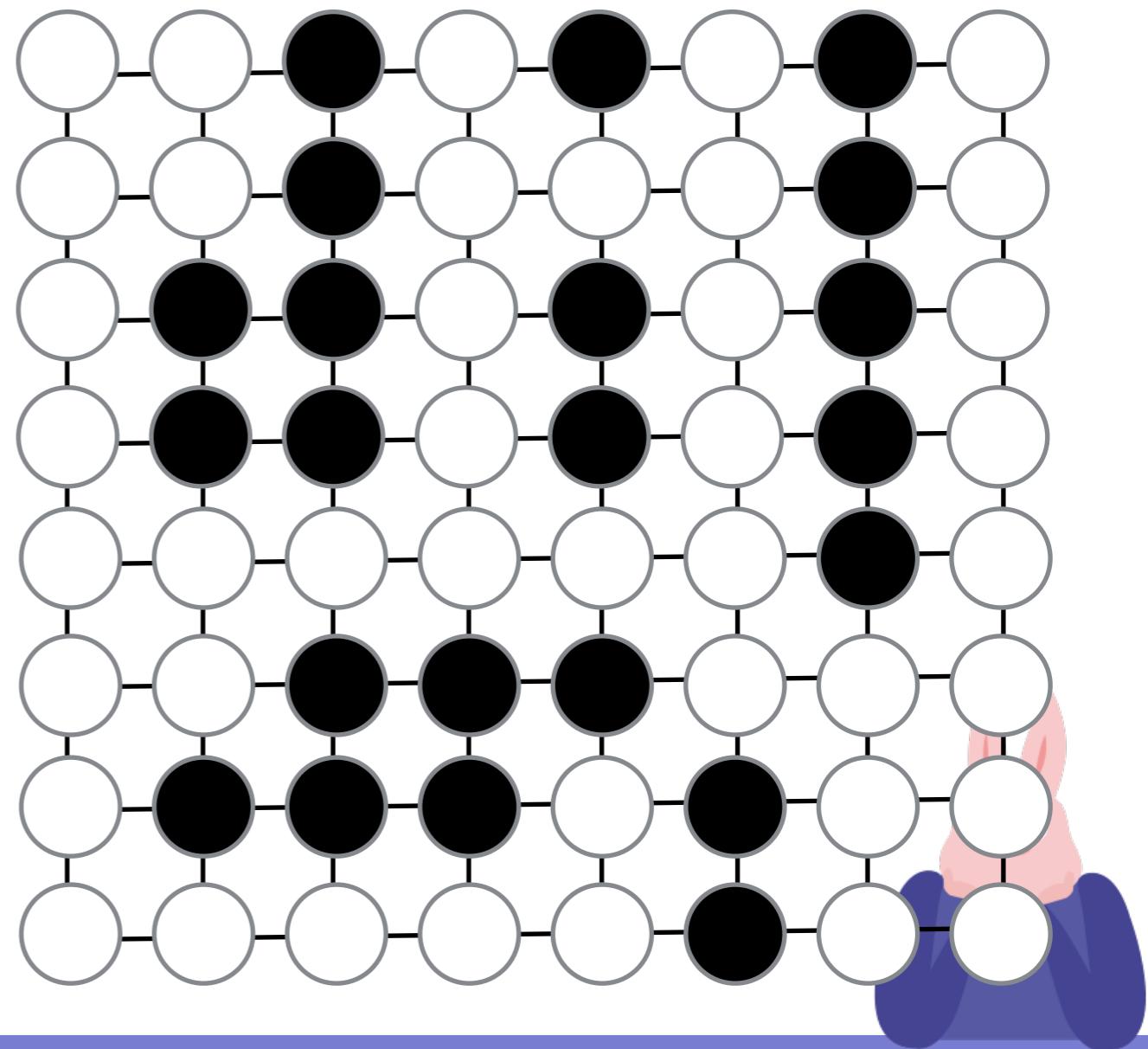
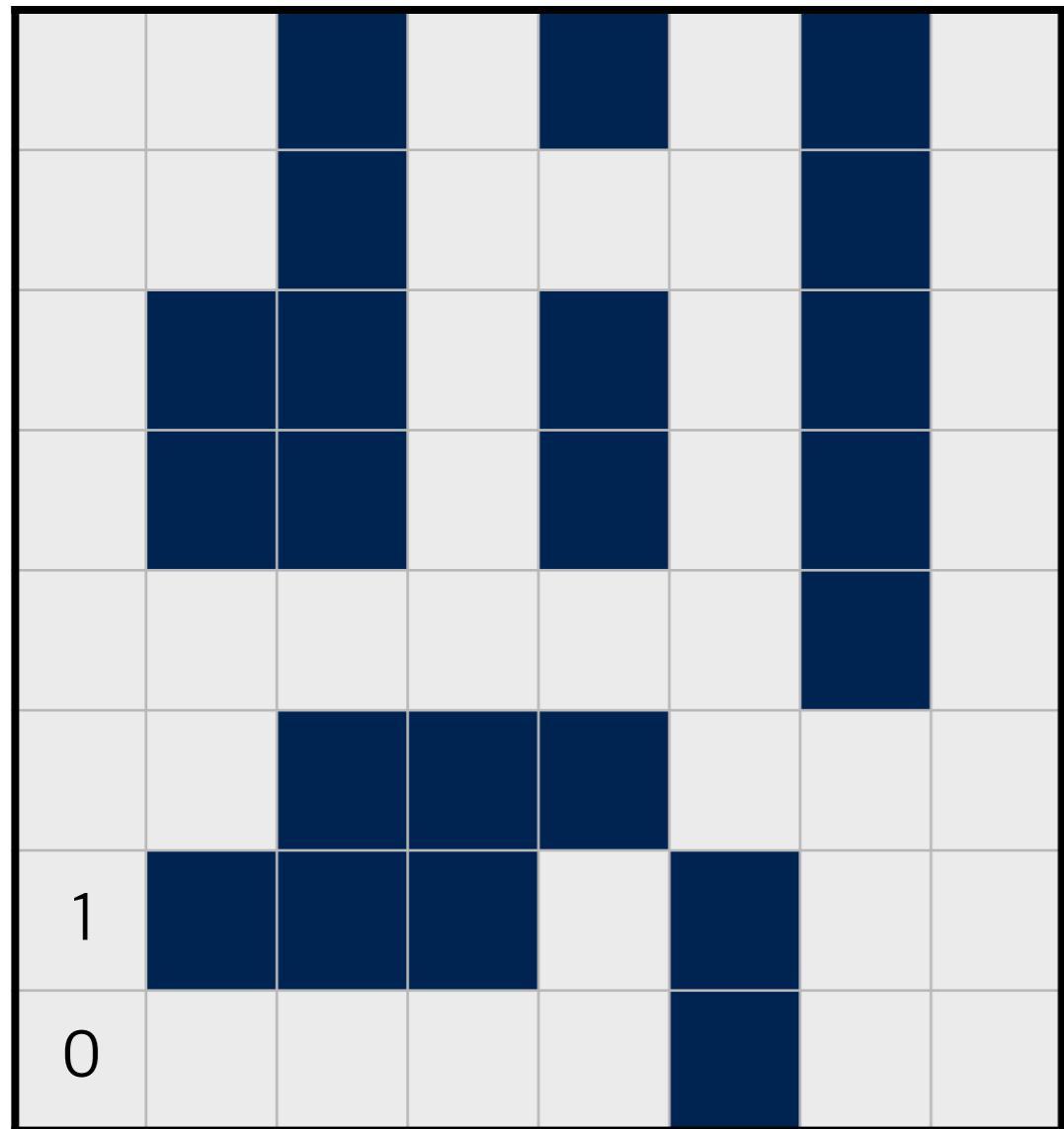
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



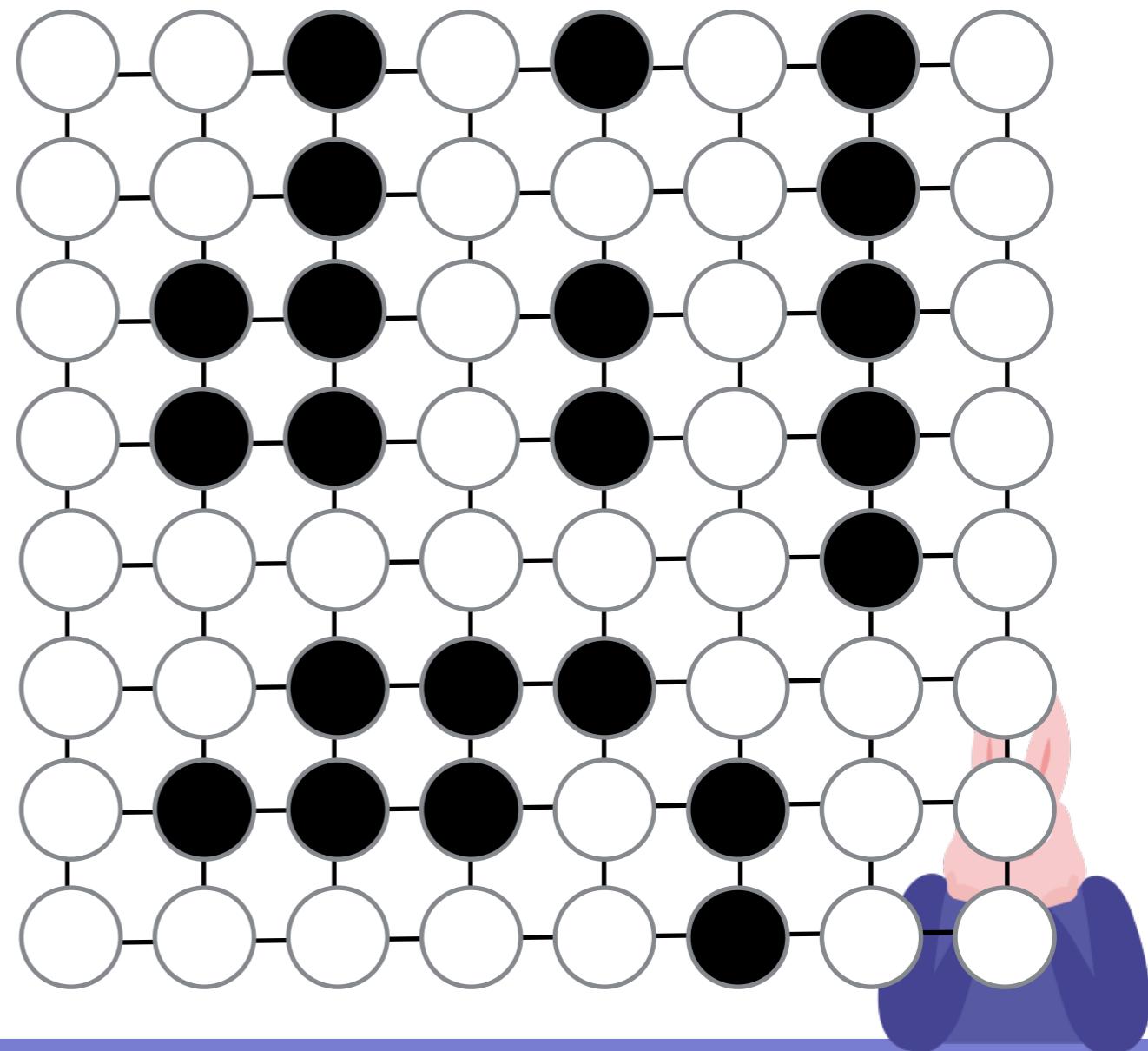
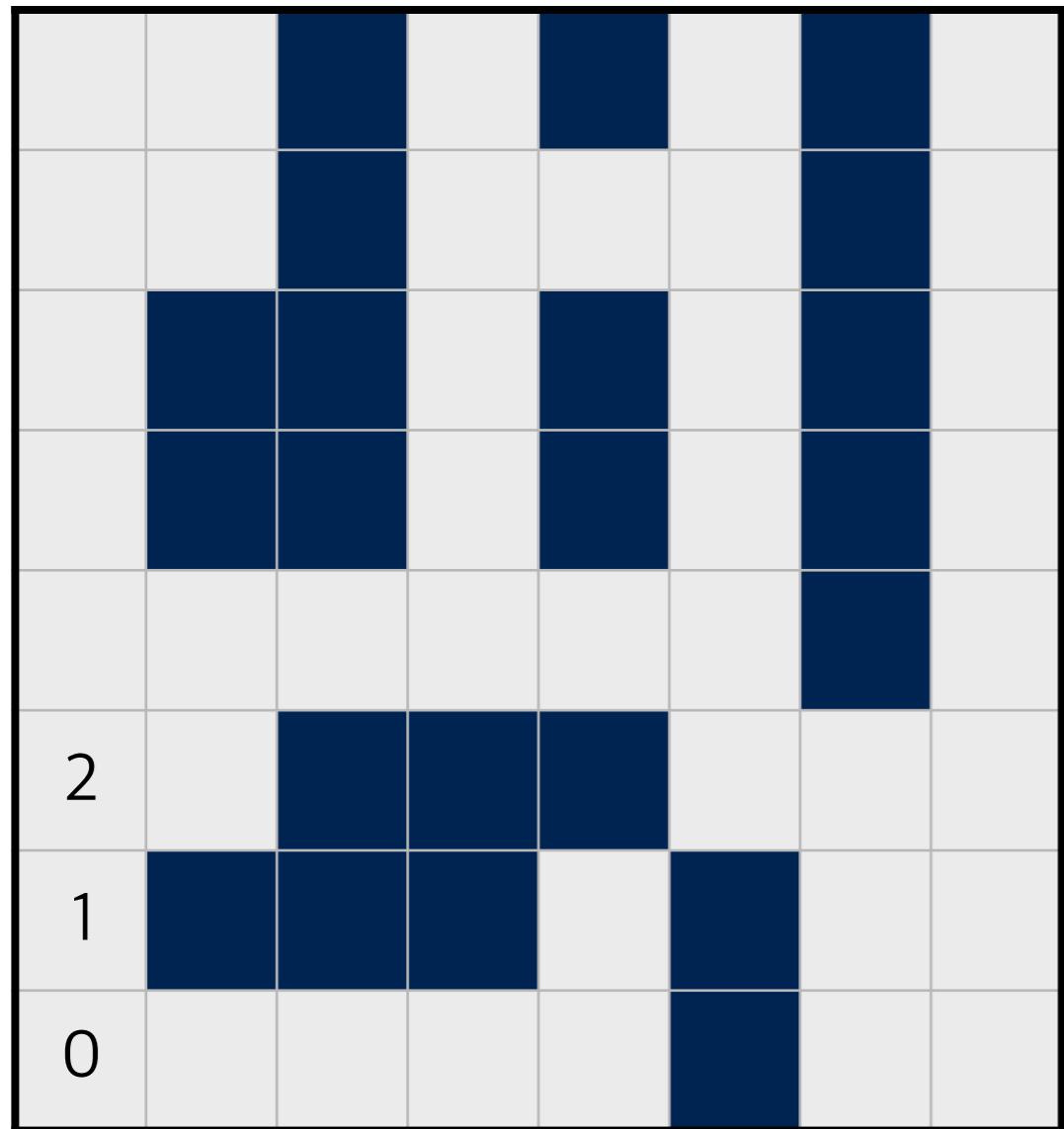
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



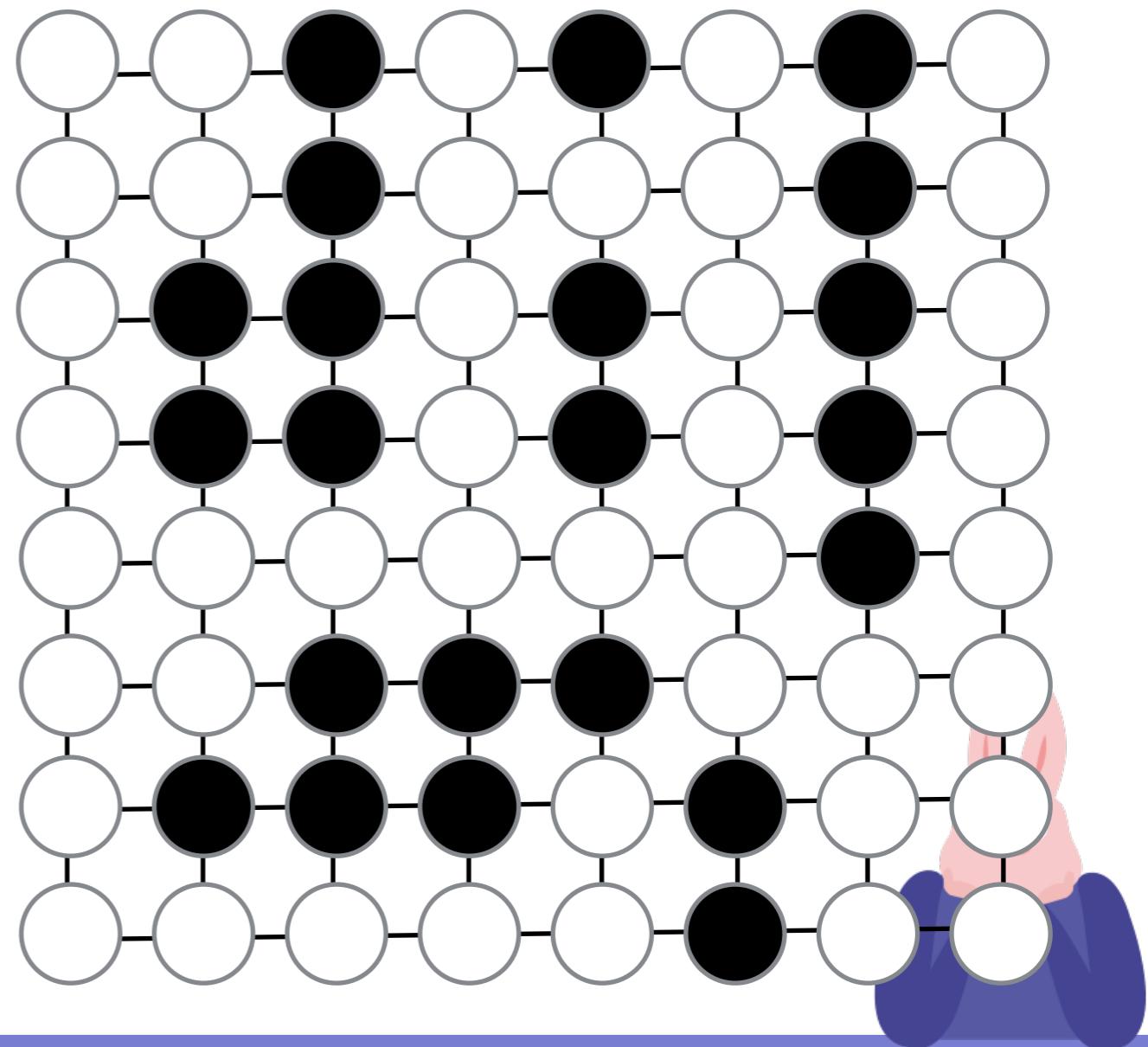
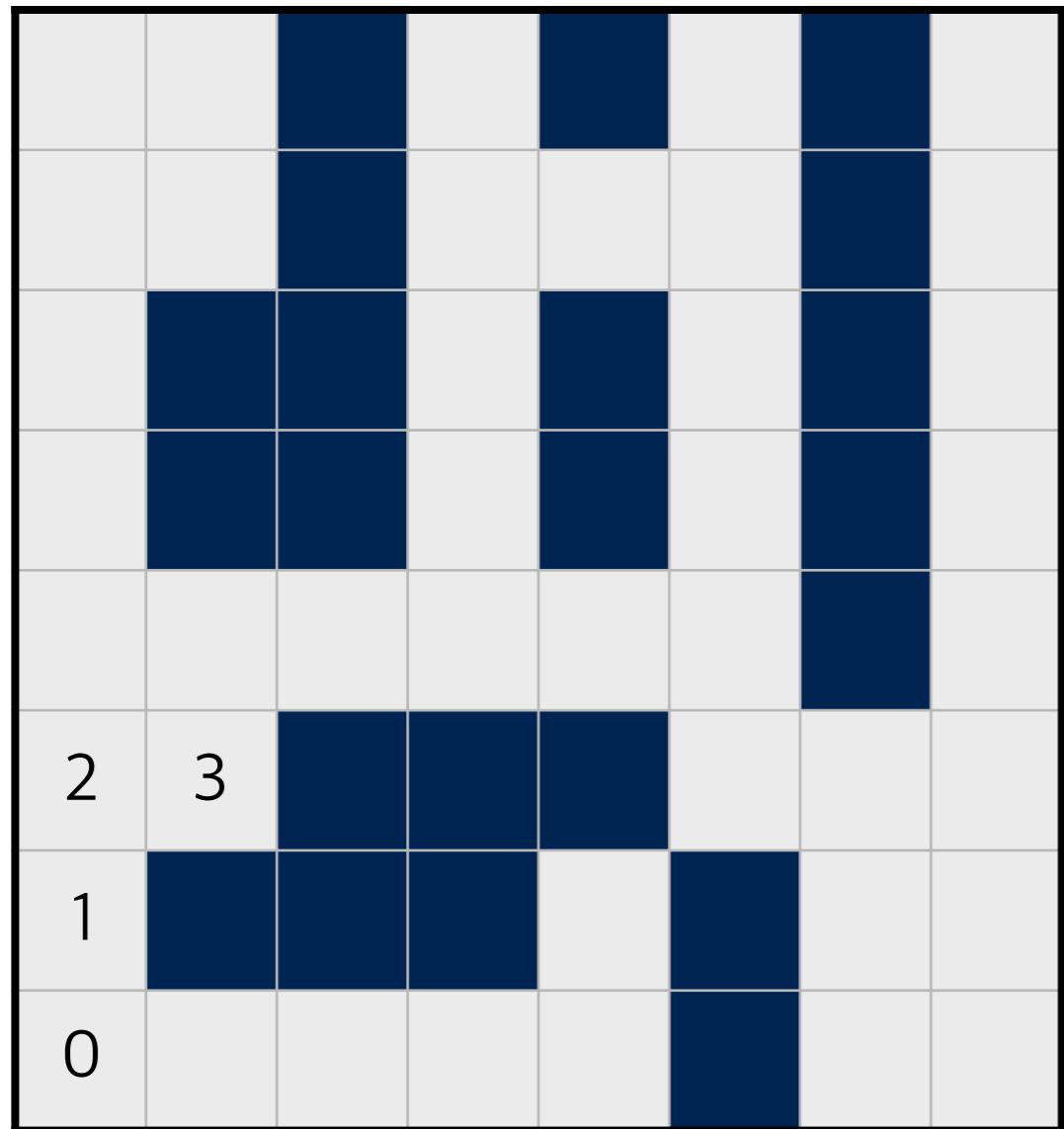
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



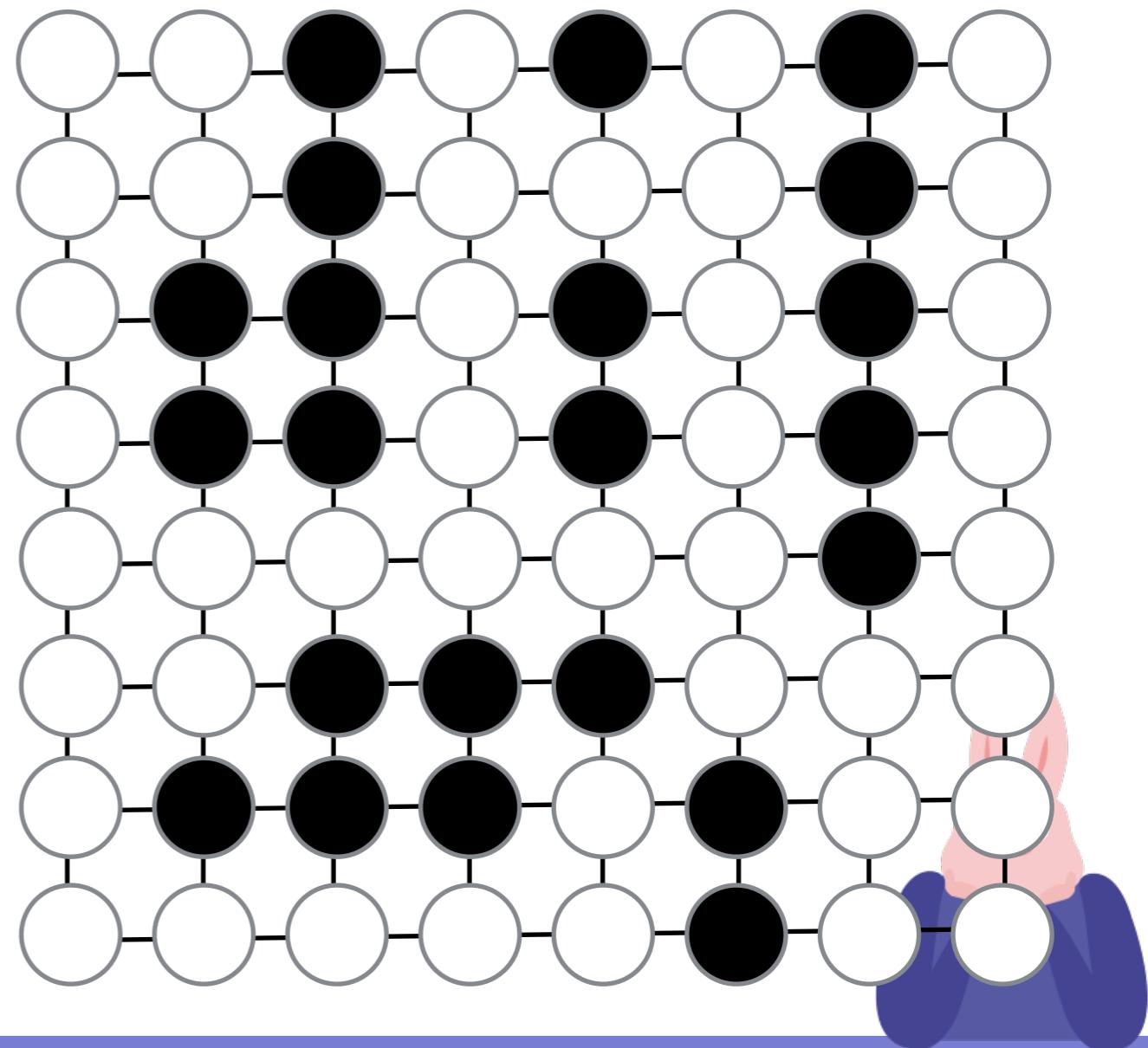
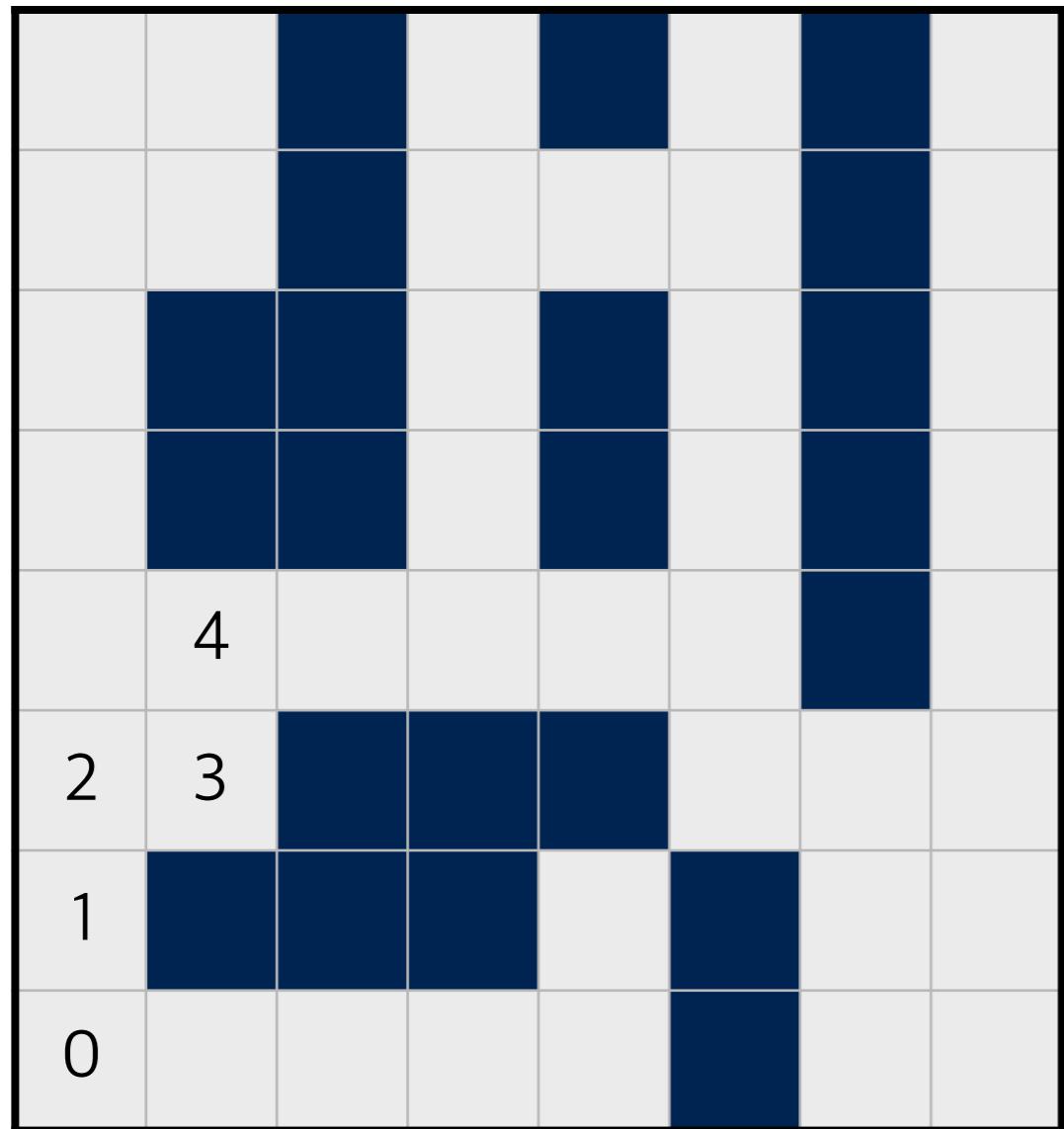
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



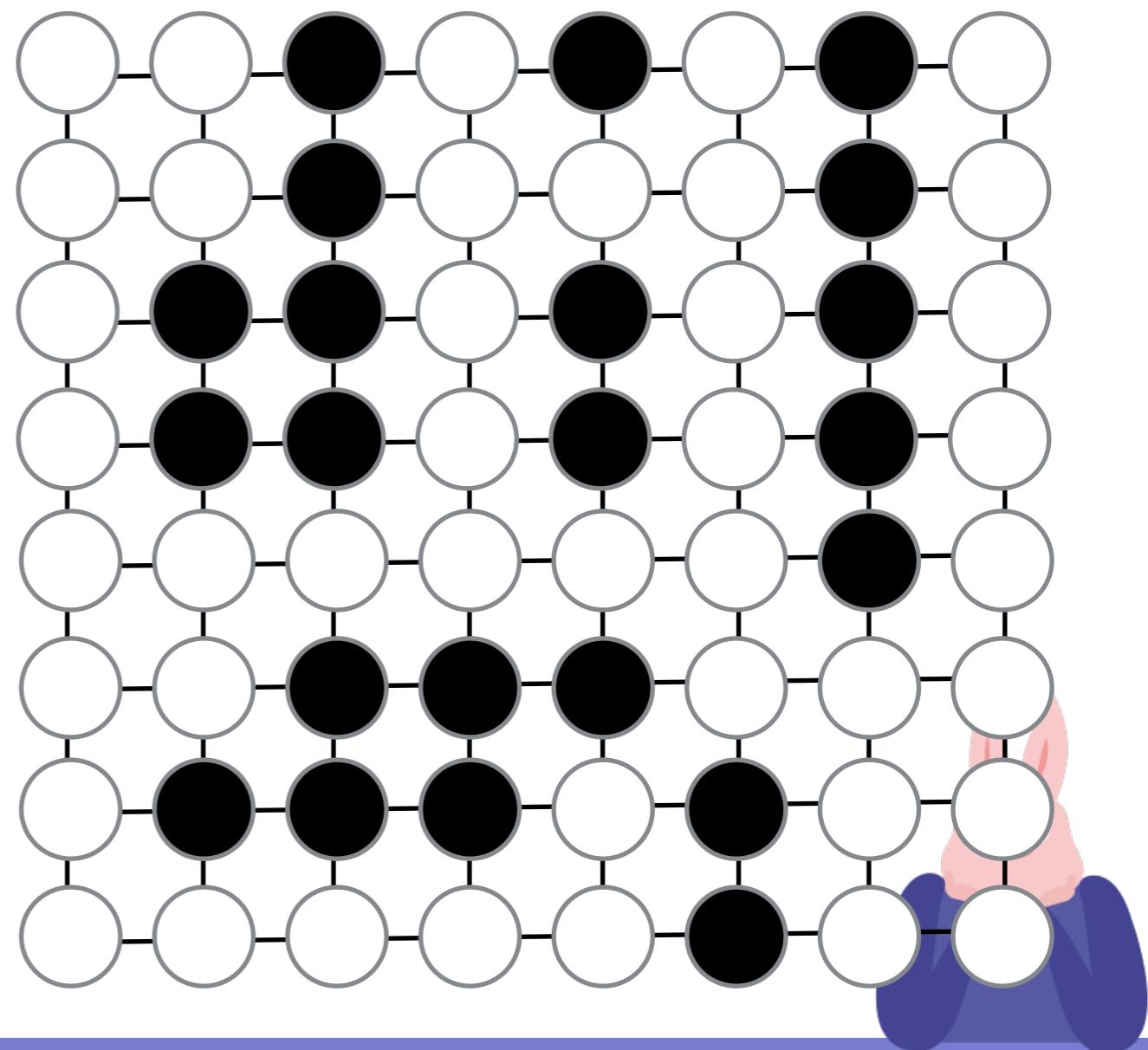
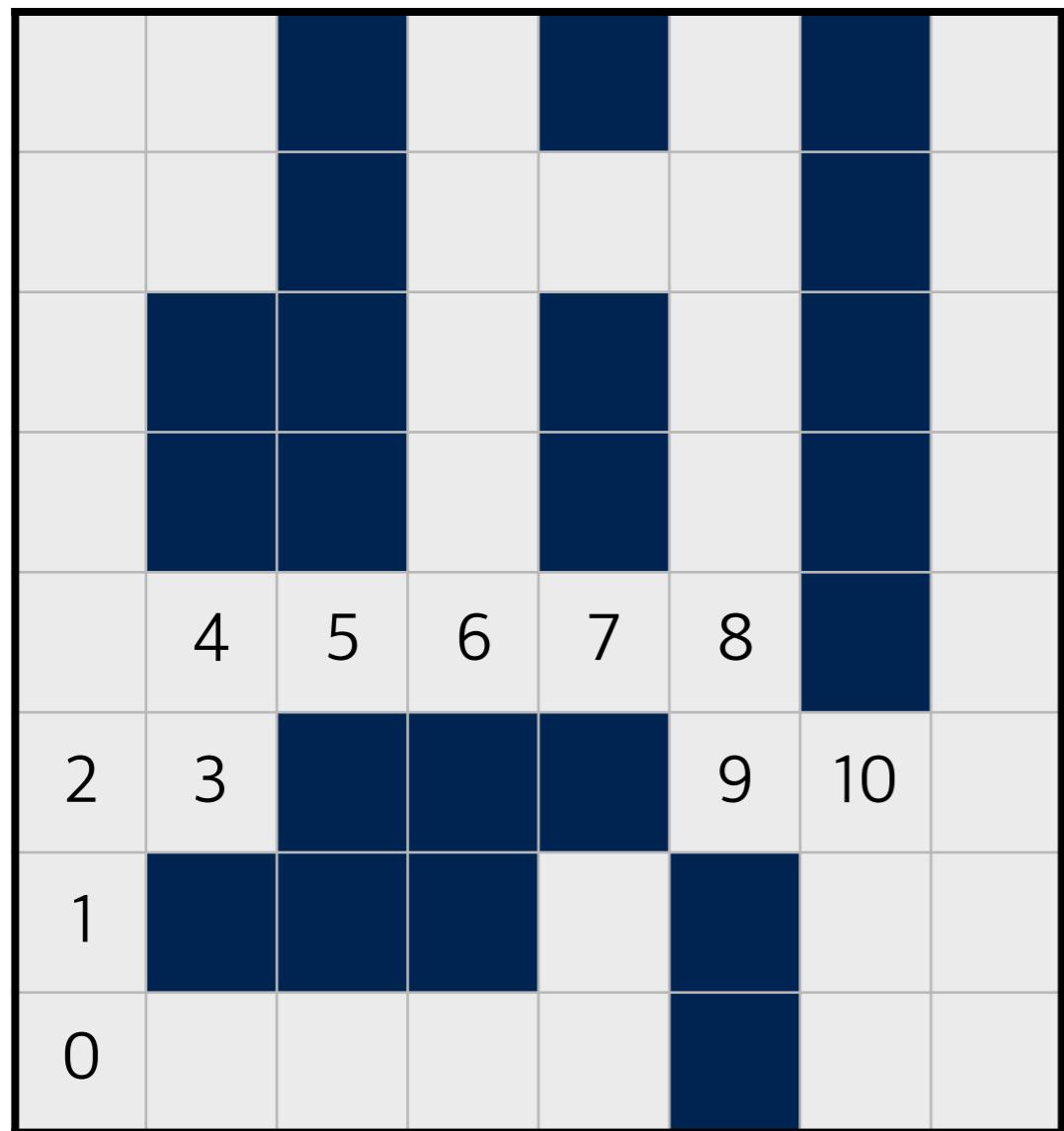
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



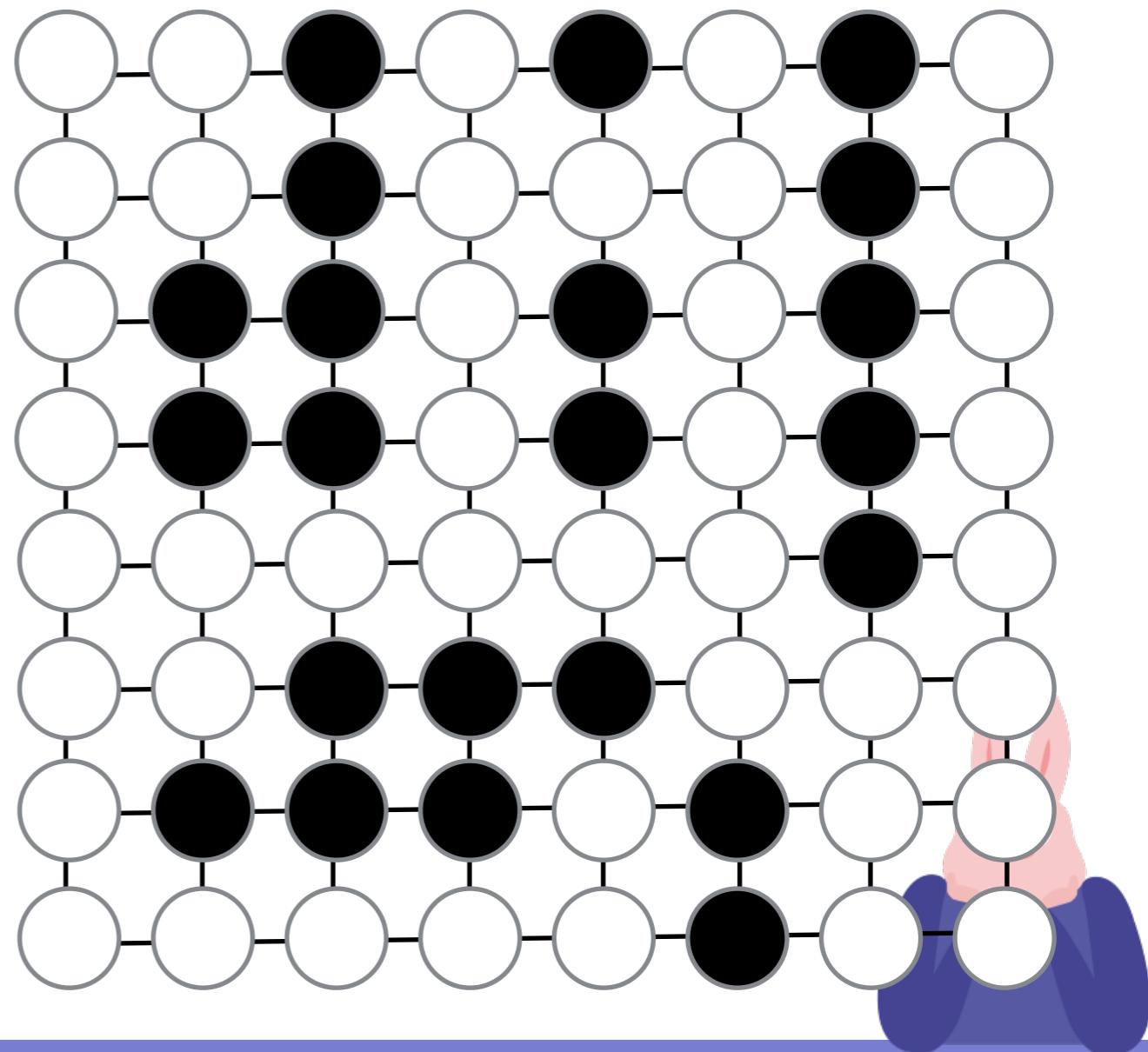
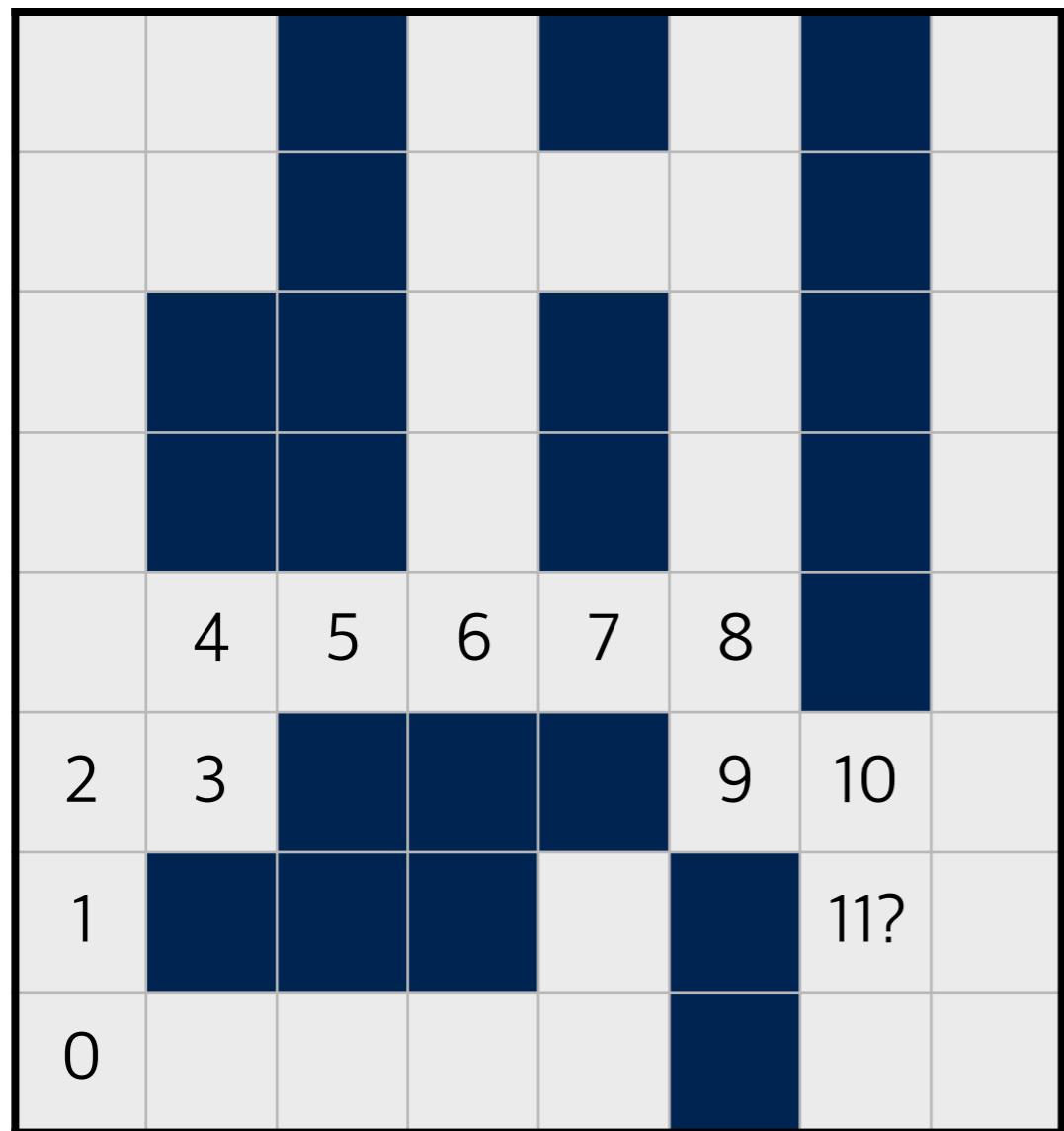
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



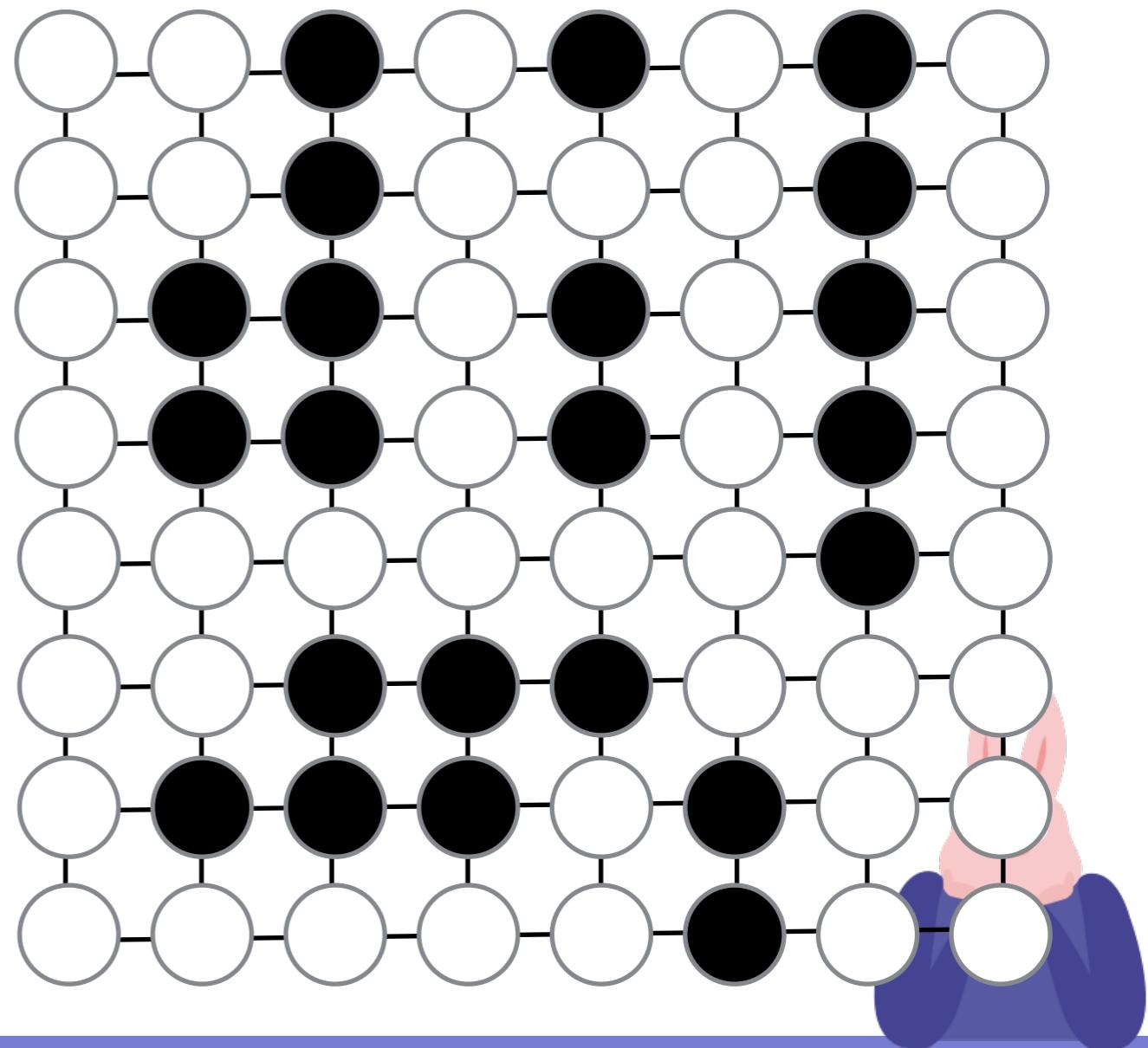
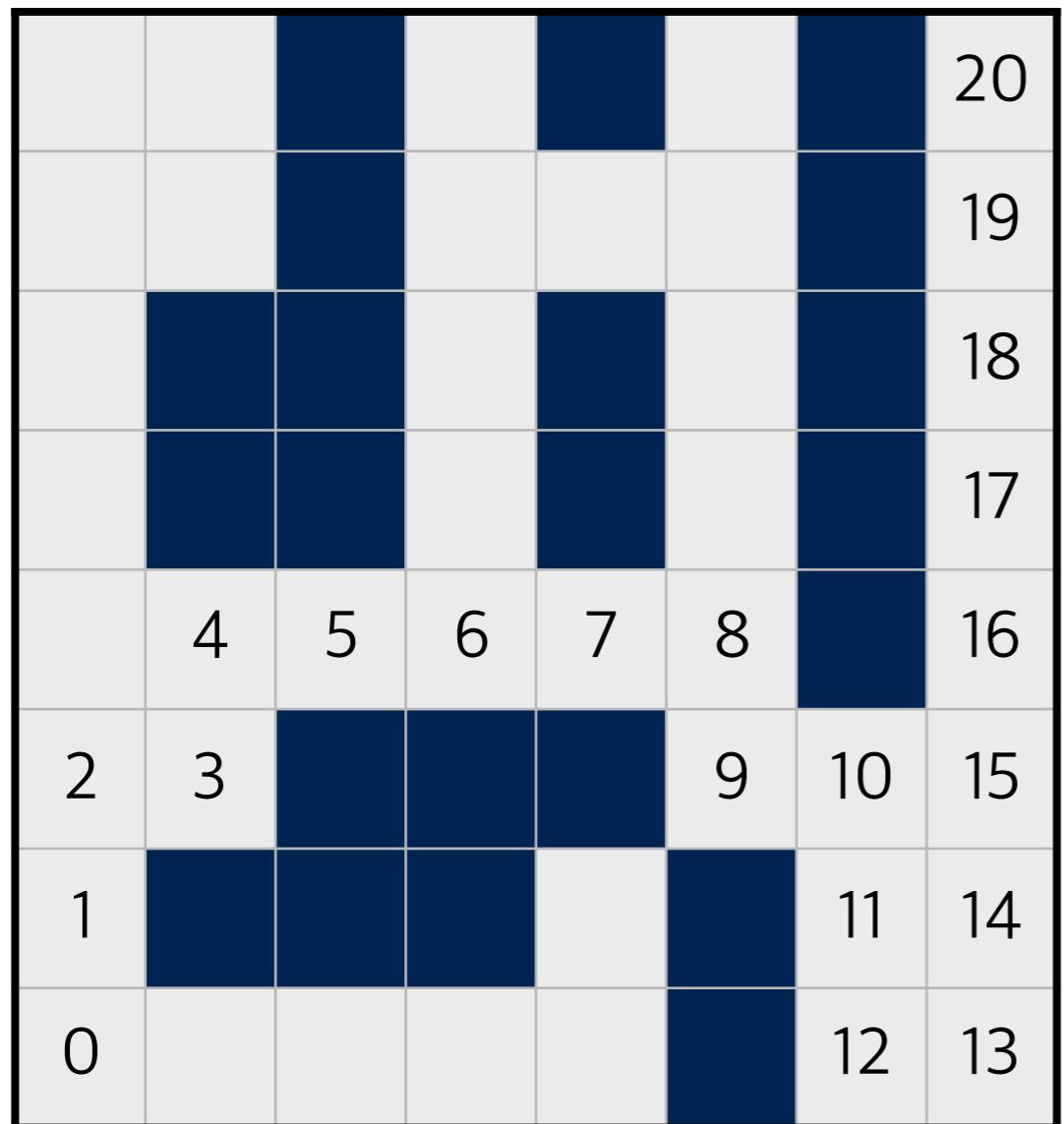
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



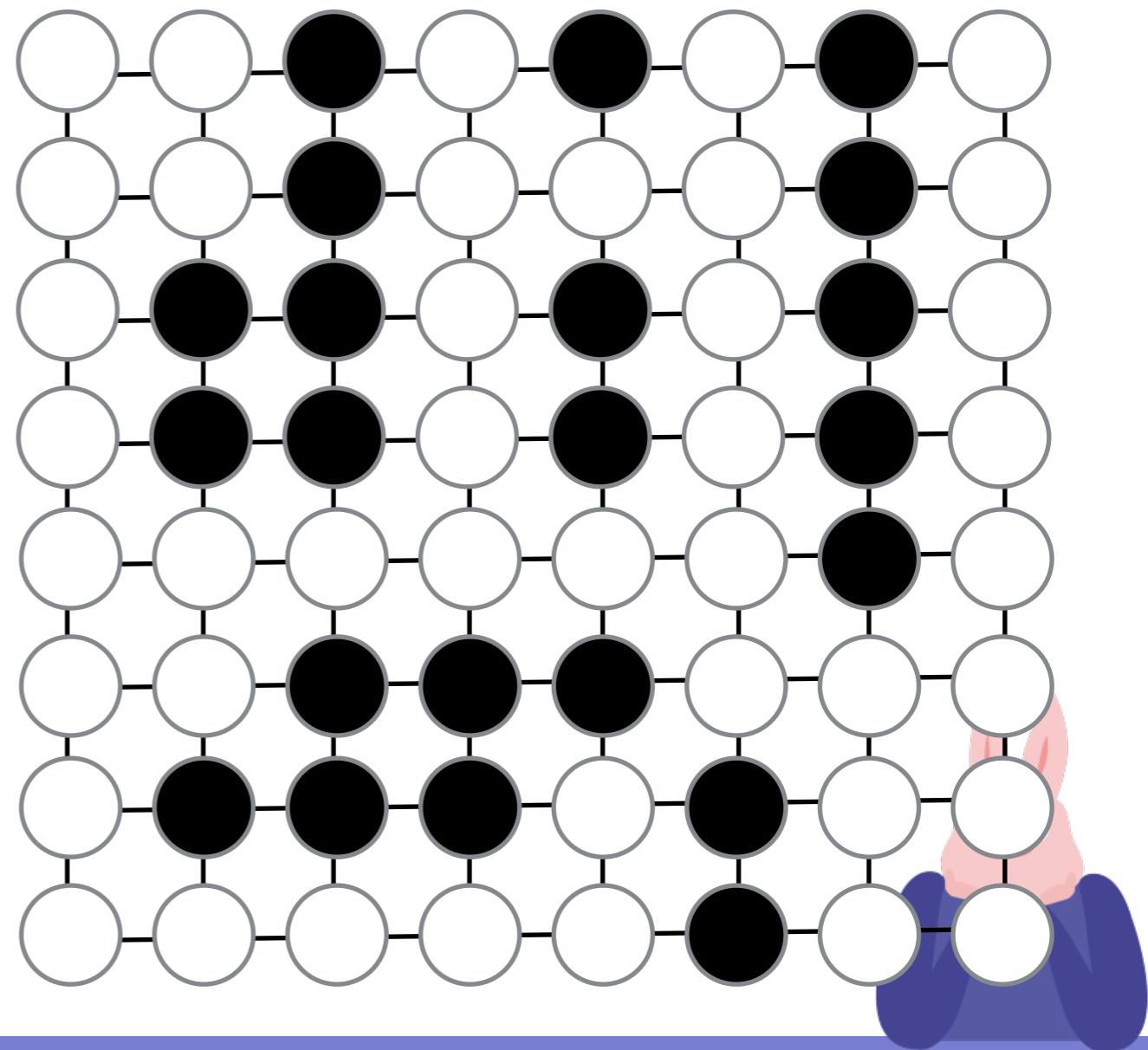
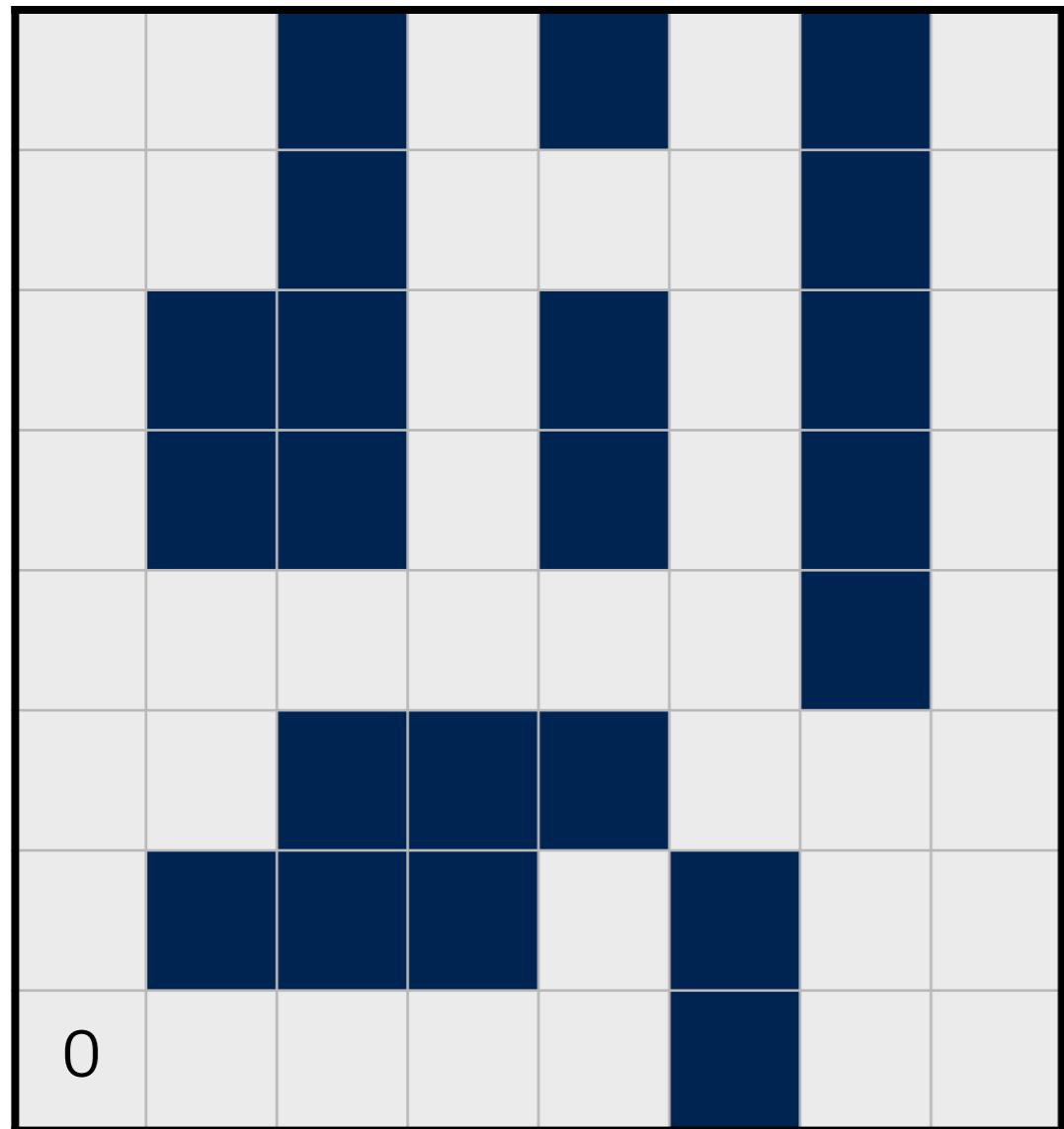
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



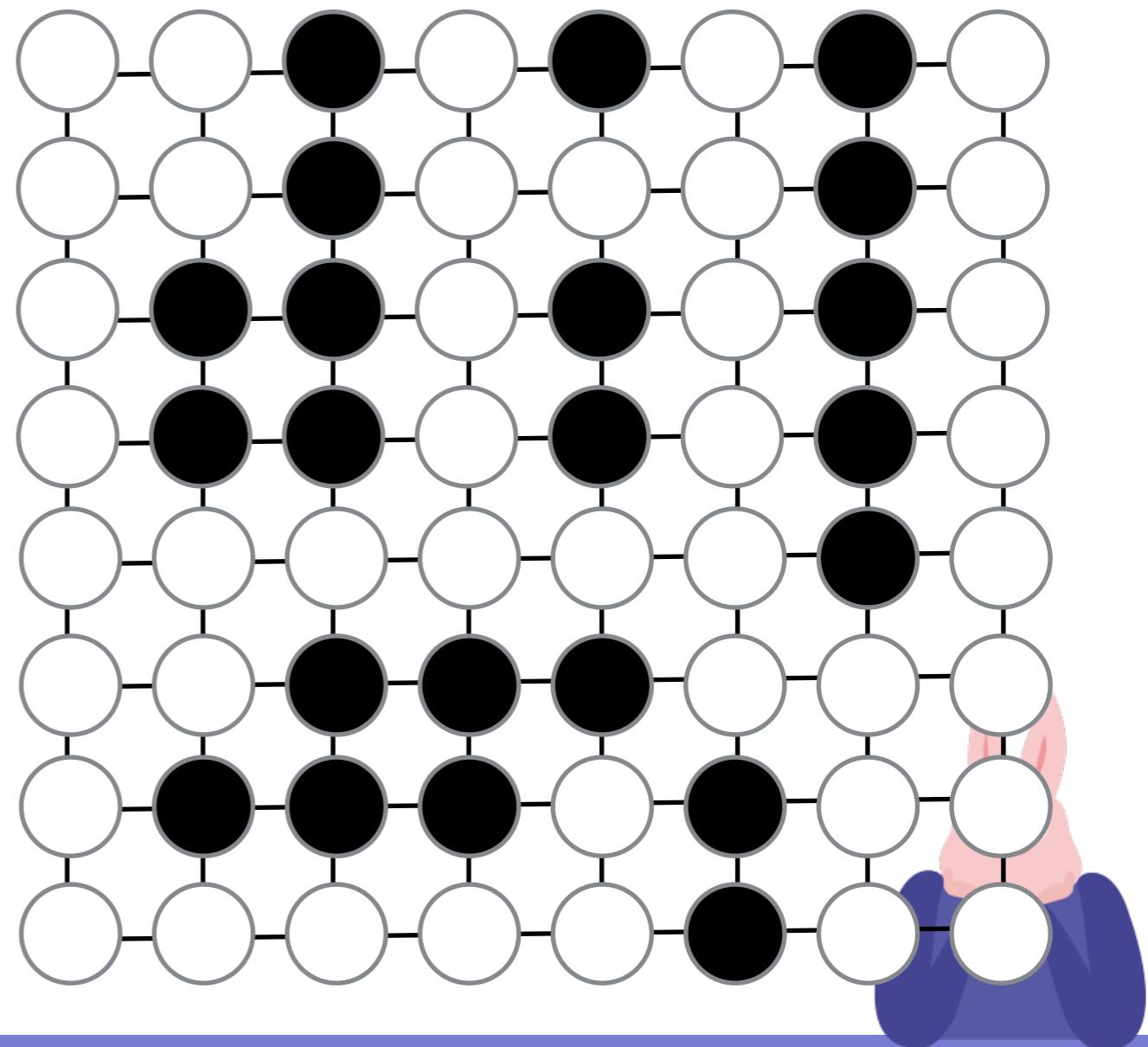
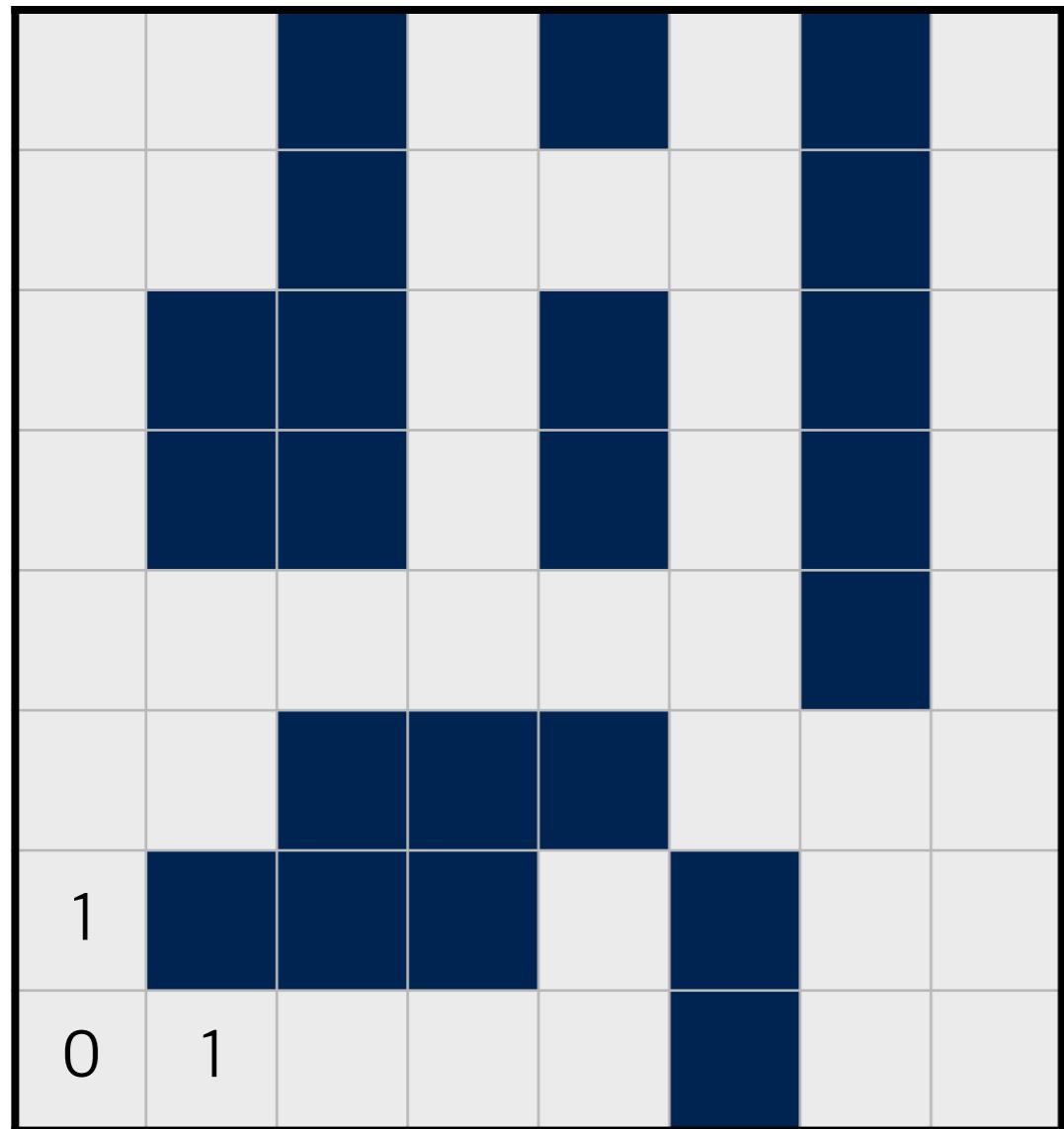
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



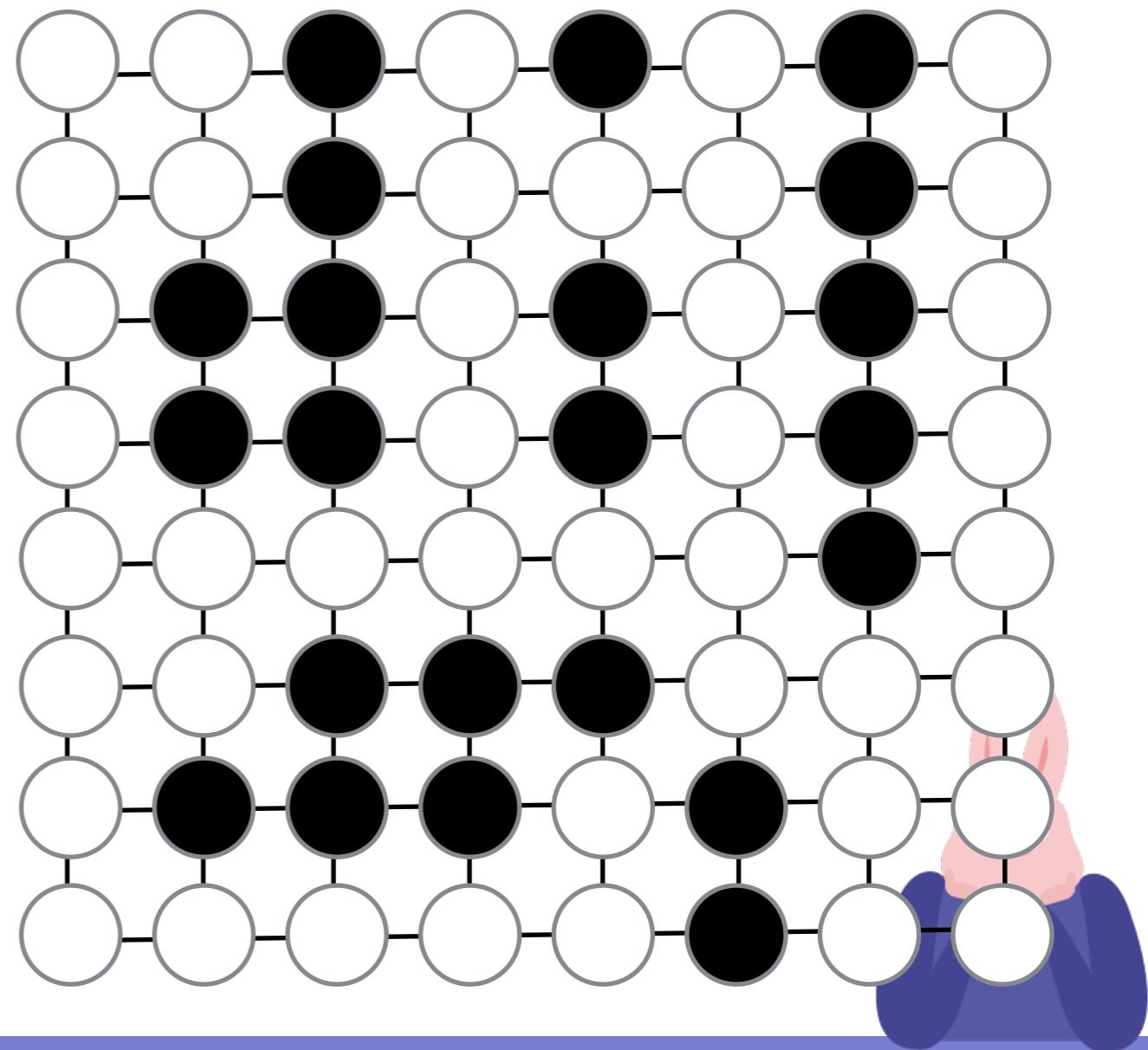
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



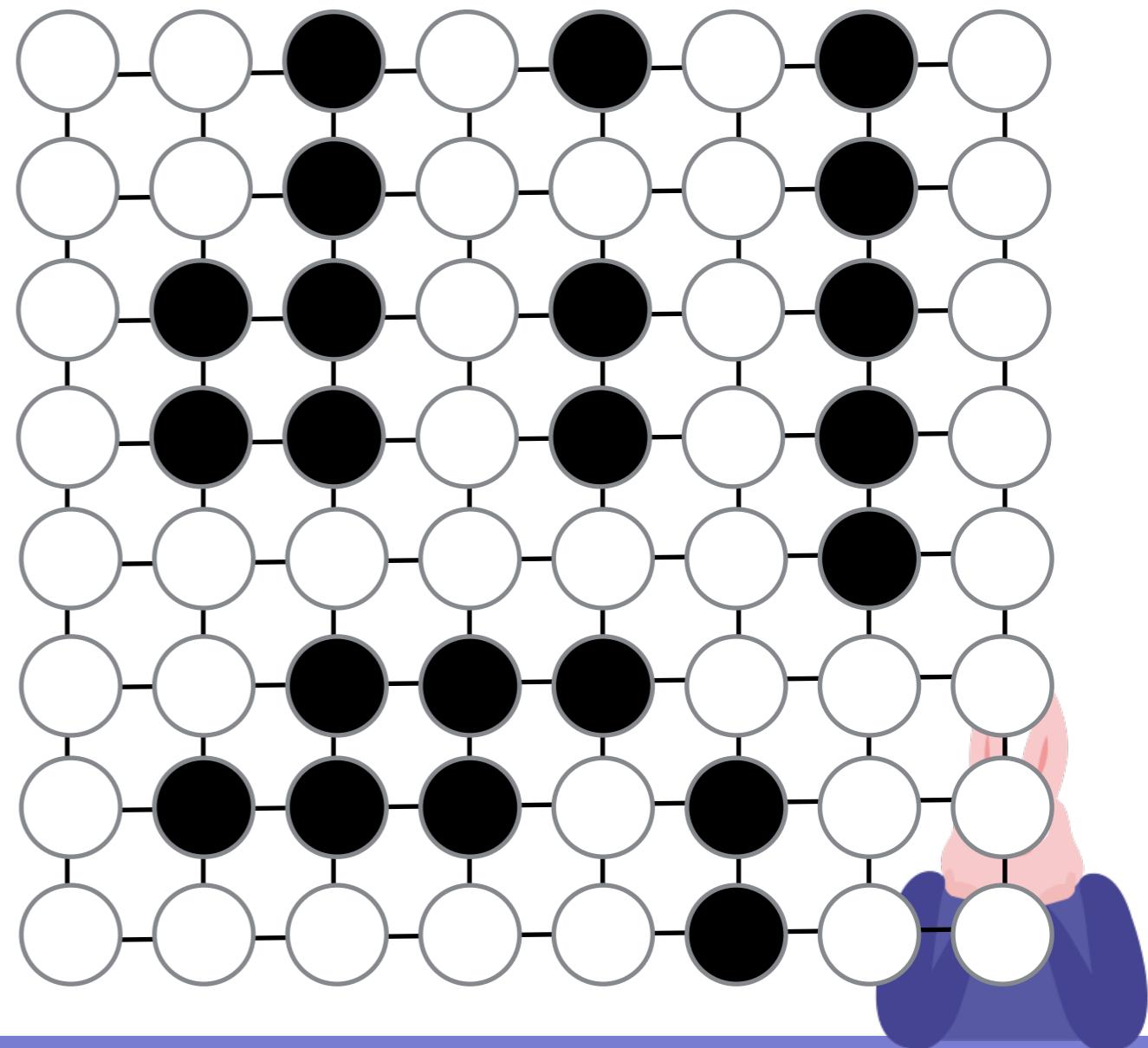
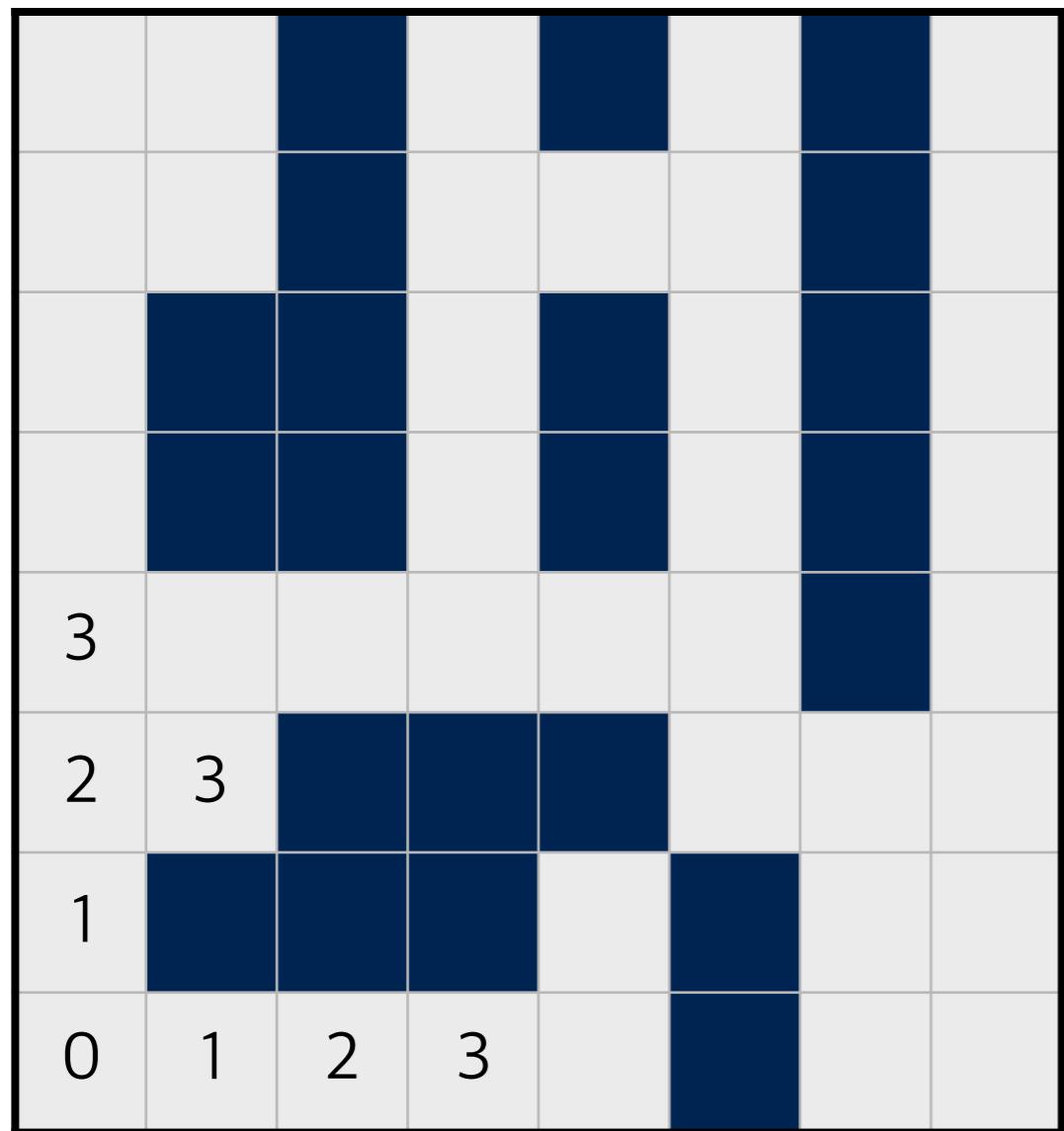
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



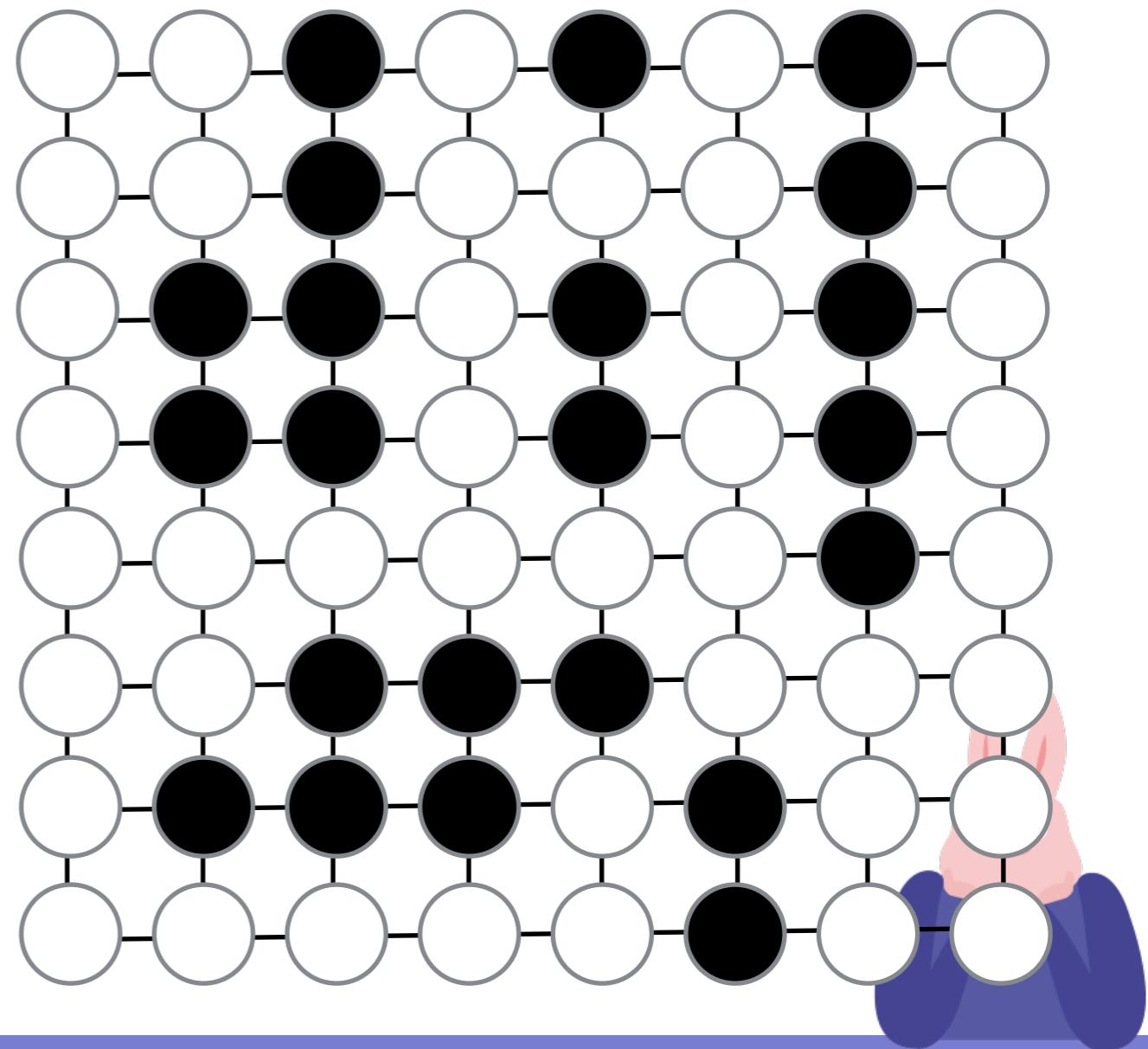
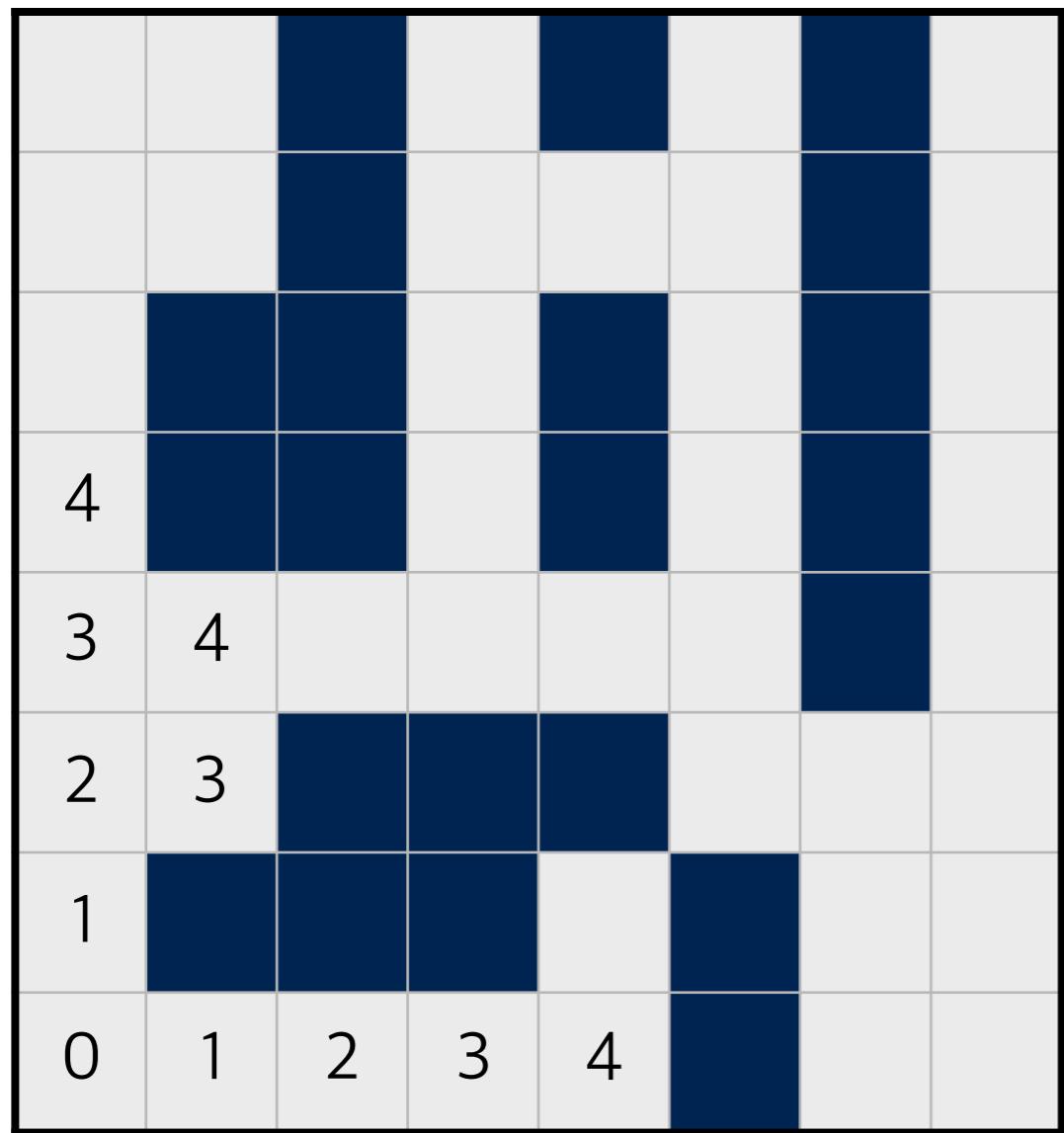
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



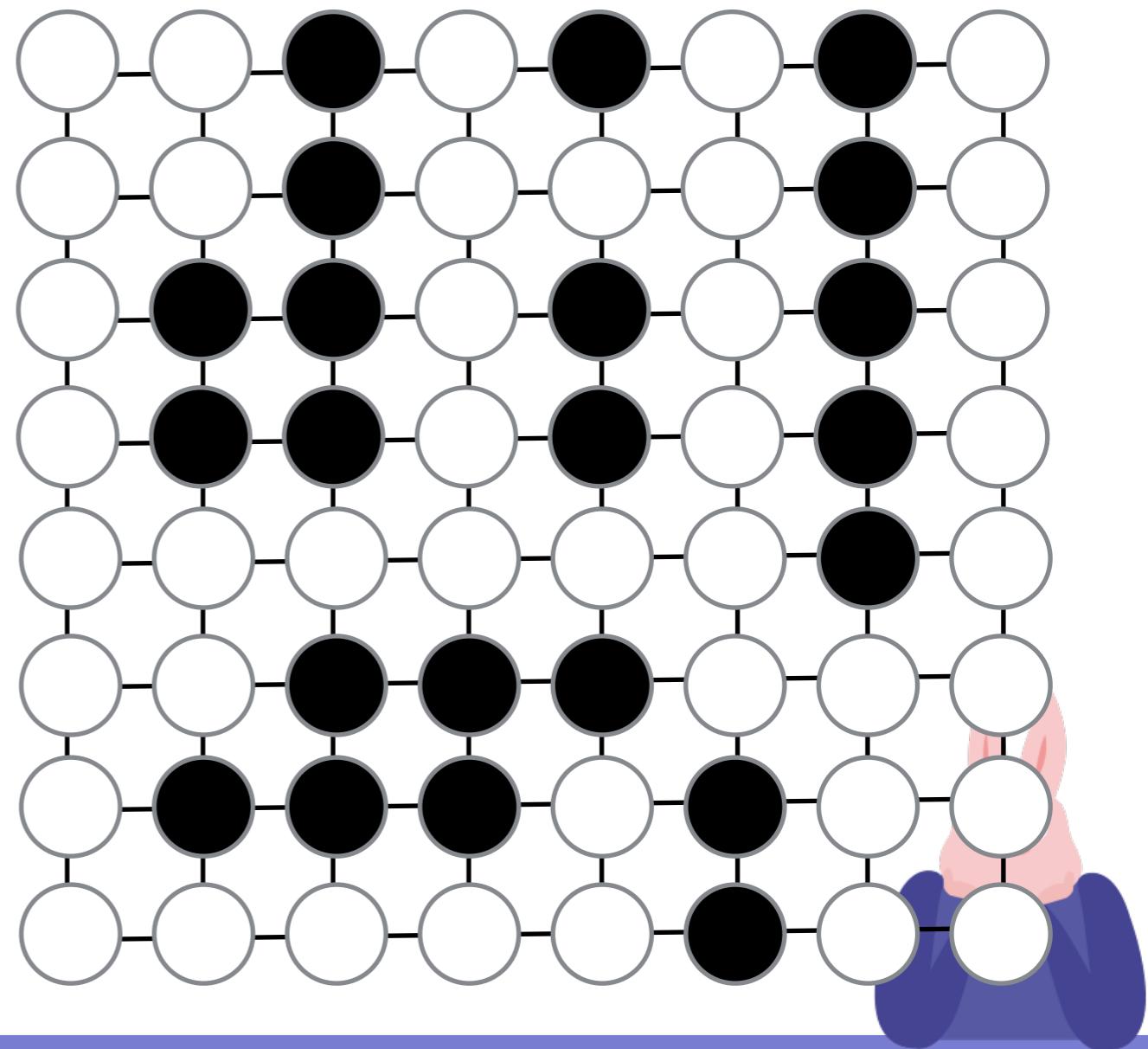
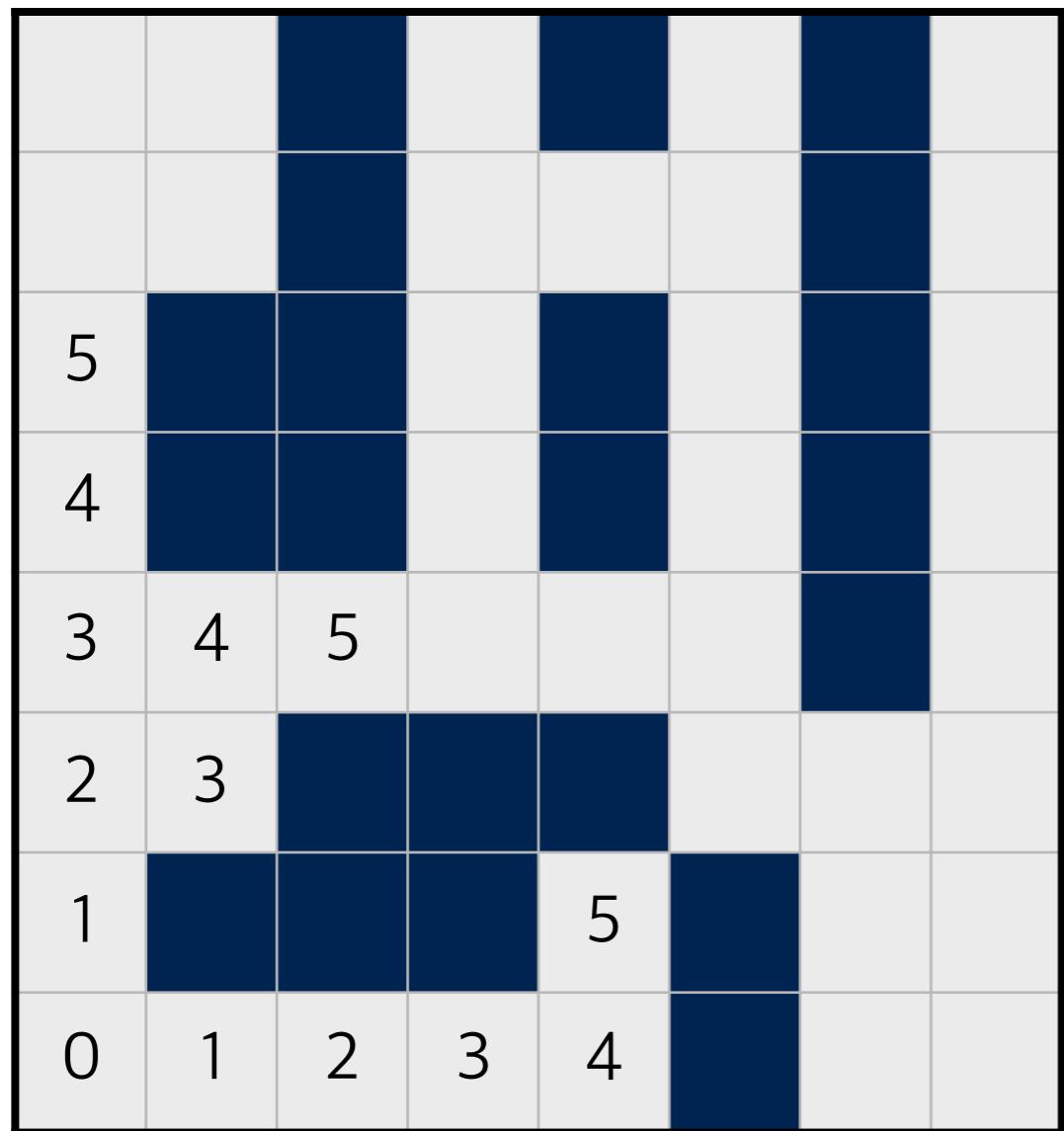
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



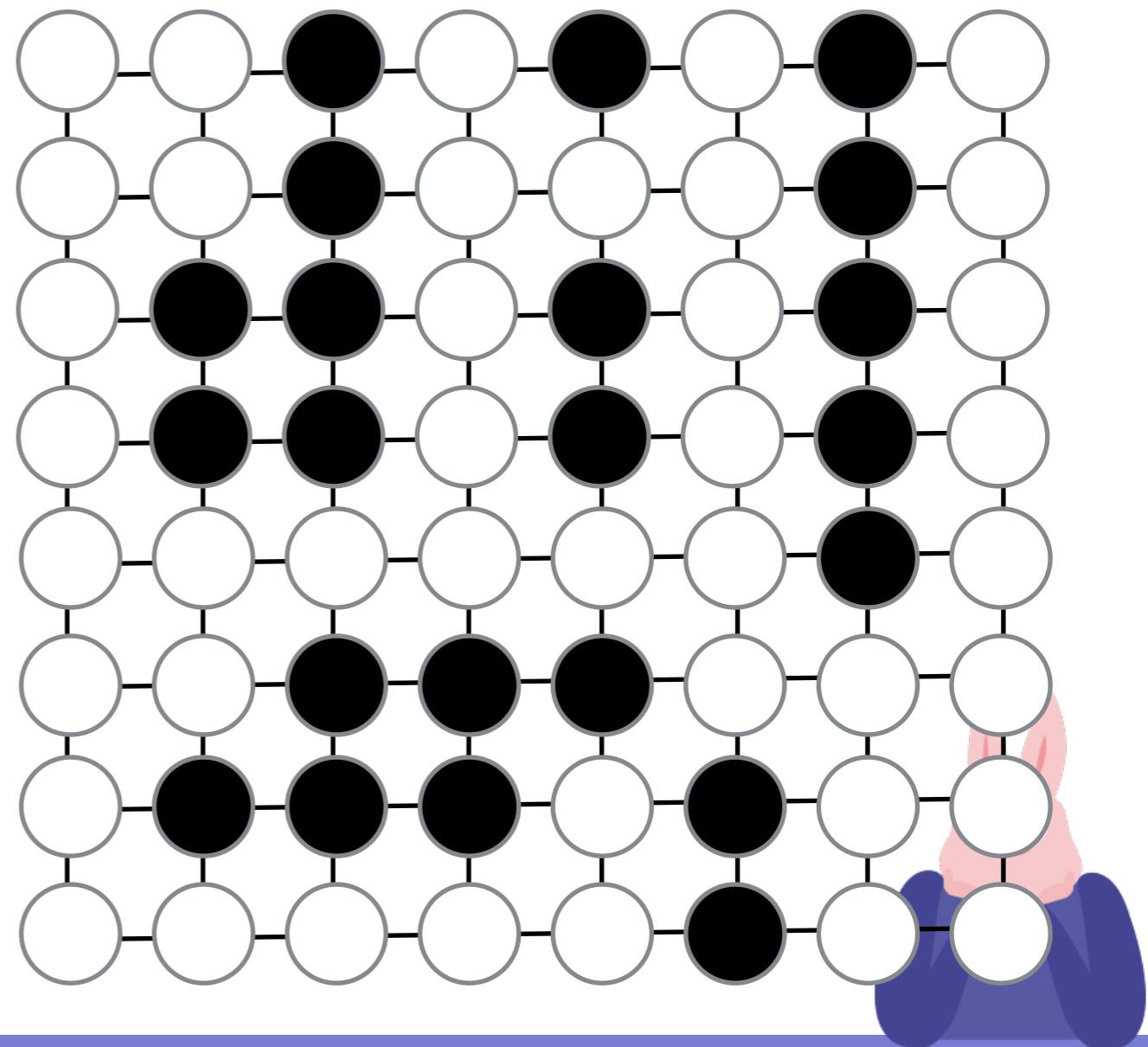
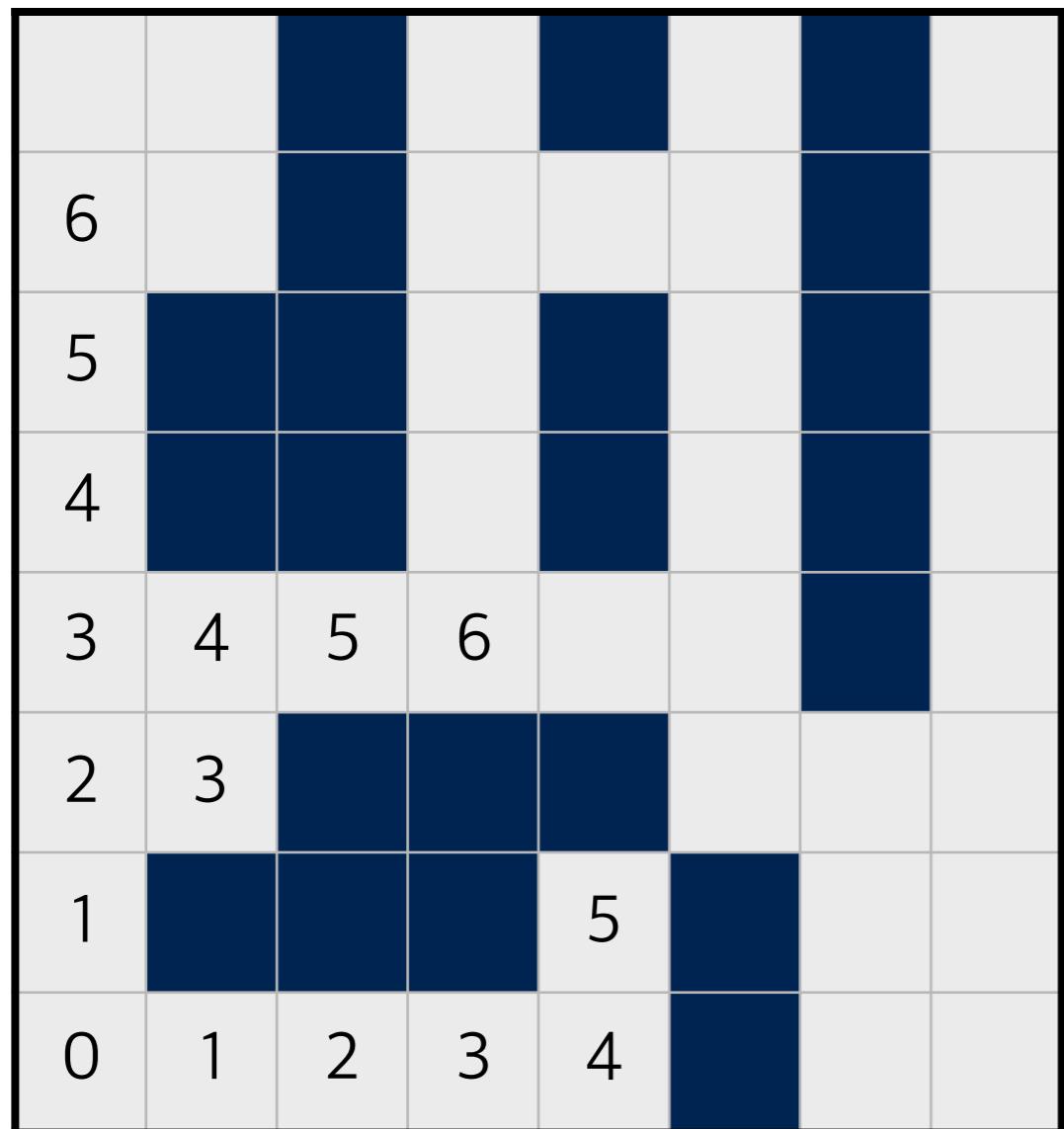
# DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



# DFS와 BFS의 응용 : 미로찾기

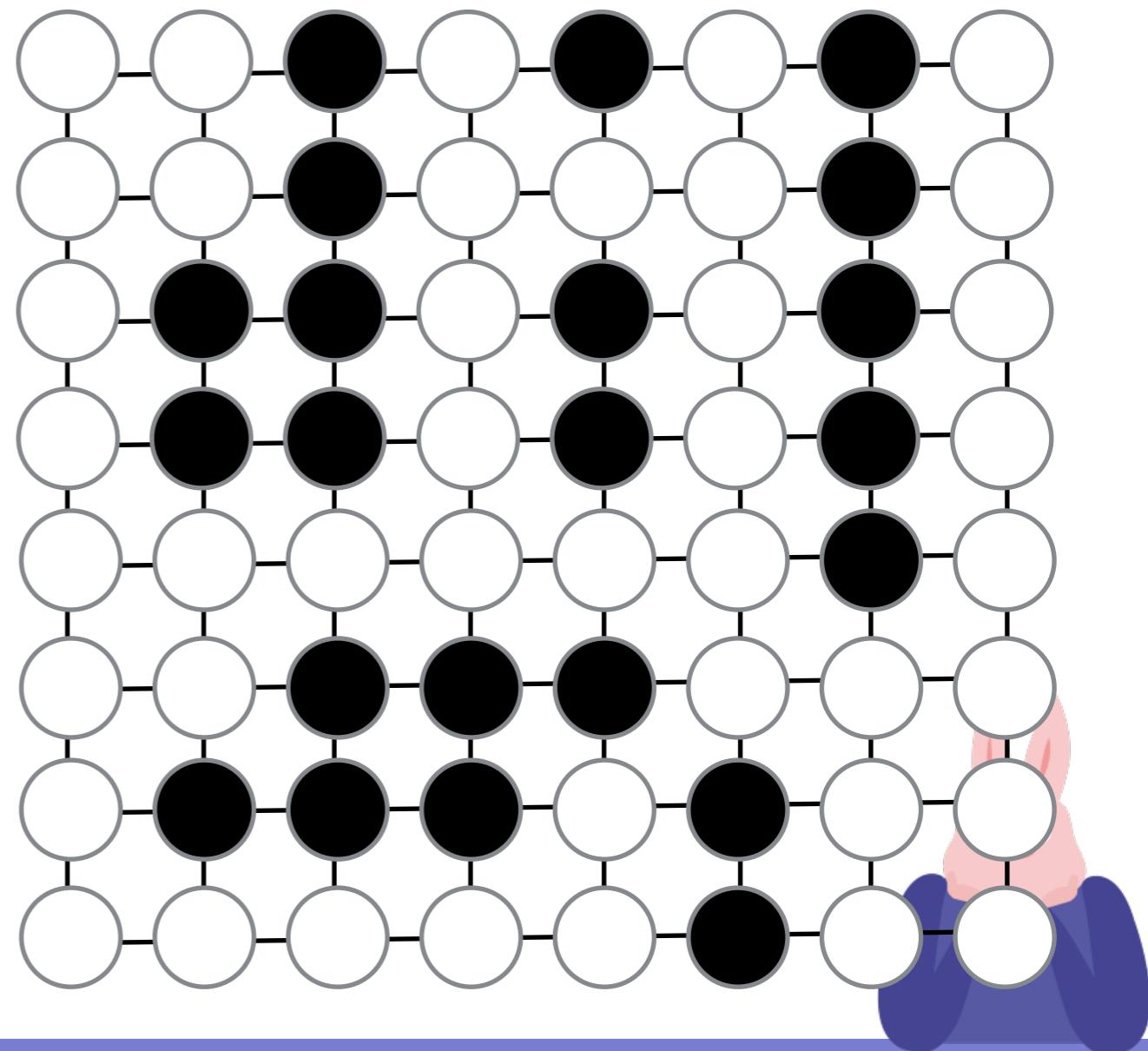
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장



# DFS와 BFS의 응용 : 미로찾기

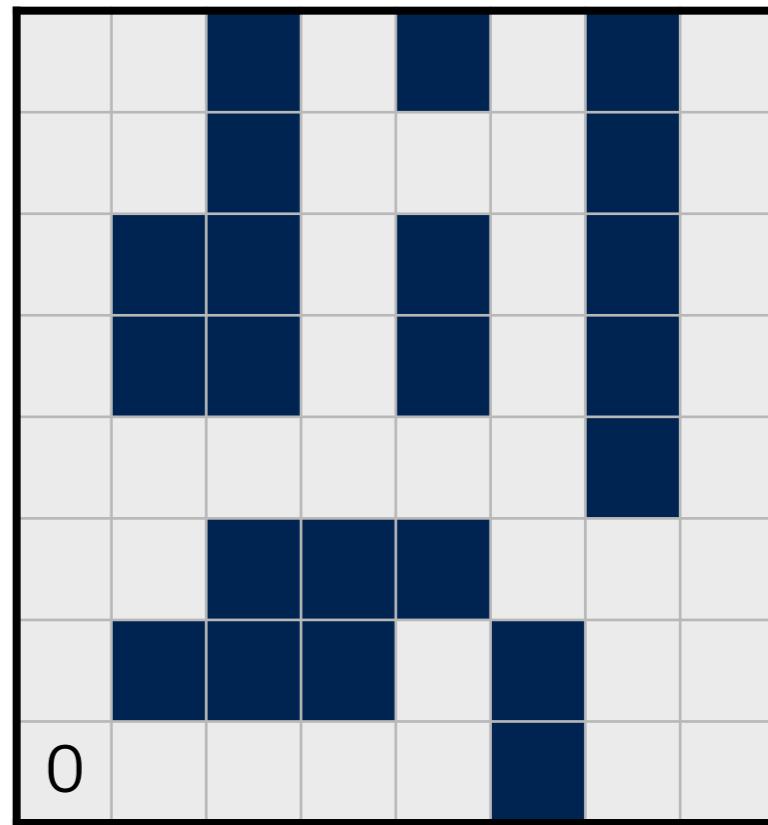
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

7	8		10		12		16
6	7		9	10	11		15
5			8		10		14
4			7		9		13
3	4	5	6	7	8		12
2	3				9	10	11
1				5		11	12
0	1	2	3	4		12	13



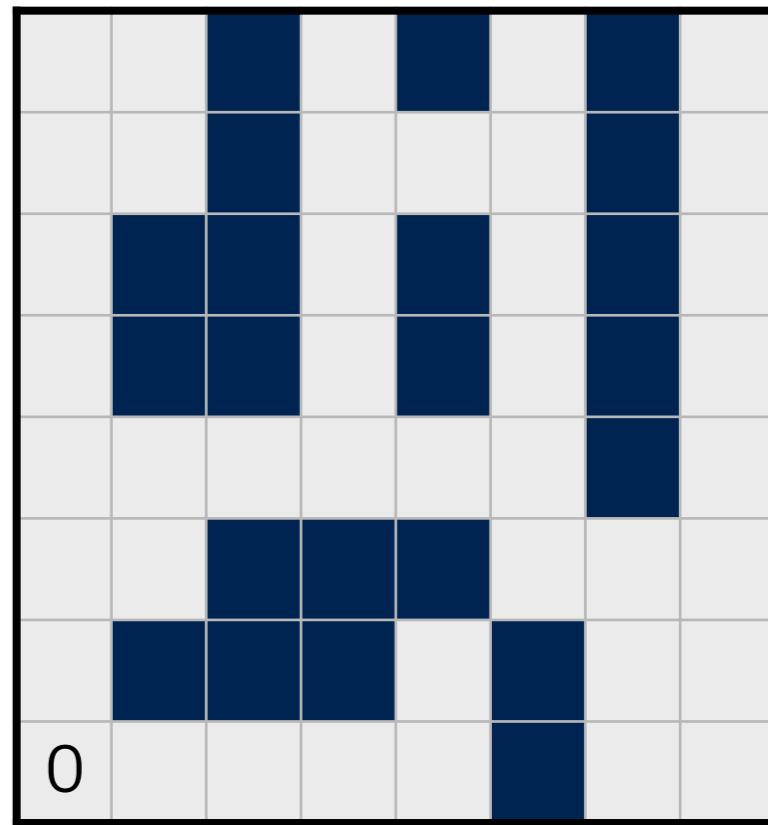
# BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 최단거리를 구할 수 있다



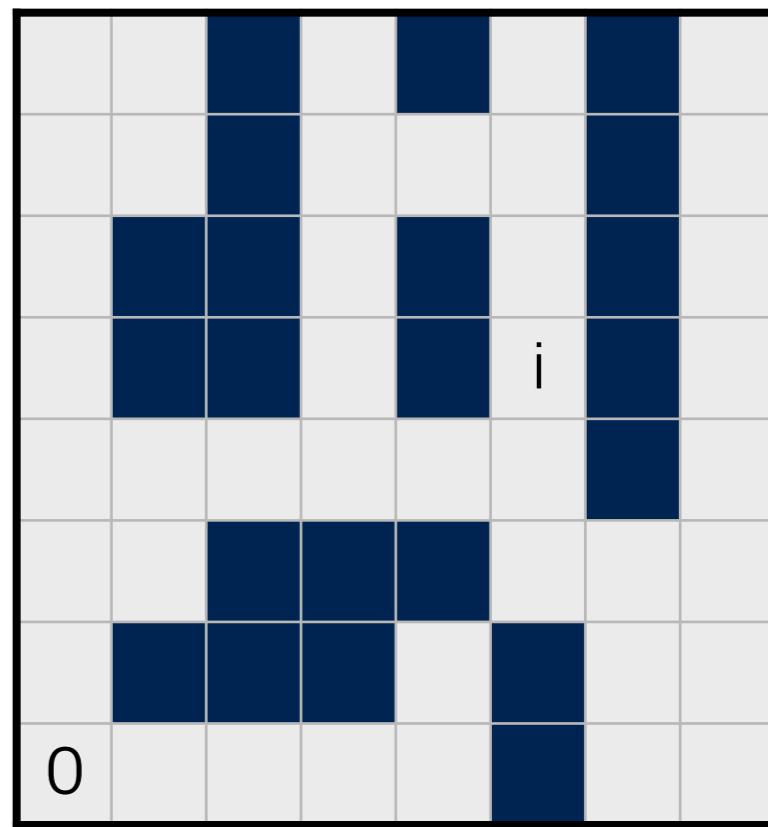
# BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 최단거리를 구할 수 있다
- 증명 (귀류법)



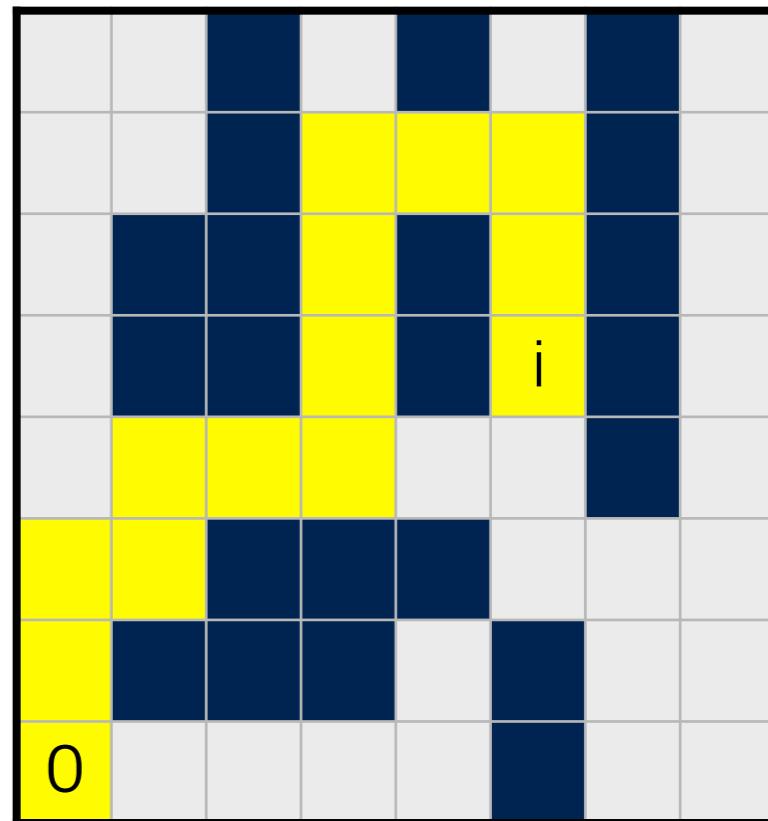
# BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 최단거리를 구할 수 있다
- 증명 (귀류법)
  - 만약 최단거리를 구하지 못했다고 가정하자  
그리고, 최단거리를 구하지 못한 칸을  $i$ 라 하자



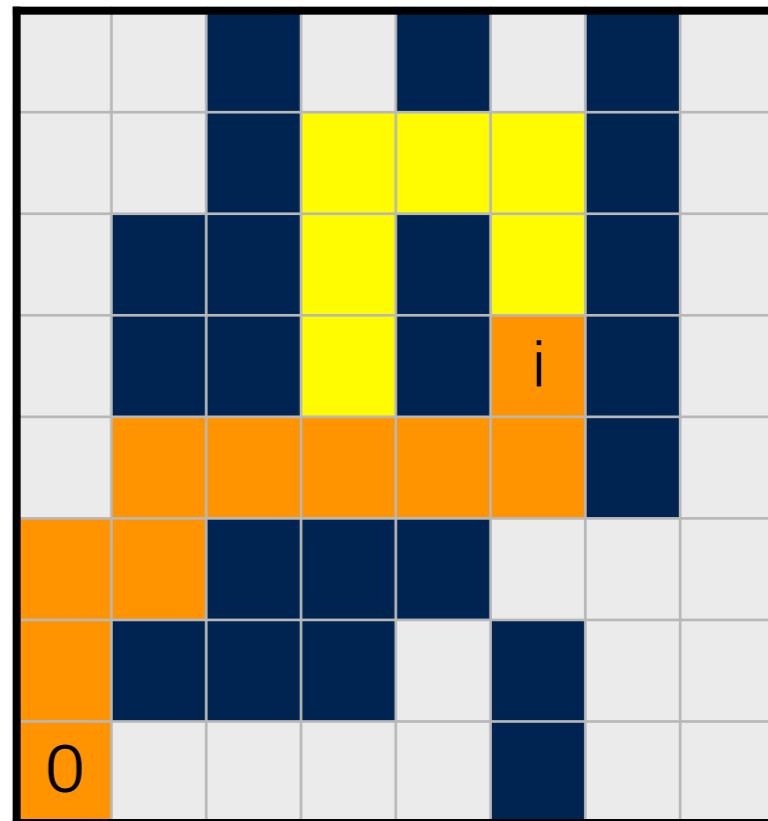
# BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 최단거리를 구할 수 있다
- 증명 (귀류법)
  - 만약 최단거리를 구하지 못했다고 가정하자  
그리고, 최단거리를 구하지 못한 칸을  $i$ 라 하자
  - 현재  $i$  까지 따라온 경로를  $P(i)$ 라 하자  
최단거리를 구하지 못했으므로,  $P(i)$ 는 최단경로가 아니다.



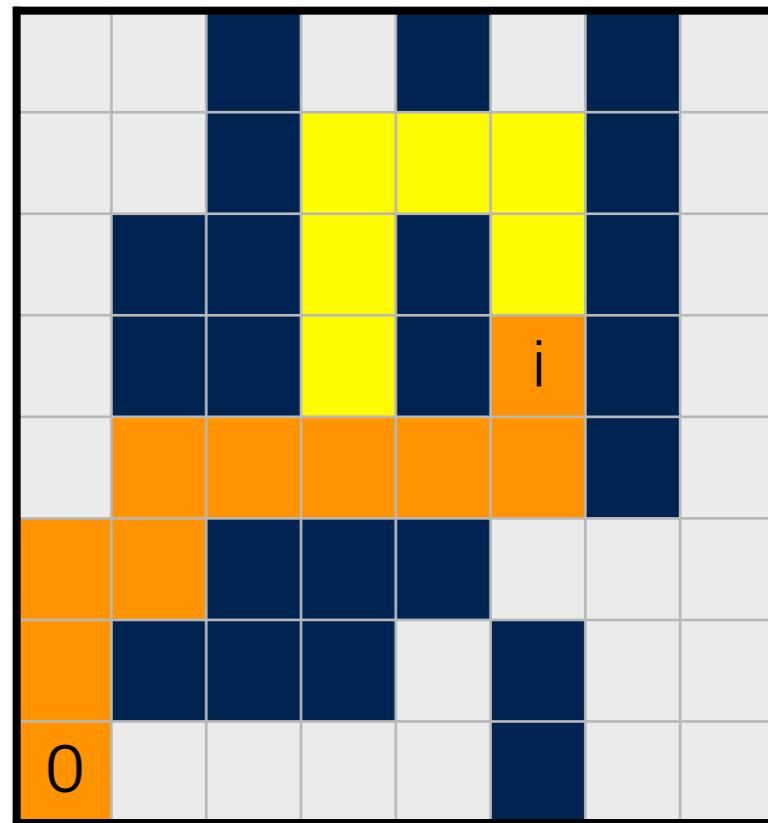
# BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 최단거리를 구할 수 있다
- 증명 (귀류법)
  - 만약 최단거리를 구하지 못했다고 가정하자  
그리고, 최단거리를 구하지 못한 칸을  $i$ 라 하자
  - 현재  $i$  까지 따라온 경로를  $P(i)$ 라 하자  
최단거리를 구하지 못했으므로,  $P(i)$ 는 최단경로가 아니다.
  - 구하지 못한 최단경로를  $S(i)$ 라 하자



# BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 최단거리를 구할 수 있다
  - 증명 (귀류법)
    - 만약 최단거리를 구하지 못했다고 가정하자  
그리고, 최단거리를 구하지 못한 칸을  $i$ 라 하자
    - 현재  $i$  까지 따라온 경로를  $P(i)$ 라 하자  
최단거리를 구하지 못했으므로,  $P(i)$ 는 최단경로가 아니다.
    - 구하지 못한 최단경로를  $S(i)$ 라 하자
    - BFS의 알고리즘에 따르면,  $P(i)$ 를  $S(i)$ 보다 먼저 구할 수 없다  
따라서 가정은 모순이다



# [활동문제 5] 미로찾기

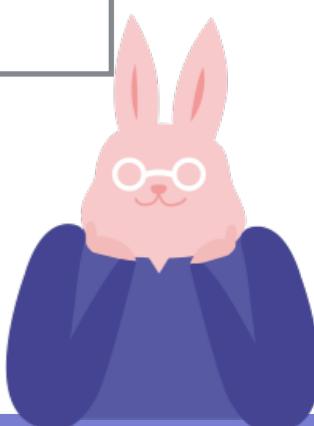
- 시작점에서 출발하여 끝점에 도달하기 위한 최단경로를 출력  
(시작점은 항상 왼쪽 아래, 끝점은 항상 오른쪽 위)

입력의 예

```
5 6
0 1 0 1 1 0
0 1 0 0 1 0
0 0 0 0 1 0
0 1 1 0 0 0
0 1 0 0 0 0
```

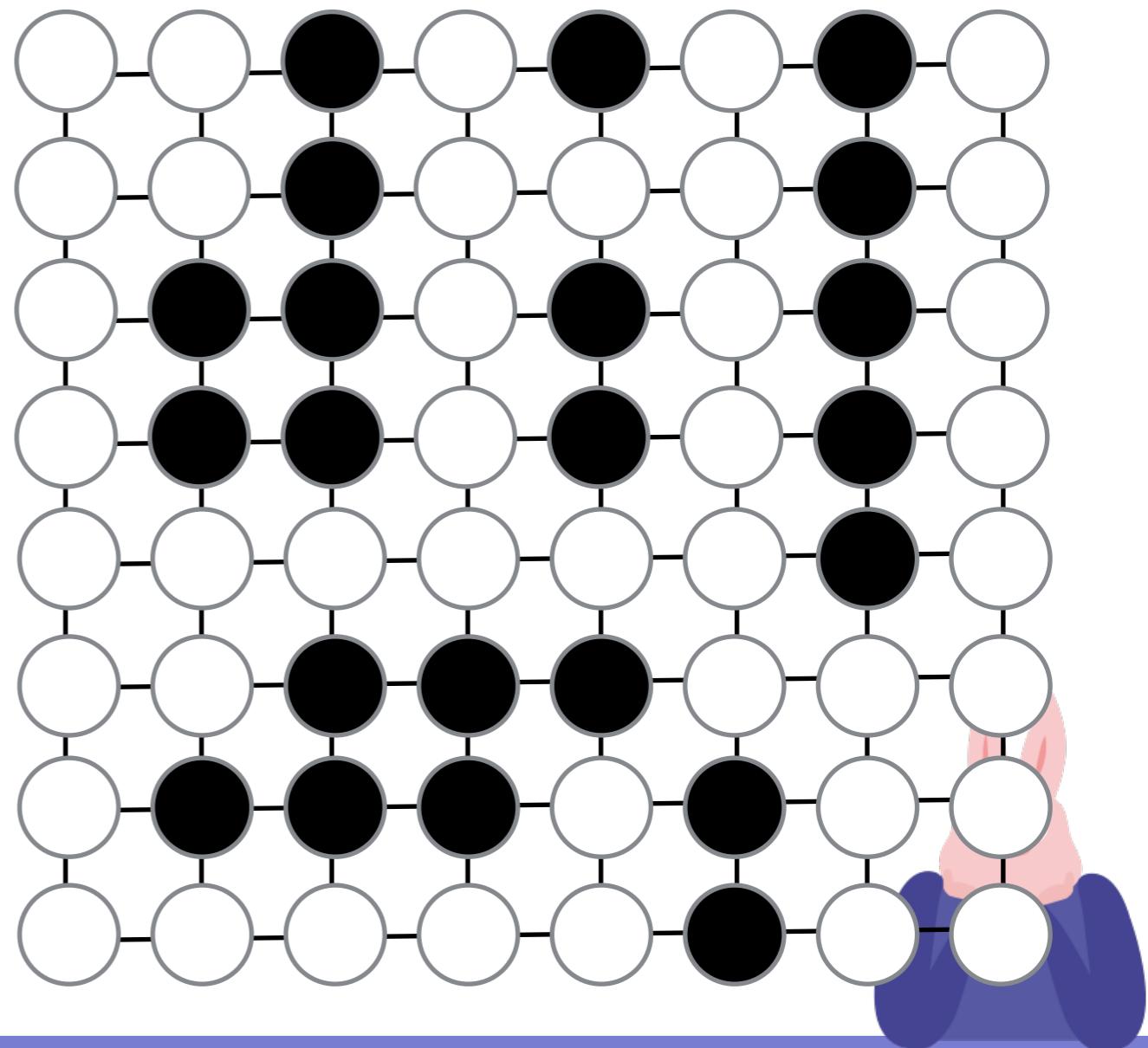
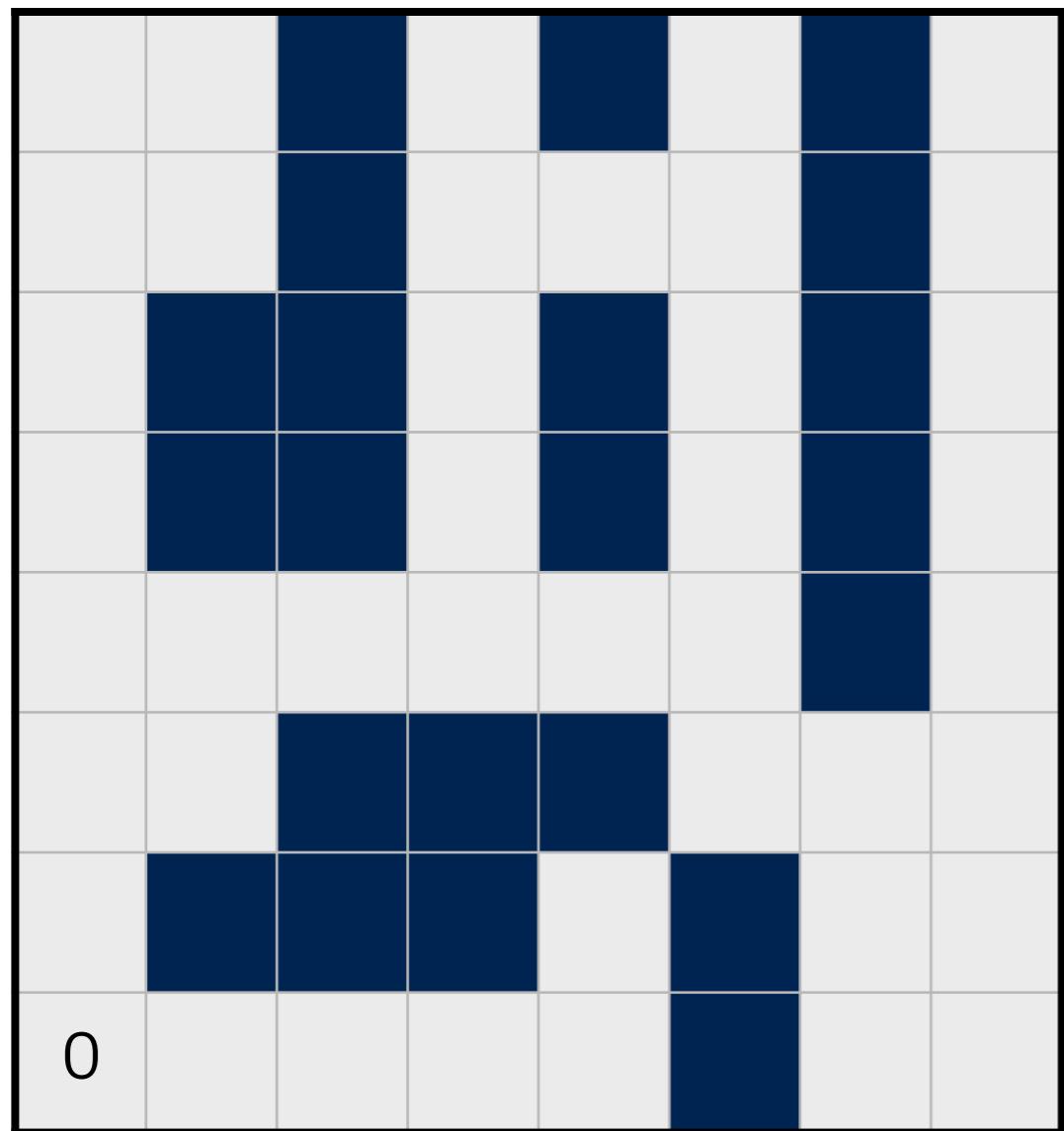
출력의 예

```
11
```



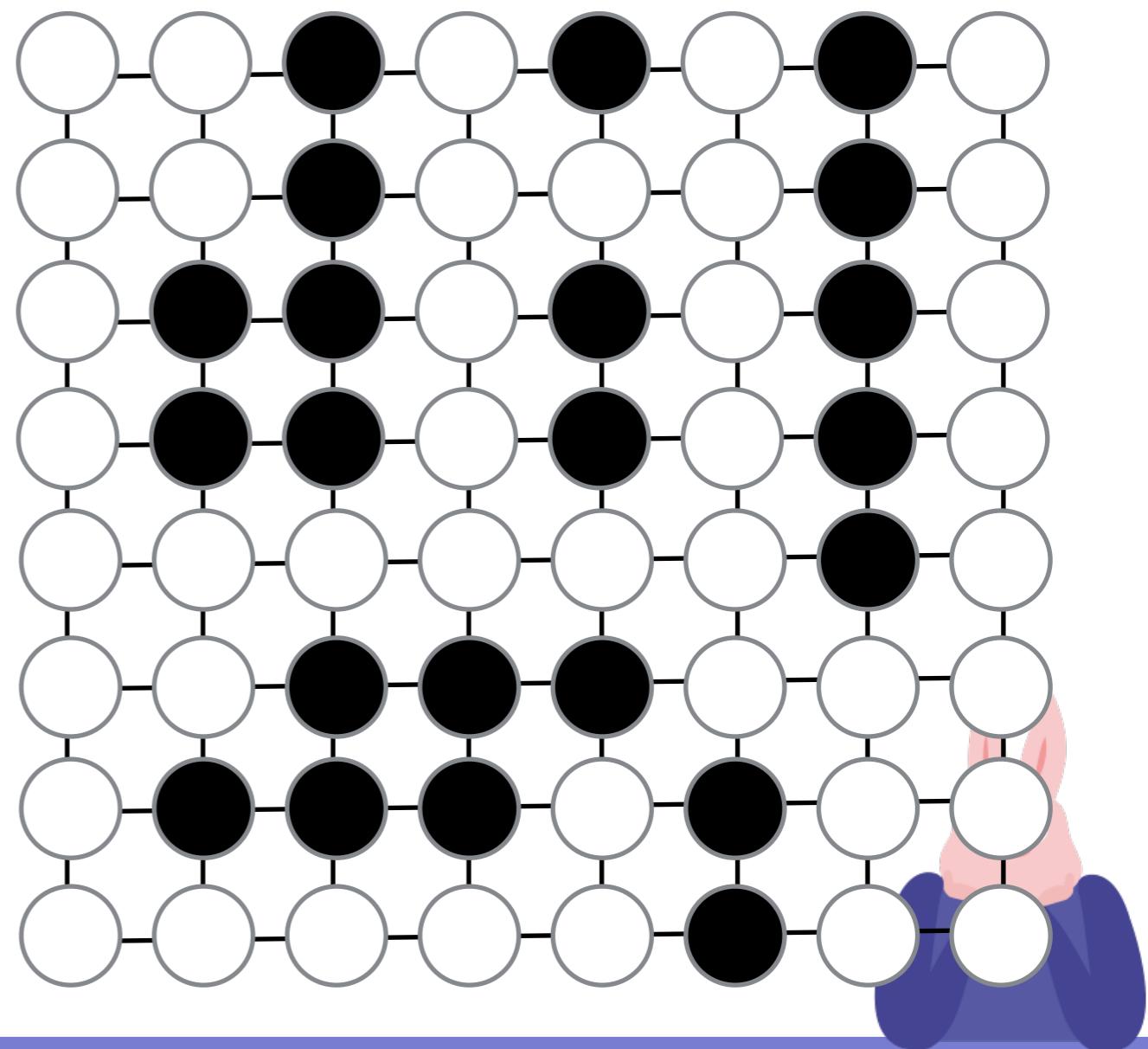
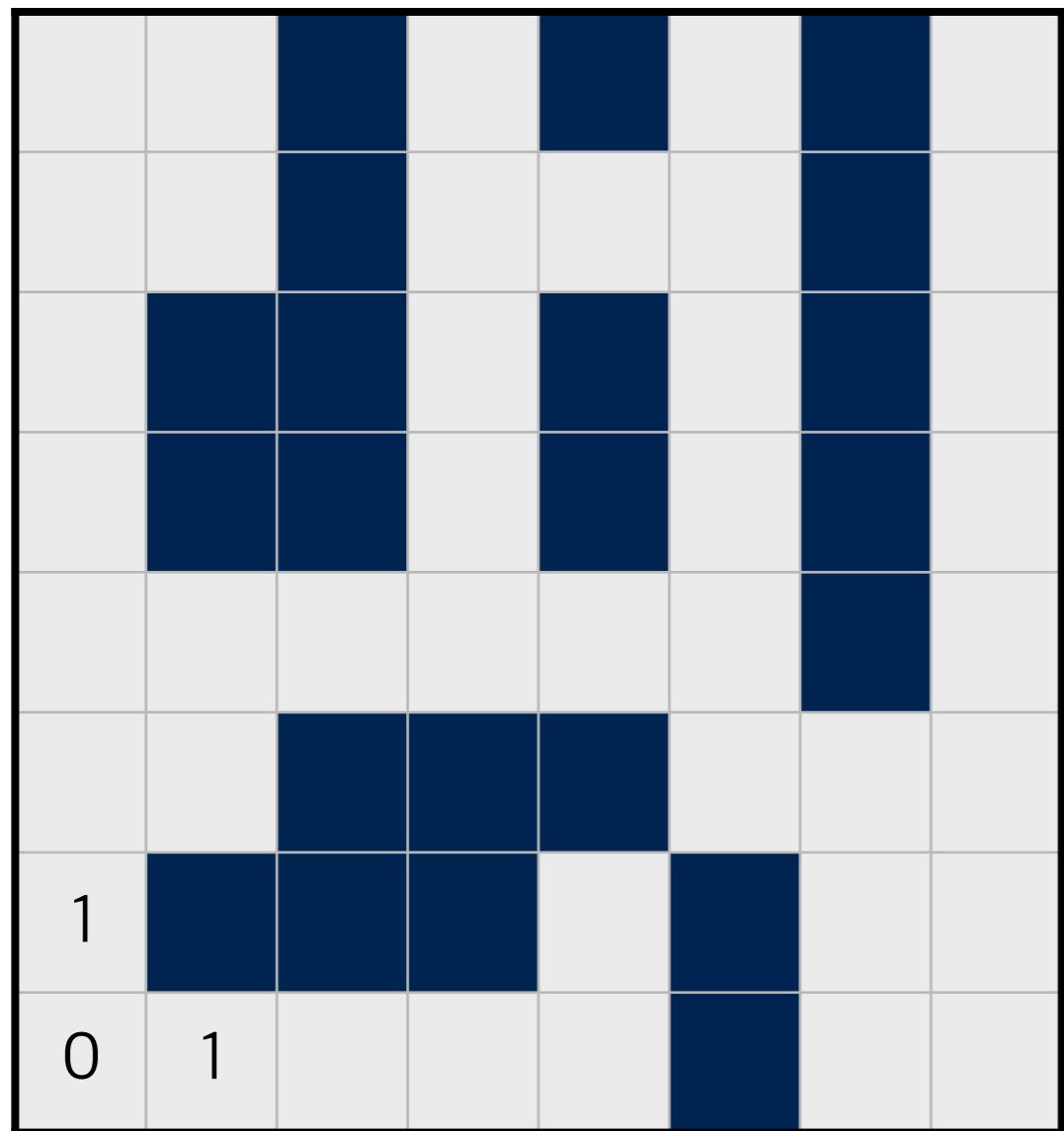
# BFS의 응용 : Flood Fill

- 물이 차오르는 듯 하여 Flood fill 이라 부름



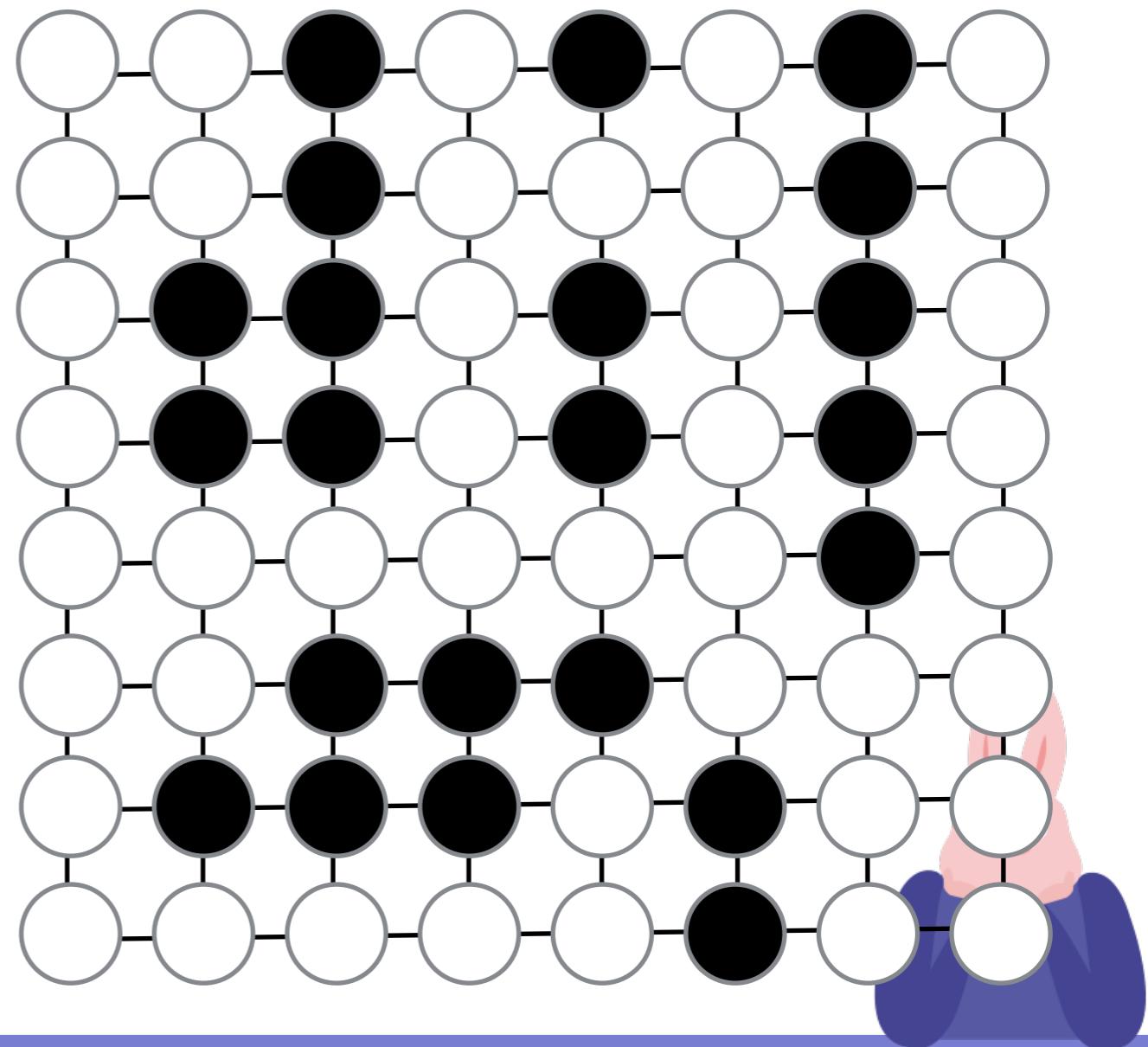
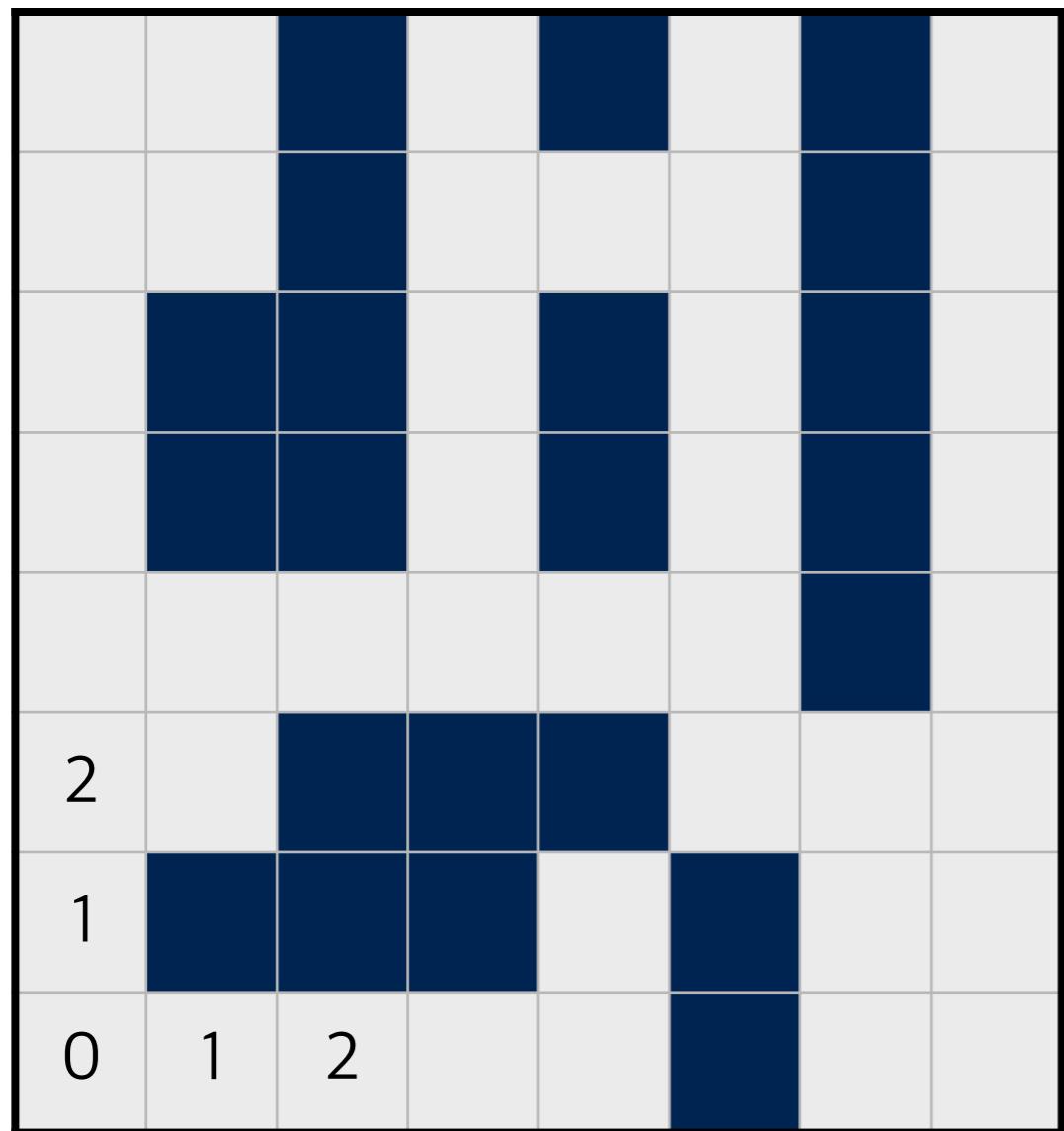
# BFS의 응용 : Flood Fill

- 물이 차오르는 듯 하여 Flood fill 이라 부름



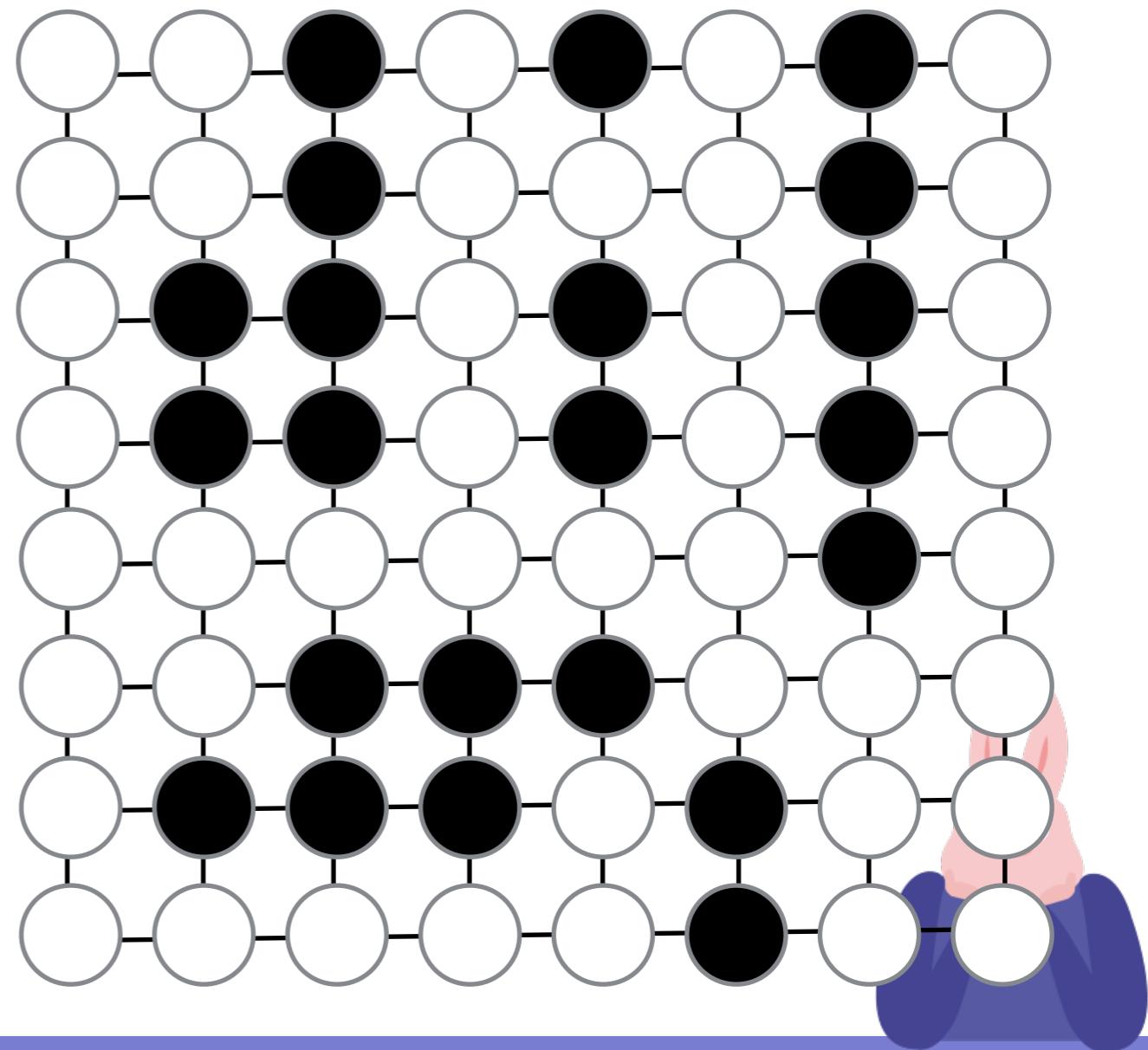
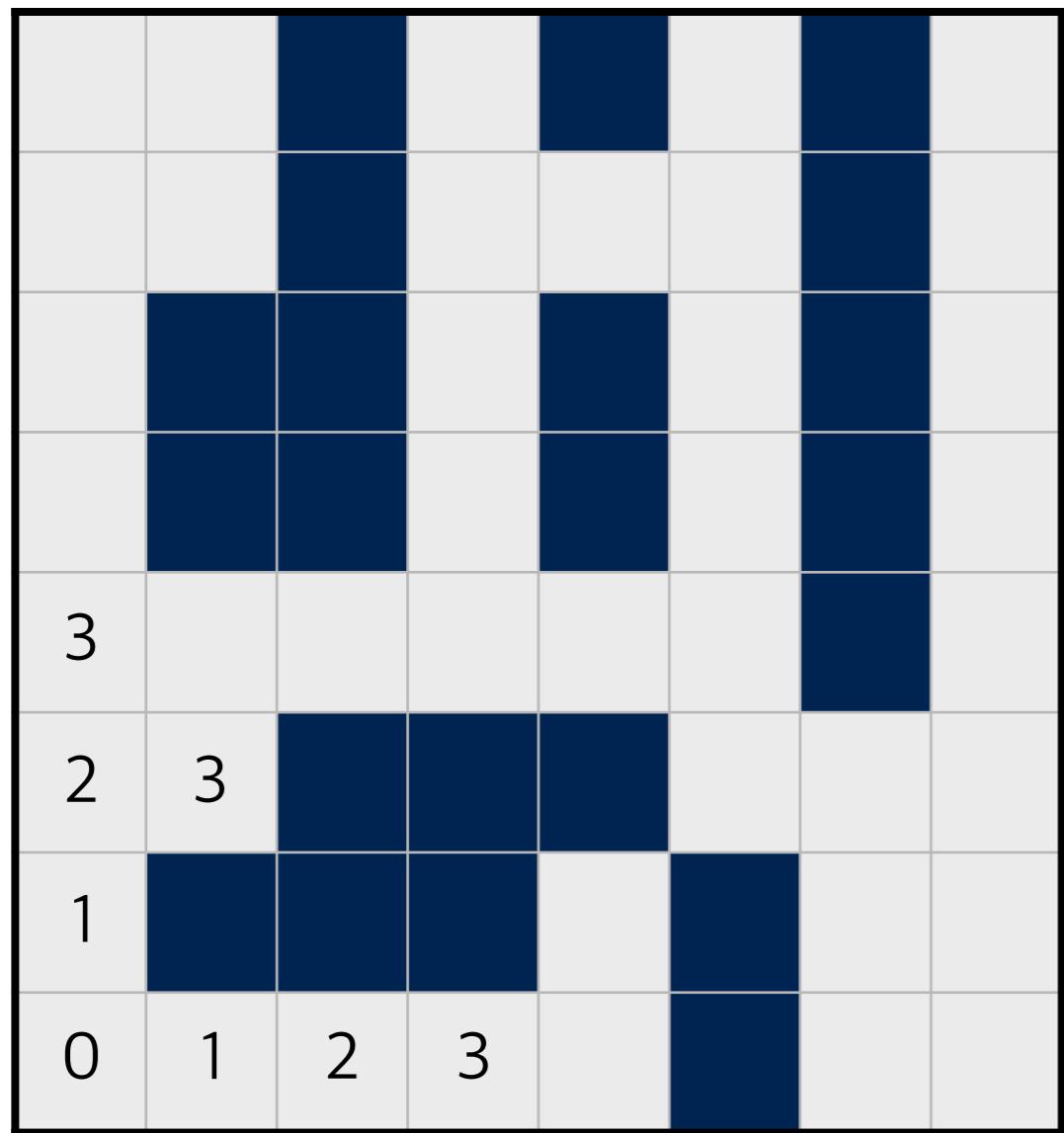
# BFS의 응용 : Flood Fill

- 물이 차오르는 듯 하여 Flood fill 이라 부름



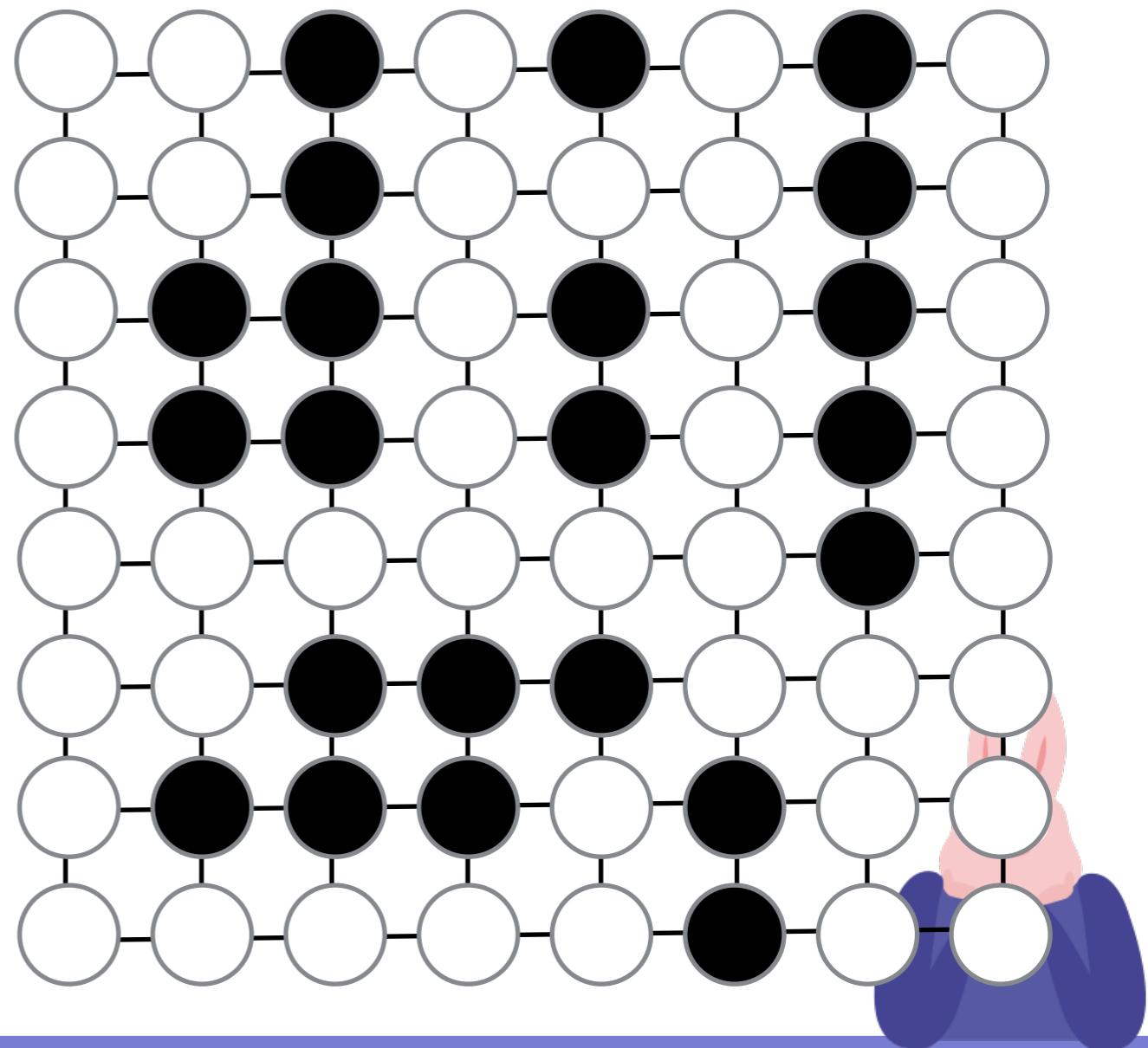
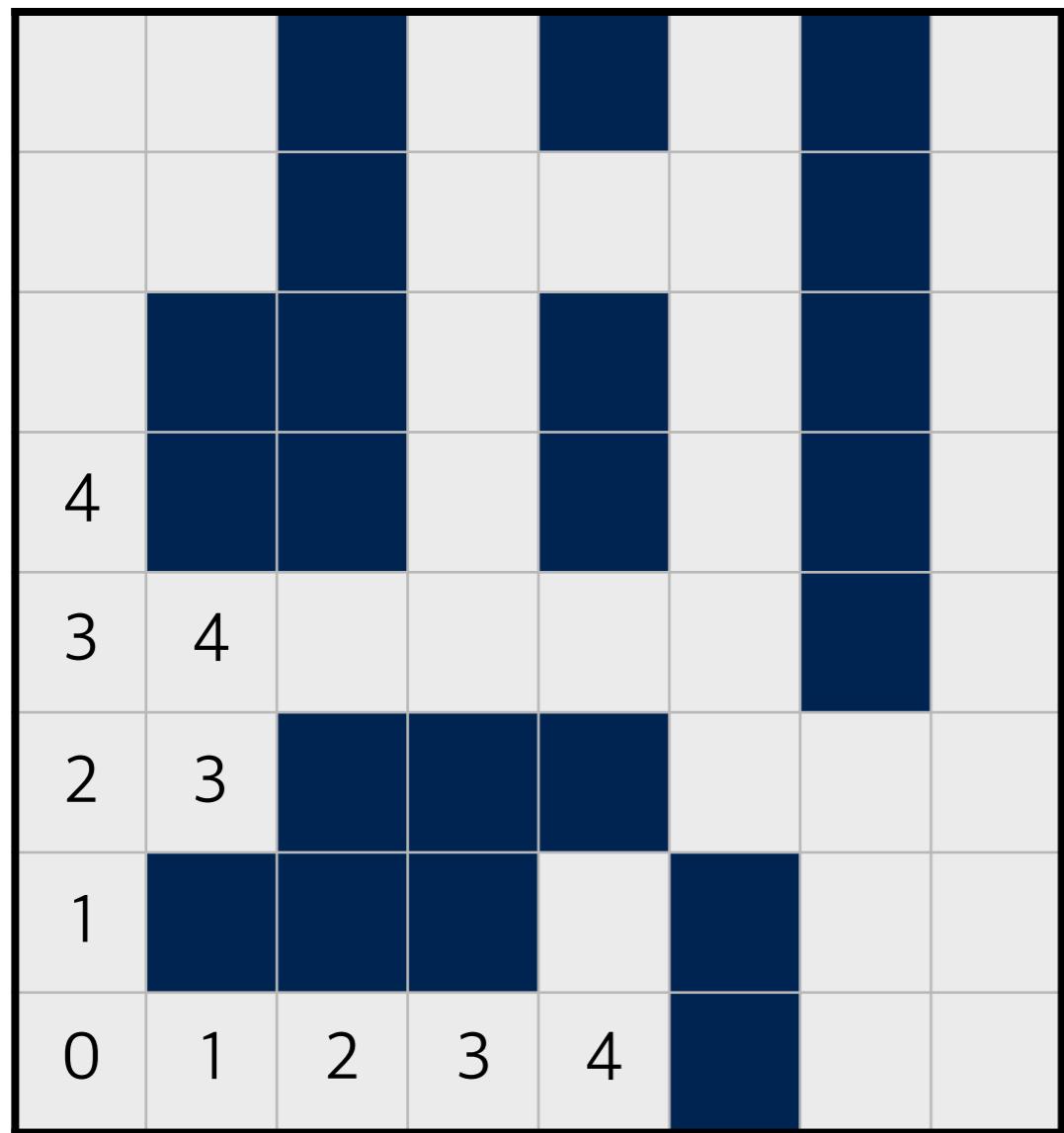
# BFS의 응용 : Flood Fill

- 물이 차오르는 듯 하여 Flood fill 이라 부름



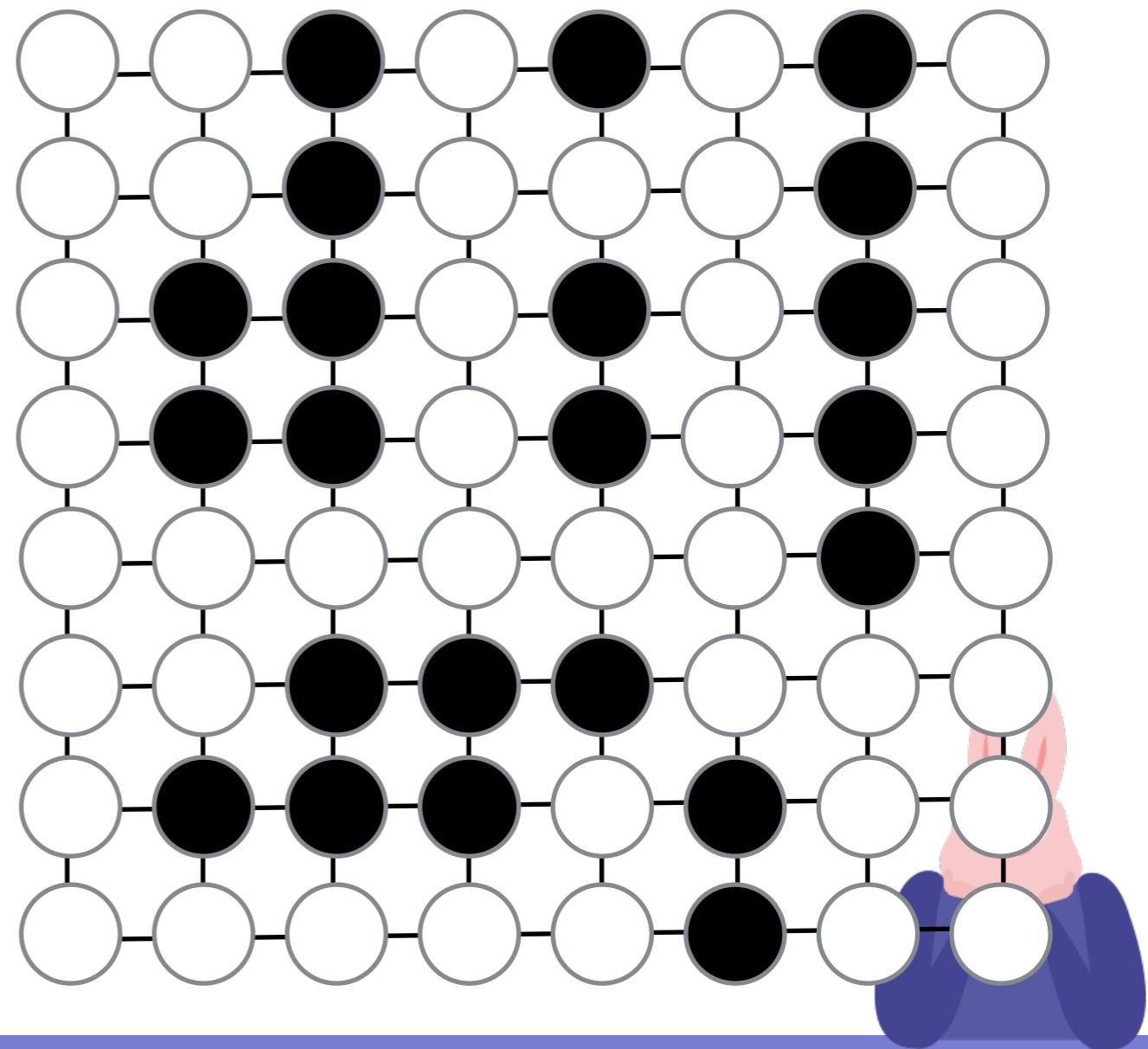
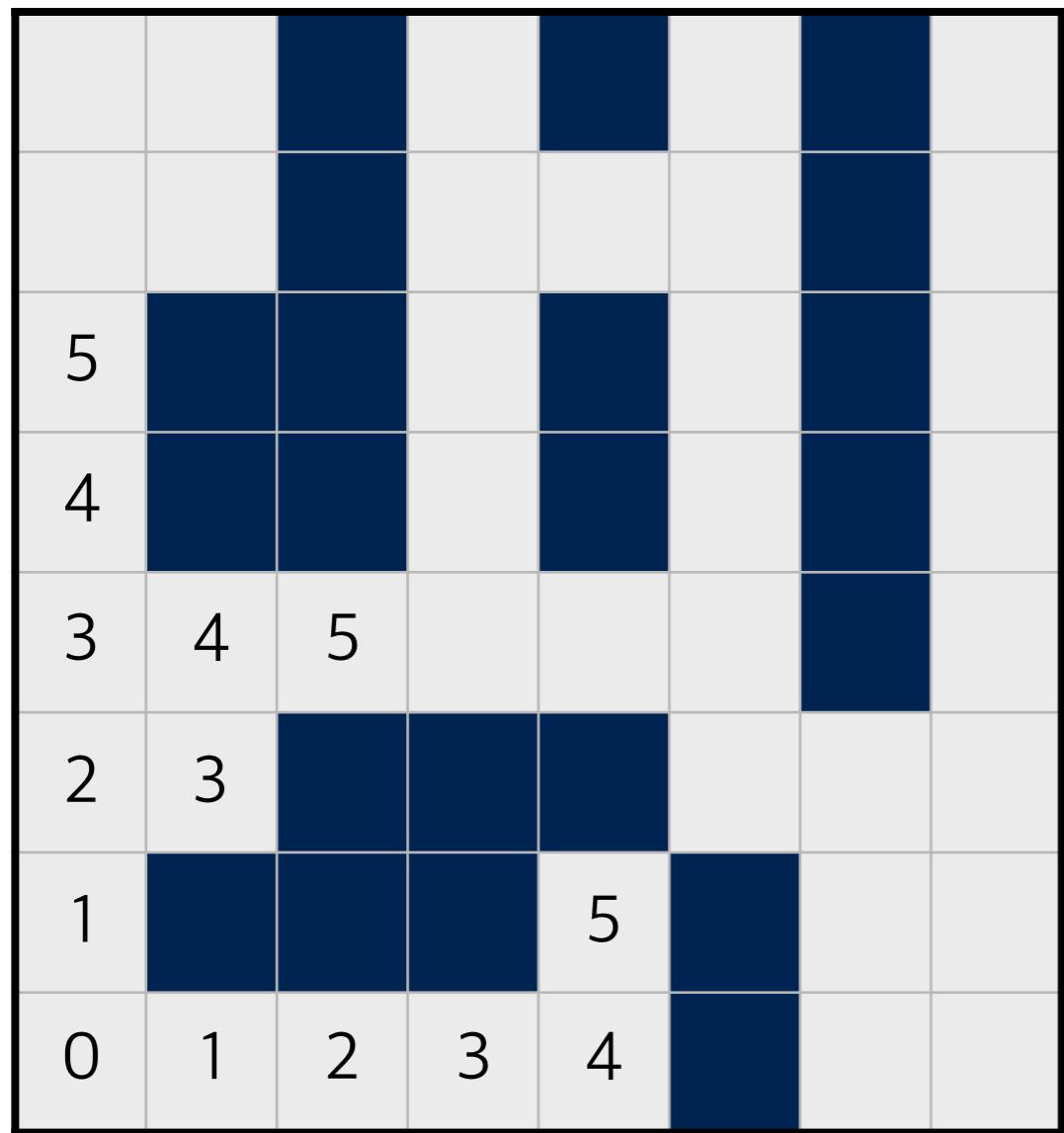
# BFS의 응용 : Flood Fill

- 물이 차오르는 듯 하여 Flood fill 이라 부름



# BFS의 응용 : Flood Fill

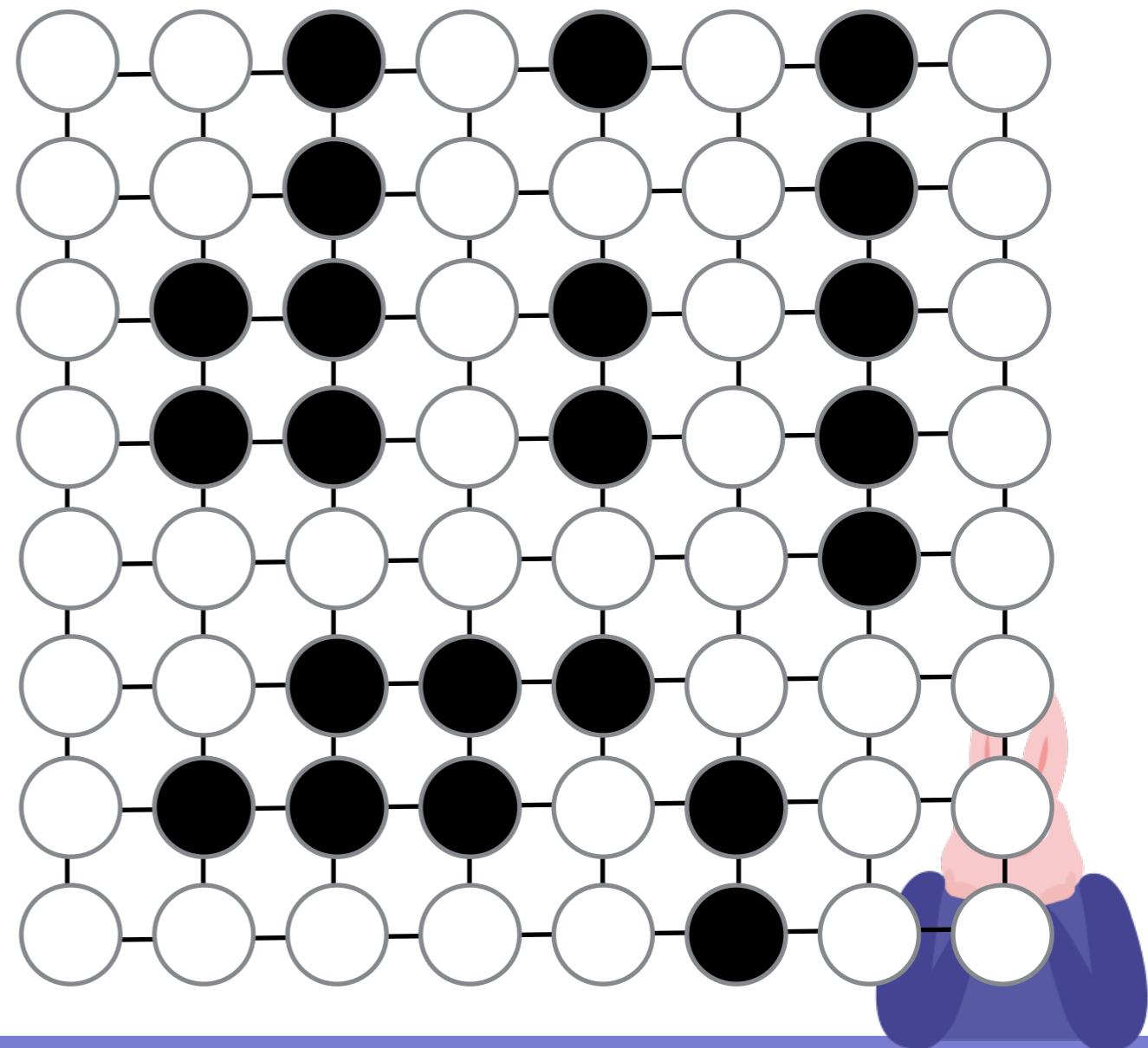
- 물이 차오르는 듯 하여 Flood fill 이라 부름



# BFS의 응용 : Flood Fill

- 물이 차오르는 듯 하여 Flood fill 이라 부름

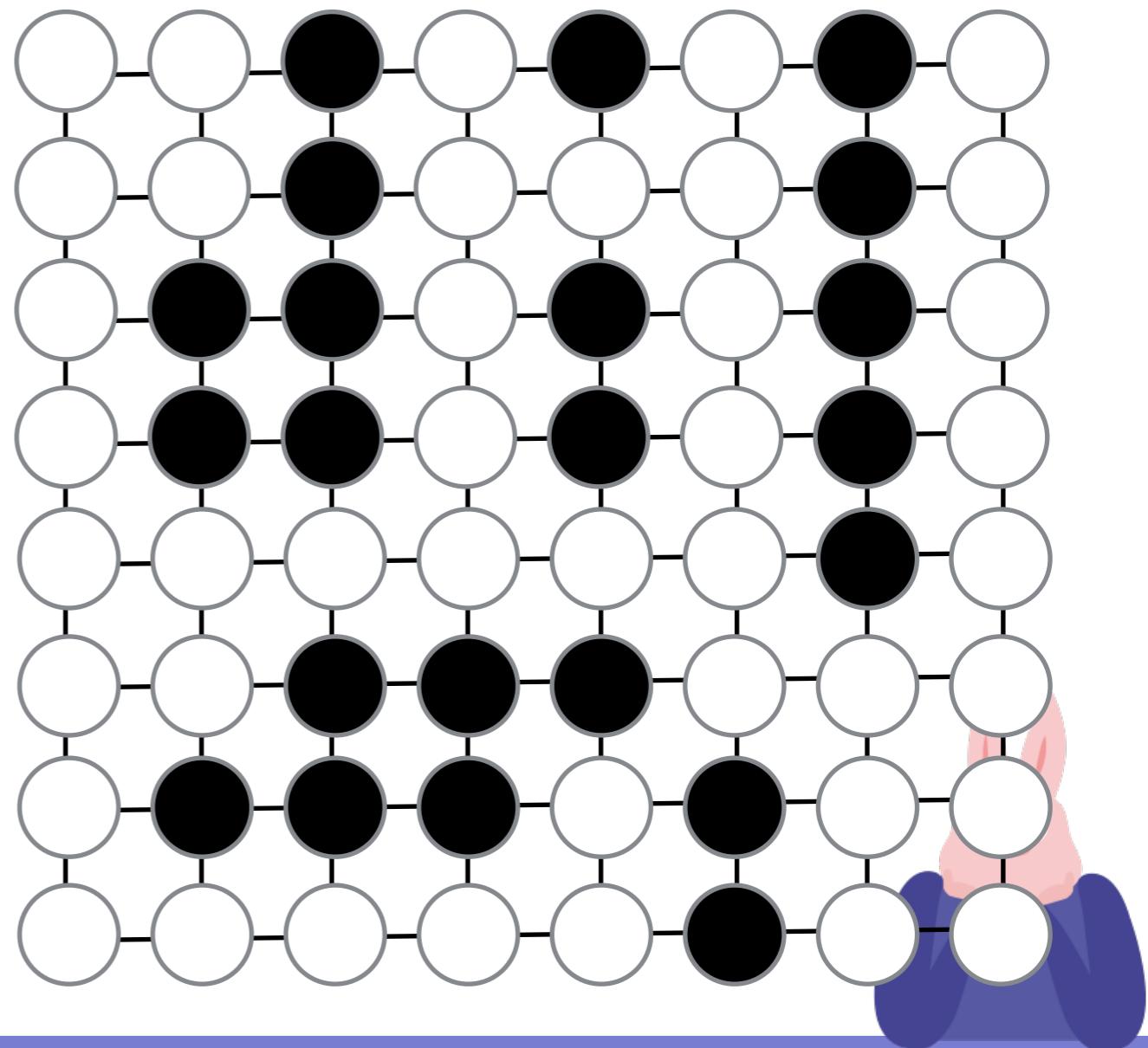
6						
5						
4						
3	4	5	6			
2	3					
1				5		
0	1	2	3	4		



# BFS의 응용 : Flood Fill

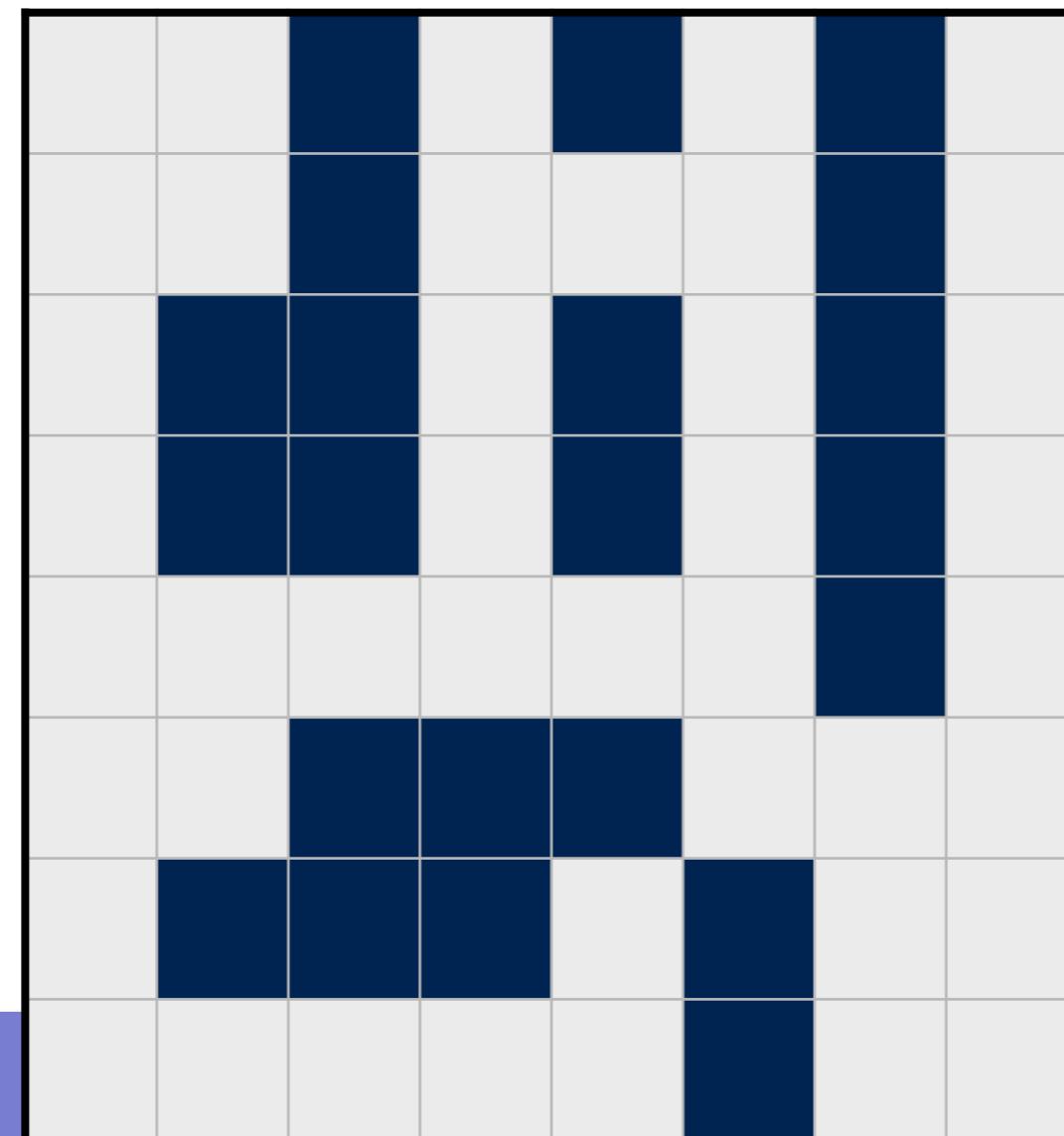
- 물이 차오르는 듯 하여 Flood fill 이라 부름

7							
6	7						
5							
4				7			
3	4	5	6	7			
2	3						
1					5		
0	1	2	3	4			



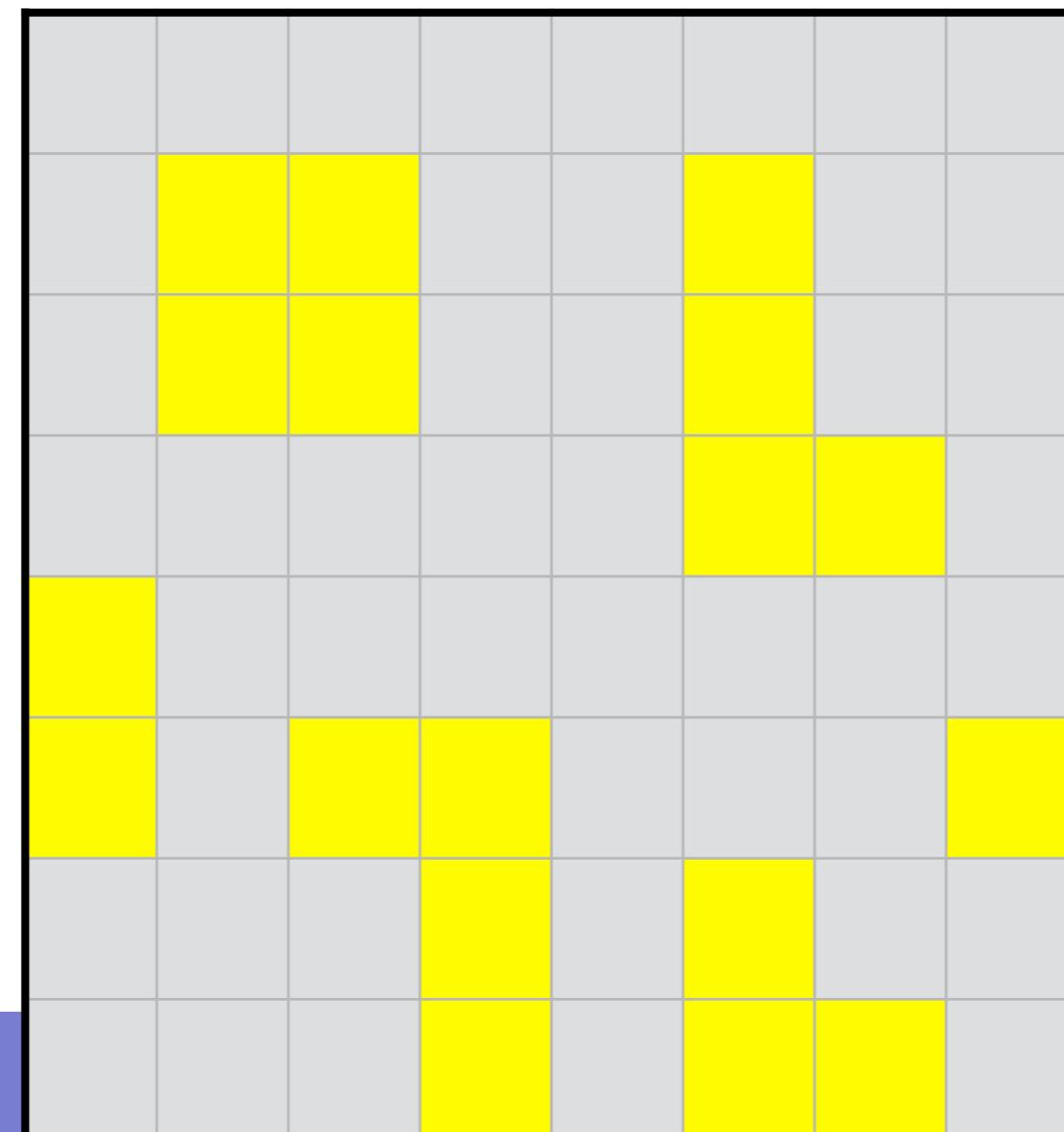
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 인접한 블럭의 집합에 색칠하기



# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에 몇개의 서로 다른 덩어리가 있는가 ?



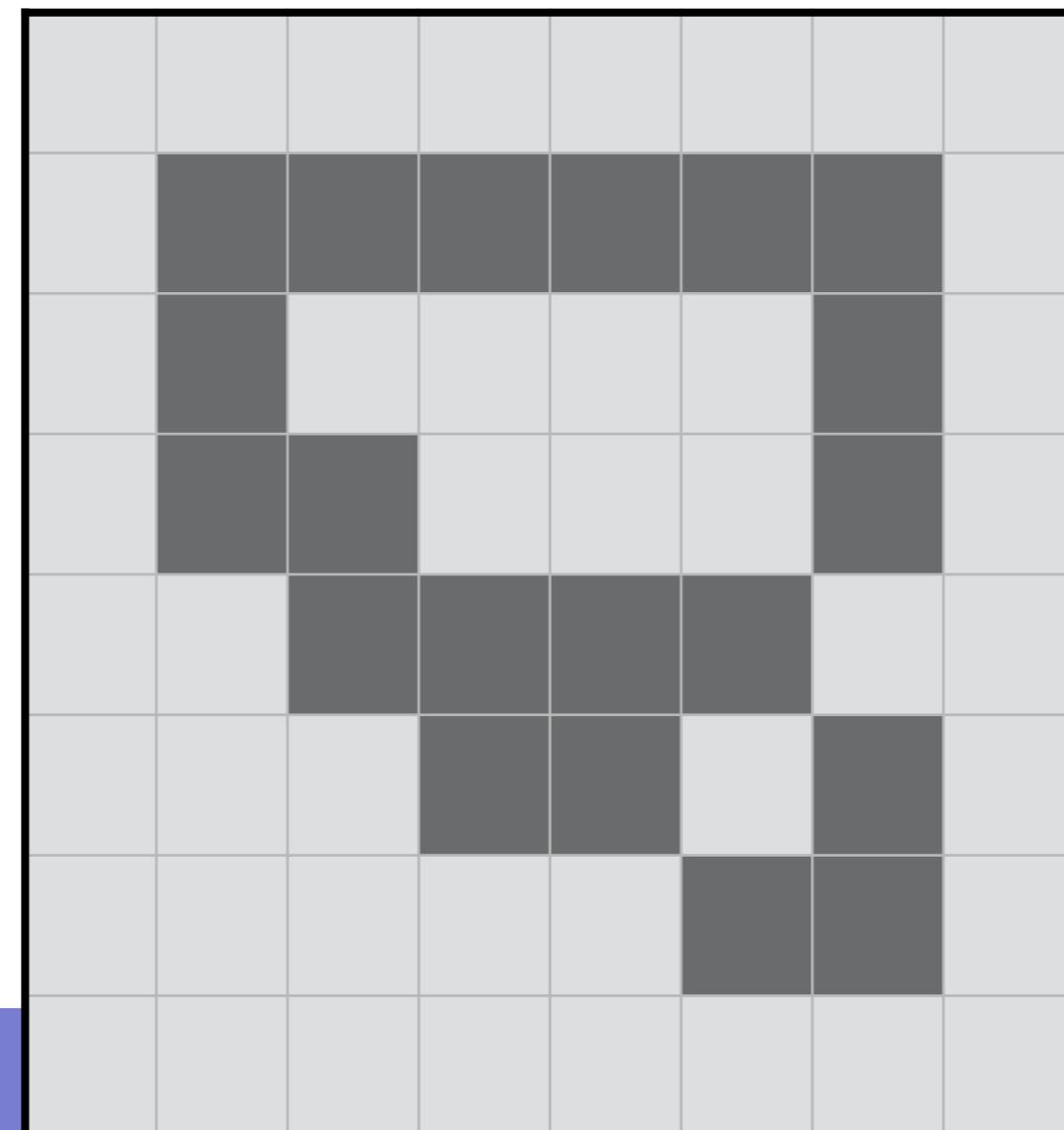
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에 몇개의 서로 다른 덩어리가 있는가 ? 6개!



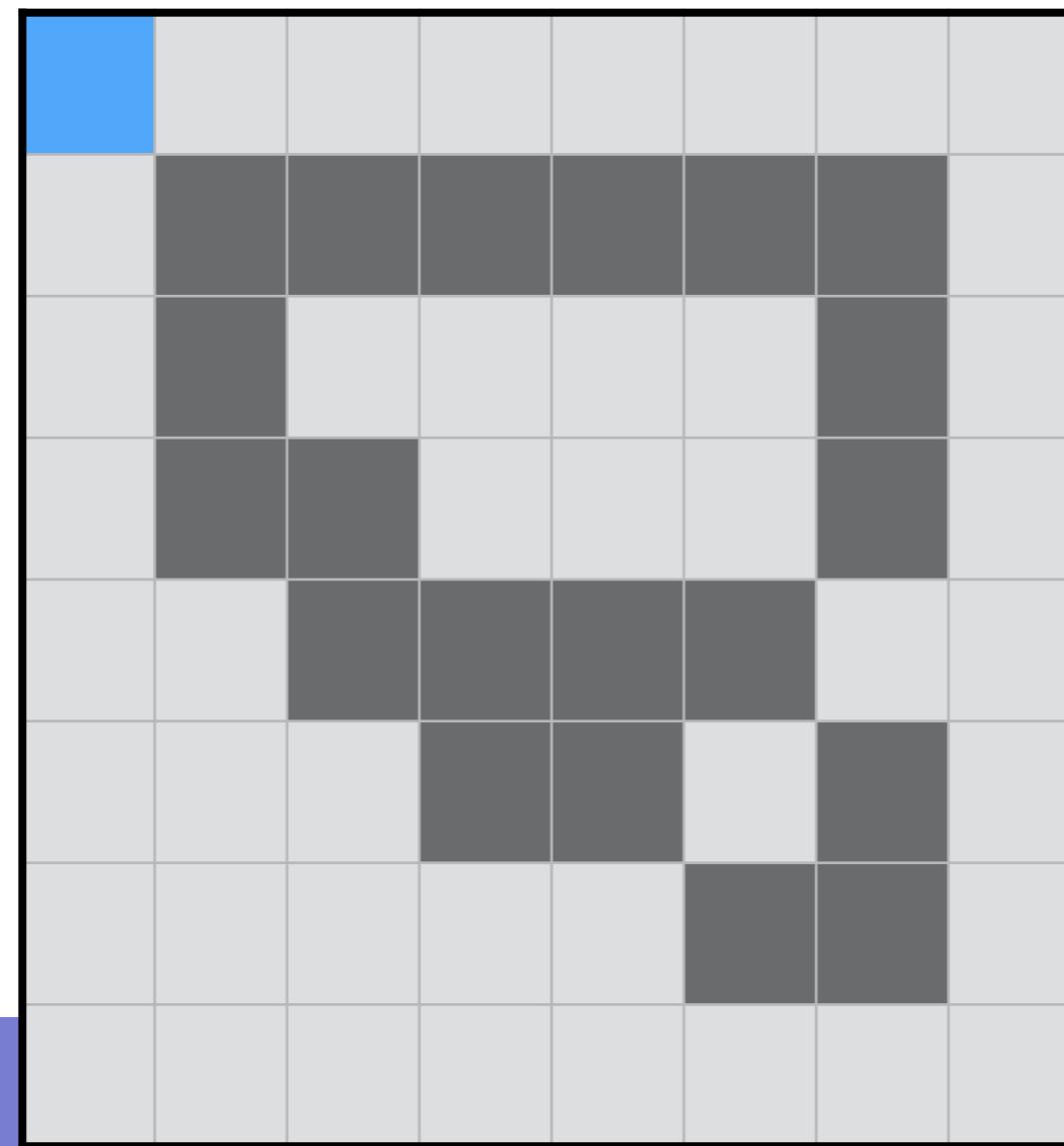
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



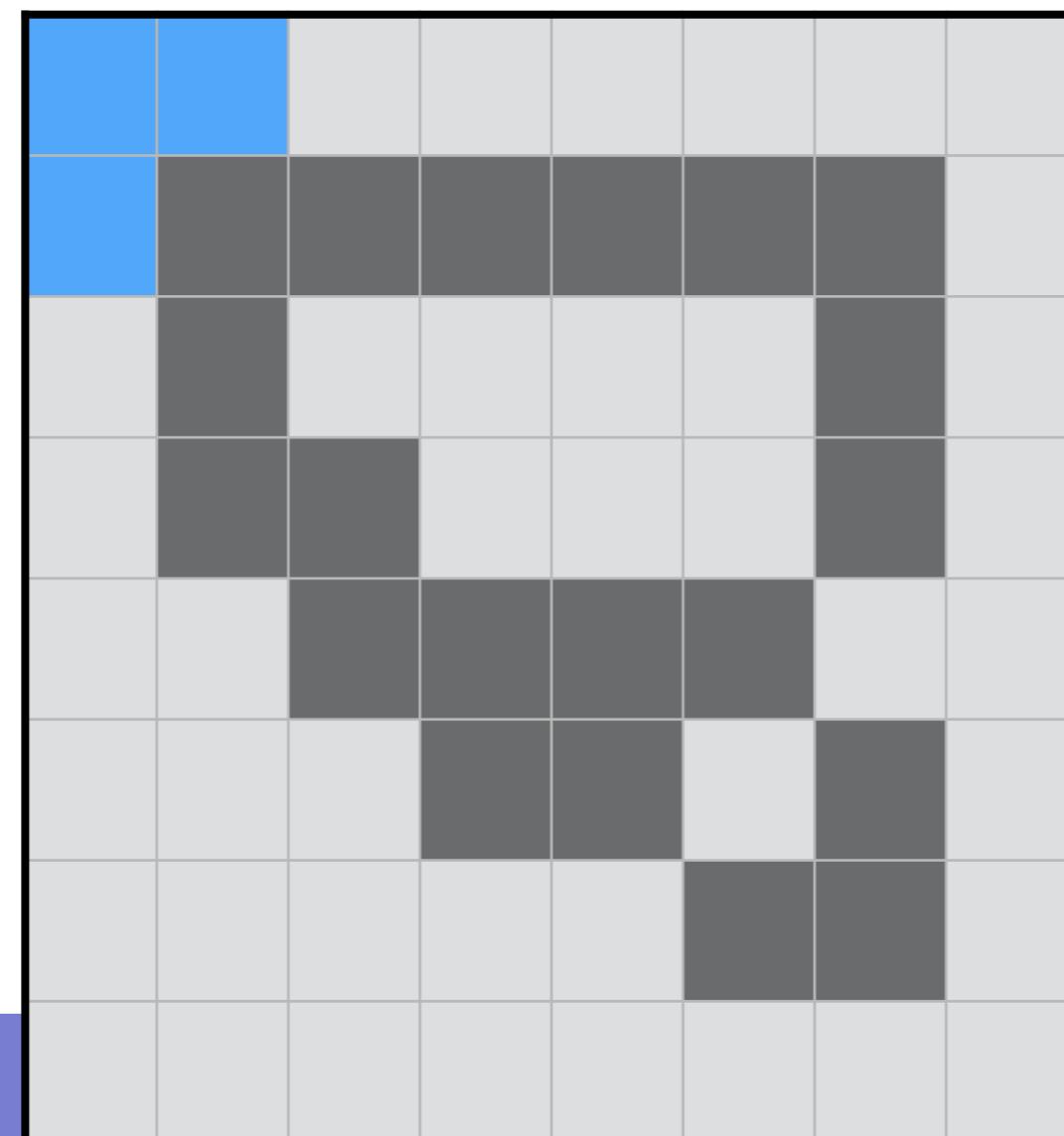
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



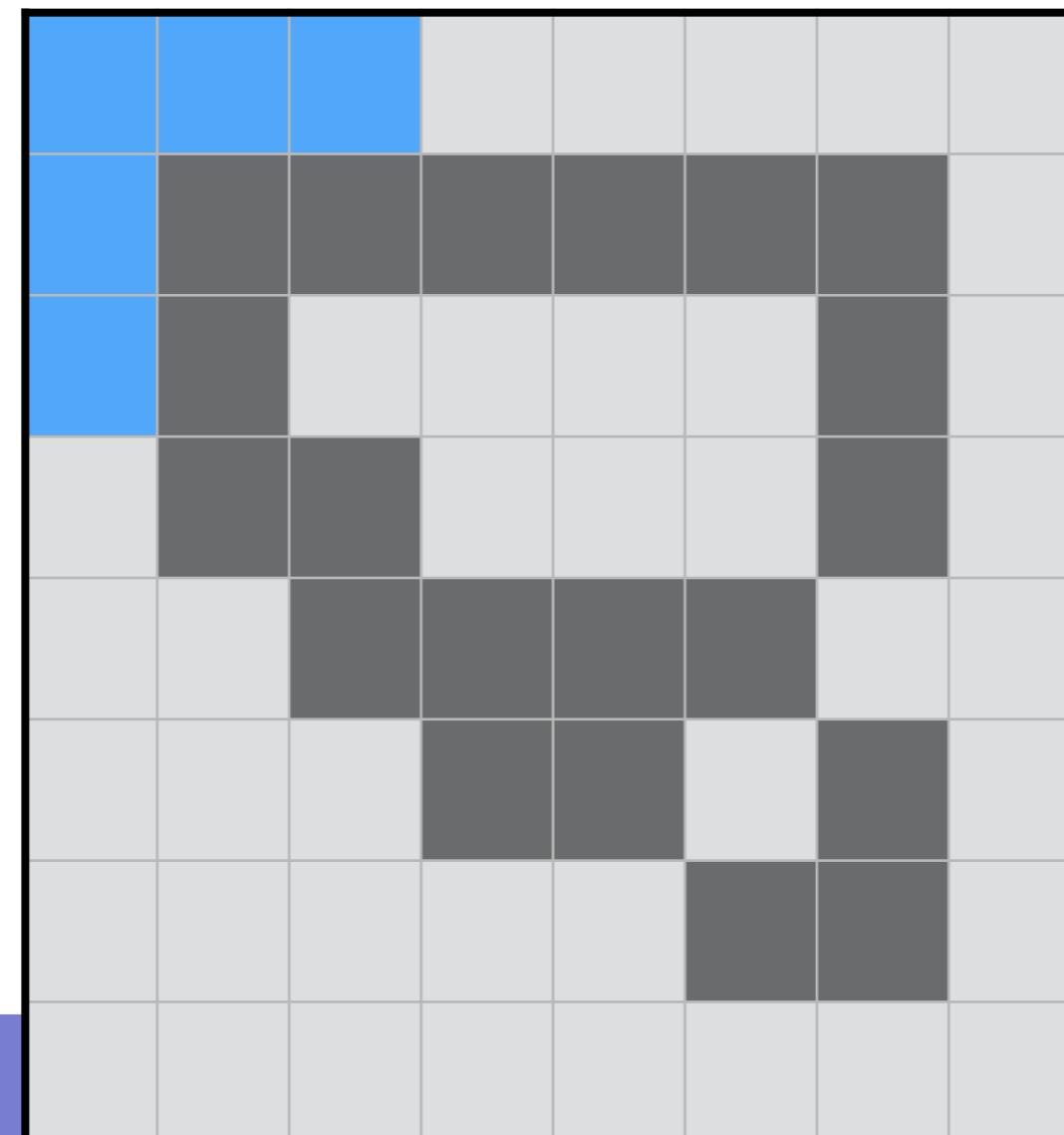
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



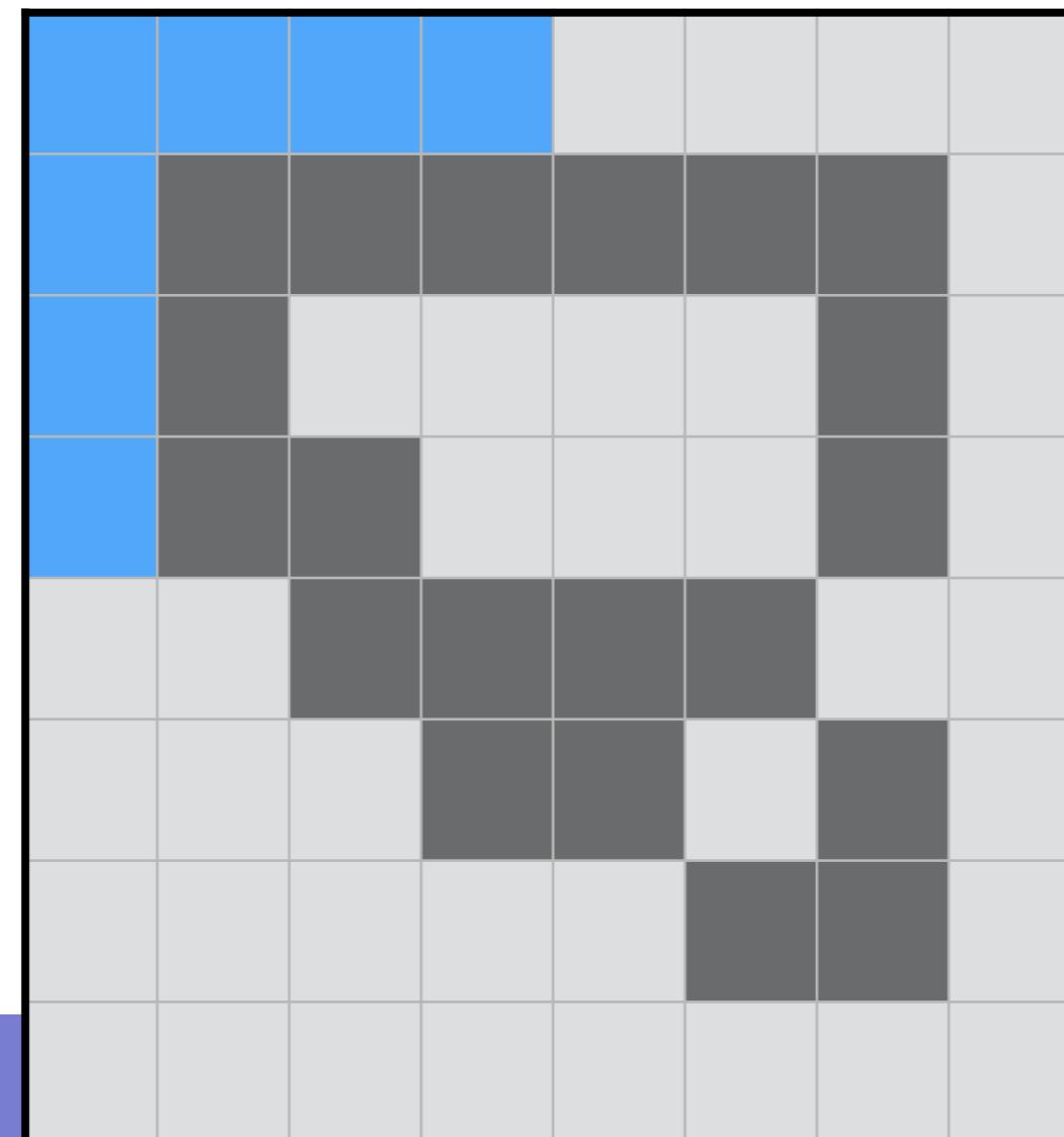
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



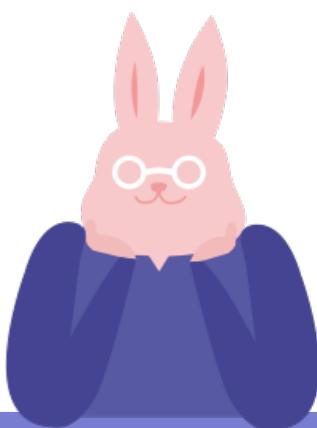
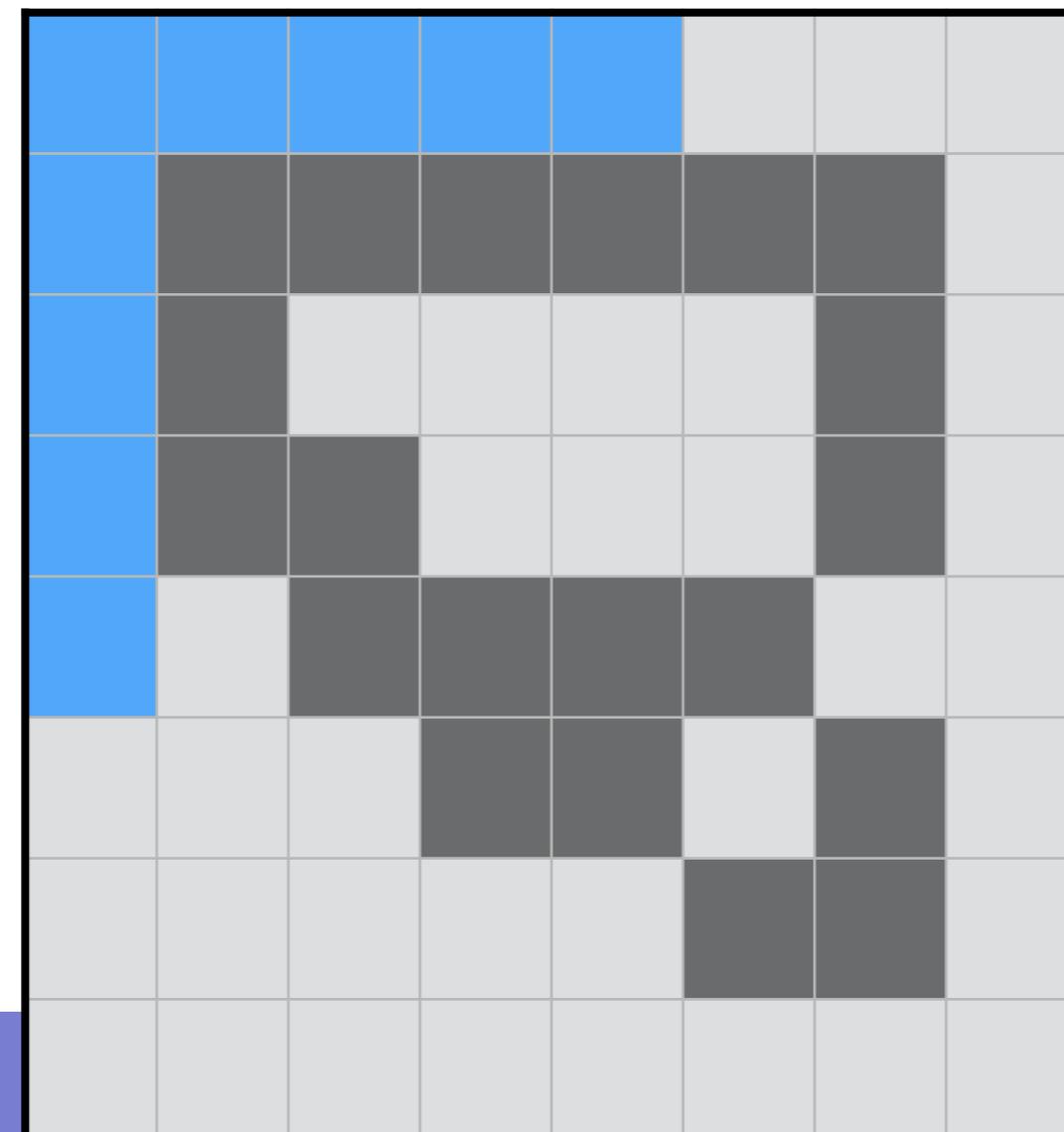
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



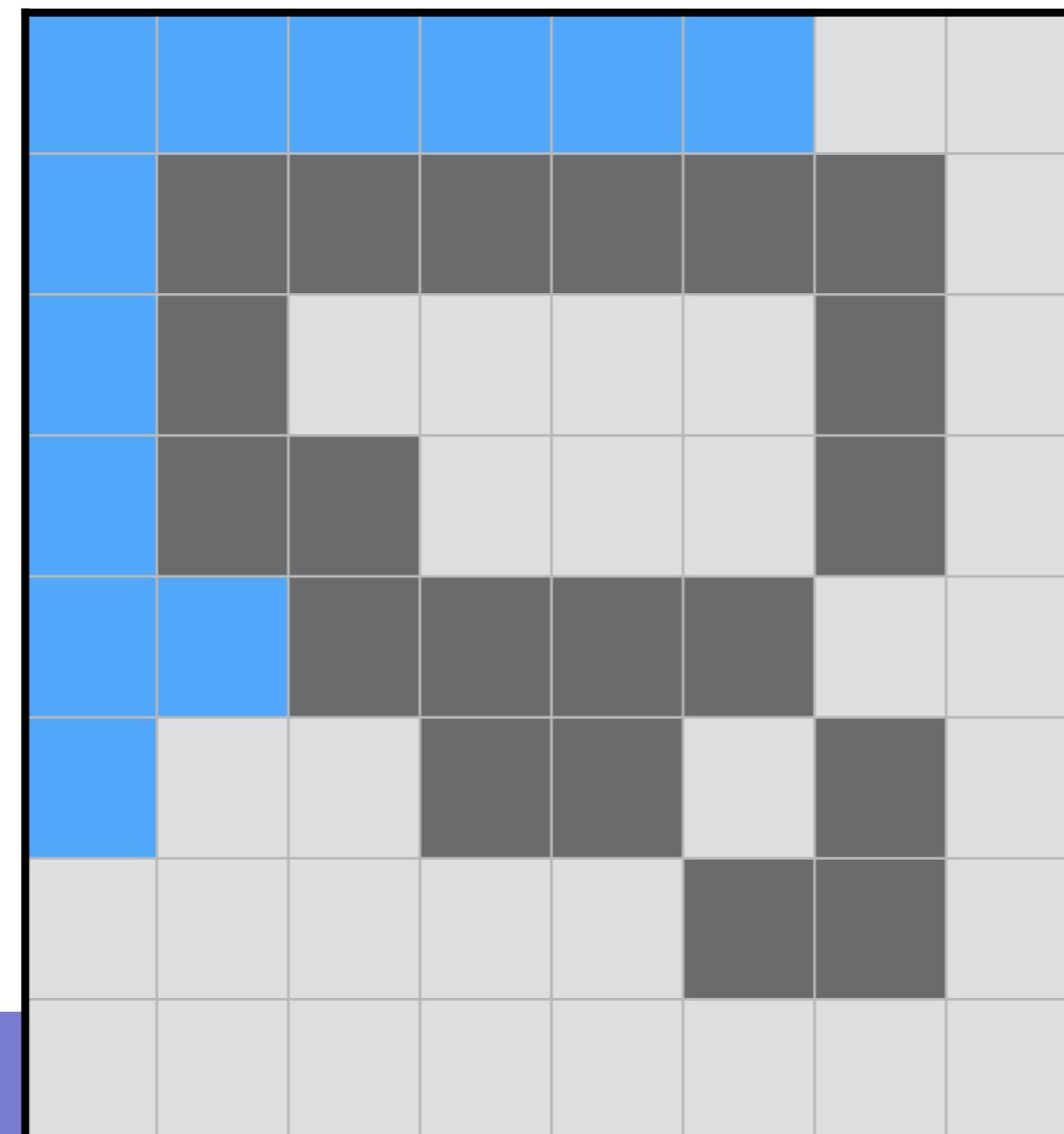
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



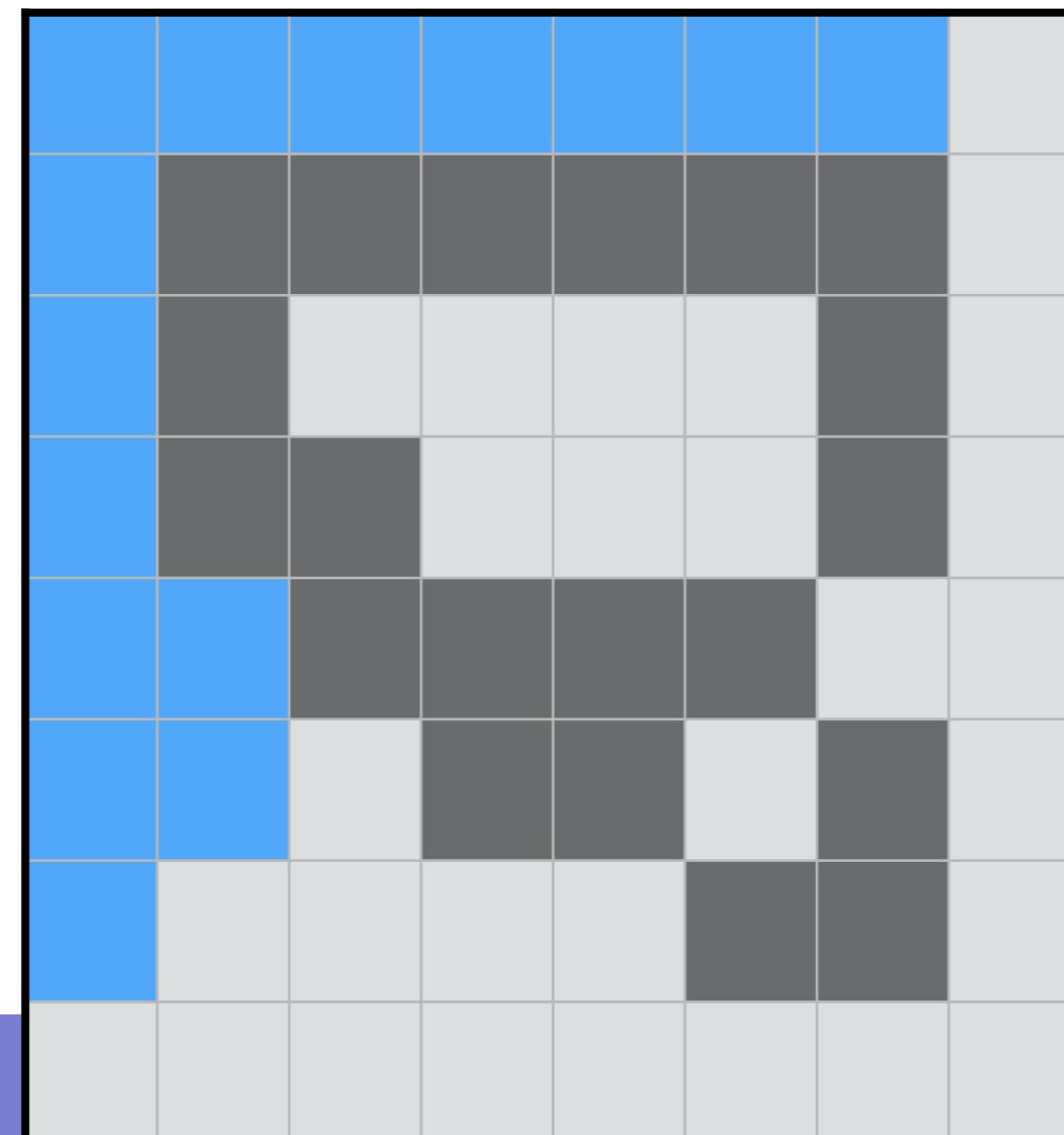
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



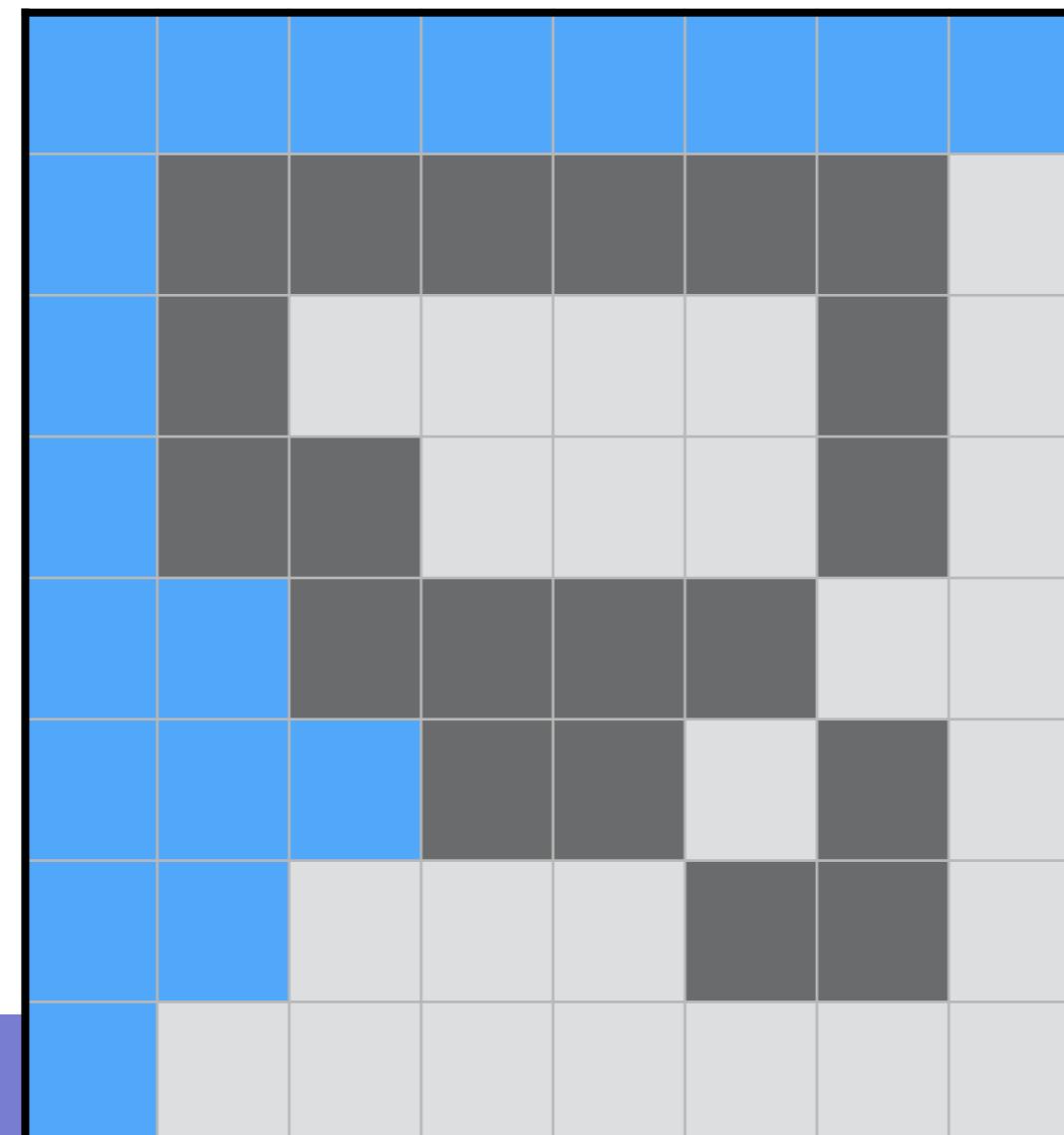
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



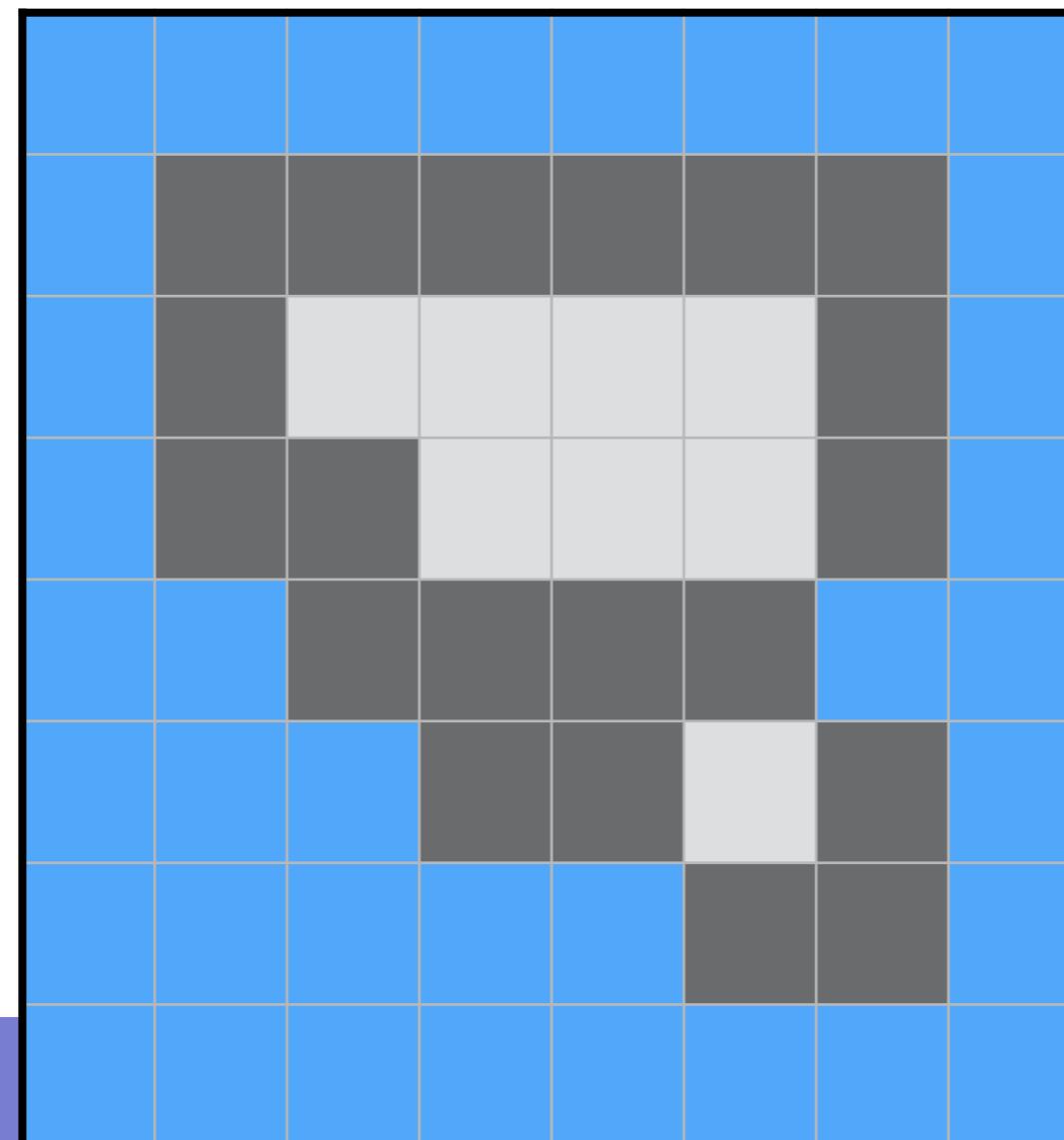
# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



# BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라



# [활동문제 2] 미로찾기

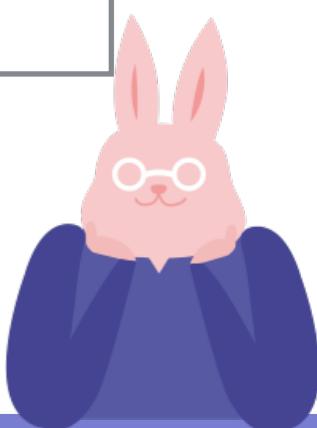
- 시작점에서 출발하여 끝점에 도달하기 위한 최단거리를 출력  
(시작점은 항상 왼쪽 아래, 끝점은 항상 오른쪽 위)

입력의 예

```
5 6
0 1 0 1 1 0
0 1 0 0 1 0
0 0 0 0 1 0
0 1 1 0 0 0
0 1 0 0 0 0
```

출력의 예

```
11
```



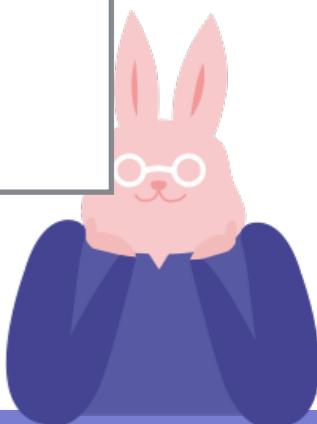
# [예제] 바이러스

- k마리의 바이러스가 1초 후에는  $k*2$  마리, 혹은  $k/3$ 마리가 됨  
N마리가 되기 위해 걸리는 최소 시간과 그 경로는 ?  
(처음엔 1마리에서 시작, k는 항상 10000 이하여야 함)

입력의 예

출력의 예

6	1	2	4	8	16	5	10
---	---	---	---	---	----	---	----



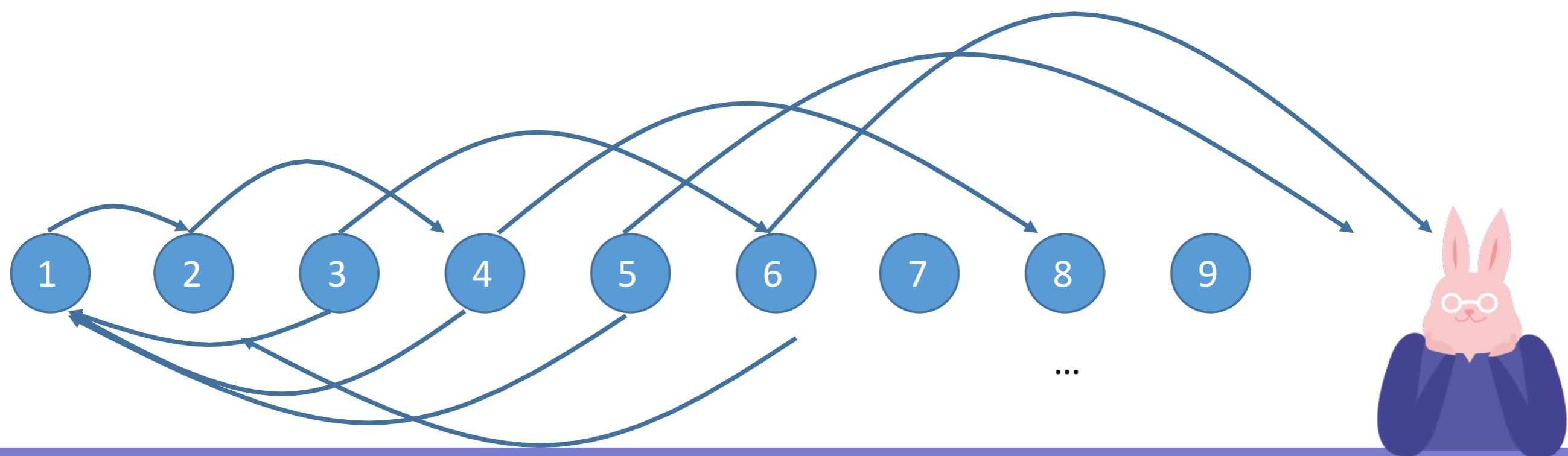
# Graph 문제의 어려운 점

- 이 문제를 풀기 위해서 그래프를 어떻게 이용해야 하는가 ?
- **Modeling**
  - 정점은 무엇을 의미하는가 ?
  - 간선은 무엇을 의미하는가 ?



# [예제] 바이러스

- 각 정점은 바이러스가 몇 마리 있는지에 대한 상태
  - 정점 X : 현재 바이러스가 X마리 있는 상태이다
- 각 간선은 바이러스의 마릿수의 변화를 나타냄
  - $X \rightarrow Y$  : X 마리에서 Y마리로 변화할 수 있음



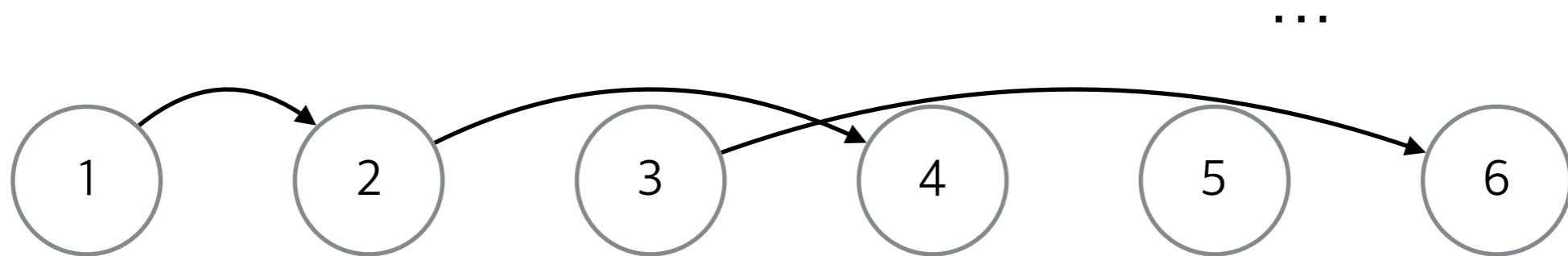
# [예제] 바이러스

- 각 정점은 바이러스가 몇 마리 있는지에 대한 상태
  - 정점 X : 현재 바이러스가 X마리 있는 상태이다
- 각 간선은 바이러스의 마릿수의 변화를 나타냄
  - $X \rightarrow Y$  : X 마리에서 Y마리로 변화할 수 있음
- 정점 1에서 정점 N으로 가는 최단경로는 무엇인가 ?
  - BFS로 해결



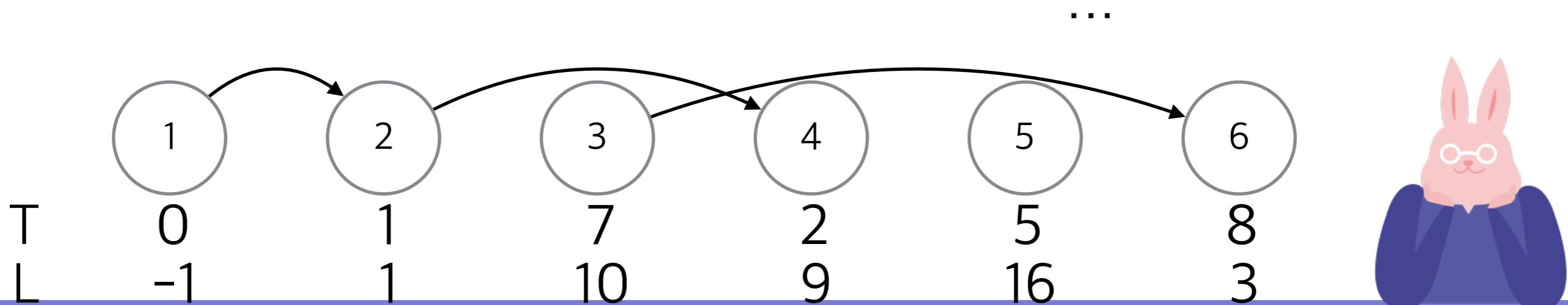
# [예제] 바이러스

- 거리는 어떻게 구하는가 ?
  - $D(x)$  : Node x 에 도달하기 위한 거리
- 경로는 어떻게 구하는가 ?
  - $L(x)$  : Node x에 도달하기 직전에 있었던 노드 번호



# [예제] 바이러스

- 거리는 어떻게 구하는가 ?
  - $D(x)$  : Node x 에 도달하기 위한 거리
- 경로는 어떻게 구하는가 ?
  - $L(x)$  : Node x에 도달하기 직전에 있었던 노드 번호



# 감사합니다!

신현규

E-mail : [hyungyu.sh@kaist.ac.kr](mailto:hyungyu.sh@kaist.ac.kr)

Kakao : yougatup

