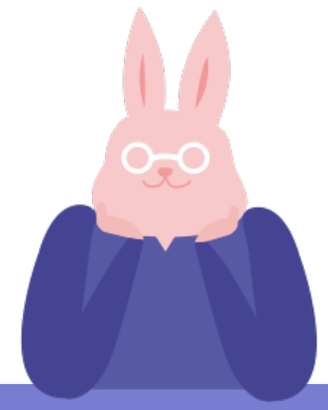
/* elice */

Time Complexity and Data Structure

2016. 11. 22.

신현규



/* elice */

1

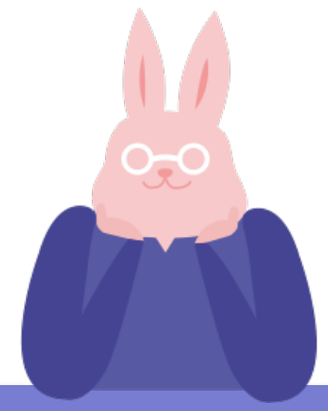
2

3

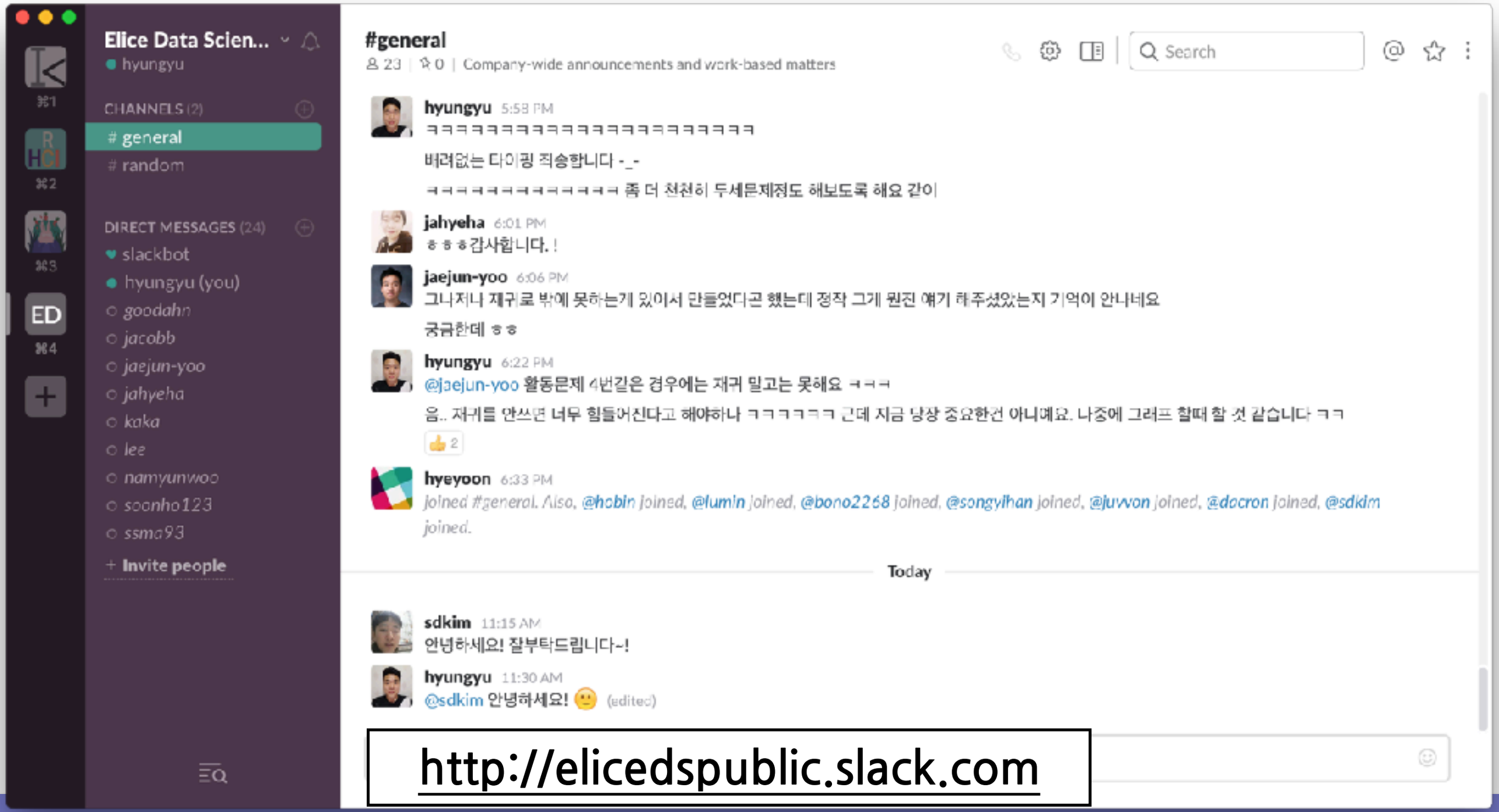
4

5

6

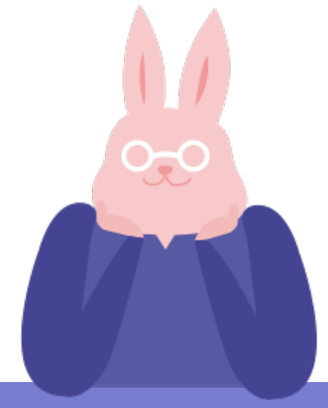


Slack


Elice Data Scien...
hyungyu
CHANNELS (2)
general
random
DIRECT MESSAGES (24)
slackbot
hyungyu (you)
goodahn
jacobb
jaejun-yoo
jahyeha
kaka
lee
namyunwoo
soonho123
ssma93
+ Invite people
#general
23 | 0 | Company-wide announcements and work-based matters
Search
hyungyu 5:58 PM
ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ
버려없는 다이굉 직송합니다 -_-
ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ 좀 더 천천히 두세문제정도 해보도록 해요 같이
jahyeha 6:01 PM
ㅎㅎㅎ감사합니다.!
jaejun-yoo 6:06 PM
그나저나 재귀로 밖에 못하는게 있어서 만들었다곤 했는데 정작 그게 뭔진 얘기 하주셨었는지 기억이 안나네요
궁금한데 ㅎㅎ
hyungyu 6:22 PM
@jaejun-yoo 활동은제 4번길은 경우에는 재귀 밀고는 못해요 ㅋㅋㅋ
음.. 재귀를 안쓰면 너무 힘들어진다고 해야하나 ㅋㅋㅋㅋㅋㅋ 근데 지금 당장 중요한건 아니예요. 나중에 그러프 할때 할 것 같습니다 ㅋㅋ
2
hyeyoon 6:33 PM
joined #general. Also, @hobin joined, @lumin joined, @bono2268 joined, @songylhan joined, @juvvon joined, @docron joined, @sdkim joined.
Today
sdkim 11:15 AM
안녕하세요! 잘부탁드립니다~!
hyungyu 11:30 AM
@sdkim 안녕하세요! (edited)
http://elicedspublic.slack.com

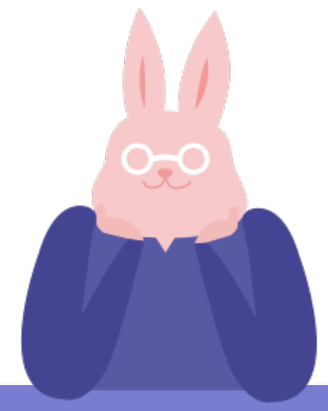
피드백

- 파이썬(과 코딩자체)에 익숙하지 않아요
- 타이핑 스피드가 너무 빨라요
- 재귀 별로 안좋아하는데 재귀만 해서 아쉬웠어요
- 제한시간때문에 돌겠어요



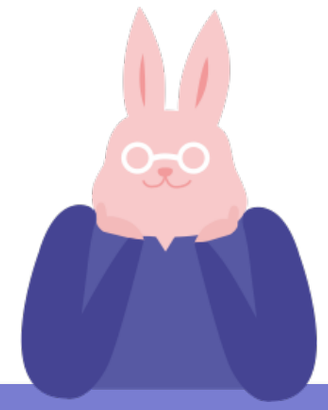
권장하는 실습 방법

- 코딩을 처음 배울때는 양으로 승부하는게 좋습니다
  - 다음주 안으로 방대한 양의(?) 파이썬 코딩에 익숙해지기 위한 문제를 제공해 드리겠습니다.
- 활동문제 바닥부터 다시 짜 보기



지난 시간 요약

- 함수를 최소 단위로 하여 **추상적 논리**에 집중하자
  - 함수의 정의를 명확히 하자
  - 재귀 함수의 디자인
    1. 값을 구할 함수 f를 정의한다
    2. f의 값을 구한다. 점화식 형태를 얻는다.
    3. 정답이 어느 함수의 값인지를 찾는다.



[활동문제 0] 숫자 피라미드

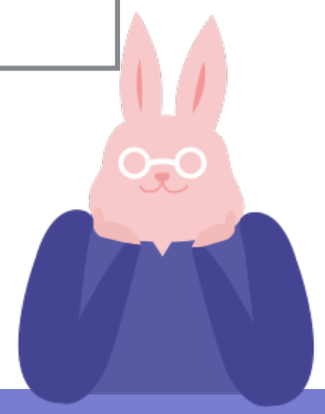
- 아래 예제와 같이 크기 n의 숫자 피라미드를 출력

입력의 예

```
5
```

출력의 예

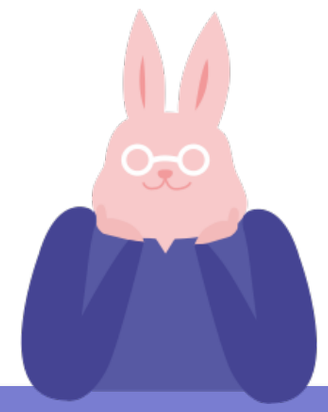
```
    1
   234
  98765
 1234567
765432198
```



/* elice */

```
    1
   234
  98765
 1234567
765432198
```

```
for i in range(n) :
    printLine(n-i+1)

    if i % 2 == 0 :
        rightDirection(num, cnt)
        /* get next number */
    else :
        leftDirection(num, cnt)
        /* get next number */
```



[활동문제 1] 달팽이 숫자 출력

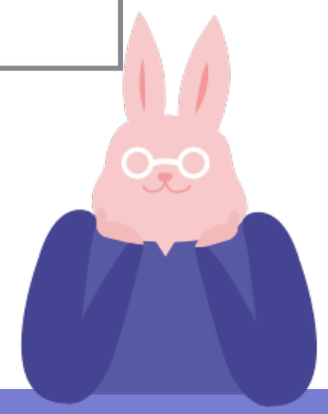
- 아래와 같이 달팽이 모양으로 숫자를 출력

입력의 예

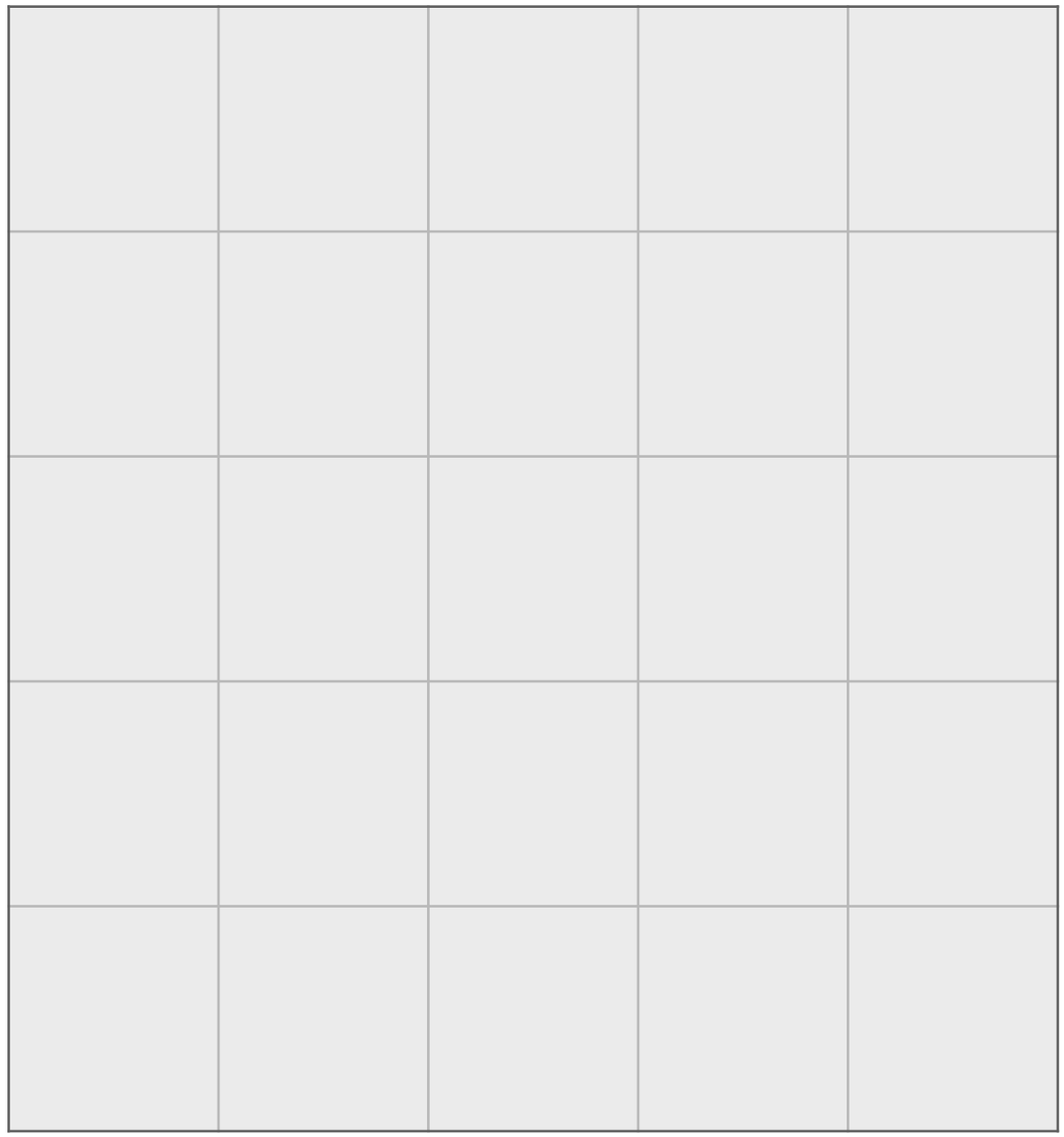
```
5
```

출력의 예

```
1  2  3  4  5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

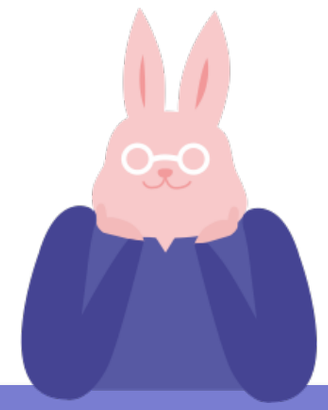


[활동문제 1] 달팽이 숫자 출력



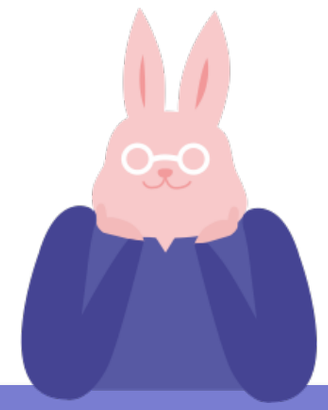
[활동문제 1] 달팽이 숫자 출력

1	2	3	4	5



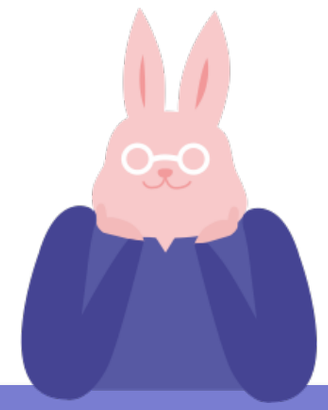
[활동문제 1] 달팽이 숫자 출력

1	2	3	4	5
				6
				7
				8
				9



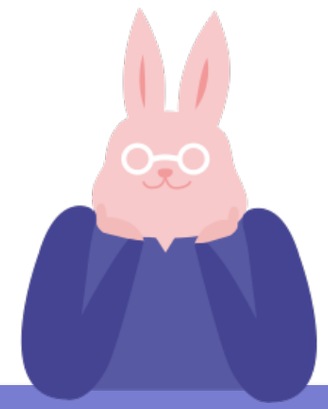
[활동문제 1] 달팽이 숫자 출력

1	2	3	4	5
				6
				7
				8
13	12	11	10	9



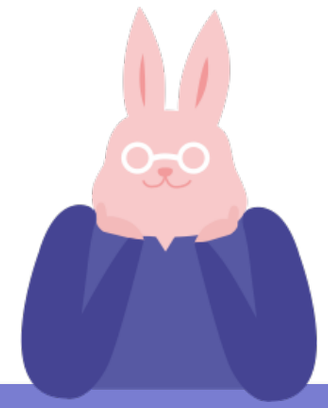
[활동문제 1] 달팽이 숫자 출력

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



[활동문제 1] 달팽이 숫자 출력

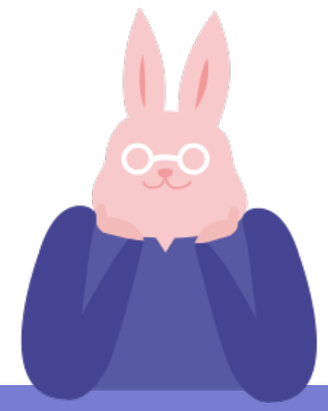
1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9



/* elice */

[활동문제 1] 달팽이 숫자 출력

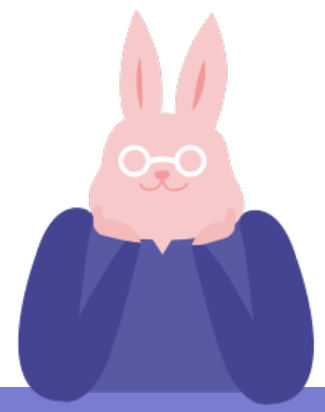
```
y = 0
x = 0
number = 1
While (number <= n*n) :
    arr[y][x] = number
    number = number + 1

    (y, x) = getNextCoordinate()
```



[활동문제 1] 달팽이 숫자 출력

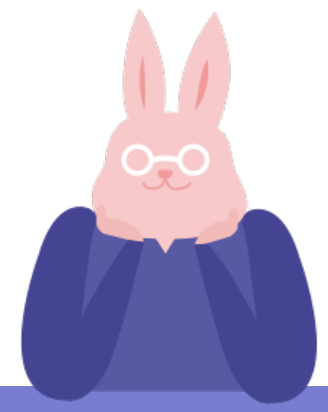
```
y = 0
x = 0
number = 1

While (number <= n*n) :
    arr[y][x] = number
    number = number + 1

    (y, x, dir) =
        getNextCoordinate(y, x, dir)
```

```
getNextCoordinate(y, x, dir):
 while (True) :
  (dy, dx) = getDyDx(dir)

  if isOkay(arr, y+dy, x+dx):
    return (y+dy, x+dx, dir)
  else :
    dir = (dir + 1) % 4
```



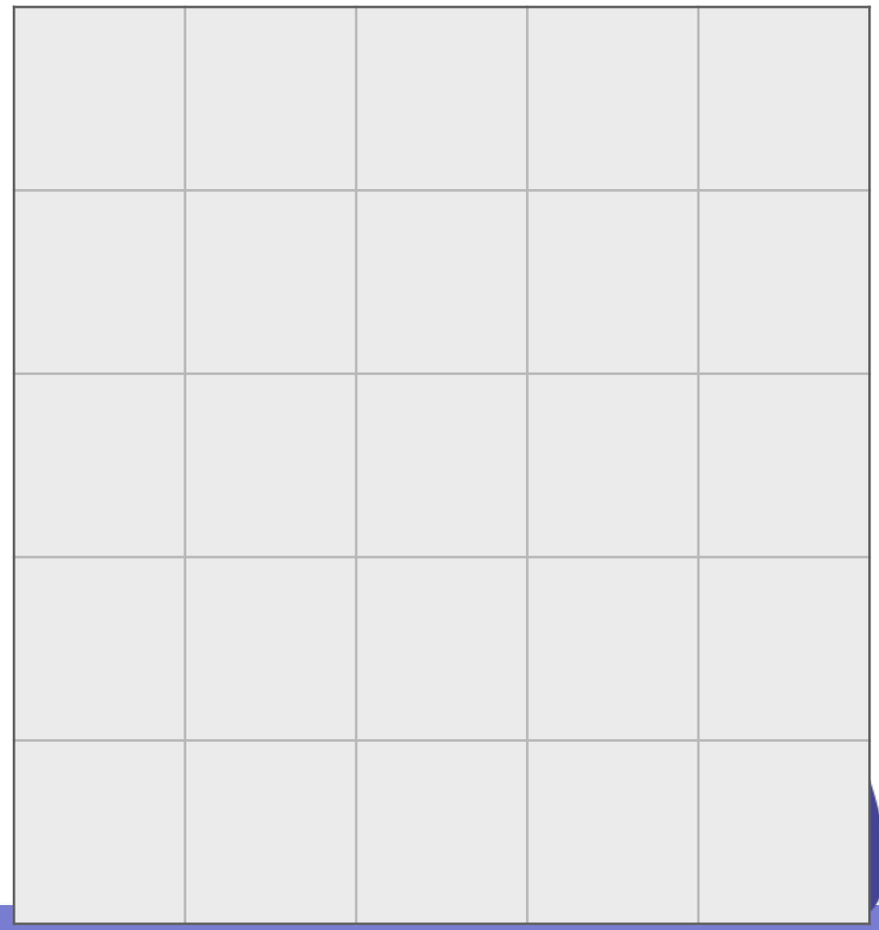
[활동문제 1] 달팽이 숫자 출력

- Recursion



[활동문제 1] 달팽이 숫자 출력

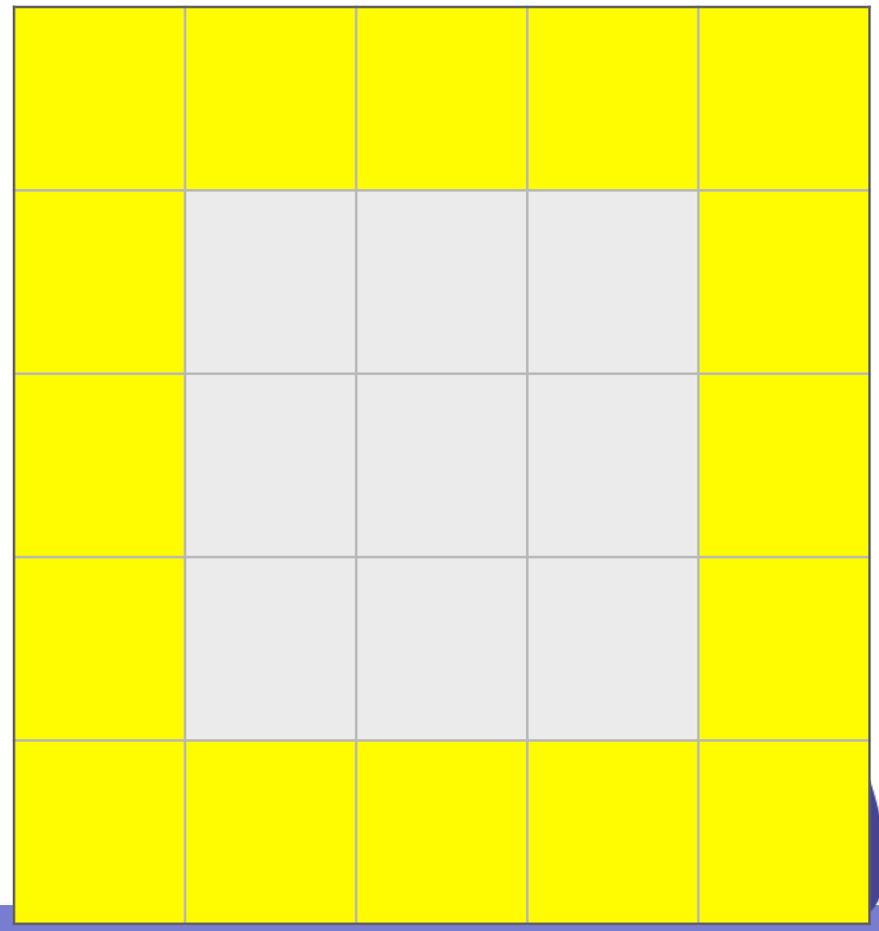
- Recursion
  - Snail(arr, y, x, n)
    : (y, x)를 (0, 0)으로 하여 크기 n의 달팽이 숫자를 저장하는 함수



[활동문제 1] 달팽이 숫자 출력

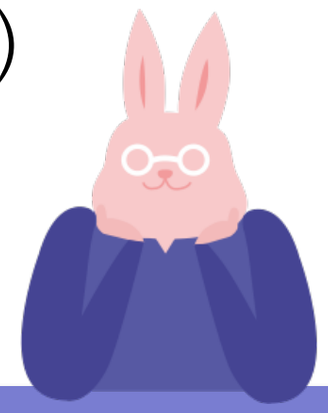
- Recursion
  - Snail(arr, y, x, n)
    : (y, x)를 (0, 0)으로 하여 크기 n의 달팽이 숫자를 저장하는 함수
  - Base condition
    - n == 1 or n == 2
  - Snail 계산
    - Snail(arr, y, x, n) =
      getYellowPart(y, x, n)

      Snail(arr, y+1, x+1, n-2)



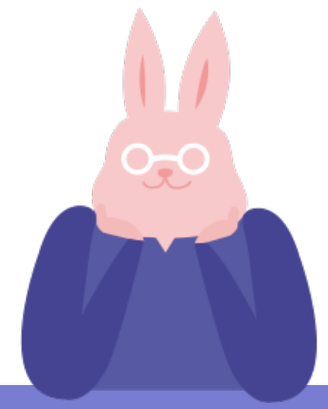
커리큘럼

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)



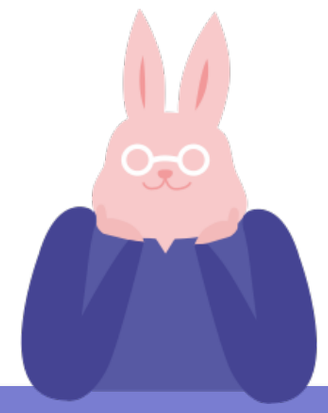
강의 구성

- 알고리즘의 정확성 증명과 시간복잡도
  - 유클리드 호제법
  - 약수 구하기
  - 소수 구하기
- 자료구조
  - 스택
  - 큐
  - 트리
  - 그래프
- 트리순회
- 트리구조의 이용 : 힙
- 그래프 순회
  - 깊이우선탐색
  - 너비우선탐색
- 그래프 순회의 이용
  - 미로찾기
  - Flood Fill



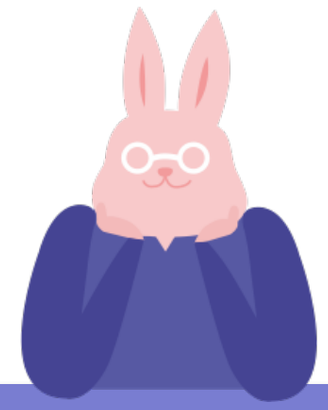
문제를 푸는 과정

1. 문제를 정확히 이해한다
2. 알고리즘을 개발한다
3. 알고리즘이 문제를 제대로 푼다는 것을 증명한다
4. 알고리즘이 제한 시간 내에 동작한다는 것을 보인다
5. 알고리즘을 직접 코드로 작성한다
6. ~~제출 후 틀렸다는 것을 확인하고 디버깅한다~~
   제출 후 만점이라는 것을 확인하고 행복해 한다



문제를 푸는 과정

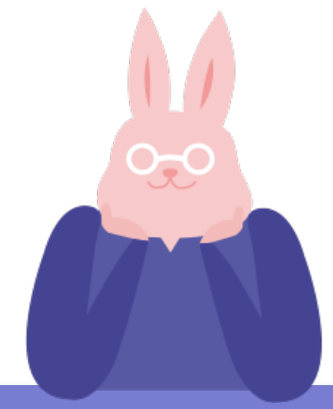
1. 문제를 정확히 이해한다
2. 알고리즘을 개발한다
3. **알고리즘이 문제를 제대로 푼다는 것을 증명한다**
4. **알고리즘이 제한 시간 내에 동작한다는 것을 보인다**
5. 알고리즘을 직접 코드로 작성한다
6. ~~제출 후 틀렸다는 것을 확인하고 디버깅한다~~
   제출 후 만점이라는 것을 확인하고 행복해 한다



알고리즘의 정확성 증명

- 주어진 알고리즘이 문제에 대한 정답임을 증명

GCD(x, y) {
  y if x mod y == 0
  GCD(y, x mod y) otherwise
}



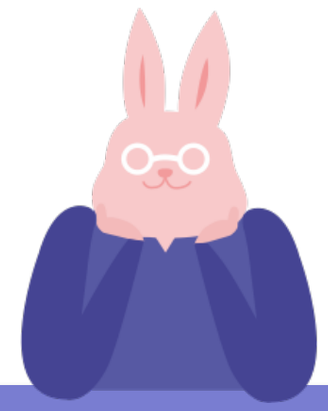
유클리드 호제법의 증명

GCD(x, y) = GCD(y, x mod y)　　단, **x ≥ y**

주장 : GCD(x, y) = GCD(x-y, y)

증명

1. x, y의 공약수 집합는 x-y, y의 공약수 집합과 같다
2. 따라서 GCD(x, y)와 GCD(x-y, y)는 같다

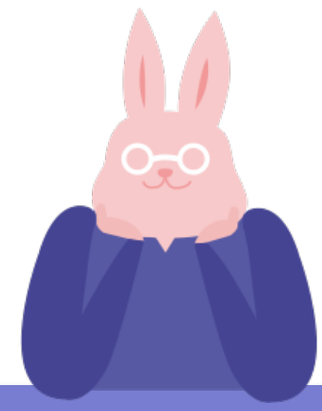


유클리드 호제법의 증명

1. x, y의 공약수 집합은 x-y, y의 공약수 집합과 같다

   => (a가 x, y를 모두 나누면, a는 x-y, y를 모두 나눈다)

   a가 x, y를 모두 나눈다고 가정하자. 그러면 x = ax', y = ay' 라 할 수 있다.
   따라서 x-y = ax' - ay' = a(x' - y') 이므로, x-y는 a로 나누어 떨어진다.
   (x ≥ y 이므로, x'-y' ≥ 0 이다)

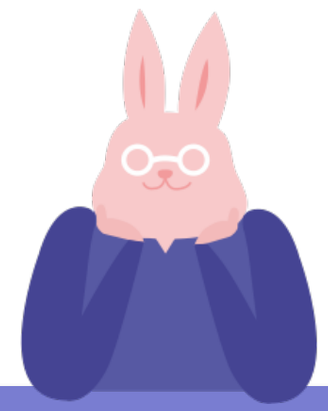


유클리드 호제법의 증명

1. x, y의 공약수 집합은 x-y, y의 공약수 집합과 같다

   <= (a가 x-y, y를 모두 나누면, a는 x, y를 모두 나눈다)

   a가 x-y, y를 모두 나눈다고 가정하자. 그러면 x-y = at', y = ay' 라 할 수 있다.
   따라서 x = at' + ay' = a(t' + y')이므로, x는 a로 나누어 떨어진다.



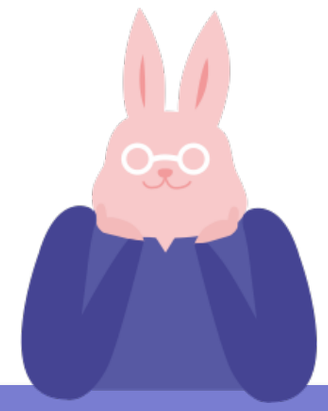
유클리드 호제법의 증명

GCD(x, y) = GCD(y, x mod y) 단, **x ≥ y**

주장 : GCD(x, y) = GCD(x-y, y)

증명

1. x, y의 공약수 집합는 x-y, y의 공약수 집합과 같다
2. 따라서 GCD(x, y)와 GCD(x-y, y)는 같다



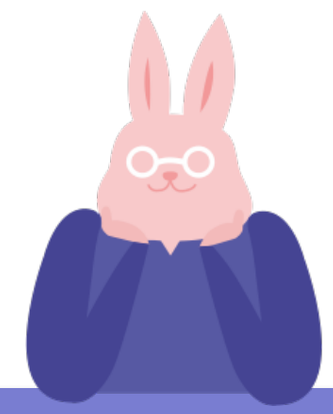
시간복잡도

- 알고리즘이 대략 몇 번의 연산을 필요로 하는가 ?

```
for i in range(n) :
    sum = sum + 1
```

```
for i in range(n) :
    for j in range(n) :
        sum = sum + 1
```

```
for i in range(n) :
    for j in range(i) :
        sum = sum + 1
```



실험

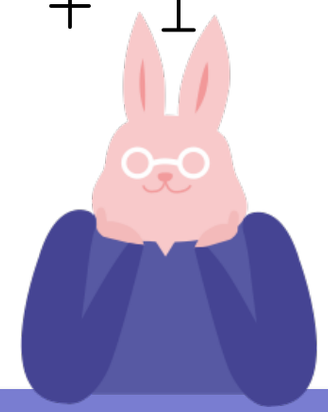
- 아래 코드에 대하여 Elice에서 수행 시간을 측정

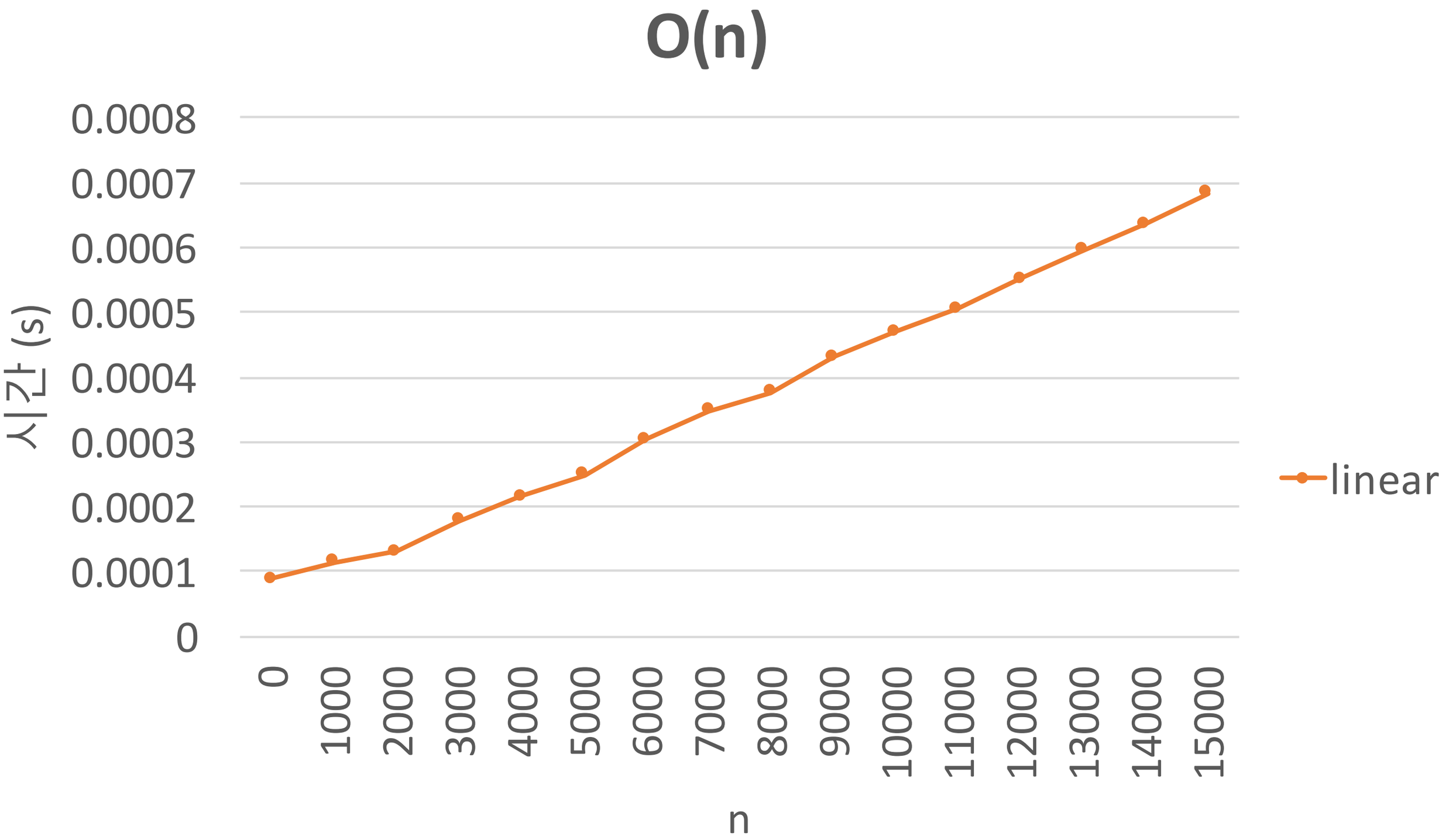
Linear, O(n)

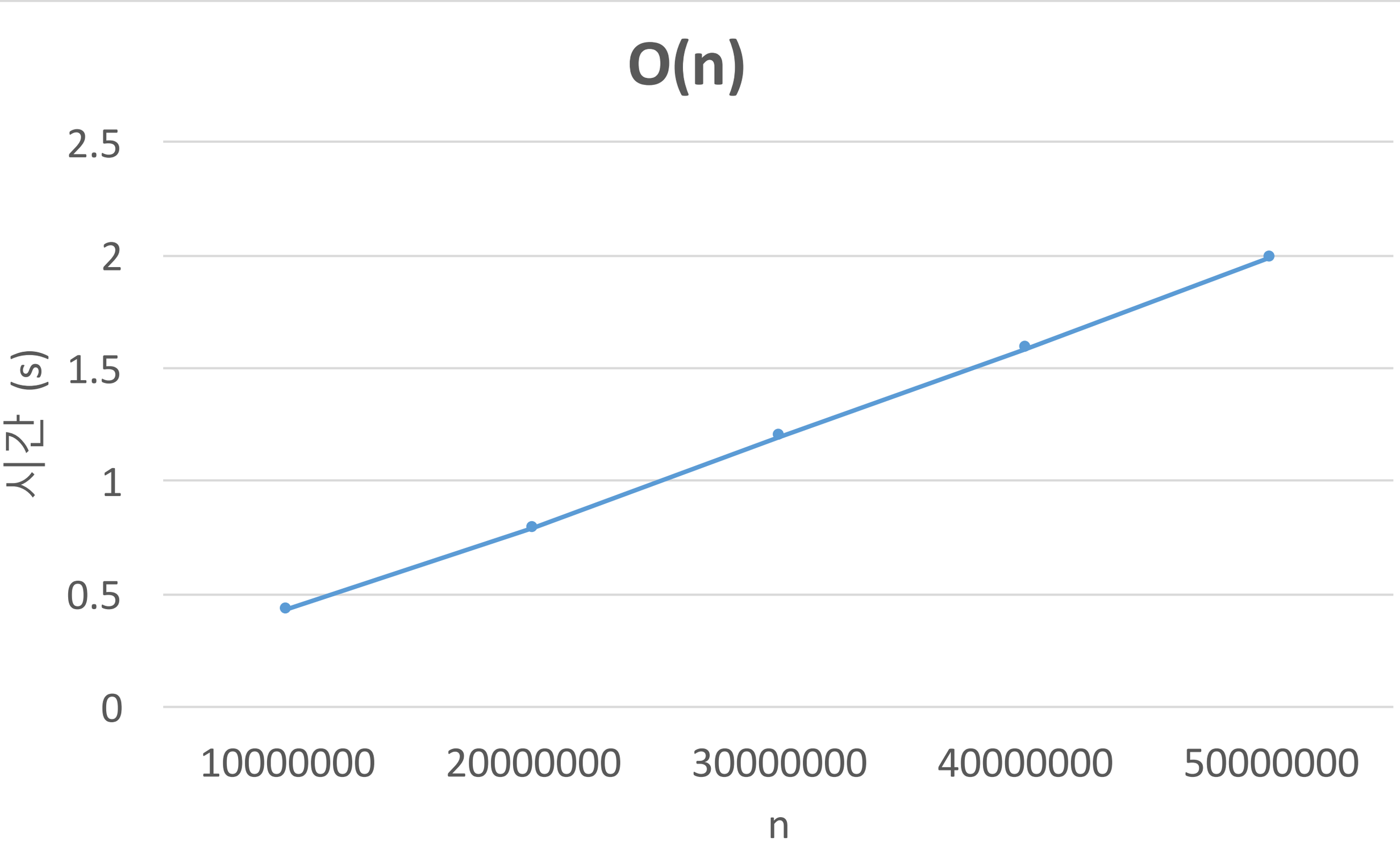
```
for i in range(n) :
    sum = sum + 1
```

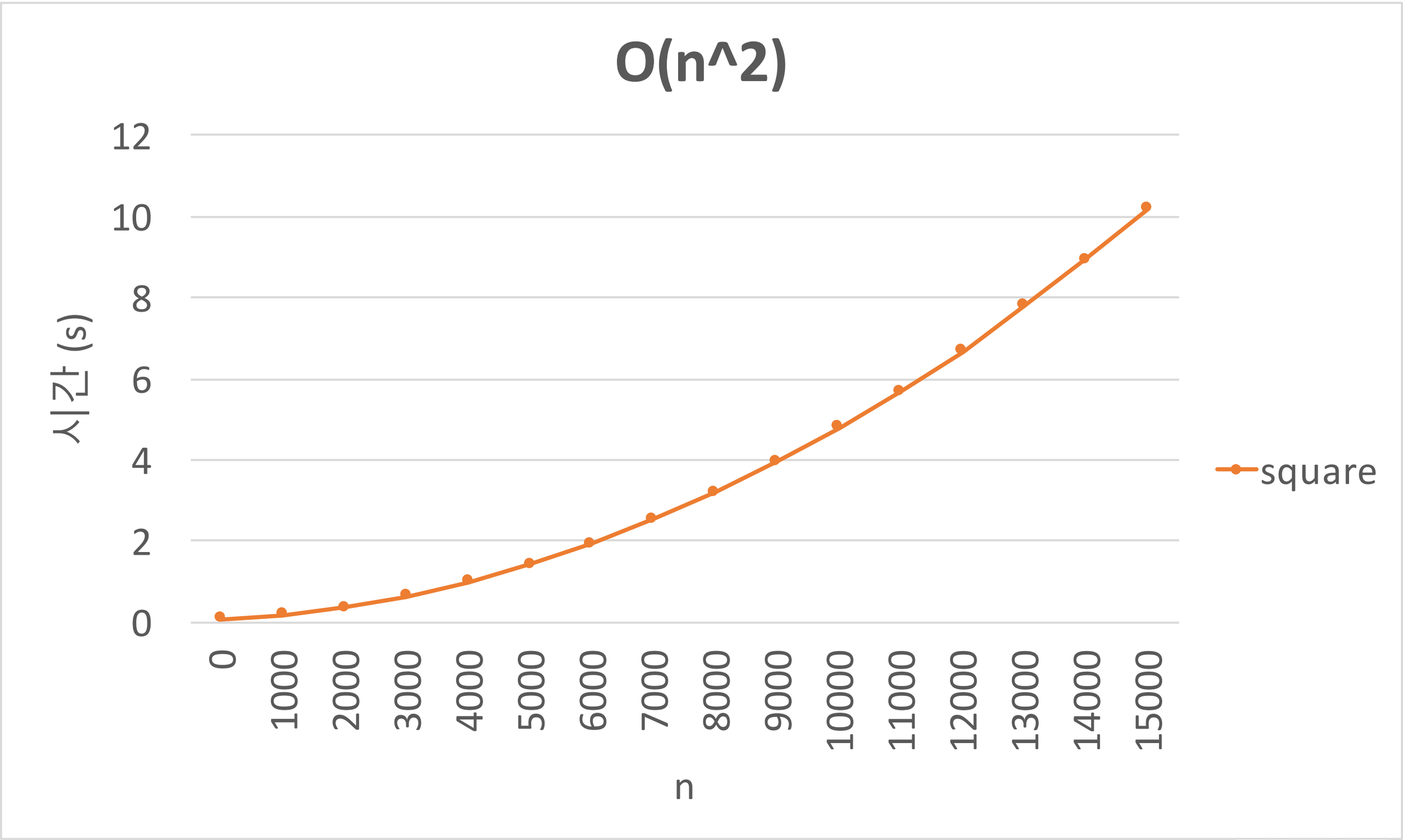
Square, O(n^2)

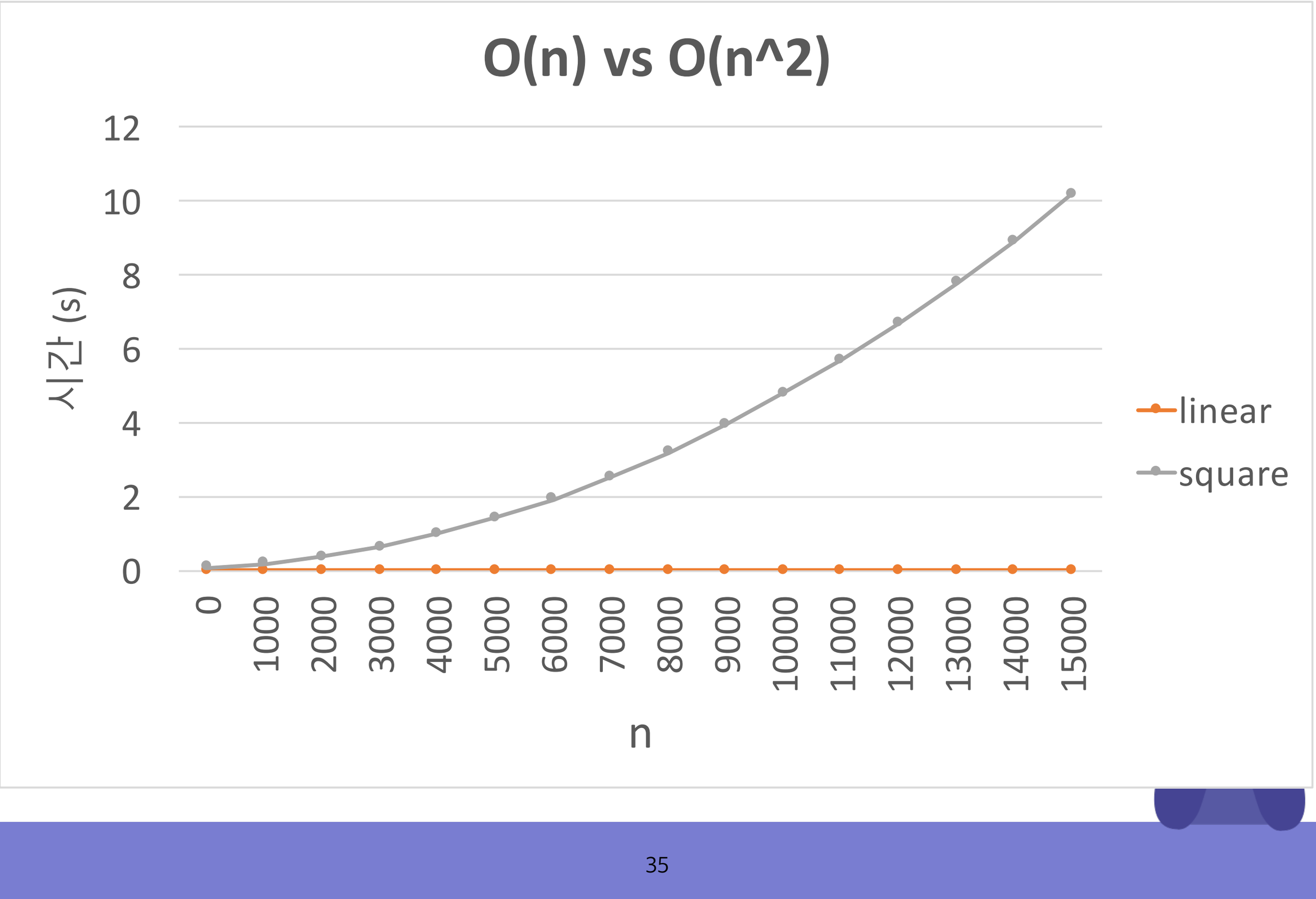
```
for i in range(n) :
    for j in range(n) :
        sum = sum + 1
```




O(n)
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
시간 (s)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
n
linear

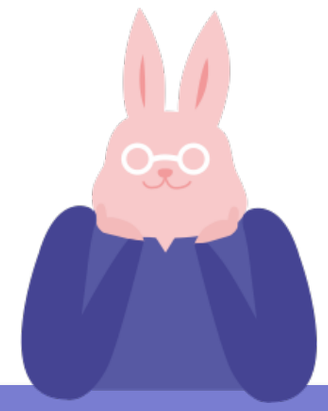

O(n)
2.5
2
1.5
1
0.5
0
시간 (s)
10000000
20000000
30000000
40000000
50000000
n


O(n^2)
12
10
8
6
4
2
0
시간 (s)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
n
square


O(n) vs O(n^2)
12
10
8
6
4
2
0
시간 (s)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
n
linear
square

1초에 연산할 수 있는 횟수

- 약 1000만번의 연산은 1초 내에 나온다
- 본인 알고리즘의 시간복잡도를 계산해보고, 대략 1초가 걸릴지 예상해 보아야 함



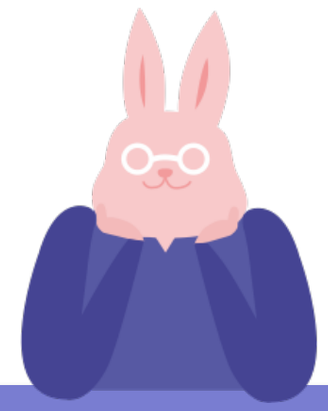
유클리드 호제법의 시간복잡도

$$\mathbf{GCD(x, y) = GCD(y, x \bmod y)} \qquad \text{단, } \mathbf{x \geq y}$$

주장 : 만약 $x \geq y$ 이면, $x \bmod y < x/2$ 이다.

증명 :

1. $y \geq x/2$인 경우 주장이 성립한다.
2. $y < x/2$인 경우 주장이 성립한다.



유클리드 호제법의 시간복잡도

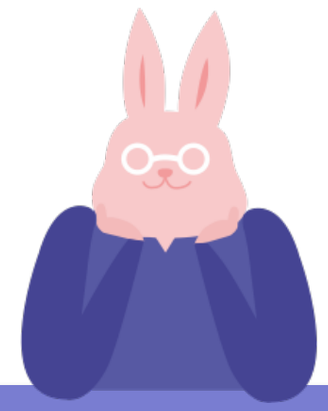
GCD(x, y) = GCD(y, x mod y) 단, **x ≥ y**

주장 : 만약 x ≥ y 이면, x mod y < x/2 이다.

증명 :

1. y ≥ x/2인 경우 주장이 성립한다.
2. y < x/2인 경우 주장이 성립한다.

x mod y = x - y < x/2



유클리드 호제법의 시간복잡도

GCD(x, y) = GCD(y, x mod y) 단, **x ≥ y**

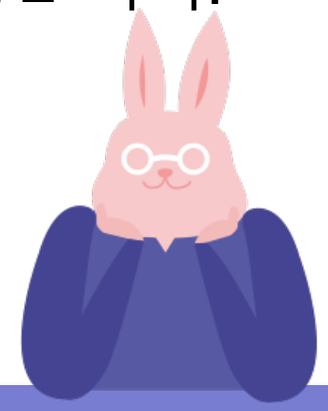
주장 : 만약 x ≥ y 이면, x mod y < x/2 이다.

증명 :

1. y ≥ x/2인 경우 주장이 성립한다.
2. y < x/2인 경우 주장이 성립한다.

나머지는 y보다 작아야 하므로

x mod y < y < x/2 이다.



유클리드 호제법의 시간복잡도

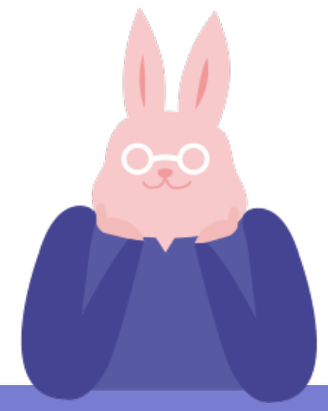
$$\mathbf{GCD(x, y) = GCD(y, x \bmod y)} \qquad \text{단, } \mathbf{x \geq y}$$

주장 : 만약 $x \geq y$ 이면, $x \bmod y < x/2$ 이다.

- 즉, 두 번의 나머지 연산 과정을 거치면 x, y가 적어도 절반으로 떨어진다

GCD(573, 238) = GCD(228, 97) = GCD(97, 34) = …

x, y 모두 절반 이하로 떨어짐!



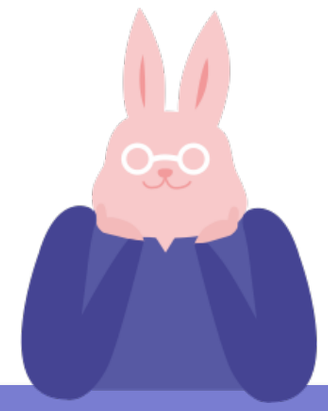
유클리드 호제법의 시간복잡도

$$\mathbf{GCD(x, y) = GCD(y, x \bmod y)} \qquad \text{단, } \mathbf{x \geq y}$$

주장 : 만약 $x \geq y$ 이면, $x \bmod y < x/2$ 이다.

- 즉, 두 번의 나머지 연산 과정을 거치면 x, y가 적어도 절반으로 떨어진다
  - 계속 절반으로 떨어지다가 y가 1까지 떨어지면 멈춘다

- 따라서 시간복잡도는 **O(log y)**이다.

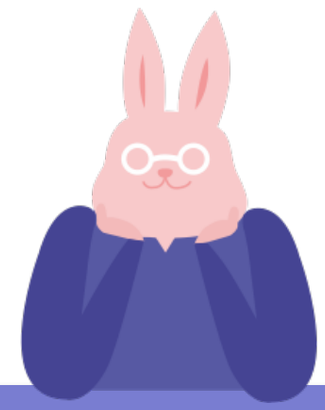


왜 유클리드 호제법을 쓰는가 ?

- 단순하게 최대공약수를 구하는 방법은 O(y)가 걸린다

```
result = -1

for i in range(1, y+1, 1) :
    if y % i == 0 and x % i == 0 :
        result = max(result, i)

print result
```

단순 최대공약수 알고리즘



왜 유클리드 호제법을 쓰는가 ?

- 유클리드 호제법은 O(log y)가 걸린다

```
result = -1

for i in range(1, y+1, 1) :
    if y % i == 0 and x % i == 0 :
        result = max(result, i)

print result
```

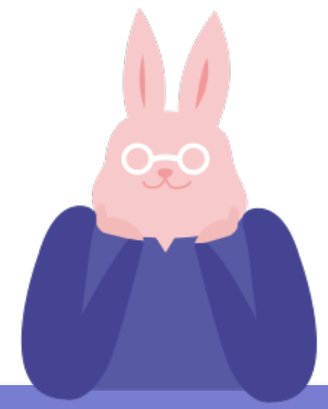
단순 최대공약수 알고리즘

O(y)

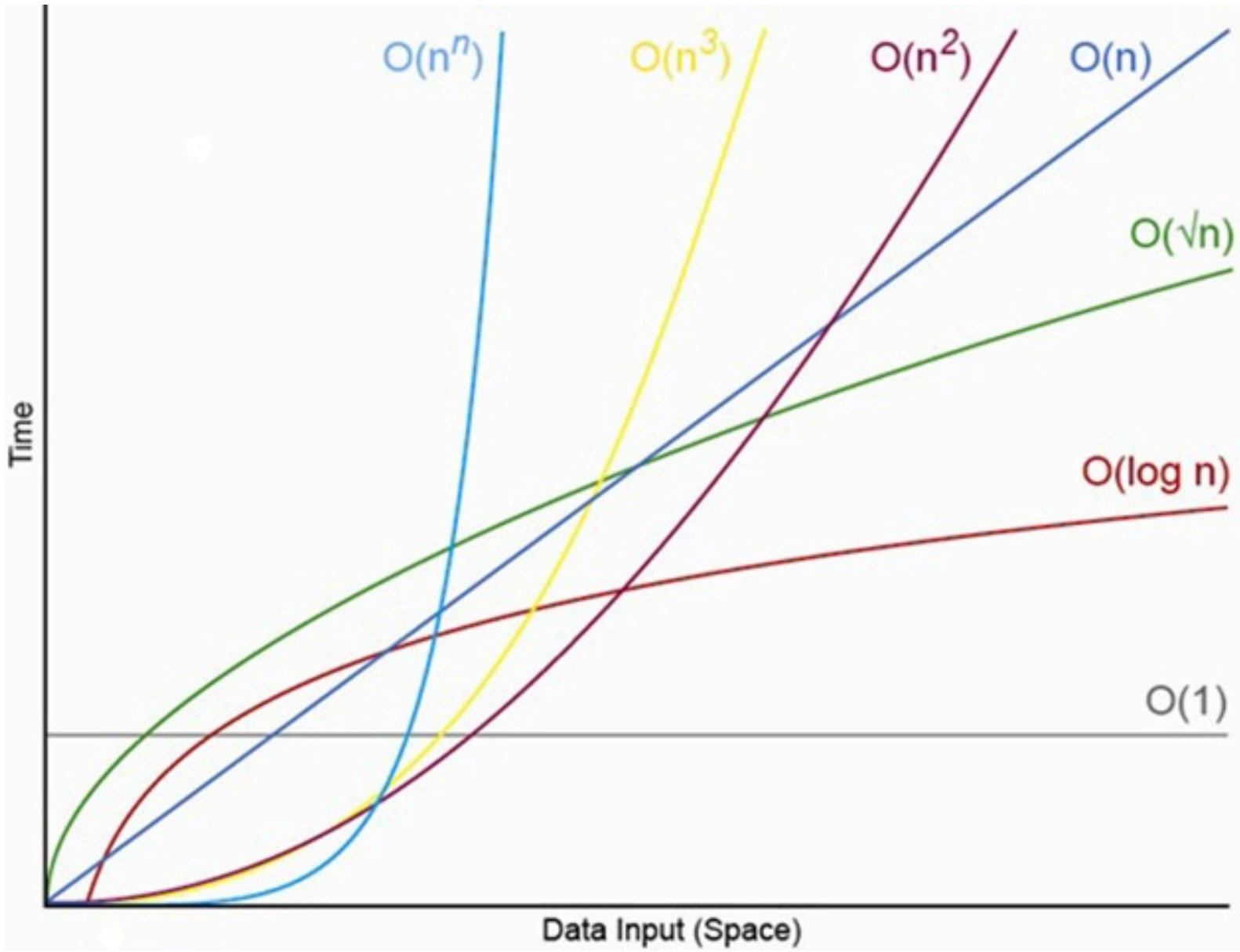
```
def GCD(a, b) :
    if a % b == 0 :
        return b

    return GCD(a, a%b)

print GCD(x, y)
```

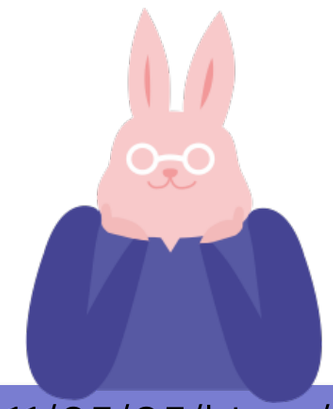
유클리드 호제법

O(log y)



O(n)과 O(log n)의 비교

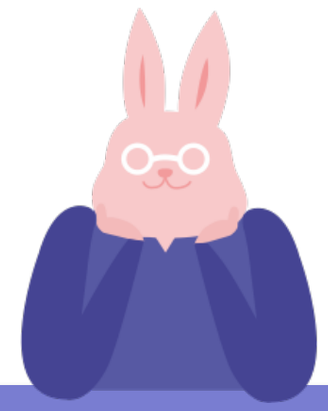




https://apelbaum.wordpress.com/2011/05/05/big-o/

O(n)과 O(log n)의 비교

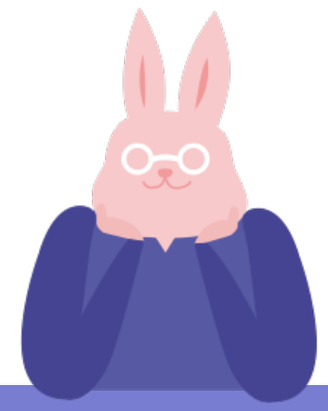
- GCD(10^30, 10^40)
  - O(n) 알고리즘 : 죽을때까지 결과가 나오지 않는다
  - O(log n) 알고리즘 : 1초도 안걸린다



약수 구하는 풀이의 시간복잡도

1. [lo, hi] 내의 모든 숫자의 약수의 개수를 하나씩 다 구하기

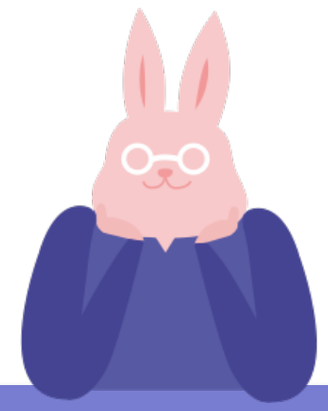
2. 에라토스테네스의 체를 이용하여 약수를 구하기



소수인지 판별하기

1. 하나의 숫자가 입력으로 주어지고, 소수인지 판별하기

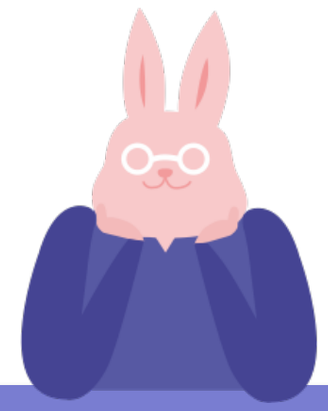
2. [lo, hi] 구간 내에 존재하는 소수 개수 구하기



소수인지 판별하기

1. 하나의 숫자가 입력으로 주어지고, 소수인지 판별하기
    - 2부터 (n-1) 까지 모든 수로 나누어본다 : O(n)

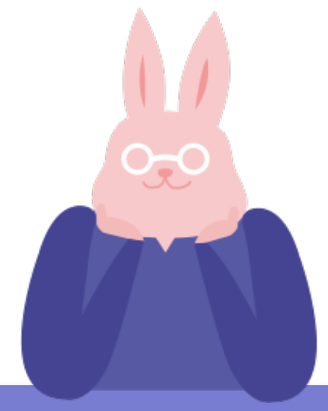
2. [lo, hi] 구간 내에 존재하는 소수 개수 구하기



소수인지 판별하기

1. 하나의 숫자가 입력으로 주어지고, 소수인지 판별하기
   - 2부터 (n-1) 까지 모든 수로 나누어본다 : O(n)
   - 2부터 (루트 n) 까지 모든 수로 나누어본다 : O(sqrt(n))

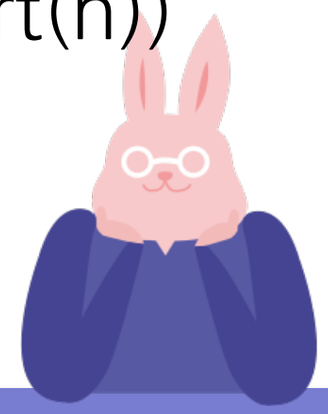
2. [lo, hi] 구간 내에 존재하는 소수 개수 구하기



소수인지 판별하기

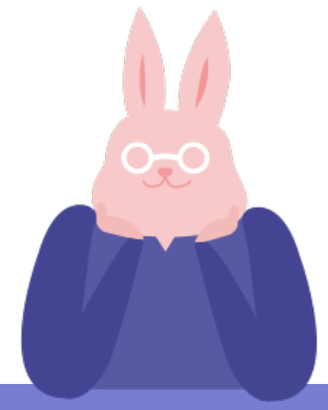
1. 하나의 숫자가 입력으로 주어지고, 소수인지 판별하기
   - 2부터 (n-1) 까지 모든 수로 나누어본다 : O(n)
   - 2부터 (루트 n) 까지 모든 수로 나누어본다 : O(sqrt(n))

2. [lo, hi] 구간 내에 존재하는 소수 개수 구하기
   - 구간 내의 숫자에 대해서 각각 소수인지 판별
   - 구간 내의 숫자에 대해서 각각 소수인지 판별 : O(n * sqrt(n))
   - 에라토스테네스의 체를 이용 : O(n log n)

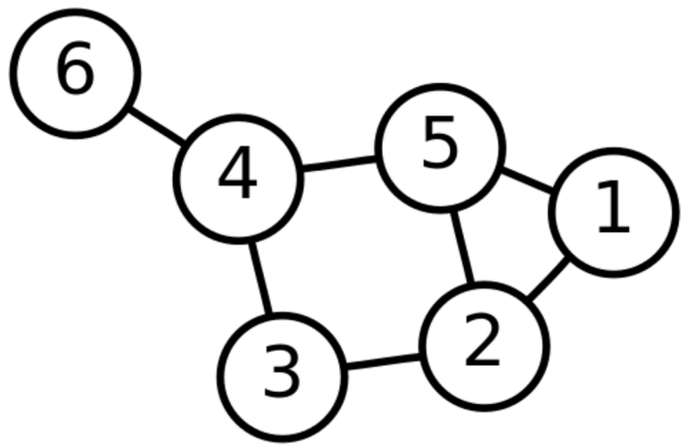


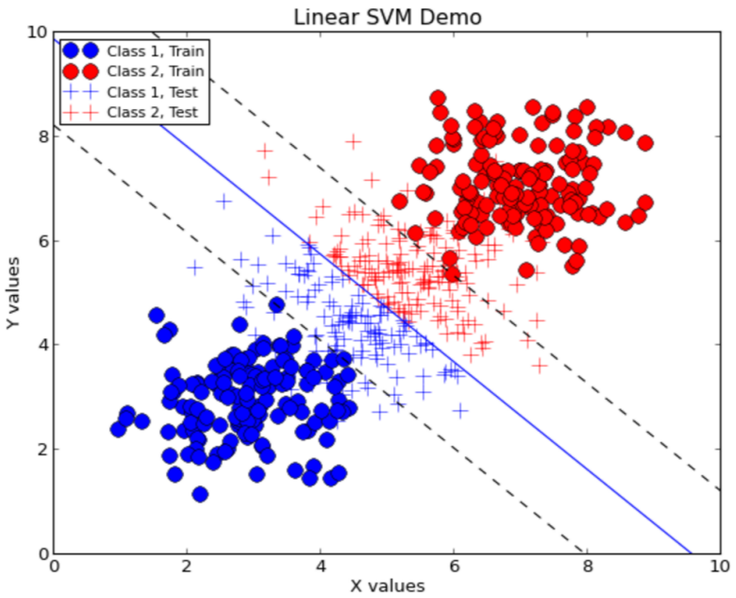
문제를 푸는 과정 (Remind)

1. 문제를 정확히 이해한다
2. 알고리즘을 개발한다
3. **알고리즘이 문제를 제대로 푼다는 것을 증명한다**
4. **알고리즘이 제한 시간 내에 동작한다는 것을 보인다**
5. 알고리즘을 직접 코드로 작성한다
6. ~~제출 후 틀렸다는 것을 확인하고 디버깅한다~~
   제출 후 만점이라는 것을 확인하고 행복해 한다



컴퓨터공학의 기본 커리큘럼





1. 프로그래밍 언어

C/C++
Python
Matlab

2. 자료구조

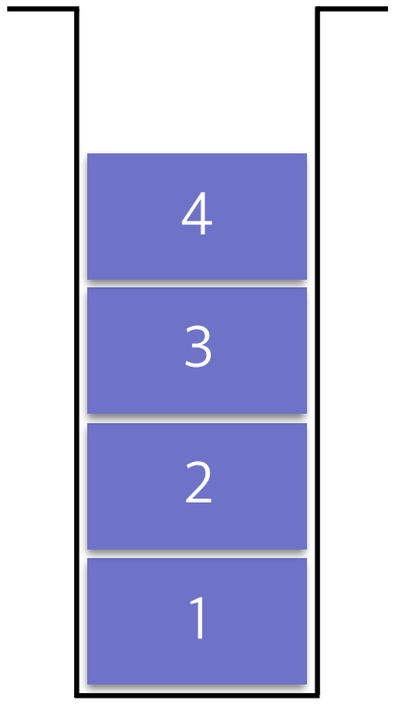
Stack
Queue
Tree

3. 알고리즘

Brute-Force
Divide & Conquer
Dynamic Programming

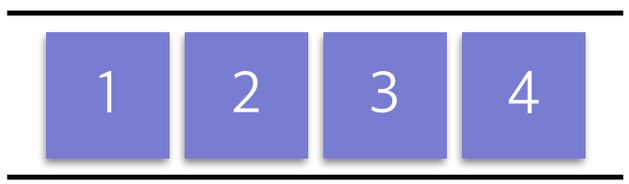
자료구조

- 자료를 저장하는 구조
  - 변수, 배열, list, …



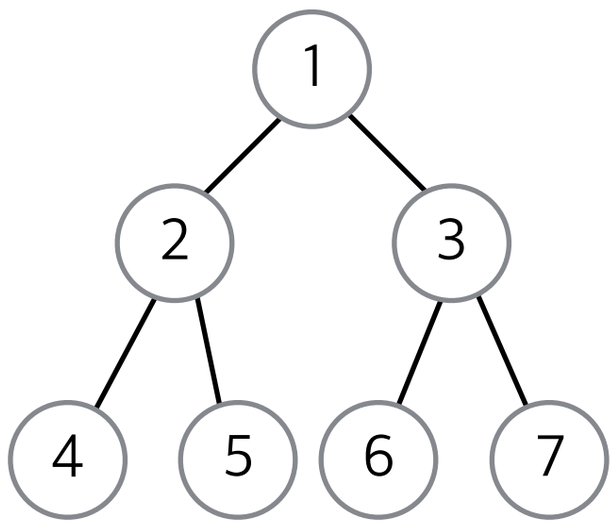
Stack

Last In First Out

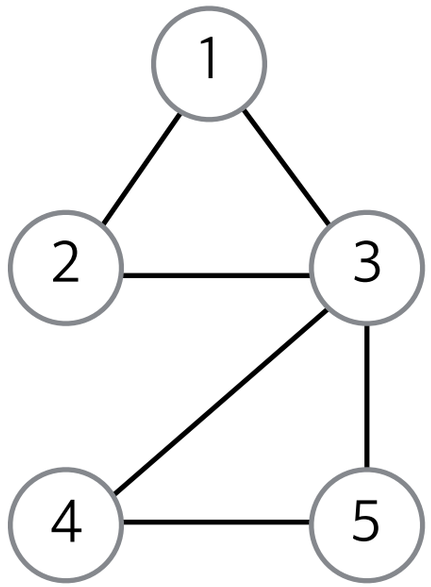


Queue

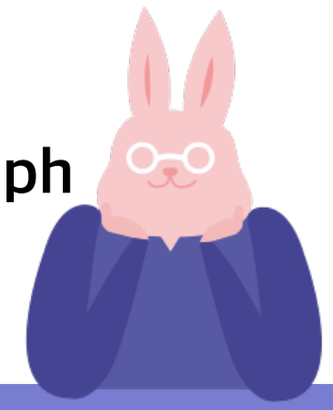
First In First Out



Tree

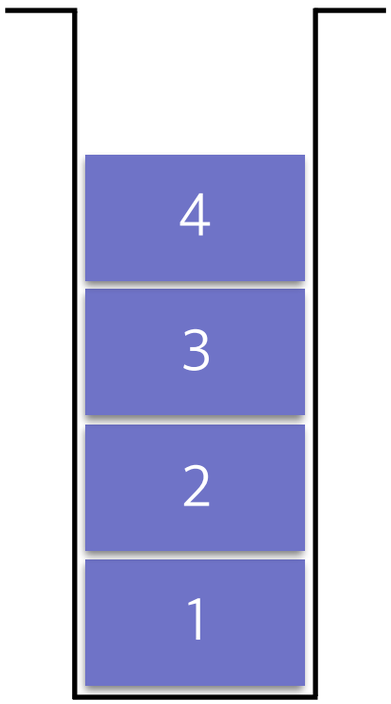


Graph



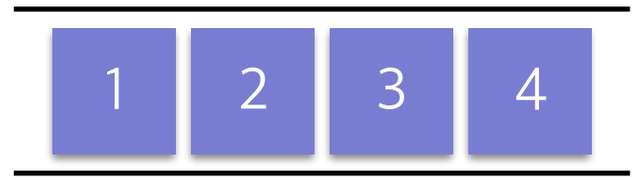
스택, 큐

- 중요한 연산 : Push, Pop



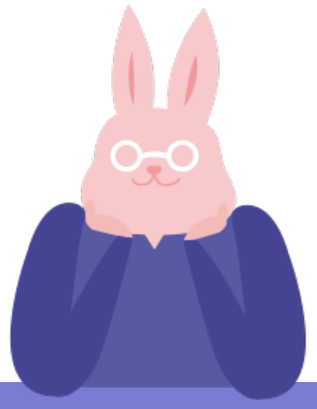
Stack

Last In First Out



Queue

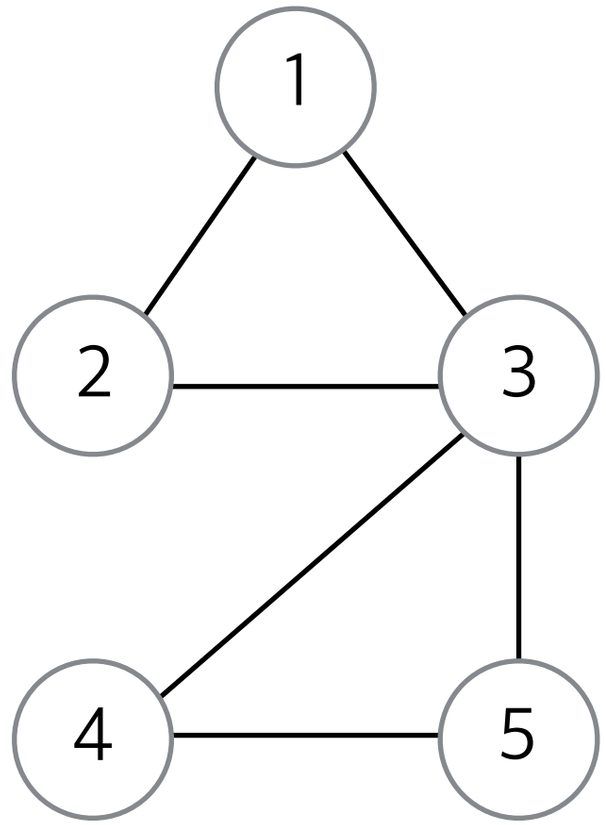
First In First Out



그래프

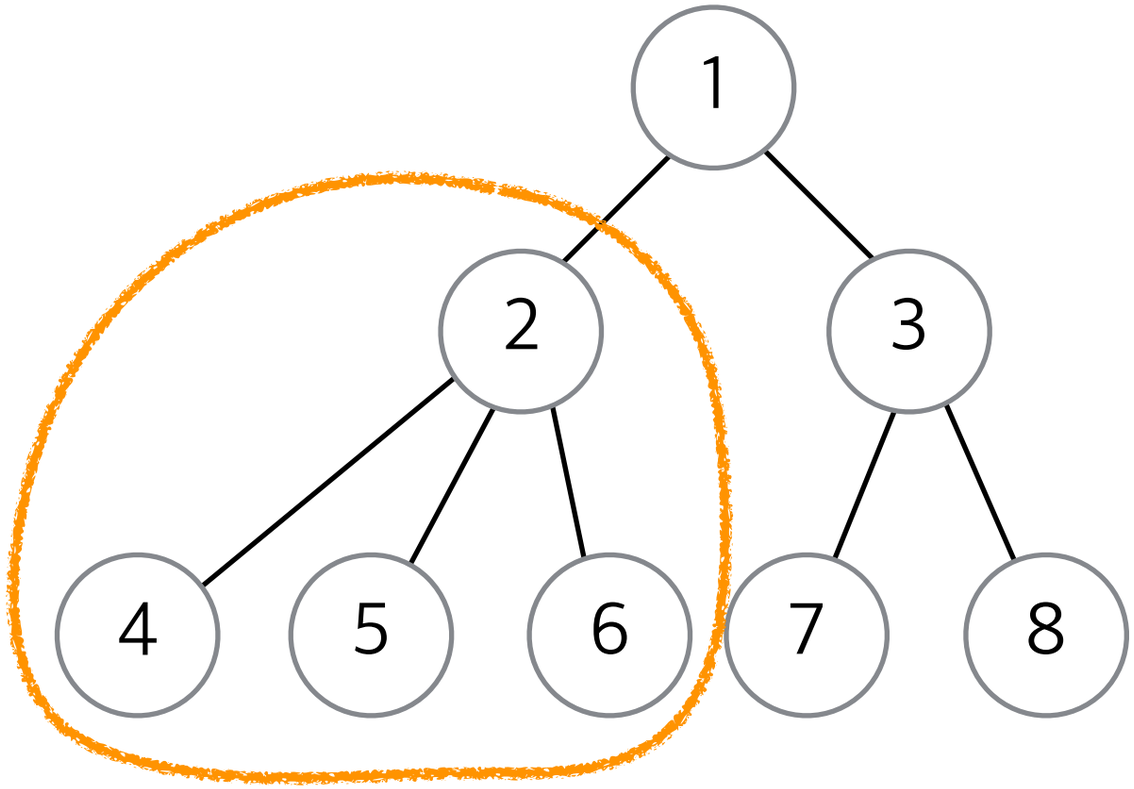
- Node와 Edge로 이루어져 있는 자료구조
  - Node와 Edge에 정보를 저장한다

- 용어
  - Path : 두 노드 사이의 길
  - Cycle : 자기 자신으로 돌아오는 길



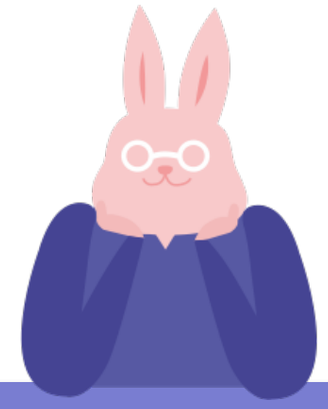
트리

- Cycle이 없는 graph



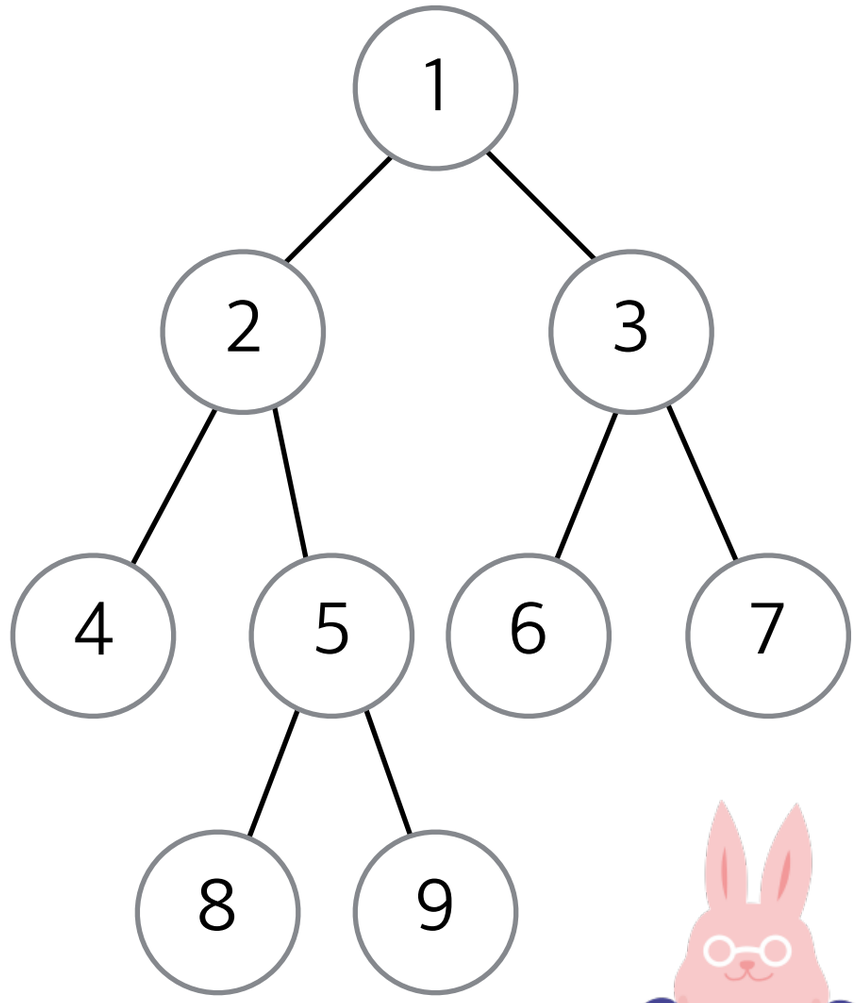
2를 루트로 하는 subtree

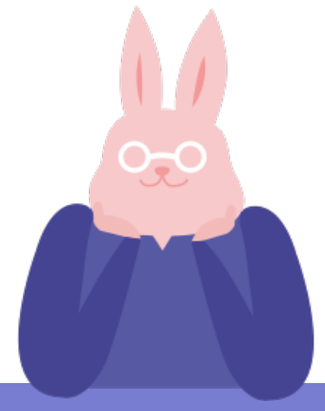
- 용어
  - Root : 가장 위에 있는 노드
  - Child : 자식 노드
  - Level : Root까지의 거리
  - Sub-tree : 트리의 일부로 이루어진 다른 트리



이진 트리

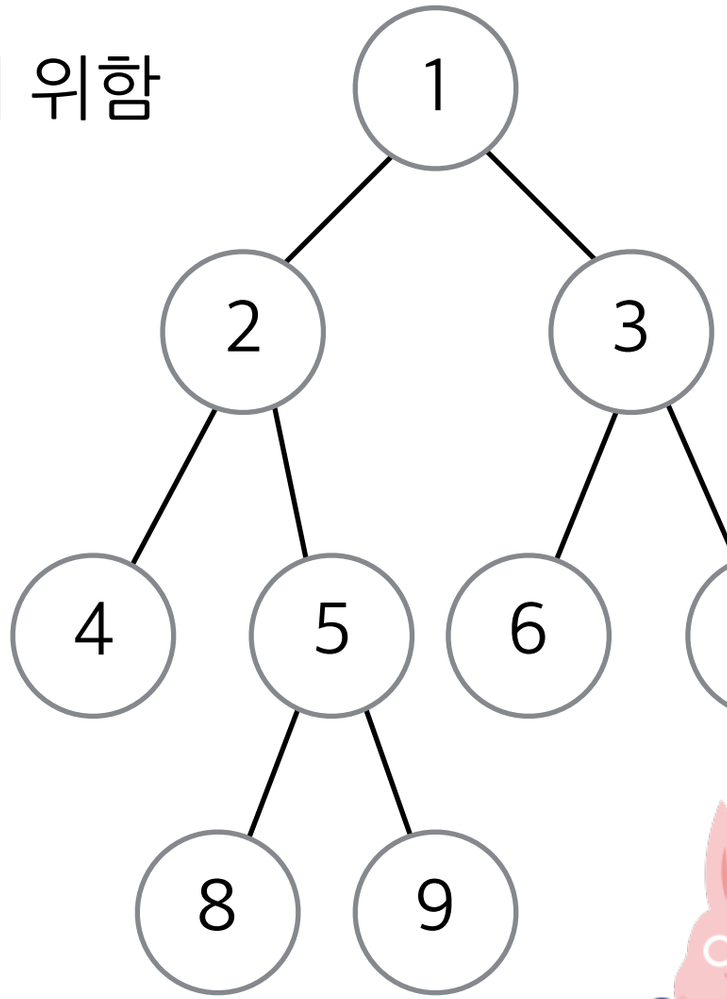
- 최대 두개의 자식노드만을 가지는 트리
- 용어
  - Left child : 왼쪽 자식노드
  - Right child : 오른쪽 자식노드



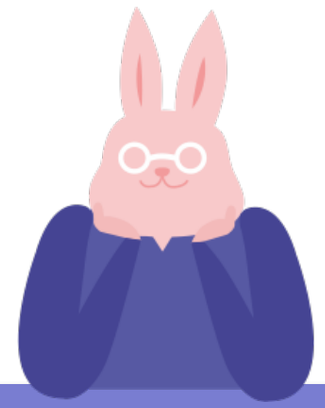


이진 트리 순회

- 이진트리의 Node 들을 순회하는 알고리즘
  - 이진트리에 무슨 값이 저장되어 있는지 알기 위함

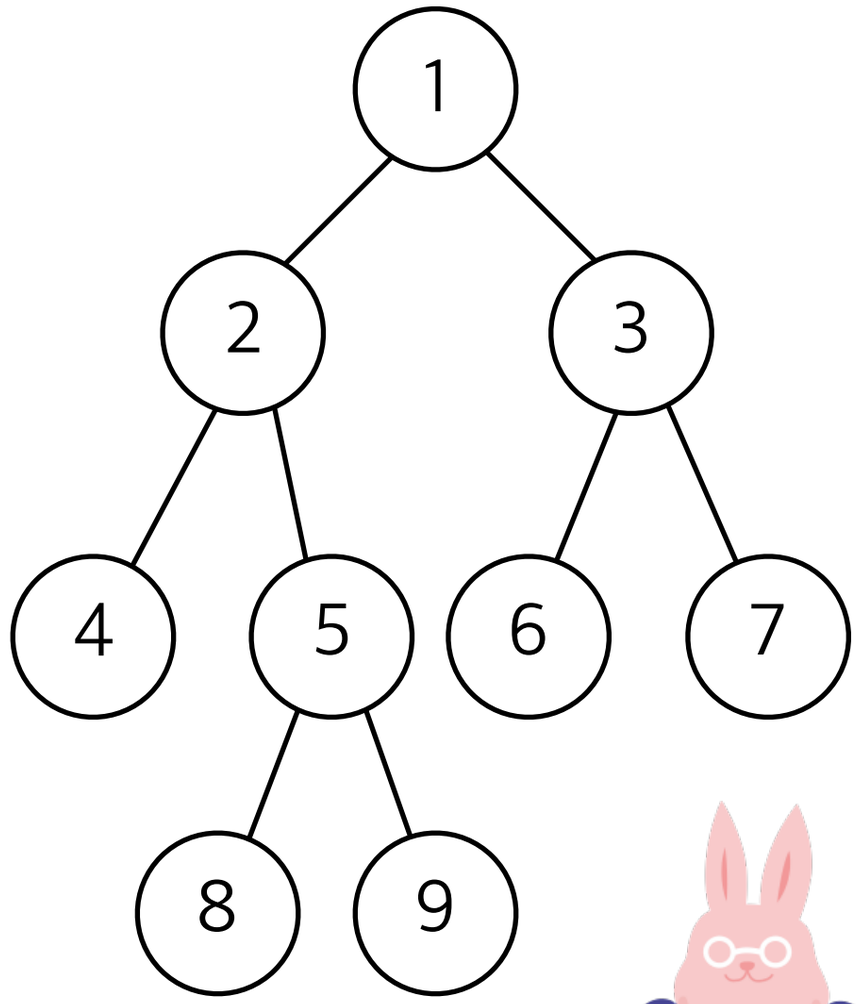


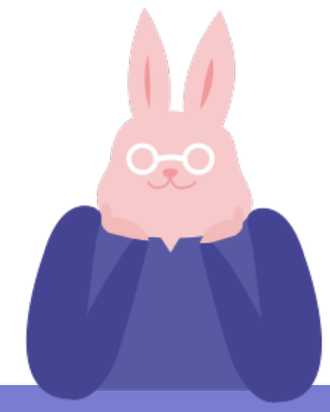
- 세 가지 방법
  1. 전위순회 (Pre-order)
  2. 중위순회 (In-order)
  3. 후위순회 (Post-order)



이진 트리 순회: 1. 전위순회

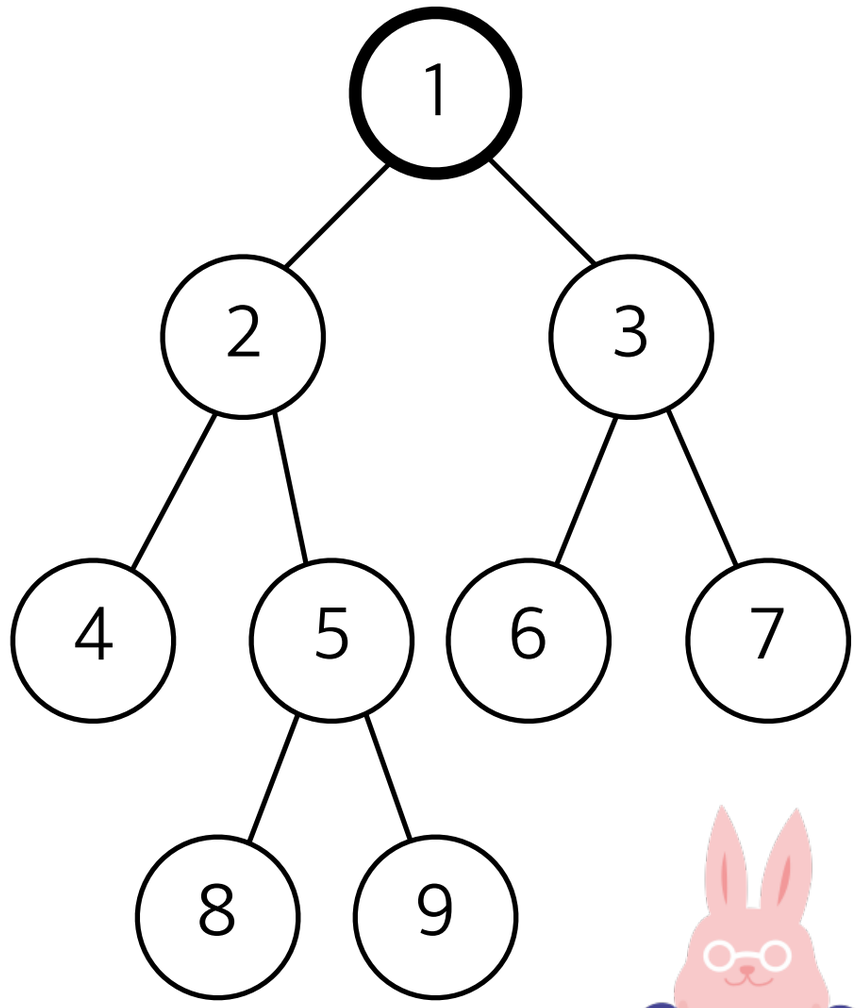
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

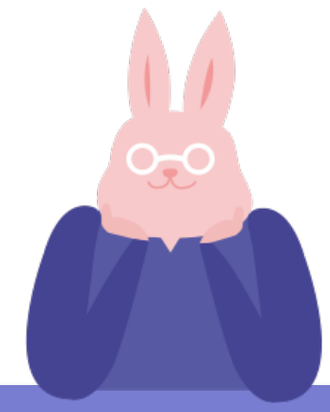




이진 트리 순회: 1. 전위순회

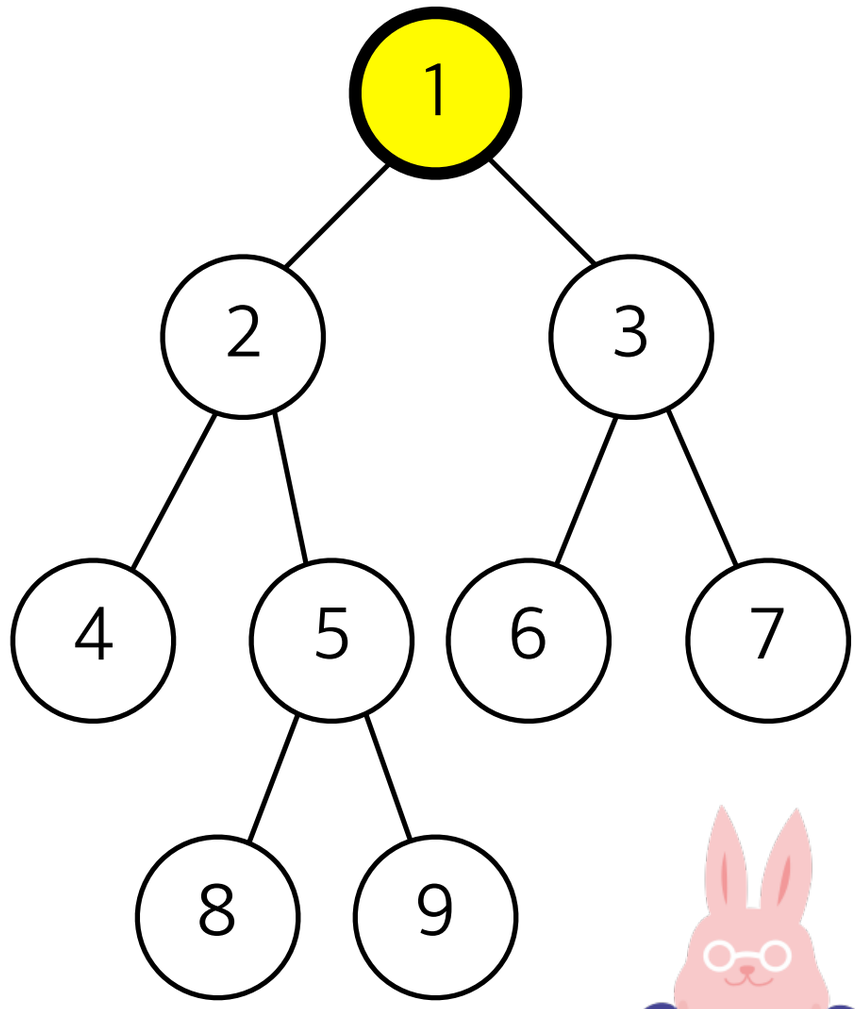
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠





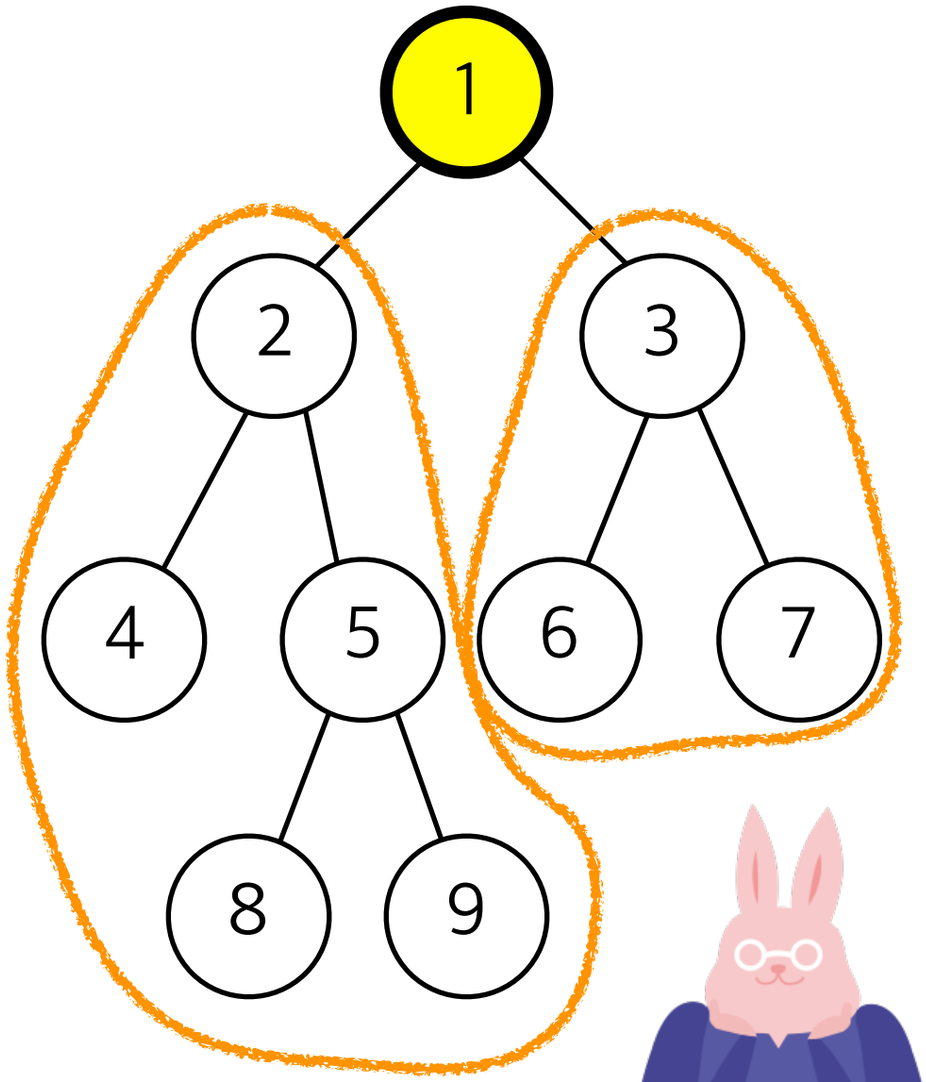
이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



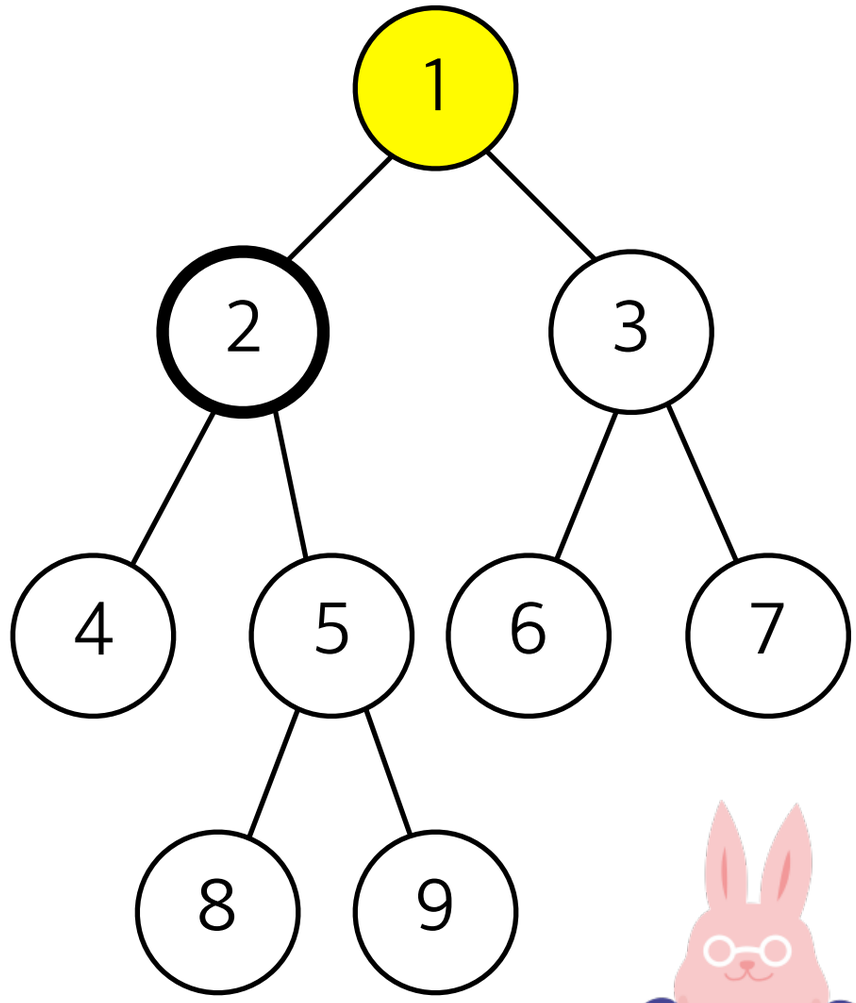
이진 트리 순회: 1. 전위순회

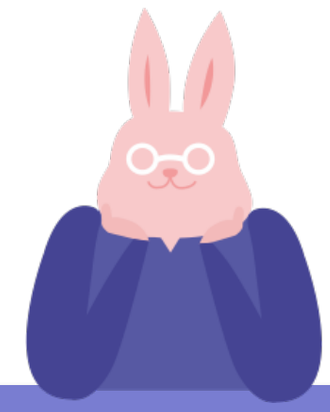
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



이진 트리 순회: 1. 전위순회

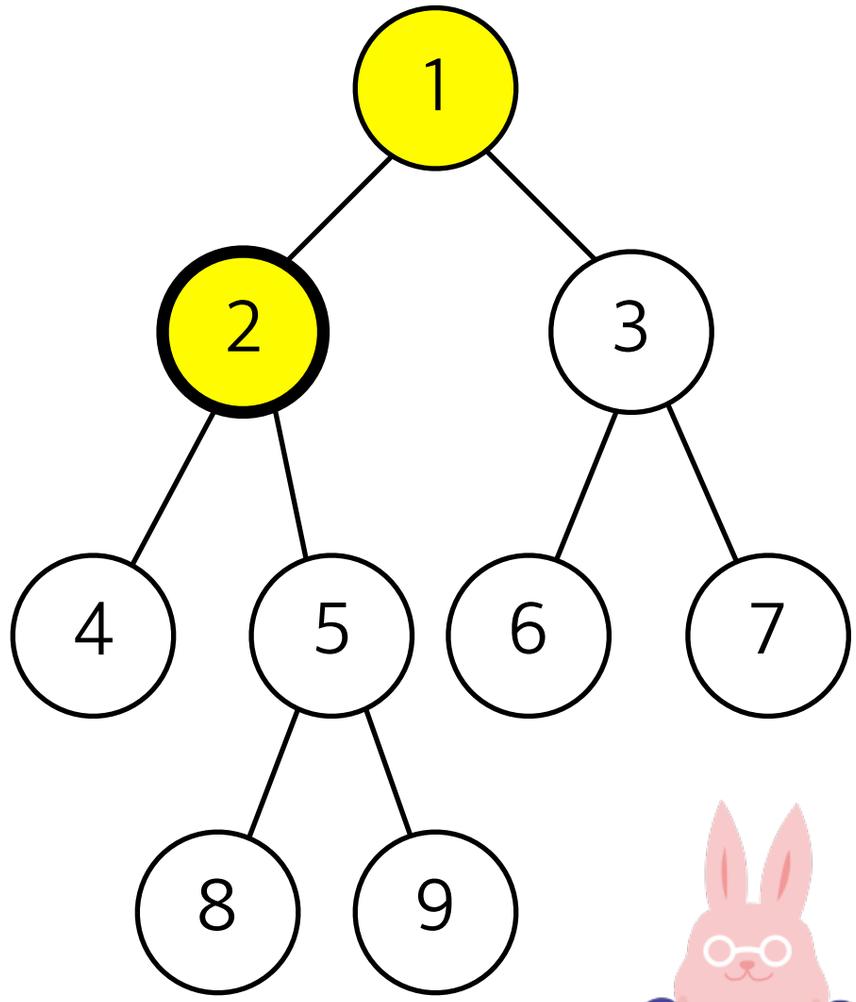
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠





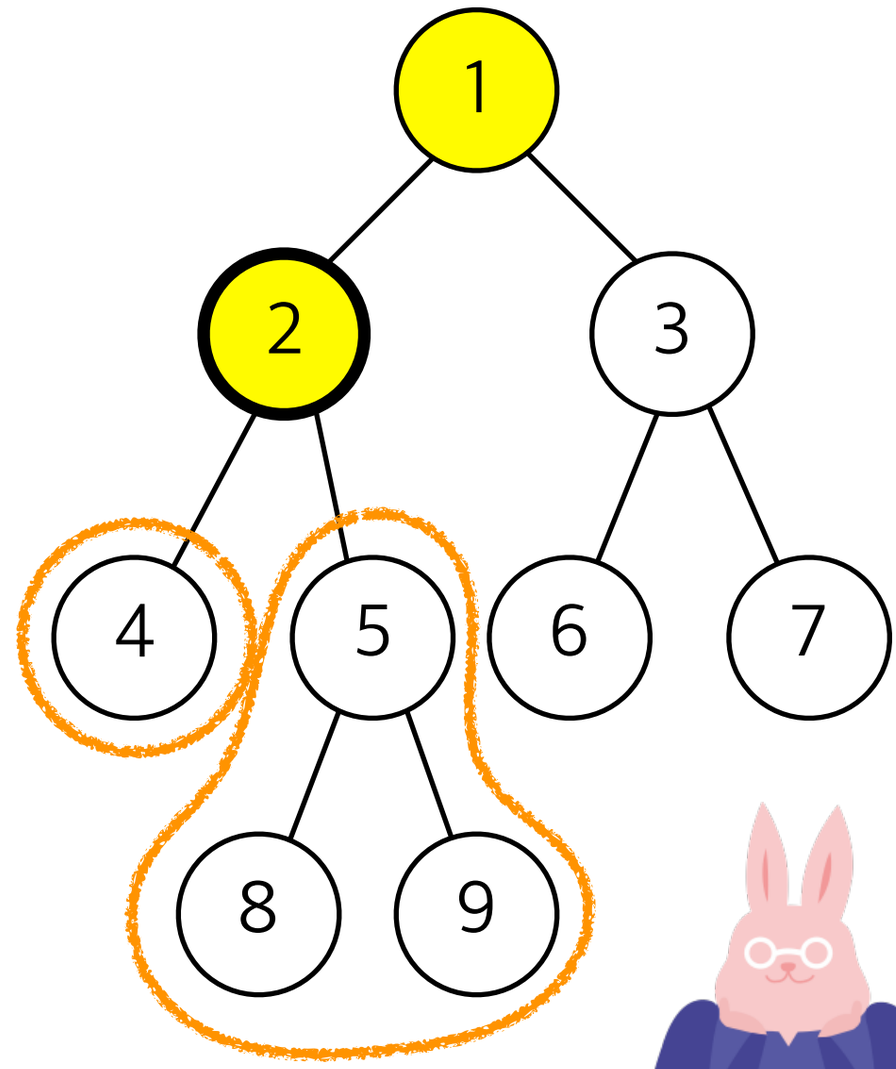
이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



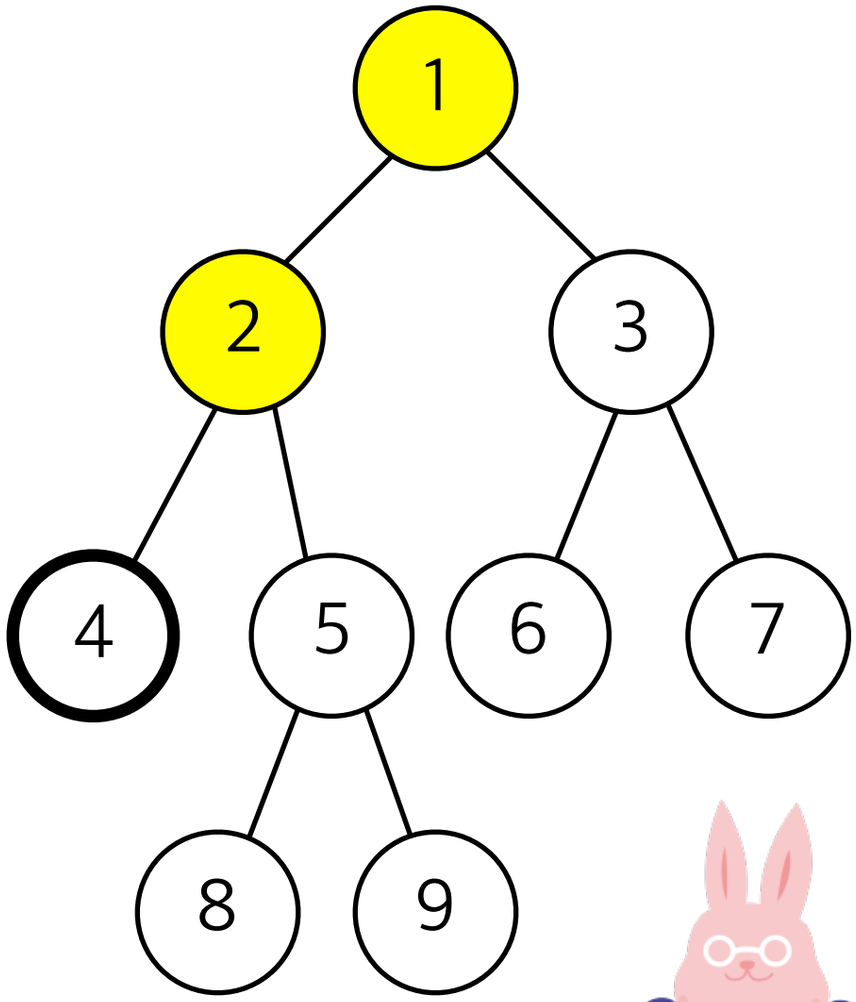
이진 트리 순회: 1. 전위순회

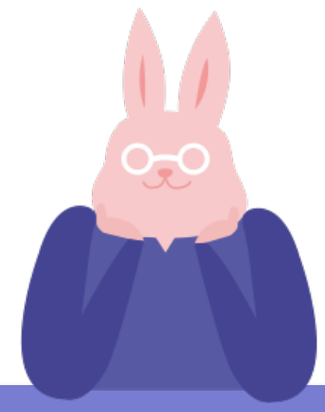
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



이진 트리 순회: 1. 전위순회

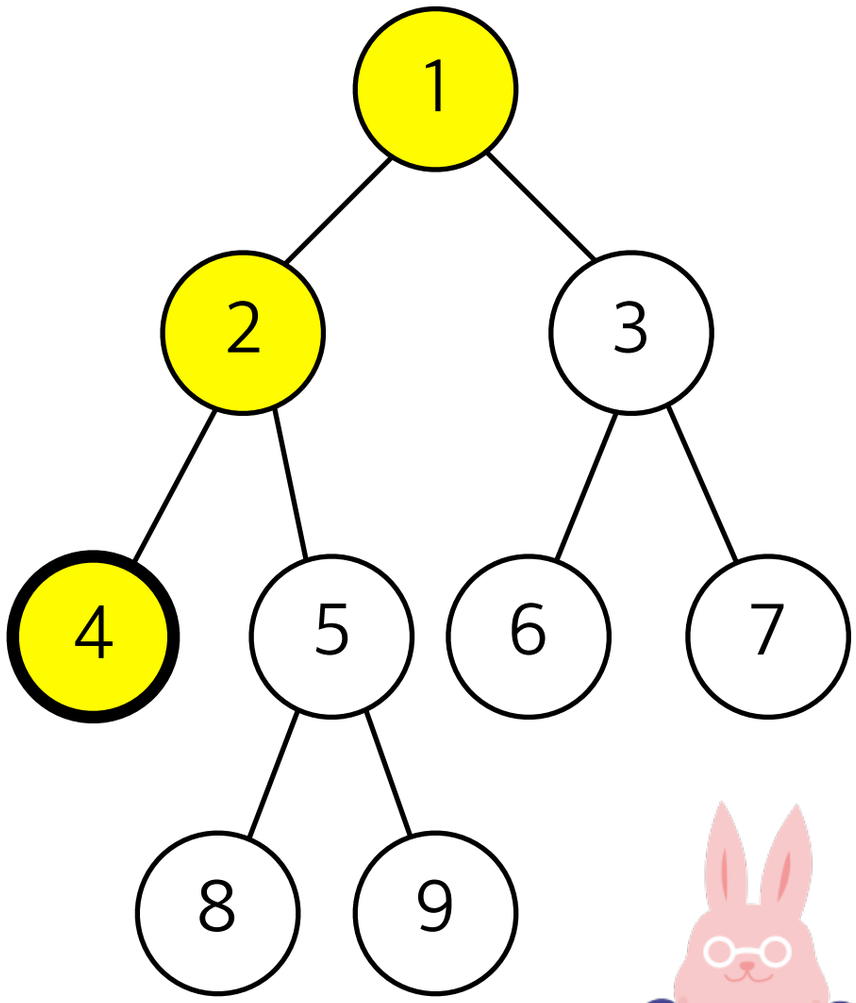
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

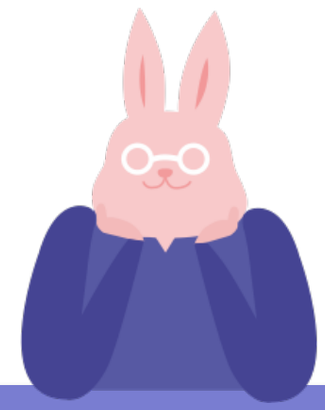




이진 트리 순회: 1. 전위순회

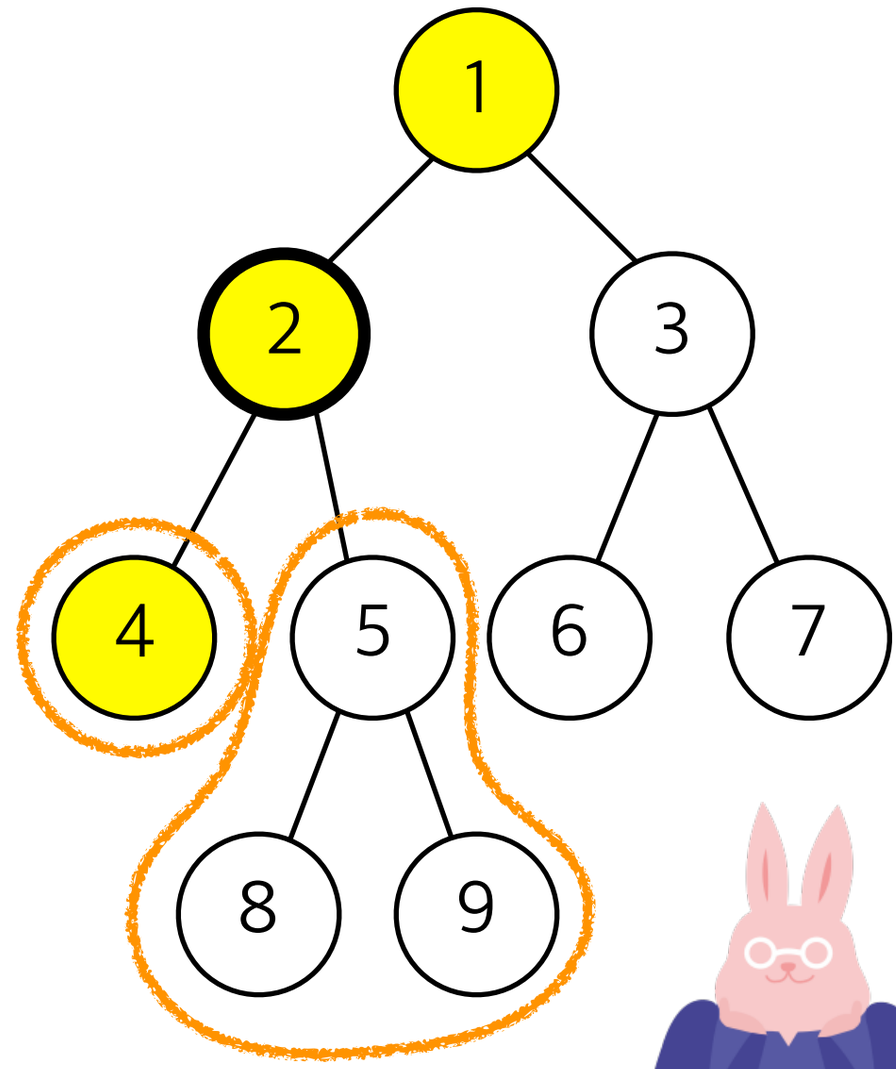
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠





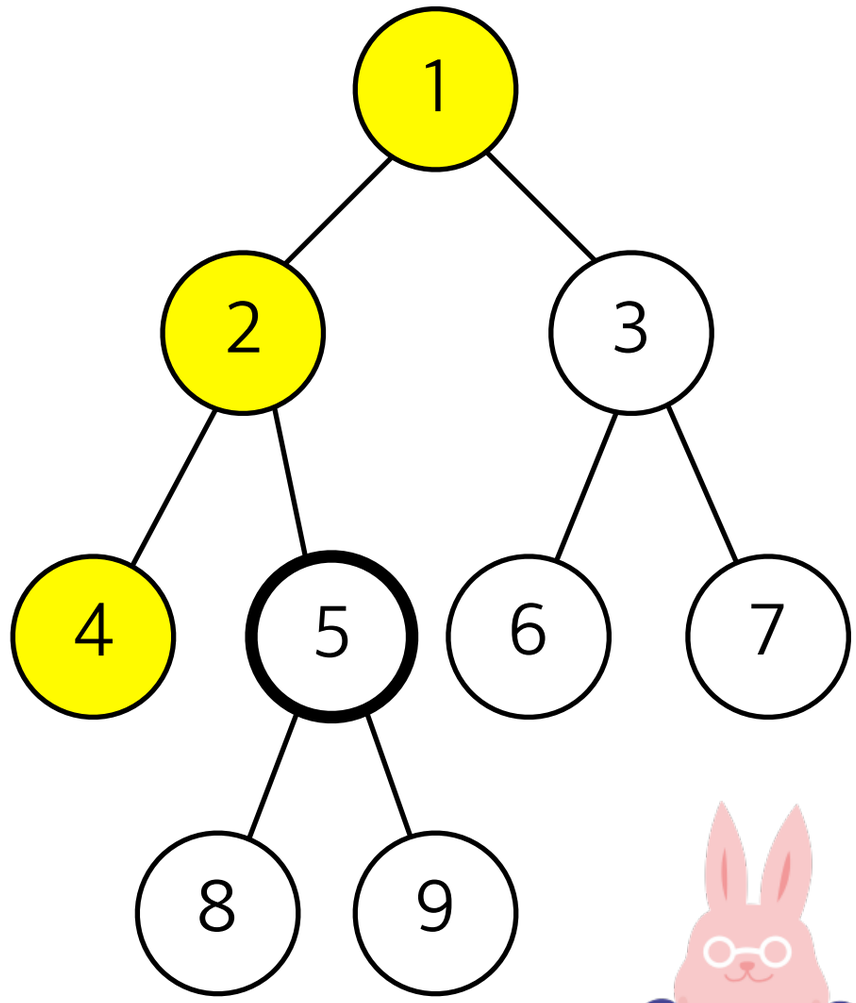
이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



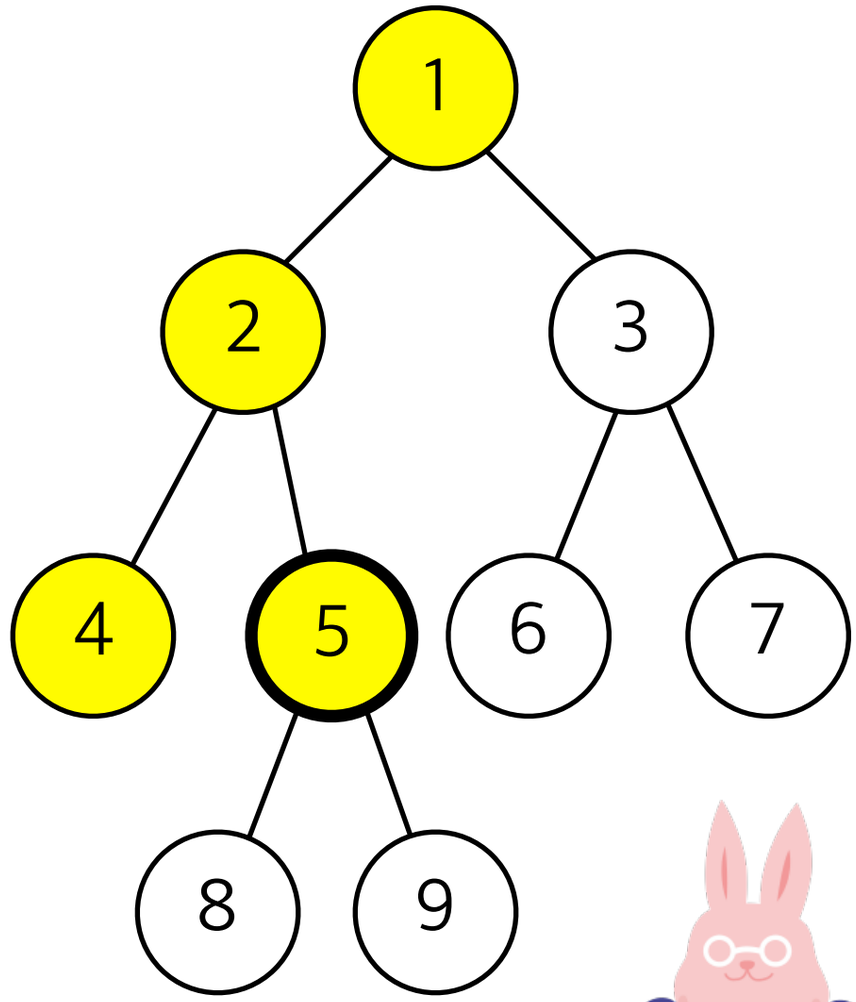
이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



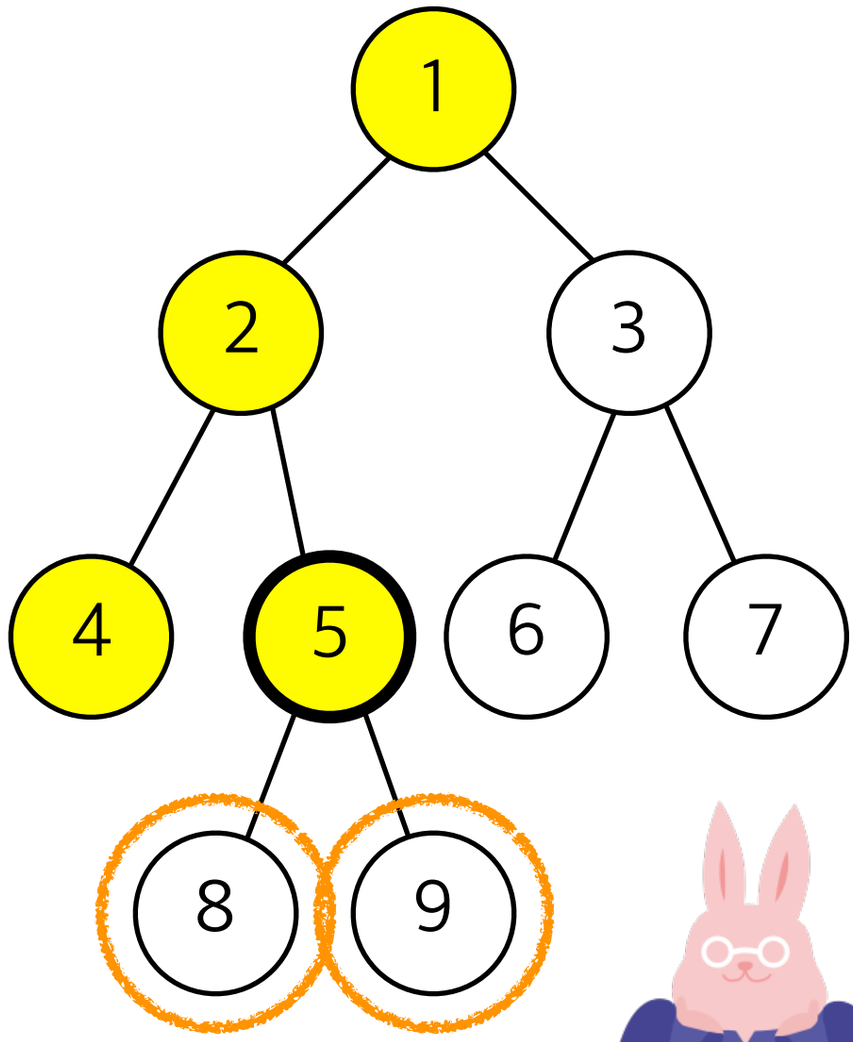
이진 트리 순회: 1. 전위순회

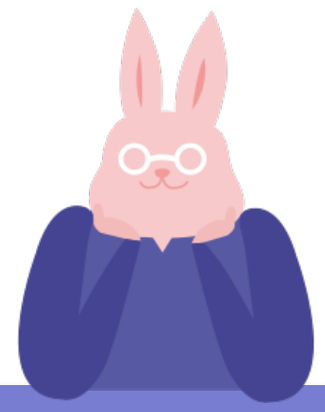
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



이진 트리 순회: 1. 전위순회

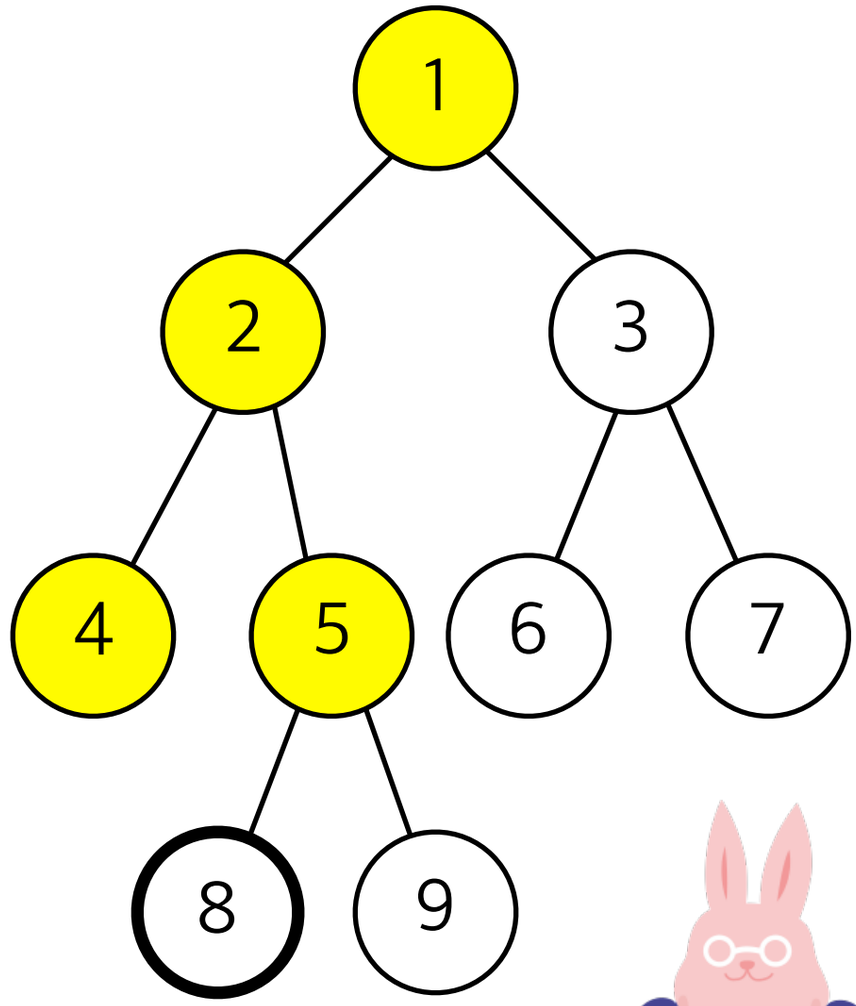
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

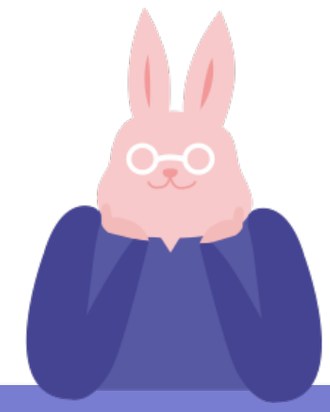




이진 트리 순회: 1. 전위순회

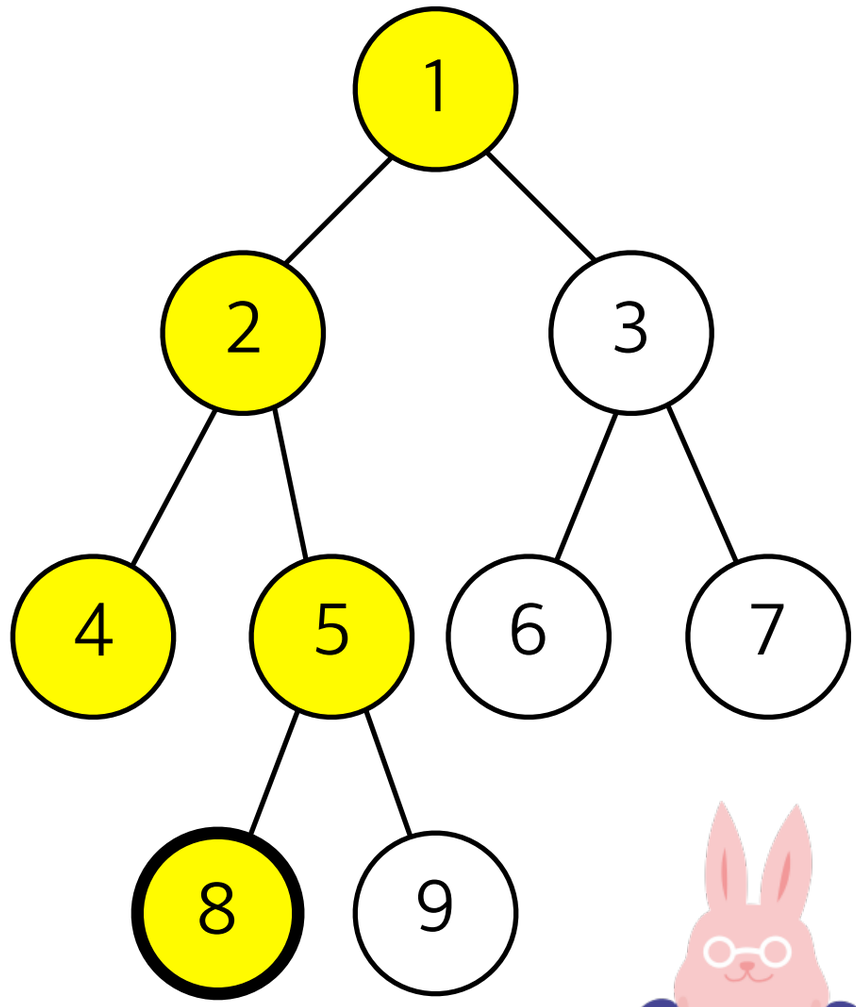
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

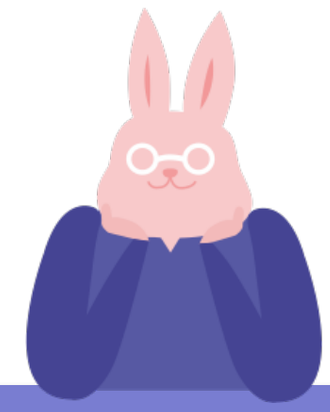




이진 트리 순회: 1. 전위순회

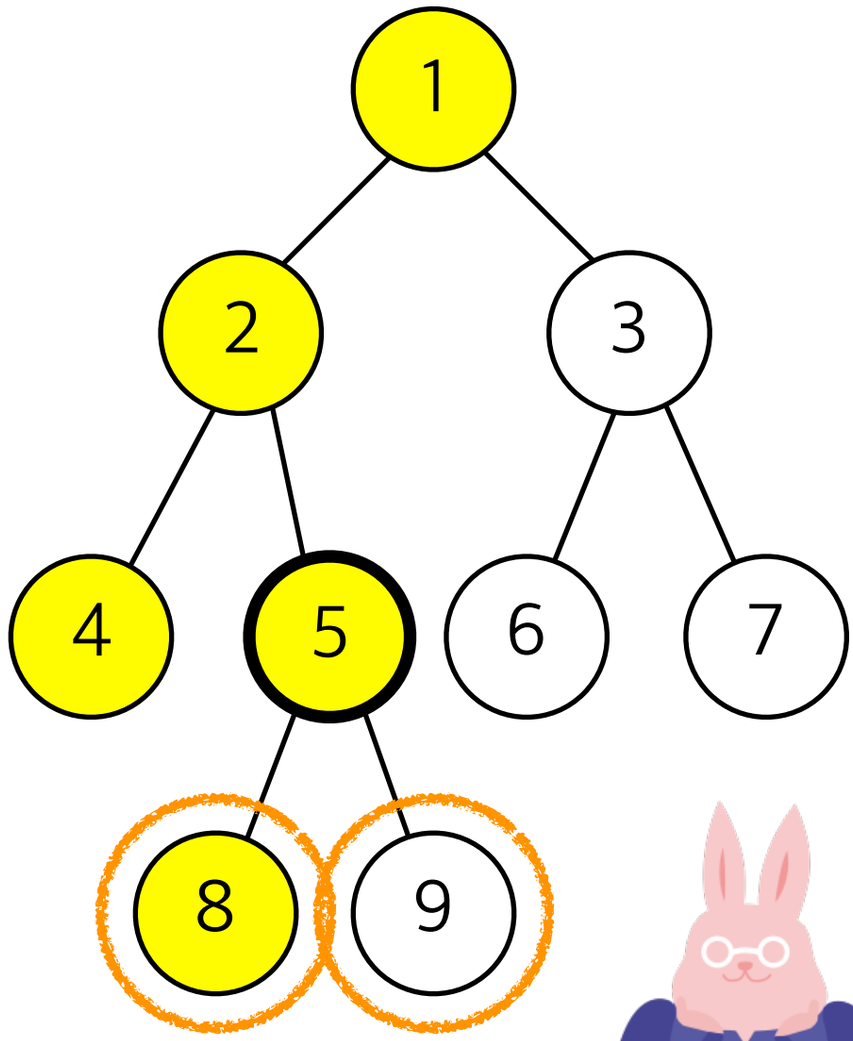
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

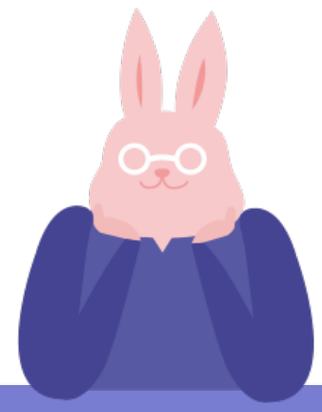




이진 트리 순회: 1. 전위순회

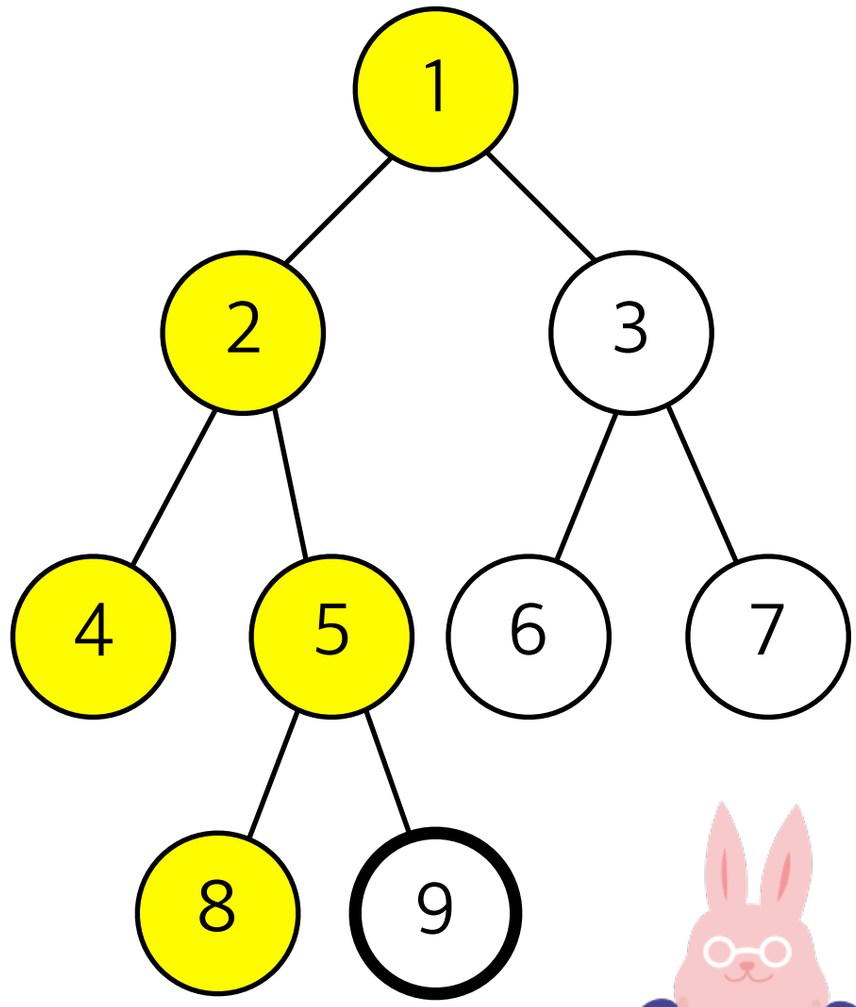
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠





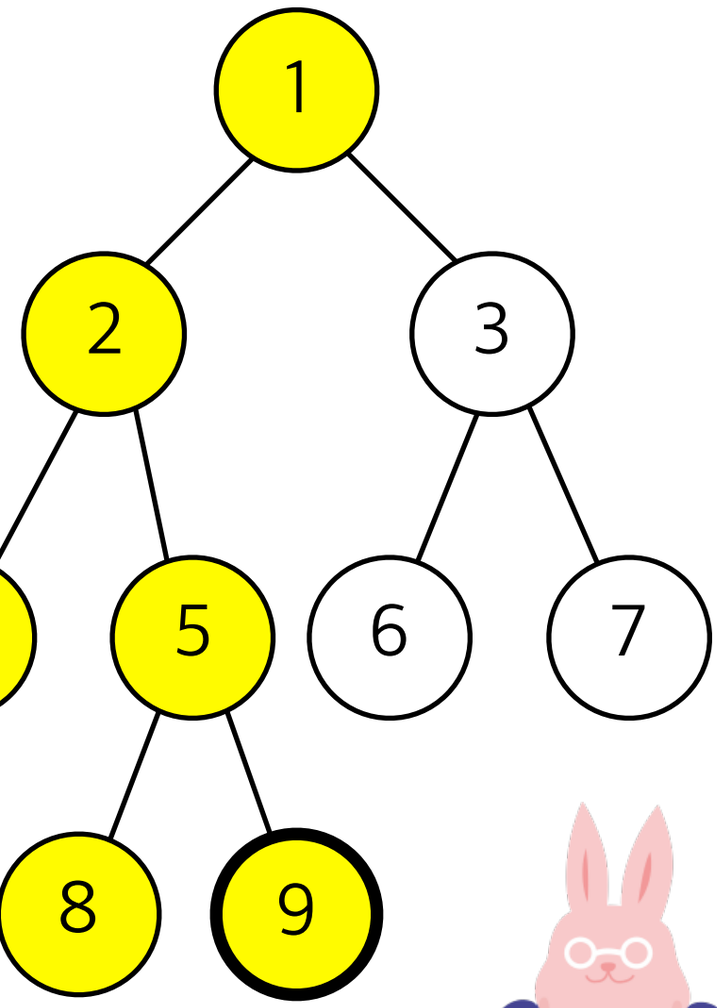
이진 트리 순회: 1. 전위순회

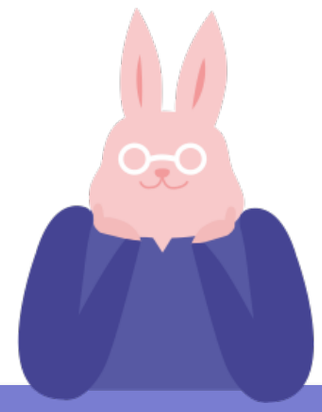
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



이진 트리 순회: 1. 전위순회

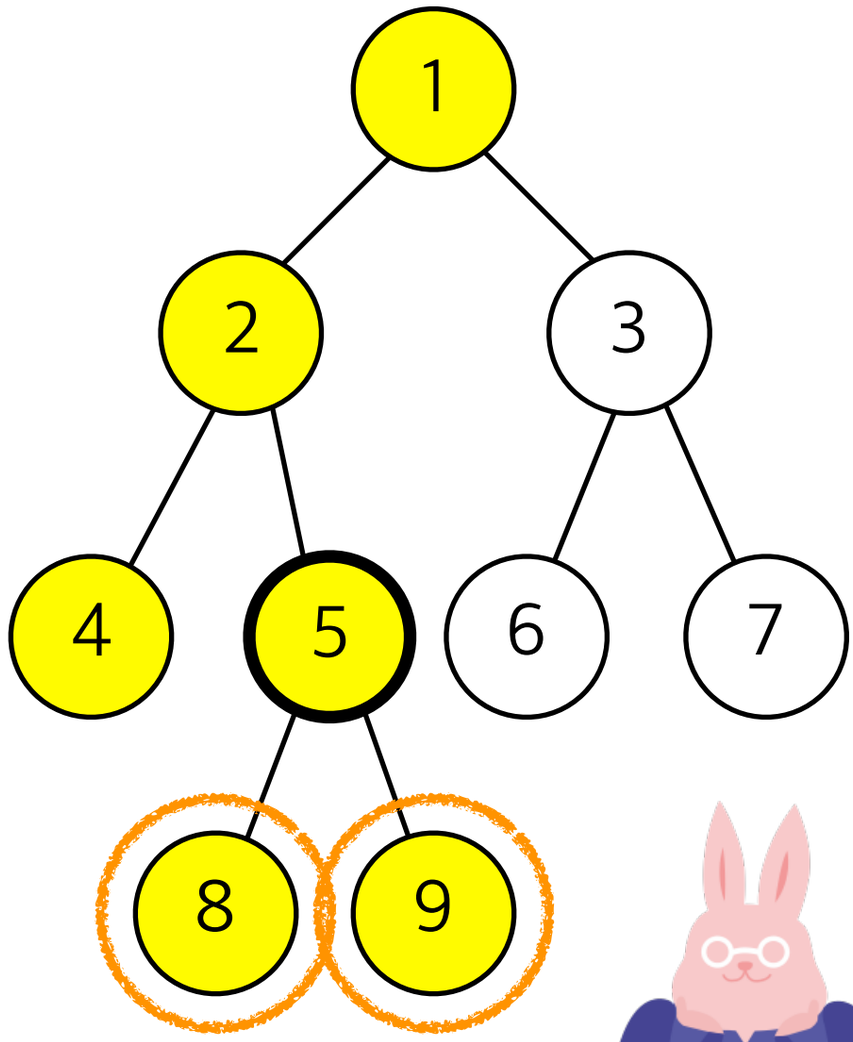
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠





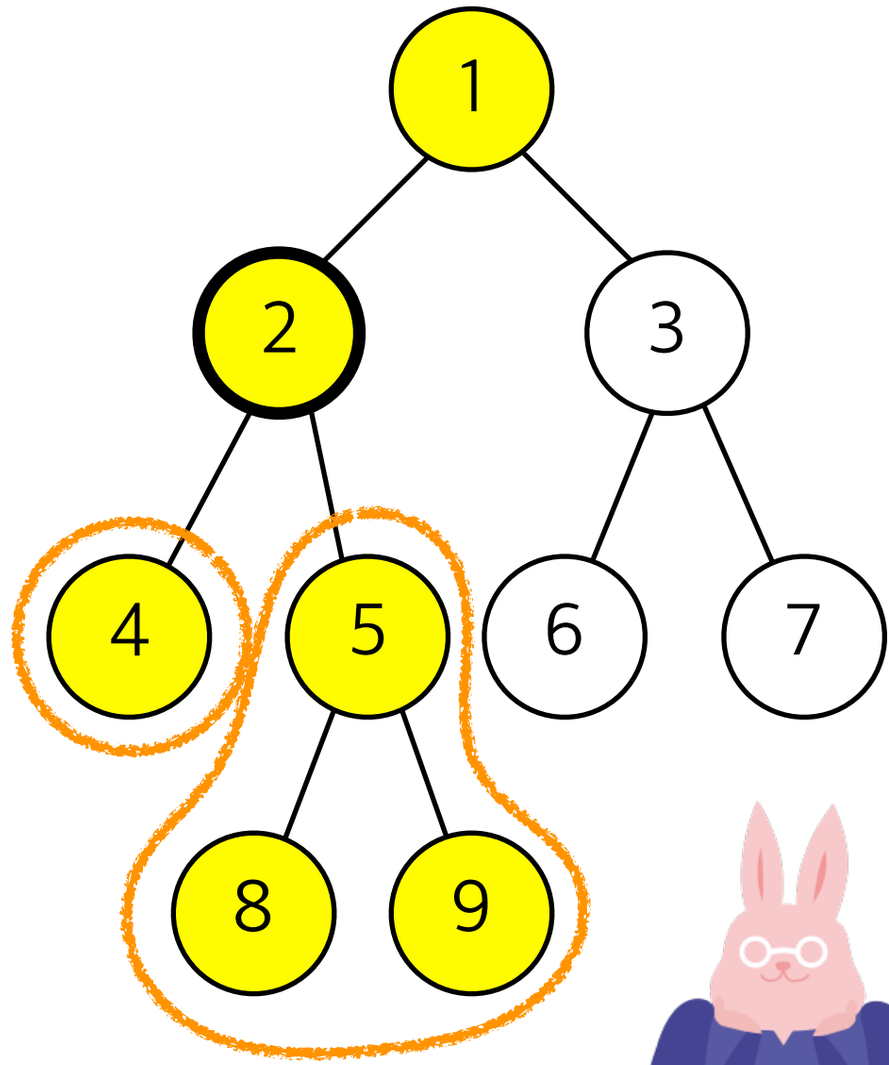
이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



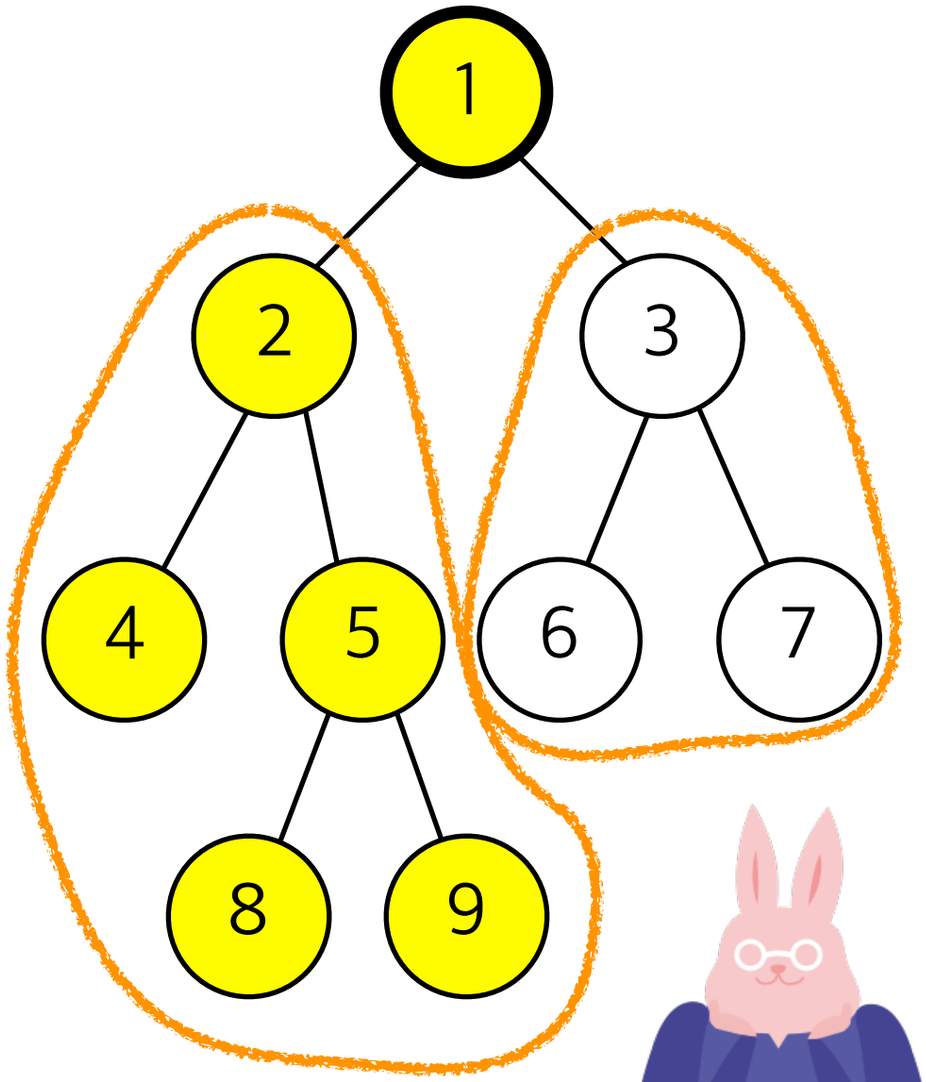
이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



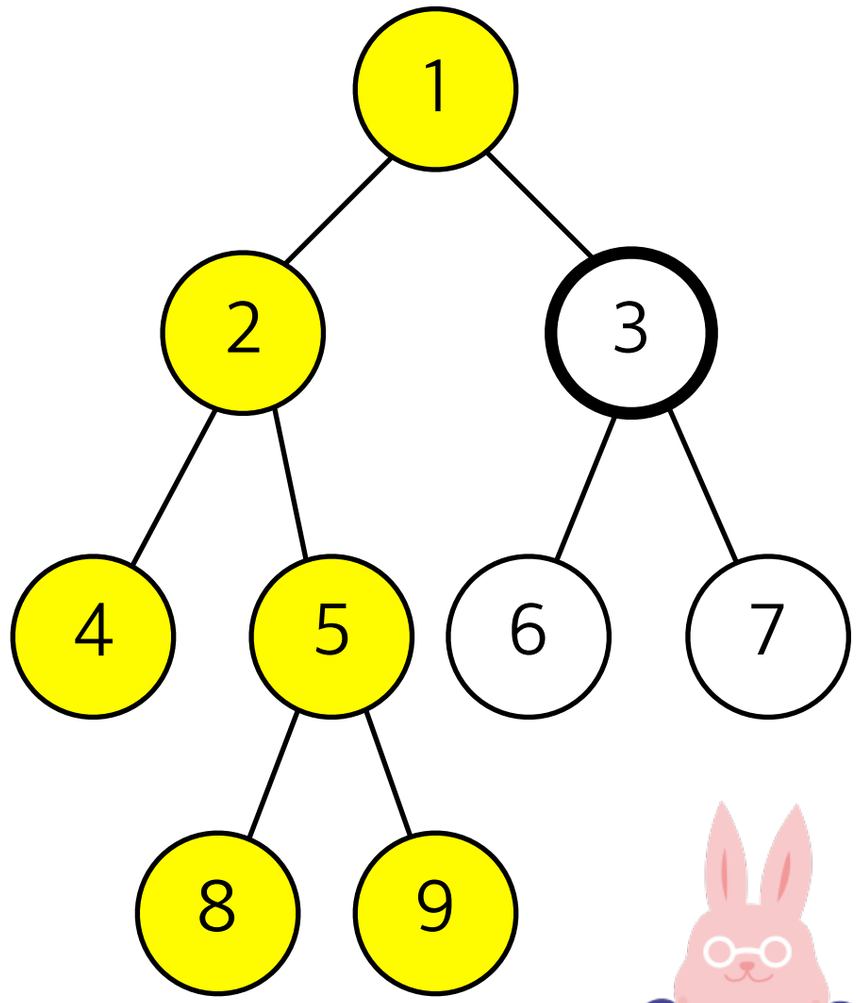
이진 트리 순회: 1. 전위순회

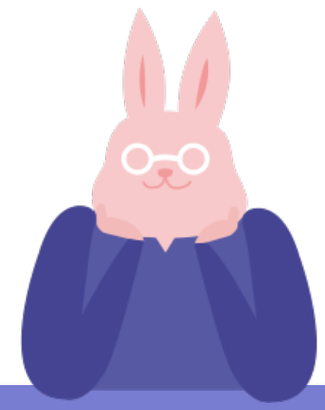
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



이진 트리 순회: 1. 전위순회

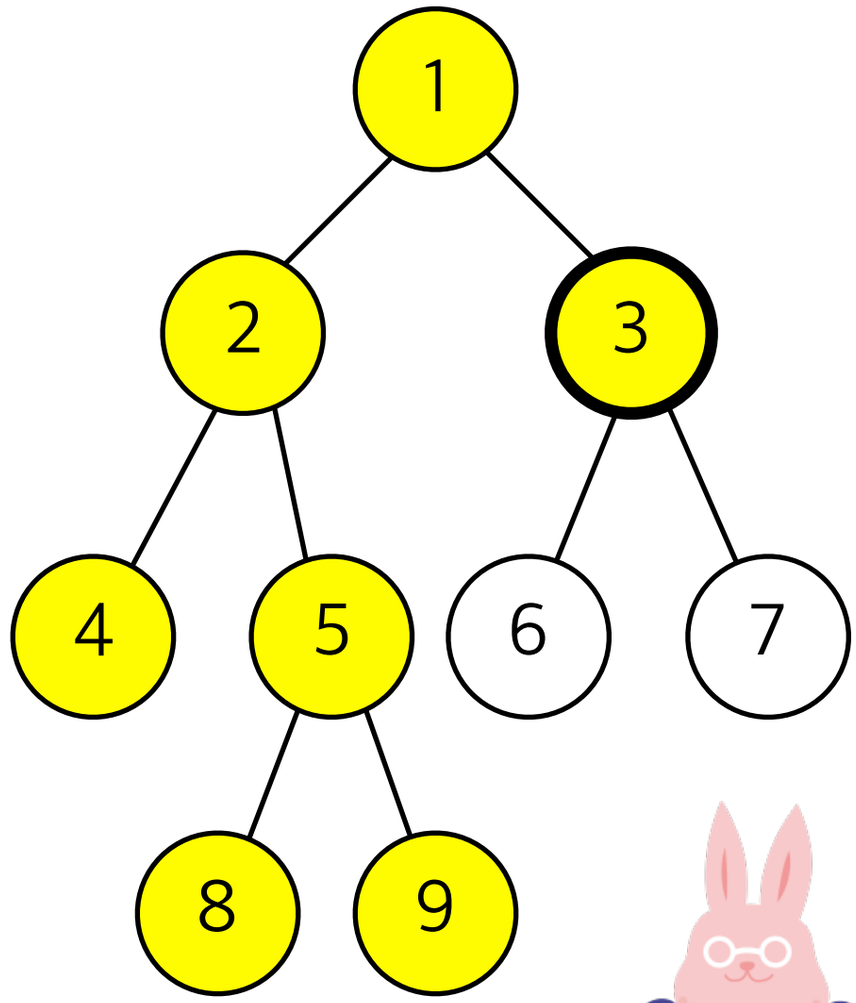
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

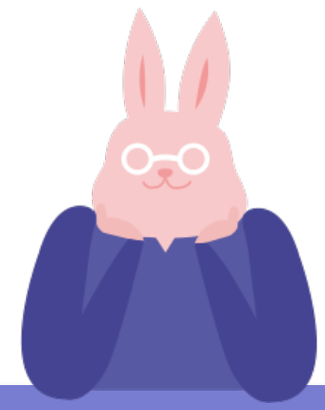




이진 트리 순회: 1. 전위순회

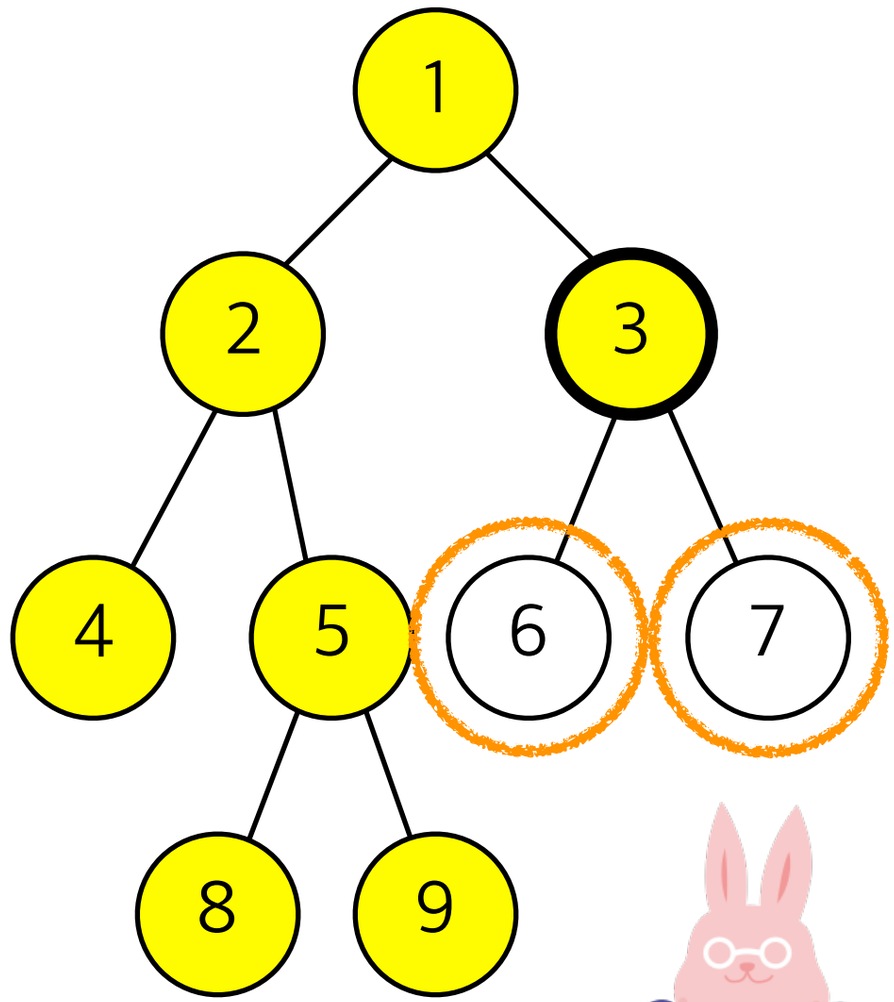
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

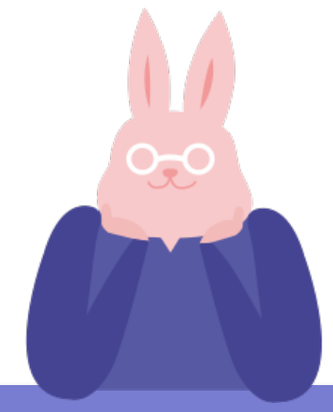




이진 트리 순회: 1. 전위순회

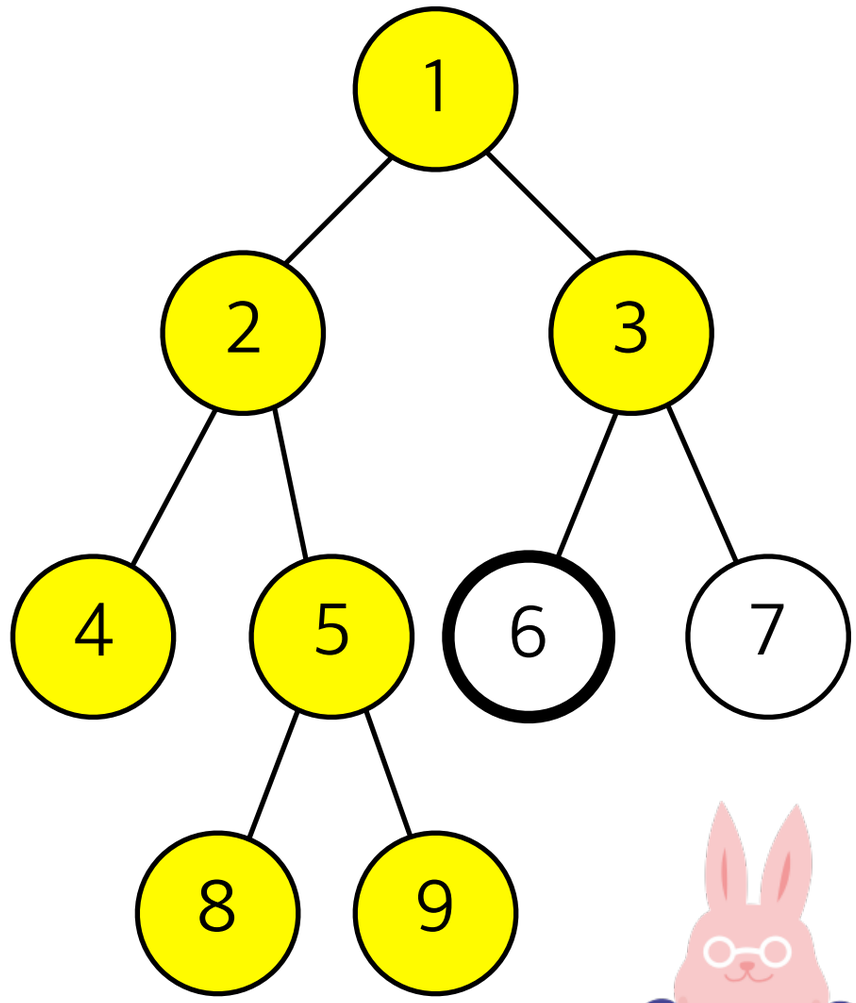
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠





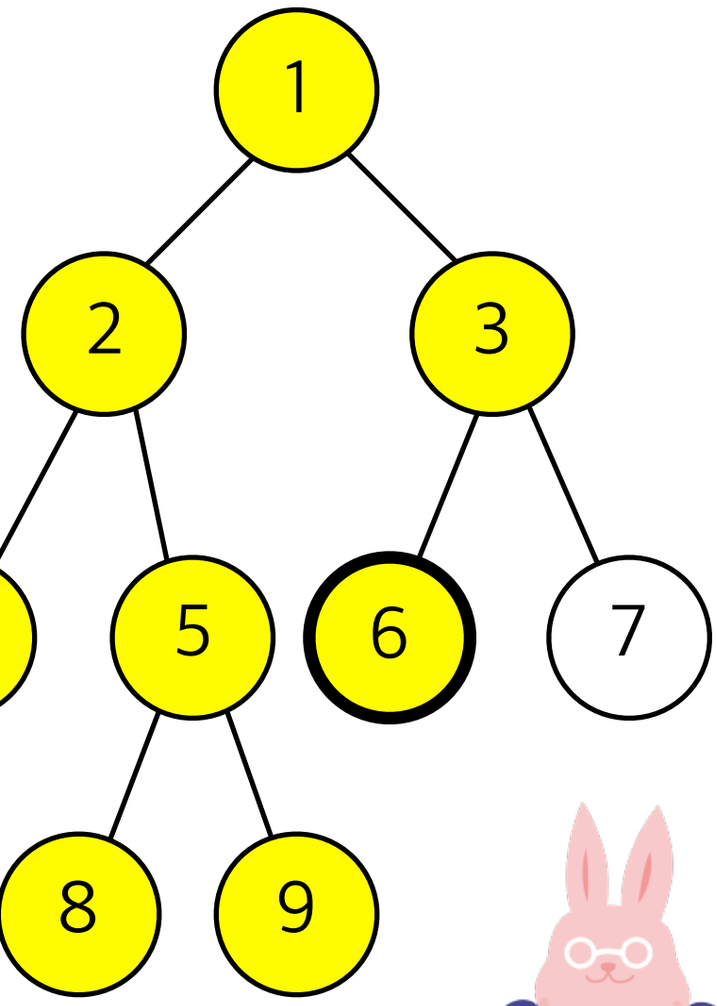
이진 트리 순회: 1. 전위순회

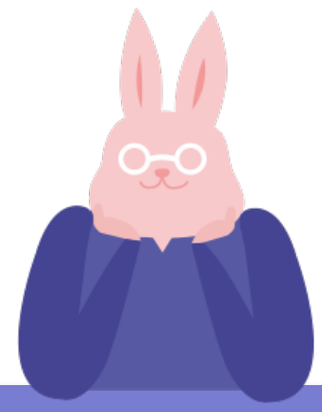
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



이진 트리 순회: 1. 전위순회

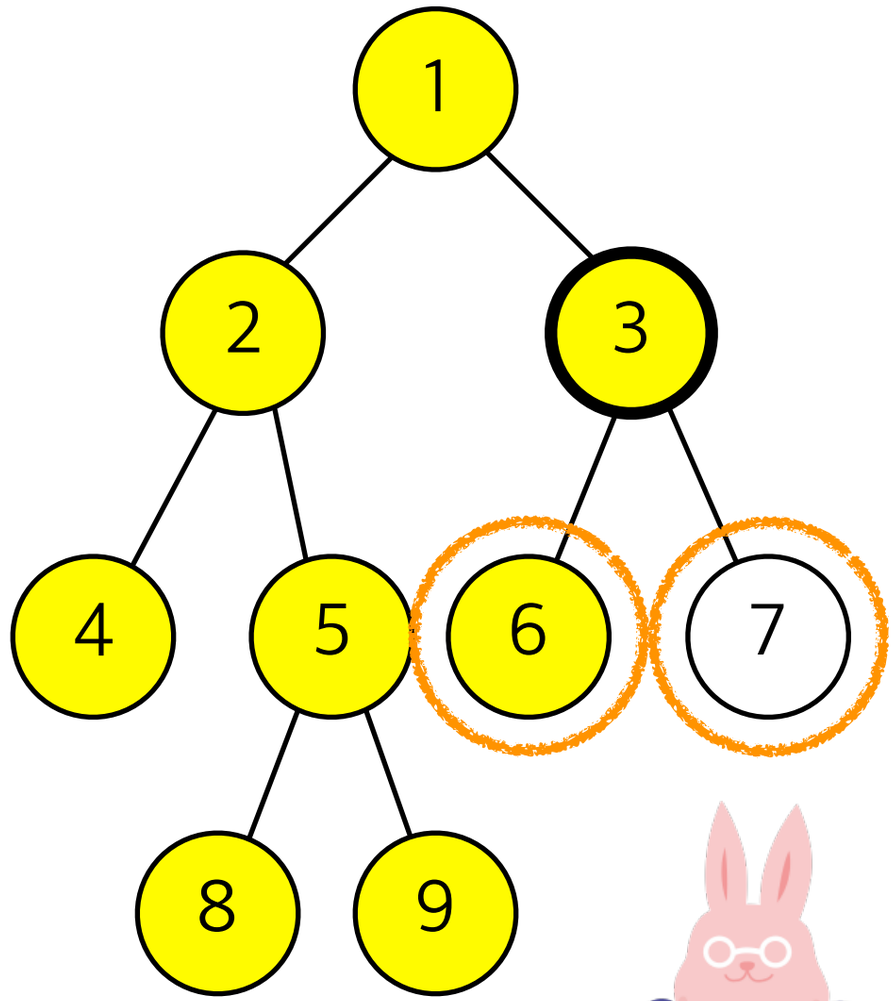
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

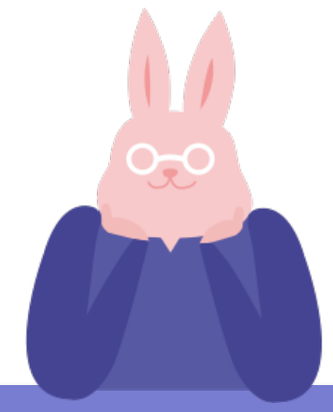




이진 트리 순회: 1. 전위순회

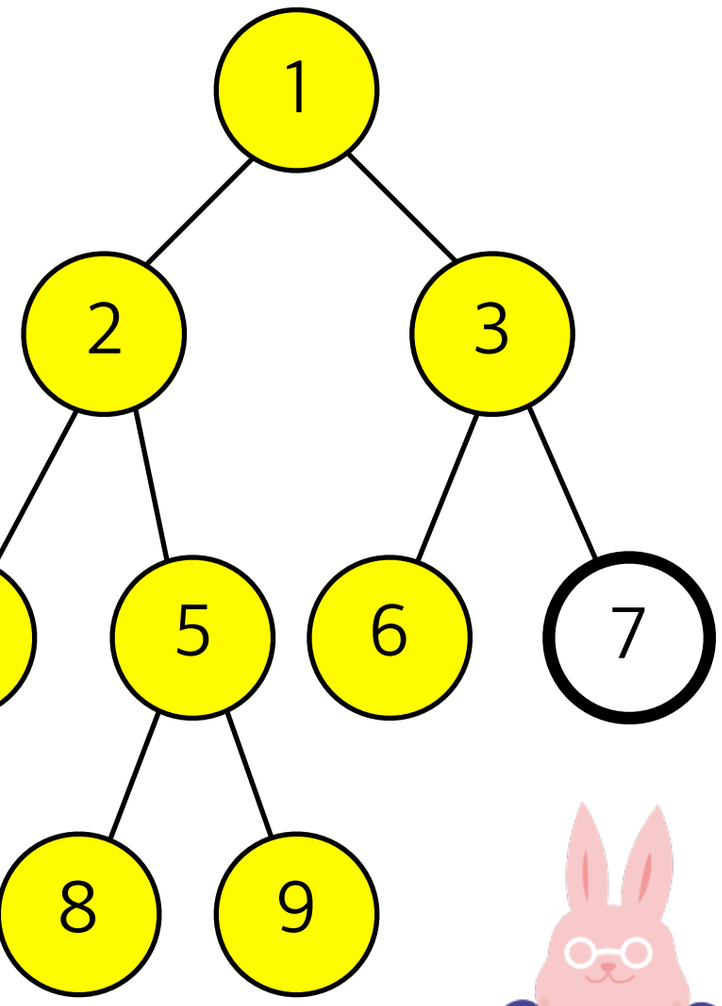
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠





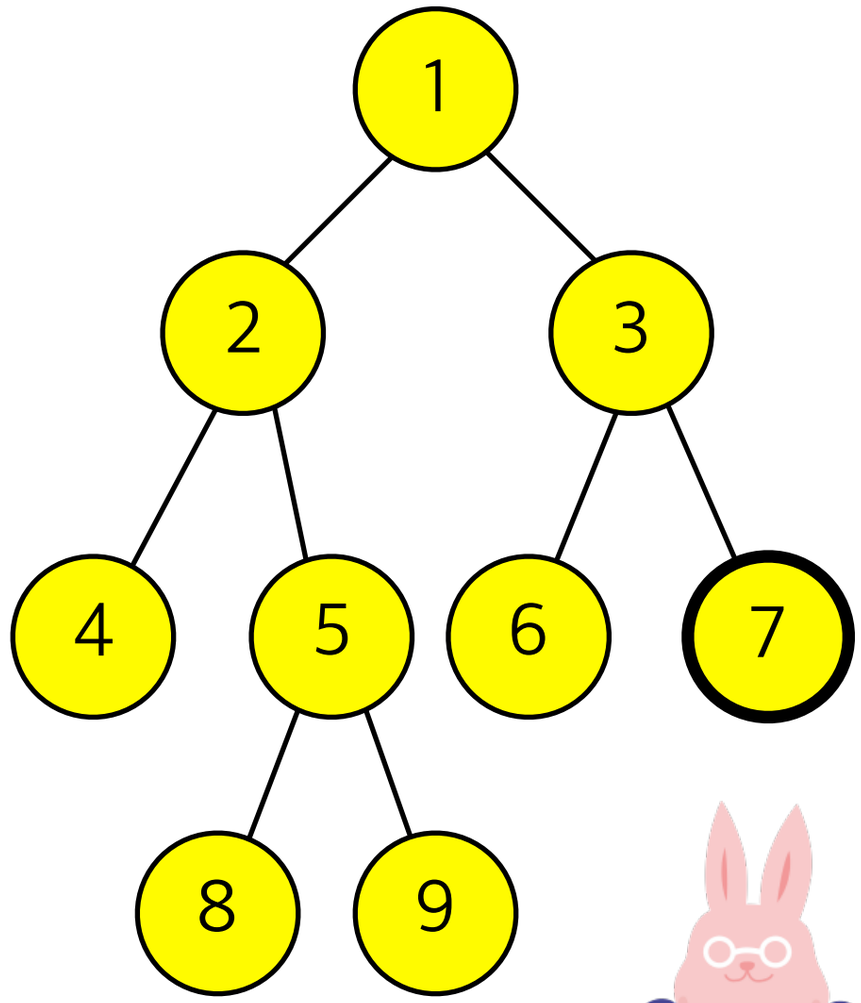
이진 트리 순회: 1. 전위순회

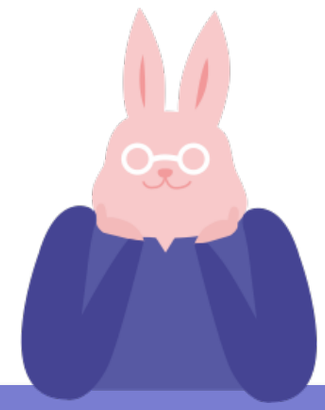
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



이진 트리 순회: 1. 전위순회

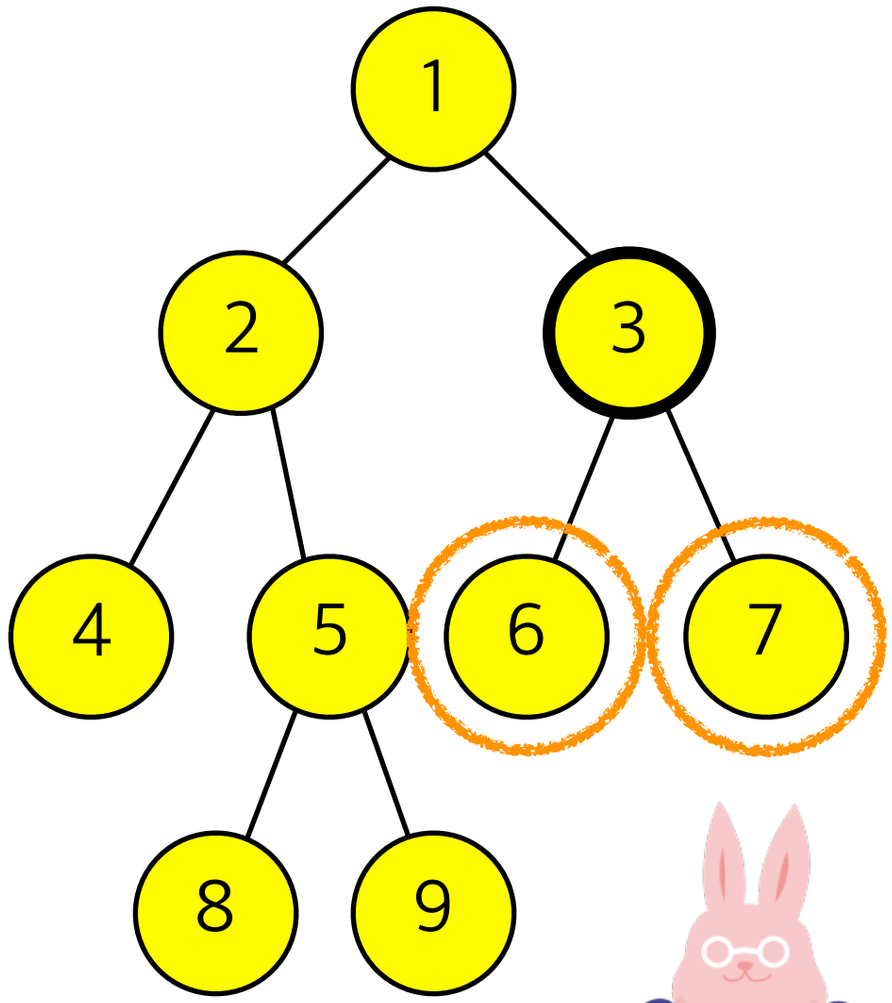
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

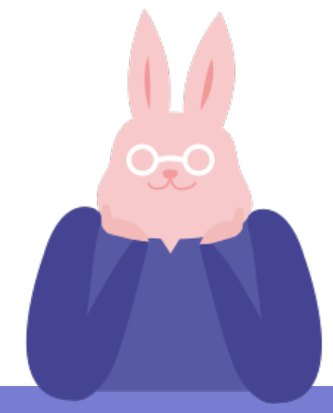




이진 트리 순회: 1. 전위순회

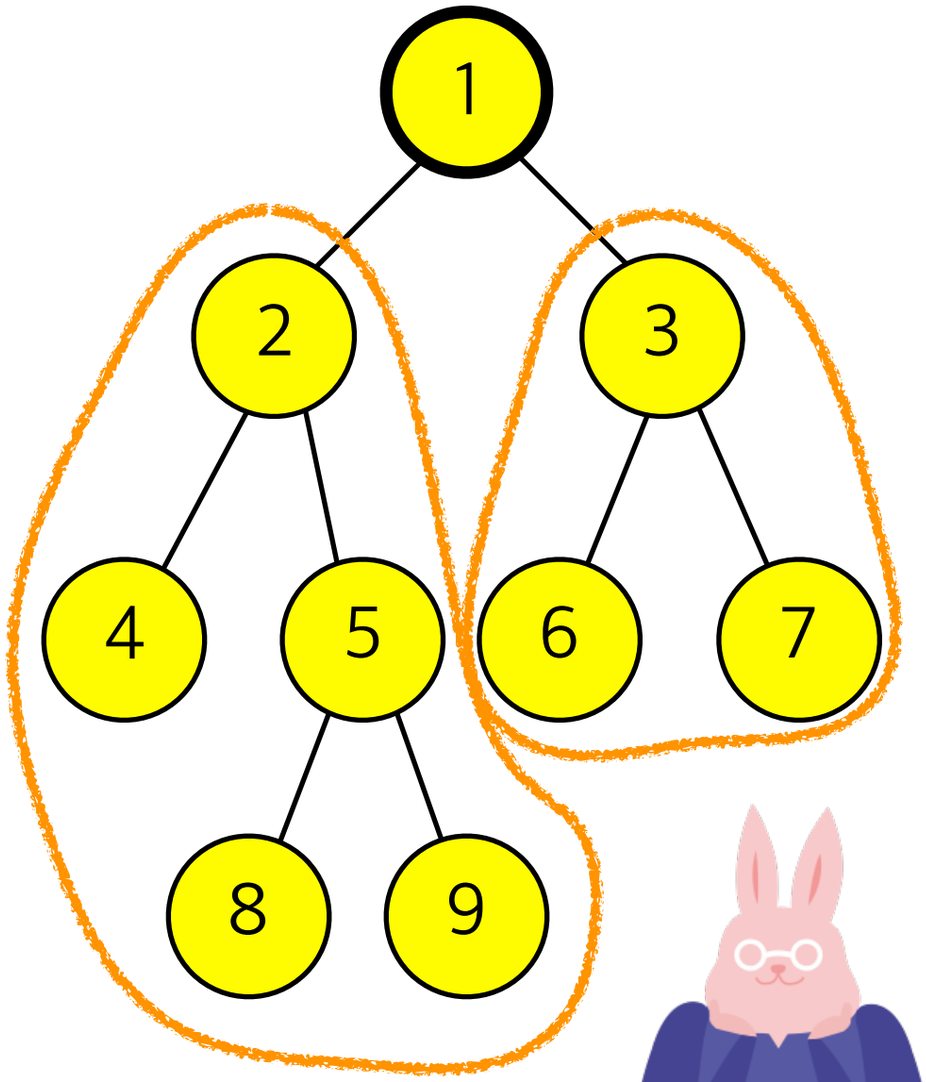
- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠





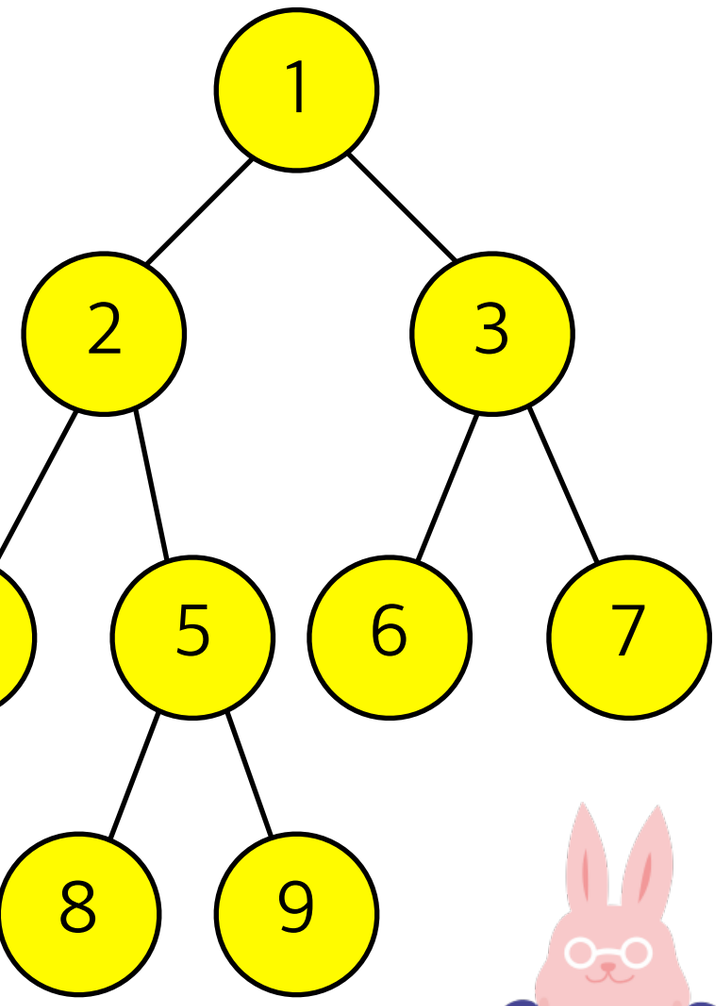
이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

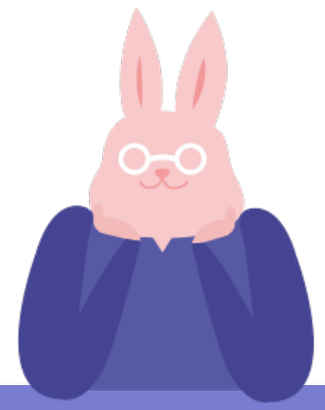


이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠

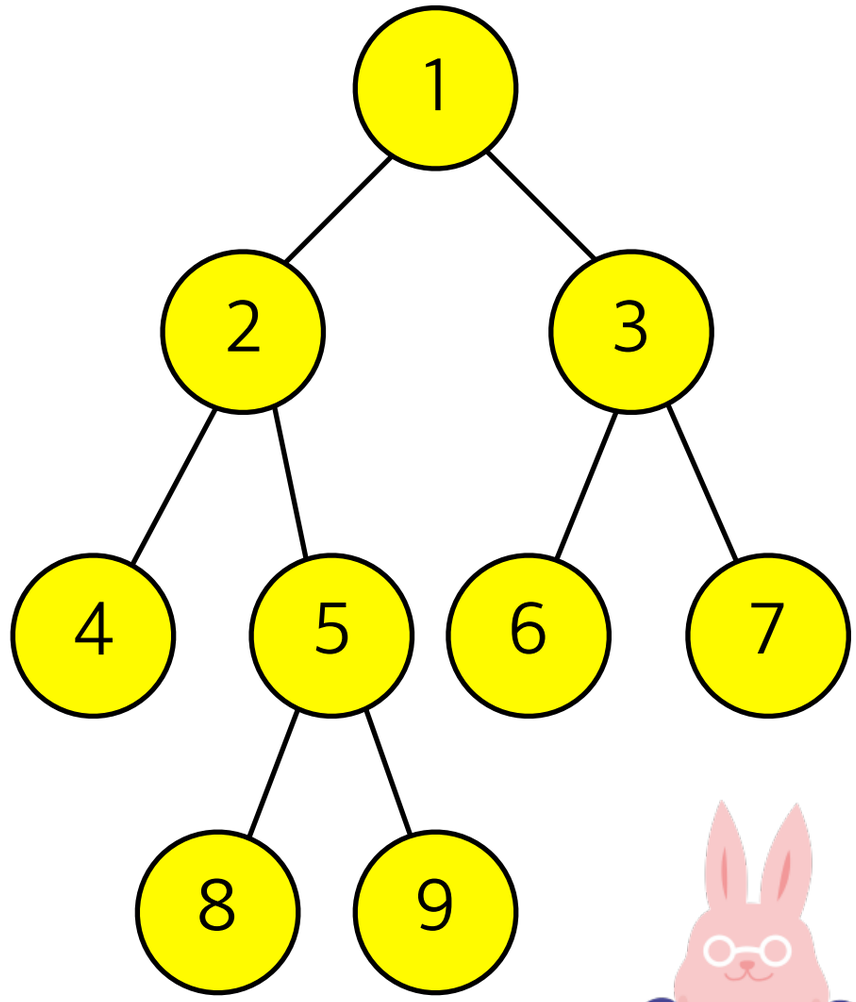


- 방문한 순서는 ?



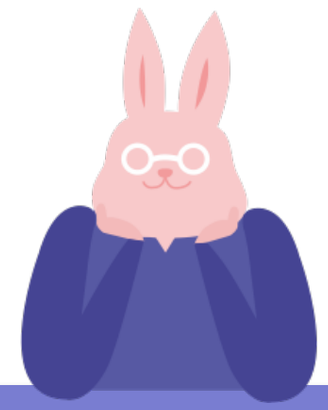
이진 트리 순회: 1. 전위순회

- 알고리즘
  1. Root노드 색칠
  2. 왼쪽 subtree를 전위순회로 색칠
  3. 오른쪽 subtree를 전위순회로 색칠



- 색칠한 순서는 ?

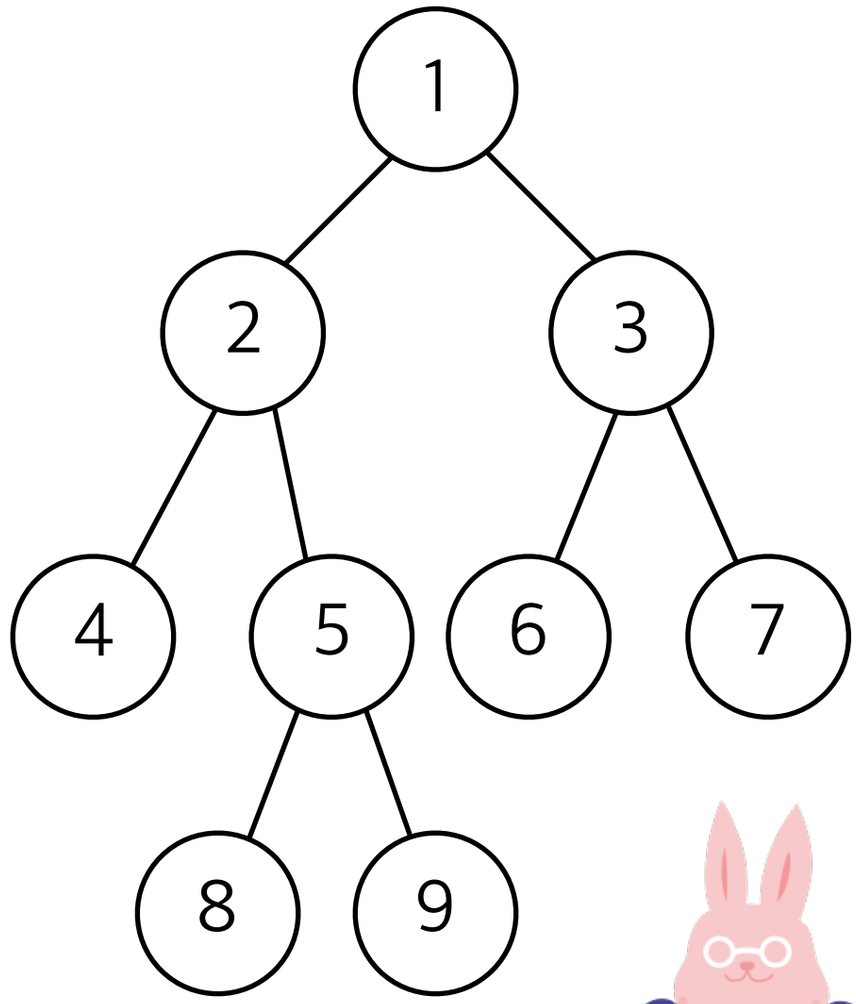
  1 - 2 - 4 - 5 - 8 - 9 - 3 - 6 - 7

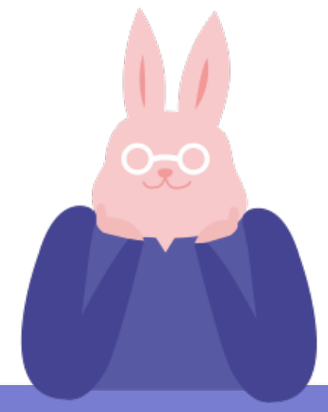


이진 트리 순회 : 2. 중위순회

- 알고리즘
  1. 왼쪽 subtree를 중위순회로 색칠
  2. Root노드 색칠
  3. 오른쪽 subtree를 중위순회로 색칠

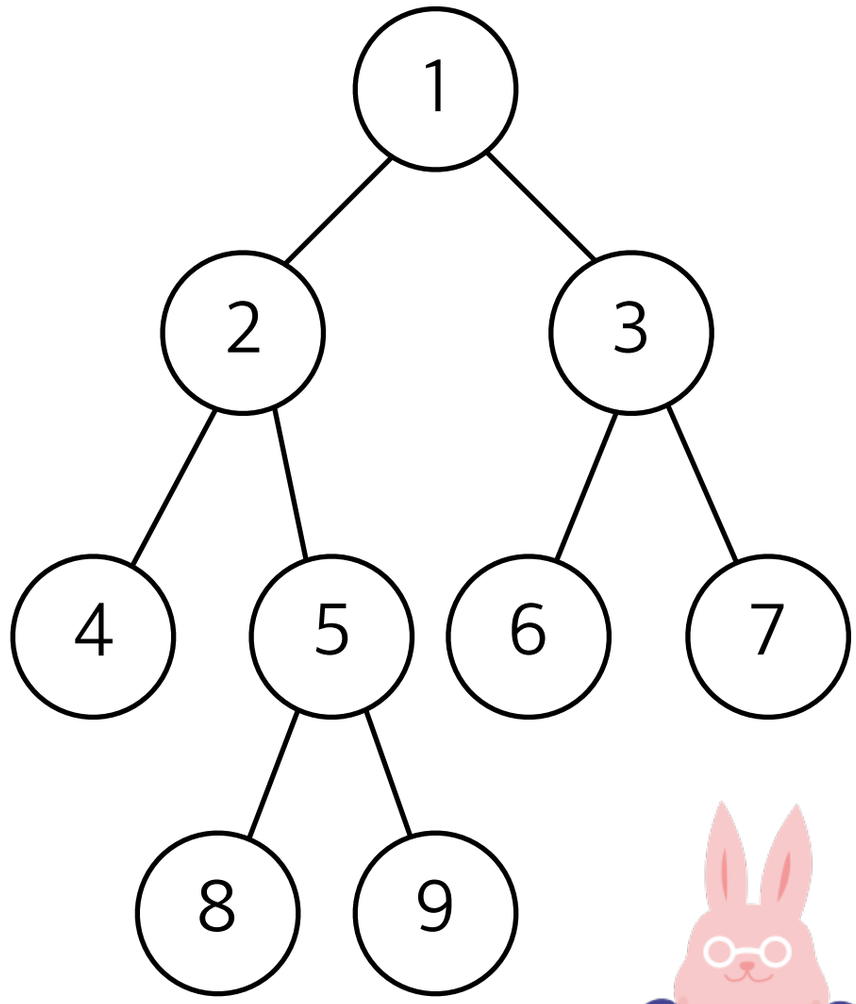
- 색칠한 순서는 ?





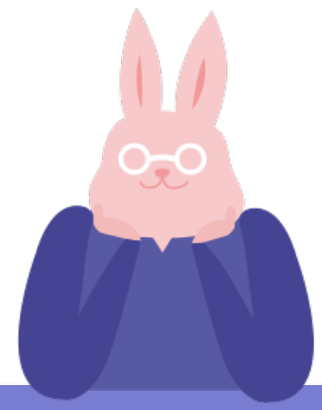
이진 트리 순회 : 2. 중위순회

- 알고리즘
  1. 왼쪽 subtree를 중위순회로 색칠
  2. Root노드 색칠
  3. 오른쪽 subtree를 중위순회로 색칠



- 색칠한 순서는 ?

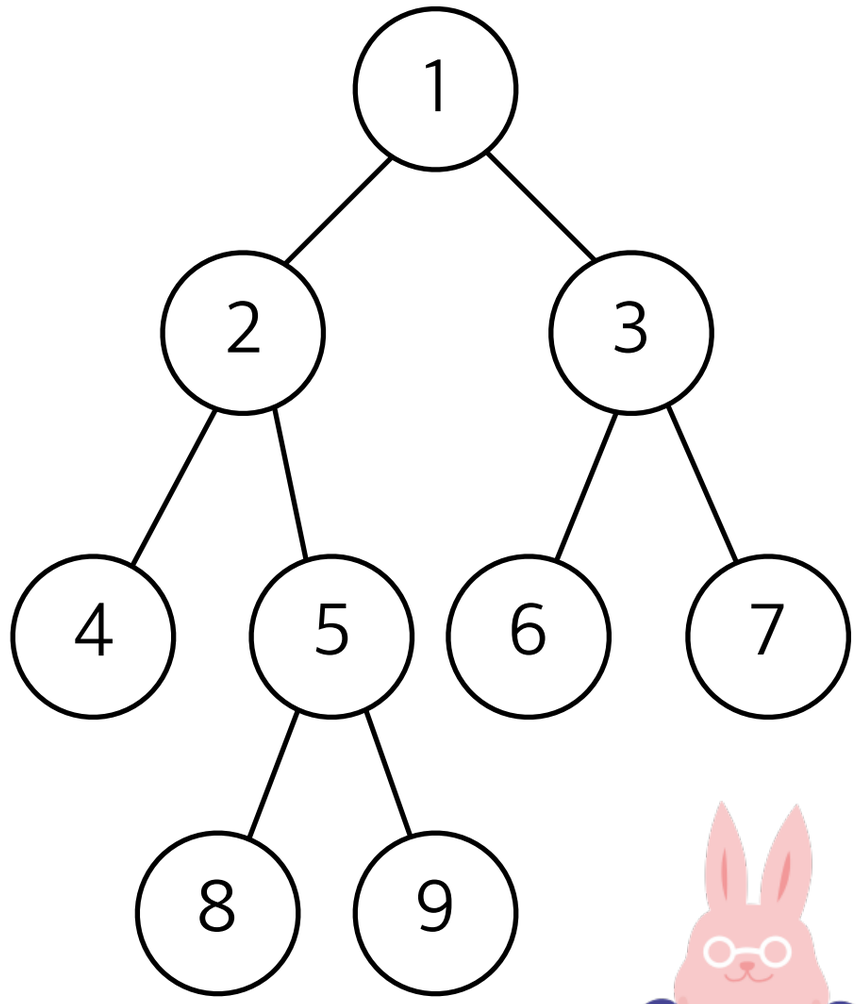
  4 - 2 - 8 - 5 - 9 - 1 - 6 - 3 - 7

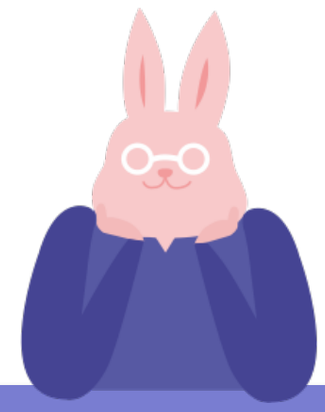


이진 트리 순회 : 3. 후위순회

- 알고리즘
  1. 왼쪽 subtree를 후위순회로 색칠
  2. 오른쪽 subtree를 후위순회로 색칠
  3. Root노드 색칠

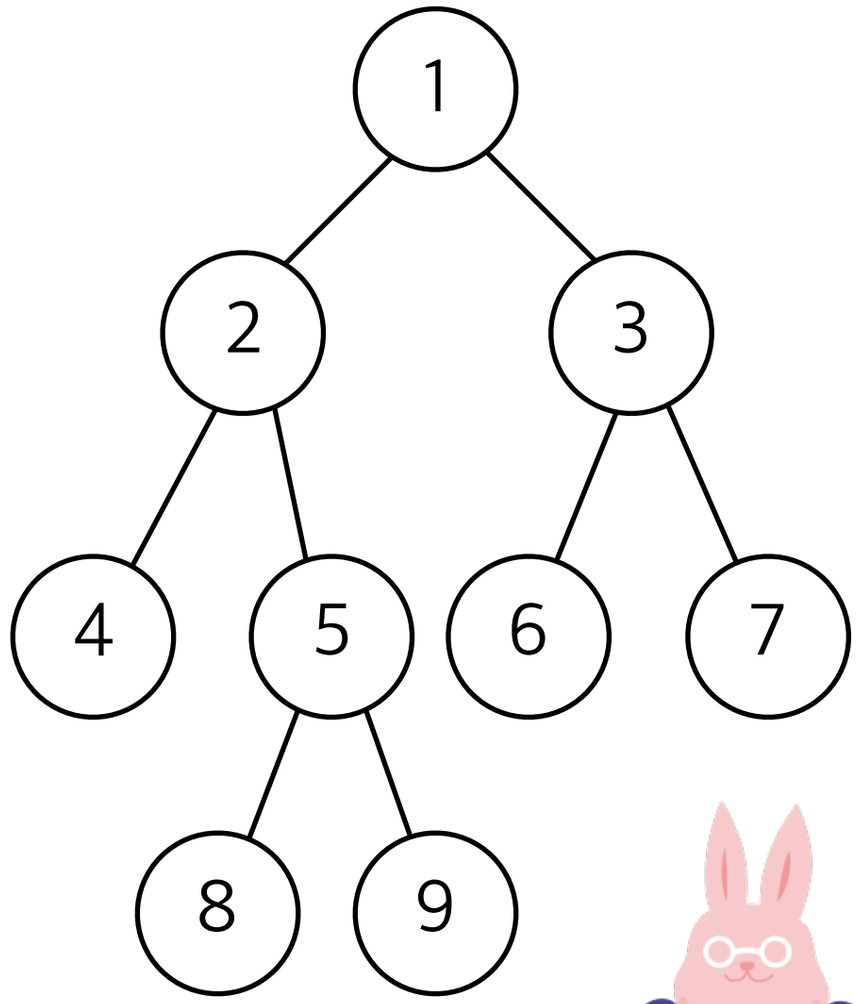
- 색칠한 순서는 ?





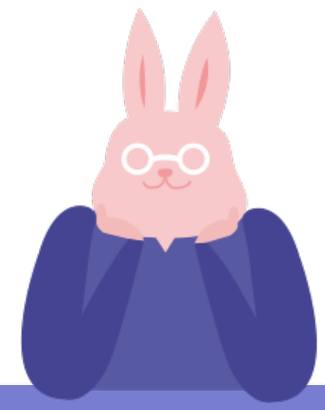
이진 트리 순회 : 3. 후위순회

- 알고리즘
  1. 왼쪽 subtree를 후위순회로 색칠
  2. 오른쪽 subtree를 후위순회로 색칠
  3. Root노드 색칠



- 색칠한 순서는 ?

  4 - 8 - 9 - 5 - 2 - 6 - 7 - 3 - 1



[활동문제 2] 후위 순회 결과 출력

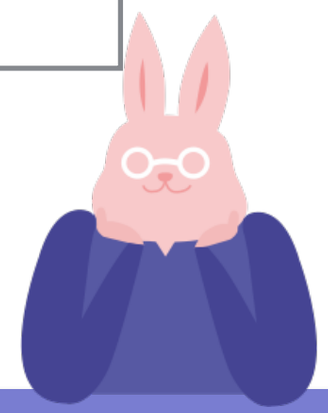
- 트리를 입력받아 후위순회 결과를 출력

입력의 예

```
5
0 1 2
1 3 4
2 -1 -1
3 -1 -1
4 -1 -1
```

출력의 예

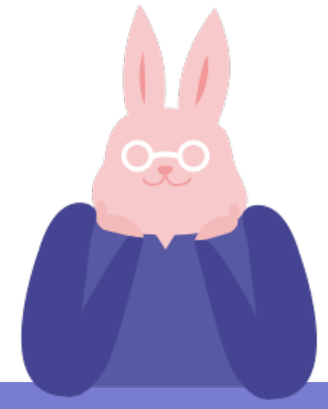
```
3 4 1 2 0
```



트리 구조의 이용 : 최소힙

- 요구사항
  - N개의 숫자 중에서 최솟값을 출력할 수 있어야 한다
  - N개의 숫자 중에서 최솟값을 삭제할 수 있어야 한다
  - 임의의 숫자를 추가할 수 있어야 한다

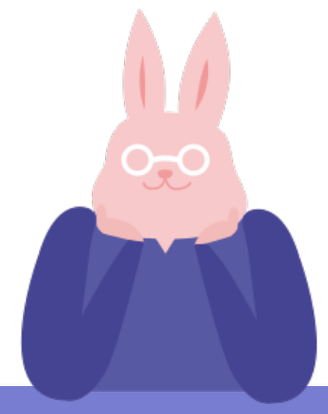
4 5 1 15 18 14 12 13



트리 구조의 이용 : 최소힙

- 요구사항
  - N개의 숫자 중에서 최솟값을 출력할 수 있어야 한다 : O(1)
  - N개의 숫자 중에서 최솟값을 삭제할 수 있어야 한다 : O(n)
  - 임의의 숫자를 추가할 수 있어야 한다 : O(1)

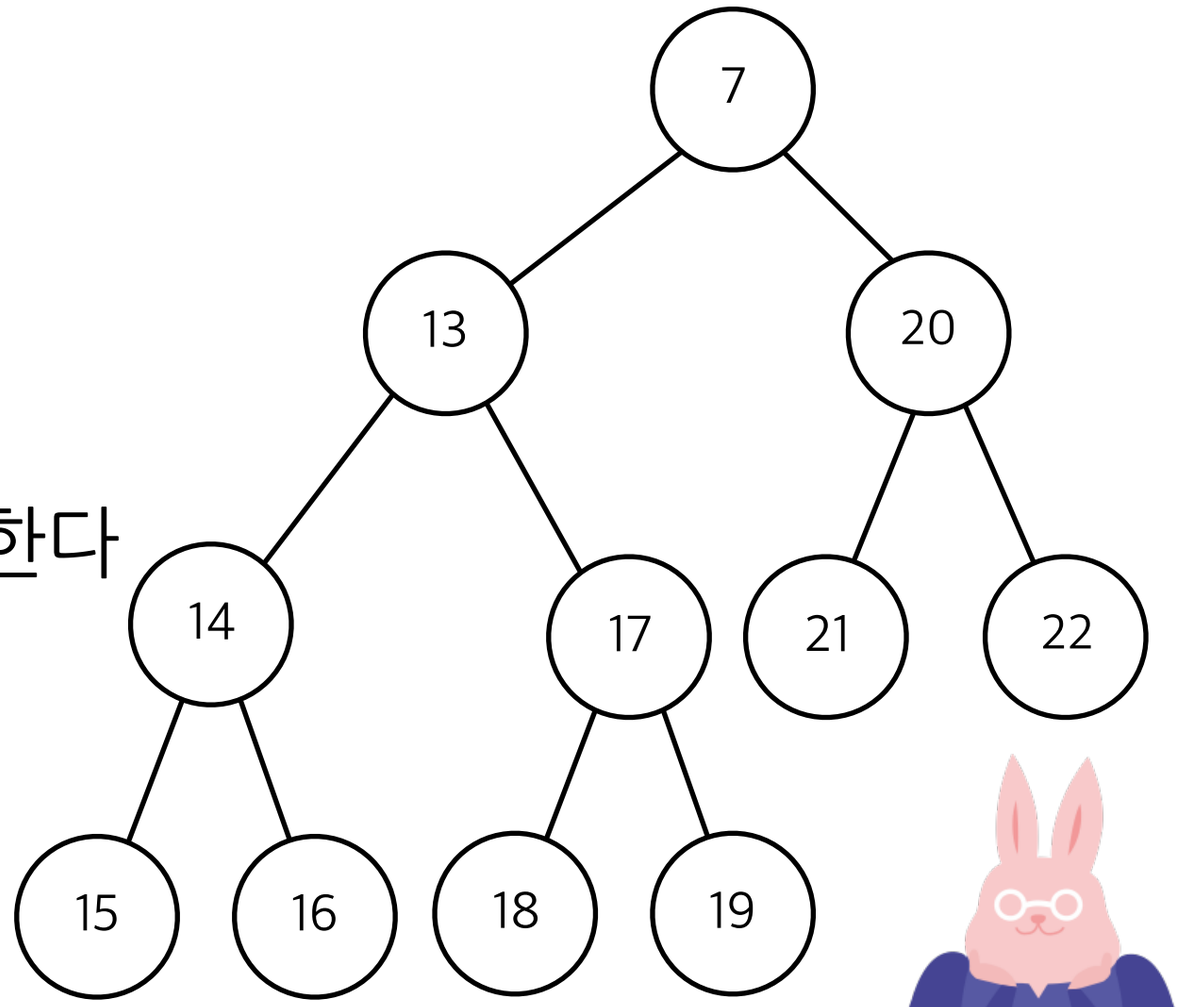
4 5 1 15 18 14 12 13



트리 구조의 이용 : 최소힙

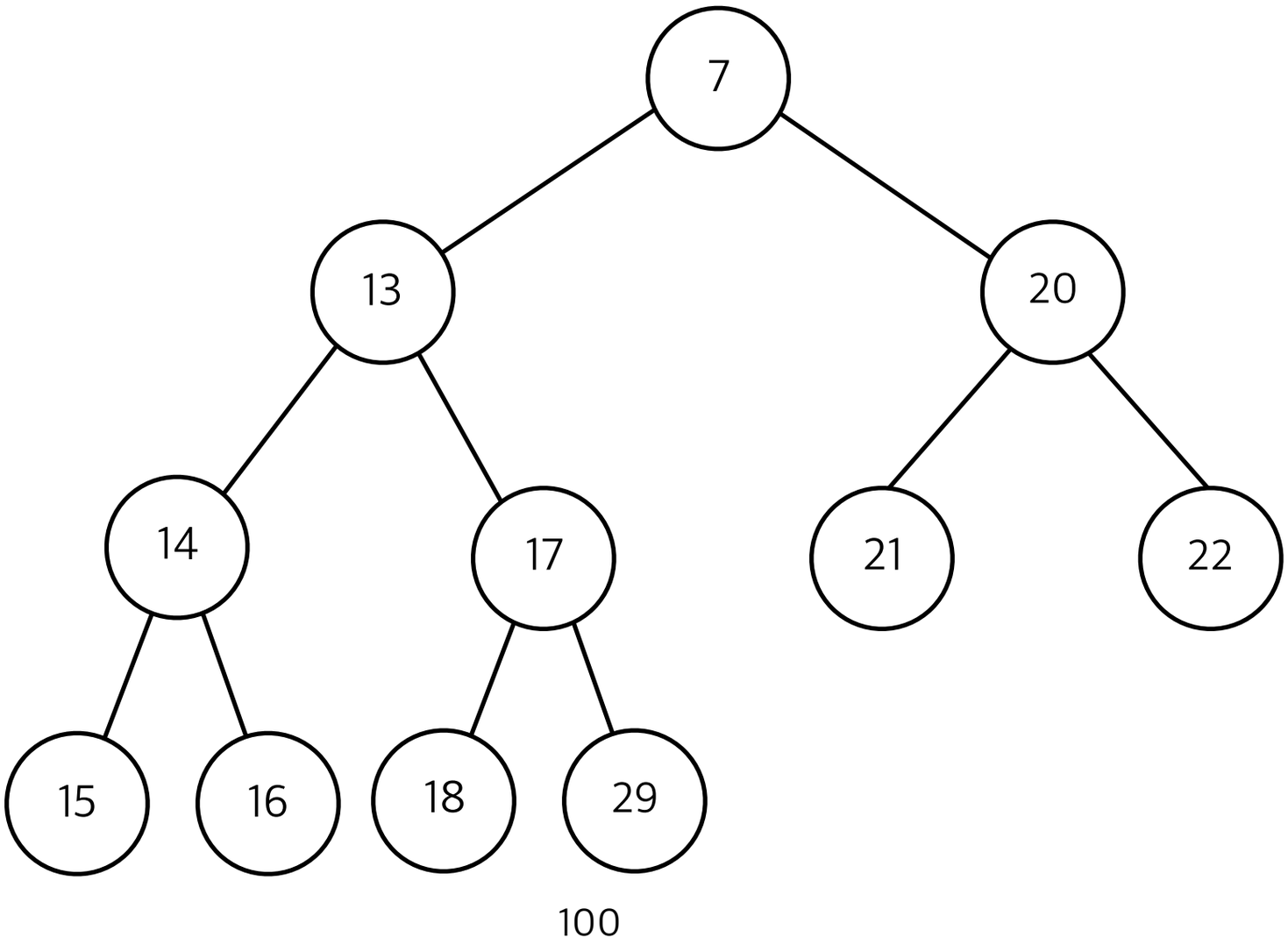
- 부모노드의 값이 자식노드의 값보다 작은 완전 이진 트리

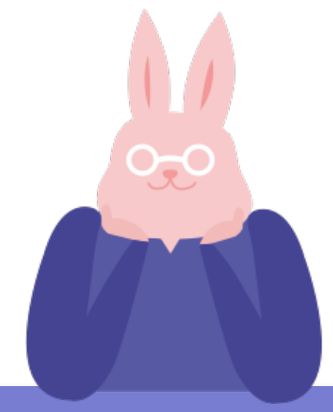
- 지원하는 연산
  - Top() : 최솟값을 반환한다
  - Push(x) : 값 x를 힙에 넣는다
  - Pop() : 최솟값을 힙에서 제거한다



트리 구조의 이용 : 최소힙의 Top()

- 루트에 항상 최솟값이 있다 : O(1)

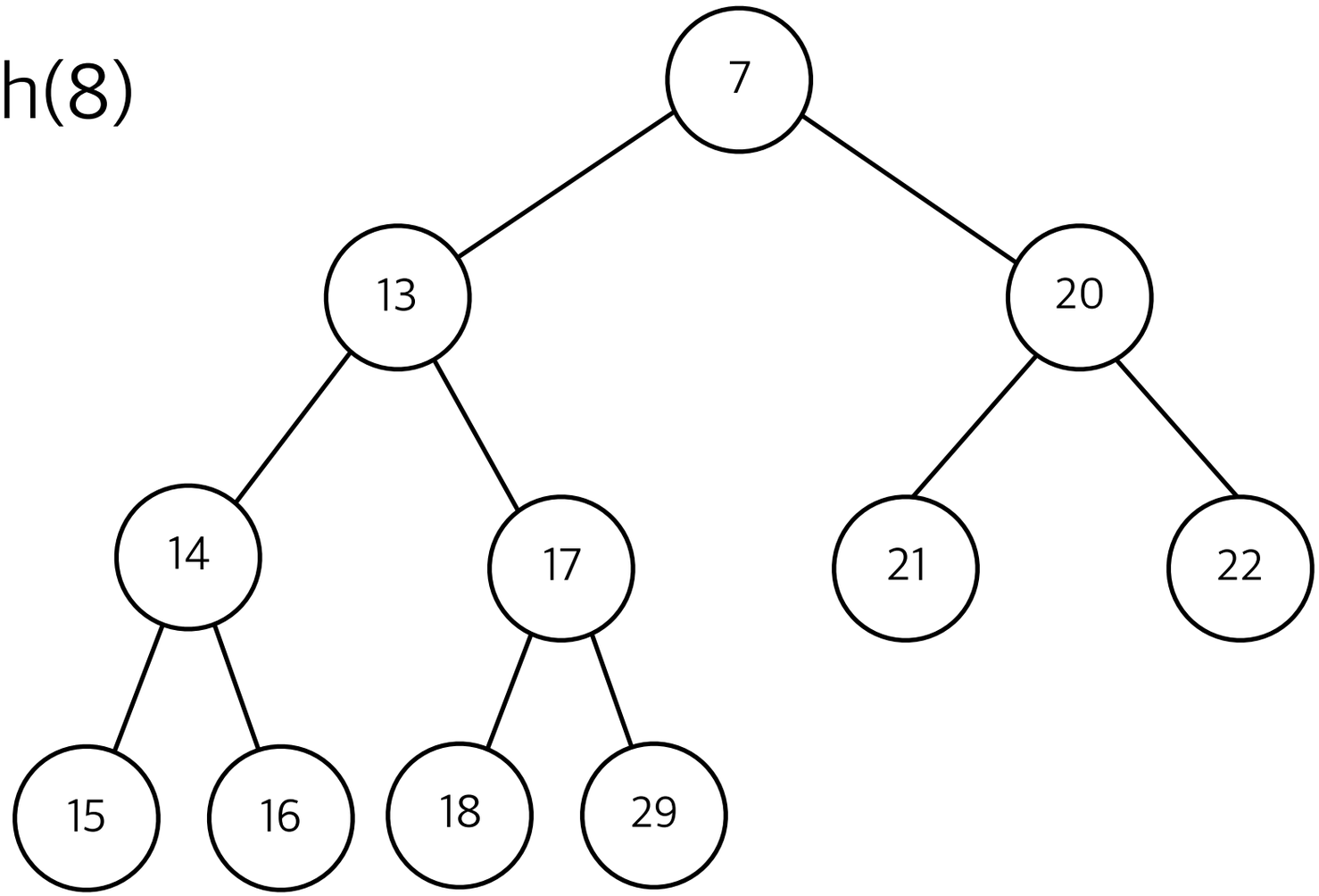


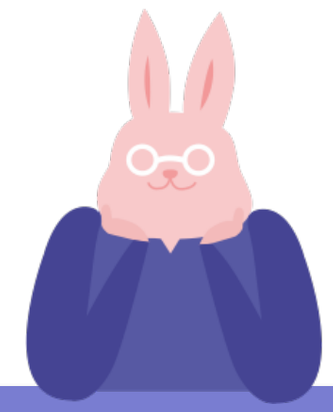


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(8)

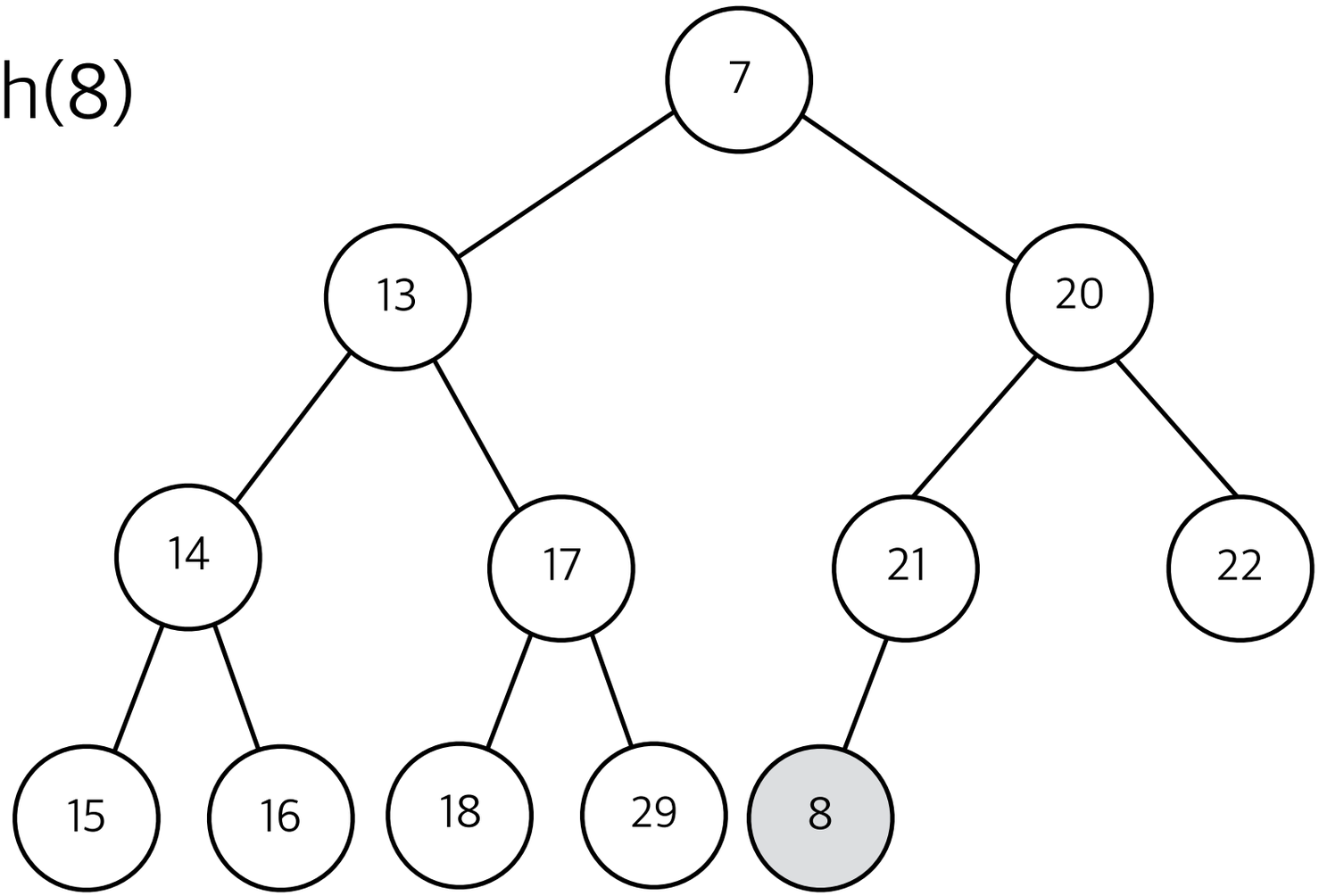


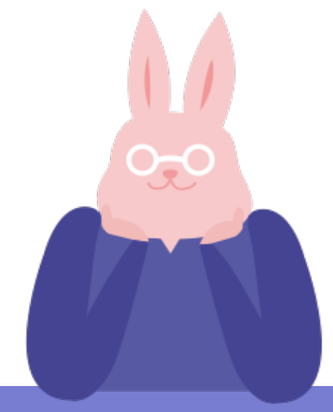


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(8)

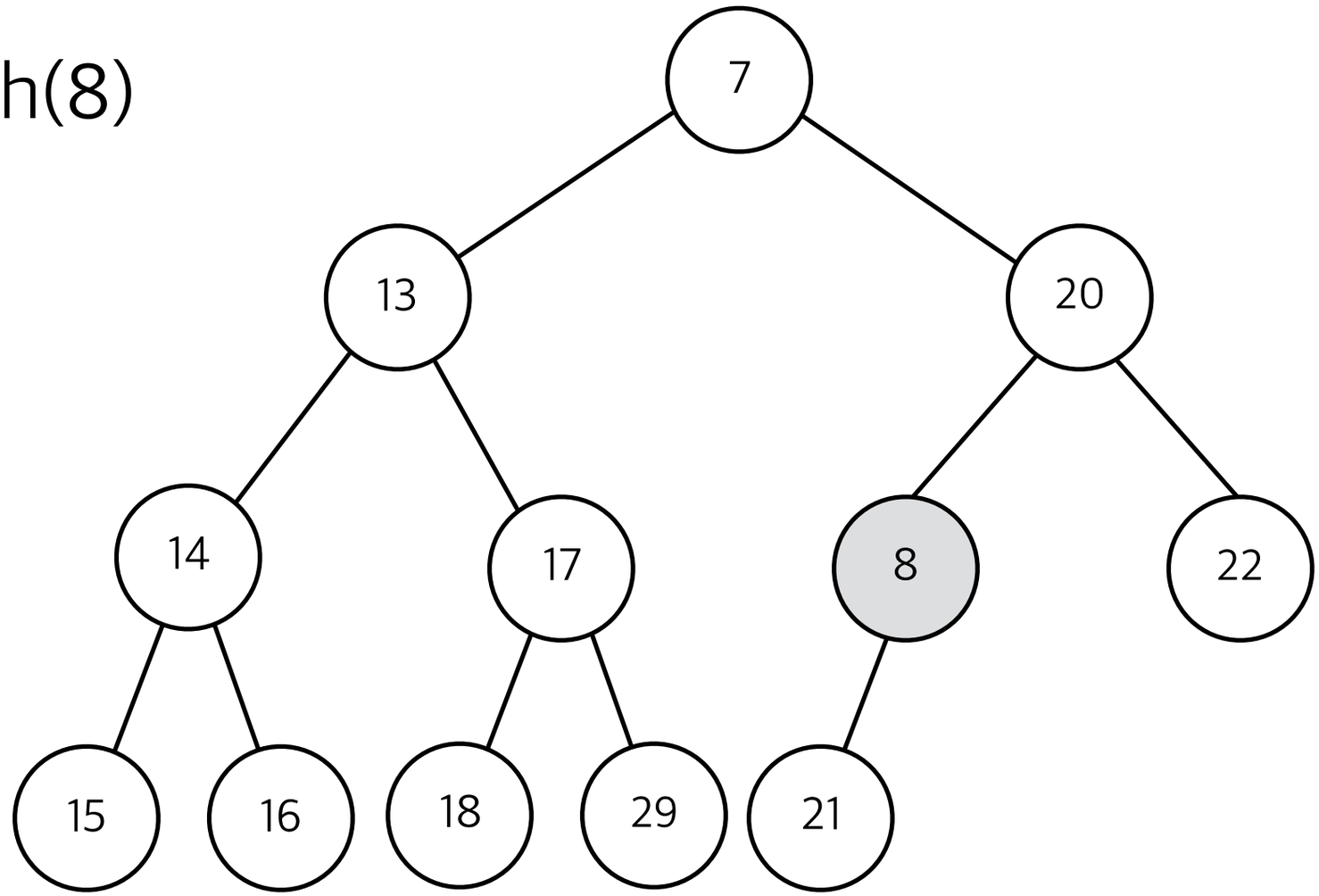


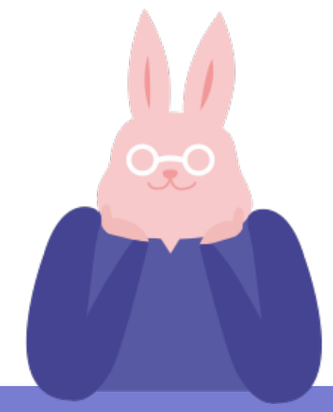


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(8)

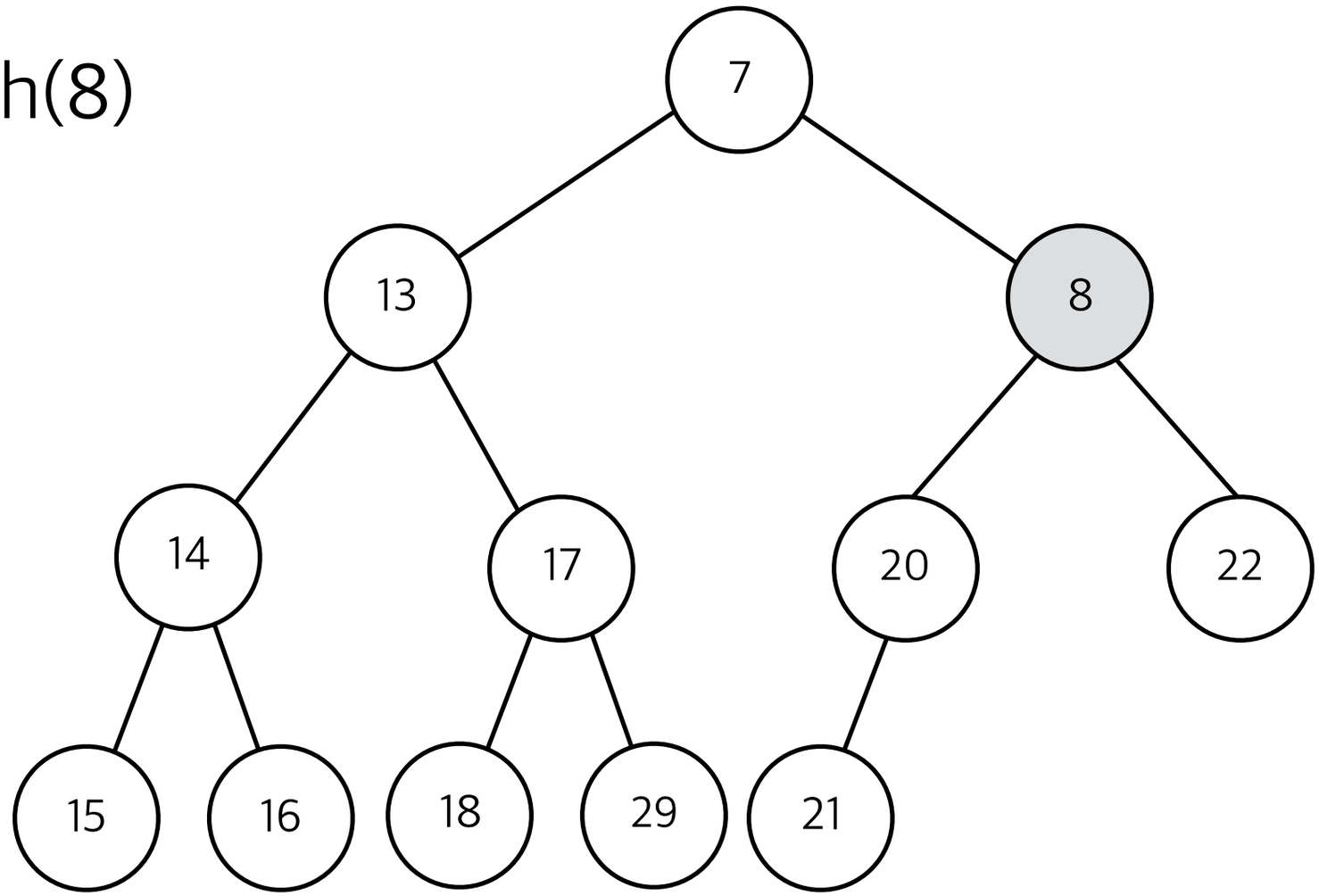


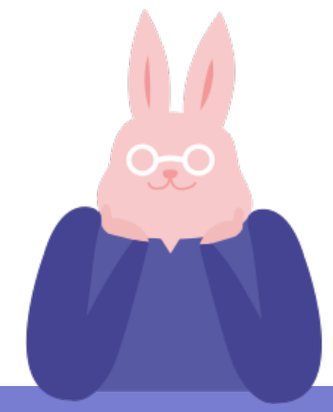


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(8)

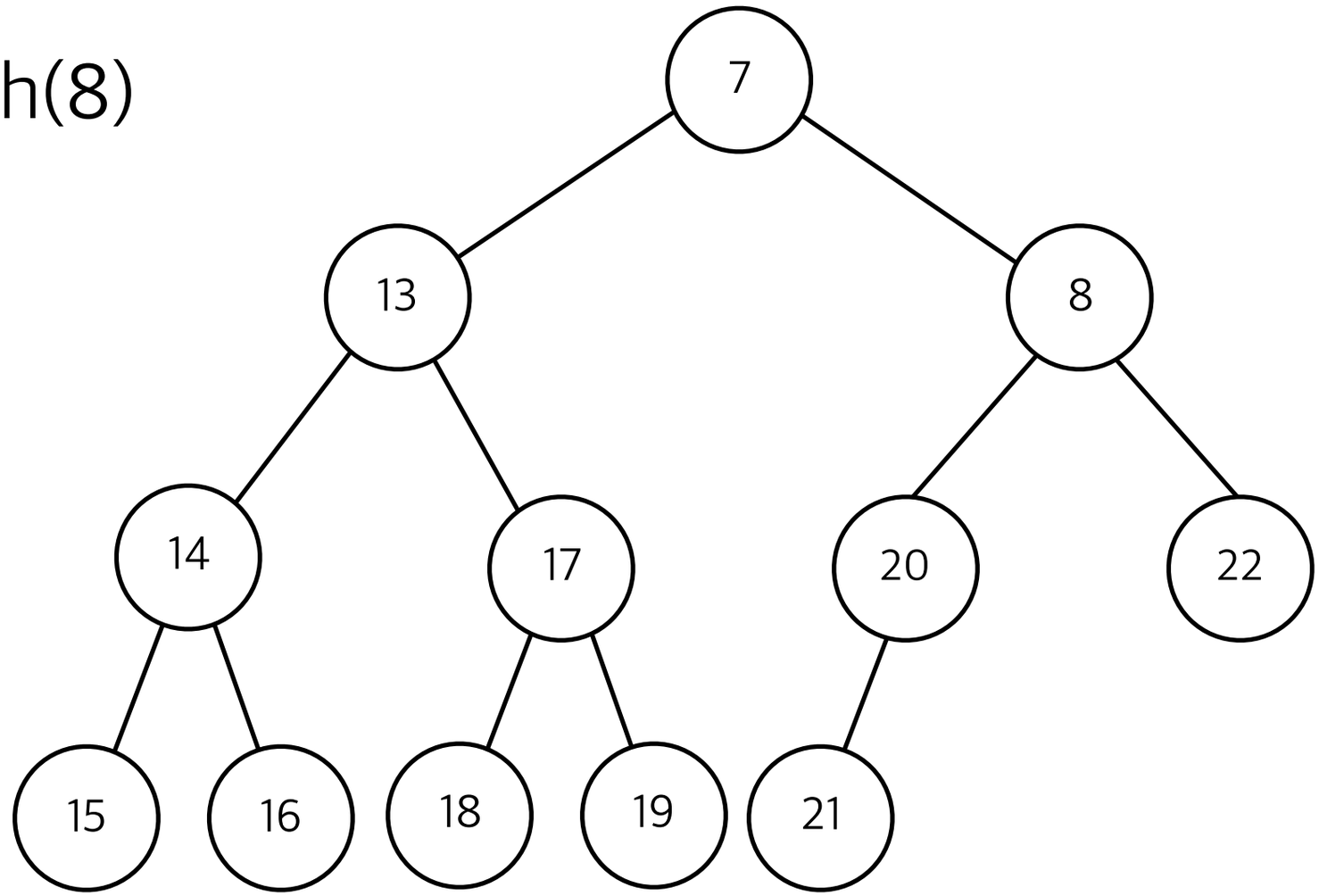


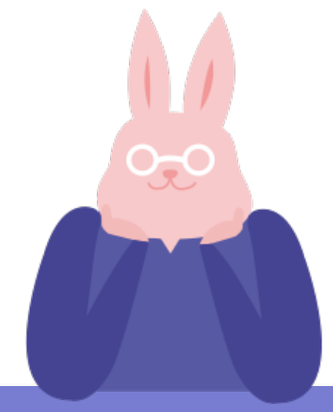


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(8)

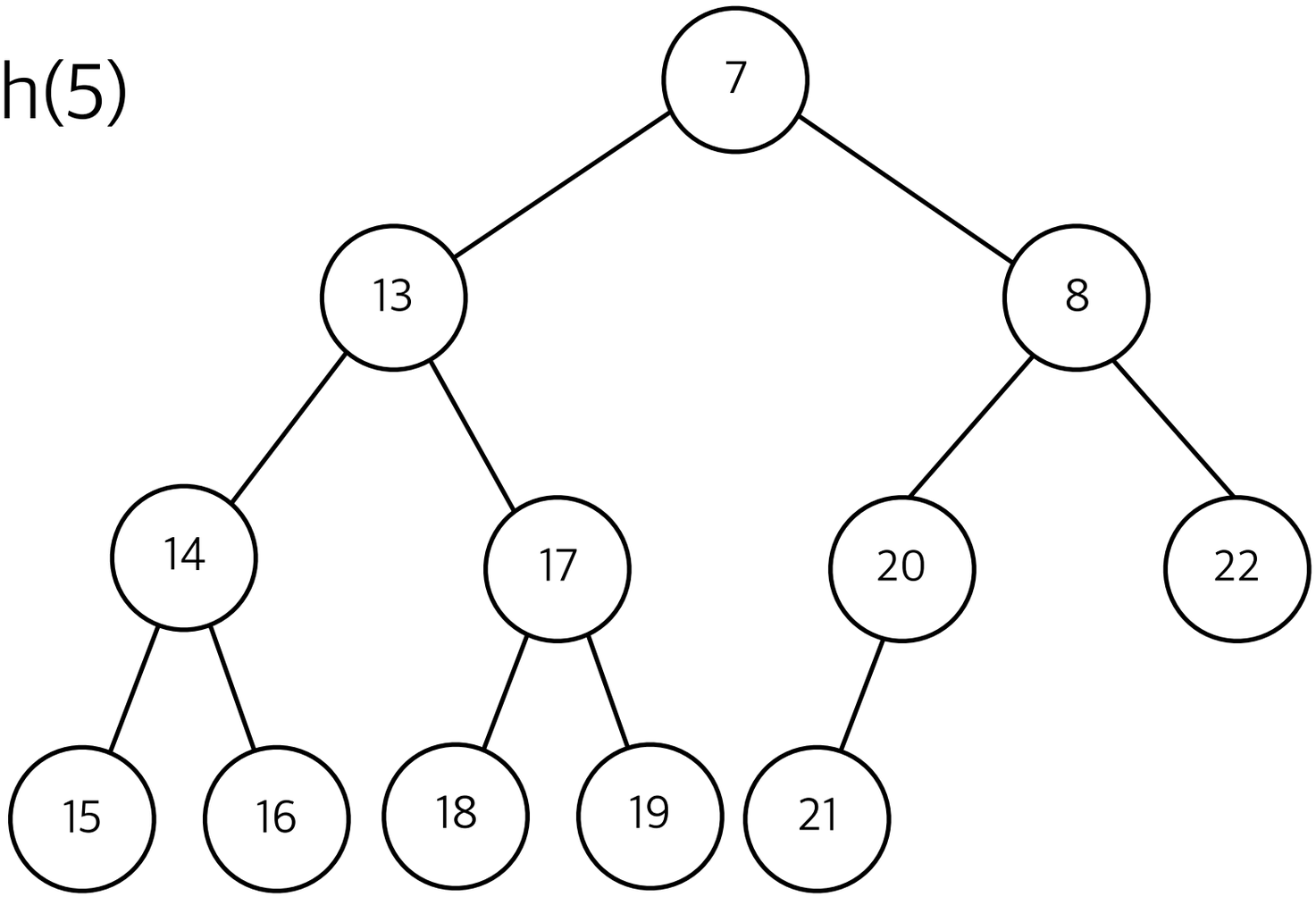


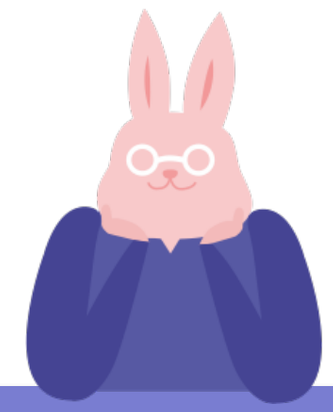


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(5)

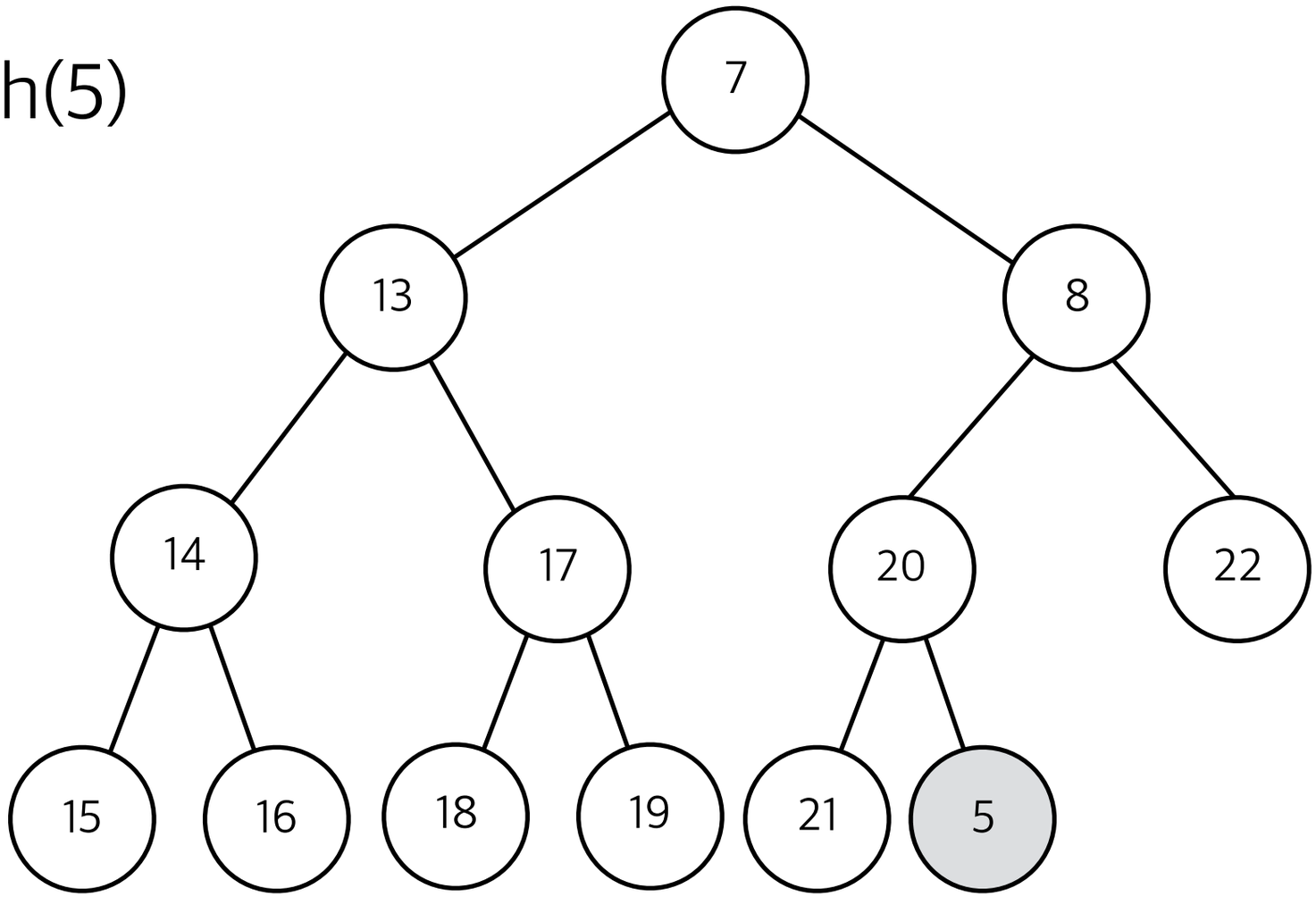


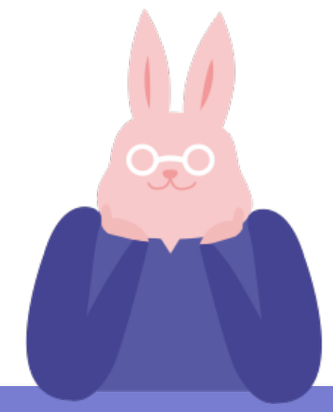


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(5)

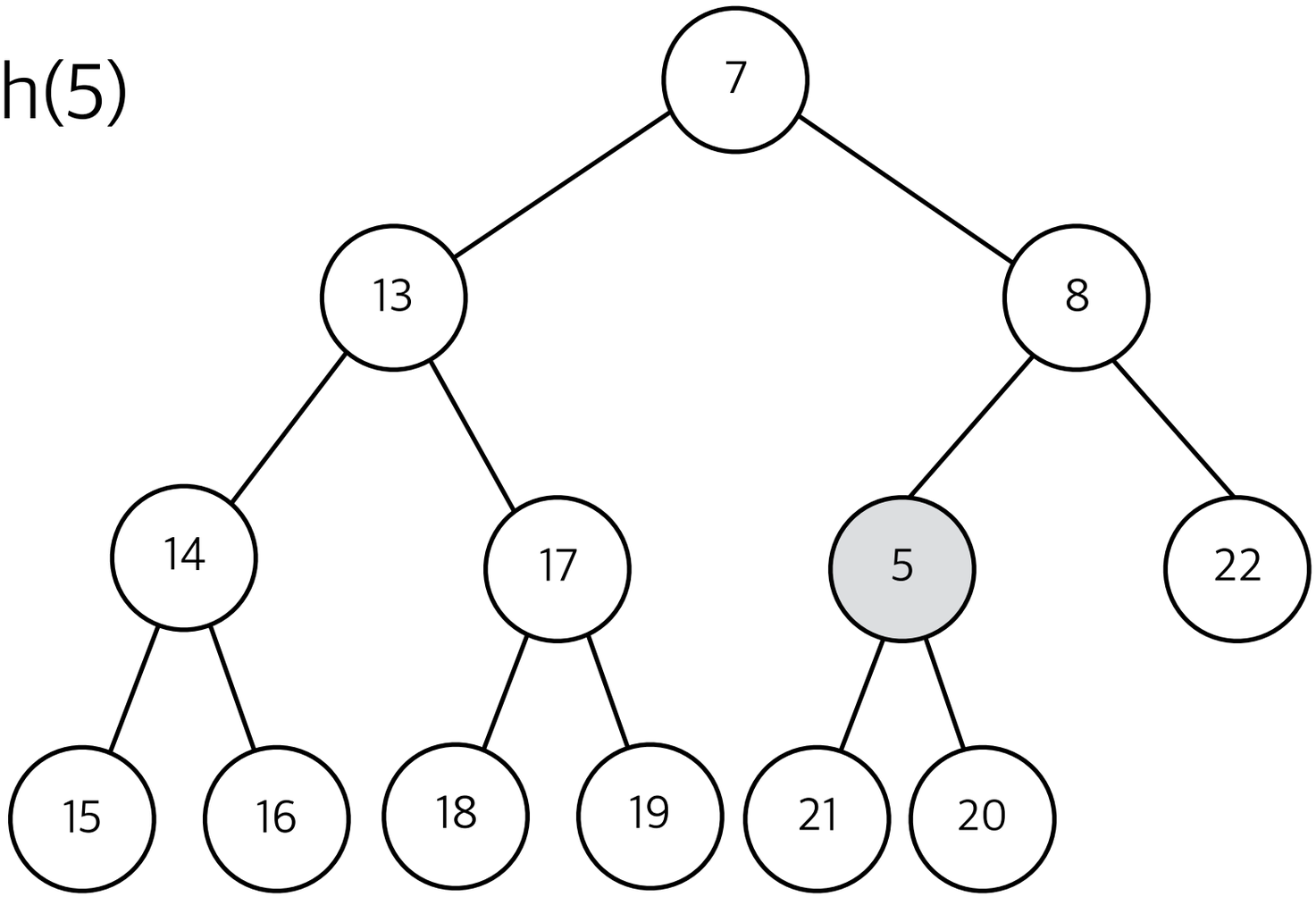


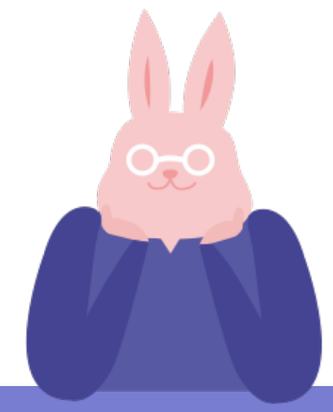


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(5)

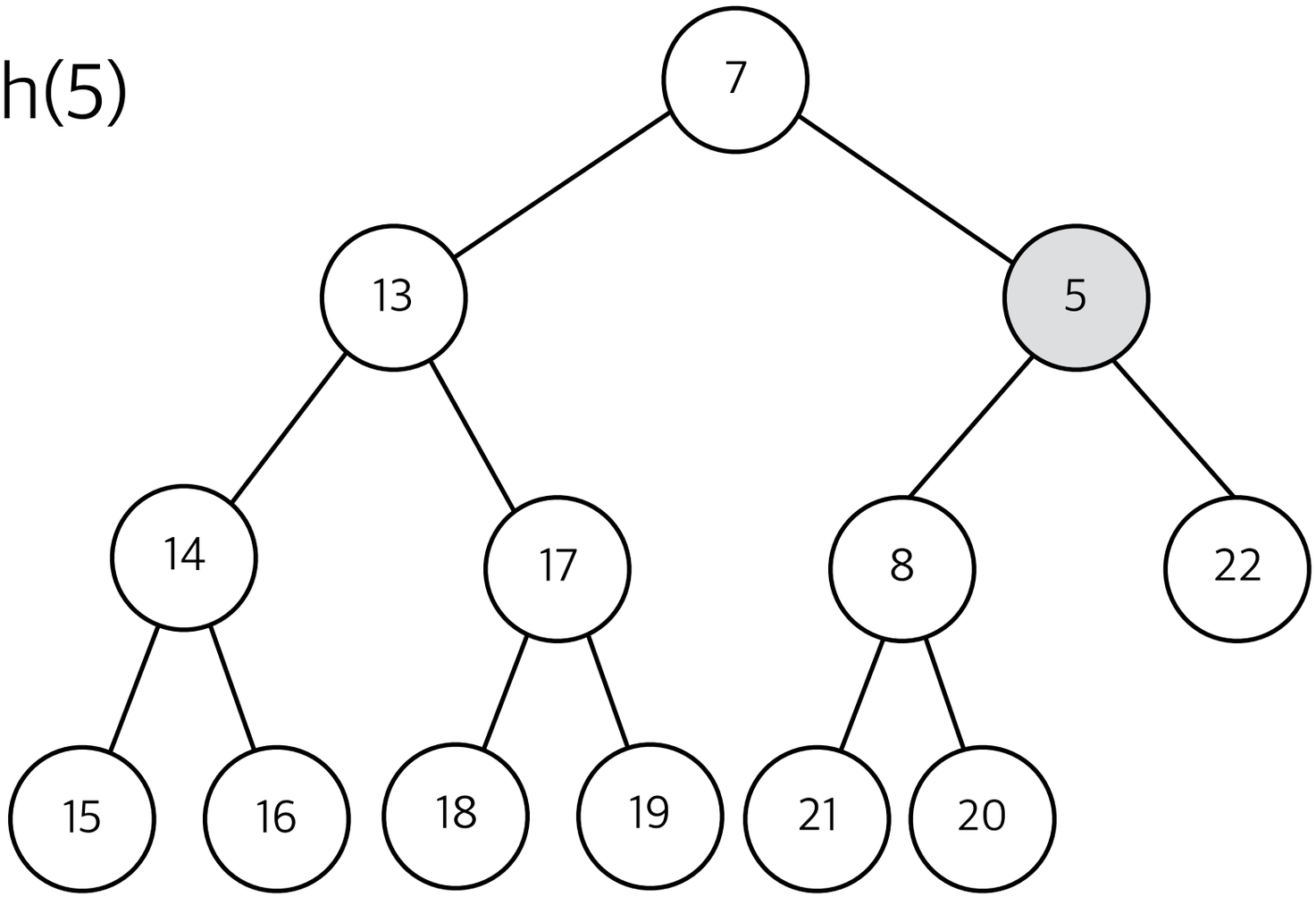


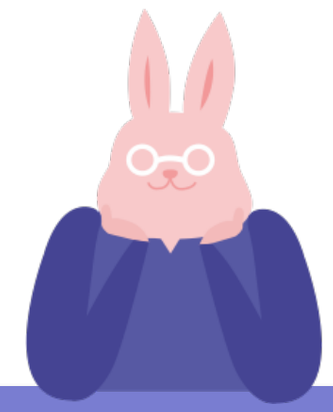


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

Push(5)

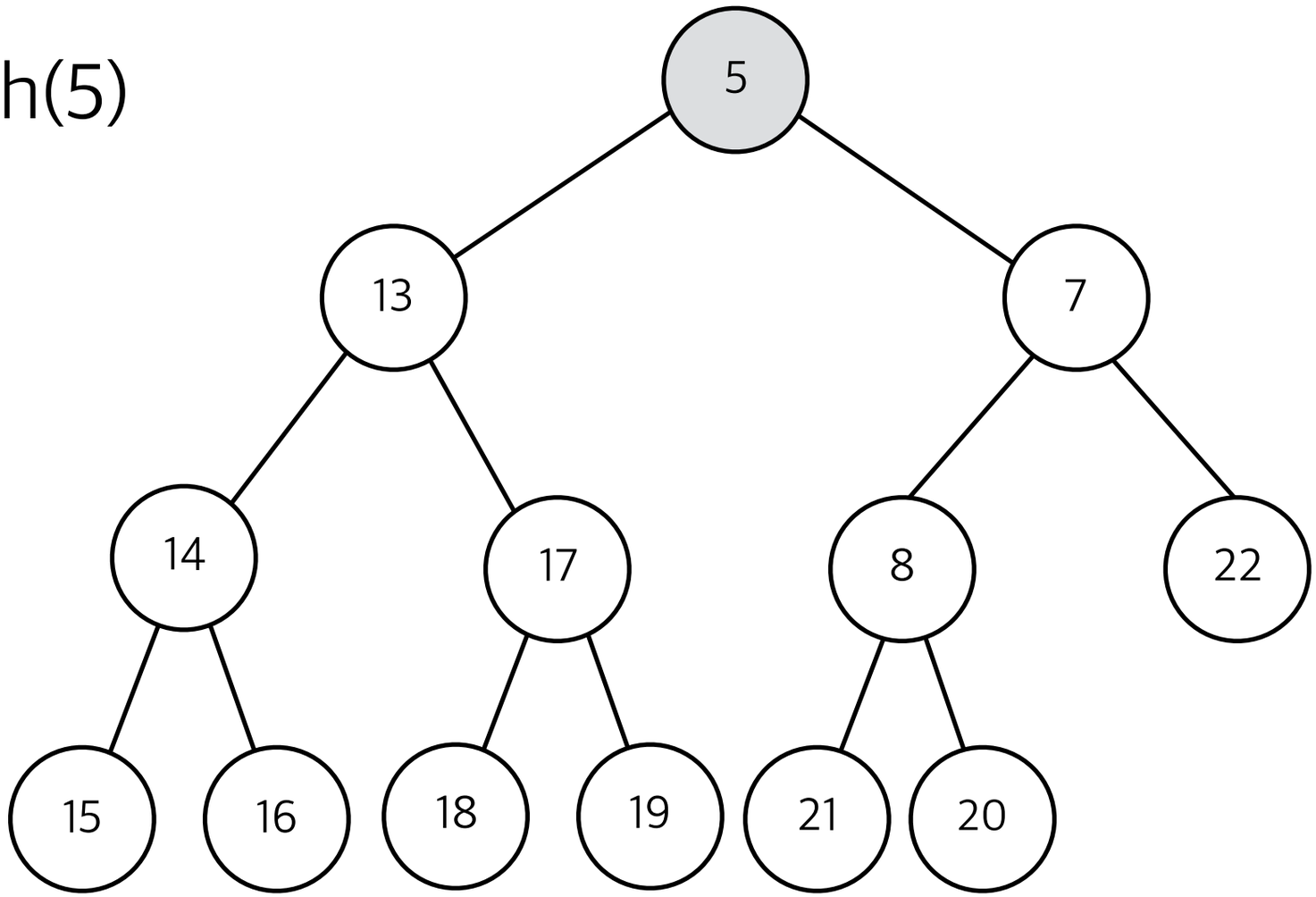


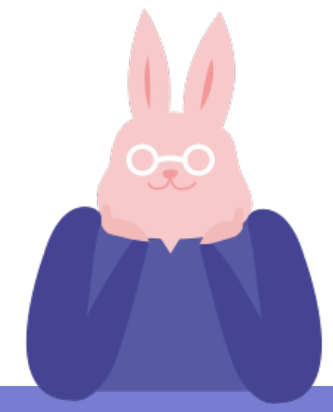


트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다

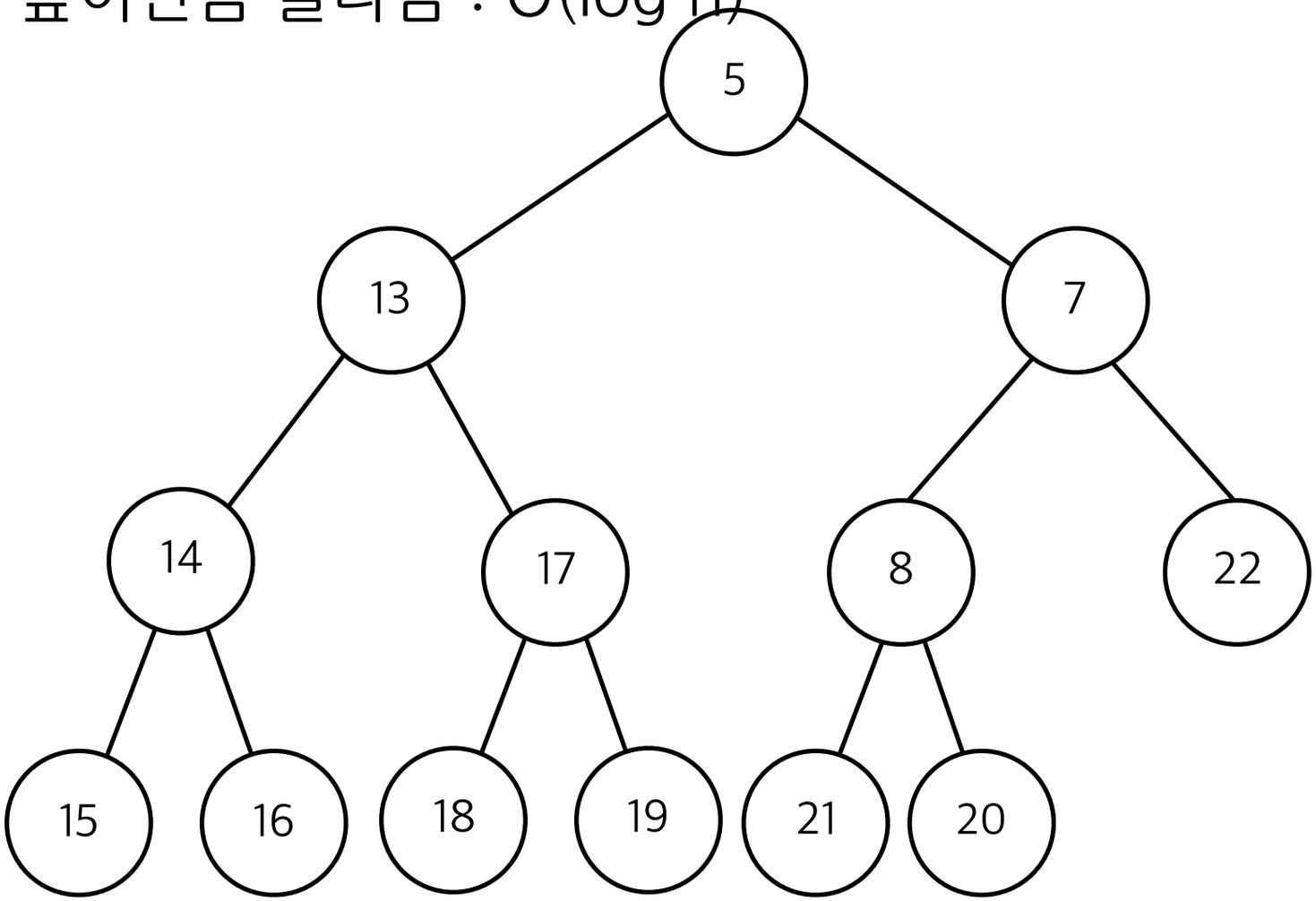
Push(5)

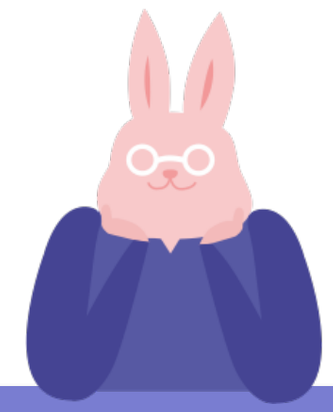




트리 구조의 이용 : 최소힙의 Push(x)

- 힙의 마지막 자리에 Node를 만들고 위치를 재정비 한다
  - 트리의 높이만큼 올라감 : O(log n)

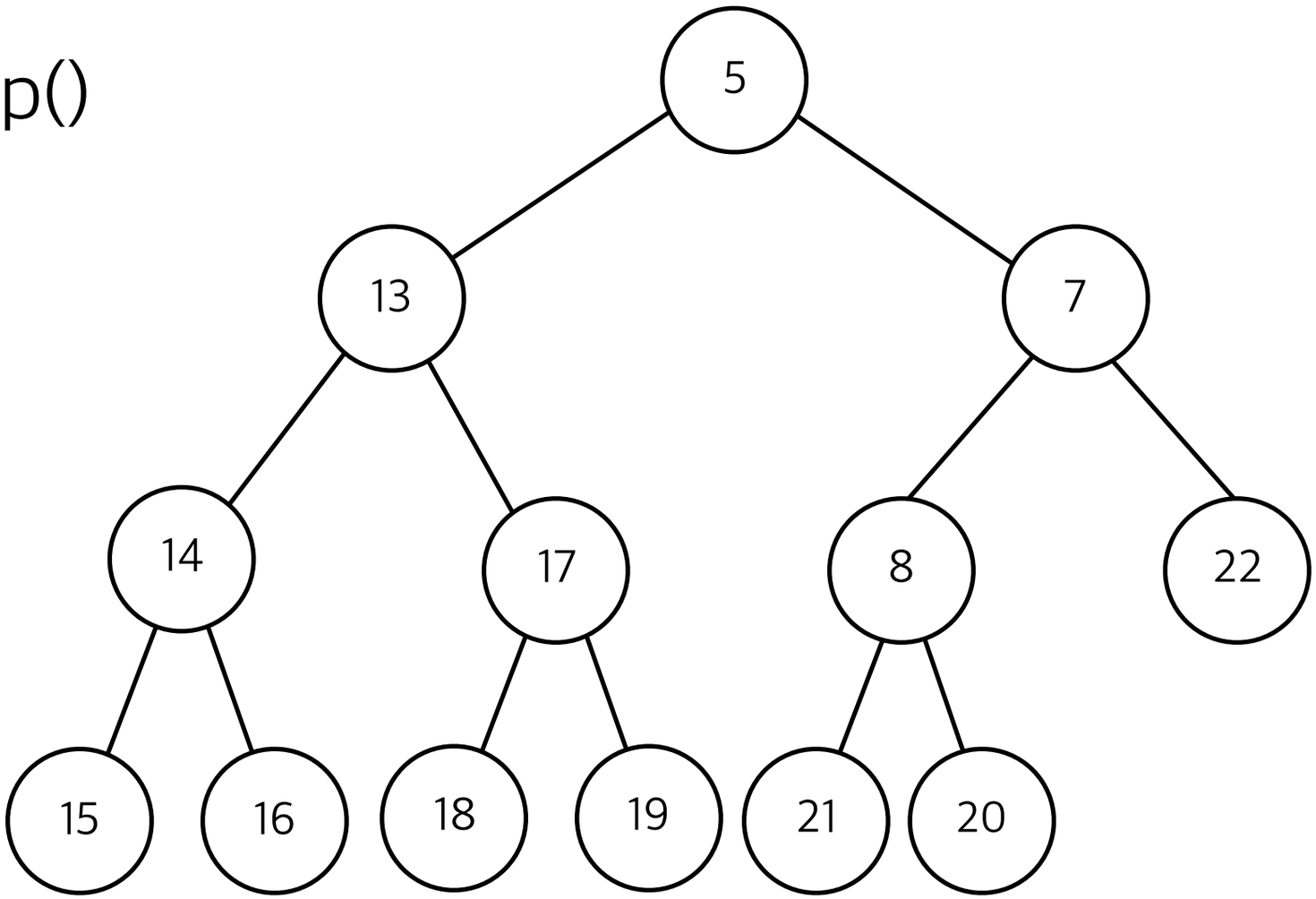


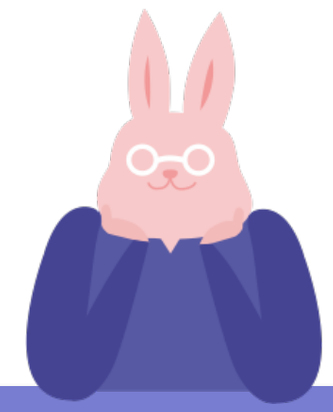


트리 구조의 이용 : 최소힙의 Pop()

- 힙의 마지막 자리에 있는 Node를 Root로 올린다

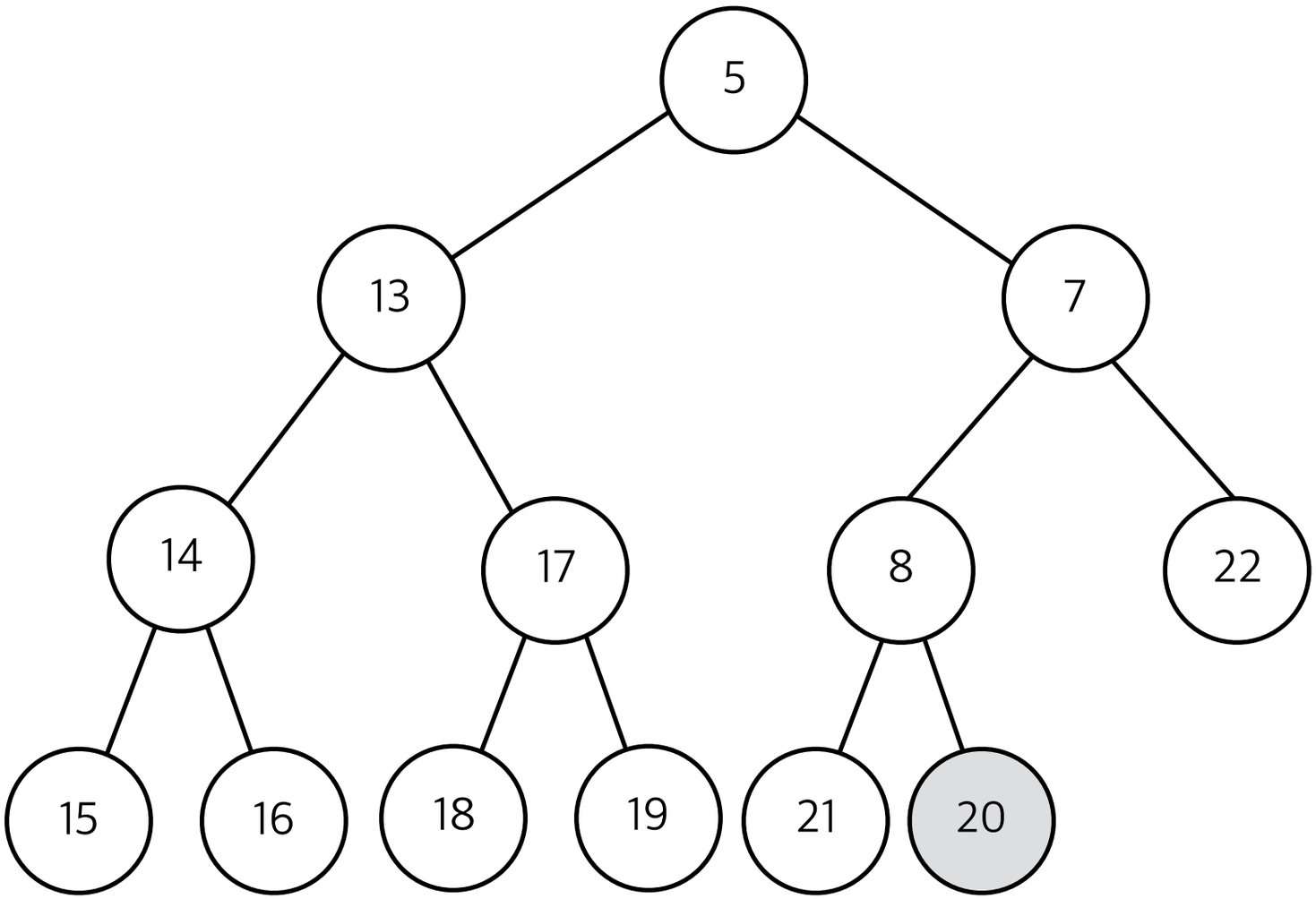
Pop()

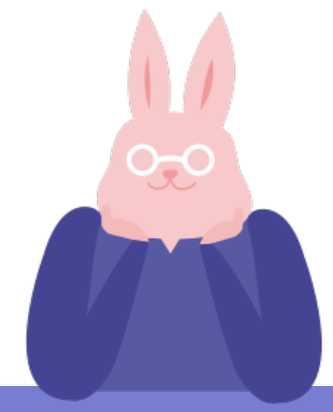




트리 구조의 이용 : 최소힙의 Pop()

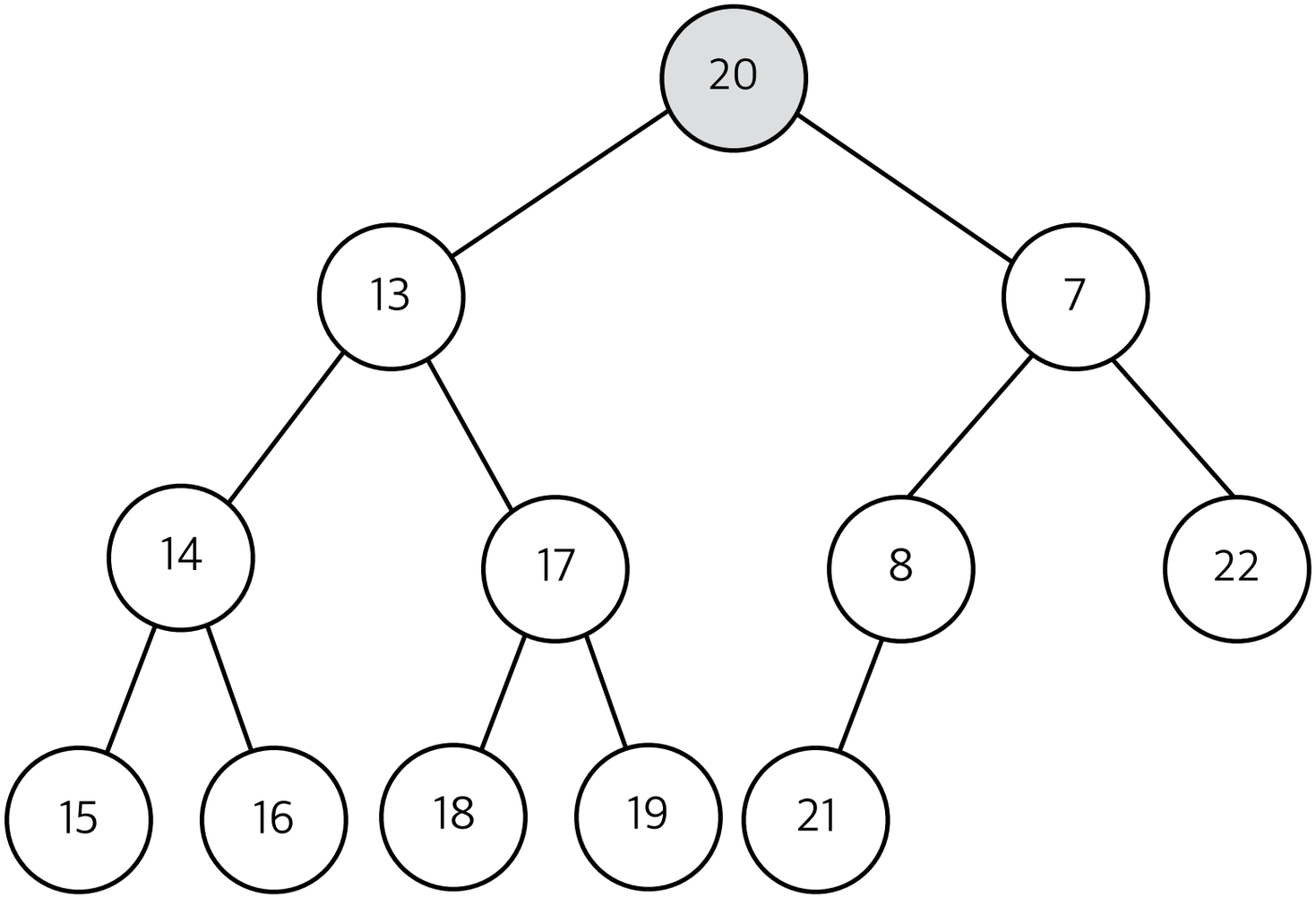
- 힙의 마지막 자리에 있는 Node를 Root로 올린다

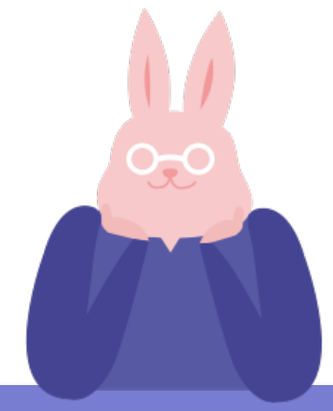




트리 구조의 이용 : 최소힙의 Pop()

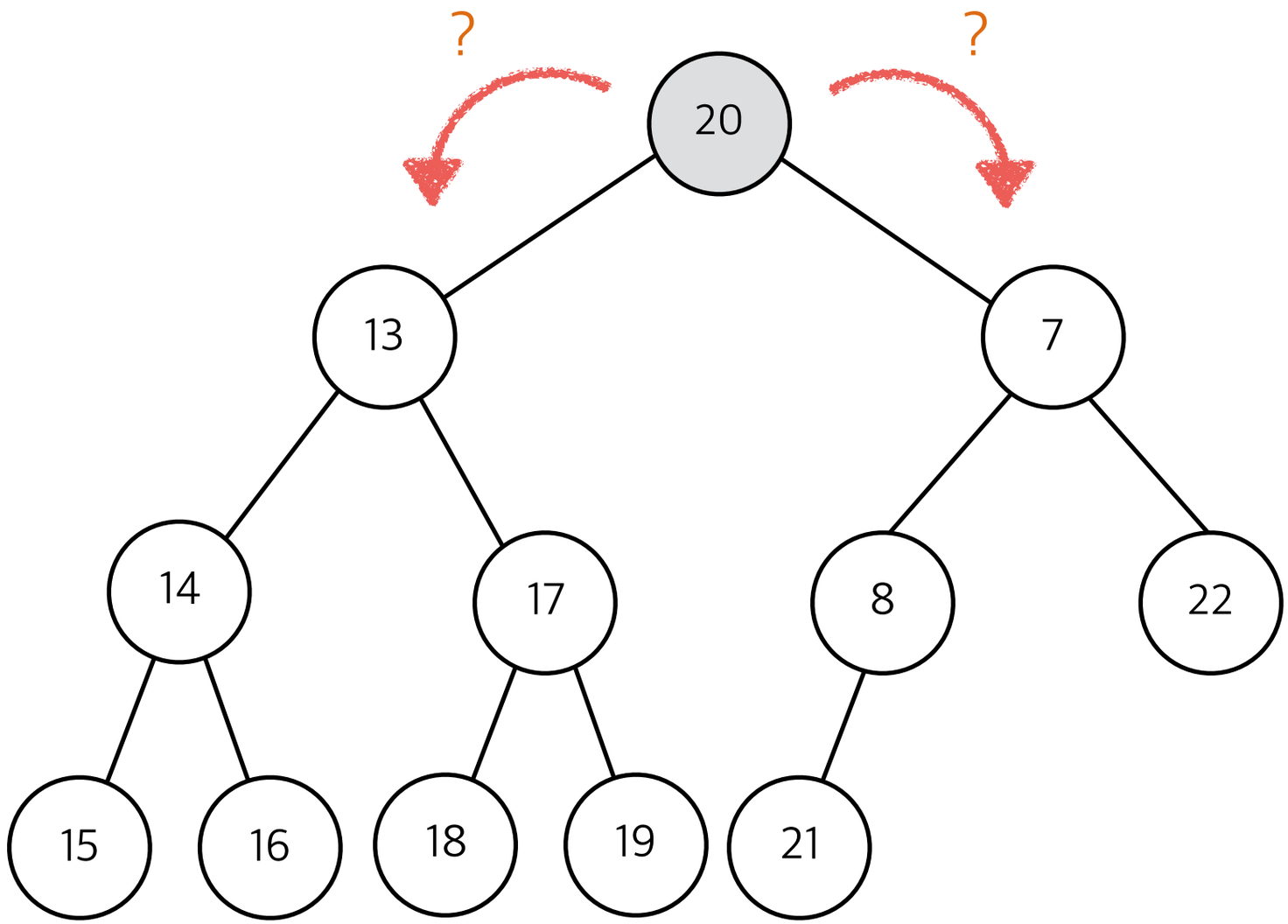
- 힙의 마지막 자리에 있는 Node를 Root로 올린다

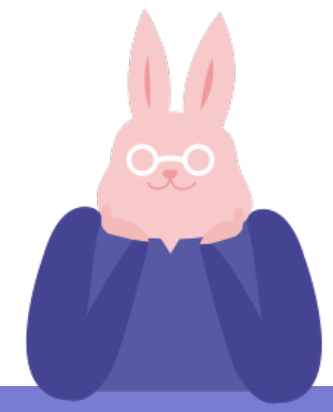




트리 구조의 이용 : 최소힙의 Pop()

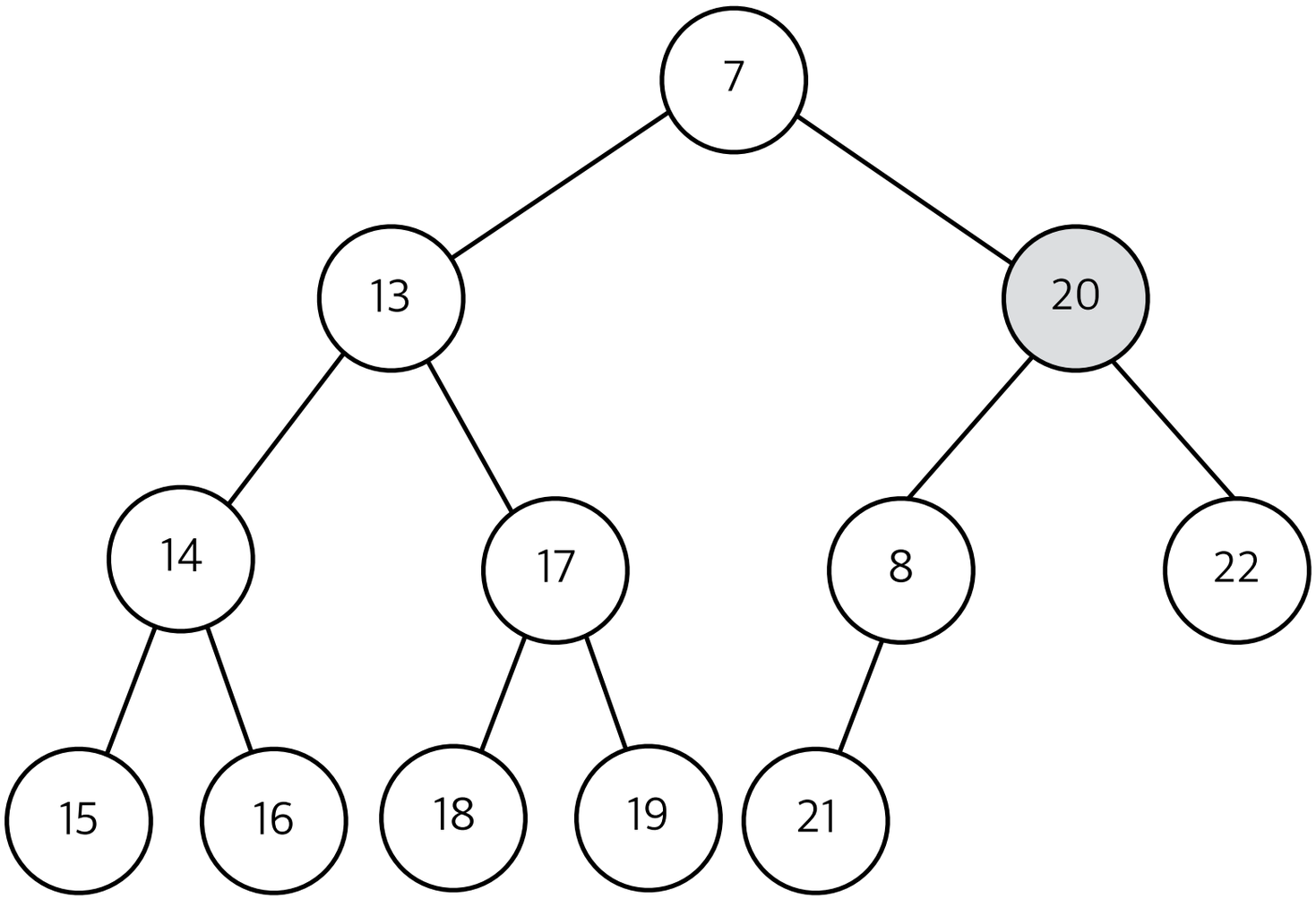
- 힙의 마지막 자리에 있는 Node를 Root로 올린다

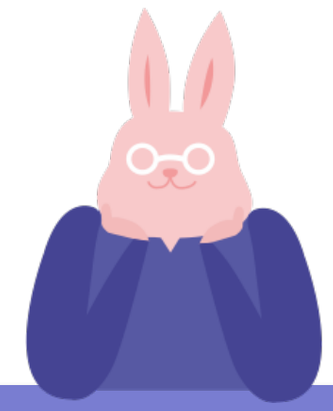




트리 구조의 이용 : 최소힙의 Pop()

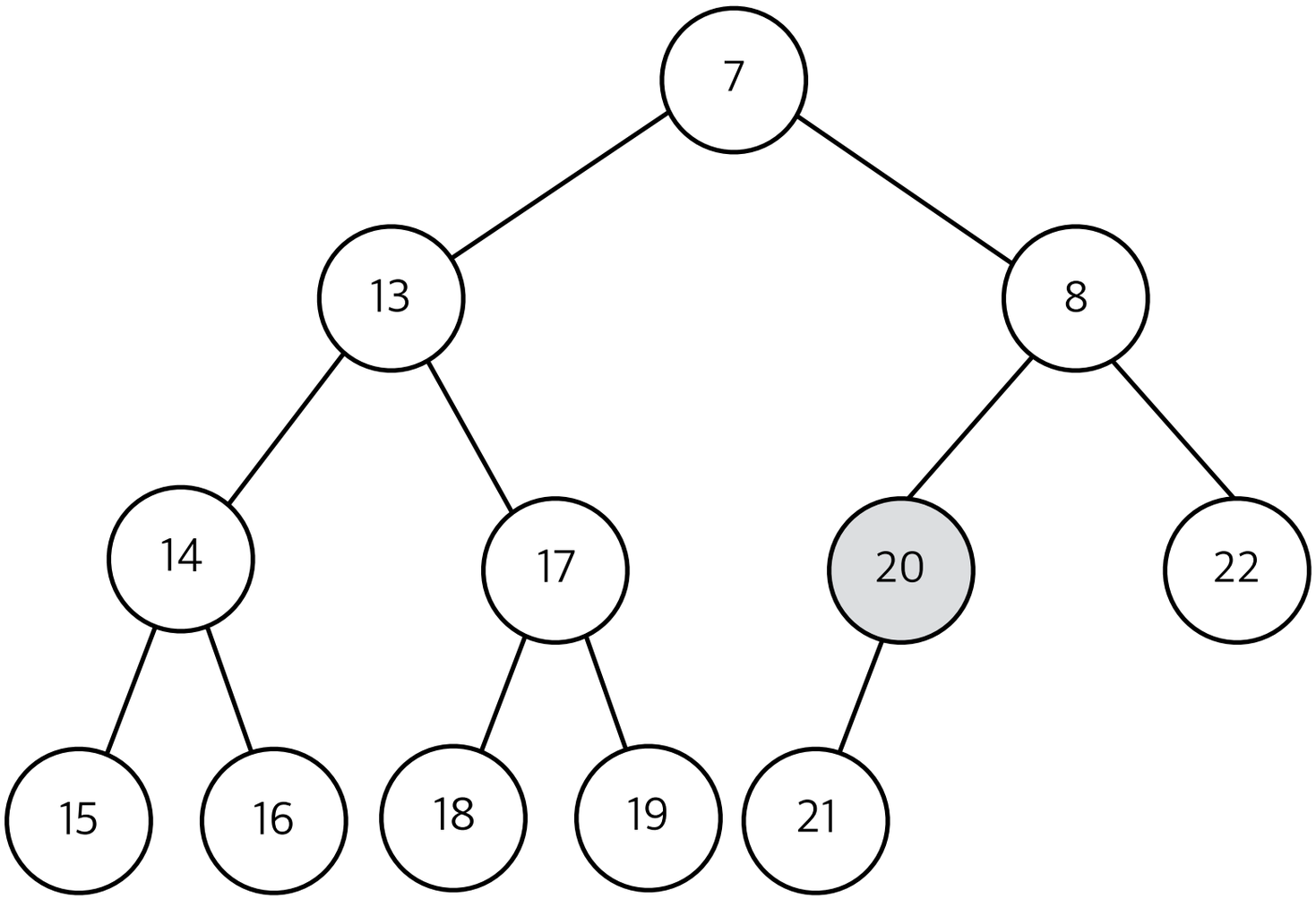
- 힙의 마지막 자리에 있는 Node를 Root로 올린다

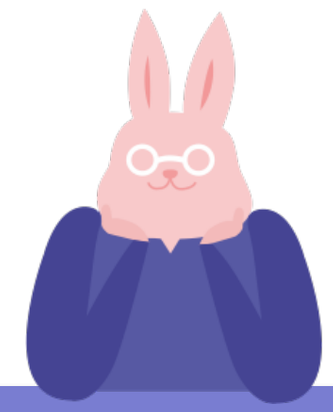




트리 구조의 이용 : 최소힙의 Pop()

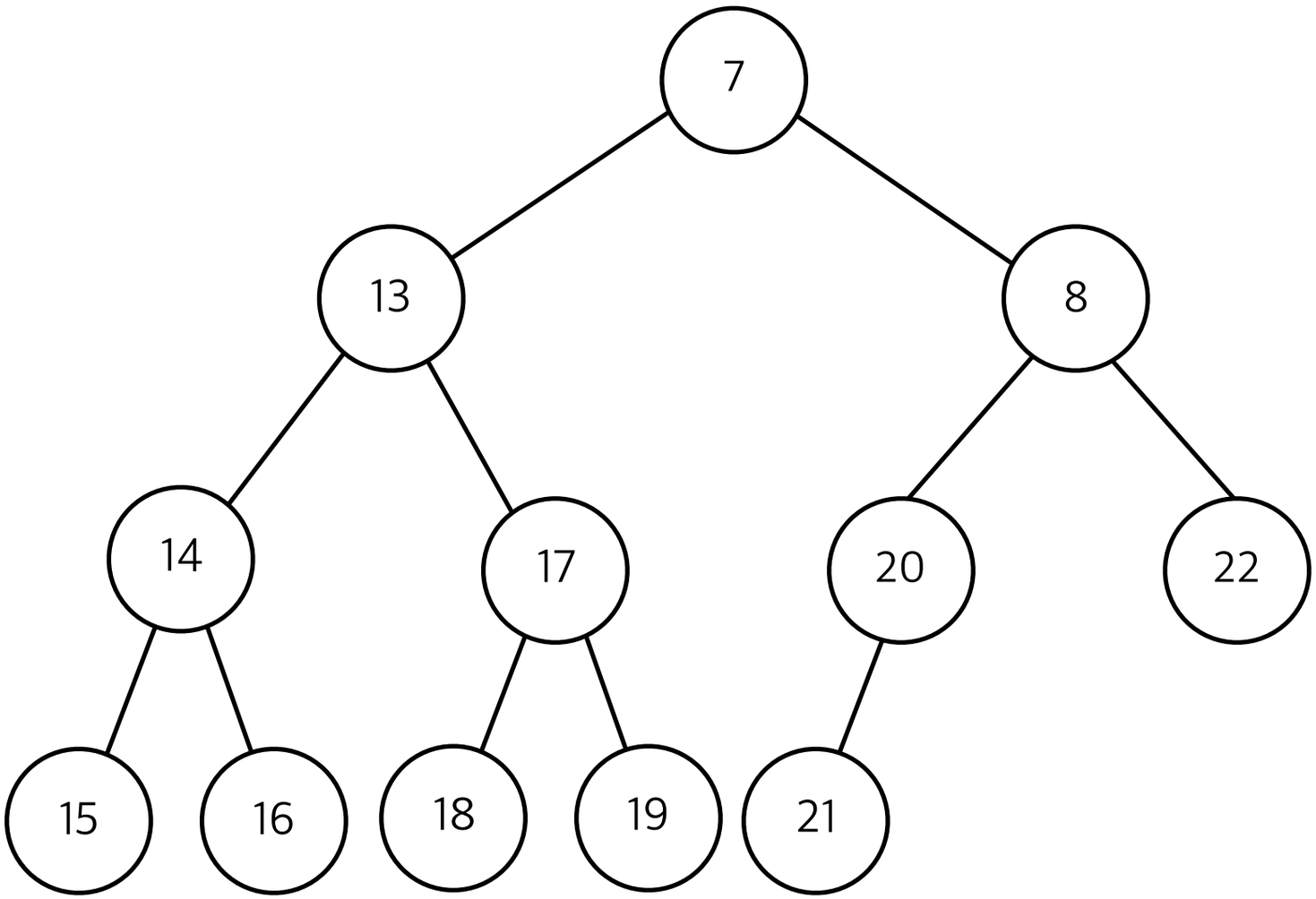
- 힙의 마지막 자리에 있는 Node를 Root로 올린다

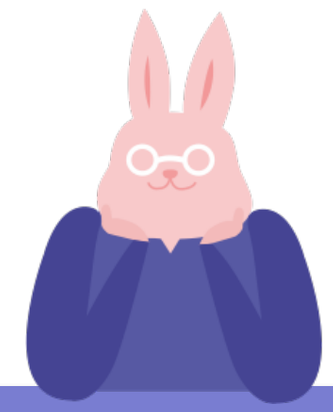




트리 구조의 이용 : 최소힙의 Pop()

- 힙의 마지막 자리에 있는 Node를 Root로 올린다

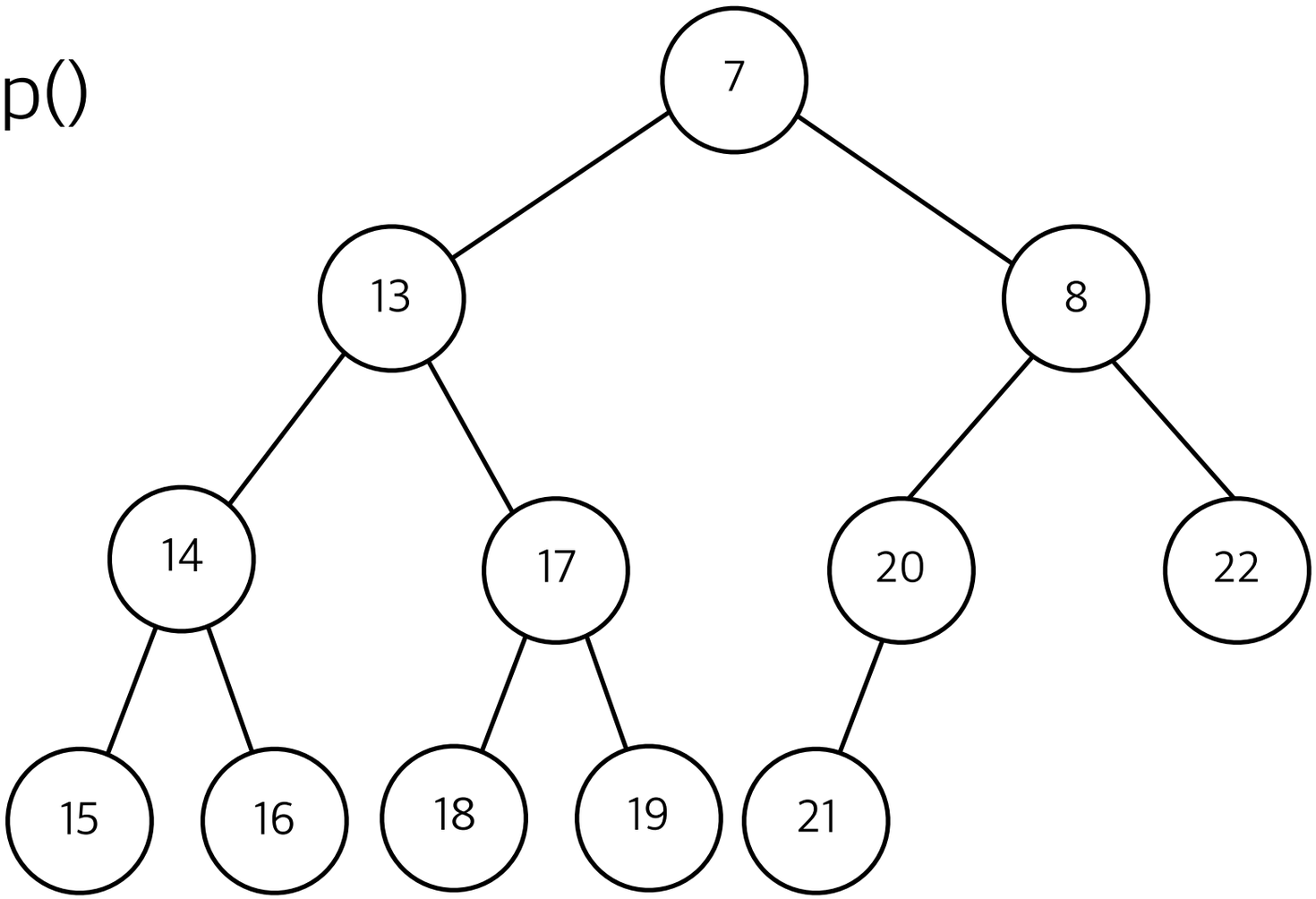


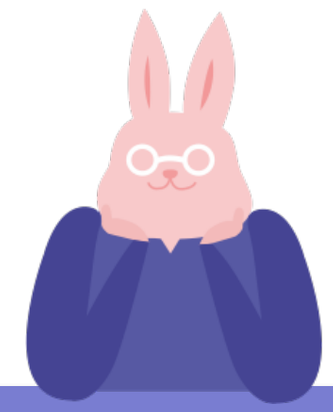


트리 구조의 이용 : 최소힙의 Pop()

- 힙의 마지막 자리에 있는 Node를 Root로 올린다

Pop()

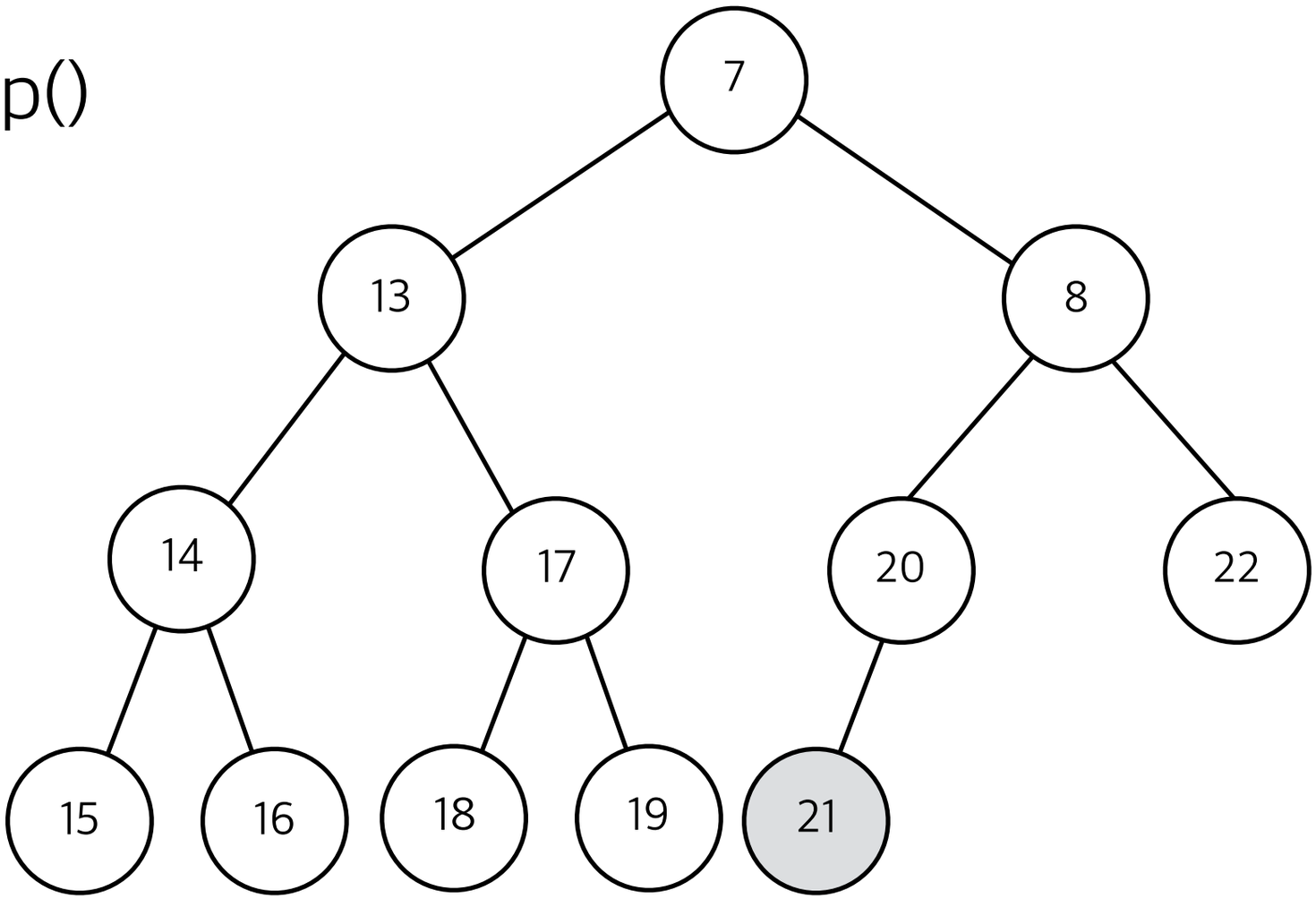


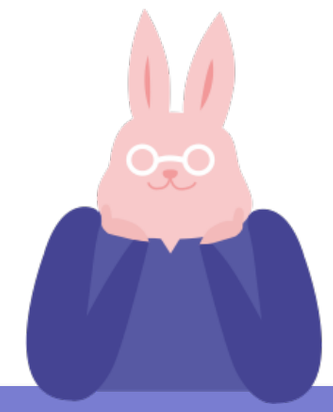


트리 구조의 이용 : 최소힙의 Pop()

- 힙의 마지막 자리에 있는 Node를 Root로 올린다

Pop()

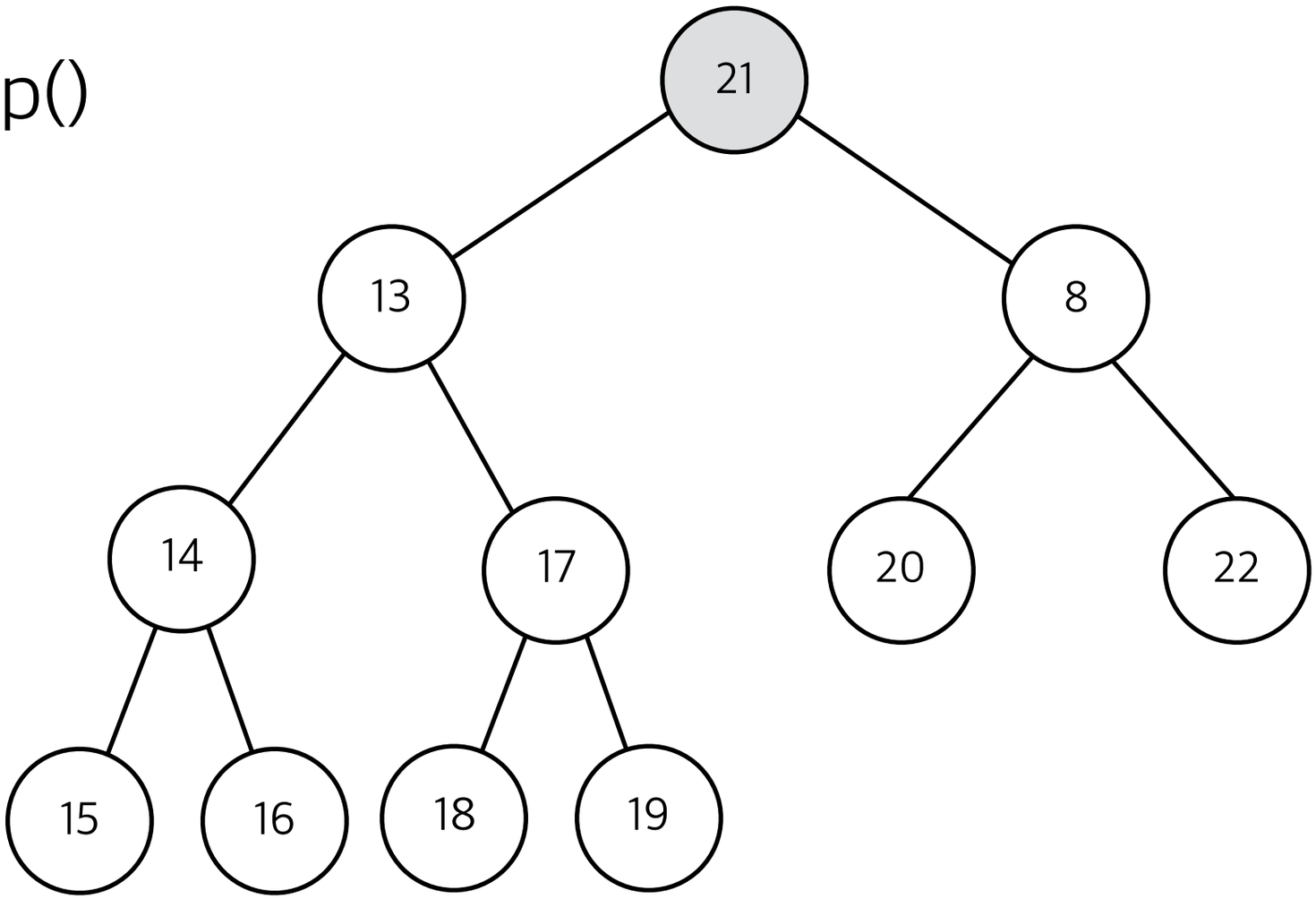


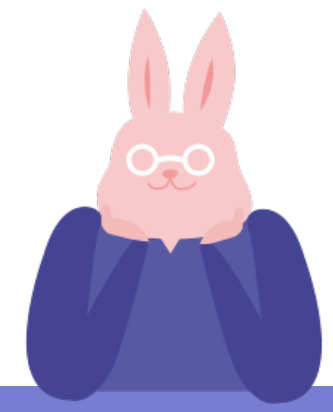


트리 구조의 이용 : 최소힙의 Pop()

- 힙의 마지막 자리에 있는 Node를 Root로 올린다

Pop()

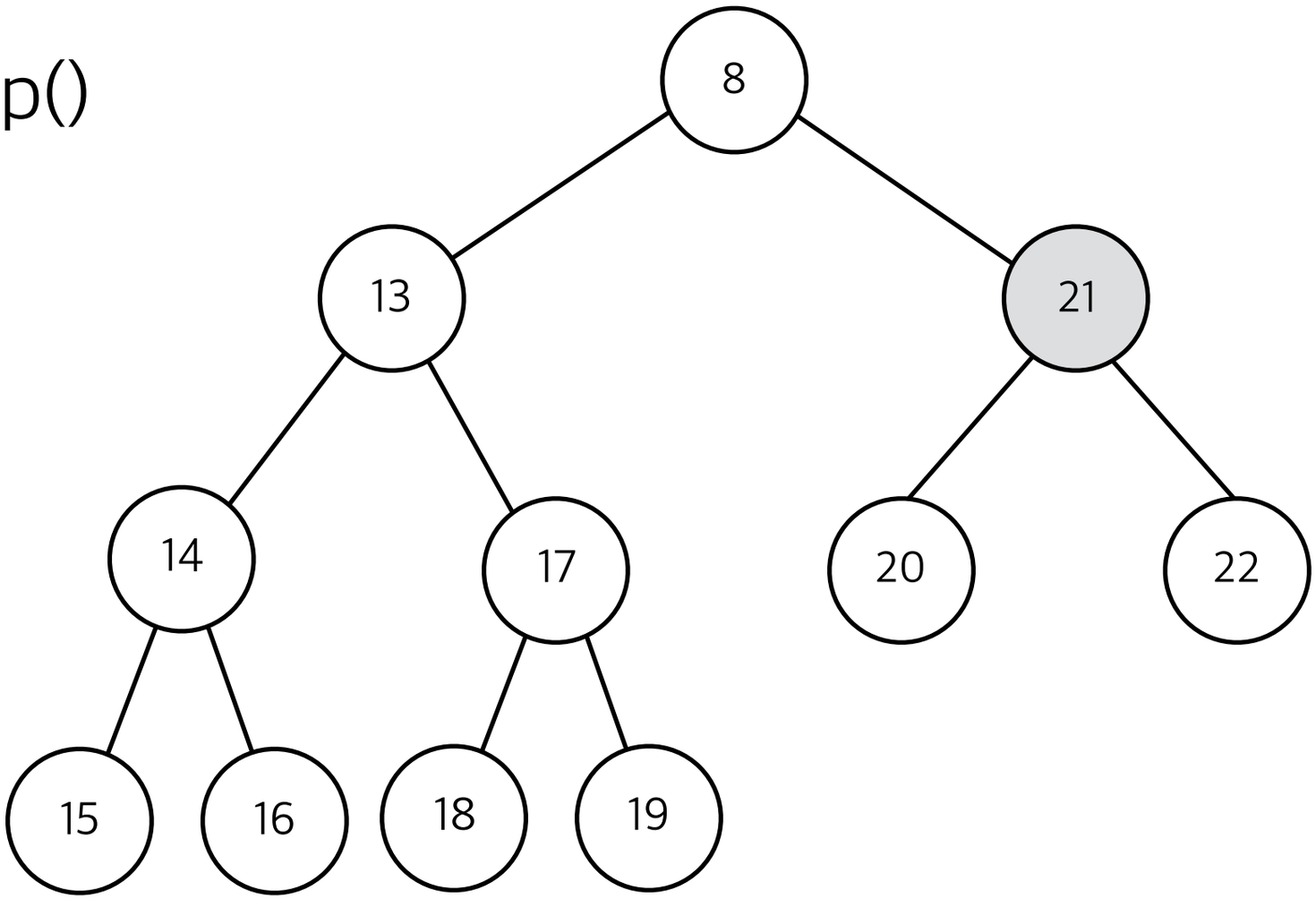


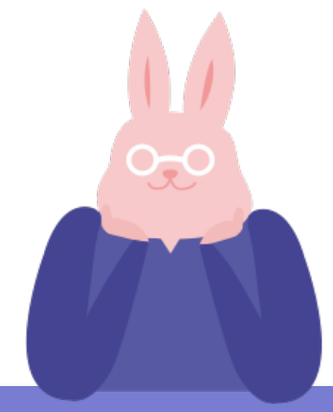


트리 구조의 이용 : 최소힙의 Pop()

- 힙의 마지막 자리에 있는 Node를 Root로 올린다

Pop()

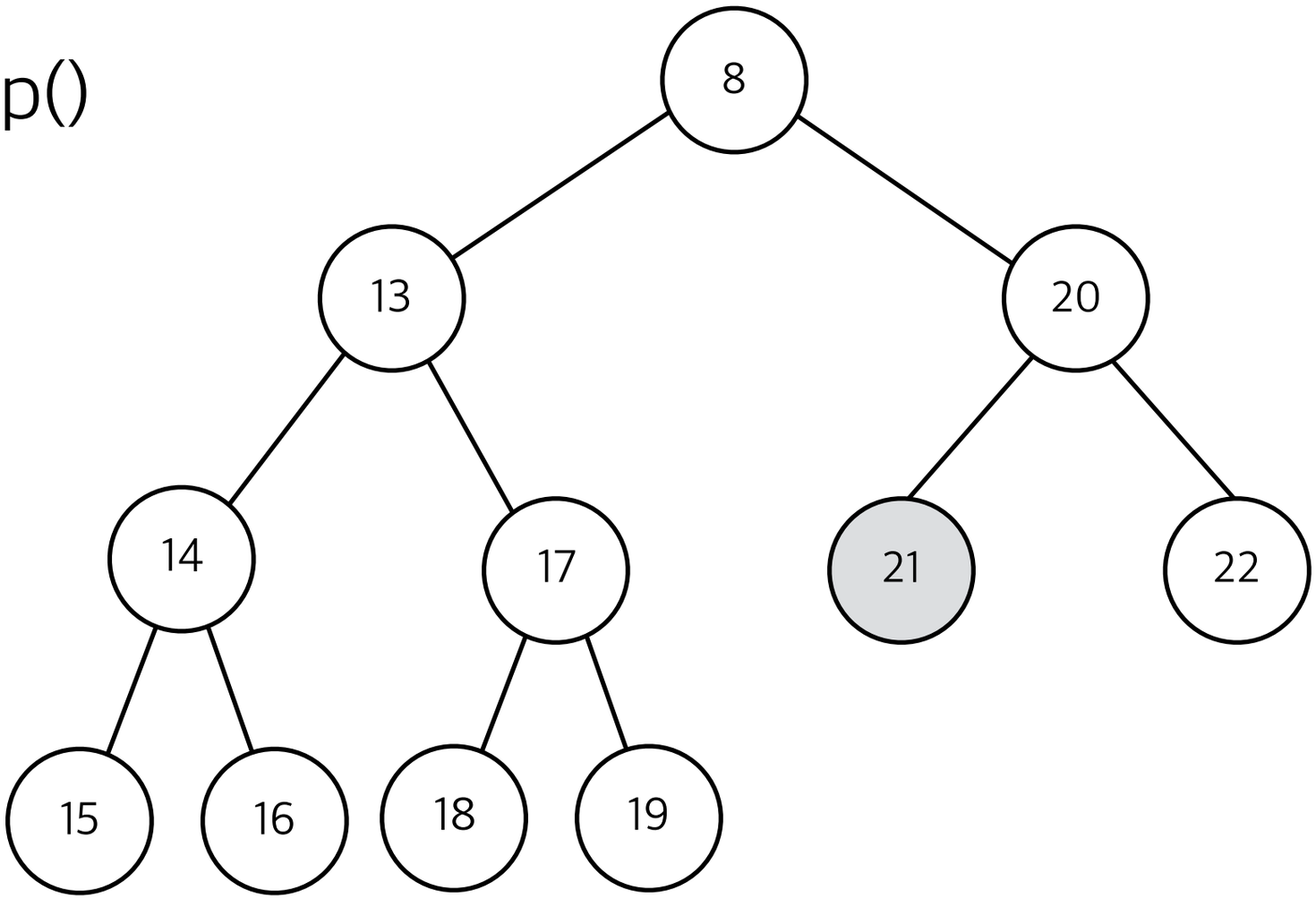


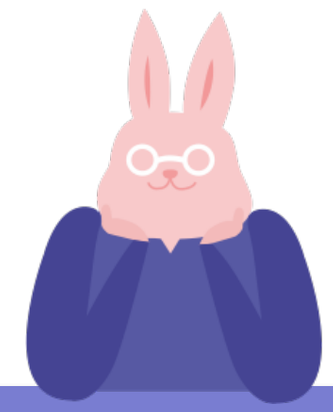


트리 구조의 이용 : 최소힙의 Pop()

- 힙의 마지막 자리에 있는 Node를 Root로 올린다

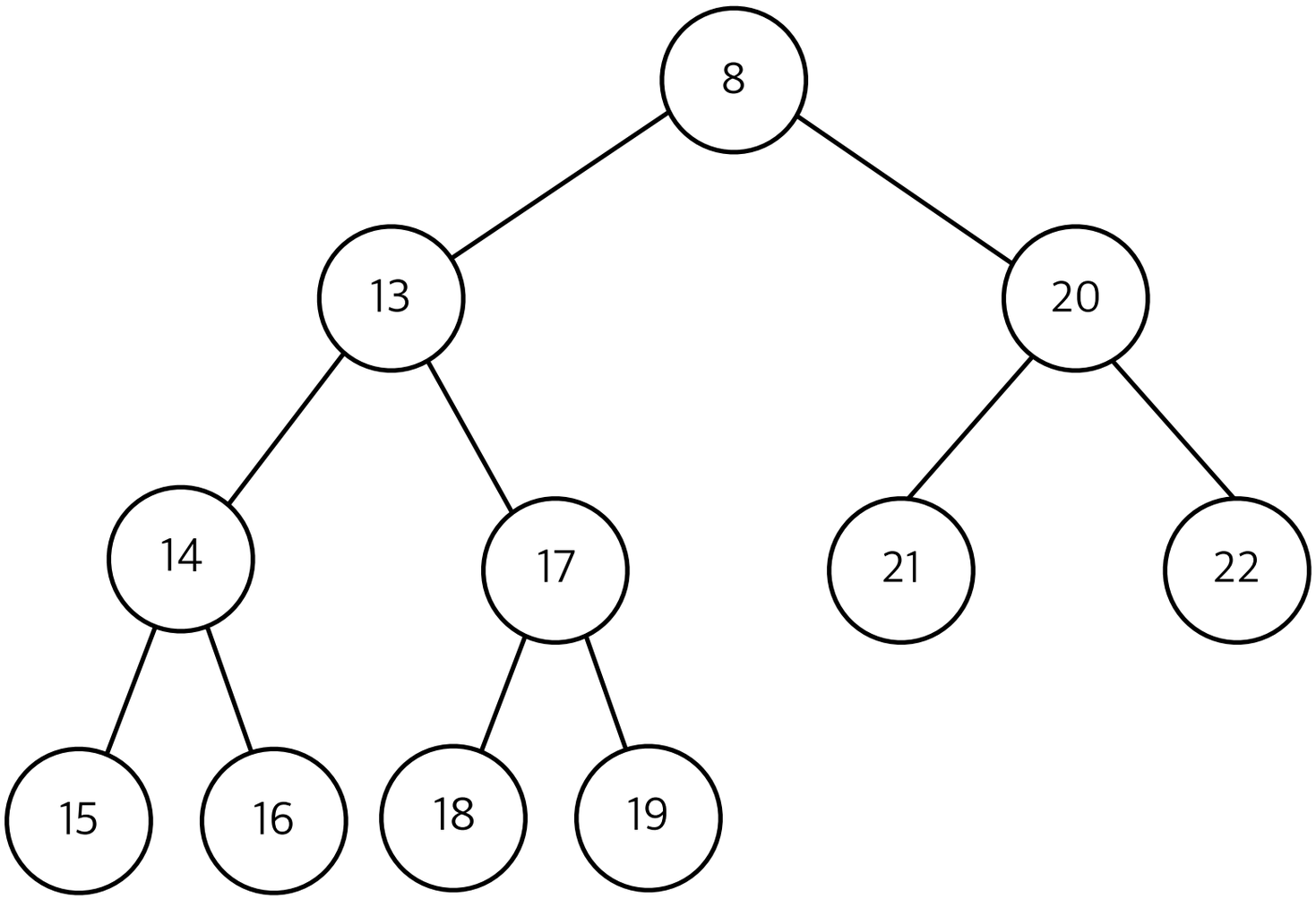
Pop()

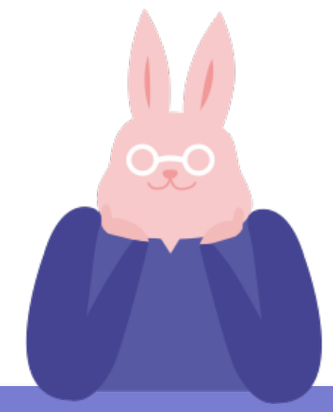




트리 구조의 이용 : 최소힙의 Pop()

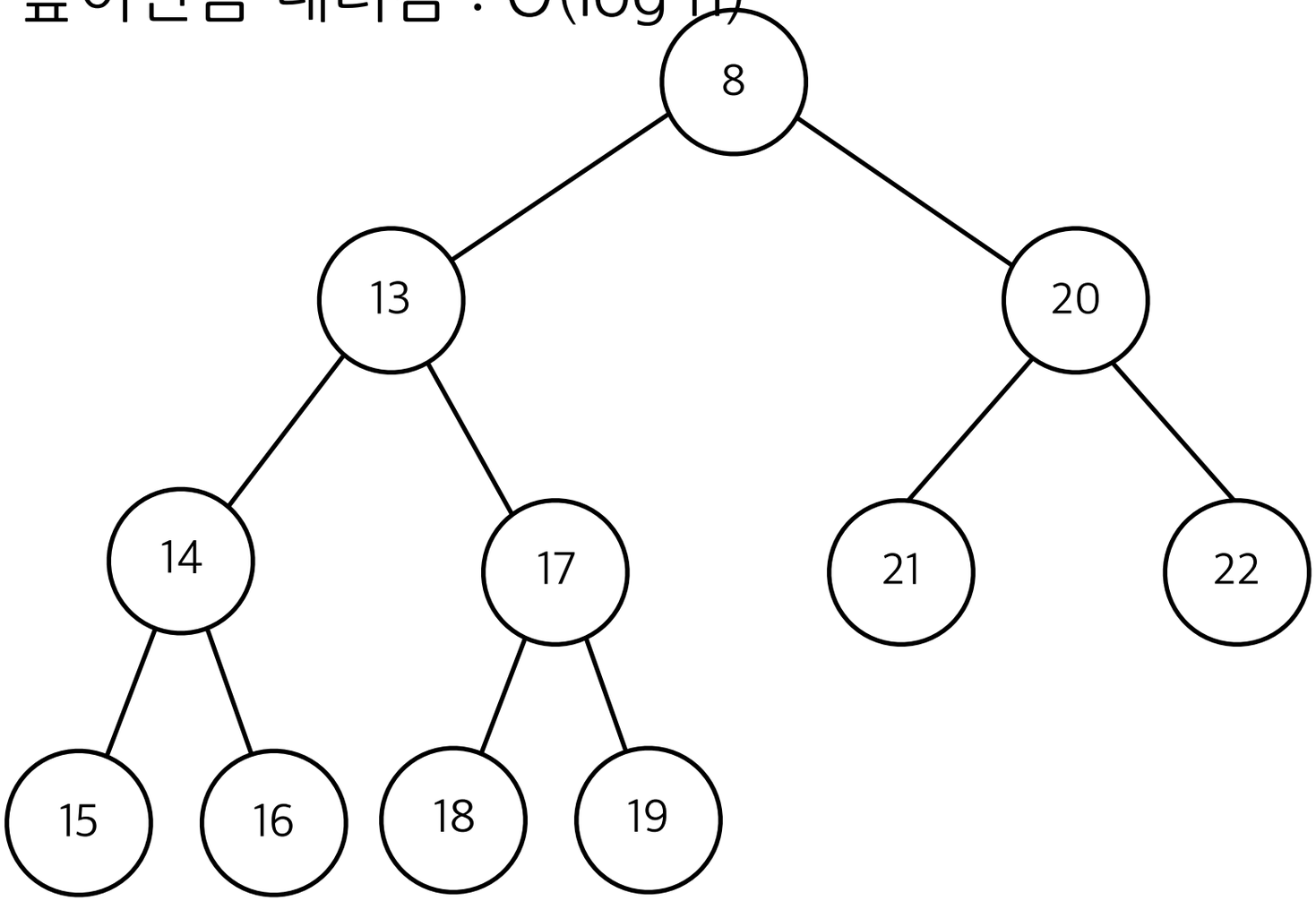
- 힙의 마지막 자리에 있는 Node를 Root로 올린다

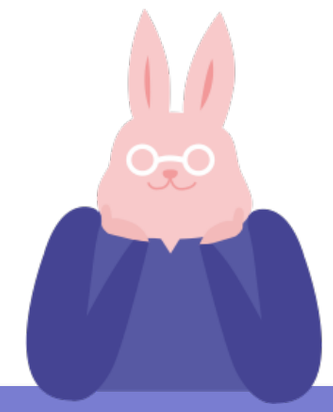




트리 구조의 이용 : 최소힙의 Pop()

- 힙의 마지막 자리에 있는 Node를 Root로 올린다
  - 트리의 높이만큼 내려감 : O(log n)

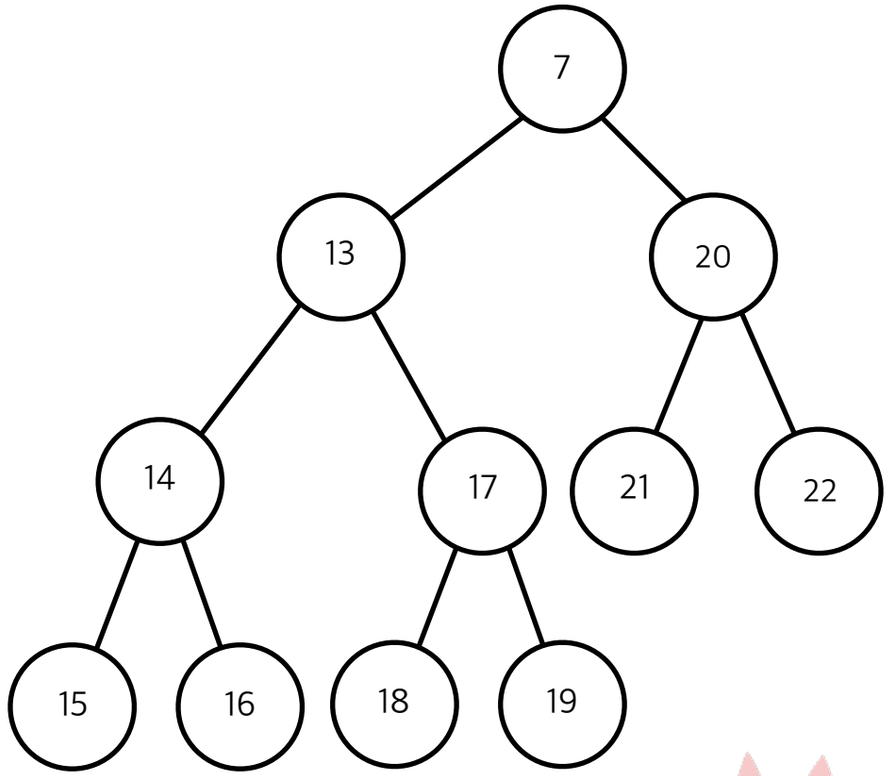


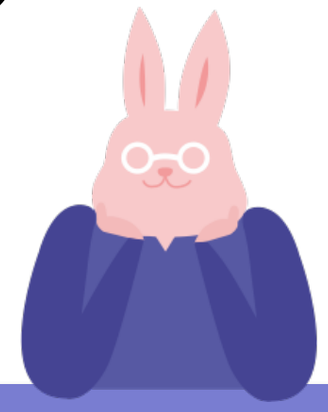


트리 구조의 이용 : 최소힙 요약

- 부모노드의 값이 자식노드의 값보다 작은 완전 이진 트리

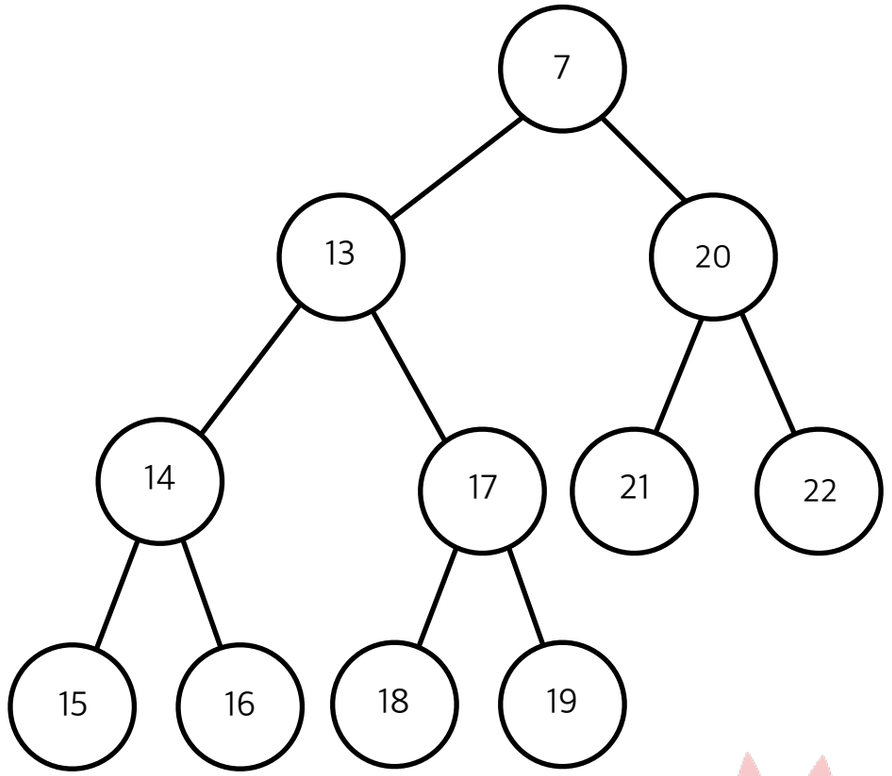
- 지원하는 연산
  - Top() : 최솟값을 반환한다, O(1)
  - Push(x) : 값 x를 힙에 넣는다, O(log n)
  - Pop() : 최솟값을 힙에서 제거한다, O(log n)

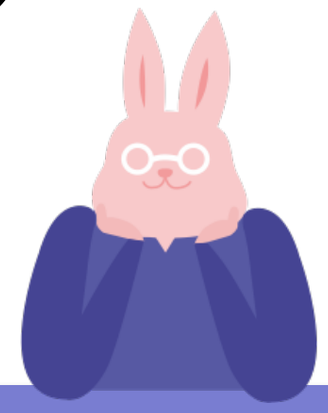




트리 구조의 이용 : 힙 구현

- Heap은 1차원 array만으로 구현해도 충분하다!
  - Tree[i] = Node i가 담고있는 값
  - Tree[i*2] = Node i의 왼쪽 자식
  - Tree[i*2+1] = Node i의 오른쪽 자식





[활동문제 3] 힙정렬 구현하기

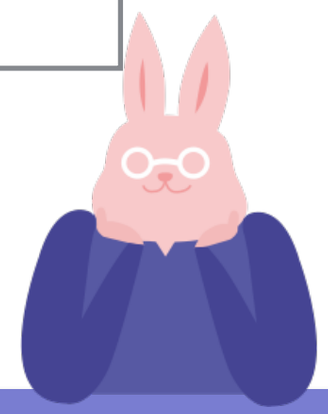
- Heap을 이용하여 정렬을 구현

입력의 예

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

출력의 예

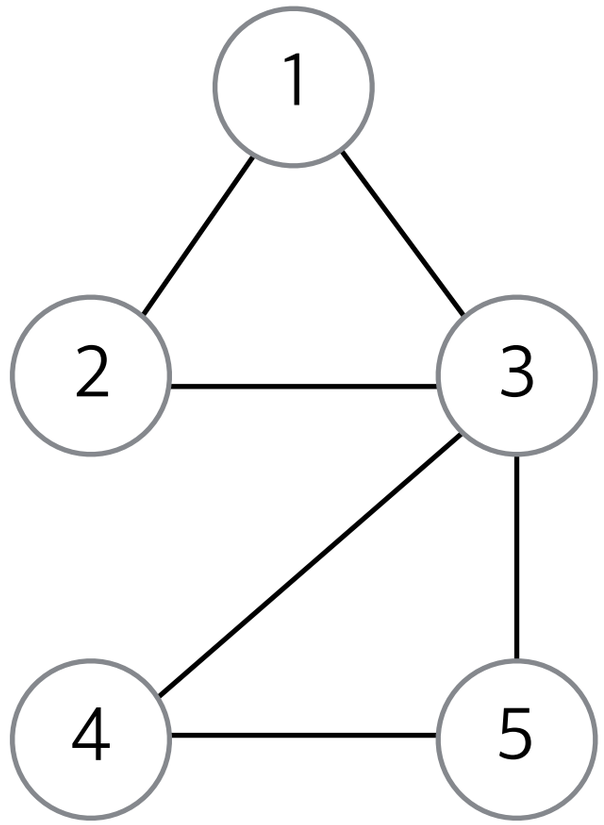
```
1 2 3 4 5
```



그래프 (Remind)

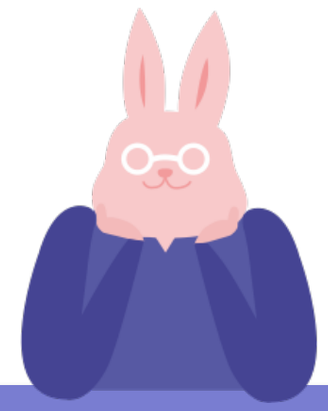
- Node와 Edge로 이루어져 있는 자료구조
  - Node와 Edge에 정보를 저장한다

- 용어
  - Path : 두 노드 사이의 길
  - Cycle : 자기 자신으로 돌아오는 길



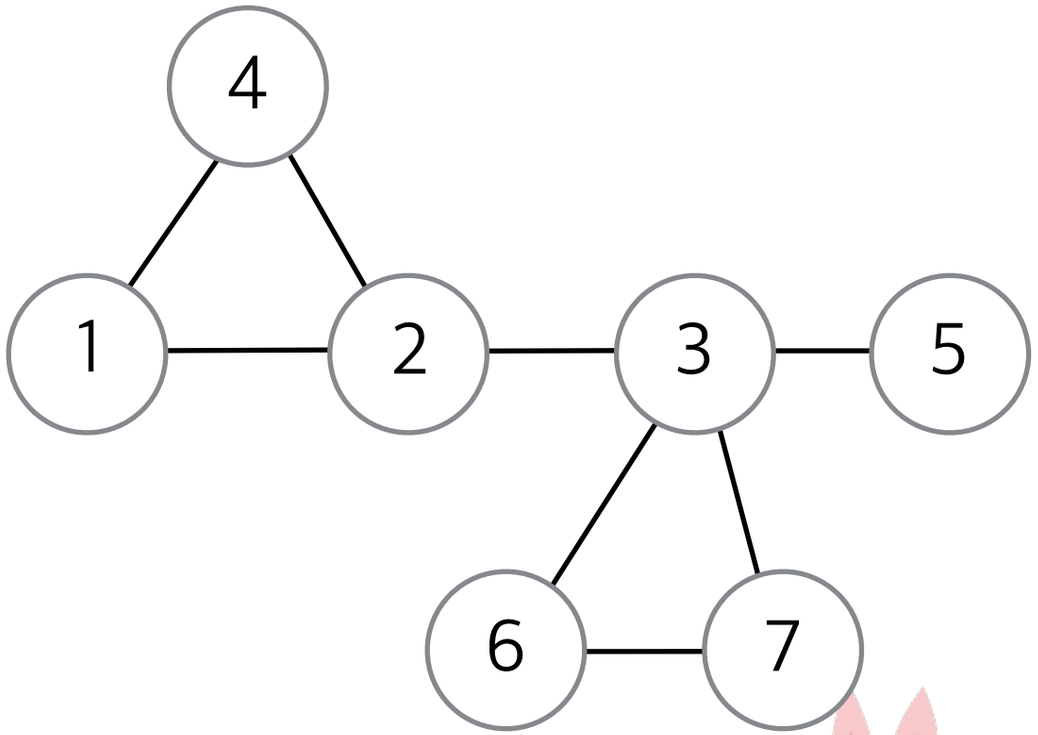
그래프 순회

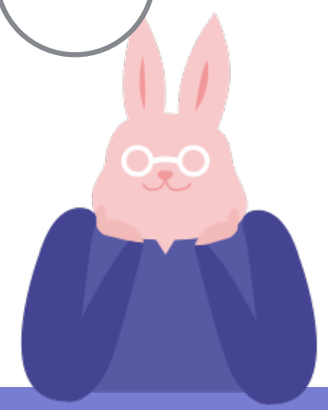
- 깊이 우선 탐색 (DFS)
- 너비 우선 탐색 (BFS)



깊이 우선 탐색 (DFS)

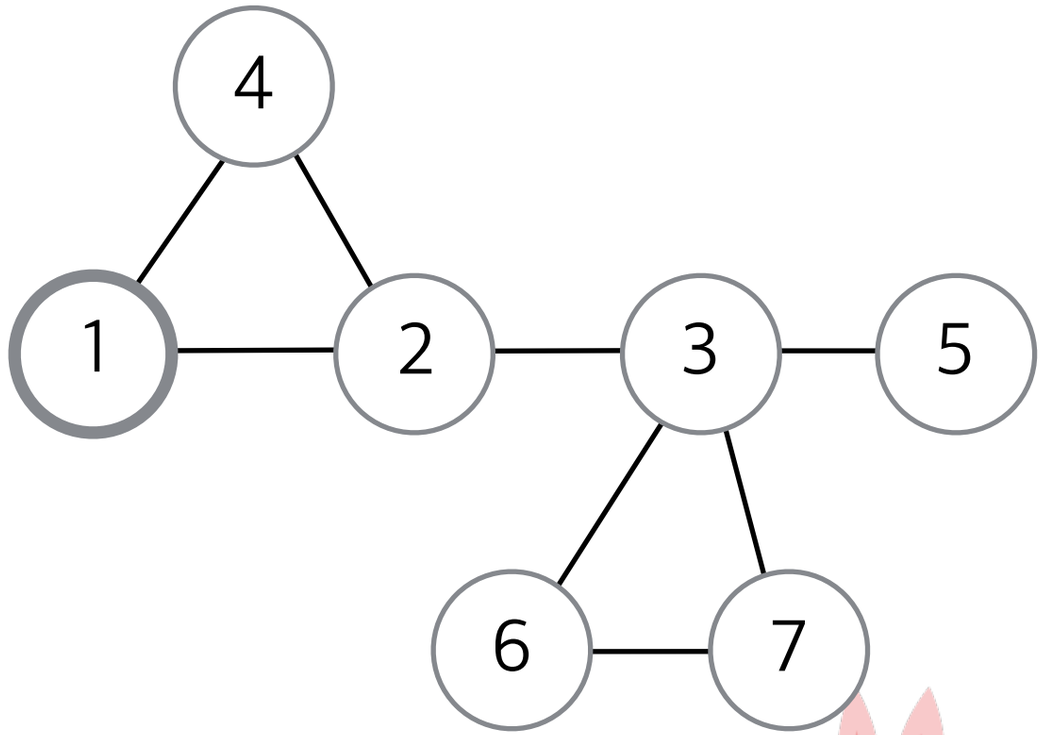
- 인접한 Node 중에서 방문하지 않은 Node로 간다





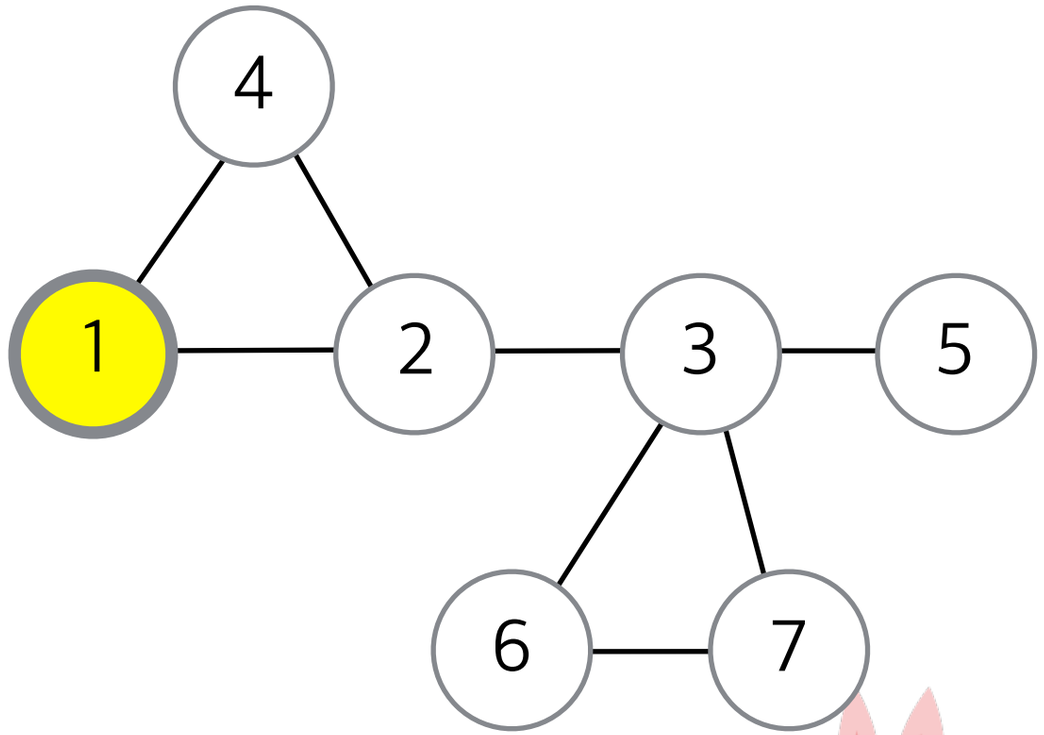
깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



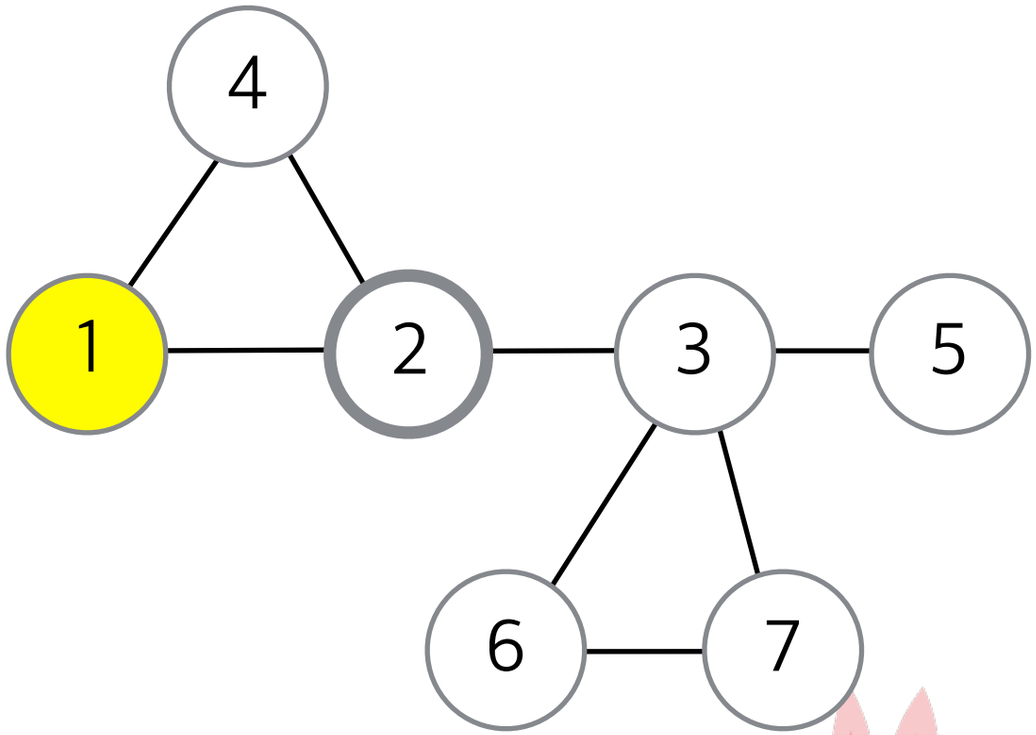
깊이 우선 탐색 (DFS)

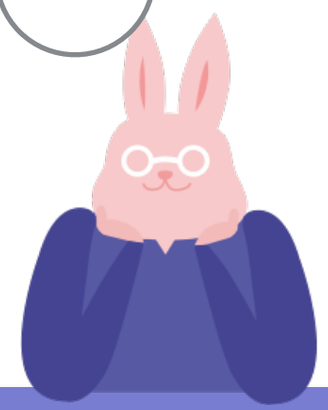
- 인접한 Node 중에서 방문하지 않은 Node로 간다



깊이 우선 탐색 (DFS)

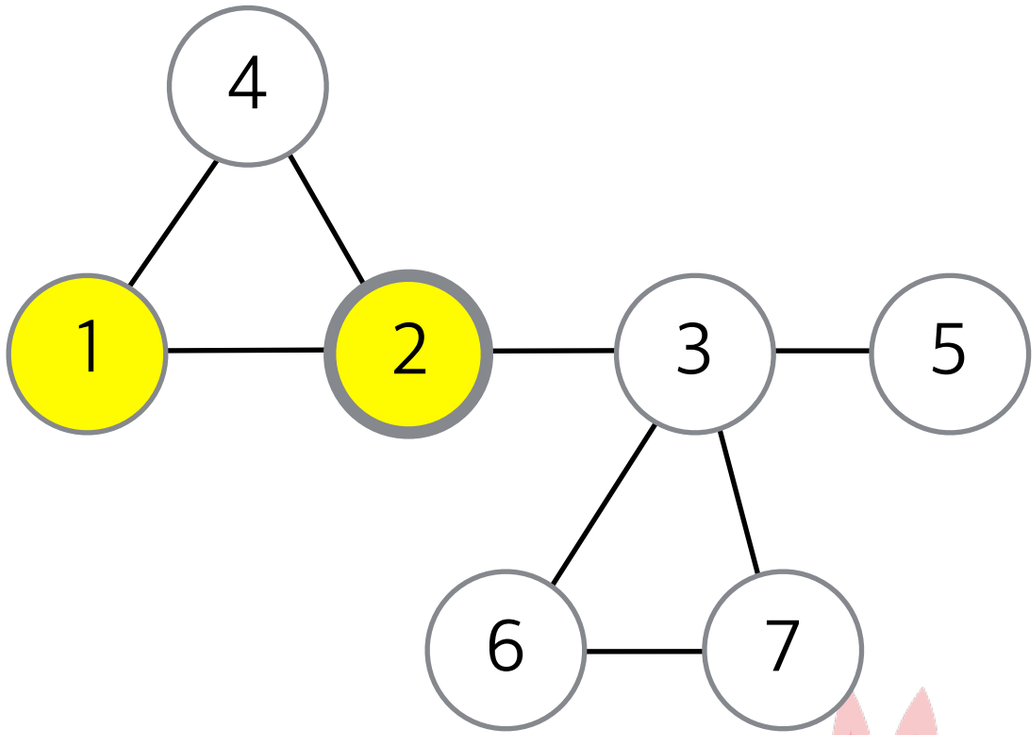
- 인접한 Node 중에서 방문하지 않은 Node로 간다

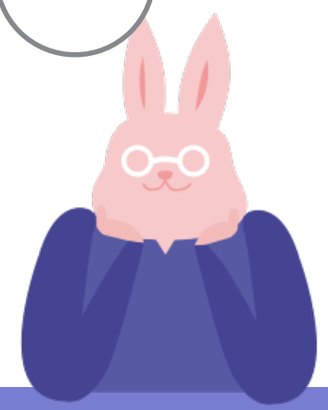




깊이 우선 탐색 (DFS)

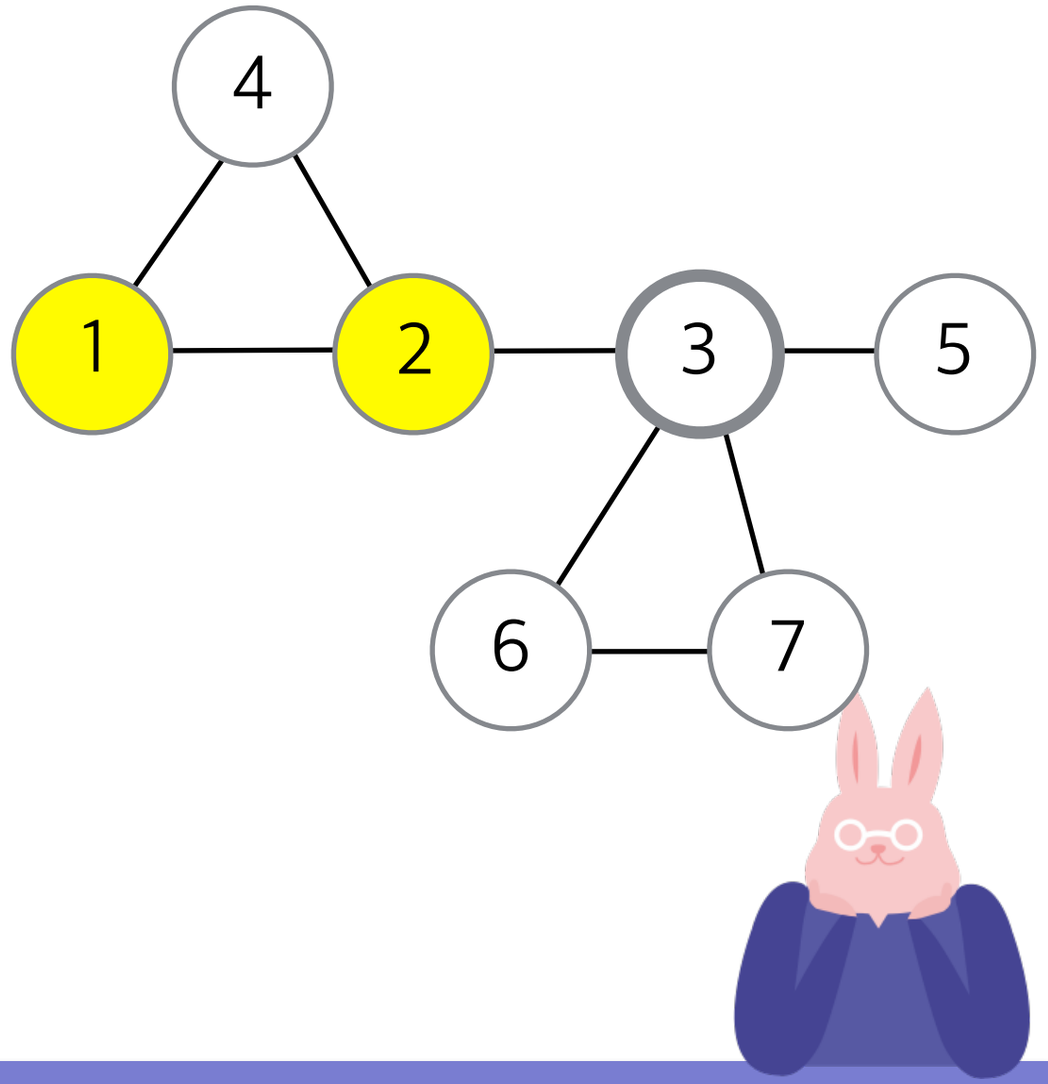
- 인접한 Node 중에서 방문하지 않은 Node로 간다





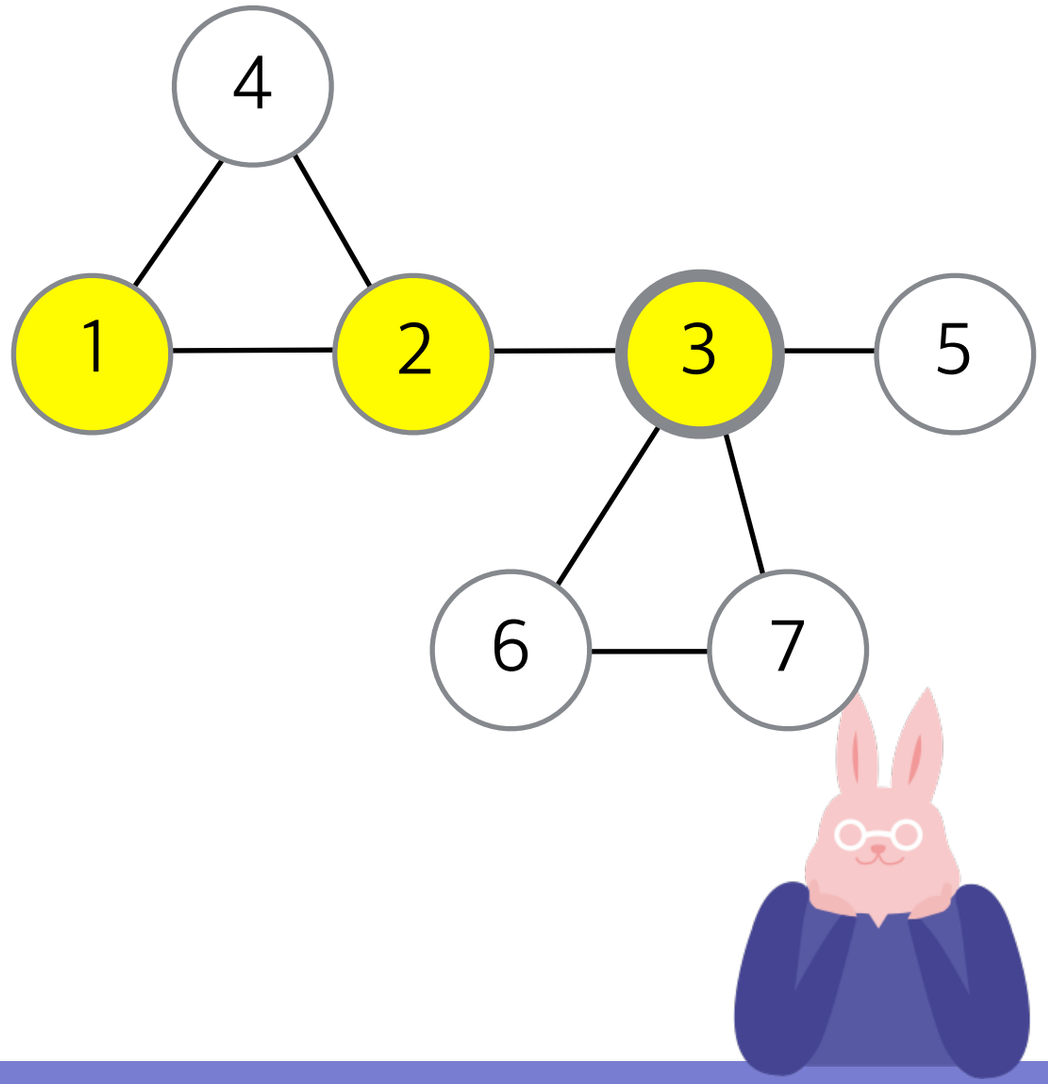
깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



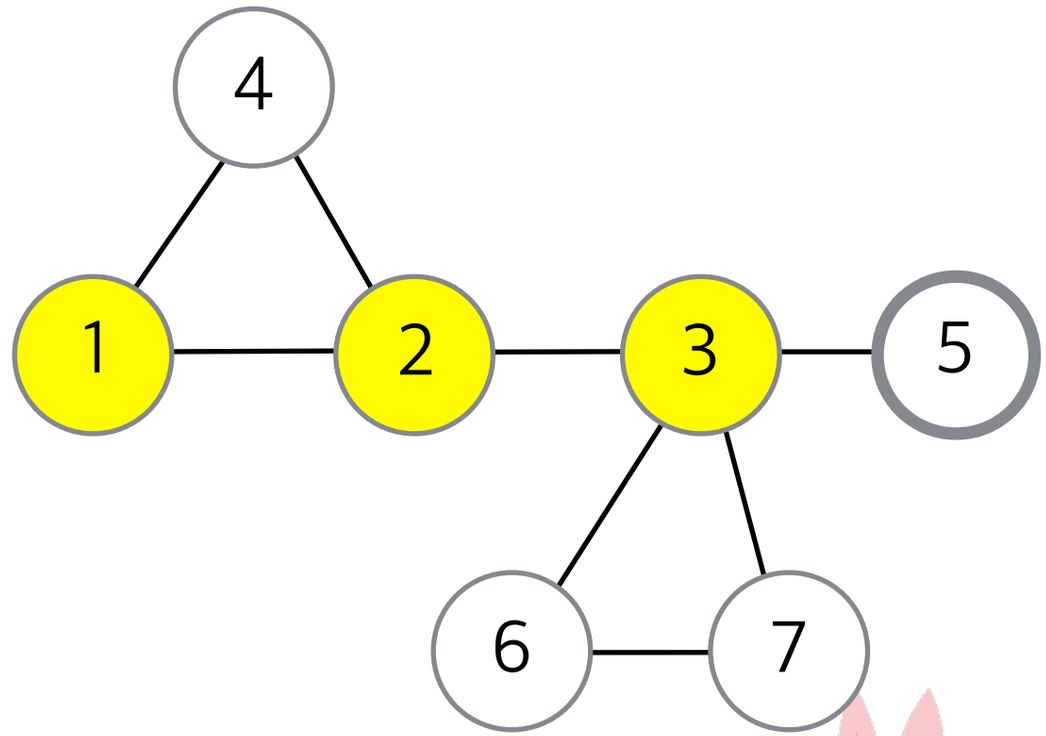
깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



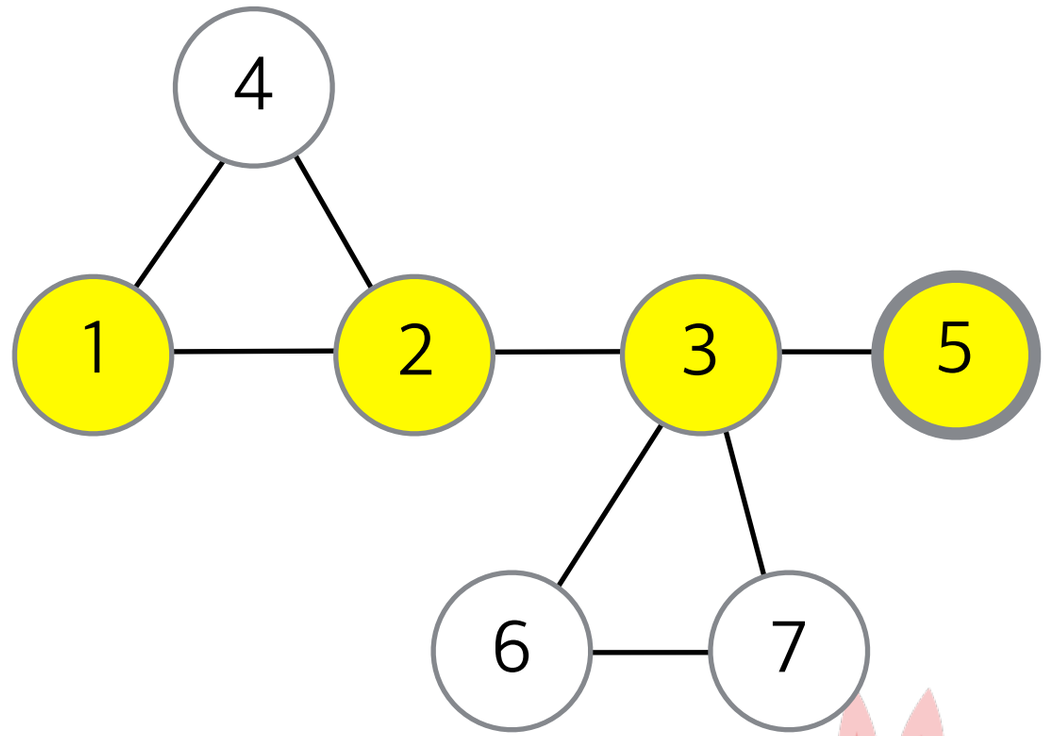
깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



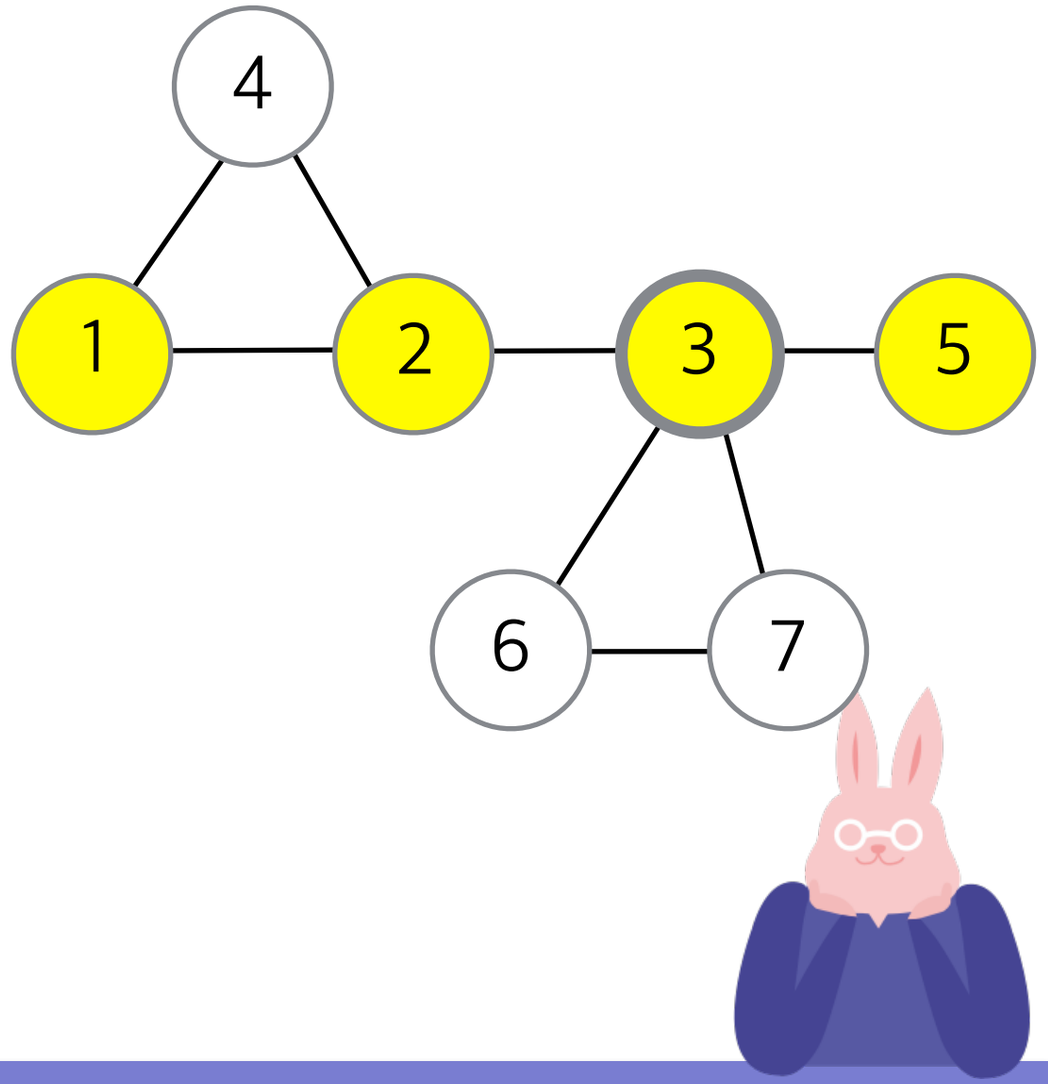
깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



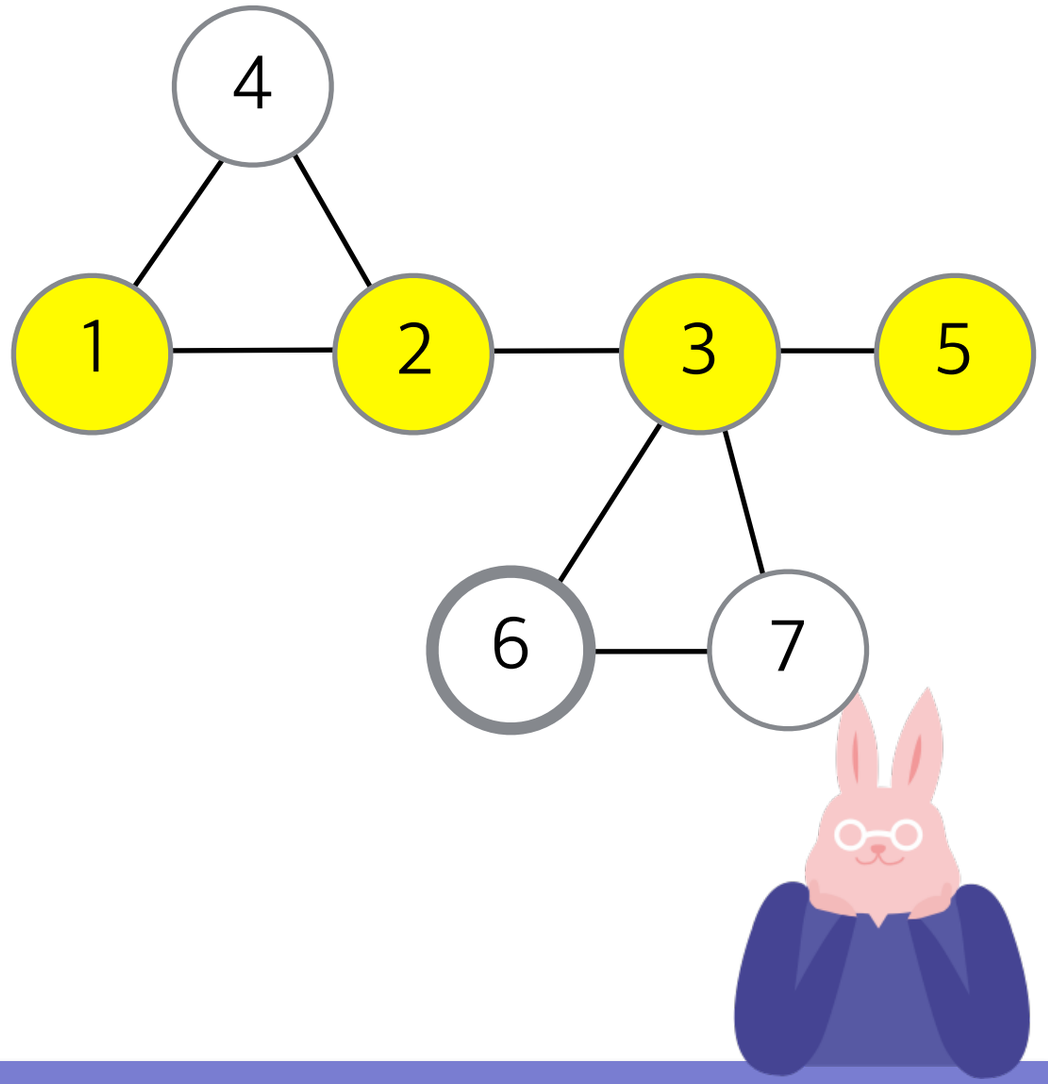
깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



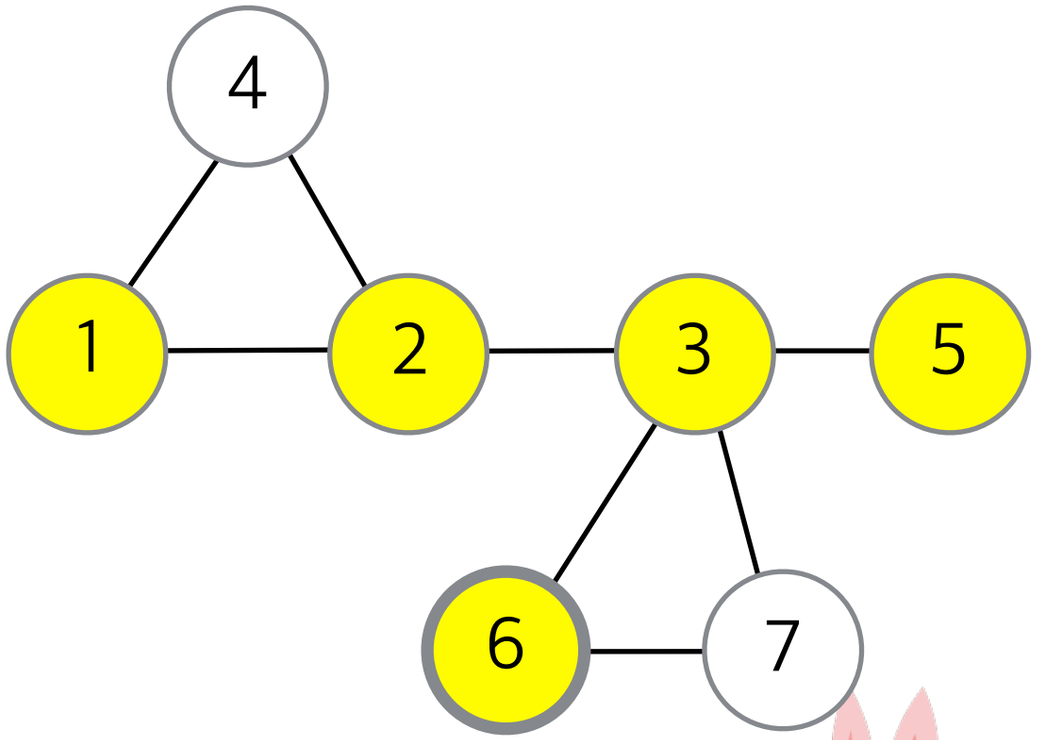
깊이 우선 탐색 (DFS)

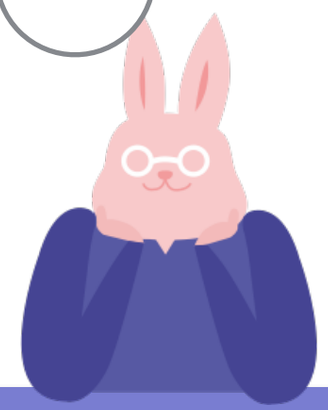
- 인접한 Node 중에서 방문하지 않은 Node로 간다



깊이 우선 탐색 (DFS)

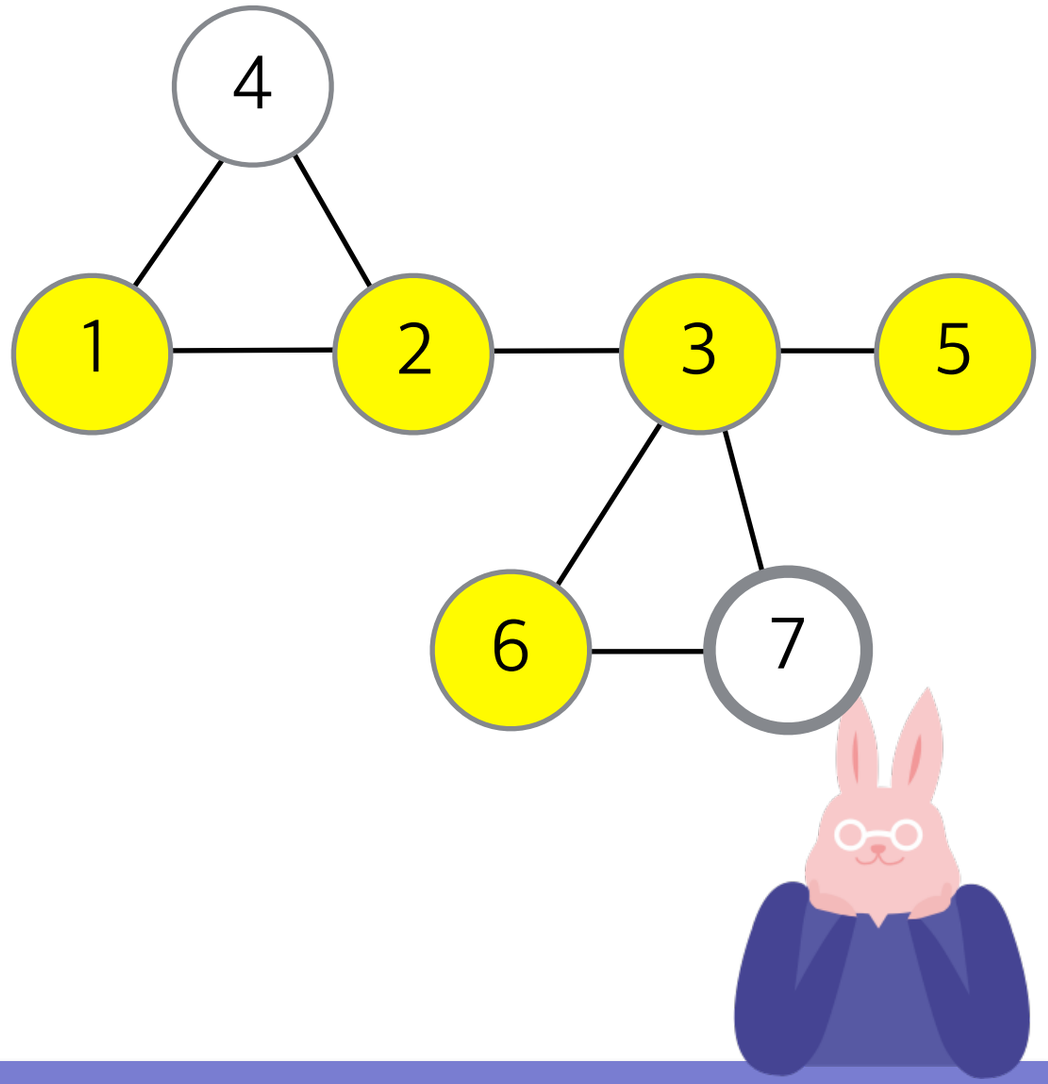
- 인접한 Node 중에서 방문하지 않은 Node로 간다





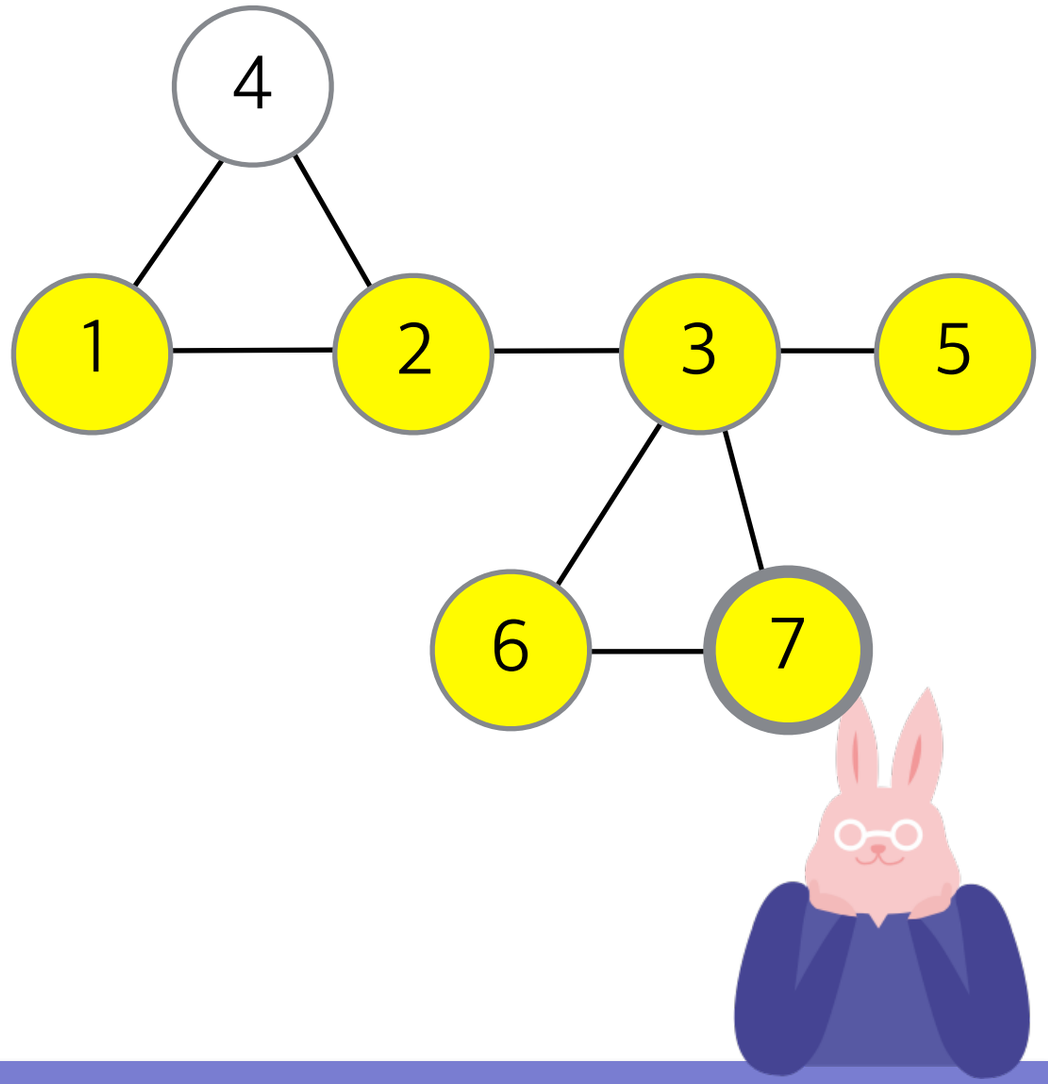
깊이 우선 탐색 (DFS)

- 인접한 Node 중에서 방문하지 않은 Node로 간다



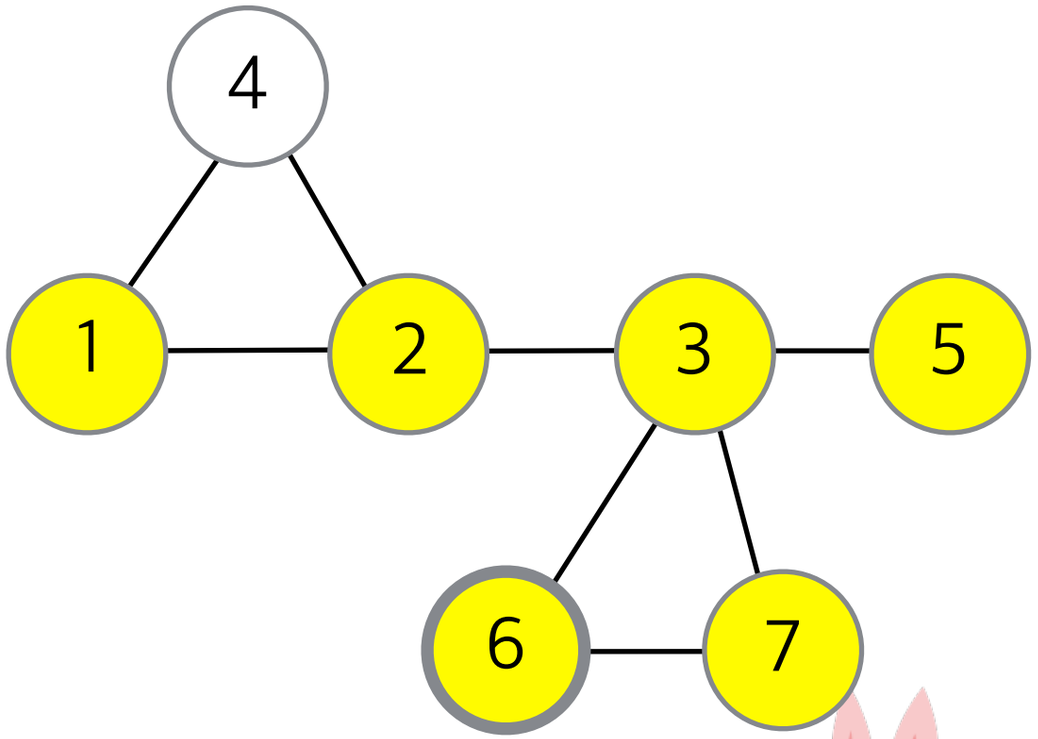
깊이 우선 탐색 (DFS)

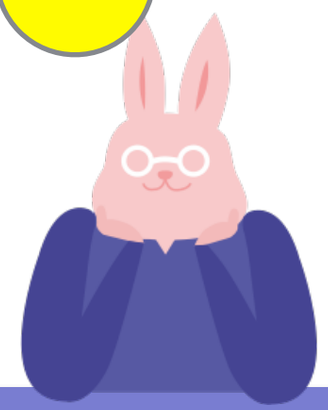
- 인접한 Node 중에서 방문하지 않은 Node로 간다



깊이 우선 탐색 (DFS)

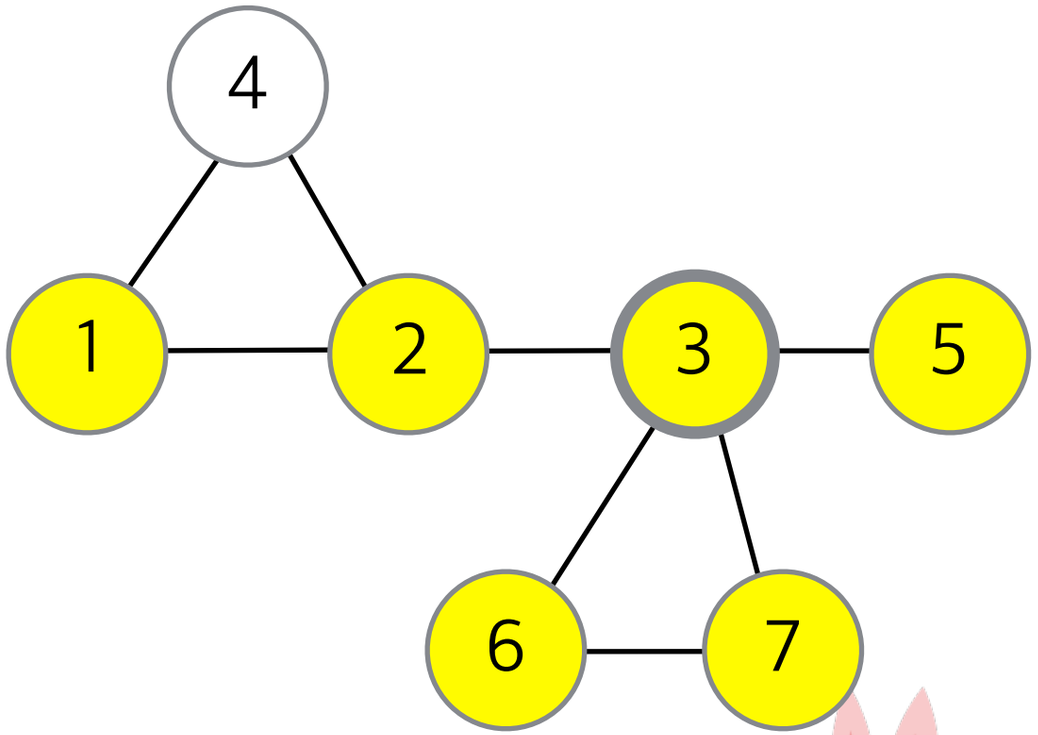
- 인접한 Node 중에서 방문하지 않은 Node로 간다





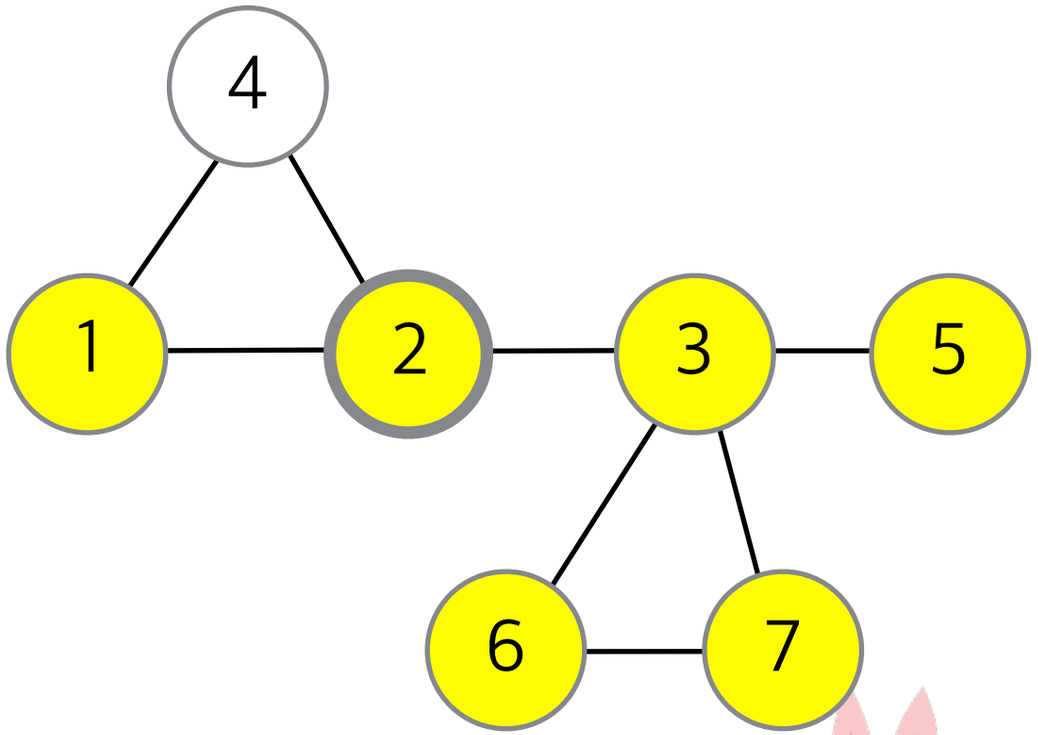
깊이 우선 탐색 (DFS)

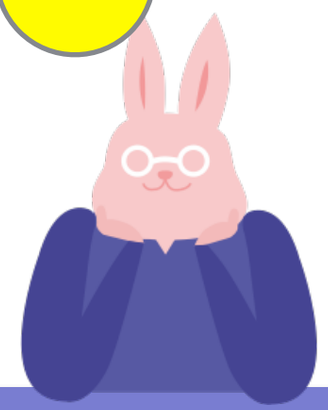
- 인접한 Node 중에서 방문하지 않은 Node로 간다



깊이 우선 탐색 (DFS)

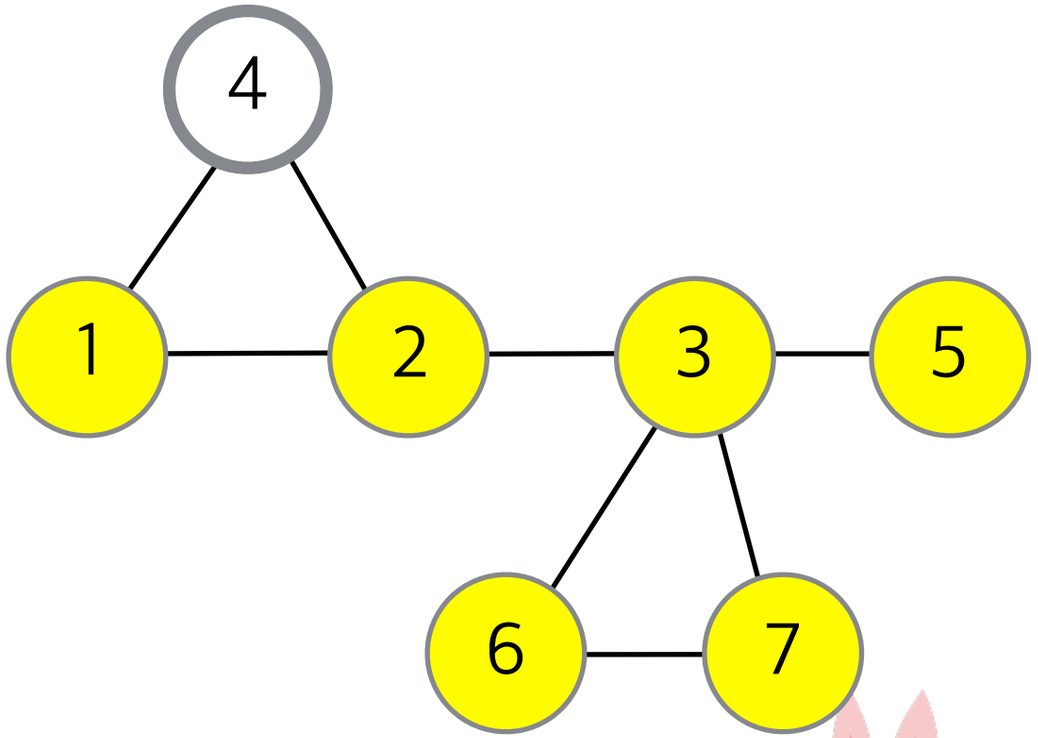
- 인접한 Node 중에서 방문하지 않은 Node로 간다

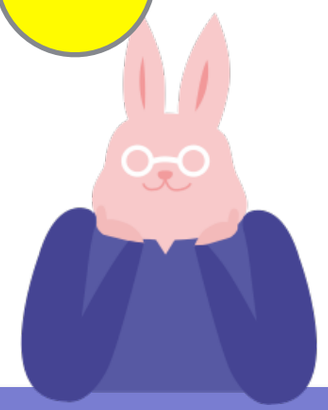




깊이 우선 탐색 (DFS)

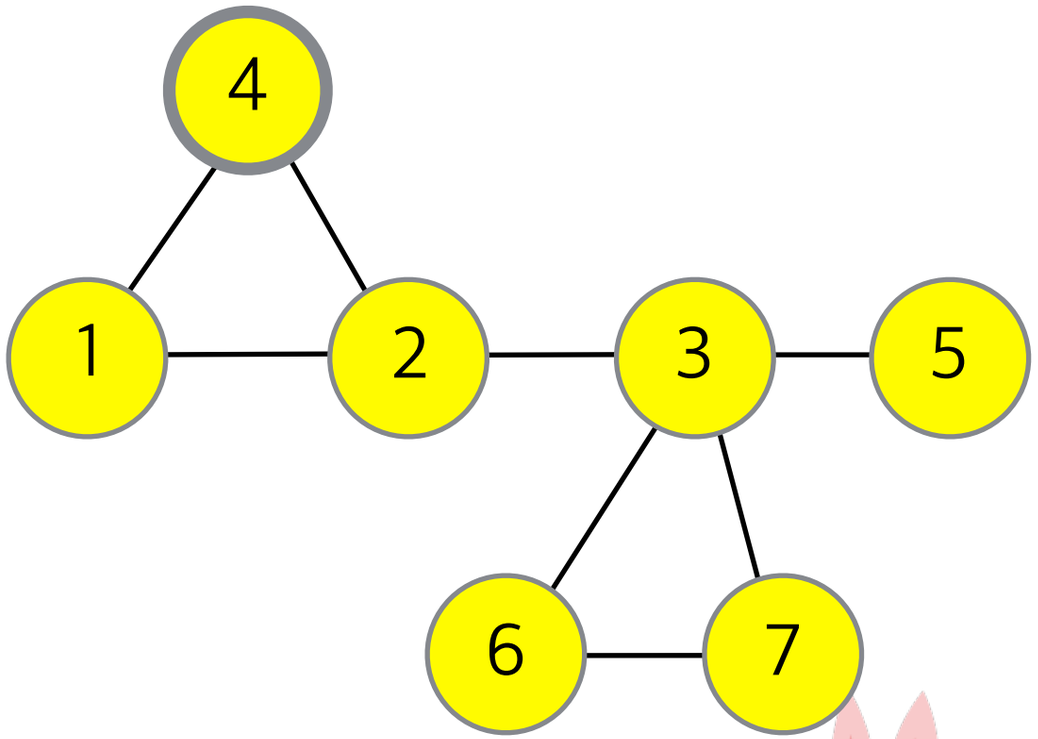
- 인접한 Node 중에서 방문하지 않은 Node로 간다

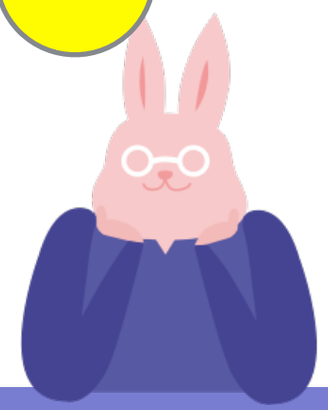




깊이 우선 탐색 (DFS)

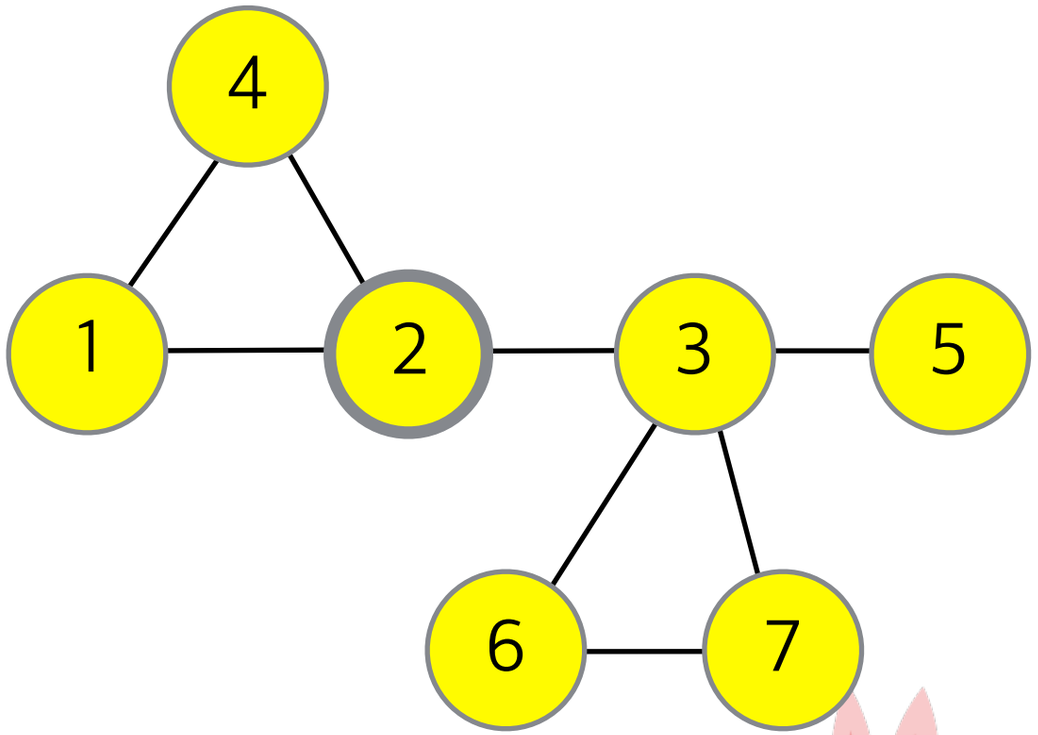
- 인접한 Node 중에서 방문하지 않은 Node로 간다

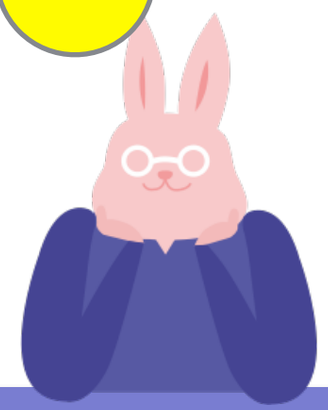




깊이 우선 탐색 (DFS)

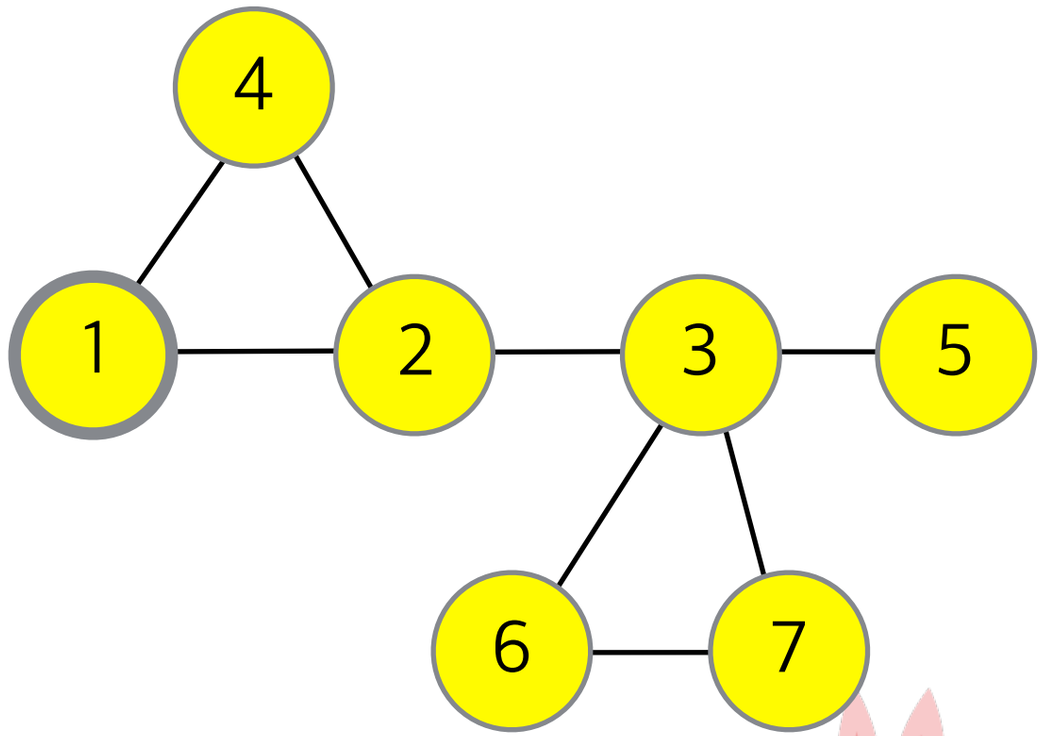
- 인접한 Node 중에서 방문하지 않은 Node로 간다

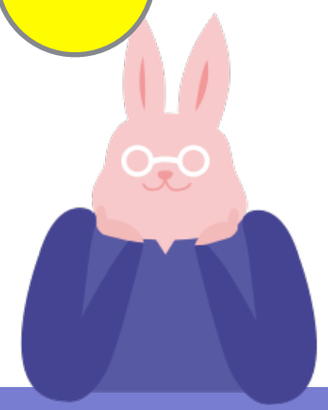




깊이 우선 탐색 (DFS)

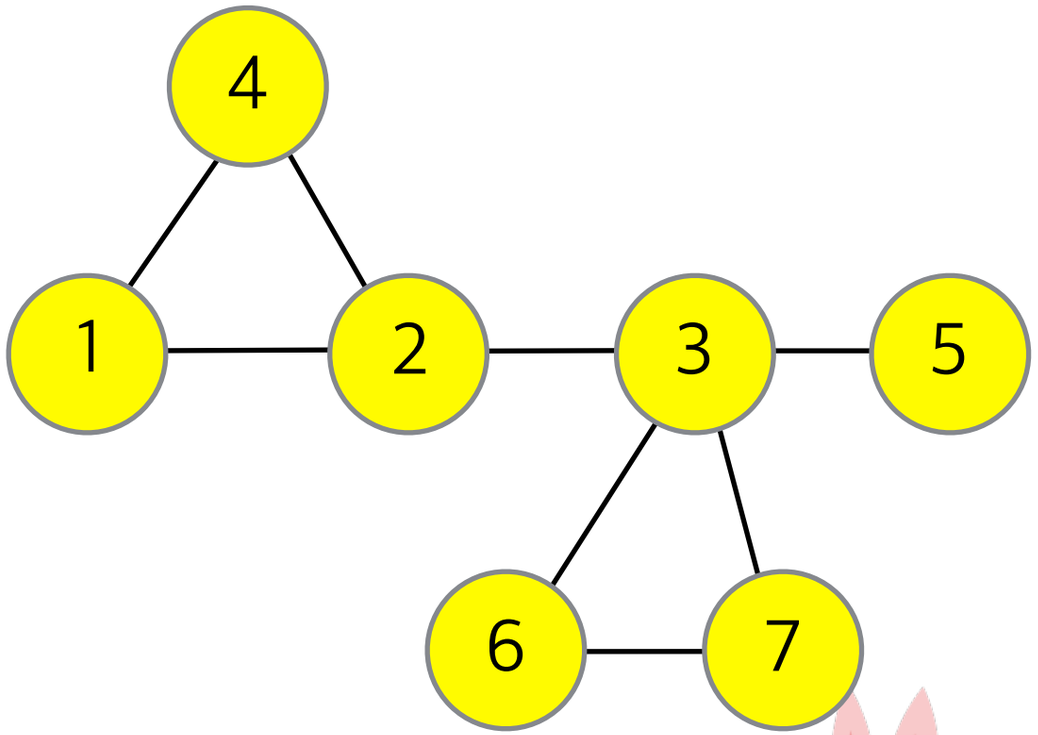
- 인접한 Node 중에서 방문하지 않은 Node로 간다

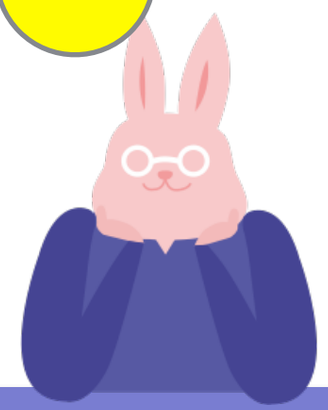




깊이 우선 탐색 (DFS)

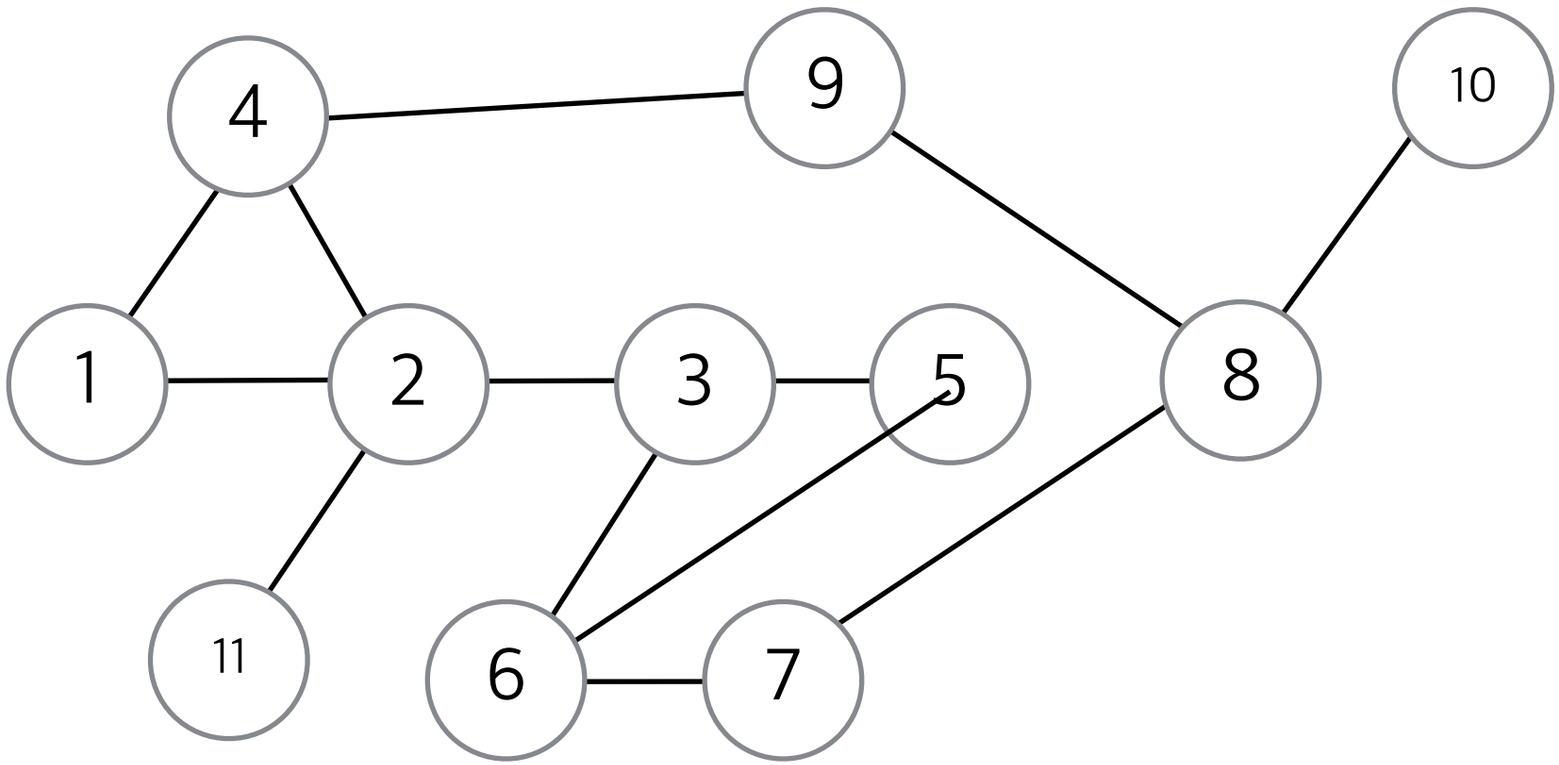
- 인접한 Node 중에서 방문하지 않은 Node로 간다



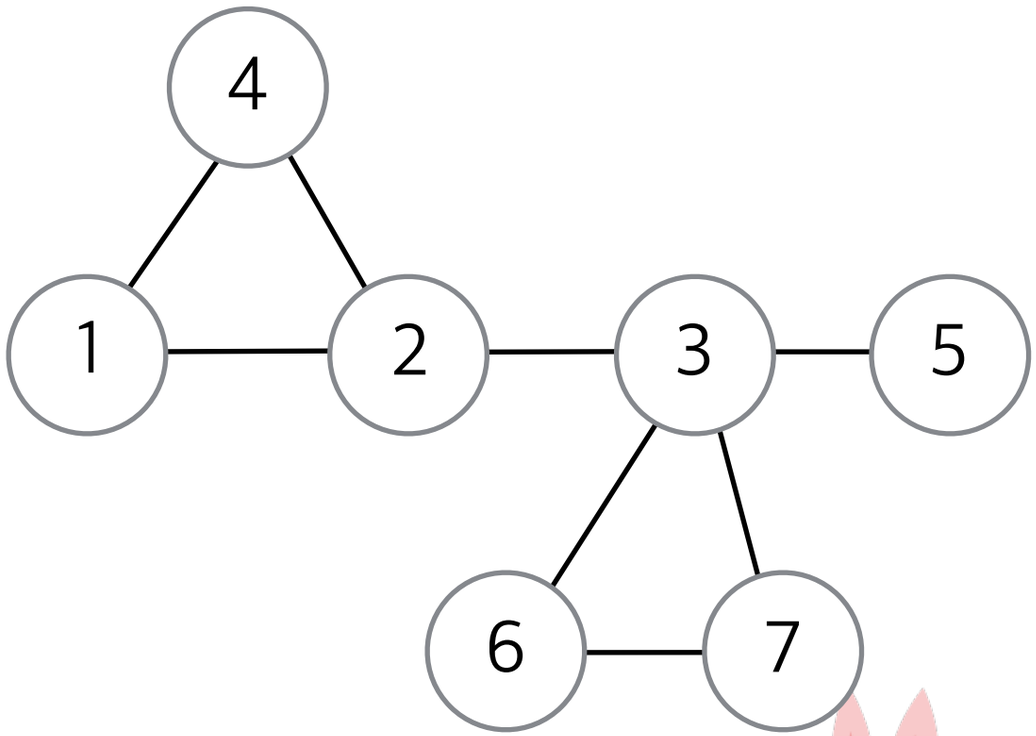


깊이 우선 탐색 (DFS)

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

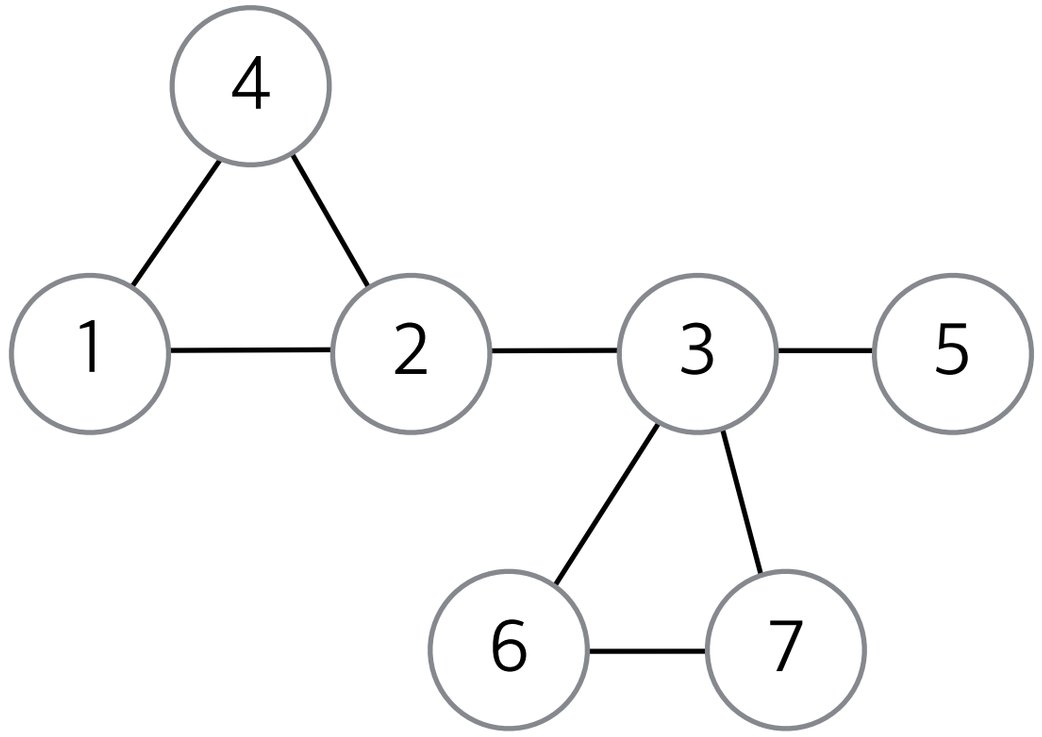


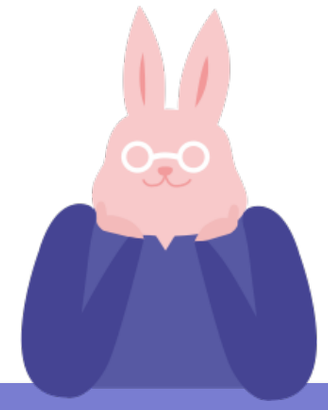
깊이 우선 탐색 (DFS)



깊이 우선 탐색 (DFS) 구현

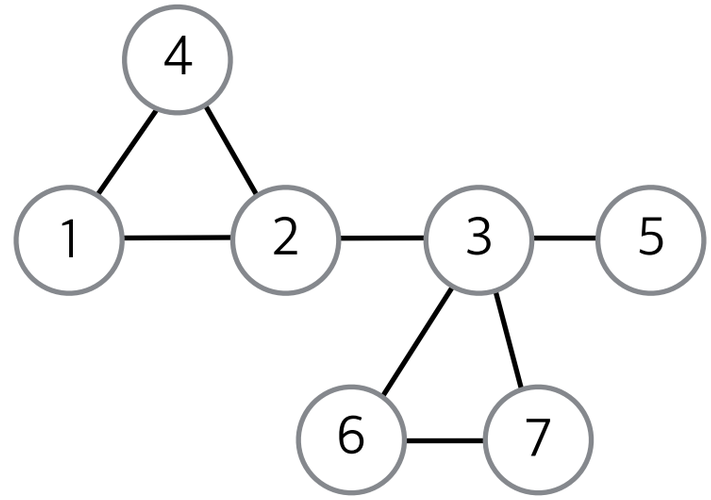
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

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

    return result
```





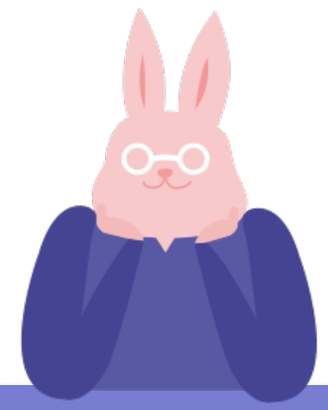
깊이 우선 탐색 (DFS) 구현

```
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, 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]

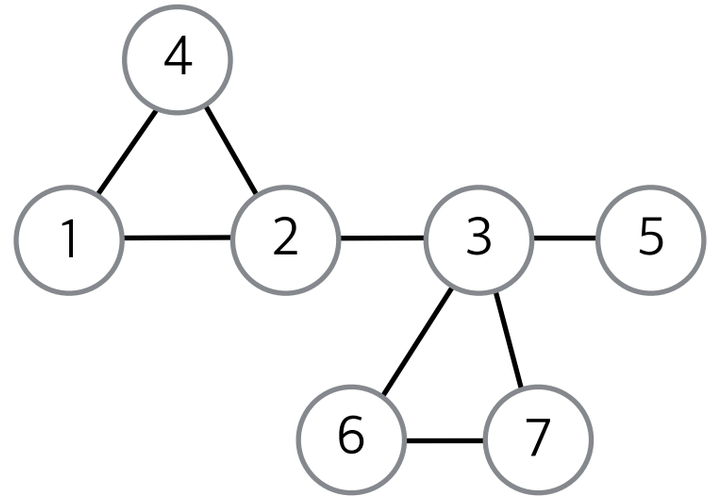


깊이 우선 탐색 (DFS) 구현

```
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
```

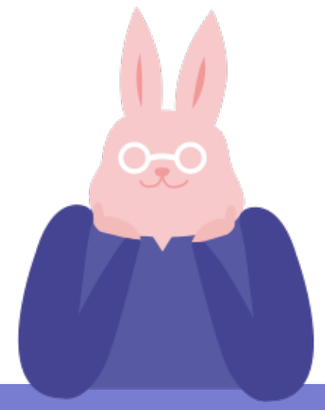
DFS(graph, 1, visited)

result = null



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

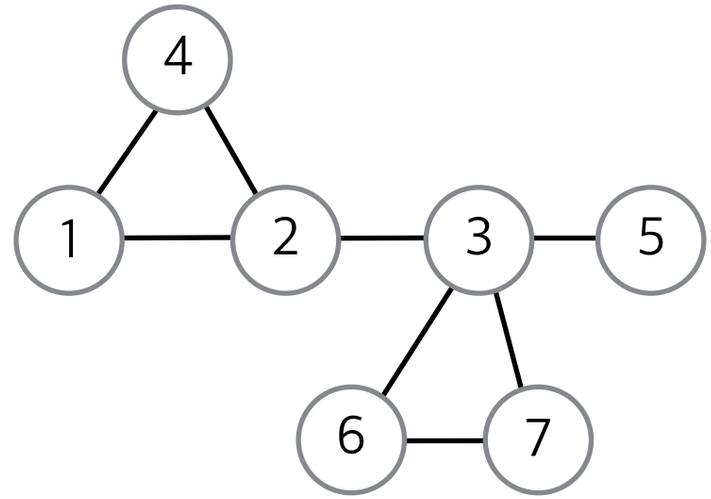


깊이 우선 탐색 (DFS) 구현

```
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
```

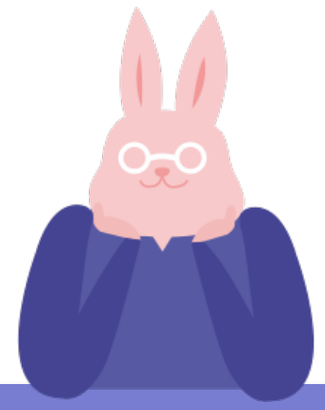
DFS(graph, 1, visited)

result = null



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

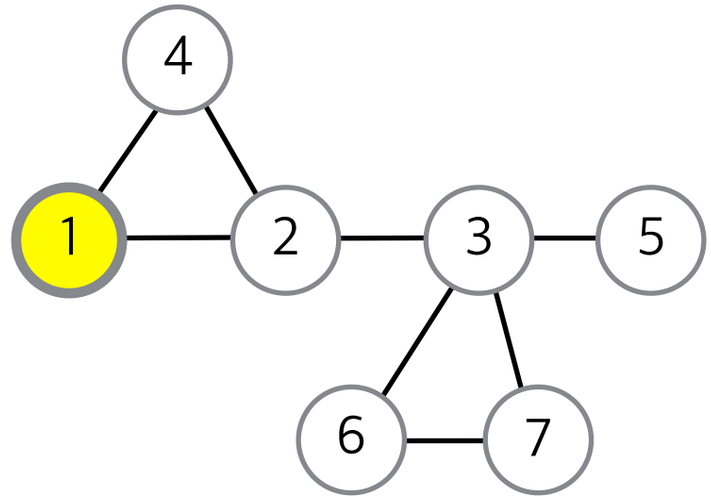


깊이 우선 탐색 (DFS) 구현

```
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
```

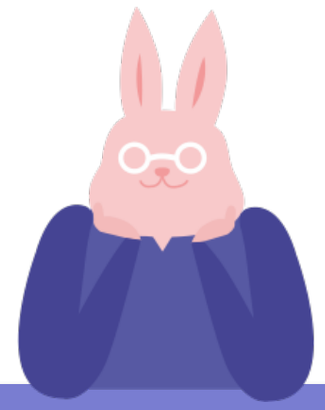
DFS(graph, 1, visited)

result = null



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

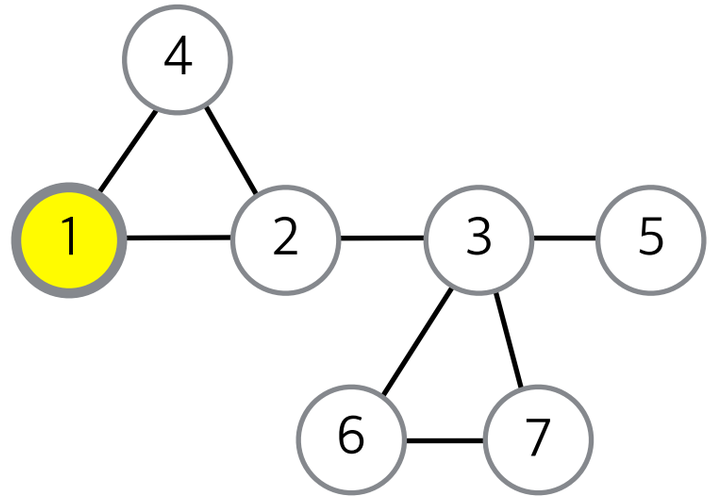


깊이 우선 탐색 (DFS) 구현

```
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
```

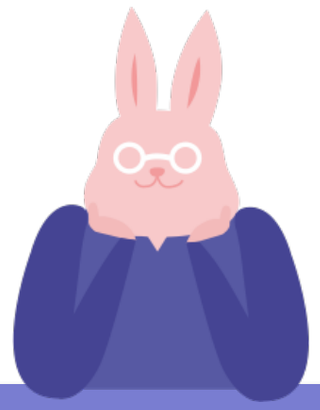
DFS(graph, 1, visited)

result = [1]



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

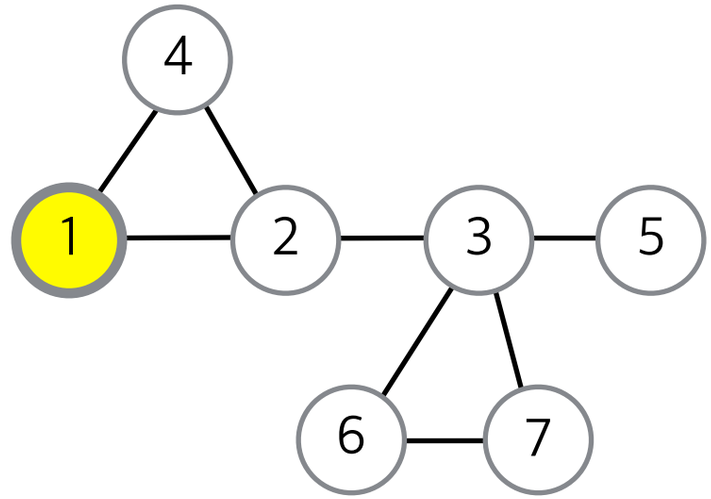


깊이 우선 탐색 (DFS) 구현

```
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
```

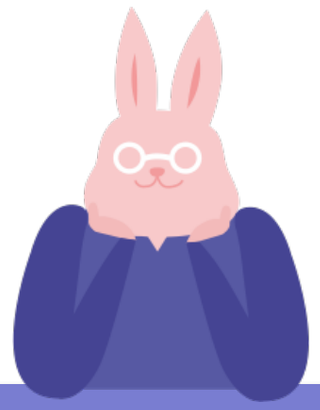
DFS(graph, 1, visited)

result = [1], v = 2



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

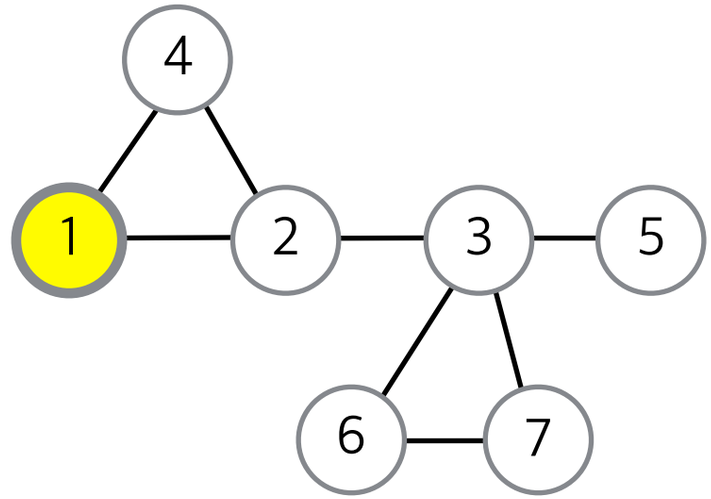


깊이 우선 탐색 (DFS) 구현

```
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
```

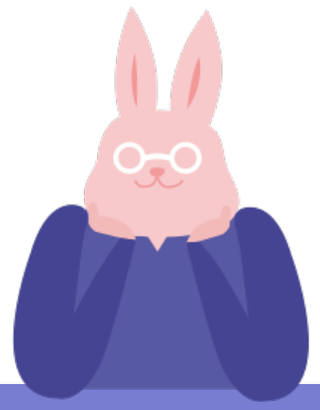
DFS(graph, 1, visited)

result = [1], v = 2



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

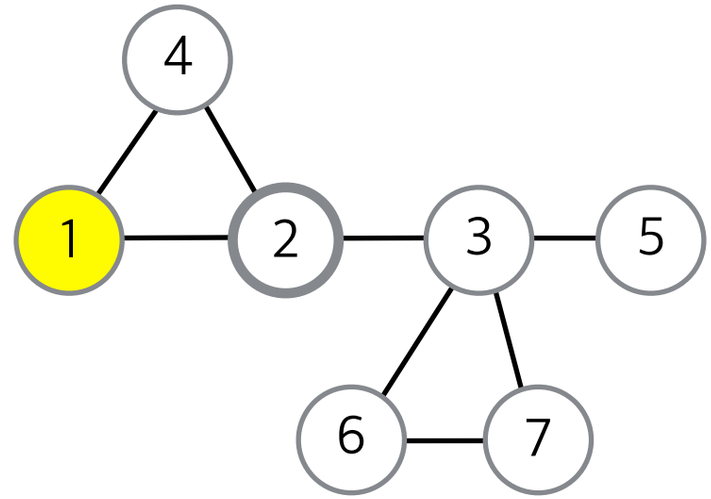
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    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

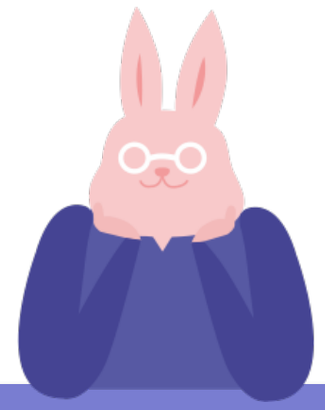
DFS(graph, 2, visited)

result = null



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

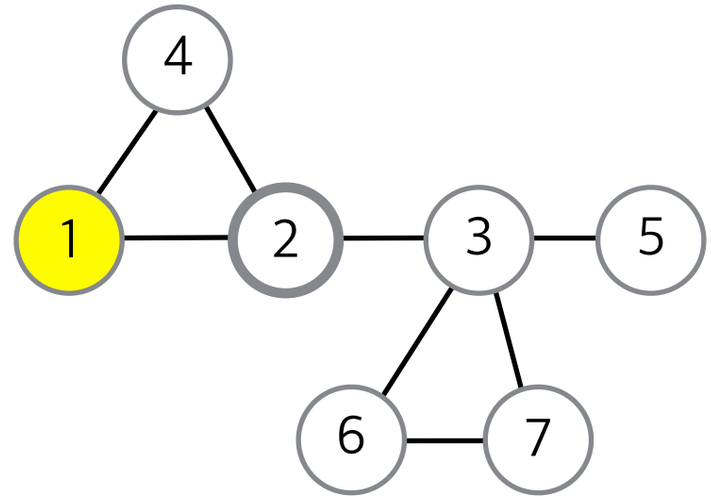
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    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

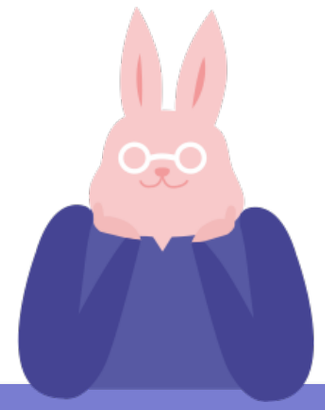
DFS(graph, 2, visited)

result = null



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

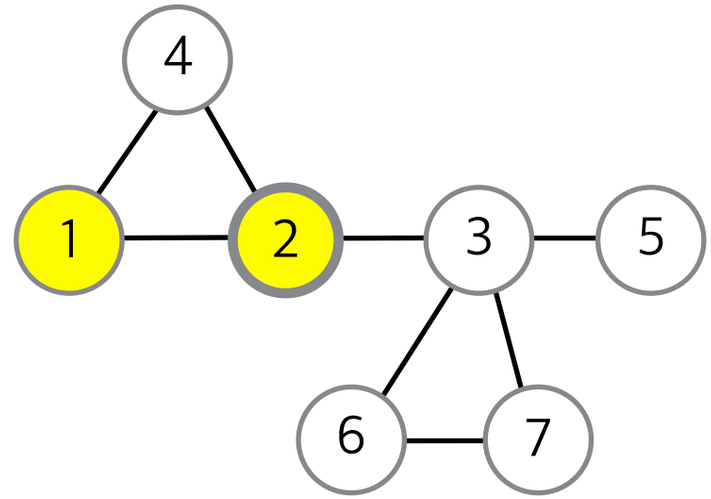
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    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

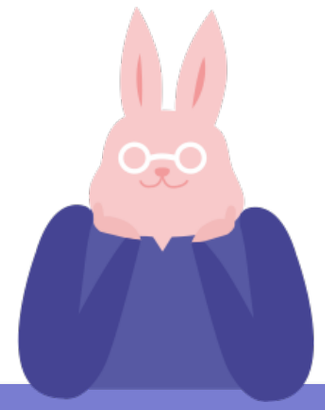
DFS(graph, 2, visited)

result = null



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

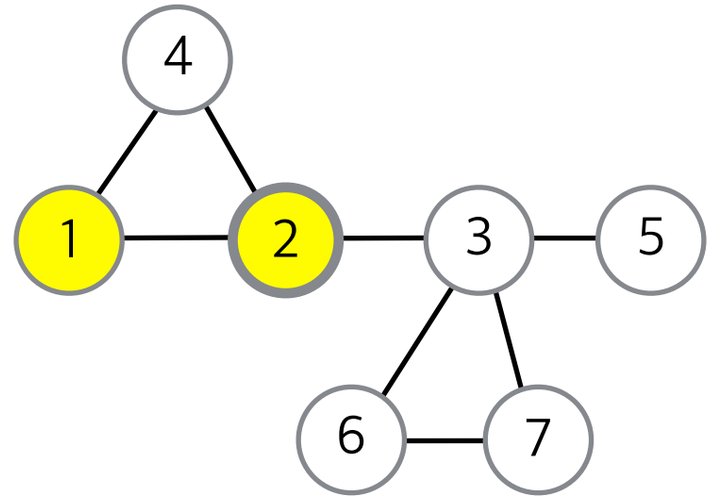
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    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

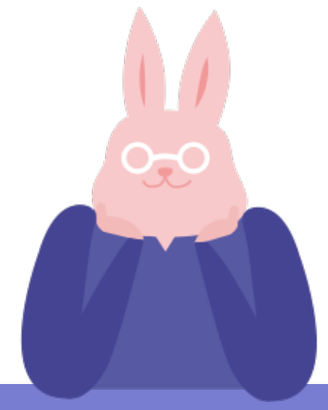
DFS(graph, 2, visited)

result = [2]



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

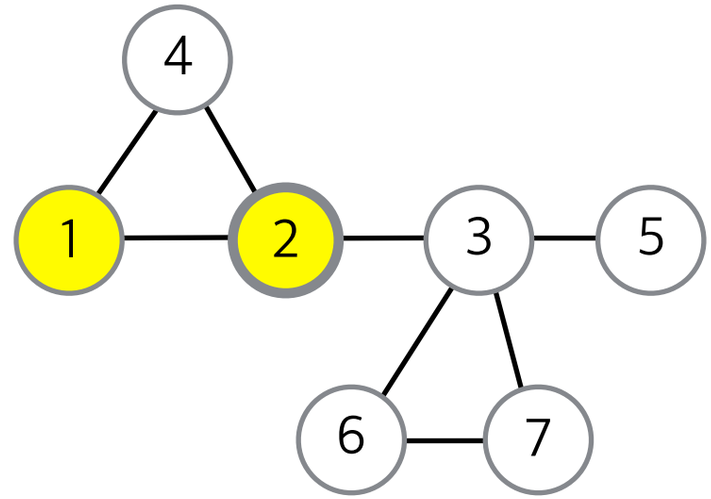
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    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

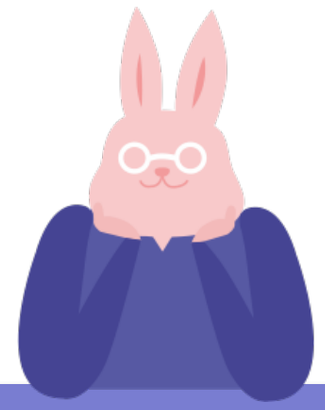
DFS(graph, 2, visited)

result = [2], v = 3



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

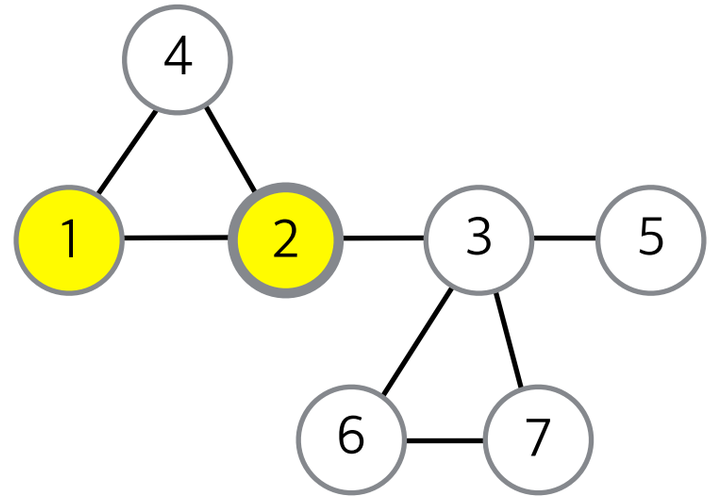
```
def DFS(graph, x, visited) :
      visited[x] = True
      result = [x]

      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

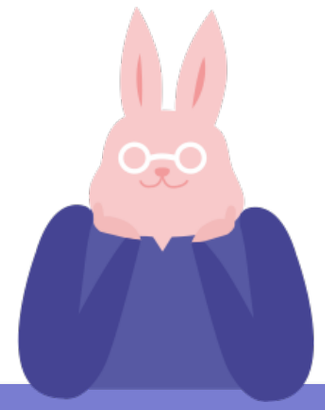
DFS(graph, 2, visited)

result = [2], v = 3



visited= [F, T, T, F, F, F, F, 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)

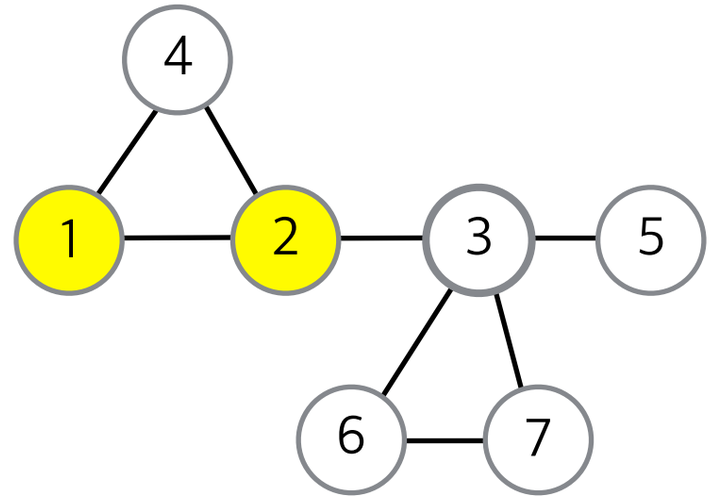
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

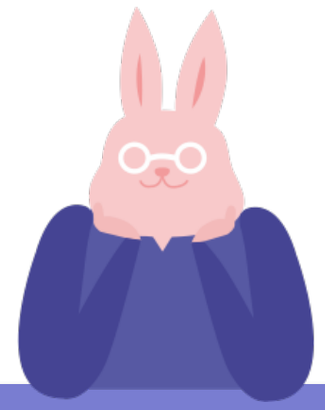
DFS(graph, 3, visited)

result = null



visited= [F, T, T, F, F, F, F, 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)

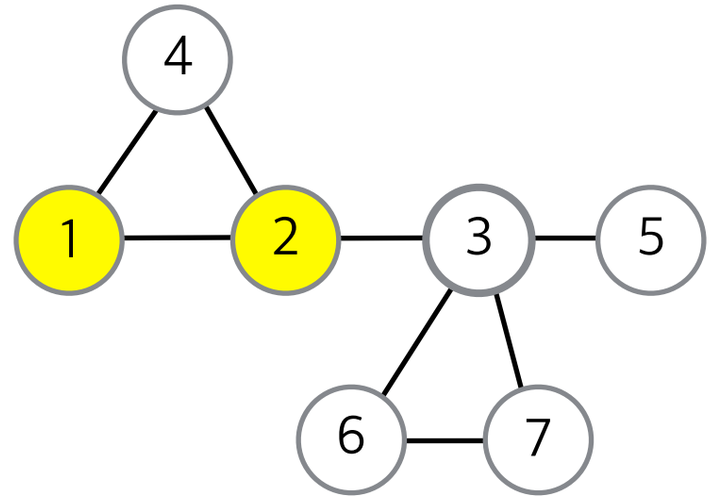
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

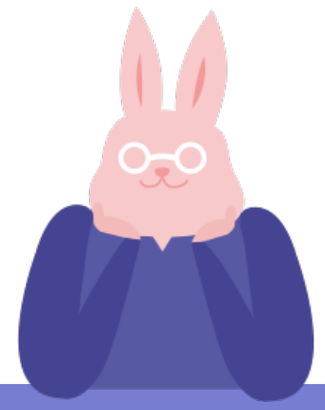
DFS(graph, 3, visited)

result = null



visited= [F, T, T, F, F, F, F, 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)

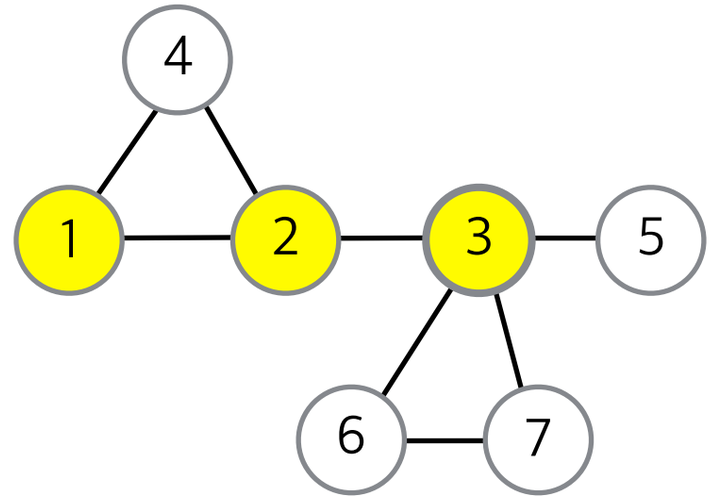
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

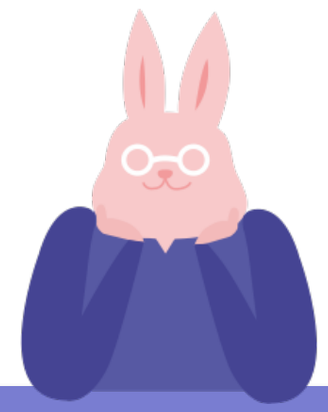
DFS(graph, 3, visited)

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)

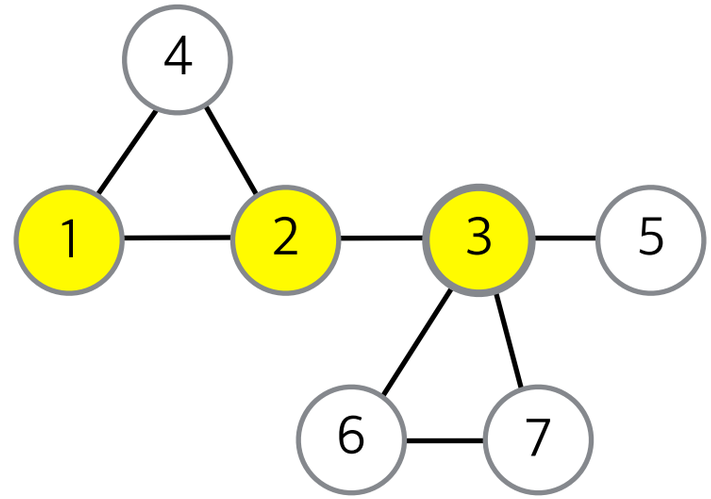
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

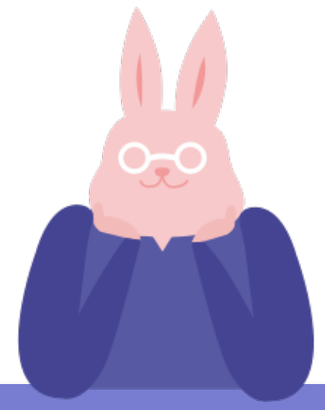
DFS(graph, 3, visited)

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)

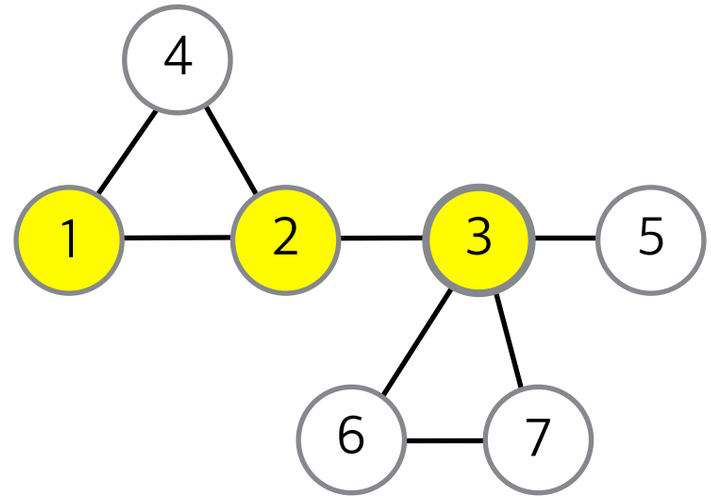
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

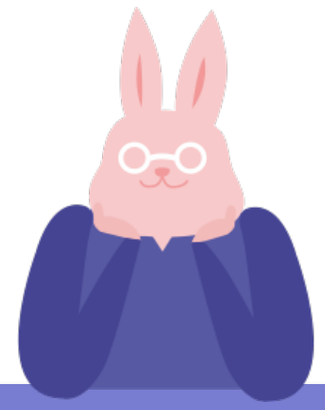
DFS(graph, 3, visited)

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)

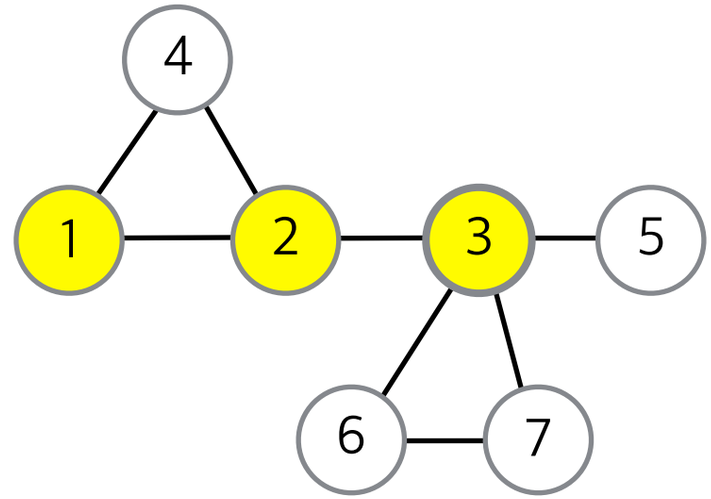
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

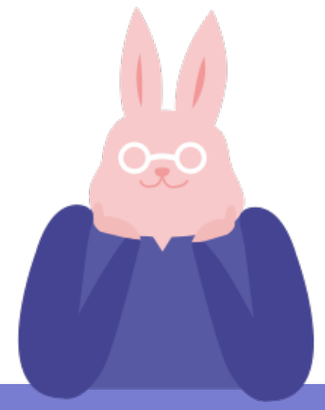
DFS(graph, 3, visited)

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)

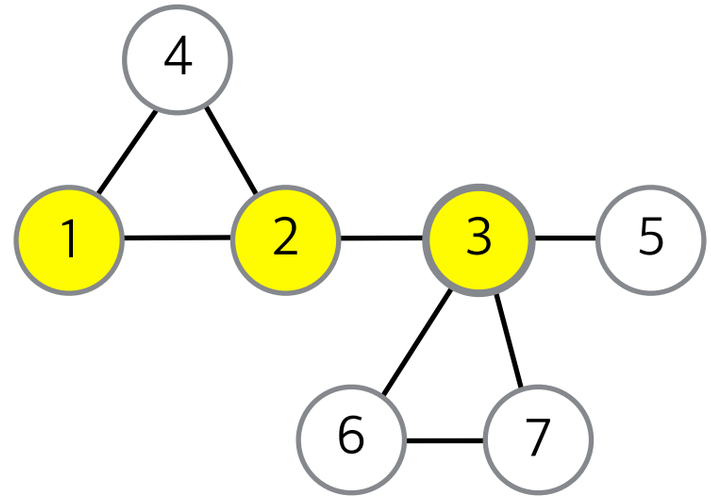
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

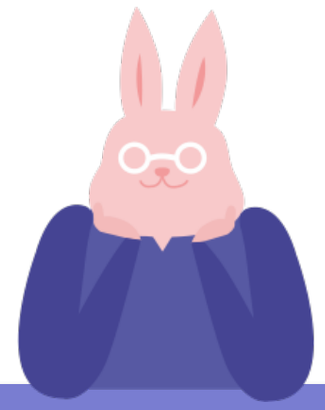
DFS(graph, 3, visited)

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)

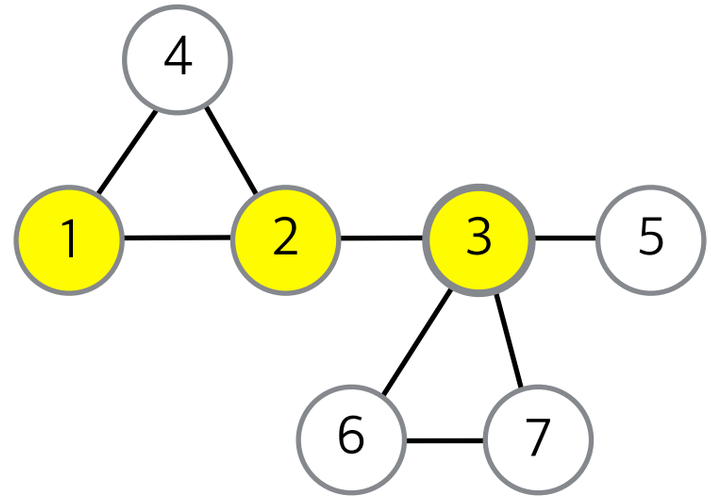
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

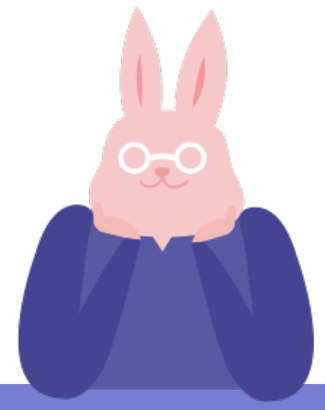
DFS(graph, 3, visited)

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)

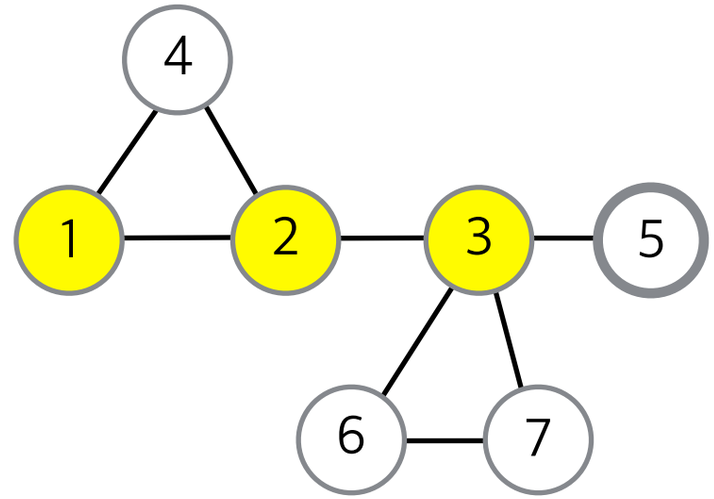
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

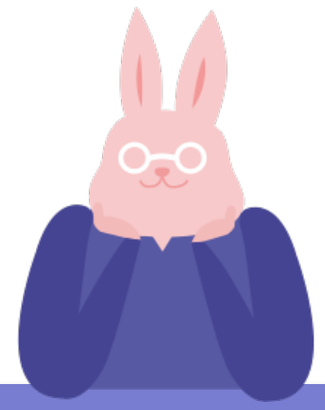
DFS(graph, 5, visited)

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)

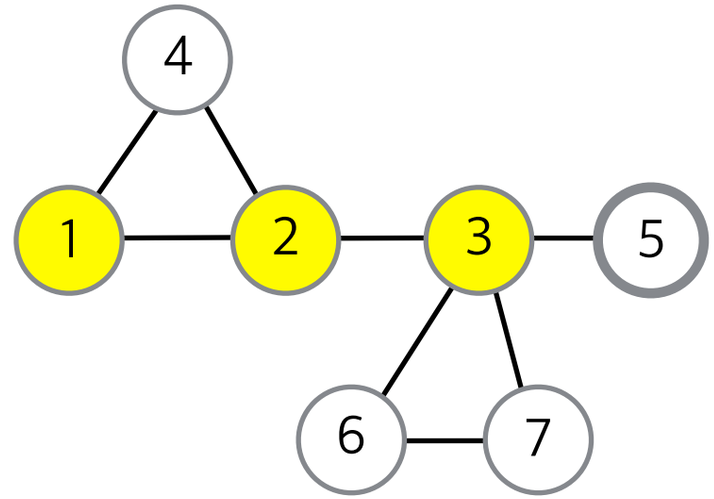
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

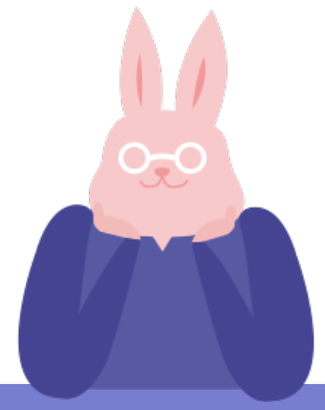
DFS(graph, 5, visited)

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)

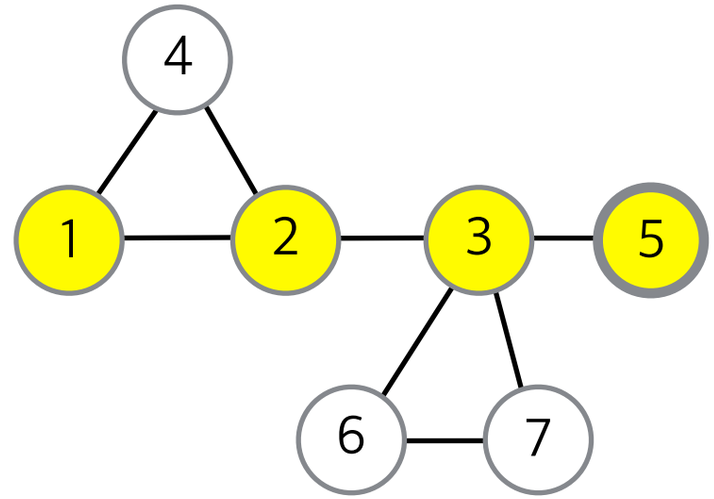
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

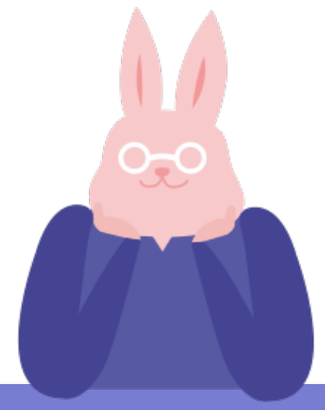
DFS(graph, 5, visited)

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)

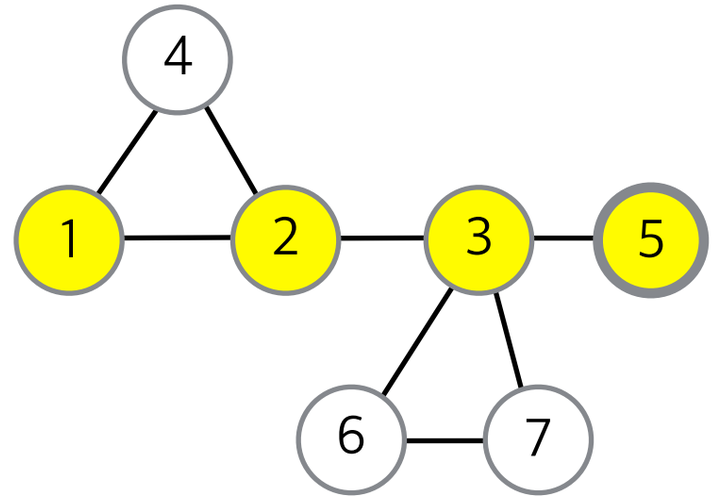
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

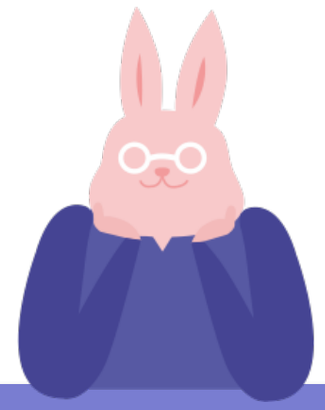
DFS(graph, 5, visited)

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)

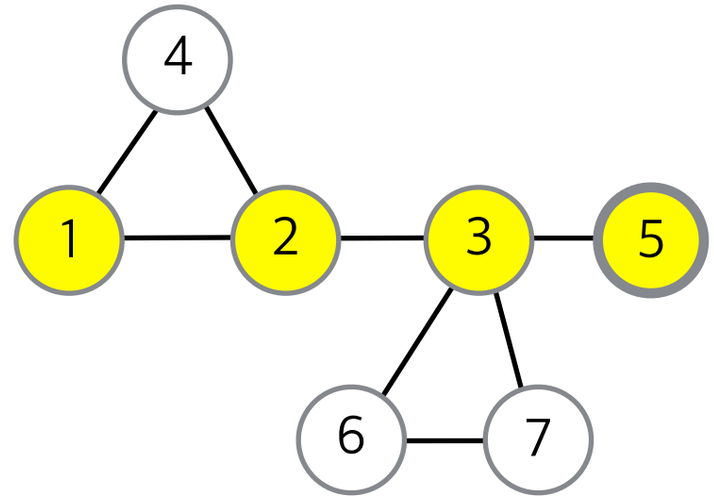
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

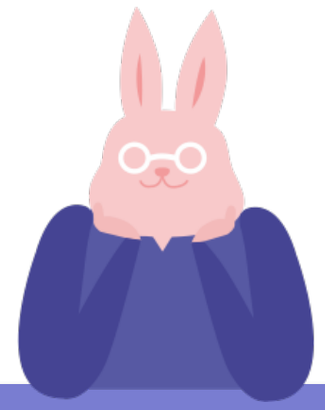
DFS(graph, 5, visited)

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)

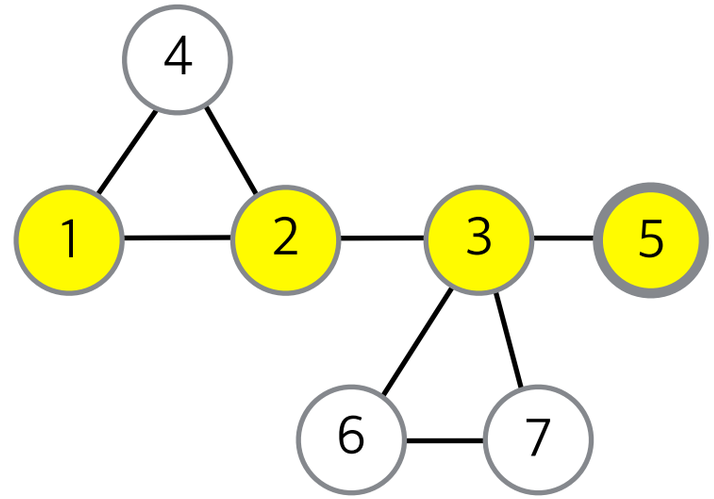
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

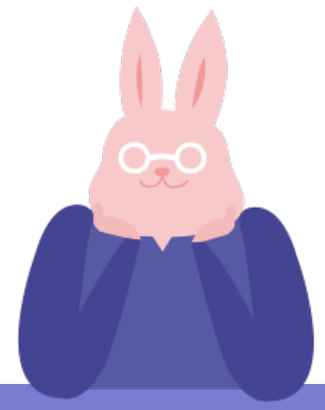
DFS(graph, 5, visited)

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)

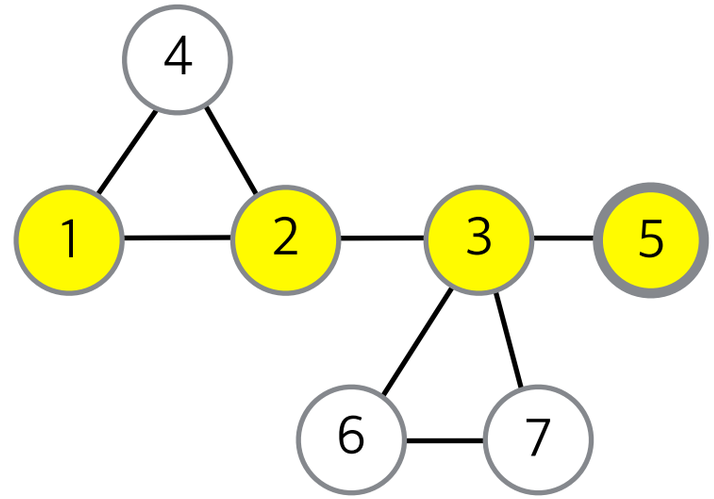
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

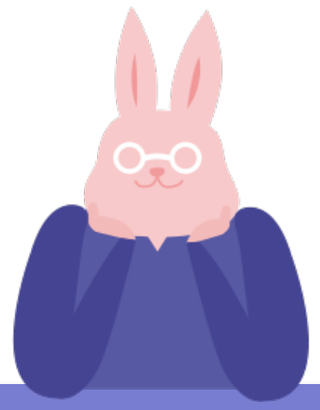
DFS(graph, 5, visited)

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)

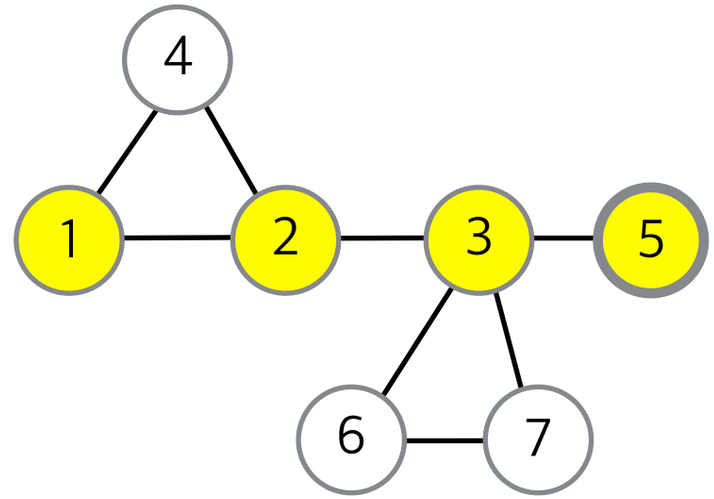
result = [2], v = 3

DFS(graph, 3, visited)

result = [3], v = 5

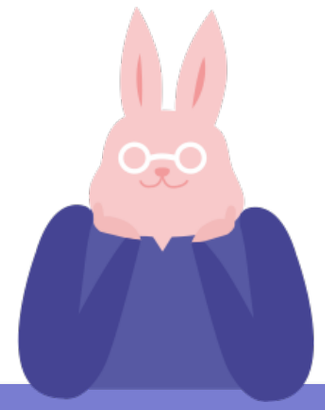
DFS(graph, 5, visited)

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

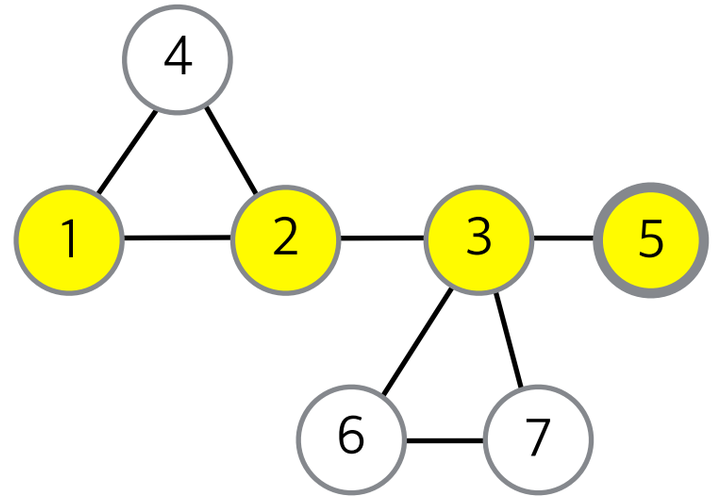
DFS(graph, 3, visited)

result = [3], v = 5

DFS(graph, 5, visited)

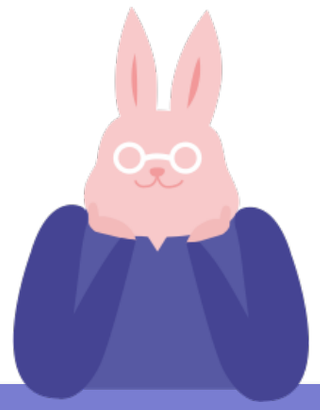
result = [5], v = 3

[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)

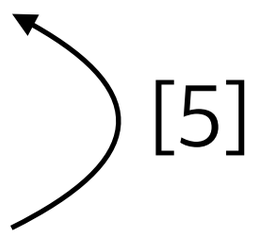
result = [1], v = 2

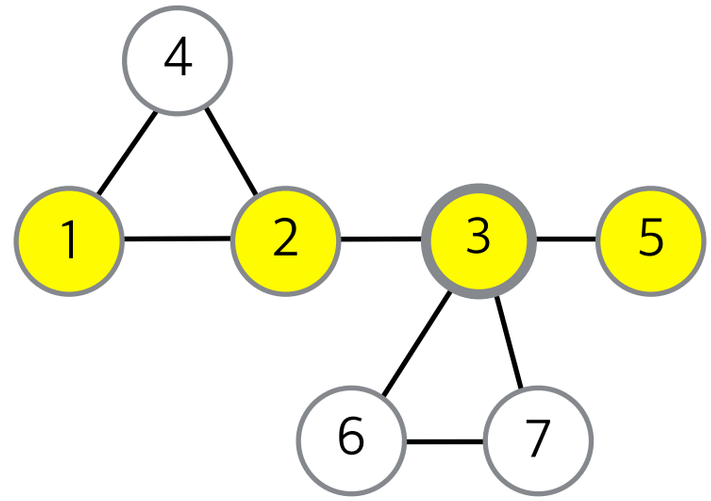
DFS(graph, 2, visited)

result = [2], v = 3

DFS(graph, 3, visited)

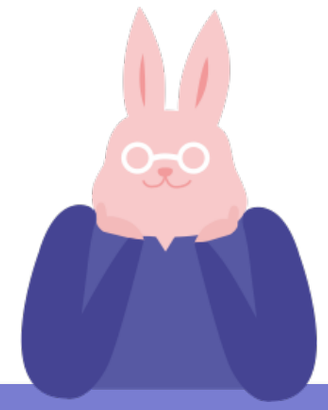
result = [3], v = 5





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)

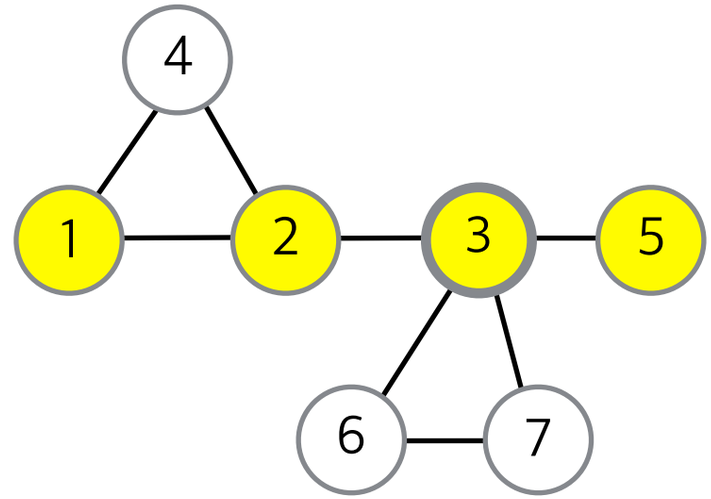
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

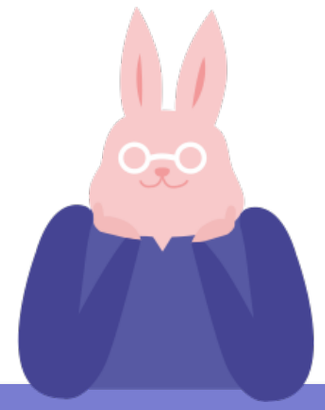
DFS(graph, 3, visited)

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)

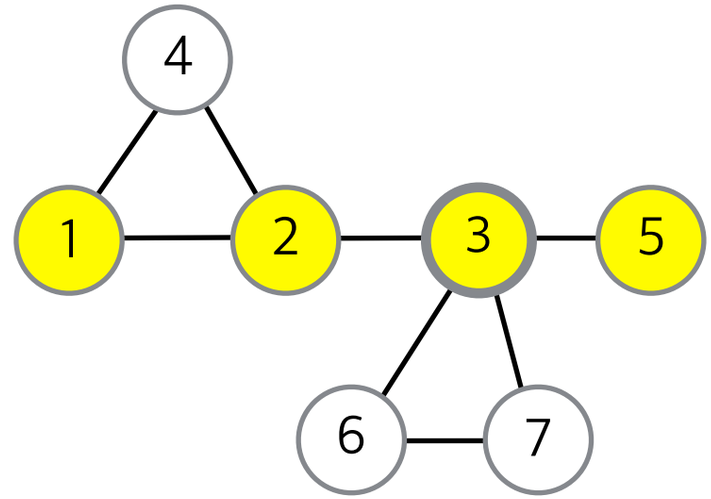
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

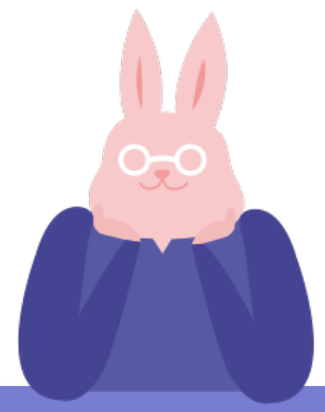
DFS(graph, 3, visited)

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)

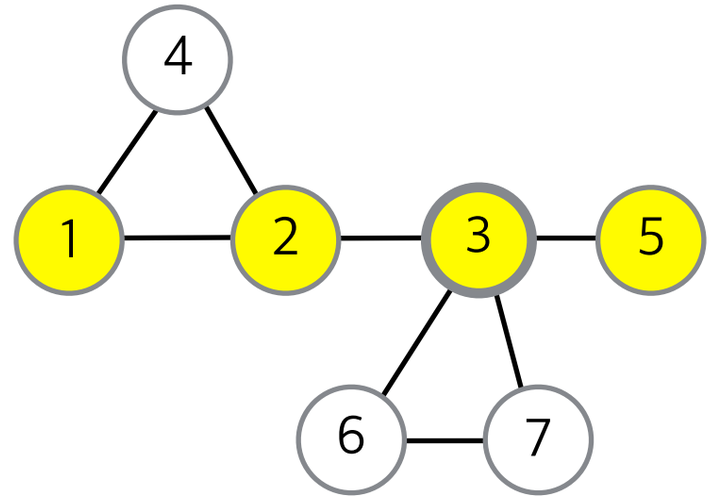
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

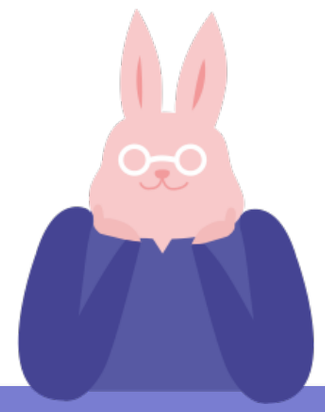
DFS(graph, 3, visited)

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)

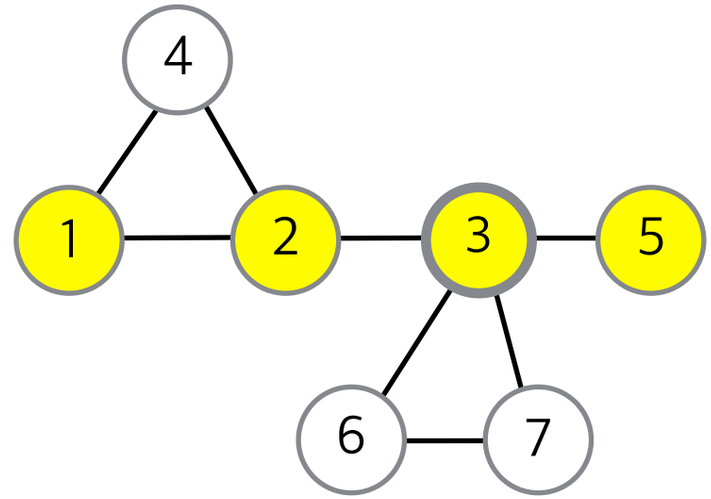
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

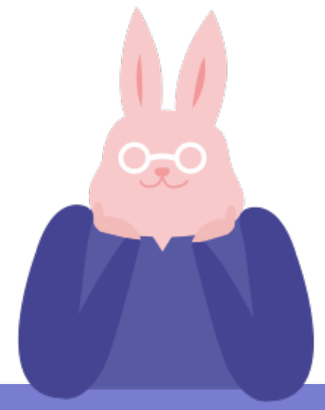
DFS(graph, 3, visited)

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

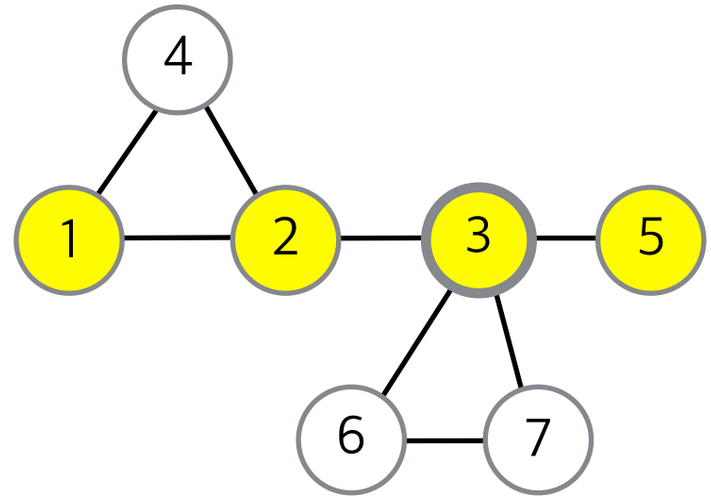
DFS(graph, 2, visited)

result = [2], v = 3

DFS(graph, 3, visited)

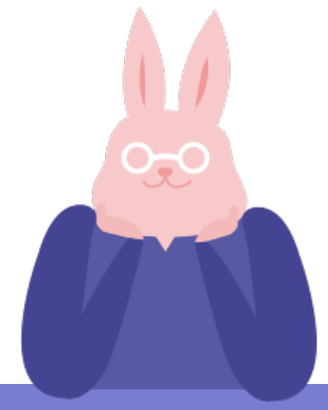
result = [3, 5], v = 6

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)

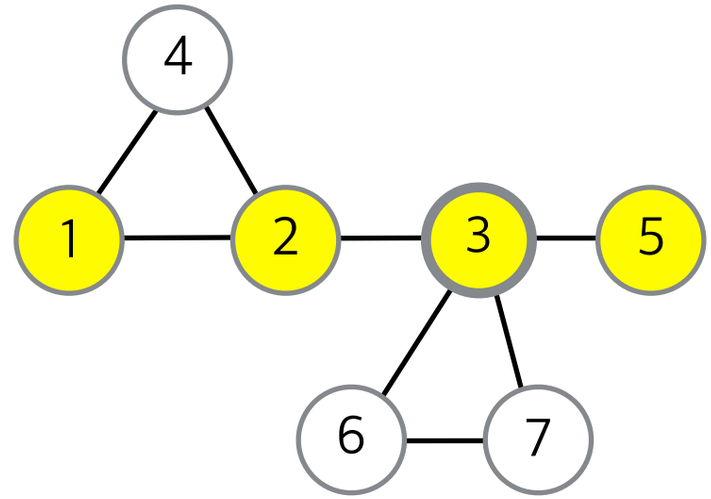
result = [2], v = 3

DFS(graph, 3, visited)

result = [3, 5], v = 6

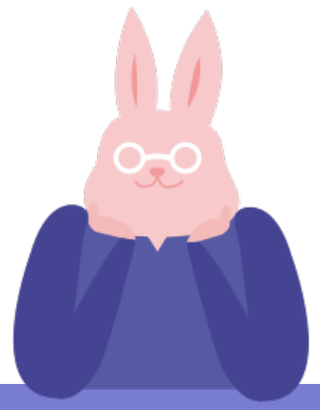
DFS(graph, 6, visited)

[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)

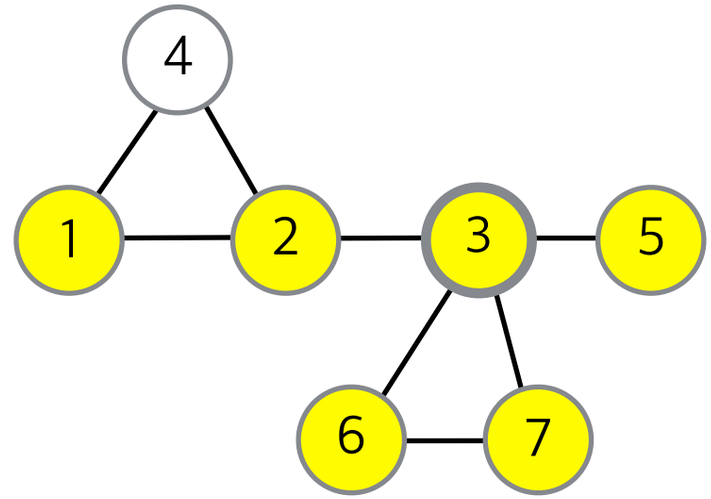
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

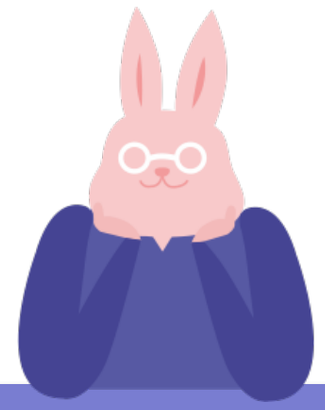
DFS(graph, 3, visited)

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)

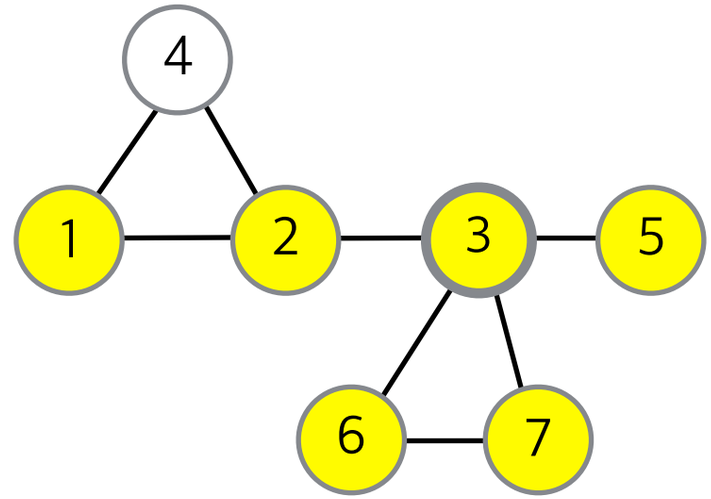
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

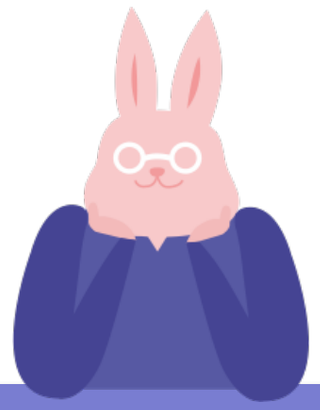
DFS(graph, 3, visited)

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)

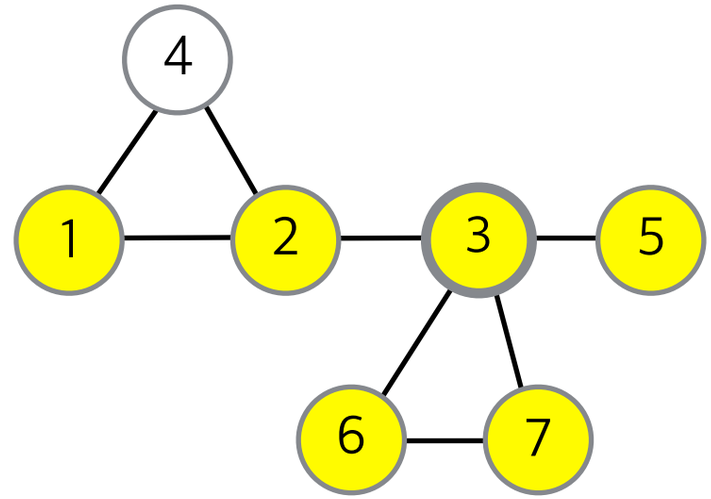
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

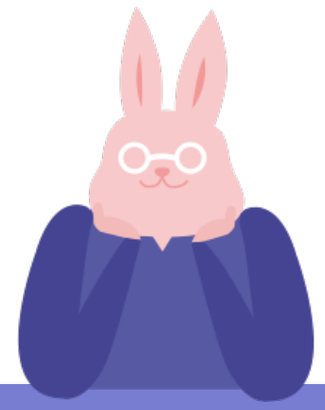
DFS(graph, 3, visited)

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)

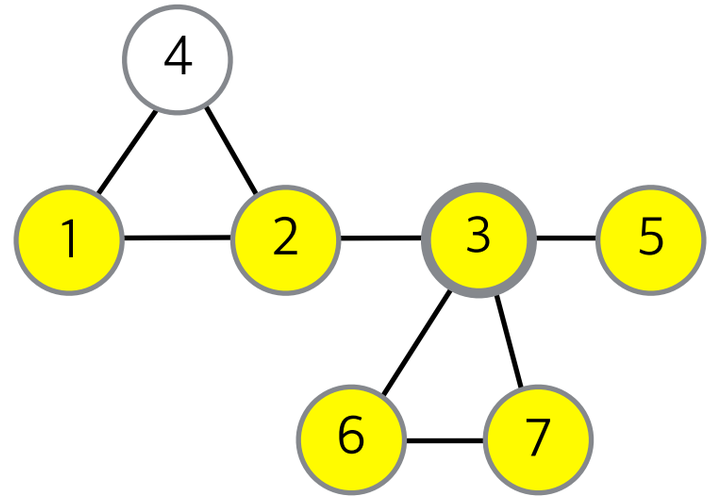
result = [1], v = 2

DFS(graph, 2, visited)

result = [2], v = 3

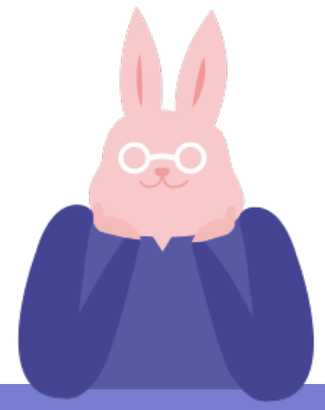
DFS(graph, 3, visited)

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

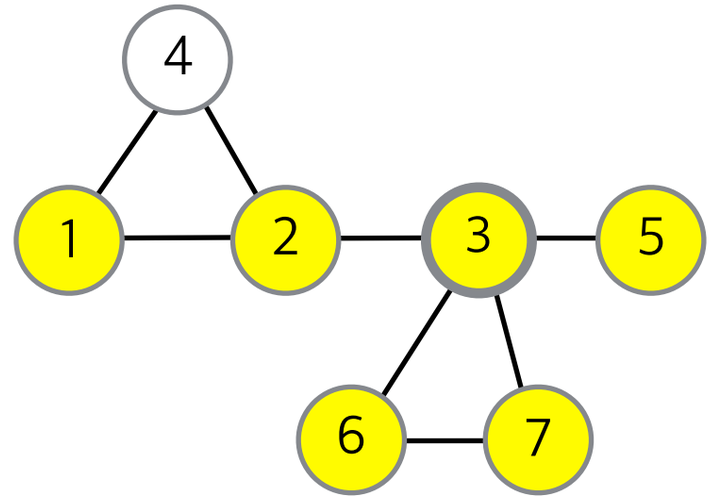
DFS(graph, 2, visited)

result = [2], v = 3

DFS(graph, 3, visited)

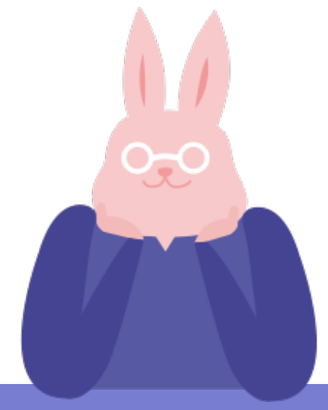
result = [3, 5, 6, 7], v = 7

[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
```

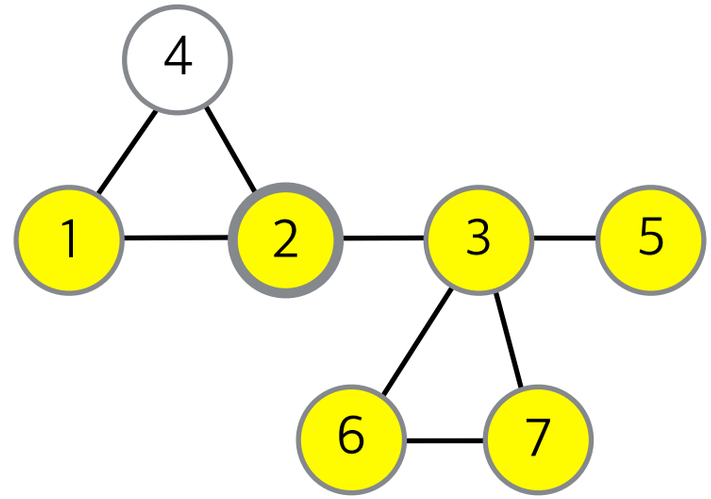
DFS(graph, 1, visited)

result = [1], v = 2

DFS(graph, 2, visited)

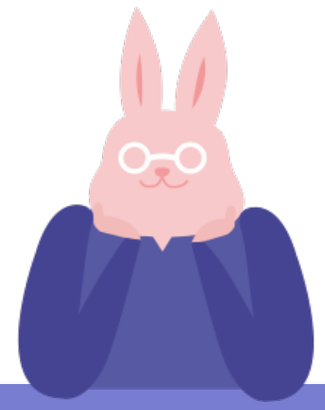
result = [2], v = 3

[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

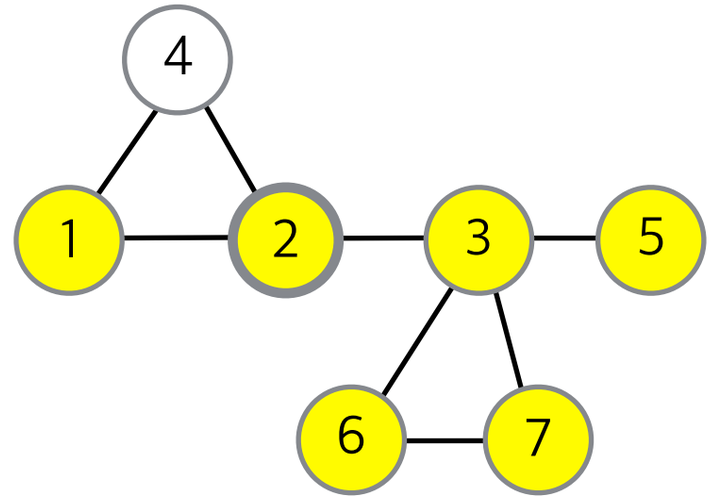
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    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

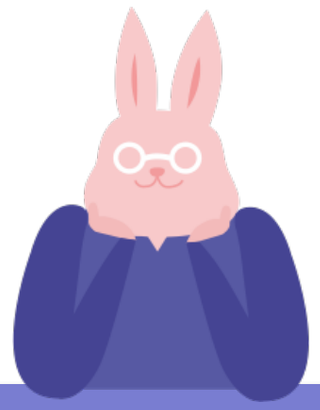
DFS(graph, 2, visited)

result = [2, 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

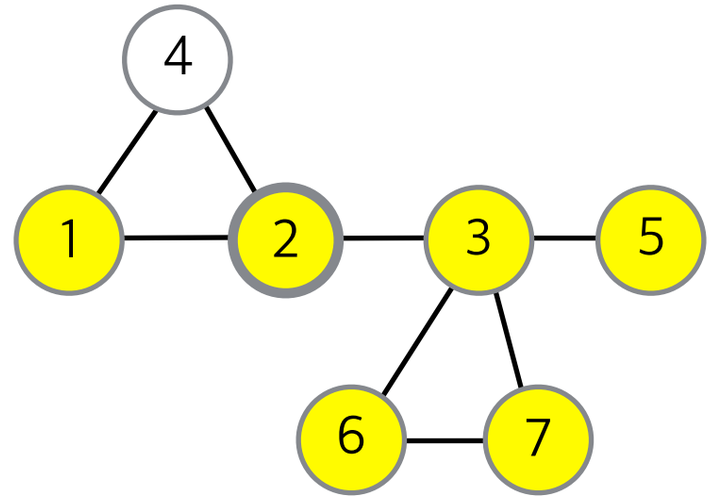
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    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

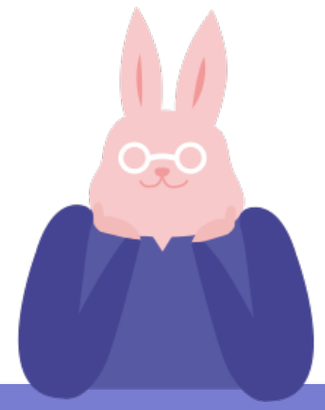
DFS(graph, 2, visited)

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

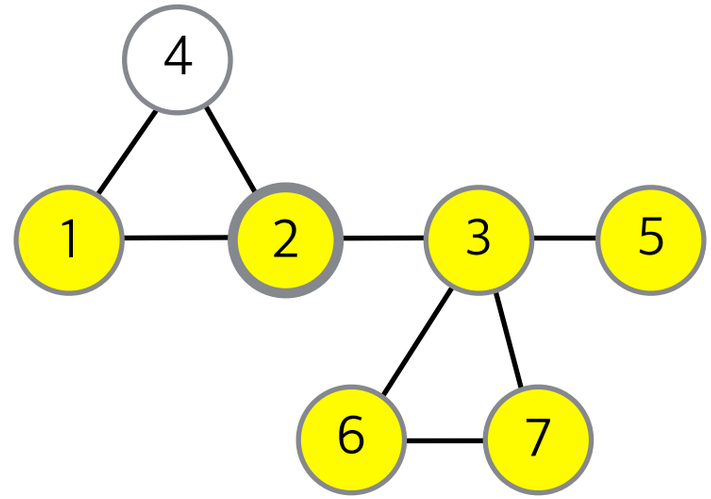
```
def DFS(graph, x, visited) :
      visited[x] = True
      result = [x]

      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

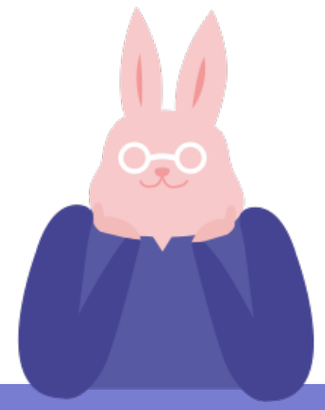
DFS(graph, 2, visited)

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
```

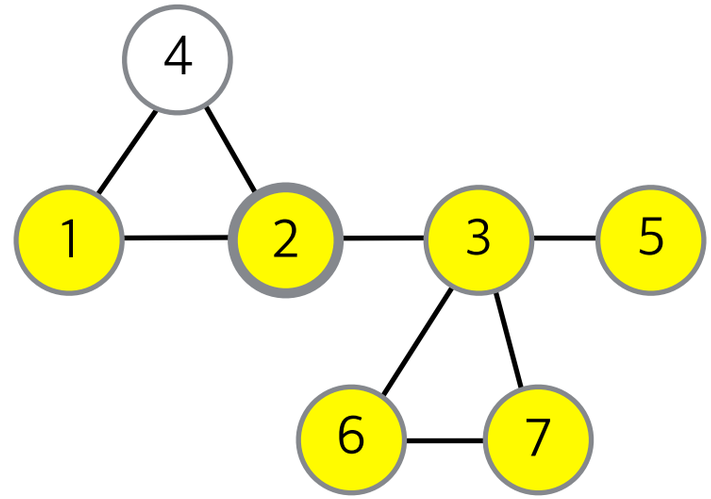
DFS(graph, 1, visited)

result = [1], v = 2

DFS(graph, 2, visited)

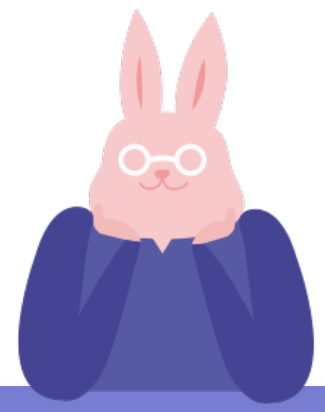
result = [2, 3, 5, 6, 7], v = 4

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
```

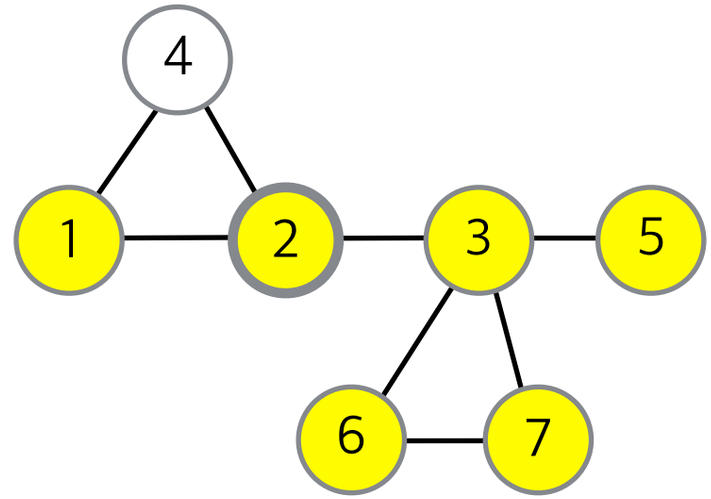
DFS(graph, 1, visited)

result = [1], v = 2

DFS(graph, 2, visited)

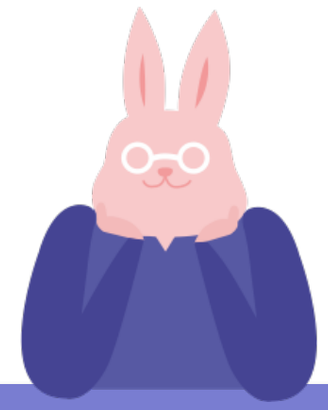
result = [2, 3, 5, 6, 7], v = 4

[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

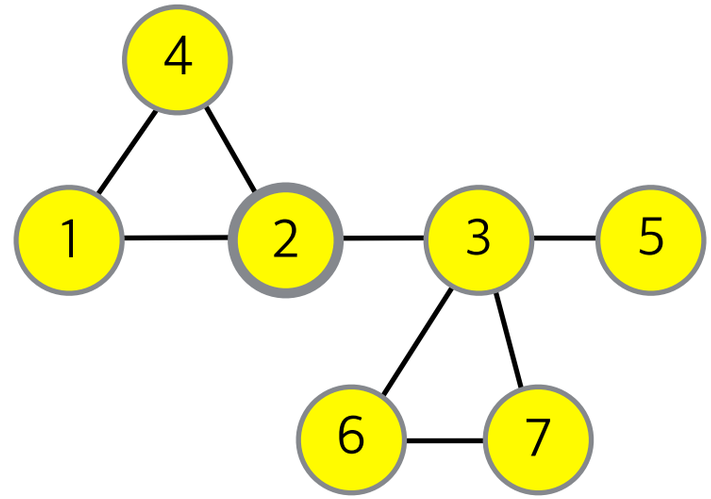
```
def DFS(graph, x, visited) :
    visited[x] = True
    result = [x]

    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

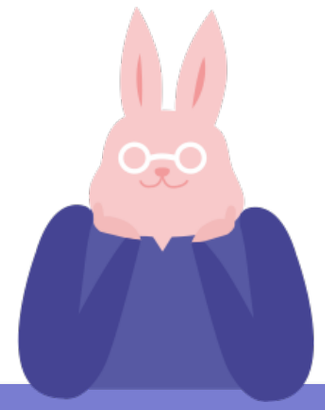
DFS(graph, 2, visited)

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

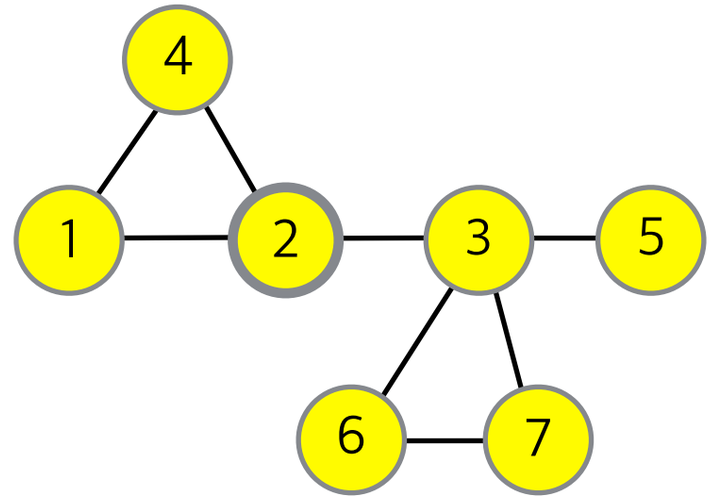
```
def DFS(graph, x, visited) :
      visited[x] = True
      result = [x]

      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

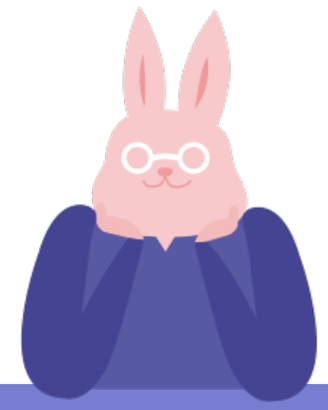
DFS(graph, 2, visited)

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]



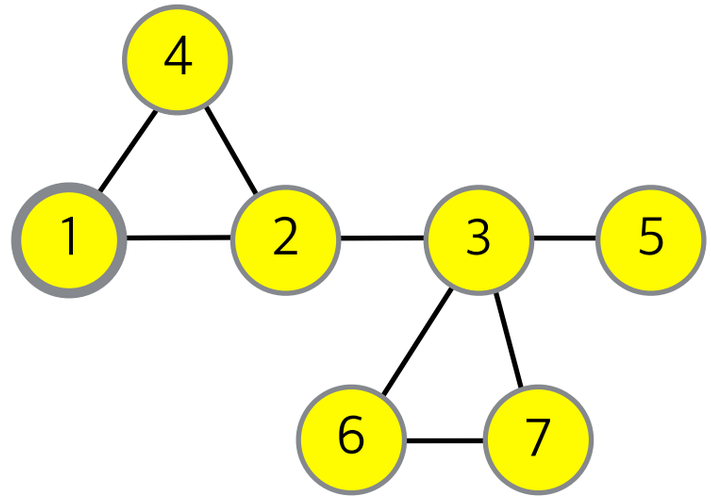
깊이 우선 탐색 (DFS) 구현

```
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)

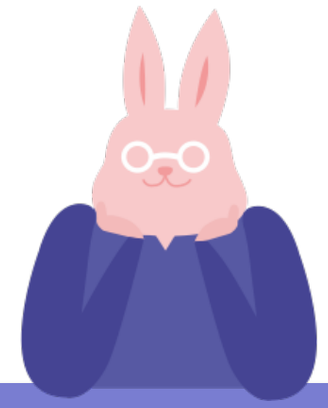
result = [1], v = 2

[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]

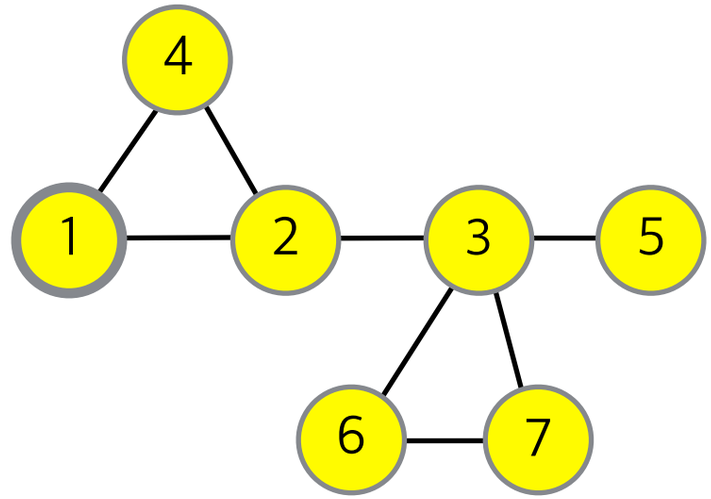


깊이 우선 탐색 (DFS) 구현

```
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
```

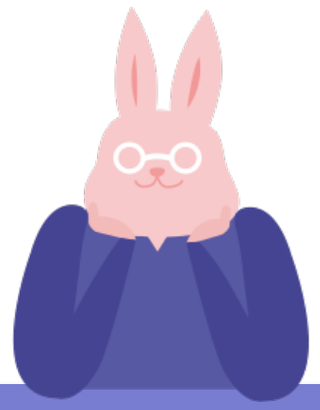
DFS(graph, 1, visited)

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]

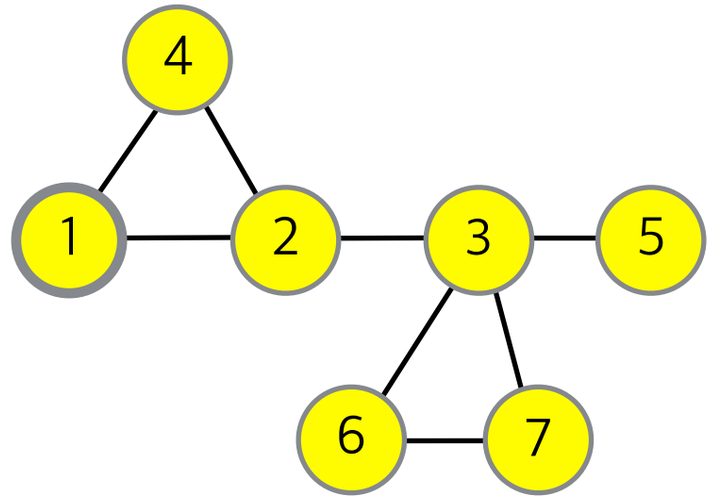


깊이 우선 탐색 (DFS) 구현

```
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
```

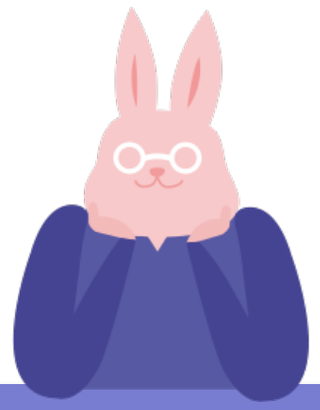
DFS(graph, 1, visited)

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]

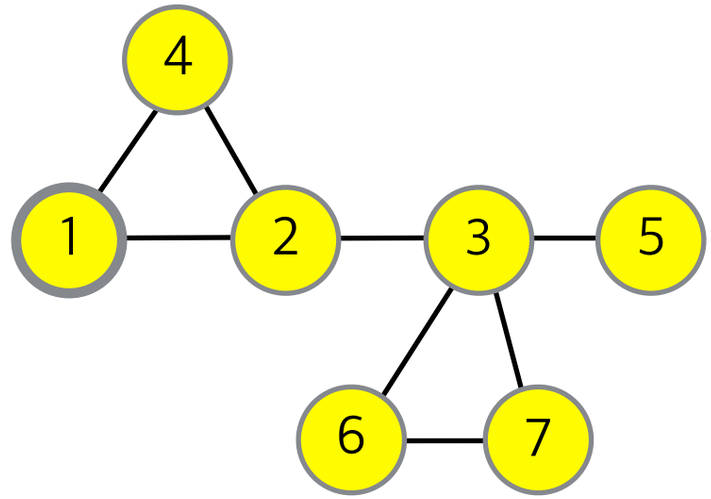


깊이 우선 탐색 (DFS) 구현

```
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
```

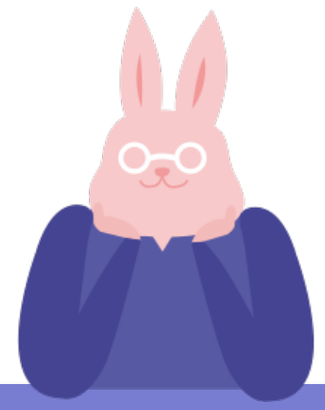
DFS(graph, 1, visited)

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



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

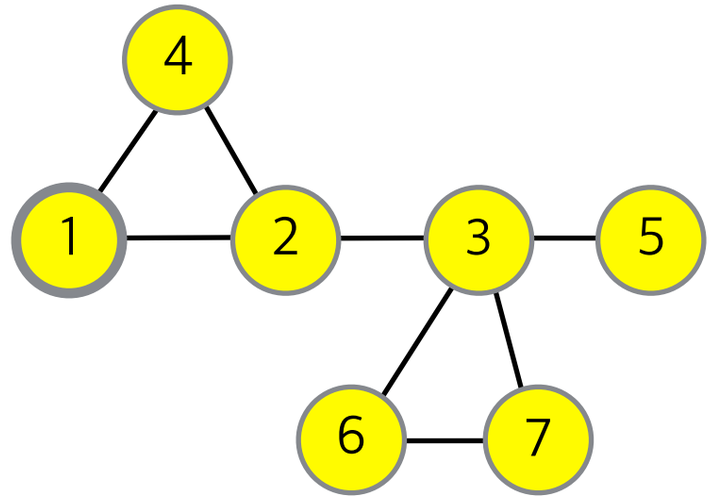


깊이 우선 탐색 (DFS) 구현

```
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
```

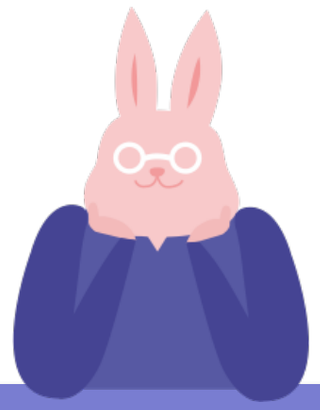
DFS(graph, 1, visited)

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]



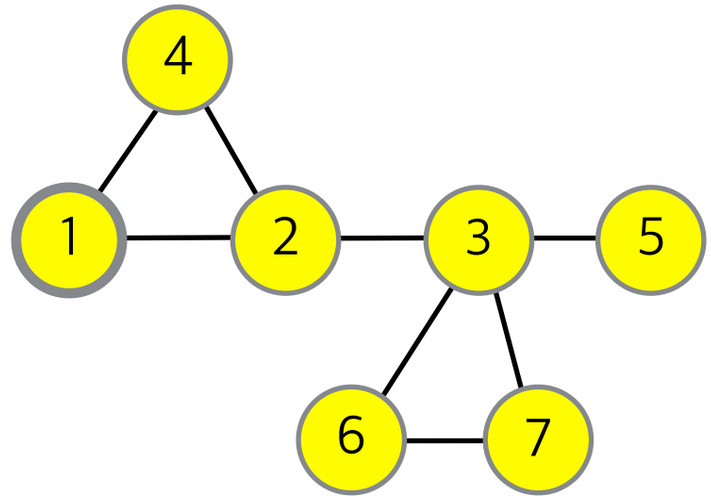
깊이 우선 탐색 (DFS) 구현

```
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
```

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

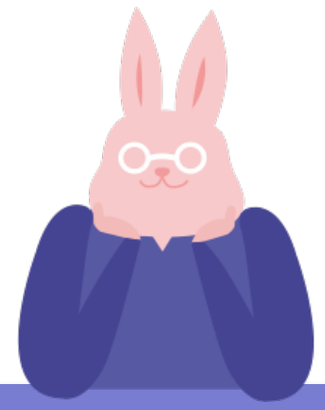
DFS(graph, 1, visited)

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



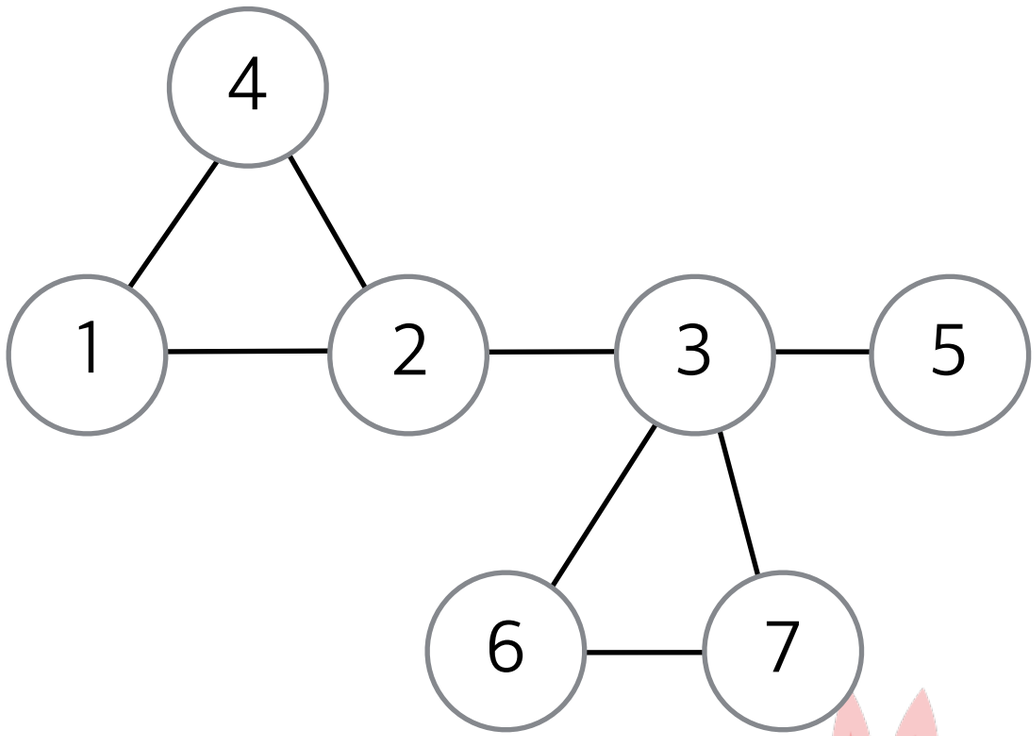
visited= [F, T, T, T, T, T, T, T]

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]



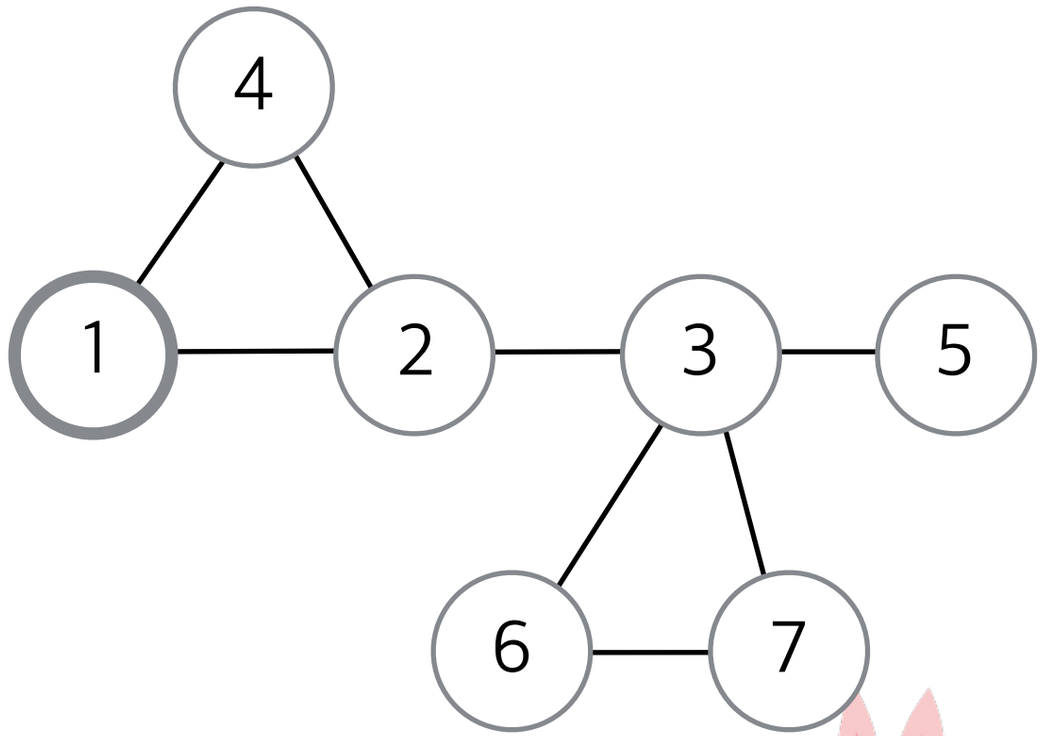
너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다



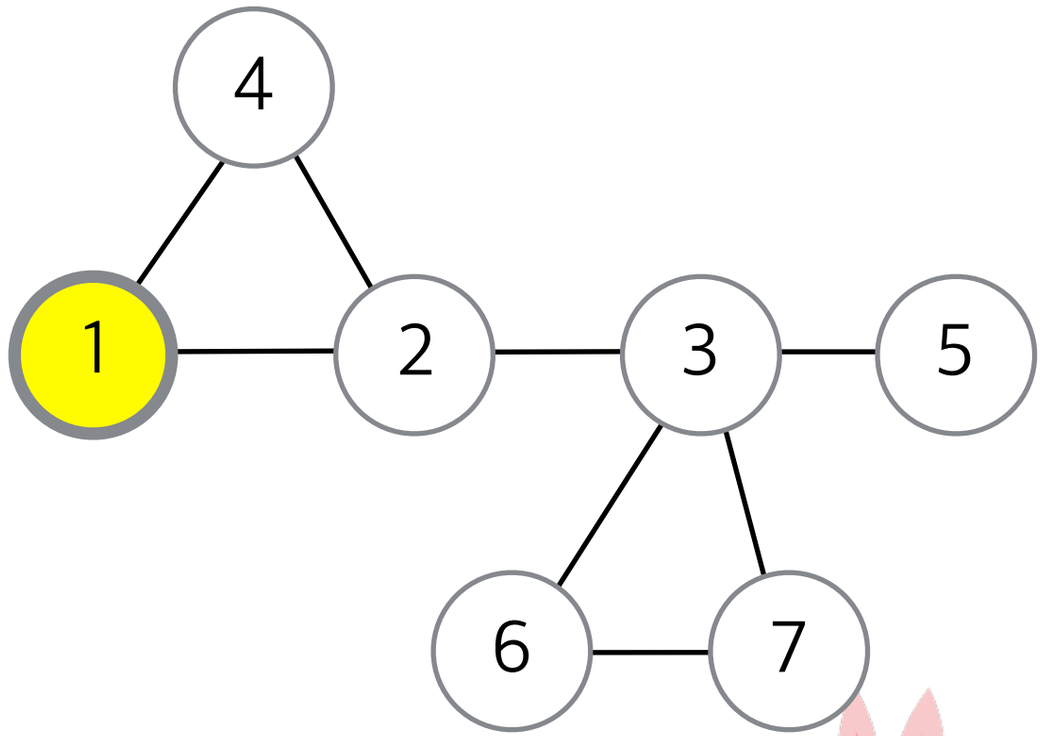
너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다



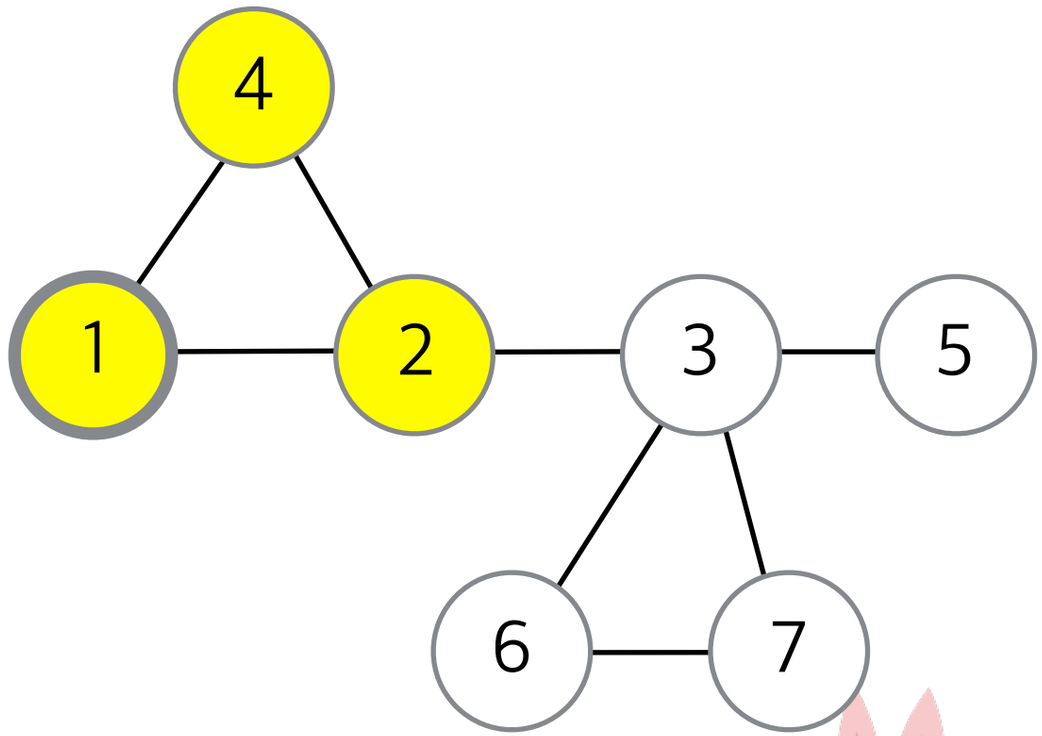
너비 우선 탐색 (BFS)

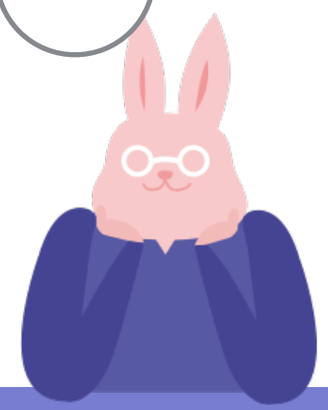
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다



너비 우선 탐색 (BFS)

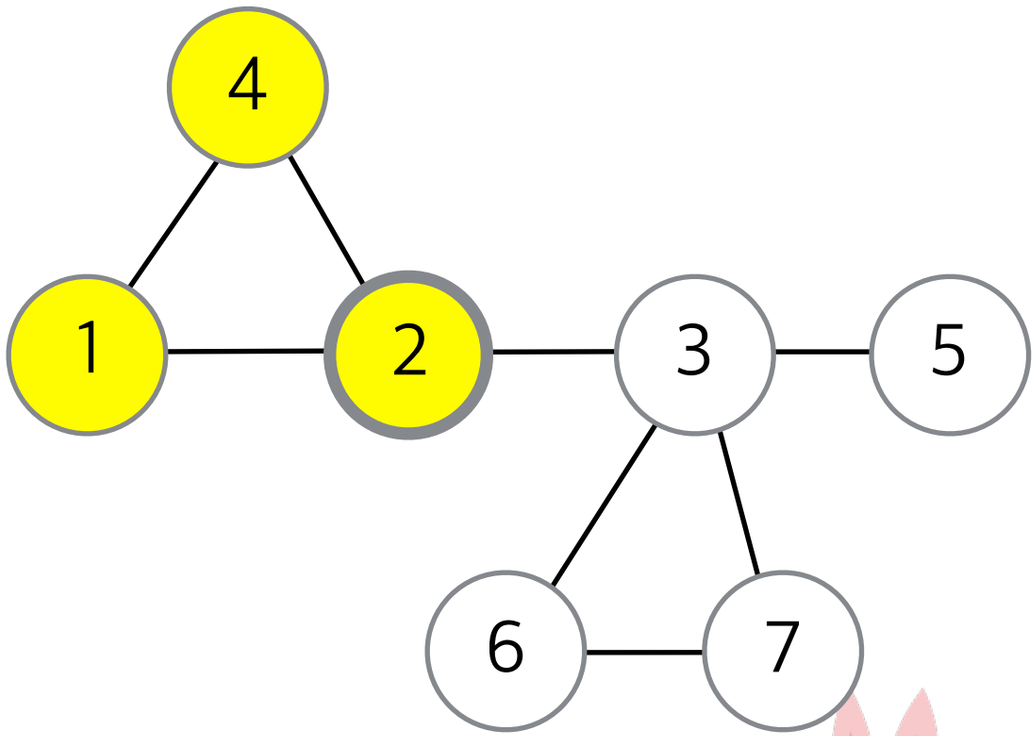
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다

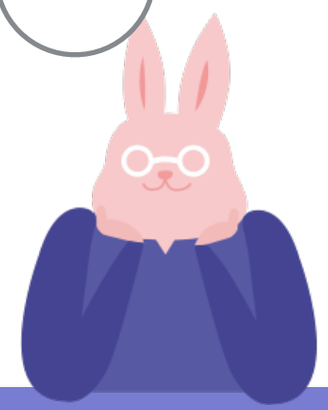




너비 우선 탐색 (BFS)

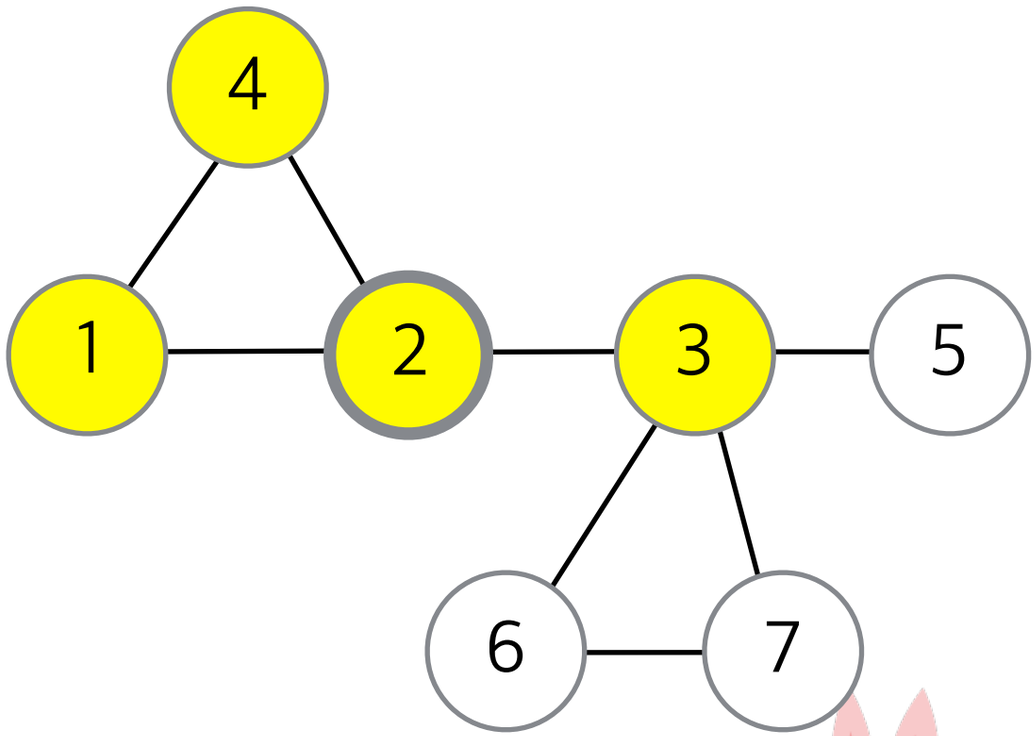
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다





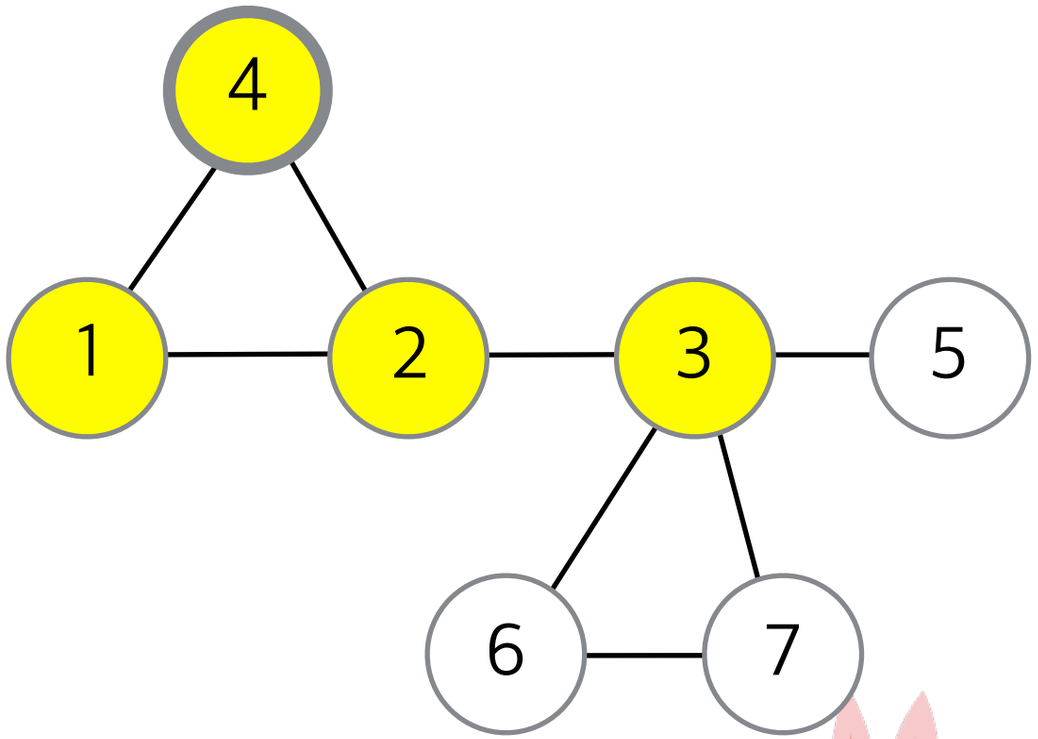
너비 우선 탐색 (BFS)

- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다



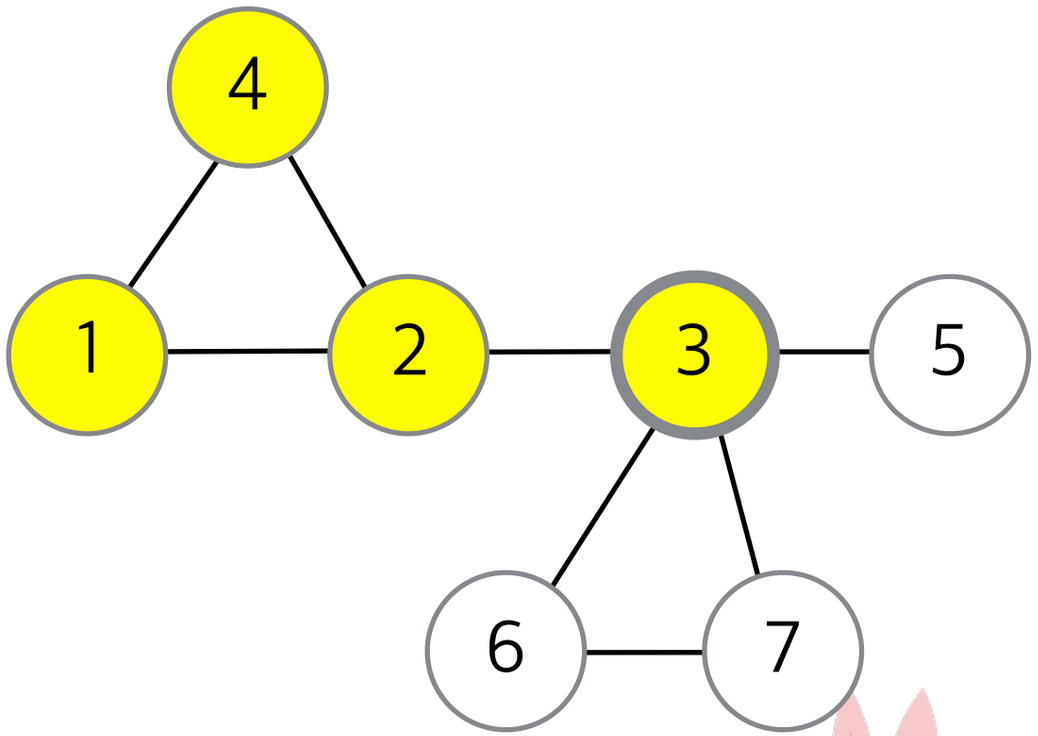
너비 우선 탐색 (BFS)

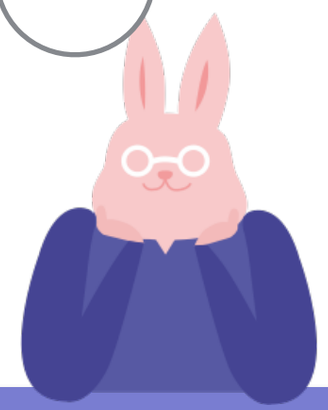
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다



너비 우선 탐색 (BFS)

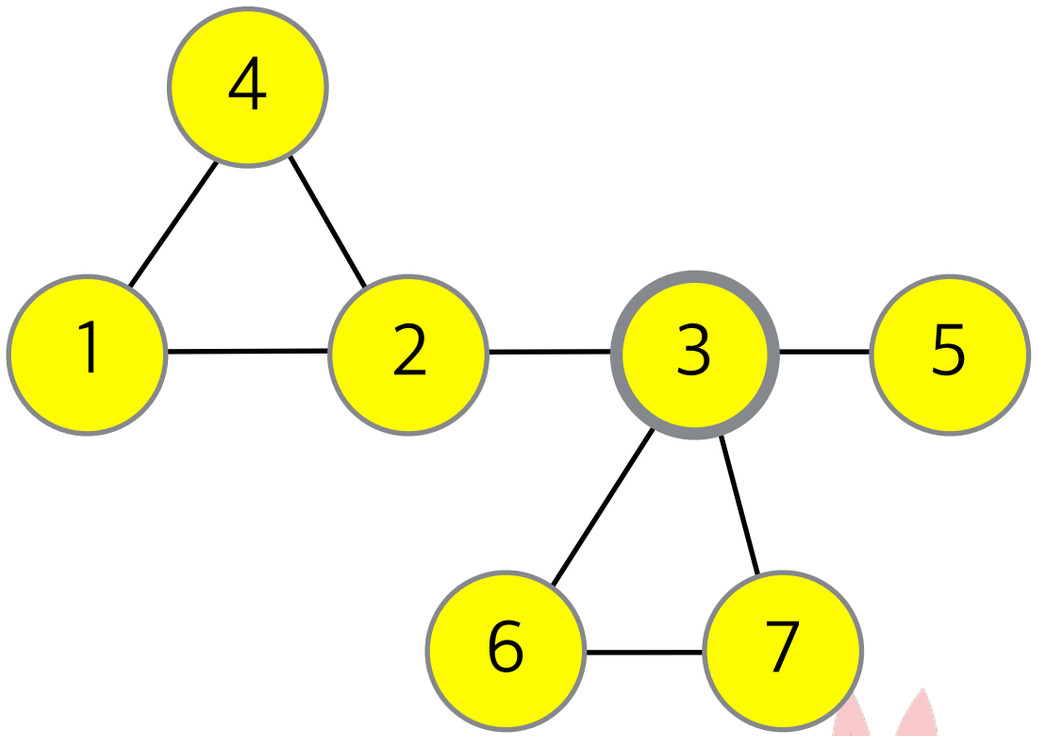
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다

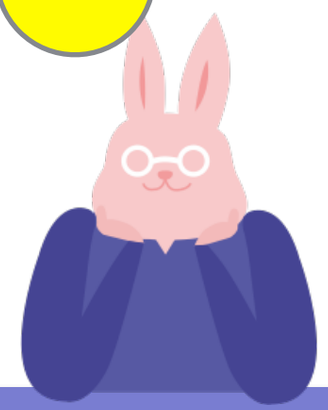




너비 우선 탐색 (BFS)

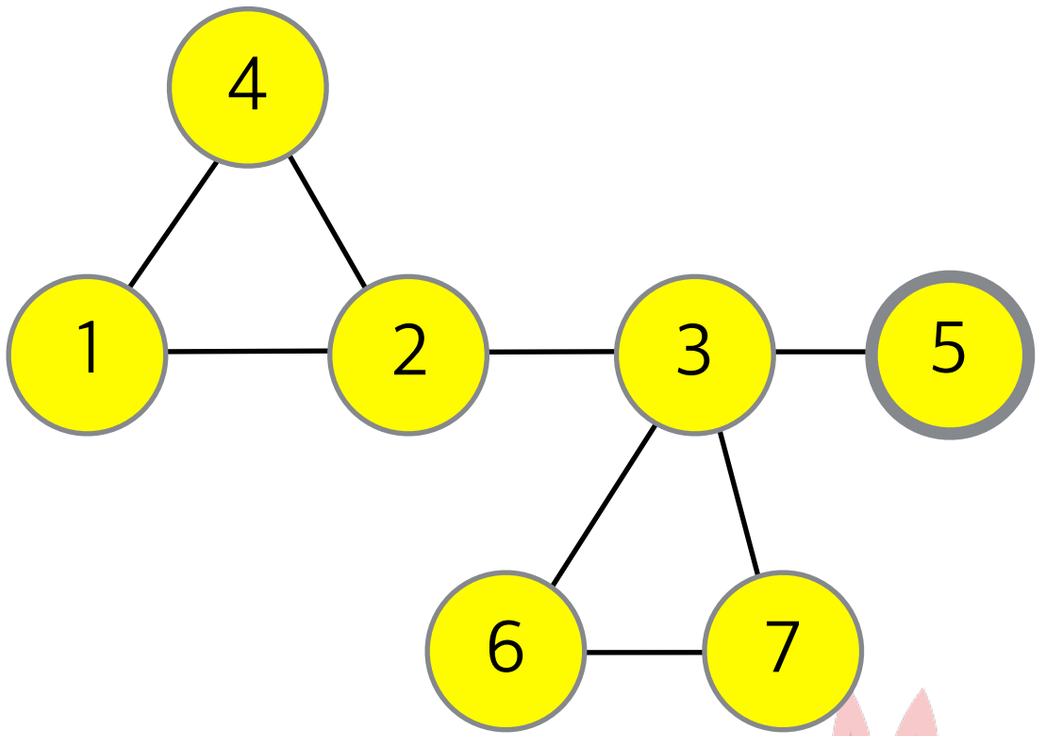
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다

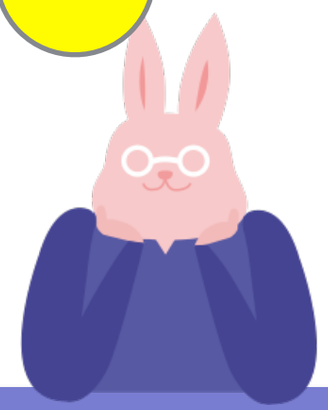




너비 우선 탐색 (BFS)

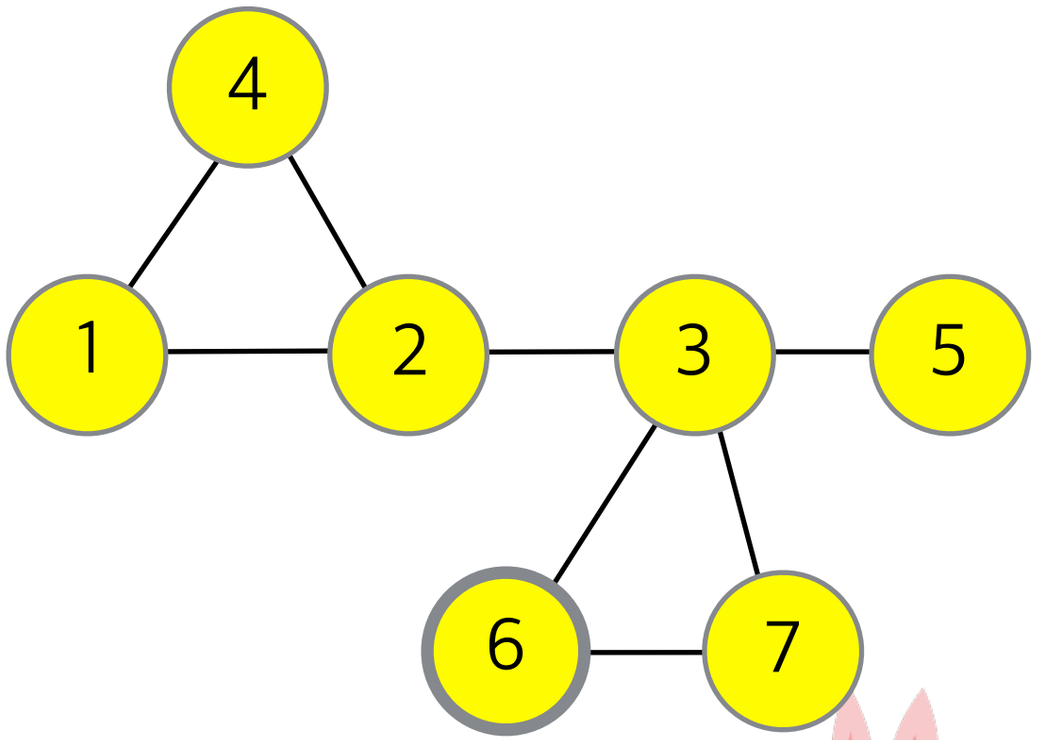
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다

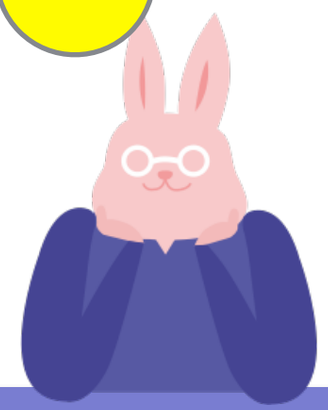




너비 우선 탐색 (BFS)

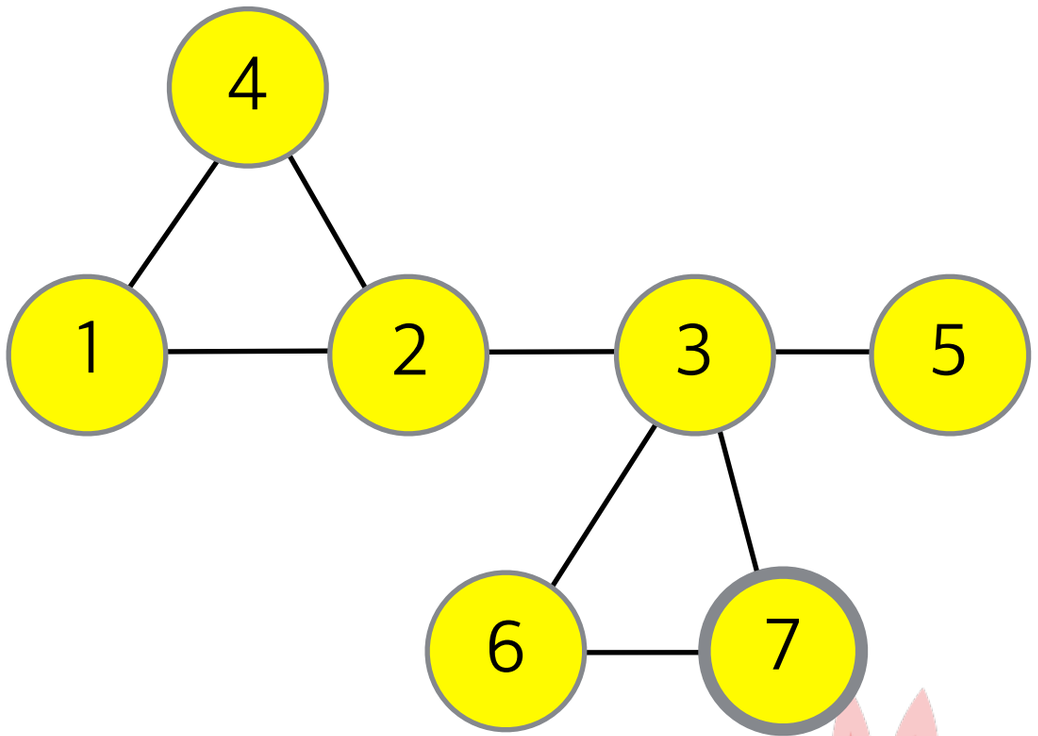
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다





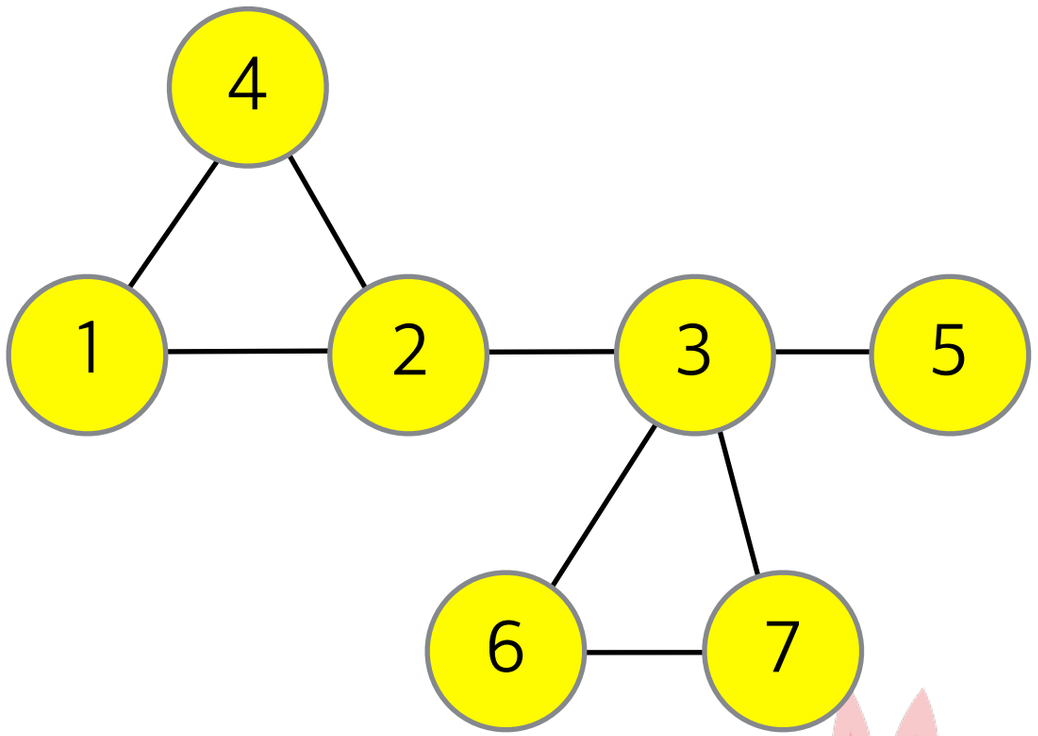
너비 우선 탐색 (BFS)

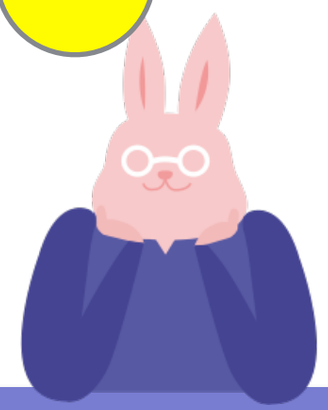
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다



너비 우선 탐색 (BFS)

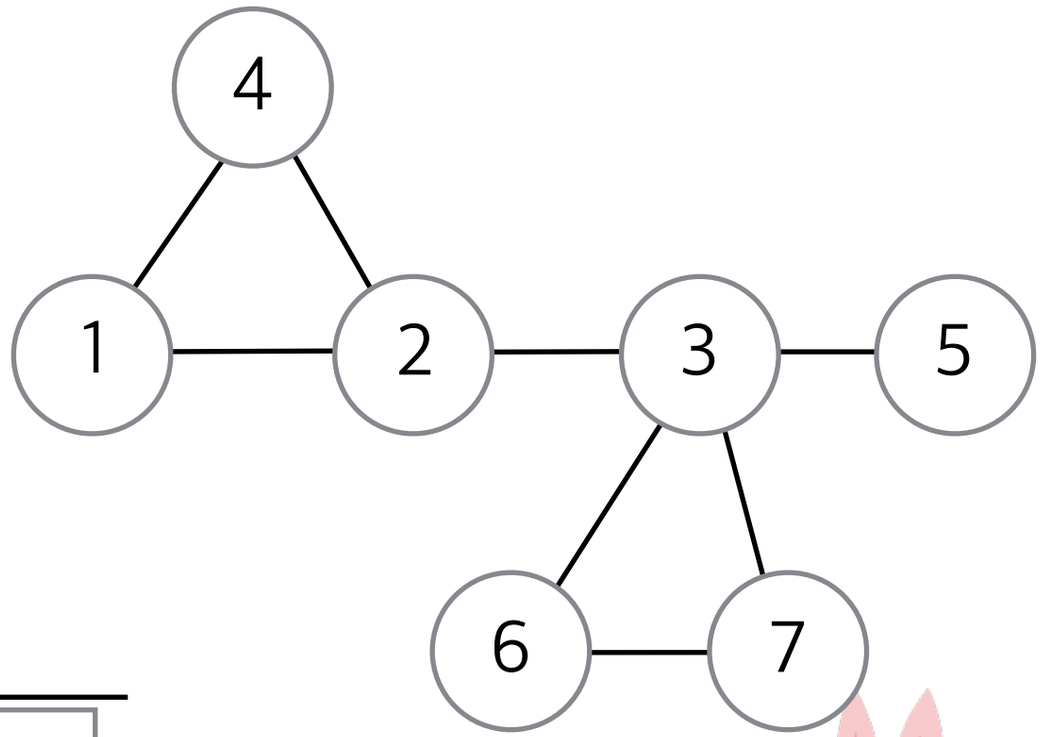
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다

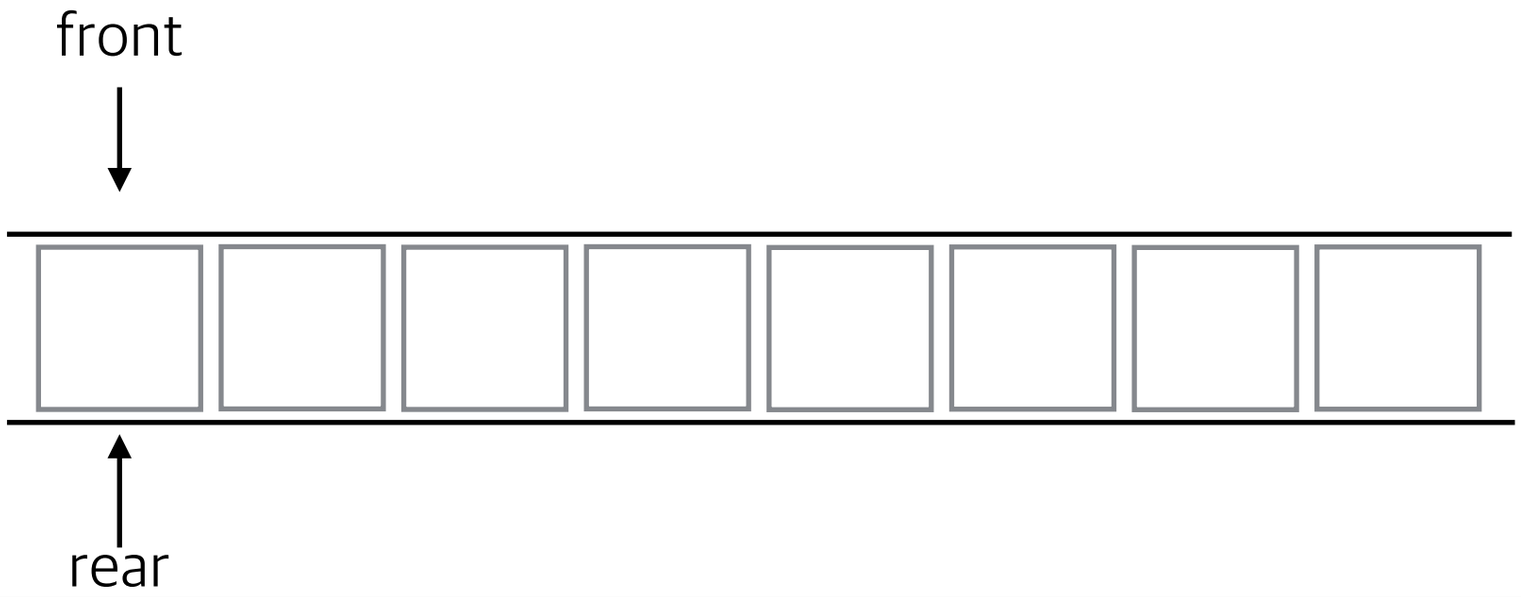


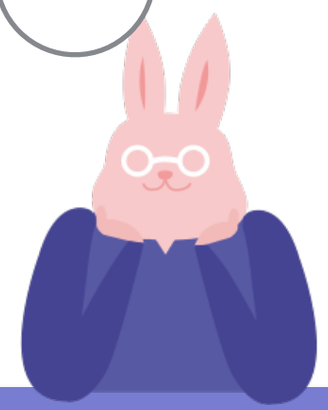


너비 우선 탐색 (BFS)

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

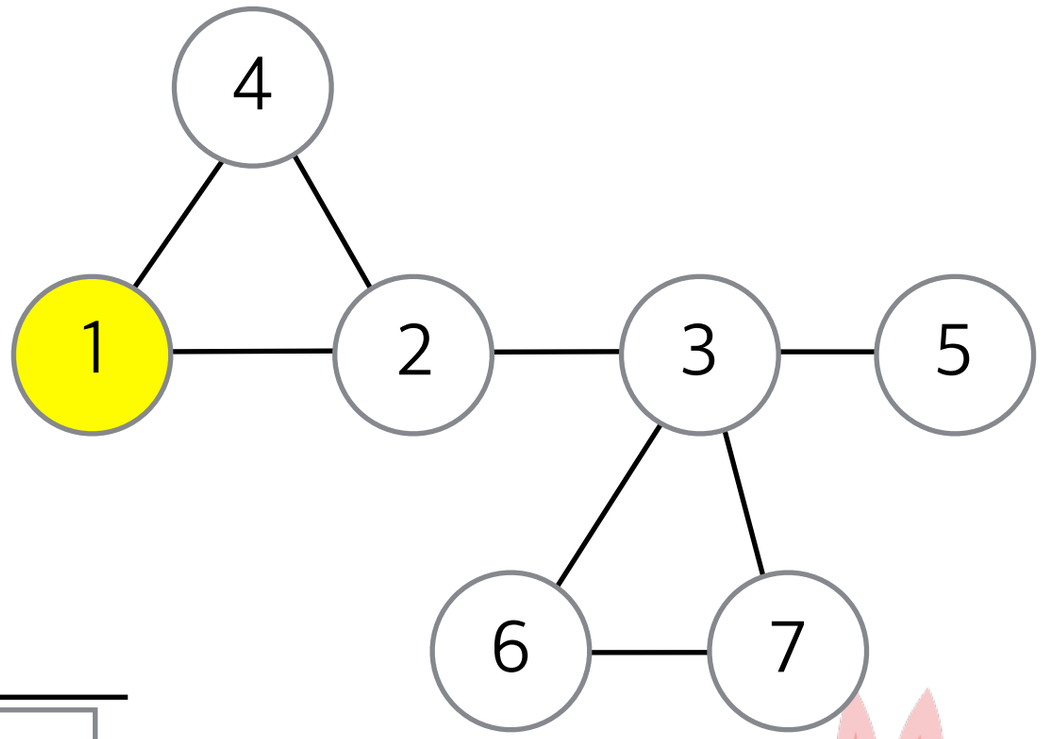




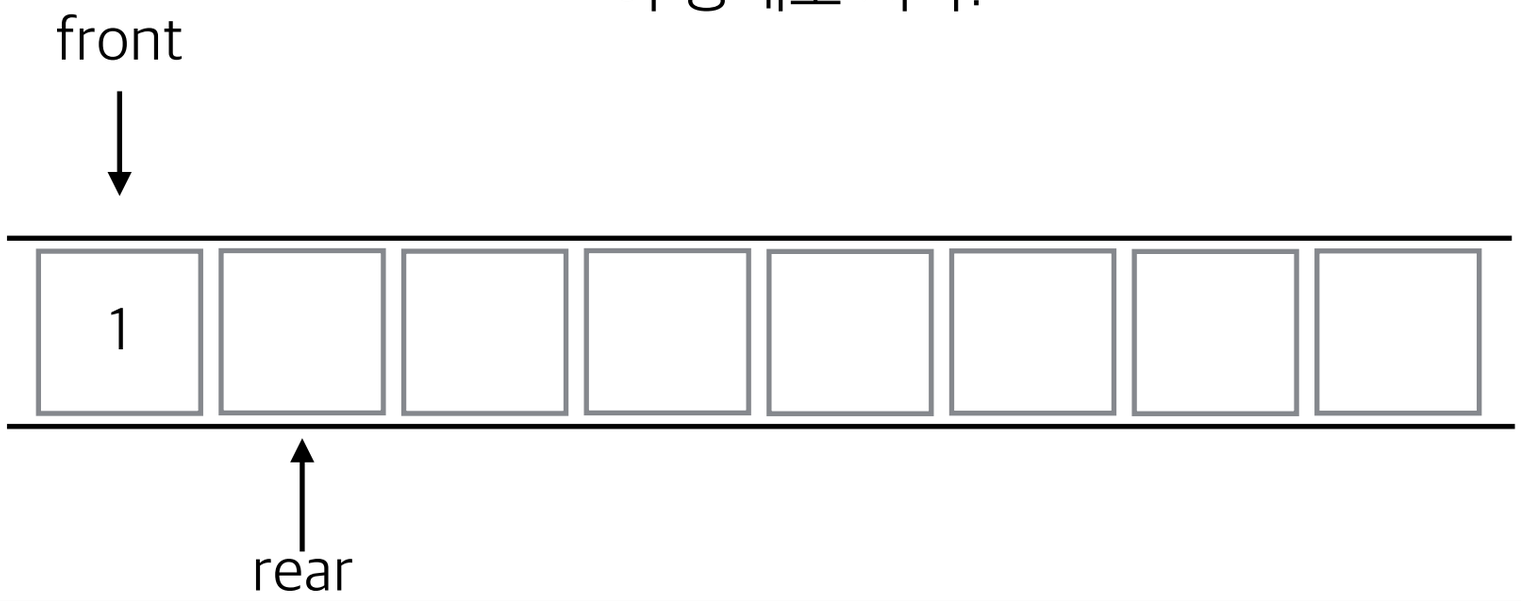


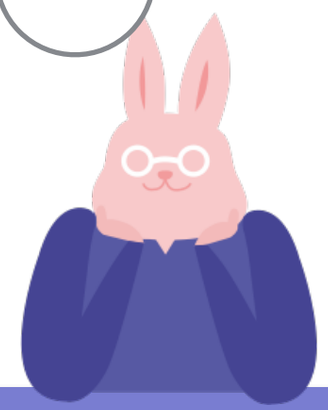
너비 우선 탐색 (BFS)

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



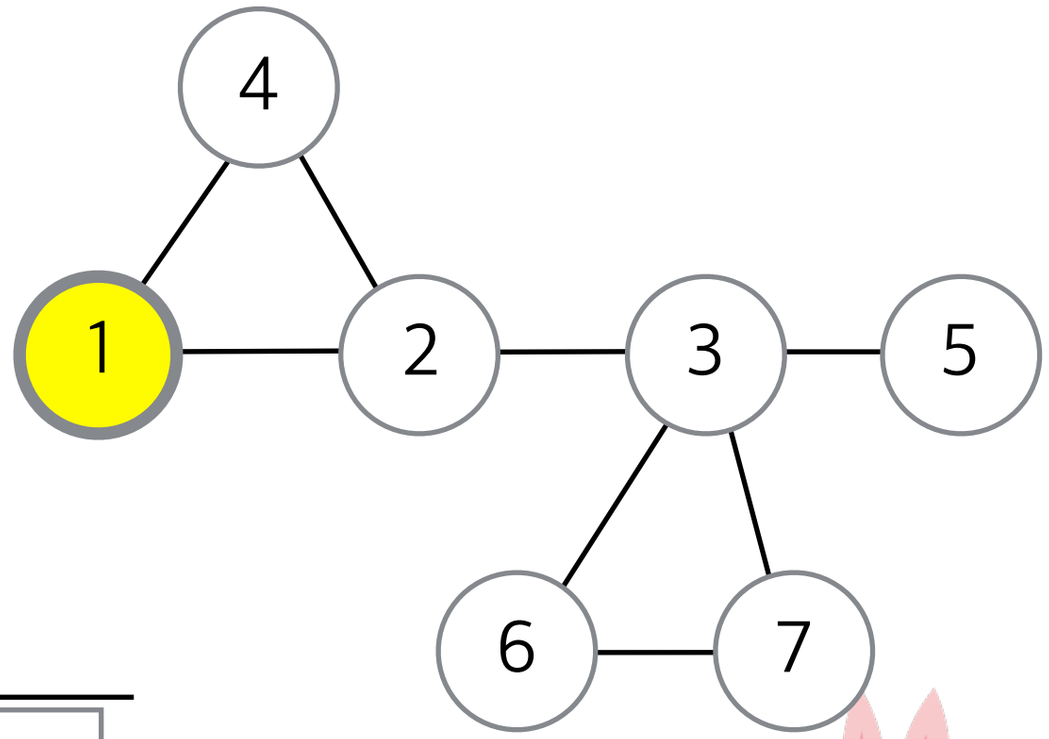
이 상태로 시작!

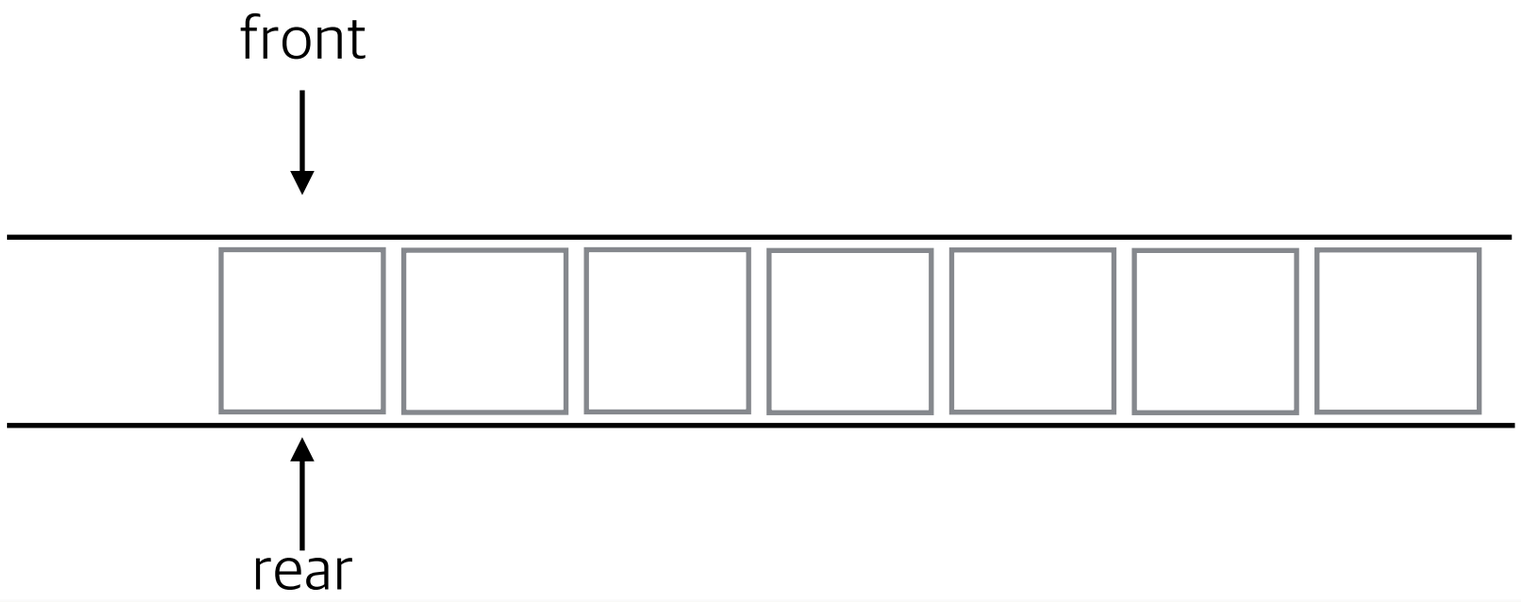




너비 우선 탐색 (BFS)

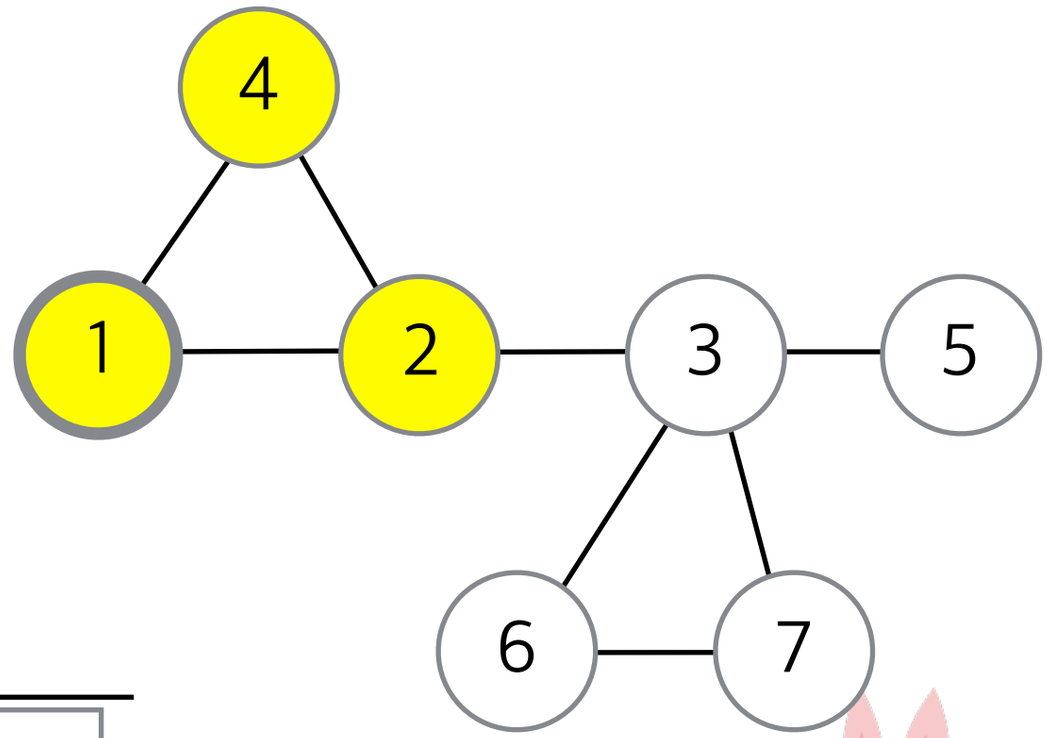
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다
  1. **Node 선택**
  2. 인접한 노드 색칠

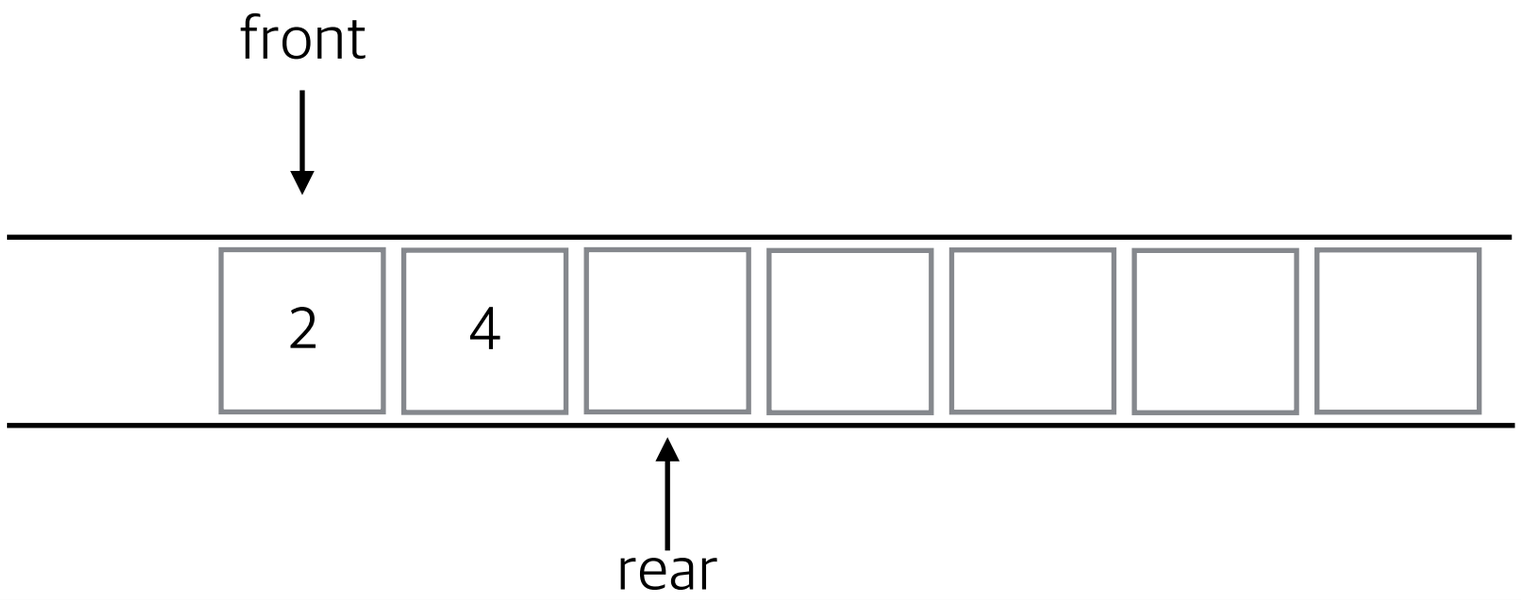




너비 우선 탐색 (BFS)

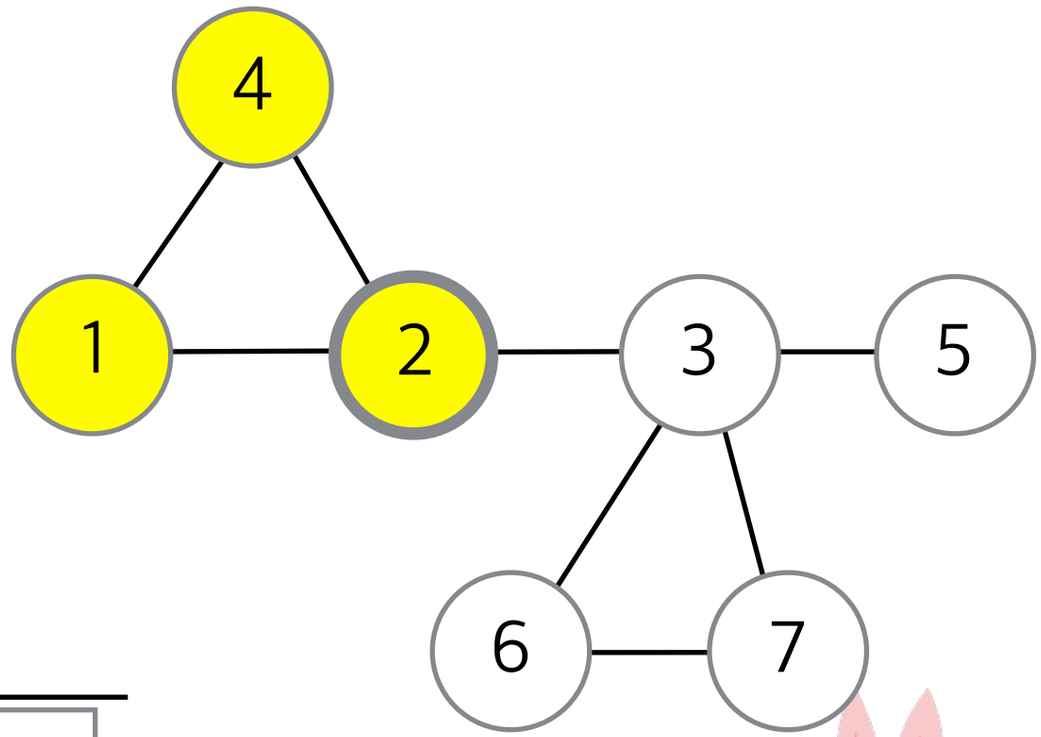
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다
  1. Node 선택
  2. **인접한 노드 색칠**

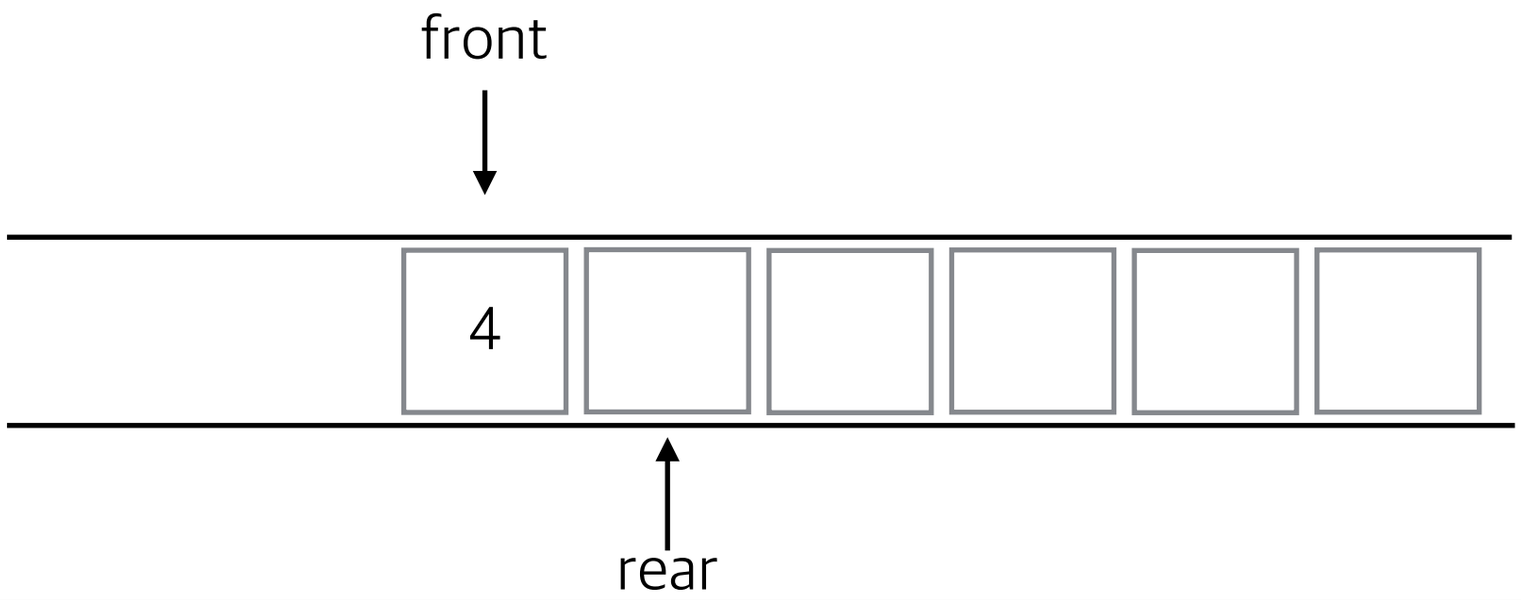


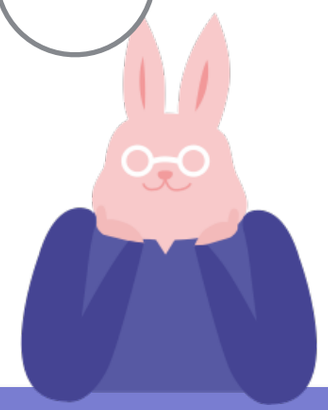


너비 우선 탐색 (BFS)

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

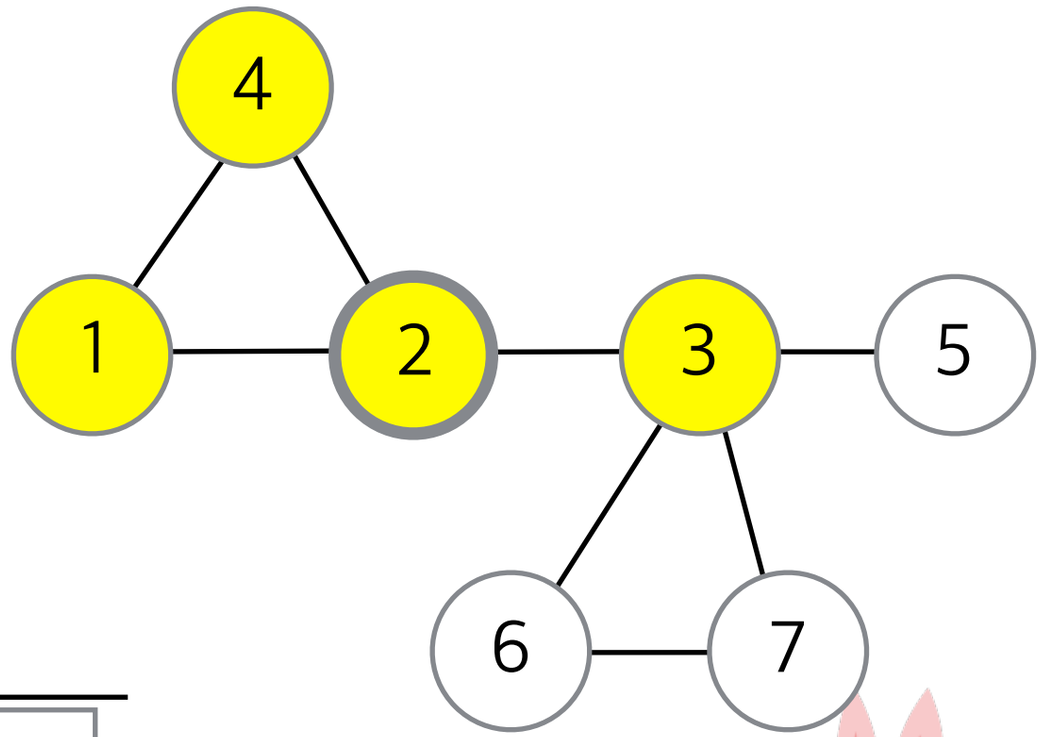


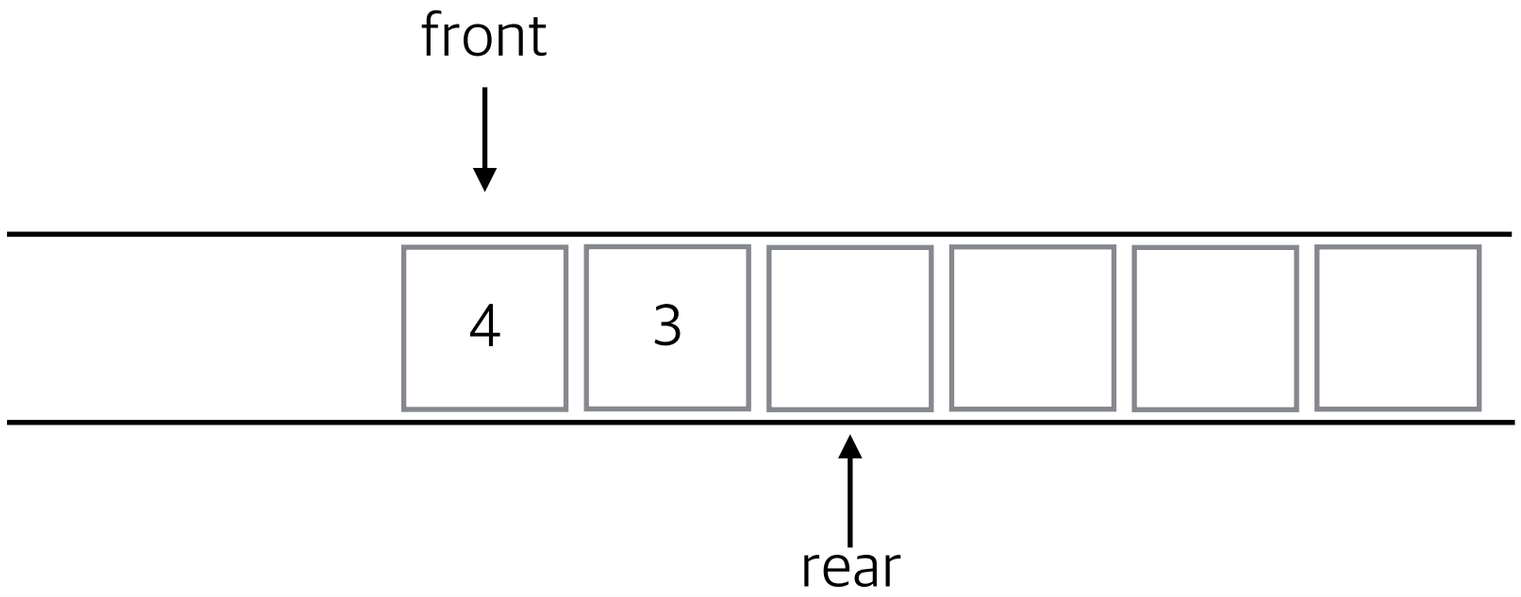




너비 우선 탐색 (BFS)

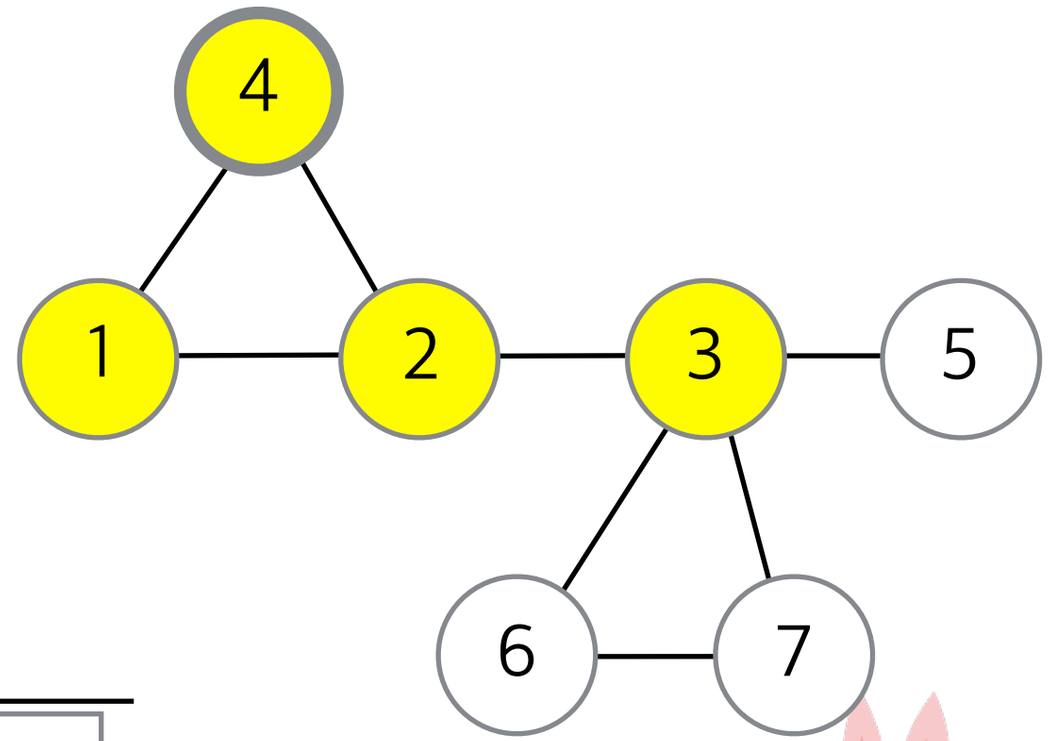
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다
  1. Node 선택
  2. **인접한 노드 색칠**

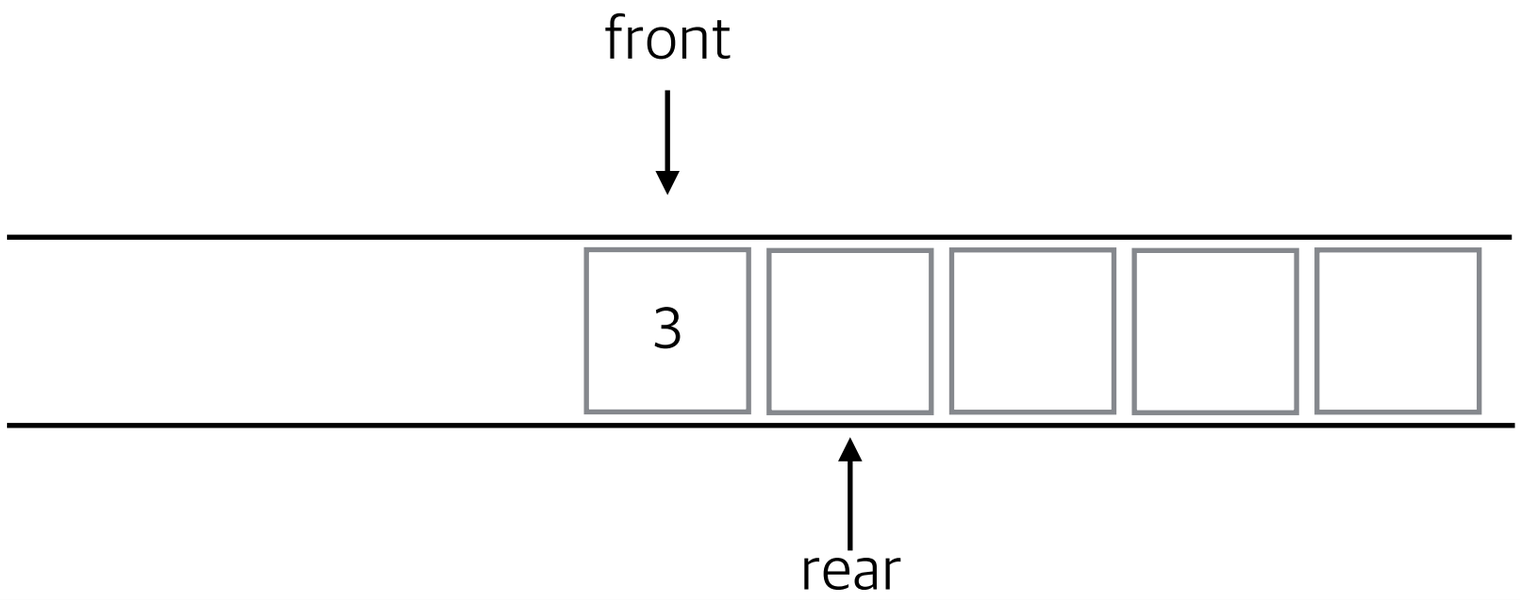


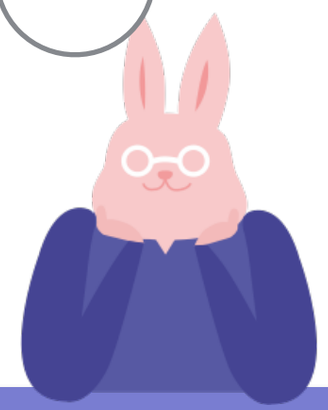


너비 우선 탐색 (BFS)

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

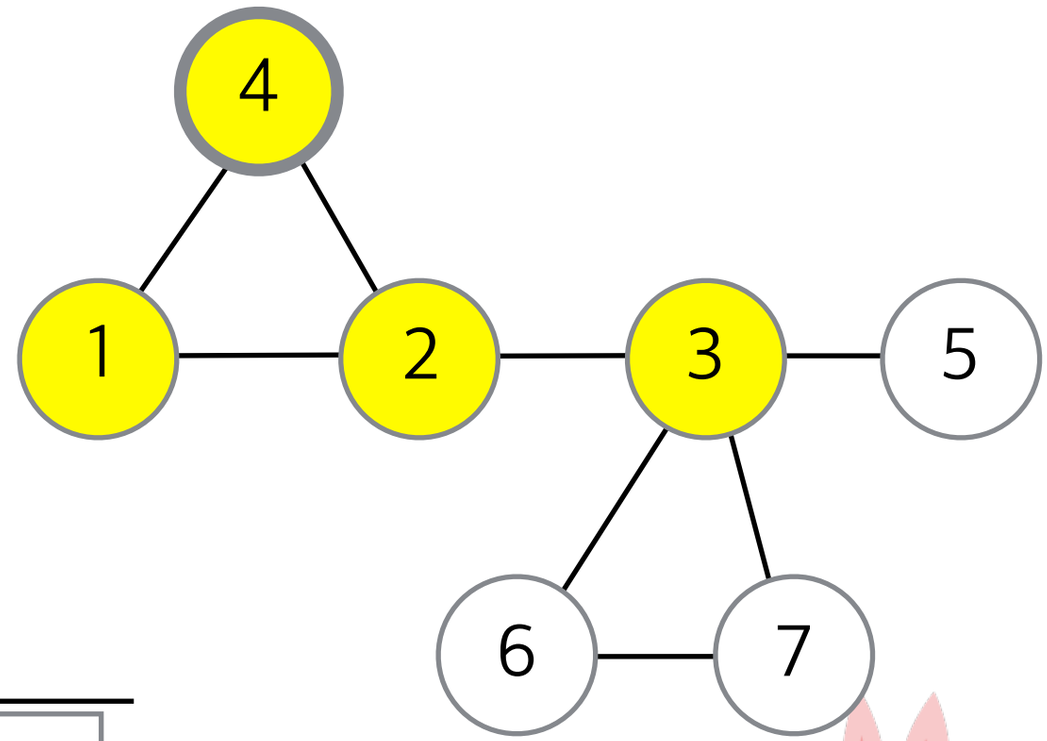


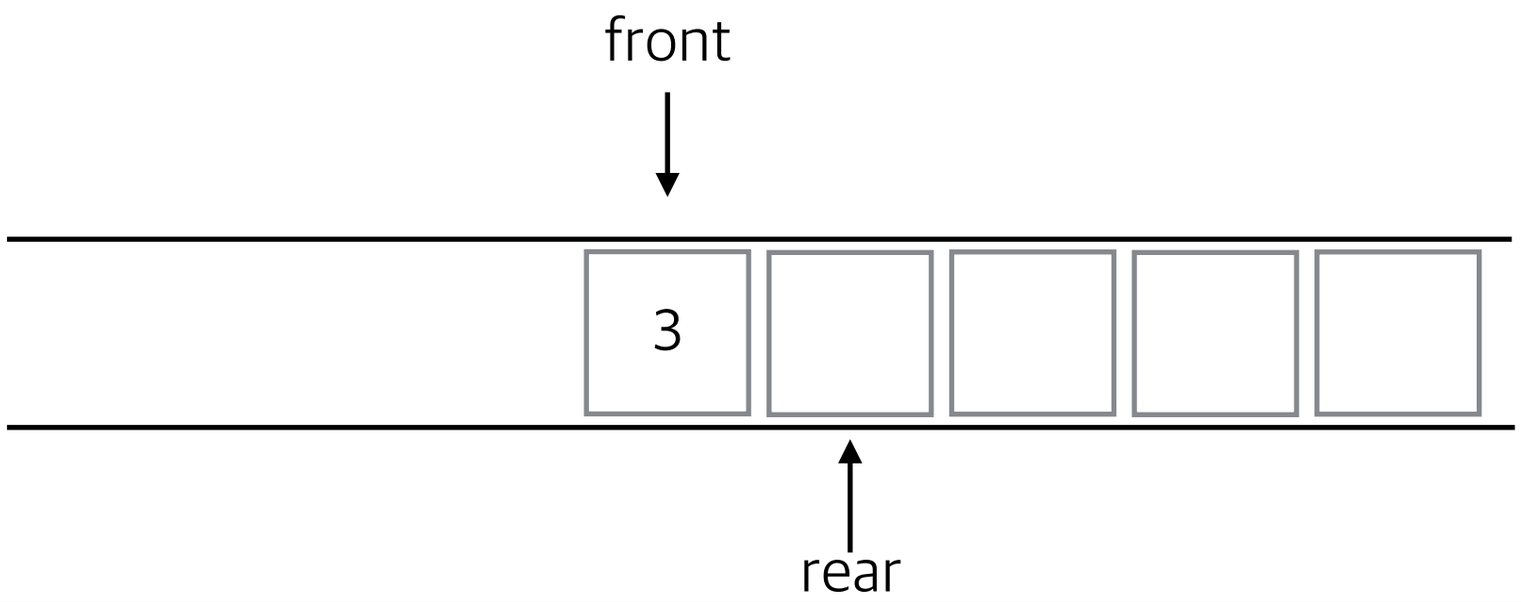




너비 우선 탐색 (BFS)

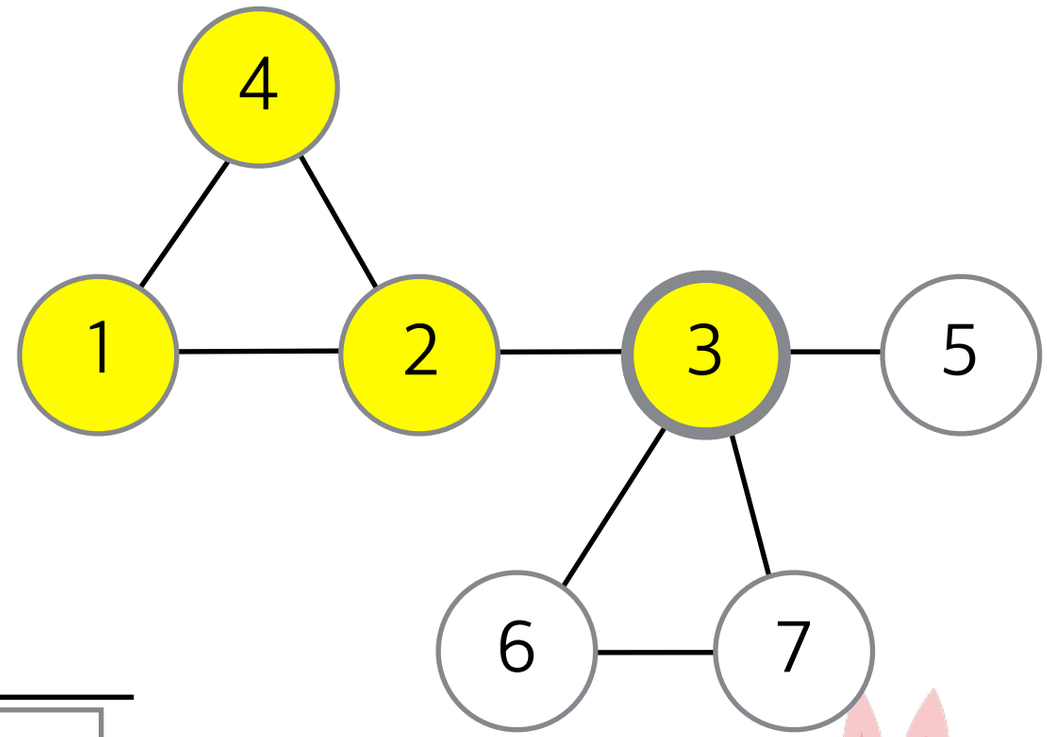
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다
  1. Node 선택
  2. **인접한 노드 색칠**

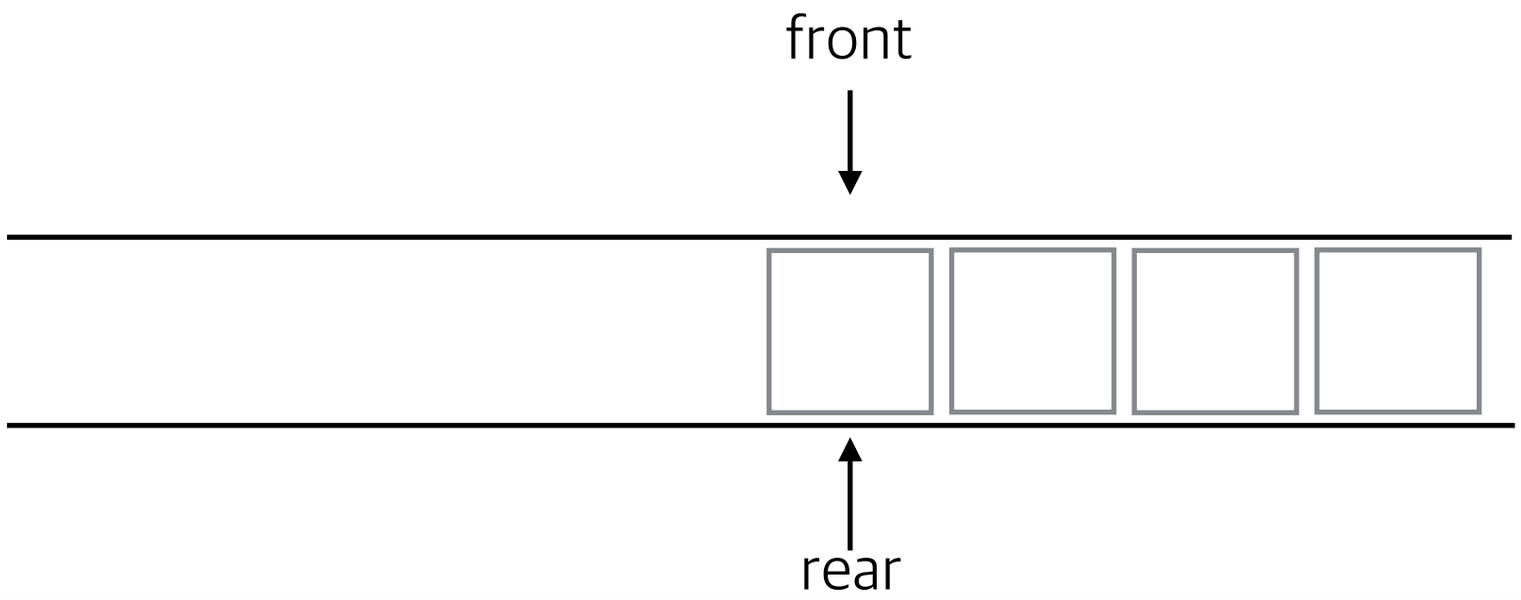


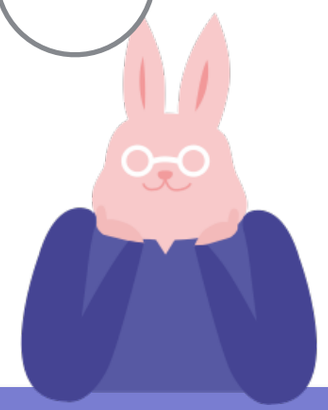


너비 우선 탐색 (BFS)

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

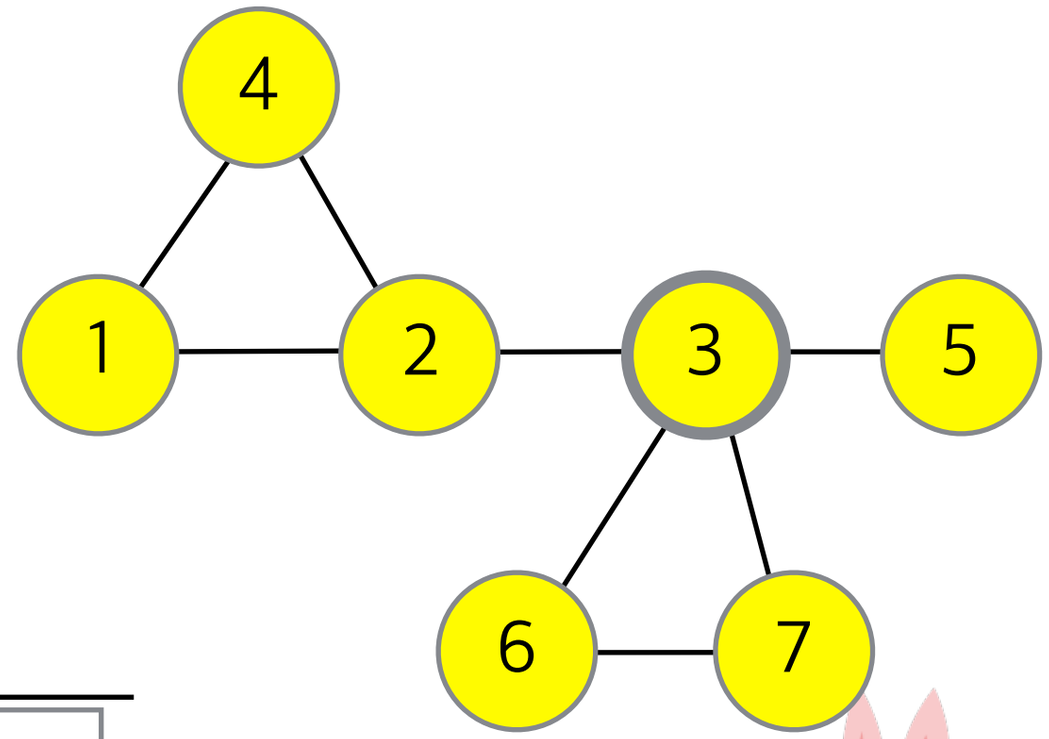


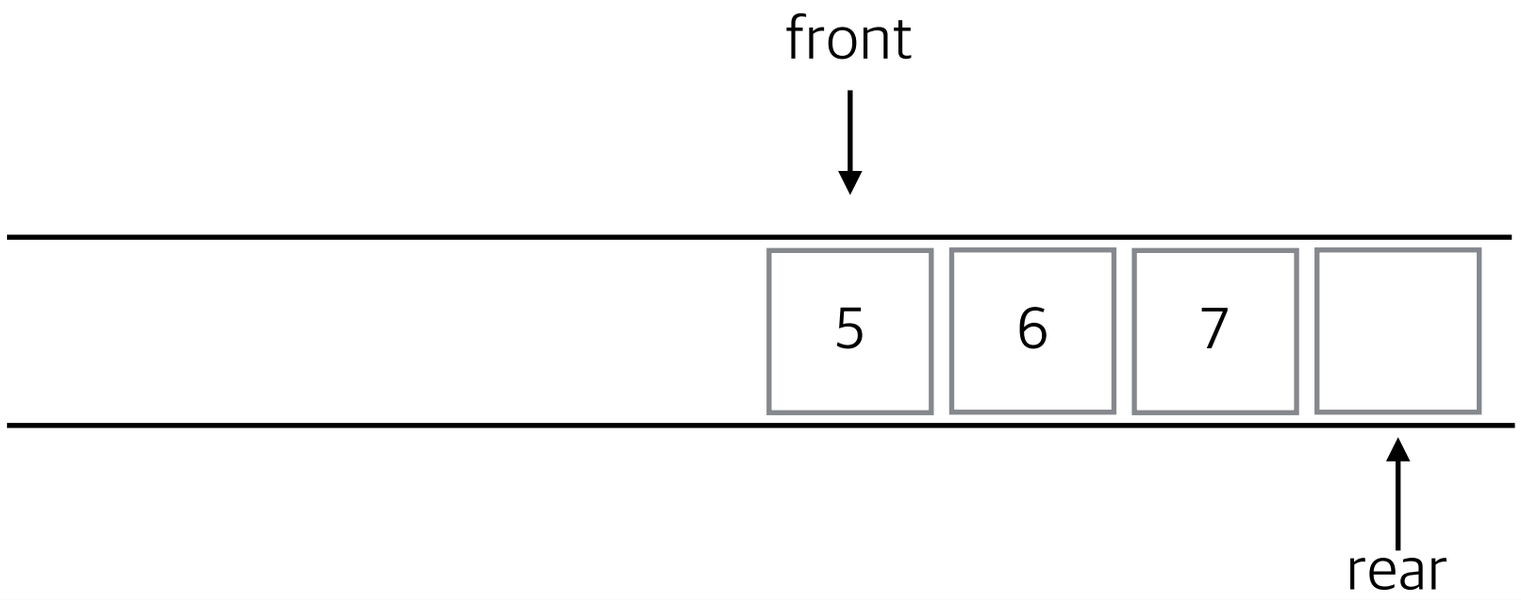




너비 우선 탐색 (BFS)

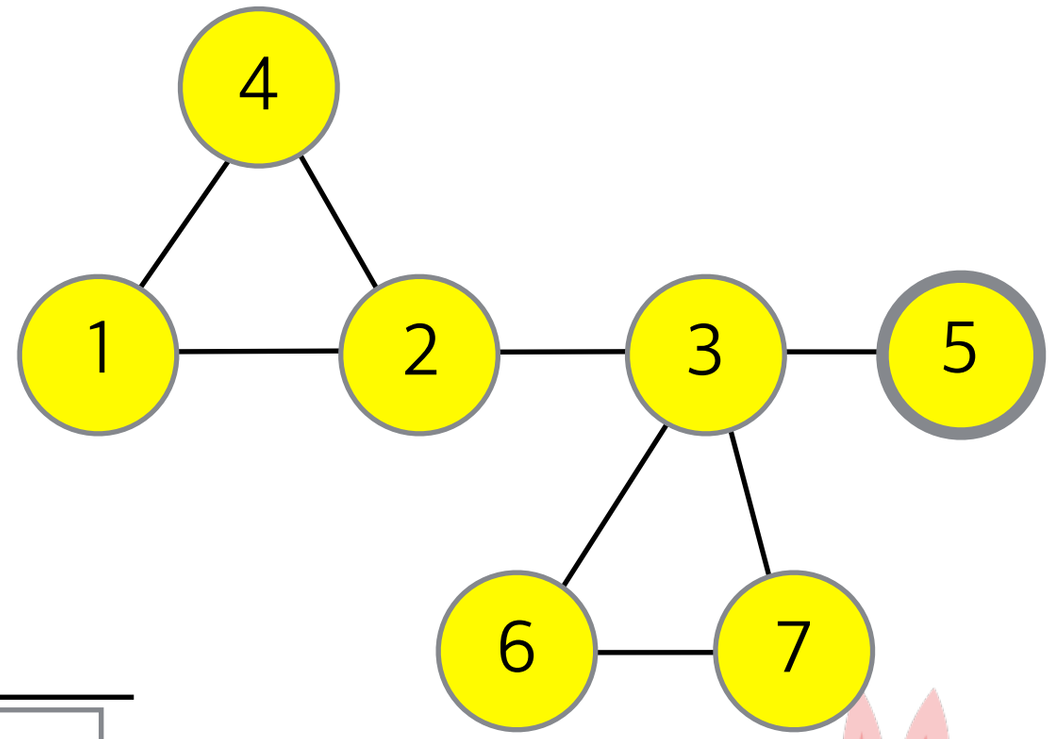
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다
  1. Node 선택
  2. **인접한 노드 색칠**

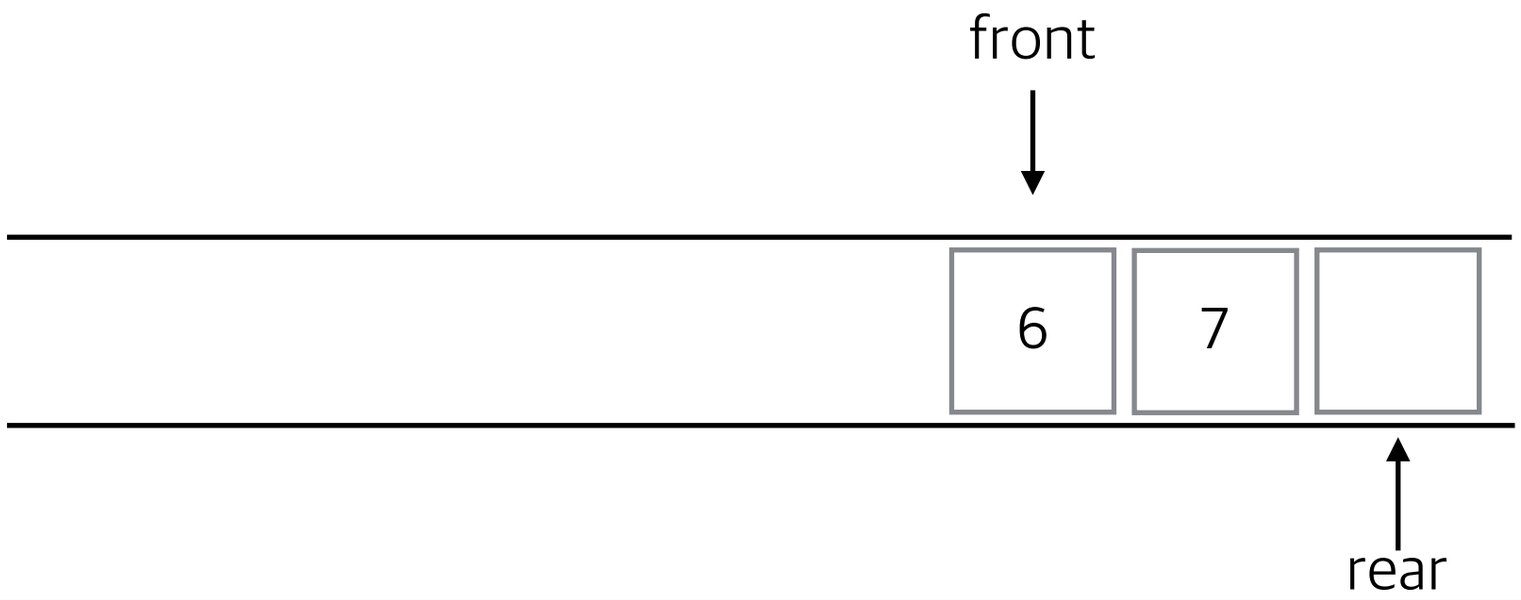


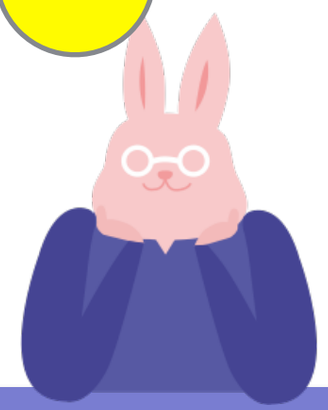


너비 우선 탐색 (BFS)

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

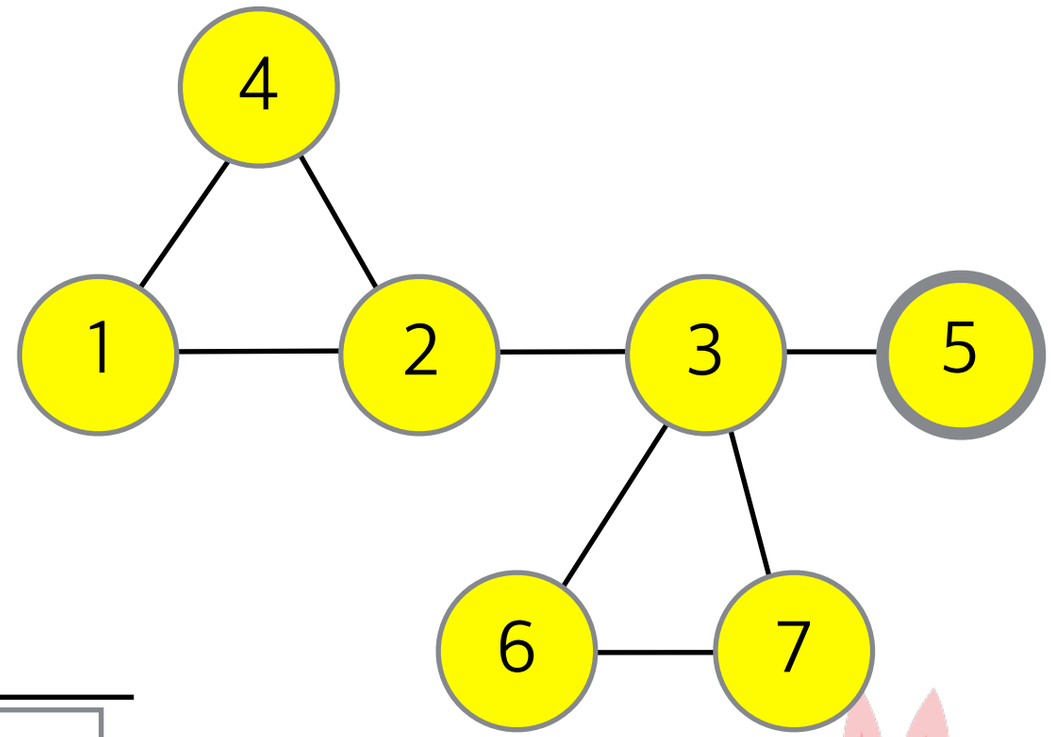


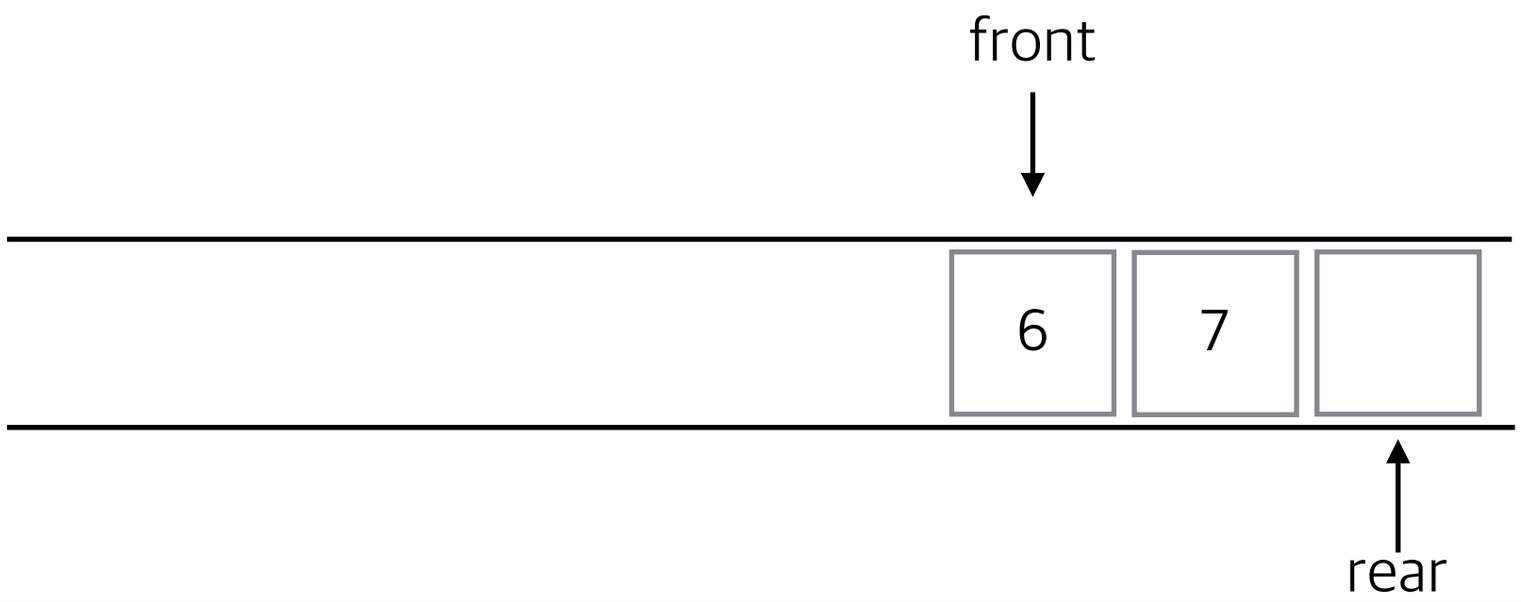


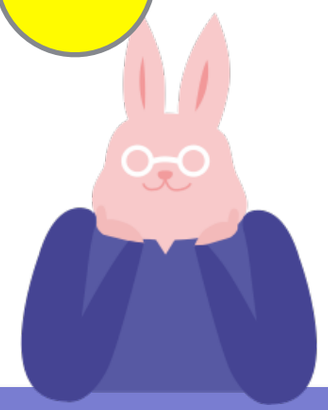


너비 우선 탐색 (BFS)

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

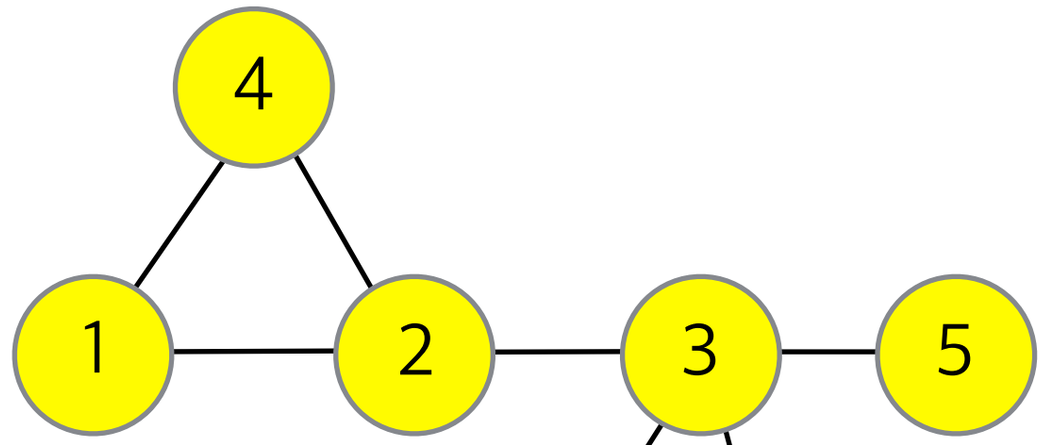


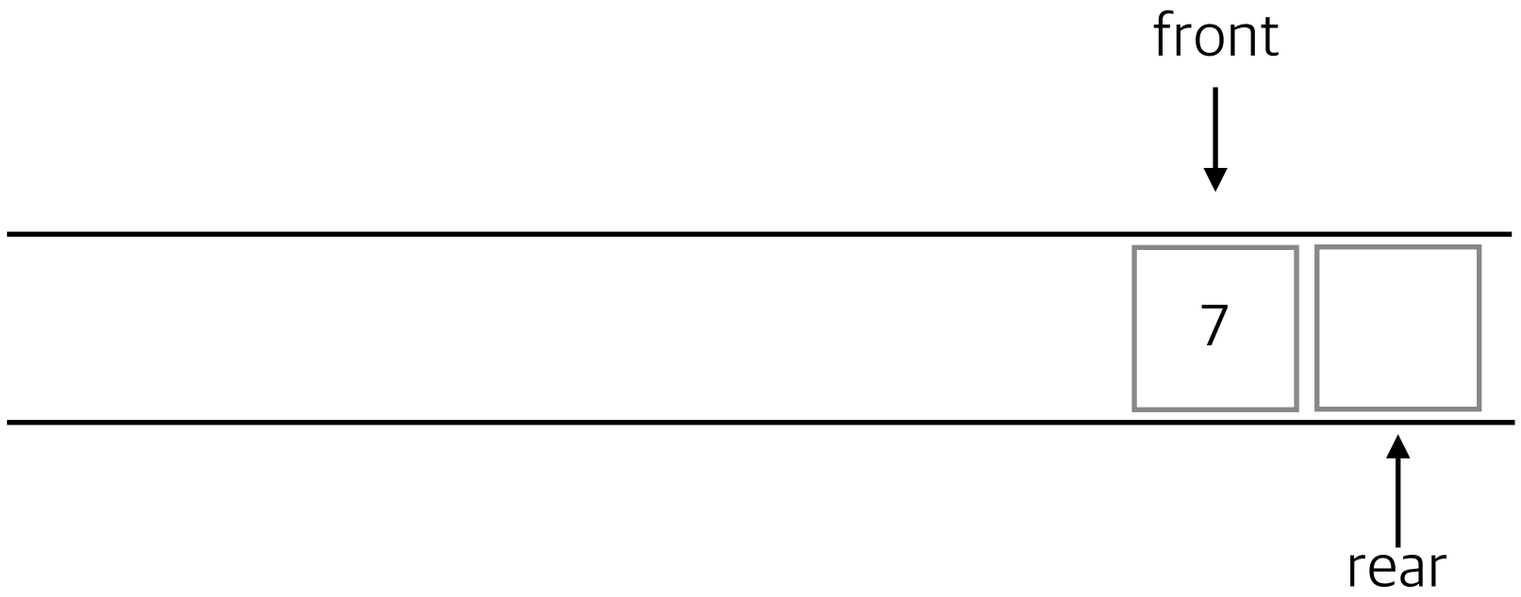


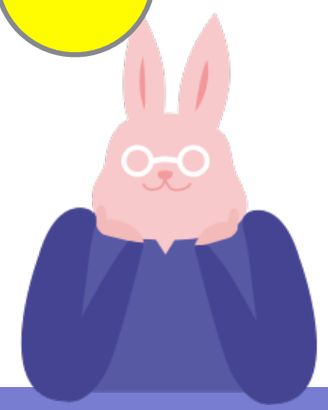


너비 우선 탐색 (BFS)

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

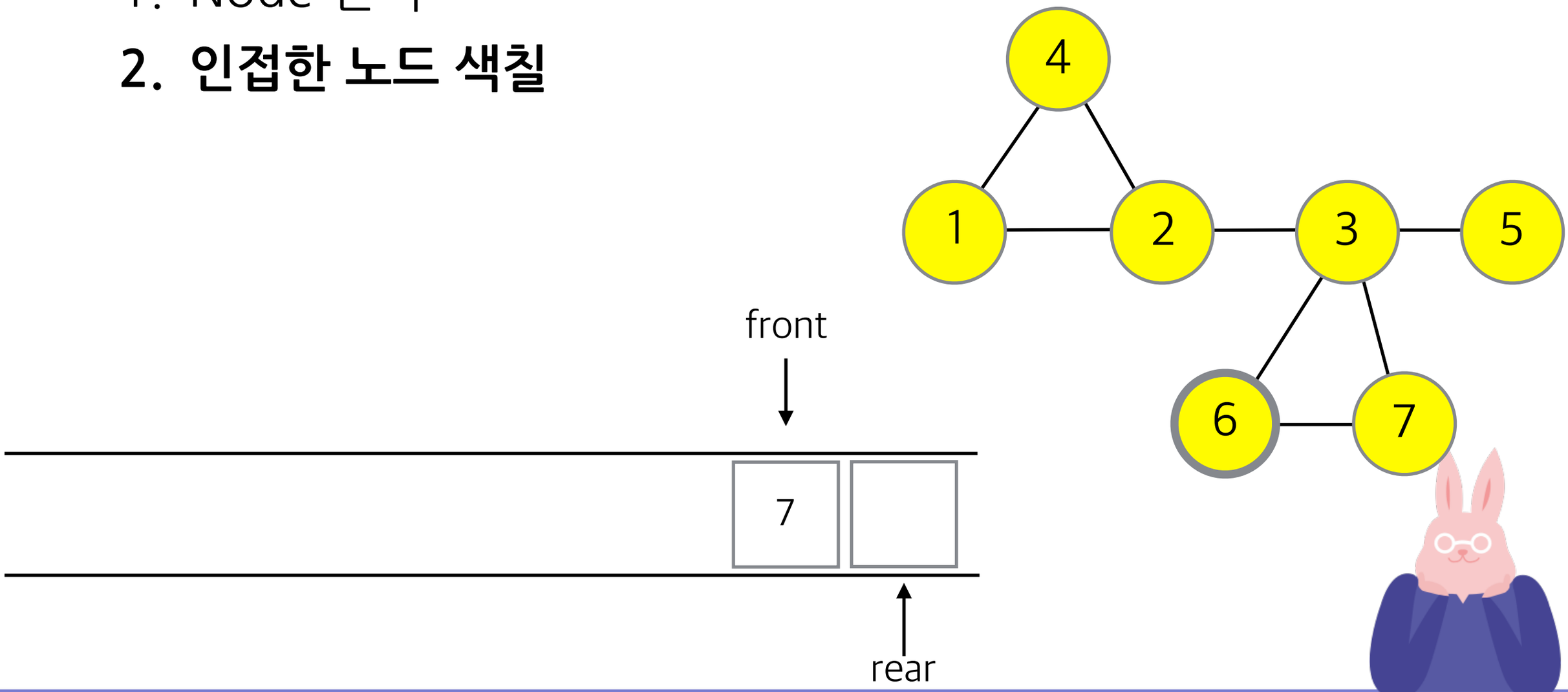






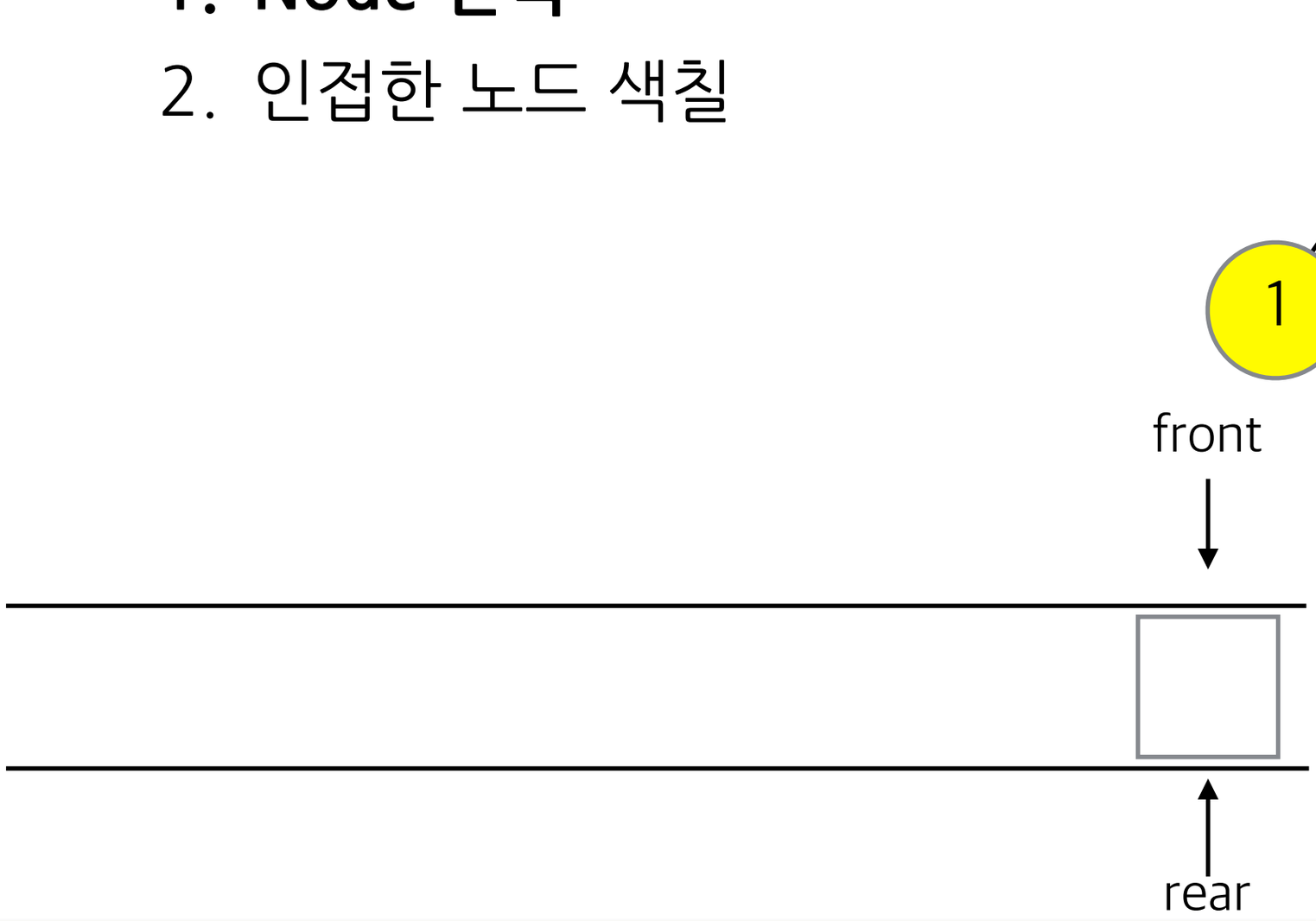
너비 우선 탐색 (BFS)

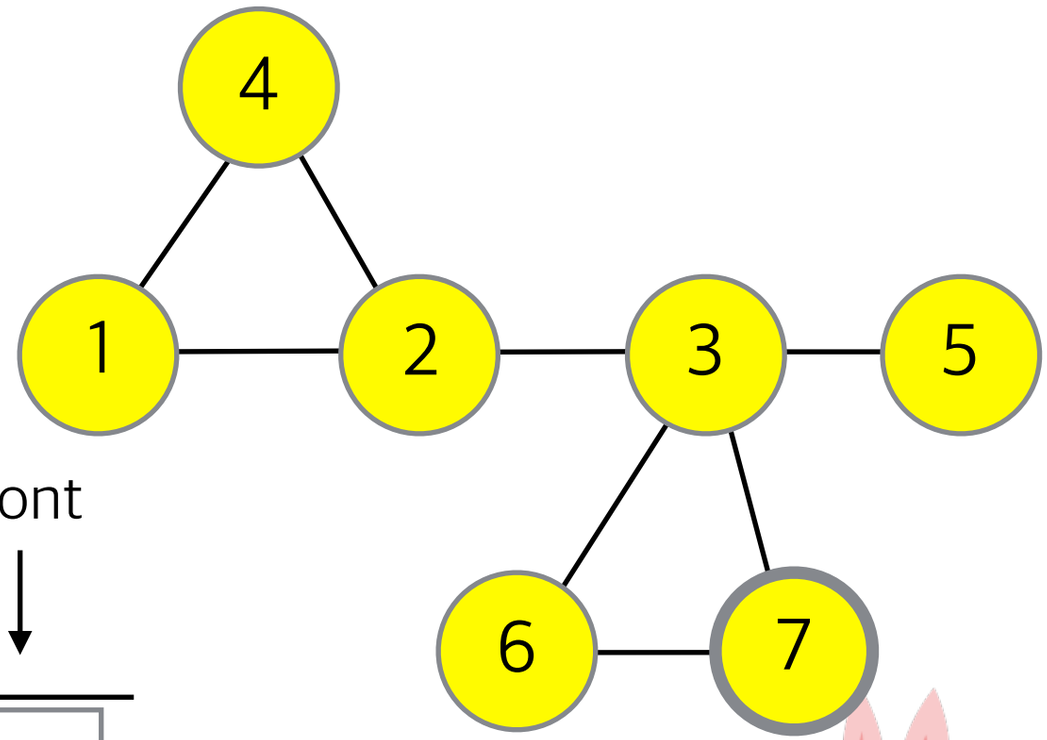
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다
  1. Node 선택
  2. **인접한 노드 색칠**



너비 우선 탐색 (BFS)

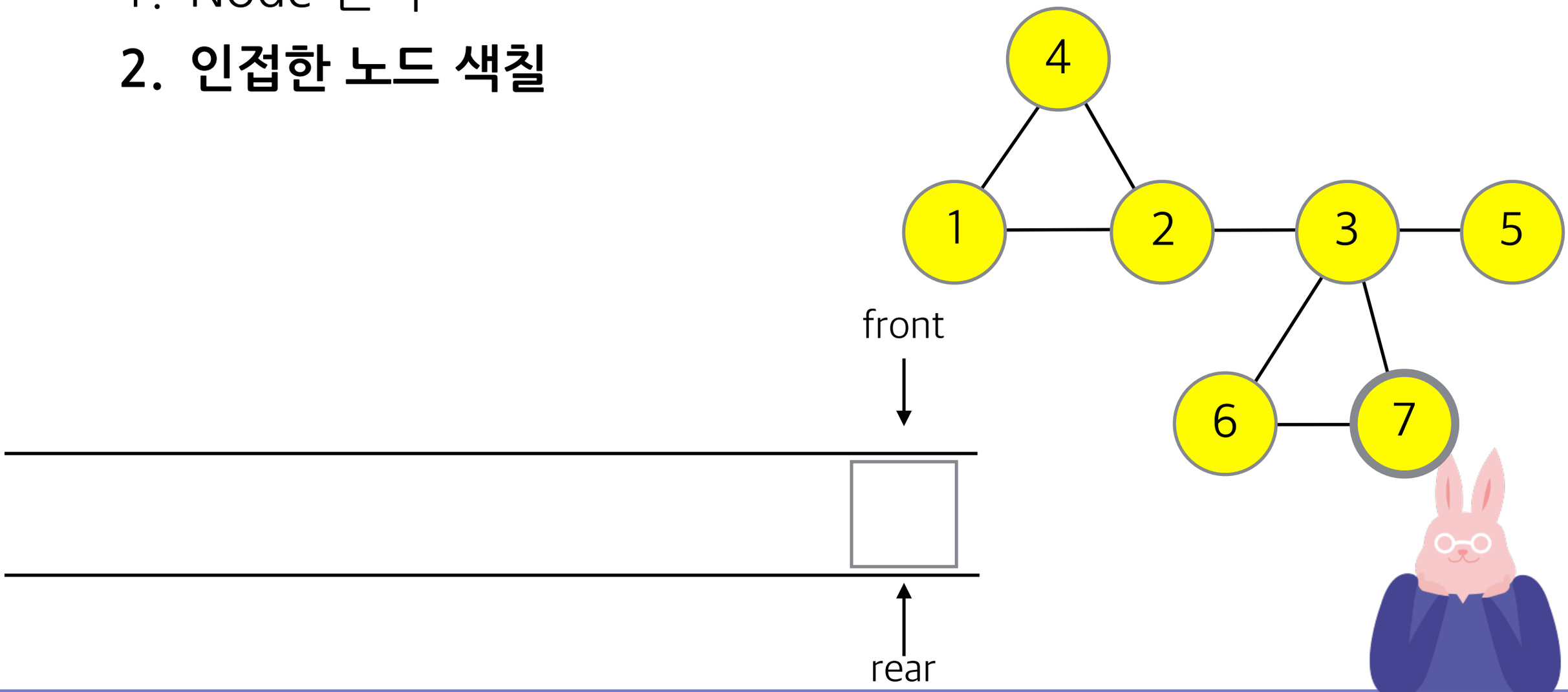
- 인접한 노드들을 우선 모두 색칠해 두고 뻗어나간다
  1. **Node 선택**
  2. 인접한 노드 색칠





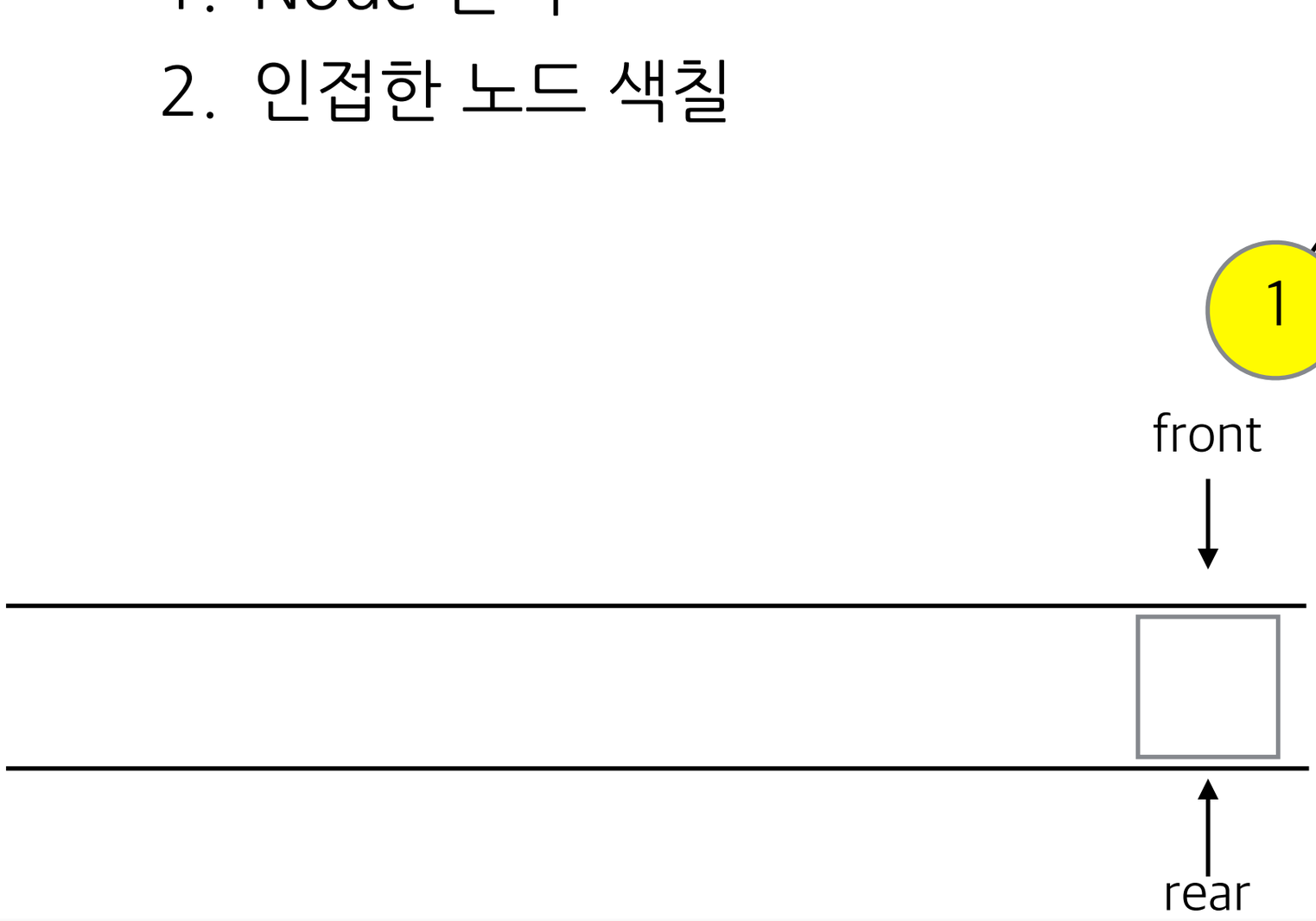
너비 우선 탐색 (BFS)

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



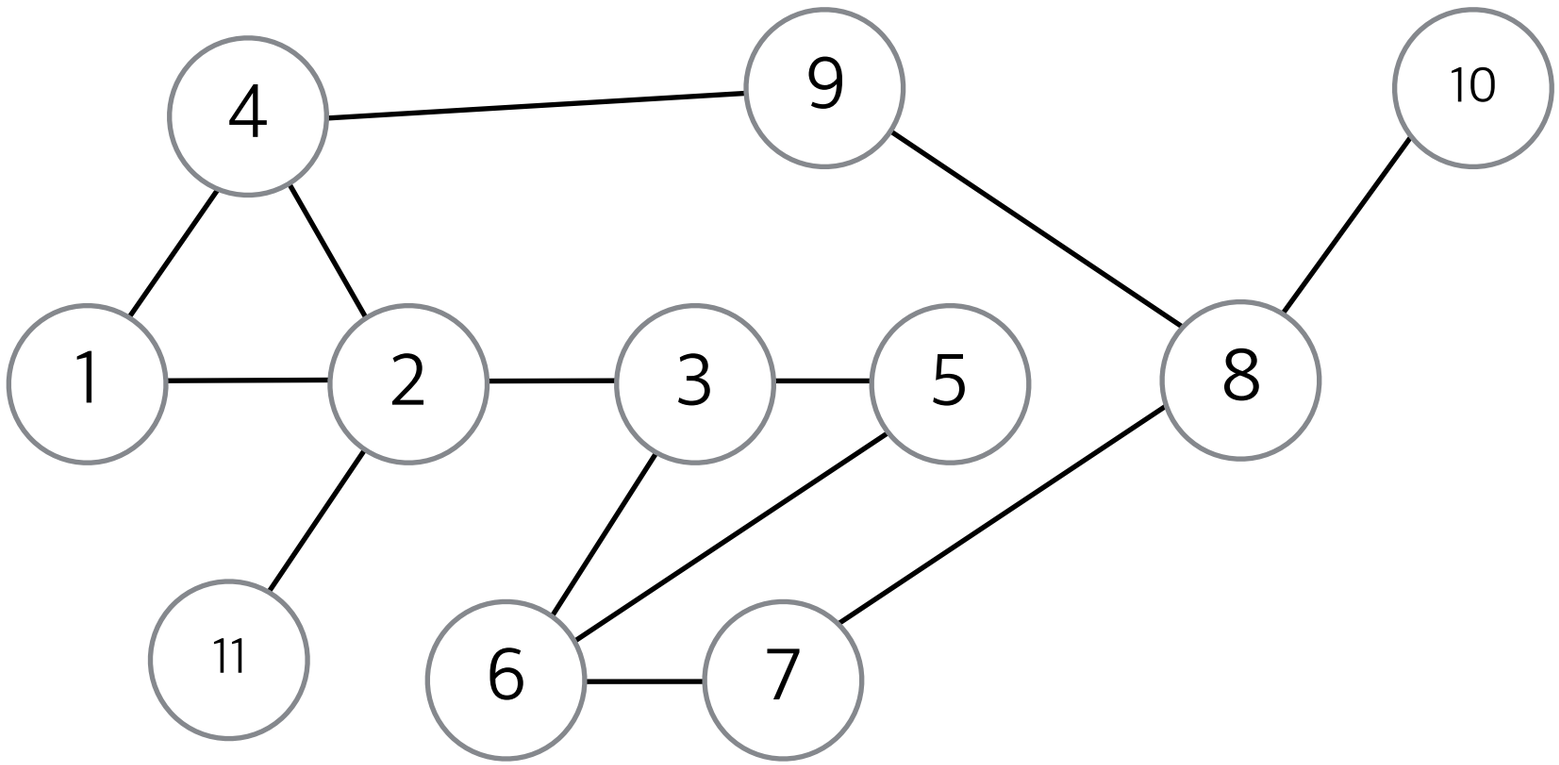
너비 우선 탐색 (BFS)

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



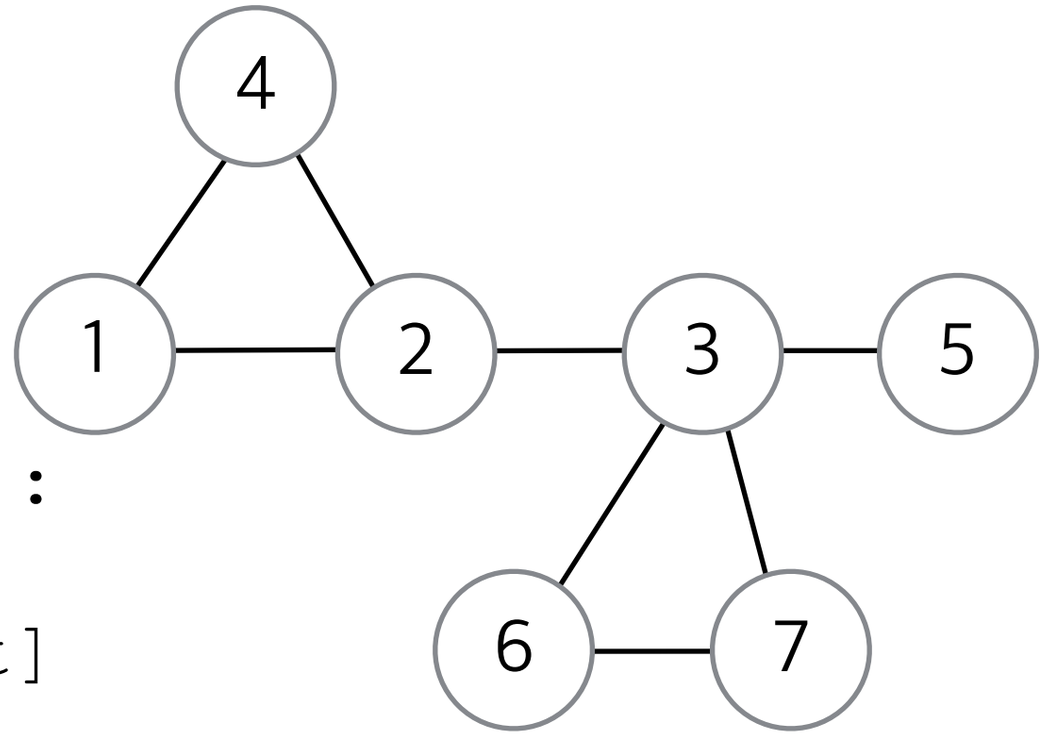
너비 우선 탐색 (BFS)

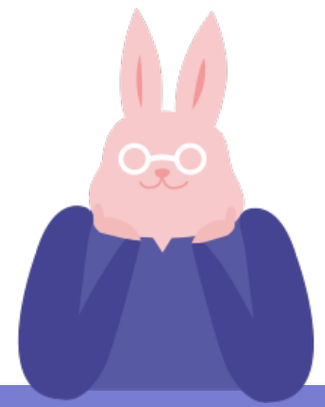
- 다음 그래프에 대하여 1을 시작으로 BFS한 결과는 ?
  - 단, 인접한 노드가 여러개일 경우 노드 번호가 작은 노드부터 방문



너비 우선 탐색 (BFS) 구현

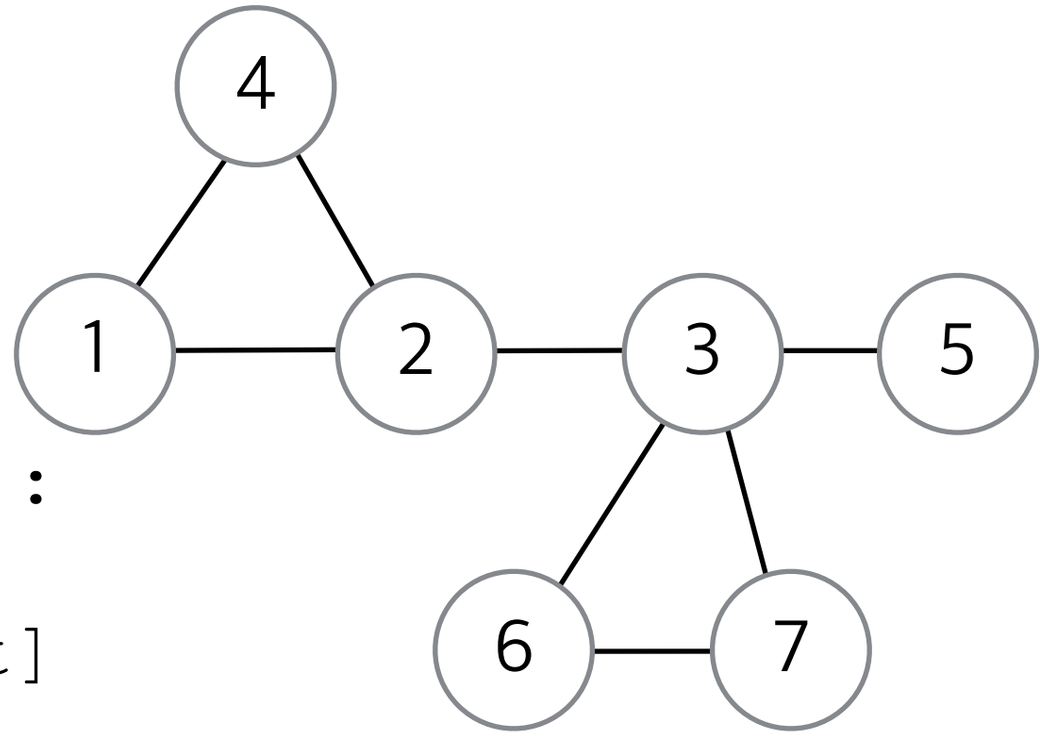
```
def BFS(graph, x, visited) :
    result = []
    visited[x] = True
    Queue.put(x)

    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
```

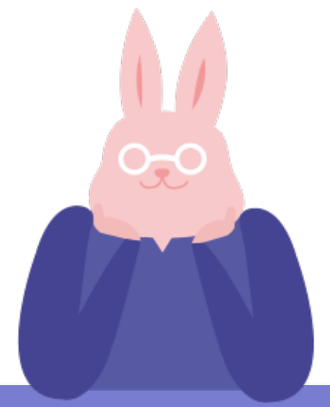




너비 우선 탐색 (BFS) 구현

```
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
```

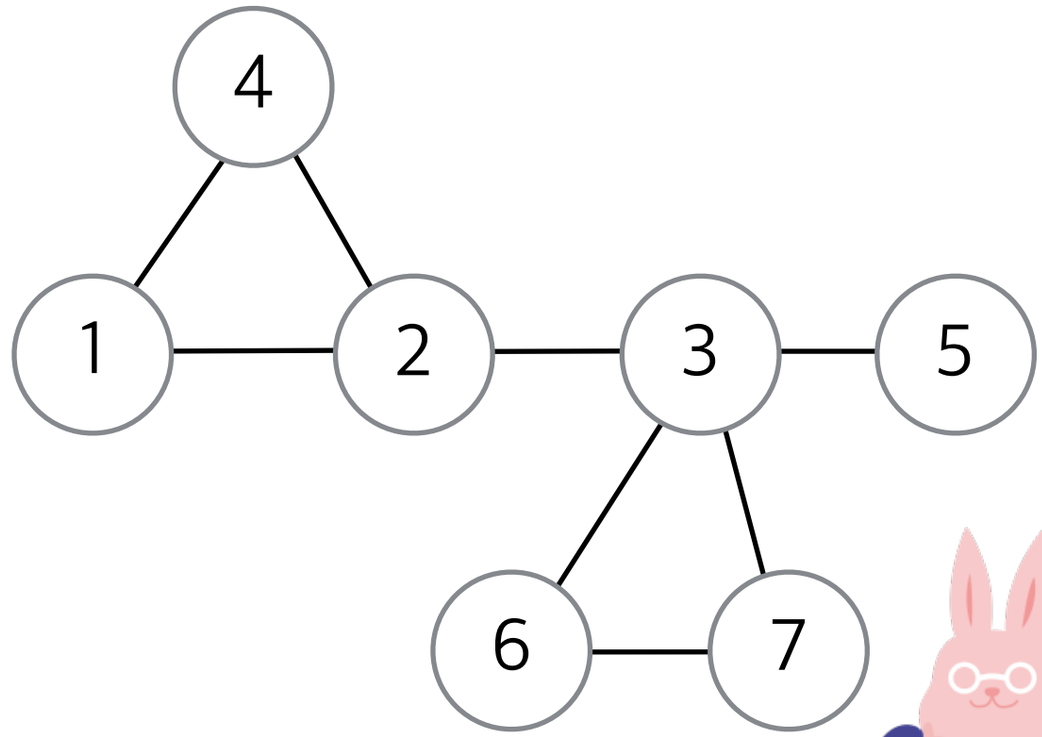




DFS와 BFS의 정확성 증명 및 시간복잡도

- 정확성 증명
  - 질문 : 모든 Node와 Edge를 방문하는가 ?

- 시간복잡도



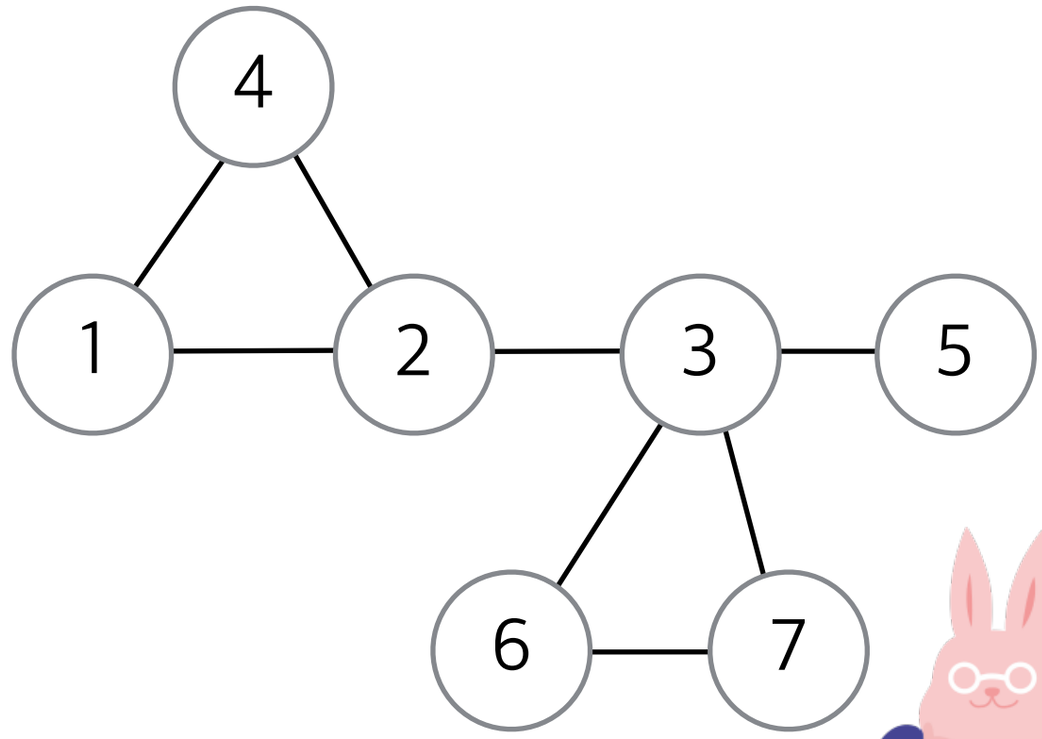
DFS와 BFS의 정확성 증명 및 시간복잡도

- 정확성 증명
  - 질문 : 모든 Node와 Edge를 방문하는가 ? **YES!**

- 시간복잡도
  - O(V+E)

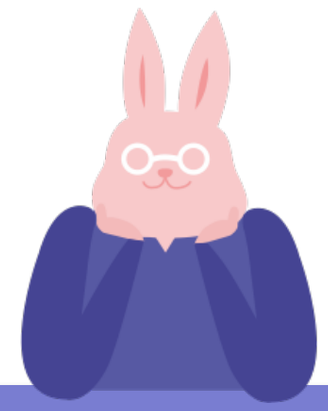
  V : Node의 갯수

  E : Edge의 개수



DFS와 BFS 마무리

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



[활동문제 4] DFS와 BFS

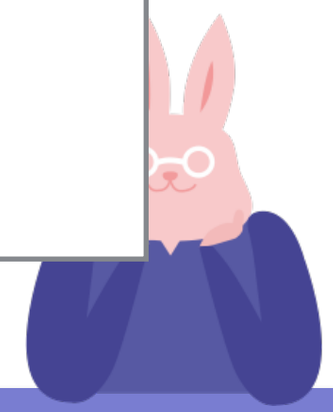
- DFS와 BFS한 결과를 각각 출력

입력의 예

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

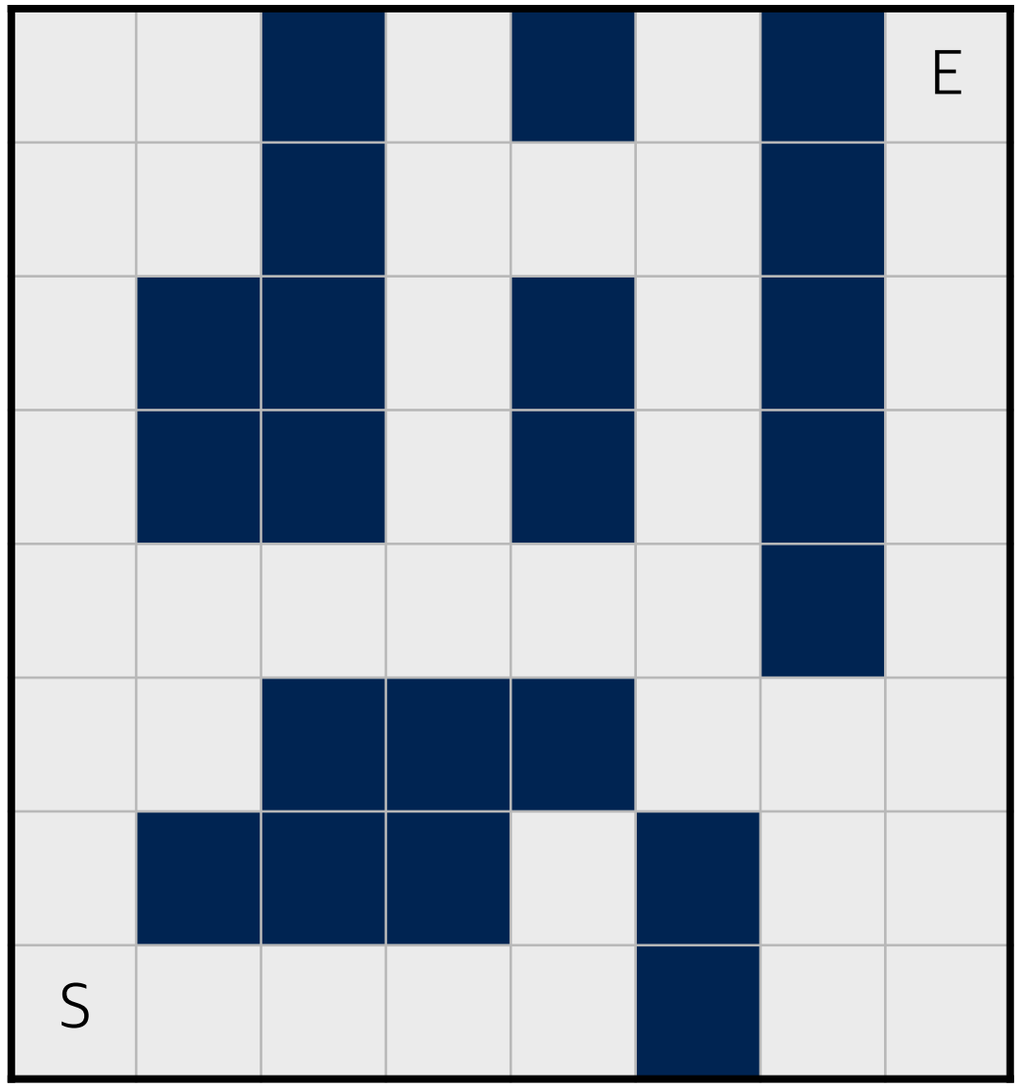
출력의 예

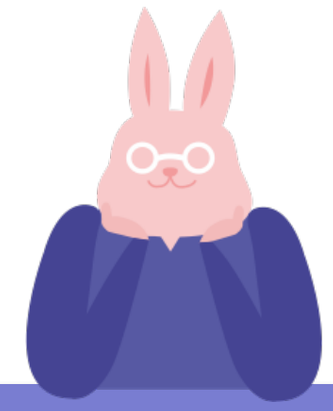
```
0 1 2 4 5 6 3
0 1 3 2 4 5 6
```



DFS와 BFS의 응용 : 미로찾기

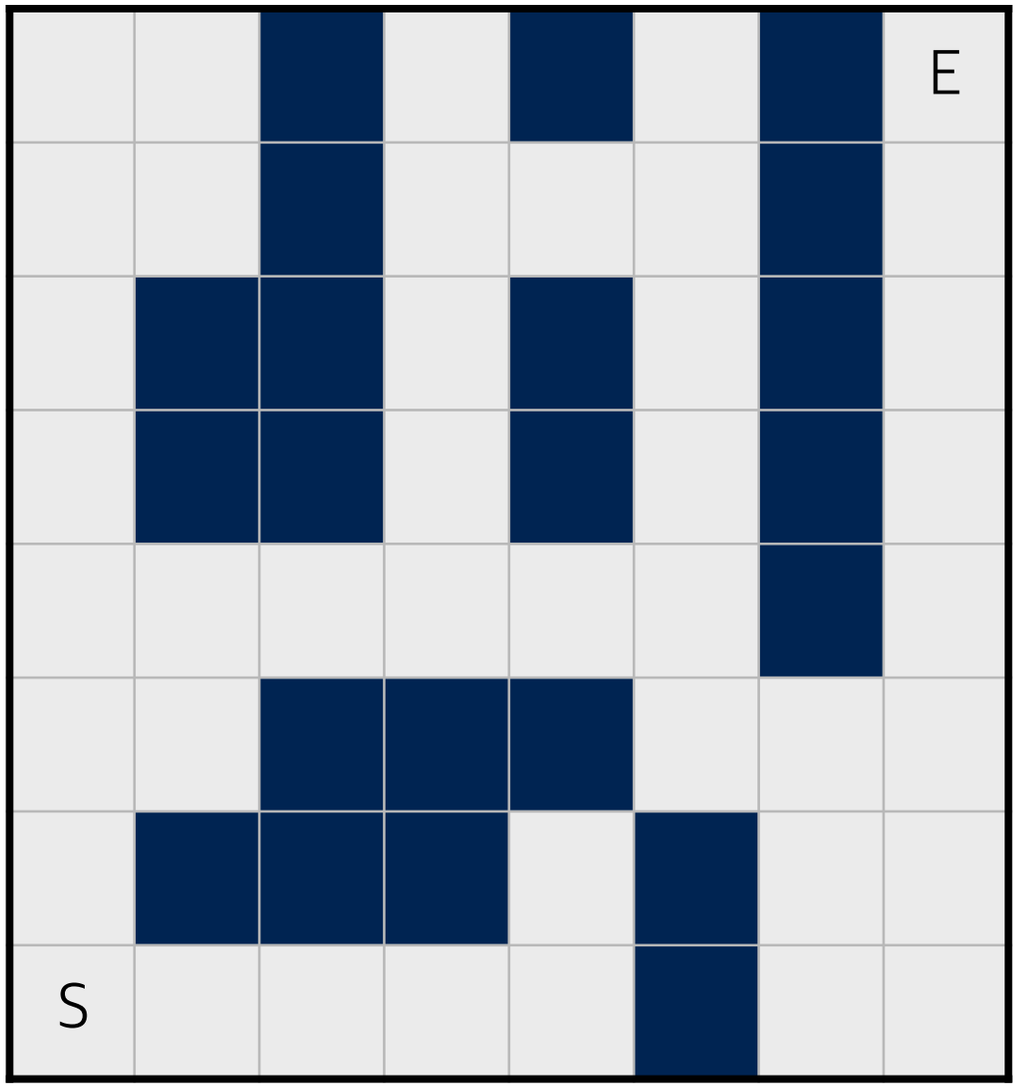
- 시작점에서 도착점까지 갈 때의 최단거리를 출력하라

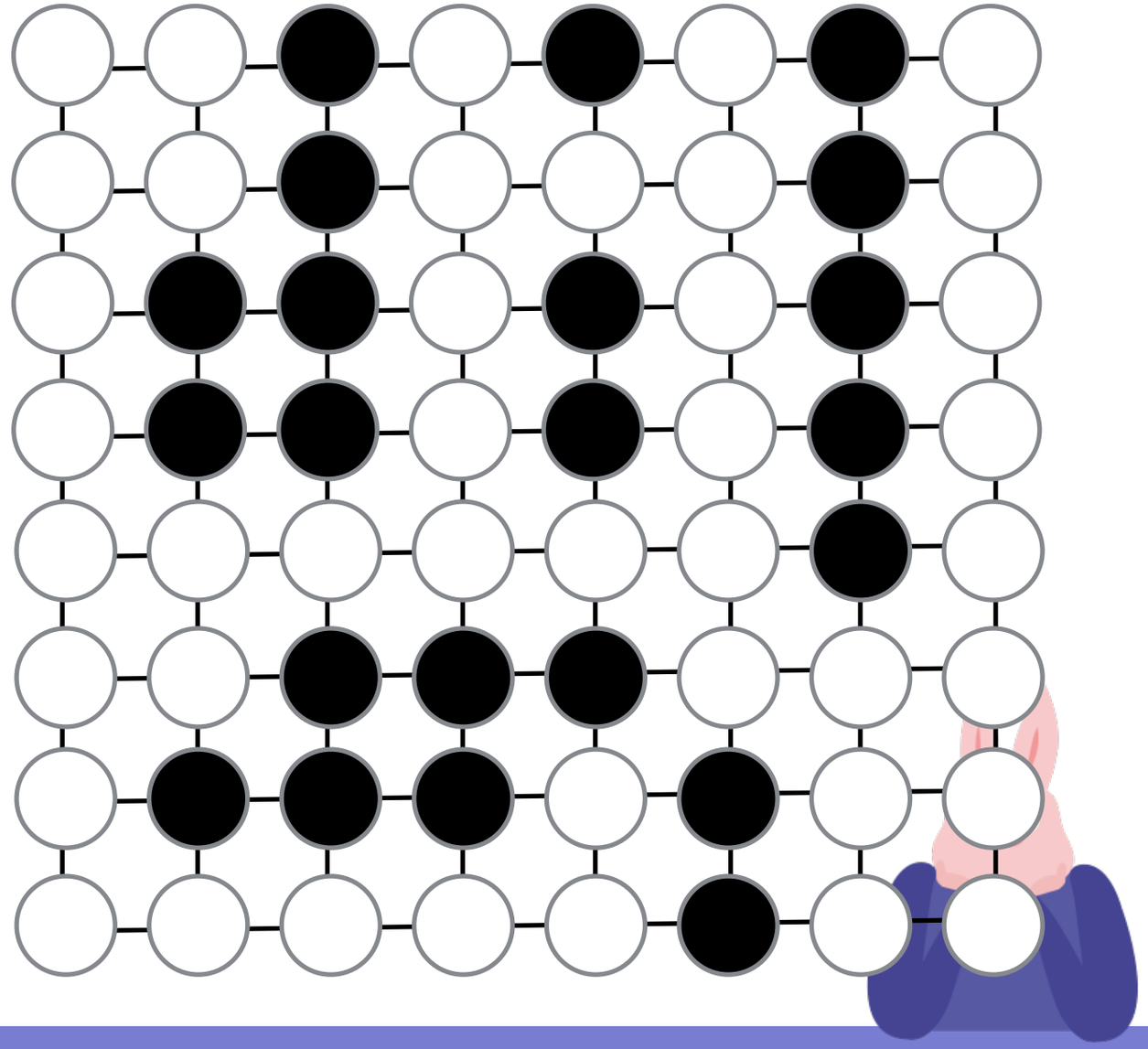




DFS와 BFS의 응용 : 미로찾기

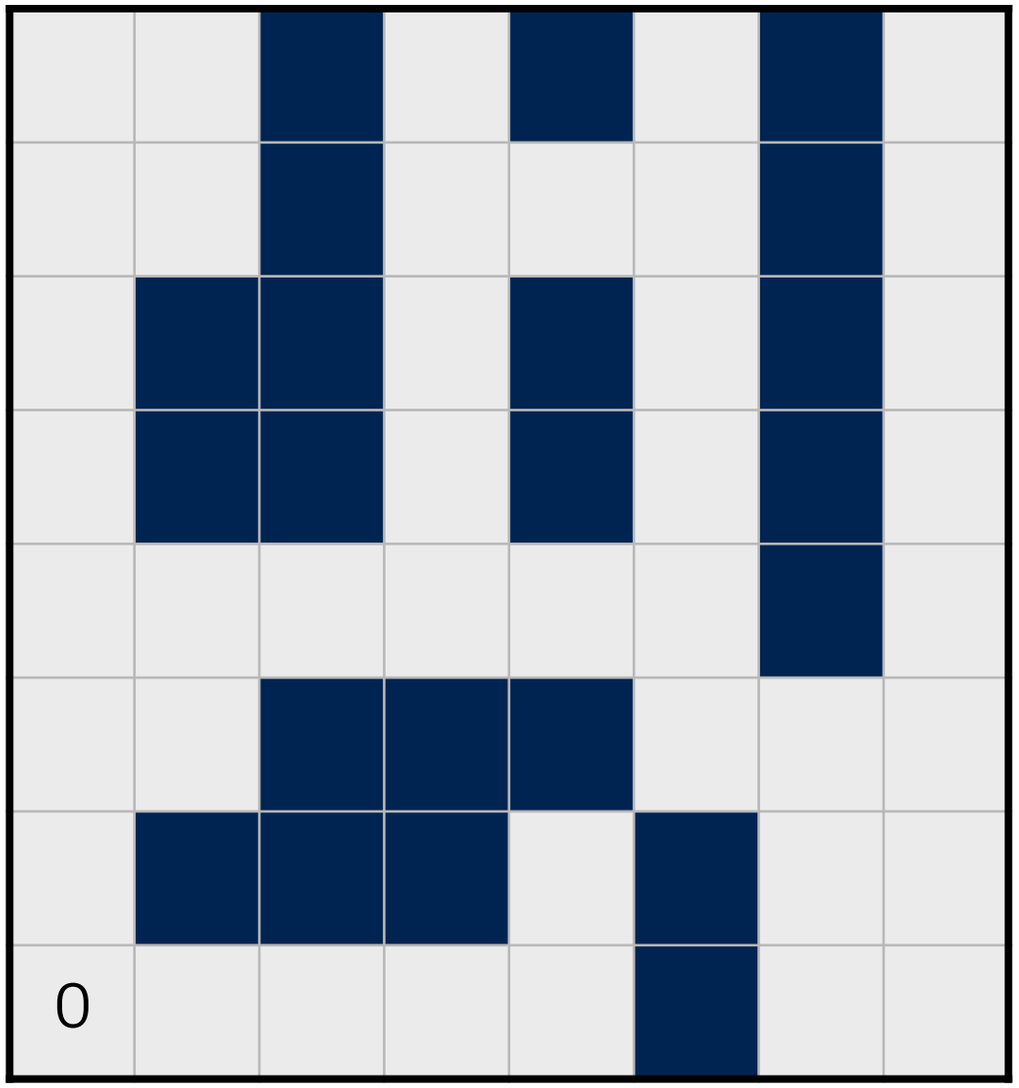
1. Graph로 변환

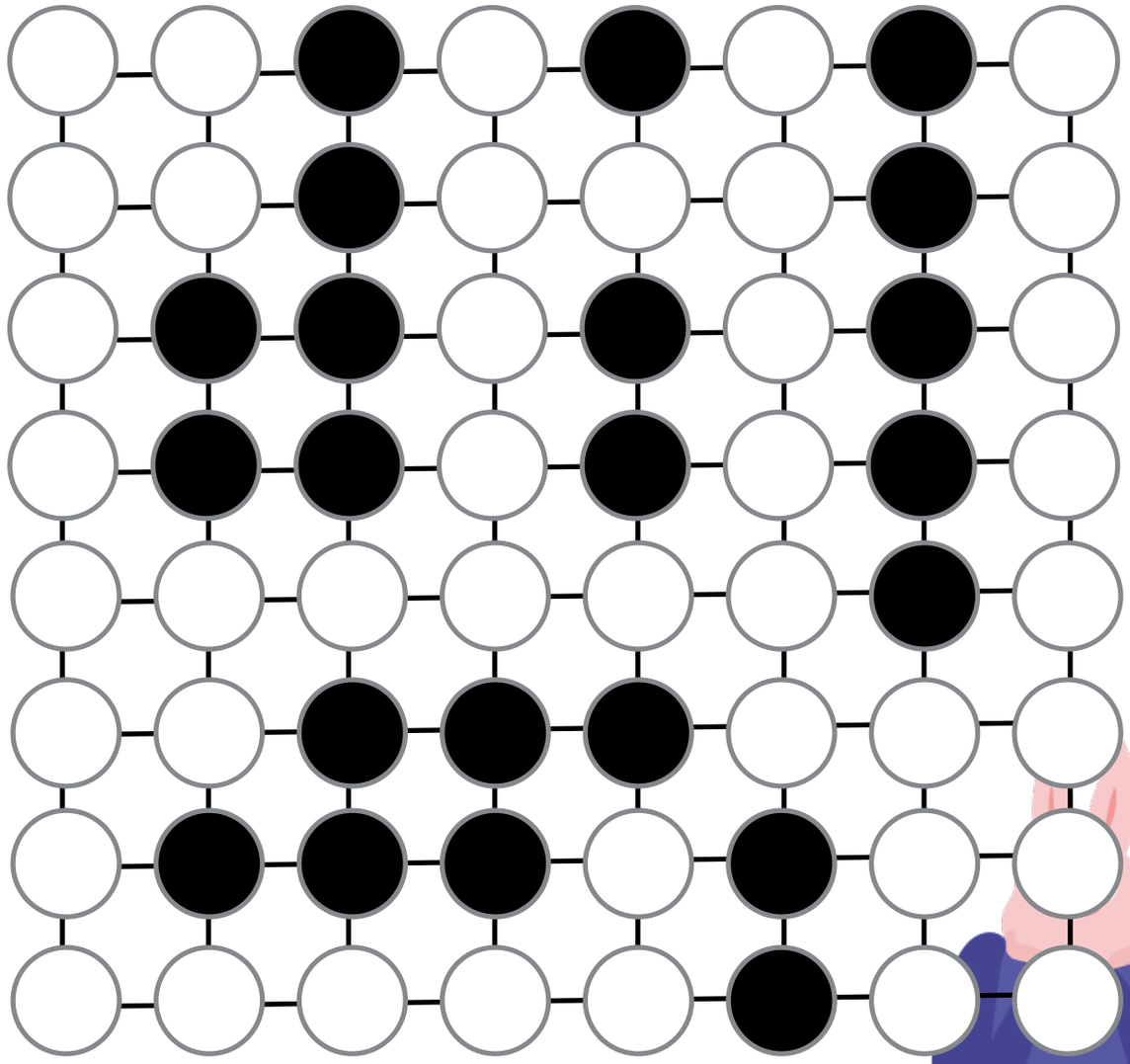




DFS와 BFS의 응용 : 미로찾기

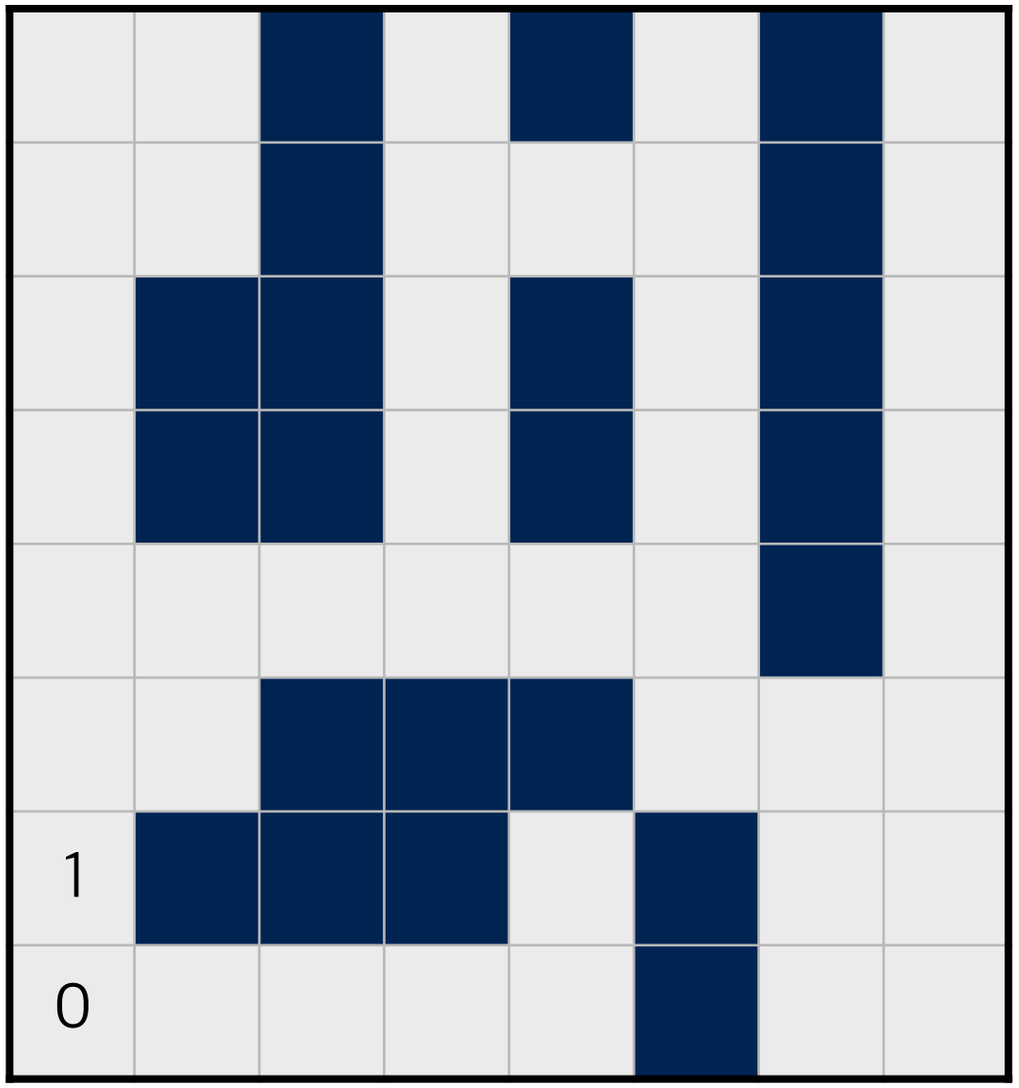
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

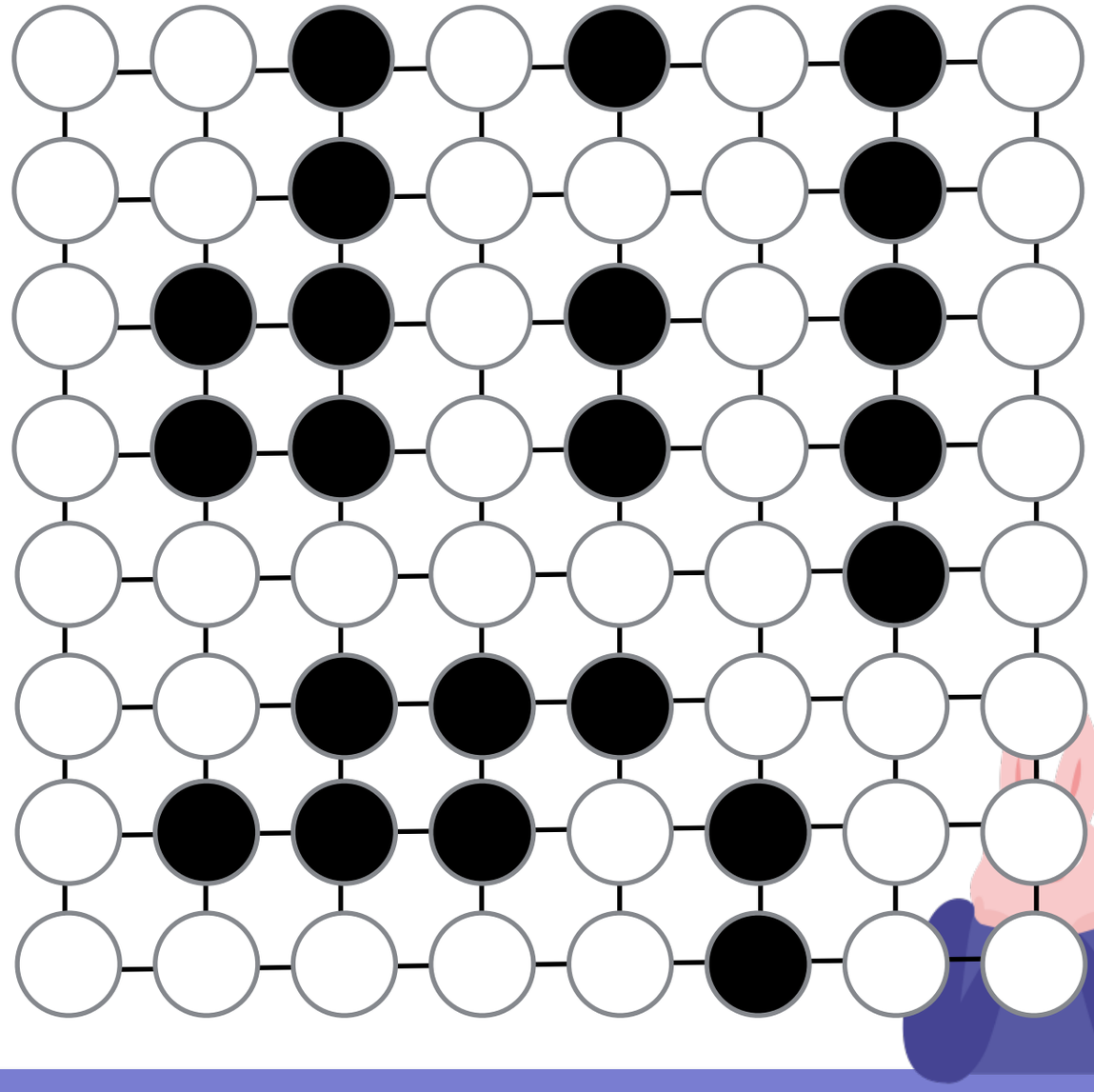




DFS와 BFS의 응용 : 미로찾기

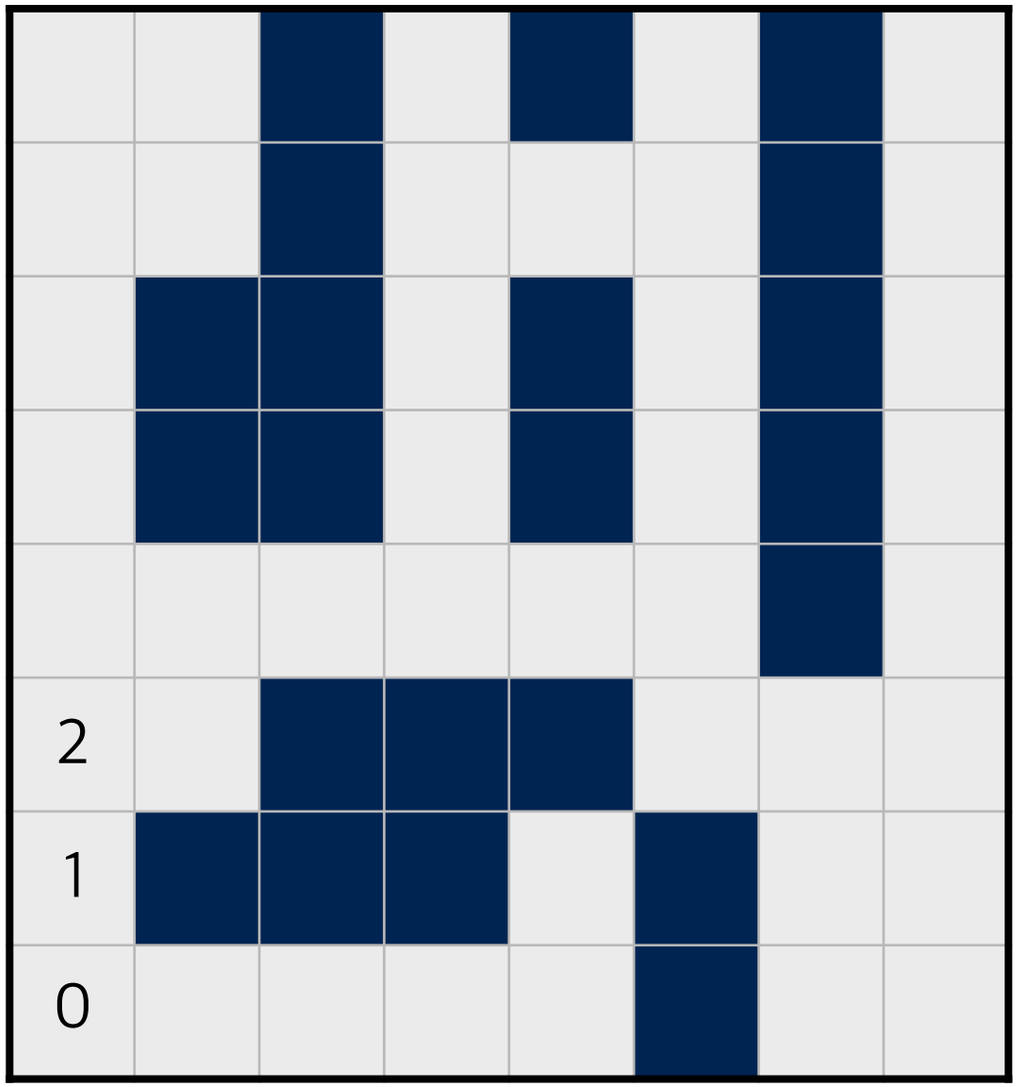
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

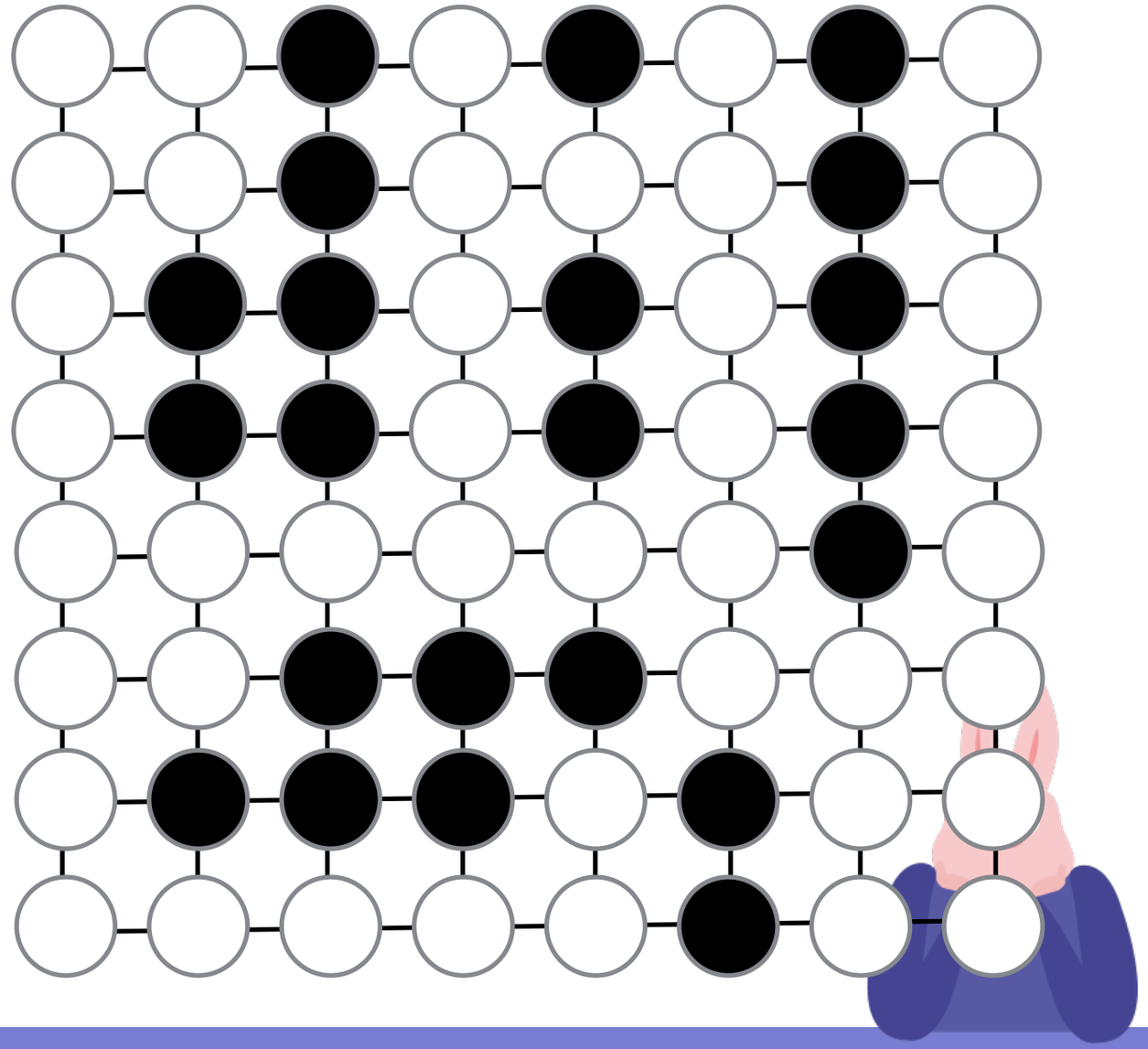




DFS와 BFS의 응용 : 미로찾기

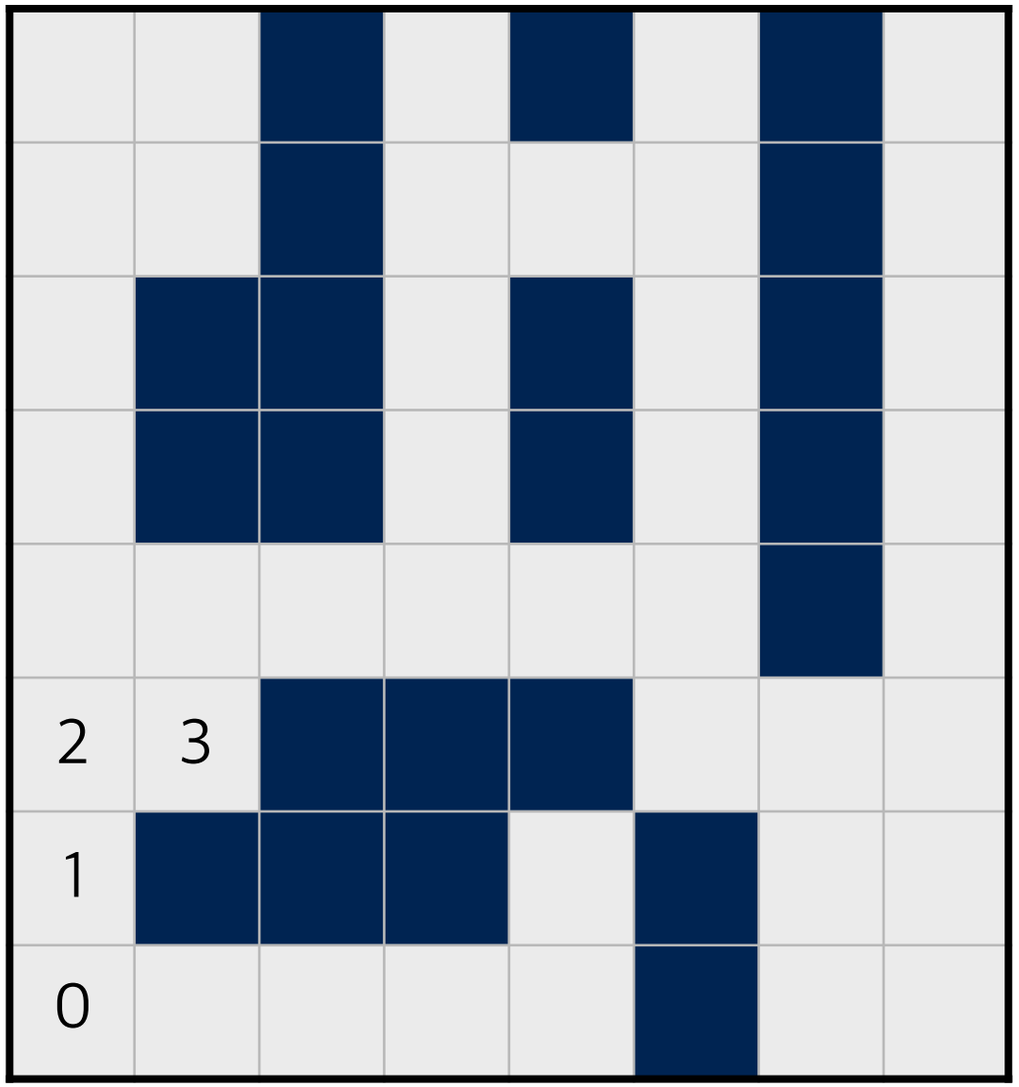
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

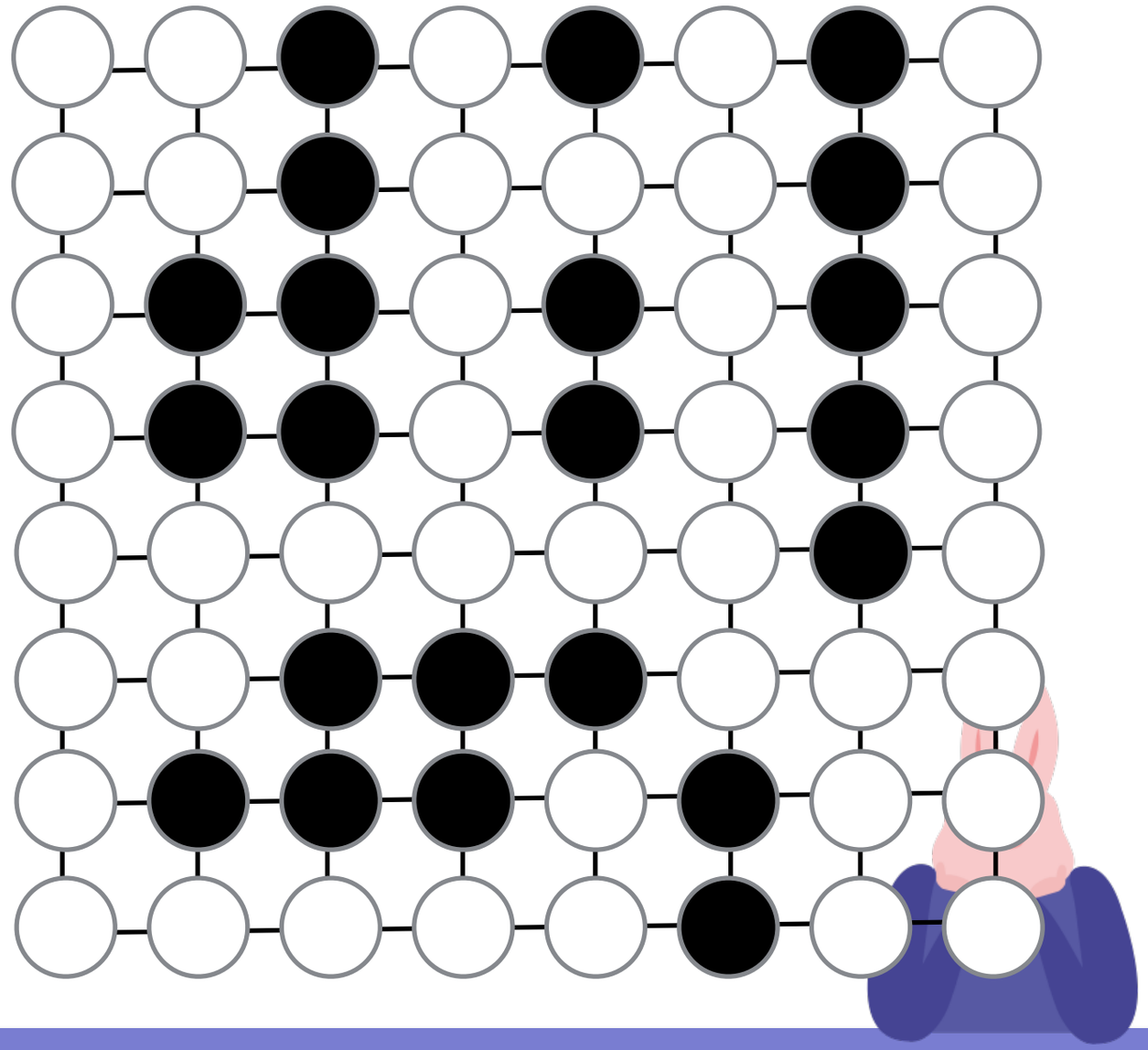




DFS와 BFS의 응용 : 미로찾기

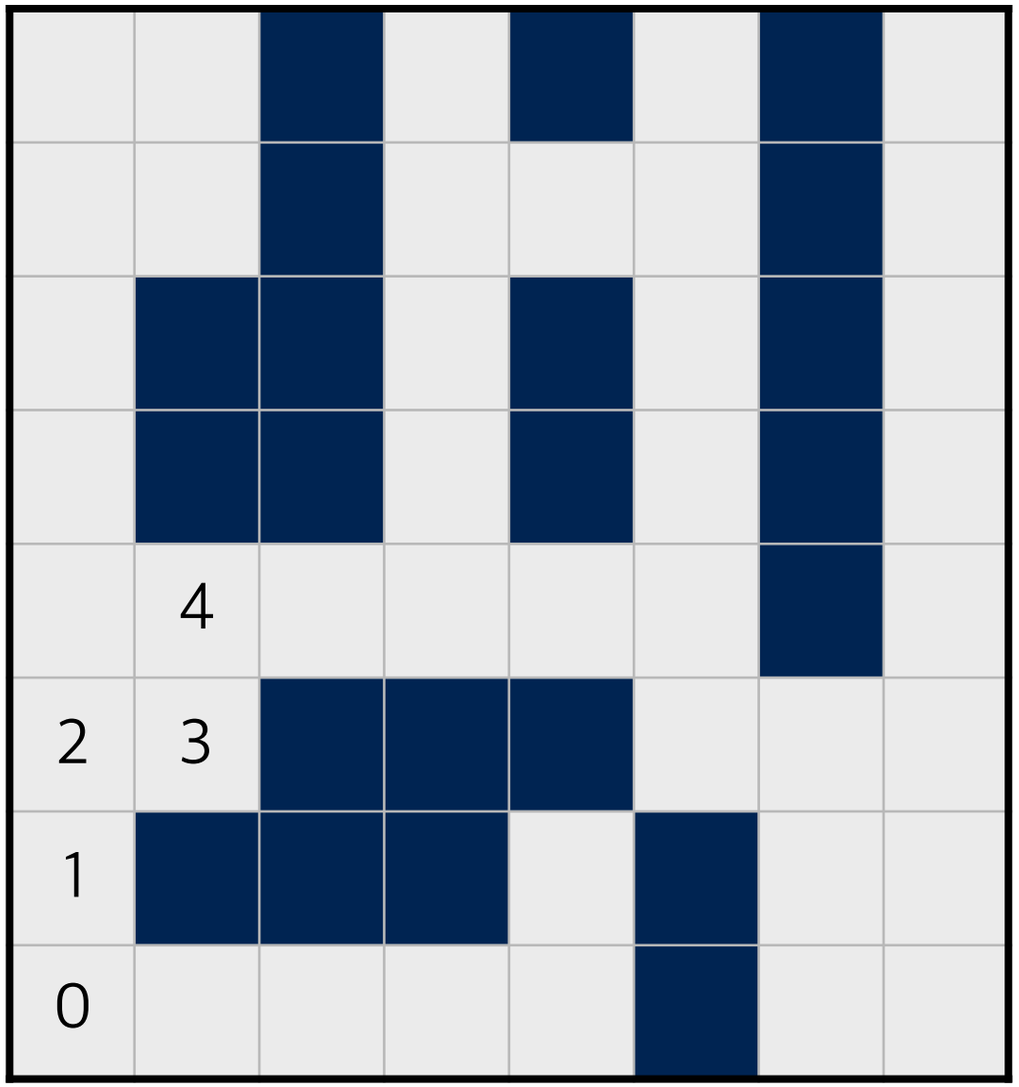
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

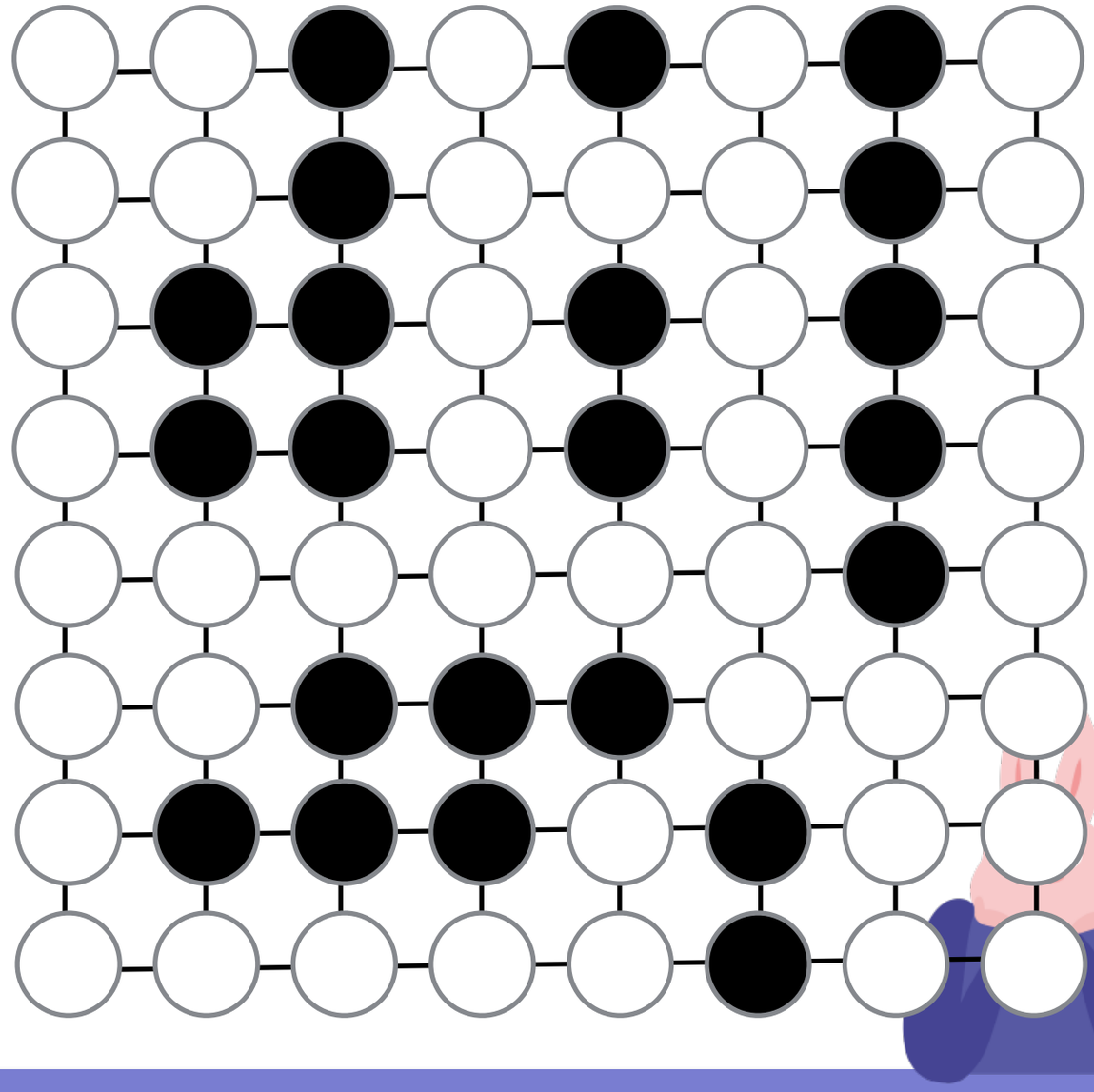




DFS와 BFS의 응용 : 미로찾기

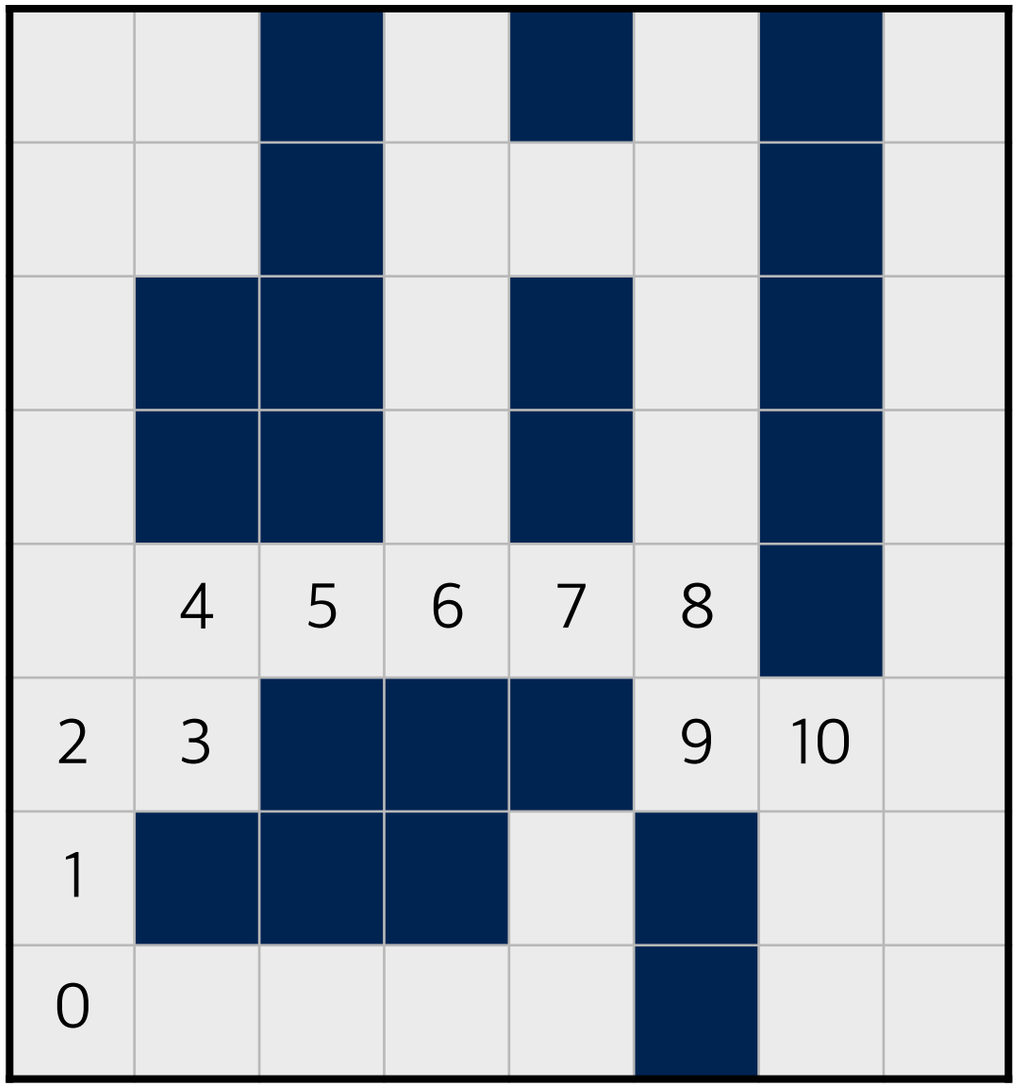
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

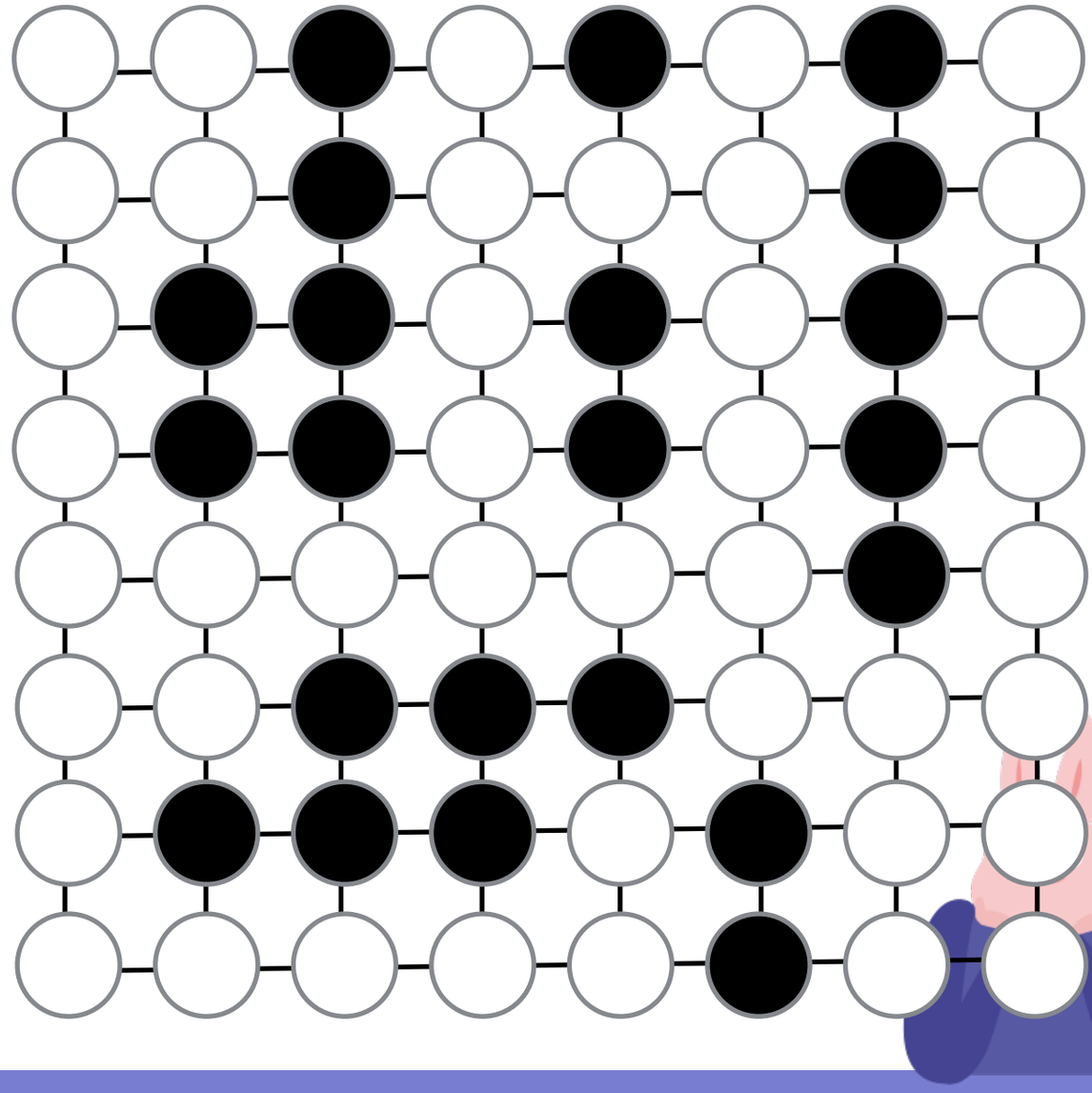




DFS와 BFS의 응용 : 미로찾기

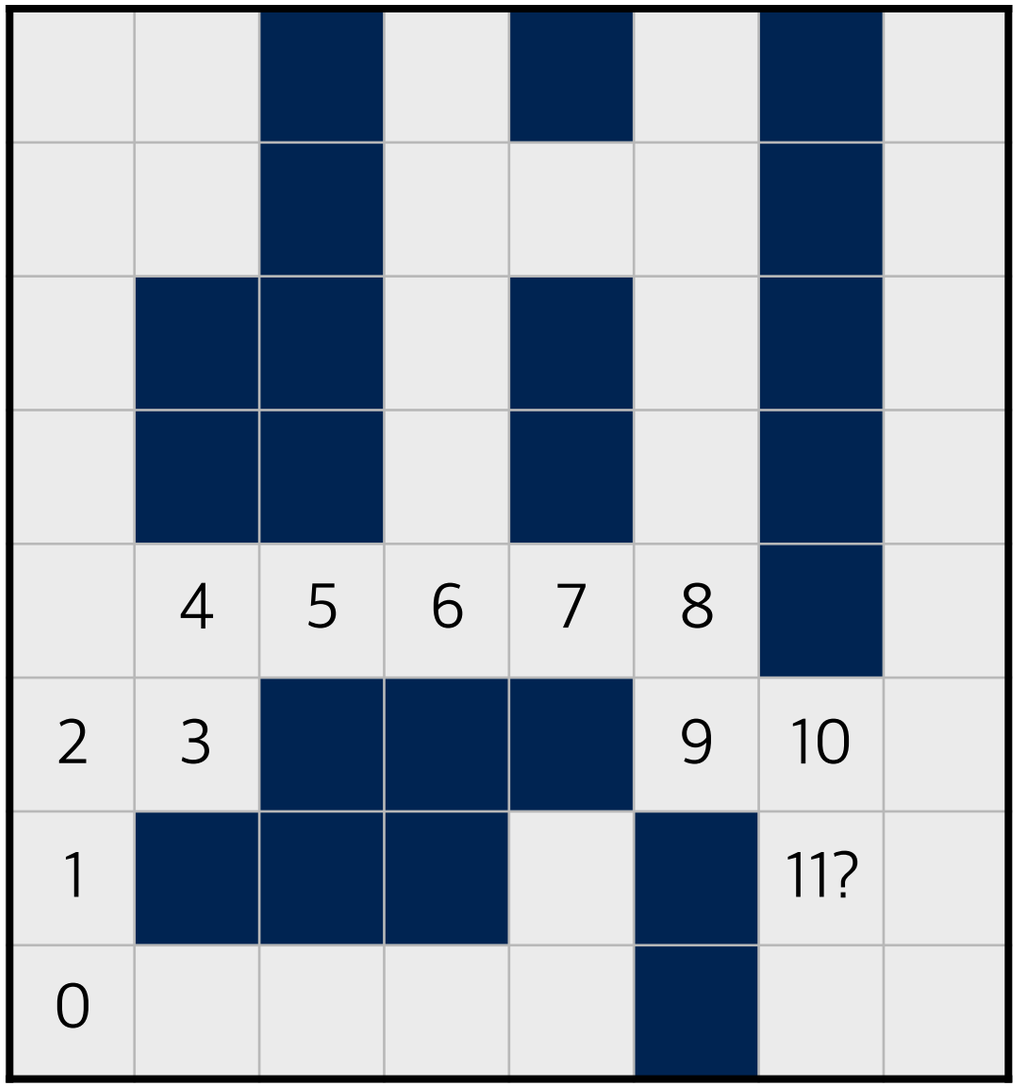
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

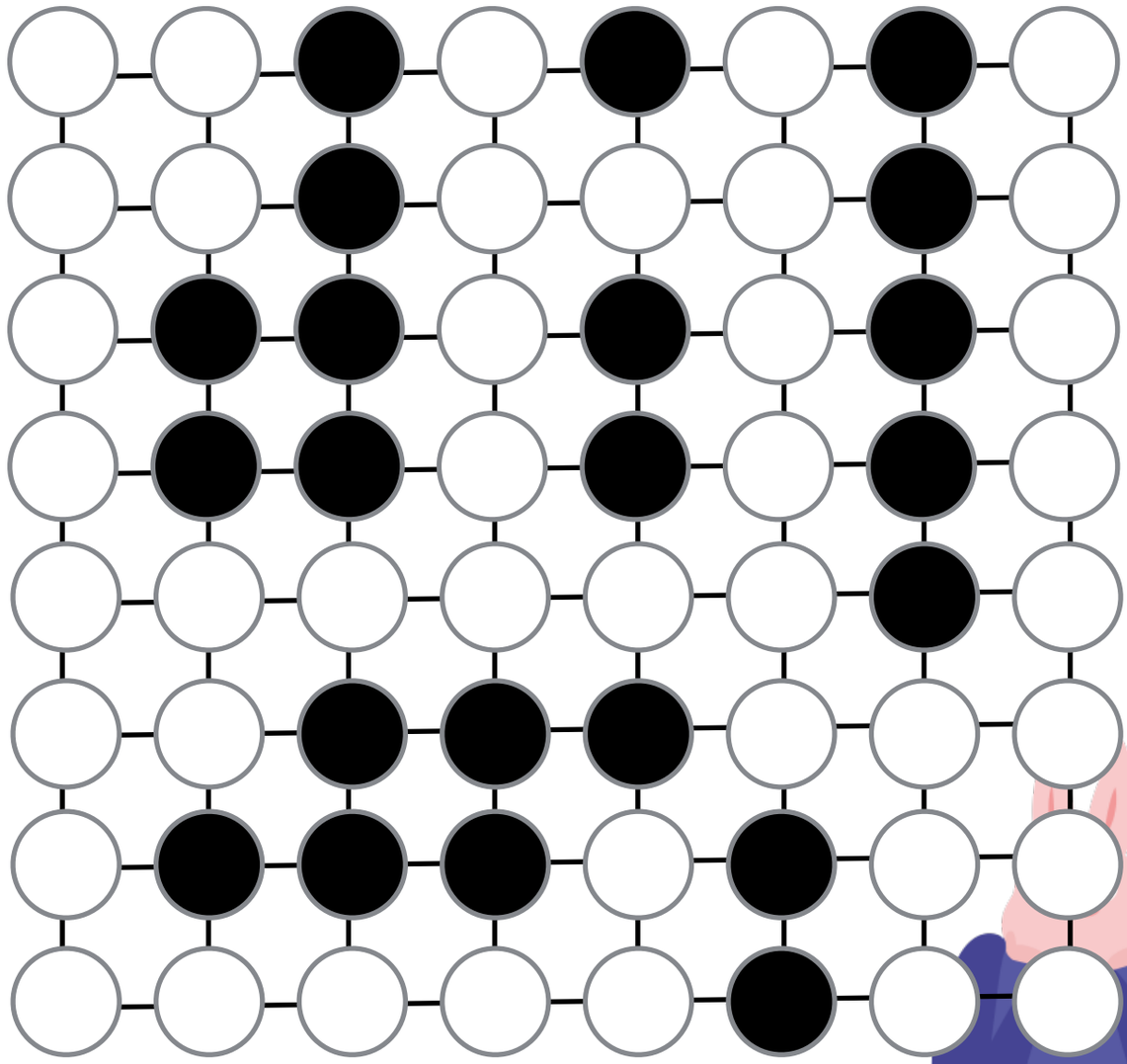




DFS와 BFS의 응용 : 미로찾기

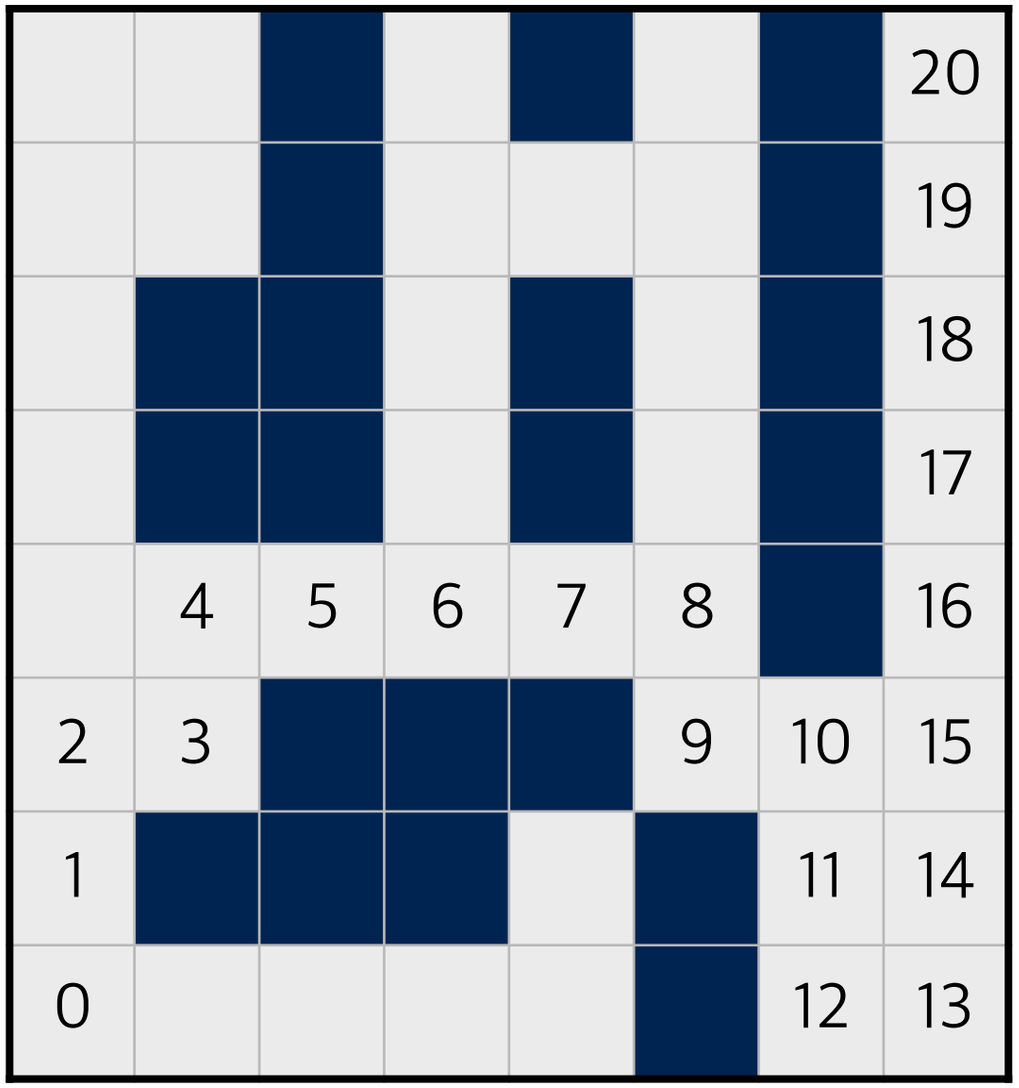
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

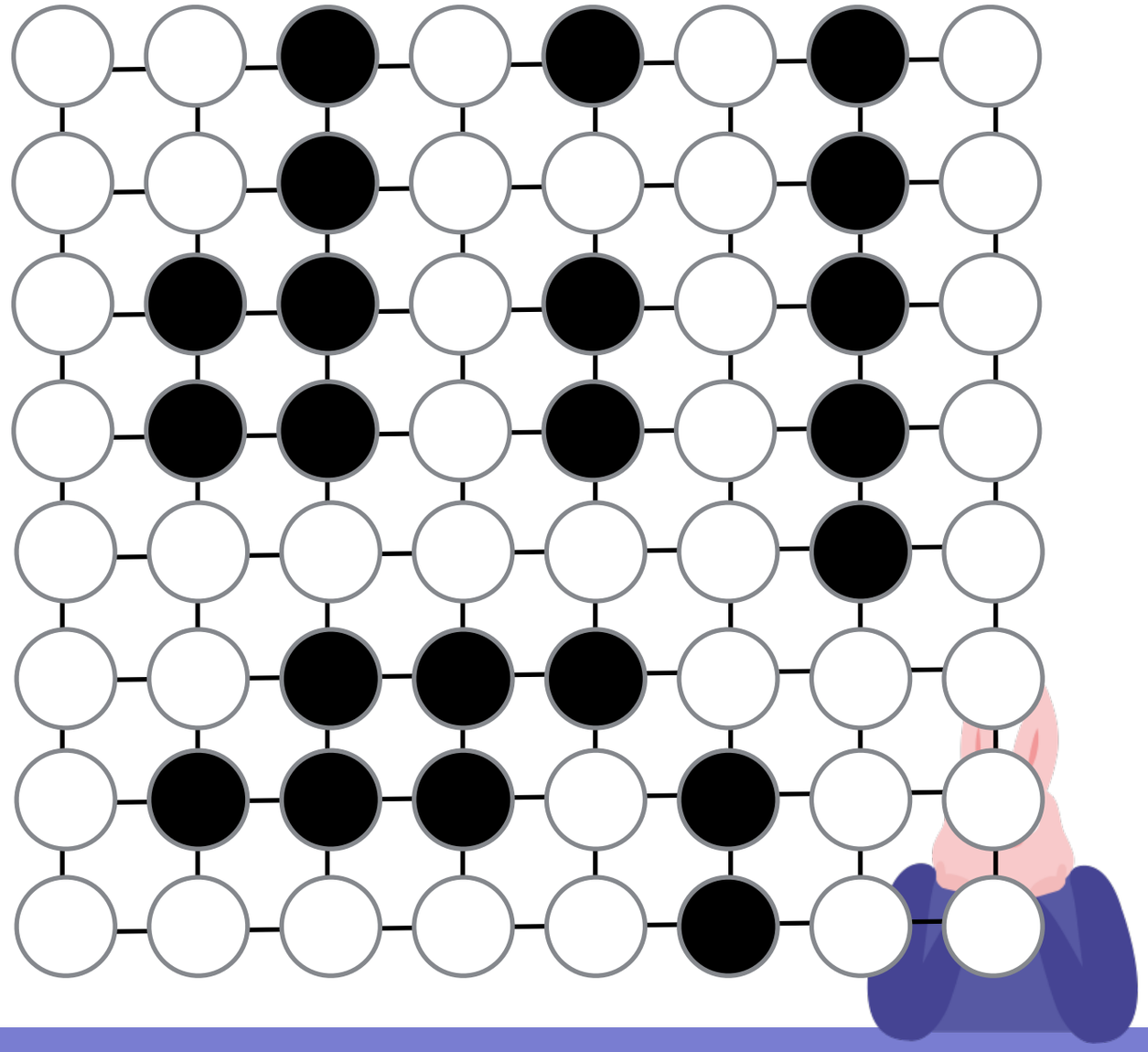




DFS와 BFS의 응용 : 미로찾기

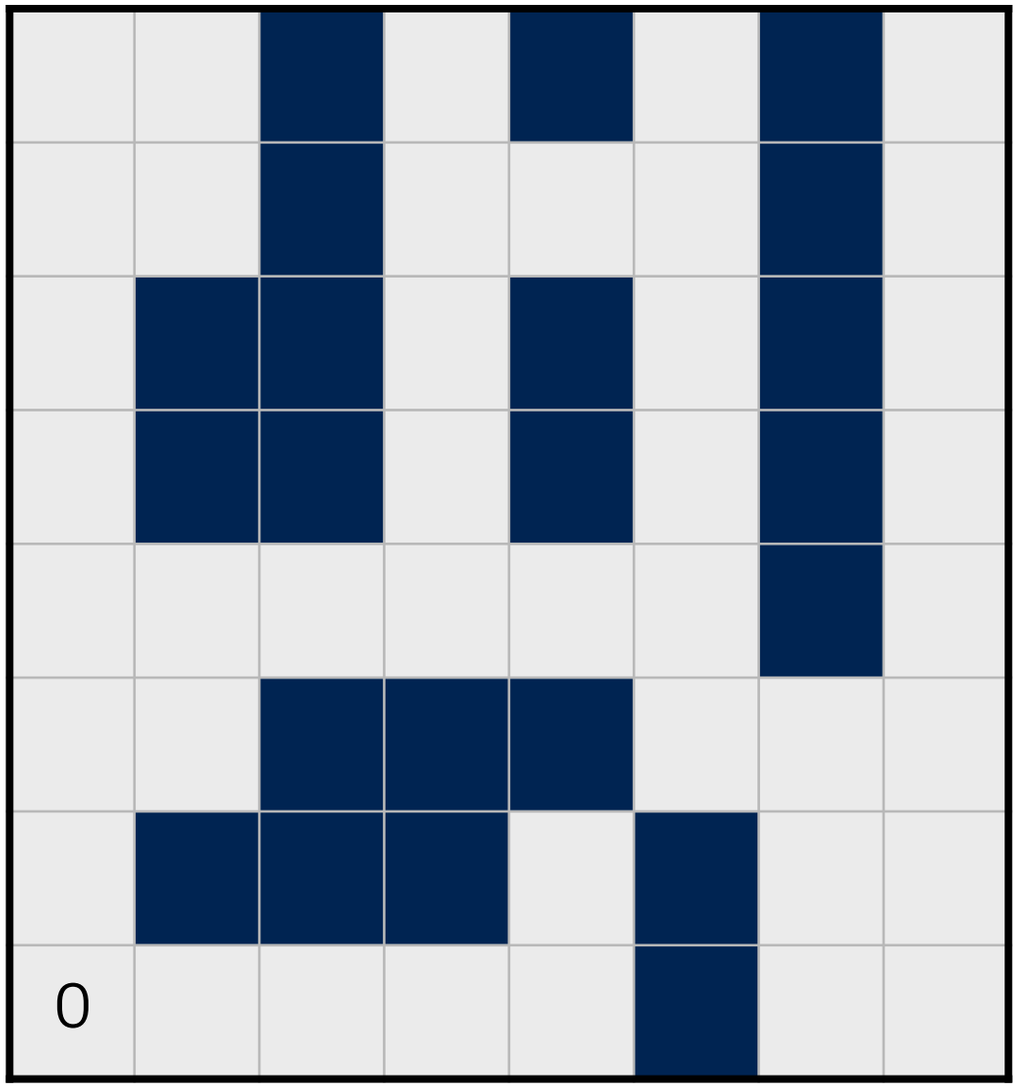
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

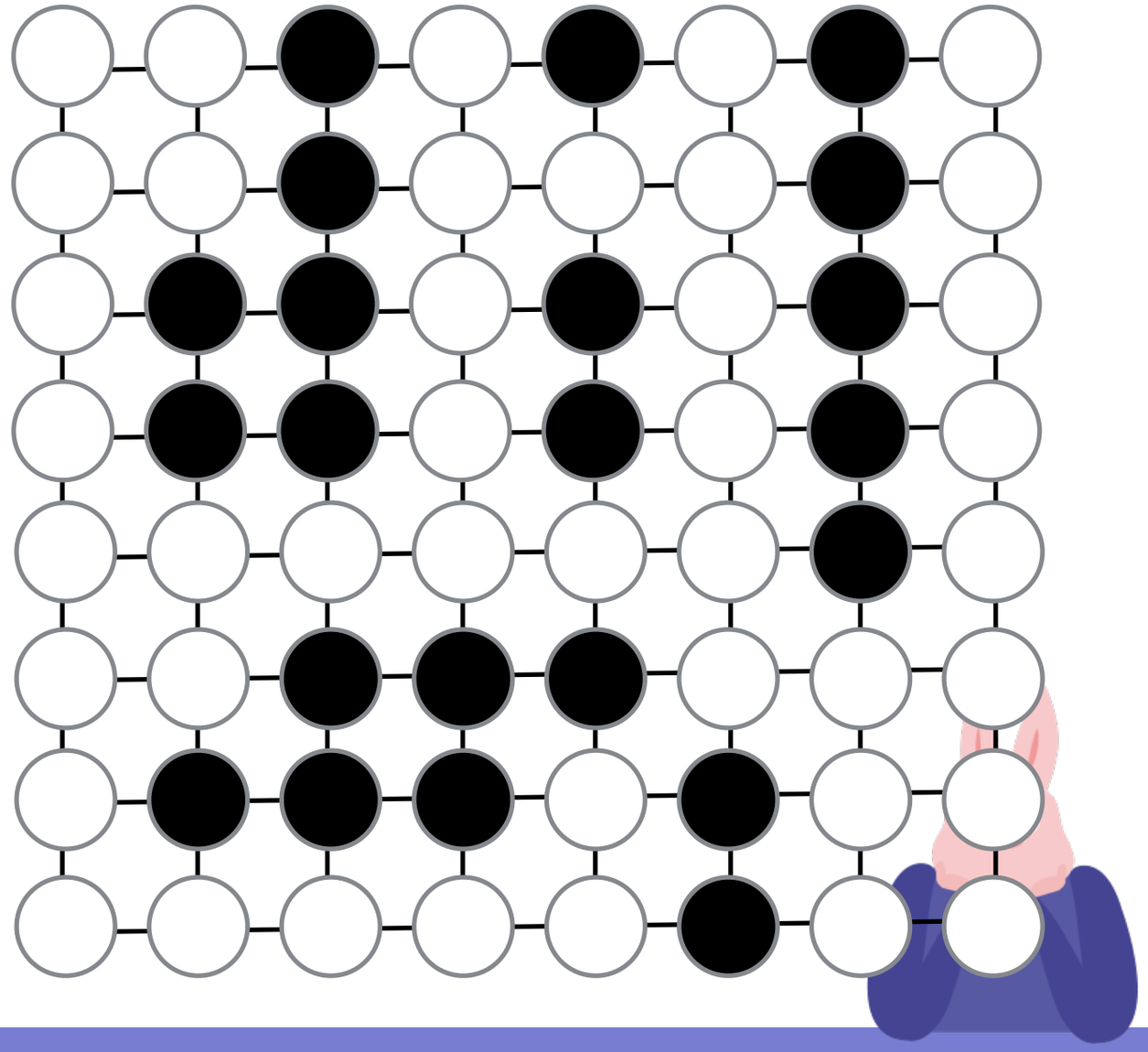




DFS와 BFS의 응용 : 미로찾기

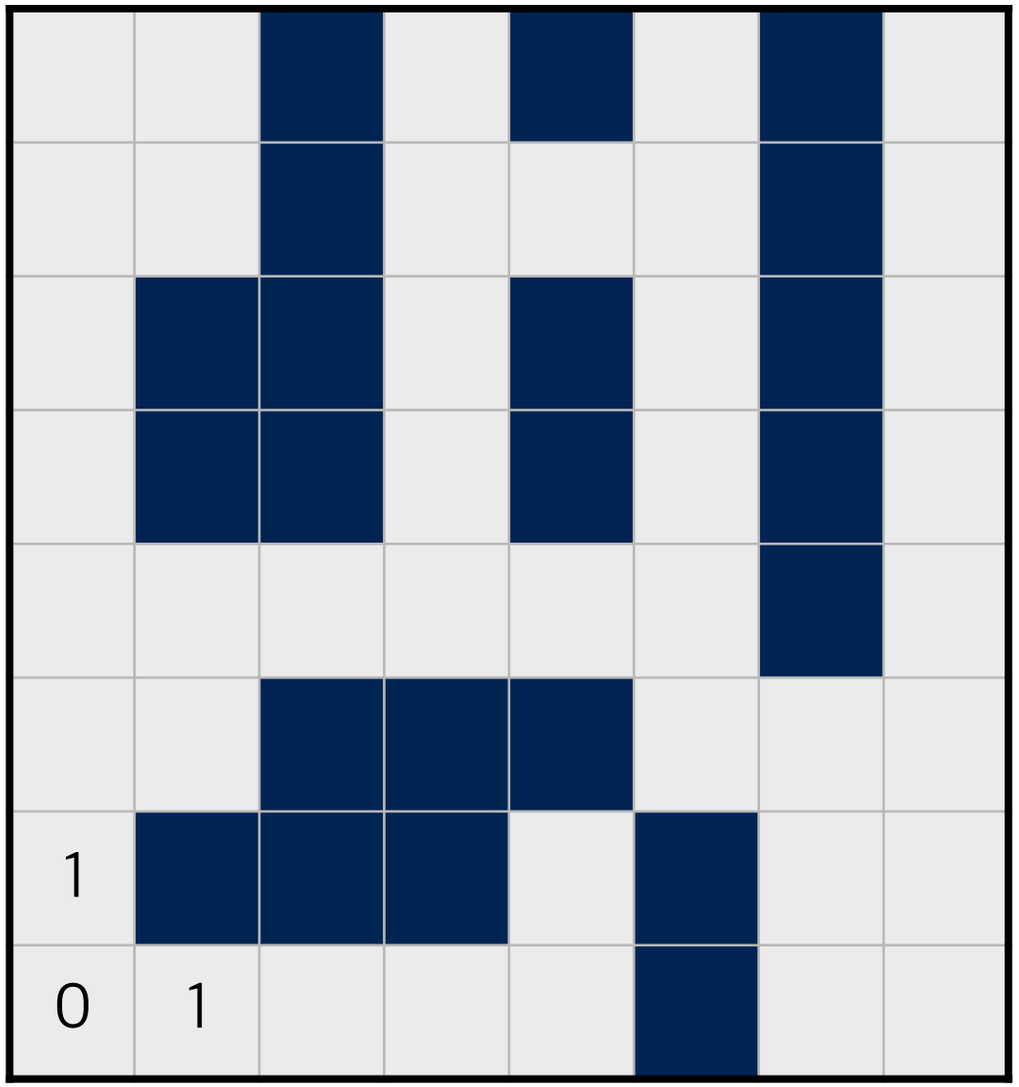
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

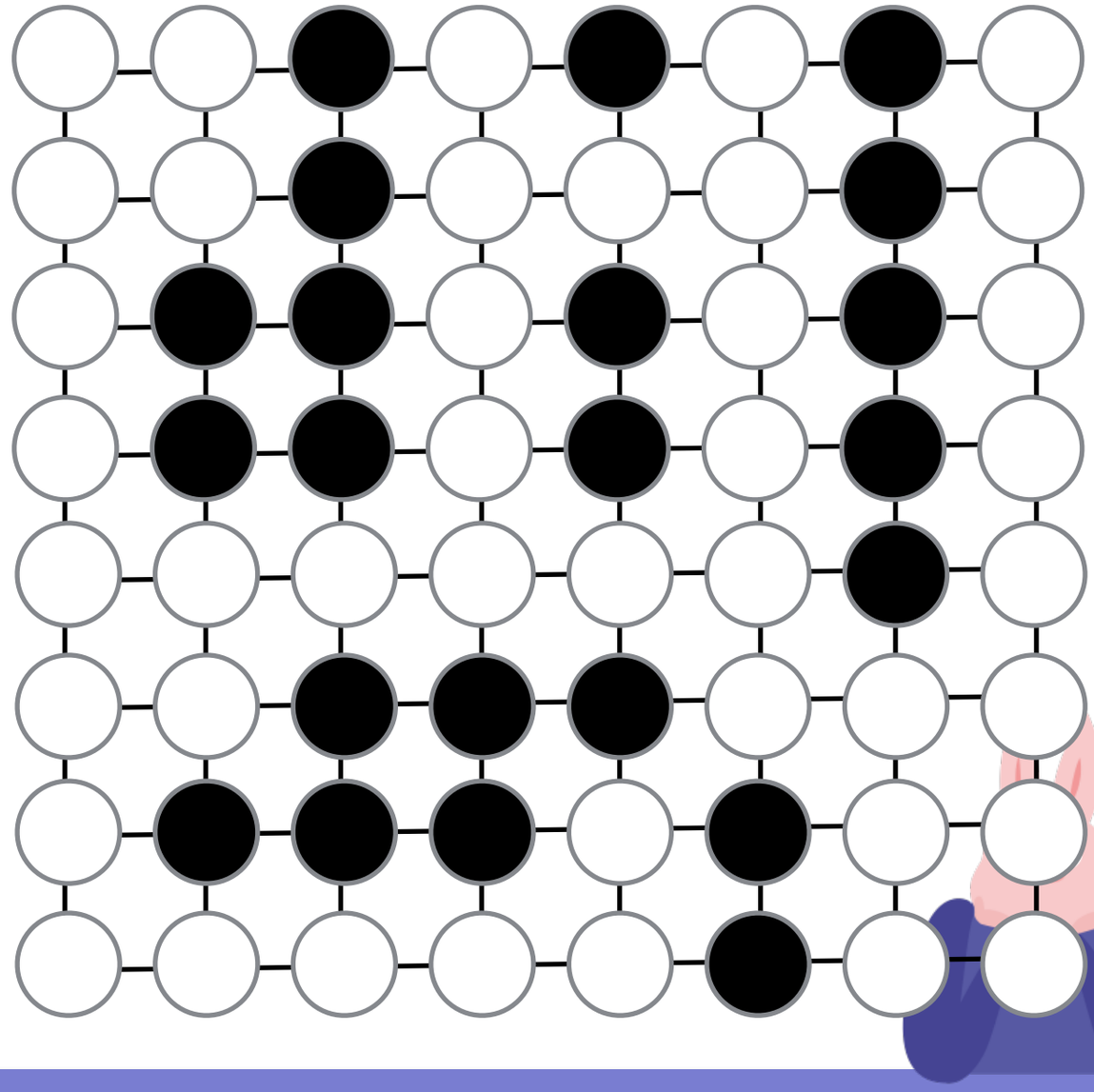




DFS와 BFS의 응용 : 미로찾기

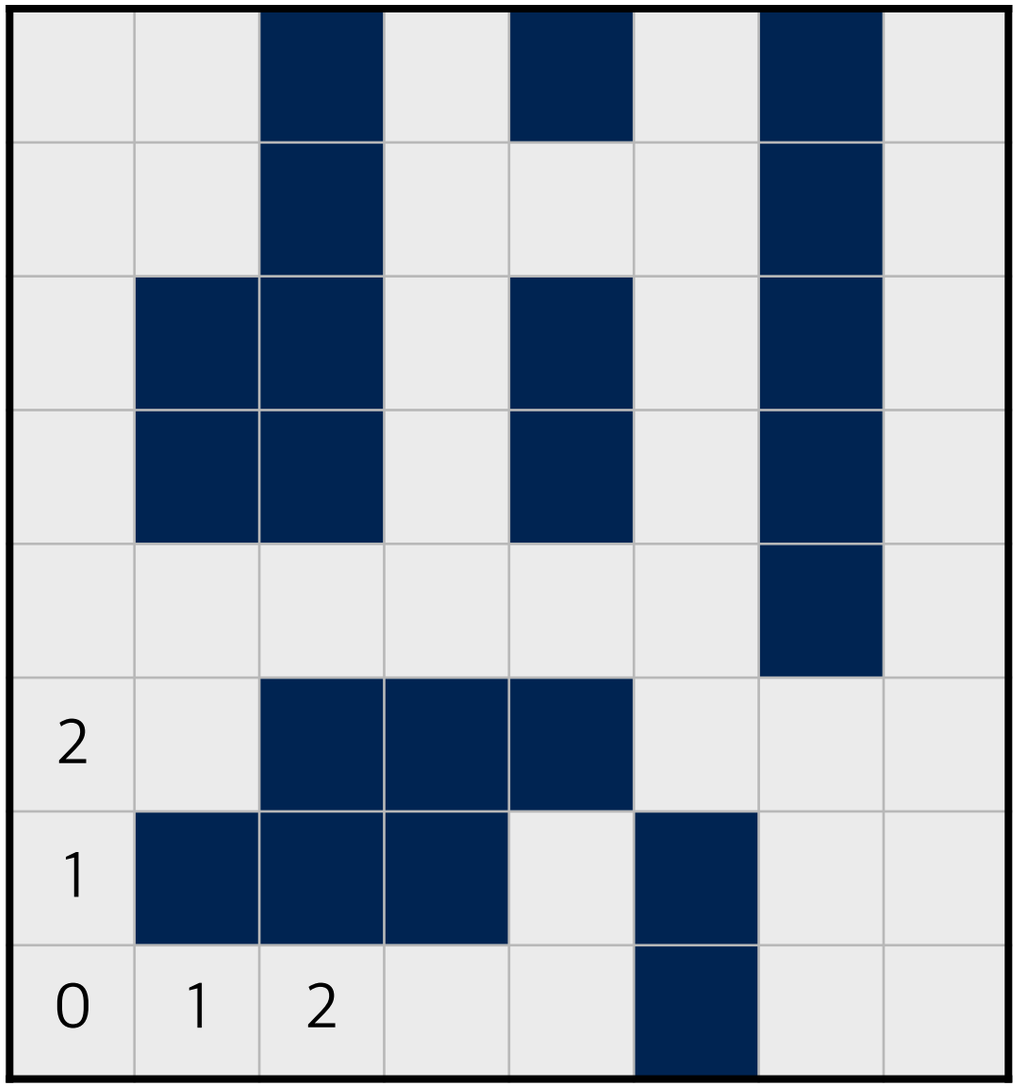
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

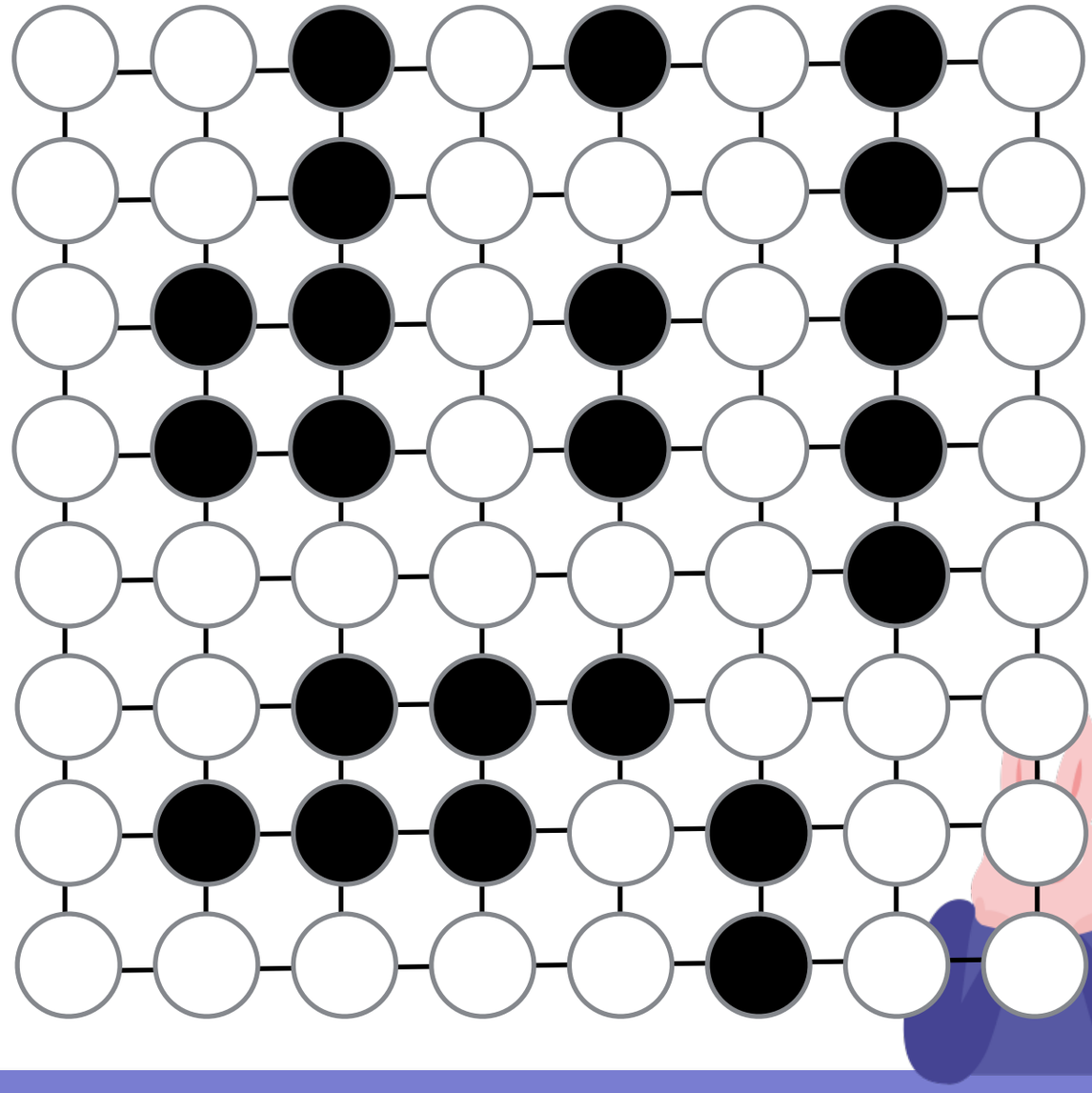




DFS와 BFS의 응용 : 미로찾기

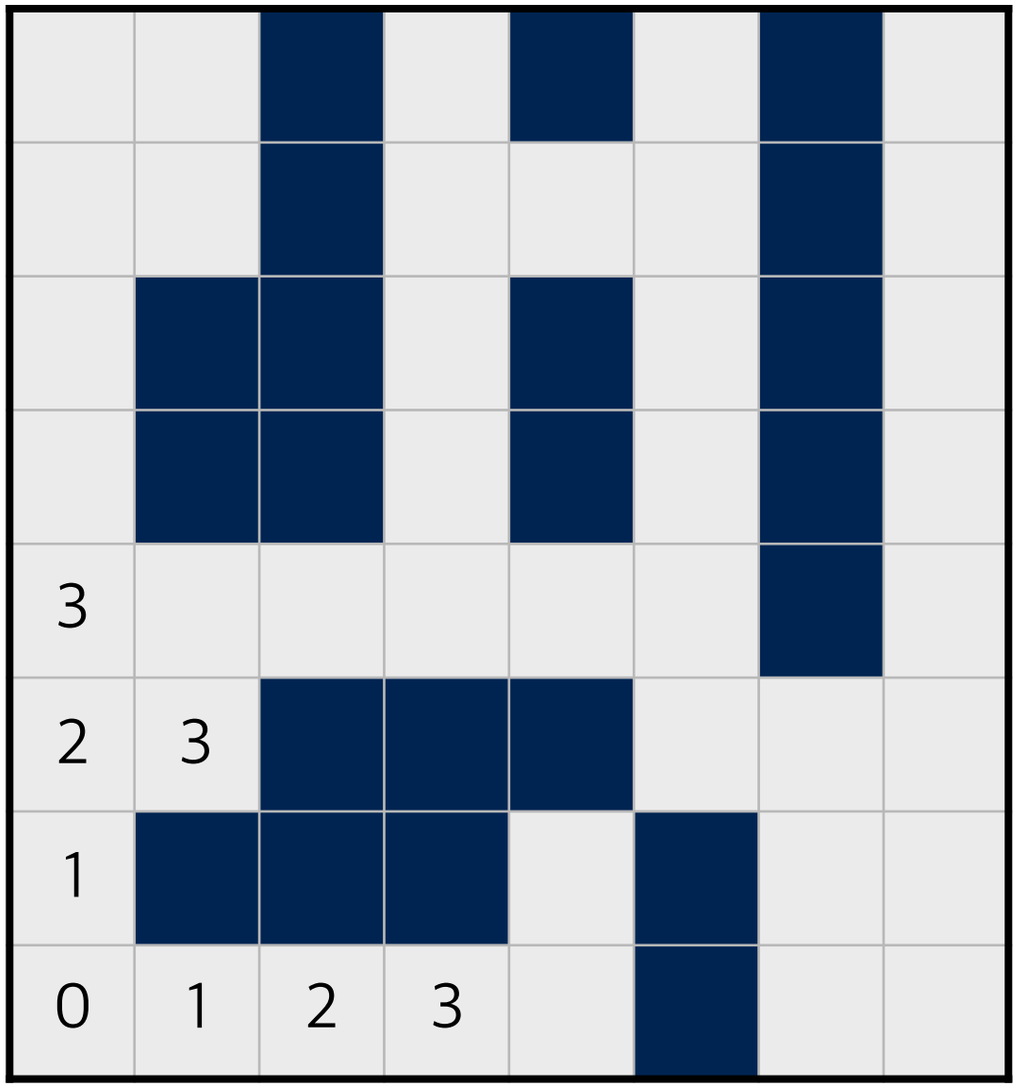
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

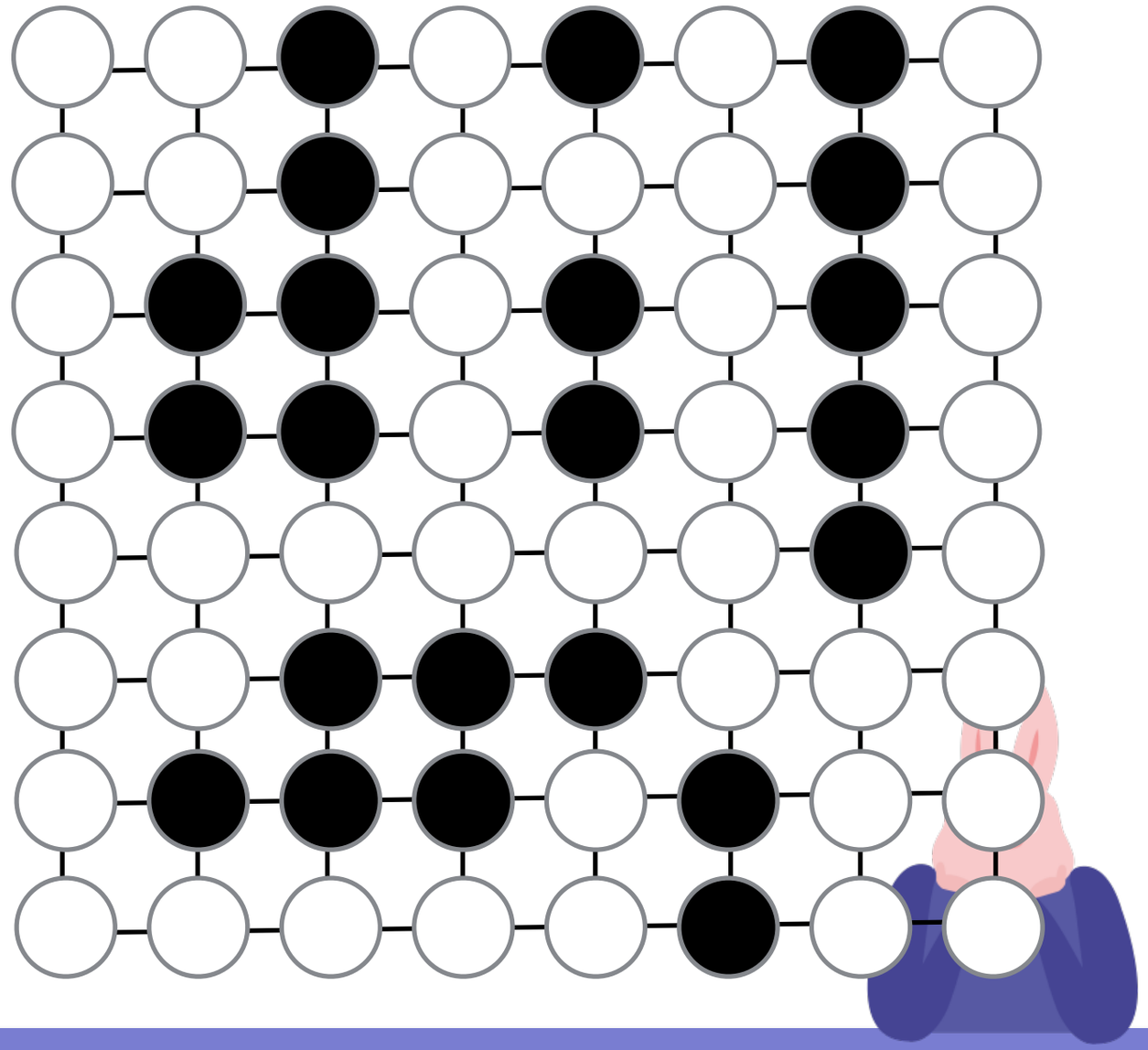




DFS와 BFS의 응용 : 미로찾기

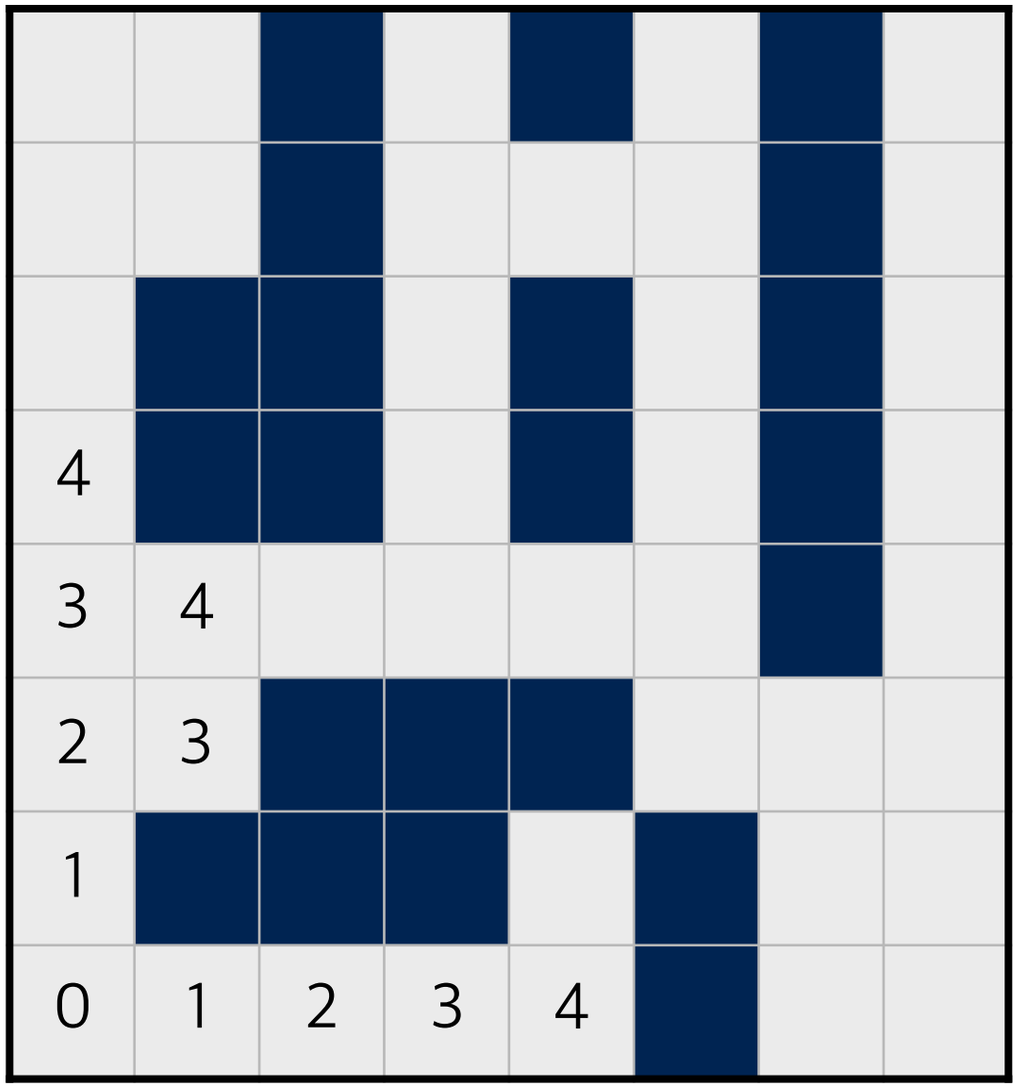
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

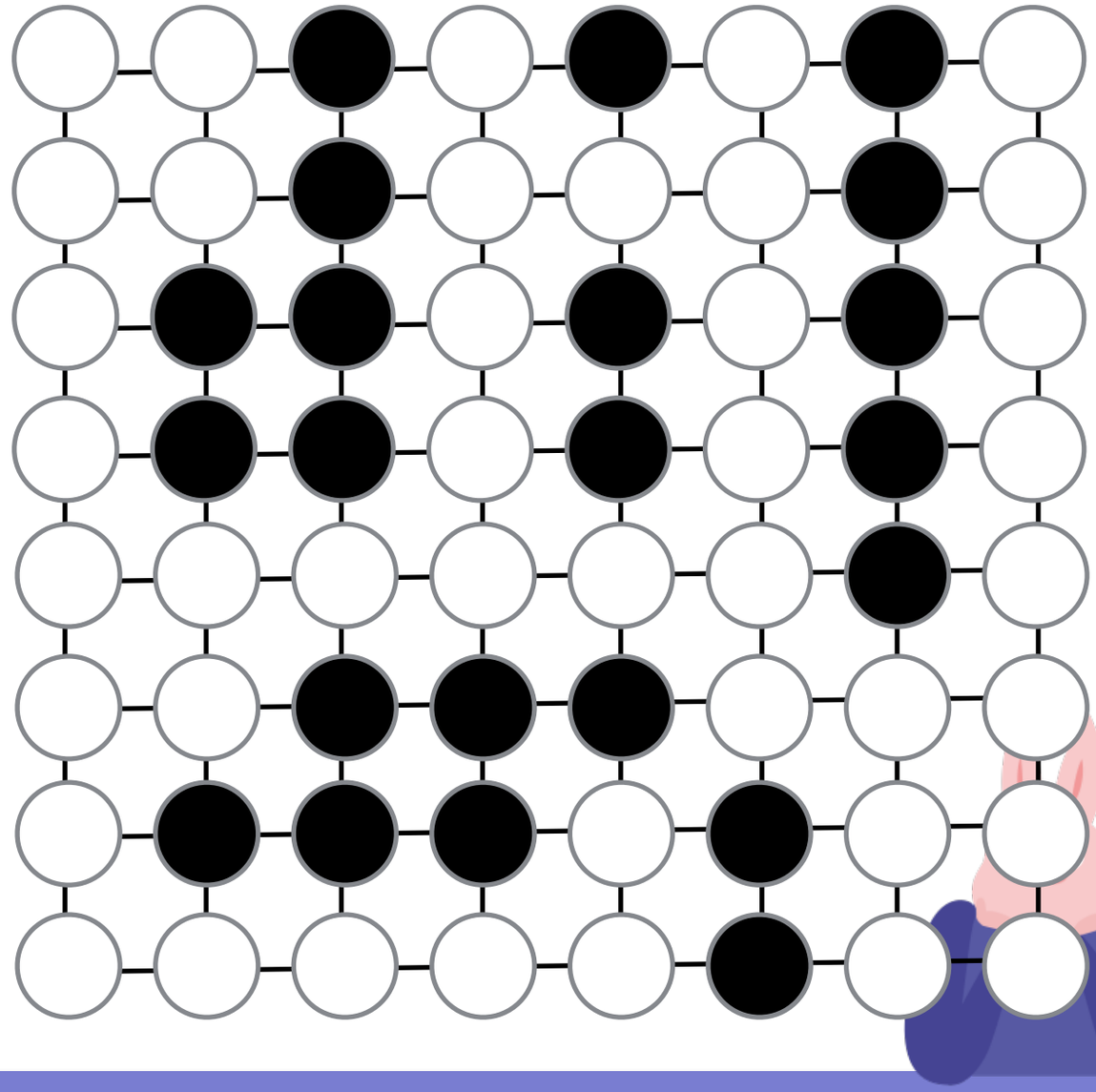




DFS와 BFS의 응용 : 미로찾기

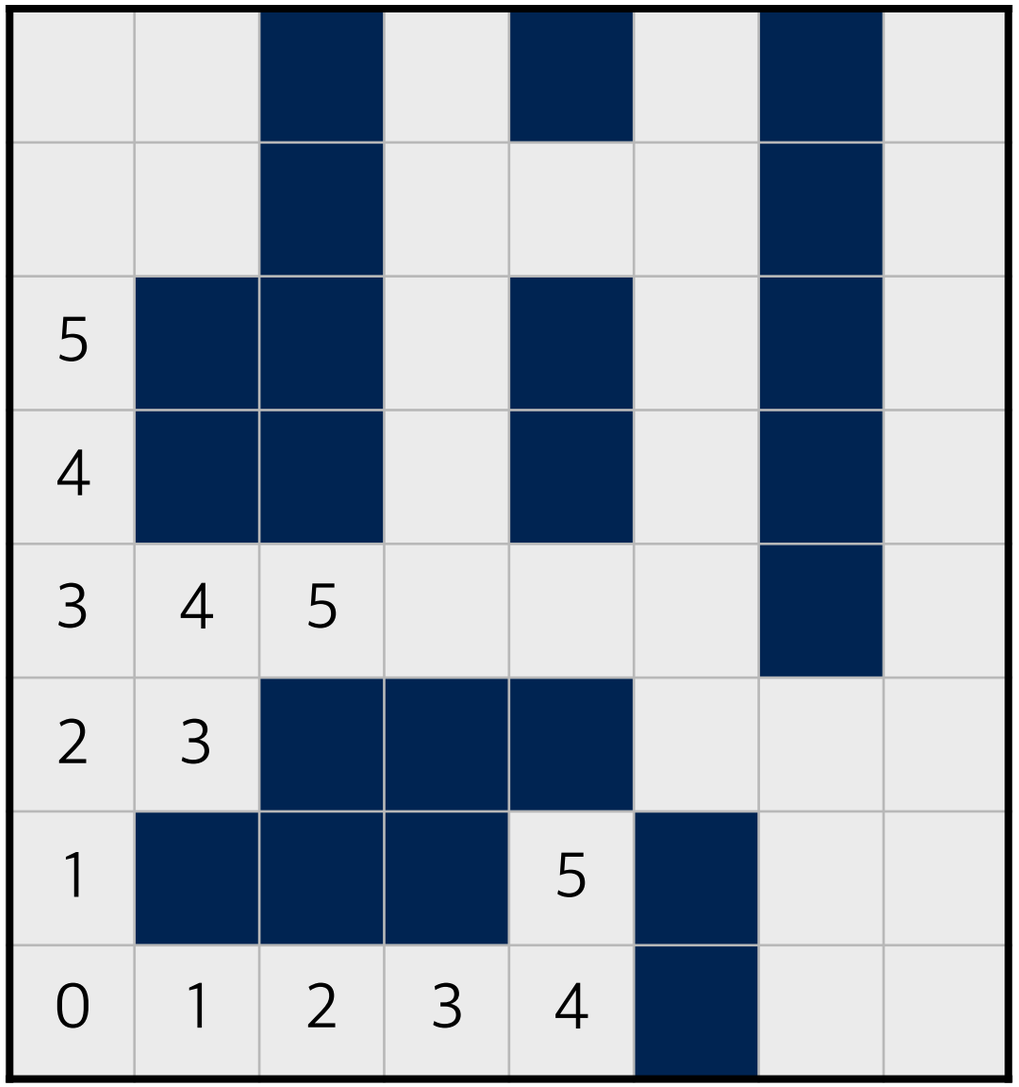
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

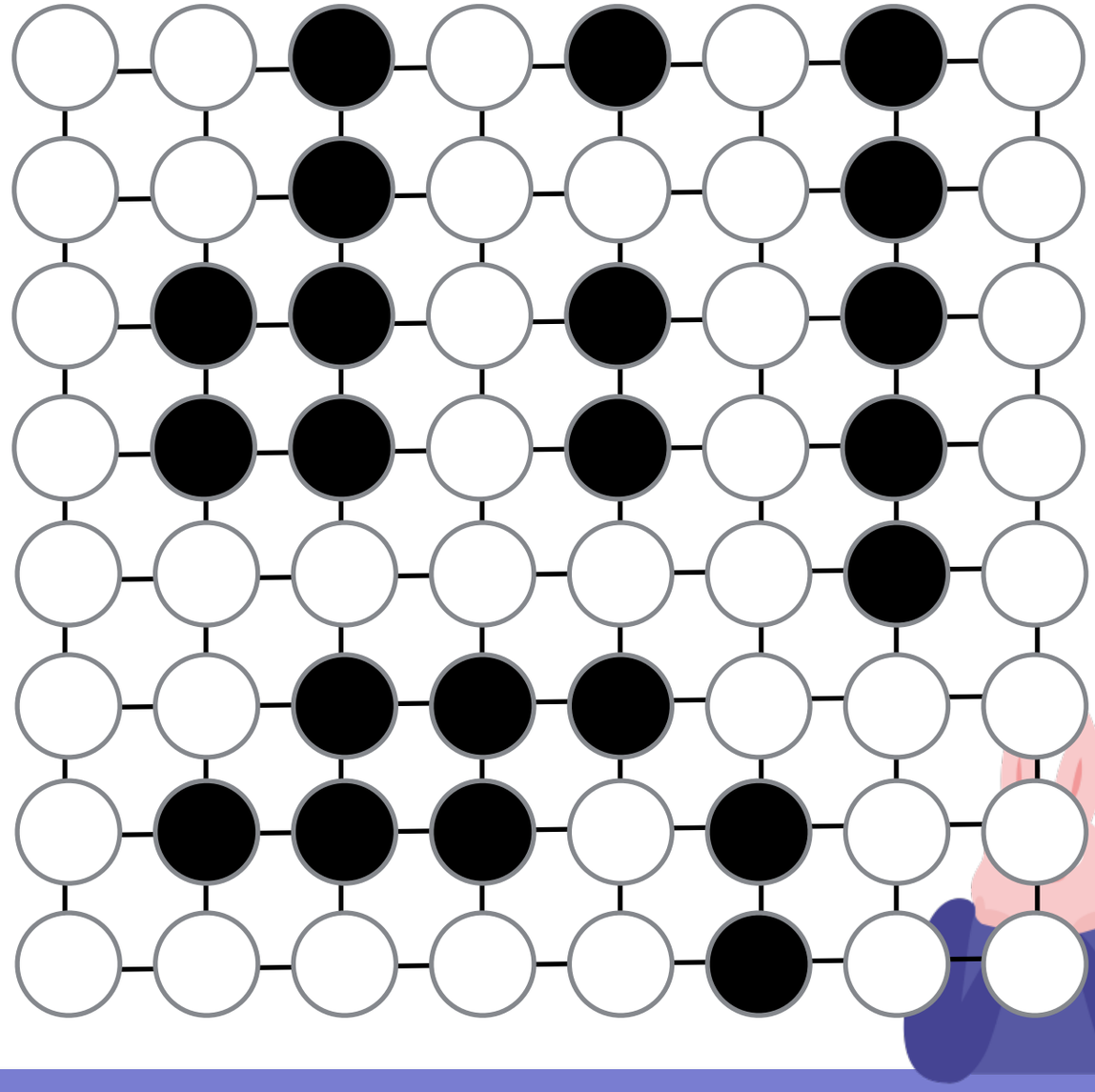




DFS와 BFS의 응용 : 미로찾기

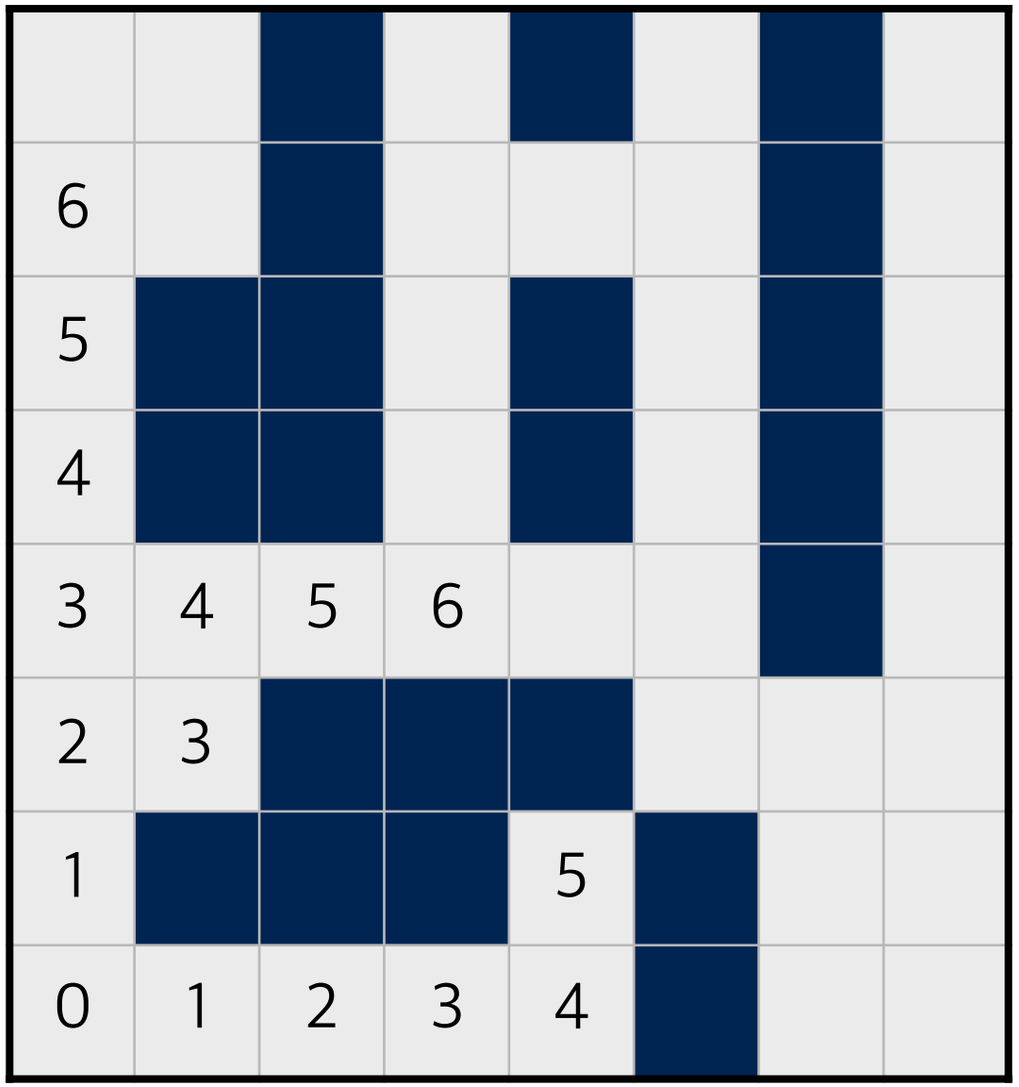
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

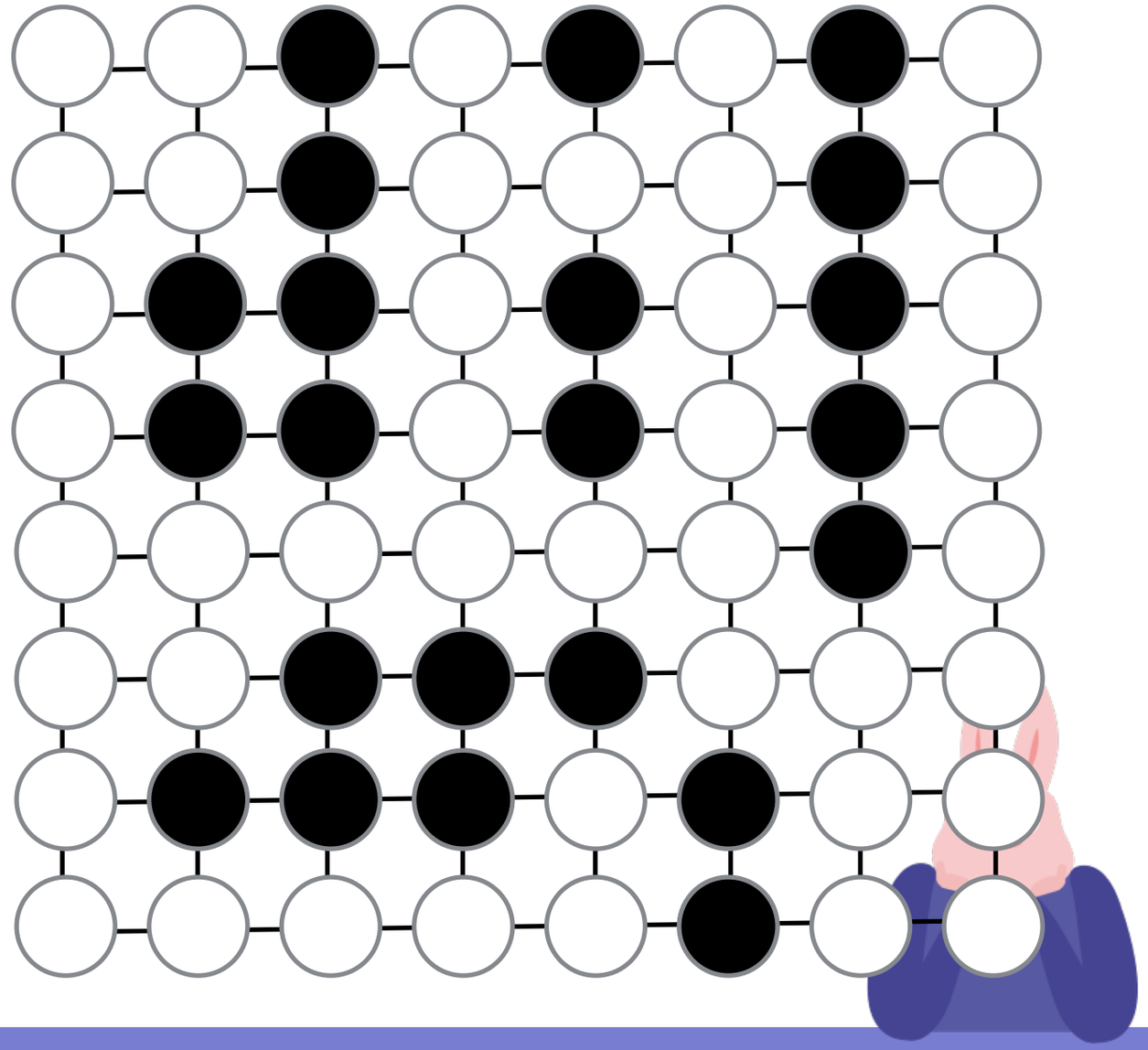




DFS와 BFS의 응용 : 미로찾기

2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

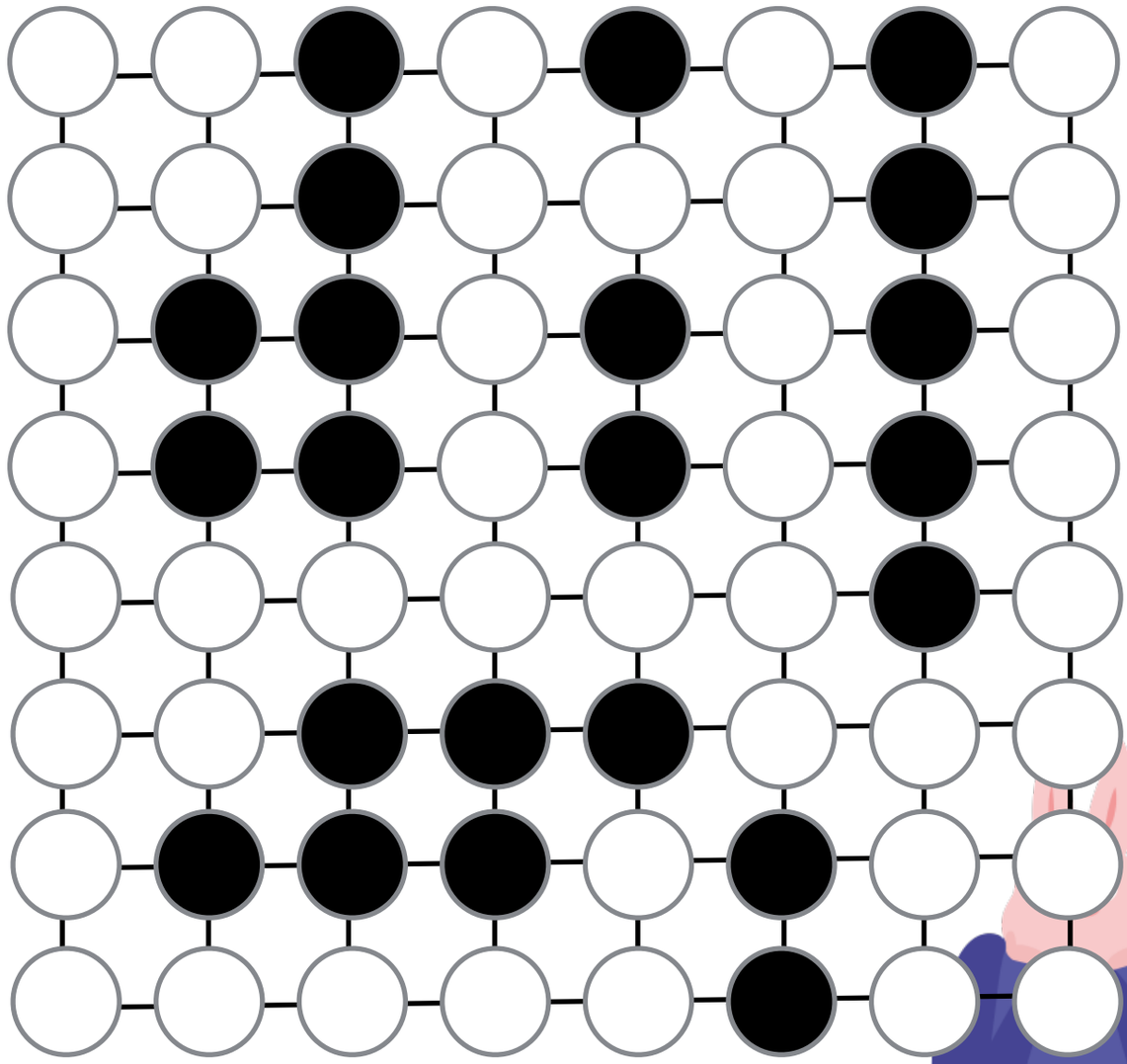




DFS와 BFS의 응용 : 미로찾기

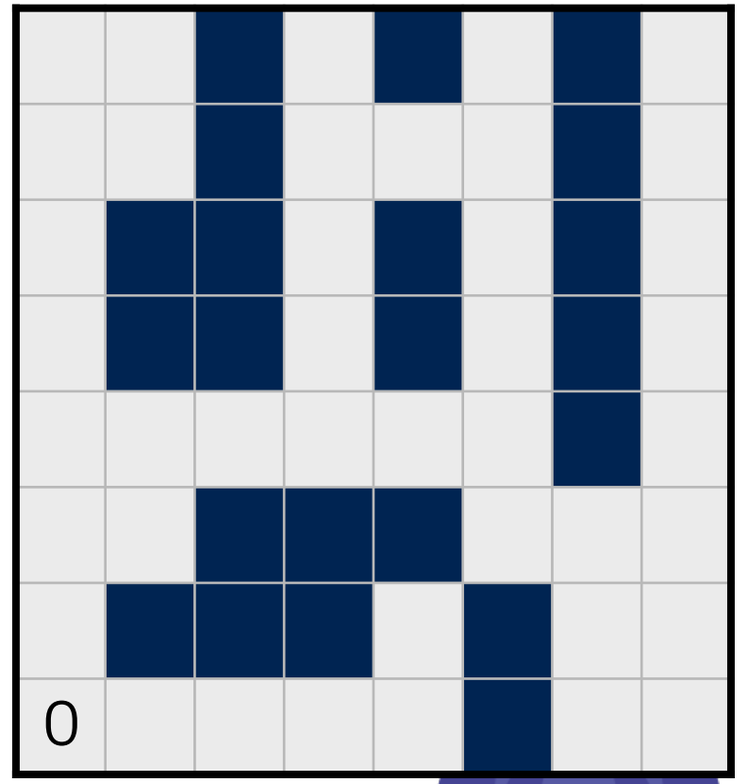
2. 각 노드에, 그 노드에 도착하기 위한 최단거리를 저장

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)

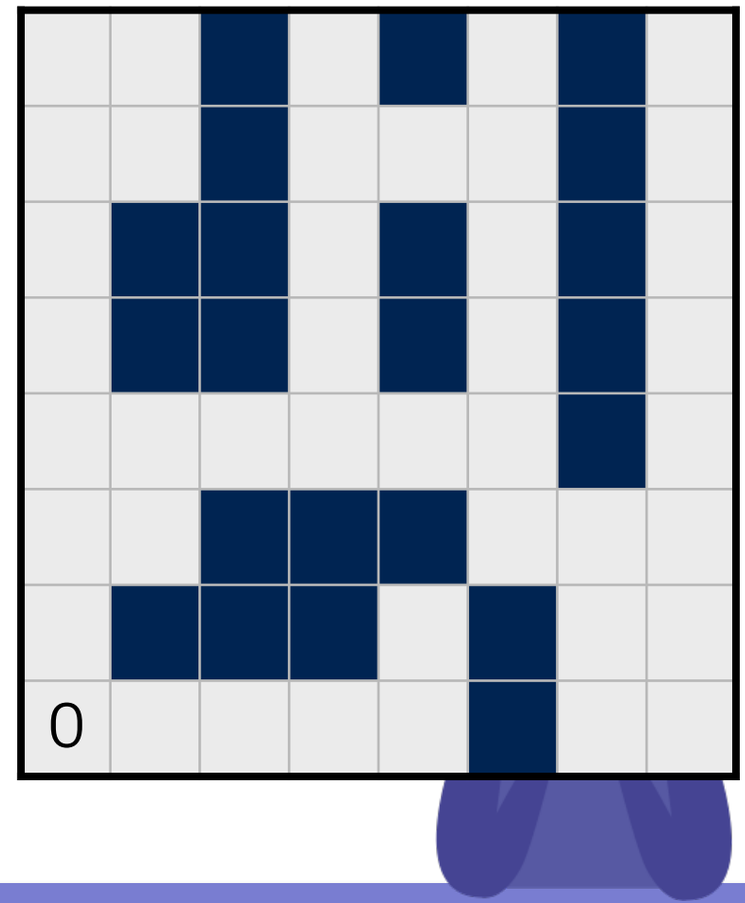
- 주장 : BFS를 이용하여 방문하면 **최단거리**를 구할 수 있다



BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 **최단거리**를 구할 수 있다

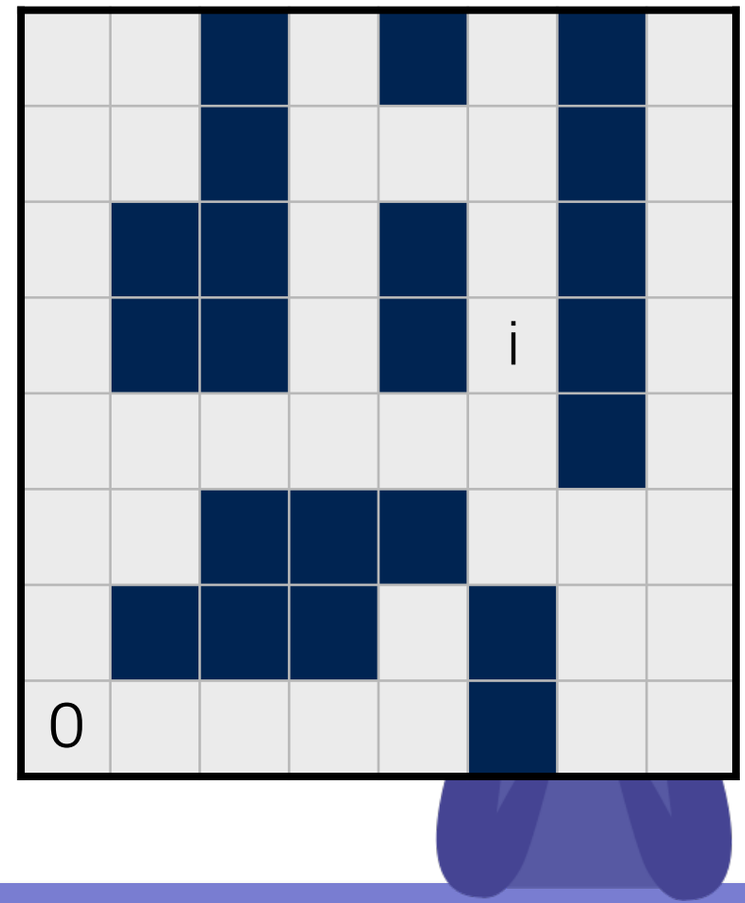
- 증명 (귀류법)



BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 **최단거리**를 구할 수 있다

- 증명 (귀류법)
  - 만약 최단거리를 구하지 못했다고 가정하자
    그리고, 최단거리를 구하지 못한 칸을 i라 하자

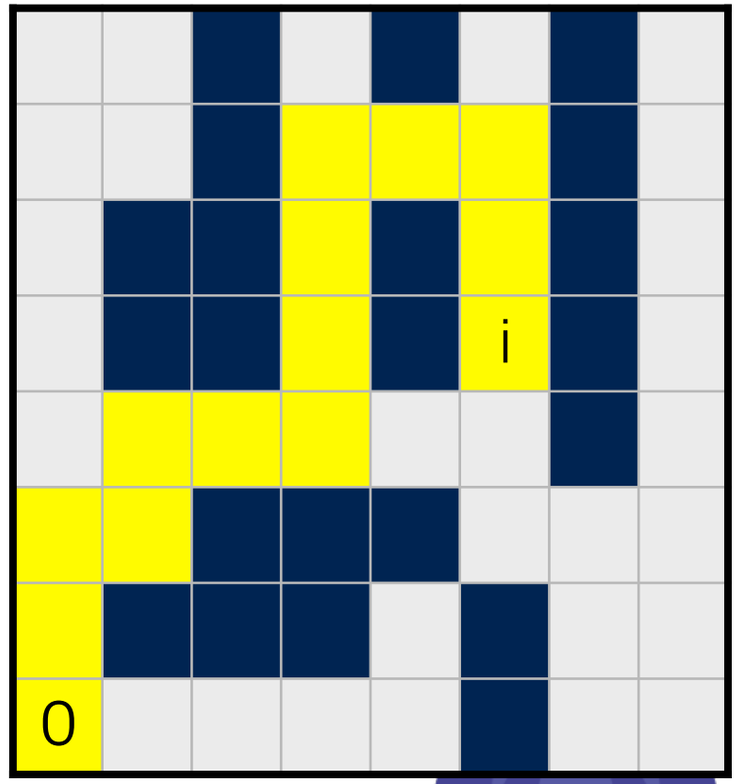


BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 **최단거리**를 구할 수 있다

- 증명 (귀류법)
  - 만약 최단거리를 구하지 못했다고 가정하자
    그리고, 최단거리를 구하지 못한 칸을 i라 하자

  - 현재 i 까지 따라온 경로를 P(i)라 하자
    최단거리를 구하지 못했으므로, P(i)는 최단경로가 아니다.



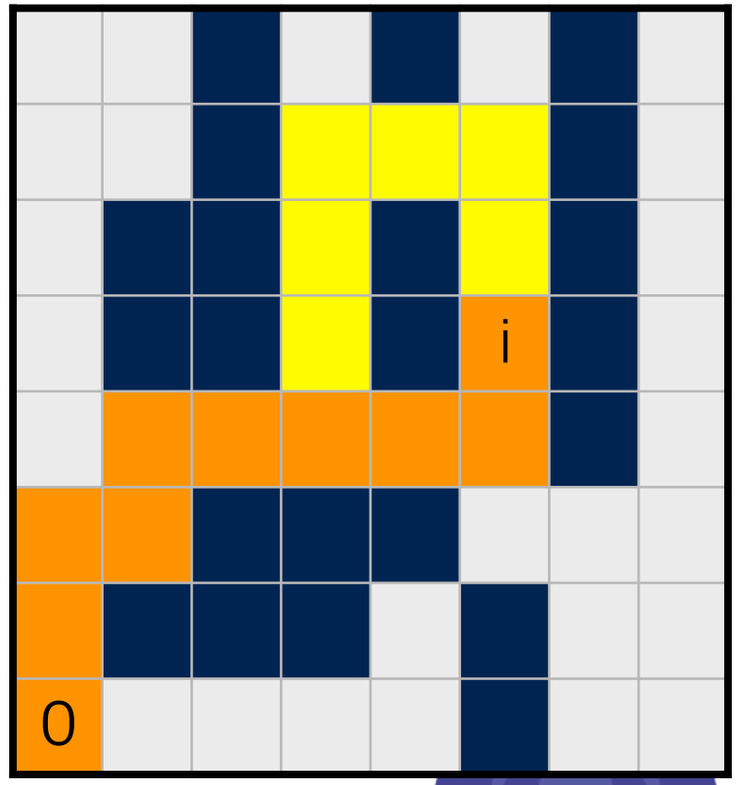
BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 **최단거리**를 구할 수 있다

- 증명 (귀류법)
  - 만약 최단거리를 구하지 못했다고 가정하자
    그리고, 최단거리를 구하지 못한 칸을 i라 하자

  - 현재 i 까지 따라온 경로를 P(i)라 하자
    최단거리를 구하지 못했으므로, P(i)는 최단경로가 아니다.

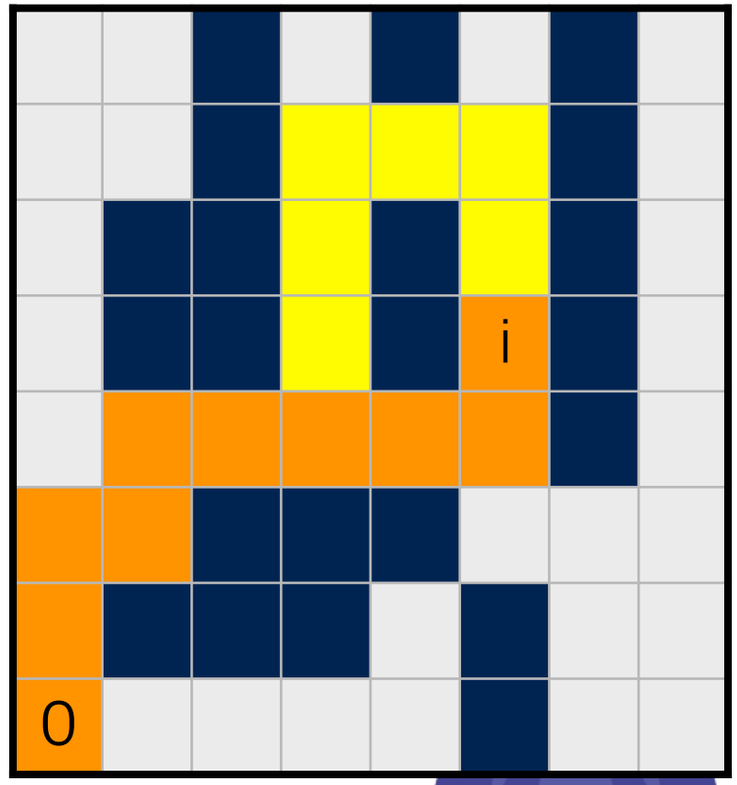
  - 구하지 못한 최단경로를 S(i)라 하자



BFS의 응용 : 미로찾기 (정확성 증명)

- 주장 : BFS를 이용하여 방문하면 **최단거리**를 구할 수 있다

- 증명 (귀류법)
  - 만약 최단거리를 구하지 못했다고 가정하자
    그리고, 최단거리를 구하지 못한 칸을 i라 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

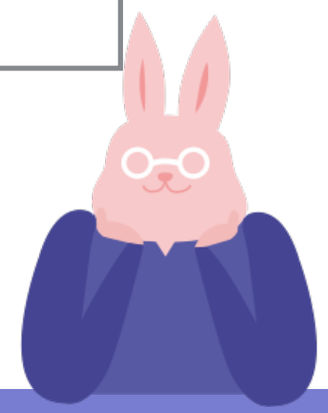
- 시작점에서 출발하여 끝점에 도달하기 위한 최단경로를 출력
  (시작점은 항상 왼쪽 아래, 끝점은 항상 오른쪽 위)

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
```

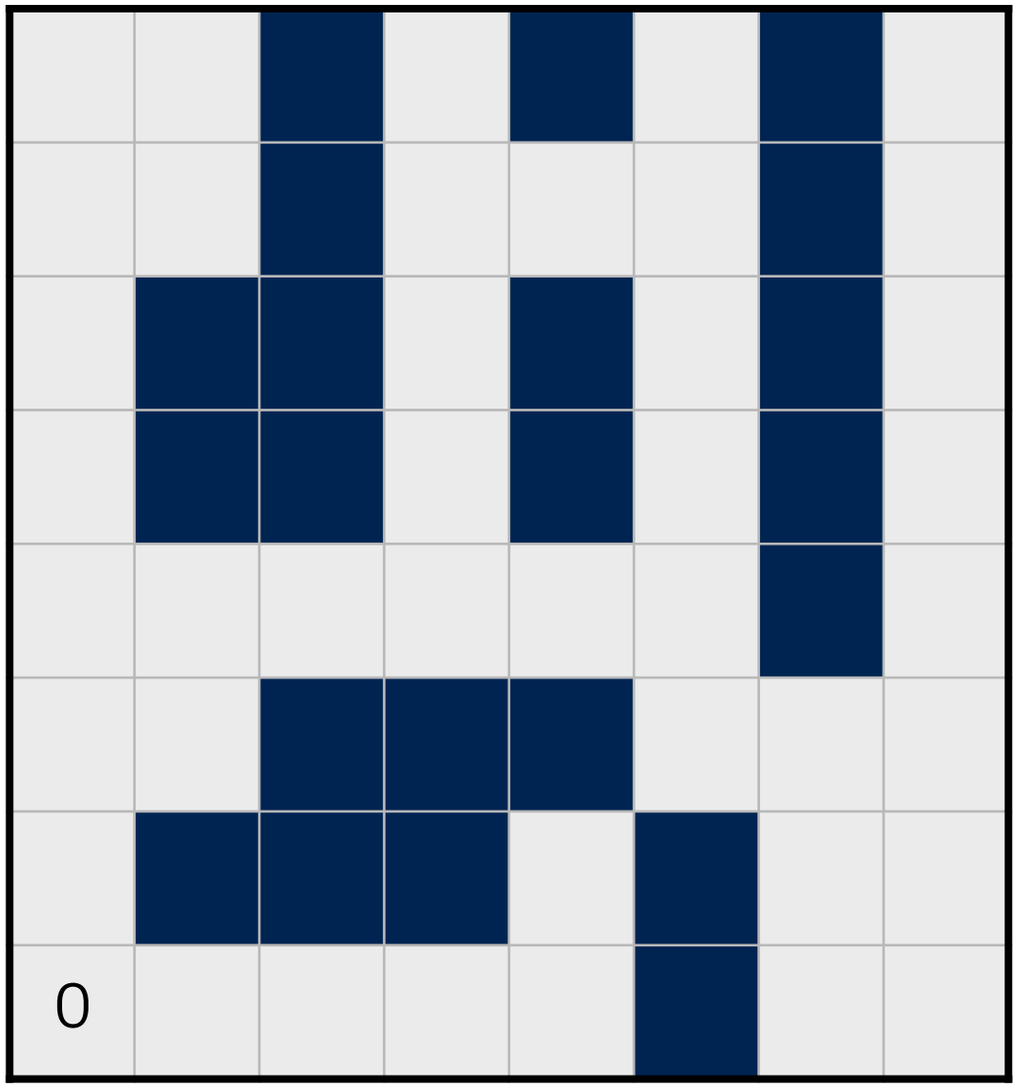
출력의 예

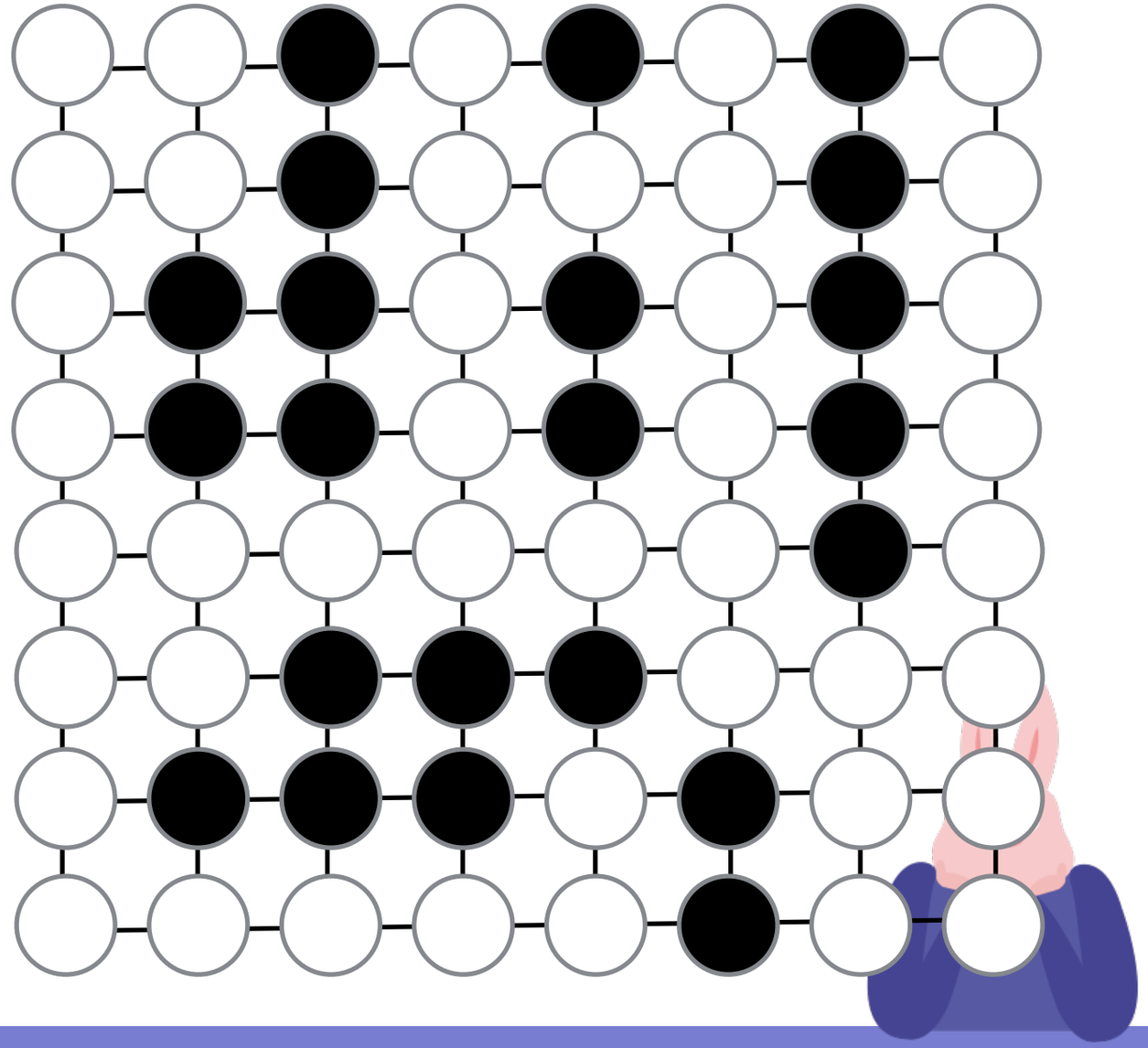
```
11
```



BFS의 응용 : Flood Fill

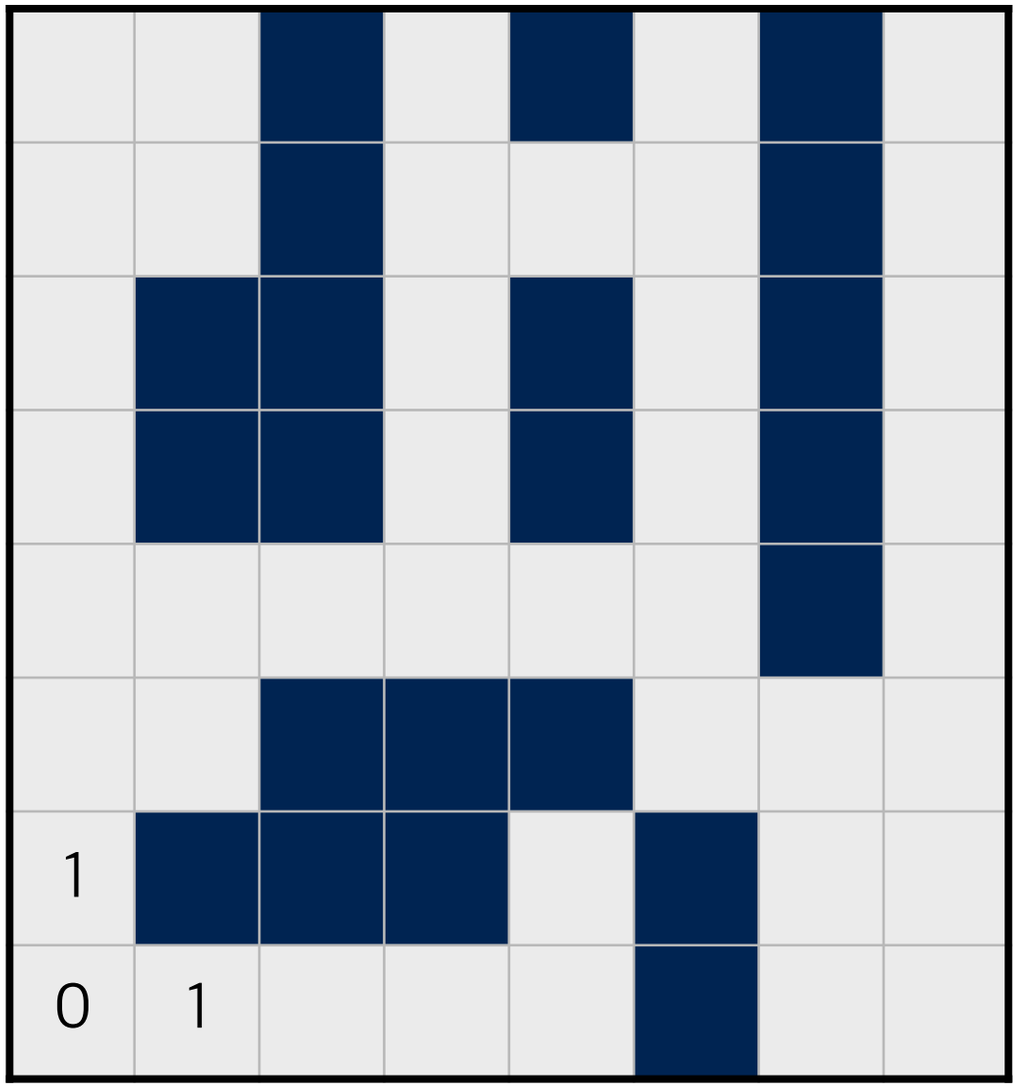
- 물이 차오르는 듯 하여 Flood fill 이라 부름

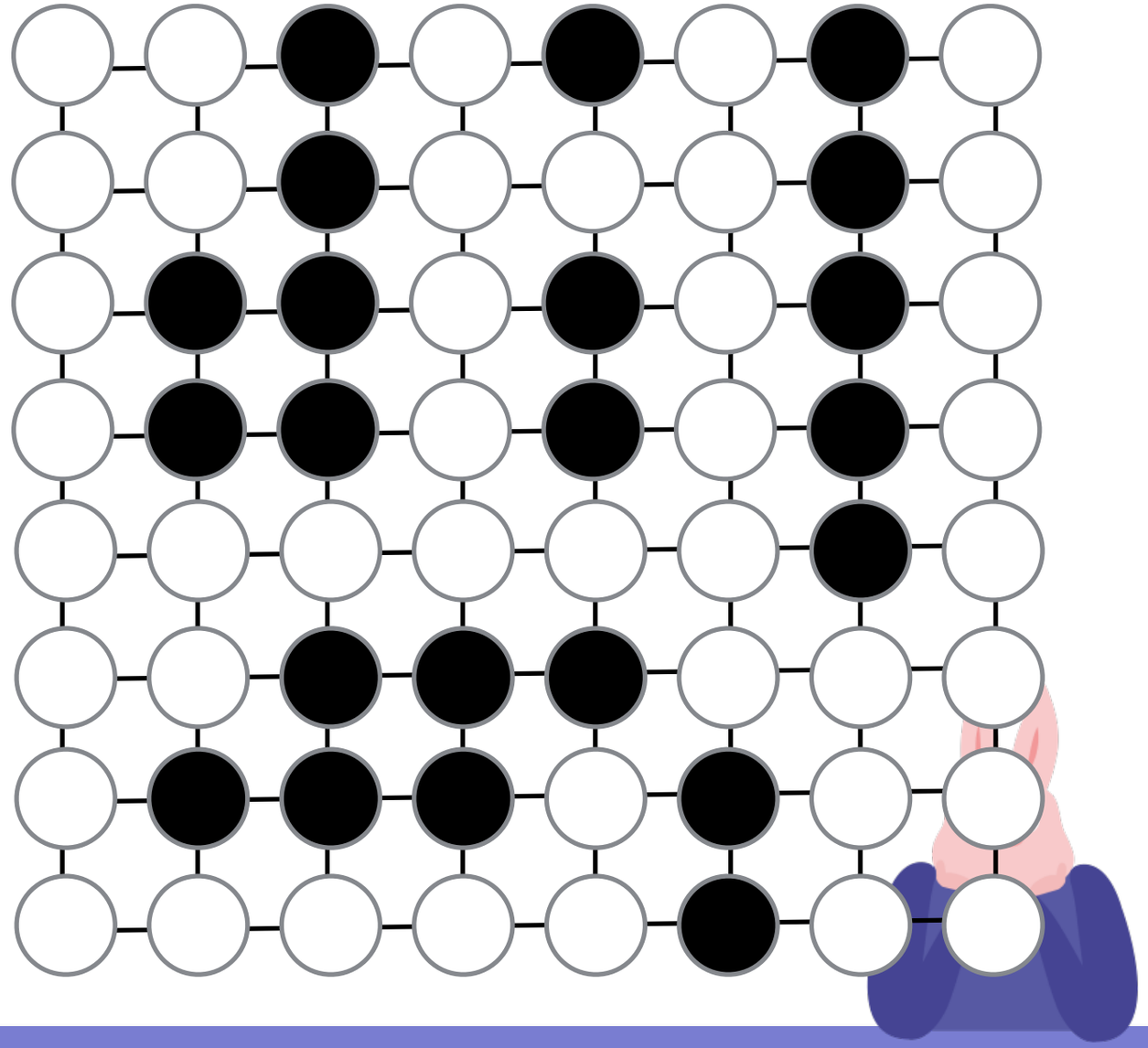




BFS의 응용 : Flood Fill

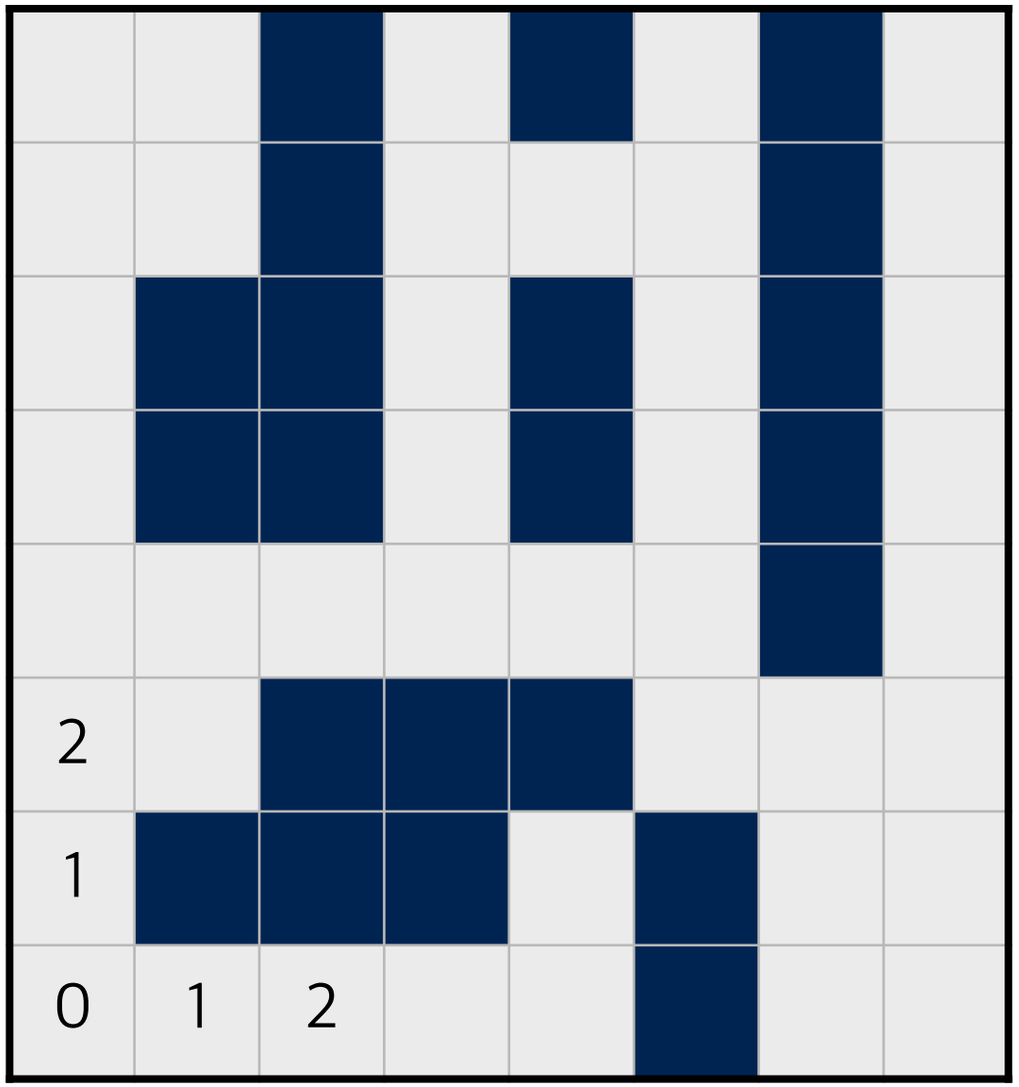
- 물이 차오르는 듯 하여 Flood fill 이라 부름

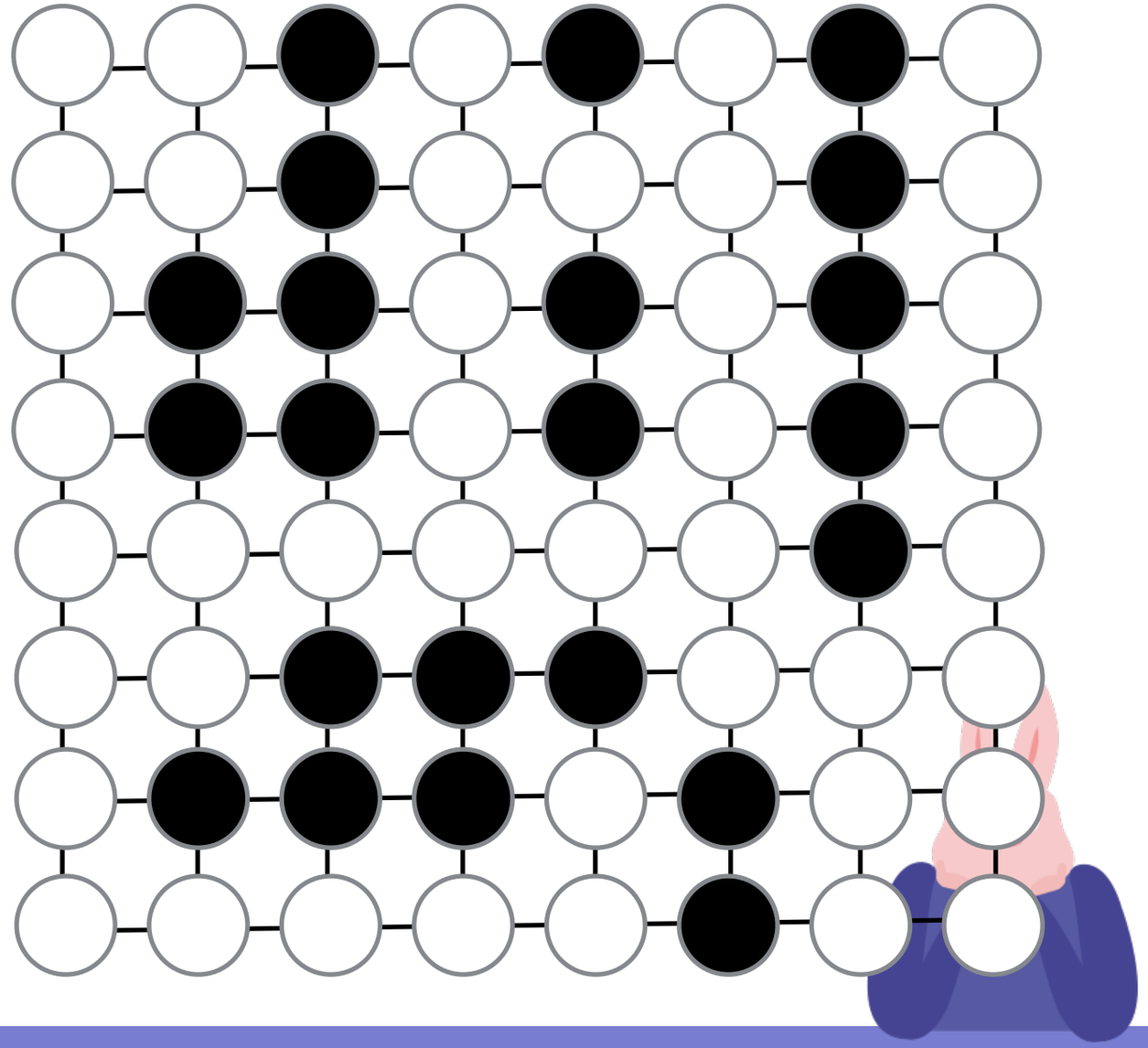




BFS의 응용 : Flood Fill

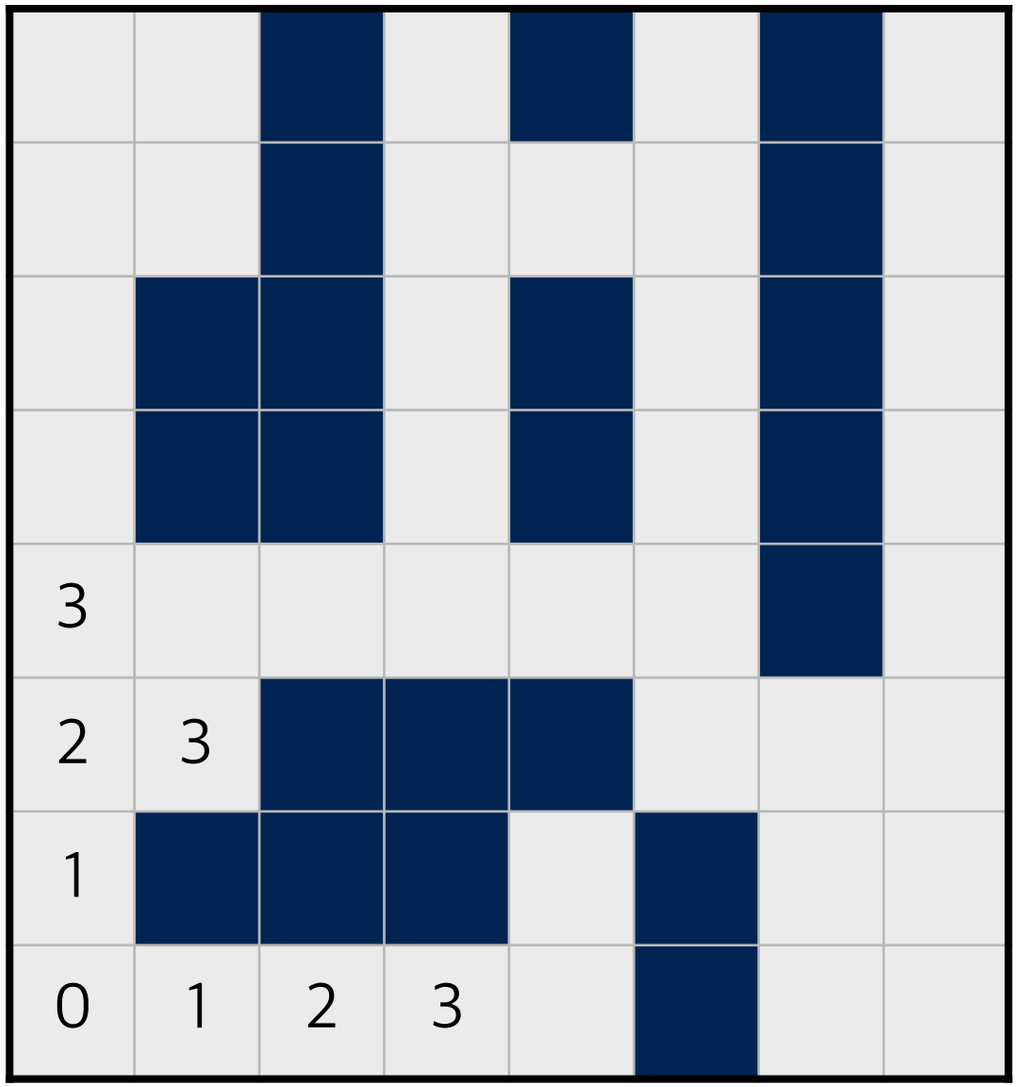
- 물이 차오르는 듯 하여 Flood fill 이라 부름

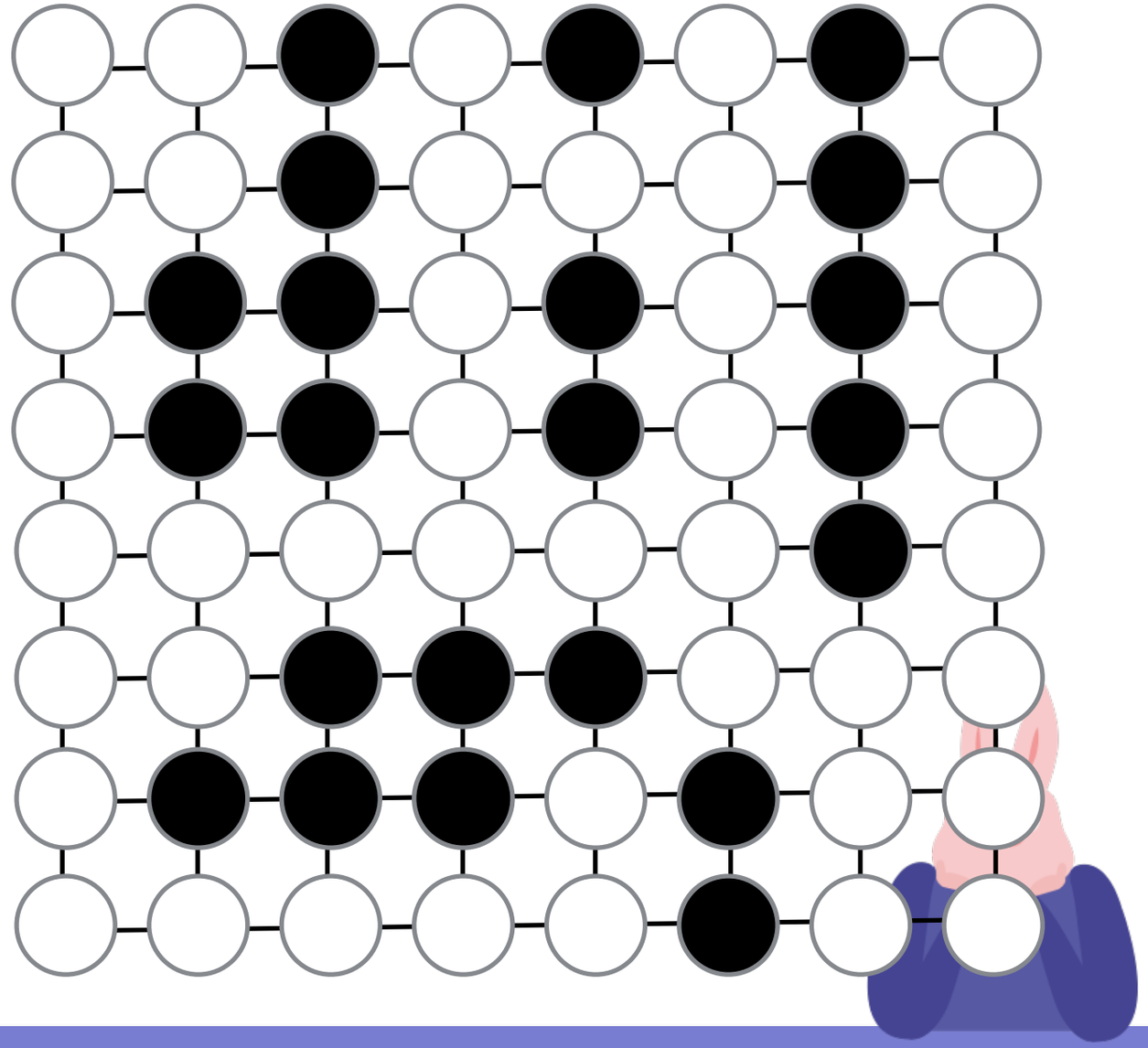




BFS의 응용 : Flood Fill

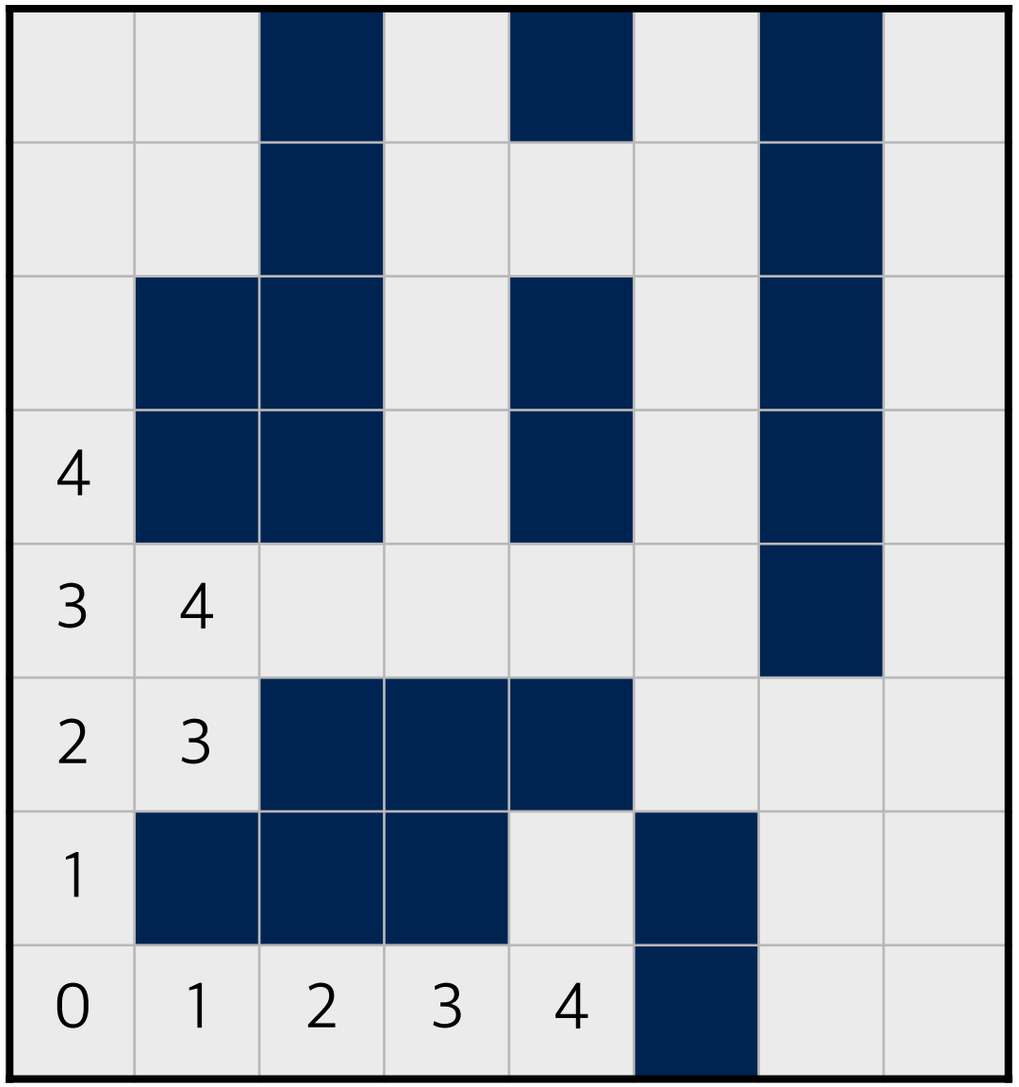
- 물이 차오르는 듯 하여 Flood fill 이라 부름

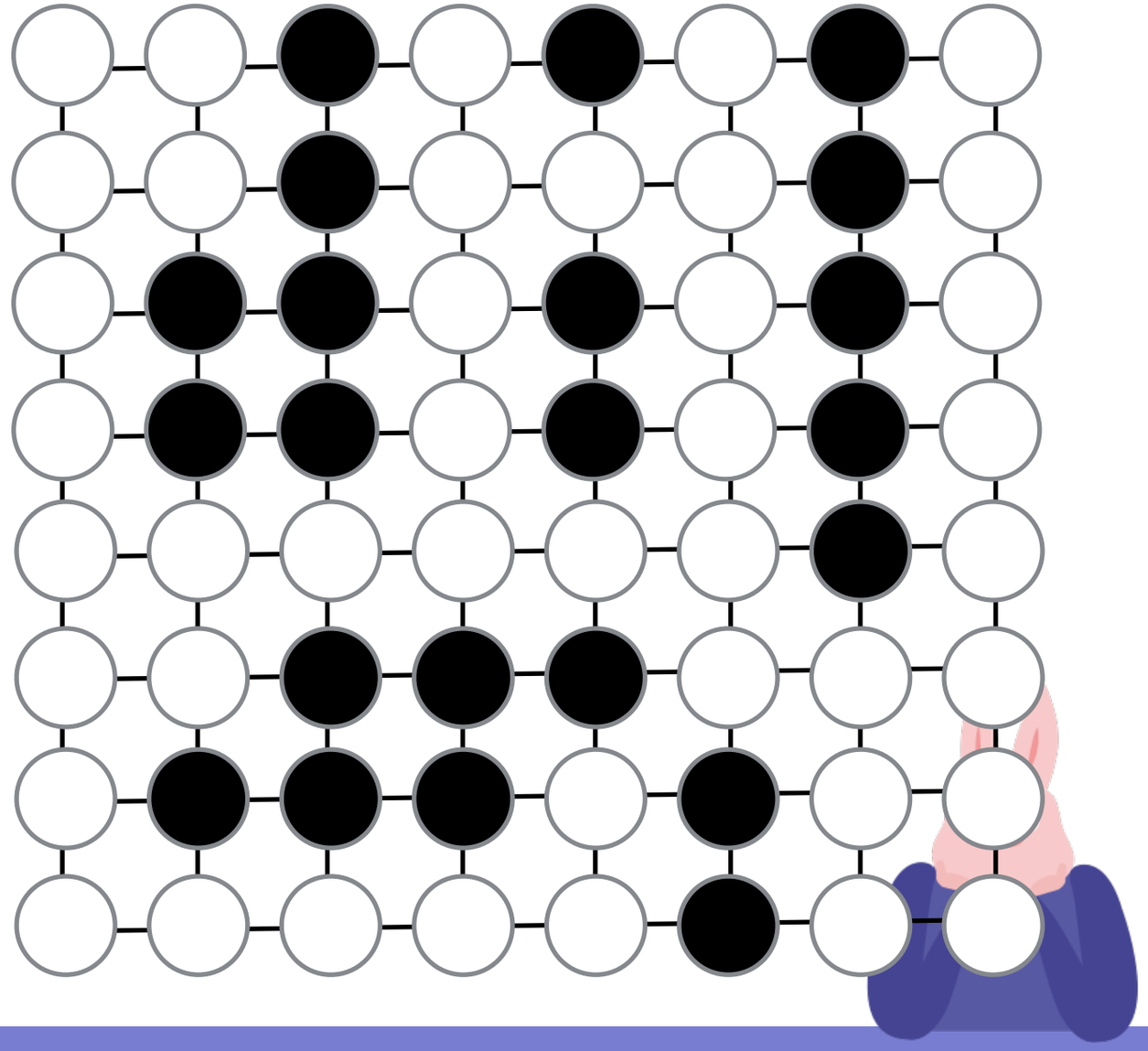




BFS의 응용 : Flood Fill

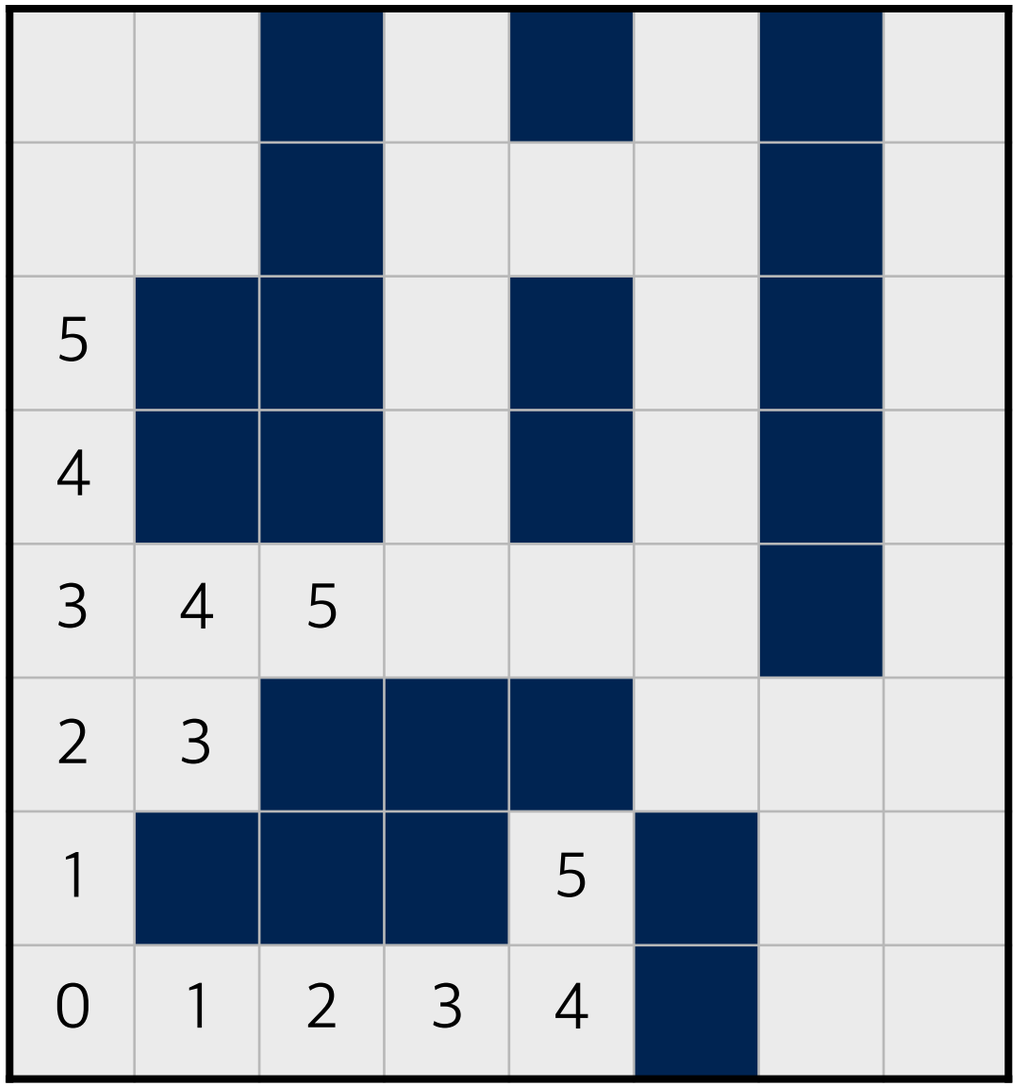
- 물이 차오르는 듯 하여 Flood fill 이라 부름

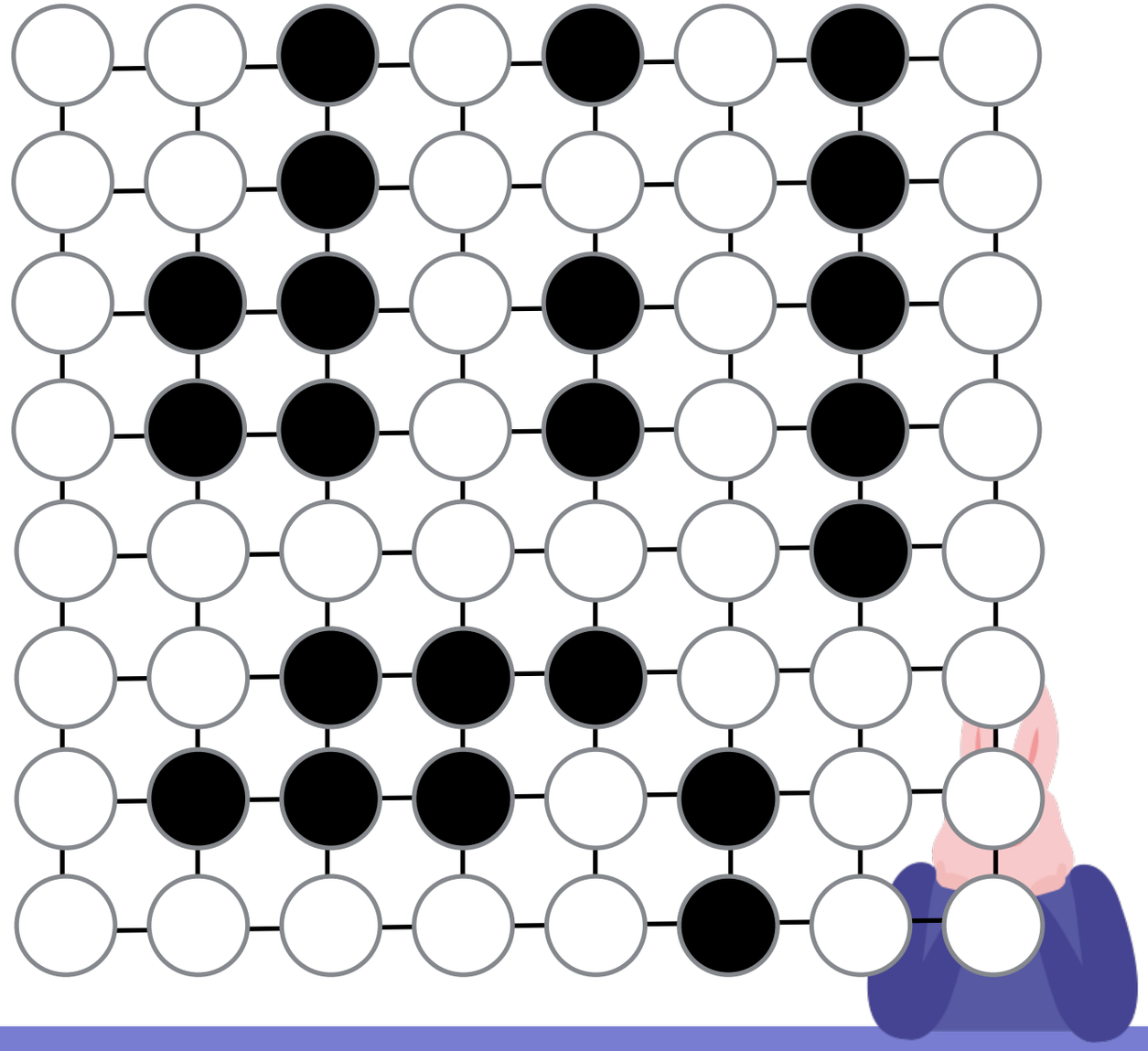




BFS의 응용 : Flood Fill

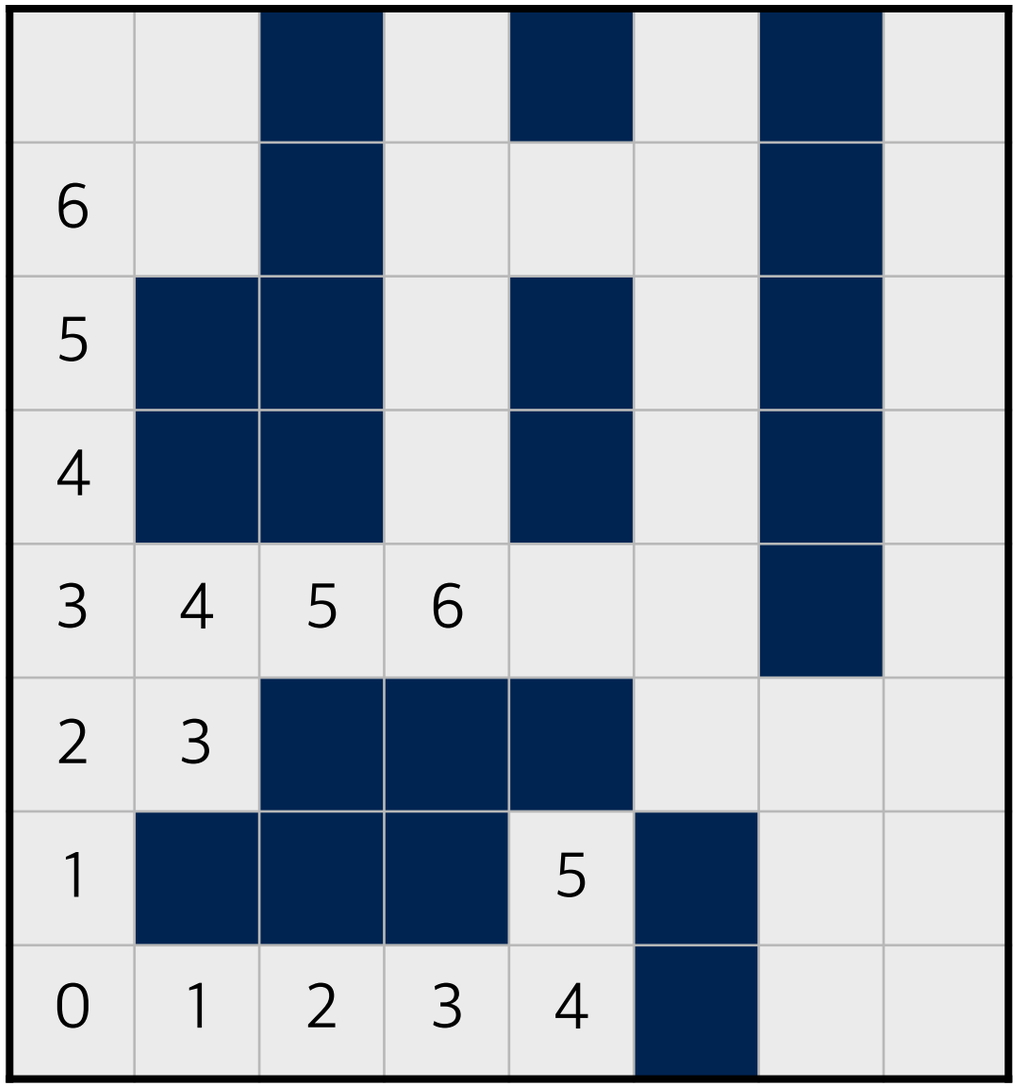
- 물이 차오르는 듯 하여 Flood fill 이라 부름

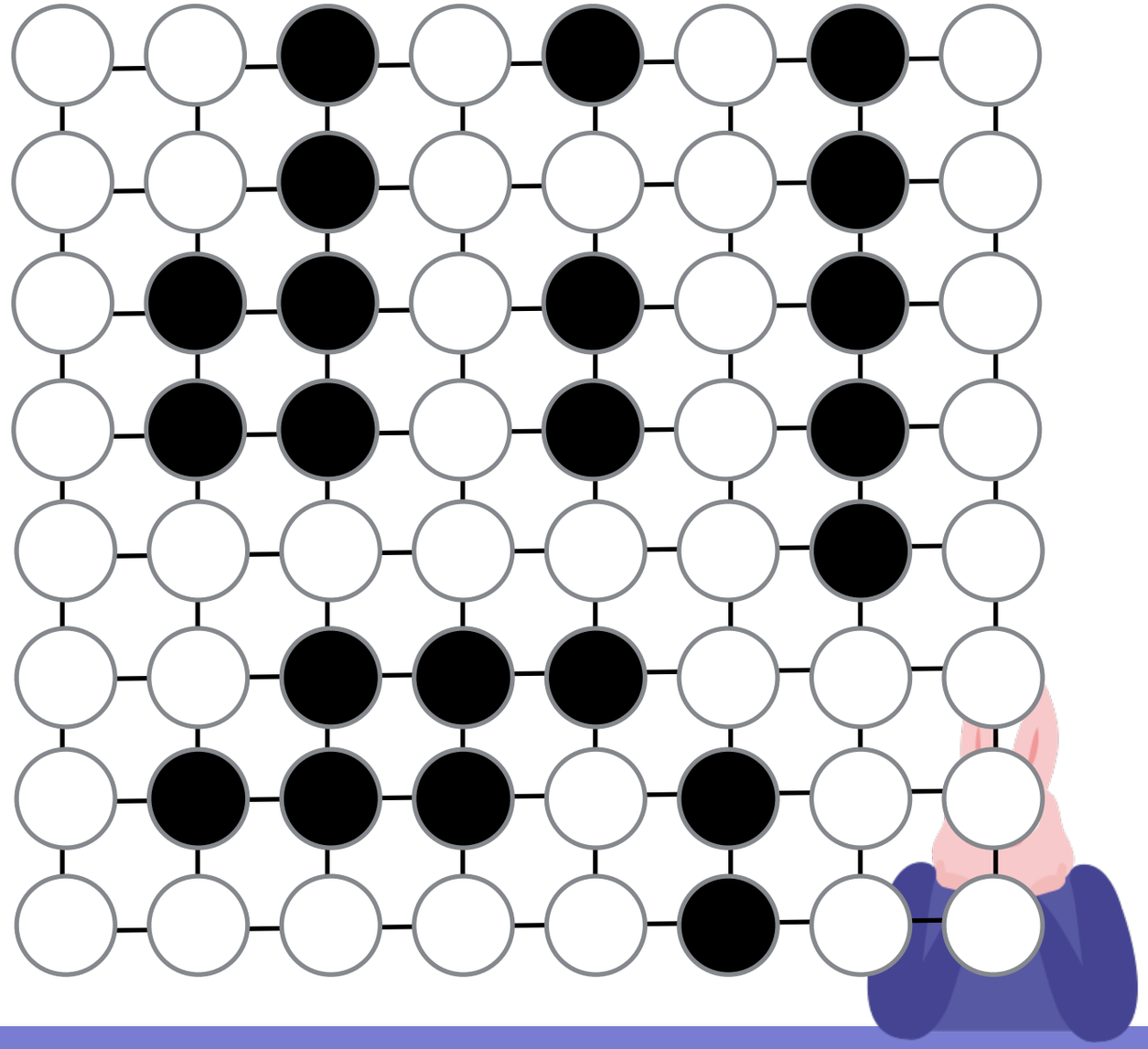




BFS의 응용 : Flood Fill

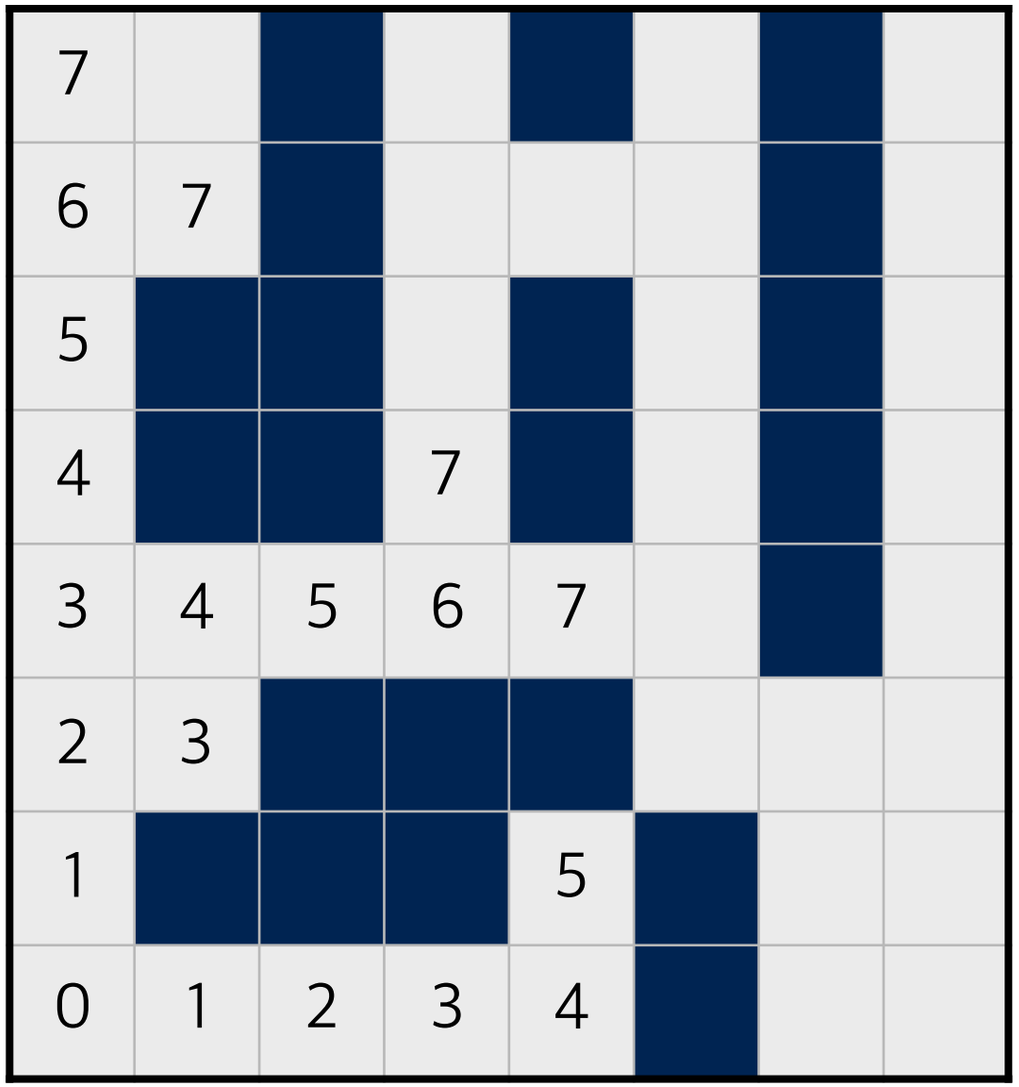
- 물이 차오르는 듯 하여 Flood fill 이라 부름

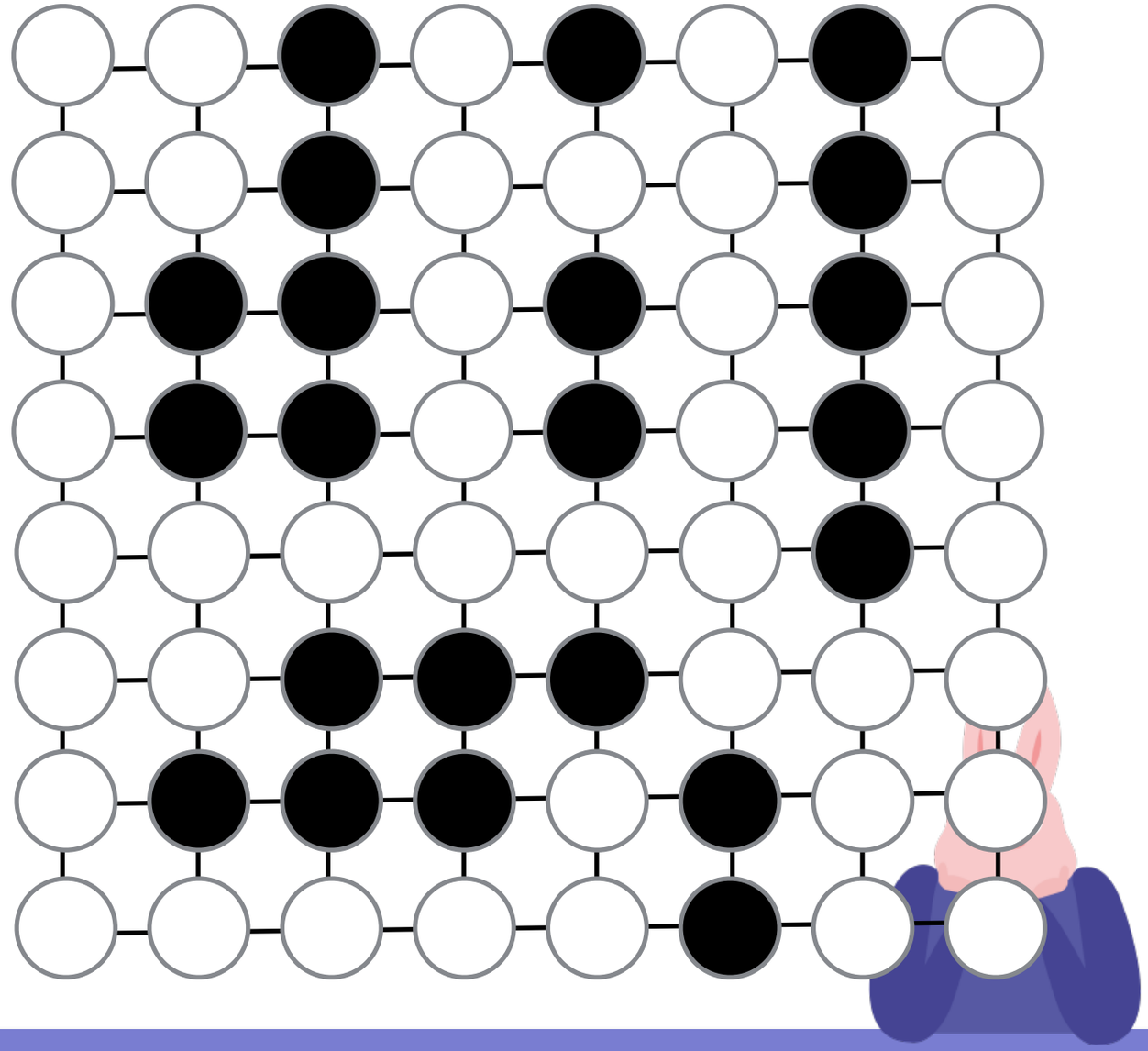




BFS의 응용 : Flood Fill

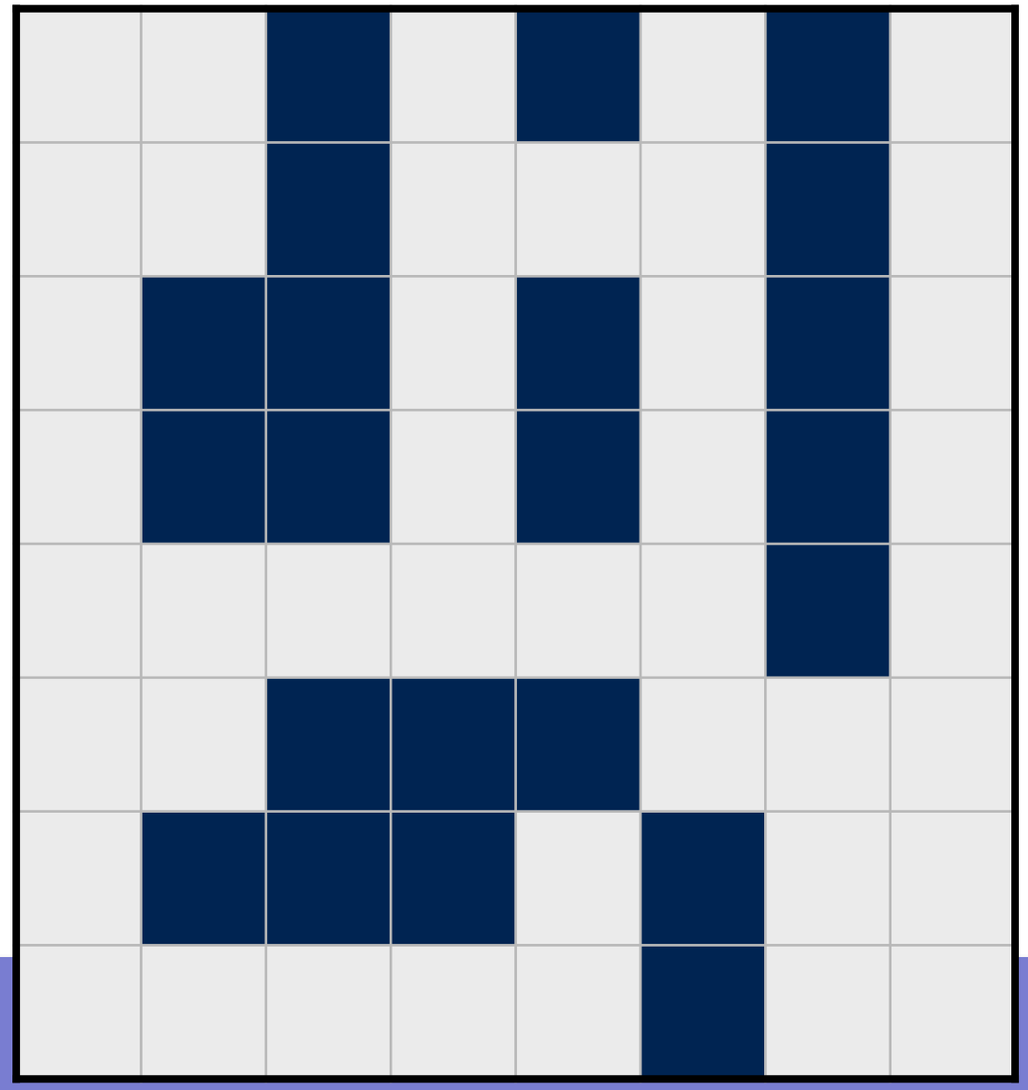
- 물이 차오르는 듯 하여 Flood fill 이라 부름

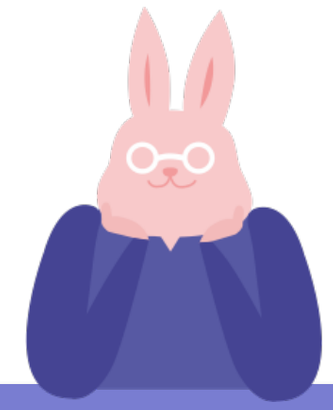




BFS의 응용 : Flood Fill

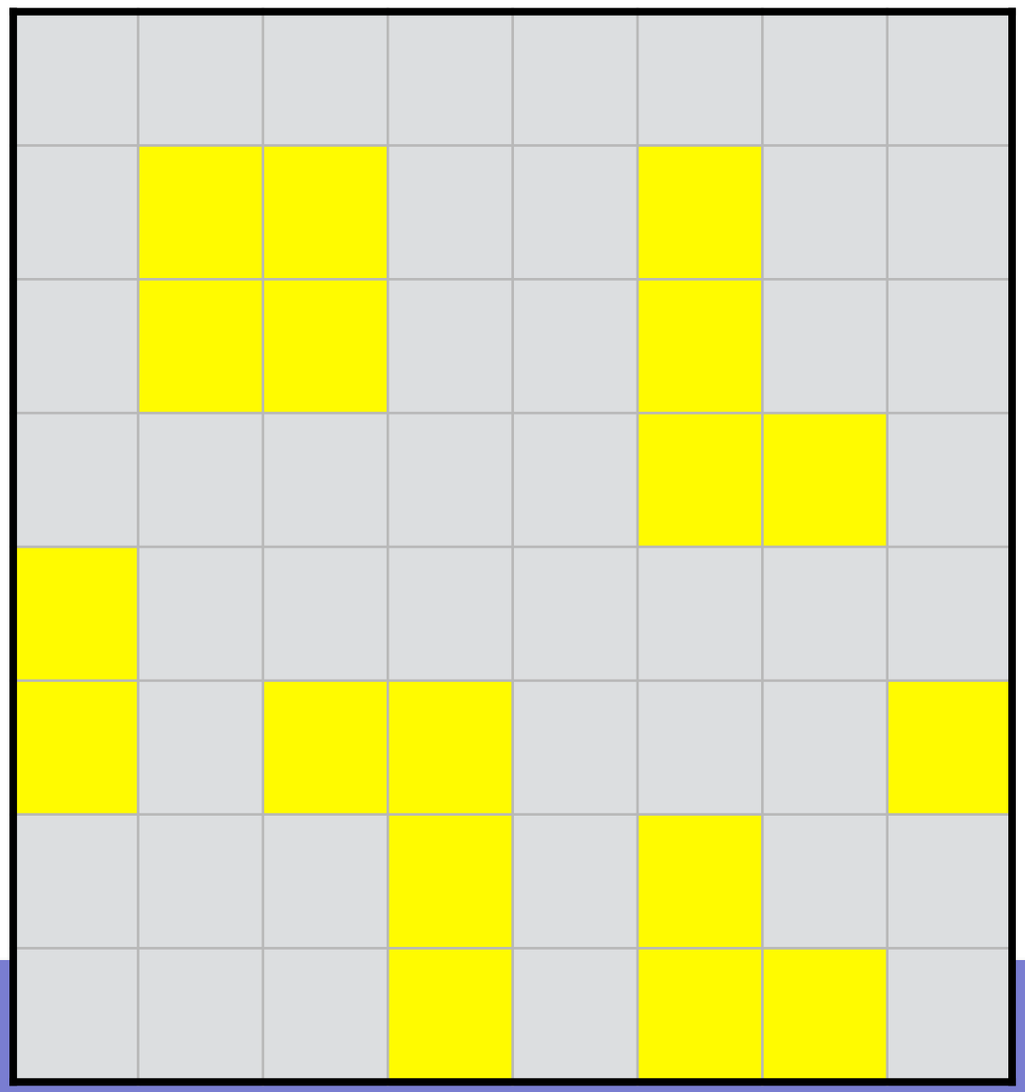
- 언제 쓰나?
  - 인접한 블럭의 집합에 색칠하기

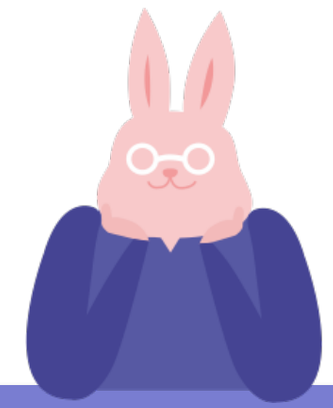




BFS의 응용 : Flood Fill

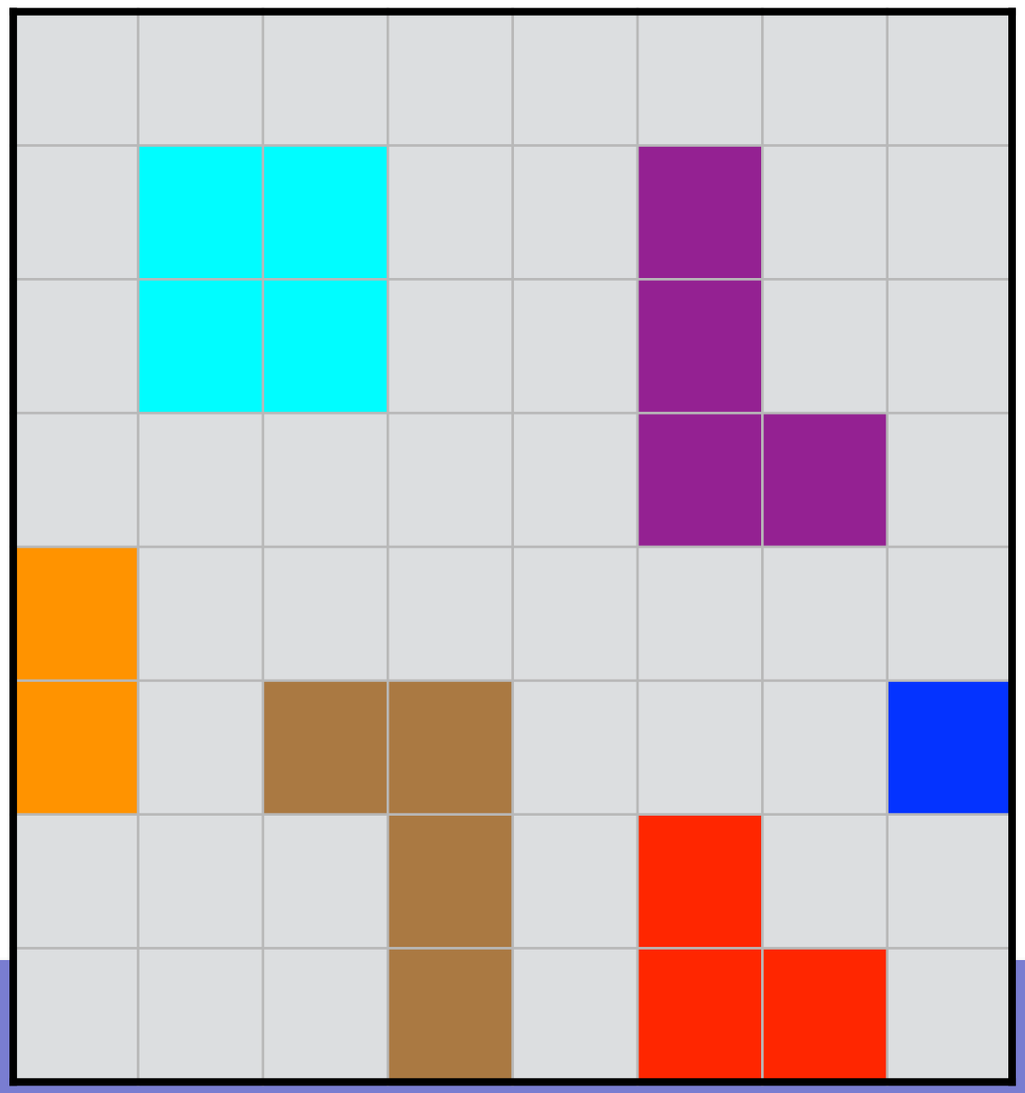
- 언제 쓰나?
  - 아래의 그림에 몇개의 서로 다른 덩어리가 있는가 ?

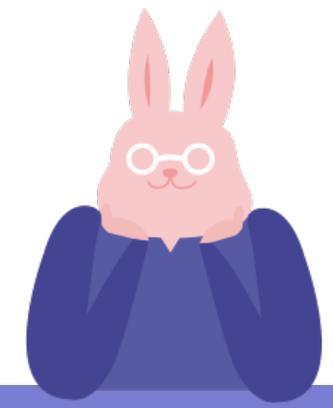




BFS의 응용 : Flood Fill

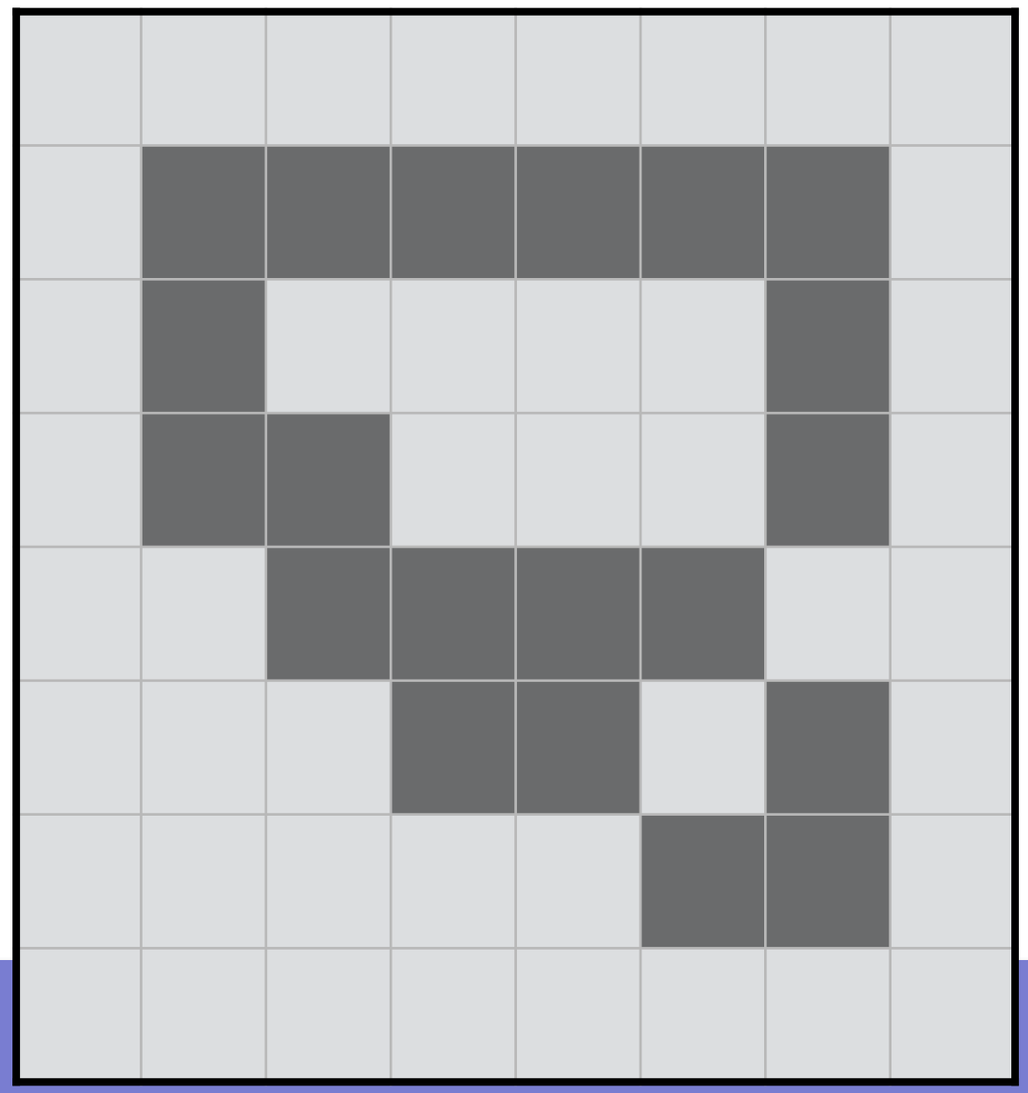
- 언제 쓰나?
  - 아래의 그림에 몇개의 서로 다른 덩어리가 있는가 ? **6개!**

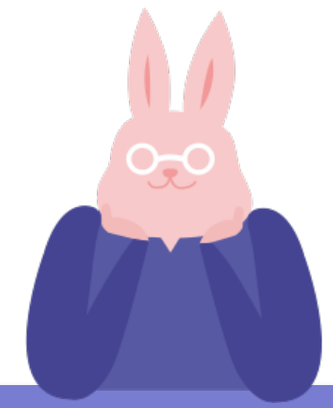




BFS의 응용 : Flood Fill

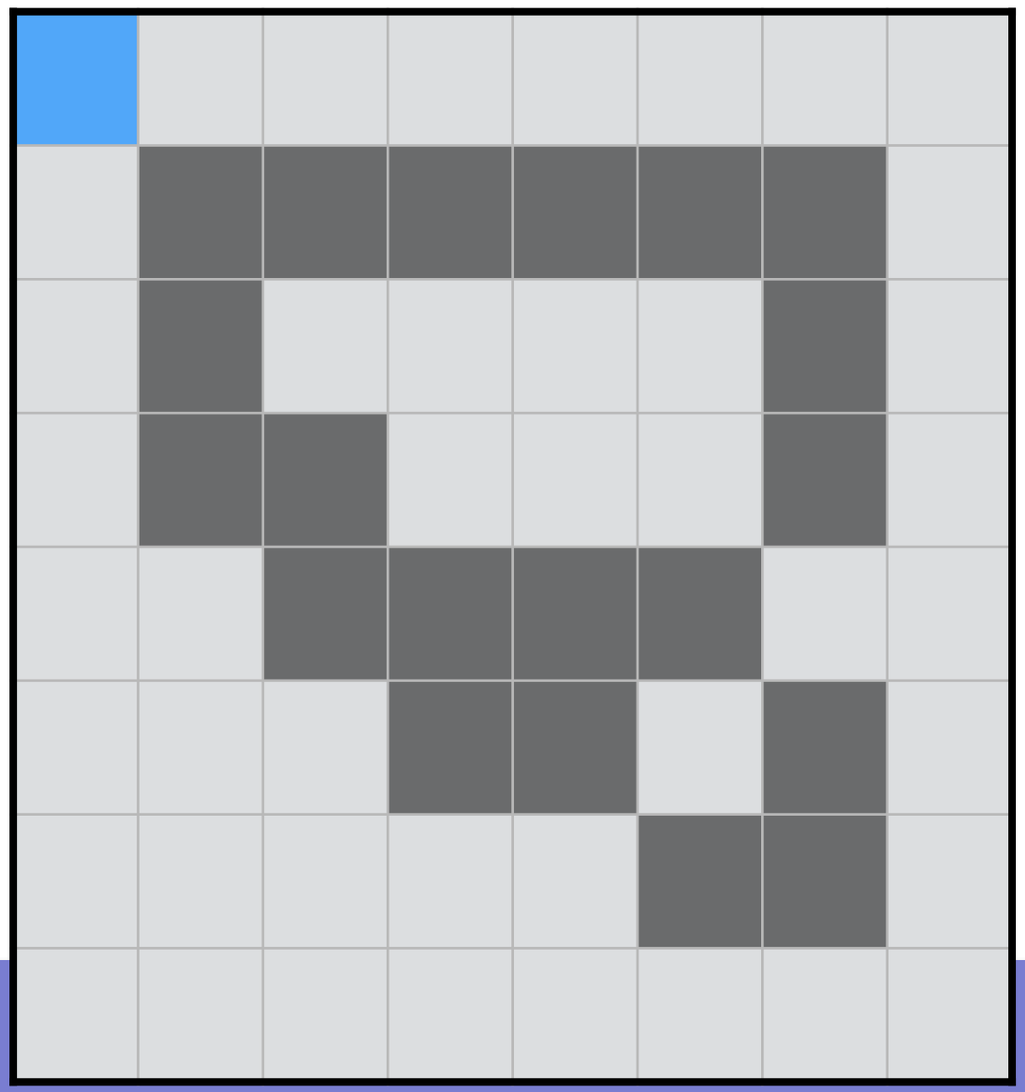
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

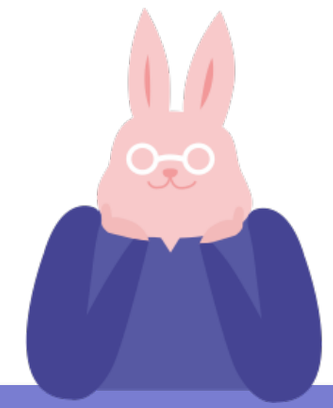




BFS의 응용 : Flood Fill

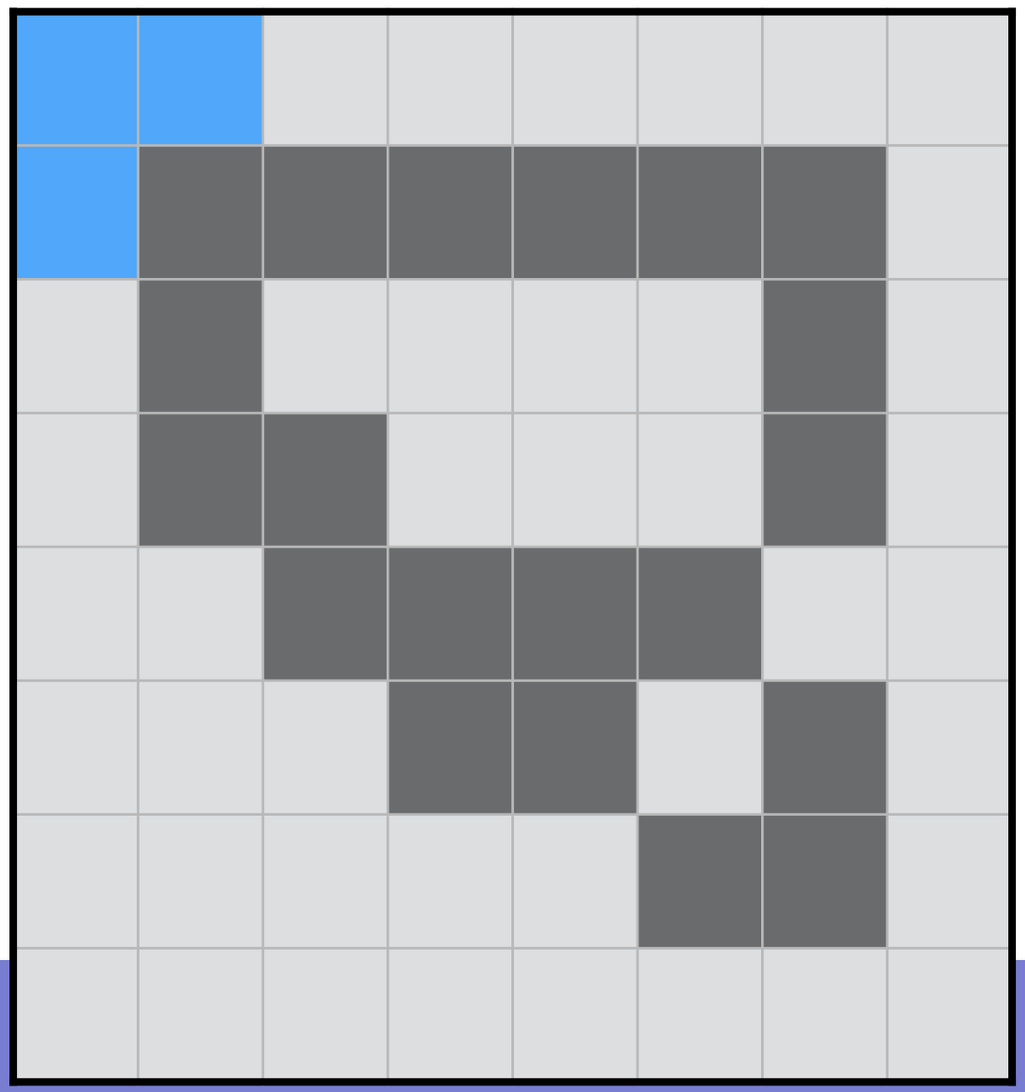
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

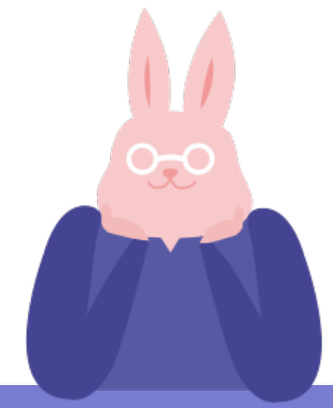




BFS의 응용 : Flood Fill

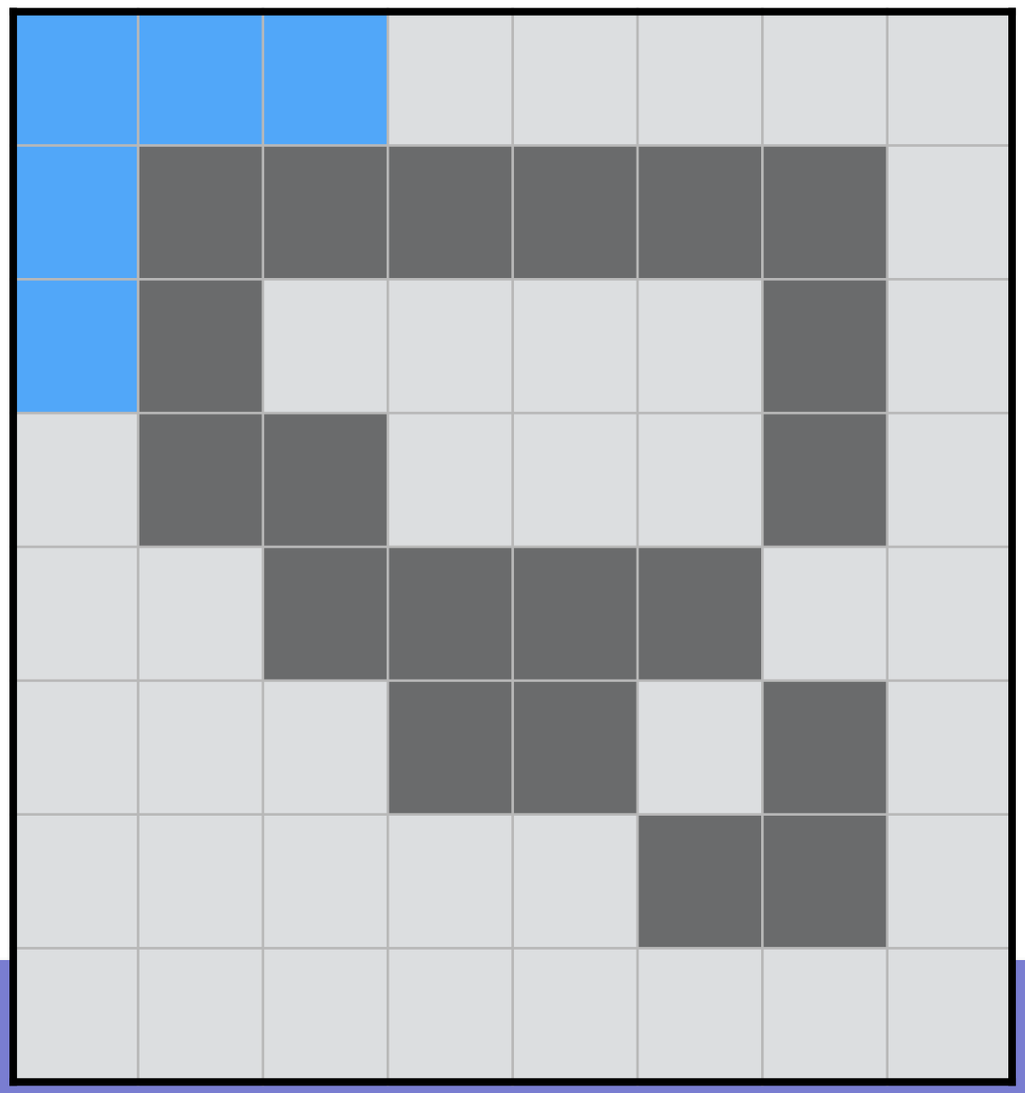
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

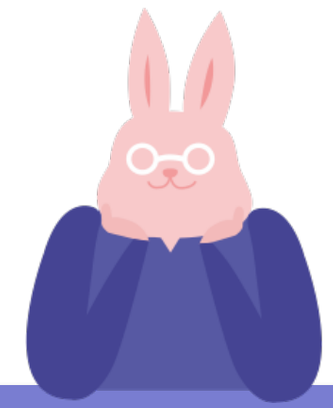




BFS의 응용 : Flood Fill

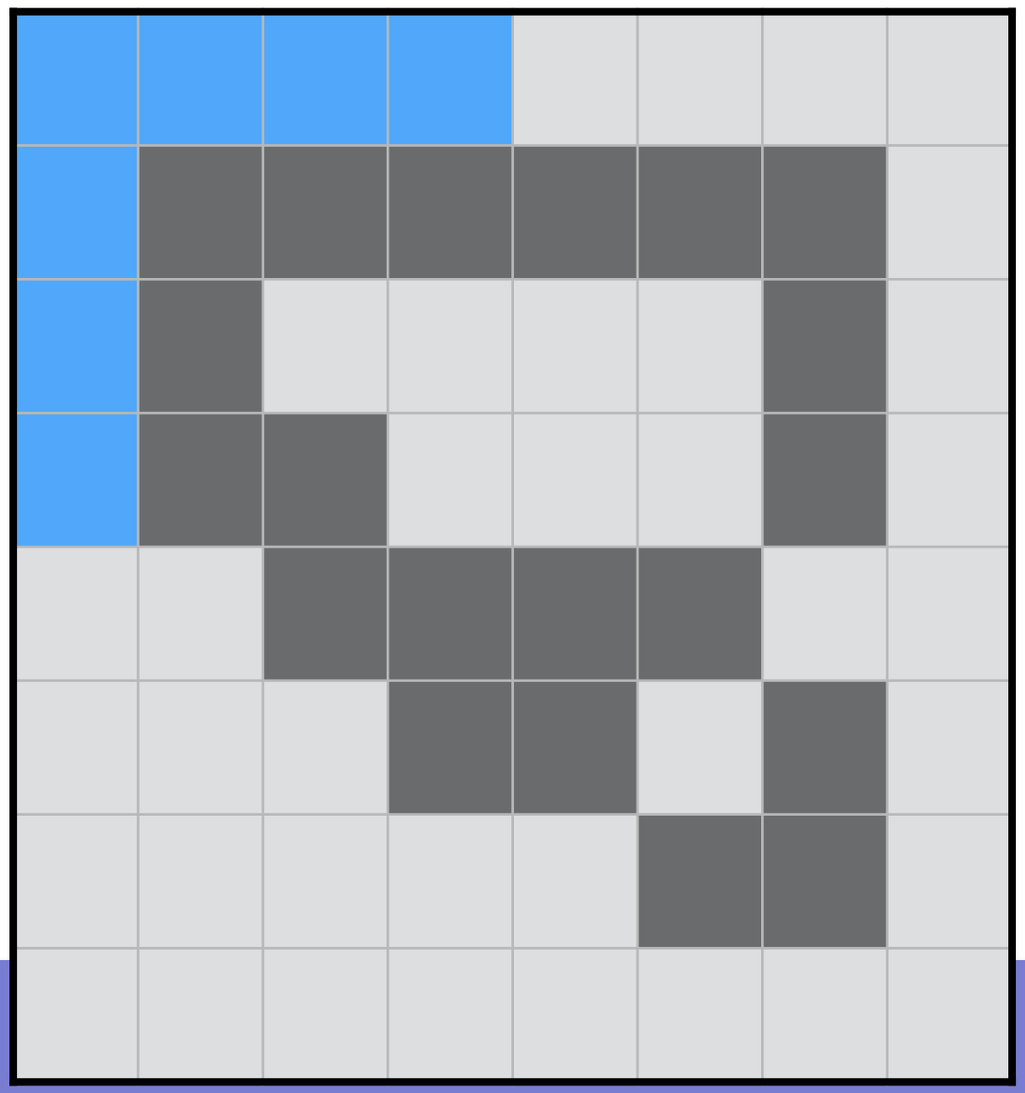
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

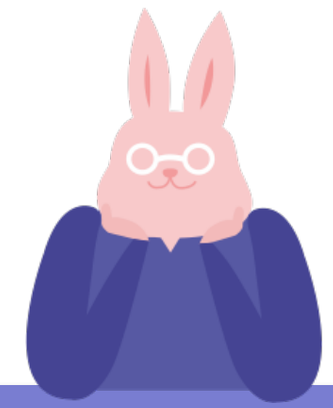




BFS의 응용 : Flood Fill

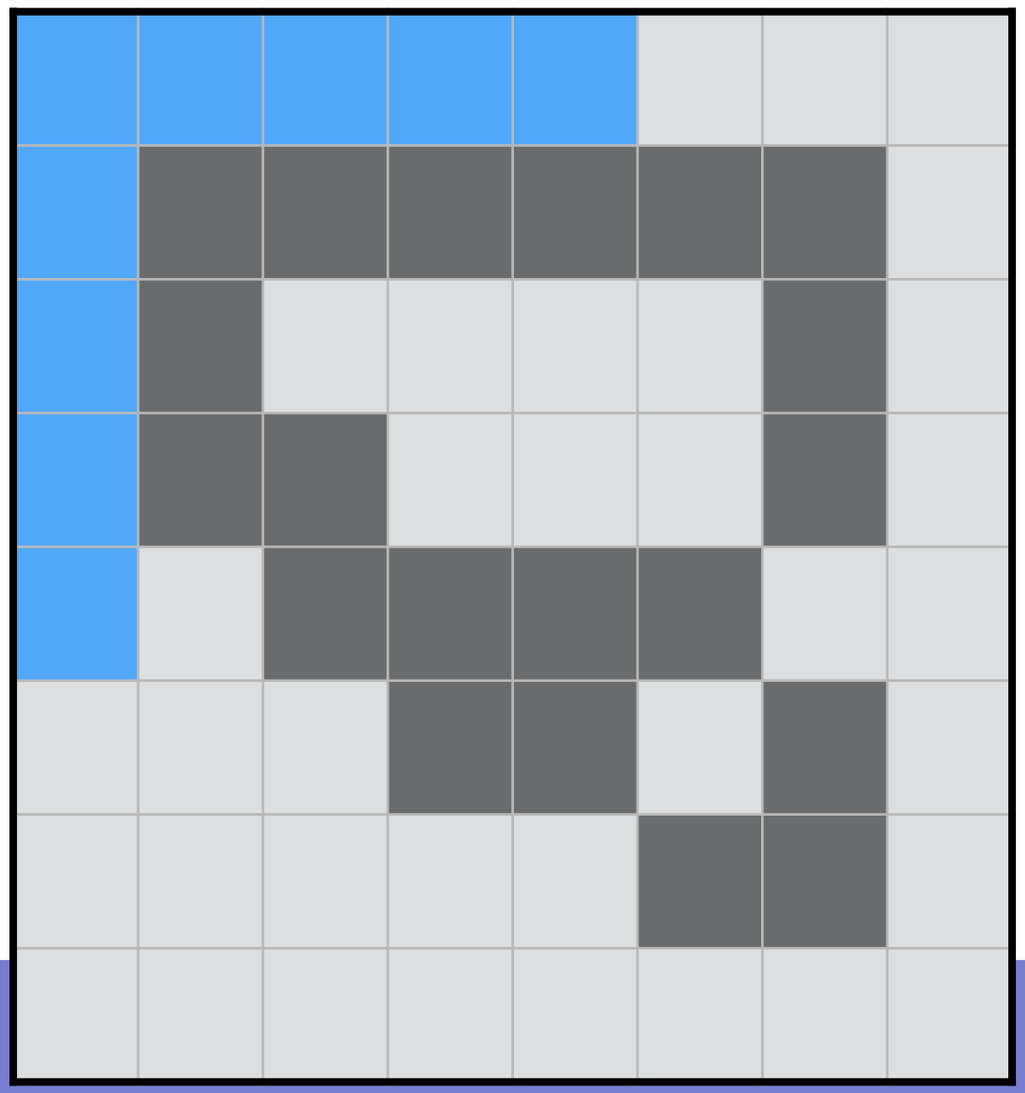
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

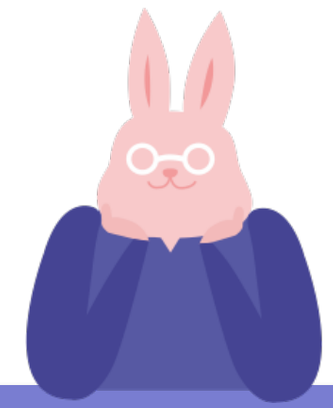




BFS의 응용 : Flood Fill

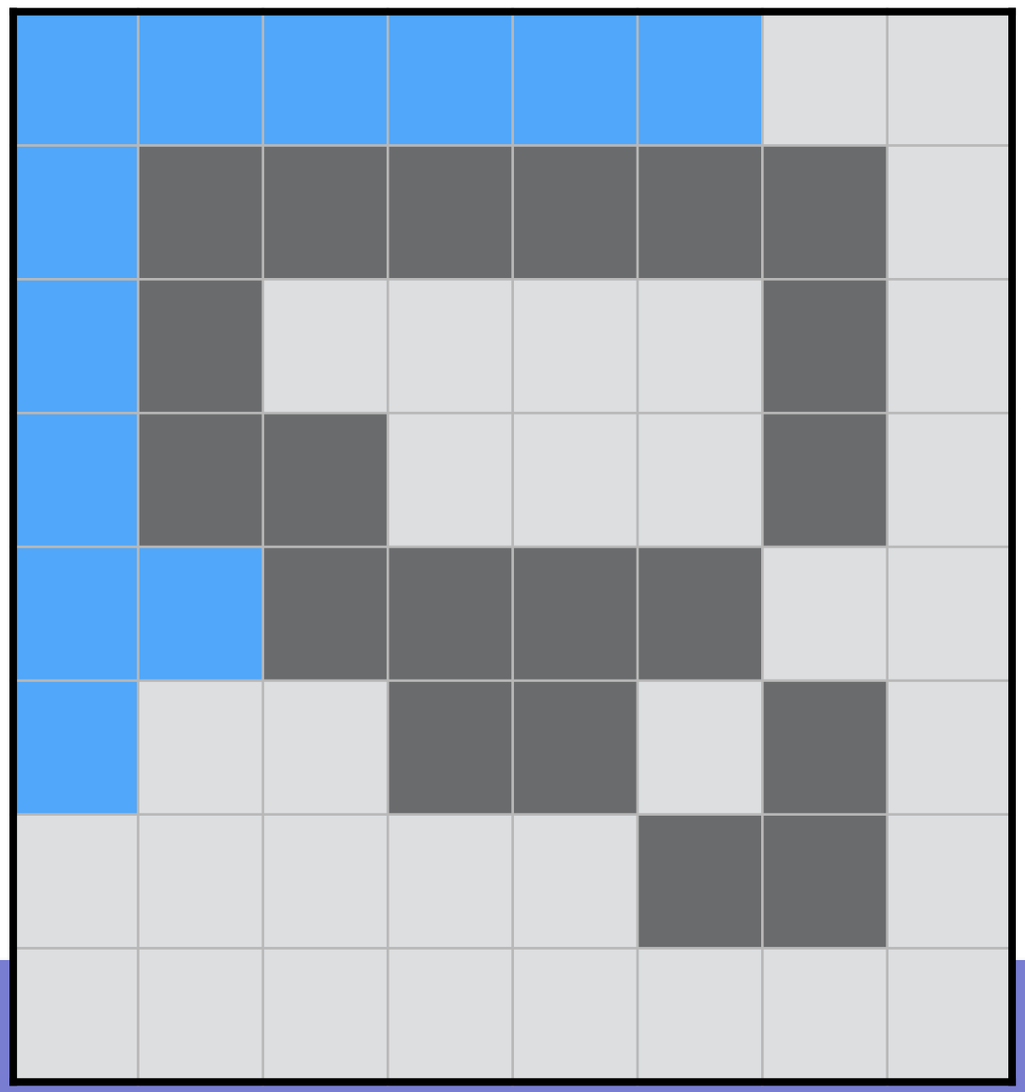
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

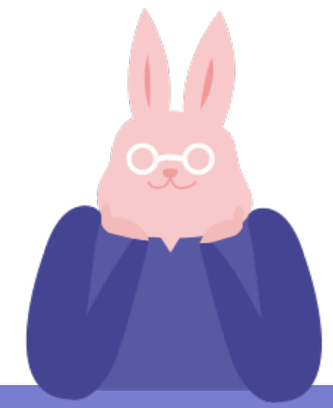




BFS의 응용 : Flood Fill

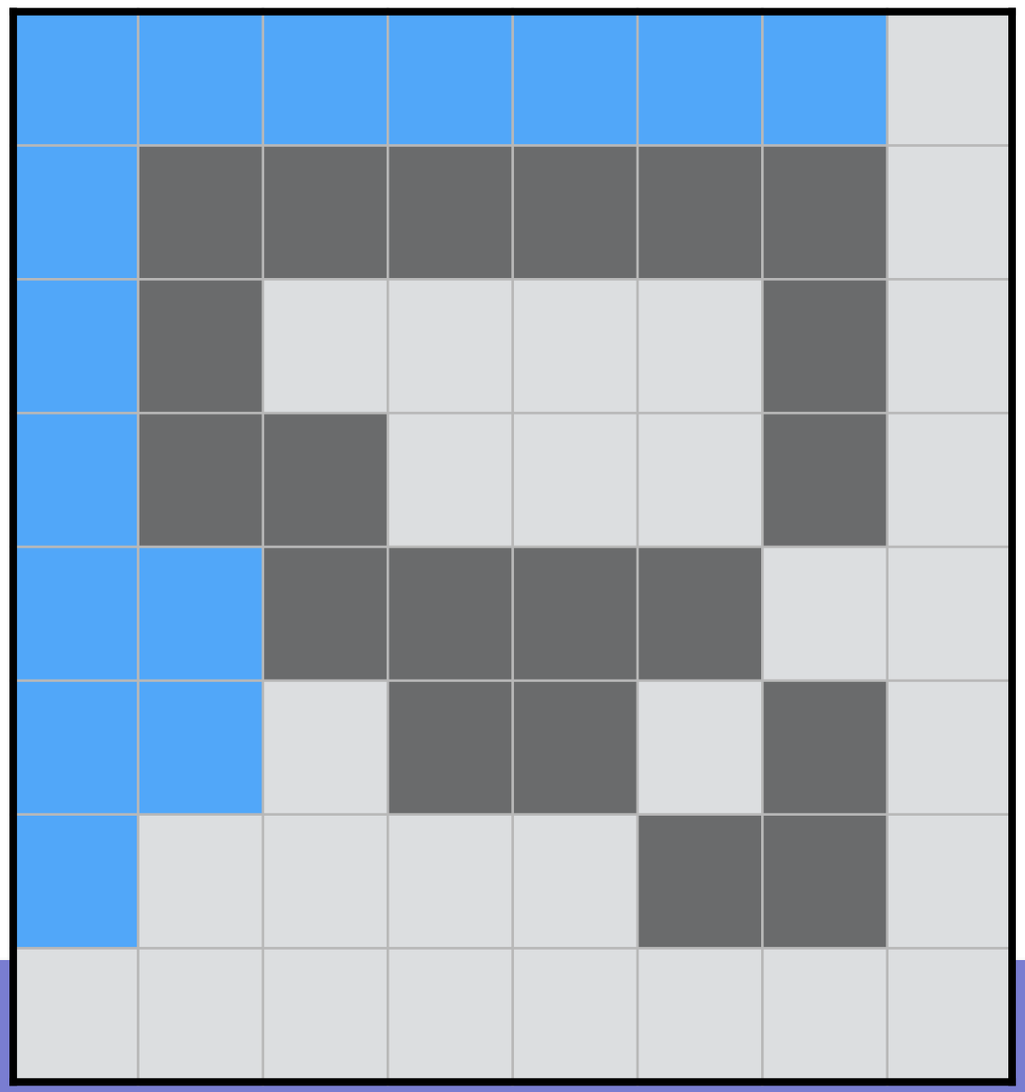
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

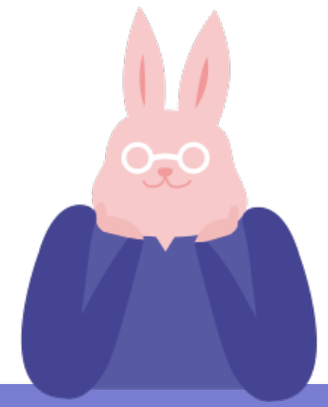




BFS의 응용 : Flood Fill

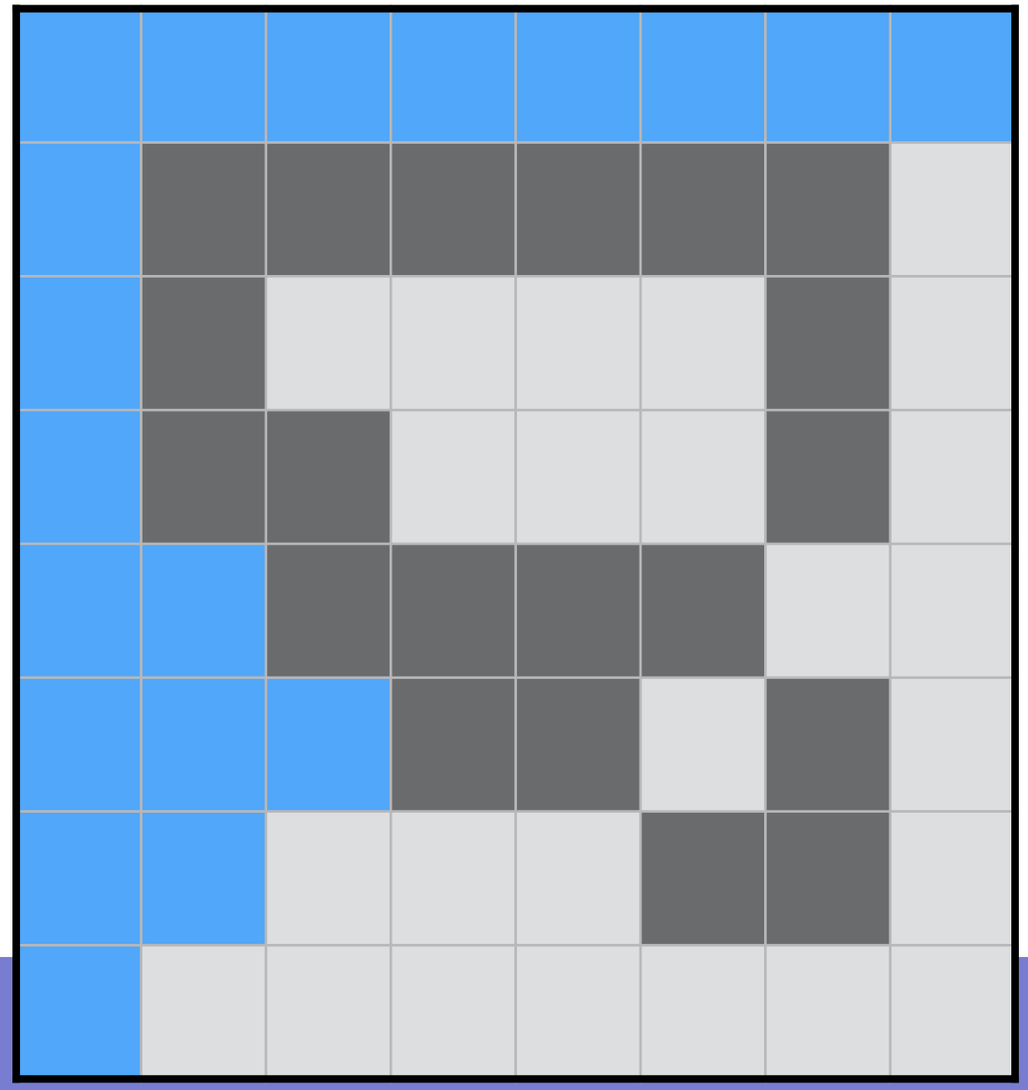
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

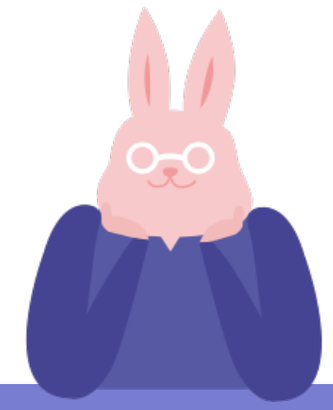




BFS의 응용 : Flood Fill

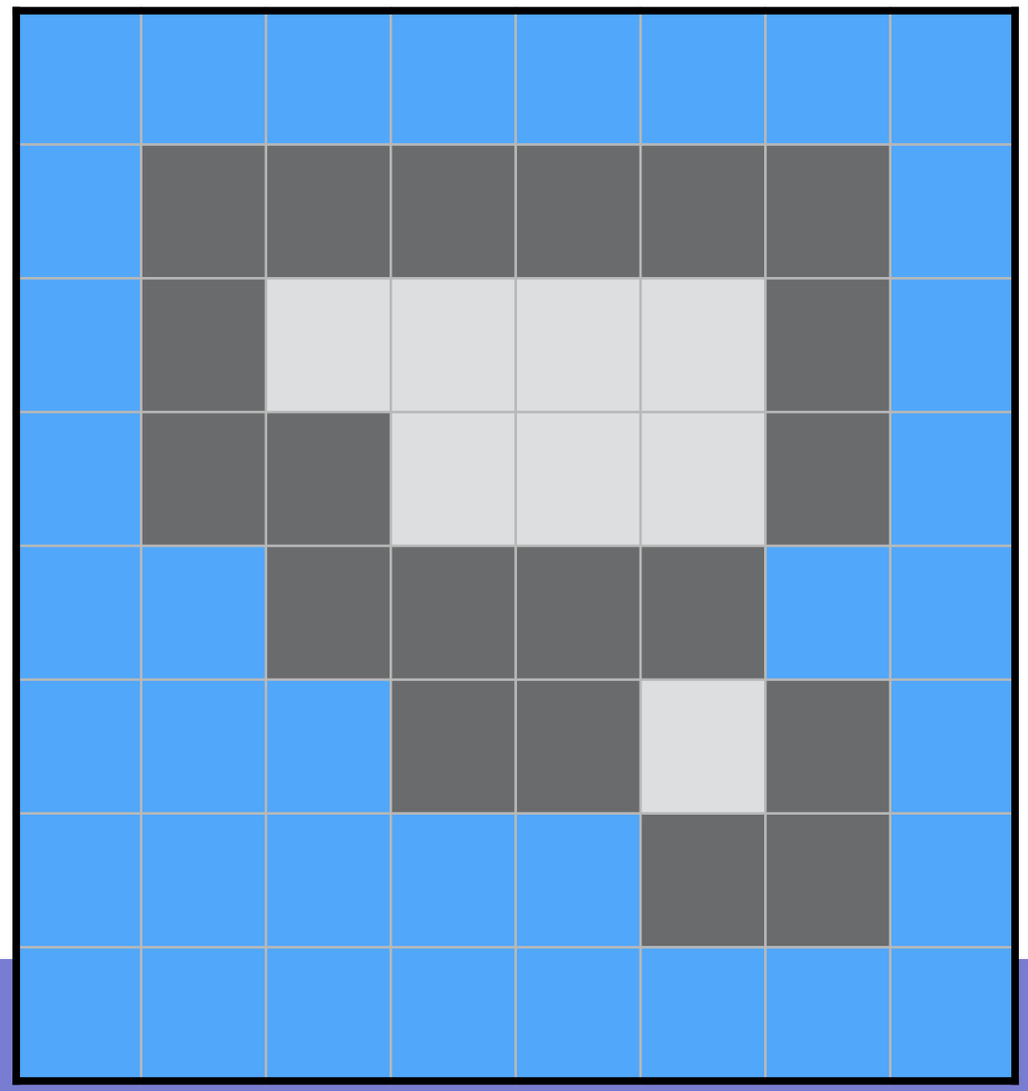
- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라

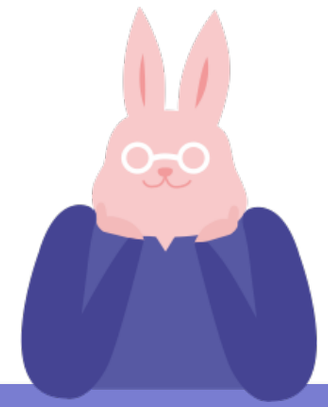




BFS의 응용 : Flood Fill

- 언제 쓰나?
  - 아래의 그림에서 외부공기와 내부공기를 판별하라





/* elice */

감사합니다!

신현규

E-mail : hyungyu.sh@kaist.ac.kr

Kakao : yougatup