

# 자료구조 2

최백준 [choi@startlink.io](mailto:choi@startlink.io)

---

# 스택

---

# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

- 문자열 S에서 폭발 문자열 T를 모두 지우는 문제
- 문자열: mirkovC4nizCC44
- 폭발 문자열: C4
- 결과: mirkovniz

# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

- 스택에 넣는 것은 (문자열의 인덱스, 폭발 문자열에서 인덱스)
- 현재 문자가 폭발 문자열의 첫 번째 문자와 같으면 스택에 넣는다
- 다른 경우에 스택이 비어있으면 그냥 넘어간다
- 다른 경우에 스택이 비어있지 않으면, 스택의 가장 위에 있는 것의 폭발 문자열의 인덱스를 가져온다. 이 인덱스를  $p$ 라고 한다.
- 현재 문자가 폭발 문자열의  $p+1$ 문자와 같으면, 스택에 넣는다. ( $p+1$ 이 폭발 문자열의 마지막 문자면, 폭발 문자열을 찾은 것이다. 스택에서 폭발 문자열을 지운다)
- 다르면, 스택을 모두 비워버린다.

# 문자열 폭발

5

<https://www.acmicpc.net/problem/9935>

- 폭발 문자열의 길이가 1이면, 스택을 사용할 수가 없기 때문에, 그냥 for문을 돌면서 체크해준다.

# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

- 폭발 문자열이 abc인 예시
- abaabcbcd -> ababcd -> abd
- **a**baabcbcd
- 폭발 문자열의 첫 번째 문자와 같기 때문에, 스택에 넣는다.
- 스택: (0, 0)
- a**b**aabcbcd
- 첫 번째 문자와 다르기 때문에, 스택의 가장 위에 있는 것의 폭발 문자열 인덱스를 가져온다.
- 이 인덱스가 0이고, b는 폭발 문자열의 0+1번째 문자와 같기 때문에, 스택에 넣는다.
- 스택: (0, 0), (1, 1)

# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

- ab**a**abcbcd
- 폭발 문자열의 첫 번째 문자와 같기 때문에, 스택에 넣는다.
- 스택: (0, 0), (1, 1), (2, 0)
- aba**a**abcbcd
- 폭발 문자열의 첫 번째 문자와 같기 때문에, 스택에 넣는다.
- 스택: (0, 0), (1, 1), (2, 0), (3, 0)

# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

- abaa**b**cbcd
- 첫 번째 문자와 다르기 때문에, 스택의 가장 위에 있는 것의 폭발 문자열 인덱스를 가져온다.
- 이 인덱스가 0이고, b는 폭발 문자열의 0+1번째 문자와 같기 때문에, 스택에 넣는다.
- 스택: (0, 0), (1, 1), (2, 0), (3, 0), (4, 1)



# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

- abaab**c**bcd
- 첫 번째 문자와 다르기 때문에, 스택의 가장 위에 있는 것의 폭발 문자열 인덱스를 가져온다.
- 이 인덱스가 1이고, c는 폭발 문자열의 0+2번째 문자와 같기 때문에, 스택에 넣는다.
- 스택: (0, 0), (1, 1), (2, 0), (3, 0), (4, 1), (5, 2)
- 2는 폭발 문자열의 마지막 인덱스이기 때문에, 총 3개를 스택에서 뺀다.
- 스택: (0, 0), (1, 1), (2, 0), **(3, 0), (4, 1), (5, 2)**
- 3, 4, 5번째 문자는 사라져야 한다.
- 스택: (0, 0), (1, 1), (2, 0)

# 문자열 폭발

10

<https://www.acmicpc.net/problem/9935>

- abaabc**b**cd
- 첫 번째 문자와 다르기 때문에, 스택의 가장 위에 있는 것의 폭발 문자열 인덱스를 가져온다.
- 이 인덱스가 0이고, b는 폭발 문자열의 0+1번째 문자와 같기 때문에, 스택에 넣는다.
- 스택: (0, 0), (1, 1), (2, 0), (6, 1)

# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

- abaabcb**c**d
- 첫 번째 문자와 다르기 때문에, 스택의 가장 위에 있는 것의 폭발 문자열 인덱스를 가져온다.
- 이 인덱스가 1이고, c는 폭발 문자열의 0+2번째 문자와 같기 때문에, 스택에 넣는다.
- 스택: (0, 0), (1, 1), (2, 0), (6, 1), (7, 2)
- 2는 폭발 문자열의 마지막 인덱스이기 때문에, 총 3개를 스택에서 뺀다.
- 스택: (0, 0), (1, 1), **(2, 0), (6, 1), (7, 2)**
- 2, 6, 7번째 문자는 사라져야 한다.
- 스택: (0, 0), (1, 1)

# 문자열 폭발

12

<https://www.acmicpc.net/problem/9935>

- abaabcbcd
- 첫 번째 문자와 다르기 때문에, 스택의 가장 위에 있는 것의 폭발 문자열 인덱스를 가져온다.
- 스택: (0, 0), (1, 1)
- 이 인덱스가 1이고, d는 폭발 문자열의 0+2번째 문자와 다르기 때문에, 스택을 비운다.
- 스택: (비어있음)

# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

- 스택에 넣고 빼는 과정에서 지워져야 하는 문자로 체크하지 않은 글자를 모두 출력하면 된다.
- 이 방법의 예외는 폭발 문자열의 길이가 1일 때이다.
- 이 때는, 한 글자 이기 때문에, 그냥 지우면 된다.

# 문자열 폭발

<https://www.acmicpc.net/problem/9935>

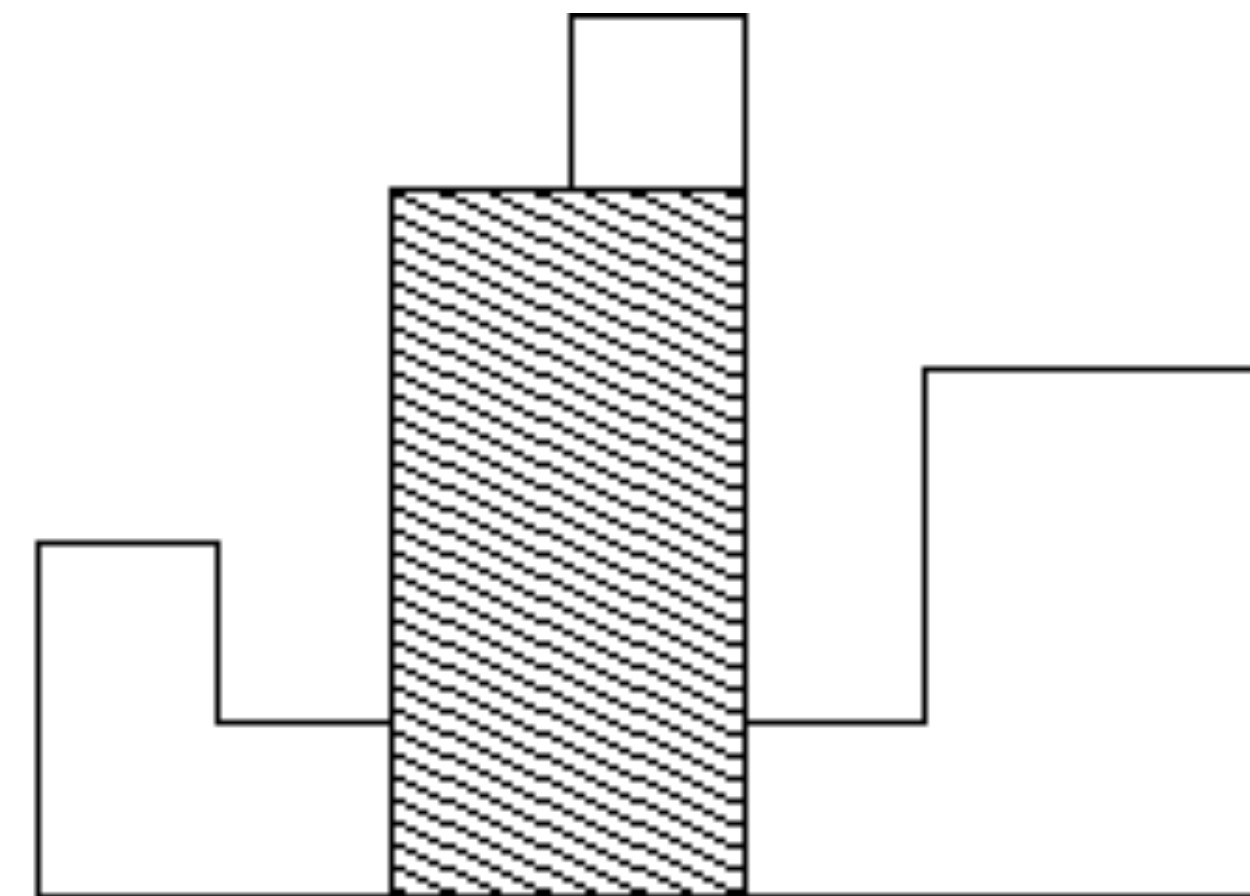
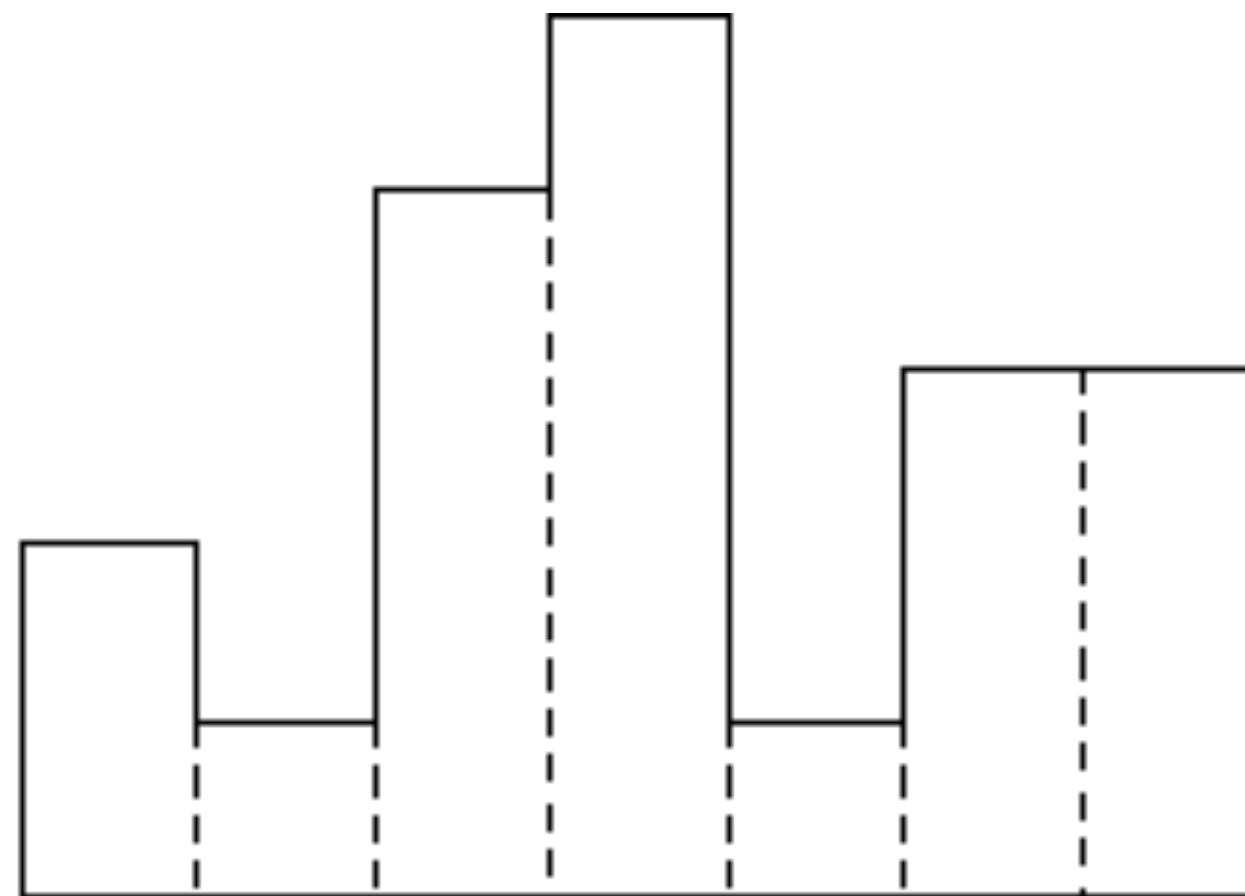
- 소스: <http://codeplus.codes/9cd7821835cf4d0b843523d3527eae72>

# 히스토그램에서 가장 큰 직사각형

15

<https://www.acmicpc.net/problem/6549>

- 히스토그램이 주어졌을 때, 가장 큰 직사각형을 찾는 문제

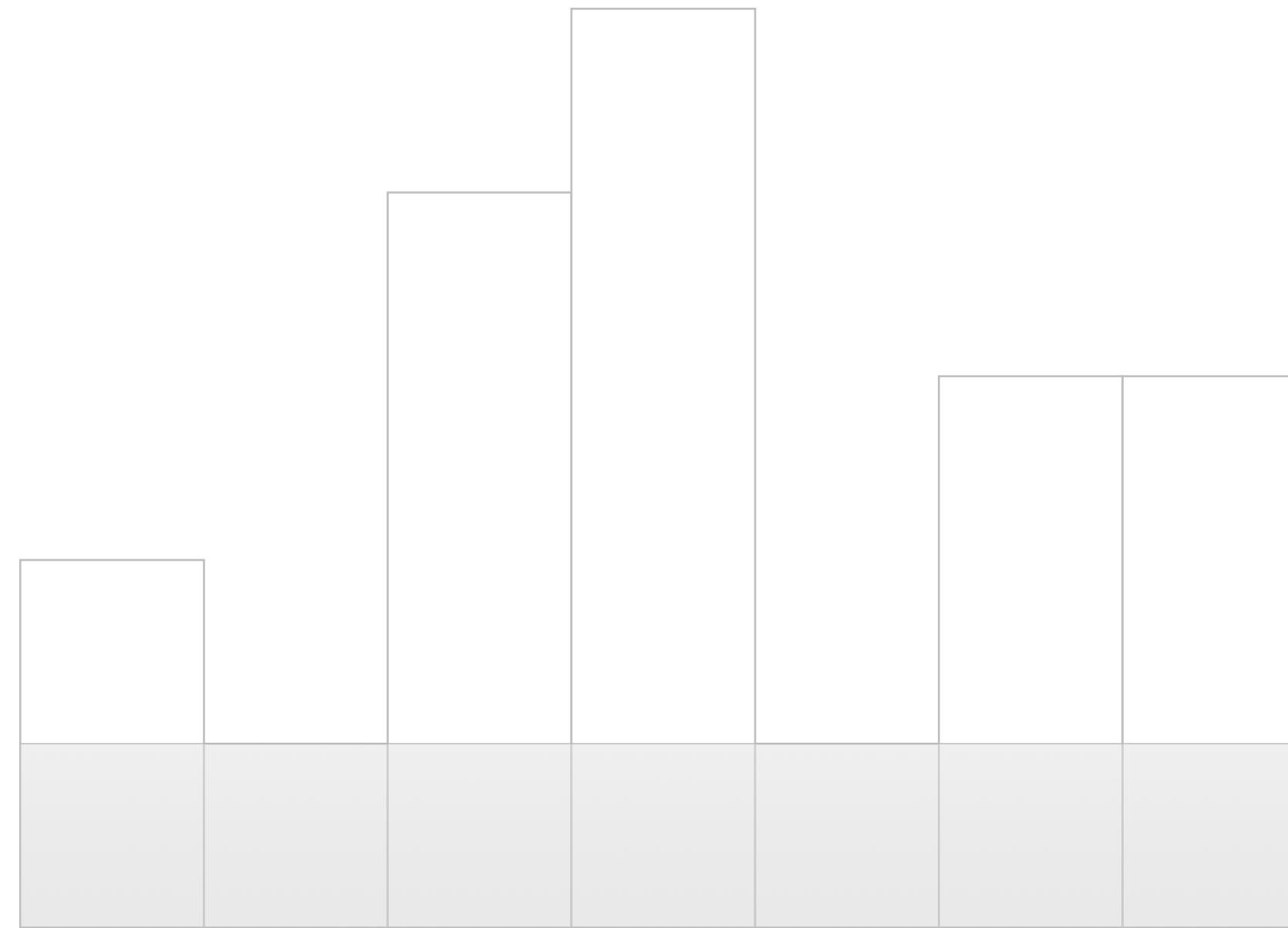


# 히스토그램에서 가장 큰 직사각형

16

<https://www.acmicpc.net/problem/6549>

- 가장 왼쪽 끝과 오른쪽 끝을 변으로 하는 가장 큰 직사각형의 높이는?
- 높이 : 히스토그램에서 가장 높이가 낮은 막대의 높이





# 히스토그램에서 가장 큰 직사각형

<https://www.acmicpc.net/problem/6549>

- 모든 막대  $x$ 에 대해서,  $x$ 를 높이로 하면서, 만들 수 있는 가장 큰 직사각형을 찾아야 함
- $x$ 를 높이로 하면서 만들 수 있는 가장 큰 직사각형은
- $x$ 의 왼쪽에 있는 막대 중에  $x$ 보다 높이가 작은 첫 번째 막대  $left$ 와
- $x$ 의 오른쪽에 있는 막대 중에서  $x$ 보다 높이가 작은 첫 번째 막대  $right$ 를
- 찾아야 한다

# 히스토그램에서 가장 큰 직사각형

<https://www.acmicpc.net/problem/6549>

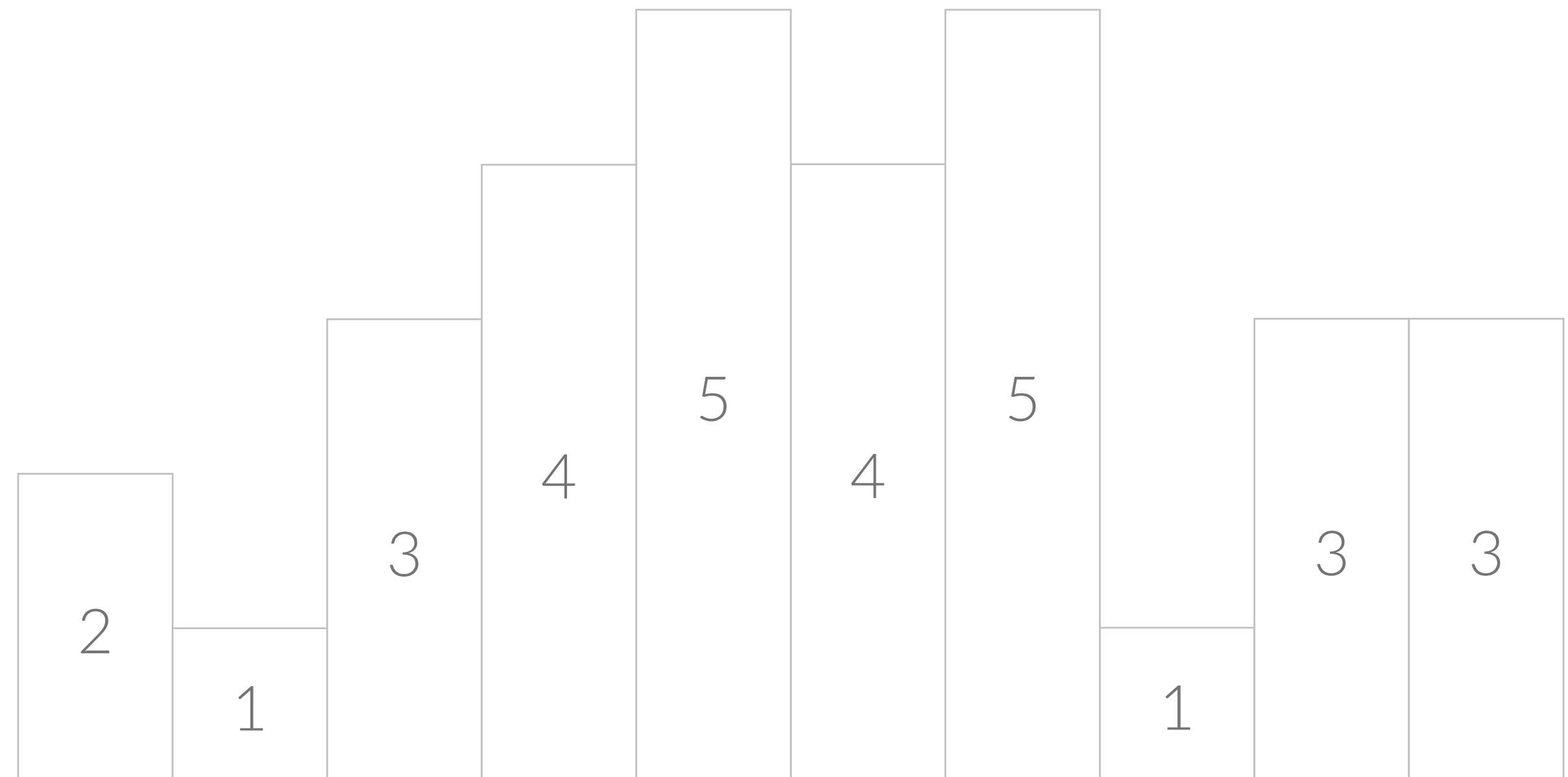
- 스택에 막대를 넣기 전에
- 스택의 가장 위에 있는 막대  $top$ 과 현재 넣으려고 하는 막대  $x$ 를 비교해야 한다
- $top$ 의 높이가  $x$ 의 높이보다 크면
- $top$ 을 높이로 하는 직사각형은  $x$ 를 지나갈 수 없다
- $top$ 을 높이로 하는 직사각형의  $right$ 는  $x-1$ 이다
- $top$ 을 높이로 하는 직사각형의  $left$ 는  $top$  다음에 스택에 들어있는 막대
- $left$ 와  $right$ 를 구했기 때문에,  $top$ 을 높이로 하는 넓이를 구할 수 있다

# 히스토그램에서 가장 큰 직사각형

19

<https://www.acmicpc.net/problem/6549>

- 0번 막대, 높이 2
- 스택이 비어있기 때문에, 막대 번호 0을 스택에 넣는다
- 스택: 0

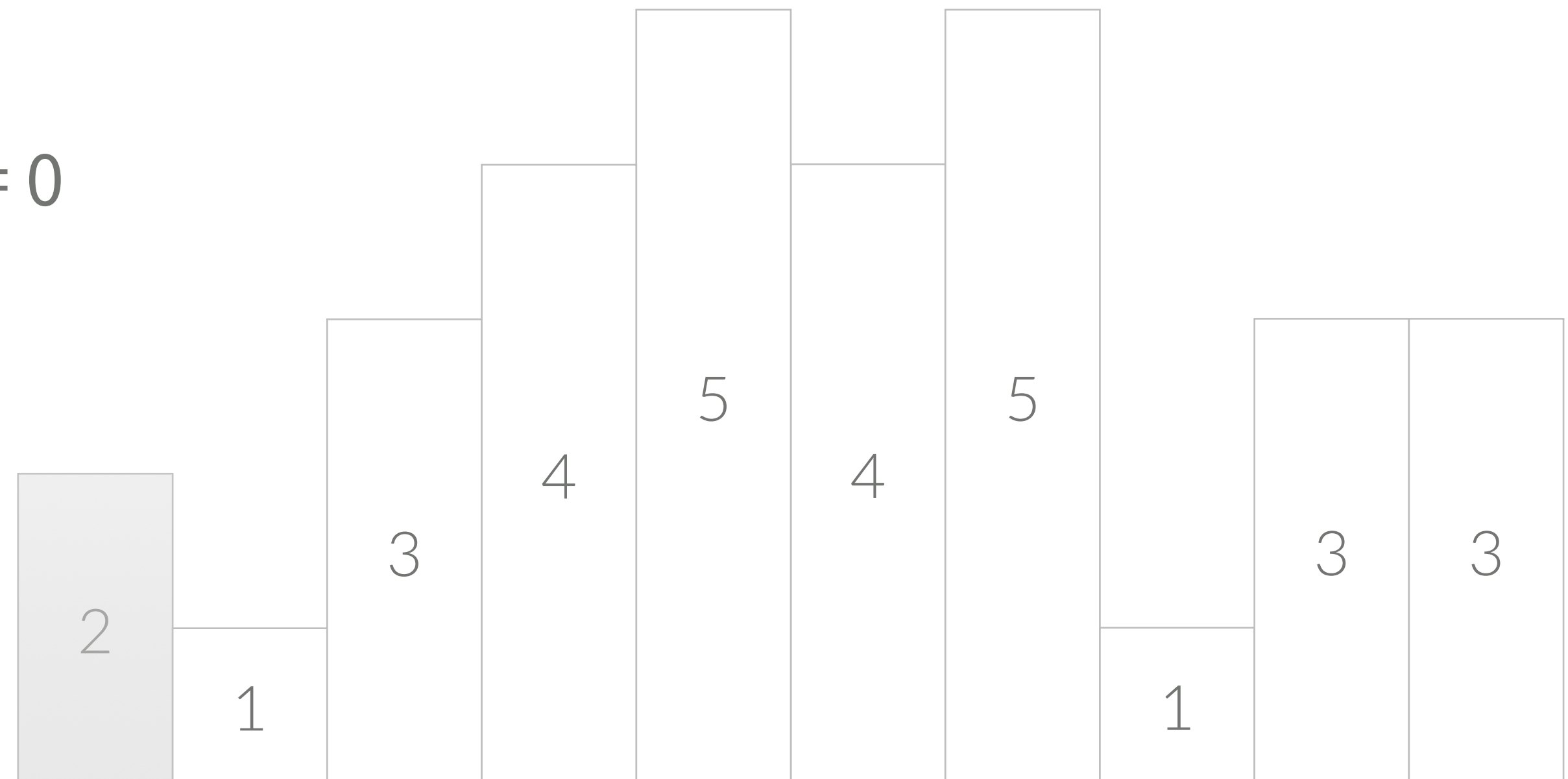


# 히스토그램에서 가장 큰 직사각형

20

<https://www.acmicpc.net/problem/6549>

- 1번 막대, 높이 1
- 스택의 가장 위에 있는 막대보다 높이가 작다
- 0번 막대의 오른쪽 끝 =  $1 - 1 = 0$
- pop을 하면 스택이 비어있기 때문에 왼쪽 끝 = 0
- 넓이: 2
- 스택: 1

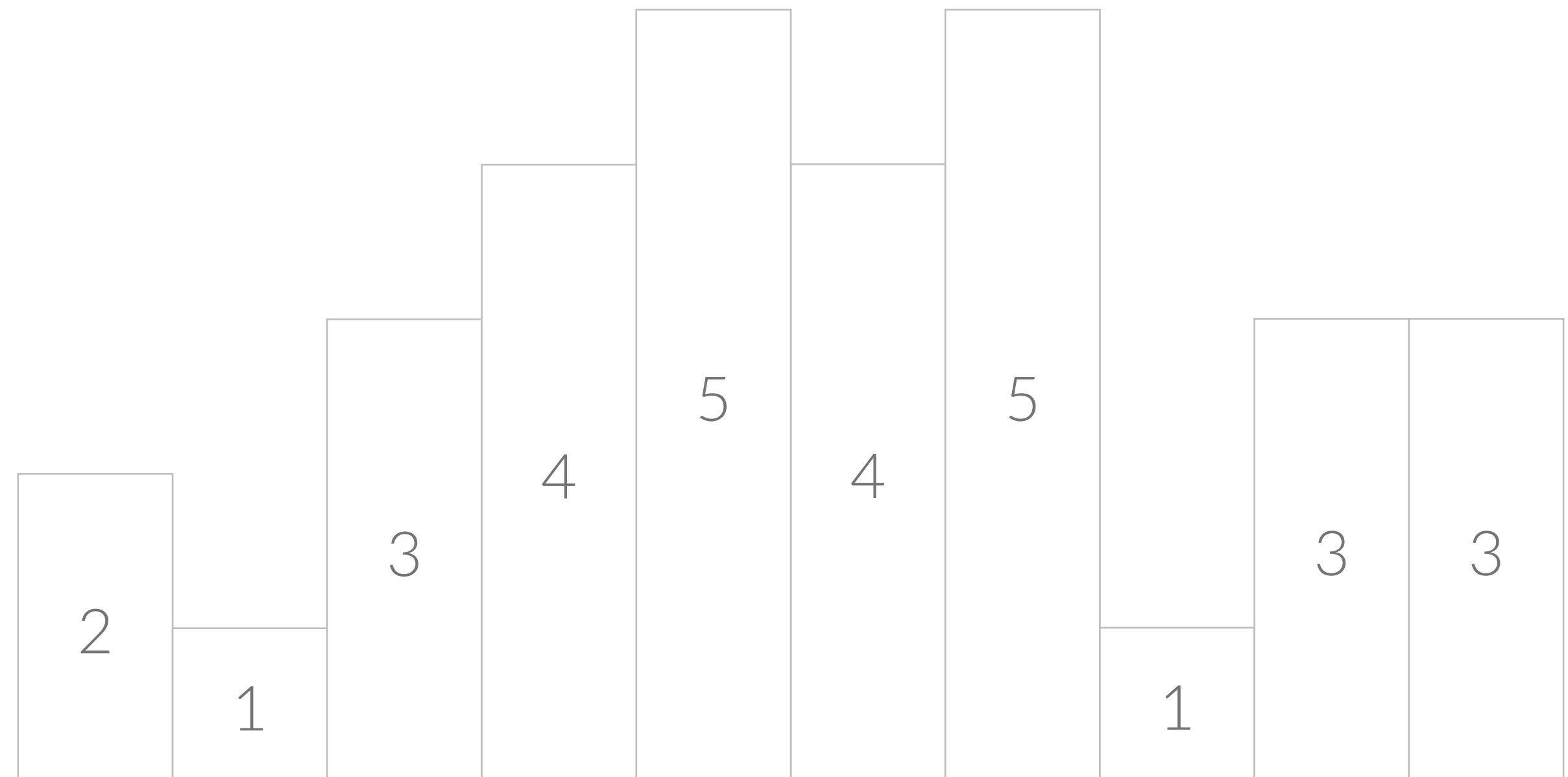


# 히스토그램에서 가장 큰 직사각형

21

<https://www.acmicpc.net/problem/6549>

- 2번 막대, 높이 3
- 스택의 가장 위에 있는 막대 1보다 높이가 크거나 같기 때문에 push
- 스택: 1 2

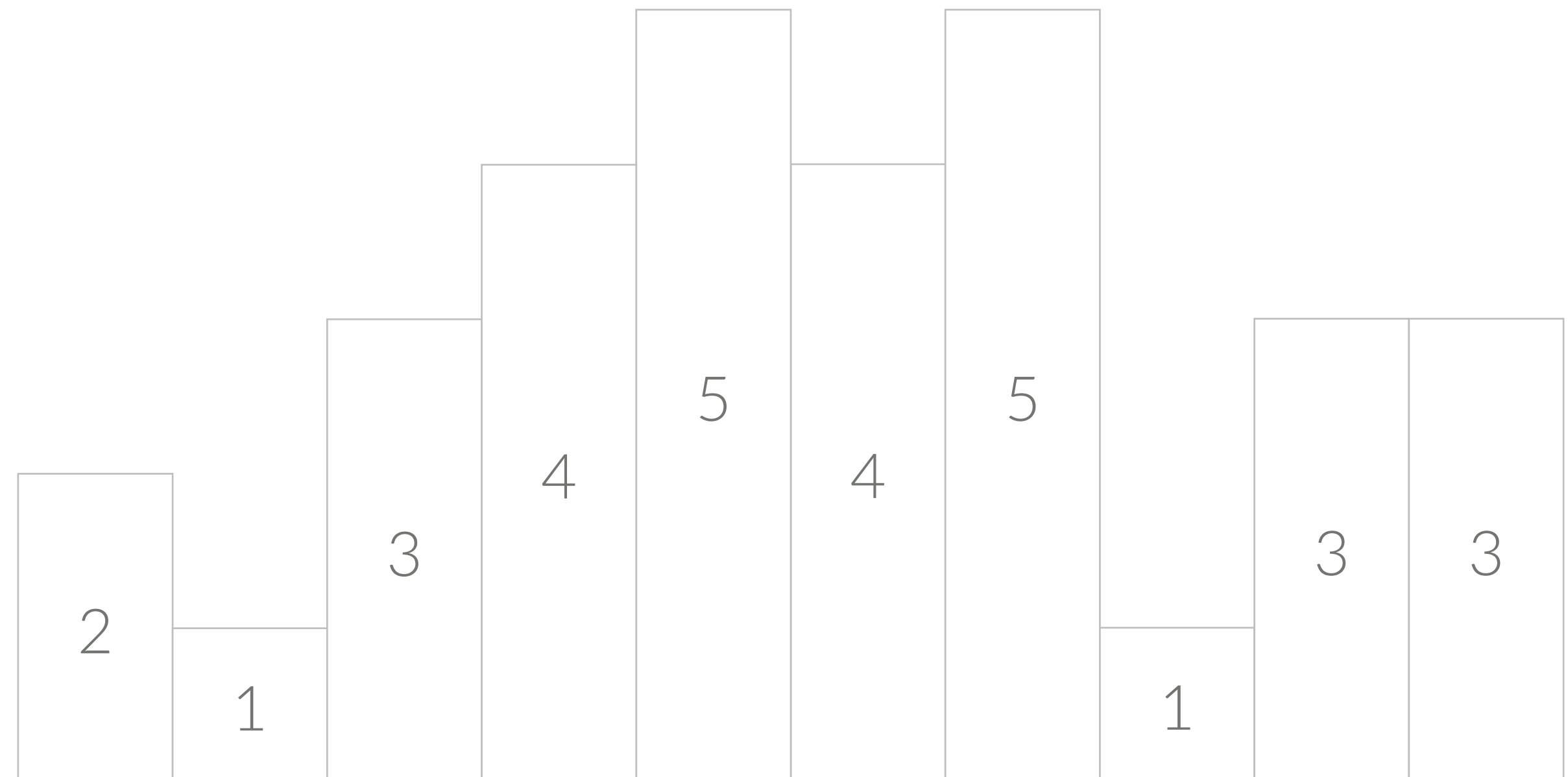


# 히스토그램에서 가장 큰 직사각형

22

<https://www.acmicpc.net/problem/6549>

- 3번 막대, 높이 4
- 스택의 가장 위에 있는 막대 2보다 높이가 크거나 같기 때문에 push
- 스택: 1 2 3

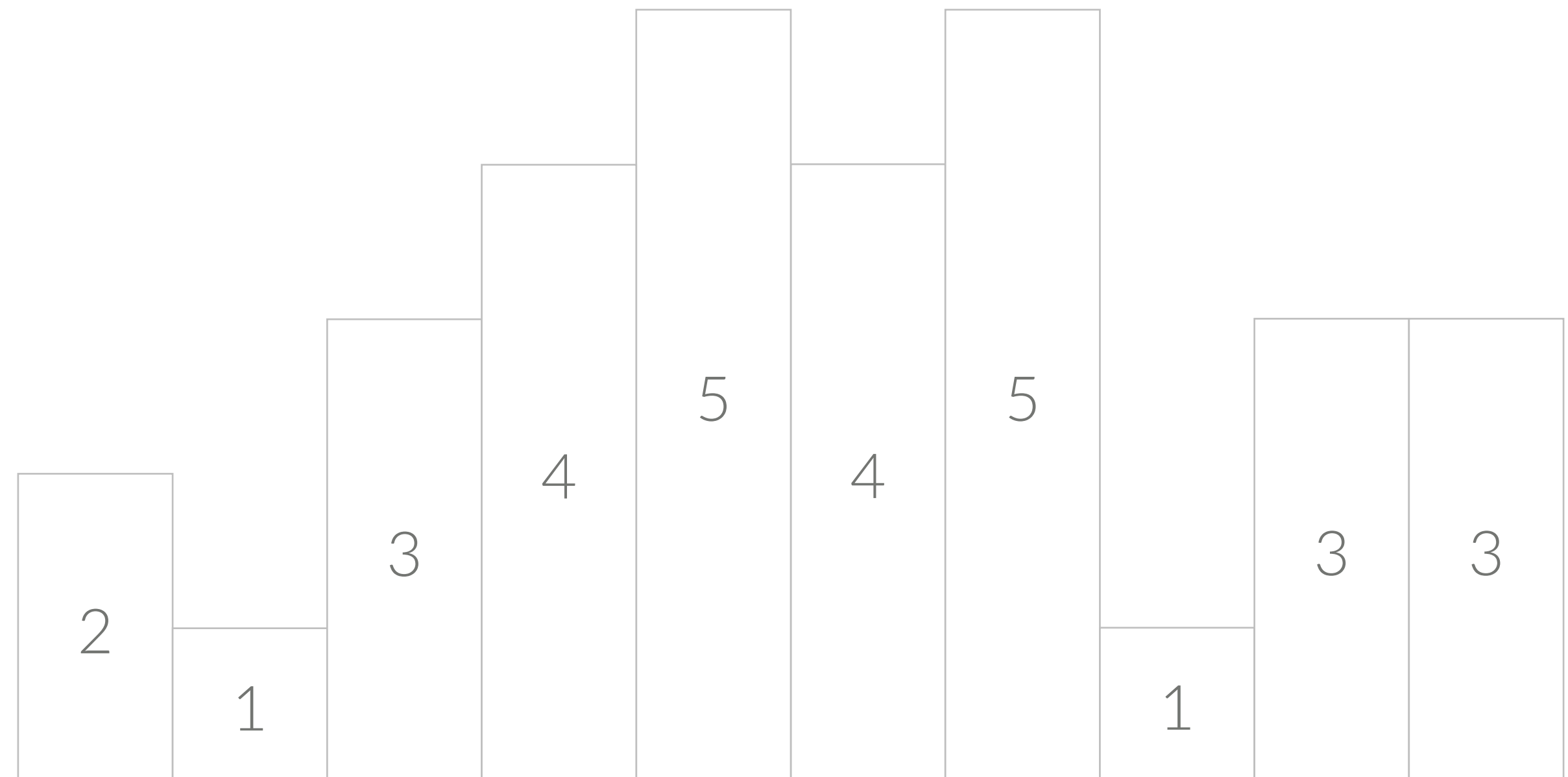


# 히스토그램에서 가장 큰 직사각형

23

<https://www.acmicpc.net/problem/6549>

- 4번 막대, 높이 5
- 스택의 가장 위에 있는 막대 3보다 높이가 크거나 같기 때문에 push
- 스택: 1 2 3 4

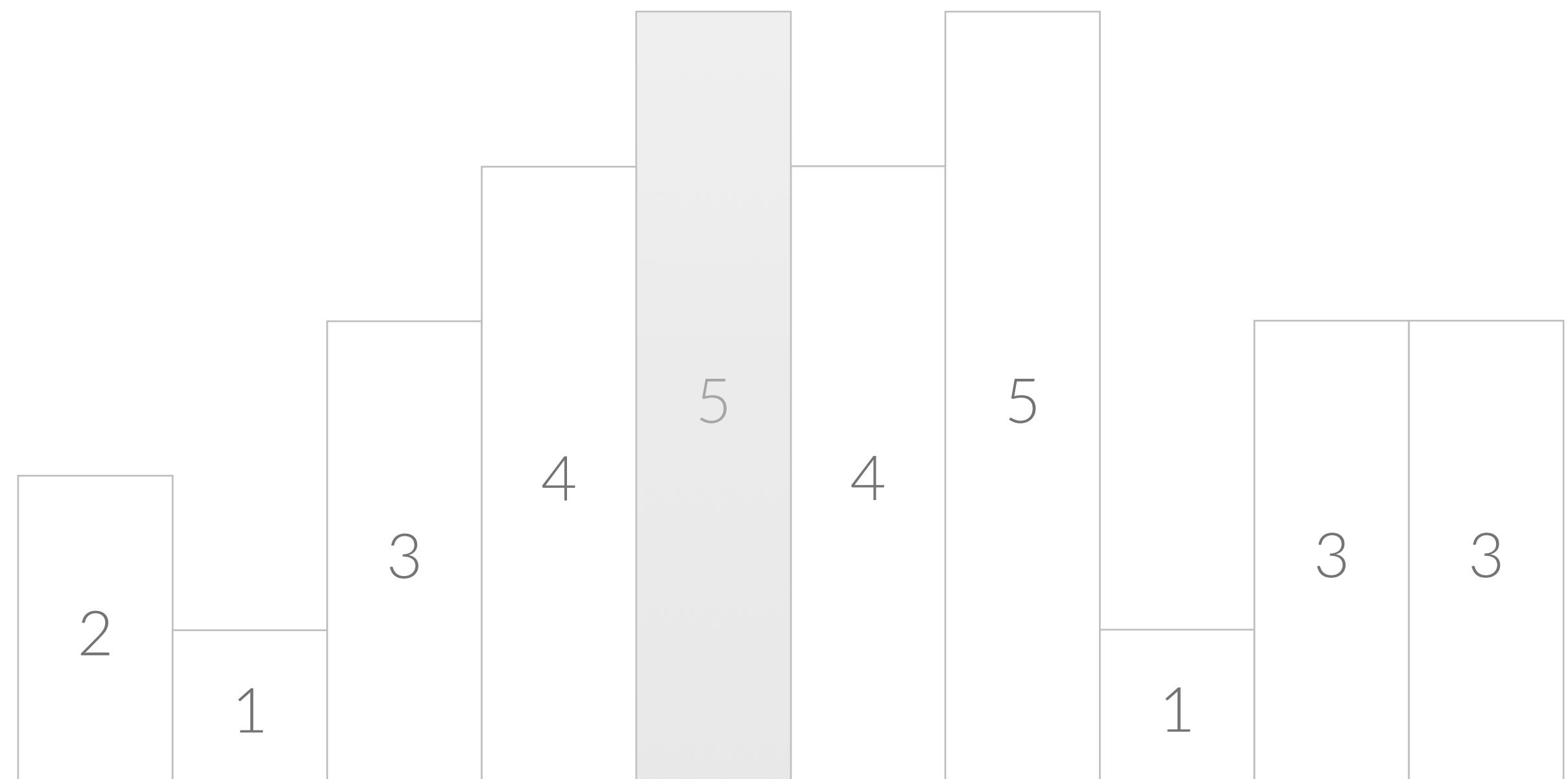


# 히스토그램에서 가장 큰 직사각형

24

<https://www.acmicpc.net/problem/6549>

- 5번 막대, 높이 4
- 스택의 가장 위에 있는 막대 4번 (높이 5)이 현재 높이보다 크다
- $\text{right} = 5 - 1 = 4$ ,  $\text{left} = 3 + 1 = 4$
- 4번 막대로 만들 수 있는 직사각형 넓이: 5
- 스택: 1 2 3



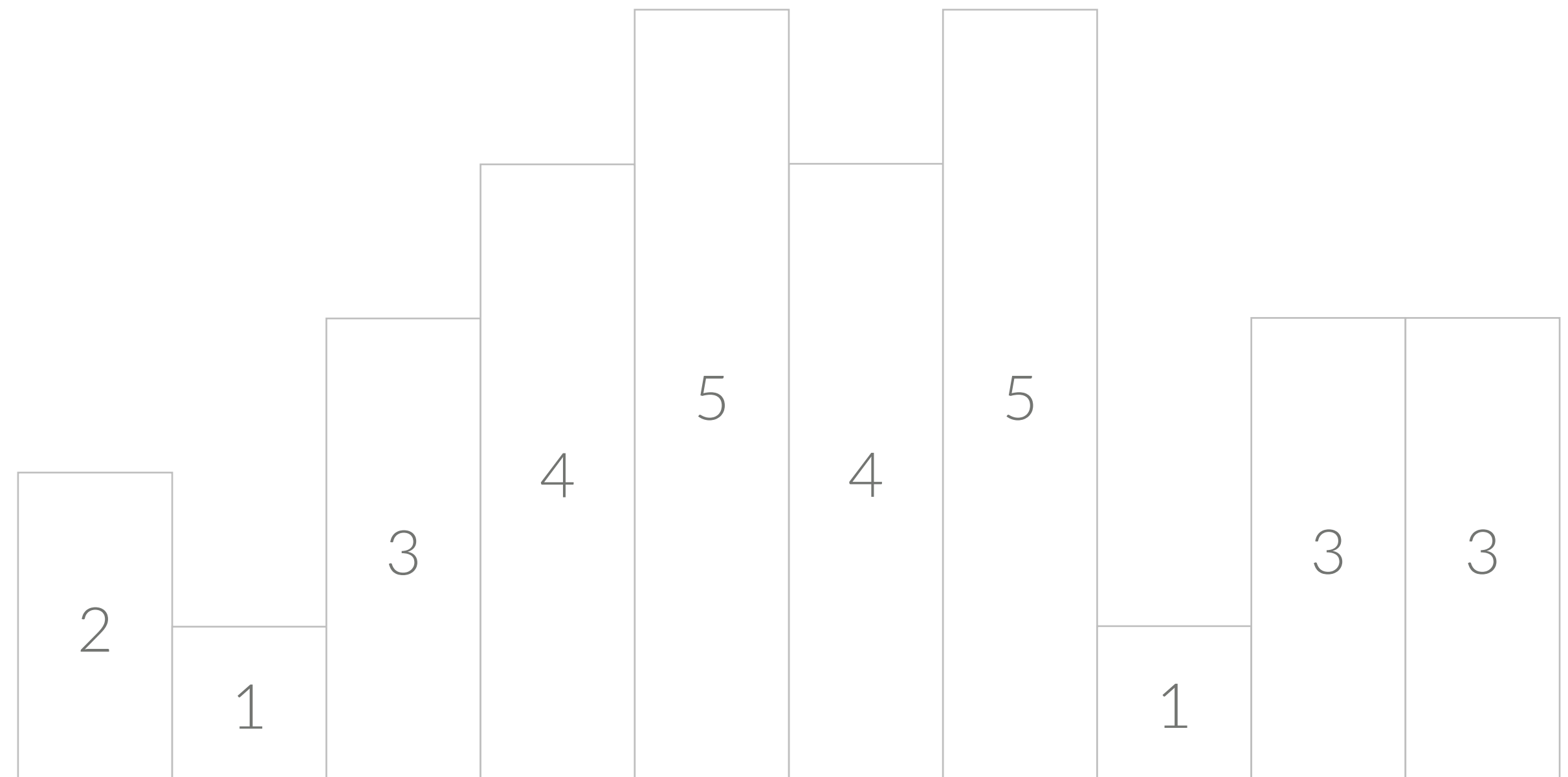


# 히스토그램에서 가장 큰 직사각형

25

<https://www.acmicpc.net/problem/6549>

- 5번 막대, 높이 4
- 스택의 가장 위에 있는 막대 3보다 높이가 크거나 같기 때문에 push
- 스택: 1 2 3 5

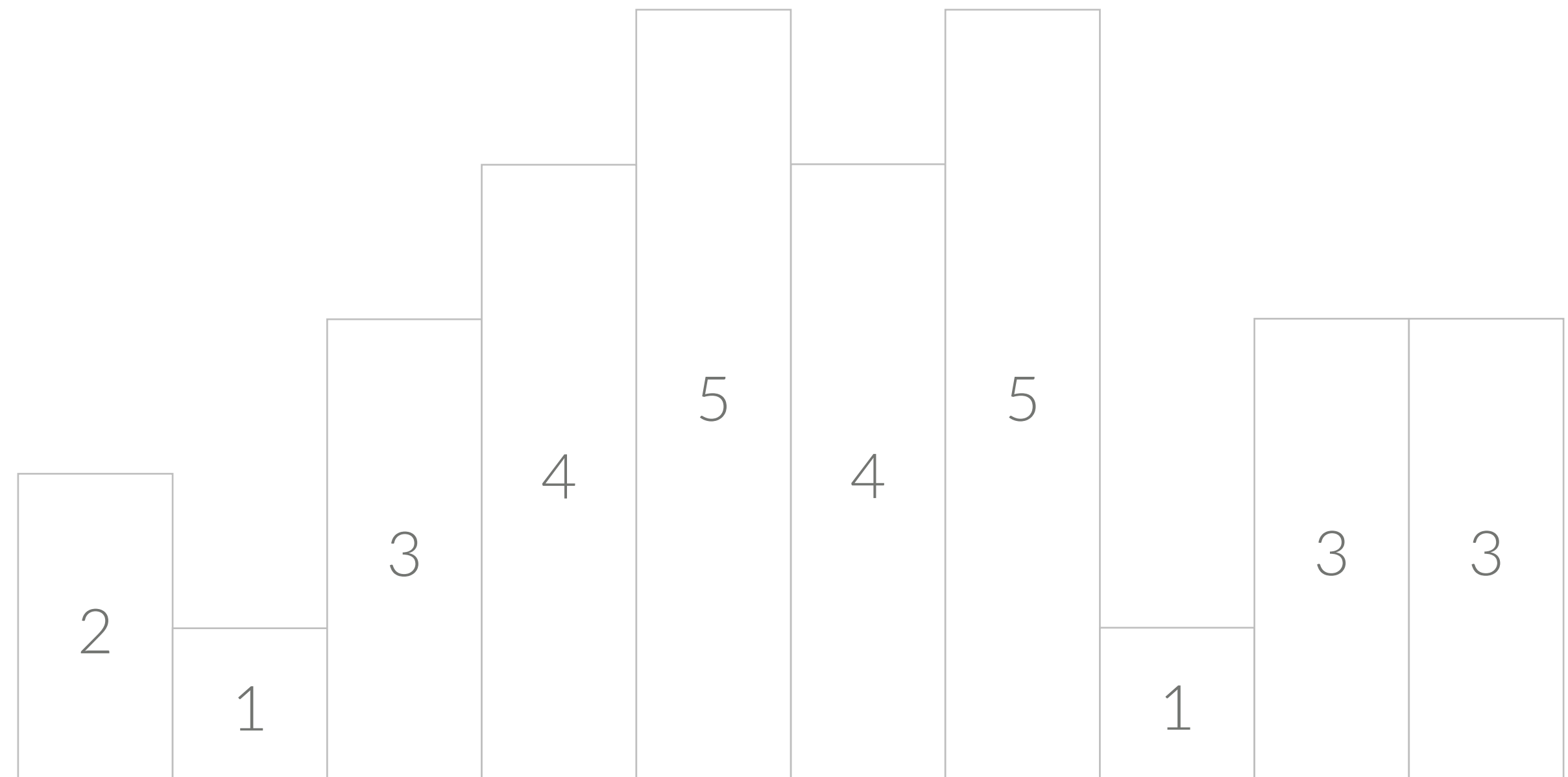


# 히스토그램에서 가장 큰 직사각형

26

<https://www.acmicpc.net/problem/6549>

- 6번 막대, 높이 5
- 스택의 가장 위에 있는 막대 5보다 높이가 크거나 같기 때문에 push
- 스택: 1 2 3 5 6

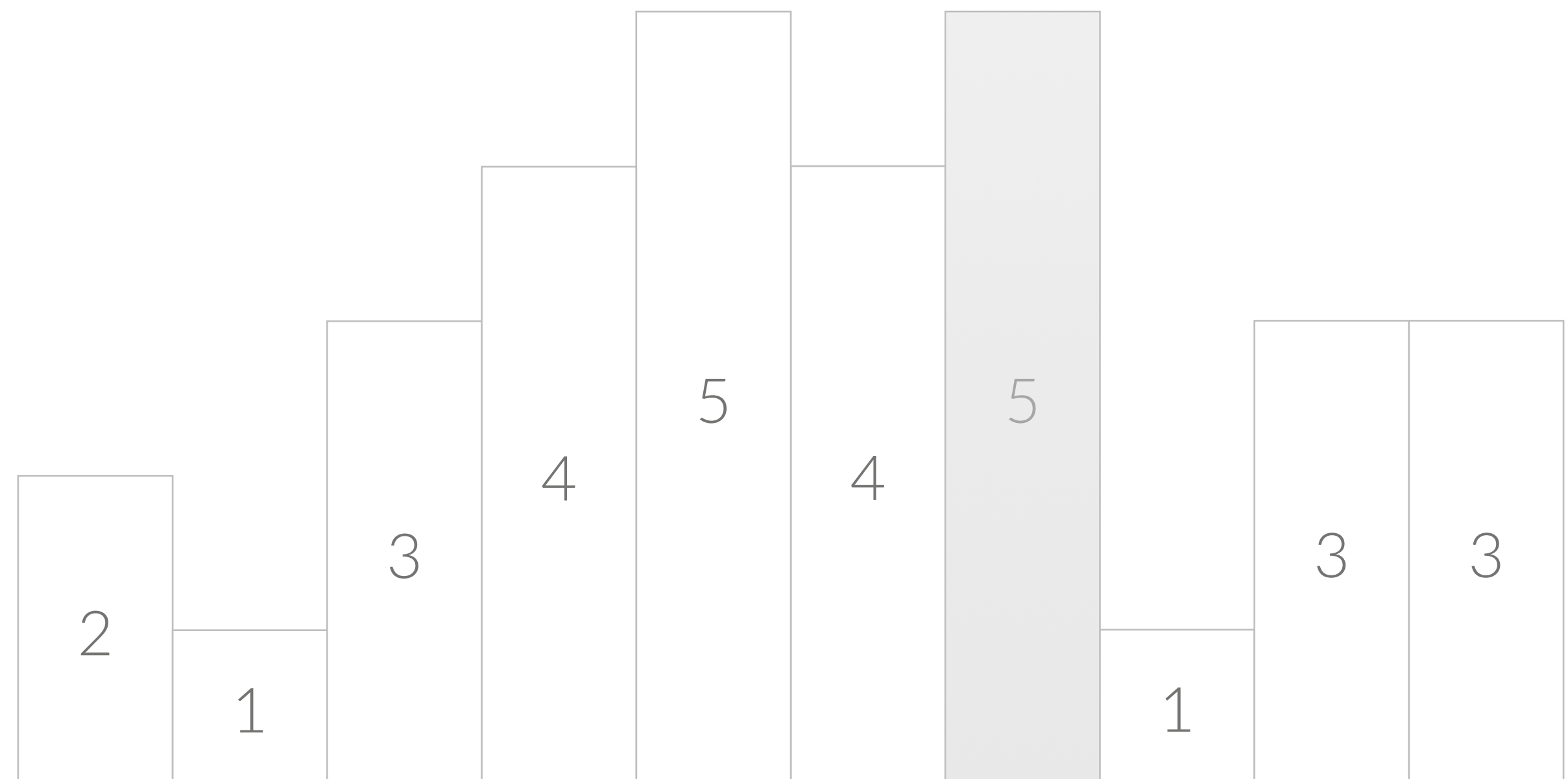


# 히스토그램에서 가장 큰 직사각형

27

<https://www.acmicpc.net/problem/6549>

- 7번 막대, 높이 1
- 스택의 가장 위에 있는 막대 6번 (높이 5)이 현재 높이보다 크다
- $\text{right} = 7 - 1 = 6$ ,  $\text{left} = 5 + 1 = 6$
- 6번 막대로 만들 수 있는 직사각형 넓이: 5
- 스택: 1 2 3 5

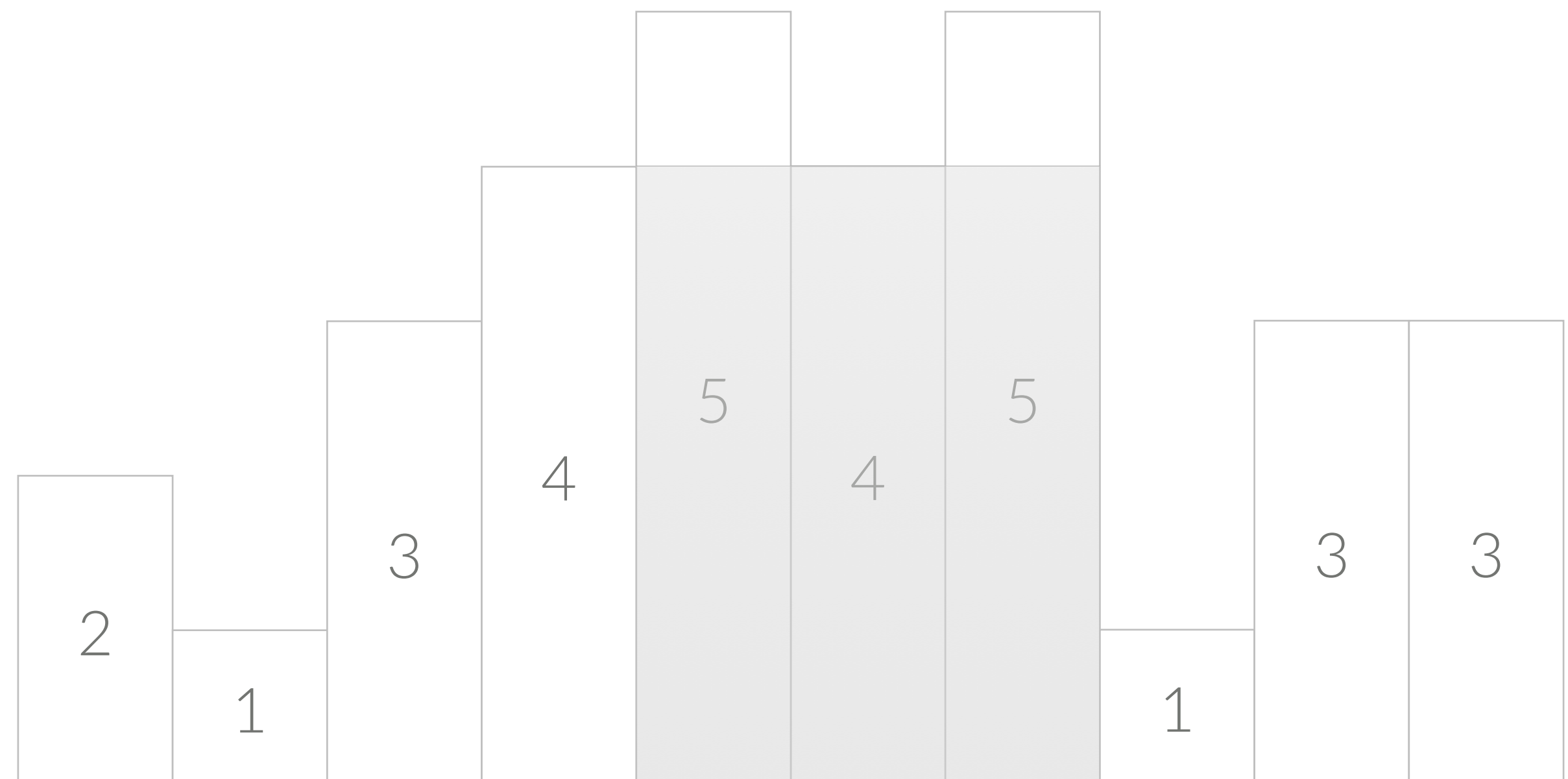


# 히스토그램에서 가장 큰 직사각형

28

<https://www.acmicpc.net/problem/6549>

- 7번 막대, 높이 1
- 스택의 가장 위에 있는 막대 5번 (높이 4)이 현재 높이보다 크다
- $\text{right} = 7 - 1 = 6$ ,  $\text{left} = 3 + 1 = 4$
- 5번 막대로 만들 수 있는 직사각형 넓이: 12
- 스택: 1 2 3

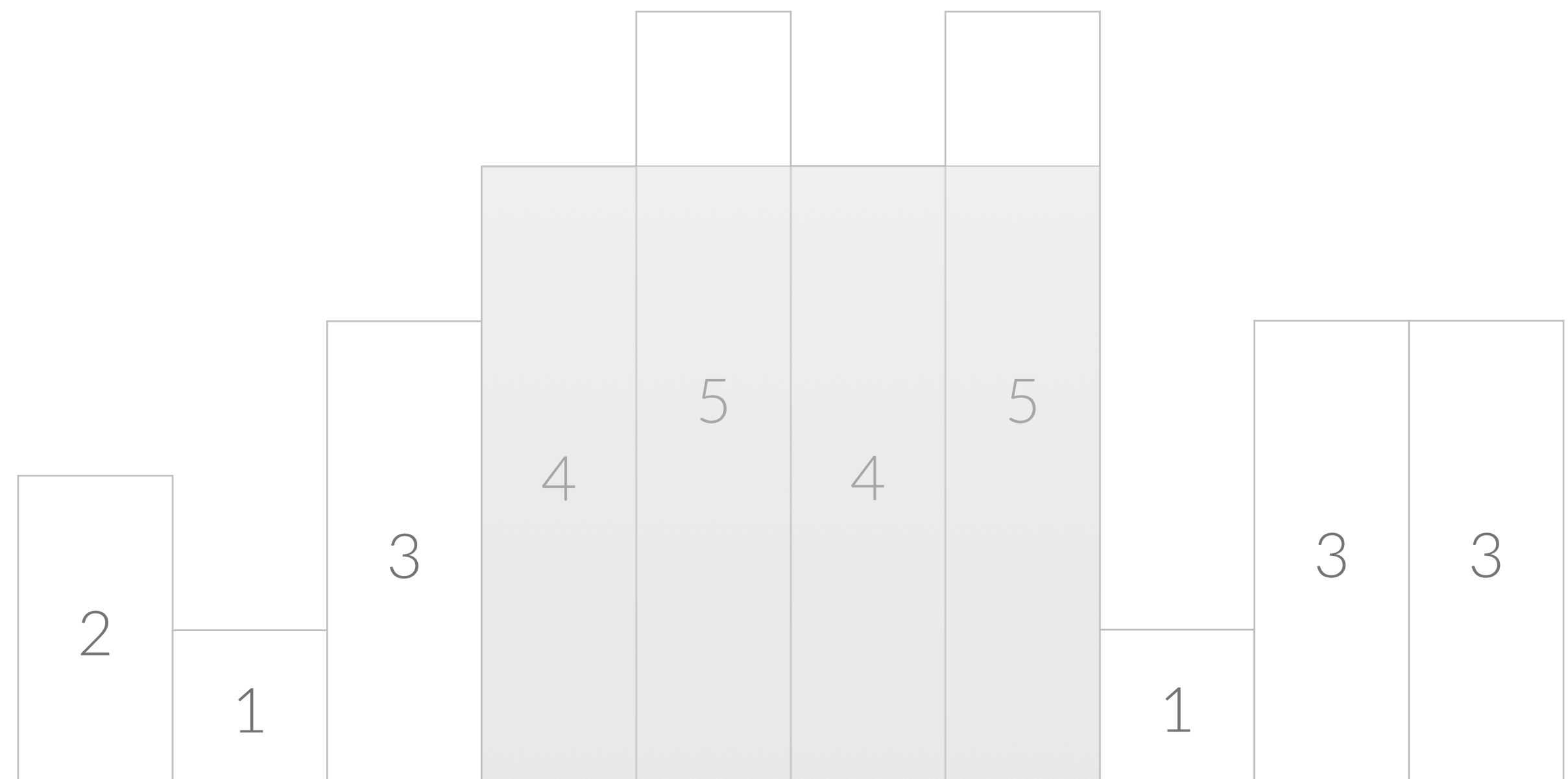


# 히스토그램에서 가장 큰 직사각형

29

<https://www.acmicpc.net/problem/6549>

- 7번 막대, 높이 1
- 스택의 가장 위에 있는 막대 3번 (높이 4)이 현재 높이보다 크다
- $\text{right} = 7 - 1 = 6$ ,  $\text{left} = 2 + 1 = 3$
- 3번 막대로 만들 수 있는 직사각형 넓이: 16
- 스택: 1 2

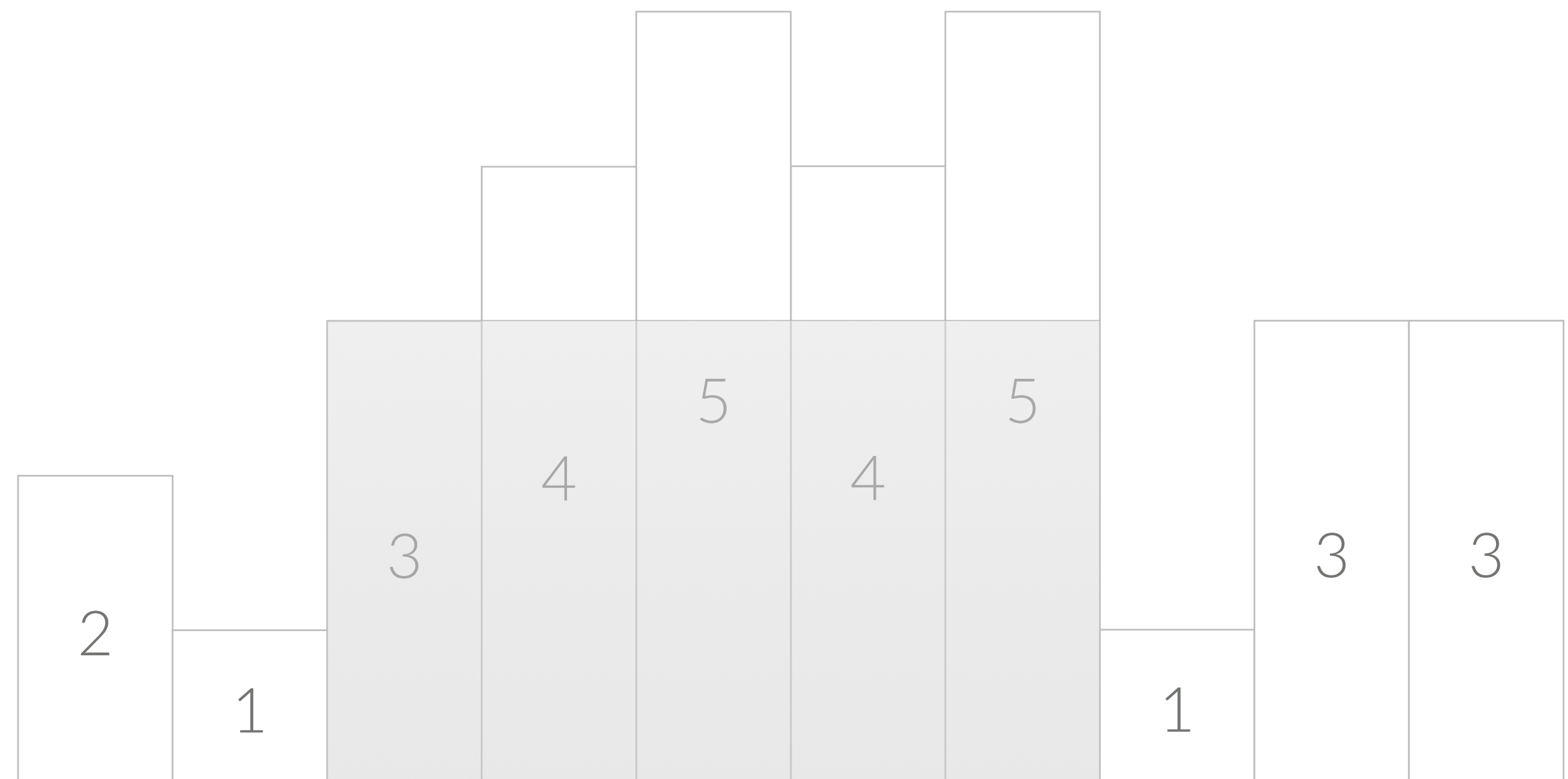


# 히스토그램에서 가장 큰 직사각형

30

<https://www.acmicpc.net/problem/6549>

- 7번 막대, 높이 1
- 스택의 가장 위에 있는 막대 2번 (높이 3)이 현재 높이보다 크다
- $\text{right} = 7 - 1 = 6$ ,  $\text{left} = 1 + 1 = 2$
- 2번 막대로 만들 수 있는 직사각형 넓이: 15
- 스택: 1

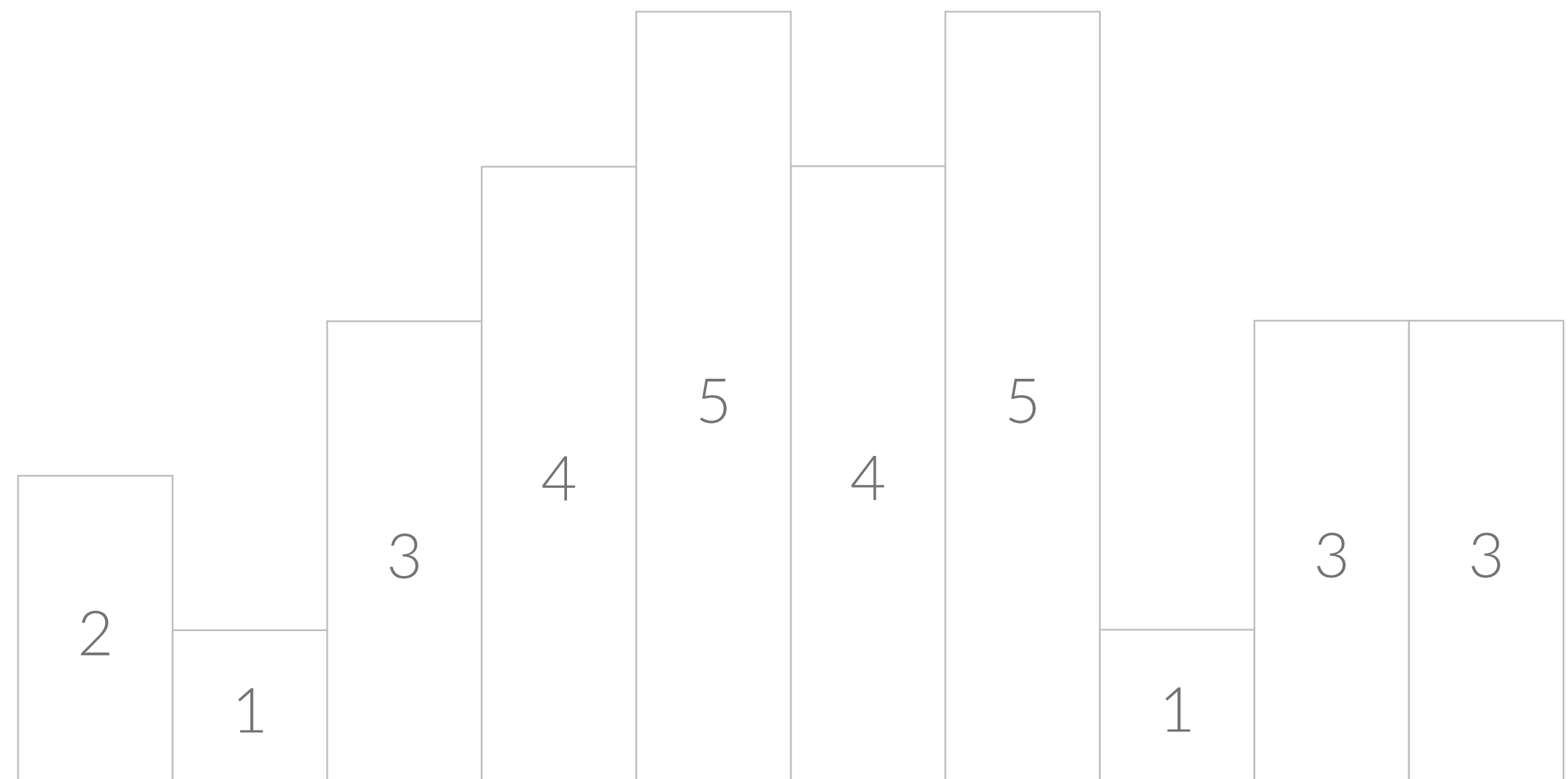


# 히스토그램에서 가장 큰 직사각형

31

<https://www.acmicpc.net/problem/6549>

- 7번 막대, 높이 1
- 스택의 가장 위에 있는 막대 1번 (높이 1)이 현재 높이보다 크거나 같기 때문에
- 스택에 현재 막대 번호 7번을 push
- 스택: 1 7

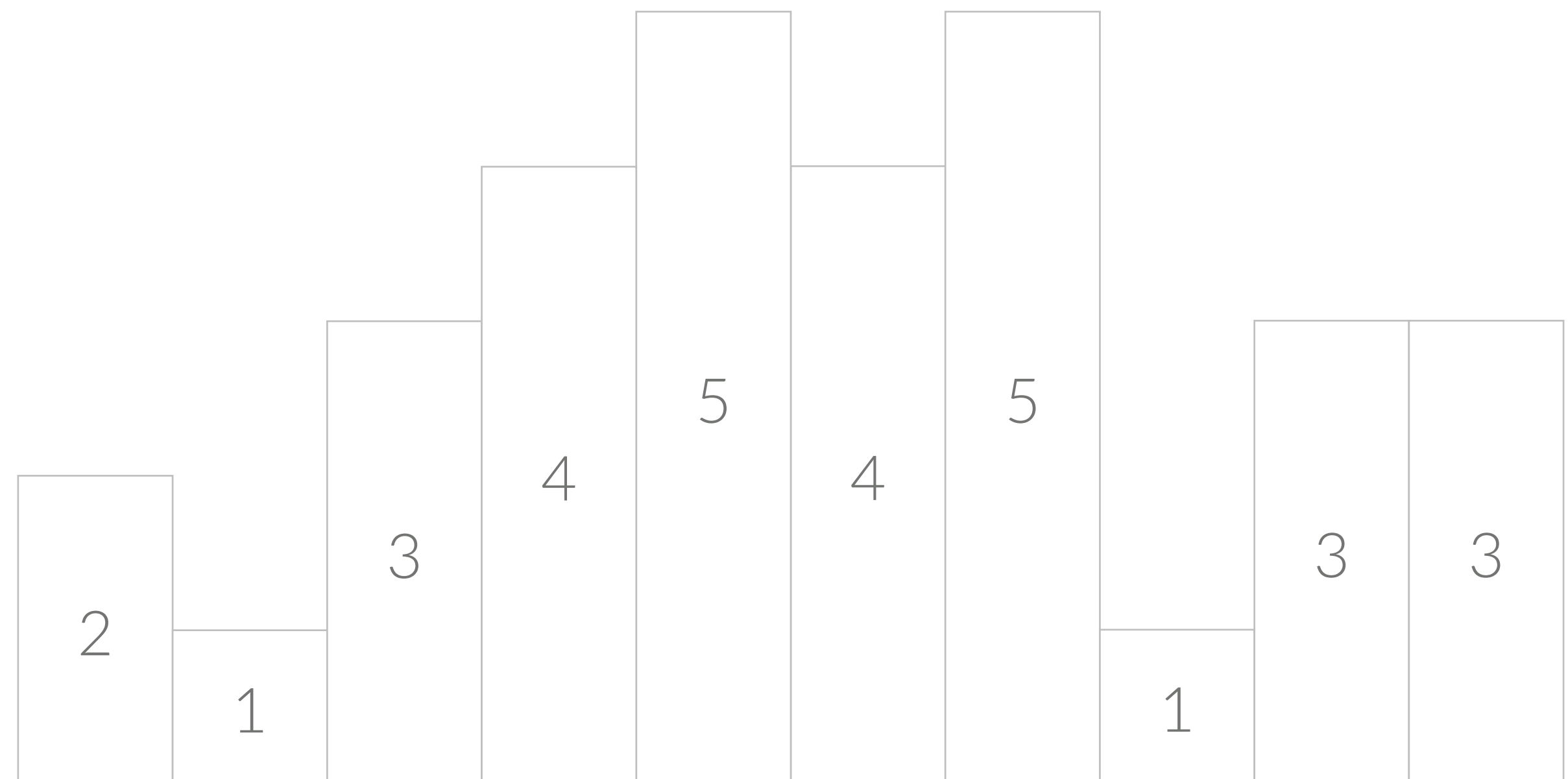


# 히스토그램에서 가장 큰 직사각형

32

<https://www.acmicpc.net/problem/6549>

- 8번 막대, 높이 3
- 스택의 가장 위에 있는 막대 7번 (높이 1)이 현재 높이보다 크거나 같기 때문에
- 스택에 현재 막대 번호 8번을 push
- 스택: 1 7 8



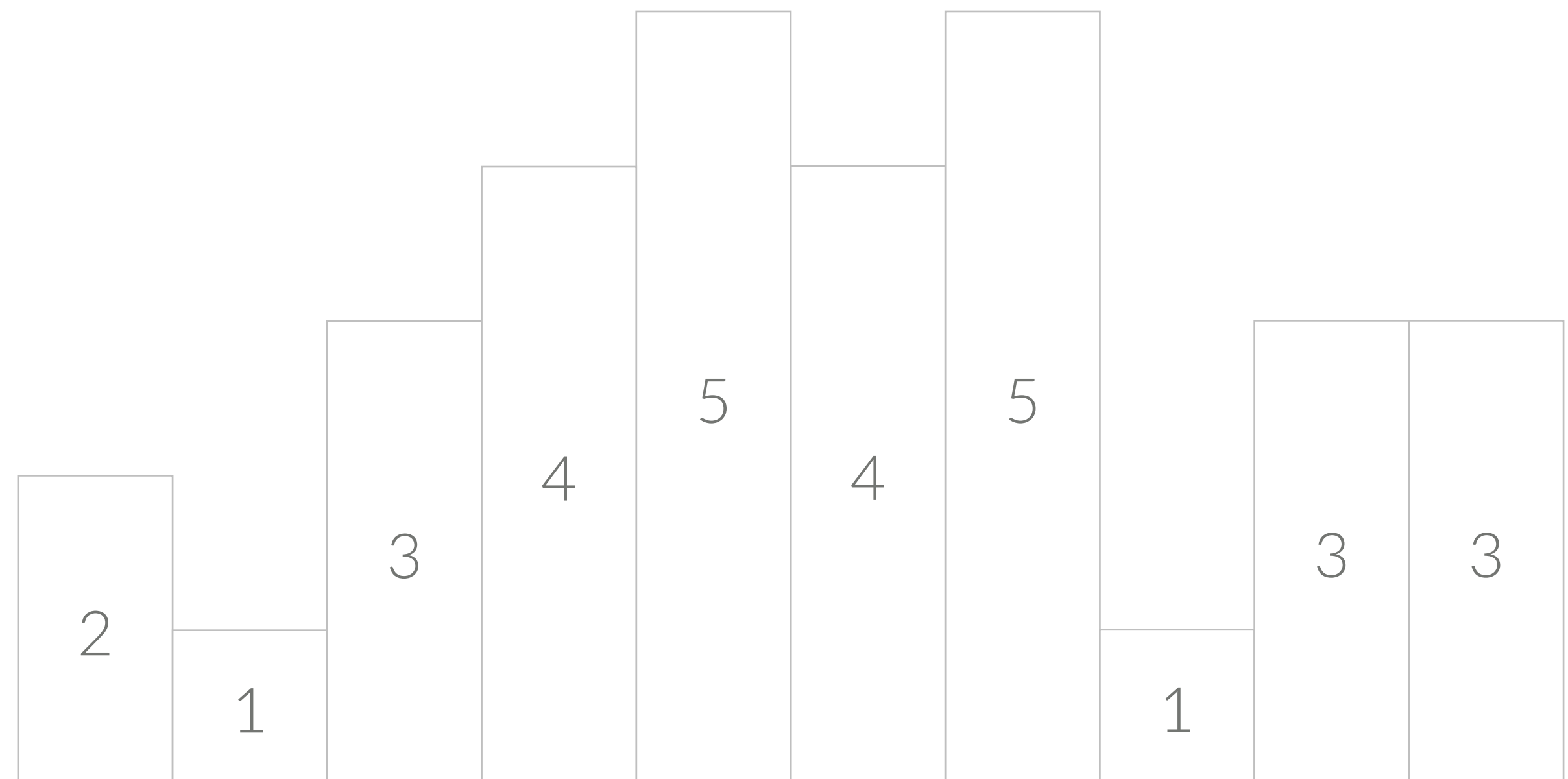


# 히스토그램에서 가장 큰 직사각형

33

<https://www.acmicpc.net/problem/6549>

- 9번 막대, 높이 3
- 스택의 가장 위에 있는 막대 8번 (높이 3)이 현재 높이보다 크거나 같기 때문에
- 스택에 현재 막대 번호 9번을 push
- 스택: 1 7 8 9

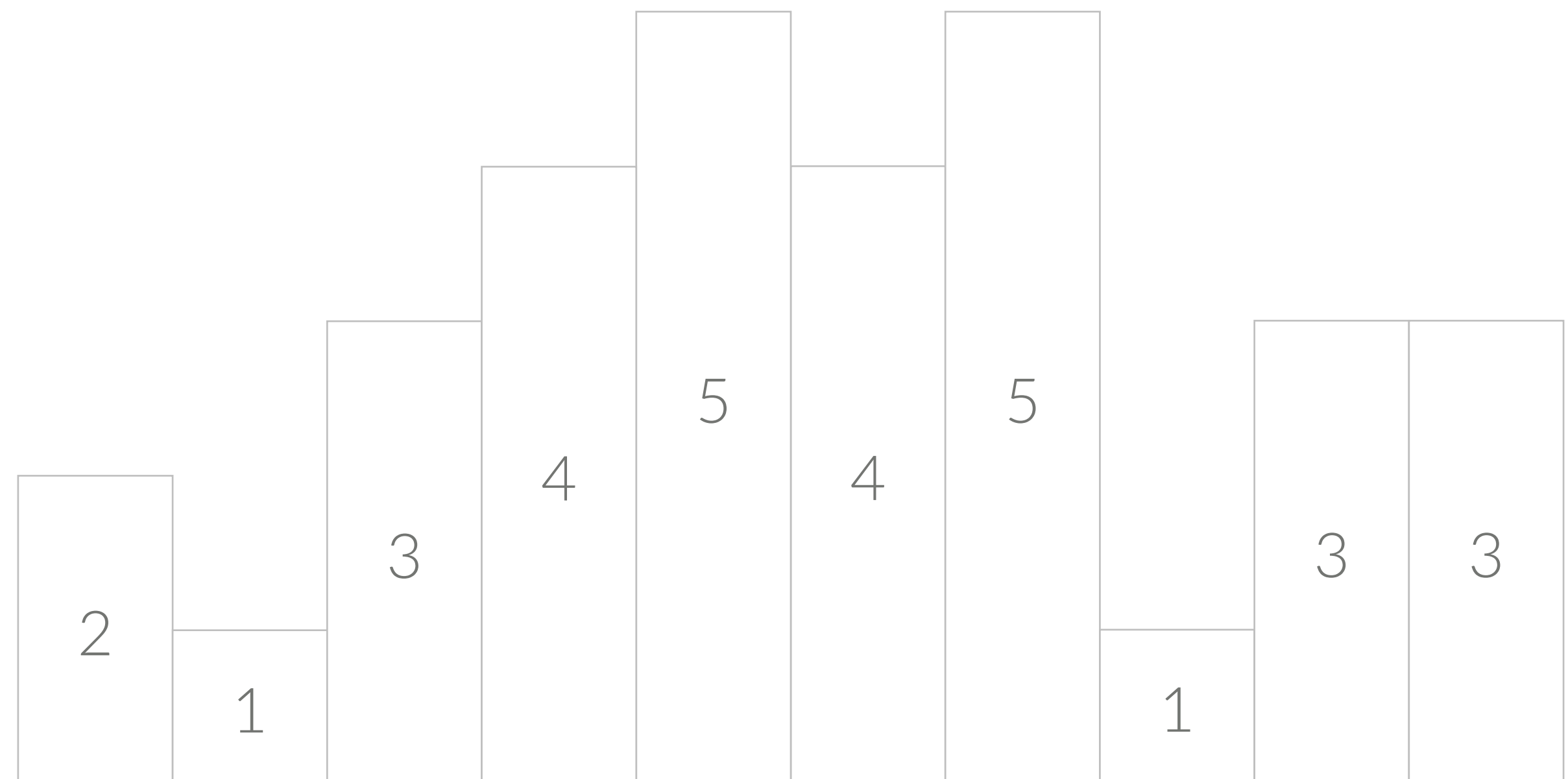


# 히스토그램에서 가장 큰 직사각형

34

<https://www.acmicpc.net/problem/6549>

- 모든 막대가 스택에 들어갔고
- 지금부터는  $\text{right} = n-1$  인 경우를 처리할 차례

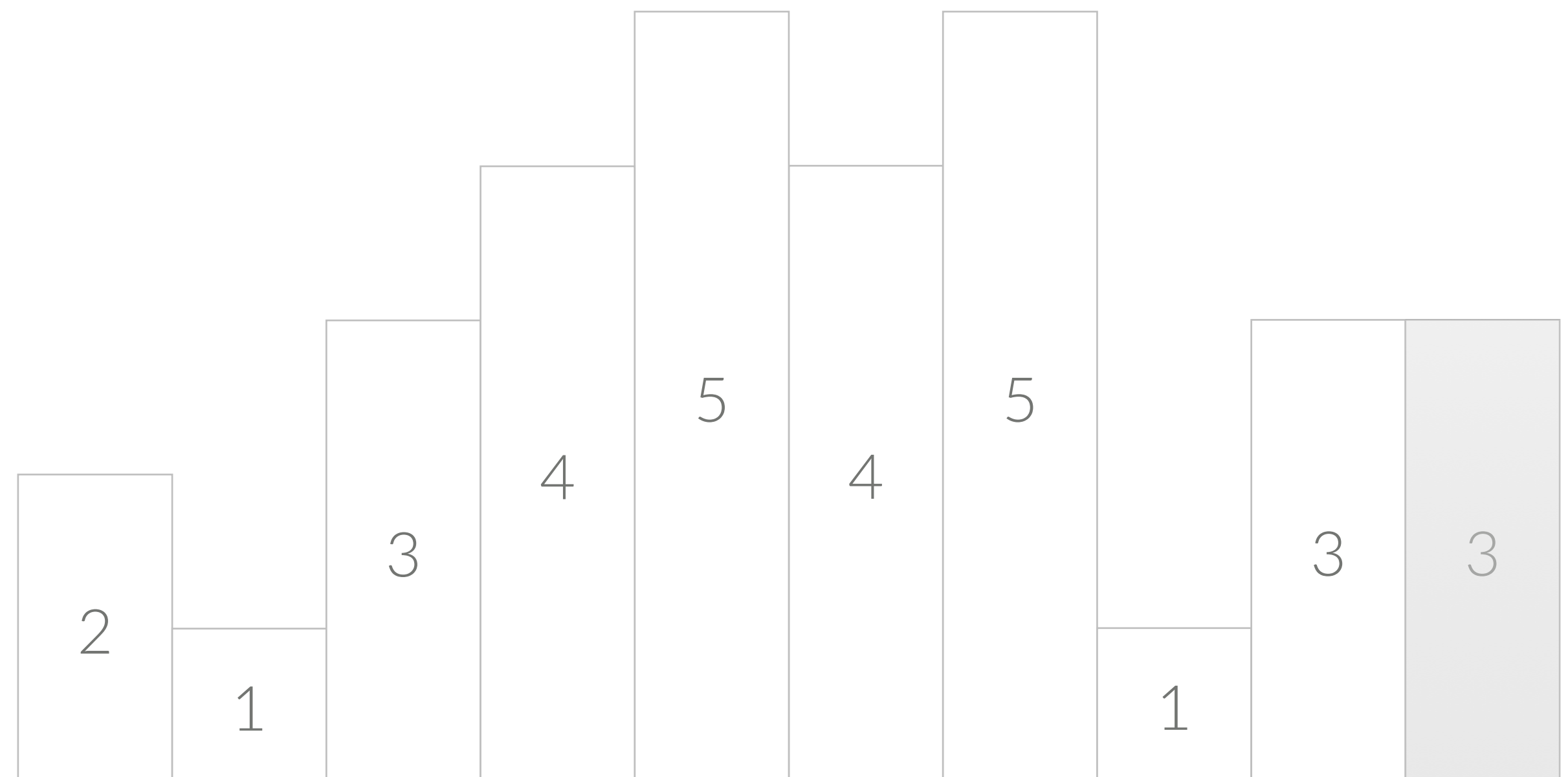


# 히스토그램에서 가장 큰 직사각형

35

<https://www.acmicpc.net/problem/6549>

- 스택: 1 7 8 9
- 9로 만들 수 있는 가장 큰 직사각형
- $\text{Right} = 9$
- $\text{Left} = 8 + 1 = 9$

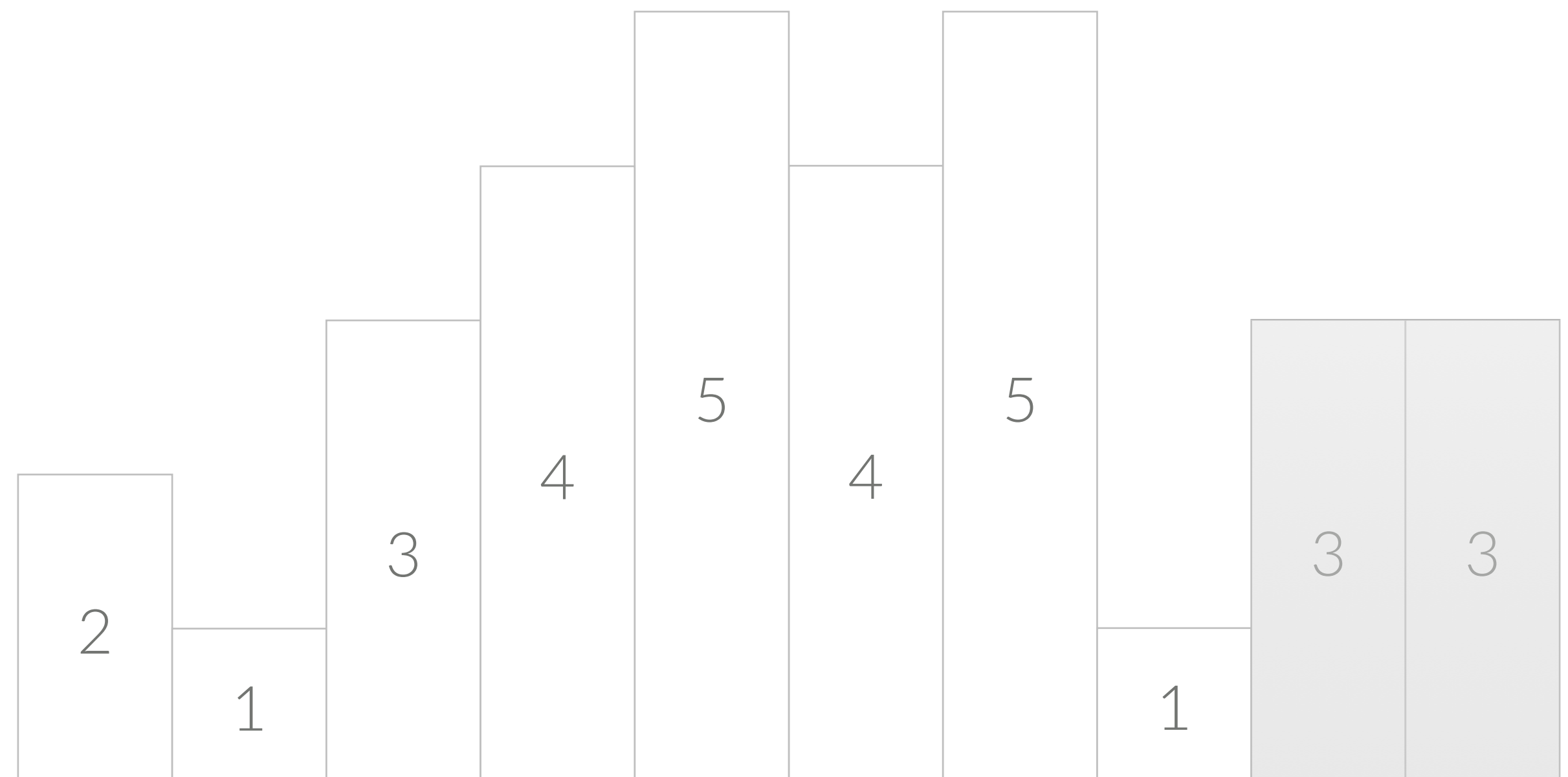


# 히스토그램에서 가장 큰 직사각형

36

<https://www.acmicpc.net/problem/6549>

- 스택: 1 7 8
- 8로 만들 수 있는 가장 큰 직사각형
- $\text{Right} = 9$
- $\text{Left} = 7 + 1 = 8$

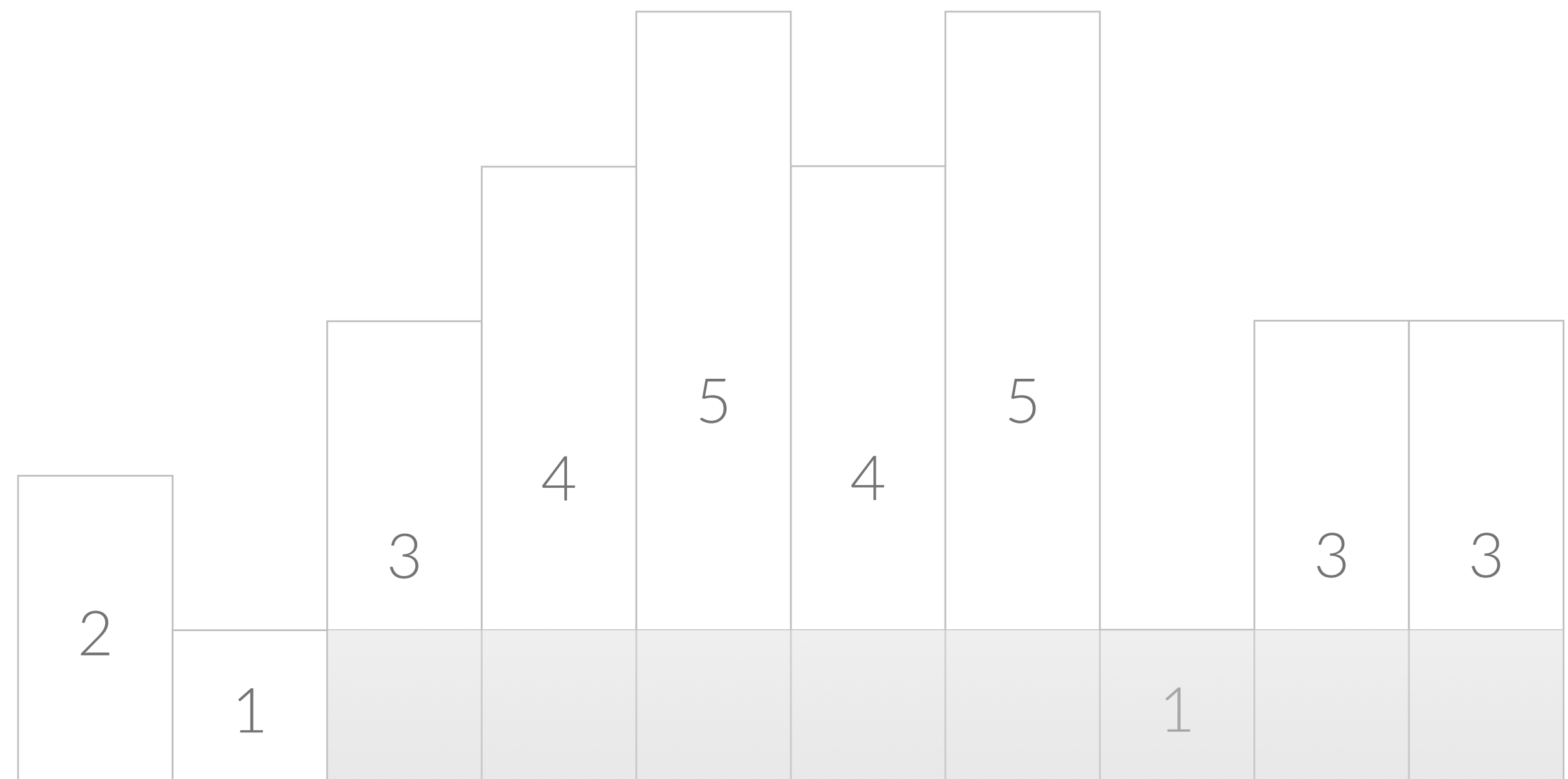


# 히스토그램에서 가장 큰 직사각형

37

<https://www.acmicpc.net/problem/6549>

- 스택: 1 7
- 7로 만들 수 있는 가장 큰 직사각형
- Right = 9
- Left = 1 + 1 = 2

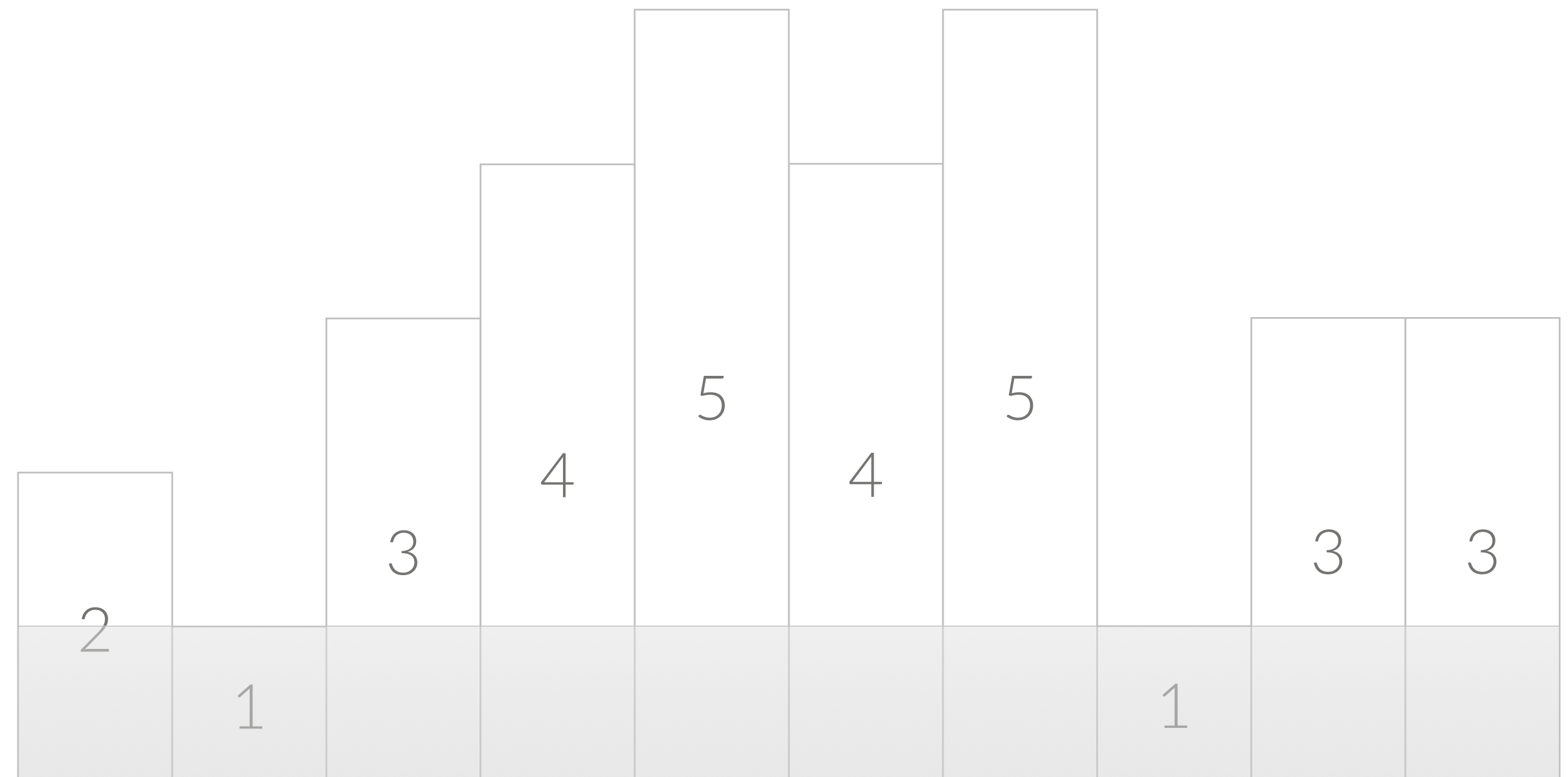


# 히스토그램에서 가장 큰 직사각형

38

<https://www.acmicpc.net/problem/6549>

- 스택: 1
- 1로 만들 수 있는 가장 큰 직사각형
- Right = 9
- Left = 0



# 히스토그램에서 가장 큰 직사각형

<https://www.acmicpc.net/problem/6549>

```
stack<int> s;
int ans = 0;
for (int i=0; i<n; i++) {
    int left = i;
    while(!s.empty() && a[s.top()] > a[i]) {
        int height = a[s.top()];
        s.pop();
        int width = i;
        if (!s.empty()) width = (i - s.top() - 1);
        if (ans < width*height) ans = width*height;
    }
    s.push(i);
}
```

# 히스토그램에서 가장 큰 직사각형

40

<https://www.acmicpc.net/problem/6549>

```
while (!s.empty()) {  
    int height = a[s.top()];  
    s.pop();  
    int width = n;  
    if (!s.empty()) {  
        width = n-s.top()-1;  
    }  
    if (ans < width*height) {  
        ans = width*height;  
    }  
}
```



# 히스토그램에서 가장 큰 직사각형

41

<https://www.acmicpc.net/problem/6549>

- 소스: <http://codeplus.codes/a21dcca711954ae5b979c3e980764fd5>

# 오아시스 재결합

<https://www.acmicpc.net/problem/3015>

- N명이 한 줄로 줄을 서있다
- A와 B가 서로 볼 수 있으려면, 두 사람 사이에 A 또는 B보다 키가 큰 사람이 없어야 한다
- 줄에 서있는 사람의 키가 주어졌을 때, 서로 볼 수 있는 쌍의 수 구하기

# 오아시스 재결합

<https://www.acmicpc.net/problem/3015>

- 먼저, 문제에 나와있지 않은 조건을 하나 추가해서 문제를 푼다
- 모든 사람의 키가 모두 서로 다르다고 가정
- A 뒤에 B가 줄을 섰다
- B가 A보다 키가 크다
- .....A.....B.....
- B의 뒤에 있는 사람들은 절대로 A를 볼 수 없다

# 오아시스 재결합

<https://www.acmicpc.net/problem/3015>

- 한 명씩 줄을 서는 경우를 생각
- 줄을 설 때마다 자기 앞에 총 몇 명을 볼 수 있는지 계산해보기
- 스택에 들어있는 사람은 뒤에 줄을 서는 사람이 볼 수도 있는 사람
- 스택에서 이미 나온 사람은 뒤에 줄을 서는 사람이 절대 볼 수 없는 사람
- 따라서, 스택에는 항상 키가 작아지는 순으로 저장되게 됨
- 즉, 스택의 top이 스택에 들어있는 사람 중에서 키가 가장 작은 사람

# 오아시스 재결합

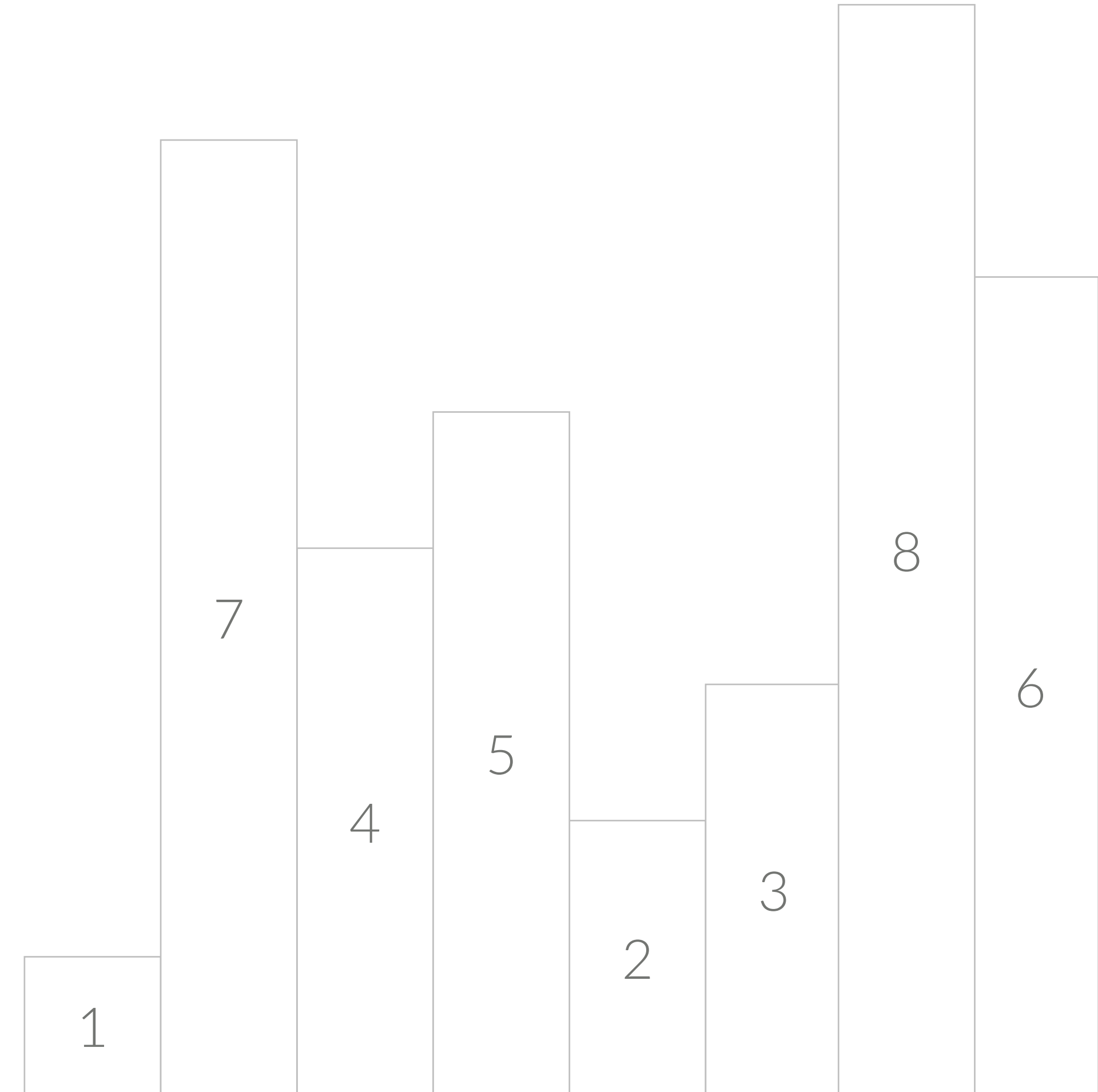
<https://www.acmicpc.net/problem/3015>

- 각 사람 A가 들어올 때마다 다음을 검사한다
  - 스택의 가장 위에 있는 사람 S와 키를 비교
  - S보다 A가 키가 크면
  - A의 뒤에 줄을 서는 사람은 절대로 A를 볼 수 없음
  - 따라서, 정답에 1을 더하고
  - S를 스택에서 빼고 위의 과정을 반복
- 이 때, 스택이 비어있지 않으면 정답에 1을 더한다
  - 왜냐하면, 이제 스택의 가장 위에 있는 사람은 A보다 키가 큰 사람인데, 그 사람은 볼 수 있기 때문
- 이제, A를 스택에 추가한다.

# 오아시스 재결합

<https://www.acmicpc.net/problem/3015>

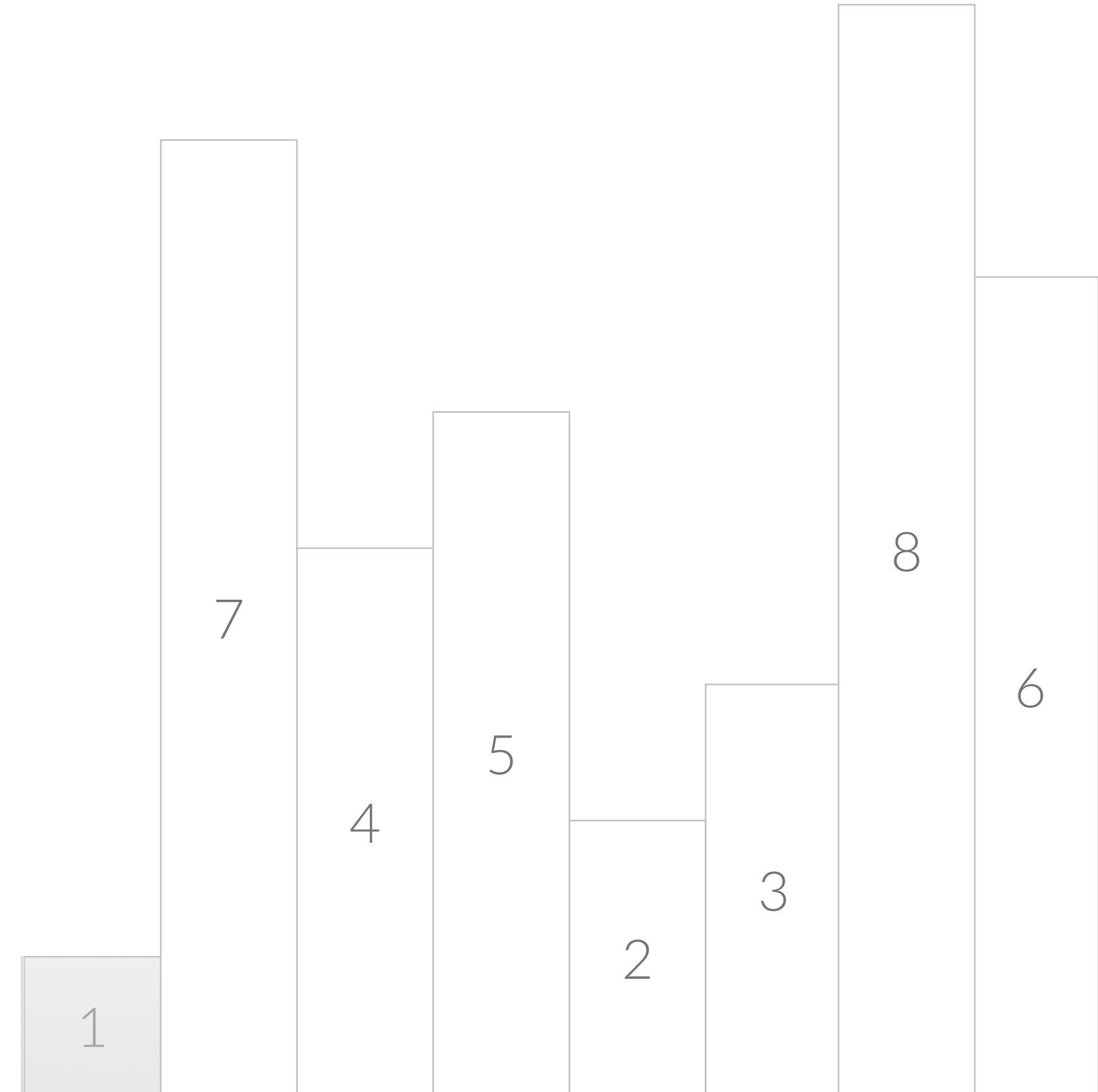
- 1 7 4 5 2 3 8 6의 경우를 생각해보자
- 볼 수 있는 쌍의 수: 11개



# 오아시스 재결합

<https://www.acmicpc.net/problem/3015>

- 스택: 1

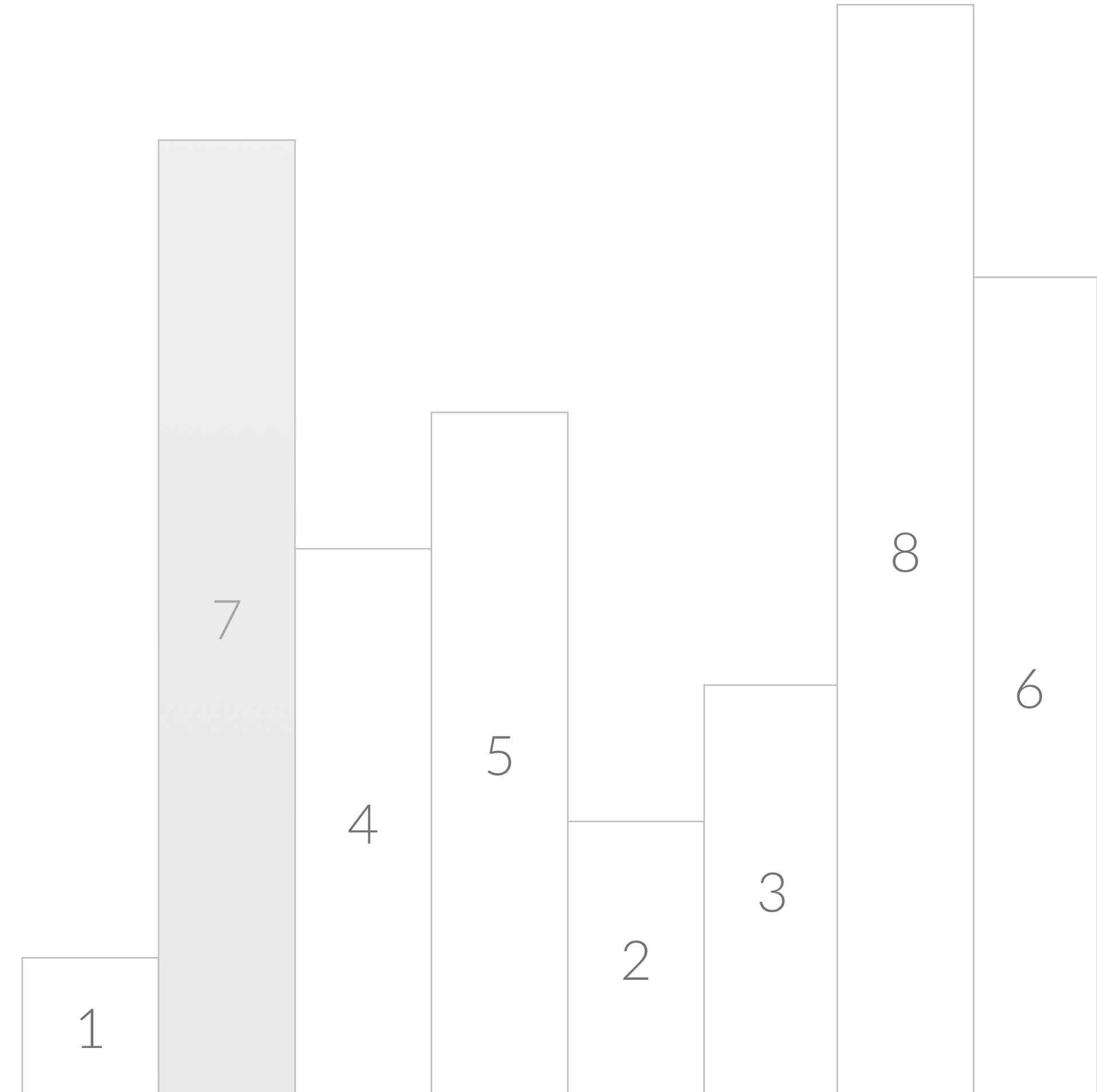


# 오아시스 재결합

48

<https://www.acmicpc.net/problem/3015>

- 스택: 1
- 1은 7보다 키가 작다
- 7의 뒤에 있는 사람은 절대로 1을 볼 수 없다
- 정답에 1을 더해주고 (1과 7)
- 1을 스택에서 제거하고 7을 스택에 추가
- 스택: 7



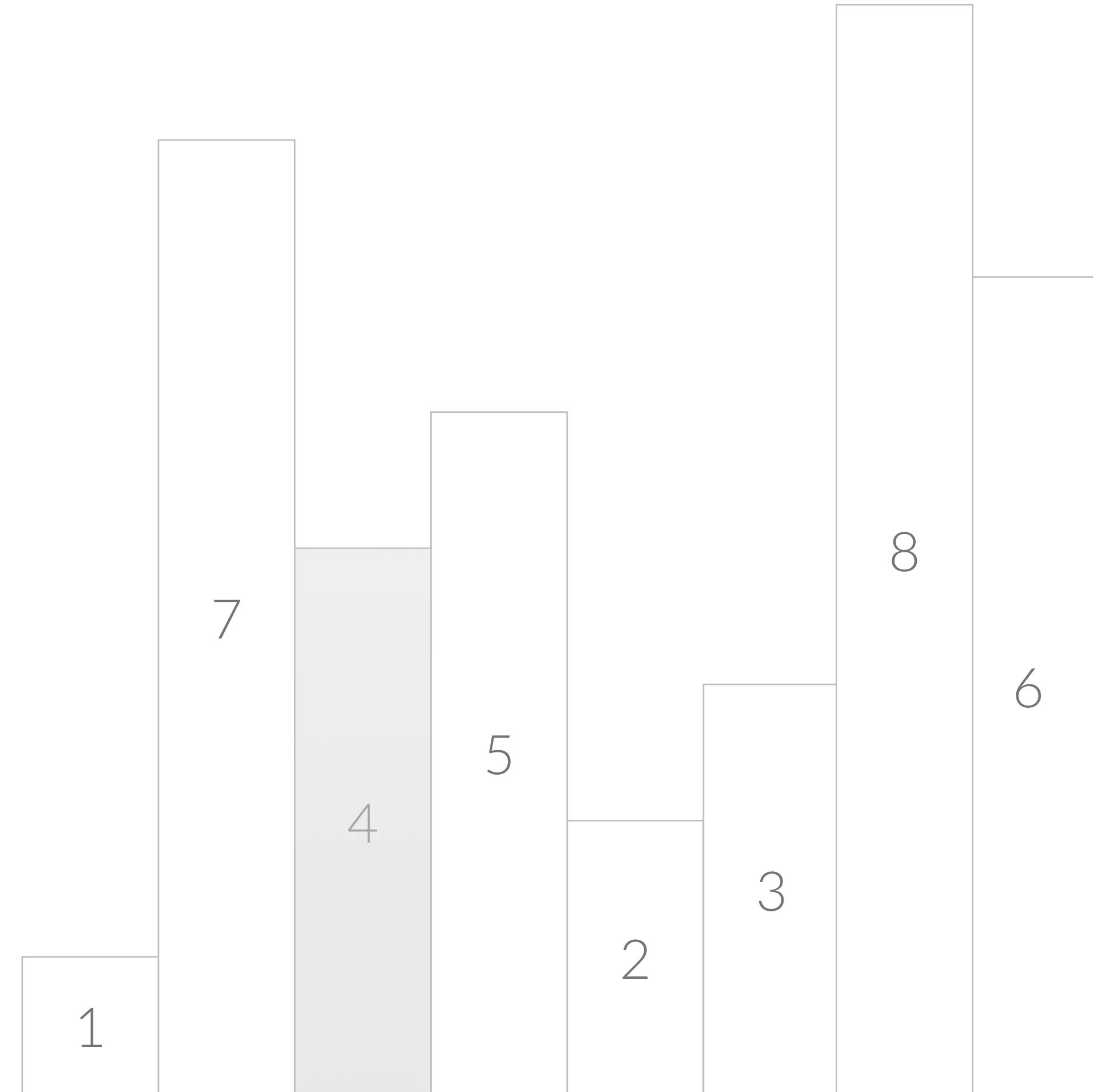


# 오아시스 재결합

49

<https://www.acmicpc.net/problem/3015>

- 스택: 7
- 7은 4보다 키가 크다
- 따라서, 스택에는 아무 일도 일어나지 않는다
- 스택이 비어있지 않기 때문에
- 정답에 1을 추가한다 (7과 4)
- 4를 스택에 추가한다
- 스택: 7 4

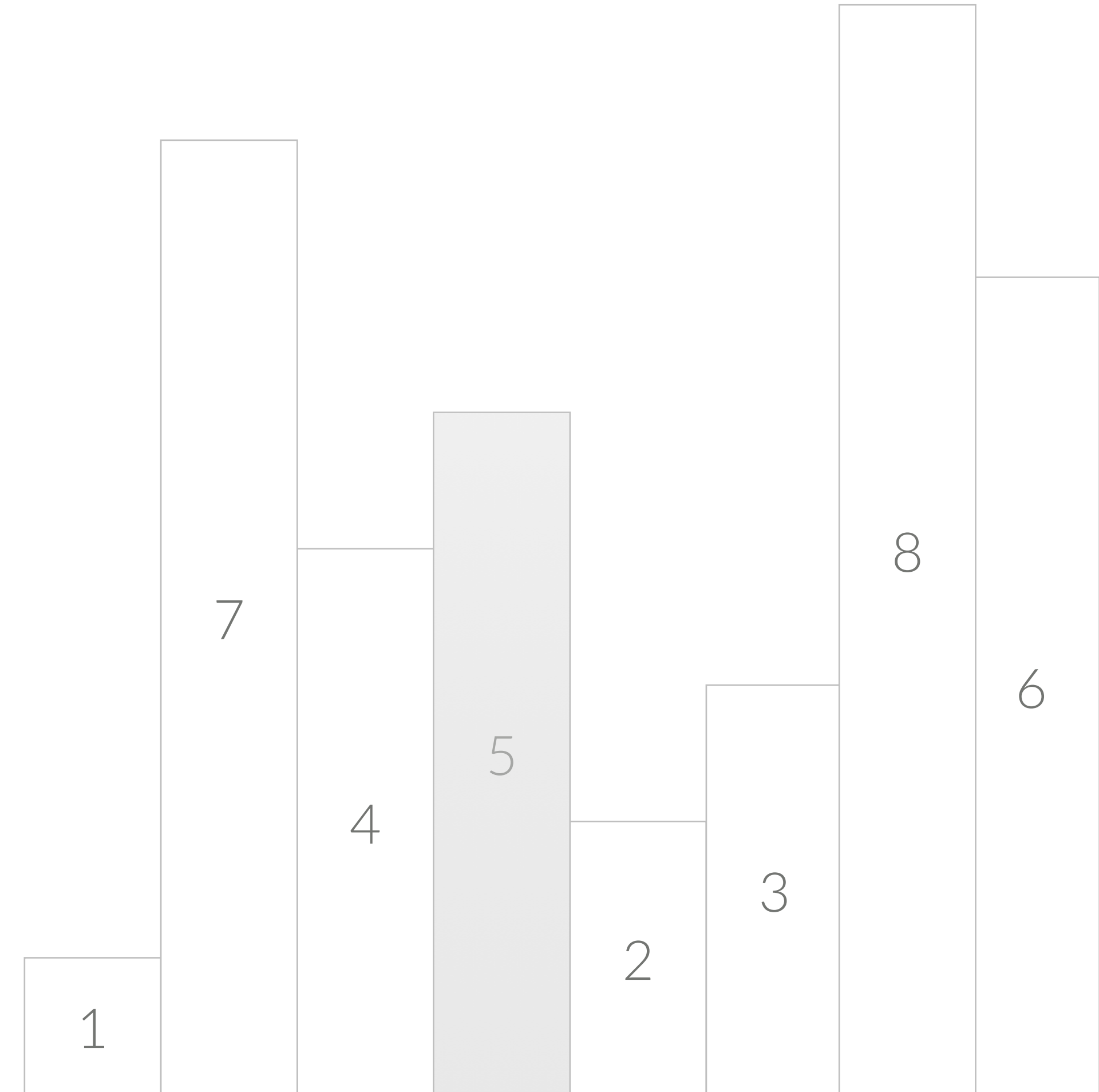


# 오아시스 재결합

50

<https://www.acmicpc.net/problem/3015>

- 스택: 7 4
- 4는 5보다 키가 작다
- 4를 스택에서 제거하고
- 정답에 1을 더한다 (4와 5)
- 스택: 7
- 7은 5보다 키가 크다
- 스택이 비어있지 않기 때문에
- 정답에 1을 더하고 (7과 5)
- 5를 스택에 추가한다

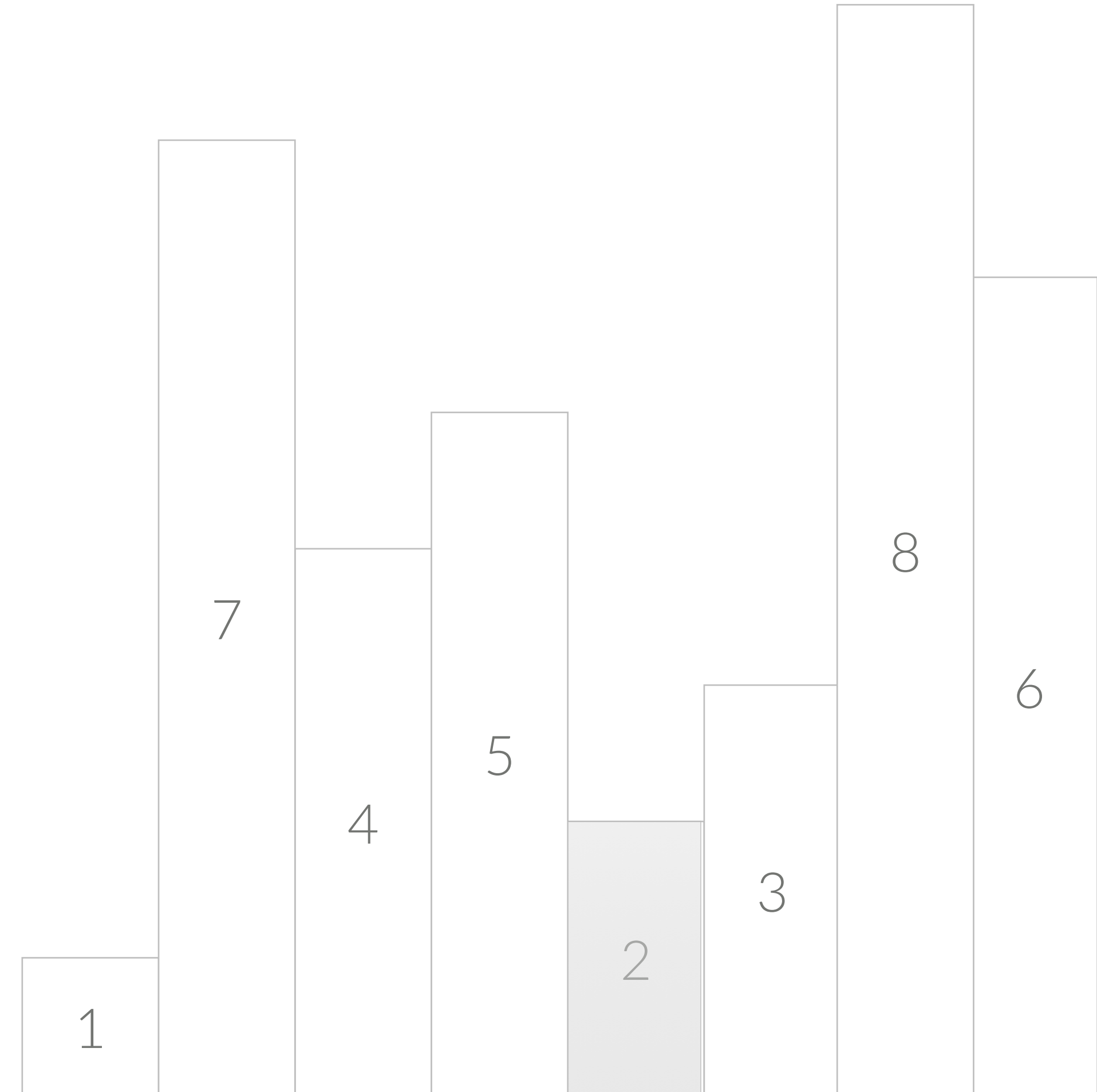


# 오아시스 재결합

51

<https://www.acmicpc.net/problem/3015>

- 스택: 7 5
- 2는 5보다 키가 작기 때문에 스택에 추가한다
- 스택이 비어있지 않기 때문에
- 정답에 1을 더한다 (5와 2)
- 스택: 7 5 2

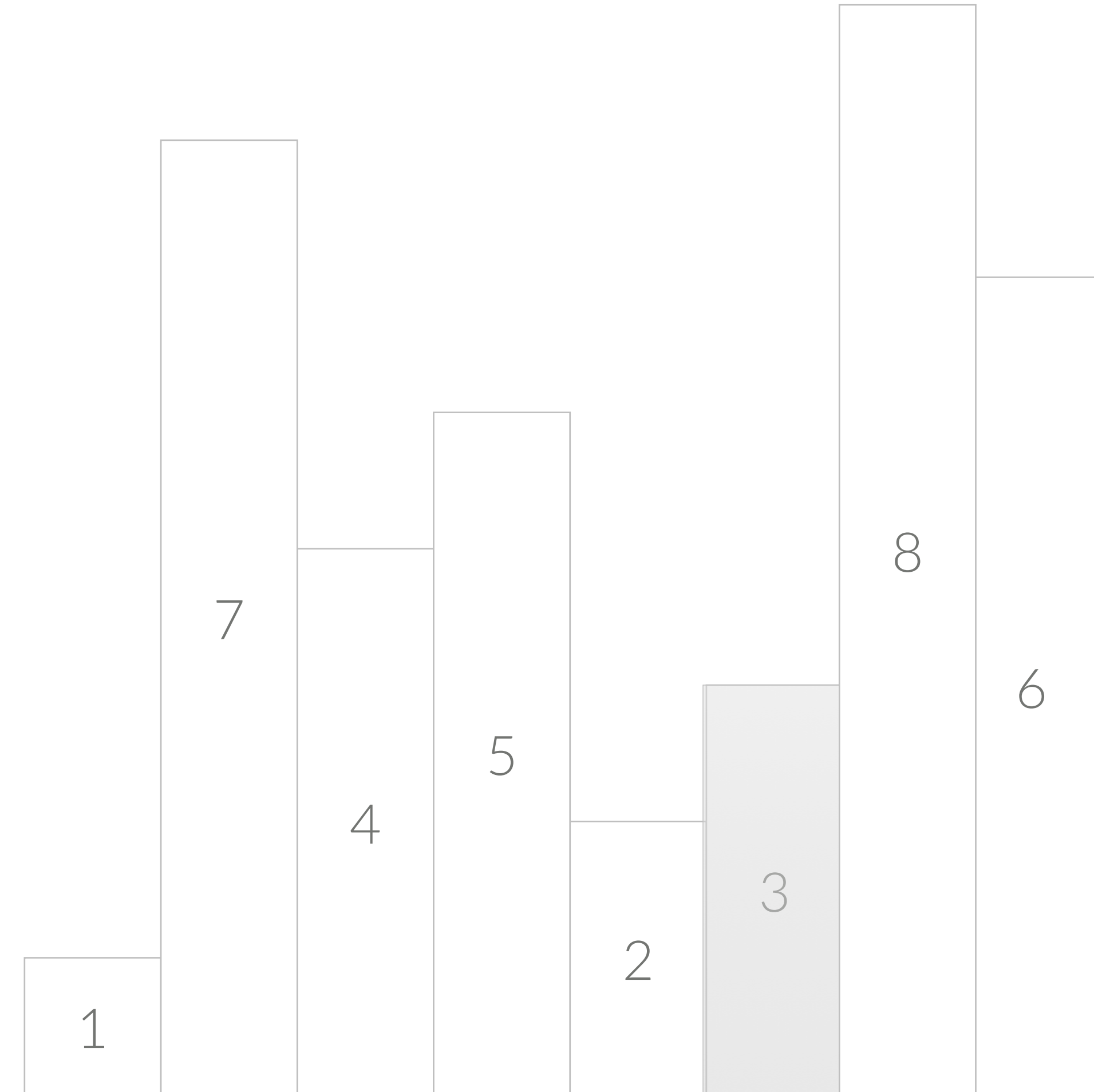


# 오아시스 재결합

52

<https://www.acmicpc.net/problem/3015>

- 스택: 7 5 2
- 2는 3보다 키가 작다.
- 3의 뒤에 있는 사람은 2를 절대로 볼 수가 없다
- 정답에 1을 더해준다 (2와 3)
- 스택: 7 5
- 5는 3보다 키가 크다.
- 3을 스택에 추가하고, 정답에 1을 더한다 (5와 3)
- 스택: 7 5 3

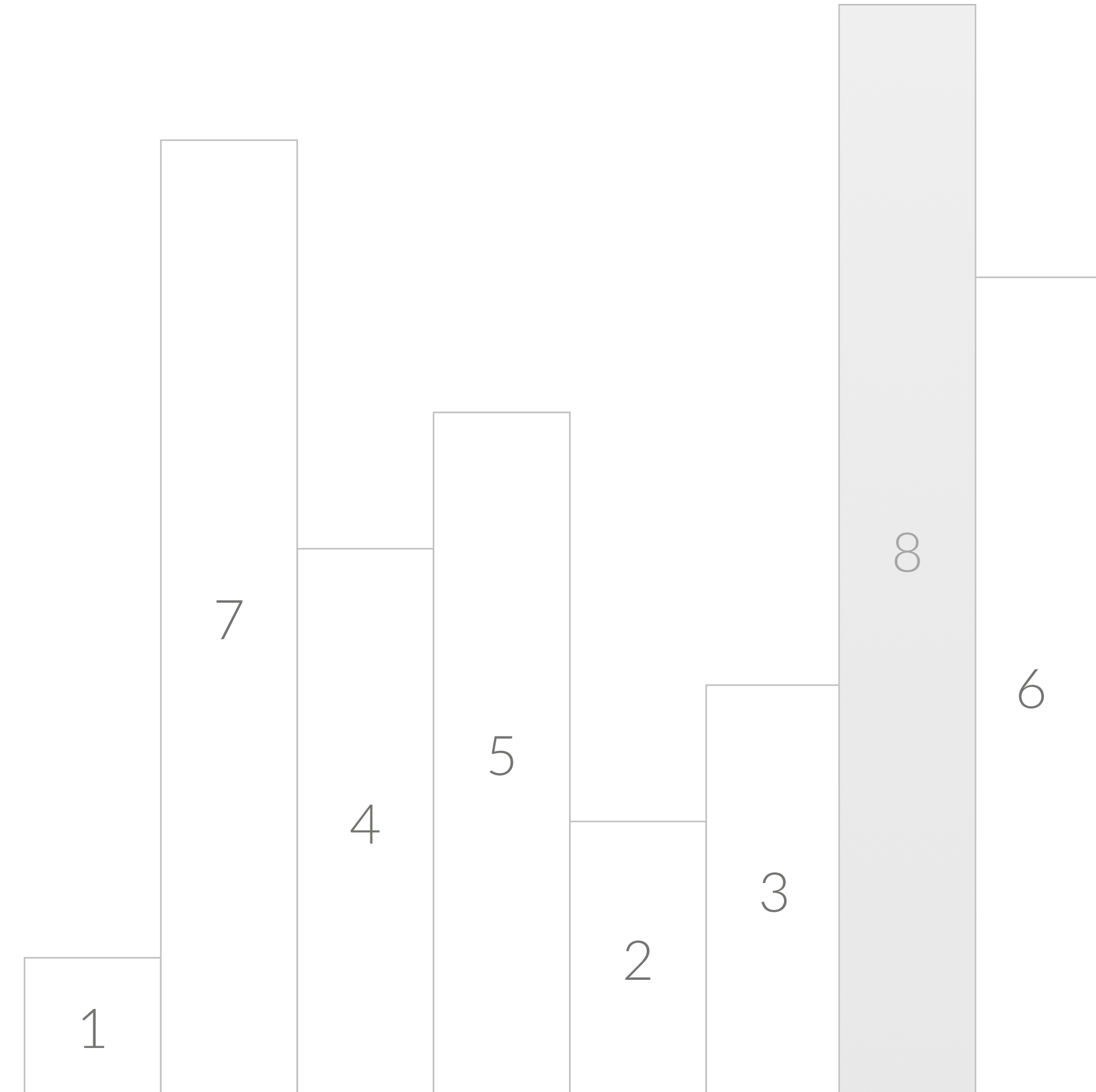


# 오아시스 재결합

53

<https://www.acmicpc.net/problem/3015>

- 스택: 7 5 3
- 8은 3보다 키가 크다
- 8의 뒤에 있는 사람을 3를 볼 수 없다
- 3를 스택에서 빼고 정답에 1을 더한다 (3와 8)
- 스택: 7 5
- 8은 5보다 키가 크다
- 8의 뒤에 있는 사람은 5를 볼 수 없다
- 5를 스택에서 빼고 정답에 1을 더한다 (5와 8)
- 스택: 7

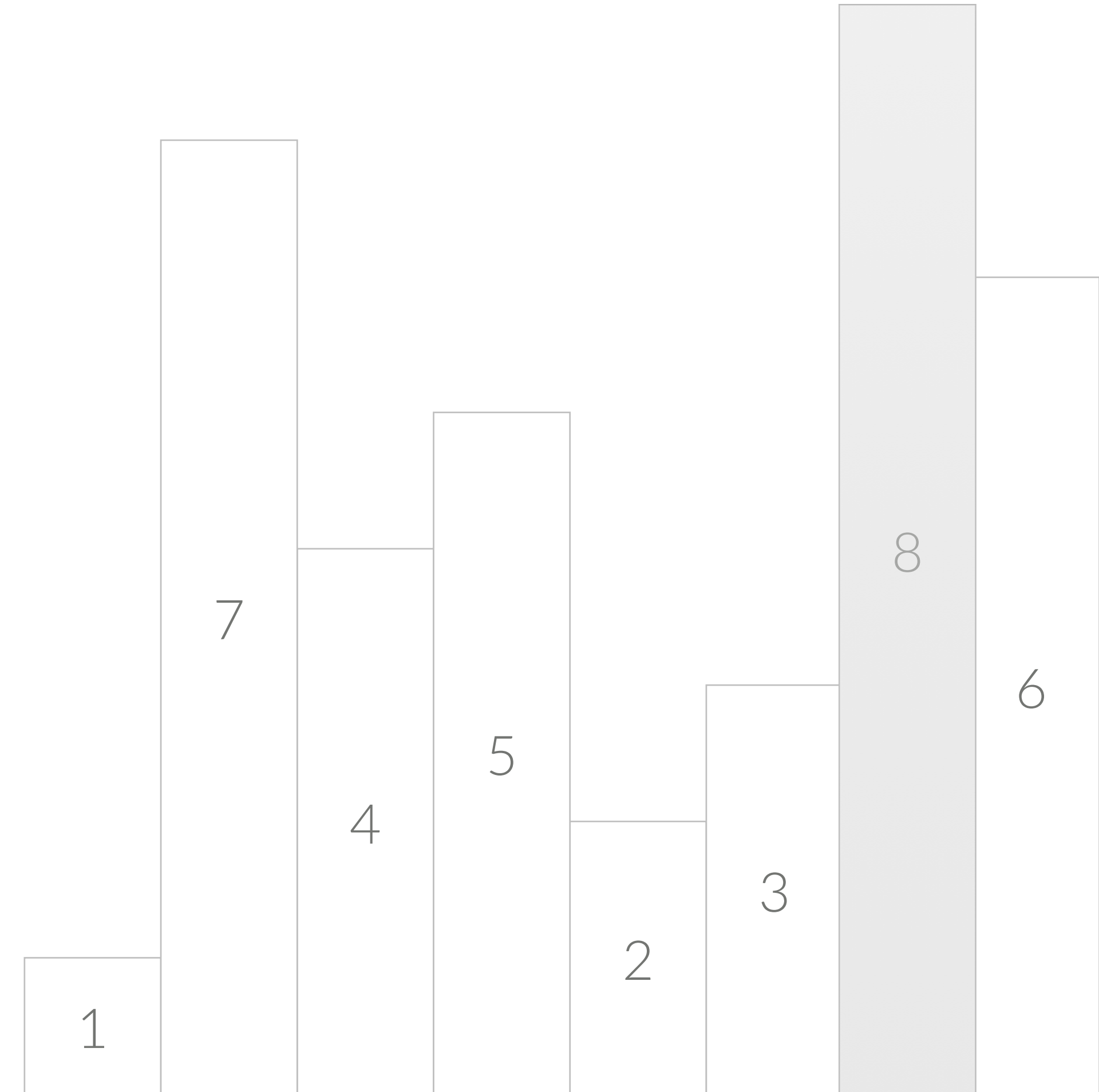


# 오아시스 재결합

54

<https://www.acmicpc.net/problem/3015>

- 스택: 7
- 8은 7보다 키가 크다
- 8의 뒤에 있는 사람을 7를 볼 수 없다
- 7를 스택에서 빼고 정답에 1을 더한다 (7와 8)
- 스택:
- 스택이 비어있으니 이제 8을 넣는다
- 스택: 8

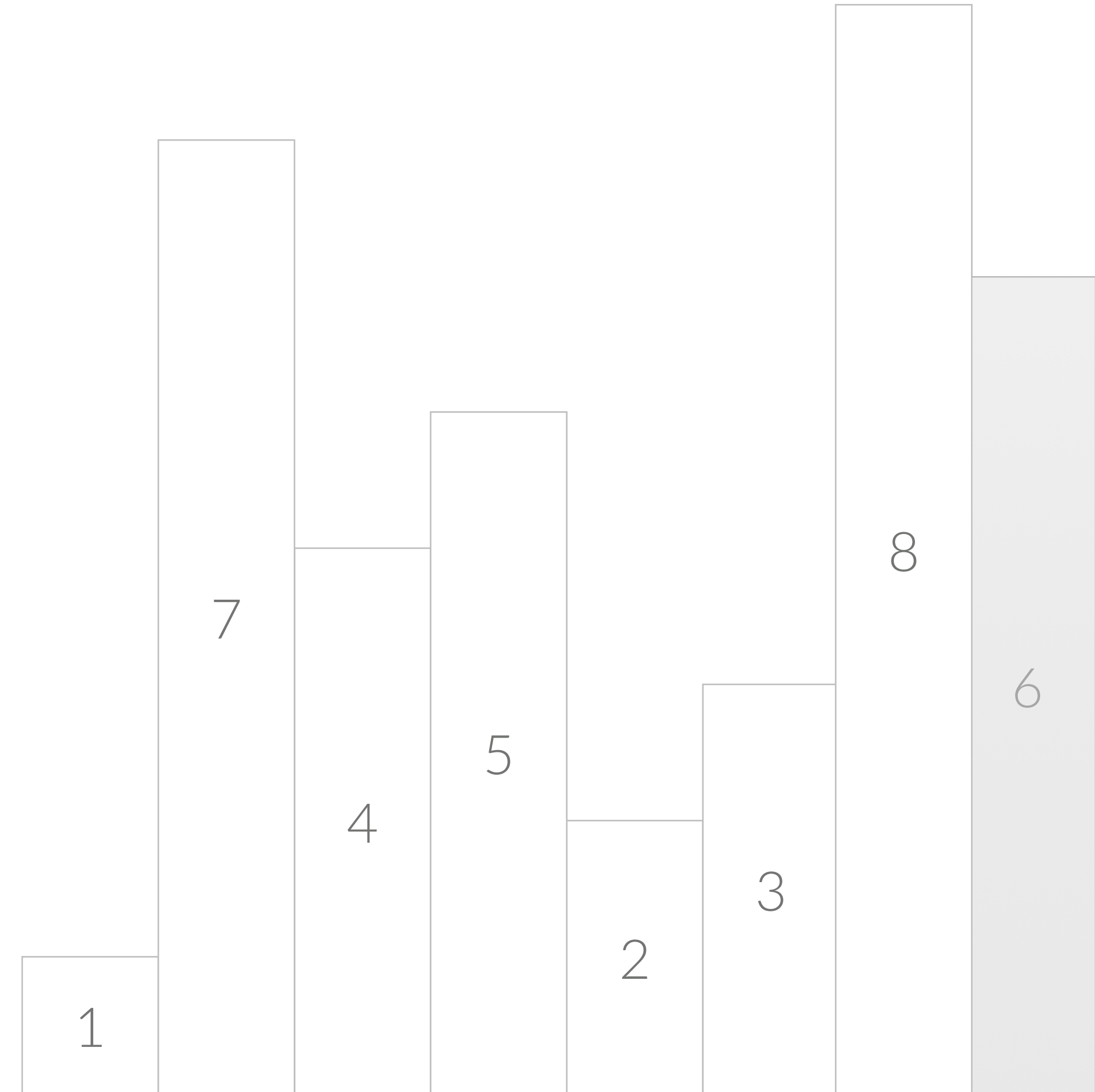


# 오아시스 재결합

55

<https://www.acmicpc.net/problem/3015>

- 스택: 8
- 6은 8보다 키가 작다
- 스택이 비어있지 않기 때문에
- 정답에 1을 더하고 (8과 6)
- 스택에 6을 추가한다
- 스택: 8 6



# 오아시스 재결합

<https://www.acmicpc.net/problem/3015>

```
for (int i=0; i<n; i++) {  
    while (!s.empty()) {  
        if (s.top() < a[i]) {  
            ans += 1;  
            s.pop();  
        } else {  
            break;  
        }  
    }  
    if (!s.empty()) ans += 1;  
    s.push(a[i]);  
}
```



# 오아시스 재결합

<https://www.acmicpc.net/problem/3015>

- 이제 원래 문제를 풀어야 한다
- 키가 같은 사람이 들어오기 때문에
- 스택에 키와, 그 키를 가진 사람의 수를 넣어서 문제를 푼다

# 오아시스 재결합

<https://www.acmicpc.net/problem/3015>

```
for (int i=0; i<n; i++) {
    pair<int,int> p = {a[i], 1};
    while (!s.empty()) {
        if (s.top().first <= a[i]) {
            ans += (long long)s.top().second;
            if (s.top().first == a[i]) {
                p.second += s.top().second;
            }
            s.pop();
        } else {
            break;
        }
    }
    if (!s.empty()) ans += 1LL;
    s.push(p);
}
```

# 오아시스 재결합

59

<https://www.acmicpc.net/problem/3015>

- 소스: <http://codeplus.codes/4b0438d27b884546be0f0dd38d28c491>

# 유니온 파인드

---

# 유니온 파인드

Union Find

61

- 상호 배타적 집합(Disjoint-set)이라고도 한다.
- 2가지 연산으로 이루어져 있다.
  1. Find:  $x$ 가 어떤 집합에 포함되어 있는지 찾는 연산
  2. Union:  $x$ 와  $y$ 가 포함되어 있는 집합을 합치는 연산
- 구현은 트리를 이용한다.
- $\text{parent}[i] = i$ 의 parent가 저장되어 있음

# 유니온 파인드

Union Find

62

- 가장 처음에는  $\text{parent}[i] = i$ 로 초기화 한다.



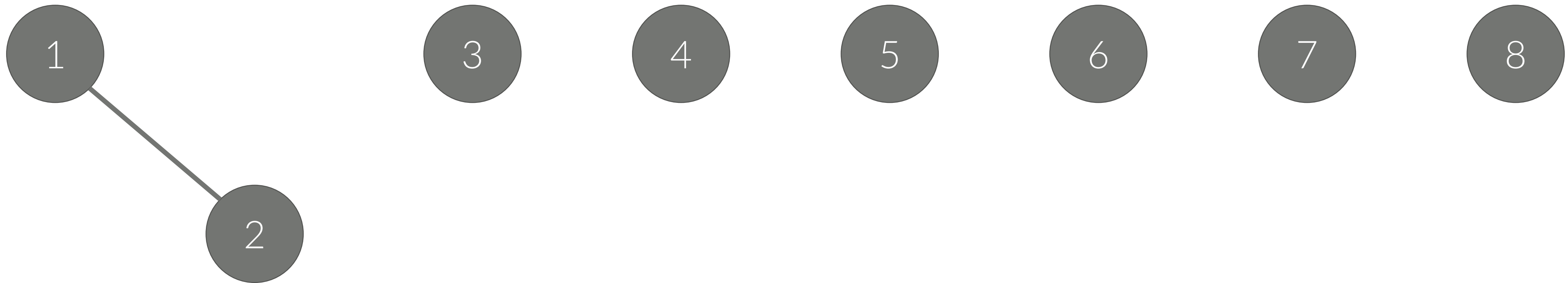
i	1	2	3	4	5	6	7	8
P[i]	1	2	3	4	5	6	7	8

# 유니온 파인드

Union Find

63

- Union (1, 2)



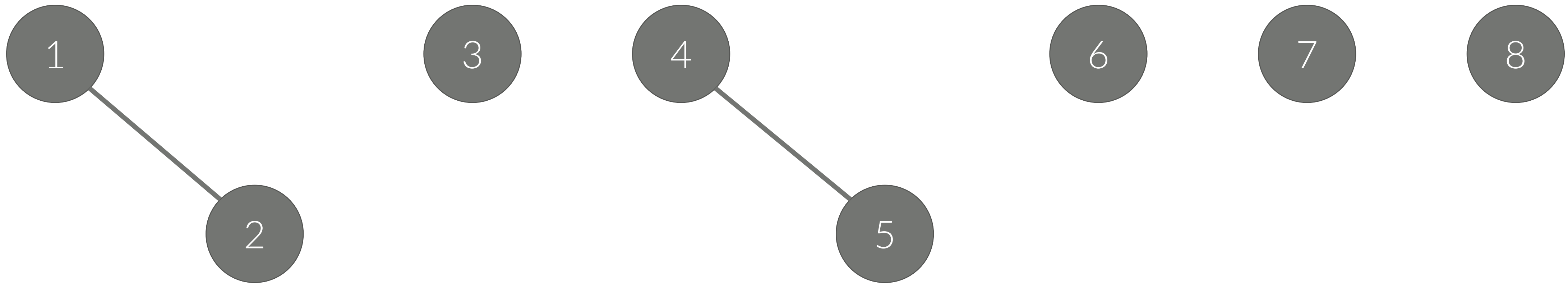
i	1	2	3	4	5	6	7	8
P[i]	1	1	3	4	5	6	7	8

# 유니온 파인드

Union Find

64

- Union (4, 5)



i	1	2	3	4	5	6	7	8
P[i]	1	1	3	4	4	6	7	8

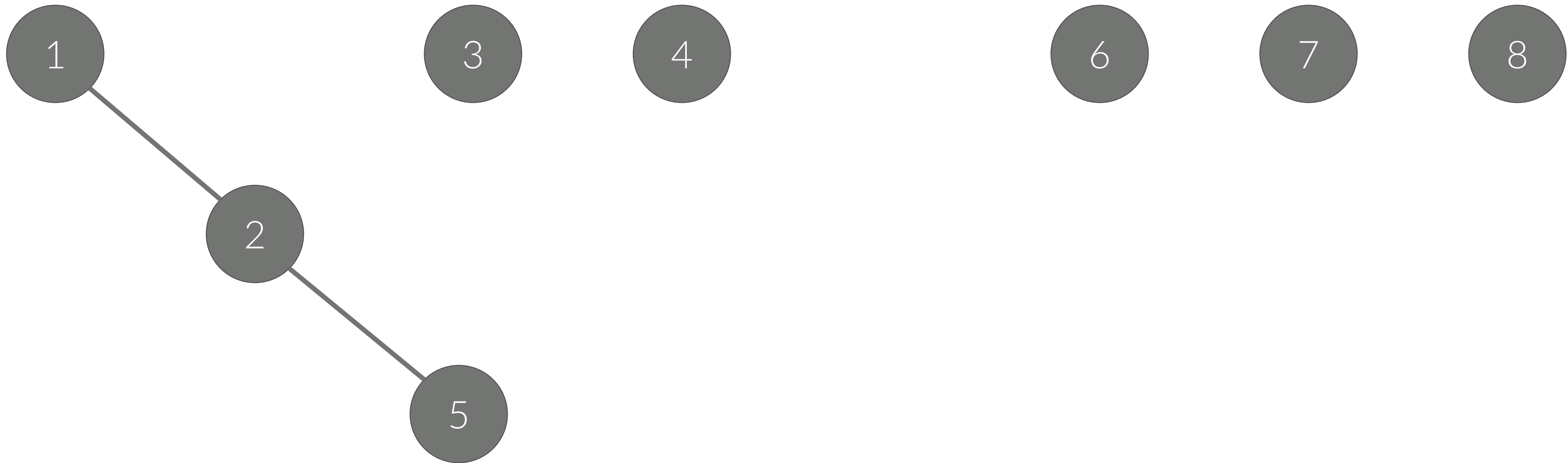


# 유니온 파인드

Union Find

65

- Union (2, 5)



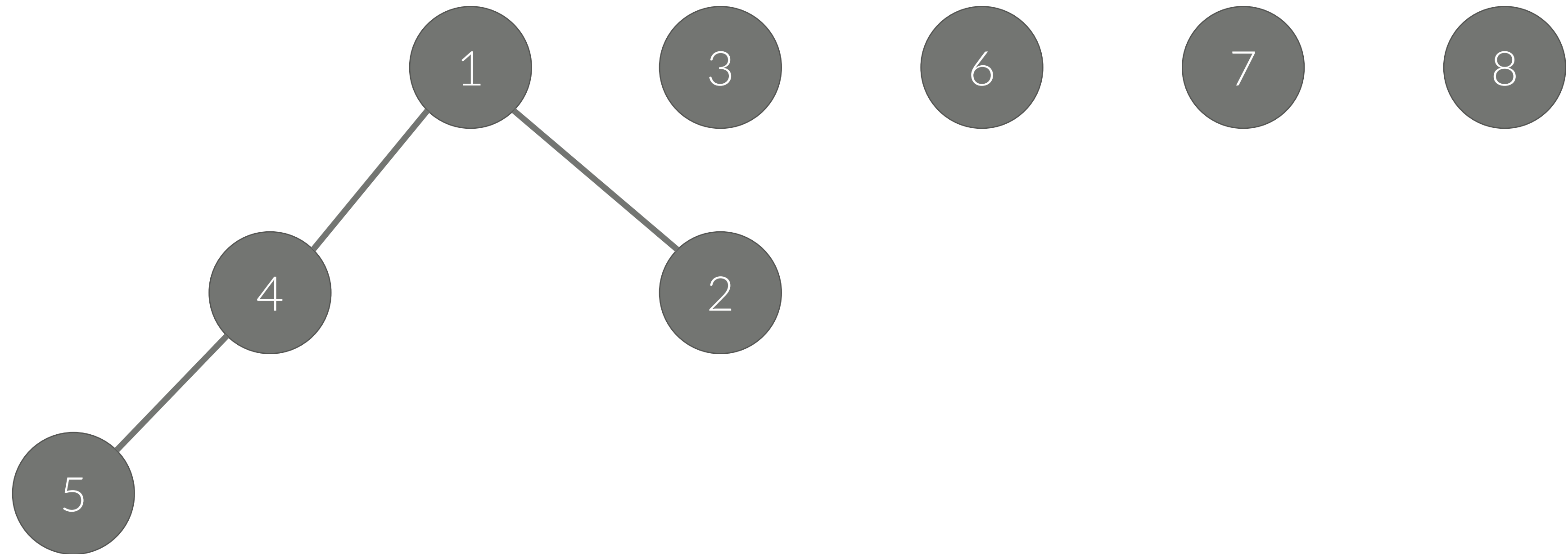
i	1	2	3	4	5	6	7	8
P[i]	1	1	3	4	2	6	7	8

# 유니온 파인드

66

Union Find

- Union(2, 5)
- Find(2) = 1
- Find(5) = 4
- Union(1, 4)

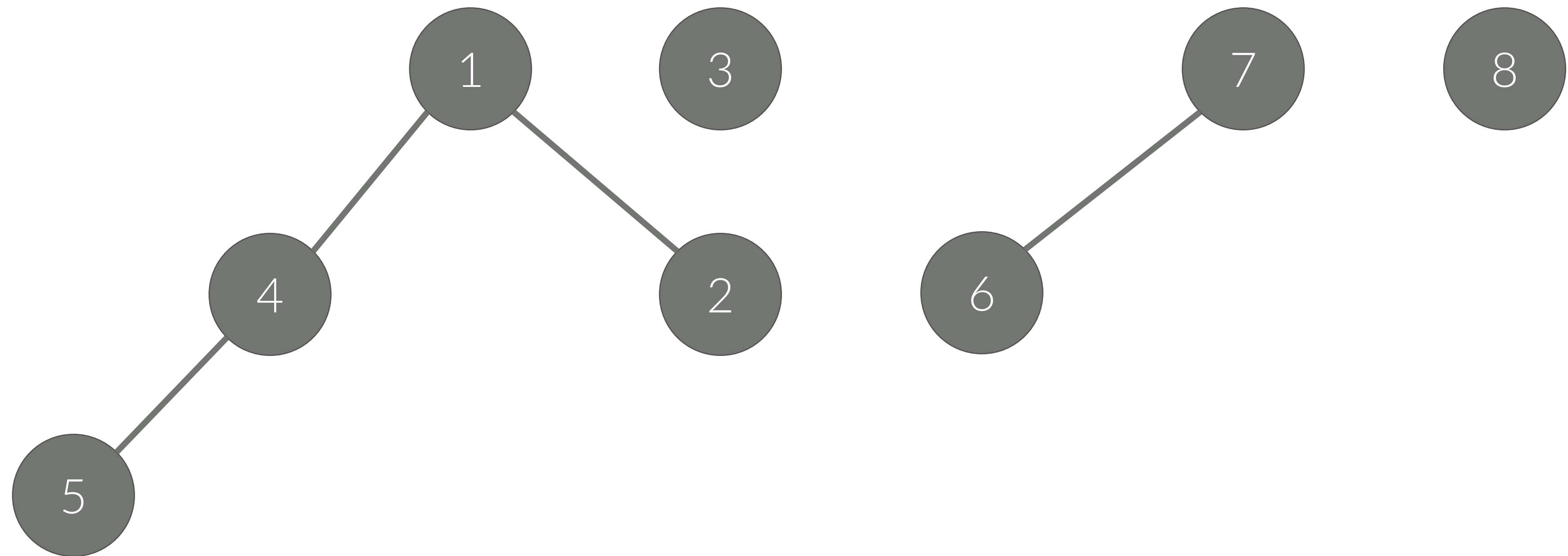


i	1	2	3	4	5	6	7	8
P[i]	1	1	3	1	4	6	7	8

# 유니온 파인드

Union Find

- Union(7, 6)



i	1	2	3	4	5	6	7	8
P[i]	1	1	3	1	4	7	7	8

# 유니온 파인드

## Union Find

- Find의 구현

```
int Find(int x) {  
    if (x == parent[x]) {  
        return x;  
    } else {  
        return Find(parent[x]);  
    }  
}
```

# 유니온 파인드

## Union Find

69

- Union의 구현
- Union(x, y): y의 parent를 x로 설정한다.

```
int Union(int x, int y) {  
    x = Find(x);  
    y = Find(y);  
    parent[y] = x;  
}
```

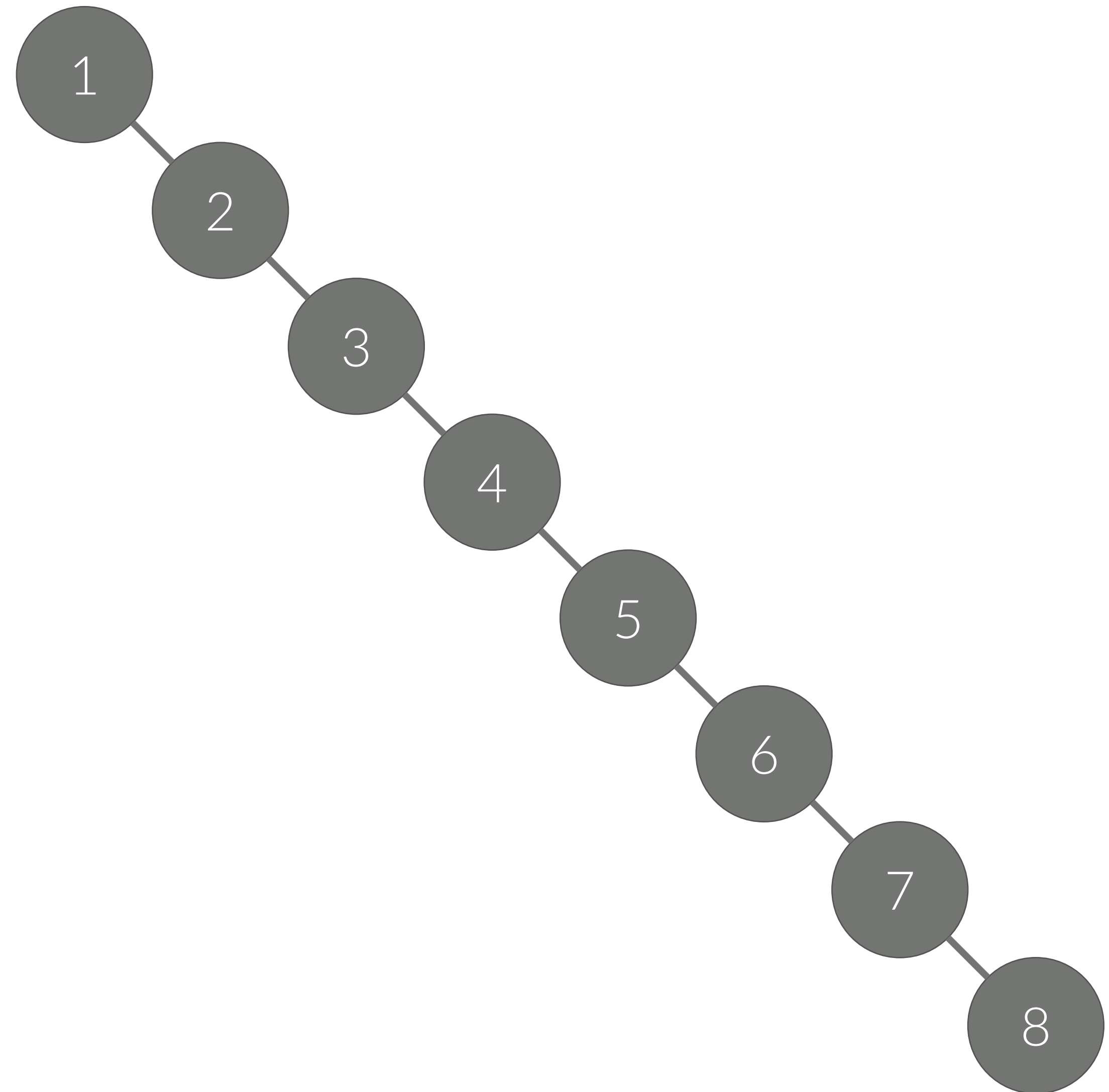
# 유니온 파인드

## Union Find

- Union의 구현
- Union(x, y): y의 parent를 x로 설정한다.

```
int Union(int x, int y) {  
    x = Find(x);  
    y = Find(y);  
    parent[y] = x;  
}
```

- 오른쪽 그림과 같은 문제가 생길 수 있다.
- find의 시간 복잡도가  $O(N)$ 이다.

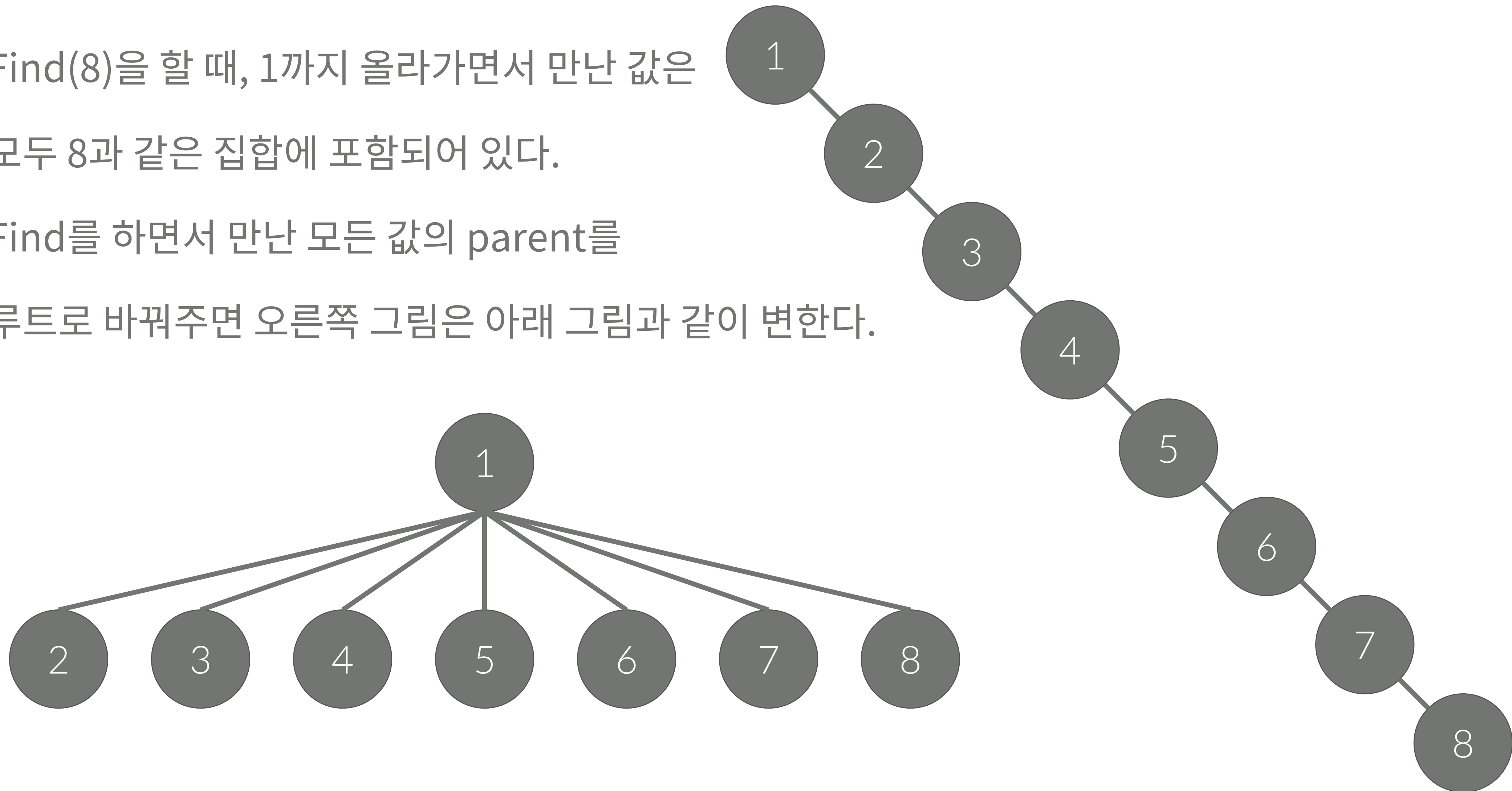


# 유니온 파인드

Union Find

71

- Find(8)을 할 때, 1까지 올라가면서 만난 값은
- 모두 8과 같은 집합에 포함되어 있다.
- Find를 하면서 만난 모든 값의 parent를
- 루트로 바꿔주면 오른쪽 그림은 아래 그림과 같이 변한다.

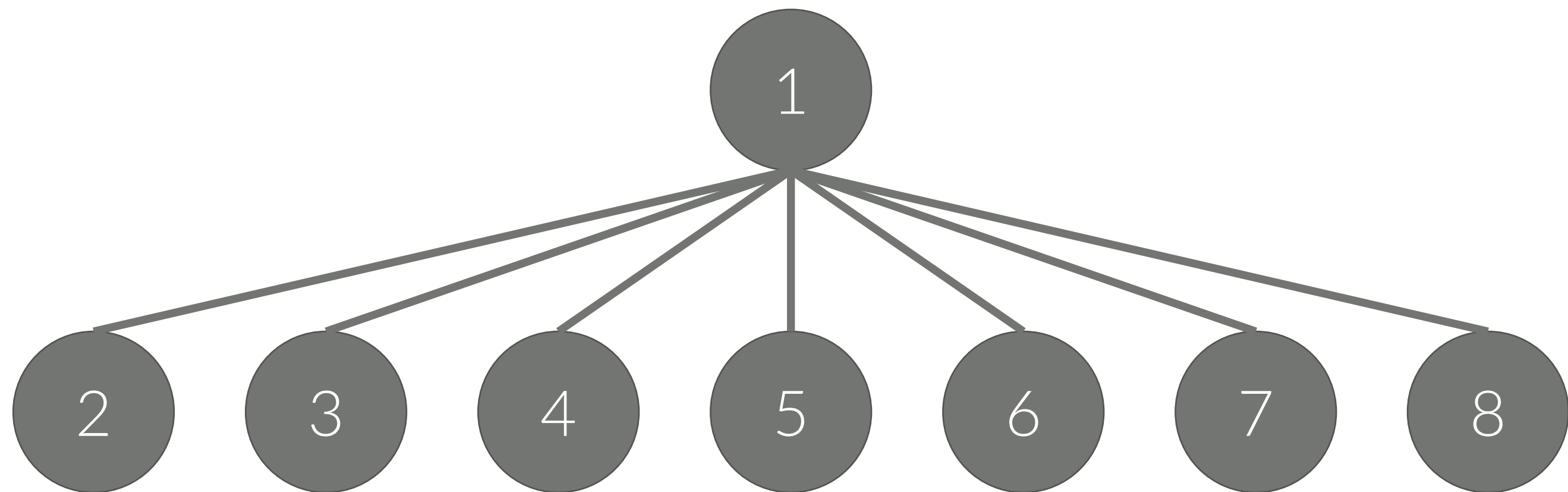


# 유니온 파인드

## Union Find

- 이 방식을 경로 압축이라고 한다.

```
int Find(int x) {  
    if (x == parent[x]) {  
        return x;  
    } else {  
        int y = Find(parent[x]);  
        parent[x] = y;  
        return y;  
    }  
}
```



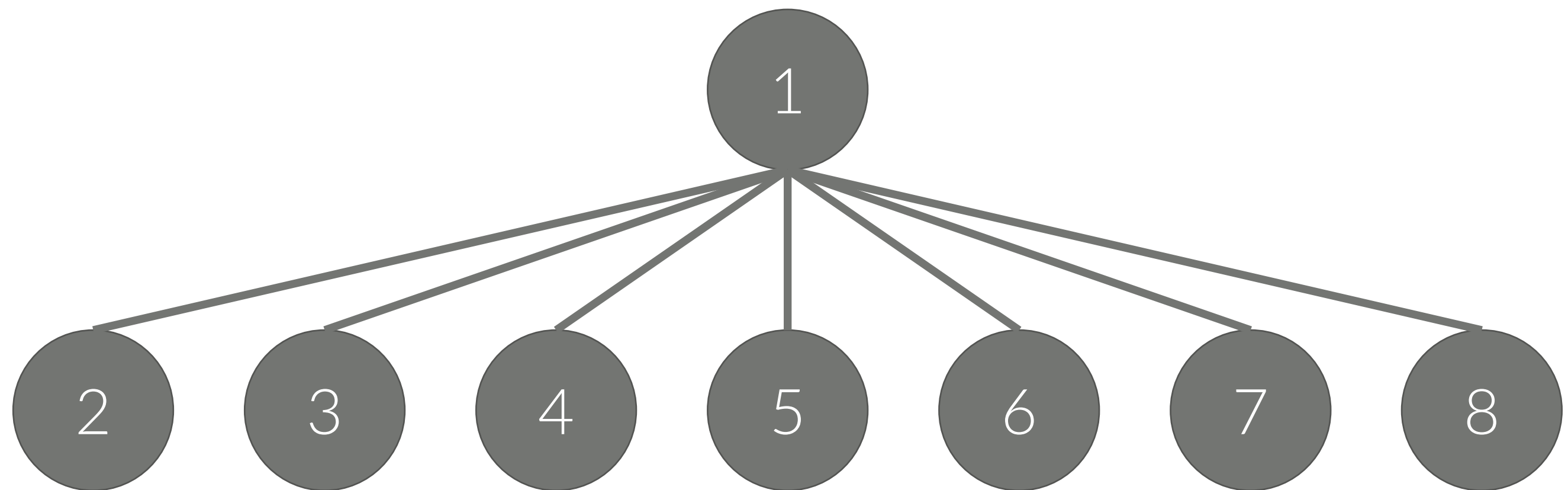


# 유니온 파인드

## Union Find

- 아래와 같이 구현할 수도 있다.

```
int Find(int x) {  
    if (x == parent[x]) {  
        return x;  
    } else {  
        return parent[x] = Find(parent[x]);  
    }  
}
```



# 유니온 파인드

Union Find

74

- Union을 구현할 때, 트리의 랭크를 기준으로 할 수 있다.
- 트리의 랭크는 높이와 같은 의미를 갖지만, 경로 압축을 사용하면 높이와 다른 값을 가질 수도 있다. 따라서, 높이 대신 랭크라는 표현을 사용한다.
- 랭크가 작은 것을 랭크가 높은 것의 자식으로 만든다.

# 유니온 파인드

## Union Find

```
int Union(int x, int y) {  
    x = Find(x);  
    y = Find(y);  
    if (x == y) {  
        return;  
    }  
    if (rank[x] < rank[y]) swap(x, y);  
    parent[y] = x;  
    if (rank[x] == rank[y]) {  
        rank[x] = rank[y] + 1;  
    }  
}
```

# 유니온 파인드

Union Find

76

- 경로 압축과 랭크를 이용한 유니온을 사용하면
- 연산의 시간 복잡도는  $O(\alpha(N))$  이다.
- $\alpha(N)$ 은 애커만 역함수 (Inverse Ackermann Function)로  $N < 2^{2^{2^{2^{16}}}}$  이면 5보다 작다.

# 집합의 표현

<https://www.acmicpc.net/problem/1717>

- 초기에  $\{0\}, \{1\}, \{2\}, \dots \{n\}$  이 각각  $n+1$ 개의 집합을 이루고 있다
- 여기에 합집합 연산과, 두 원소가 같은 집합에 포함되어 있는지를 확인하는 연산을 수행
- Disjoint-set 자료구조를 구현하는 문제

# 집합의 표현

<https://www.acmicpc.net/problem/1717>

- 소스: <http://codeplus.codes/ba5ed0c3656246cf8d49ab7b7969a3ac>

# 바이러스

<https://www.acmicpc.net/problem/2606>

- 컴퓨터가 연결된 상태가 주어진다
- 1번 컴퓨터가 웜 바이러스에 걸렸을 때, 웜 바이러스에 걸리게 되는 컴퓨터의 수를 구하는 문제

# 바이러스

<https://www.acmicpc.net/problem/2606>

- 유니온 파인드로 풀 수 있다.
- 모든 간선을 Union하면 된다.



# 바이러스

<https://www.acmicpc.net/problem/2606>

- 소스: <http://codeplus.codes/b4a83a10718042fba540fab23222d562>
- 제한이 크지 않아 경로 압축만 사용했다.

한

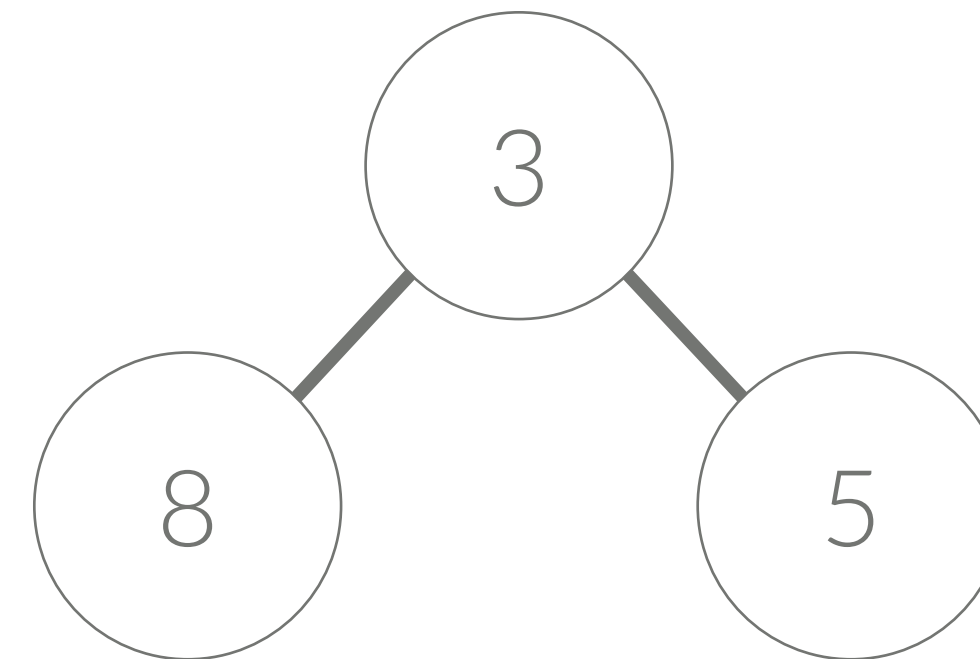
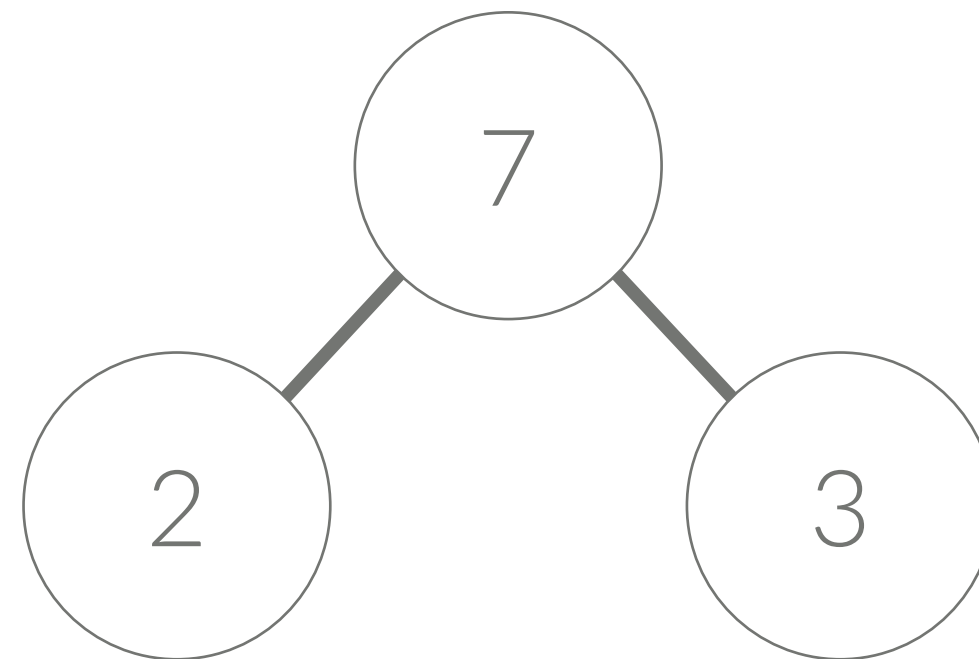
---

# 힙

## Heap

83

- 최대 힙
  - 부모 노드는 자식 노드에 들어있는 값보다 크다
- 최소 힙
  - 부모 노드는 자식 노드에 들어있는 값보다 작다.
- 힙은 완전 이진 트리 (Complete Binary Tree) 이다.

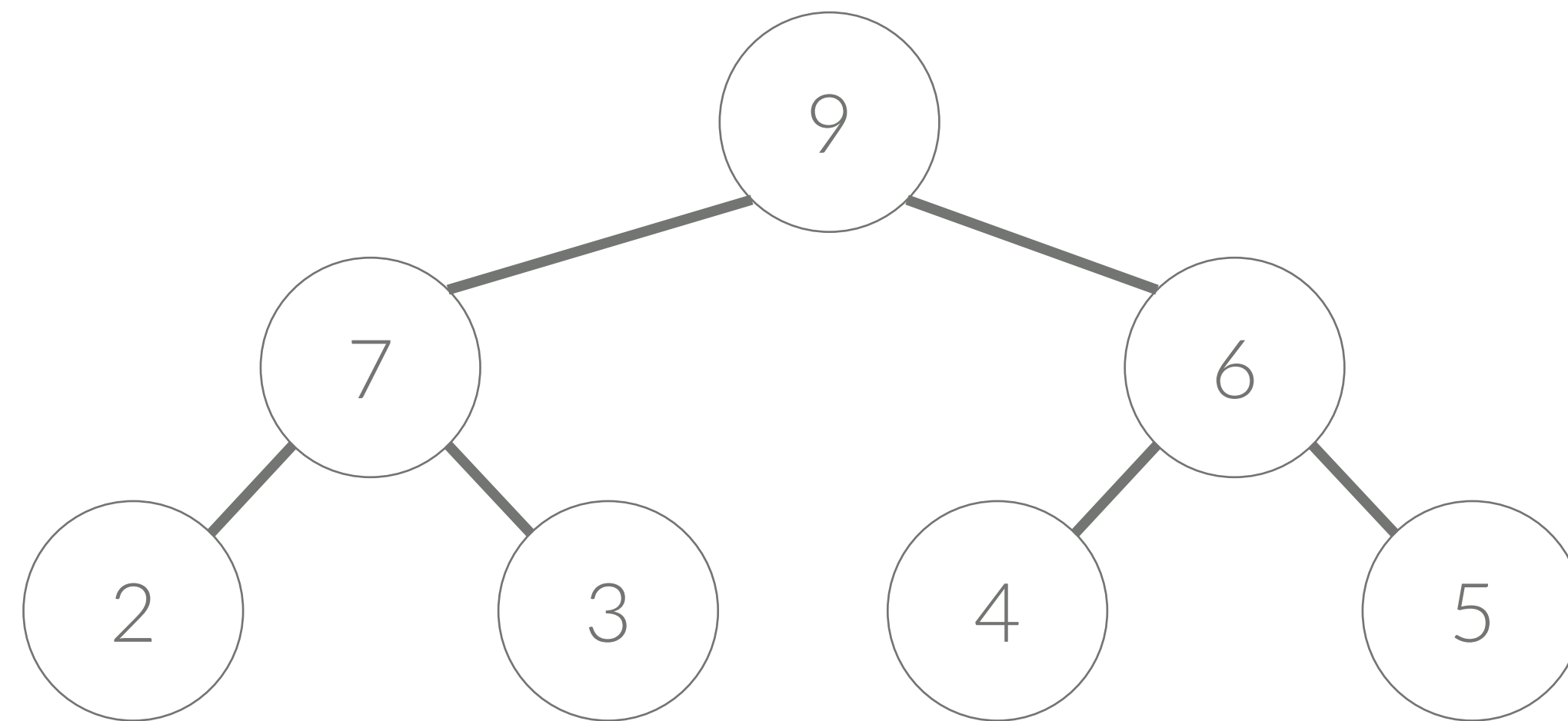


# 최대 힙

Max-Heap

84

- 최대 힙에서 가장 큰 값은 루트에 들어가 있다.
- N개가 힙에 들어가있으면 높이는  $\lg N$ 이 된다.

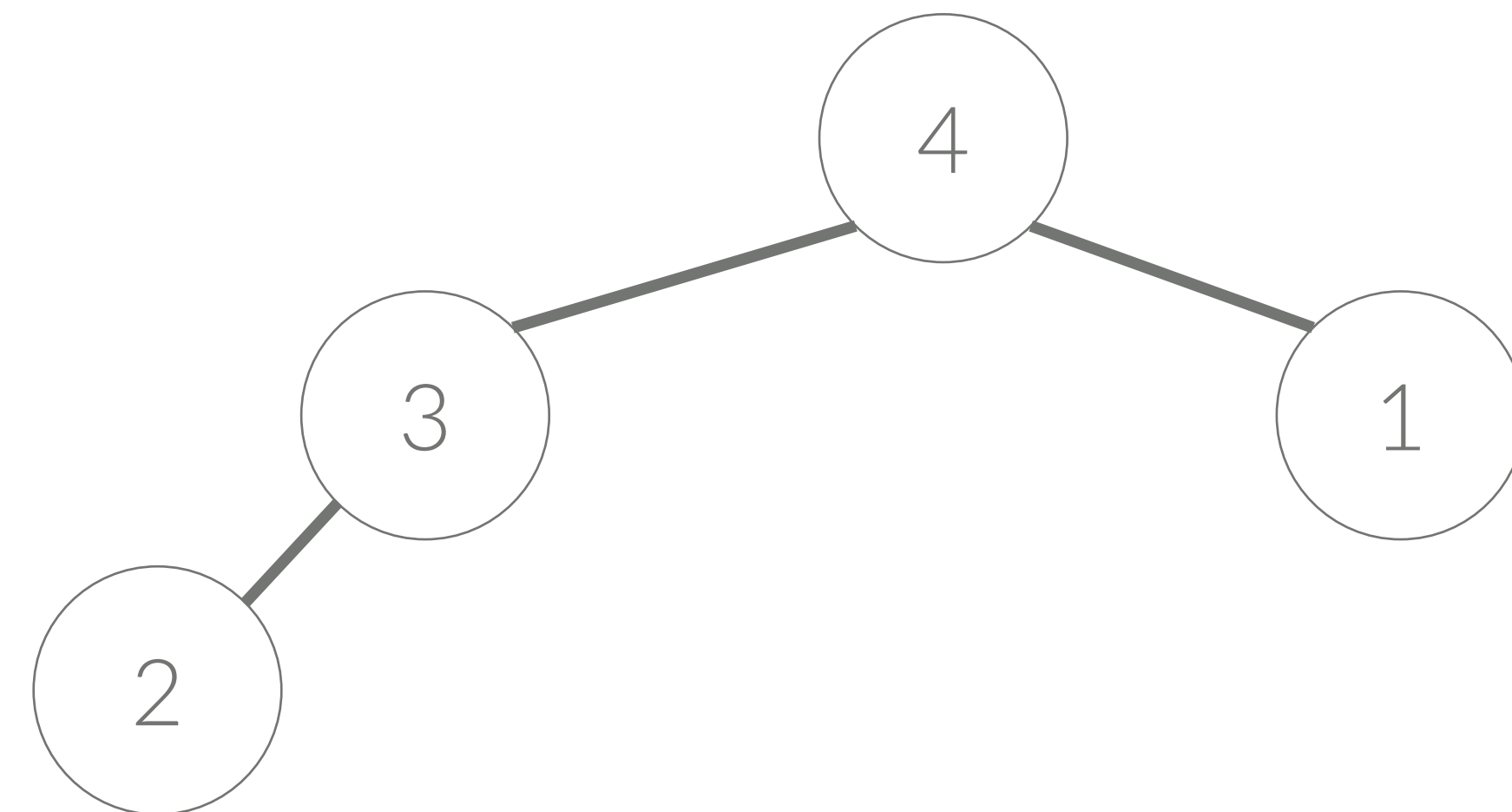


# 최대 힙 삽입

Max-Heap

85

- 가장 마지막 위치에 새로운 수를 넣는다.
- 그 수와 부모를 계속해서 비교해가면서
- 루트 < 자식 이면 두 수를 바꿔준다.
- 이제 최대 힙에 5를 넣어보자

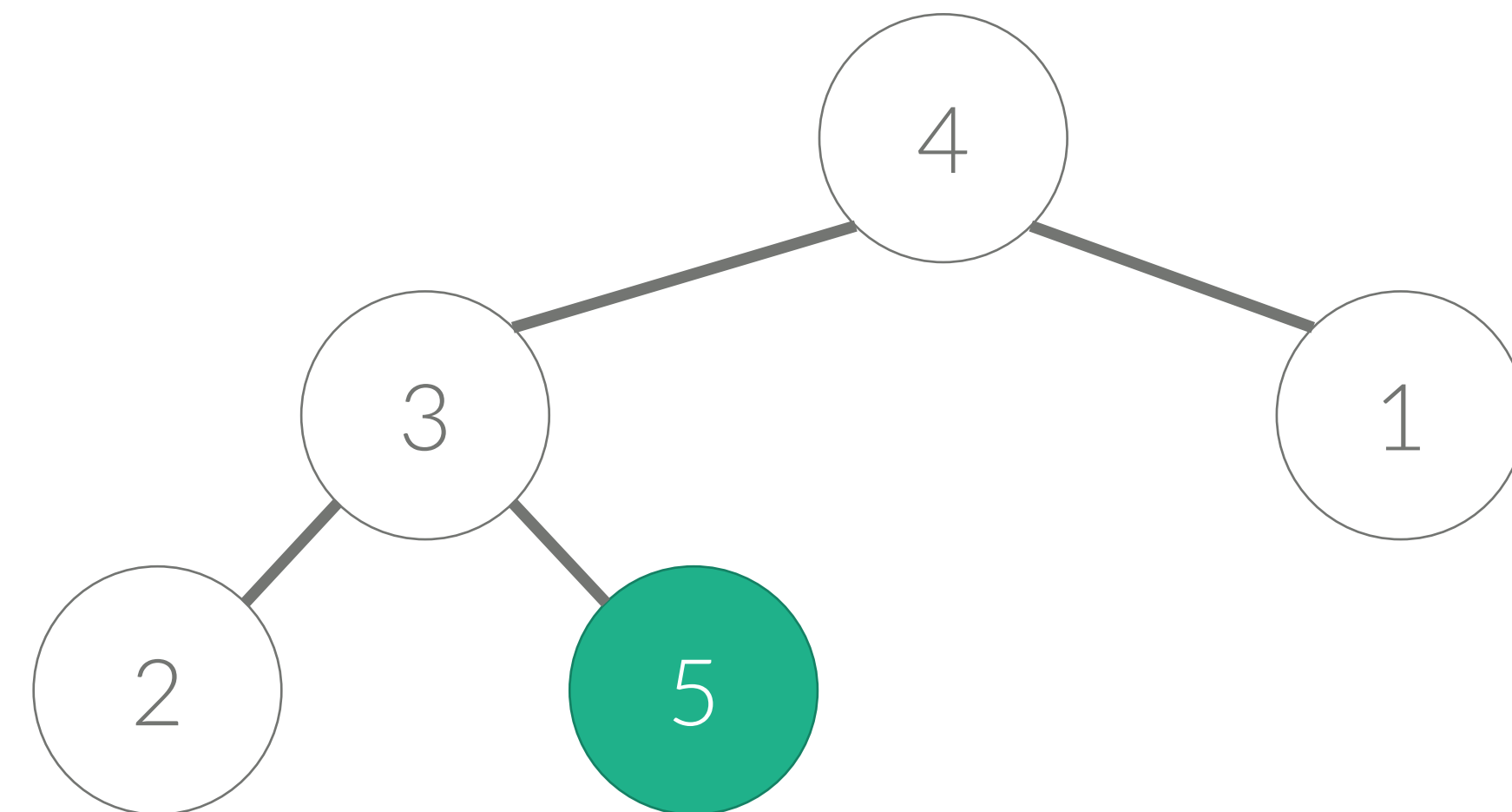


# 최대 힙 삽입

Max-Heap

- 5의 위치

86

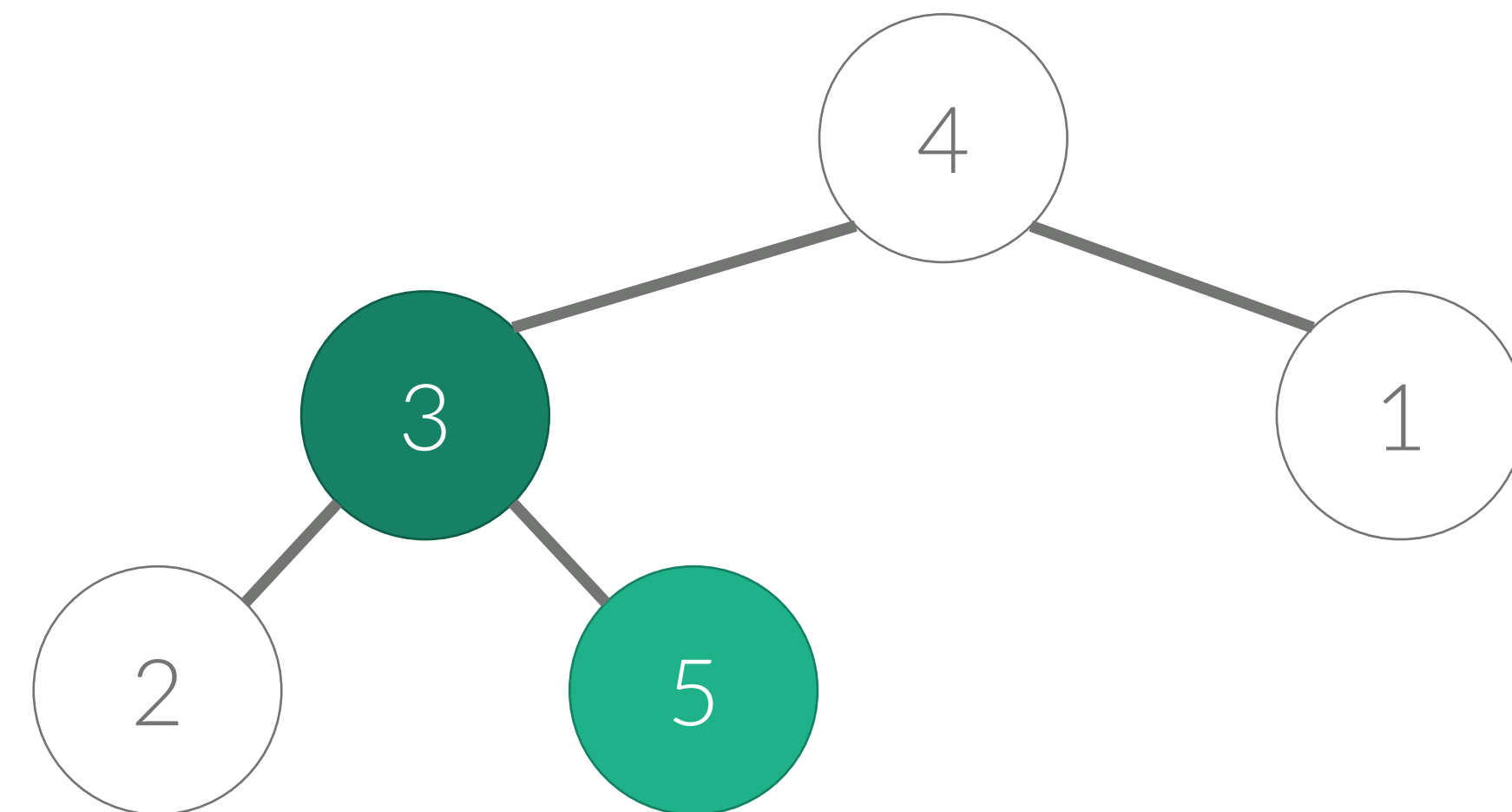


# 최대 힙 삽입

Max-Heap

87

- 5와 부모를 비교한다.
- 이 경우에  $3 < 5$  이기 때문에 최대 힙의 성질을 만족하지 않는다
- 따라서 두 수를 교환

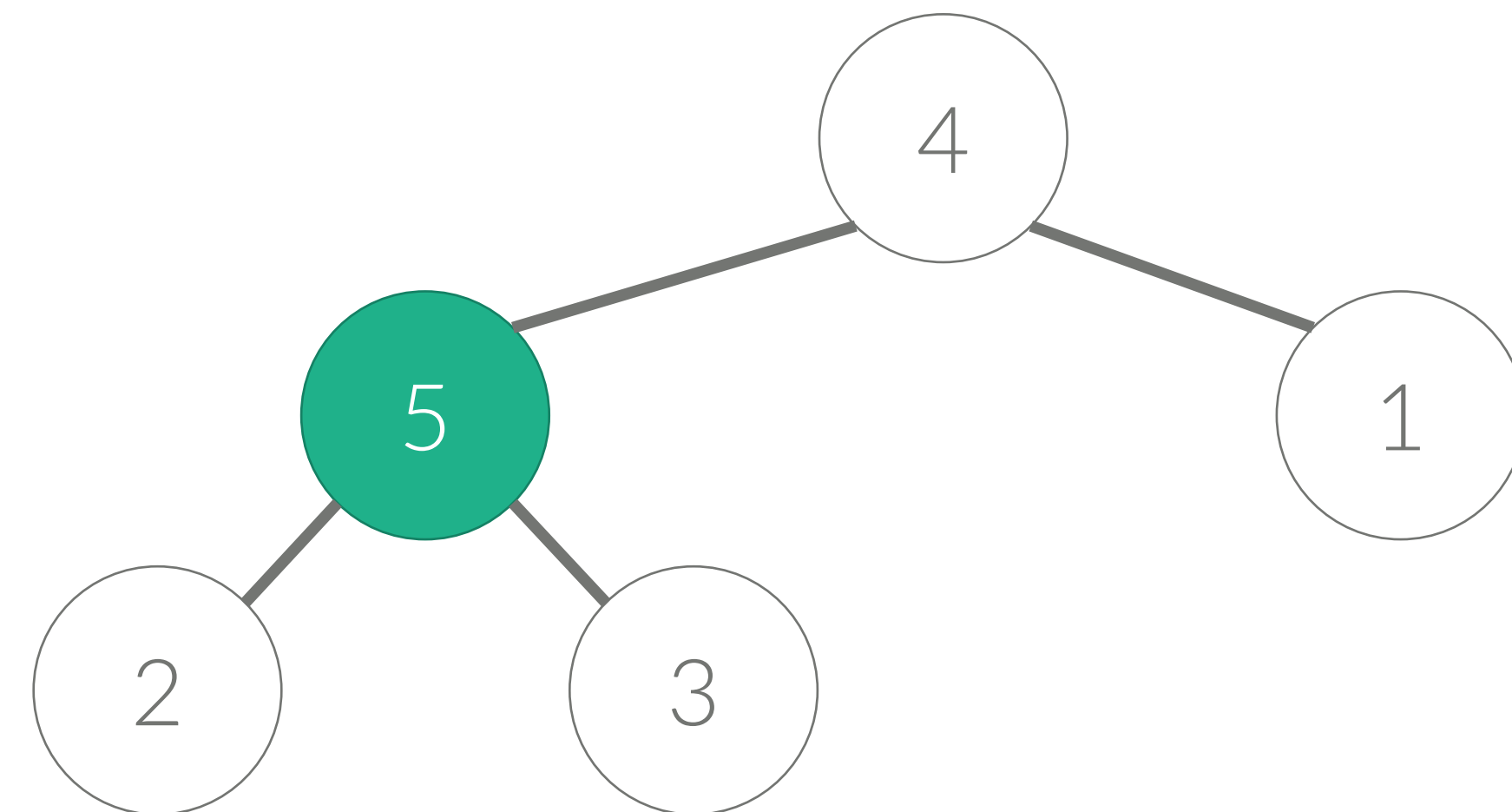


# 최대 힙 삽입

Max-Heap

88

- 5와 부모를 비교한다.
- 이 경우에  $3 < 5$  이기 때문에 최대 힙의 성질을 만족하지 않는다
- 따라서 두 수를 교환



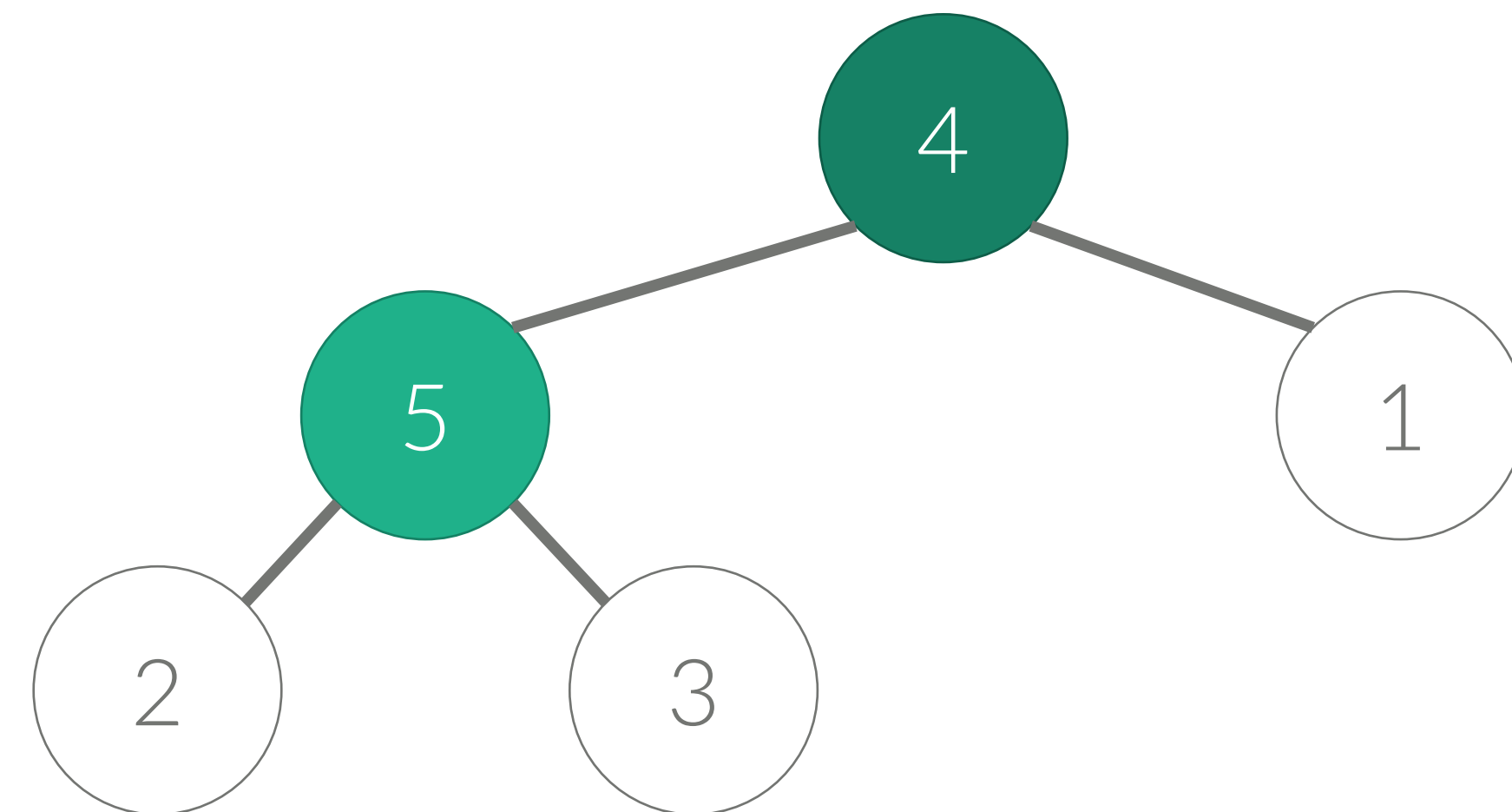


# 최대 힙 삽입

Max-Heap

89

- 5와 부모를 비교한다.
- 이 경우에  $4 < 5$  이기 때문에 최대 힙의 성질을 만족하지 않는다
- 따라서 두 수를 교환

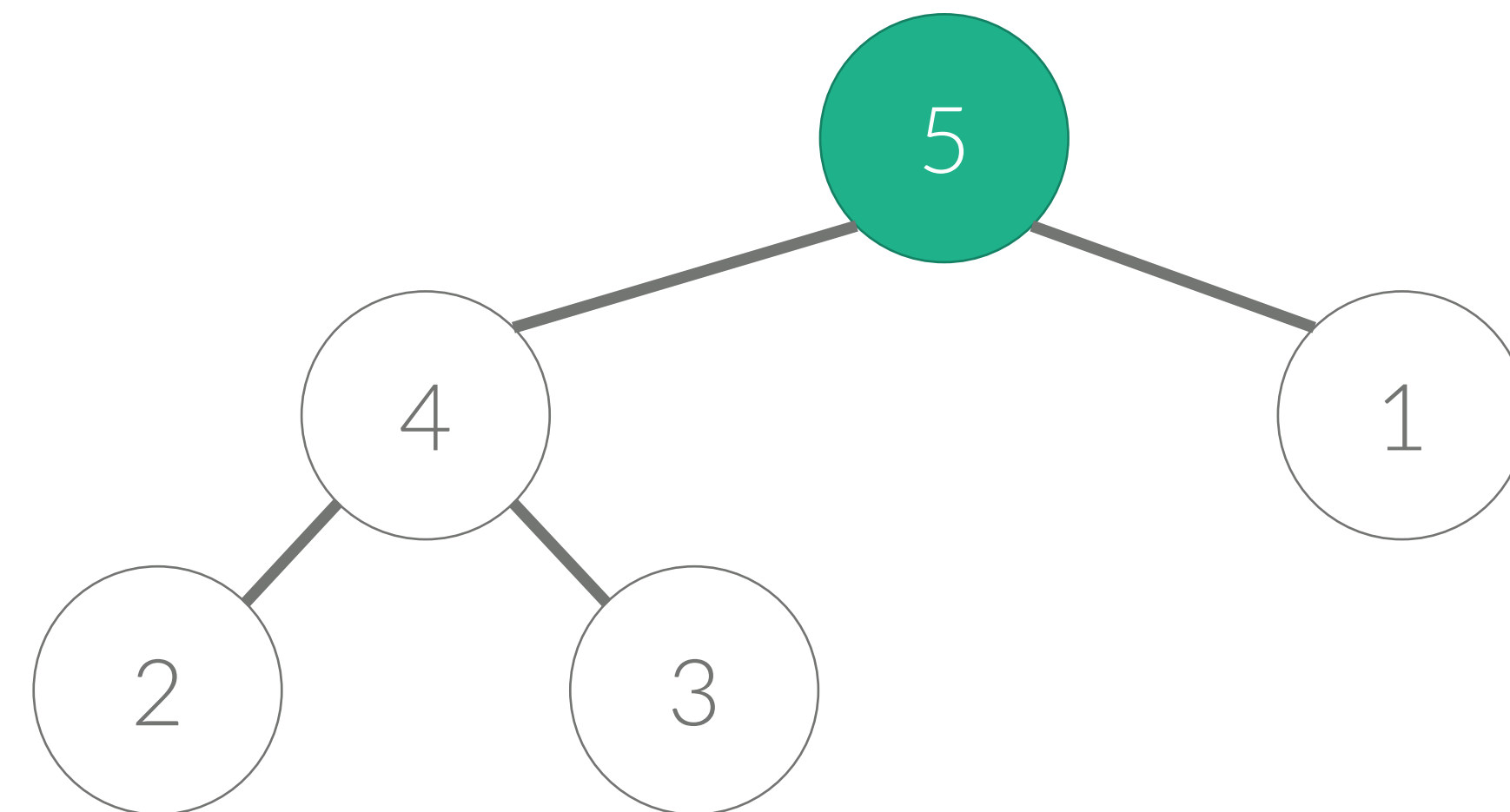


# 최대 힙 삽입

Max-Heap

90

- 5와 부모를 비교한다.
- 이 경우에  $4 < 5$  이기 때문에 최대 힙의 성질을 만족하지 않는다
- 따라서 두 수를 교환

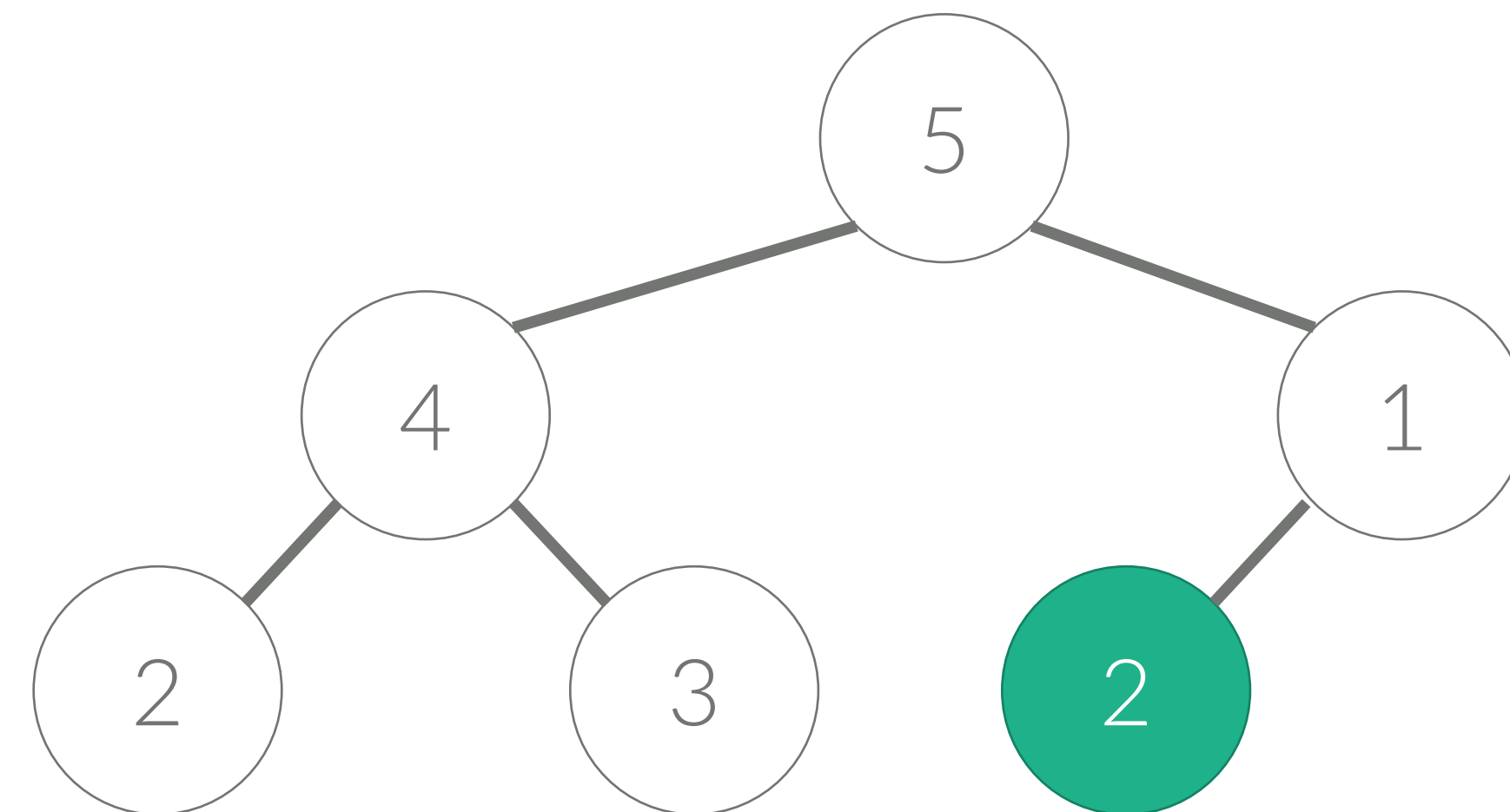


# 최대 힙 삽입

Max-Heap

- 2를 넣는다

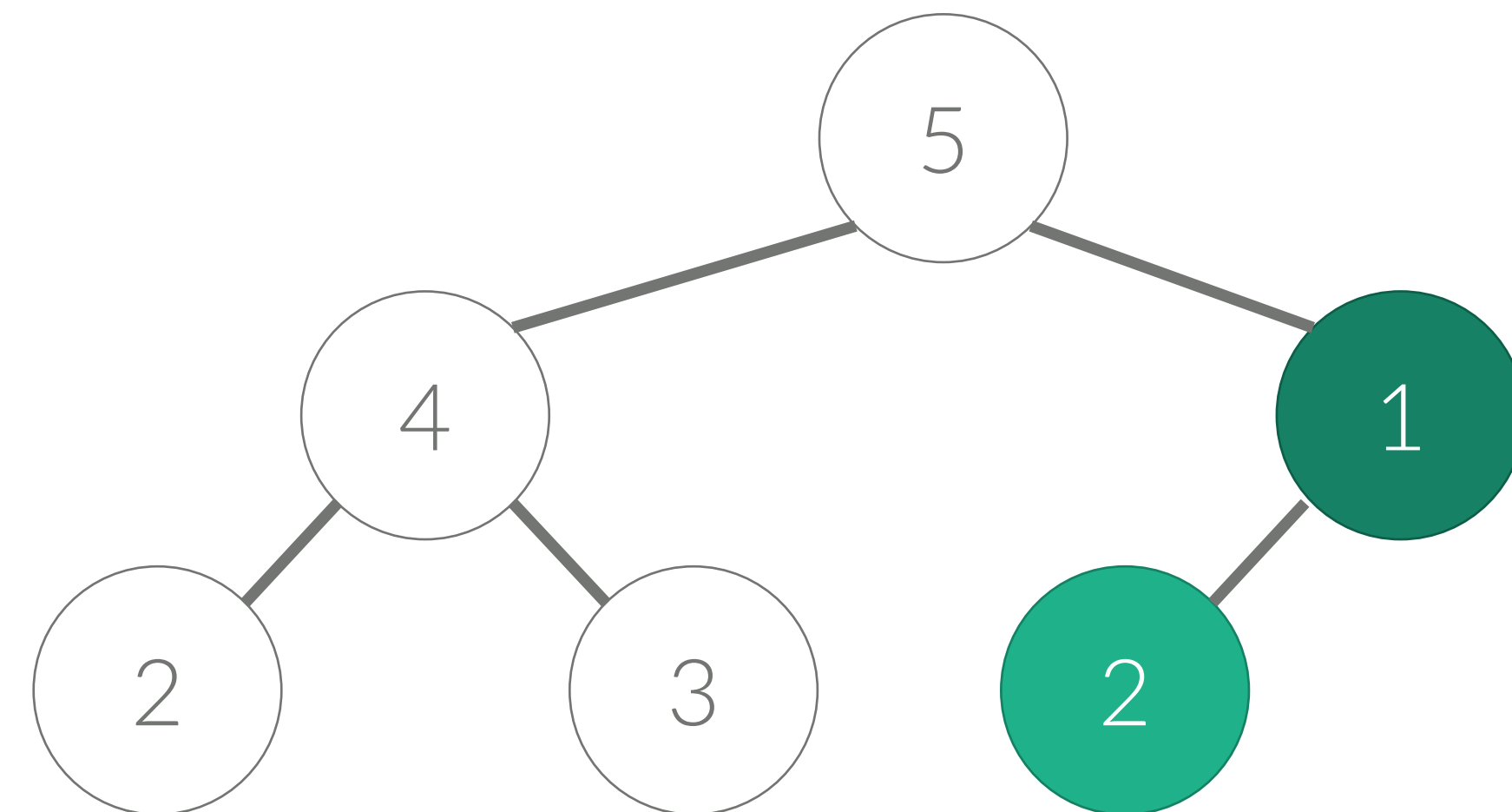
91



# 최대 힙 삽입

Max-Heap

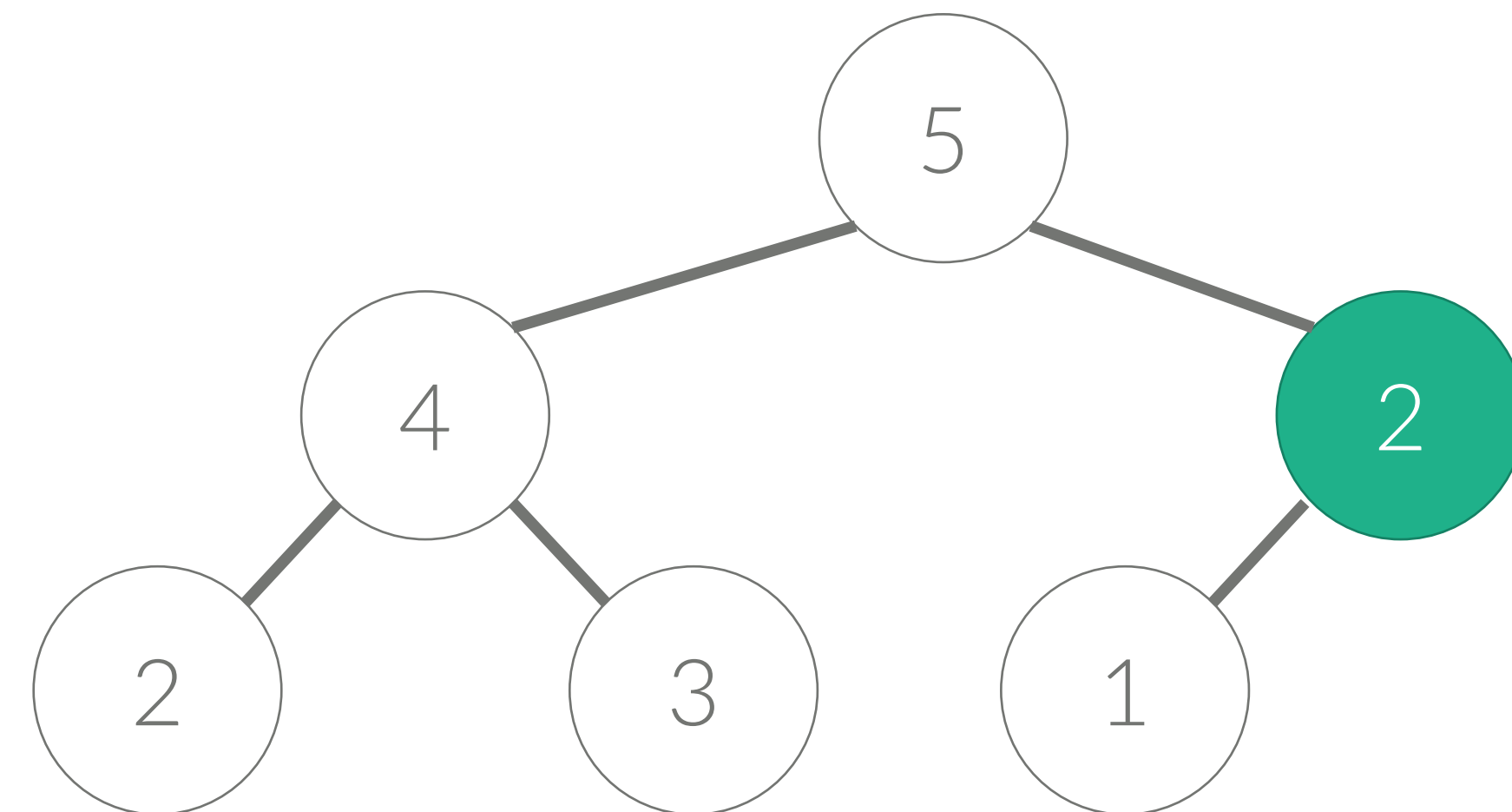
- 2와 2의 부모인 1을 비교
- $1 < 2$  이기 때문에 교환



# 최대 힙 삽입

Max-Heap

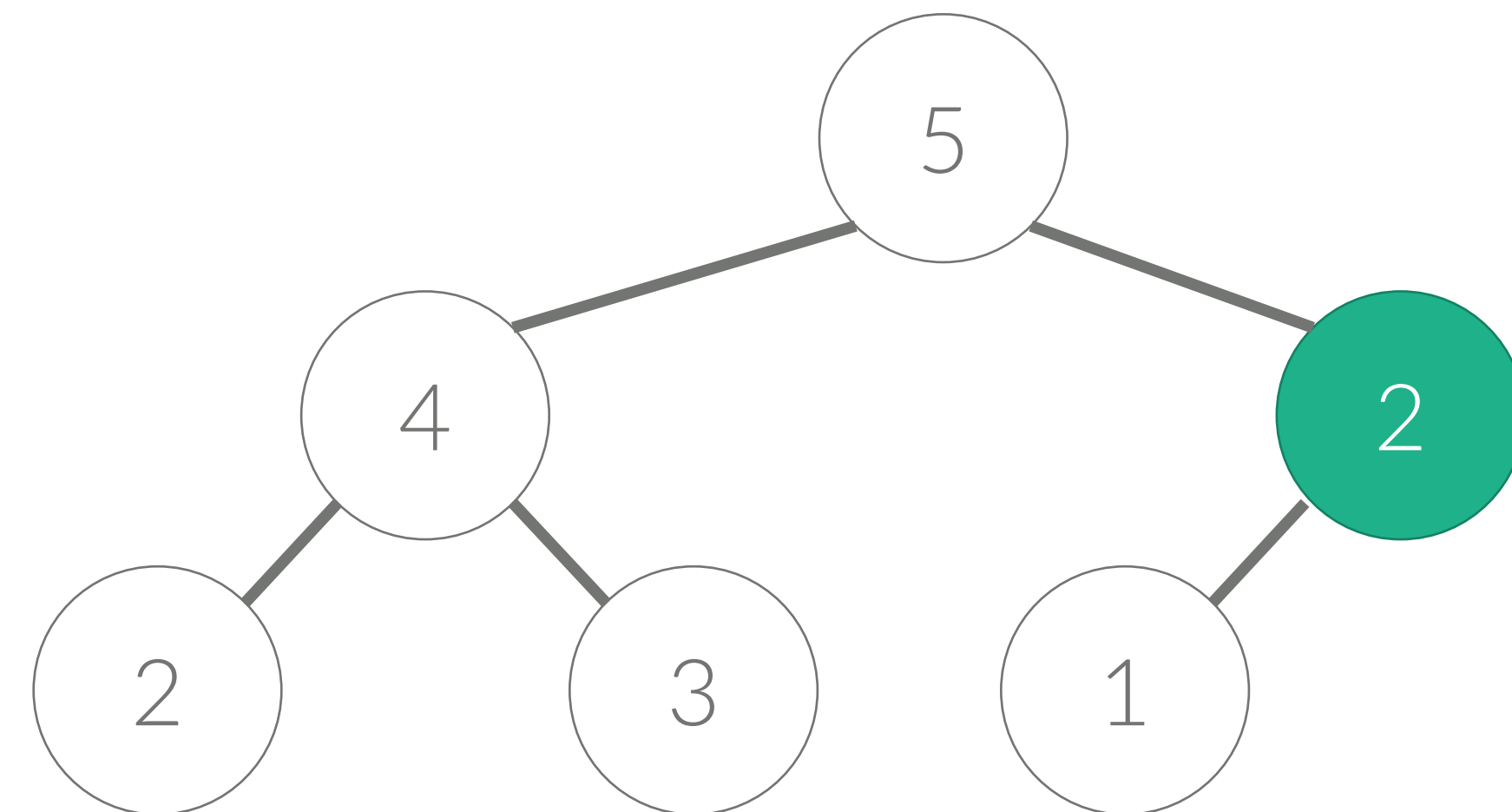
- 2와 2의 부모인 1을 비교
- $1 < 2$  이기 때문에 교환



# 최대 힙 삽입

Max-Heap

- 2와 2의 부모인 5을 비교
- $5 > 1$  이기 때문에 여기서 종료

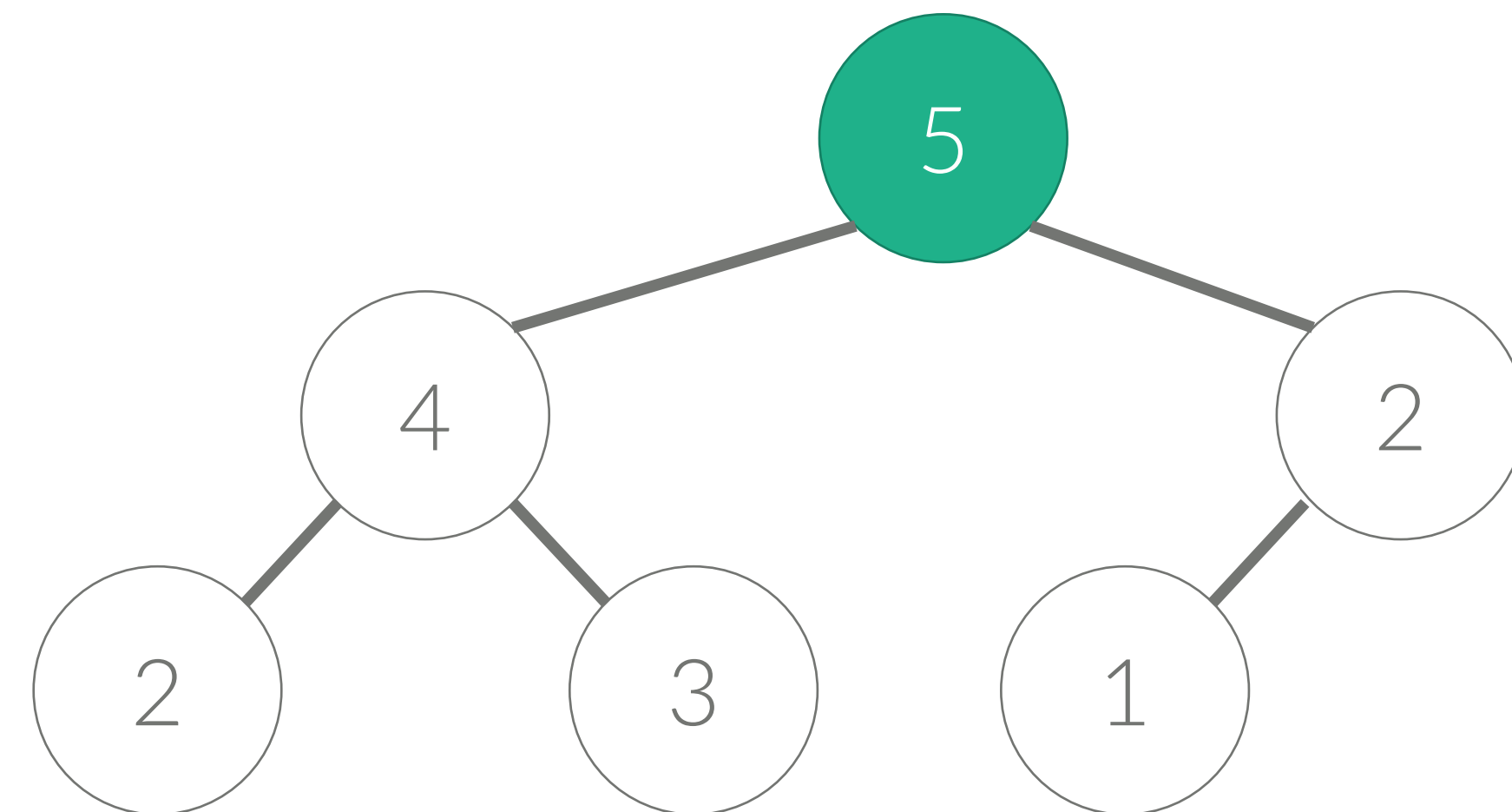


# 최대 힙 제거

Max-Heap

95

- 루트를 가장 마지막에 있는 값으로 바꿈

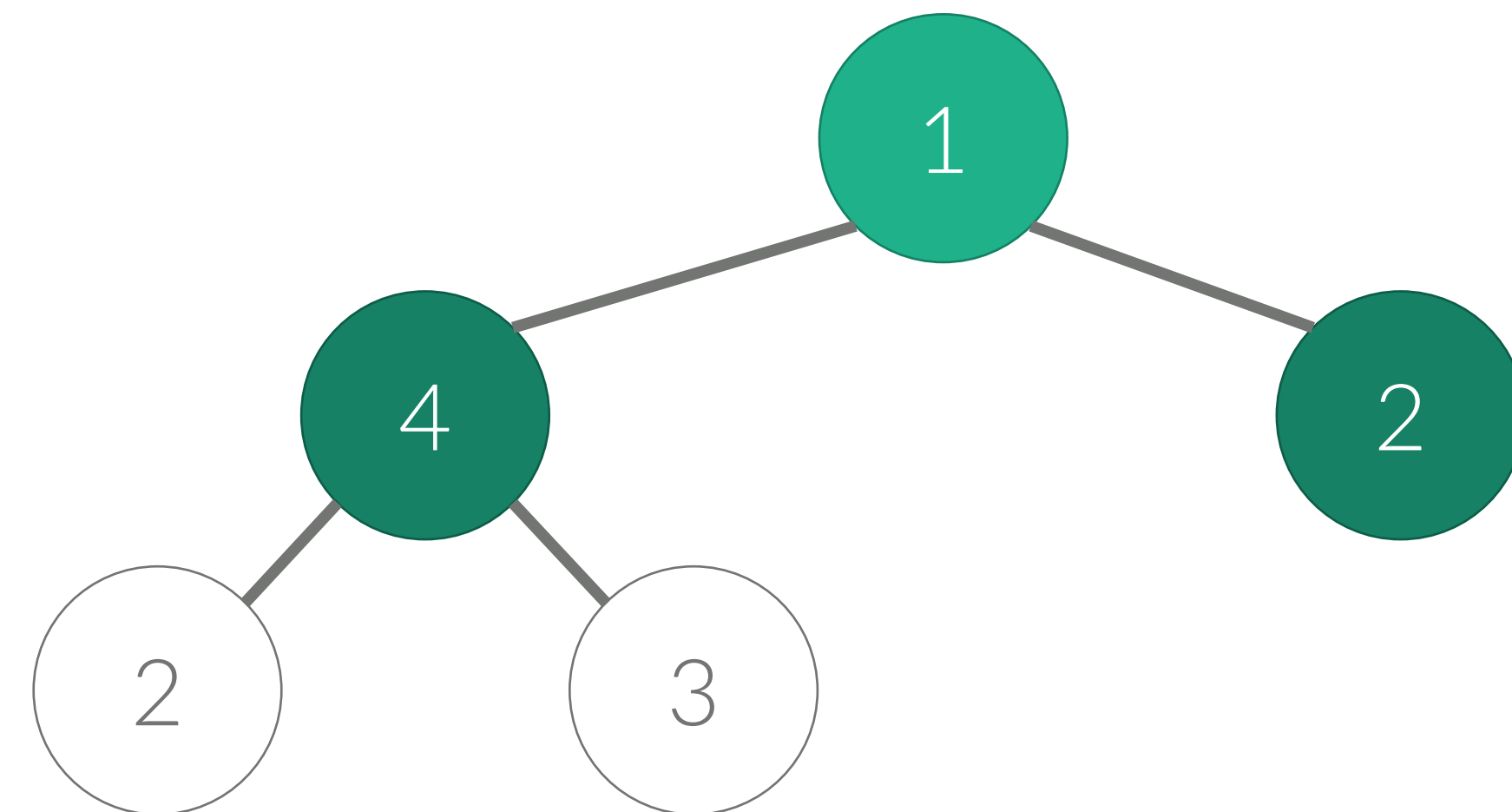


# 최대 힙 제거

Max-Heap

96

- 자식과 비교하면서 아래로 내려감
- 최대 힙이기 때문에
- 루트 > 자식 을 만족하려면
- 4와 1을 바꿔야 함



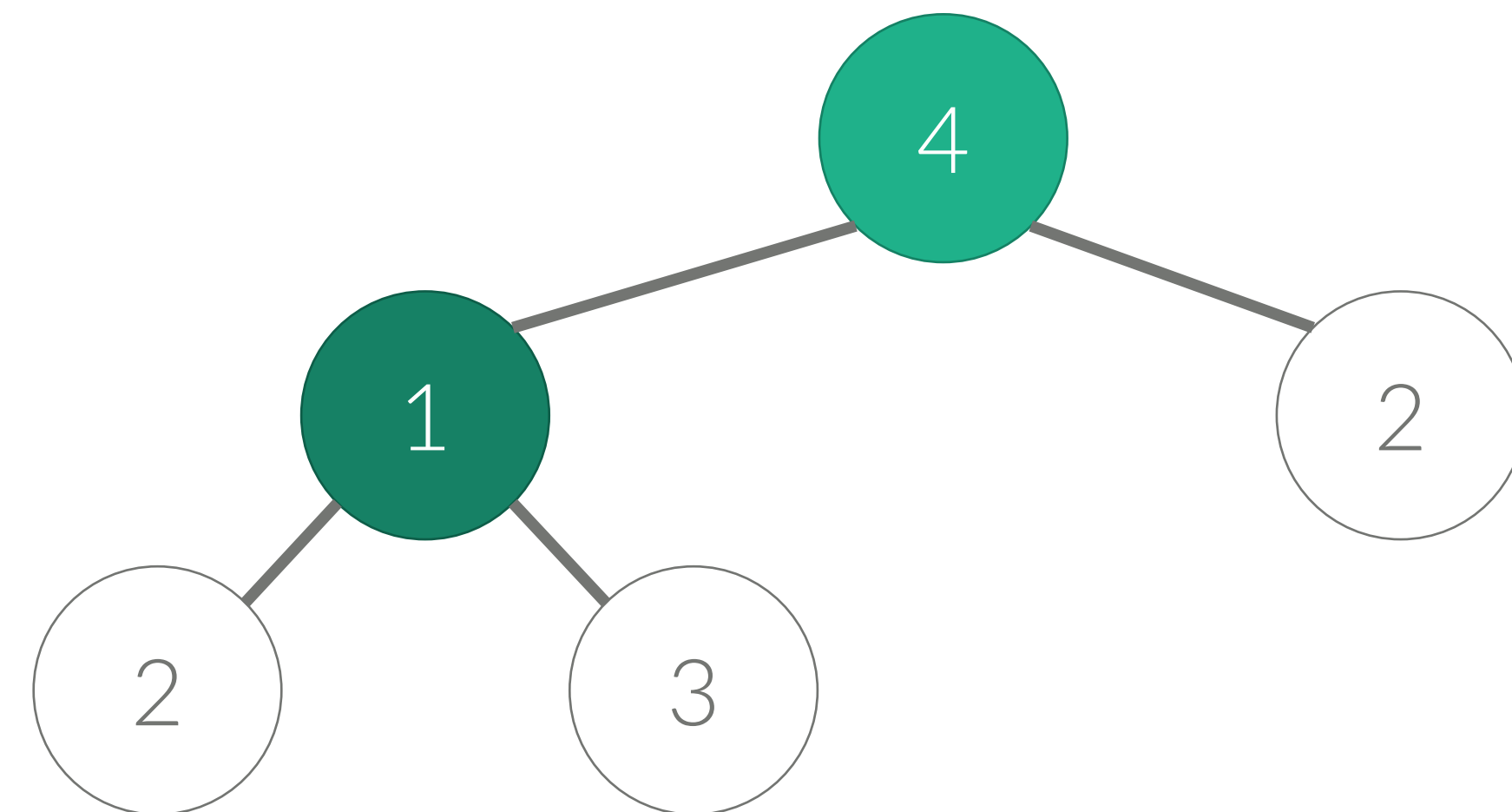


# 최대 힙 제거

Max-Heap

97

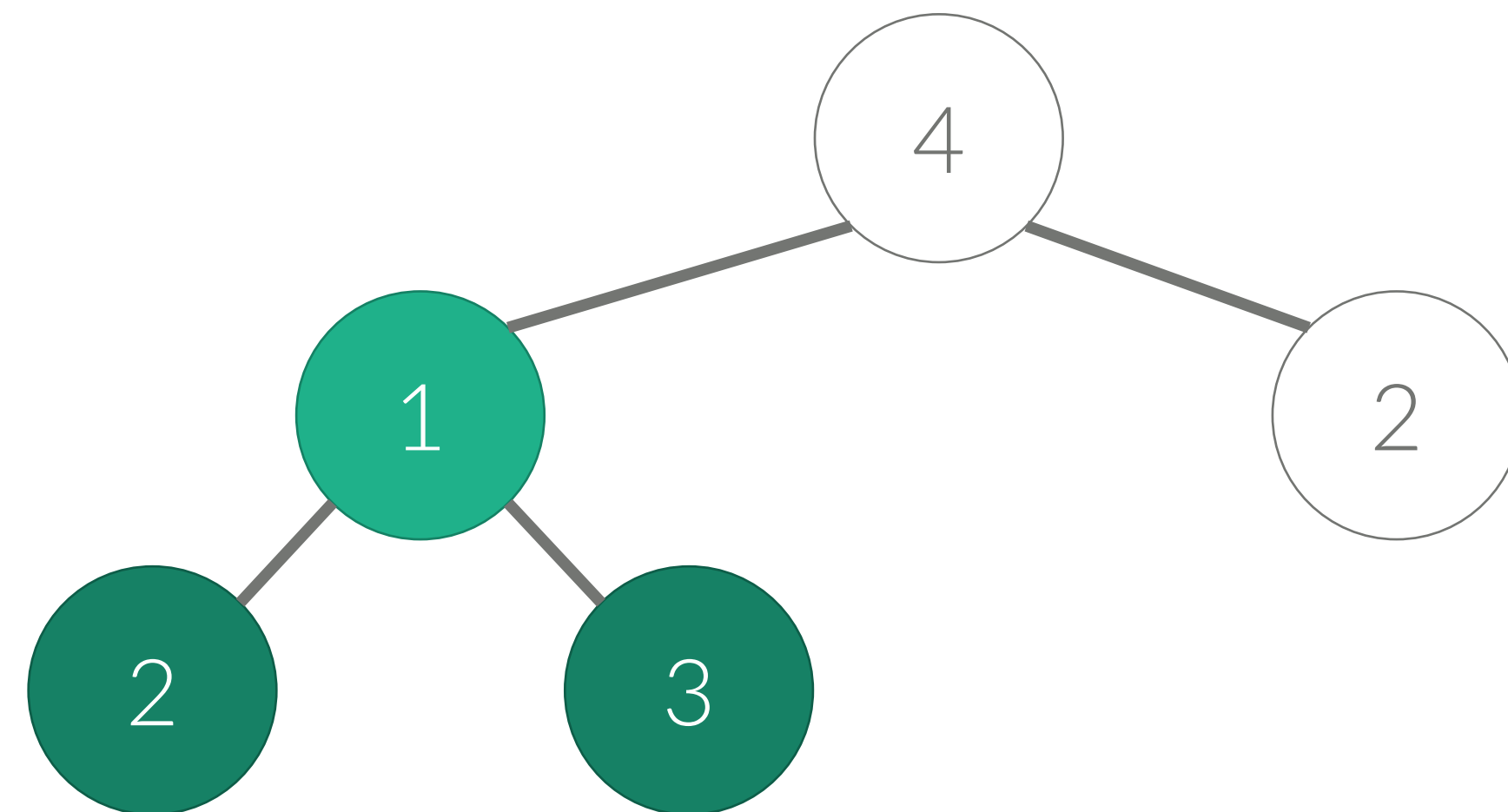
- 자식과 비교하면서 아래로 내려감
- 최대 힙이기 때문에
- 루트 > 자식 을 만족하려면
- 4와 1을 바꿔야 함



# 최대 힙 제거

Max-Heap

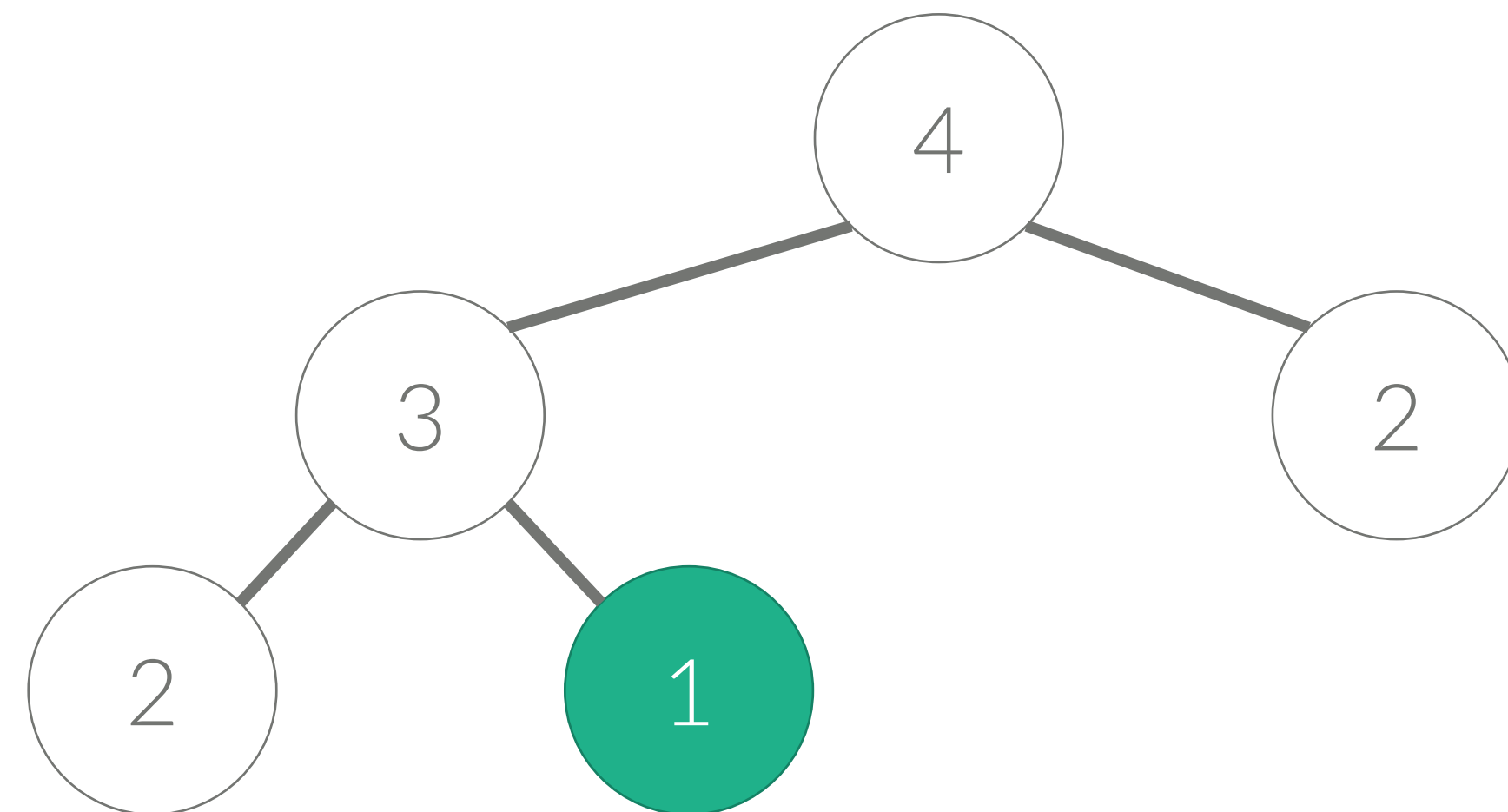
- 자식과 비교하면서 아래로 내려감



# 최대 힙 제거

Max-Heap

- 자식과 비교하면서 아래로 내려감

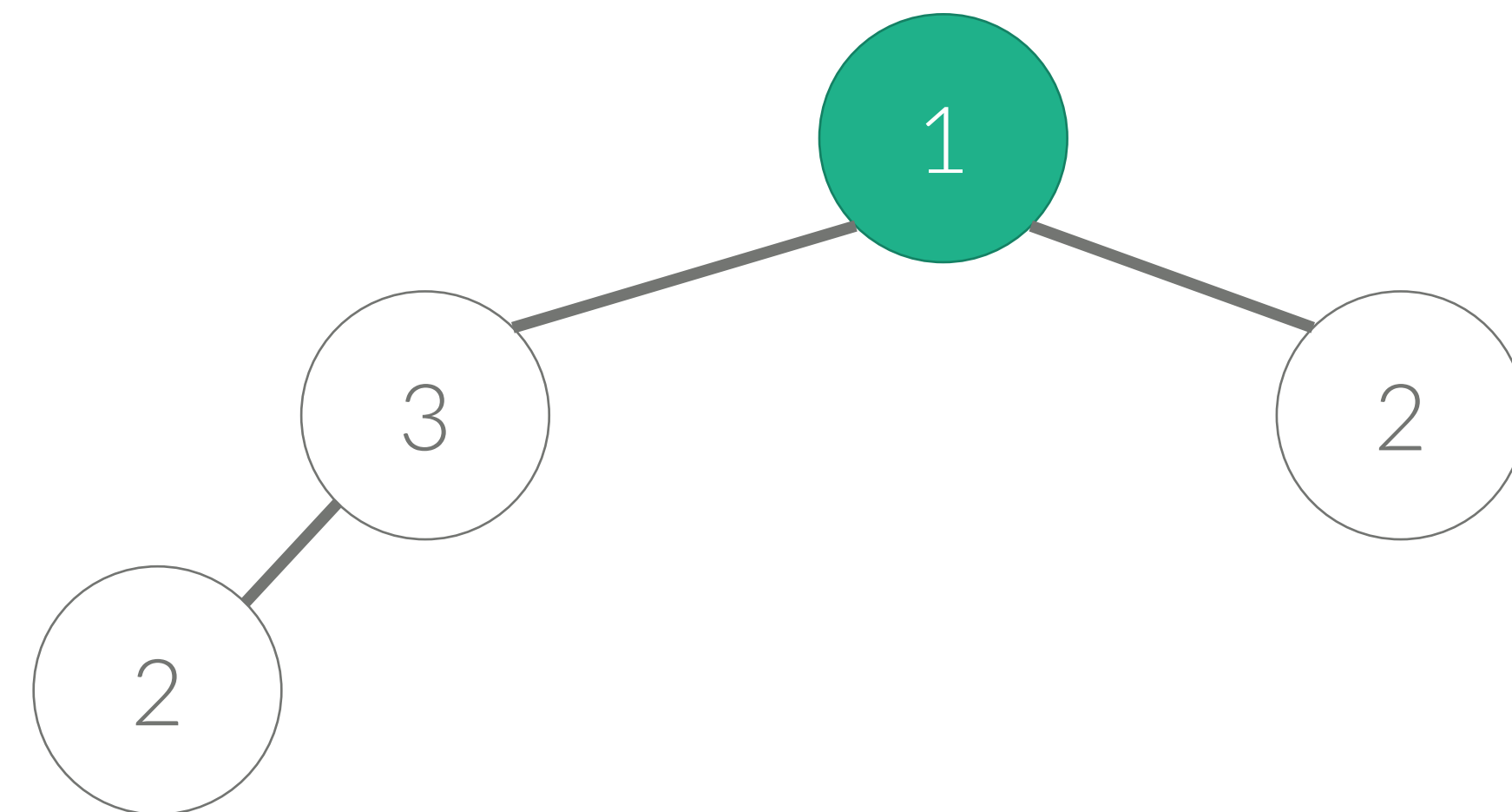


# 최대 힙 제거

Max-Heap

- 한 번 더 제거

100

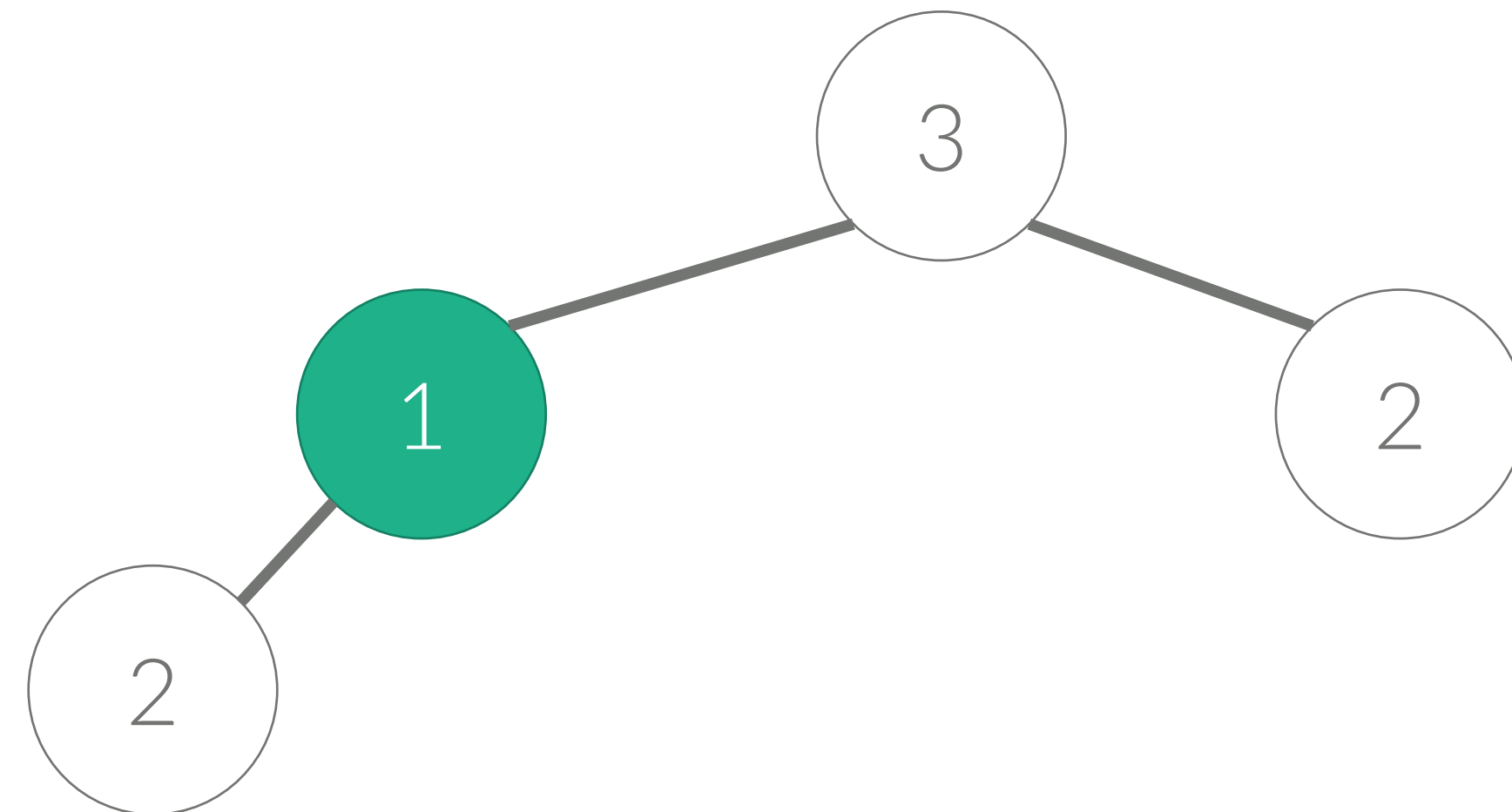


# 최대 힙 제거

Max-Heap

101

- 자식과 비교하면서 아래로 내려감

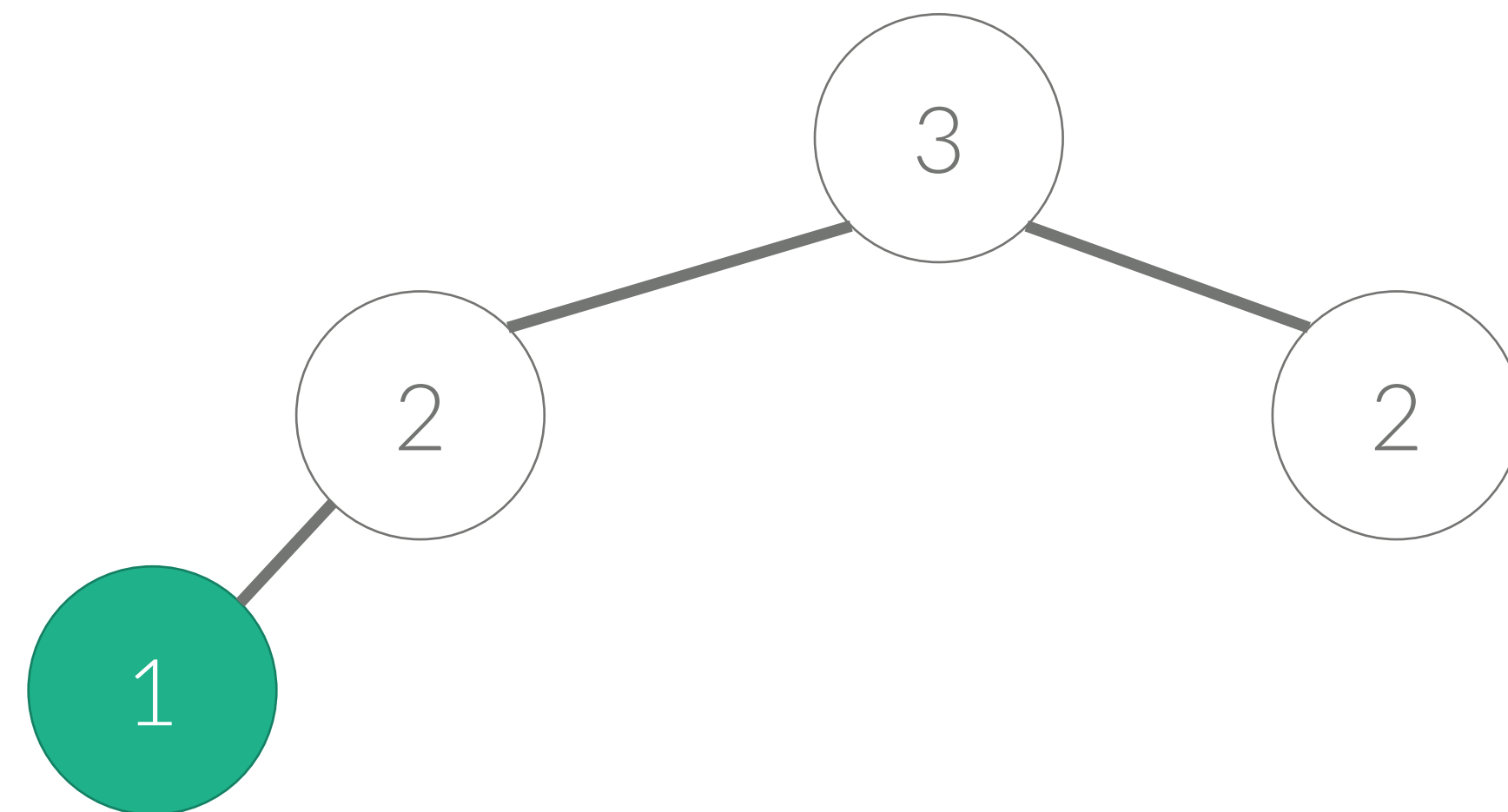


# 최대 힙 제거

Max-Heap

- 자식과 비교하면서 아래로 내려감

102



# 힙

## Heap

103

- C++은 `priority_queue`, Java는 `PriorityQueue`를 사용할 수 있다.
- C++은 최대 힙, Java는 최소 힙이다.
- C++의 경우에 최소 힙을 사용하려면 다음 둘 중 하나를 이용하면 된다.
  - `priority_queue<int, vector<int>, greater<int>> q;`
  - 모든 값에 -1을 곱해서 넣고, 뺀 후에는 -1을 곱해서 사용한다. 이 방법은 힙에 들어갈 수 있는 것이 자료형의 전체 범위일 때는 사용할 수 없다.
- Java의 경우에는 `Compator`를 구현하거나, -1을 곱하는 방법을 사용한다.

# 최대 힙

<https://www.acmicpc.net/problem/11279>

- 직접 구현 소스: <http://codeplus.codes/d0cd13af92044c8b8e3f85d829a1f01f>
- 라이브러리 사용 소스: <http://codeplus.codes/227bbccb3d4341449c0b532342de53d1>



# 최소 힙

105

<https://www.acmicpc.net/problem/1927>

- 소스: <http://codeplus.codes/bb84c82656094622979c13f81785987a>

# 가운데를 말해요

106

<https://www.acmicpc.net/problem/1655>

- N개의 수가 주어졌을 때
- 각각의 수가 주어졌을 때 마다 지금까지 나온 수의 중간값을 구하는 문제
- $1 \leq N \leq 100,000$

# 가운데를 말해요

<https://www.acmicpc.net/problem/1655>

- 수가 주어질 때마다 정렬하고 중간값을 구하면 시간 복잡도는  $O(N^2 \lg N)$  이다.
- 수를 링크드 리스트에 넣으면 매번 정렬할 필요가 없기 때문에  $O(N^2)$  이다.
- 모두 시간이 오래 걸린다.

# 가운데를 말해요

108

<https://www.acmicpc.net/problem/1655>

- 수를  $N/2$ ,  $N/2$ 개로 나눠서 왼쪽과 오른쪽으로 나눠서 쪼갬다.
- 왼쪽: 최대 힙
- 오른쪽: 최소 힙

# 가운데를 말해요

109

<https://www.acmicpc.net/problem/1655>

- 수를  $N/2$ ,  $N/2$ 개로 나눠서 왼쪽과 오른쪽으로 나눠서 쪼갬.
- 왼쪽: 최대 힙
- 오른쪽: 최소 힙
- 항상 왼쪽과 오른쪽 크기의 차이를 1보다 작거나 같게 만든다
- 짝수: 0, 홀수: 1 (왼쪽이 더 큼)

# 가운데를 말해요

110

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

$L =$ 

5
---

$R =$

# 가운데를 말해요

111

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L = 

3	5
---	---

R =

# 가운데를 말해요

112

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L = 

3	5
---	---

R =

L = 

3
---

R = 

5
---



# 가운데를 말해요

113

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L = 

3	4
---	---

R = 

5
---

# 가운데를 말해요

114

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L = 

2	3	4
---	---	---

R = 

5
---

# 가운데를 말해요

115

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L = 

2	3	4
---	---	---

R = 

5
---

L = 

2	3
---	---

R = 

4	5
---	---

# 가운데를 말해요

116

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L = 

2	3
---	---

R = 

4	5	6
---	---	---

# 가운데를 말해요

117

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L = 

2	3
---	---

L = 

2	3	4
---	---	---

R = 

4	5	6
---	---	---

R = 

5	6
---	---

# 가운데를 말해요

118

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L =

1	2	3	4
---	---	---	---

R =

5	6
---	---

# 가운데를 말해요

119

<https://www.acmicpc.net/problem/1655>

- $A = [5, 3, 4, 2, 6, 1]$

L = 

1	2	3	4
---	---	---	---

R = 

5	6
---	---

L = 

1	2	3
---	---	---

R = 

4	5	6
---	---	---

# 가운데를 말해요

120

<https://www.acmicpc.net/problem/1655>

- 소스: <http://codeplus.codes/95316a9075df4d50ab6da93241e5c923>



# 이진 검색 트리

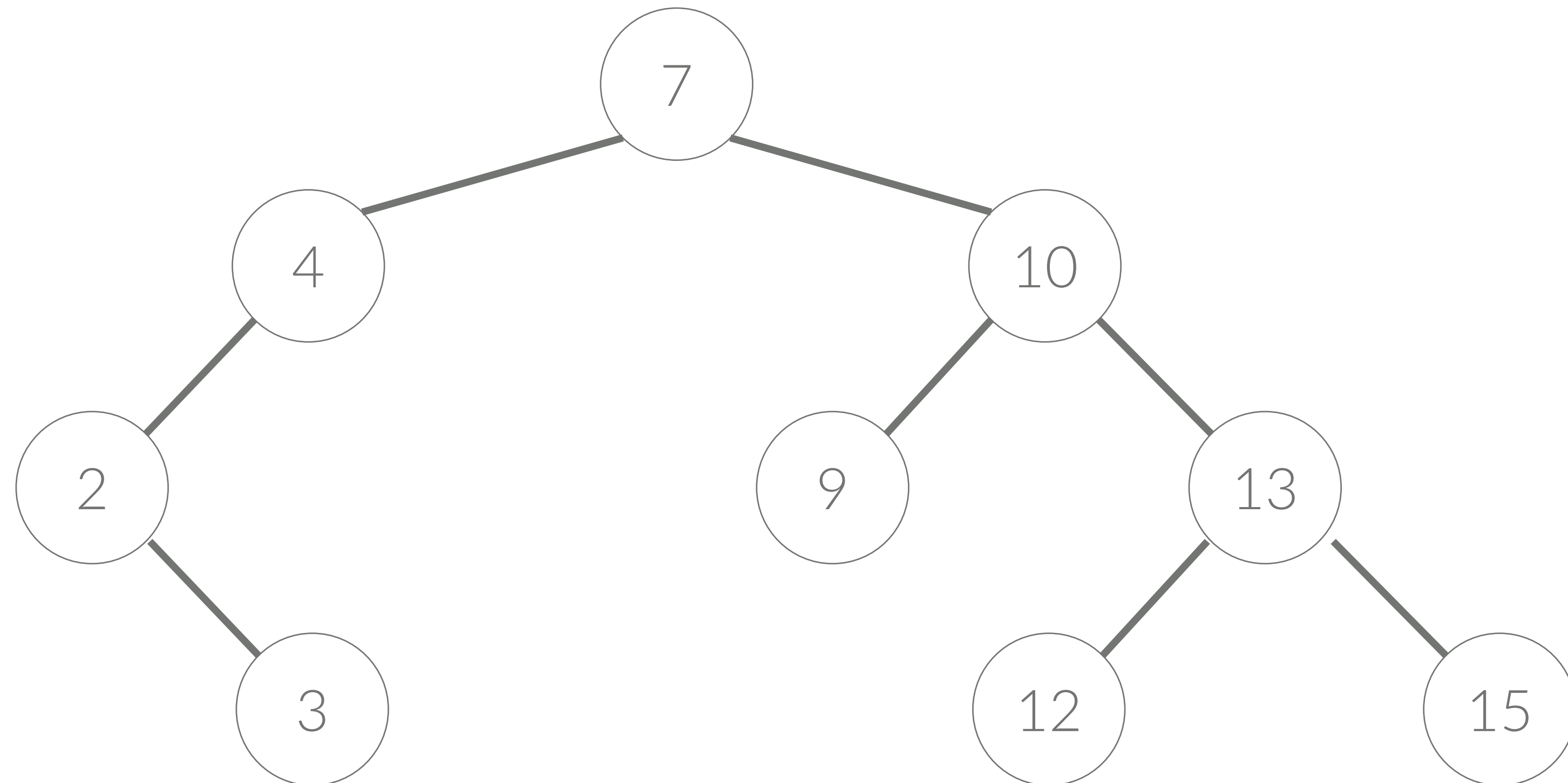
---

# 이진 검색 트리

Binary Search Tree

122

- 이진 트리
- 현재 노드의 왼쪽 서브 트리에는 항상 현재 노드의 값보다 작은 값이 들어있고
- 현재 노드의 오른쪽 서브 트리에는 항상 현재 노드의 값보다 큰 값이 들어있다



# 이진 검색 트리

123

## Binary Search Tree

- 트리의 노드에는 자료를 저장할 변수(data), 왼쪽 자식을 가르키는 변수(left), 오른쪽 자식을 가르키는 변수(right)

```
struct Node {  
    int data;  
    Node *left;  
    Node *right;  
    Node(int data) {  
        this->data = data;    }  
        this->left = NULL;  
        this->right = NULL;  
    }  
};
```

```
struct BST {  
    Node *root;  
    BST() {  
        this->root = NULL;  
    }  
};
```

# 이진 검색 트리 삽입

## Binary Search Tree

- 가장 첫 노드를 추가하는 경우에는 root를 새로운 노드로 설정해주면 된다

```
Node *insert(Node *node, int data) {  
    if (node == NULL) {  
        return new Node(data);  
    }  
}
```

# 이진 검색 트리 삽입

125

Binary Search Tree

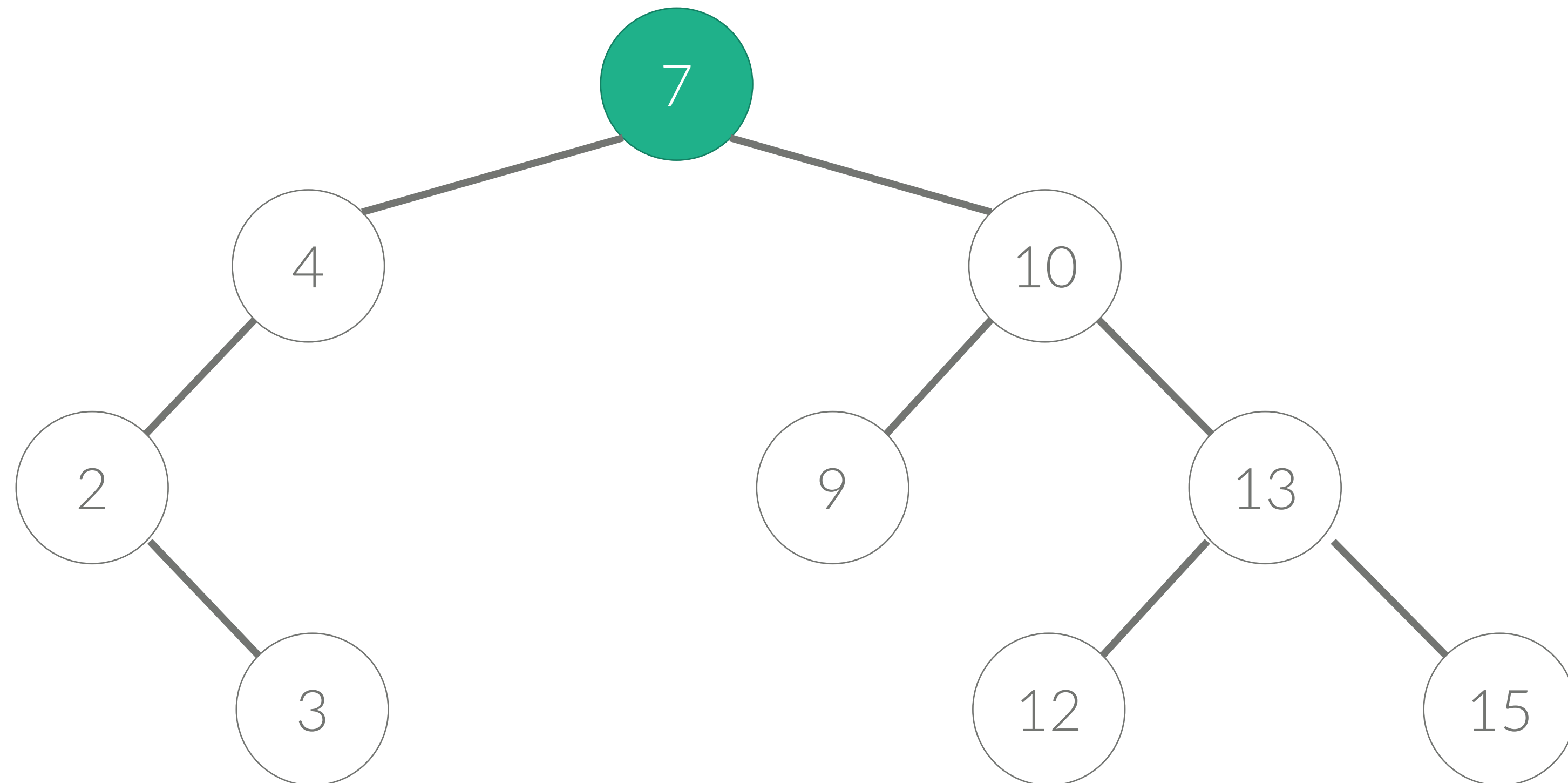
- 가장 첫 노드를 추가하는 경우에는 root를 새로운 노드로 설정해주면 된다
- 그 이외의 경우에는 노드에 포함된 값과 대소비교를 하면서 왼쪽 또는 오른쪽으로 이동하면 된다

# 이진 검색 트리 삽입

Binary Search Tree

126

- 8을 삽입하는 경우

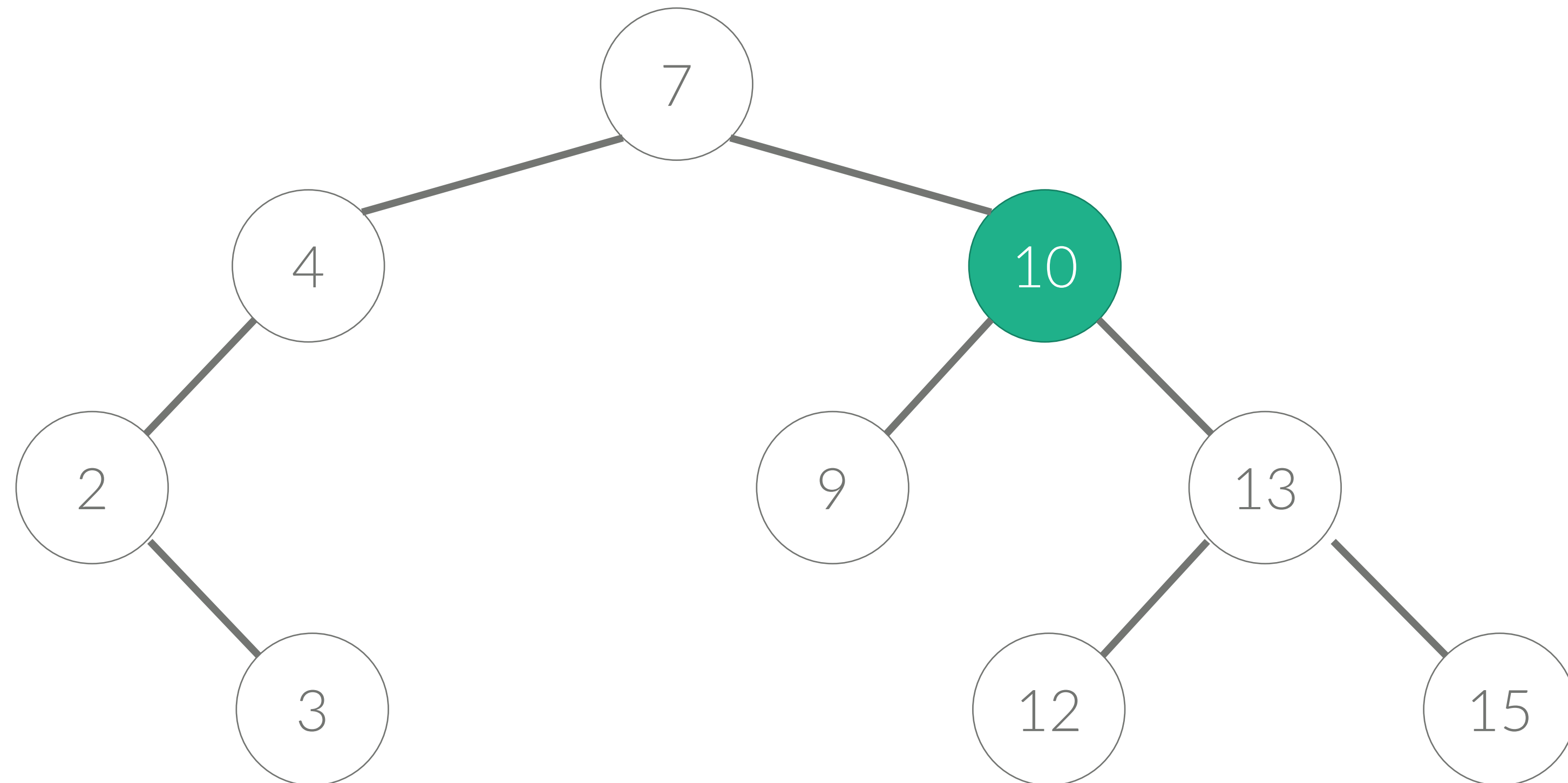


# 이진 검색 트리 삽입

127

Binary Search Tree

- 8을 삽입하는 경우

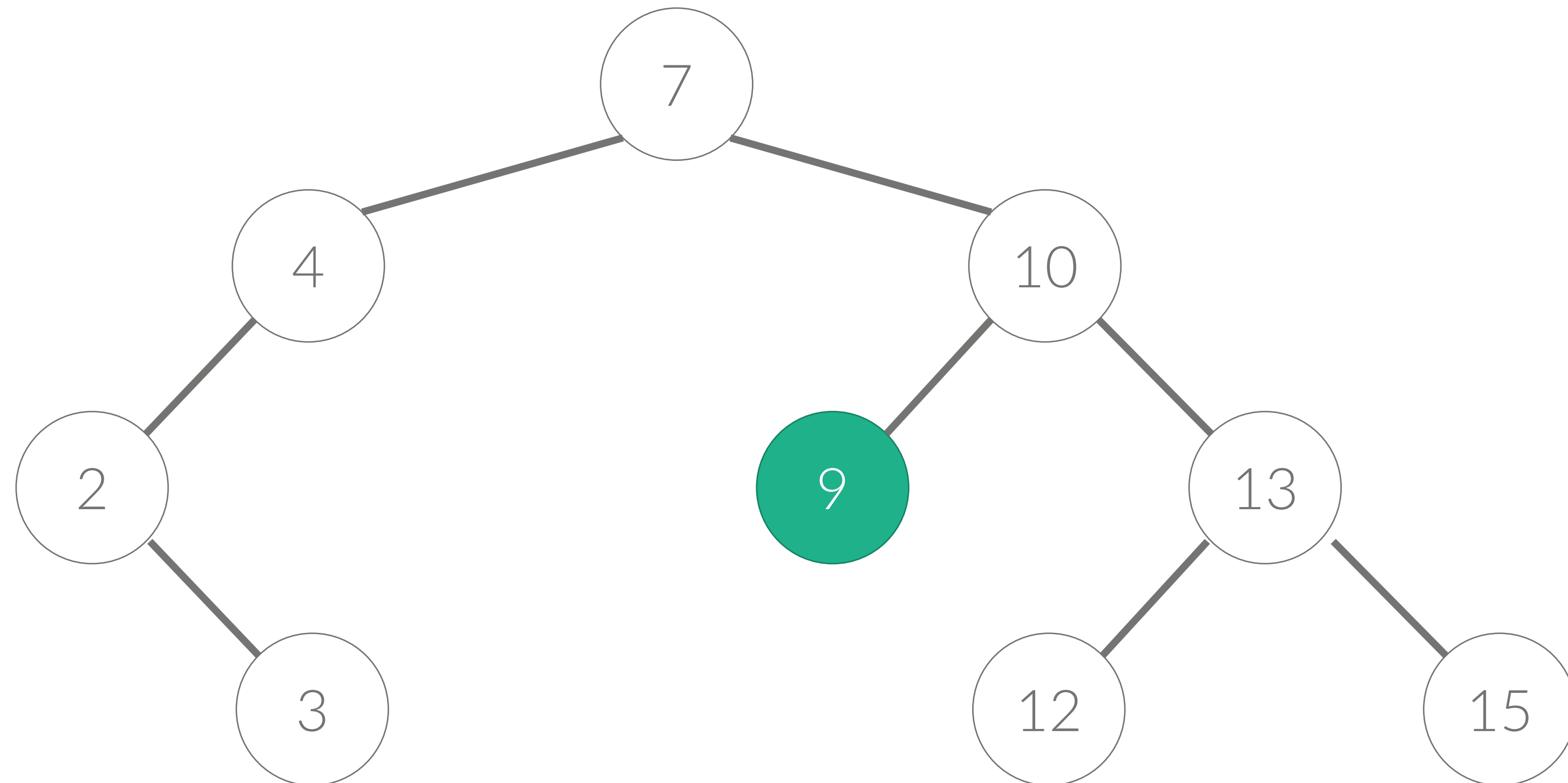


# 이진 검색 트리 삽입

Binary Search Tree

128

- 8을 삽입하는 경우



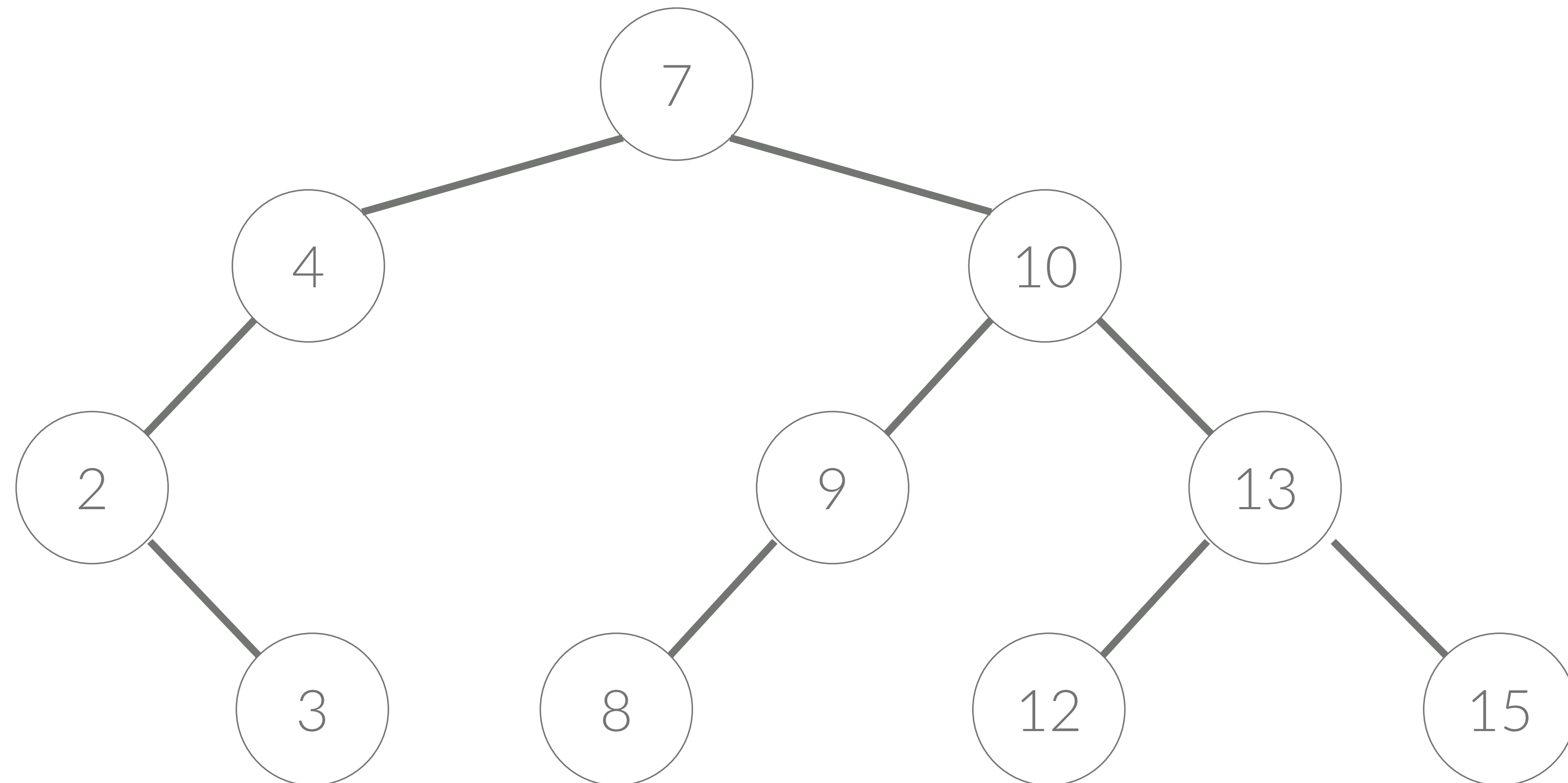


# 이진 검색 트리 삽입

129

Binary Search Tree

- 8을 삽입하는 경우



# 이진 검색 트리 삽입

Binary Search Tree

```
Node *p = this->root;
while (true) {
    if (p->data < node->data) {
        if (p->right == NULL) {
            p->right = node;
            break;
        } else p = p->right;
    } else {
        if (p->left == NULL) {
            p->left = node;
            break;
        } else p = p->left;
    }
}
```

# 이진 검색 트리 삽입

## Binary Search Tree

- 재귀로 구현할 수 있다

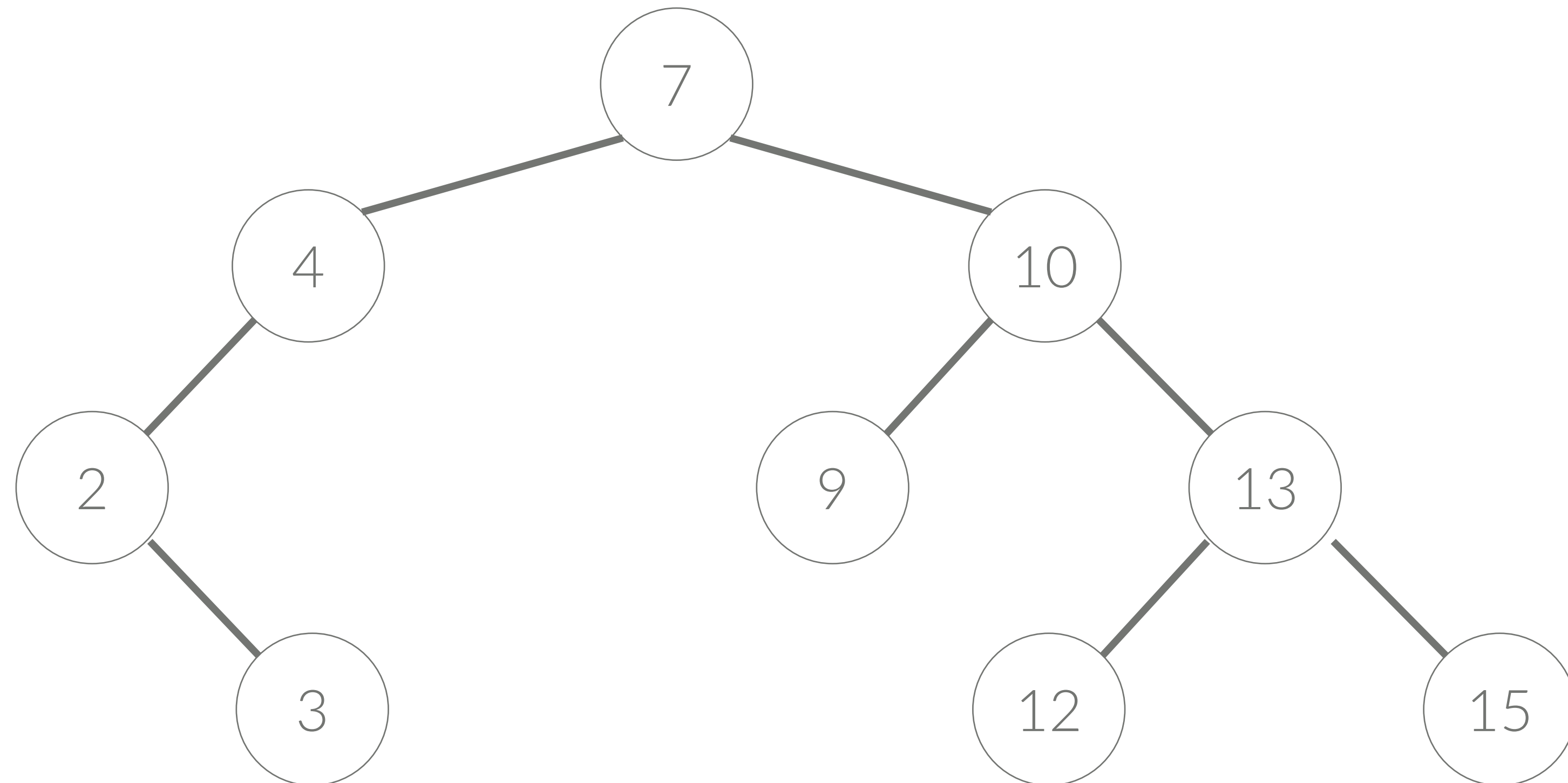
```
Node *insert(Node *node, int data) {  
    if (node == NULL) return new Node(data);  
    if (data < node->data) {  
        node->left = insert(node->left, data);  
    } else {  
        node->right = insert(node->right, data);  
    }  
    return node;  
}  
  
void insert(int data) {  
    this->root = insert(this->root, data);  
}
```

# 이진 검색 트리

Binary Search Tree

132

- 트리를 인오더 순회를 하면, BST에 저장된 값을 오름차순으로 구할 수 있다



# 이진 검색 트리

## Binary Search Tree

- 트리를 인오더 순회를 하면, BST에 저장된 값을 오름차순으로 구할 수 있다

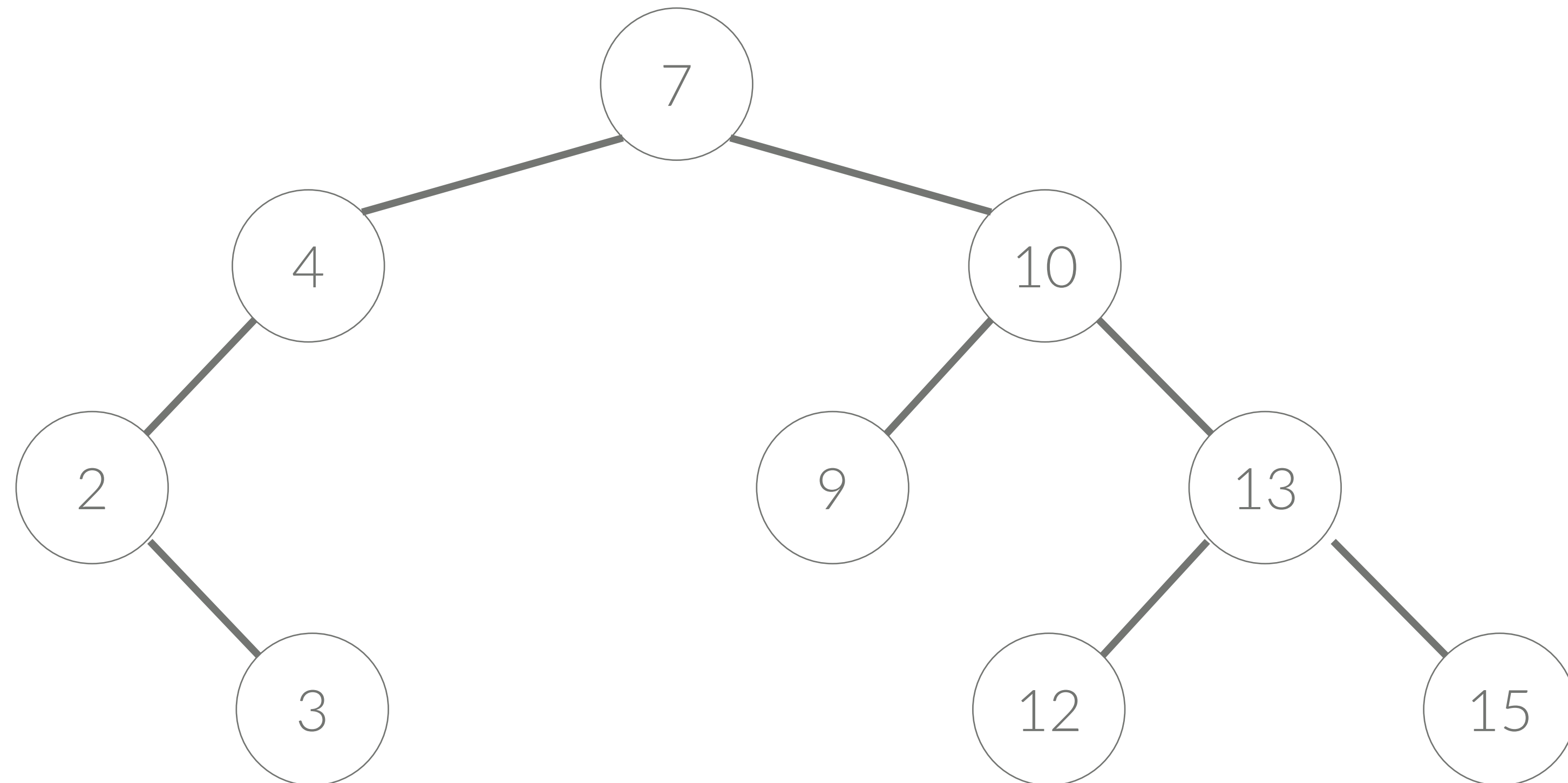
```
void inorder(Node *node) {  
    if (node == NULL) return;  
    inorder(node->left);  
    cout << node->data << ' ';  
    inorder(node->right);  
}  
  
void inorder() {  
    inorder(this->root);  
    cout << '\n';  
}
```

# 이진 검색 트리 삭제

134

## Binary Search Tree

- 삭제의 경우에는 총 3가지 경우의 수가 있다
- 1. 자식이 0개인 경우
- 2. 자식이 1개인 경우
- 3. 자식이 2개인 경우

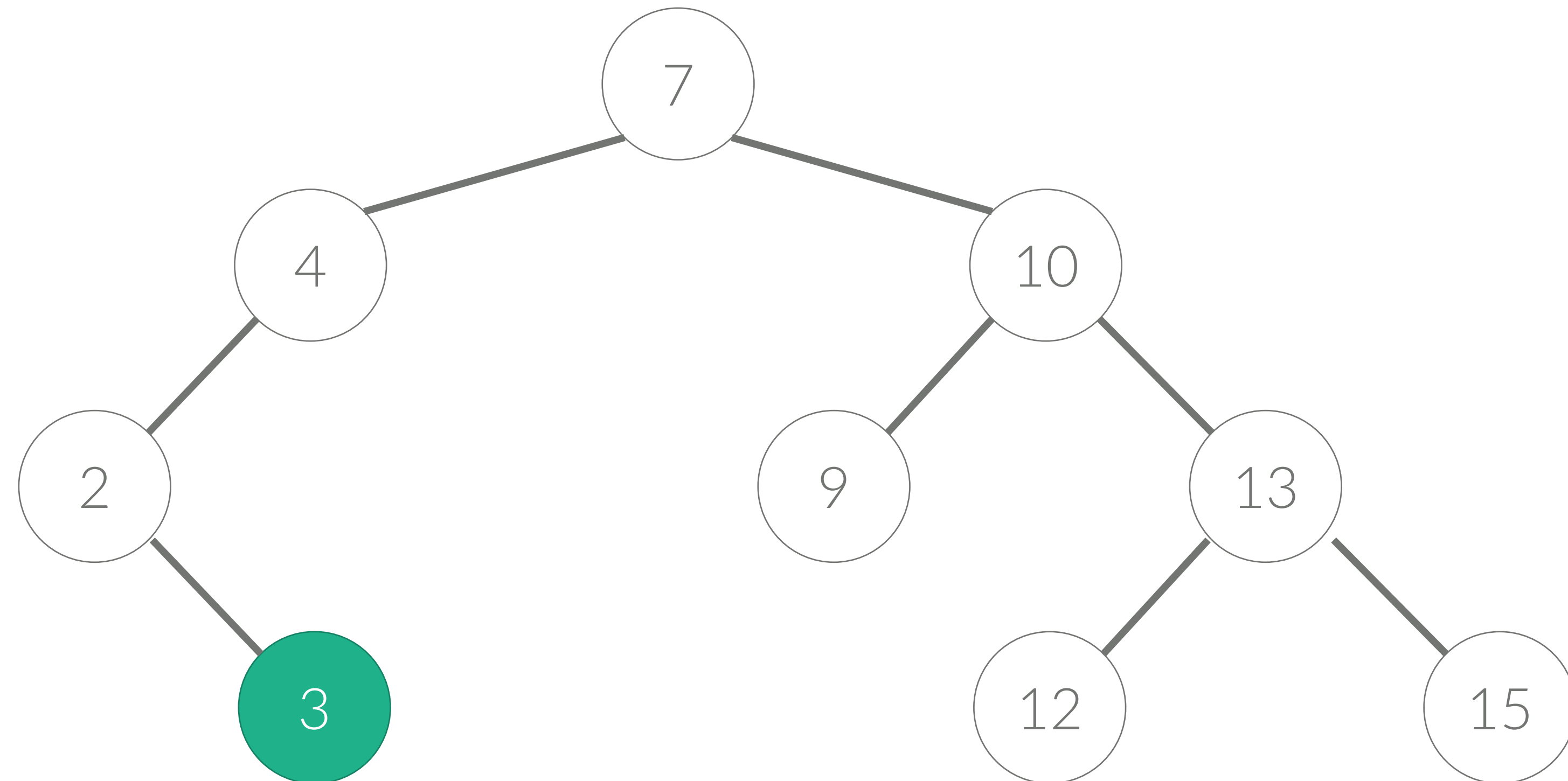


# 이진 검색 트리 삭제

Binary Search Tree

- 자식이 0개인 경우
- 그냥 지우면 된다

135

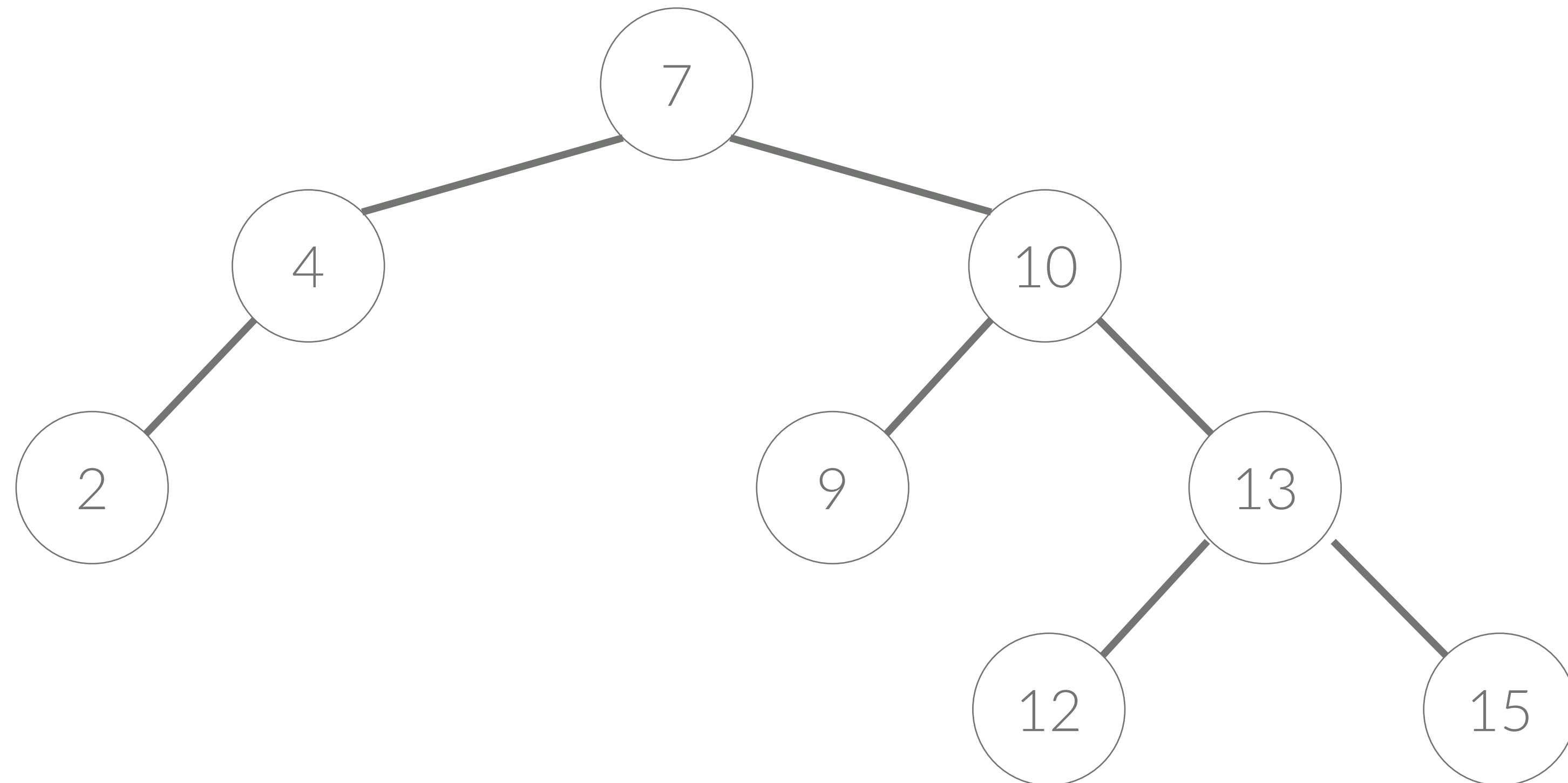


# 이진 검색 트리 삭제

136

Binary Search Tree

- 자식이 0개인 경우
- 그냥 지우면 된다



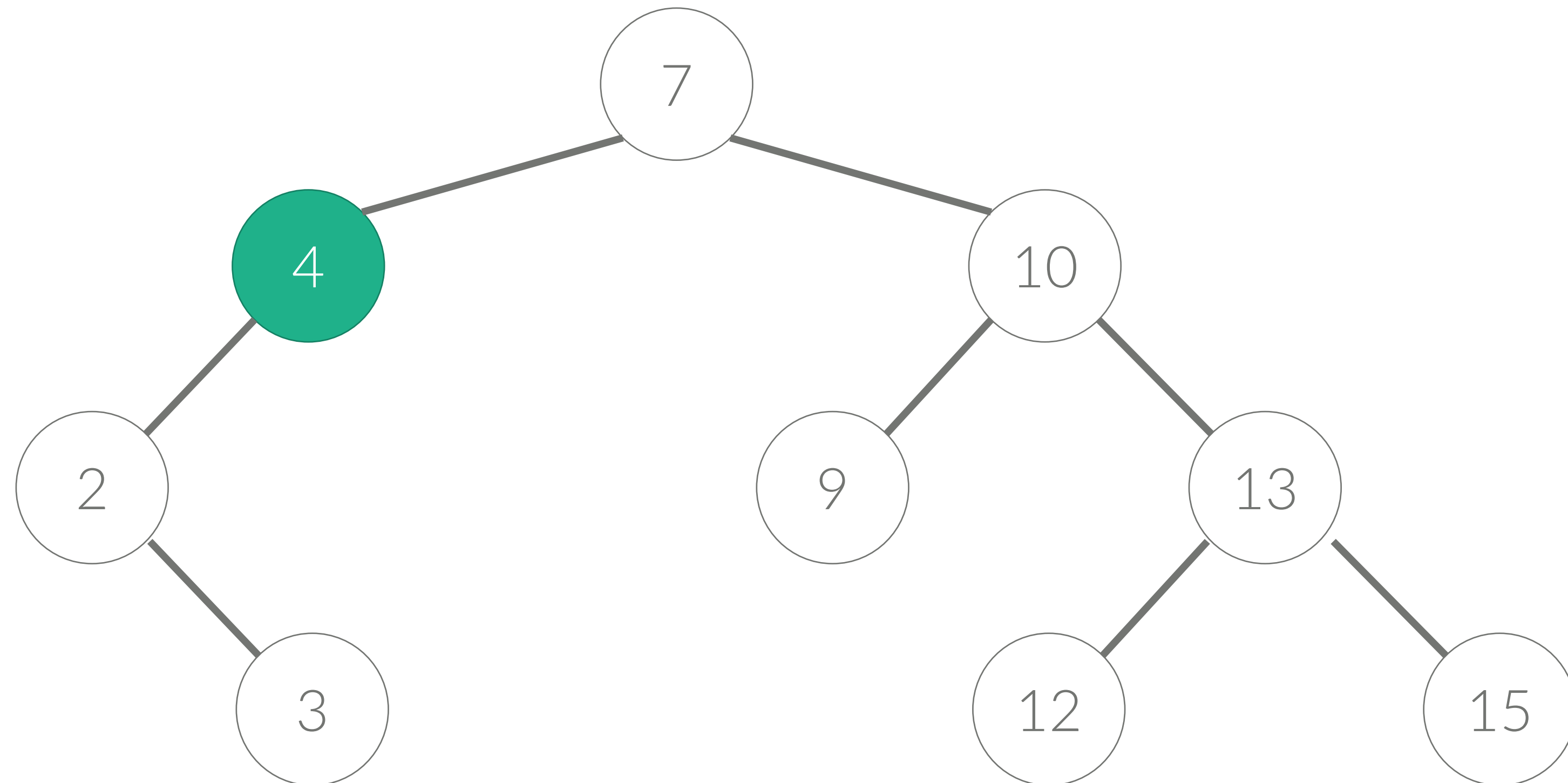


# 이진 검색 트리 삭제

137

Binary Search Tree

- 자식이 1개인 경우
- 노드를 지우고 자식을 이어 붙이면 된다

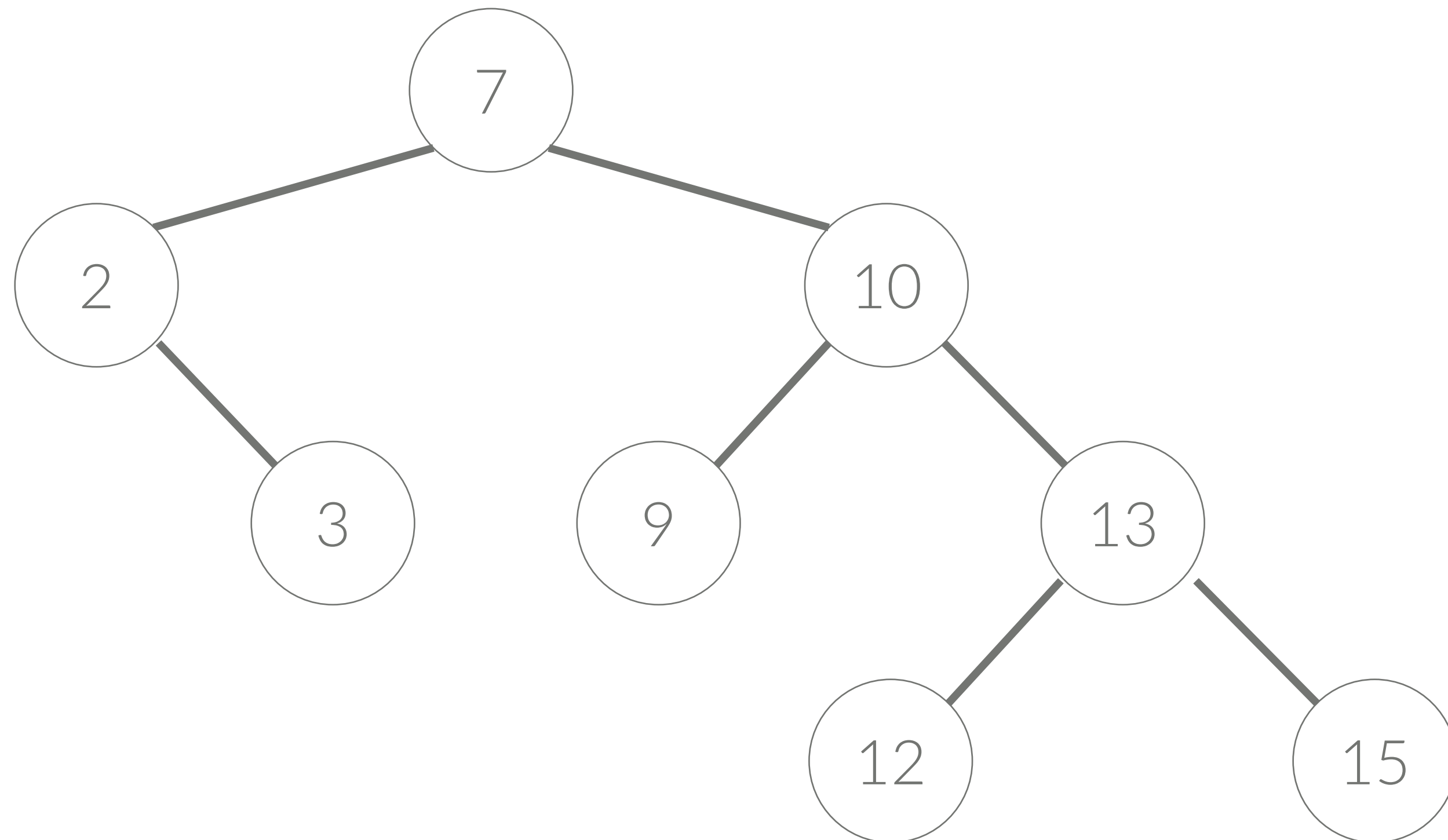


# 이진 검색 트리 삭제

Binary Search Tree

138

- 자식이 1개인 경우
- 노드를 지우고 자식을 이어 붙이면 된다

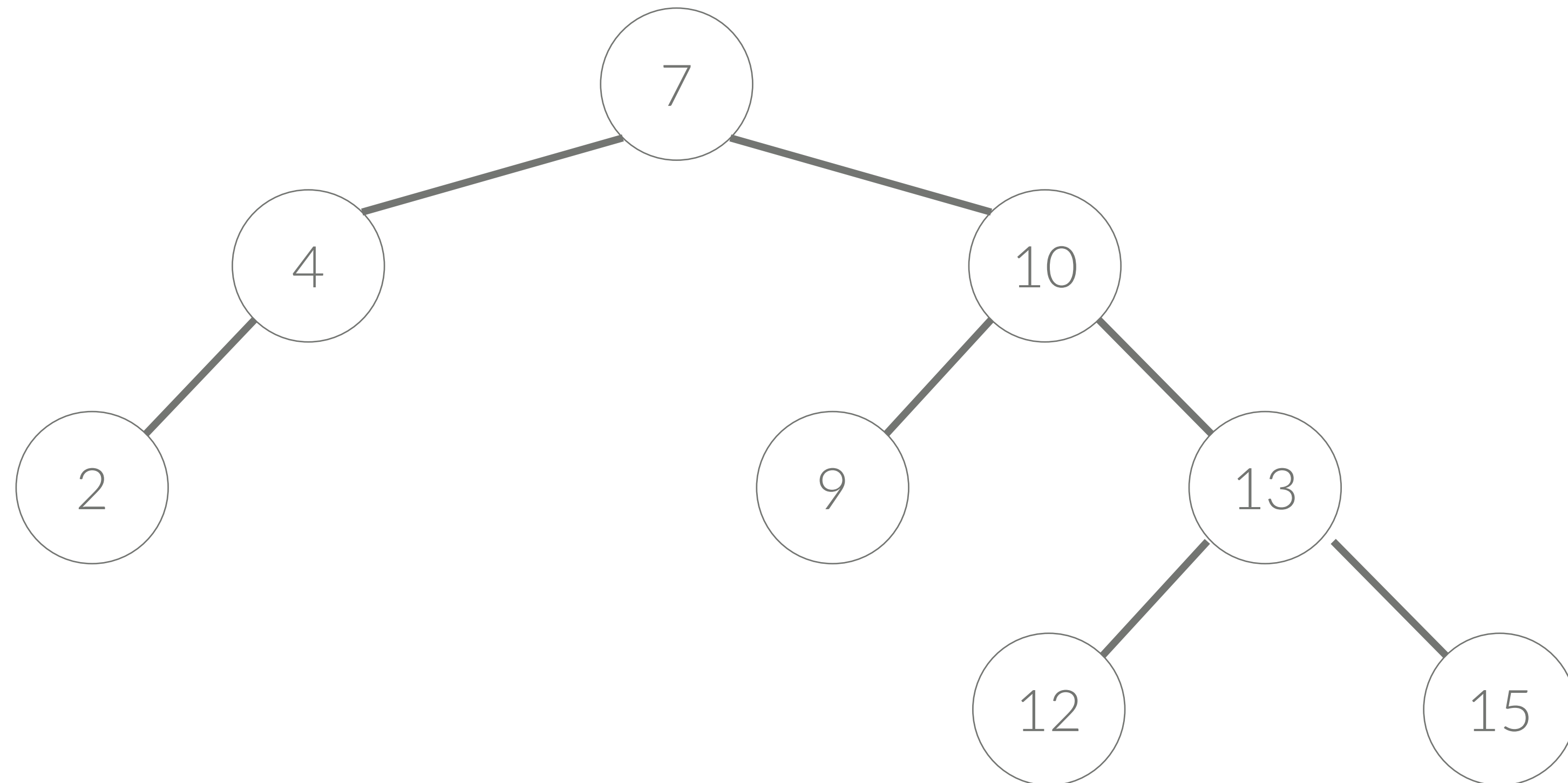


# 이진 검색 트리 삭제

139

Binary Search Tree

- 자식이 2개인 경우
- 지우려는 노드의 값을, in-order successor 노드의 값으로 바꿔준 다음
- in-order successor 노드를 지운다

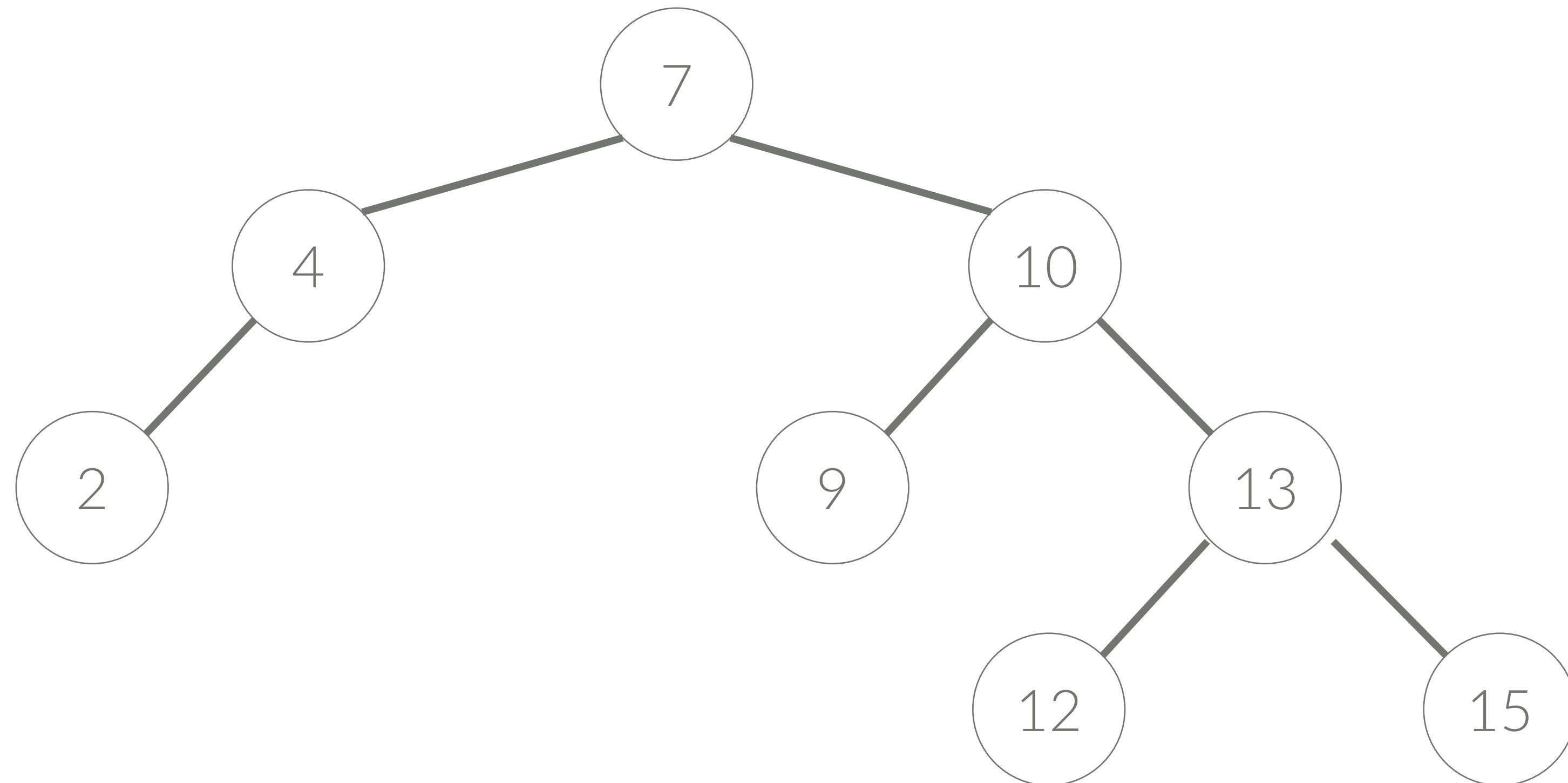


# 이진 검색 트리 삭제

140

## Binary Search Tree

- in-order successor란, 인오더 순회 했을 때, 바로 다음에 오는 값을 말한다.
- 7의 in-order successor는 9이고
- 10의 in-order successor는 12이다



# 이진 검색 트리 삭제

Binary Search Tree

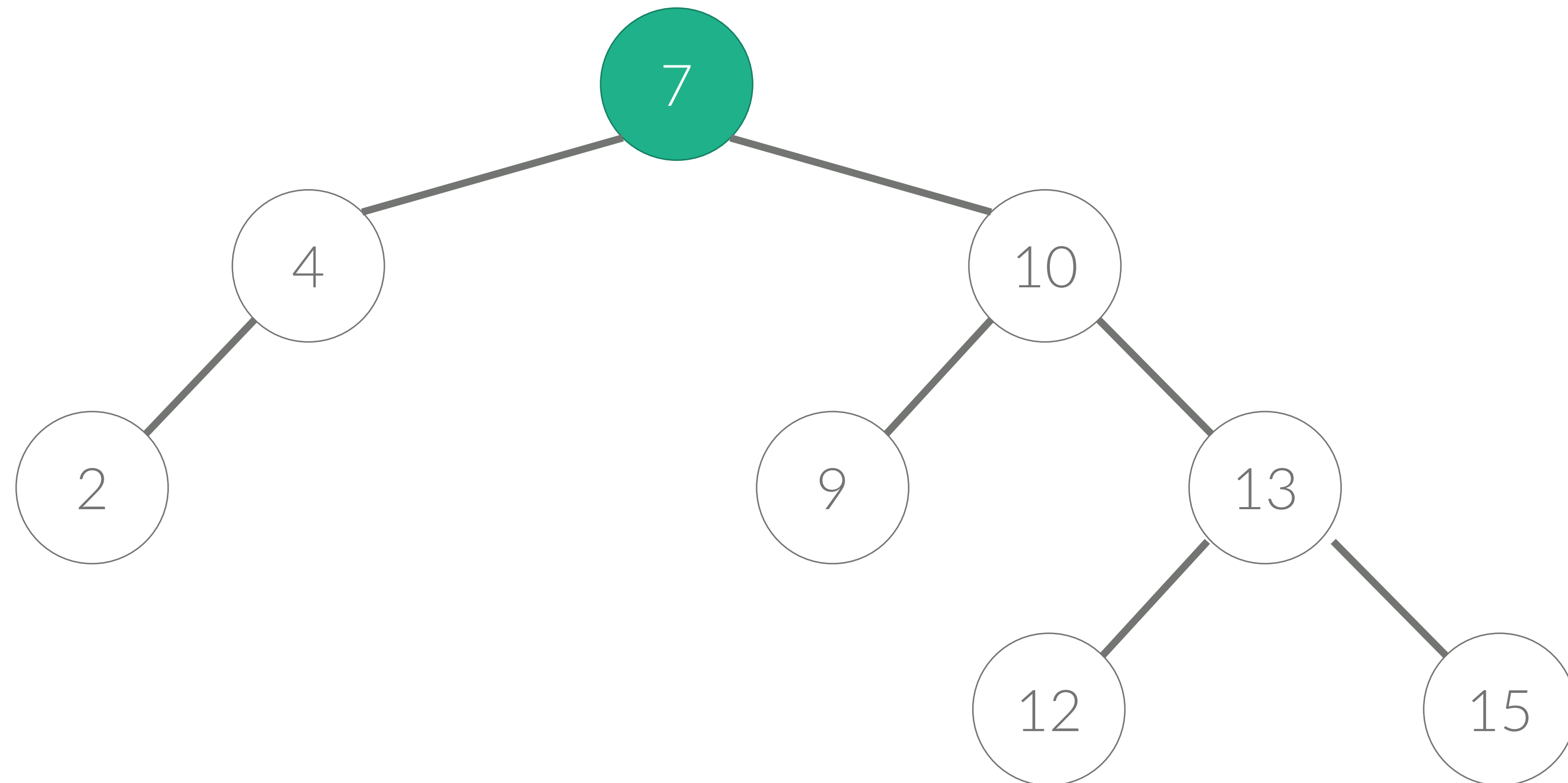
```
Node *inorder_successor(Node *node) {  
    Node *p = node;  
    while (p->left != NULL) {  
        p = p->left;  
    }  
    return p;  
}
```

# 이진 검색 트리 삭제

Binary Search Tree

142

- 7을 지우는 경우

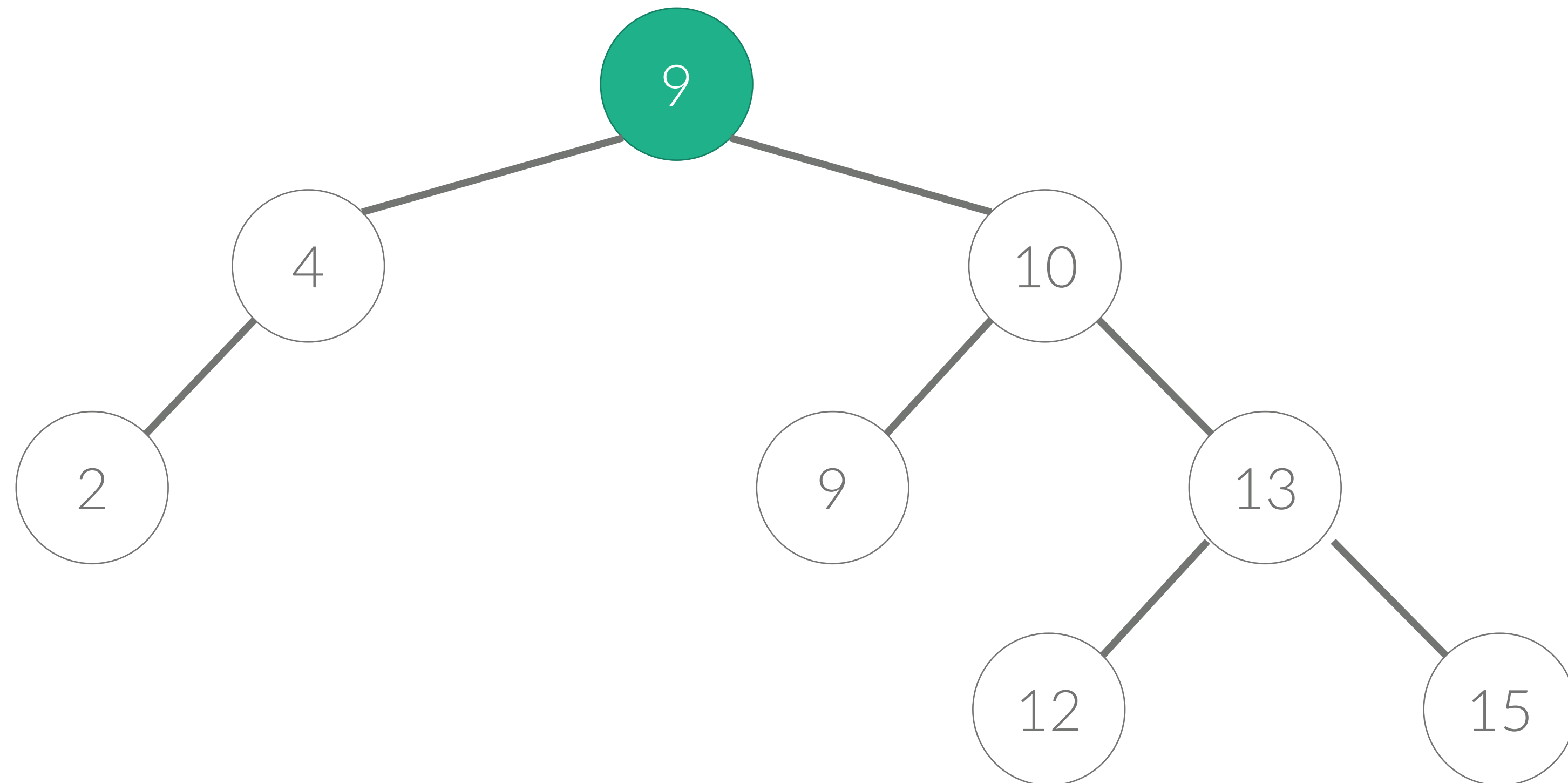


# 이진 검색 트리 삭제

Binary Search Tree

143

- 7을 지우는 경우

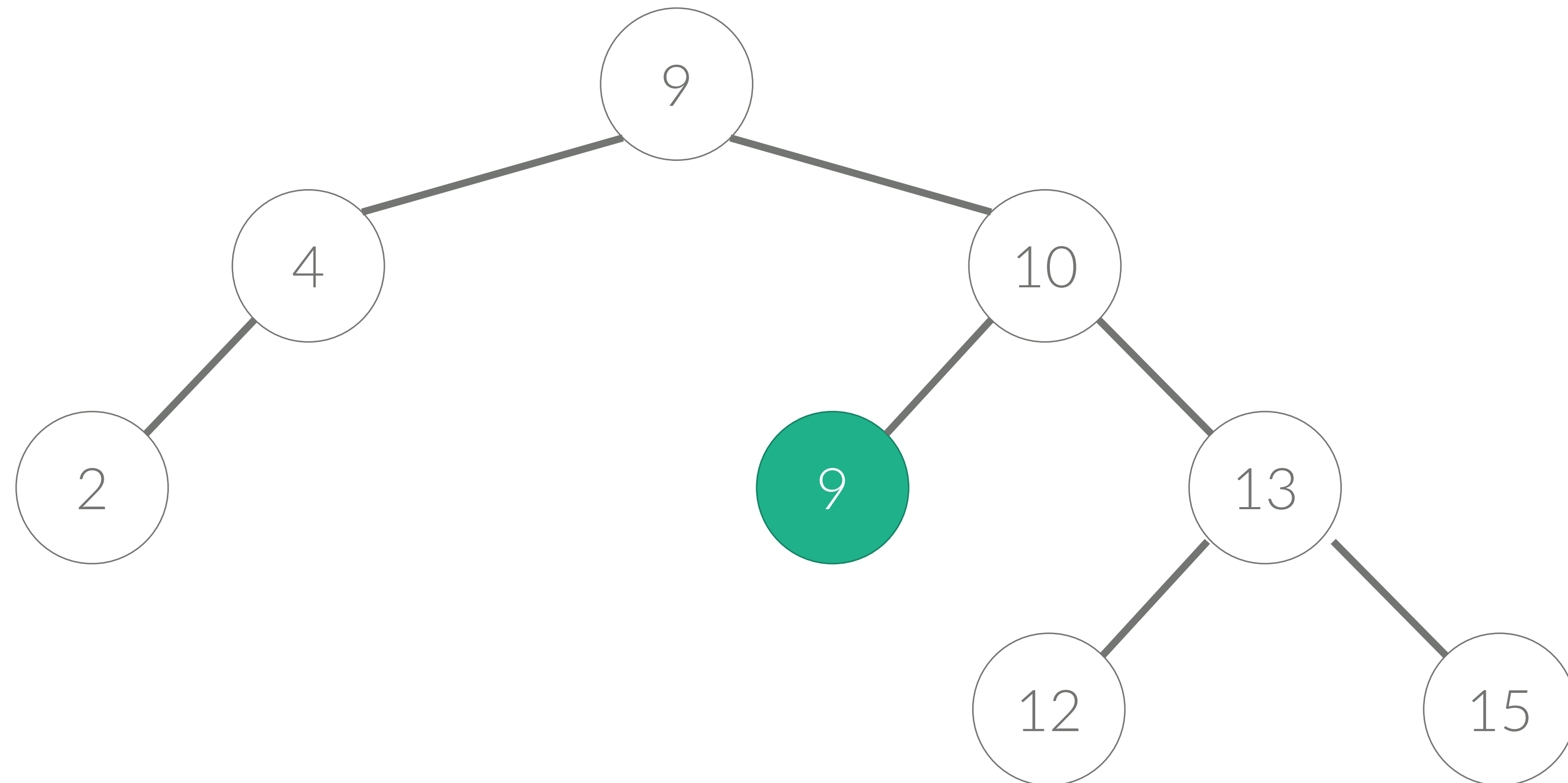


# 이진 검색 트리 삭제

Binary Search Tree

- 7을 지우는 경우

144



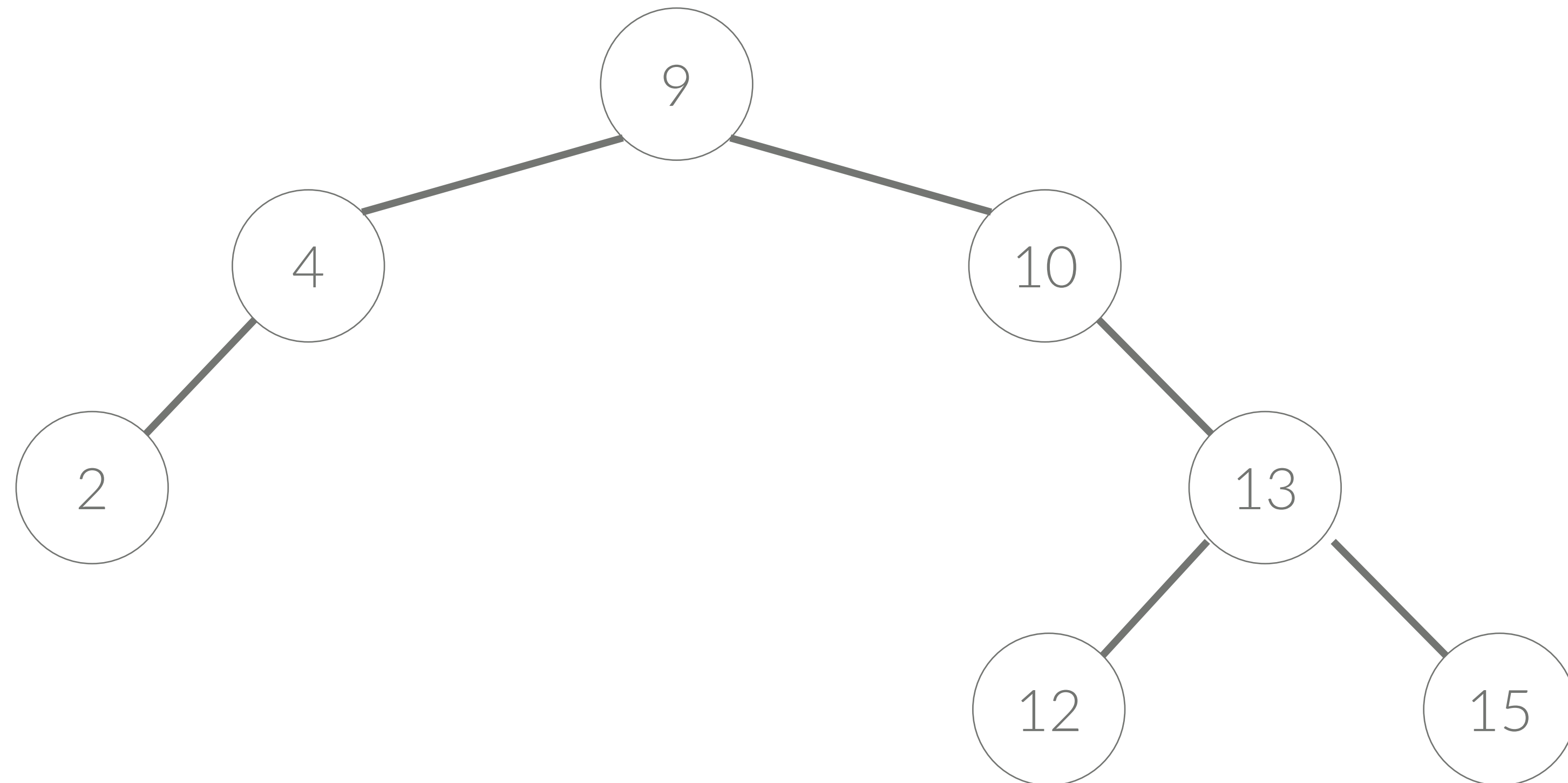


# 이진 검색 트리 삭제

Binary Search Tree

145

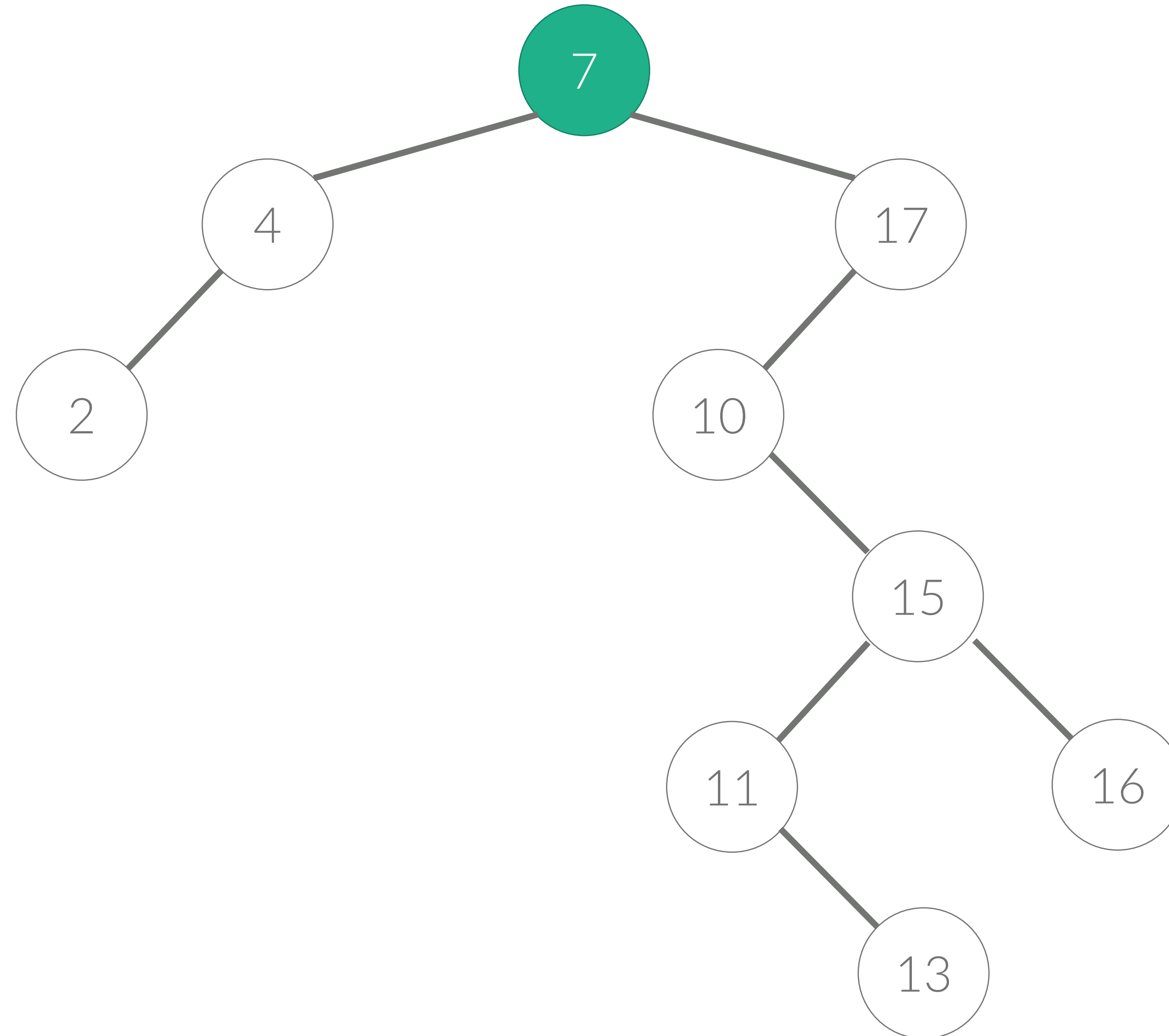
- 7을 지우는 경우



# 이진 검색 트리 삭제

Binary Search Tree

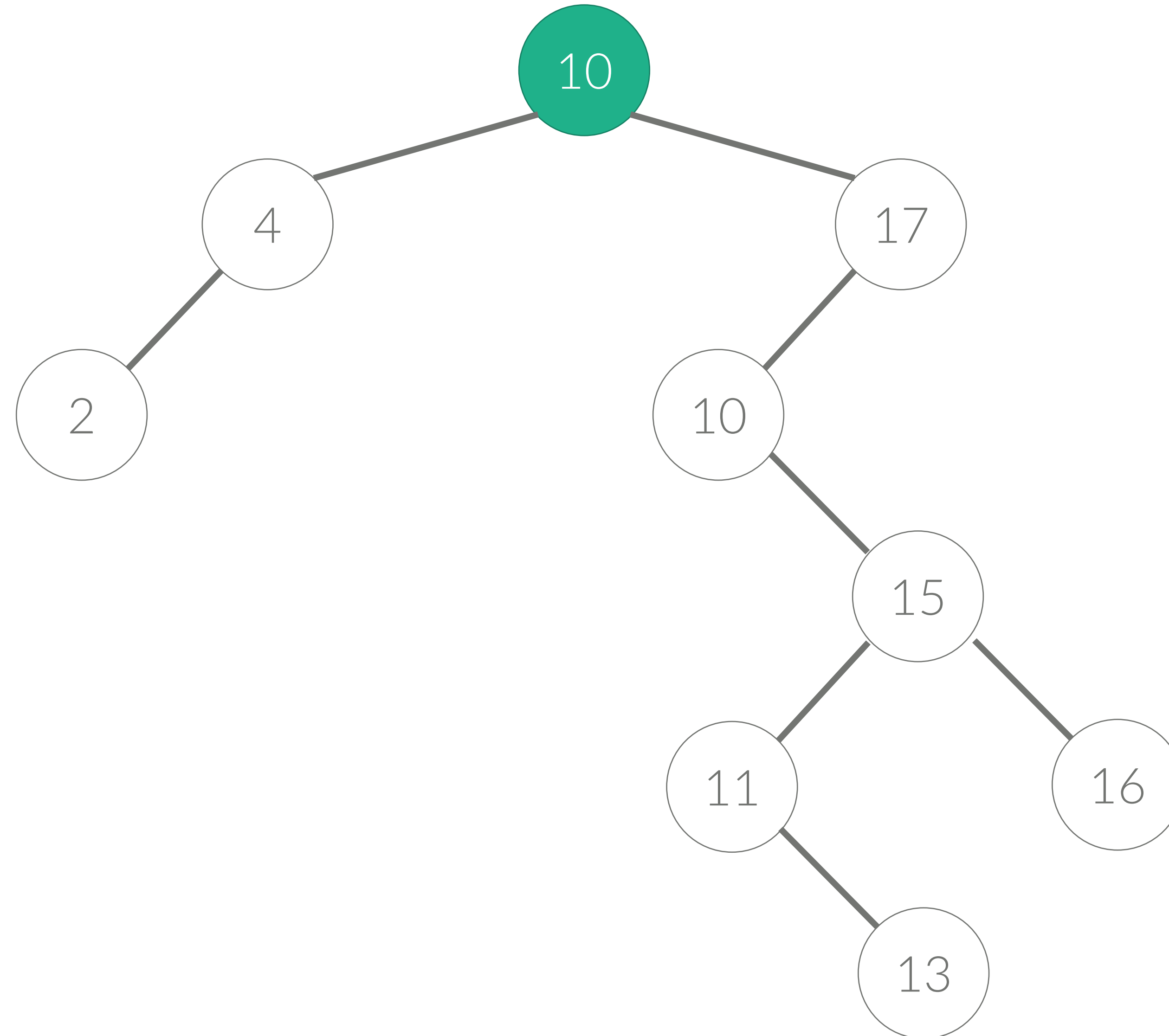
- 7을 지우는 경우



# 이진 검색 트리 삭제

Binary Search Tree

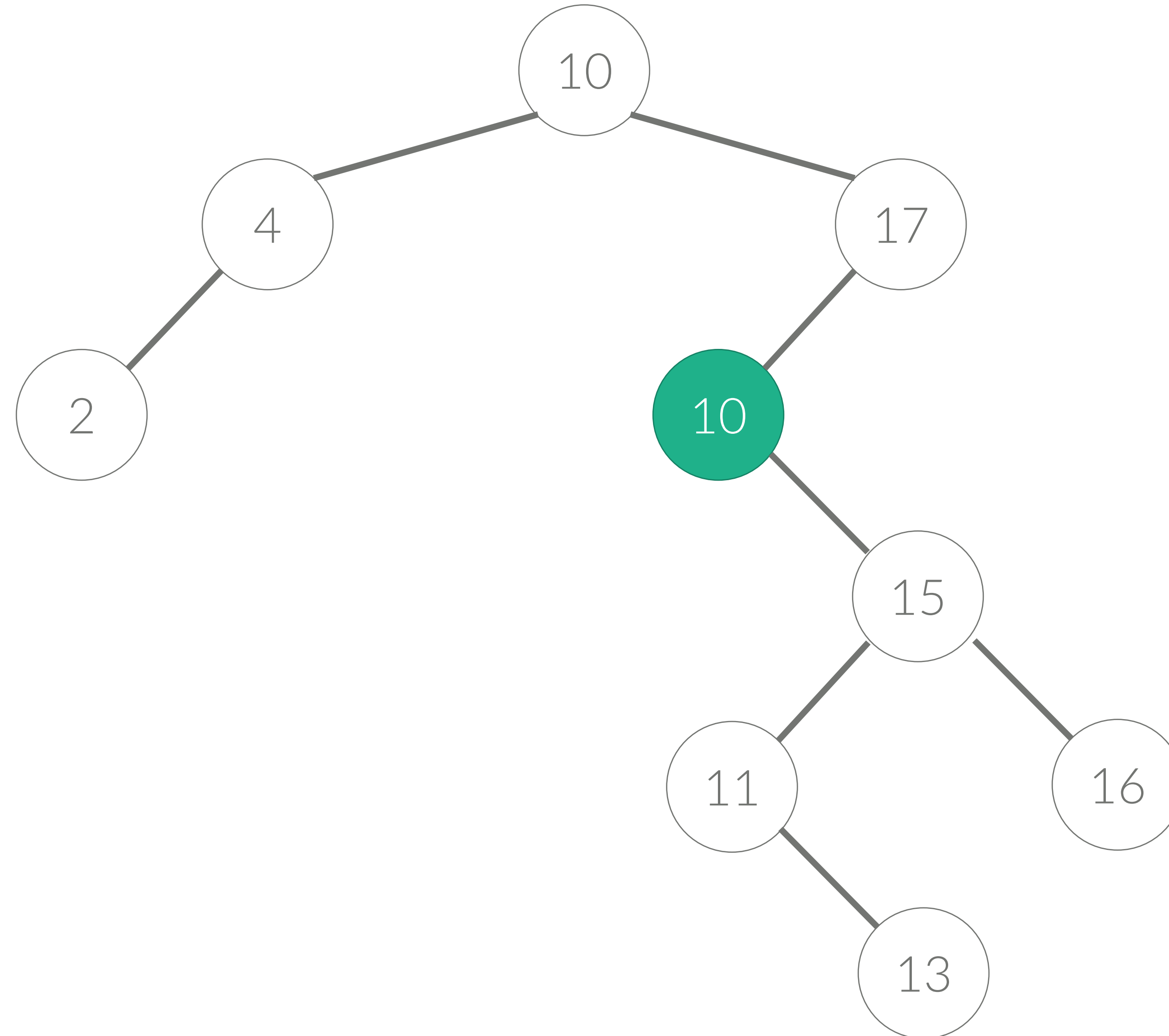
- 7을 지우는 경우



# 이진 검색 트리 삭제

Binary Search Tree

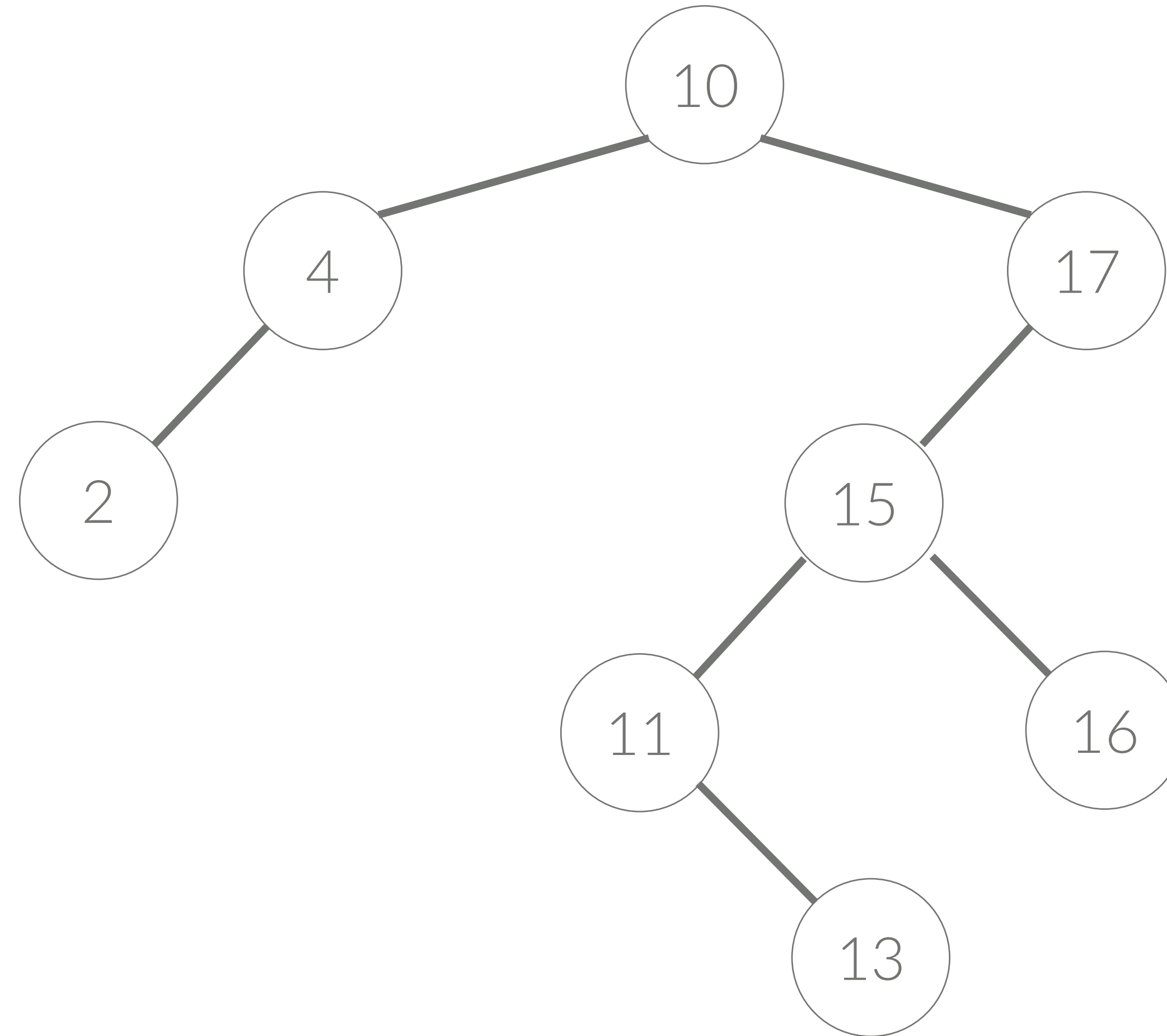
- 7을 지우는 경우



# 이진 검색 트리 삭제

Binary Search Tree

- 7을 지우는 경우



# 이진 검색 트리 삭제

## Binary Search Tree

```
Node *remove(Node *node, int data) {  
    if (node == NULL) {  
        return node;  
    }  
    if (data < node->data) {  
        node->left = remove(node->left, data);  
    } else if (data > node->data) {  
        node->right = remove(node->right, data);  
    } else {  
        // 다음 페이지  
    }  
    return node;  
}
```

# 이진 검색 트리 삭제

## Binary Search Tree

```
if (node->left == NULL) {
    Node *temp = node->right;
    node = NULL;
    return temp;
} else if (node->right == NULL) {
    Node *temp = node->left;
    node = NULL;
    return temp;
}

Node *temp = inorder_successor(node->right);
node->data = temp->data;
node->right = remove(node->right, temp->data);
```

# 이진 검색 트리

Binary Search Tree

152

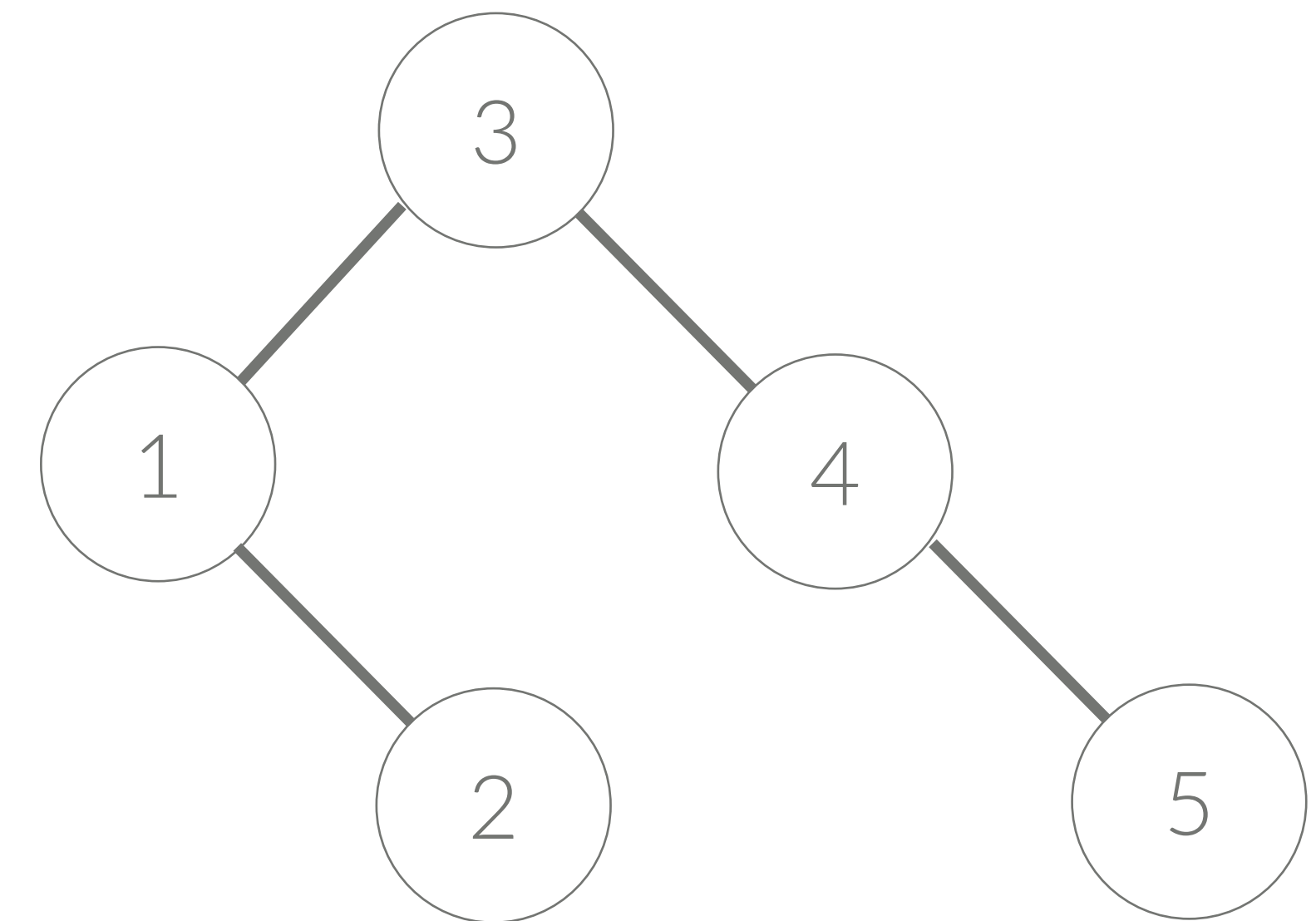
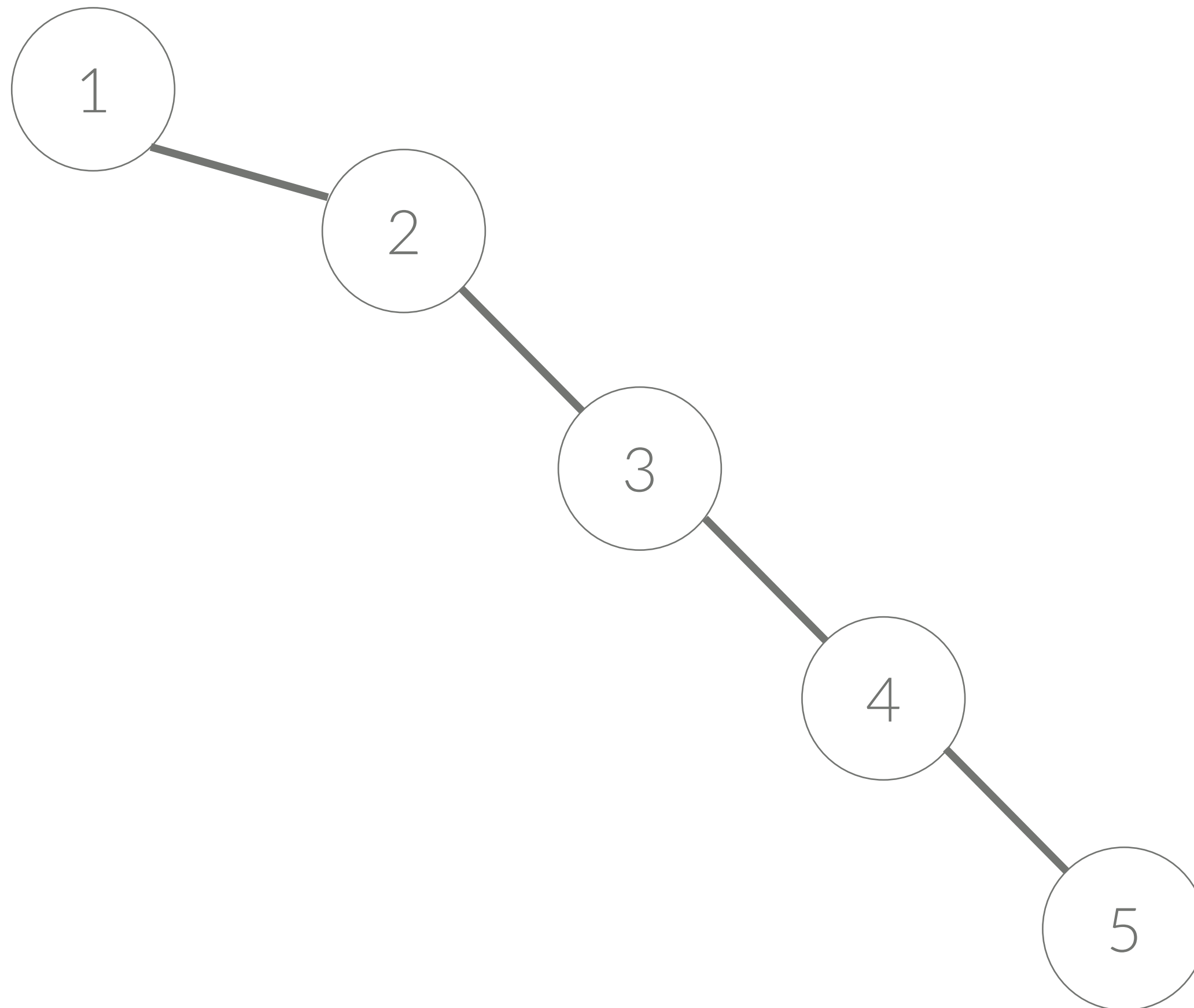
- 소스: <http://codeplus.codes/f1039acf964f4db39f48c2291d75807f>



# 이진 검색 트리

## Binary Search Tree

- 같은 데이터도 어떤 순서로 넣냐에 따라서 트리의 형태가 달라지게 된다
- 1, 2, 3, 4, 5를 1, 2, 3, 4, 5로 넣은 트리과 3, 1, 2, 4, 5로 넣은 트리의 모양은 다르다



# 이진 검색 트리

## Binary Search Tree

154

- BST의 삽입/삭제/검색은 시간 복잡도가  $O(h)$ 가 된다
- 높이의 최대값은  $N$ 이 될 수 있기 때문에,  $O(N)$ 이 될 수 있다
- 따라서, 균형이 맞춰진 트리를 사용하는 것이 좋다

# 이진 검색 트리

Binary Search Tree

155

- 균형이 맞춰져 있는 BST는
- AVL-Tree
- Red-black Tree
- Splay Tree
- Treap 이 있다

# 이진 검색 트리

156

## Binary Search Tree

- C++의 경우에는 `set`, `map`, Java는 `TreeSet`, `TreeMap`을 이용할 수 있다.
- 정렬된 상태가 필요 없다면
- C++은 `unordered_set`, `unordered_map`, Java는 `HashSet`, `HashMap`을 이용할 수 있다.

# 회사에 있는 사람

157

<https://www.acmicpc.net/problem/7785>

- 회사 모든 사람의 출입카드 시스템 로그를 가지고 있다.
- 로그는 어떤 사람이 들어갔는지, 나갔는지가 기록되어 있다.
- 로그가 주어졌을 때, 회사에 있는 모든 사람을 구하는 문제

# 회사에 있는 사람

158

<https://www.acmicpc.net/problem/7785>

- 소스: <http://codeplus.codes/2e88a3eab0d340d0950c0247299f213d>

# 듣보잡

159

<https://www.acmicpc.net/problem/1764>

- 듣도 못한 사람과 보도 못한 사람의 명단이 주어졌을 때
- 듣도 보도 못한 사람의 명단을 구하는 문제

# 등보잡

160

<https://www.acmicpc.net/problem/1764>

- map 소스: <http://codeplus.codes/3a32707334c94b338b8c016264feb06f>
- set 소스: <http://codeplus.codes/806a5d048d22433baf934c693025c48c>
- 투 포인터 소스: <http://codeplus.codes/3245e0431d3d459896624a6b0383bf2a>