고 급 문 제 해 결

<문제 4.11>

두 사각형끼리 겹치는지 확인

2018117543 한재성

Chapter 4
Primitive types

4.11 – 두 사각형끼리 겹치는지 확인

5.11 RECTANGLE INTERSECTION

This problem is concerned with rectangles whose sides are parallel to the X-axis and Y-axis. See Figure 5.2 for examples.

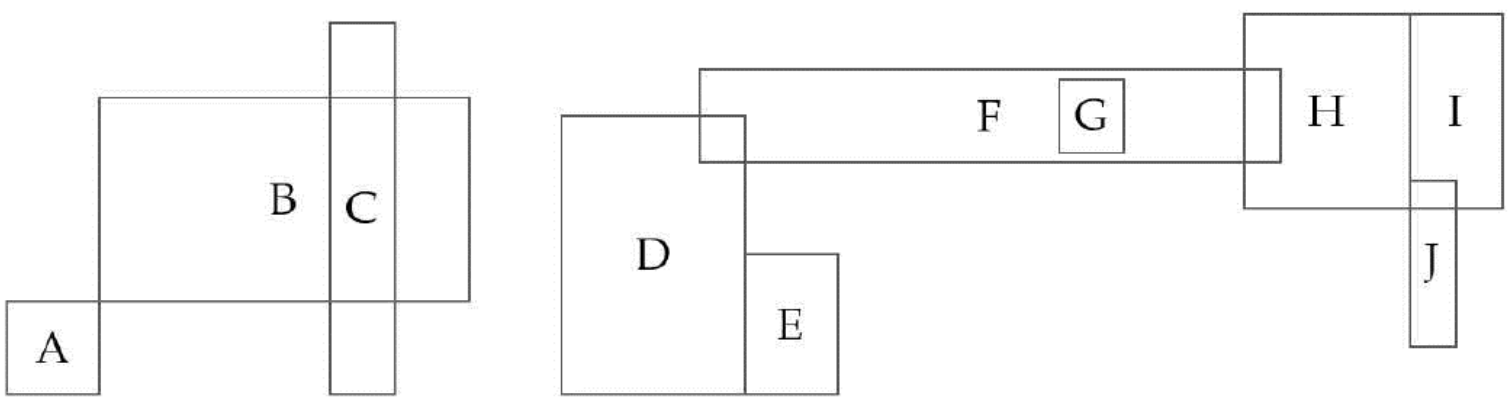
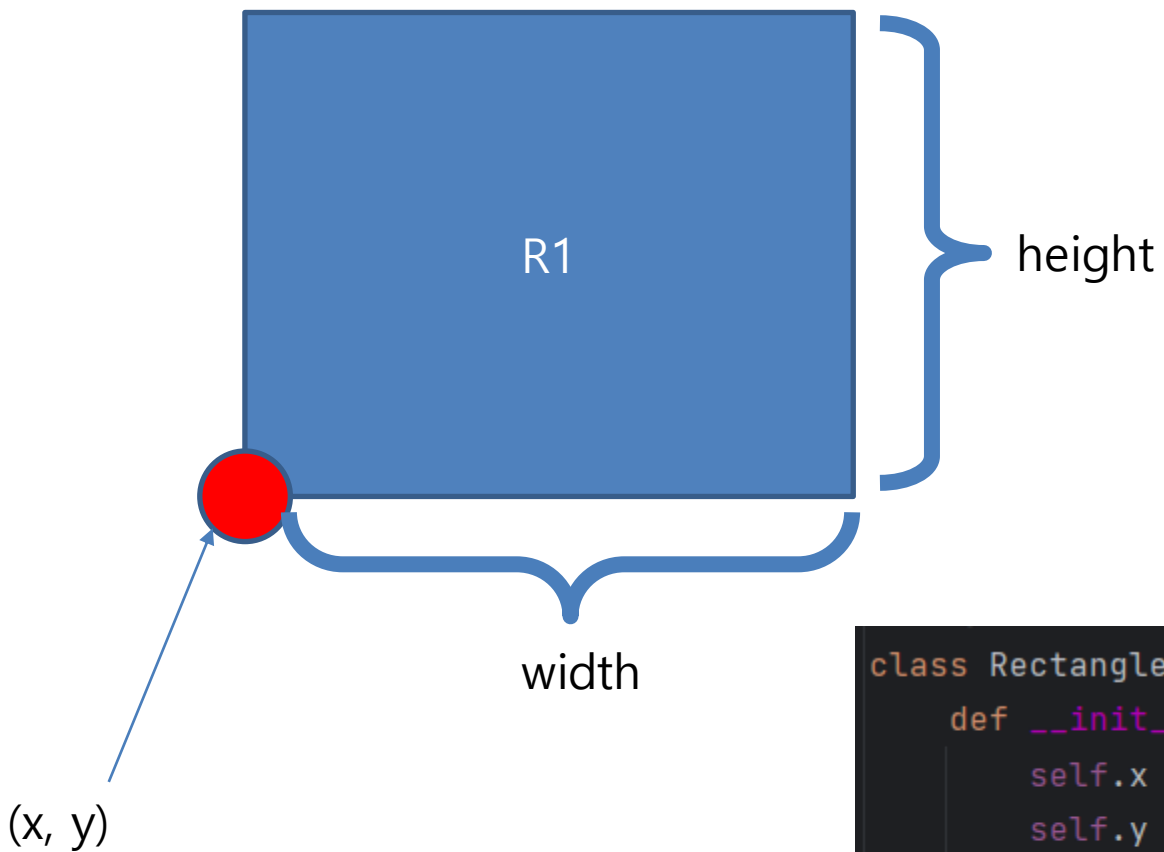


Figure 5.2: Examples of XY-aligned rectangles.

사각형 표현 방법

사각형의 표현




```
class Rectangle():
    def __init__(self, x=0, y=0, w=0, h=0):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
```

Idea

Rectangle intersection의 판단

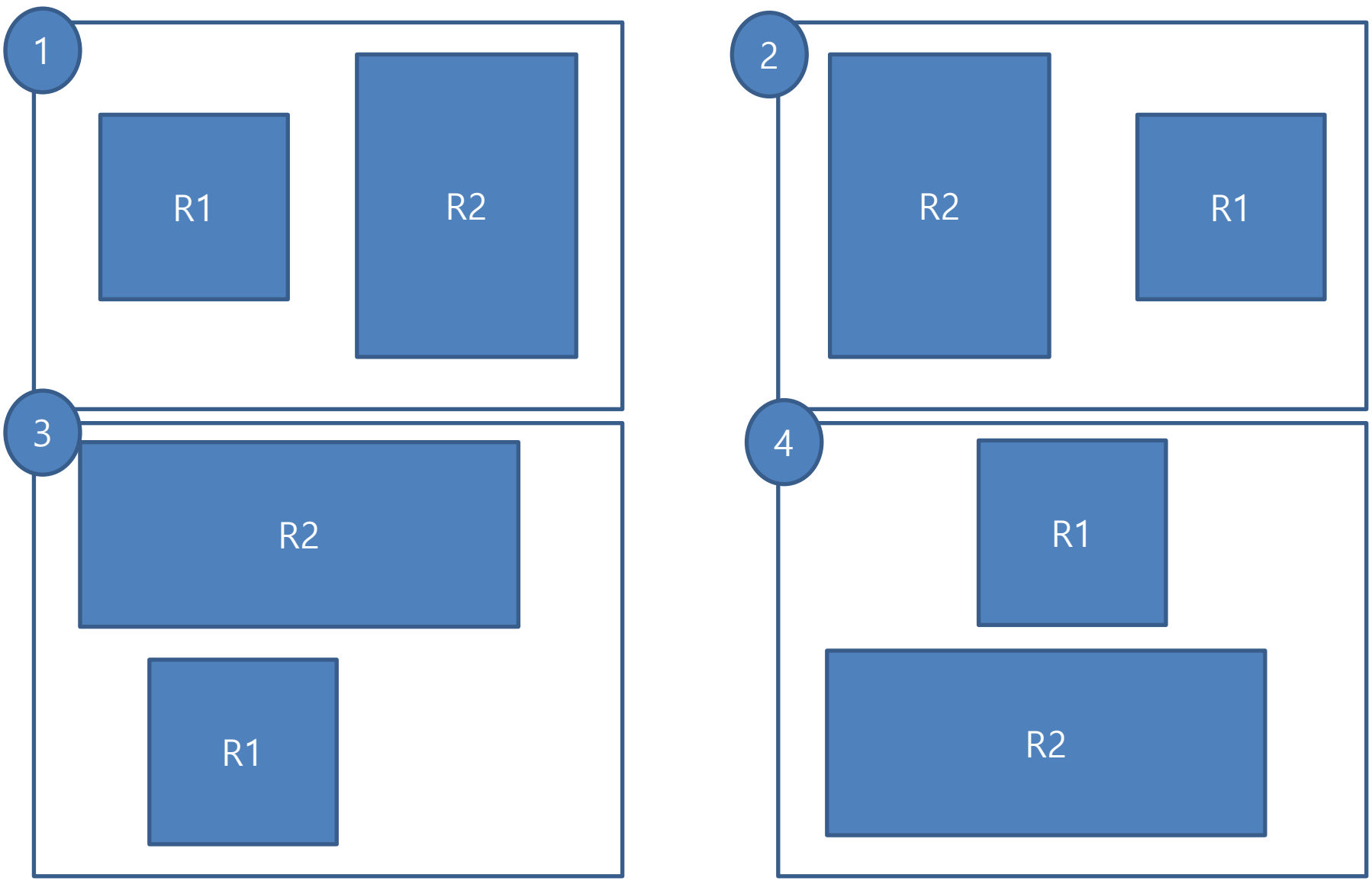
- 꼭짓점 혹은 모서리가 겹치더라도 겹친 것으로 판단
- 한 사각형이, 다른 사각형에 온전히 포함되더라도 겹쳤다고 판단

거꾸로 생각하기

- 모든 경우의 수를 따지지 말고, 겹치지 않을 때를 먼저 생각하자.
- 그리고 나서, 그 조건을 Negation시킨 것이 곧 겹칠 때의 조건.

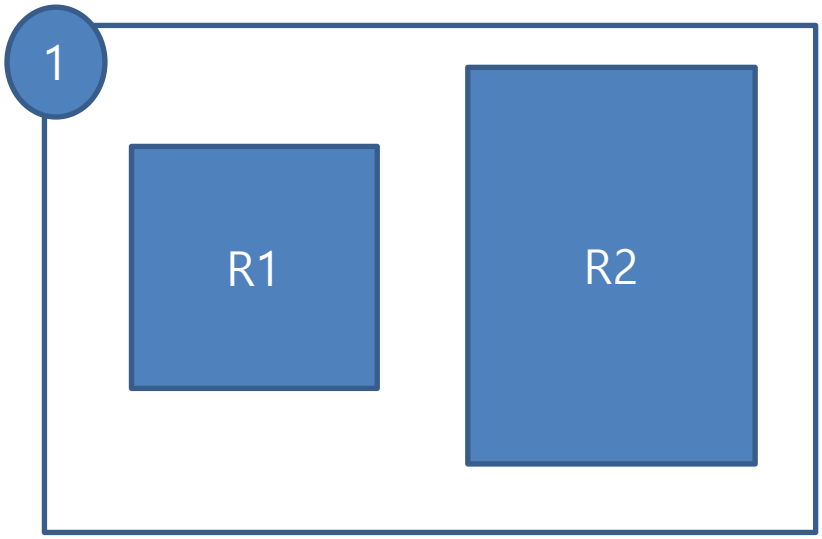
Idea

사각형끼리 겹치지 않을 때는 ?

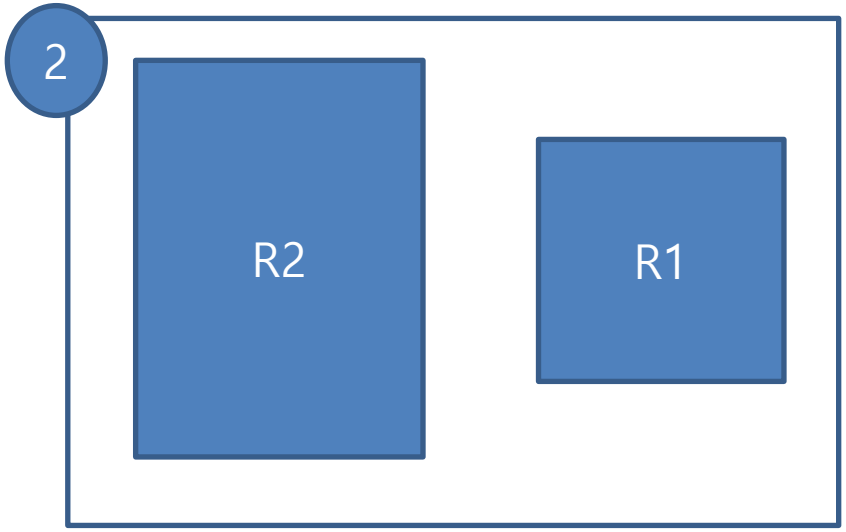


Idea

사각형끼리 겹치지 않을 때는 ?



R1.x + R1.w < R2.x

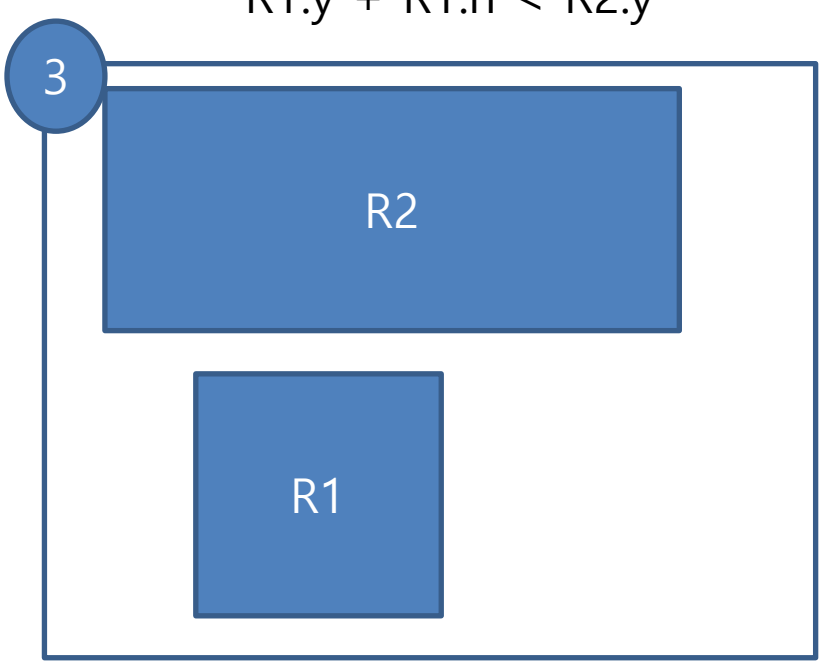


R2.x + R2.w < R1.x

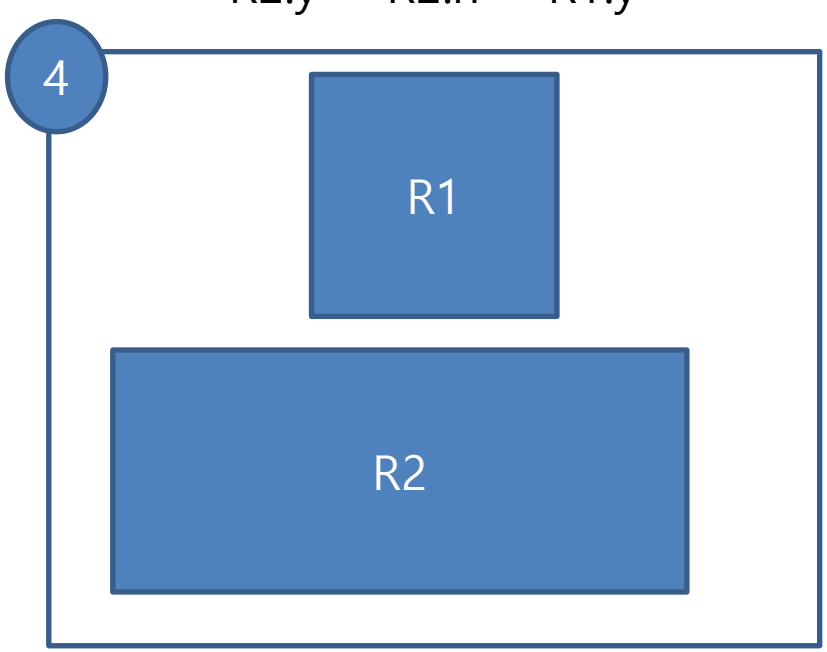
Idea

사각형끼리 겹치지 않을 때는 ?

R1.y + R1.h < R2.y



R2.y + R2.h < R1.y



Idea

사각형끼리 겹치지 않을 때의 총 조건

(R1.y + R1.h < R2.y) OR (R2.y + R2.h < R1.y)

OR (R1.y + R1.h < R2.y) OR (R2.y + R2.h < R1.y)

Negation

사각형끼리 겹칠 때의 조건

```
def is_intersect(R1: Rectangle, R2: Rectangle):
    return (R1.x <= R2.x + R2.w) and (R1.x + R1.w >= R2.x) \
        and (R1.y <= R2.y + R2.h) and (R1.y + R1.h >= R2.y)
```

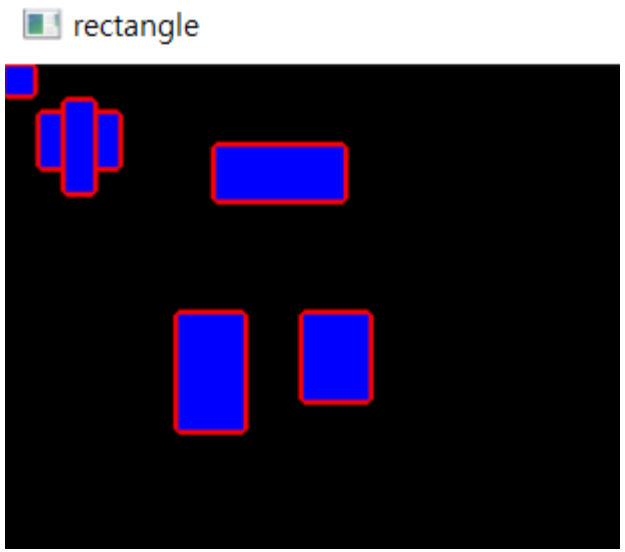
Code

```
L1 = [Rectangle(1, 1, 10, 10), Rectangle(15, 20, 30, 20), Rectangle(25, 15, 10, 35),
     Rectangle(70, 100, 25, 45), Rectangle(85, 33, 50, 20), Rectangle(120, 100, 25, 33)]

L2 = [Rectangle(1, 1, 10, 10), Rectangle(12, 20, 30, 20), Rectangle(1, 1, 10, 5),
     Rectangle(70, 100, 25, 45), Rectangle(85, 33, 50, 20), Rectangle(120, 100, 25, 33)]

L = L1
# Step 1
for i in range(len(L)):
    for j in range(i + 1, len(L)):
        if is_intersect(L[i], L[j]):
            print(f'{i}, {j} -> intersect.')
        else:
            print(f'{i}, {j} -> not intersect.')
```

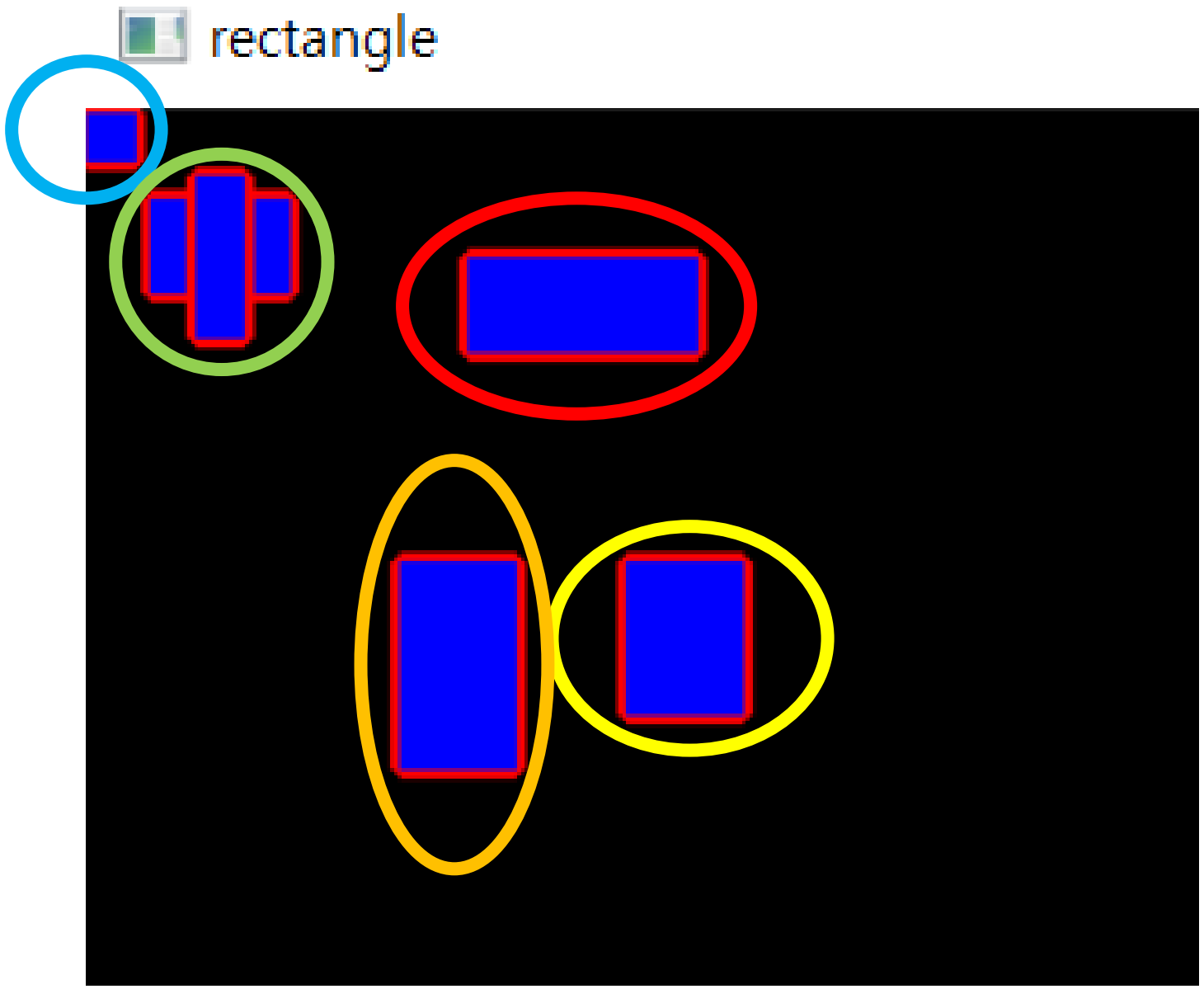
결과

사각형끼리 겹치는 것을 파악 가능

0, 1 -> not intersect.
0, 2 -> not intersect.
0, 3 -> not intersect.
0, 4 -> not intersect.
0, 5 -> not intersect.
1, 2 -> intersect.
1, 3 -> not intersect.
1, 4 -> not intersect.
1, 5 -> not intersect.
2, 3 -> not intersect.
2, 4 -> not intersect.
2, 5 -> not intersect.
3, 4 -> not intersect.
3, 5 -> not intersect.
4, 5 -> not intersect.

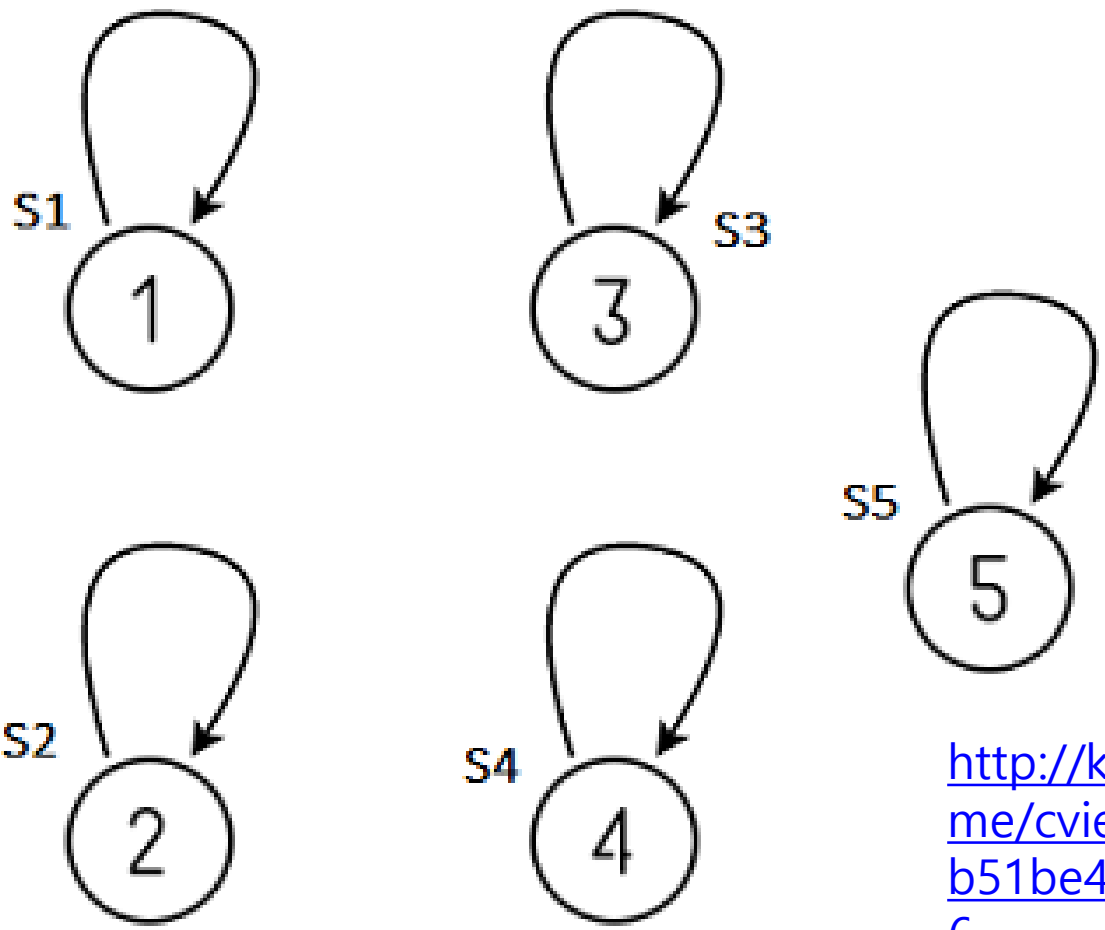


Grouping?



Union-Find

Disjoint Set(처음에는 외딴 섬)

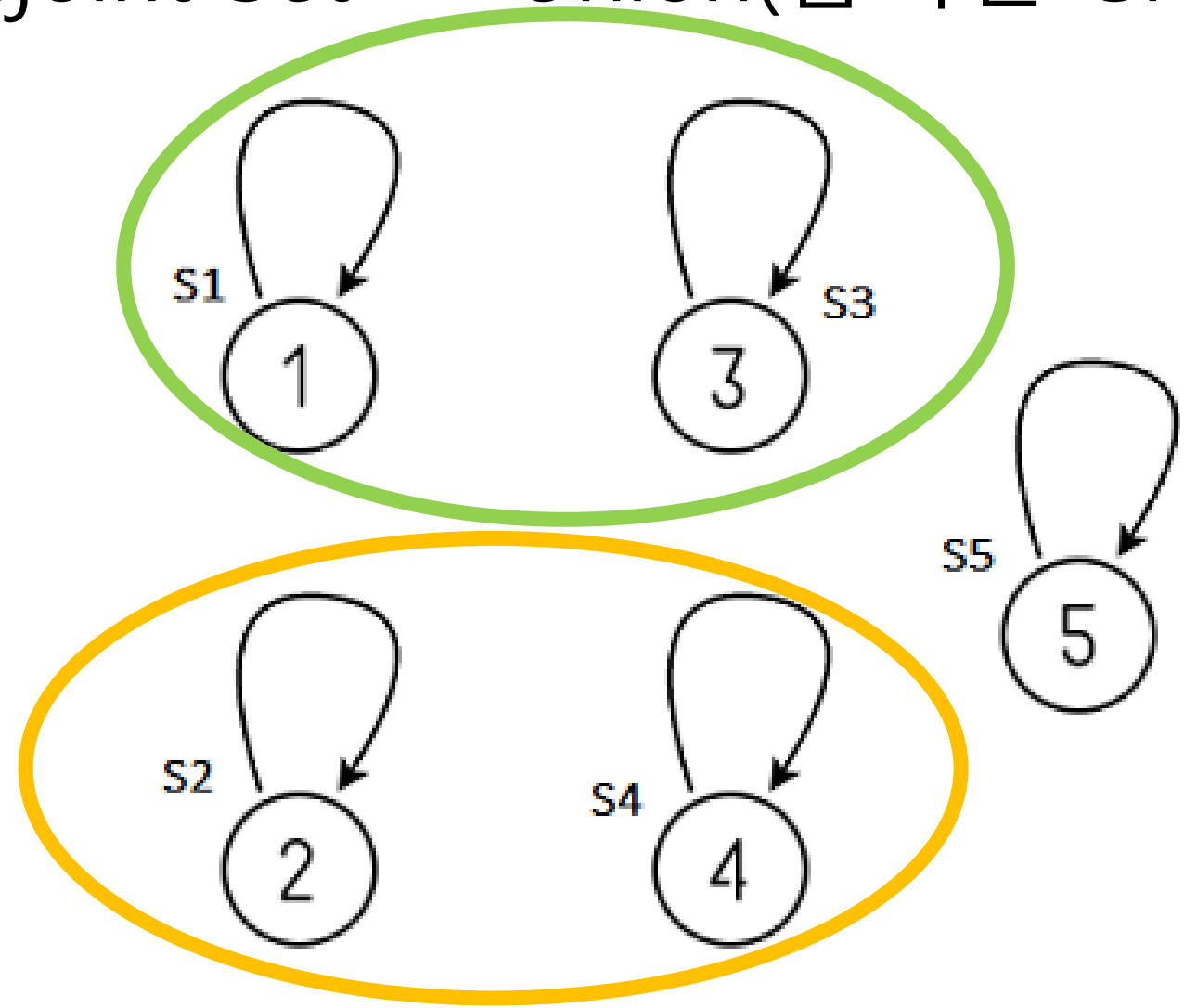


http://kocw.net/home/cview.do?cid=b51be49fd9538786

참고)여기서부터의 강의자료는 Kocw에 공개된 영남대학교 조행래 교수님의 자료구조론 강의자료를 인용하였습니다.

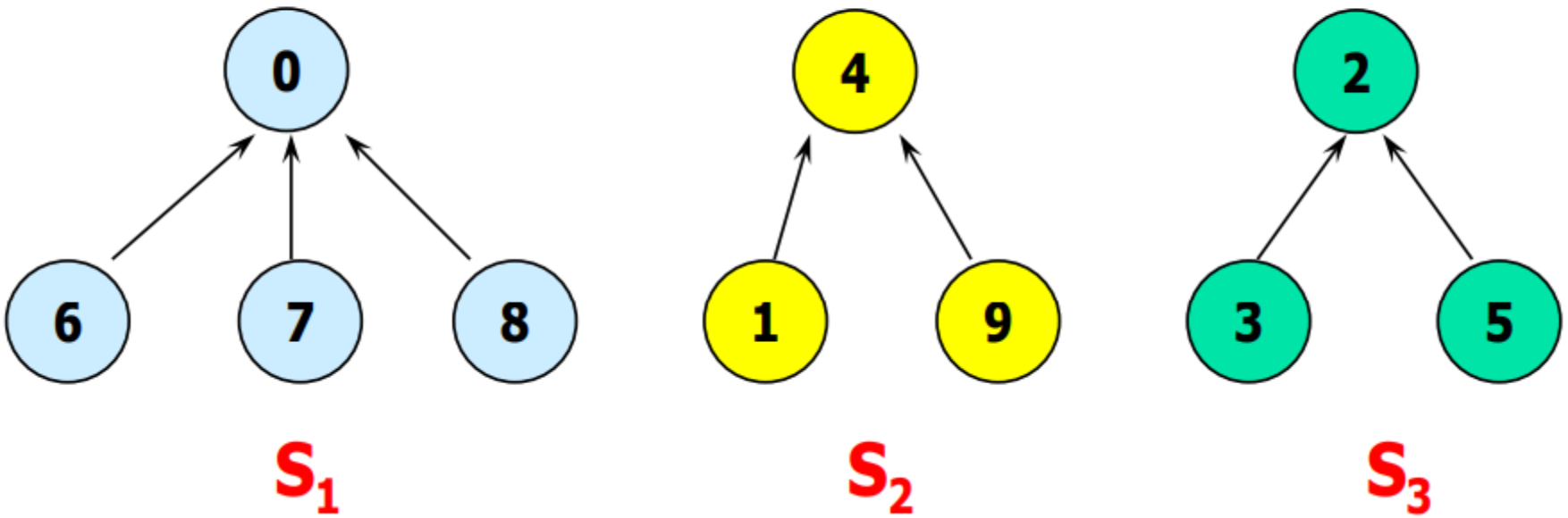
Union-Find

Disjoint Set -> Union(합치면 Group)



Union-Find

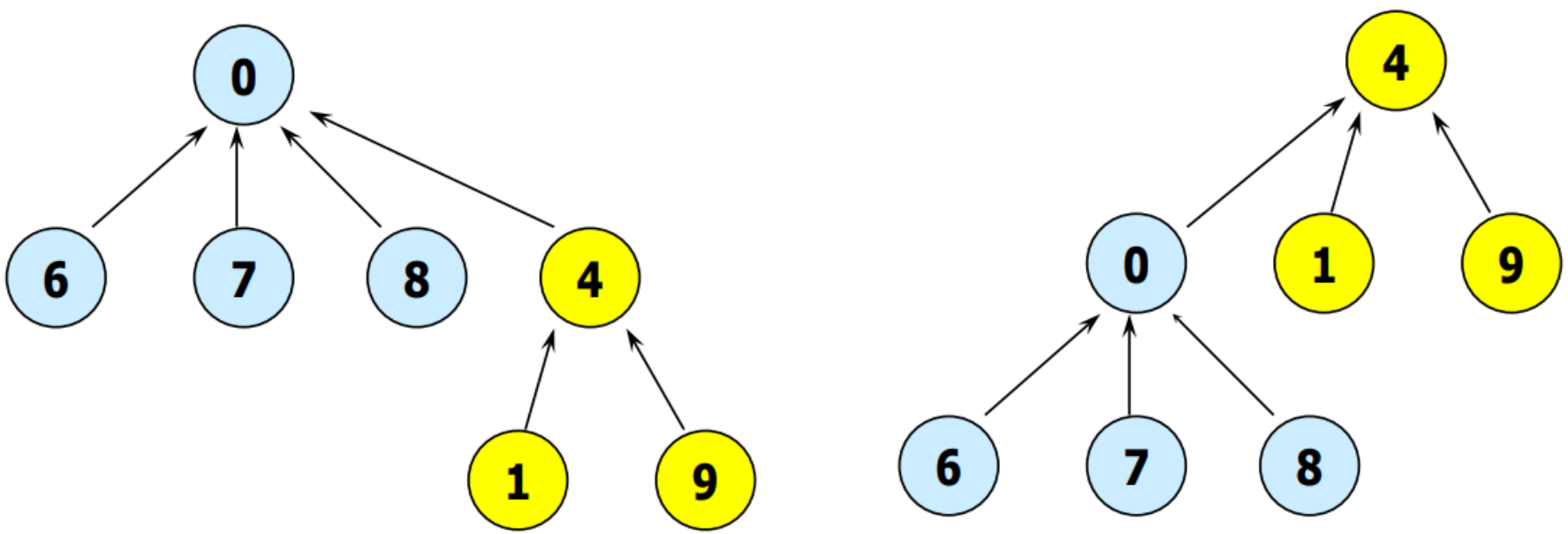
$S_1 = \{0, 6, 7, 8\}, S_2 = \{1, 4, 9\}, S_3 = \{2, 3, 5\}$



i	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
parent	-1	4	-1	2	-1	2	0	0	0	4

http://kocw.net/home/cview.do?cid=b51be49fd9538786

Union-Find



$S_1 \cup S_2$ or $S_2 \cup S_1$

http://kocw.net/home/cview.do?cid=b51be49fd9538786

Union-Find

```python
def simple_union(P: List[int], i: int, j: int) -> None:
    # 주의 : i, j는 항상 root여야 함.
    P[i] = j
```

```python
def simple_find(P: List[int], i: int) -> int:
    while P[i] >= 0:
        i = P[i]
    return i
```

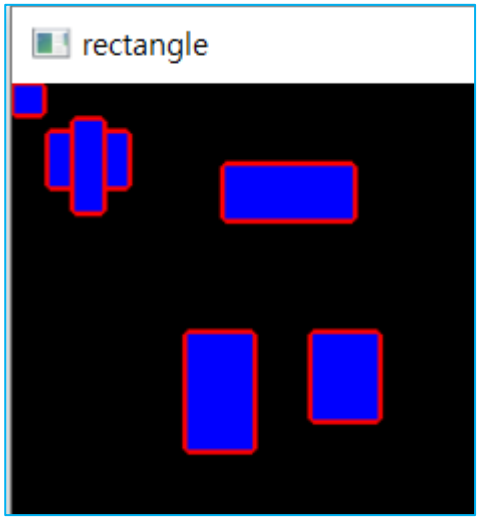
Code

Union-Find 를 이용하여 Rectangle Grouping

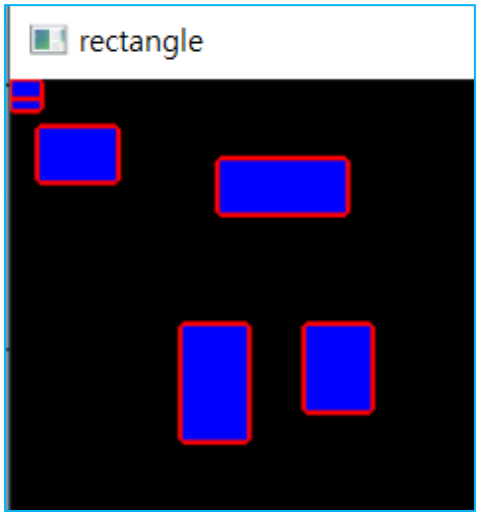
```
for i in range(len(L)):
    for j in range(i + 1, len(L)):
        R1, R2 = L[i], L[j]
        if is_intersect(R1, R2):
            root_left = simple_find(R, i)
            root_right = simple_find(R, j)

            if root_left != root_right:
                simple_union(R, root_left, root_right)
```

결과

```
L1 = [Rectangle(1, 1, 10, 10), Rectangle(15, 20, 30, 20), Rectangle(25, 15, 10, 35),
     Rectangle(70, 100, 25, 45), Rectangle(85, 33, 50, 20), Rectangle(120, 100, 25, 33)]

L2 = [Rectangle(1, 1, 10, 10), Rectangle(12, 20, 30, 20), Rectangle(1, 1, 10, 5),
     Rectangle(70, 100, 25, 45), Rectangle(85, 33, 50, 20), Rectangle(120, 100, 25, 33)]
```



[-1, -1, -1, -1, -1, -1]
[-1, 2, -1, -1, -1, -1]
{0: [0], 2: [2, 1], 3: [3], 4: [4], 5: [5]}

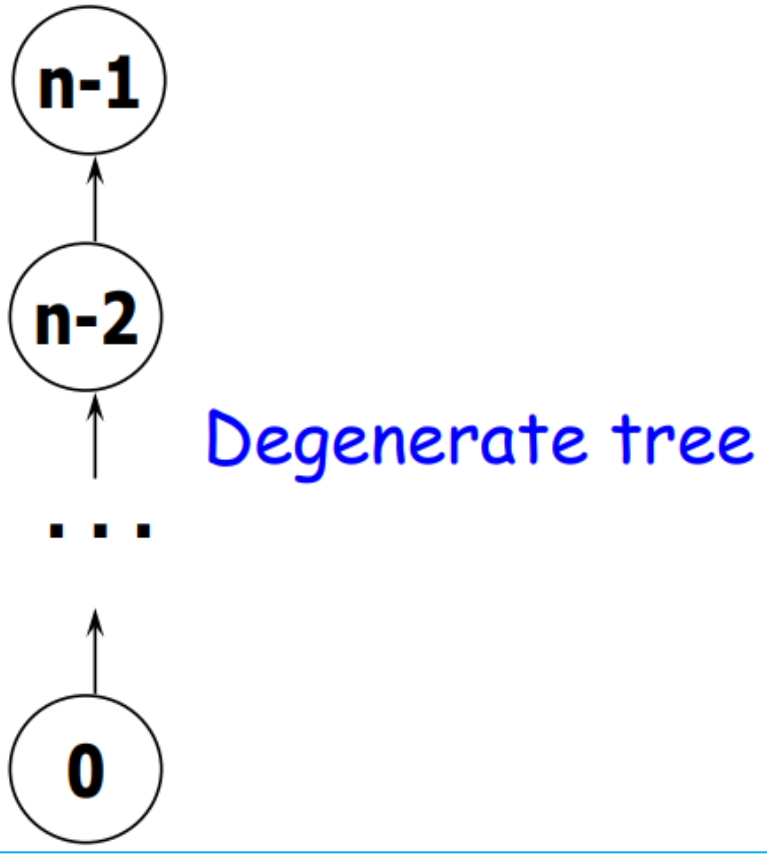


[-1, -1, -1, -1, -1, -1]
[2, -1, -1, -1, -1, -1]
{2: [2, 0], 1: [1], 3: [3], 4: [4], 5: [5]}

참고)Weighted Union

Simple Union의 문제점: Degenerate tree

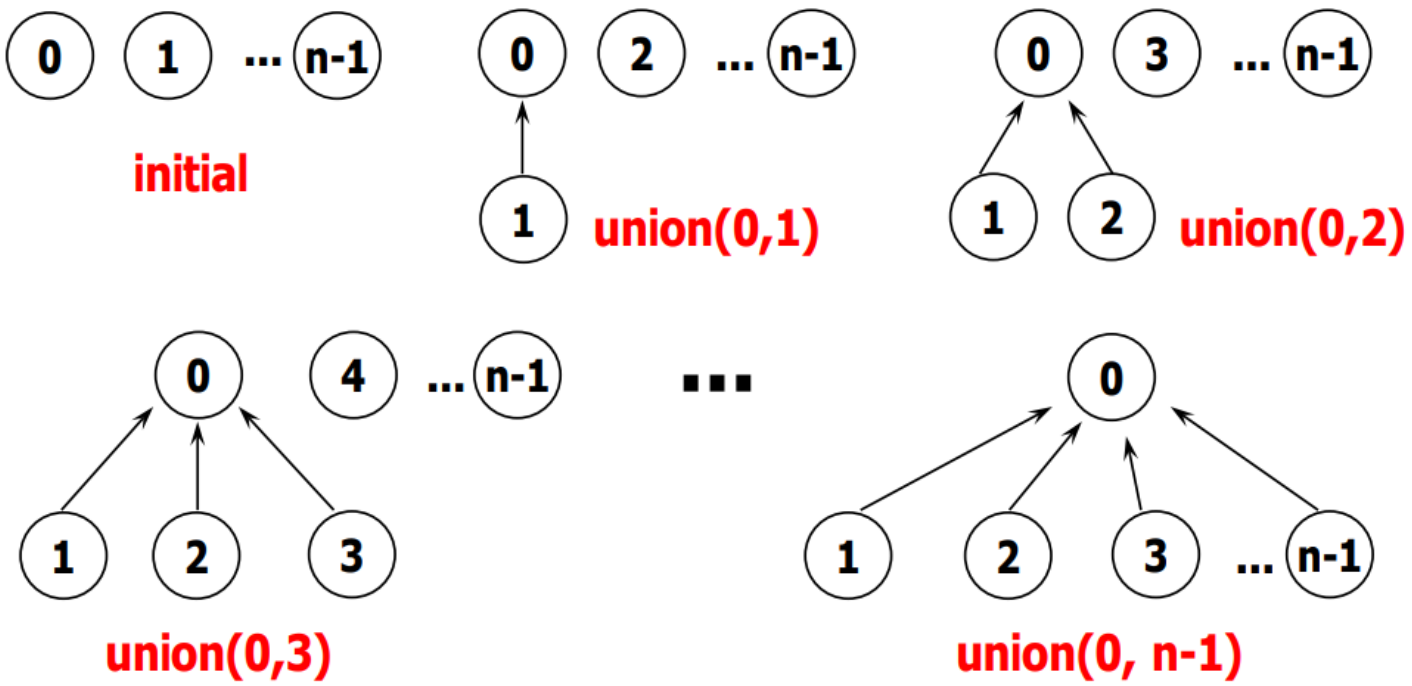
union(0, 1), union(1, 2), ..., union(n-2, n-1)



참고)Weighted Union

Simple Union의 문제점 해결 방법 : 큰 쪽에 몰아주자

- i를 가진 트리의 노드 수와 j를 가진 트리의 노드 수 비교
  - (i 트리의 노드 수 < j 트리의 노드 수) → j가 i의 부모
  - Otherwise → i가 j의 부모
  - ☞ Union-by-rank



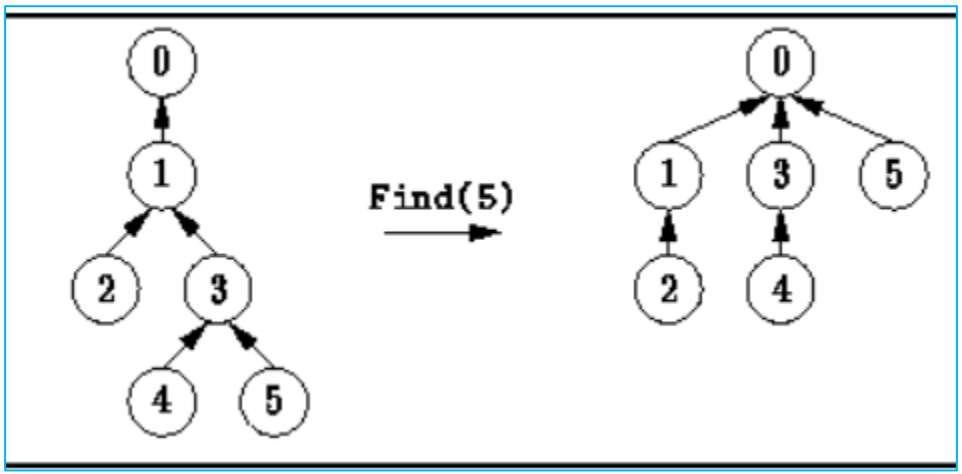
참고)Collapsing Find

Simple Find의 개선 : leaf로부터 타고 올라가는 노드들을 Root 밑에 바로 붙이자

정의 [**Collapsing Rule – 붕괴 규칙**]

- **find(i)**의 실행 과정에 만나는 **root**가 아닌 노드들을 **root**의 **child**로 변경

☞ Path compression



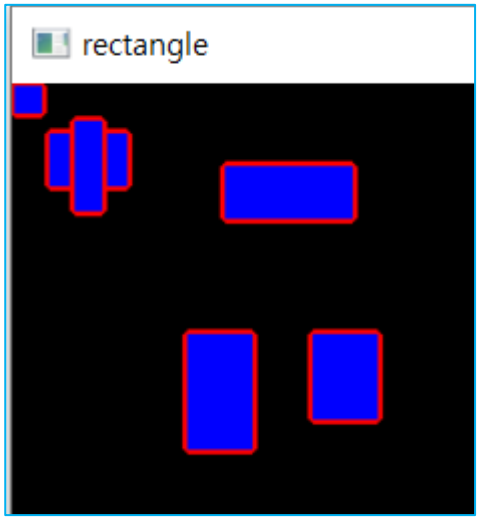
https://book.huihoo.com/data-structures-and-algorithms-with-object-oriented-design-patterns-in-c++/html/page410.html

Code

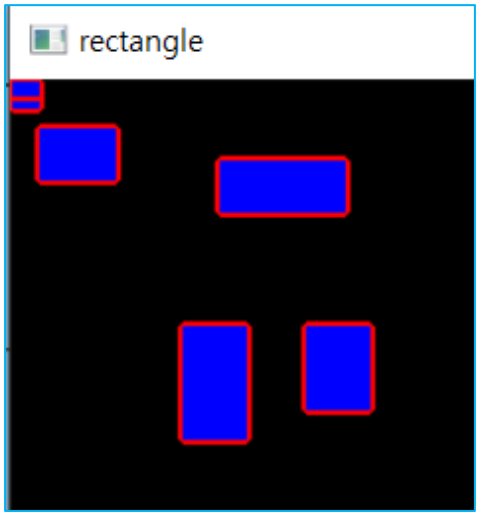
```
for i in range(len(L)):
    for j in range(i + 1, len(L)):
        R1, R2 = L[i], L[j]
        if is_intersect(R1, R2):
            root_left = collapsing_find(R, i)
            root_right = collapsing_find(R, j)

            if root_left != root_right:
                weighted_union(R, root_left, root_right)
```

결과

```
L1 = [Rectangle(1, 1, 10, 10), Rectangle(15, 20, 30, 20), Rectangle(25, 15, 10, 35),
     Rectangle(70, 100, 25, 45), Rectangle(85, 33, 50, 20), Rectangle(120, 100, 25, 33)]

L2 = [Rectangle(1, 1, 10, 10), Rectangle(12, 20, 30, 20), Rectangle(1, 1, 10, 5),
     Rectangle(70, 100, 25, 45), Rectangle(85, 33, 50, 20), Rectangle(120, 100, 25, 33)]
```



[-1, -1, -1, -1, -1, -1]
[-1, -2, 1, -1, -1, -1]
{0: [0], 1: [1, 2], 3: [3], 4: [4], 5: [5]}



[-1, -1, -1, -1, -1, -1]
[-2, -1, 0, -1, -1, -1]
{0: [0, 2], 1: [1], 3: [3], 4: [4], 5: [5]}

Summary

- 문제 4.11 : 사각형이 겹치는지 확인하기
  - ✓ 거꾸로 생각해서 조건 찾기
    - 사각형이 겹치지 않는 조건 -> 사각형이 겹치는 조건
  - ✓ Grouping(Union-Find)
    - Simple union, Simple find
    - Weighted union, Collapsing find

들어 주셔서 감사합니다