선택 트리
Union-Find 알고리즘

강 지 훈

jhkang@cnu.ac.kr

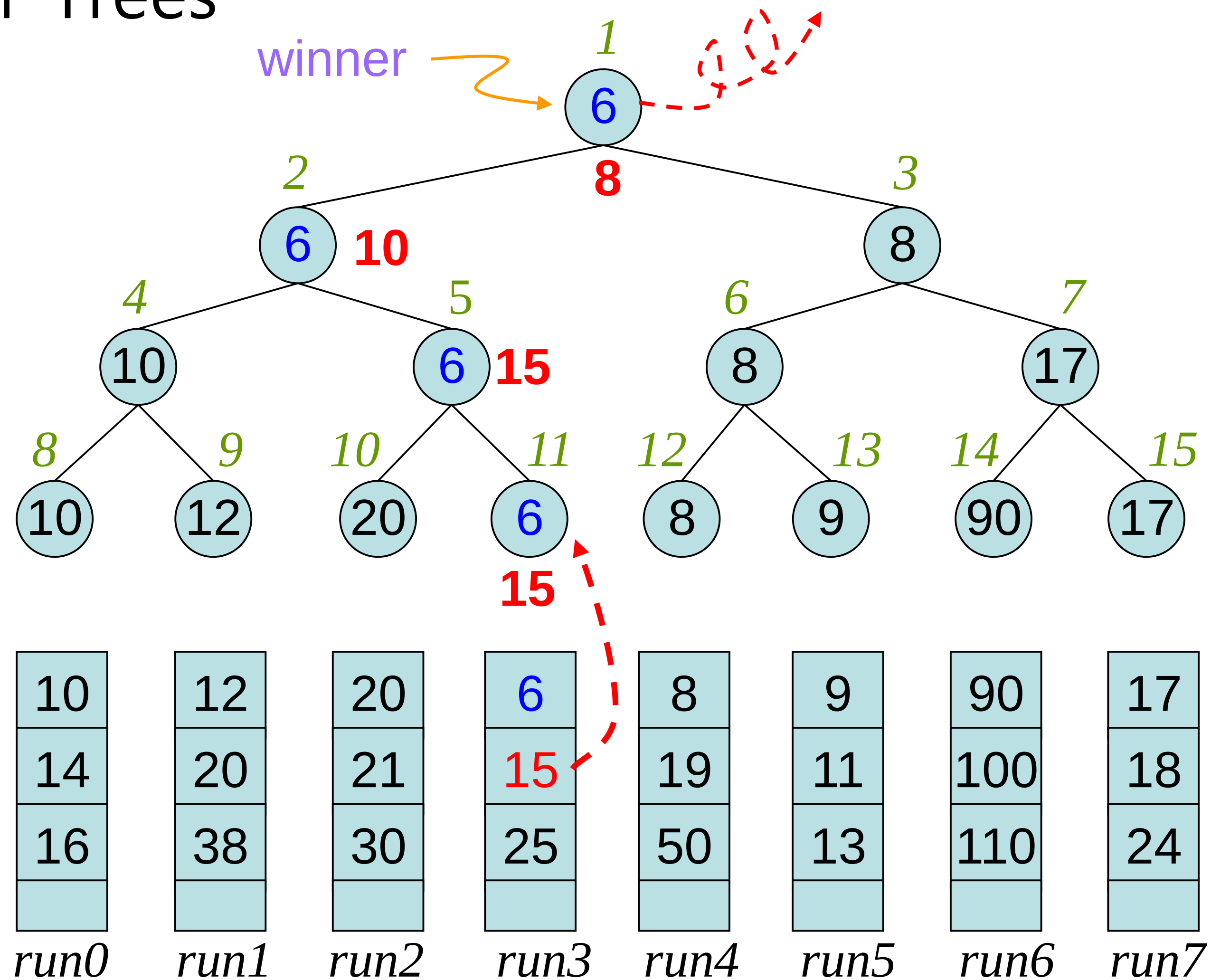
충남대학교 컴퓨터공학과

Selection Trees

Winner Trees
Loser Trees

Selection Trees

Winner Trees



Winner Tree의 구현

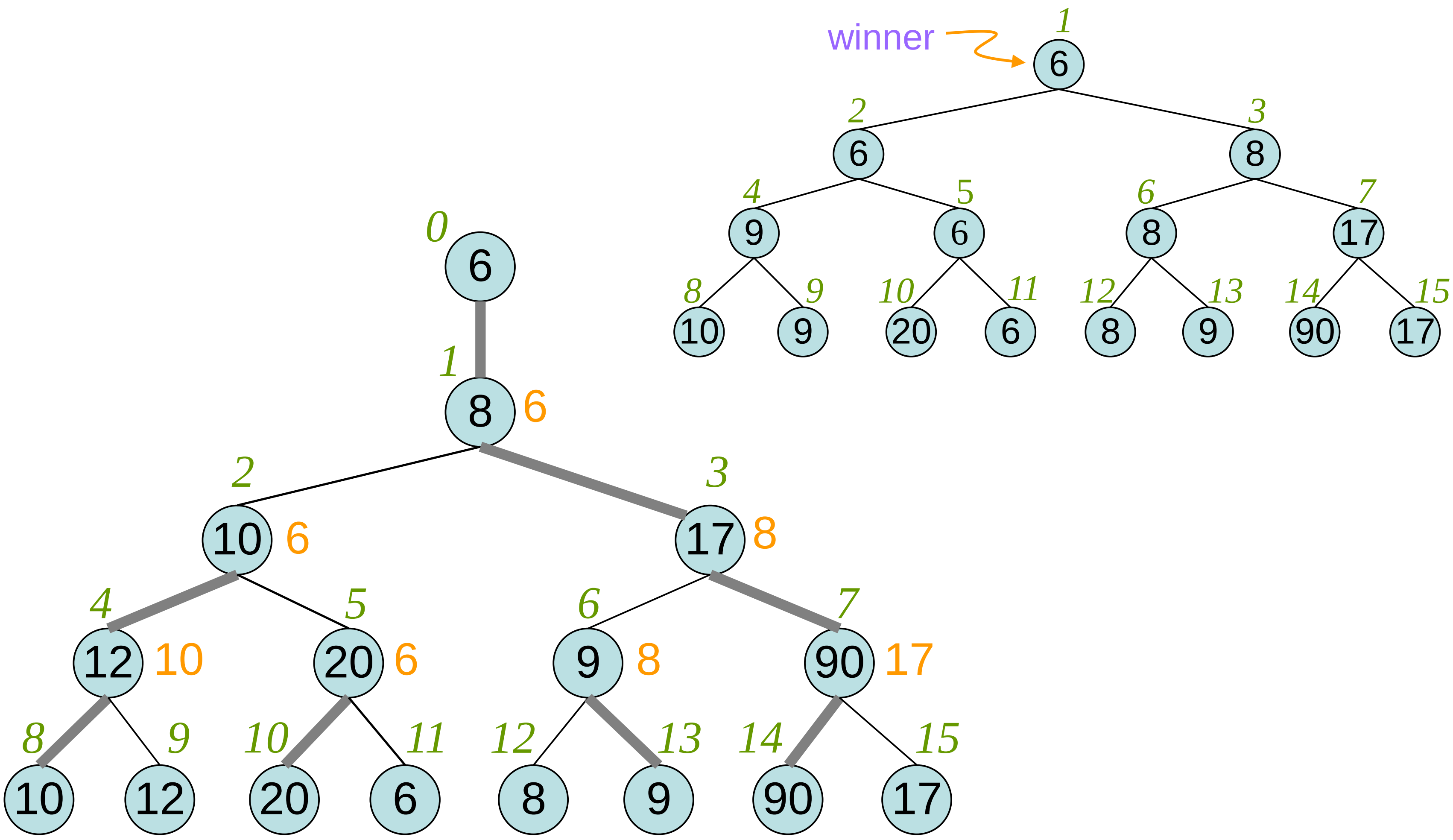
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	3	3	4	0	3	4	7	0	1	2	3	4	5	6	7

0	1	2	3	4	5	6	7
10	12	20	6	8	9	90	17

run0	run1	run2	run3	run4	run5	run6	run7
10	12	20	6	8	9	90	17
14	20	21	15	19	11	100	18
16	38	30	25	50	13	110	24

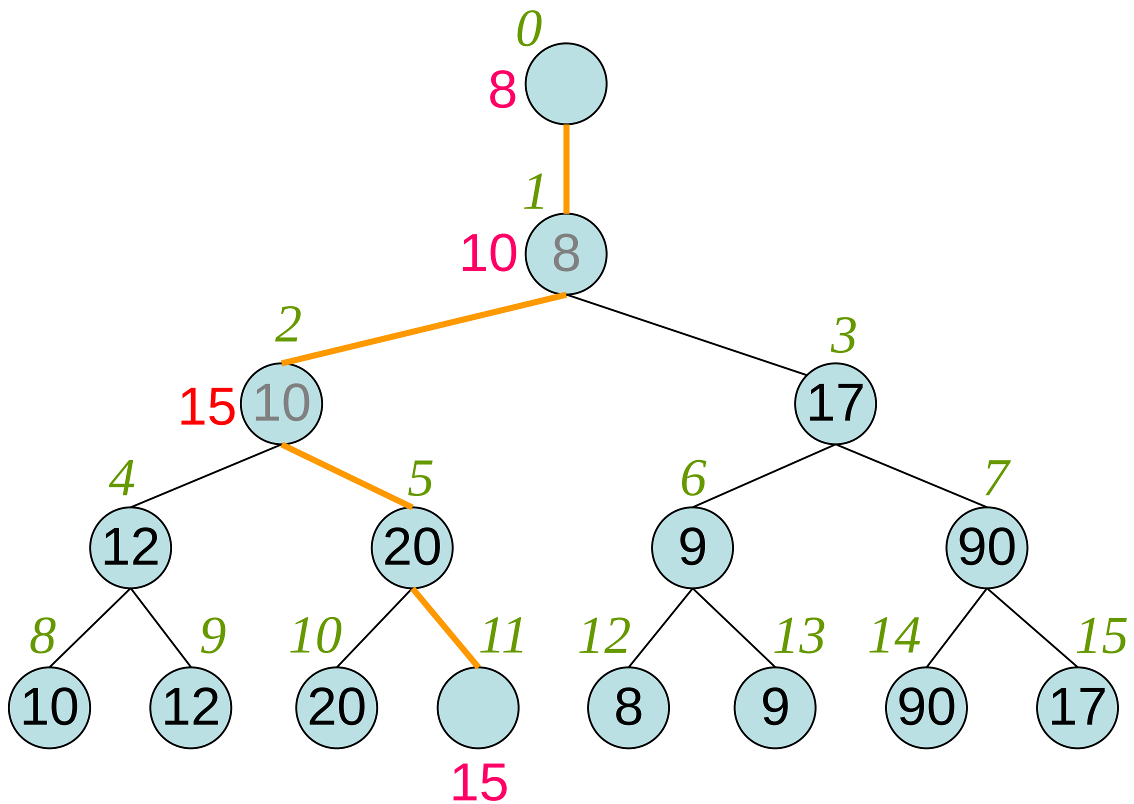
Loser Trees

- 초기화



Loser Tree

- 비교는 부모노드와만 하면 된다.



Pair-wise Disjoint Sets
&
Union-Find Algorithm

Pair-wise Disjoint sets

- 어떠한 집합도 다른 집합과 겹치는 원소를 가지고 있지 않다. (pair-wise disjoint sets)
  - 따라서, 어떠한 원소도 단 하나의 집합에만 속해 있다.

- Example:
  - S1 = {0, 6, 7, 8}
  - S2 = {1, 4, 9}
  - S3 = {2, 3, 5}

❑ 두 개의 기본 연산

- Union(Si, Sj): 겹치는 원소가 없는 두 집합을 합한다.
  - Union(S1, S2) = { 0, 6, 7, 8, 1, 4, 9 }
- Find(i): 원소 i 가 속해있는 집합을 찾는다
  - Find(3) = S3
  - Find(8) = S1

S1 = {0, 6, 7, 8}
S2 = {1, 4, 9}
S3 = {2, 3, 5}

- 하나의 집합을 다른 집합과 구분할 수만 있으면 충분하다.
  - 집합의 이름은 중요하지 않다.
  - 집합 자체가 중요.

DisjointSets 의 공개함수

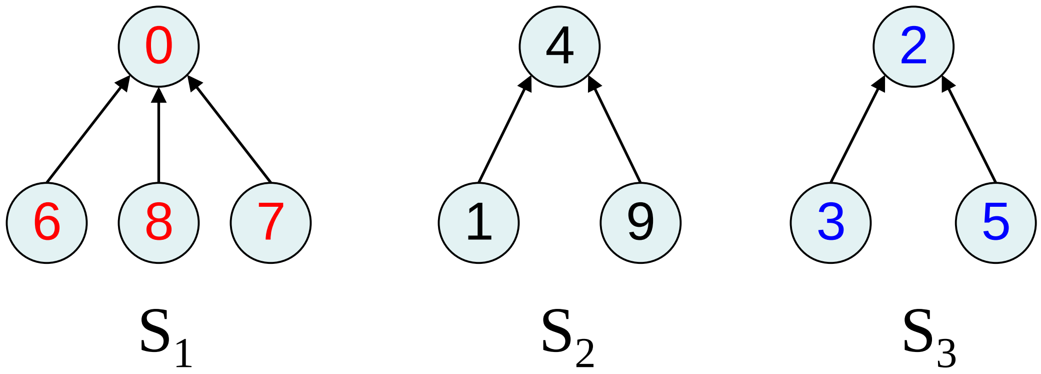
- DisjointSets (int numberOfElements) ; // 생성자
  - 원소의 개수가 주어진다.
  - 맨 처음에는 각각의 원소가 하나의 집합이다. 그러므로 초기에는 원소의 개수만큼 집합이 존재하게 된다.

- void union (SetID set1, SetID set2) ;
  - 두 집합을 하나의 집합으로 합한다.

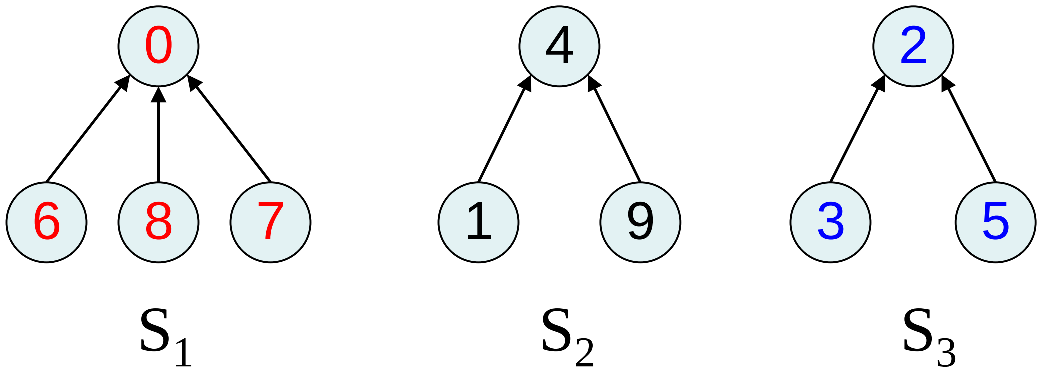
- SetID find (Element e) ;
  - 주어진 원소를 갖는 집합을 찾는다.

Union-Find 알고리즘을 이용한 "DisjointSets" 의 구현

❑ 가능한 표현 방법



❑ 표현 방법:



- 배열을 사용
  - 각 원소는 부모 원소를 가리킨다.
  - 루트는 값으로 '-1'을 갖는다.
  - 루트의 레이블, 즉 루트 원소의 배열 인덱스는 집합의 이름으로 사용된다.

S1 → 0　　　S2 → 4　　　S3 → 2

0	1	2	3	4	5	6	7	8	9
-1	4	-1	2	-1	2	0	0	0	4

Class DisjointSets의 비공개 변수들

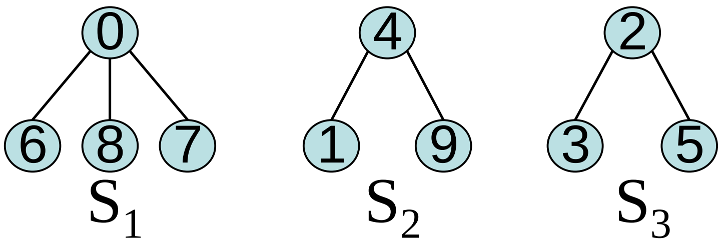
```
public class DisjointSets {
    // 비공개 인스턴스 변수
    private int _numberOfElements ;
            // DisjointSets 객체가 다룰 수 있는 원소의 개수
    private int _parent[] ;
            // 각 원소의 부모를 가리키는 정보를 저장하는 배열
            // _numberOfElements 크기로 생성되어야 한다.
```

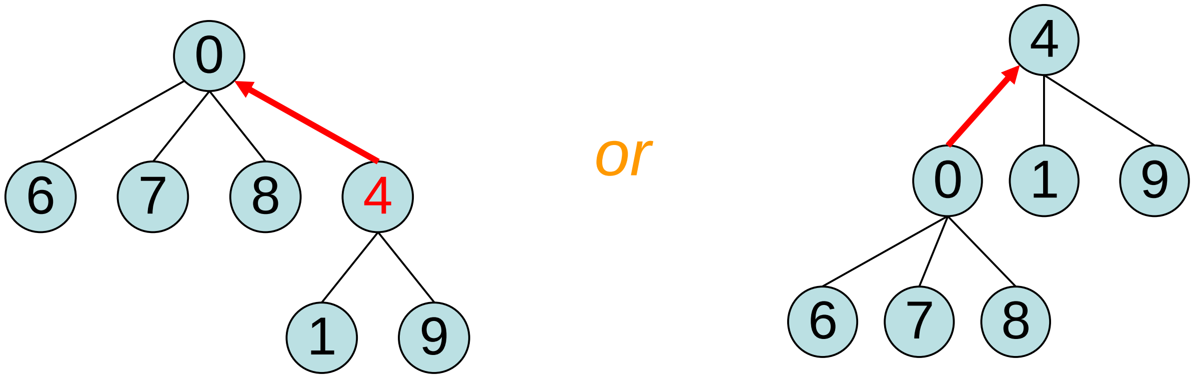
❑ 생성자

```
public class DisjointSets {
    ……
    // 생성자
    public DisjointSets (int givenNumberOfElements)
    {
        this._numberOfElements = givenNumberOfElements ;
        this._parent = new int[givenNumberOfElements] ;
    }
```

Union

- Union(S_1, S_2)
  = { 0, 6, 7, 8, 1, 4, 9 }





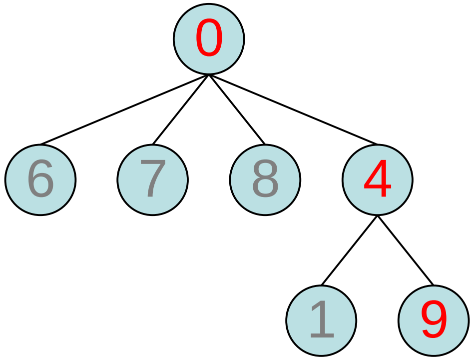
0	1	2	3	4	5	6	7	8	9
-1	4	-1	2	0	2	0	0	0	4

```
void union1 (int i, int j)
{
    this._parent[i] = j ; // or, this._parent[j] = i
}
```

Find

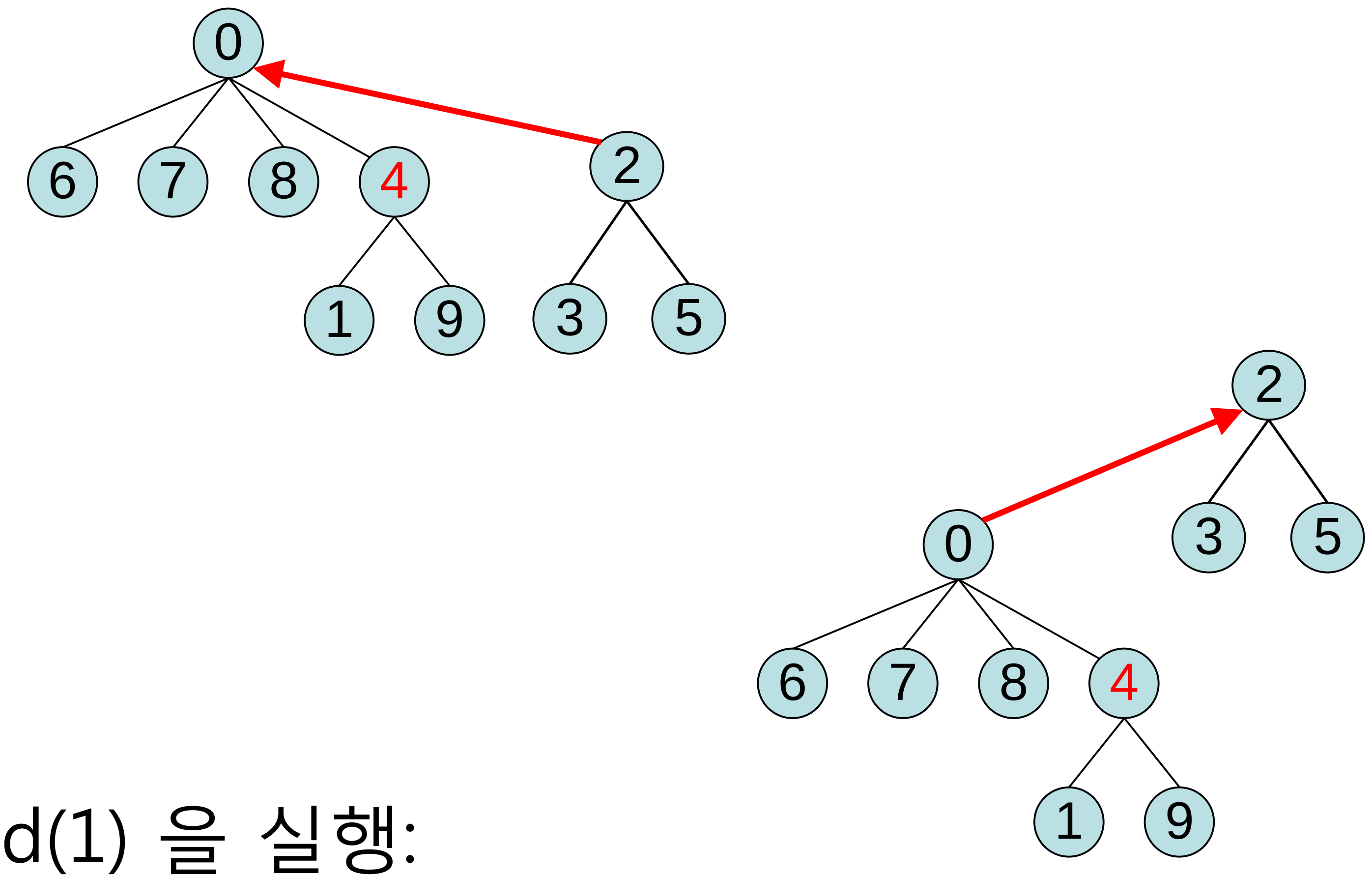
- Find (9) = 0

```
int find1 (int i)
{
   while (this._parent[i] >= 0 ) {
      i = this._parent[i] ;
   }
   return i ;
}
```



0	1	2	3	4	5	6	7	8	9
-1	4	-1	2	0	2	0	0	0	4

Union 한 후에 트리의 높이는?

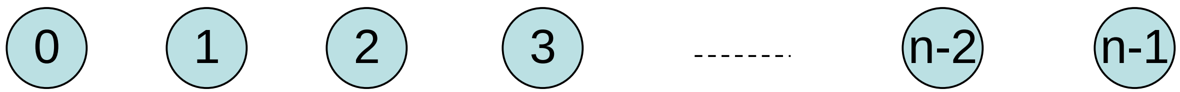


- Find(1) 을 실행:
  - 어느 쪽의 결과가 더 좋은가?

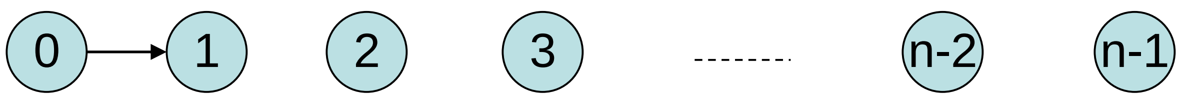
무엇이 문제인가?

union1() and find1()의 분석

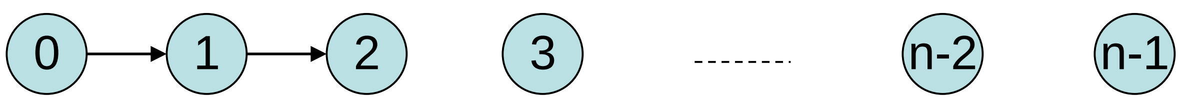
- 초기화: $s_i = \{\ i\ \},\ 0 \le i < n$.



- 처리 순서:
  - Union(0,1), find(0),

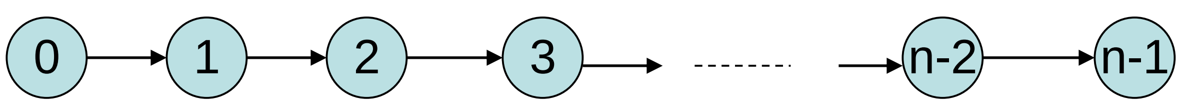


  - Union(1,2), find(0),



  ……

  - Union(n-2, n-1), find(0)



단순한 방법의 비효율성

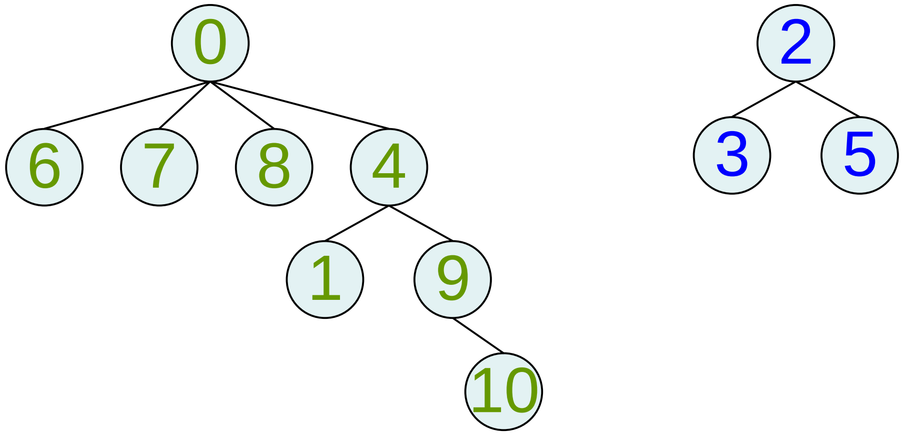
- 분석
  - union1(i, i+1): $c_1 = O(1)$
  - find1(0): $c_2 \cdot i = O(i)$
  - Totally,

$$\sum_{i=1}^{n-1}(c_1 + c_2 \bullet i) = O(n^2)$$

- find1()의 비효율성을 피할 방법은 ?

Weighting Rule for union(i,j)

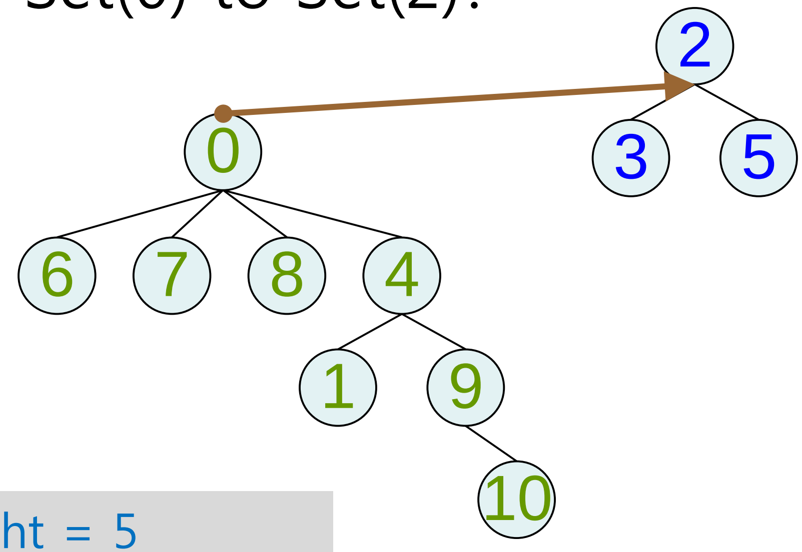
- 예:
  - Set(0): Height = 5, Number of elements = 8
  - Set(2): Height = 2, Number of elements = 3



❑ Weighting Rule for union(i,j)

- 예:
  - Set(0): Height = 5, Number of elements = 8
  - Set(2): Height = 2, Number of elements = 3

  - If we attach Set(0) to Set(2)?

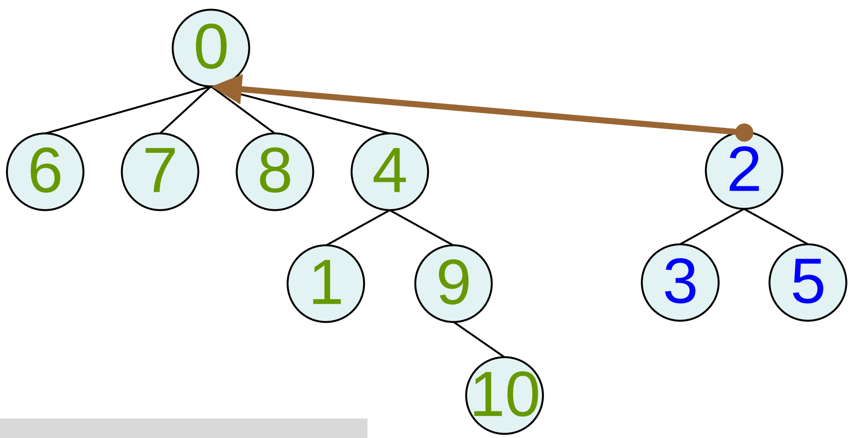


Height = 5

Number of elements = 11

Weighting Rule for union(i,j)

- 예:
  - Set(0): Height = 5, Number of elements = 8
  - Set(2): Height = 2, Number of elements = 3
  - Set(2) 를 Set(1) 에 붙인다면?

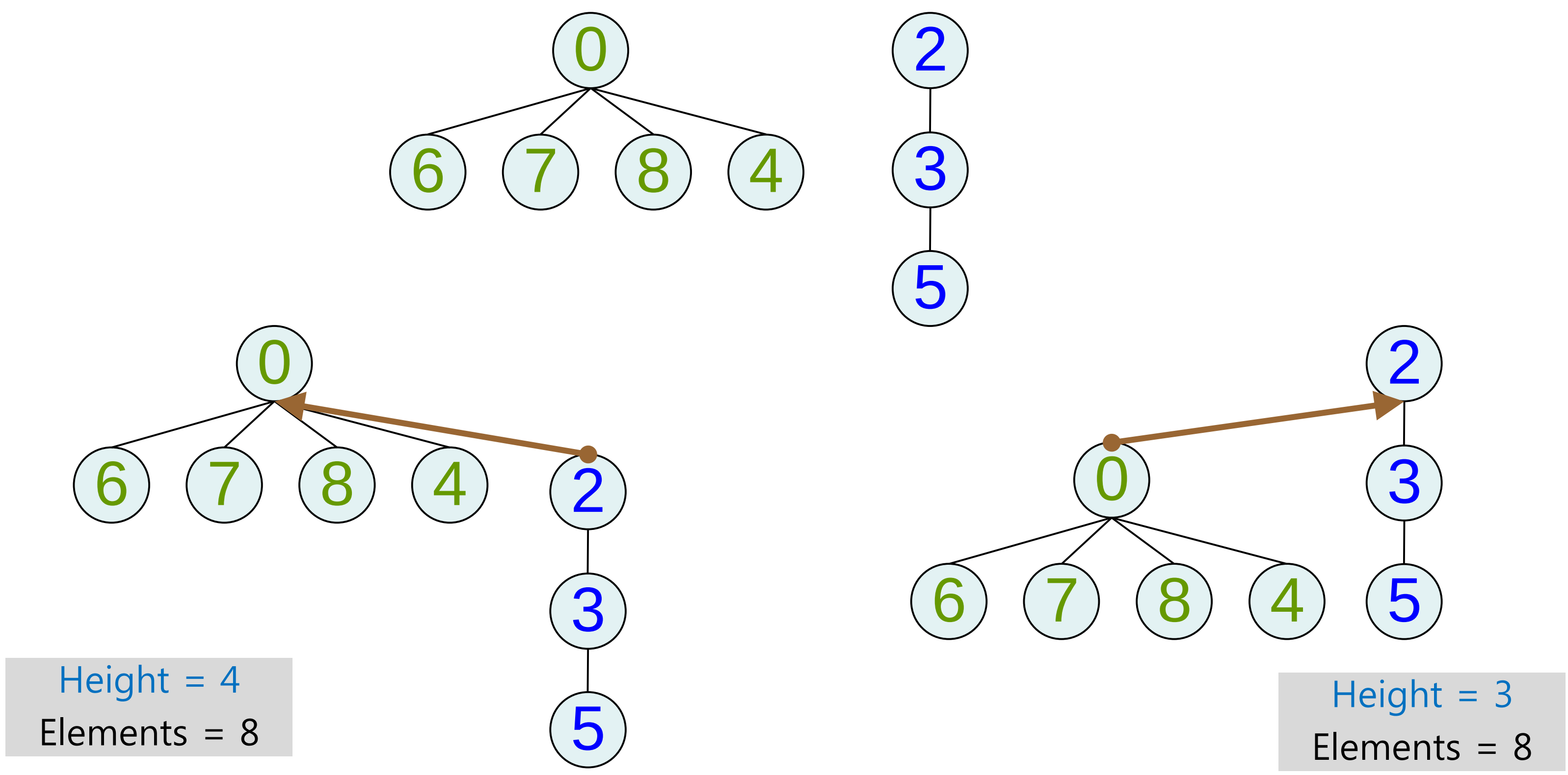


Height = 4

Number of elements = 11

❑ 예외 상황?

- 통상적이지 않은 경우:
  - Set(0): Height = 2, Number of elements = 5
  - Set(2): Height = 3, Number of elements = 3



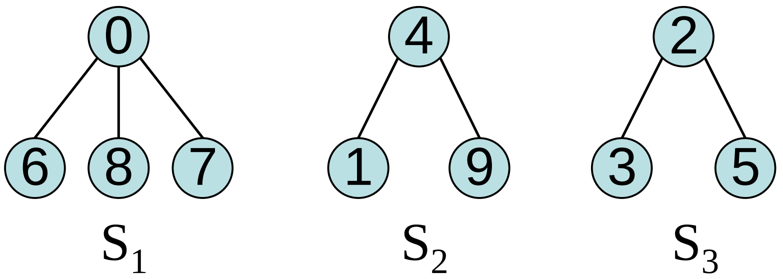
Weighting Rule for union(i,j)

- 아이디어
  - set(i) 가 set(j) 보다 더 많은 원소를 가지고 있다면, set(i) 가 set(j) 보다 키가 더 클 것이다.
  - 두 집합을 union 한 후의 집합의 높이가 증가하지 않으면 좋을 것이다.
  - 그러므로, 개수가 작은 집합의 트리를 개수가 많은 집합의 트리에 붙이자.
- Weighting Rule:
  - If (#i < #j), then { j 를 i 의 부모로 만든다 }
  - If (#i ≥ #j), then { i 를 j 의 부모로 만든다 }
    - #i: 집합 i 의 원소의 개수
    - #j: 집합 j 의 원소의 개수

Weighting Rule for union(i,j)

- 구현
  - 각 root i 는 -1 대신에 #i 의 음수 값을 갖게 한다.

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




```
void union2 (int i, int j)
{
    if (this._parent[i] >= this._parent[j])  { // #i < #j
         this._parent[i] = j ;
    }
    else { // #i >= #j
          this._ parent[j] = i ;
    }
}
```

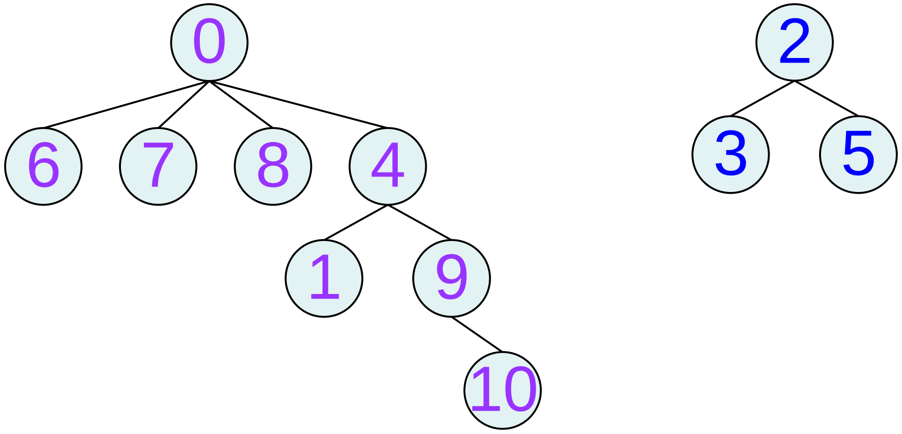
Union2()의 분석

- Lemma
  - T는 union2() 의 결과로 만들어진 노드의 개수가 n인 트리라고 하자.
  - 그러면, no node in T의 어떠한 노드도 그 레벨이 ($\lfloor \log_2 n \rfloor + 1$) 보다 크지 않다

- 분석
  - 최악의 경우
    - (n-1) unions
    - m finds
    - Totally, $O(n + m \cdot \log_2 n)$

Collapsing Rule for find(i)

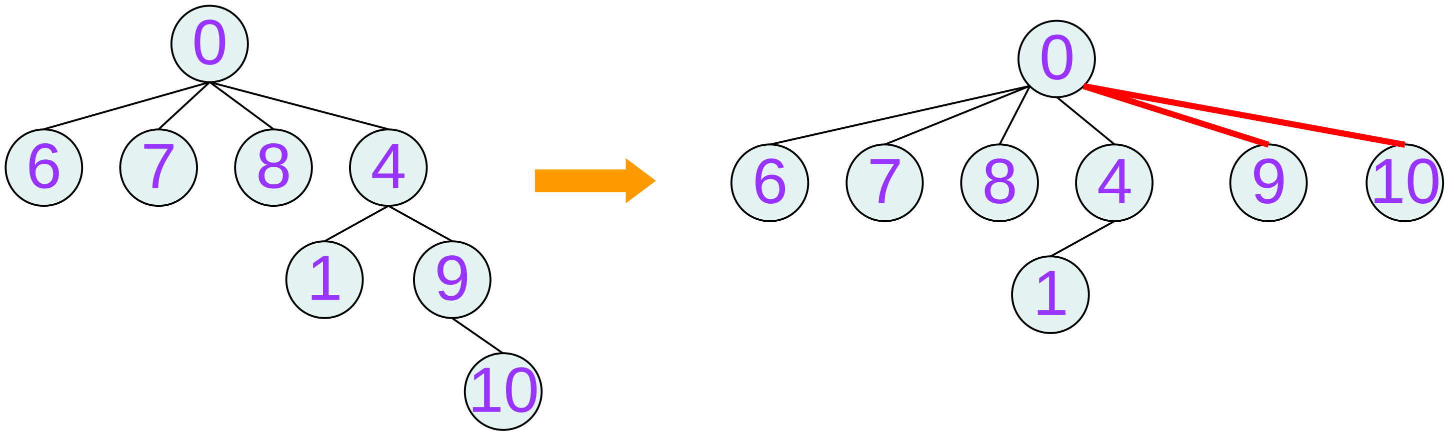
- 아이디어
  - 트리의 키를 될 수 있는대로 높지 않게 유지한다면 find() 할 때 유리할 것이다.
- Rule
  - j 가 노드 i 부터 루트까지의 경로 상의 노드라면, j 를 root의 자식 노드로 만든다.



0	1	2	3	4	5	6	7	8	9	10
-8	4	-3	2	0	2	0	0	0	4	9

find2(i) 의 구현

- 주어진 노드 i의 루트를 찾은 후에, i로부터 루트까지의 경로를 다시 스캔 하면서, 경로 상의 모든 노드를 루트의 자식노드로 만든다.
- 예: find2(10) = 0



0	1	2	3	4	5	6	7	8	9	10
-8	4	-3	2	0	2	0	0	0	0	0

❑ 분석

- Lemma
  - T (m,n) 은 m (≥ n)번의 find()와 (n – 1) 번의 union()이 섞여서 실행될 때 요구되는 최대 시간이라고 하자.
그러면, 어떤 양의 상수 k_1 과 k_2 에 대해,
$k_1 m\alpha(m,n) \le T(m,n) \le k_2 m\alpha(m,n)$
이 성립한다.
  - 여기서, 모든 현실 상황에서는 $\alpha(m,n) = \Theta(1)$ 이라고 가정해도 무방하다.
  - 따라서 위의 Lemma의 결과를 다음과 같이 말할 수 있다.
$\Rightarrow T(m,n) = \Theta(m\alpha(m,n)) \approx \Theta(m)$

- 다시 말하면, 통상적으로 union()보다 find()를 많이 실행하게 될 것이며, 그 경우 전체적으로 요구되는 시간 T (m,n) 은 결국 find()의 회수에 비례한다.

Ackerman's function

$$A(p,q) = \begin{cases} 2q & \text{if } p = 0 \\ 0 & \text{if } q = 0 \ \&\ p \geq 1 \\ 2 & \text{if } p \geq 1 \ \&\ q = 1 \\ A(p\text{-}1, A(p, q\text{-}1)) & \text{if } p \geq 1 \ \&\ q \geq 2 \end{cases}$$

- A(3,4) = ?
- α(m,n) = ?

Ackerman's function

- A(3,4) 는 매우 크다:

$$A(3,4) = 2^{2^{2^{\cdot^{\cdot^{2}}}}} \left.\vphantom{2^{2^{2^{2}}}}\right\} \text{65,536 twos}$$

- 실제적인 목적으로, 임의의 n에 대해 A(3,4) > $\log_2 n$ 이라고 가정해도 무방하다.
- 함수 $\alpha(m,n)$ 은 다음과 같이 정의된다.

$$\alpha(m,n) = \min\{z \geq 1 \mid A(z, 4\left\lceil \frac{m}{n} \right\rceil) > \log_2 n\}$$

- 따라서, $\alpha(m,n) \leq 3$.

- Note:
  $\alpha(m,n)$ 이 매우 천천히 증가하는 함수이지만, 그렇다고 해서 그 복잡도가 m 에 비례하는 것은 아니다.

"선택 트리"
"Union-Find 알고리즘"
[끝]