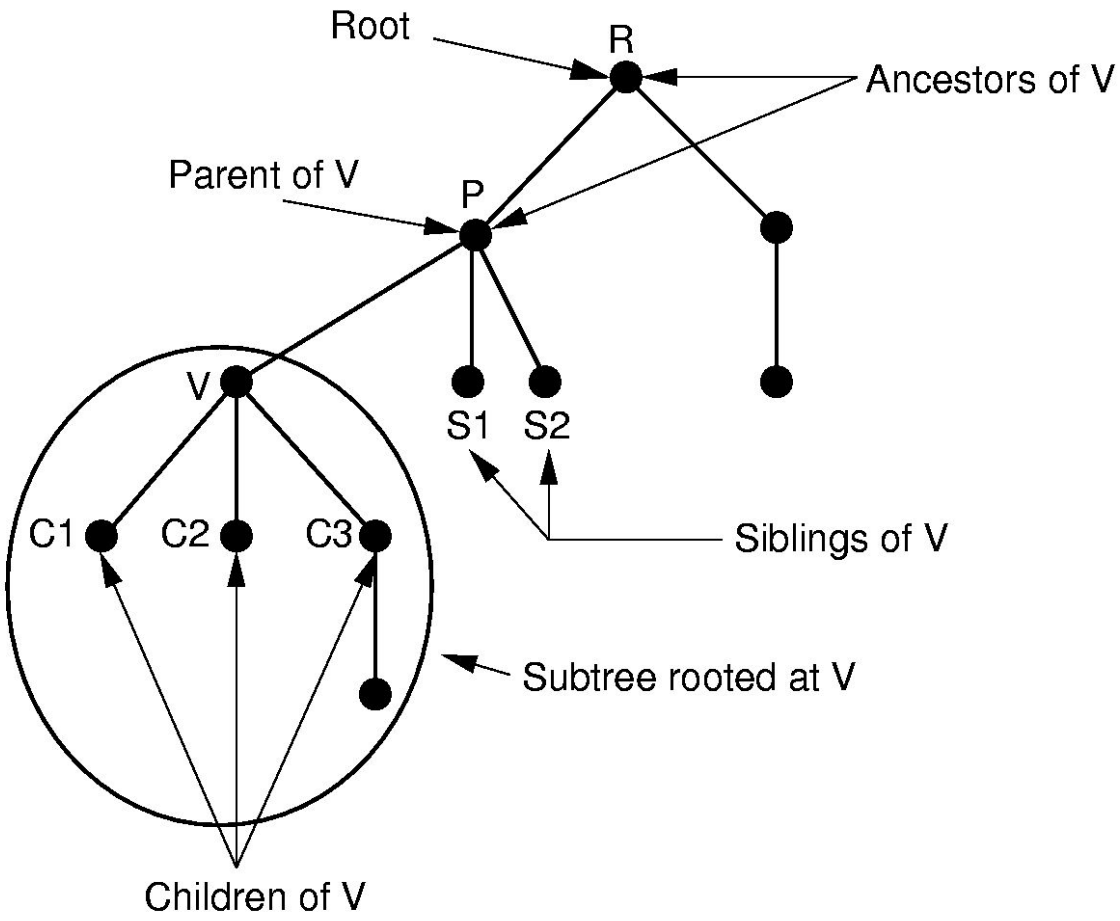
자료구조

L08 Parent Pointer Impl. of Trees

2022년 1학기

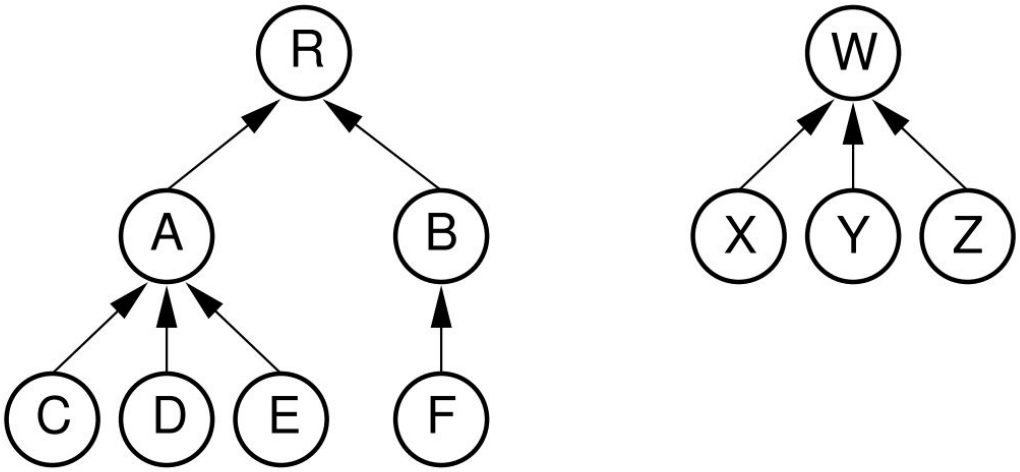
국민대학교 소프트웨어학부

General Trees



Parent Pointer Implementation

- Not for a general tree.
  - 노드의 자식들을 조회하는 등의 중요한 operation을 수행하기에는 적절하지 않음...



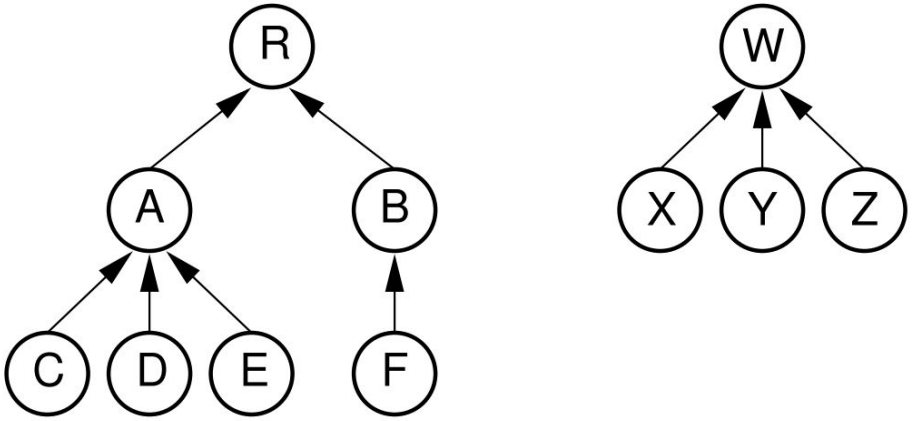
Node Index	0	1	2	3	4	5	6	7	8	9	10
Label	R	A	B	C	D	E	F	W	X	Y	Z
Parent's Index	/	0	0	1	1	1	2	/	7	7	7

Equivalence Class Problem

- Parent pointer representation은 다음 질문에 적합:
  - 두 아이템이 같은 Class에 있는가?
  - 예) BOJ 1717번 문제

```
/** Determine if nodes in different trees */
public boolean differ(int a, int b) {
  Integer root1 = FIND(a);
  Integer root2 = FIND(b);
  return root1 != root2;
}
```

Find function
returns the root

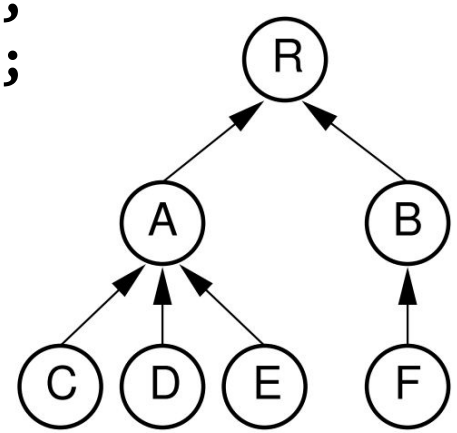


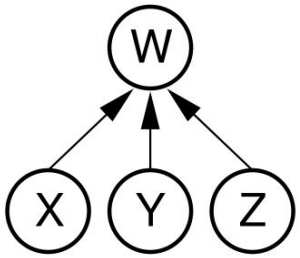
Find

```
/** return the root of the curr's tree */
public Integer FIND(Integer curr) {
  if (array[curr] == null) return curr;
  while (array[curr] != null)
    curr = array[curr];
  return curr;
}
```

array[curr]:
id가 curr인 노드의
부모 노드 id

```
/** Determine if nodes in different trees */
public boolean differ(int a, int b) {
  Integer root1 = FIND(a);
  Integer root2 = FIND(b);
  return root1 != root2;
}
```





Union/Find

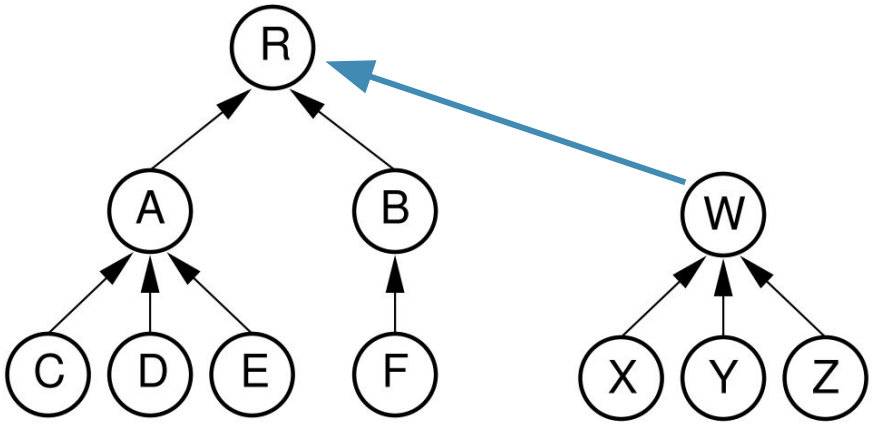
- Equivalence class problem
  - **UNION**: 두 집합을 합친다
  - **FIND**: 아이템이 어느 집합에 있는지 확인한다
    → 두 아이템이 같은 집합에 있는지 확인한다
- 응용
  - Graph connectivity
  - Clustering
  - Disjoint-set
  - Spanning tree (크루스칼 알고리즘)
- 입력 형태: 동일한 class에 속하는 아이템 쌍 목록

Implementation for Union/Find

- Equivalence class problem
  - **UNION**: 두 집합을 합친다
  - **FIND**: 아이템이 어느 집합에 있는지 확인한다
    → 두 아이템이 같은 집합에 있는지 확인한다
- **단순한 방법**
  - UNION of A and B: A ∪ B 에 속한 모든 i의 id(i) 값을 업데이트한다 → O(n)
  - FIND of i: id(i)를 리턴 → O(1)
- **Parent Pointer Implementation을 사용한 방법**
  - UNION of A and B: O(FIND(i)) → O(log n)
  - FIND of i: O(log n). Path compression을 사용한다면 대부분의 경우 O(1)

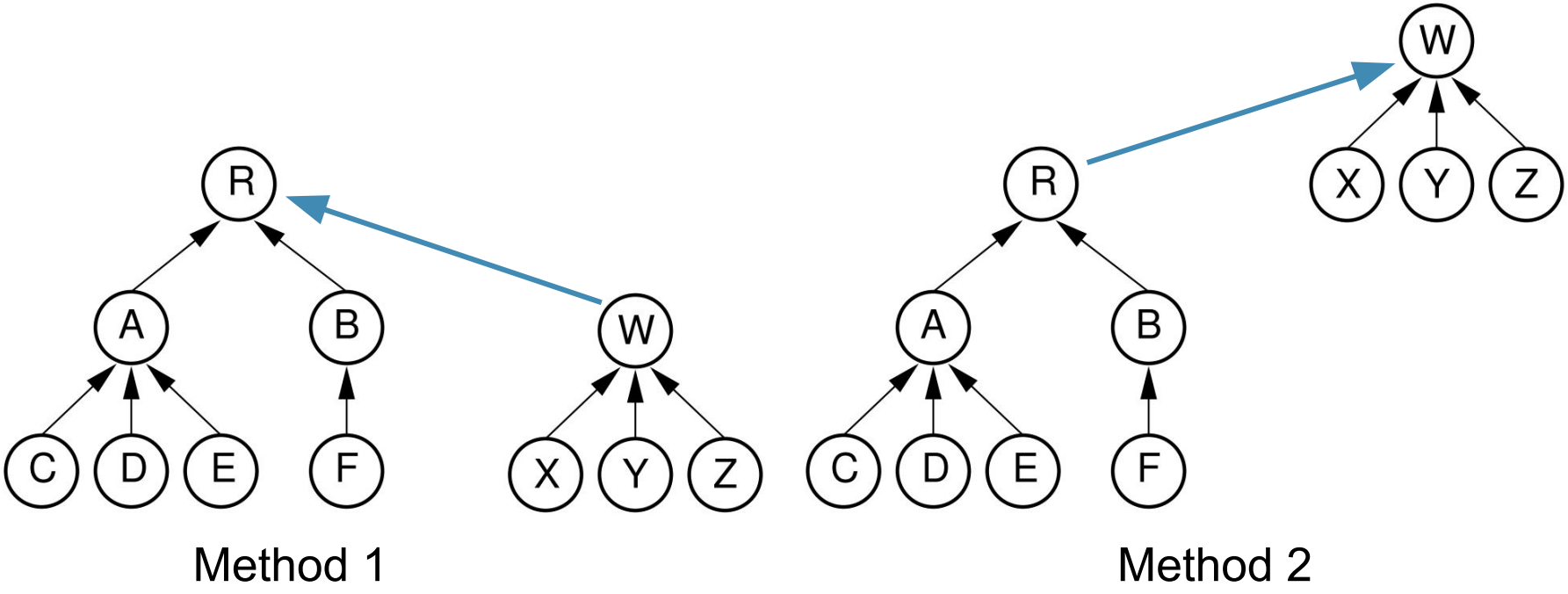
Union

```
/** Merge two subtrees */
public void UNION(int a, int b) {
  Integer root1 = FIND(a); // Find a's root
  Integer root2 = FIND(b); // Find b's root
  if (root1 != root2) array[root2] = root1;
}
```



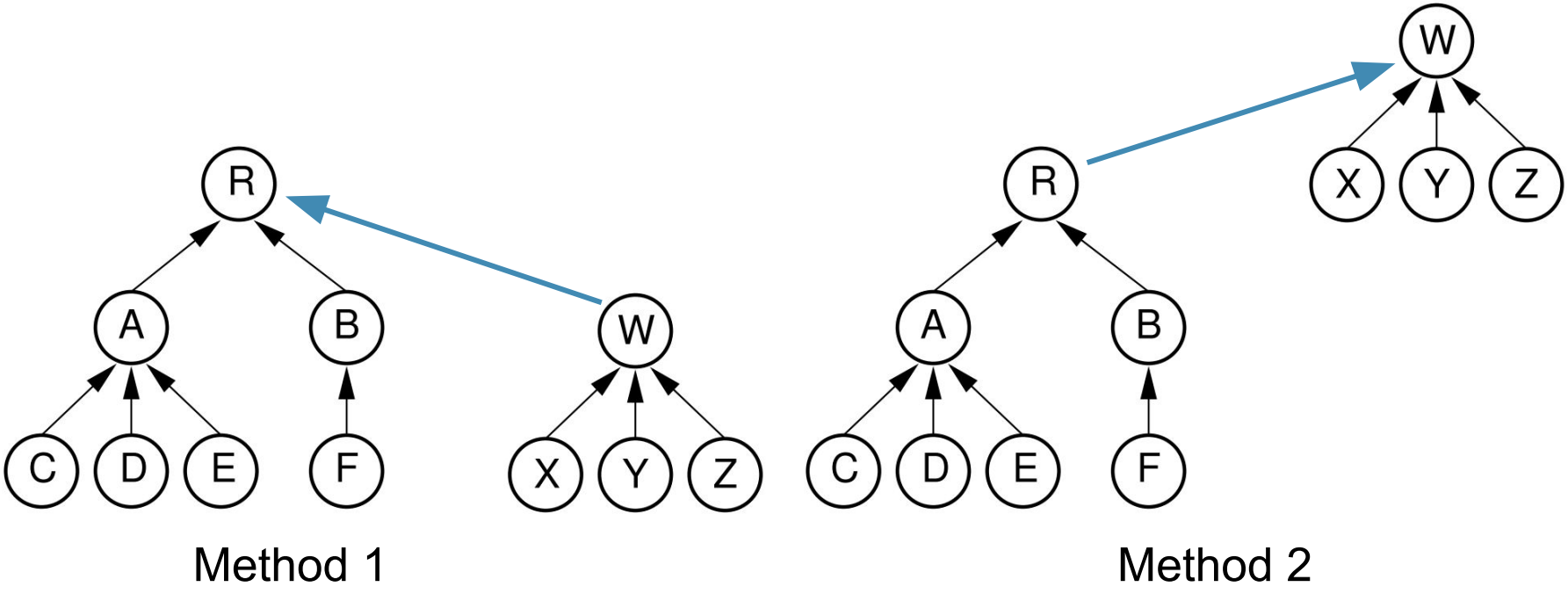
Union

- Union 할 때, 노드의 깊이를 최소로 유지하고 싶다. (Why?)
- 다음 두 가지 Union 방법 중에 무엇이 더 나을까? Why?



Union

- Weighted union rule: 노드 수가 적은 tree를 노드 수가 많은 tree에 붙인다.
  - Tree의 Height가 작게 유지되는 효과!

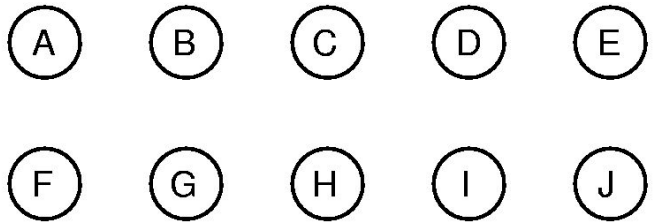


Union

- (Theorem) Assume n nodes in n independent equivalent classes. Performing Union operations in any order using weighted union rule results in a tree with depth at most log n.
- (Proof)
  - Consider a node v with the maximum depth in the final tree. Initially, v's depth was 0. v's depth increased only when the subtree A containing v is merged with another tree B, and |A| < |B|. Then,
    v's final depth = (# of times v's depth increased) <= log n.

Equiv. Class Processing

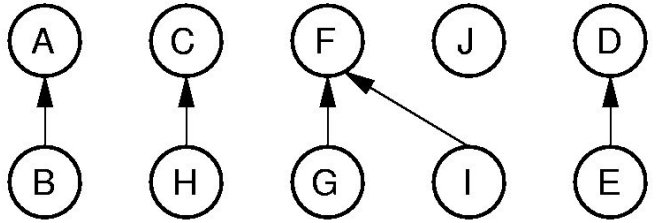
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



(a)

Equivalence Relation
(A,B), (C,H), (G,F), (D,E), (I,F)

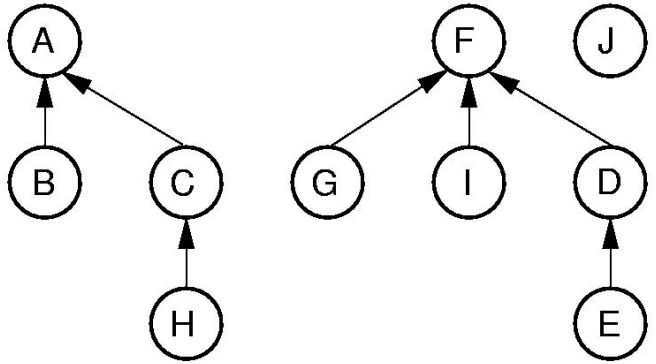
	0			3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



(b)

Equivalence Relation
(H,A), (E,G)

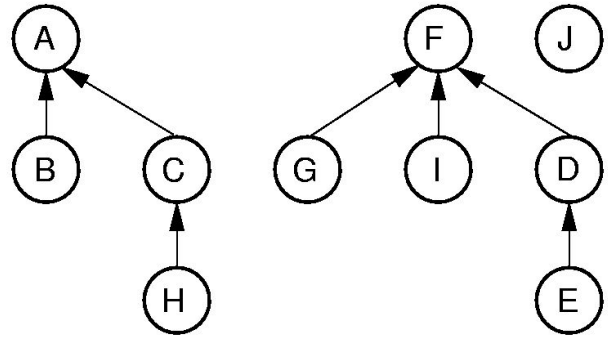
	0	0	5	3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



(c)

Equiv. Class Processing

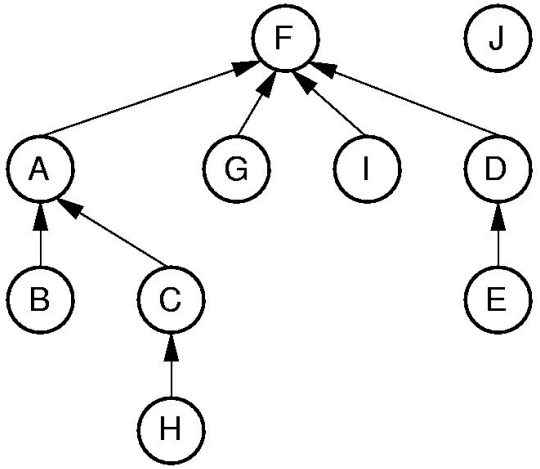
	0	0	5	3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



(c)

Equivalence Relation
(H,E)

5	0	0	5	3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9

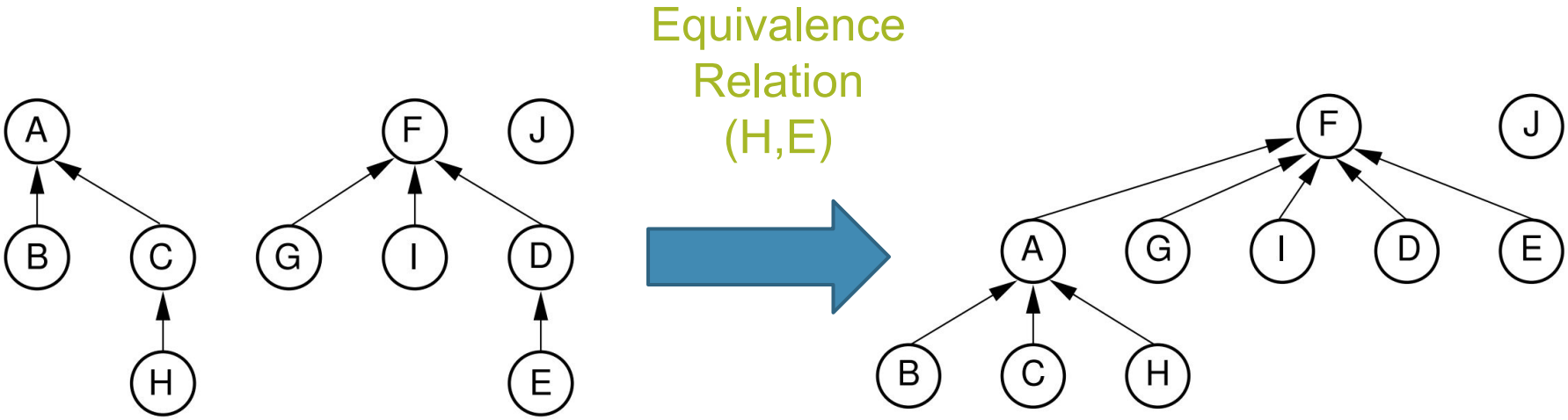


Method 1

(d)

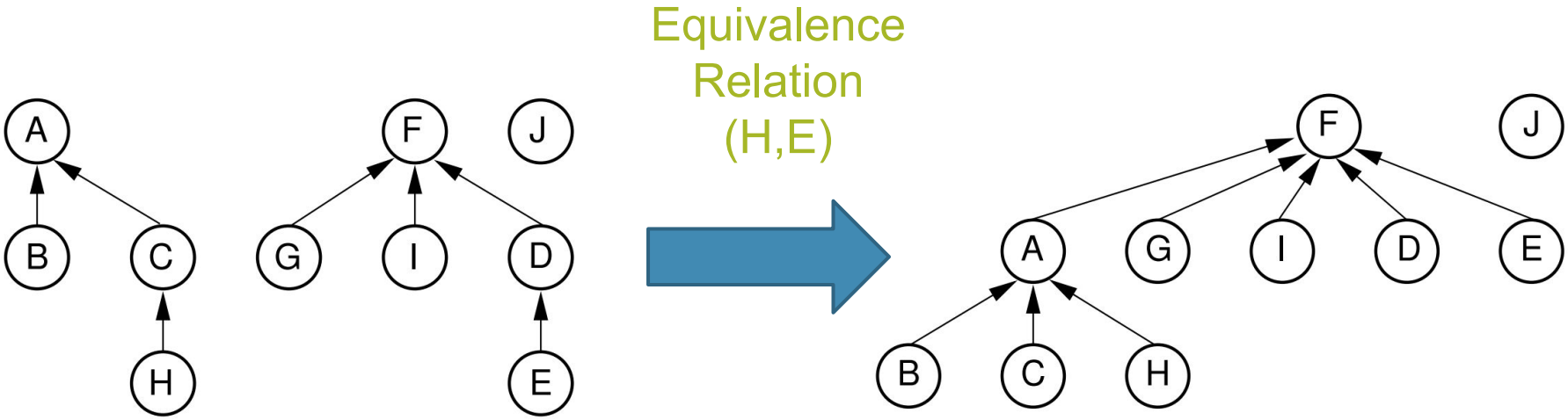
Path Compression

- 트리의 Height를 더 줄일 수는 없을까?
  - **Path Compression**:
    FIND 연산 중에, path 상의 모든 노드를 root 노드에 연결



Path Compression

```
/** FIND for path compression */
public Integer FIND(Integer curr) {
  if (array[curr] == null) return curr;
  array[curr] = FIND(array[curr]);
  return array[curr];
}
```



Questions?