1️⃣

알고리즘 : 집합의 처리

집합(set)의 처리

상호배타적 집합(disjoint set)만을 대상으로 한다. 즉 교집합 연산은 다루지 않는다.

상호배타적 집합 처리에 필요한 작업(지원하는 연산)

- Mask-Set(x) : 원소 x로만 이루어진 집합을 생성한다.
- Find-Set(x) : 원소 x가 속한 집합을 알아낸다.
- Union(x, y) : 원소 x가 속한 집합과 원소 y가 속한 집합의 합집합을 구한다.

연결리스트(Linked List)를 이용한 집합의 처리

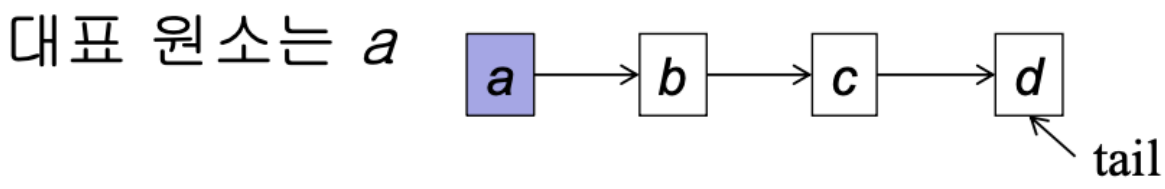
의미

각 원소 당 하나의 노드를 만들고, 같은 집합에 속한 원소들을 하나의 연결 리스트로 관리한다.

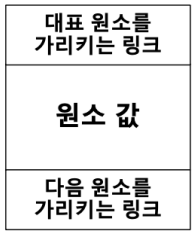
- 집합의 대표 원소 = Linked List의 맨 앞의 원소
- 마지막 원소를 가리키는 변수 tail을 가짐. (Union에 이용)

예시

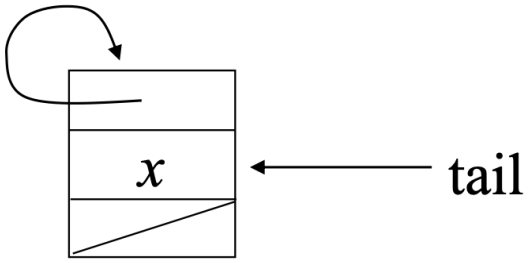
{a, b, c, d}를 표현하는 연결리스트



노드 구조



원소가 하나인 집합

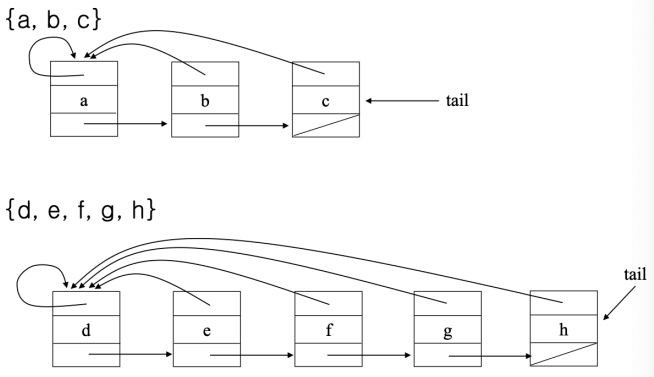


Make-Set 예시

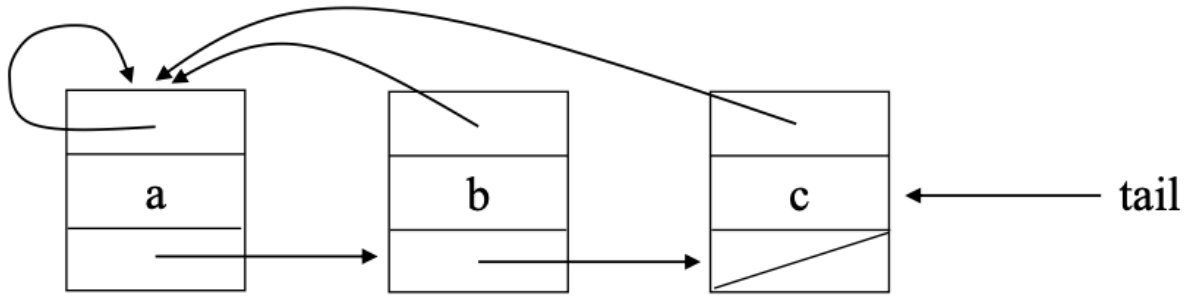
Make-Set(x) : 원소 x로만 이루어진 집합 {x}를 만드는 연산

1. 노드를 하나 만들어 원소 x를 저장한다.
2. 대표 원소 링크는 자신을 가리키도록 한다.
3. 다음 원소는 없으므로 다음 원소 링크는 NIL이 된다.

원소가 여러 개인 집합

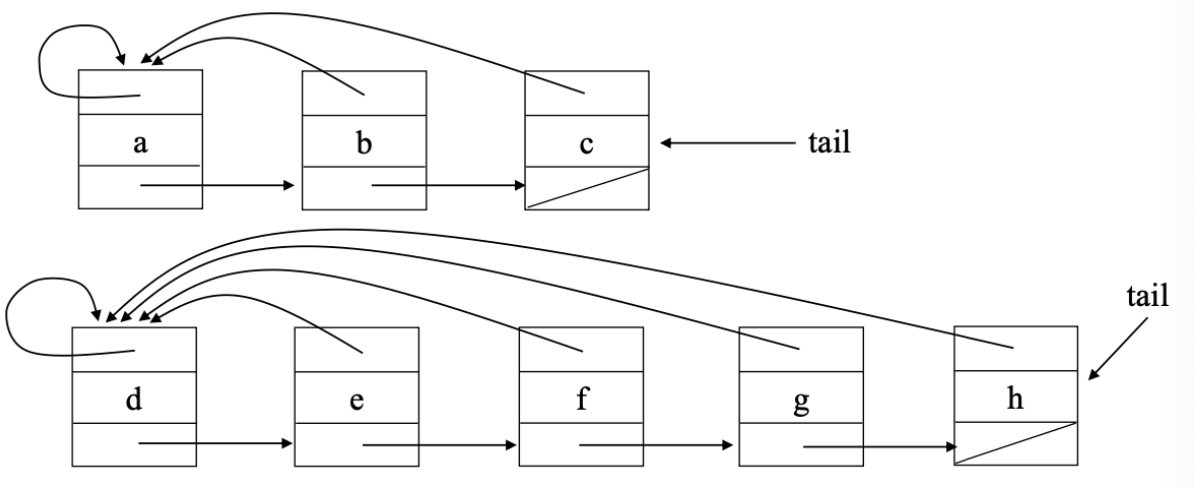


Find-Set 예시

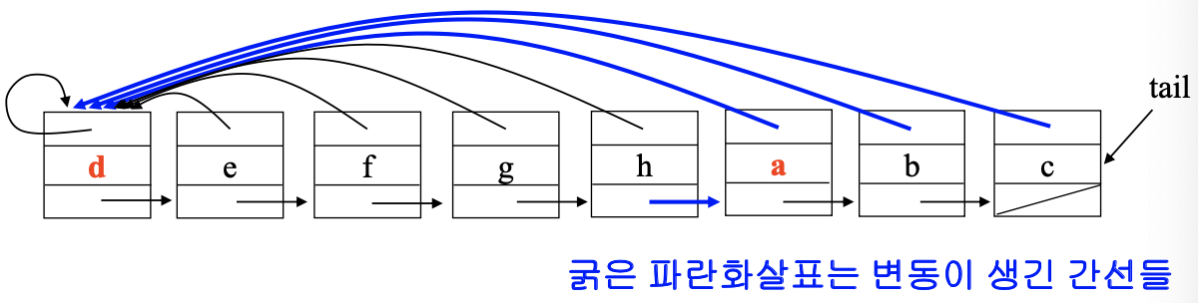


Find-Set(a) : 원소 a가 속한 집합을 알아냄 -> 대표 원소 a
Find-Set(b) : 원소 b가 속한 집합을 알아냄 -> 대표 원소 a
Find-Set(c) : 원소 c가 속한 집합을 알아냄 -> 대표 원소 a

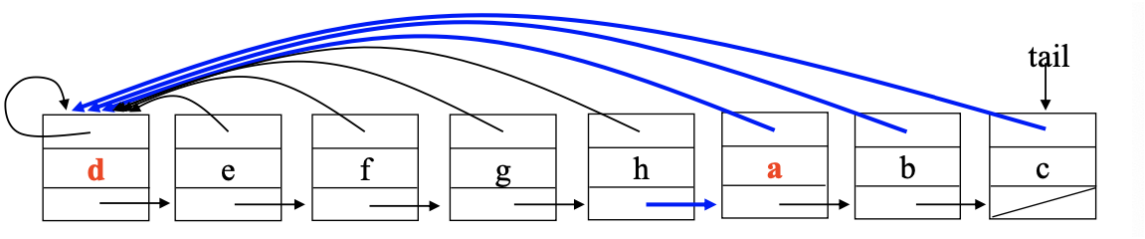
Union 예시



Union(c, g)연산 : Find-Set(c)와 Find-Set(g)를 수행해 각각의 대표 원소를 알아낸 후, 한 집합을 다른 집합의 뒤에 붙인다.

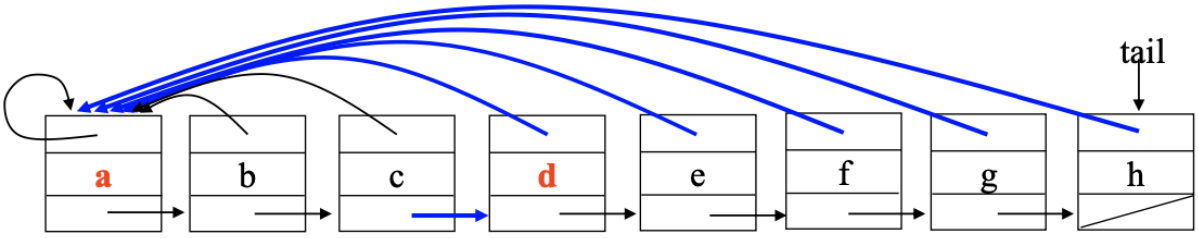


- Weighted Union



무게를 고려한 유니온 연산으로 Linked List로 된 두 집합의 합집합을 구할 때 큰 집합 뒤에 작은 집합을 붙인다.
대표 원소를 가리키는 링크 갱신 작업으로 최소화 할 수 있다.

Weighted Union이 아닌 예시



수행시간

Linked List로 표현한 배타적 집합에서 Weighted Union을 사용할 때 m번의 Make-Set, Find-Set, Union 연산 중 n번이 Make-Set이라면 **총 수행시간은 O(m+ nlogn)**이다.

증명

- Make-Set이 n번 이므로 원소의 수는 n개
- Make-Set, Find-Set은 각각 O(1)이므로 모두 합쳐도 **O(m)**이다.
- Union에서 '임의의 원소 x에 대해 대표 원소 포인터 갱신 수 ≤ log2n' 인데 x가 속한 집합은 갱신이 일어날 때마다 1 -> 2 -> 2^2 -> 2^3 -> ... 이상의 비율로 커지므로 총 **대표 원소 갱신 수 <= n*log2n**이다.
- **전체 수행시간 = O(m+ nlogn)**

트리(Tree)를 이용한 집합의 처리

의미

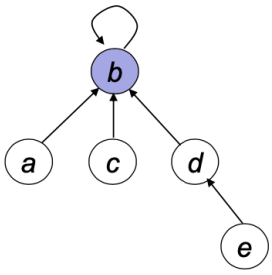
각 원소 당 하나의 노드를 만들고, 같은 집합에 속한 원소들을 하나의 트리로 관리한다.

- 일반적인 트리와 다르게 child가 parent를 가리키는 구조이다.

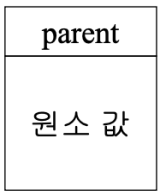
- 집합의 대표 원소 = 트리의 루트

예시

{a, b, c, d, e}를 표현하는 트리

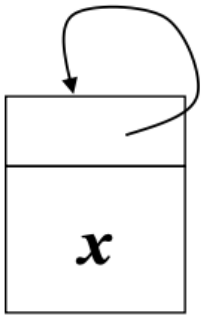


노드 구조

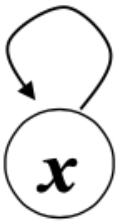


부모를 가리키는 링크

원소가 하나인 집합



간단하게 표기



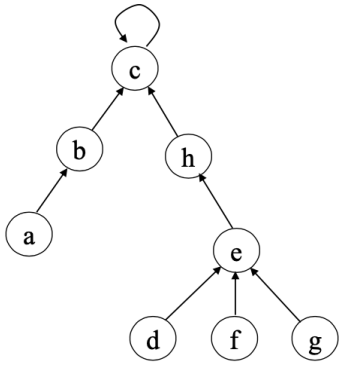
Make-Set 예시

Make-Set(x) : 원소 x로만 이루어진 집합 {x}를 만드는 연산

1. 노드를 하나 만들어 원소 x를 저장한다.
2. 부모 링크는 자신을 가리키도록 한다.

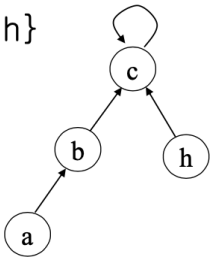
원소가 여러 개인 집합

{a, b, c, h, d, e, f, g}

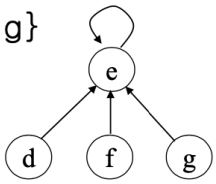


Find-Set 예시

{a, b, c, h}



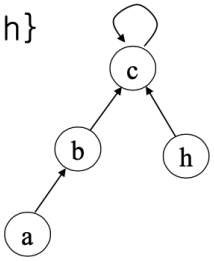
{d, e, f, g}



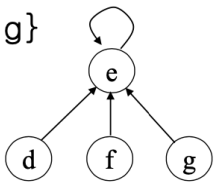
Find-Set(a) : 원소 a가 속한 집합을 알아냄 -> 대표 원소 c
Find-Set(e) : 원소 e가 속한 집합을 알아냄 -> 대표 원소 e
Find-Set(g) : 원소 g가 속한 집합을 알아냄 -> 대표 원소 e

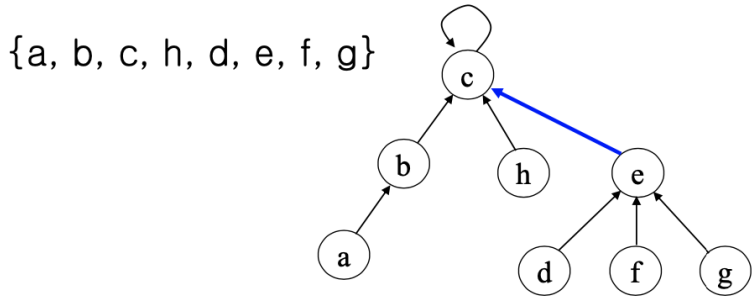
Union 예시

{a, b, c, h}



{d, e, f, g}





Tree를 이용한 집합 처리 알고리즘

Mask-Set(x)

```
Make-Set(x)       // 노드 x를 유일한 원소로 하는 집합을 만든다.
{
  x.parent <- x;
}
```

Union(x, y)

```
Union(x)        // 노드 x가 속한 집합에 노드 y가 속한 집합을 합친다.
{
  Find-Set(y).parent <- Find-Set(x);
}
```

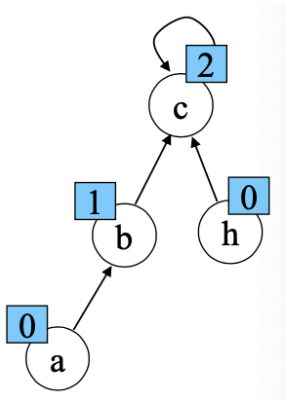
Find-Set(x)

```
Find-Set(x)       // 노드 x가 속한 집합을 알아낸다.
{           // 즉 노드 x가 속한 트리의 루트 노드를 리턴한다.
  if(x = x.parent)
      return x;
    else
      return Find-Set(x.parent);  // 재귀 사용ㅇ
}
```

연산의 효율을 높이는 방법 (= 트리의 높이를 낮추는 방법)

1. rank를 이용한 Union

Union연산을 수행해 트리가 확장될 때, 트리의 높이를 가능한 낮게 유지할 수 있도록 한다.



rank의 개념

각 노드에 랭크(rank)라는 이름의 필드를 두어 자신을 루트로 하는 서브트리의 높이를 저장한다.
하나의 노드로 이루어진 트리(서브트리)의 높이는 0으로 정의한다.

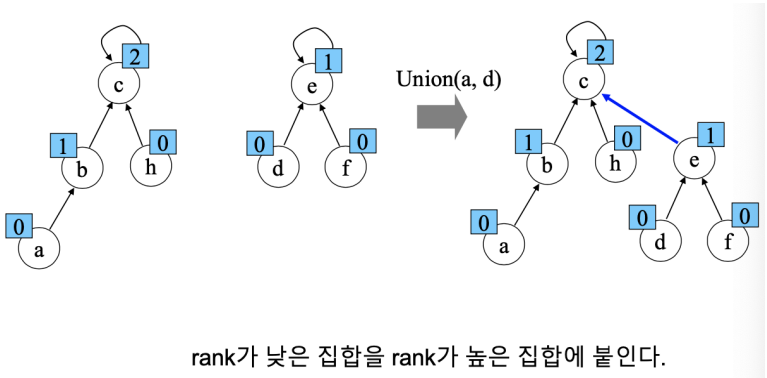
집합의 랭크(rank)

루트 노드의 rank가 그 집합의 rank이다.

랭크를 이용한 Union

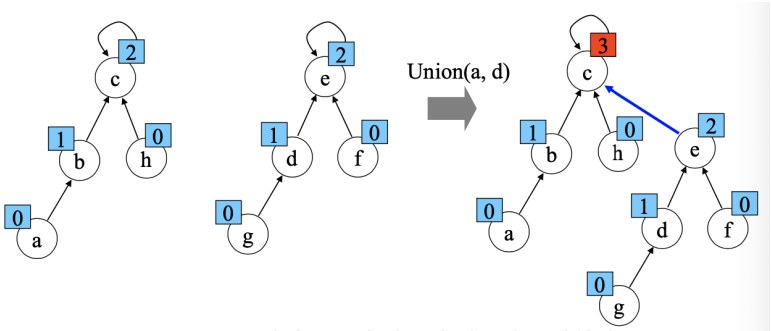
Union 연산 시 rank가 낮은 집합을 rank가 높은 집합에 붙인다.

랭크가 변하지 않는 예



rank가 낮은 집합을 rank가 높은 집합에 붙인다.

랭크가 증가하는 예



두 트리의 rank가 같으면 랭크가 증가한다.

랭크를 이용한 알고리즘

Mask-Set(x)

```
Make-Set(x)       // 노드 x를 유일한 원소로 하는 집합을 만든다.
{
  x.parent <- x;
    x.rank <- 0;
}
```

Union(x, y)

```
Union(x)        // 노드 x가 속한 집합과 노드 y가 속한 집합을 합친다.
{
  xx <- Find-Set(x);
    yy <- Find-Set(y);
  if(xx.rank > yy.rank)
      yy.parent <- xx;
    else {
      xx.parent <- yy;
        if (xx.rank = yy.rank)
          yy.rank <- yy.rank + 1;
    }
}
```

Find-Set(x)은 랭크를 이용하지 않은 알고리즘과 동일하다.

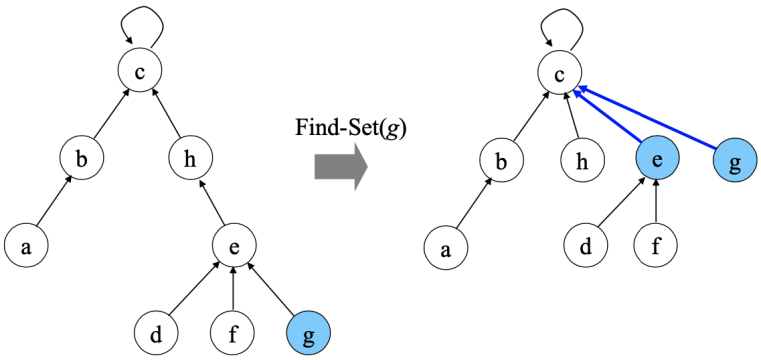
2. 경로 압축 (path compression)

Find-Set 연산을 수행하는 과정에서 경로의 길이를 줄이는 시도를 한다.

개념

Find-Set을 수행하는 과정에서 만나는 모든 노드들에 대해, 현재 부모를 가리키는 대신, 직접 root를 가리키도록 parent 포인터를 바꿔준다. 이 과정에서 트리의 높이가 줄어들 가능성이 높다.

예시



경로 압축을 이용한 알고리즘

Find-Set(x)

```
Find-Set(x)       // 노드 x가 속한 집합을 알아낸다.
{           // 즉 노드 x가 속한 트리의 루트 노드를 리턴한다.
  if(x.parent != x)           // x가 대표 원소(루트 노드)가 아니면
      x.parent <- Find-Set(x.parent);   // x의 parent의 대표 원소를 찾아 x의 parent로 삼는다.

    return x.parent;
}
```

수행시간

랭크를 이용한 Union

- 랭크가 k인 노드를 대표로 하는 집합의 원소 수는 최소한 2k개다.
- 원소 수가 n인 집합을 표현하는 트리에서 임의의 노드의 랭크는 **O(log n)**이다.
- m번의 Make-Set, Union, Find-Set 중 n번이 Make-Set이라면, 이들의 총 **수행 시간은 O(mlogn)**이다.

Tree로 표현한 배타적 집합에서 랭크를 이용한 Union과 경로압축을 이용한 Find-Set을 동시에 사용

m번의 Make-Set, Union, Find-Set 중 n번이 Make-Set일 때 이들의 총 수행시간은 **O(m log*n)**이다.

+) log*n 은 “로그스타 n” 이라고 읽고, 다음과 같이 정의된다.
log*n = min {k : loglog ...logn ≤ 1}

즉 현실적인 n값에 대해 logn는 상수라고 봐도 될 정도로 작은 값이다. 따라서 **O(*m logn*) 은 사실상 O(m)으로 선형시간**이다.

요약

상호 배타적 집합을 표현하는 대표적인 방법 : **Linked List를 이용한 방법, Tree를 이용한 방법**
상호 배타적 집합 처리에 필요한 작업 : **Make-Set, Find-Set, Union**
Linked List를 이용한 방법은 잘 구현하면 **Make-Set, Find-Set은 O(1), Union은 평균 O(log n) 시간을 소요**한다.
트리를 이용한 방법은 잘 구현하면 m번의 Make-Set, Find-Set, Union 작업에 **O(m) 시간**을 소요한다.