#11-3. 서로소 집합

나정휘

https://justiceHui.github.io/

서로소 집합

서로소 집합족

- 교집합이 공집합인 집합으로 구성된 집합족
  - 두 집합 A, B가 A = B이거나 $A \cap B = \emptyset$
  - 서로소인 두 정수 A, B의 소인수분해를 생각해 보자.
- 각 원소가 속한 집합을 유일하게 특정할 수 있음

서로소 집합

Union Find (또는 Disjoint Set)

- 서로소 집합을 관리하는 자료구조
  - Init : 모든 원소가 자기 자신만을 원소로 하는 집합에 속하도록 초기화
  - Union(u, v) : u가 속한 집합과 v가 속한 집합을 병합
  - Find(v) : v가 속한 집합을 반환

- 위 3가지 연산을 빠르게 구현하는 것이 목표

서로소 집합

Union Find

- 각 집합을 Rooted Tree로 표현한다고 생각해 보자.
  - Init은 포레스트에서 모든 간선을 제거하는 것과 동일
  - Find(v)는 v가 속한 트리의 루트 정점을 반환하면 됨
  - Union(u, v)는 u가 속한 트리의 루트를 v가 속한 트리의 루트의 자식으로 넣어주면 됨

  - 시간 복잡도는?

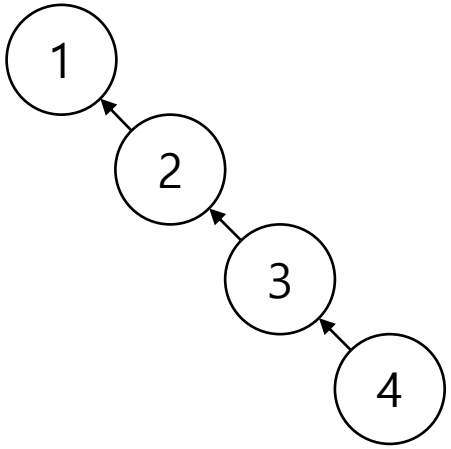
서로소 집합

```cpp
int P[101010];

void Init(int n){
    for(int i=1; i<=n; i++) P[i] = i;
}

int Find(int v){
    if(v == P[v]) return v;
    return Find(P[v]);
}

void Union(int u, int v){
    int u_root = Find(u), v_root = Find(v);
    if(u_root != v_root) P[u_root] = v_root;
}
```

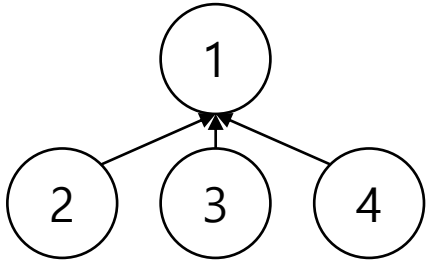
서로소 집합

Union Find

- Init : O(N)
- Find : 트리의 높이를 h라고 하면 O(h)
- Union : Find와 동일



- 최악의 경우 O(N)
- 최선의 경우 O(1)



질문?

서로소 집합

최적화

- 트리의 높이를 줄여야 함
  - Union by Rank
  - Union by Size
  - Path Compression

- 3개 중 하나만 사용해도 M번 연산했을 때 O(M log N)이 보장됨
  - Union by Rank, Union by Size는 O(log N)
  - Path Compression은 amortized O(log N)

- Union by Rank와 Path Compression을 함께 사용하면 amortized O(log* N)
  - log* N : N을 1 이하로 만들기 위해서 필요한 log의 횟수
  - log* N = 1 + log* (log N)

서로소 집합

Union by Rank

- 높이가 낮은 트리를 높은 트리 아래로 넣는 방법
  - Rank[i] : i를 루트로 하는 트리 높이의 상한
  - 만약 두 트리의 높이가 동일하면 Union 후 Rank 1 증가

- rank가 k인 정점은 최대 $N/2^{k-1}$개 있음을 증명 가능
- 따라서 트리의 높이는 최대 $\log_2 N$

```cpp
int P[101010], R[101010];

void Init(int n){
    for(int i=1; i<=n; i++) P[i] = i, R[i] = 1;
}

int Find(int v){
    if(v == P[v]) return v;
    return Find(P[v]);
}

void Union(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return;
    if(R[u] > R[v]) swap(u, v);
    P[u] = v;
    if(R[u] == R[v]) R[v] += 1;
}
```

서로소 집합

Union by Size

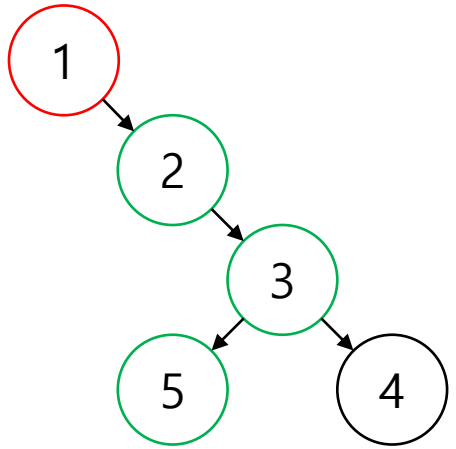
- 정점이 적은 트리를 많은 트리 아래로 넣는 방법
  - Size[i] = i를 루트로 하는 트리의 정점 개수
  - u를 v 밑으로 넣으면 Size[v]는 Size[u] 만큼 증가

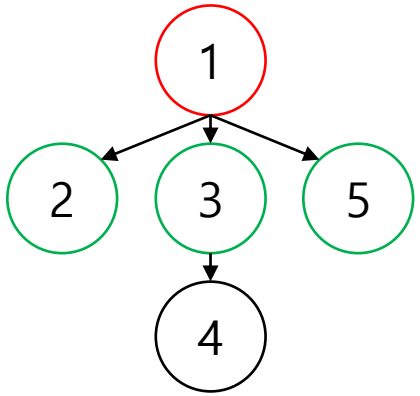
- 어떤 트리가 다른 트리 밑으로 붙을 때마다
- 트리의 크기는 2배 이상 증가함

- 따라서 각 정점의 높이는 최대 $\log_2 N$번 증가함

```cpp
int P[101010], S[101010];

void Init(int n){
    for(int i=1; i<=n; i++) P[i] = i, R[i] = 1;
}

int Find(int v){
    if(v == P[v]) return v;
    return Find(P[v]);
}

void Union(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return;
    if(S[u] > S[v]) swap(u, v);
    P[u] = v; S[v] += S[u];
}
```

Union Find





Path Compression

- 경로 압축
  - Find를 통해 루트를 찾았다면
  - 루트까지의 경로 상에 있는 정점을 모두 루트의 바로 밑(자식)으로 붙임

- 한 번의 연산에서는 O(N) 만큼의 시간이 걸릴 수 있음
  - Union(N, N-1), Union(N-1, N-2), ... , Union(2, 1) 하면 높이 N

- M(≥ N번)의 연산을 하면 O(M log N)이 됨을 증명할 수 있음
- 따라서 시간 복잡도는 amortized O(log N)
  - 증명은 생략

- Union by Rank + Path Compression은 amortized O(log* N)
  - 증명은 (링크) 참고

```
int P[101010];

void Init(int n){
    for(int i=1; i<=n; i++) P[i] = i;
}

int Find(int v){
    if(v == P[v]) return v;
    return P[v] = Find(P[v]);
}

void Union(int u, int v){
    u = Find(u); v = Find(v);
    if(u != v) P[u] = v;
}
```

질문?

Union Find – 예시 1

BOJ 1976 여행 가자

- 그래프 G와 수열 A가 주어짐
- 수열 A에서 인접한 정점 A[i-1]과 A[i]가 G에서 연결되어 있는지 확인하는 문제

- 그래프의 연결 요소를 Union Find로 관리하면 됨

```cpp
#include <bits/stdc++.h>
using namespace std;

int N, M, A[1010], P[222];
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
void Merge(int u, int v){ if(Find(u) != Find(v)) P[Find(u)] = Find(v); }

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    iota(P+1, P+N+1, 1);
    for(int i=1; i<=N; i++){
        for(int j=1; j<=N; j++){
            int t; cin >> t;
            if(t) Merge(i, j);
        }
    }
    for(int i=1; i<=M; i++) cin >> A[i];

    bool res = true;
    for(int i=2; i<=M; i++) res &= Find(A[i-1]) == Find(A[i]);
    cout << (res ? "YES" : "NO");
}
```

질문?