

#3. 서로소 집합

나정휘

<https://justicehui.github.io/>

서로소 집합

서로소 집합족

- 교집합이 공집합인 집합으로 구성된 집합족
 - 두 집합 A, B 가 $A = B$ 이거나 $A \cap B = \emptyset$
 - 서로소인 두 정수 A, B 의 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) : u 가 속한 집합과 v 가 속한 집합을 병합
 - Find(v) : v 가 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 v 가 속한 트리의 루트 정점을 반환하면 됨
 - Union(u, v)는 u 가 속한 트리의 루트를 v 가 속한 트리의 루트의 자식으로 넣어주면 됨
- 시간 복잡도는?

서로소 집합



```
int P[101010];

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

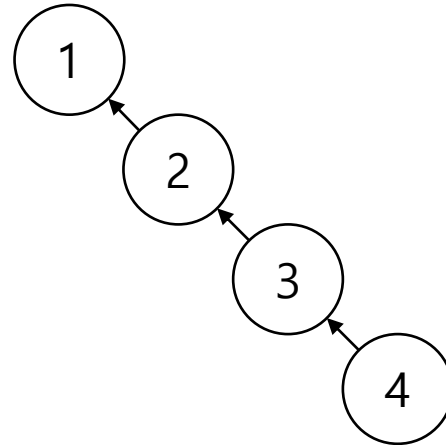
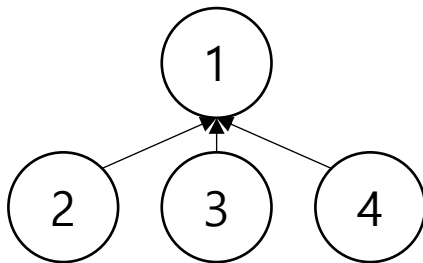
int Find(int v){
    if(v == P[v]) return v;
    return Find(P[v]);
}

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 h 라고 하면 $O(h)$
- Union : Find와 동일
- 최악의 경우 $O(N)$
- 최선의 경우 $O(1)$



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 $\log_2 N$



```
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
 - $Size[i]$ = i 를 루트로 하는 트리의 정점 개수
 - u 를 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



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

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

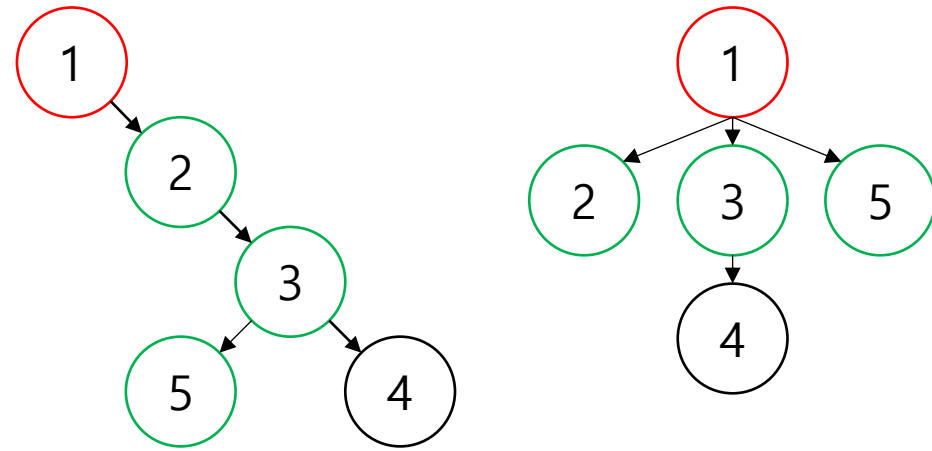
int Find(int v){
    if(v == P[v]) return v;
    return Find(P[v]);
}

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];
}
```

서로소 집합

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
- $M(\geq N\text{번})$ 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;
}
```

서로소 집합 - 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



```
#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");
    11
}
```

서로소 집합

Small to Large

- 그래프 G와 수열 A가 주어짐
- Union by Size의 시간 복잡도 계산을 다시 생각해 보자.
 - 크기가 작은 트리를 큰 트리 밑에 붙이는 방식으로 합병
 - Find(x)의 값이 바뀔 때마다, x가 속한 트리의 크기는 2배 이상으로 증가함
 - $2 * \min(a, b) \leq a + b$ 이기 때문
 - 따라서 각 정점의 깊이는 최대 $\log_2 n$ 번 증가함
- Union Find에서 각 집합의 원소를 실제로 관리해야 한다면?
 - Union 할 때 크기가 작은 집합의 원소를 크기가 큰 집합으로 옮기면
 - 각 원소의 이동 횟수는 최대 $\log_2 n$ 번

서로소 집합 - 예시 2

BOJ 17469 트리의 색깔과 쿼리

- N개의 정점으로 구성된 트리가 주어지고, 각 정점은 1 이상 10만 이하의 색을 갖고 있음
- 아래 두 가지 쿼리를 처리해야 함
 - 1 a: 정점 a와 a의 부모 정점을 연결하는 간선을 제거한다.
 - 2 a: 정점 a에서 갈 수 있는 정점만 보았을 때, 색깔의 종류의 개수를 출력한다.
- 1번 쿼리는 N-1번, 2번 쿼리는 Q번 주어짐
- Union Find는 간선을 삭제할 수 없음
 - 쿼리를 뒤에서부터 처리하면 간선이 없는 그래프에 간선을 하나씩 추가하는 상황이라고 생각할 수 있음
- 컴포넌트마다 색깔의 종류를 관리해야 함
 - 중복을 허용하지 않는 자료구조(e.g., `std::set`)을 이용해서 색깔의 종류를 관리
 - small to large를 이용하면 자료구조에 색깔을 삽입하는 연산을 총 $O(n \log n)$ 번 수행함
 - `std::set`을 사용하면 시간 복잡도는 $O(n \log^2 n)$

서로소 집합 - 예시 2

BOJ 17469 트리의 색깔과 쿼리

- Pa[x]: x번 정점의 부모 정점
- O[i]: i번째 쿼리의 종류
- V[i]: i번째 쿼리의 피연산자
- R[i]: 2번 쿼리의 정답 (뒤에서부터 출력해야 함)

```
#include <bits/stdc++.h>
using namespace std;
constexpr int S = 101010;

int P[S], Pa[S];
set<int> st[S];
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
void Merge(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return;
    if(st[u].size() > st[v].size()) swap(u, v);
    for(auto i : st[u]) st[v].insert(i);
    st[u].clear();
    P[u] = v;
}

int N, Q, O[1110101], V[1110101], R[1010101];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q; iota(P, P+S, 0);
    for(int i=2; i<=N; i++) cin >> Pa[i];
    for(int i=1,c; i<=N; i++) cin >> c, st[i].insert(c);
    for(int i=1; i<N+Q; i++) cin >> O[i] >> V[i];

    for(int i=N+Q-1, idx=0; i; i--){
        if(O[i] == 1) Merge(V[i], Pa[V[i]]);
        else R[++idx] = st[Find(V[i])].size();
    }
    for(int i=Q; i; i--) cout << R[i] << "\n";
}
```