Kruskal

1115 이정우

목차

1 자료구조 개요

Union-Find는 서로소 집합을 효율적으로 표현하고 관리하기 위한 자료구조로, 원소들이 어떤 집합에 속하는지 확인하는 find연산과 두 집합을 합치는 union 연산을 제공한다. 내부적으로는 Tree구조를 사용하며, 경로압축 및 랭크 기반 합치기 기법을 적용해 상수시간에 가까운 효율로 동작이 가능하다.

2 자료구조 상세

2.1 변수와 ADT

2.1.1 변수

- `parent: any[size]`
  - 각 원소가 가진 대푯값을 표시하는 배열이다.

- `Root"char": int`
  - 자신의 대푯값을 저장하는 변수이다.
- `n: int`
  - 정점의 갯수를 저장하는 변수이다.

- `m: int`
  - 연산할 횟수를 저장하는 변수이다.

2.1.2 ADT

- `my_union() -> void`
  - 두 노드를 병합하는 함수이다.
- `find() -> int`
  - 노드의 대푯값을 반환한다.

2.1.3 Exception

- 특별한 예외 상황은 존재하지 않는다.

2.2 구현 상세 및 핵심 코드

2.2.1 my_union()

b와 c를 매개변수로 받고 두 변수(노드)를 병합하는 함수이다. find()함수를 통해 각 노드가 속한 집합의 대푯값을 찾고 그 대푯값중 더 작은 수가 b, c노드가 속한 집합의 대푯값이 된다.

```
    void my_union(int b, int c){
    int RootX = find(b);
    int RootY = find(c);
    if(a!=y){
        if(a>y)parent[a] = parent[y];
        else parent[y] = parent[a];
    }
}
```

2.2.2 find()

노드의 대푯값을 반환하는 기능을 가진 함수이다. 노드 a의 대푯값이 자신을 가르키면 계속해서 재귀적으로 자신의 뿌리를 찾고 그렇지 않으면 최종값은 그 노드의 대푯값이 되므로 그 값을 반환한다.

```
int find(int a){
    if(parent[a]==a)return a;
    return parent[a] = find(parent[a]);
}
```

2.3 복잡도 분석

- 위 함수를 바탕으로 Union-Find를 하게 된다면 처음에는 O(n)의 시간 복잡도를 가지지만 계속해서 parent의 값이 갱신되므로 경로를 찾는 연산 횟수가 낮아지기 때문에 나중에는 O(1)의 시간복잡도를 가지게 된다. 이를 아커만 함수 역함수로 부르며, O(a(n))으로 표현된다.

2.4 전체 코드

```
#include <stdio.h>

int parent[1000001]={0,};
```

```
int find(int a){
    if(parent[a]==a)return a;
    return parent[a] = find(parent[a]);
}

void my_union(int b, int c){
    int RootX = find(b);
    int RootY = find(c);
    if(RootX!=RootY){
        if(RootX>RootY)parent[RootX] = parent[RootY];
        else parent[RootY] = parent[RootX];
    }
}

int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i=1; i<=n; i++){
        parent[i] = i;
    }
    for(int i=1; i<=m; i++){
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        if(a==0){
            my_union(b, c);
        }
        else{
            if(find(b)==find(c))printf("YES\n");
            else printf("NO\n");
        }
    }
    return 0;
}
```

3 활용 방안

1. 서버/네트워크 연결 체크
    - Union-Find를 활용하여 연결된 네트워크를 빠르고 쉽게 확인할수있다.

4 인상깊었던 점

이전에는 DFS, BFS, 다익스트라, 플로이드 워셜, 벨만-포드 등 최단경로, 즉 정점과 정점 사이의 최단거리를 초점에 뒀다면 이번에는 이 노드가 다른노드와 연결되있는지 확인하는 자료구조라 새로웠다. 또한 단순히 같은 경로에 있다는것이 아니라 집합이라는 수학 개념을 활용하여 자료구조를 정의했다는것이 인상깊었다.

5 참고문헌

유니온 파인드(Union-Find)