六 森林与并查集整理

6.1 基础知识点总结

> · 森林只有两种遍历方式，前序和后序
> · Quick-Find算法，是基于染色的思想解决联通性问题模型的方法，注意头的变化，联通判断O(1)，合并操作O(n)
> · Quick-Union算法，当前节点记录其前驱节点的地址，联通判断与合作操作都基于树的高度
> · Weighted Quick-Union算法，依据节点权值来确定谁是根，联通判断与合并操作都是log(N)

6.2 题型总结

> 题型一 分组问题，求合并后的树的个数

```
#include <stdio.h>
#include <stdlib.h>

typedef struct JointSet {
    int *father, *rank;
} JointSet;

JointSet *init(int n);
int find_set(JointSet *, int);
int merge(JointSet *, int, int);
void clear(JointSet *);

void swap(int *a, int *b) {
    int *temp = a;
    a = b;
    b = temp;
}

int max(int a, int b) {
    return (a > b ? a : b);
}

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    JointSet *s = init(n);
    for (int i = 0; i < m; i++) {
        int node1, node2;
        scanf("%d%d", &node1, &node2);
        merge(s, node1, node2);
    }
    int ans = 0;
    for (int i = 0; i < n; i++) {
        if (s->father[i] == i) ans += 1;
    }
    printf("%d", ans);
    clear(s);
    return 0;
}

JointSet *init(int n) {
    JointSet *s = (JointSet *)malloc(sizeof(JointSet));
    s->father = (int *)malloc(sizeof(int) * n);
    s->rank = (int *)malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++) {
        s->father[i] = i;
        s->rank[i] = 1;
    }
    return s;
}

int find_set(JointSet *s, int node) {
    if (s->father[node] != node)
        s->father[node] = find_set(s, s->father[node]);
    return s->father[node];
}

int merge(JointSet *s, int node1, int node2) {
    int ans1 = find_set(s, node1);
    int ans2 = find_set(s, node2);
    if (ans1 != ans2) {
        if (s->rank[ans1] > s->rank[ans1]) swap(&ans1, &ans2);
        s->father[ans1] = ans2;
        s->rank[ans2] = max(s->rank[ans1] + 1, s->rank[ans2]);
        return 1;
    }
    return 0;
}

void clear(JointSet *s) {
    if (s == NULL) return ;
    free(s->father);
    free(s->rank);
    free(s);
    return ;
}
```

> 题型二 冗余关系，就是在合并的时候父节点相同的情况，在合并操作加入变量统计即可

```
int merge(JointSet *s, int node1, int node2) {
    int ans1 = find_set(s, node1);
    int ans2 = find_set(s, node2);
    if (ans1 != ans2) {
        if (s->rank[ans1] > s->rank[ans1]) swap(&ans1, &ans2);
        s->father[ans1] = ans2;
        s->rank[ans2] = max(s->rank[ans1] + 1, s->rank[ans2]);
        return 1;
    }
    count += 1;
    return 0;
}
```