并查集

并查集（Union-Find Set），也称为不相交集数据结构（Disjointed Set Data Structure）。是指一系不相交的集合（Sets），提供合并（Union）和查找(Find)两种操作。

用来解决连通性问题

并查集，在一些有N个元素的集合应用问题中，我们通常是在开始时让每个元素构成一个单元素的集合，然后按一定顺序将属于同一组的元素所在的集合合并，其间要反复查找一个元素在哪个集合中。

每个集合通过一个代表来识别，代表即集合中的某个成员，通常选择根做这个代表。

我们通常用树来表示一个集合。

（1）初始化操作：

开始每个数的代表都是他们自己所在集合的代表。

（2）合并操作：

合并两个元素所在的集合，所以就需要知道所在数的根节点，此时我们就需要也个大哥。

并查集需要选择一个人当作他们的大哥，但他们都是一个类，都是亲戚关系。

（3）查找操作：

将两棵树合并为一棵树需要find操作，

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

typedef struct DisjointSet{
    //声明一个数组    存储他的父亲节点
    int *father;
    int *rank;
} DisjointSet;

void init(DisjointSet *s, int size) {
    //申请空间
    s->father = (int *)malloc(sizeof(int) * size);
    s->rank = (int *)malloc(sizeof(int) * size);
    for (int i = 0; i < size; ++i) {
        s->father[i] = i;//将其全部初始化为自己
        s->rank[i] = 1;
    }
}

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

```c
int max(int a, int b) { //找最大值
    return a > b ? a : b;
}

int find_set(DisjointSet *s, int node) {//返回他的根节点
    if (s->father[node] != node) {
        return s->father[node] = find_set(s, s->father[node]);
    }
    return node;
}

int merge(DisjointSet *s, int node1, int node2) {//合并操作
    int ancestor1 = find_set(s, node1);  //node1的根节点
    int ancestor2 = find_set(s, node2); //node2的根节点
    if (ancestor1 != ancestor2) {  //如果两个人老大不同就合并操作
        if (s->rank[ancestor1] > s->rank[ancestor2]) {//判断谁的深度大
            swap(&ancestor1, &ancestor2);//将深度大的定义为ａｎｃｅｓｔｏｒ２
        }
        s->father[ancestor1] = ancestor2;//将ａｎｃｅｓｔｏｒ２设置为ａｎｃｅｓｔｏｒ１的父亲节点
        s->rank[ancestor2] = max(s->rank[ancestor2], s->rank[ancestor1] + 1);
        //合并后深度变为rank1的深度加上一 和 原来ｒａｎｋ２中深度教大的
        return 1;//成功返回１
    }
    return 0;//两个人本身就在同一个集合
}

void clear(DisjointSet *s) {//清除操作
    free(s->father);
    free(s->rank);
    free(s);
}

int main() {
    DisjointSet *dsu = (DisjointSet *)malloc(sizeof(DisjointSet));
    //申请空间
    init(dsu, 100);
    int m, x, y;
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d%d", &x, &y);
        int ans = merge(dsu, x, y);
        if (ans) {
            printf("success\n");
        } else {
            printf("failed\n");
        }
    }
    clear(dsu);
    return 0;
}
```

(1) union-find

首先最基本的并查集，直接将root1当作父节点

```
void union(int *s, int root1, int root2){
         s[root2] = root1;//将root1作为root2的新树根
}
```

在进行合并操作时什么情况下合并所需要的次数最小？

在进行合并时高度越小的查找操作越快 下面的quick优化7

```
int find (int *node, int x) {
    if (node[x] == x) return x;
    else return find(node, x);
}
```

find此时返回的是所查找到的根节点

(2) Quick union-find

1、按秩合并

在一种极端情况下如N个元素退化为一条链，而查找时就会遍历整条连时间复杂度为O(n)

为了避免这种情况，我们可以再合并的时候尽可能的让树的深度不要过深

我们就需要申请一个新的数组 rank 存储深度

将 rank 数组全部初始化为 1

```
int merge(DisjointSet *s, int node1, int node2) {
    int ancestor1 = find_set(s, node1);//查找node1的根节点
    int ancestor2 = find_set(s, node2);//查找node2的根节点
    if (ancestor1 != ancestor2) {  //两个不属于一个集合合并
        if (s->rank[ancestor1] > s->rank[ancestor2]) {
            swap(&ancestor1, &ancestor2);
        }
        //将深度大的定义为ａｎｃｅｓｔｏｒ２
        s->father[ancestor1] = ancestor2;//将ａｎｃｅｓｔｏｒ２设置为ａｎｃｅｓｔｏｒ１的父亲节点
        s->rank[ancestor2] = max(s->rank[ancestor2], s->rank[ancestor1] + 1);
        //合并后深度变为原来的深度加上一个根和原来ｒａｎｋ２深度教大的
        return 1;
    }
    return 0;
}
```

2、路径压缩

原来的只是将根节点返回，我们可以将自身节点直接连接到其根节点当中

```
int find_set(DisjointSet *s, int node) {
    if (s->father[node] != node) {
        s->father[node] = find_set(s, s->father[node]);//将找到的根节点返回
    }
    return s->father[node];
}
```

(3) 习题

poj 1611

题意：

有n个学生分为m组，将n个学生编号，有一个同学患病其所在的组也被标记为疑似患病，已知0号学生疑似患病，求一共有多少个学生疑似患病。

题目讲解：

我们可以利用并查集，先申请一个数组存储n个人之后给每个人编上号，之后在申请一个数组存储i作为老大时的集合人数，之后进行并查集合并。

代码：

```
#include<stdio.h>
#include <stdlib.h>
int sum[30011];//老大存储集合中的人数
//模板
int find(int *node, int value) {
    if (node[value] != value) {
        node[value] = find(node, node[value]);
    }
    return node[value];
}

void bing(int *node, int x, int y) {
    int a1 = find(node, x);
    int a2 = find(node, y);
    if (a1 != a2) {
        node[a1] = a2;
        sum[a2] += sum[a1];//集合中人数加一个
    }
}

int main() {
    int m, n, p, hh;
    while(scanf("%d %d", &n, &m) != EOF) {
        if (m == 0 && n == 0) break;
        int num[30011] = {0}, oo;
        for (int i = 0; i <= n; i++) {
            num[i] = i;
            sum[i] = 1;//将sum数组初始化为一
        }
        for (int i = 0; i < m; i++) {
```

```
            scanf("%d", &p);
            scanf("%d", &hh);//将之后输入的全部和h h合并
            for (int i = 1; i < p; i++) {
                scanf("%d", &oo);
                bing(num, hh, oo);
            }
        }
        printf("%d\n", sum[find(num, 0)]);//直接找0点所在集合的人数即可
    }
}
```

poj 2236

题意：

n台电脑， d表示两台电脑的最大通信距离，输入n个点的坐标，接下来O表示将电脑修好， S表示查看是否通信。

必须修好才可以进行查看是否能通信

代码：

```
#include<stdio.h>
#include<stdlib.h>
int d;//输入的d值，方便调用
typedef struct Node {
    int x, y;//坐标值
    int cmd;//第i个电脑
} Node;
//模板
int find(Node *root, int key) {
    if (root[key].cmd == key) return key;
    return root[key].cmd = find(root, root[key].cmd);
}

void bing(Node *node, int x, int y) {
    int a1 = find(node, x);
    int a2 = find(node, y);
    if (a1 != a2) {
        if ((node[x].x - node[y].x) * (node[x].x - node[y].x) + (node[x].y - node[y].y) *
(node[x].y - node[y].y) <= d * d) {
            //两个点的距离不超过d
            node[a2].cmd = a1;
        }
    }
}

int main() {
    int n, x, y, p, q, heal[1111] = {0};
    char ch;
    scanf("%d %d", &n, &d);
    //申请空间
    Node *node = (Node *)malloc(sizeof(Node) * (n + 5));
```

```
    for (int i = 0; i <= n; i++) {//n个电脑
        node[i].cmd = i;
    }
    for (int i = 1; i <= n; i++) {//将n个电脑的坐标存储
        scanf("%d %d", &node[i].x, &node[i].y);
    }
    while(scanf("\n%c", &ch) != EOF) {
        if (ch == 'O') {//修电脑
            scanf("%d", &p);
            heal[p] = 1;//标记已经被修好的电脑
            for (int i = 1; i <= n; i++) {
                if (heal[i] != 0 && i != p)   //和被修好的电脑进行合并
                bing(node, i, p);
            }
        } else {
            scanf("%d %d", &p, &q);//查找是否是联通的
            if (find(node, p) == find(node, q)) {
                printf("SUCCESS\n");
            } else {
                printf("FAIL\n");
            }
        }
    }
}
```

杭电oj 1213

题意：

一个人过生日邀请了很多朋友，但有些朋友之间是不认识的，所以他们不能坐在一个桌子上，求最多要多少张桌子

代码：

```
//模板题
#include<stdio.h>
#include <stdlib.h>
//模板
int find(int *node, int value) {
    if (node[value] != value) {
        node[value] = find(node, node[value]);
    }
    return node[value];
}
void bing(int *node, int x, int y) {
    int a1 = find(node, x);
    int a2 = find(node, y);
    if (a1 != a2) {
        node[a1] = a2;
    }
}

int main() {
    int m, n, x, father[100001], a, b;
```

```
    scanf("%d", &x);
    for (int i = 0; i < x; i++) {
        scanf("%d %d", &n, &m);
        for (int i = 1; i <= n; i++) {
            father[i] = i;
        }
        while (m--) {
            scanf("%d %d", &a, &b);
            bing(father, a, b);//将a，b合并
        }
        int num = 0;
        for (int i = 1; i <= n; i++) {
            //查找有多少个头节点即多少个集合
            if (father[i] == i) num++;
        }
        printf("%d\n", num);
    }
    return 0;
}
```

杭电oj 1232

题意：

n个城镇，题目要求使任意两个城镇都能通最少还要多少条道路 我们可以利用并查集将所有输入的并在一起　并便利一遍看看有几个根节点即可

```
#include<stdio.h>
#include <stdlib.h>
//模版
int find(int *node, int value) {
    if (node[value] != value) {
        node[value] = find(node, node[value]);
    }
    return node[value];
}

void bing(int *node, int x, int y) {
    int a1 = find(node, x);
    int a2 = find(node, y);
    if (a1 != a2) {
        node[a1] = a2;
    }
}
//
int main() {
    int m, n;
    while (scanf("%d", &m) && m != 0) {
        scanf("%d", &n);
        int x, y, num = 0;
```

```
        int *node = (int *)malloc(sizeof(int) * (m + 1));
        for (int i = 0; i < m; i++) {
            node[i] = i;
        }
        //合并操作  因为我习惯从0开始所以在合并时就减一个进行合并
        for (int i = 0; i < n; i++) {
            scanf("%d %d", &x, &y);
            bing(node, x - 1, y - 1);
            num++;
        }
        //标记数组
        int nx[1001] = {0}, max = 0;
        //nx用于计数，查找有多少个集合
        for (int i = 0; i < m; i++) {
            if (nx[find(node, i)] == 0) {
                nx[find(node, i)] = 1;
                max++;
            }
        }
        printf("%d\n", max - 1);
        free(node);
    }
    return 0;
}
```

落谷 p1111

题意：

修公路，给出了每个公路修的时间，此题求最少需要的时间，我们知道他要求任意的两个村庄都能够通车,即所有的村庄都在一个集合中 题目还要求我们时间最短我们直接按时间来排序一下即可 所以我们直接用并查集按照我们排好的序列来进行合并并计算时间 按照返回值正确的个数和村庄的个数 - 1相同时就是最小的时间 如果结束后所得值小于村庄个数 - 1返回 - 1 注意的是 排序的时间必须是O(nlogn)的 洛谷过不去

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

typedef struct Node {
    int x, y, z;
} Node;
//模版
int find(int *father, int node) {
    if (father[node] != node) {
        return find(father, father[node]);
    }
    return father[node];

}
//快排
void sort(Node *num, int l, int r) {
    if (r <= l) return ;
```

```
    int x = l, y = r;
    Node mi = num[l];
    while (x < y) {
        while (x < y && num[y].z >= mi.z) --y;
        if (x < y) num[x++] = num[y];
        while (x < y && num[x].z <= mi.z) ++x;
        if (x < y) num[y--] = num[x];
    }
    num[x] = mi;
    sort(num, l, x - 1);
    sort(num, x + 1, r);
    return ;
}

int main() {
    int m, n;
    scanf("%d %d", &m, &n);
    //为申请node数据类型空间
    Node *p = (Node *) malloc (sizeof(Node) * 100001);
    for (int i = 0; i < n; i++) {
        scanf("%d %d %d", &p[i].x, &p[i].y, &p[i].z);
    }
    int num = 0, hh = 0;
    //按照时间来排序
    sort(p, 0, n);
    int father[100001];
    for (int i = 0; i <= m+5; i++) {
        //将数组初始化
        father[i] = i;
    }
    for (int i = 0; i < n; i++) {
        int a1 = find(father, p[i].x);
        int a2 = find(father, p[i].y);
        if (a1 != a2) {
            father[a1] = a2;
            num++;
            hh = p[i].z;
        }
    }
    //最后求得总和小于城镇的数量减一个则返回 - 1
    if (num < m - 1) {
        printf("-1\n");
    } else {
        printf("%d\n", hh);
    }
        free(p);
}
```

poj 1182

题意：

三种动物，A, B, C　A吃B， B吃C， C吃D。

2 种关系 第一种说法是”1 X Y”，表示X和Y是同类。 第二种说法是”2 X Y”，表示X吃Y。

1） 当前的话与前面的某些真的话冲突，就是假话；
2） 当前的话中X或Y比N大，就是假话；
3） 当前的话表示X吃X，就是假话。

找出其中的假话。

思路：

因为要处理的是3个种类之间的关系问题，所以应当用并查集来解决。

将扩充为 3 倍的空间，并且如果 x > n || y > n判断是假话

如果 x， y 是同一种动物的话，判断是否是吃和被吃的关系，如果是就是假话，否则每种空间都进行相同的合并

如果 x， y 是吃与被吃的关系的话，我们就应该判断是否是同一个种类或者是 y 吃 x，如果是的话就是假话，否则进行 1 和 2， 2 和 3， 3 和 1 之间的合并

```
#include<stdio.h>
#include<stdlib.h>
//模板
int find(int *num, int key) {
    if (num[key] != key) {
        num[key] = find(num, num[key]);
    }
    return num[key];
}
void bing(int *num, int x, int y) {
    int a1 = find(num, x);
    int a2 = find(num, y);
    if (a1 != a2) {
        num[a1] = a2;
    }
}

int main() {
    int m, n, father[1500000], o, x, y;
    scanf("%d %d", &m, &n);
    for (int i = 0; i <= m * 3; i++) {
        father[i] = i;
    }
    int sum = 0;
    for (int i = 0; i < n; i++) {
        scanf("%d %d %d", &o, &x, &y);
        if (x > m || y > m || x < 0 || y < 0) {
            sum++;
            continue;
        }
        if (o == 1) {
            //如果 o = 1 证明是同类的关系，此时我们只需要判断 x 被 y 吃或者 y 被 x 吃就可以判断真假
            //即如果 x 和  第二个空间所在的 y 是吃或被吃的关系则是假话
            if (find(father, x) == find(father, y + m) || find(father, x + m) ==
find(father, y)) {
                sum++;
```

```
            } else {
                //将1，2，3空间的x，y分别合并
                bing(father, x, y);
                bing(father, x + m, y + m);
                bing(father, x + m * 2, y + m * 2);
            }
        } else if (o == 2 && x == y) {
            sum++;
        } else {
            //此时0 = 2证明x和y是吃与被吃的关系，我们只需要判断x和y在头结点相同就说明x和y就是同类
            //或者y吃下一个空间的x也说明x吃y是错误的，我们此时只需要判断第二个空间的x是否和y头结点是
否相同即可
            if (find(father, x) == find(father, y) || find(father, x + m) == find(father,
y)) {
                sum++;
            } else {
                //将1，2和2，3和3，4空间的x，y分别合并
                bing(father, x, y + m);
                bing(father, x + m, y + 2 * m);
                bing(father, x + 2 * m, y);
            }
        }
    }
    printf("%d\n", sum);
}
```