并查集

NEFU-付钧元，周思乐，潘航

为什么叫并查集

并查集（union-find disjoint set）是一种可以动态维护若干个不重叠的集合，支持合**并**与**查**询的数据结构。

并查集一般包含两个基本操作：

1. 查询一个元素属于哪一个集合
2. 把两个集合合并成一个大集合

存储方式

我们选择用树来表示每个集合，并为每个集合选出特定的元素（根），作为整个集合的代表。

如果给出的元素之间没有明显的树状关系，我们可以令最先被处理的元素作为该集合的代表（也可以用其他既定关系，比如编号最小的元素）。

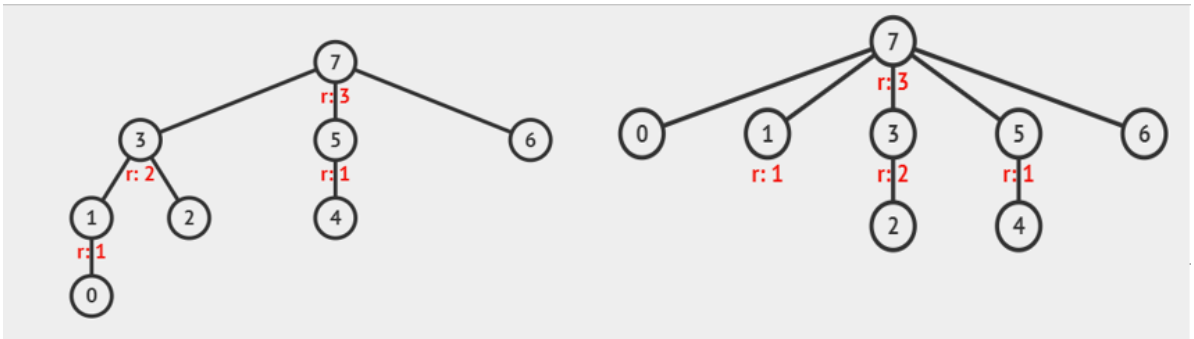
为了维护这个树状关系，每一个元素需要存储其父节点的下标。 特别地，根节点的父节点为它自己。这样我们就可以通过递归求出任一元素所在的集合了。

在合并集合时，只需要将一个集合根的父节点设置为另一集合的根即可。

并且刚开始时，我们定义其每个元素独占一个集合。

路径压缩

对于左图所示的并查集，如果我们需要多次查找0号节点所在的集合，每一次都需要3次递归，浪费了一定的时间。由于并查集大多数情况下对元素的深度这一参数并不关心，所以可以把每次访问到的点都直接指向树根。



按秩合并思想

一般不会有卡按秩合并的题，所以在这里仅作介绍。

当我们需要将两个集合合二为一时，无论将哪一个集合连接到另一个集合的下面，都能得到正确的结果。但不同的连接方法存在时间复杂度的差异。

具体来说，如果我们将一棵点数与深度都较小的集合树连接到一棵更大的集合树下，显然相比于另一种连接方案，接下来执行查找操作的用时更小。

当然，我们不总能遇到恰好如上所述的集合————点数与深度都更小。鉴于点数与深度这两个特征都很容易维护，我们常常从中择一，作为估价函数。

代码实现

```
//1. 并查集的存储
int fa[SIZE];
//2. 并查集的初始化
//初始每个元素都构成一个独立的集合
for (int i = 1; i <= SIZE; i++)
    fa[i] = i;
//3. 查询操作
int find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = find(fa[x]);
}
//4. 合并操作
void merge(int a, int b)
{
    fa[find(a)] = find(b);
}
```

最小生成树Kruskal算法

在离散数学教材P177中提到：

定义 7.4 设无向连通带权图 $G=\langle V,E,W\rangle$，T 是 G 的一棵生成树. T 各边的权之和称为 T 的权，记作 $W(T)$. G 的所有生成树中权最小的生成树称为 G 的**最小生成树**.

下面介绍求最小生成树的**避圈法**(Kruskal 算法).

设 n 阶无向连通带权图 $G=\langle V,E,W\rangle$ 有 m 条边. 不妨设 G 中没有环(若有环，将所有的环删去)，将 m 条边按权从小到大顺序排列，设为 $e_1,e_2,\cdots,e_m$.

取 e_1 在 T 中，然后依次检查 $e_2,e_3,\cdots,e_m$. 若 e_j 与 T 中的边不能构成回路，则取 e_j 在 T 中，否则弃去 e_j.

其简述了Kruskal算法的过程，但是对于`不能构成回路`如何判断并没有详细说明；

事实上，若 e_t 与 T 中的边不能构成回路，即说明 e_t 所连接的两点不在**同一个**联通集合中，这个便可以通过并查集进行判断；

具体操作可以在下面的例题天空之城 中分析；

例题

Acwing.237 程序自动分析

https://www.acwing.com/problem/content/description/239/

哈希+并查集

考虑一个约束满足问题的简化版本：假设 $x_1, x_2, x_3, \ldots$ 代表程序中出现的变量，给定 n 个形如 $x_i = x_j$ 或 $x_i \neq x_j$ 的变量相等/不等的约束条件，请判定是否可以分别为每一个变量赋予恰当的值，使得上述所有约束条件同时被满足。

例如，一个问题中的约束条件为：$x_1 = x_2$，$x_2 = x_3$，$x_3 = x_4$，$x_1 \neq x_4$，这些约束条件显然是不可能同时被满足的，因此这个问题应判定为不可被满足。

现在给出一些约束满足问题，请分别对它们进行判定。

输入格式

输入文件的第 1 行包含 1 个正整数 t，表示需要判定的问题个数，注意这些问题之间是相互独立的。

对于每个问题，包含若干行：

第 1 行包含 1 个正整数 n，表示该问题中需要被满足的约束条件个数。

接下来 n 行，每行包括 3 个整数 i, j, e，描述 1 个相等/不等的约束条件，相邻整数之间用单个空格隔开。若 $e = 1$，则该约束条件为 $x_i = x_j$；若 $e = 0$，则该约束条件为 $x_i \neq x_j$。

输出格式

输出文件包括 t 行。

输出文件的第 k 行输出一个字符串 `YES` 或者 `NO`，`YES` 表示输入中的第 k 个问题判定为可以被满足，`NO` 表示不可被满足。

数据范围

$1 \leq n \leq 10^5$
$1 \leq i, j \leq 10^9$

输入样例：

```
2
2
1 2 1
1 2 0
2
1 2 1
2 1 1
```

输出样例：

```
NO
YES
```

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

int fa[200005];//fa数组
pair<int,int>e0[200005];
    //存储需要不同值的数对，统一判冲
map<int,int>mp;//映射表
int c,ce0;

int find(int x)
{
    if(fa[x]==x)return x;
    return fa[x]=find(fa[x]);
}

int main()
{
```

```
    int t,n,a,b,e;
    cin>>t;
    while(t--)
    {
        mp.clear();
        c=1,ce0=0;//初始化
        scanf("%d",&n);
        int f=1;
        for(int i=1;i<=200000;i++)//初始化
        {
            fa[i]=i;
        }
        for(int i=1;i<=n;i++)
        {
            scanf("%d%d%d",&a,&b,&e);
            if(!mp[a])mp[a]=c++;
            if(!mp[b])mp[b]=c++;
                //若值不存在，则为其分配哈希值
            a=mp[a],b=mp[b];
            if(!e)
            {
                e0[ce0++]={a,b};
            }
            else
            {
                fa[find(a)]=find(b);//合并
            }
        }
        for(int i=0;i<ce0&&f;i++)//处理需要不同值的值对
        {
            if(find(e0[i].first)==find(e0[i].second))f=0;
        }
        if(f)printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

Acwing.257 关押罪犯

https://www.acwing.com/problem/content/259/

贪心+扩展域并查集

S 城现有两座监狱，一共关押着 N 名罪犯，编号分别为 $1 \sim N$。

他们之间的关系自然也极不和谐。

很多罪犯之间甚至积怨已久，如果客观条件具备则随时可能爆发冲突。

我们用“怨气值”（一个正整数值）来表示某两名罪犯之间的仇恨程度，怨气值越大，则这两名罪犯之间的积怨越多。

如果两名怨气值为 c 的罪犯被关押在同一监狱，他们俩之间会发生摩擦，并造成影响力为 c 的冲突事件。

每年年末，警察局会将本年内监狱中的所有冲突事件按影响力从大到小排成一个列表，然后上报到 S 城 Z 市长那里。

公务繁忙的 Z 市长只会去看列表中的第一个事件的影响力，如果影响很坏，他就会考虑撤换警察局长。

在详细考察了 N 名罪犯间的矛盾关系后，警察局长觉得压力巨大。

他准备将罪犯们在两座监狱内重新分配，以求产生的冲突事件影响力都较小，从而保住自己的乌纱帽。

假设只要处于同一监狱内的某两个罪犯间有仇恨，那么他们一定会在每年的某个时候发生摩擦。

那么，应如何分配罪犯，才能使 Z 市长看到的那个冲突事件的影响力最小？这个最小值是多少？

输入格式

第一行为两个正整数 N 和 M，分别表示罪犯的数目以及存在仇恨的罪犯对数。

接下来的 M 行每行为三个正整数 a_j，b_j，c_j，表示 a_j 号和 b_j 号罪犯之间存在仇恨，其怨气值为 c_j。

数据保证 $1 \leq a_j < b_j < N, 0 < c_j \leq 10^9$ 且每对罪犯组合只出现一次。

输出格式

输出共 1 行，为 Z 市长看到的那个冲突事件的影响力。

如果本年内监狱中未发生任何冲突事件，请输出 0。

数据范围

$N \leq 20000, M \leq 100000$

输入样例：

```
4 6
1 4 2534
2 3 3512
1 2 28351
1 3 6618
2 4 1805
3 4 12884
```

输出样例：

```
3512
```

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll N = 100005;

int fa[N] = {0};
struct dat
{
    int a, b, c;
} da[N];

bool cmp(dat a, dat b)
```

```
{
    return a.c > b.c;
}

int find(int x)
{
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}

int main()
{
    int n, m, i;
    cin >> n >> m;
    for (i = 1; i <= m; i++)
        scanf("%d%d%d", &da[i].a, &da[i].b, &da[i].c);
    sort(da + 1, da + 1 + m, cmp);
    for (i = 1; i <= 2 * n; i++) fa[i] = i;//初始化
    for (i = 1; i <= m; i++)//贪心，从大到小找到第一个发生冲突的
    {
        if (find(da[i].a) == find(da[i].b))//存在冲突
        {
            printf("%d", da[i].c);
            return 0;
        }
        else
        {
            fa[find(da[i].a)] = find(da[i].b + n);
            fa[find(da[i].b)] = find(da[i].a + n);
                //把一个加入另一个的扩展域，代表放入相异的集合
        }
    }
    printf("0");
    return 0;
}
```

牛客竞赛 天空之城

Kruskal最小生成树

时间限制：C/C++ 5秒，其他语言10秒

空间限制：C/C++ 524288K，其他语言1048576K

64bit IO Format: %lld

题目描述

天空之城有5个小镇，名字分别为Ada, Aed, Akk, Orz, Apq，他们也有相互的路径长度。
希达早已期盼着天空之城，如今她登上了天空之城，就想走遍天空之城的每一个城市，但是她希望自己走的路的长度越小越好，以节省体力和节约时间。
由于天空之城具有魔力，如果希达想再走一次自己之前走过的路，则她可以在这条路上不花费任何时间。
但是天空之城的城市太多了，他实在计算不过来，只得请你来帮帮忙了。

输入描述:

第一行，输入n，q，表示有n个城市，q条边；
第二行，输入一个名字tmp，表示希达想要从tmp城市开始行走；
接下来q行，每行输入两个名字a,b和一个数字val，表示a城市与b城市之间的距离为val.(注意可能有重边和自环)

输出描述:

帮助巴鲁计算出最短的路径长度，如果无法走遍所有城市，输出"No!"。

示例1

输入 复制

```
5 5
Orz
Ada Aed 5
Orz Ada 6
Apq Aed 8
Akk Apq 12
Aed Orz 3
```

输出 复制

```
28
```

说明

```
Ada->Aed->Orz->Aed->Apq->Akk
```

备注:

多组输入输出（以EOF结束），保证数据组数不超过 10 。

1 <= n <= 5000, 1 <= q <= 200000, 1 <= val <= 1e9. 每个城市的名字长度不超过10。

保证 $\sum q \leq 200000$ 。

在有前面Kruskal最小生成树的基础上，具体到这道题，在合并集合时同时还需要维护**联通集合节点数**和**联通代价**两个变量；

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

int fa[5003], cnt[5003]; //fa存该城所在集合的根节点（城市），cnt存以该城为根的集合城市数
ll len[5003];            //len存以该城为根的集合内部的连通代价

struct road
{
    int fr, to;
    ll lrd;
} rod[200005];

bool cmp(road a, road b)
{
    return a.lrd < b.lrd;
}

map<string, int> mp; //存城市名与编号的映射

int find(int x)
{
    if (fa[x] != x)
        fa[x] = find(fa[x]);
    return fa[x];
}

int main()
{
    int n, q, i;
    while (~scanf("%d%d", &n, &q))
    {
        mp.clear();
        int ccty = 0, glen;
        string g1, g2;
        cin >> g1;
        mp[g1] = (++ccty); //将开始城设为1
        for (i = 1; i <= q; i++)
        {
            cin >> g1;
            cin >> g2;
            scanf("%d", &glen);
            if (!mp[g1])
                mp[g1] = ++ccty;
            if (!mp[g2])
                mp[g2] = ++ccty;
            rod[i].fr = mp[g2], rod[i].to = mp[g1], rod[i].lrd = glen; //存路
        }
        if (ccty != n)
        {
            printf("No!\n");
            continue;
        } //如果提到的城市数!=n，则必有城市在路网之外
        sort(rod + 1, rod + 1 + q, cmp);
        for (i = 1; i <= n; i++)
            fa[i] = i, cnt[i] = 1, len[i] = 0; //初始化并查集
```

```
        for (i = 1; i <= q; i++)
        {
            int frn = rod[i].fr, ton = rod[i].to;
            if (find(frn) != find(ton))
            {
                if (find(frn) > find(ton))
                    swap(frn, ton);                                  //保证两集以较小根为新根
                cnt[find(frn)] += cnt[find(ton)];                    //处理城市数
                len[find(frn)] += len[find(ton)] + rod[i].lrd; //处理连通代价
                fa[find(ton)] = find(frn);
            }
        }
        if (cnt[1] == n)
            printf("%lld\n", len[1]);
        else
            printf("No!\n");
    }
    return 0;
}

```

洛谷P2024 食物链

https://www.luogu.com.cn/problem/P2024

题目描述

展开

动物王国中有三类动物 A,B,C，这三类动物的食物链构成了有趣的环形。A 吃 B，B 吃 C，C 吃 A。

现有 N 个动物，以 1 - N 编号。每个动物都是 A,B,C 中的一种，但是我们并不知道它到底是哪一种。

有人用两种说法对这 N 个动物所构成的食物链关系进行描述：

- 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 。

此人对 N 个动物，用上述两种说法，一句接一句地说出 K 句话，这 K 句话有的是真的，有的是假的。当一句话满足下列三条之一时，这句话就是假话，否则就是真话。

- 当前的话与前面的某些真的话冲突，就是假话
- 当前的话中 X 或 Y 比 N 大，就是假话
- 当前的话表示 X 吃 X，就是假话

你的任务是根据给定的 N 和 K 句话，输出假话的总数。

输入格式

第一行两个整数，N，K，表示有 N 个动物，K 句话。

第二行开始每行一句话（按照题目要求，见样例）

输出格式

一行，一个整数，表示假话的总数。

输入输出样例

输入 #1 复制

```
100 7
1 101 1
2 1 2
2 2 3
2 3 3
1 1 3
2 3 1
1 5 5
```

输出 #1 复制

```
3
```

说明/提示

$1 \le N \le 5 * 10^4$

$1 \le K \le 10^5$

解法一：扩展域并查集

若`fa[a]=b`，则说明a与b为同类；

若`fa[a+n]=b`，则说明a吃b；

若`fa[a+n]=b`，则说明a被b吃；

通过这个扩展域维护物种间关系即可；

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
int fa[150004];
int find(int x)
{
    if(fa[x]==x)return x;
    else return find(fa[x]);
}
int main()
{
    int n,k,ans=0,o,a,b,i;
    cin>>n>>k;
    for(i=1;i<=3*n;i++)fa[i]=i;
    while(k--)
    {
        scanf("%d%d%d",&o,&a,&b);
        if(a>n||b>n){ans++;continue;}
        if(o==1)//同类
        {
            if(find(a+n)==find(b)||find(a+2*n)==find(b))ans++;
            else
            {
                fa[find(a)]=find(b);
                fa[find(a+n)]=find(b+n);
                fa[find(a+2*n)]=find(b+2*n);
                    //按同类维护
```

```
            }
        }
        else if(o==2)//a吃b
        {
            if(find(a)==find(b)||find(a)==find(b+n))ans++;
            else
            {
                fa[find(a+n)]=find(b);
                fa[find(a+2*n)]=find(b+n);
                fa[find(a)]=find(b+2*n);
            }
        }
    }
    cout<<ans;
}
```

解法二：并查集维护深度

如果对于同类的a和b，我们将其定义为同深度 $d[a] = d[b]$;

如果有a吃b，定义有 $d[a] = d[b] + 1$;

以此策略维护的话，如果$(d[a] - d[b] - 1)\%3 == 0$则说明a吃b，如果$(d[a] - d[b])\%3 == 0$则说明a与b同级;

```
#include <bits/stdc++.h>
using namespace std;
const int N = 5e4 + 4;
int fa[N];
int d[N];
int n, k;
int res = 0;
void init()
{
    for (int i = 1; i <= n; i++)
        fa[i] = i;
}
int find(int x) //路径压缩
{
    if (x != fa[x])
    {
        int t = find(fa[x]); //保存好p[x],以免直接将p[x]变为根节点
        d[x] += d[fa[x]];
        fa[x] = t;
    }
    return fa[x];
}
int main()
{
    scanf("%d%d", &n, &k);
    init();
    while (k--)
    {
        int x, a, b;
        scanf("%d%d%d", &x, &a, &b);
        if (a > n || b > n)
            res++;
        else
```

```
        {
            int faa = find(a), fab = find(b);
            if (x == 1)
            {
                if (faa == fab && (d[a] - d[b]) % 3)
                    res++;            //判断在同一集合时，判断其距离是否满足
                else if (faa != fab) //不在同一集合时，说明没有关系，直接加入即可，维护距离
                {
                    fa[faa] = fab;
                    d[faa] = d[b] - d[a];//d[a]+d[faa]=d[b]
                }
            }
            else
            {
                if (faa == fab && (d[a] - d[b] - 1) % 3)
                    res++;
                else if (faa != fab)
                {
                    fa[faa] = fab;
                    d[faa] = d[b] - d[a] + 1;//d[a]+d[faa]=d[b]+1
                }
            }
        }
    }
    printf("%d\n", res);
    return 0;
}
```

Acwing.145 超市

https://www.acwing.com/problem/content/description/147/

超市里有 N 件商品，每件商品都有利润 p_i 和过期时间 d_i，每天只能卖一件商品，过期商品不能再卖。

求合理安排每天卖的商品的情况下，可以得到的最大收益是多少。

输入格式

输入包含多组测试用例。

每组测试用例，以输入整数 N 开始，接下来输入 N 对 p_i 和 d_i，分别代表第 i 件商品的利润和过期时间。

在输入中，数据之间可以自由穿插任意个空格或空行，输入至文件结尾时终止输入，保证数据正确。

输出格式

对于每组产品，输出一个该组的最大收益值。

每个结果占一行。

数据范围

$0 \leq N \leq 10000$,
$1 \leq p_i, d_i \leq 10000$
最多有 14 组测试样例

输入样例:

```
4  50 2  10 1   20 2   30 1

7  20 1   2 1   10 3  100 2   8 2
   5 20  50 10
```

输出样例:

```
80
185
```

并查集除了维护具象的集合，还可以维护更抽象一些的集合，比如此题中并查集维护的便是 `能卖出商品的日期` 的集合；

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

int fa[10004];
pair<int, int> gds[10004];

int find(int x) //路径压缩
{
    return x == fa[x] ? x : fa[x] = find(fa[x]);
}

bool cmp(pair<int, int> a, pair<int, int> b)
{
    return a.first > b.first;
}

int main()
{
    int n;
    while (~scanf("%d", &n))
    {
        ll ans = 0;
```

```
        int mxd = 0; //日期最大值
        for (int i = 1; i <= n; i++)
        {
            scanf("%d%d", &gds[i].first, &gds[i].second);
            mxd = max(mxd, gds[i].second);
        }
        for (int i = 0; i <= mxd; i++) //初始化
            fa[i] = i;
        sort(gds + 1, gds + 1 + n, cmp); //排序后贪心
        for (int i = 1; i <= n; i++)
        {
            int fi = find(gds[i].second);
            if (fi) //如果fi!=0，说明还有日期可卖此商品
            {
                ans += gds[i].first;
                fa[fi] = fi - 1; //令该日期被使用
            }
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

参考资料

《算法竞赛进阶指南》

AcWing.com

并查集 - OI wiki

扩展阅读

可持久化

重构树

树上启发式合并

最小生成树

LCA