

并查集

简单介绍

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

引入 P1551 亲戚

• 操作: 初始化

假定我们有若干个元素, 这些元素互相之间并没任何关系, 那么我们需要区别这些元素。

不妨令这些元素的下标为 $1 \sim n$

与离散教科书里的等价类相似, 我们对于一个集合, 提取出来一个代表元表示这个集合。

并且刚开始, 我们定义其代表集合的元素为本身。

设查找代表元的操作为 $F(x)$

```
1 for(int i = 1; i <= n ; i ++ ) F[i] = i;
```

• 操作: 并

对于两个不相交的集合若要将其合并, 那么我们只需要改变其代表元即可。

例如, 我们有 4 倍数的集合, 和元素有 $x * 4 - 2$ 的两个集合。

$$S_1 = \{4, 8, 12, 16 \dots\}$$

$$S_2 = \{2, 6, 10, 14 \dots\}$$

取 S_1 的代表元 4, 和 S_2 的代表元 2。

然后将 $F(4)$ 改为 2 即可

```
1  F(4) = 2 // 以下写作 fa
```

• 操作：查

如何查找呢，有上面的操作，我们便可以知道，我们可以通过递归或者循环一直迭代到代表元即可。

```
1  int find(int x){
2      if(fa[x] == x) return x;
3      return find(fa[x]);
4  }
5
6  // or
7
8  int find(int t){
9      int t;
10     while(fa[t] != t){
11         t = fa[t];
12     }
13     return t;
14 }
```

随后，我们 S_1 中所有的集合的代表元都可以找到 4，然后 4 就可以找到 2

如此，简单的并查集已经完成了。

如上，我们不难发现，这个结构非常像树的结构

其次，我们每次操作只要找到代表元即可，那么，对我们来说，这个树的中间值，其实根本不重要。我们只要知道代表元就行了。

因此，我们还需要改变中间变量。

• 操作：路径压缩

代码操作起来也非常简单

```

1  int find(int x){
2      if(fa[x] == x) return x;
3      return fa[x] = find(fa[x]);
4  }
5
6  // or
7
8  int find(int x){
9      t = x;
10     while(t != fa[t]) t = fa[t];
11     while(x != fa[x]) x = fa[x], fa[x] = t;
12     return t;
13 }

```

最后代码可变为这样

```

1  int find(int x){
2      if(fa[x] == x) return x;
3      return fa[x] = find(fa[x]);
4  }
5
6  // or
7
8  int find(int x){
9      t = x;
10     while(t != fa[t]) t = fa[t];
11     while(x != fa[x]) x = fa[x], fa[x] = t;
12     return t;
13 }
14
15 void merge(int x ,int y){ fa[find(x)] = find(y); }
16

```

那么上面的题目的答案就是

[code](#)

下面来看几个简单的应用

• 例题 A1 - 【模板】并查集

[code](#)

Okay ~ 有没有发现这两个题目描述的就是上面的东西呢？

这些不再讲解，下面稍微加强一下。

• 例题 A2 - [Acwing.237 程序自动分析](#)

哈希+并查集



image-20210810164936527

离散化，即可。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  int fa[200005]; // fa数组
6  pair<int,int> e0[200005];
7      //存储需要不同值的数对，统一判冲
8  map<int,int> mp; //映射表
9  int c, ce0;
10
11 int find(int x)
12 {
13     if(fa[x]==x) return x;
14     return fa[x]=find(fa[x]);
15 }
16
17 int main()
18 {
19     int t,n,a,b,e;
20     cin>>t;
21     while(t-->0)
22     {
23         mp.clear();
24         c=1, ce0=0; //初始化
25         scanf("%d",&n);
26         int f=1;
27         for(int i=1;i<=200000;i++) //初始化
28         {
29             fa[i]=i;
30         }
31         for(int i=1;i<=n;i++)
32         {
33             scanf("%d%d%d",&a,&b,&e);
34             if(!mp[a]) mp[a]=c++;
35             if(!mp[b]) mp[b]=c++;
36             //若值不存在，则为其分配哈希值
37             a=mp[a], b=mp[b];
38             if(!e)
39             {
40                 e0[ce0++]={a,b};
41             }
42             else
43             {
```

```

44         fa[find(a)]=find(b); //合并
45     }
46 }
47 for(int i=0;i<ce0&&f;i++) //处理需要不同值的值对
48 {
49     if(find(e0[i].first)==find(e0[i].second))f=0;
50 }
51 if(f)printf("YES\n");
52 else printf("NO\n");
53 }
54 return 0;
55 }

```

• 例题 A3 - 点一成零

并查集+逆元

[code](#)

• 例题 A4 - 天空之城

Kruskal最小生成树

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；

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  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
6  ll len[5003];           //len存以该城为根的集合内部的连通代价
7
8  struct road
9  {
10     int fr, to;
11     ll lrd;
12 } rod[200005];
13
14 bool cmp(road a, road b)
15 {
16     return a.lrd < b.lrd;
17 }
18
19 map<string, int> mp; //存城市名与编号的映射
20
21 int find(int x)

```

```

22     {
23         if (fa[x] != x)
24             fa[x] = find(fa[x]);
25         return fa[x];
26     }
27
28     int main()
29     {
30         int n, q, i;
31         while (~scanf("%d%d", &n, &q))
32         {
33             mp.clear();
34             int ccty = 0, glen;
35             string g1, g2;
36             cin >> g1;
37             mp[g1] = (++ccty); //将开始城设为1
38             for (i = 1; i <= q; i++)
39             {
40                 cin >> g1;
41                 cin >> g2;
42                 scanf("%d", &glen);
43                 if (!mp[g1])
44                     mp[g1] = ++ccty;
45                 if (!mp[g2])
46                     mp[g2] = ++ccty;
47                 rod[i].fr = mp[g2], rod[i].to = mp[g1], rod[i].lrd = glen; //存路
48             }
49             if (ccty != n)
50             {
51                 printf("No!\n");
52                 continue;
53             } //如果提到的城市数!=n, 则必有城市在路网之外
54             sort(rod + 1, rod + 1 + q, cmp);
55             for (i = 1; i <= n; i++)
56                 fa[i] = i, cnt[i] = 1, len[i] = 0; //初始化并查集
57             for (i = 1; i <= q; i++)
58             {
59                 int frn = rod[i].fr, ton = rod[i].to;
60                 if (find(frn) != find(ton))
61                 {
62                     if (find(frn) > find(ton))
63                         swap(frn, ton); //保证两集以较小
64
65                     根为新根
66                     cnt[find(frn)] += cnt[find(ton)]; //处理城市数
67                     len[find(frn)] += len[find(ton)] + rod[i].lrd; //处理连通代价
68                     fa[find(ton)] = find(frn);
69                 }
70             }
71             if (cnt[1] == n)
72                 printf("%lld\n", len[1]);
73             else
74                 printf("No!\n");

```

```
73     }
74     return 0;
75 }
76
```

[code](#)

"扩展域"和“带边权”的并查集

边权： 我们可以想象这是一个个树木组成的森林，这些树与树之间的连线都有一个值，我们便可以把这个值成为边权

扩展域： 将一颗树扩展成多棵树，这样既可以根据相互之间的关系来推演

• 例题B1 - [银河英雄传说](#) -- 带边权

[code](#)

• 例题B2 - [关押犯罪](#) -- 带边权 / 扩展域

扩展域：


 image-20210810165601669

 image-20210810165626047

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const ll N = 100005;
5
6  int fa[N] = {0};
7  struct dat
8  {
9      int a, b, c;
10 } da[N];
11
12 bool cmp(dat a, dat b)
13 {
14     return a.c > b.c;
15 }
16
17 int find(int x)
18 {
```

```

19     return fa[x] == x ? x : fa[x] = find(fa[x]);
20 }
21
22 int main()
23 {
24     int n, m, i;
25     cin >> n >> m;
26     for (i = 1; i <= m; i++)
27         scanf("%d%d%d", &da[i].a, &da[i].b, &da[i].c);
28     sort(da + 1, da + 1 + m, cmp);
29     for (i = 1; i <= 2 * n; i++) fa[i] = i; //初始化
30     for (i = 1; i <= m; i++) //贪心，从大到小找到第一个发生冲突的
31     {
32         if (find(da[i].a) == find(da[i].b)) //存在冲突
33         {
34             printf("%d", da[i].c);
35             return 0;
36         }
37         else
38         {
39             fa[find(da[i].a)] = find(da[i].b + n);
40             fa[find(da[i].b)] = find(da[i].a + n);
41             //把一个加入另一个的扩展域，代表放入相异的集合
42         }
43     }
44     printf("0");
45     return 0;
46 }

```

带边权：

[code](#)

• 例题B3 - 食物链 -- 扩展域 / 带边权

- 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；

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  int fa[150004];

```



```

5  int find(int x)
6  {
7      if(fa[x]==x)return x;
8      else return find(fa[x]);
9  }
10 int main()
11 {
12     int n,k,ans=0,o,a,b,i;
13     cin>>n>>k;
14     for(i=1;i<=3*n;i++)fa[i]=i;
15     while(k--)
16     {
17         scanf("%d%d%d",&o,&a,&b);
18         if(a>n||b>n){ans++;continue;}
19         if(o==1)//同类
20         {
21             if(find(a+n)==find(b)||find(a+2*n)==find(b))ans++;
22             else
23             {
24                 fa[find(a)]=find(b);
25                 fa[find(a+n)]=find(b+n);
26                 fa[find(a+2*n)]=find(b+2*n);
27                 //按同类维护
28             }
29         }
30         else if(o==2)//a吃b
31         {
32             if(find(a)==find(b)||find(a)==find(b+n))ans++;
33             else
34             {
35                 fa[find(a+n)]=find(b);
36                 fa[find(a+2*n)]=find(b+n);
37                 fa[find(a)]=find(b+2*n);
38             }
39         }
40     }
41     cout<<ans;
42 }

```

- 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;

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 5e4 + 4;

```

```

4   int fa[N];
5   int d[N];
6   int n, k;
7   int res = 0;
8   void init()
9   {
10      for (int i = 1; i <= n; i++)
11          fa[i] = i;
12  }
13  int find(int x) //路径压缩
14  {
15      if (x != fa[x])
16      {
17          int t = find(fa[x]); //保存好p[x],以免直接将p[x]变为根节点
18          d[x] += d[fa[x]];
19          fa[x] = t;
20      }
21      return fa[x];
22  }
23  int main()
24  {
25      scanf("%d%d", &n, &k);
26      init();
27      while (k--)
28      {
29          int x, a, b;
30          scanf("%d%d%d", &x, &a, &b);
31          if (a > n || b > n)
32              res++;
33          else
34          {
35              int faa = find(a), fab = find(b);
36              if (x == 1)
37              {
38                  if (faa == fab && (d[a] - d[b]) % 3)
39                      res++; //判断在同一集合时,判断其距离是否满足
40                  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
41                      {
42                          fa[faa] = fab;
43                          d[faa] = d[b] - d[a]; //d[a]+d[faa]=d[b]
44                      }
45              }
46              else
47              {
48                  if (faa == fab && (d[a] - d[b] - 1) % 3)
49                      res++;
50                  else if (faa != fab)
51                  {
52                      fa[faa] = fab;
53                      d[faa] = d[b] - d[a] + 1; //d[a]+d[faa]=d[b]+1
54                  }

```

```
55         }
56     }
57 }
58 printf("%d\n", res);
59 return 0;
60 }
```

例题与习题

例题：

题目名称	来源	涉及算法
亲戚	洛谷	并查集
并查集	洛谷	并查集
程序自动分析	Acwing	哈希 并查集
点一成零 (付费题)	牛客竞赛	并查集 逆元
天空之城	牛客竞赛	生成树 并查集
银河英雄传说	Acwing	带边权的并查集
关押犯罪	Acwing	带边权 / 扩展域 并查集
食物链	Acwing	带边权 / 扩展域 并查集

习题

A 组

题目名称	来源	涉及算法
最小生成树	LibreOJ	生成树 并查集
City	东北赛	生成树 并查集
Igor In the Museum	Codeforces	并查集 dfs
X-Plosives	UVA	并查集
Corporative Network	UVA	并查集
Junk-Mail Filter	HDU	并查集
修复公路	计算客	并查集
奶酪	LibreOJ	并查集

B 组

题目名称	来源	涉及算法
Islands	UVA	枚举 排序 并查集
Exclusive-OR	UVA	并查集
Destroying Array	Codeforces	并查集 线段树
The Door Problem	codeforces	并查集 2-SAT
9102	计蒜客	并查集
The Child and Zoo	Codeforces	并查集
奇偶游戏	POJ	并查集

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