### 集合与并查集

在修建道路时，为了让尽可能多的点连通，需要修建连通两个点的公路,这就需要随时询问两个点是否已经连通。

若将已经连通的点看作一个集合，那么修建一条公路的意义，就是合并两个集合，那么如何快速查询两个点是否是属于同一个集合，以及快速的合并两个集合，十分重要。

§3.1 集合与并查集

**一、 集合与并查集的概念**

**集合**是由一个或多个确定的元素所构成的整体。集合中的元素如下三个特征：

确定性：一个元素要么属于集合，要么不属于集合。

互异性：集合中的元素互不相同。

无序性：集合中的元素没有先后顺序。

**并查集**是一个可以维护集合的数据结构，它能高效支持集合的基本操作。

1、合并两个集合。

2、查询两个指定元素是否属于同一个集合。

需要注意的是，由于计算机存储结构的限制，并查集维护的集合是离散意义下的集合，而不是广义的集合。集合中的元素是有限的。

**二、集合的存储**

1、数组存储

存储一个集合最简单的方式就是直接用数组。我们将一个集合的所有元素按某种特定顺序存储在数组里。使用数组存储集合，可以支持较丰富的集合操作，但是维护集合的时间复杂度较高，对于几乎所有操作，单次操作的时间复杂度都是和集合大小成正比的。

在 C++ 语言中，我们可以使用 STL 中的 vector 来实现数组存储集合。

2、链表存储

可以模仿数组存储方式，将集合中的元素存储在一个链表里。链表的一大好处是可以避免元素的复制，这对于合并操作是比较有帮助的，在定位元素所在的集合的两端后，直接将两个集合的端点相接即可合并完成。

然而，使用链表维护集合的最坏情况时间复杂度仍然是与集合大小成正比的，在时间效率上还不够优秀。

在 C++ 语言中，我们可以使用 STL 中的 list 来实现链表存储集合。如果题目对时间效率的要求较高，也可以选择自行实现链表。

值得一提的是，假设元素总数是 N，且仅需要支持合并和查询操作，那么上述两种方式可以采用启发式合并的技术（即每次将较小集合合并入较大集合并修改较小集合所有元素的信息）做到总时间复杂度 O(N\*log2 N)。虽然单次操作的时间复杂度较高，但是可以证明总时间复杂度是可以接受的。

3、[森林存储](#_Toc1127886396)

[注意到一个元素只可能属于一个集合，所以我们可以为每一个集合 G 选取一个代表元 w(G)。于是查询两个元素 u, v 是否属于同一集合实际上就是询问对于 u 所属的集合 U 和 v 所属的集合V，是否有 w(U) = w(V)，即集合 U, V 的代表元是否相同。](#_Toc1127886396)

[通过上述转化，我们就把需要维护的信息从比较难以维护的集合转化为了较容易维护的元素。对于每个元素，维护它所属的集合的代表元，查询时直接比较即可。](#_Toc1127886396)

[但是合并时需要改变其中一个集合中所有元素的代表元，时间复杂度仍然非常高，如何优化呢？](#_Toc1127886396)

[注意到，合并操作的复杂度远高于查询操作的复杂度，这启发我们通过一定的方式，提高查询操作的复杂度，降低合并操作的复杂度。](#_Toc1127886396)

[我们并不需要 O(1) 知道每个元素所属集合的代表元，这启发我们用森林来维护代表元。对于每一个元素 x, 维护其父亲 fa[x]。特殊地，对于属于一个集合的代表元，有fa[x] == x。一开始每个元素都是其所在集合的代表元，即 fa[x] = x。](#_Toc1127886396)

[当需要合并两个元素 u, v 所在集合时，我们先找到 u, v 所在集合 U, V 的代表元 w(U) 和 w(V)，令 fa[w(U)] = fa[w(V)] 或 fa[w(V)] = fa[w(U)]。](#_Toc1127886396)

[如何找到一个元素所在集合的代表元呢？根据上述的合并方式，容易发现，对于一个点 x，我们不停找 x 的父亲，即令 x = fa[x]，直到 x 是代表元时 (fa[x] == x) 停止，此时 x 就是我们要找的代表元了。](#_Toc1127886396)

[相关代码如下：](#_Toc1127886396)

[const int maxn = 100000;](#_Toc1127886396)

[int fa[maxn];](#_Toc1127886396)

[int Find(int x) {](#_Toc1127886396)

[if (fa[x] == x) return x;](#_Toc1127886396)

[return Find(fa[x]);](#_Toc1127886396)

[}](#_Toc1127886396)

[void Union(int u, int v) {](#_Toc1127886396)

[if (Find(u) != Find(v)) fa[Find(u)] = Find(v);](#_Toc1127886396)

[}](#_Toc1127886396)

不幸地是，上述方法看似优秀，实际上“找代表元”这一操作的最坏情况时间复杂度仍然比较高，因为森林的深度可能比较大。如果我们不停地从一个深度较大的点向上寻找代表元，时间复杂度就令人难以接受。

下面我们来讲讲上述操作的优化操作。

§3.2 并查集的基本操作

**一、 启发式合并**

运用启发式合并的思想，我们尝试每次将较小的集合并入较大的集合。由于要判断集合之间的大小关系，每一个元素需要维护其所在集合的大小。可以证明，在运用启发式合并技术后，并查集的总时间复杂度将至 O(N\*log2N)。

相关代码如下：

const int maxn = 100000;

int fa[maxn], siz[maxn];

int Find(int x) {

if (fa[x] == x) return x;

return Find(fa[x]);

}

void Union(int u, int v) {

u = Find(u); v = Find(v);

if (siz[u] > siz[v]) swap(u, v);

fa[u] = v; siz[v] += siz[u];

}

**二、路径压缩技术**

更进一步地，我们发现集合中只有代表元是有用的，而寻找代表元时经过中间点是冗余的，如果我们将这些中间点在访问过后直接将它们的父亲指向集合的代表元，在后续访问中，就不会产生这一部分的冗余了。

这是一个非常大的优化。可以证明，结合启发式合并技术，并查集运用这种路径压缩技术后时间复杂度是 ，其中 是阿克曼函数的反函数，增长非常缓慢，在 N 可接受的范围内，一般可以认为是不超过 4 的常数。

相关代码如下：

const int maxn = 100000;

int fa[maxn], siz[maxn];

int Find(int x) {

if (fa[x] == x) return x;

return fa[x] = Find(fa[x]);

}

void Union(int u, int v) {

u = Find(u); v = Find(v);

if (siz[u] > siz[v]) swap(u, v);

fa[u] = v; siz[v] += siz[u];

}

§3.3 并查集的应用

**例3.1**  亲戚

或许你并不知道，你的某个朋友是你的亲戚。他可能是你的曾祖父的外公的女婿的外甥的表姐的孙子。如果能得到完整的家谱，判断两个人是否亲戚应该是可行的， 但如果两个人的最近公共祖先与他们相隔好几代，使得家谱十分庞大，那么检验亲戚关系实非人力所能及.在这种情况下，最好的帮手就是计算机。  
 为了将问题简化，你将得到一些亲戚关系的信息，如同Marry和Tom是亲戚，Tom和Ben是亲戚，等等。从这些信息中，你可以推出Marry和Ben是亲戚。请写一个程序，对于我们的关心的亲戚关系的提问，以最快的速度给出答案。问题涉及的人的个数不超过20000，所给出的亲戚关系数量不超过1000000，询问数不超过1000000。

**解答：**

本题比较简单，可以直接套用并查集的模型，将同一个家族（互为亲戚）的人合并到同一个集合内，询问时只要查询两个人是否属于同一个集合即可作出判断。

**例3.2** 银河英雄传说[[1]](#footnote-0)

公元五八○一年，地球居民迁至金牛座α第二行星，在那里发表银河联邦创立宣言，同年改元为宇宙历元年，并开始向银河系深处拓展。  
 宇宙历七九九年，银河系的两大军事集团在巴米利恩星域爆发战争。泰山压顶集团派宇宙舰队司令莱因哈特率领十万余艘战舰出征，气吞山河集团点名将杨威利组织麾下三万艘战舰迎敌。  
 杨威利擅长排兵布阵，巧妙运用各种战术屡次以少胜多，难免恣生骄气。在这次决战中，他将巴米利恩星域战场划分成 列，每列依次编号为 。之后，他把自己的战舰也依次编号为 ，让第 号战舰处于第 列 ，形成“一字长蛇阵”，诱敌深入。这是初始阵形。当进犯之敌到达时，杨威利会多次发布合并指令，将大部分战舰集中在某几列上，实施密集攻击。合并指令为 ，含义为第 号战舰所在的整个战舰队列，作为一个整体（头在前尾在后）接至第 号战舰所在的战舰队列的尾部。显然战舰队列是由处于同一列的一个或多个战舰组成的。合并指令的执行结果会使队列增大。  
 然而，老谋深算的莱因哈特早已在战略上取得了主动。在交战中，他可以通过庞大的情报网络随时监听杨威利的舰队调动指令。  
 在杨威利发布指令调动舰队的同时，莱因哈特为了及时了解当前杨威利的战舰分布情况，也会发出一些询问指令： 。该指令意思是，询问电脑，杨威利的第 号战舰与第 号战舰当前是否在同一列中，如果在同一列中，那么它们之间布置有多少战舰。  
 作为一个资深的高级程序设计员，你被要求编写程序分析杨威利的指令，以及回答莱因哈特的询问。  
 最终的决战已经展开，银河的历史又翻过了一页……  
 指令共有 条。

**解答：**

在维护并查集的同时维护更多的信息。我们使每一艘战舰的fa为其所在列的头部的战舰。那么就可以维护每个点到其目前fa的距离dis。在进行Find操作时，如果修改了某个点的fa，那么相应的也修改这个点的dis。更具体地，新的dis值为原来的dis值加上原来fa的dis值。查询时，只需先查询两个点是否在同一个集合内，之后答案就是这两个点的dis之差。注意在合并的时候，由于要保证每个点的fa是其列头的点，所以进行Union操作时两点的顺序要注意。  
 时间复杂度 。

**例3.3**  关押罪犯[[2]](#footnote-1)

S城现有两座监狱，一共关押着 名罪犯，编号分别为 。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 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 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 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？  
 存在仇恨的罪犯对数不超过 。

**解答：**

首先，应用贪心的思想，我们将矛盾关系根据仇恨值从大到小排序。这样我们依次将每一对罪犯拆开安放在两个监狱中，直到无法做到为止，所得的结果就是尽可能小的。  
 如果有两个罪犯要被拆开，那么我们称这两个罪犯互相视为敌人。根据“敌人的敌人是我的朋友”的思想，我们维护每个罪犯其本身以及其敌人的并查集。我们用x表示一个罪犯，并用x’表示其敌人。注意这里的“敌人”只代表一种关系，而非某个确切的敌人。当要拆开一对罪犯x和y时，我们在并查集中合并x和y’，同时合并x’和y。如果在合并前发现x和y已经在一个集合中，或者x’和y’已经在一个集合中，那么这一对罪犯将无法再被拆开。此时这对关系的仇恨值即为答案。

**例3.4**  食物链[[3]](#footnote-2)

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。  
 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。  
 你的任务是根据给定的N（1<=N<=50,000）和K句话（0<=K<=100,000），输出假话的总数。

**解答：**

不妨对于每只动物 i 创建 3 个元素 A(i), B(i), C(i) 分别代表动物i 属于集合 A, B, C 的情况。以这 3N 个元素创建并查集，两个元素属于同一集合表示这两种情况同时发生或不发生。

对于两种类型的话，

1. x 和 y 同类：分别合并 A(x) 和 A(y), B(x) 和 B(y), C(x) 和 C(y) 所在集合。
2. X 吃 y，分别合并 A(x) 和 B(y), B(x) 和 C(y), C(x) 和 A(y) 所在集合。

在执行合并前，先判断是否已经产生冲突，如果产生冲突，则这句话是假话。

时间复杂度 。

本章小结

本章讲述了集合和并查集的基本定义，以及如何使用并查集这一数据结构高效地完成集合的一些基本操作，并给出了并查集的两种优化技术。并查集在联赛提高组及以上难度的题目中较为常见。它简洁但并不无用，常常在其他算法中作为一个高效的辅助工具出现。

1. 该题来源于NOI2002 [↑](#footnote-ref-0)
2. 该题来源于NOIP2010 [↑](#footnote-ref-1)
3. 该题来源于poj1182 [↑](#footnote-ref-2)