# 并查集

李淳风

长郡中学

2024 年 6 月 1 日

# 并查集

并查集作为一种常用的数据结构，用于动态维护若干个不重叠的集合，并支持合并与查询。详细地说，并查集包括以下两个基本操作：

- Get，查询一个元素属于哪一个集合；
- Merge，把两个集合合并成一个大集合。

# 并查集

并查集作为一种常用的数据结构，用于动态维护若干个不重叠的集合，并支持合并与查询。详细地说，并查集包括以下两个基本操作：

- Get，查询一个元素属于哪一个集合；
- Merge，把两个集合合并成一个大集合。

要实现上述操作，首先我们需要思考如何表示一个集合。
我们可以给每一个集合单独编号，用编号来区分集合。但注意到需要维护的若干个集合没有重叠，所以我们可以直接为每个集合选择一个该集合中的元素，作为整个集合的“代表”。

# 并查集

并查集作为一种常用的数据结构，用于动态维护若干个不重叠的集合，并支持合并与查询。详细地说，并查集包括以下两个基本操作：

- Get，查询一个元素属于哪一个集合；
- Merge，把两个集合合并成一个大集合。

要实现上述操作，首先我们需要思考如何表示一个集合。
我们可以给每一个集合单独编号，用编号来区分集合。但注意到需要维护的若干个集合没有重叠，所以我们可以直接为每个集合选择一个该集合中的元素，作为整个集合的“代表”。
之后，我们需要判断一个元素属于哪个集合。我们可以直接维护一个数组 $f$，用 $f_x$ 表示元素 $x$ 所处集合的“代表”。这样可以快速查询，但是在合并集合时需要进行大量修改。

# 并查集

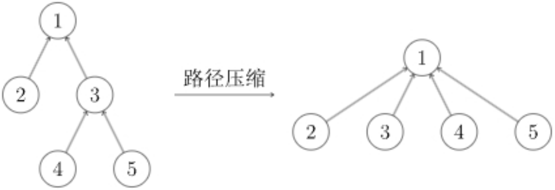
并查集作为一种常用的数据结构，用于动态维护若干个不重叠的集合，并支持合并与查询。详细地说，并查集包括以下两个基本操作：

- Get，查询一个元素属于哪一个集合；
- Merge，把两个集合合并成一个大集合。

要实现上述操作，首先我们需要思考如何表示一个集合。
我们可以给每一个集合单独编号，用编号来区分集合。但注意到需要维护的若干个集合没有重叠，所以我们可以直接为每个集合选择一个该集合中的元素，作为整个集合的“代表”。
之后，我们需要判断一个元素属于哪个集合。我们可以直接维护一个数组 $f$，用 $f_x$ 表示元素 $x$ 所处集合的“代表”。这样可以快速查询，但是在合并集合时需要进行大量修改。
另一种思路是使用一个树形结构存储每个集合，树上的每个节点都是一个元素，树根是集合的代表元素，整个并查集就是一个森林（由若干棵树组成）。我们可以维护一个 $fa$ 数组来表示这个森林，$fa_x$ 表示 $x$ 的父节点。这样在合并时只需要连接两个树根即可，但在查询时需要通过 $fa$ 数组来找到根节点。为了提高查询效率，并查集引入了路径压缩和按秩合并两种思想。

# 路径压缩

实际上，我们在查询时只关心每个集合的“代表”，也就是每棵树的根节点是什么，但对一棵树的具体形态并不关心。例如，下图中的两棵树对于我们而言是等价的：




因此我们可以在每次执行 Get 操作的同时，把访问过的每个节点（也就是所查询元素的全部祖先）都直接指向树根。这种优化方式被称为路径压缩。采用路径压缩优化的并查集，每次 Get 操作的均摊复杂度为 $O(\log N)$。

# 路径压缩

普通查询代码：

```
int find(int x){//查询 x 所在集合的代表
  return fa[x]==x?x:find(fa[x]);
}
```

路径压缩代码：

```
int find(int x){//查询 x 所在集合的代表
  return fa[x]==x?x:fa[x]=find(fa[x]);
}
```

大部分情况下，并查集只需要一个路径压缩就能达到要求的复杂度了。但一定要注意的是，只采用路径压缩优化的并查集，最坏情况下的复杂度是 $O(N\log N)$。

# 按秩合并

按秩合并，是并查集的第二种优化。所谓“秩”，有两种定义，既可以指树的深度，也可以指集合的大小。无论采用哪种定义，我们都可以把集合的秩存储在树根上，在合并的时候比较两个根节点的秩，把秩较小的根节点作为秩更大的根节点的子节点。
当“秩”被定义为集合的大小时，“按秩合并”也称为“启发式合并”，是数据结构中一种重要的思想，不局限于并查集中。启发式合并的原则是，合并两个数据结构时，把“小的结构”合并到“大的结构”中，并且本次合并的代价只与“小的结构”有关。这样一来，如果每次合并是线性复杂度，那么全部合并的总复杂度就是 $O(N \log N)$。
所以单独采用“按秩合并”优化（秩被定义为集合大小）的并查集，每次 Get 操作的均摊复杂度也是 $O(\log N)$。

# 简单实现

同时采用“路径压缩”和“按秩合并”优化的并查集，每次 Get 操作的均摊复杂度可以进一步降低到 $O(\alpha(n))$。其中 $\alpha(n)$ 为反阿克曼函数，它是一个比 $\log N$ 增长还要慢的函数，$\forall N \leq 2^{2^{10^{19729}}}$，都有 $\alpha(N) \leq 5$，所以我们在使用中可以把它近似为一个常数。
在实际使用中，我们一般只用路径压缩就足够了。
并查集的初始化，最初每个元素各自构成一个集合：

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

并查集的 Get 操作：

```
int Get(int x){//查询 x 所在集合的代表
    return fa[x]==x?x:fa[x]=Get(fa[x]);
}
```

并查集的 Merge 操作：

```
void Merge(int x,int y){//合并 x 和 y 所在的集合
    int a=Get(x),b=Get(y);
    if(a==b) return;
    fa[a]=b;
}
```

# 例题

## 程序自动分析

有 $m$ 个变量 $x_1, x_2, \cdots, x_m$，和 $n$ 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，请判断是否可以为每个变量赋予一个合适的值，使得上述约束条件同时满足。
$1 \leq n \leq 10^5, 1 \leq m \leq 10^9$

# 例题

## 程序自动分析

有 $m$ 个变量 $x_1, x_2, \cdots, x_m$，和 $n$ 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，请判断是否可以为每个变量赋予一个合适的值，使得上述约束条件同时满足。
$1 \leq n \leq 10^5, 1 \leq m \leq 10^9$

首先，虽然 $m$ 很大，但是 $n$ 不大，而我们有用的变量最多只有 $2n$ 个，因此我们可以首先进行离散化操作。
接着，如果我们把一个变量看作无向图中的一个点，那么每个相等关系就对应无向图中的一条边，所有相等的点构成了一个连通块。因此，我们可以直接使用并查集进行维护，碰到相等关系就把两个点所在的集合合并。
最后我们再扫描所有的不等关系，判断所有不等号两边的两个点是否在同一个集合内即可。

# 例题

## Supermarket

一共有 $n$ 个商品，每个商品有利润 $p_i$ 和过期时间 $d_i$，每天只能卖一个商品，过期商品不能再卖。求如何操作才能使利润最大。
$1 \leq n, p_i, d_i \leq 10000$。

# 例题

## Supermarket

一共有 $n$ 个商品，每个商品有利润 $p_i$ 和过期时间 $d_i$，每天只能卖一个商品，过期商品不能再卖。求如何操作才能使利润最大。
$1 \le n, p_i, d_i \le 10000$。

这道题是一道贪心题，这里给出一种使用了并查集的做法。
对于每一个商品，我们肯定尽量晚卖出，这样才能给其它商品留下更充足的时间。所以我们把所有商品按照利润从大到小排序，依次考虑能否卖出，现在要做的就是找出每一个商品能卖出的最晚时间。
我们根据天数建立一个并查集，起初每一天自己构成一个集合。对于一件商品，如果它在 $d$ 天之后过期，就在并查集中查询 $d$ 的树根，记为 $r$。若 $r$ 大于 $0$，我们就可以在第 $r$ 天把商品卖出，并把 $r$ 的父亲设为 $r-1$，表示第 $r$ 天已经被占用。

# 并查集

从这两道题可以看出，并查集擅长对于维护具有传递性的关系。例如 A 与 B 有某种关系，B 与 C 也有这种关系，那么可以推出 A 与 C 也有关系，这就叫传递性。例如程序自动分析中，无向图的连通性就具有传递性，这类关系基本都可以使用并查集来维护。
而在 Supermarket 这道题中，我们的并查集实际上维护了一个数组中“位置”的占用情况。每个“位置”所在集合的代表就是从它开始往前数，第一个空闲的位置。当某个位置被占用，就把它合并到前一个位置所在的集合中去。需要注意的是如果这样做，就不能使用按秩合并优化了，只能使用路径压缩。

# 带权并查集

并查集维护的是若干棵树，而我们知道，树上的边是可以带边权的。实际上，并查集也可以带上边权。
我们新开一个数组 $d$，$d_x$ 表示 $x$ 到它的父亲 $fa_x$ 之间的边权。
而这个数组也可以在路径压缩时方便地一起维护了：

```
int Get(int x){//查询 x 所在集合的代表
  if(fa[x]==x) return x;
  int root=Get(fa[x]);
  d[x]+=d[fa[x]];
  return fa[x]=root;
}
```

# 例题

## 银河英雄传说

有一个划分为 N 列地星际战场和 N 艘战舰，初始时第 $i$ 艘战舰位于第 $i$ 列。现在有 M 条指令，每条指令为下列两种格式之一。

- M i j，表示让第 $i$ 号战舰所在列的全部战舰保持原有顺序，接在第 j 号战舰所在列的尾部。
- C i j，表示询问第 $i$ 号战舰和第 $j$ 号战舰当前是否处于同一列中，如果是，则输出它们之间间隔了多少战舰。

$N \leq 30000, M \leq 5 * 10^5$。

# 例题

## 银河英雄传说

有一个划分为 N 列地星际战场和 N 艘战舰，初始时第 $i$ 艘战舰位于第 $i$ 列。现在有 M 条指令，每条指令为下列两种格式之一。

- M i j，表示让第 $i$ 号战舰所在列的全部战舰保持原有顺序，接在第 j 号战舰所在列的尾部。
- C i j，表示询问第 $i$ 号战舰和第 $j$ 号战舰当前是否处于同一列中，如果是，则输出它们之间间隔了多少战舰。

$N \le 30000, M \le 5 * 10^5$。

首先，如果只询问两艘战舰是否处在同一列，那么就是并查集板子题。现在要知道间隔，我们该怎么办呢？上一页已经给出答案了，就是带边权的并查集。

# 银河英雄传说

由于在这里战舰只会排成一列，我们只需要知道两艘战舰到队首的距离，就能知道它们之间的距离。
所以我们需要在并查集中维护每个点到树的根节点的距离。
Get 函数之前已经讲了，现在再来看看 Merge 函数。
当我们合并两个集合，也就是两列战舰时，会给排在后面的所有战舰的 $d$ 数组，加上前面那些战舰的数量。
所以我们还需要另外维护一个 $size$ 数组，$size_x$ 表示 $x$ 的集合大小。

# 银河英雄传说

由于在这里战舰只会排成一列，我们只需要知道两艘战舰到队首的距离，就能知道它们之间的距离。
所以我们需要在并查集中维护每个点到树的根节点的距离。
Get 函数之前已经讲了，现在再来看看 Merge 函数。
当我们合并两个集合，也就是两列战舰时，会给排在后面的所有战舰的 $d$ 数组，加上前面那些战舰的数量。
所以我们还需要另外维护一个 $size$ 数组，$size_x$ 表示 $x$ 的集合大小。

```
void Merge(int x,int y){//合并 x 和 y 所在的集合
  int a=Get(x),b=Get(y);
  if(a==b) return;
  fa[a]=b,d[a]=size[b];
  size[b]+=size[a];
}
```

# 例题

## Parity game

小 A 和小 B 在玩一个游戏。首先，小 A 写了一个由 0 和 1 组成的序列 S，长度为 N。
然后，小 B 向小 A 提出了 M 个问题，每次询问指定两个数 $l$ 和 $r$，小 A 回答 $S_l, S_l+1, \cdots, S_r$ 中存在奇数个 1 还是偶数个 1。
但是小 B 发现小 A 的回答出现了自相矛盾的情况。现在请你检查这 M 个回答，并指出在多少个回答之后可以确定小 A 的回答自相矛盾。
$N \leq 10^9, M \leq 10000$

# 例题

## Parity game

小 A 和小 B 在玩一个游戏。首先，小 A 写了一个由 0 和 1 组成的序列 S，长度为 N。
然后，小 B 向小 A 提出了 M 个问题，每次询问指定两个数 $l$ 和 $r$，小 A 回答 $S_l, S_l + 1, \cdots, S_r$ 中存在奇数个 1 还是偶数个 1。
但是小 B 发现小 A 的回答出现了自相矛盾的情况。现在请你检查这 M 个回答，并指出在多少个回答之后可以确定小 A 的回答自相矛盾。
$N \leq 10^9, M \leq 10000$

同样可以发现 $N$ 那么大就是逗你玩的，有用的最多只有 $2M$ 个。
其次像这一类涉及到区间和的问题，一般可以尝试着往前缀和的方向想想看。
用 W 来表示序列 S 的前缀和，那么如果 $[l, r]$ 中有偶数个 1，等价于 $W_r$ 与 $W_{l-1}$ 奇偶性相同，有奇数个 1 则意味着奇偶性不同。
但是奇偶性和相等关系不同，没有办法直接传递，该怎么办呢?

# Parity game

一个比较直接的想法就是同样使用带权并查集。因为带权并查集的权值实际上是一个点到树的根节点的信息，因此只要两个点之间的信息可以通过它们到根节点的信息计算得出，就可以使用带权并查集。
在之前的题目中，两艘战舰之间的间隔是它们到根节点，也就是队首战舰的距离之差；在这道题中，如果我们用 $d_x = 1$ 来表示 $x$ 与 $fa_x$ 奇偶性不同，$d_x = 0$ 表示奇偶性不同，我们同样可以通过计算异或和的方式，通过根节点这么一个中间人来判断同一个集合内，两点之间的奇偶情况。
注意，因为我们并查集的权值都经过了根节点这么一个“中间人”，在合并 $x$，$y$ 所在的两个集合的时候还需要分别考虑 $x$ 到根、$y$ 到根的异或和。设 $ans$ 为小 A 的回答（奇偶性不同时，也就是奇数个 1 时，$ans = 1$），也就要让合并之后 $d_x \ xor \ d_y = ans$。
如果 $x$，$y$ 已经在一个集合中了，我们就需要进行查询操作，通过 $d_x \ xor \ d_y$ 计算得到奇偶性是否相同，再和小 A 的回答做比较。

# Parity game

当然，除了带权并查集之外，我们还有另一种方法，使用“拓展域”的并查集。

把每个节点 $x$ 拆成两个节点 $x_{odd}, x_{even}$，其中 $x_{odd}$ 表示 $W_x$ 是奇数，$x_{even}$ 表示 $W_x$ 是偶数。我们也把这两个节点称为 $x$ 的“奇数域”和“偶数域”。

这样一来，如果 $x$ 和 $y$ 奇偶性不同，就意味着 $x$ 是奇数和 $y$ 是偶数等价，$x$ 是偶数和 $y$ 是奇数等价，把 $x_{odd}$ 和 $y_{even}$ 连边，$x_{even}$ 和 $y_{odd}$ 连边，使用并查集维护无向图连通性。对于奇偶性相同的情况，同理处理即可。

在询问的时候如果 $x_{odd}$ 和 $y_{odd}$ 在同一个集合内，则说明奇偶性相同；若 $x_{odd}$ 和 $y_{even}$ 在同一个集合内，则说明奇偶性不同；否则说明 $x$ 和 $y$ 之间的关系还不能确定，根据回答连边并维护关系即可。

# 例题

## 食物链

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唤醒。其中 A 吃 B，B 吃 C，C 吃 A。

现在有 N 个动物，编号为 1 到 N。每个动物都是 A、B、C 三类中的一类，但我们并不知道它是哪一类。有人用两种说法对这 N 个动物构成的食物链关系进行了描述：

- 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。

此人用这两种说法对这 N 个动物进行了 K 次描述。这 K 句话有真有假，当一句话满足下列三条之一时，就是假话：

- 当前的话与前面的真话冲突，就是假话；
- 当前话中的 X 或 Y 比 N 大，就是假话；
- 当前的话表示 X 吃 X，就是假话。

现在请问这 K 句话中有多少句是假话。$1 \leq N \leq 50000, 0 \leq K \leq 10^5$。

# 食物链

这道题仍然可以使用带权并查集或者“拓展域”的并查集解决。
如果使用“拓展域”方法进行求解的话，同样把每个动物 $x$ 拆成三个节点，同类域 $x_{self}$，捕食域 $x_{eat}$ 和天敌域 $x_{enemy}$。

# 食物链

这道题仍然可以使用带权并查集或者“拓展域”的并查集解决。
如果使用“拓展域”方法进行求解的话，同样把每个动物 $x$ 拆成三个节点，同类域 $x_{self}$，捕食域 $x_{eat}$ 和天敌域 $x_{enemy}$。
若一句话说“$x$ 与 $y$ 是同类”，则说明“$x$ 的同类”与“$y$ 的同类”一样，“$x$ 捕食的物种”与“$y$ 捕食的物种”一样，“$x$ 的天敌”和“$y$ 的天敌”一样。此时，我们给 $x_{self}$ 和 $y_{self}$ 连边，$x_{eat}$ 和 $y_{eat}$ 连边，$x_{enemy}$ 和 $y_{enemy}$ 连边。使用并查集维护无向图的连通性。
若一句话说“$x$ 吃 $y$”，说明“$x$ 捕食的物种”与“$y$ 的同类”一样，“$x$ 的同类”和“$y$ 的天敌”一样，“$x$ 的天敌”和“$y$ 捕食的物种”一样 (因为题目中的食物链为环形)。同样连边用并查集维护即可。

# 食物链

这道题仍然可以使用带权并查集或者“拓展域”的并查集解决。
如果使用“拓展域”方法进行求解的话，同样把每个动物 $x$ 拆成三个节点，同类域 $x_{self}$，捕食域 $x_{eat}$ 和天敌域 $x_{enemy}$。
若一句话说“$x$ 与 $y$ 是同类”，则说明“$x$ 的同类”与“$y$ 的同类”一样，“$x$ 捕食的物种”与“$y$ 捕食的物种”一样，“$x$ 的天敌”和“$y$ 的天敌”一样。此时，我们给 $x_{self}$ 和 $y_{self}$ 连边，$x_{eat}$ 和 $y_{eat}$ 连边，$x_{enemy}$ 和 $y_{enemy}$ 连边。使用并查集维护无向图的连通性。
若一句话说“$x$ 吃 $y$”，说明“$x$ 捕食的物种”与“$y$ 的同类”一样，“$x$ 的同类”和“$y$ 的天敌”一样，“$x$ 的天敌”和“$y$ 捕食的物种”一样 (因为题目中的食物链为环形)。同样连边用并查集维护即可。
在处理每句话之前，我们需要判断它的真假，主要是判断和之前的话是否有着冲突。有两种信息与“$x$ 与 $y$ 是同类”矛盾：

- $x_{eat}$ 与 $y_{self}$ 在同一集合，说明 $x$ 吃 $y$；
- $x_{self}$ 与 $y_{eat}$ 在同一集合，说明 $y$ 吃 $x$。

判断“$x$ 吃 $y$”是否与之前的话矛盾同理。

# 食物链

这道题同样可以使用“边带权”的方法来解决。

我们定义 $d_x = 0$ 表示 $x$ 和 $fa_x$ 是同类，$d_x = 1$ 表示 $x$ 吃 $fa_x$，$d_x = -1$ 表示 $fa_x$ 吃 $x$。由于食物链是一个大小为 3 的环，所以所有边权都可以在模 3 的意义下进行加减法运算。实际上，之前题目中，边权的异或操作就是模 2 意义下的加减法。

# 食物链

这道题同样可以使用“边带权”的方法来解决。
我们定义 $d_x = 0$ 表示 $x$ 和 $fa_x$ 是同类，$d_x = 1$ 表示 $x$ 吃 $fa_x$，$d_x = -1$ 表示 $fa_x$ 吃 $x$。由于食物链是一个大小为 3 的环，所以所有边权都可以在模 3 的意义下进行加减法运算。实际上，之前题目中，边权的异或操作就是模 2 意义下的加减法。
所以对于一个集合中的两个节点 $x$ 和 $y$，如果 $d_x - d_y \equiv 0 \pmod 3$ 则说明 $x$ 和 $y$ 是同类，如果 $d_x - d_y \equiv 1 \pmod 3$ 则说明 $x$ 吃 $y$，如果 $d_x - d_y \equiv 2 \pmod 3$ 则说明 $y$ 吃 $x$。