数据结构-基本操作

并查集

并查集 - OI Wiki (oi-wiki.org)

维护集合、无向图连通性（允许加边）

优化：

- 路径压缩（一般情况下必须加）
- 按秩合并（一般写启发式合并把 `size` 小的合并进大的中，支持撤销掉当前最后一次合并操作）

两个优化都加的时候单次操作时间复杂度为 $\mathcal{O}(\alpha(n))$，只加其中一个单次时间复杂度最差为 $\mathcal{O}(\log n)$

P4185 [USACO18JAN] MooTube G

离线，把边和询问都按 k 降序排序，用并查集维护连通块，按 k 从大到小处理询问，每次把本次新增的能用的边加进并查集，答案为该点所在的连通块大小 -1（去掉自己）

常见拓展

边带权并查集

在每个点上维护其到父亲节点的信息，结合路径压缩，对该点 `getroot` 后可以得到其到根的信息

```
int getroot(int x)
{
    if(x==fa[x])
        return x;
    int fx=getroot(fa[x]);
    f[x]+=f[fa[x]];
    return fa[x]=fx;
}
```

在合并两个集合时，往往已知的是 u 和 v 的关系，需要通过一定的计算，算出两个集合的根的关系，才能连边

```
void merge(int u,int v,int w) // u->v 边权为 w
{
    int fu=getroot(u),fv=getroot(v);
    // dis[u] + w(fu, fv) - dis[v] = w
    f[fu]=w-dis[u]+dis[v];
    fa[fu]=fv;
}
```

想要得到在同一集合中的两个点关系，需要先把两个点 `getroot` 一下，再用两个 `dis` 做运算

经典例题：食物链（两者的关系为它们之间的距离模 3）

扩展域并查集（种类并查集）

以食物链为例，将一个点 x 拆成三个点 $x, x+n, x+2n$，其中和 x 在同一集合的点表示跟 x 同类的点，和 $x+n$ 在同一集合的点表示 x 会吃的那一种类的点，和 $x+2n$ 在同一集合的点表示会被 x 吃的那一种类的点。

如果 u, v 是同一类的点，将 u, v 合并，$u+n, v+n$ 合并，$u+2n, v+2n$ 合并

如果 u 吃 v，将 $u+n, v$ 合并，$u+2n, v+n$ 合并，$u, v+2n$ 合并

如果 u 被 v 吃，将 $u+2n, v$ 合并，$u, v+n$ 合并，$u+n, v+2n$ 合并

如果 u 和 $v, v+n, v+2n$ 中的一个在同一集合，说明两者之间有关系，如果不符合这次给出的关系，就出现了矛盾，或者说不能出现在本次合并后 $v, v+n, v+2n$ 其中至少两个处于同一集合的情况

P9869 [NOIP2023] 三值逻辑

因为对于一个点，有些点会跟它相同，有些点会跟它相反，所以考虑使用扩展域并查集，将一个点拆成 i 和 $i+n$ 这两个点分别表示和 i 相等的元素和相反的元素所在的集合，可以再加三个点，分别表示 T, F, U

先求出每个数最终的值是 T, F, U 还是跟哪个数的初始值相等或相反，设这个数组为 a，初始让 $a[i]=i$，然后遇到 TFU 就直接赋值，对于 $+$ 就把 $a[x]$ 赋值成 $a[y]$，对于 $-$ 就把 $a[x]$ 赋值成和 $a[y]$ 相反的那个点

求出 a 数组后，将 i 和 $a[i]$ 合并，$i+n$ 和 $a[i]$ 的反点合并

如果最终 i 和 $i+n$ 在同一集合中，该集合就必须填成 U，打上标记，U 属于的集合也要打标记，最后再统计一下有多少点 i 属于打标记的集合即可

本题也可以使用边带权并查集，两点间的距离模 2 为 0 说明这两个点值相同，为 1 说明两个点值相反，合并时如果两点已经在同一集合里且距离不等于这条边的权值，就说明这个集合必须填 U

也可以使用二分图染色，两点之间边权为 0 表示同色，边权为 1 表示异色，如果染色失败，这个连通块就要全填 U

区间染色（跳过已经染过的）

$fa[i]$ 表示 $\geq i$ 处第一个没被染色的地方，染色后合并 $i, i+1$（注意方向），用 $findroot$ 再找下一个要染色的点

ST 表

ST 表 - OI Wiki (oi-wiki.org)

设 $st[i][j]$ 表示以 i 为起点，长度为 2^j 的区间的最值，初始化 $st[i][0]$，由 $st[i][j-1]$ 和 $st[i+(1<<(j-1))][j-1]$ 合并出 $st[i][j]$

建表的时间复杂度为 $\mathcal{O}(n \log n)$

查询 $[l, r]$ 的区间时只需要合并从 l 开始长度为 $2^{\log_2 r-l+1}$ 的区间和以 r 结尾长度为 $2^{\log_2 r-l+1}$ 的区间，重叠部分不影响最值，其中 $\log_2 x$ 的值可以预处理出来 $\log_2 x = \log_2 \lfloor \frac{n}{2} \rfloor + 1$

查询的时间复杂度为 $\mathcal{O}(1)$

堆

堆简介 - OI Wiki (oi-wiki.org)

删除元素时，将最后一个元素放到堆顶，从上往下调整

插入元素时，将其放到最后，从下往上调整

单次操作时间复杂度 $\mathcal{O}(\log n)$，一般使用 `priority_queue`，默认是大根堆，重载小于号内写的逻辑表达式和一般情况反过来

如果从下往上建堆（保证一个点的两个子树形成堆后，再把这个点从上往下调整），可以做到 $\mathcal{O}(n)$ 建堆

单调队列

单调队列 - OI Wiki (oi-wiki.org)

维护滑动窗口（双指针型区间）的区间最值

从队首弹出出界的元素，从队尾出掉不优的元素，把新元素放入队尾

总时间复杂度 $\mathcal{O}(n)$

单调栈

单调栈 - OI Wiki (oi-wiki.org)

不弹队首的单调队列，可以求出每个点的前驱后继（前边/后边第一个比它大/小的元素）

处理到 i 时的单调栈中的元素为 $a[1,\dots,i]$ 这段数组的后缀最值

树状数组

树状数组 - OI Wiki (oi-wiki.org)

$BIT[i]$ 存储到 $i-lowbit(i)+1$ 到 i 的信息，支持单点修改，前缀查询，时间复杂度 $\mathcal{O}(\log n)$

线段树

线段树 - OI Wiki (oi-wiki.org)

支持单点修改，单点查询，区间修改（需要 `lazy` 标记），区间查询

区间修改和查询时每层最多访问 4 个节点

做矩形操作相关的扫描线时，往往需要让叶子节点代表一小段的信息，而不是单独一个点的信息

单次操作的时间复杂度均为 $\mathcal{O}(\log n)$

U128816 重复询问II

可以将询问转化为 $[l,r]$ 中每个元素的前驱位置的最大值是否 $\geq l$，此时修改操作会影响 id，修改前 id 的后继，修改后 id 的后继这三个点的前驱信息，可以用 `set` 维护每种颜色出现在了哪些位置，这样可以快速找到修改前、后的后继点，也可以找到修改后这些受影响的点的新前驱

需要小心边界，可以把每个 `set` 初始时都放入 0 和 $n+1$ 两个哨兵，但这样时间空间常数都有点大

总时间复杂度 $\mathcal{O}((n+q)\log n)$

P4513 小白逛公园

单点修改，查询区间最大子段和

最大子段和是可以分治的问题，$[l,r]$ 的最大子段和是 $[l,mid]$ 中的最大子段和，$[mid+1,r]$ 中的最大子段和，和 $[l,mid]$ 中以 mid 为结尾的最大子段和 $lsum$ 加上 $[mid+1,r]$ 中以 $mid+1$ 为开头的最大子段和 $rsum$，三者中的最大值，因此我们可以用线段树维护这三个信息和区间和 sum，共四个信息。

$pushup$ 时，$lsum[p]$ 取 $lsum[p<<1]$ 和 $sum[p<<1]+lsum[p<<1|1]$ 中的较大值，$rsum[p]$ 取 $rsum[p<<1|1]$ 和 $sum[p<<1|1]+rsum[p<<1]$ 中的较大值，$query$ 时合并两个区间的查询结构时同理

总时间复杂度 $\mathcal{O}((n+m)\log n)$

P6619 [省选联考 2020 A/B 卷] 冰火战士

当指定了哪些人参加比赛之后，双方消耗的总能量一定是两方能量的较小值的二倍

因此选定了温度后，一定是让能参加比赛的人都参加最好，固定一个温度 x，温度 $\leq x$ 的冰系战士可以参赛，温度 $\geq x$ 的火系战士可以参赛

如果从小到大枚举温度，我们显然可以用两个桶维护温度为某个值的冰系/火系战士的能量值之和，从左往右推出这场比赛两方的能量，计算出总能量，取最优

但是询问次数和温度范围都比较大，我们必须快速找到合适的温度，找到合适的温度后，算双方的能量可以用值域线段树区间查询来做。

可以发现，随着温度的上升，冰系战士的总能量在增加，火系战士的总能量在减少，两者中的最小值最大的地方是冰系战士的总能量反超火系战士的总能量这一瞬间的前后，因此我们可以二分找出这一时刻，得到了一个复杂度不超过 $\mathcal{O}((n+q)\log^2 n)$ 的解法

如果不是先二分再区间查询，而是在线段树上二分，可以把时间复杂度降到一个 $\log$，因为火系战士那边需要的是温度 $\geq x$ 的战士的能量值之和，需要转化成总和减去 $< x$ 的和，此时需要把火系战士的能量在放上线段树时放到 $x+1$ 处，表示如果决定的温度 $\geq x+1$ 时，它不能上场，需要从总能量里减掉。

这样总时间复杂度降到了一个 $\log$，但线段树常数有点大，可能被卡常

由于查询的信息是前缀查询，可以把线段树上二分改为树状数组上倍增，即从 $now=0$ 开始，如果 $now|(1<<i)$ 在这个温度还未完成反超，就把 $now|=(1<<i)$，并把树状数组上新的 now 这个地方的值加进当前询问到的区间和里，因为此时 now 下标存储的信息是 $now-(1<<i)+1$ 到 now 的信息

```
for(int i=20;i>=0;i--)
    if((ans|(1<<i))<=cnt&&now0+BIT[0][ans|(1<<i)]<=all1-now1-BIT[1][ans|(1<<i)])
    {
        ans|=(1<<i);
        now0+=BIT[0][ans];
        now1+=BIT[1][ans];
    }
```

这样我们就可以找到反超前的时刻（也就是最后一个满足冰系战士的总能量 $\leq$ 火系战士的总能量的时刻），答案还可能在刚反超的时候，因为题目要的是最佳温度的最大值，所以需要找到反超后火系战士总能量不变的最后时刻，也就是满足火系战士的总能量 $\geq$ 某个值的最后时刻，也是可以从头开始在树状数组上倍增求的。

至此，本题就解决了。

P8868 [NOIP2022] 比赛

本题前 52 分是简单的，对每个询问单独处理，枚举 q 从 1 扫到 r，用单调栈辅助线段树区间赋值维护 p 取每个值时，$[p,q]$ 中 ma，mb，$ma\times mb$，线段树维护这些信息的区间和（分别称为 $suma,sumb,sumab$）

当 q 扫到 l 到 r 的时候，在线段树上做区间询问，询问 $[l,r]$ 的区间中 $ma\times mb$ 的和，加进当前答案

时间复杂度 $\mathcal{O}(q\times n\log n)$

可以发现这个询问的答案其实是 $[l,r]$ 这个区间在 q 从 l 扫到 r 时的历史和，所以正解就是考虑怎么把这个历史和维护出来。

当 $q=i$ 的修改做完后，需要给 $[1,i]$ 这个区间里的每个数加上当前的 $ma\times mb$，考虑怎么设计 $lazy$ 标记，可以认为当前节点的子树中还未修改的 ans 可以写成 $ans+=addab\times sumab+adda\times suma+addb\times sumb+add\times len$ 这样的形式，其中 $sumab,suma,sumb$ 都是原来（做当前赋值操作前的信息），因此 $lazy$ 里就需要有 $addab,adda,addb,add$ 这四个系数信息，因为还要做区间赋值，所以还要有 $seta,setb$ 这两个表示该区间是否做了赋值操作赋值成了多少这样的信息。

修改 ma 和 mb 相关信息时，相当于是给相应区间新增一个只有赋值操作 $lazytag$，区间里每个数都加上当前的 $ma\times mb$，则是相当于给相应区间新增一个只有 $addab=1$ 的 $lazytag$

修改和 `pushdown` 时对线段树信息的修改是一样的，都是让 $ans+=suma+addab\times sumab+adda\times suma+addb\times sumb+add\times len$，然后如果有赋值标记，就需要改 $sumab,suma,sumb$ 中的其中几个信息

```
void updatetree(int p,int l,int r,tag t)
{
    tree[p].ans+=t.addab*tree[p].sumab+t.adda*tree[p].suma+t.addb*tree[p].sumb+t.add*
(r-l+1);
    if(t.seta&&t.setb)
    {
        tree[p].sumab=t.seta*t.setb*(r-l+1);
        tree[p].suma=t.seta*(r-l+1);
        tree[p].sumb=t.setb*(r-l+1);
    }
    else if(t.seta)
```

```
    {
        tree[p].sumab=t.seta*tree[p].sumb;
        tree[p].suma=t.seta*(r-l+1);
    }
    else if(t.setb)
    {
        tree[p].sumab=t.setb*tree[p].suma;
        tree[p].sumb=t.setb*(r-l+1);
    }
}
```

再考虑修改和 `pushdown` 时对之前 $lazy$ 和本次新增的 $lazy$ 的合并操作，如果之前的 $lazy$ 里有 $seta, setb$ 都有值，那么这次新添的 $lazy$ 的操作用的 ma 和 mb 的信息都是 set 后的，因此只能把这个新添的 $lazy$ 操作的影响都算在常数部分需要加的值 add 里，即

$$lazy[p].add+=t.addab*lazy[p].seta*lazy[p].setb+t.adda*lazy[p].seta+t.addb*lazy[p].setb+t.add$$

其中 t 是本次新添的 $lazy$

如果之前的 $lazy$ 里只有 $seta$，就把新添的操作中的 $addab$ 和 $addb$ 的影响算到 $addb$ 里，把 $adda$ 和 add 的影响算到 add 里，只有 $setb$ 时也类似，都没有时，就是对应变量全部相加

别忘了，如果新来的操作中有赋值标志，要更新一下当前点 $lazy$ 里的赋值标记

```
void updatelazy(int p,int l,int r,tag t)
{
    if(lazy[p].seta&&lazy[p].setb)

lazy[p].add+=t.addab*lazy[p].seta*lazy[p].setb+t.adda*lazy[p].seta+t.addb*lazy[p].setb
+t.add;
    else if(lazy[p].seta)
    {
        lazy[p].addb+=t.addab*lazy[p].seta+t.addb;
        lazy[p].add+=t.adda*lazy[p].seta+t.add;
    }
    else if(lazy[p].setb)
    {
        lazy[p].adda+=t.addab*lazy[p].setb+t.adda;
        lazy[p].add+=t.addb*lazy[p].setb+t.add;
    }
    else
    {
        lazy[p].addab+=t.addab;
        lazy[p].adda+=t.adda;
        lazy[p].addb+=t.addb;
        lazy[p].add+=t.add;
    }
    if(t.seta)
        lazy[p].seta=t.seta;
    if(t.setb)
        lazy[p].setb=t.setb;
}
```

动态开点线段树

刚开始不建树，只有当需要访问这个节点时才把节点建出来，常数较大（相比于离散化后使用普通线段树），注意所需空间的计算，区间操作一层可能需要访问 4 个节点

P3960 [NOIP2017 提高组] 列队

看上去一次离队和归队会导致每一行都有变化，不好维护，但细心研究可以发现，如果不考虑最后一列，最多就只有一行的前 $m-1$ 个数会变，因此我们考虑维护每一行的前 $m-1$ 个数，再单独维护最后一列按行号从上往下的序列

操作分两类：

1. 如果 $y \neq m$

   操作会分为以下几步：

   - 在这一行的前 $m-1$ 个数中找到第 y 个数，取出并删去
   - 将取出的数放到最后一列的末尾
   - 在最后一列中找到第 x 个数，取出并删去
   - 将取出的数放到第 x 行前 $m-1$ 个数中的最后（放好后就是第 $m-1$ 个数）

2. 如果 $y = m$

   操作会分为以下几步：

   - 在最后一列中找到第 x 个数，取出并删去
   - 将取出的数放到最后一列的末尾

因此我们需要一个支持找到第 k 个数，删数，在末尾增加一个数的数据结构，平衡树可以做，但本题的插入只在末尾，因此可以使用线段树完成。

线段树的叶子上的信息就是人的编号，另外还需要在线段树上维护这个区间有多少个人，取出并删数的时候可以把相应叶子上的人的编号和人数都设为 0，在末尾插入一个人的时候，就是单点修改把叶子上记的编号放上去，人数设为 1，不过线段树根节点代表的区间分别需要是 $n+q$ 和 $m+q$

因为本题人数较多，初始不能建树，因此需要动态开点，当一个叶子节点被创建出来时，其编号是可以计算的，当一个非叶子节点被创建出来时，需要算一下这个区间有多少人（刚建出来的时候一定是在这个区间的所有人都还在）

总时间复杂度 $\mathcal{O}(q \log n)$

P7963 [NOIP2021] 棋局

前 24 分可以搜索，测试点 $7 \sim 8$ 可以直接判定周围 4 个点

删点维护连通块肯定不好做，先离线时光倒流变成加点，维护类型 3 的边形成的连通块，然后分移动到空白格和吃子分别处理

1、空白格

首先连通块内的空白格肯定全算进答案，然后考虑类型 2 的边，需要知道从当前点出发往四个方向沿类型 2 的边走能走到的极限位置，可以用 4 个并查集分别维护，这些空白格是可以走到的，但要跟类型 3 得到的去重，肯定不能暴力判每个点是不是重的，但如果我们给所有点按从上到下从左到右的顺序编号，类型 2 横向两个方向走到的空白格就是连续区间了，所以我们需要询问当前类型 3 的连通块里一个编号区间内有多少点，这些是重复的，同理按从左到右从上到下的顺序编号可以把纵向两个方向解决。因此类型 3 的连通块除了用并查集维护以外，还需要用线段树维护，合并连通块时就是线段树合并。

对于类型 1 涉及到的 4 个点，也是同样用线段树就能判断是否需要去重。

2、吃子

需要维护每个连通块边界有哪些点，分黑白分别维护，需要对一个等级区间询问有多少个棋子在该区间，因此需要关于等级的值域线段树，合并连通块时也是线段树合并。再把类型 2 的边考虑进来，发现每个方向最多吃一个子，和类型 1 一样，需要知道这个棋子是否已经被计算过。为了能直接在现有的关于等级的值域线段树上询问出这个结果，我们可以把棋子等级稍作调整使其两两不同，这样就能做这个单点询问了。为了之前的区间询问不出问题，就需要再记录对每个棋子而言实际等级相同的棋子调整后的等级的最大值，才能得到正确的询问区间。

至此全部问题都已解决，总共使用了 5 个并查集和 4 种线段树

可持久化线段树

可持久化线段树 - OI Wiki (oi-wiki.org)

每次修改时，将本次修改涉及到的点新建一份，原先的保留，新点的不涉及修改的儿子也直接连向原有的点

常见应用：不带修在线二维数点，每个版本是一个前缀的值域线段树，显然当前版本可以由上一个版本做单点修改得到，查询第 $[l,r]$ 个版本值域 $[x,y]$ 中的点数就是 $query$ 时用第 r 个版本的信息减第 $l-1$ 个版本的信息

如果可以离线，就直接把询问拆成两个，从头开始一路扫过去，扫到相应位置时计算相应答案，就不用可持久化来保留所有版本的信息了