数据结构及其应用

Vxlimo

福建师范大学附属中学

2023 年 7 月 10 日

写在前面

写在前面

- 请认真听课并尝试理解，如果该部分你已经会了或者掉线了不想听，请不要打扰到其他同学。

写在前面

- 请认真听课并尝试理解，如果该部分你已经会了或者掉线了不想听，请不要打扰到其他同学。
- 在讲课过程中如有任何问题请立即提出，但是建议提问之前和周围同学探讨一下，尽量不要出现无意义问题。

目录

简单介绍

简单介绍

- 这两个简单数据结构我想应该绝大部分同学都已经完全掌握了，所以不再赘述。

简单介绍

- 这两个简单数据结构我想应该绝大部分同学都已经完全掌握了，所以不再赘述。
- 本质来说，这两种数据结构就是在做一件事情：维护一个序列的某些信息，同时支持一定程度的修改。

简单介绍

- 这两个简单数据结构我想应该绝大部分同学都已经完全掌握了，所以不再赘述。
- 本质来说，这两种数据结构就是在做一件事情：维护一个序列的某些信息，同时支持一定程度的修改。
- 利用 $\log$ 来达成时间和空间上的平衡，复杂度均为 $\mathcal{O}(n\log n)$。

简单介绍

- 这两个简单数据结构我想应该绝大部分同学都已经完全掌握了，所以不再赘述。
- 本质来说，这两种数据结构就是在做一件事情：维护一个序列的某些信息，同时支持一定程度的修改。
- 利用 $\log$ 来达成时间和空间上的平衡，复杂度均为 $\mathcal{O}(n \log n)$。
- 显然，树状数组能做的，线段树也一定能做。但是树状数组更好写，并且常数也更小。

简单介绍

- 这两个简单数据结构我想应该绝大部分同学都已经完全掌握了，所以不再赘述。
- 本质来说，这两种数据结构就是在做一件事情：维护一个序列的某些信息，同时支持一定程度的修改。
- 利用 $\log$ 来达成时间和空间上的平衡，复杂度均为 $\mathcal{O}(n\log n)$。
- 显然，树状数组能做的，线段树也一定能做。但是树状数组更好写，并且常数也更小。
- 简单讲一下线段树的标记永久化操作，部分条件下有用。

标记永久化

标记永久化

- 线段树区间修改的时候，会用到标记和标记下传。

标记永久化

- 线段树区间修改的时候，会用到标记和标记下传。
- 在一些情况下，标记也可以不需要下传。

标记永久化

- 线段树区间修改的时候，会用到标记和标记下传。
- 在一些情况下，标记也可以不需要下传。
- 例如区间加，区间修改等，大多只需要多记录一下修改时间，查询的时候根据修改时间讨论一下就可以了。

标记永久化

- 线段树区间修改的时候，会用到标记和标记下传。
- 在一些情况下，标记也可以不需要下传。
- 例如区间加，区间修改等，大多只需要多记录一下修改时间，查询的时候根据修改时间讨论一下就可以了。
- 在一些标记下传会破坏复杂度，比如可持久化的时候就很重要。

简单介绍

简单介绍

- 并查集是一种，维护“祖先关系”的简单数据结构，同样不做过多说明。

简单介绍

- 并查集是一种，维护“祖先关系”的简单数据结构，同样不做过多说明。
- 这里我们主要对它的复杂度进行一个分析，同时介绍一个很强大的思想。

简单介绍

- 并查集是一种，维护“祖先关系”的简单数据结构，同样不做过多说明。
- 这里我们主要对它的复杂度进行一个分析，同时介绍一个很强大的思想。
- 启发式合并。

复杂度是 $\mathcal{O}(\log n)$?

复杂度是 $\mathcal{O}(\log n)$?

- 第一次接触并查集时，大多使用的是一个叫“路径压缩”的方法来保证它的复杂度。

复杂度是 $\mathcal{O}(\log n)$?

- 第一次接触并查集时，大多使用的是一个叫“路径压缩”的方法来保证它的复杂度。
- 那这为什么是对的?

复杂度是 $\mathcal{O}(\log n)$?

- 第一次接触并查集时，大多使用的是一个叫“路径压缩”的方法来保证它的复杂度。
- 那这为什么是对的?
- 实际上，这个做法在少数情况下是会被卡的。有经验的出题人甚至会特意针对这种做法。

复杂度是 $\mathcal{O}(\log n)$?

- 第一次接触并查集时，大多使用的是一个叫“路径压缩”的方法来保证它的复杂度。
- 那这为什么是对的?
- 实际上，这个做法在少数情况下是会被卡的。有经验的出题人甚至会特意针对这种做法。
- 这里我们不打算讲，有兴趣的同学可以自行去 OI Wiki 或者其他网站上查找。

复杂度是 $\mathcal{O}(\log n)$?

- 第一次接触并查集时，大多使用的是一个叫“路径压缩”的方法来保证它的复杂度。
- 那这为什么是对的?
- 实际上，这个做法在少数情况下是会被卡的。有经验的出题人甚至会特意针对这种做法。
- 这里我们不打算讲，有兴趣的同学可以自行去 OI Wiki 或者其他网站上查找。
- 所以正确做法是什么?

启发式合并

启发式合并

- 在这之前，先来介绍一下启发式合并到底怎么启发了。就以并查集为例子。

启发式合并

- 在这之前，先来介绍一下启发式合并到底怎么启发了。就以并查集为例子。
- 每次，比较两个需要合并的集合的大小，然后把小的那个合并到大的那个上面。

启发式合并

- 在这之前，先来介绍一下启发式合并到底怎么启发了。就以并查集为例子。
- 每次，比较两个需要合并的集合的大小，然后把小的那个合并到大的那个上面。
- 就这么简单。简单到我们需要严谨证明一下。

启发式合并

- 在这之前，先来介绍一下启发式合并到底怎么启发了。就以并查集为例子。
- 每次，比较两个需要合并的集合的大小，然后把小的那个合并到大的那个上面。
- 就这么简单。简单到我们需要严谨证明一下。
- 对于合并部分，考虑每个元素，由于每次它所在的集合合并都会导致集合大小至少 $\times 2$，因此最多合并次数是 $\log n$ 次。

启发式合并

- 在这之前，先来介绍一下启发式合并到底怎么启发了。就以并查集为例子。
- 每次，比较两个需要合并的集合的大小，然后把小的那个合并到大的那个上面。
- 就这么简单。简单到我们需要严谨证明一下。
- 对于合并部分，考虑每个元素，由于每次它所在的集合合并都会导致集合大小至少 $\times 2$，因此最多合并次数是 $\log n$ 次。
- 对 n 个元素，那就是 $\mathcal{O}(n \log n)$ 的了。

启发式合并

启发式合并

- 对于查询？可能不是那么直观。

启发式合并

- 对于查询？可能不是那么直观。
- 这里有一个结论，启发式合并最多导致整棵树的深度 $+1$。

启发式合并

- 对于查询？可能不是那么直观。
- 这里有一个结论，启发式合并最多导致整棵树的深度 $+1$。
- 也就是说，启发式合并中，小的子树的深度一定不大于大的子树。

启发式合并

- 对于查询？可能不是那么直观。
- 这里有一个结论，启发式合并最多导致整棵树的深度 $+1$。
- 也就是说，启发式合并中，小的子树的深度一定不大于大的子树。
- 证明用数学归纳法分奇偶讨论，

$dep_{2k} = dep_{2k+1}, dep_{2k+2} = dep_{2k} + 1 / dep_{2k+1}$。

启发式合并

- 对于查询？可能不是那么直观。
- 这里有一个结论，启发式合并最多导致整棵树的深度 $+1$。
- 也就是说，启发式合并中，小的子树的深度一定不大于大的子树。
- 证明用数学归纳法分奇偶讨论，
  $dep_{2k} = dep_{2k+1}, dep_{2k+2} = dep_{2k} + 1 / dep_{2k+1}$。
- 于是，总深度一定是 $\log n$ 级别的，也就证明完毕了。

左偏树

左偏树

- 普通的二叉堆我们就不介绍了。这里介绍一种简单的可并堆：左偏树。

左偏树

- 普通的二叉堆我们就不介绍了。这里介绍一种简单的可并堆：左偏树。
- 可并堆，顾名思义，可以在较低的时间复杂度下完成对两个堆的合并。左偏树是其中较为简单的一种。

左偏树

- 普通的二叉堆我们就不介绍了。这里介绍一种简单的可并堆：左偏树。
- 可并堆，顾名思义，可以在较低的时间复杂度下完成对两个堆的合并。左偏树是其中较为简单的一种。
- 何为左偏？我们定义外节点为左儿子或右儿子不存在的点，$dist$ 为一个节点到离他最近的外节点的距离 $+1$，空节点的 $dist = 0$。

左偏树

- 普通的二叉堆我们就不介绍了。这里介绍一种简单的可并堆：左偏树。
- 可并堆，顾名思义，可以在较低的时间复杂度下完成对两个堆的合并。左偏树是其中较为简单的一种。
- 何为左偏？我们定义外节点为左儿子或右儿子不存在的点，$dist$ 为一个节点到离他最近的外节点的距离 $+1$，空节点的 $dist = 0$。
- 其中 $+1$ 只是为了写代码方便，没有其他含义。

左偏树

- 普通的二叉堆我们就不介绍了。这里介绍一种简单的可并堆：左偏树。
- 可并堆，顾名思义，可以在较低的时间复杂度下完成对两个堆的合并。左偏树是其中较为简单的一种。
- 何为左偏？我们定义外节点为左儿子或右儿子不存在的点，$dist$ 为一个节点到离他最近的外节点的距离 $+1$，空节点的 $dist = 0$。
- 其中 $+1$ 只是为了写代码方便，没有其他含义。
- 左偏树满足 $dist_{lson} \geq dist_{rson}$，稍微画一下图就可以发现，它的确是“左偏”的。

几个性质

几个性质

- 首先，$dist_u = dist_{rson} + 1$，这一点应该很显然。

几个性质

- 首先，$dist_u = dist_{rson} + 1$，这一点应该很显然。
- 其次，一棵 n 个点左偏树，$dist$ 的最大值应该在 $\log n$ 级别。

几个性质

- 首先，$dist_u = dist_{rson} + 1$，这一点应该很显然。
- 其次，一棵 n 个点左偏树，$dist$ 的最大值应该在 $\log n$ 级别。
- 假设根的 $dist = x$，那么这棵树至少有 $x - 1$ 层是满二叉树，这样就至少有 $2^x - 1$ 个节点了。

几个性质

- 首先，$dist_u = dist_{rson} + 1$，这一点应该很显然。
- 其次，一棵 n 个点左偏树，$dist$ 的最大值应该在 $\log n$ 级别。
- 假设根的 $dist = x$，那么这棵树至少有 $x - 1$ 层是满二叉树，这样就至少有 $2^x - 1$ 个节点了。
- 需要注意的是，左偏树的深度是没有保证的。一条向左的链同样是正确的左偏树，这一点请牢记于心。

$merge$ 操作

$merge$ 操作

- $merge$ 操作是左偏树的核心，剩下所有操作都是基于 $merge$。

$merge$ 操作

- $merge$ 操作是左偏树的核心，剩下所有操作都是基于 $merge$。
- 看一下代码。

其他操作

其他操作

- 插入元素：一个点也算作一个堆，直接合并即可。

其他操作

- 插入元素：一个点也算作一个堆，直接合并即可。
- 删除堆顶：合并根的左右儿子。

其他操作

- 插入元素：一个点也算作一个堆，直接合并即可。
- 删除堆顶：合并根的左右儿子。
- 不改变相对大小的全局操作（加乘）：在根上打标记，$merge$ 时 pushdown。

其他操作

其他操作

- 删除任意节点：儿子合并后向上更新 $dist$，并交换左右儿子维护左偏性质，直到 $dist$ 不改变为止。

其他操作

- 删除任意节点：儿子合并后向上更新 $dist$，并交换左右儿子维护左偏性质，直到 $dist$ 不改变为止。
- 令删除的节点为 u，若 u 为右儿子，$dist_{fa_u} = dist_u + 1$。

其他操作

- 删除任意节点：儿子合并后向上更新 $dist$，并交换左右儿子维护左偏性质，直到 $dist$ 不改变为止。
- 令删除的节点为 u，若 u 为右儿子，$dist_{fa_u} = dist_u + 1$。
- 若 u 为左儿子，当且仅当它和右儿子的 $dist$ 一样的时候，fa_u 的 $dist$ 才会更新。

其他操作

- 删除任意节点：儿子合并后向上更新 $dist$，并交换左右儿子维护左偏性质，直到 $dist$ 不改变为止。
- 令删除的节点为 u，若 u 为右儿子，$dist_{fa_u} = dist_u + 1$。
- 若 u 为左儿子，当且仅当它和右儿子的 $dist$ 一样的时候，fa_u 的 $dist$ 才会更新。
- 因此每层 $dist$ 均 $+1$。结合 $dist$ 的性质可知，操作次数在 $\log n$ 级别。

[SCOI2011] 棘手的操作

[SCOI2011] 棘手的操作

有 n 个节点，一开始相互不连通。

第 i 个节点的初始权值为 a_i，接下来有如下一些操作：

- U x y：加一条边，连接第 x 个节点和第 y 个节点。
- A1 x v：将第 x 个节点的权值增加 v。
- A2 x v：将第 x 个节点所在的连通块的所有节点的权值都增加 v。
- A3 v：将所有节点的权值都增加 v。
- F1 x：输出第 x 个节点当前的权值。
- F2 x：输出第 x 个节点所在的连通块中，权值最大的节点的权值。
- F3：输出所有节点中，权值最大的节点的权值。

$n, q \leq 3 \times 10^5$。

[SCOI2011] 棘手的操作

[SCOI2011] 棘手的操作

- 显然使用可并堆维护联通块。

[SCOI2011] 棘手的操作

- 显然使用可并堆维护联通块。
- A1 和 A2 操作刚刚都讲过了，A3 只要全局标记就行。

[SCOI2011] 棘手的操作

- 显然使用可并堆维护联通块。
- A1 和 A2 操作刚刚都讲过了，A3 只要全局标记就行。
- F2 操作寻找堆顶时，常见错解是暴力跳父亲。刚刚说过，左偏树的树高是没有保证的。

[SCOI2011] 棘手的操作

- 显然使用可并堆维护联通块。
- A1 和 A2 操作刚刚都讲过了，A3 只要全局标记就行。
- F2 操作寻找堆顶时，常见错解是暴力跳父亲。刚刚说过，左偏树的树高是没有保证的。
- 因此要用并查集维护联通性的同时，记一下根。

重链剖分

重链剖分

- 对于一棵树，先给出若干定义如下：

重链剖分

- 对于一棵树，先给出若干定义如下：
- 重儿子：一个点的子节点中，子树大小最大的子节点。

重链剖分

- 对于一棵树，先给出若干定义如下：
- 重儿子：一个点的子节点中，子树大小最大的子节点。
- 轻儿子：不是重儿子的其他子节点。

重链剖分

- 对于一棵树，先给出若干定义如下：
- 重儿子：一个点的子节点中，子树大小最大的子节点。
- 轻儿子：不是重儿子的其他子节点。
- 重边：一个点与其重儿子的连边。重链：首尾衔接的重边。

重链剖分

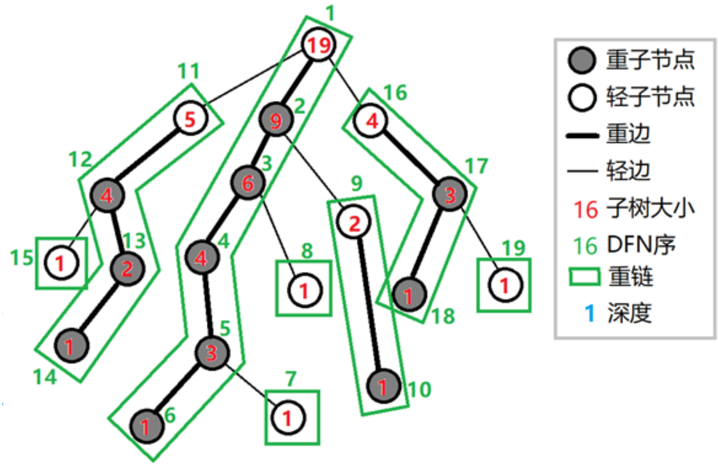
- 对于一棵树，先给出若干定义如下：
- 重儿子：一个点的子节点中，子树大小最大的子节点。
- 轻儿子：不是重儿子的其他子节点。
- 重边：一个点与其重儿子的连边。重链：首尾衔接的重边。
- 轻边：剩下的边。

重链剖分

- 对于一棵树，先给出若干定义如下：
- 重儿子：一个点的子节点中，子树大小最大的子节点。
- 轻儿子：不是重儿子的其他子节点。
- 重边：一个点与其重儿子的连边。重链：首尾衔接的重边。
- 轻边：剩下的边。
- 把单个点也看成重链，那么整棵树就被剖分成若干条重链。

重链剖分

重链剖分



重要性质

重要性质

- 任意一条父子链的轻边条数在 $\log n$ 级别。

重要性质

- 任意一条父子链的轻边条数在 $\log n$ 级别。
- 证明考虑每走过一条轻边，子树大小必定至少翻倍。

重要性质

- 任意一条父子链的轻边条数在 $\log n$ 级别。
- 证明考虑每走过一条轻边，子树大小必定至少翻倍。
- 有什么用？回到上一张图看看。

重要性质

- 任意一条父子链的轻边条数在 $\log n$ 级别。
- 证明考虑每走过一条轻边，子树大小必定至少翻倍。
- 有什么用？回到上一张图看看。
- 我们注意到，如果让每条重链的 dfn 序连续，那么每条父子链都可以按照 dfn 序被拆分成连续的 $\log n$ 段区间。

重要性质

- 任意一条父子链的轻边条数在 $\log n$ 级别。
- 证明考虑每走过一条轻边，子树大小必定至少翻倍。
- 有什么用？回到上一张图看看。
- 我们注意到，如果让每条重链的 dfn 序连续，那么每条父子链都可以按照 dfn 序被拆分成连续的 $\log n$ 段区间。
- 推广一下，树上任意一条链都可以按照 dfn 序被拆分成连续的 $\log n$ 段区间。

重要性质

- 任意一条父子链的轻边条数在 $\log n$ 级别。
- 证明考虑每走过一条轻边，子树大小必定至少翻倍。
- 有什么用？回到上一张图看看。
- 我们注意到，如果让每条重链的 dfn 序连续，那么每条父子链都可以按照 dfn 序被拆分成连续的 $\log n$ 段区间。
- 推广一下，树上任意一条链都可以按照 dfn 序被拆分成连续的 $\log n$ 段区间。
- 如果我们结合上区间修改相关的数据结构？线段树？

应用

应用

- 于是，借助树剖和线段树，现在我们可以实现树上任意一条链的一些操作。

应用

- 于是，借助树剖和线段树，现在我们可以实现树上任意一条链的一些操作。
- 顺带一提，由于 dfn 序本身的特殊性，子树的 dfn 序必然是一个连续区间，因此子树操作也可以实现。

应用

- 于是，借助树剖和线段树，现在我们可以实现树上任意一条链的一些操作。
- 顺带一提，由于 dfn 序本身的特殊性，子树的 dfn 序必然是一个连续区间，因此子树操作也可以实现。
- $\log n$ 个重链区间，加上线段树自带的 $\log n$，单次操作的复杂度是 $\mathcal{O}(\log^2 n)$。

应用

- 于是，借助树剖和线段树，现在我们可以实现树上任意一条链的一些操作。
- 顺带一提，由于 dfn 序本身的特殊性，子树的 dfn 序必然是一个连续区间，因此子树操作也可以实现。
- $\log n$ 个重链区间，加上线段树自带的 $\log n$，单次操作的复杂度是 $\mathcal{O}(\log^2 n)$。
- 并且，树剖的常数很小，很多时候吊打理论复杂度 $\mathcal{O}(\log n)$ 的 LCT。

应用

- 于是，借助树剖和线段树，现在我们可以实现树上任意一条链的一些操作。
- 顺带一提，由于 dfn 序本身的特殊性，子树的 dfn 序必然是一个连续区间，因此子树操作也可以实现。
- $\log n$ 个重链区间，加上线段树自带的 $\log n$，单次操作的复杂度是 $\mathcal{O}(\log^2 n)$。
- 并且，树剖的常数很小，很多时候吊打理论复杂度 $\mathcal{O}(\log n)$ 的 LCT。
- 看看板子题。

[ZJOI2008] 树的统计

[ZJOI2008] 树的统计

一棵树上有 n 个节点，每个节点都有一个权值 w。

有如下 q 次操作：

1. 把结点 u 的权值改为 t。
2. 询问从点 u 到点 v 的路径上的节点的最大权值。
3. 询问从点 u 到点 v 的路径上的节点的权值和。

从点 u 到点 v 的路径上的节点包括 u 和 v 本身。

$n \le 3 \times 10^4, q \le 2 \times 10^5$。

[ZJOI2008] 树的统计

[ZJOI2008] 树的统计

- 板子题，如果刚刚听懂了应该很简单。

[ZJOI2008] 树的统计

- 板子题，如果刚刚听懂了应该很简单。
- 树剖，然后转为单点修改，区间查询最大值和求和，线段树轻松解决。

何为可持久化?

何为可持久化?

- 有的时候，题目会要求我们将操作回退，也即进行 undo/redo 操作。

何为可持久化?

- 有的时候，题目会要求我们将操作回退，也即进行 undo/redo 操作。
- 使数据结构能够执行这样的操作的改造，就叫可持久化。

何为可持久化?

- 有的时候，题目会要求我们将操作回退，也即进行 undo/redo 操作。
- 使数据结构能够执行这样的操作的改造，就叫可持久化。
- 总的来说，可持久化只有一个中心思想：存档只存改过的。以下以线段树为例。

可持久化线段树

可持久化线段树

- 我们知道，无论是区间修改还是单点修改，线段树上变化的点数都只有 $\log n$ 级别个。

可持久化线段树

- 我们知道，无论是区间修改还是单点修改，线段树上变化的点数都只有 $\log n$ 级别个。
- 因此一个新的线段树，其实只有 $\log n$ 个点和原来不同。我们只要对所有修改过的点新建一个版本，而不需要复制整棵树。

可持久化线段树

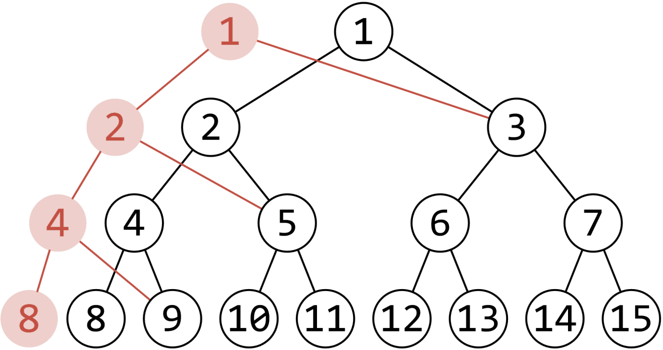
- 我们知道，无论是区间修改还是单点修改，线段树上变化的点数都只有 $\log n$ 级别个。
- 因此一个新的线段树，其实只有 $\log n$ 个点和原来不同。我们只要对所有修改过的点新建一个版本，而不需要复制整棵树。
- 需要注意的是，显然父子之间的编号关系被破坏了，因而可持久化时需要动态开点。

可持久化线段树

- 我们知道，无论是区间修改还是单点修改，线段树上变化的点数都只有 $\log n$ 级别个。
- 因此一个新的线段树，其实只有 $\log n$ 个点和原来不同。我们只要对所有修改过的点新建一个版本，而不需要复制整棵树。
- 需要注意的是，显然父子之间的编号关系被破坏了，因而可持久化时需要动态开点。
- 下面是一张经典的图。

可持久化线段树

可持久化线段树



可持久化数据结构

可持久化数据结构

- 事实上，依据这个思想，任意一个数据结构，只要新建一个版本的复杂度不高于原本修改的复杂度，都可以可持久化。

可持久化数据结构

- 事实上，依据这个思想，任意一个数据结构，只要新建一个版本的复杂度不高于原本修改的复杂度，都可以可持久化。
- 这也是左偏树，无旋 Treap 等的存在意义之一。和二叉堆不同，修改不会改变树的结构，也因而可持久化。

在做题以前

在做题以前

- 今天讲的是数据结构，但是裸数据结构题几乎不会出现。

在做题以前

- 今天讲的是数据结构，但是裸数据结构题几乎不会出现。
- 要理解一个思想：数据结构只是一个工具，题目的核心环节应当是对条件的转换。

在做题以前

- 今天讲的是数据结构，但是裸数据结构题几乎不会出现。
- 要理解一个思想：数据结构只是一个工具，题目的核心环节应当是对条件的转换。
- 数据结构承担的仅仅是“维护某个需要维护的东西”的作用。

[NOIP2017 提高组] 列队

[NOIP2017 提高组] 列队

n 行 m 列个学生，从左往右，从上往下地依次编号为 $1 \sim n \times m$。

q 次事件，每次有 1 个坐标为 (i, j) 的学生离队。

离队后，所有学生依次执行以下两个操作：

- 第一列保持不动，所有学生向左填补空位。

- 第一行保持不动，所有学生向前填补空位。

显然这样空位来到 (n, m) 的位置，此时离队的学生回到此空位。

注意：编号不会随着学生位置改变。

求每次离队的学生的编号。

$n, m, q \leq 3 \times 10^5$。

[NOIP2017 提高组] 列队

[NOIP2017 提高组] 列队

- 经典的线段树/树状数组高级运用。当然用平衡树直接暴力搞也是可以的。

[NOIP2017 提高组] 列队

- 经典的线段树/树状数组高级运用。当然用平衡树直接暴力搞也是可以的。
- 首先先手画一下可能影响到的点，发现有变动的只有修改点所在行和最后一列，并且大部分的点的相对位置都没有变。

[NOIP2017 提高组] 列队

- 经典的线段树/树状数组高级运用。当然用平衡树直接暴力搞也是可以的。
- 首先先手画一下可能影响到的点，发现有变动的只有修改点所在行和最后一列，并且大部分的点的相对位置都没有变。
- 这提示我们把最后一列抽出来单独考虑，这样就是平衡树裸题了。但是如果我们不用平衡树，可以使用惰性删除。

[NOIP2017 提高组] 列队

- 经典的线段树/树状数组高级运用。当然用平衡树直接暴力搞也是可以的。
- 首先先手画一下可能影响到的点，发现有变动的只有修改点所在行和最后一列，并且大部分的点的相对位置都没有变。
- 这提示我们把最后一列抽出来单独考虑，这样就是平衡树裸题了。但是如果我们不用平衡树，可以使用惰性删除。
- 什么叫惰性删除？就是删除时只在位置上打上一个删除标记，查询时跳过这个数就好。这样可以很好地维护点的相对位置。

[NOIP2017 提高组] 列队

[NOIP2017 提高组] 列队

- 于是，我们目前的做法是，维护 $n+1$ 个序列表示每行的前 $m-1$ 个数和最后一列。

[NOIP2017 提高组] 列队

- 于是，我们目前的做法是，维护 $n+1$ 个序列表示每行的前 $m-1$ 个数和最后一列。
- 每次操作，把修改点和所在行的最后一个点打上删除标记，并在相应的序列的末端重新加入这两个数。

[NOIP2017 提高组] 列队

- 于是，我们目前的做法是，维护 $n+1$ 个序列表示每行的前 $m-1$ 个数和最后一列。
- 每次操作，把修改点和所在行的最后一个点打上删除标记，并在相应的序列的末端重新加入这两个数。
- 现在问题是，我们如何才能越过删除标记，快速定位一个序列中某个位置的数?

[NOIP2017 提高组] 列队

- 于是，我们目前的做法是，维护 $n+1$ 个序列表示每行的前 $m-1$ 个数和最后一列。
- 每次操作，把修改点和所在行的最后一个点打上删除标记，并在相应的序列的末端重新加入这两个数。
- 现在问题是，我们如何才能越过删除标记，快速定位一个序列中某个位置的数？
- 注意到了吗？线段树还没出场！

[NOIP2017 提高组] 列队

[NOIP2017 提高组] 列队

- 用 $n+1$ 棵动态开点线段树维护这 $n+1$ 个序列的删除标记，找第 k 个数变成在线段树上二分！

[NOIP2017 提高组] 列队

- 用 $n+1$ 棵动态开点线段树维护这 $n+1$ 个序列的删除标记，找第 k 个数变成在线段树上二分！
- 问题到此解决，时间复杂度 $\mathcal{O}(q\log n)$。现在的问题是，空间够不够？

[NOIP2017 提高组] 列队

- 用 $n+1$ 棵动态开点线段树维护这 $n+1$ 个序列的删除标记，找第 k 个数变成在线段树上二分！
- 问题到此解决，时间复杂度 $\mathcal{O}(q\log n)$。现在的问题是，空间够不够？
- 删除标记最多添加 $2\times q$ 个，而单次修改最多开点是 $\log(n+q)$ 量级。完全足够。

[十二省联考 2019] 春节十二响

[十二省联考 2019] 春节十二响

给一棵 n 个点的树，每个点有一个权值 w_i。

你每次可以选取树上的一个点集，要求点集中的每个点不能是另一个点的祖先，而选出点集的代价为点集中权值最大点的权值。

问将所有点都选一遍的最小代价为多少，每次选的点集不能包含之前已经被选过的点。

$n \leq 2 \times 10^5$。

提示：若树是一条链，怎么做。

[十二省联考 2019] 春节十二响

[十二省联考 2019] 春节十二响

- 都提示了，那先考虑链的情况。容易发现只有两种可能：根有 $1/2$ 个儿子。

[十二省联考 2019] 春节十二响

- 都提示了，那先考虑链的情况。容易发现只有两种可能：根有 $1/2$ 个儿子。
- 如果是 1 个，显然只能一个一个选，答案就是 $\sum w_i$。

[十二省联考 2019] 春节十二响

- 都提示了，那先考虑链的情况。容易发现只有两种可能：根有 $1/2$ 个儿子。
- 如果是 1 个，显然只能一个一个选，答案就是 $\sum w_i$。
- 如果是 2 个，那就是两个子树里各选 1 个，w_i 较大的贡献到答案。

[十二省联考 2019] 春节十二响

- 都提示了，那先考虑链的情况。容易发现只有两种可能：根有 $1/2$ 个儿子。
- 如果是 1 个，显然只能一个一个选，答案就是 $\sum w_i$。
- 如果是 2 个，那就是两个子树里各选 1 个，w_i 较大的贡献到答案。
- 怎么配对？肯定是贪心地从大到小分别配对。

[十二省联考 2019] 春节十二响

[十二省联考 2019] 春节十二响

- 尝试推广到一般情况？先考虑多个子树。

[十二省联考 2019] 春节十二响

- 尝试推广到一般情况？先考虑多个子树。
- 对每个点维护一个集合，表示这个子树的最优解。

[十二省联考 2019] 春节十二响

- 尝试推广到一般情况？先考虑多个子树。
- 对每个点维护一个集合，表示这个子树的最优解。
- 那么，两个子树的集合配对合并后，新的集合由于内部都不能同时选，就相当于一个新的子树。

[十二省联考 2019] 春节十二响

- 尝试推广到一般情况？先考虑多个子树。
- 对每个点维护一个集合，表示这个子树的最优解。
- 那么，两个子树的集合配对合并后，新的集合由于内部都不能同时选，就相当于一个新的子树。
- 不断合并这个新的子树和另一个子树就好了，最后加上这个点本身，就是这个点的最优解集合。

[十二省联考 2019] 春节十二响

- 尝试推广到一般情况？先考虑多个子树。
- 对每个点维护一个集合，表示这个子树的最优解。
- 那么，两个子树的集合配对合并后，新的集合由于内部都不能同时选，就相当于一个新的子树。
- 不断合并这个新的子树和另一个子树就好了，最后加上这个点本身，就是这个点的最优解集合。
- 现在问题是，对于两个集合怎么配对删点？

[十二省联考 2019] 春节十二响

[十二省联考 2019] 春节十二响

- 直接暴力怎么做?

[十二省联考 2019] 春节十二响

- 直接暴力怎么做?
- 用 STL::multiset 维护集合和排序，每次合并直接遍历集合。

[十二省联考 2019] 春节十二响

- 直接暴力怎么做?
- 用 STL::multiset 维护集合和排序，每次合并直接遍历集合。
- 这样的问题是，集合的大小可能很大，如果是一条链复杂度就会达到 $\mathcal{O}(n^2 \log n)$ 级别。如何优化?

[十二省联考 2019] 春节十二响

- 直接暴力怎么做?
- 用 STL::multiset 维护集合和排序，每次合并直接遍历集合。
- 这样的问题是，集合的大小可能很大，如果是一条链复杂度就会达到 $\mathcal{O}(n^2 \log n)$ 级别。如何优化?
- 我们这里还需要两两比较大小，所以单纯可并堆（左偏树）肯定是不行的。

[十二省联考 2019] 春节十二响

[十二省联考 2019] 春节十二响

- 考虑启发式合并的思想。由于第一个子树是不需要合并的，因此对每个点，直接继承它最大的儿子的集合，往下合并。

[十二省联考 2019] 春节十二响

- 考虑启发式合并的思想。由于第一个子树是不需要合并的，因此对每个点，直接继承它最大的儿子的集合，往下合并。
- 那么对于每个点，最大操作次数是子树大小 $/2$ 级别的。

[十二省联考 2019] 春节十二响

- 考虑启发式合并的思想。由于第一个子树是不需要合并的，因此对每个点，直接继承它最大的儿子的集合，往下合并。
- 那么对于每个点，最大操作次数是子树大小 $/2$ 级别的。
- 算一下这个大概是多少。对于第一层最多是 $n/2$，第二层最多是 $n/4$，最终我们发现，总次数的量级是 $n \log n$ 级别！

[十二省联考 2019] 春节十二响

- 考虑启发式合并的思想。由于第一个子树是不需要合并的，因此对每个点，直接继承它最大的儿子的集合，往下合并。
- 那么对于每个点，最大操作次数是子树大小 $/2$ 级别的。
- 算一下这个大概是多少。对于第一层最多是 $n/2$，第二层最多是 $n/4$，最终我们发现，总次数的量级是 $n\log n$ 级别！
- 于是这个题就搞定了，总时间复杂度 $\mathcal{O}(n\log^2 n)$。

[NOI2021] 轻重边

[NOI2021] 轻重边

给一棵 n 个点的树，树上的每一条边可能是轻边或者重边。

接下来对树进行 m 次操作，在操作开始前，树上所有边都是轻边。操作有以下两种：

1. 给定两个点 u 和 v，对于 u 到 v 路径上的所有点 x（包含 u 和 v），将与 x 相连的所有边变为轻边。然后再将 u 到 v 路径上的所有边变为重边。

2. 给定两个点 u 和 v，查询当前 u 到 v 的路径有多少条重边。

$n, m \leq 10^5$。

提示：当两端的点满足什么条件时，一条边是重边？

[NOI2021] 轻重边

[NOI2021] 轻重边

- 当且仅当一条边两端的点最后一次被操作是同一次，这条边是重边。

[NOI2021] 轻重边

- 当且仅当一条边两端的点最后一次被操作是同一次，这条边是重边。
- 那么，只要对每个点记一下最后一次操作的时间（染色），问题转化成求路径上颜色段的长度 -1 的和。

[NOI2021] 轻重边

- 当且仅当一条边两端的点最后一次被操作是同一次，这条边是重边。
- 那么，只要对每个点记一下最后一次操作的时间（染色），问题转化成求路径上颜色段的长度 -1 的和。
- 先考虑序列上怎么做，线段树上每个点记一下颜色段长度 -1 的和，左端点颜色，右端点颜色。

[NOI2021] 轻重边

- 当且仅当一条边两端的点最后一次被操作是同一次，这条边是重边。
- 那么，只要对每个点记一下最后一次操作的时间（染色），问题转化成求路径上颜色段的长度 -1 的和。
- 先考虑序列上怎么做，线段树上每个点记一下颜色段长度 -1 的和，左端点颜色，右端点颜色。
- 合并两段区间的时候就判断一下中间能不能接起来，更新答案就好。修改就是区间赋值。

[NOI2021] 轻重边

[NOI2021] 轻重边

- 转移到树上？树剖！重链之间的合并和线段树上是一个道理。

[NOI2021] 轻重边

- 转移到树上？树剖！重链之间的合并和线段树上是一个道理。
- 每次操作 $\log n$ 个区间，时间复杂度 $\mathcal{O}(m\log^2 n)$。

[NOI2021] 轻重边

- 转移到树上？树剖！重链之间的合并和线段树上是一个道理。
- 每次操作 $\log n$ 个区间，时间复杂度 $\mathcal{O}(m\log^2 n)$。
- 本题还有 LCT，毛毛虫剖分等做法，有兴趣的同学自行查找。

[JLOI2015] 城池攻占

[JLOI2015] 城池攻占

有 n 个城池构成一棵树，每个城池有一个防御力 h_i。

现在有 m 个骑士，每个骑士有一个初始攻击力 s_i，每个骑士都会从一个城池出发，一路向上攻打直到到根所在的城池。

显然地，如果一个骑士的攻击力低于城池的防御力，他就会在这个城池阵亡。

否则，他会攻克这个城池，并且攻击力会加上/乘上 v_i。

m 个骑士相互独立，也即一个骑士的攻打结果对其他骑士没有影响。

求对每个骑士，他攻克的城池数；以及对每个城池，有多少个骑士阵亡在这里。

$n, m \leq 3 \times 10^5$。

[JLOI2015] 城池攻占

[JLOI2015] 城池攻占

- 当然不能一个个骑士算过去，所以我们要所有骑士一起算。

[JLOI2015] 城池攻占

- 当然不能一个个骑士算过去，所以我们要所有骑士一起算。
- 从下往上对每个城池维护攻打它的骑士的集合，在每个城池去掉阵亡的，再打上攻击力的标记。用什么数据结构？

[JLOI2015] 城池攻占

- 当然不能一个个骑士算过去，所以我们要所有骑士一起算。
- 从下往上对每个城池维护攻打它的骑士的集合，在每个城池去掉阵亡的，再打上攻击力的标记。用什么数据结构？
- 还是集合合并，考虑可并堆或者启发式合并。

[JLOI2015] 城池攻占

- 当然不能一个个骑士算过去，所以我们要所有骑士一起算。
- 从下往上对每个城池维护攻打它的骑士的集合，在每个城池去掉阵亡的，再打上攻击力的标记。用什么数据结构？
- 还是集合合并，考虑可并堆或者启发式合并。
- 如果用可并堆，在根上打标记，删除的时候 pushdown 一下，时间复杂度 $\mathcal{O}((n+m)\log m)$，做完了。

[JLOI2015] 城池攻占

- 当然不能一个个骑士算过去，所以我们要所有骑士一起算。
- 从下往上对每个城池维护攻打它的骑士的集合，在每个城池去掉阵亡的，再打上攻击力的标记。用什么数据结构？
- 还是集合合并，考虑可并堆或者启发式合并。
- 如果用可并堆，在根上打标记，删除的时候 pushdown 一下，时间复杂度 $\mathcal{O}((n+m)\log m)$，做完了。
- 如果用启发式合并，由于要把小堆的值修改一下来和大堆的标记匹配，需要用到浮点数。时间复杂度 $\mathcal{O}(m\log^2 m)$，会有点危险。

基于重链剖分的毛毛虫剖分