数据结构及应用

Data Structures and their Applications

清华大学计算机系 胡泽聪

目录

前言

什么是数据结构？

- 基础数据结构
- 高级数据结构

数据结构在OI中有何应用？

数据结构在OI中处于怎样的地位？ □

数据结构不止是结构

- **数学推导：** 如何维护数据……
- **性质利用：** 单调性、区间减法……
- **思维方法：** 分治、分块、离线……

平衡树

Balanced Trees

二叉查找树

二叉树，节点带权值。
满足左儿子权值小于等于自己，右儿子权值大于等于自己。

二叉查找树

二叉树，节点带权值。
满足左儿子权值小于等于自己，右儿子权值大于等于自己。
形态任意。（有多少种不同形态？）

二叉查找树

二叉树，节点带权值。
满足左儿子权值小于等于自己，右儿子权值大于等于自己。
形态任意。（有多少种不同形态？）
中序遍历即为权值的有序序列。 □

树的操作

- 构建；
- 插入；
- 删除；
- 查找元素；
- 寻找前驱/后继。

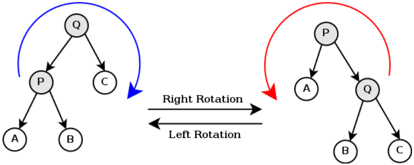
树的平衡

在插入、删除后，我们无法保证树足够平衡。

最坏情况下（顺序插入），树高可以达到 $O(n)$ 级别。而查找的复杂度是树高级别的。

我们需要一些方法调整树的高度。 □

树的旋转



执行这种旋转操作后，性质保持。
右旋又叫zig，左旋又叫zag。
可以通过旋转将树变为任意形态。

□

Splay

Splay的思路很简单。

既然形态可以任意变化，我不妨把我要操作的东西挪到根附近，这样插入删除都比较方便。

splay操作就是将点提到根：通过判断与父亲的关系不断zig或zag。

拓展一下，其实可以将点提到任意（满足性质的前提下）确定的位置。 □

Splay的各种操作

- **splay**：通过判断与父亲的关系不断zig或zag，直到成为根；
- **insert**：先将前驱提到根，再将后继提到根的右儿子，插入到根右儿子的左儿子处；
- **remove**：与insert相反。

□

单旋与双旋

光是这样，Splay是无法保证复杂度的。

一个例子就是顺序插入之后，不断访问第一个和最后一个元素。单次操作$O(n)$。

单旋与双旋

光是这样，Splay是无法保证复杂度的。

一个例子就是顺序插入之后，不断访问第一个和最后一个元素。单次操作 $O(n)$ 。

Tarjan & Sleator (1985) 提出了双旋的办法解决这个问题。

方法很简单：splay操作中，不光考虑和父亲的关系，还考虑父亲和祖父的关系。如果都是左儿子或者都是右儿子，那么先旋转父亲，再旋转自己。

直观理解，这样在一条长链的情况下，可以有效减小树高。

通过势能分析可以证明，操作的均摊复杂度为 $O(\log n)$ 。 □

Splay的区间操作

Splay如果光是这样，是没有太多前途的。

由于Splay形态高度自由（与其他平衡树对比），Splay可以完成几乎所有线段树的区间操作。

- **select**：将左端点的前驱提到根，右端点的后继提到根的右儿子；为了方便操作可以插入起止虚节点；
- **各种线段树操作**：select出来之后对子树操作，维护时稍有区别；
- **区间翻转**：select出来之后打翻转标记，push的时候处理。这个是线段树做不到的。

□

POJ3580 SuperMemo [1]

给定初始长度为 n 的序列，有 q 次操作，操作有六种：

- 区间 $+k$；
- 区间翻转；
- 区间旋转；
- 插入单个元素；
- 删除单个元素；
- 询问区间最小值。

[1] *http://poj.org/problem?id=3580*

POJ3580 SuperMemo [1]

给定初始长度为 n 的序列，有 q 次操作，操作有六种：

- 区间 $+k$；
- 区间翻转；
- 区间旋转；
- 插入单个元素；
- 删除单个元素；
- 询问区间最小值。

标准题目。

□

[1] http://poj.org/problem?id=3580

NOI2005 维修数列 [2]

给定初始长度为 n 的序列，有 q 次操作，操作有六种：

- 插入一段元素；
- 删除一段元素；
- 区间赋值；
- 区间翻转；
- 询问区间和；
- 询问区间最大子段和。

[2] *http://www.lydsy.com/JudgeOnline/problem.php?id=1500*

NOI2005 维修数列 [2]

给定初始长度为 n 的序列，有 q 次操作，操作有六种：

- 插入一段元素；
- 删除一段元素；
- 区间赋值；
- 区间翻转；
- 询问区间和；
- 询问区间最大子段和。

标准题目2。

插入一段的时候，先建出一棵完全二叉树。

最大子段和如何维护？

□

[2] *http://www.lydsy.com/JudgeOnline/problem.php?id=1500*

ZOJ2112 Dynamic Rankings [3]

给定长度为 n 的序列，有 q 次操作，操作有两种：

- 修改单个元素；
- 求区间第 k 大的元素。

[3] http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=2112

ZOJ2112 Dynamic Rankings [3]

给定长度为 n 的序列，有 q 次操作，操作有两种：

- 修改单个元素；
- 求区间第 k 大的元素。

如果单纯求第 k 大可以用可持久化线段树，修改怎么办？

[3] *http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=2112*

ZOJ2112 Dynamic Rankings [3]

给定长度为 n 的序列，有 q 次操作，操作有两种：

- 修改单个元素；
- 求区间第 k 大的元素。

如果单纯求第 k 大可以用可持久化线段树，修改怎么办？

线段树套平衡树。

查询时二分，$O(\log^3 n)$ 。

□

[3] http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=2112

JSOI2008 火星人 [4]

给定长度为 n 的字符串，有 q 次操作，操作有三种：

- 修改单个字符；
- 插入单个字符；
- 查询两个位置开始的后缀的LCP。

[4] *http://www.lydsy.com/JudgeOnline/problem.php?id=1014*

JSOI2008 火星人 [4]

给定长度为 n 的字符串，有 q 次操作，操作有三种：

- 修改单个字符；
- 插入单个字符；
- 查询两个位置开始的后缀的LCP。

维护Hash值，二分判定。$O(\log^2 n)$。

□

[4] *http://www.lydsy.com/JudgeOnline/problem.php?id=1014*

HNOI2011 括号修复 [5]

给定长度为 n 的括号序列（仅含“(”和“)”），有 q 次操作，操作有四种：

- 区间赋值；
- 区间翻转；
- 区间反转；
- 给定区间，求至少要修改多少个位置的括号，才能使得区间中的序列变为合法括号序列。

□

[5] *http://www.lydsy.com/JudgeOnline/problem.php?id=2329*

HNOI2011 括号修复 - 题解

重点在于维护，先考虑单次询问的情况。

HNOI2011 括号修复 - 题解

重点在于维护，先考虑单次询问的情况。

先把序列中所有合法的部分（连着的“()”）全部消去，最后一定剩下一堆右括号接一堆左括号。此时的最优解一定是：对于右括号和左括号的两端，分别隔一个改一个：

$$\begin{array}{c})))(((((\\ \underline{(})\underline{()}(\underline{)}(\underline{)} \end{array}$$

将之前删去的括号都插回去，不难证明仍为合法序列。因此这种方案为合法解。也可以证明这是最优解。 □

HNOI2011 括号修复 - 题解

扩展到静态多次询问问题呢？
我们要维护什么？

HNOI2011 括号修复 - 题解

扩展到静态多次询问问题呢？
我们要维护什么？

括号 $\Longrightarrow$ `+1/-1`序列

HNOI2011 括号修复 - 题解

扩展到静态多次询问问题呢？
我们要维护什么？

括号 $\Longrightarrow$ **+1/-1**序列

右括号的个数 $\Leftrightarrow$ 左起的最小值；左括号的个数 $\Leftrightarrow$ 右起的最大值。
和最大子段和差不多。 □

HNOI2011 括号修复 - 题解

别的操作呢？
翻转：

HNOI2011 括号修复 - 题解

别的操作呢？
翻转：没有影响。
赋值：

HNOI2011 括号修复 - 题解

别的操作呢？
翻转：没有影响。
赋值：没有影响。
反转：

HNOI2011 括号修复 - 题解

别的操作呢？
翻转：没有影响。
赋值：没有影响。
反转：影响大了！

- 还需要维护左起最大值和右起最小值；
- 撞上了赋值标记怎么办？

HNOI2011 括号修复 - 题解

别的操作呢？
翻转：没有影响。
赋值：没有影响。
反转：影响大了！

- 还需要维护左起最大值和右起最小值；
- 撞上了赋值标记怎么办？如果反转先来，打翻转和赋值标记；如果赋值先来，直接反转赋值标记；
- 三种操作的顺序如何？

HNOI2011 括号修复 - 题解

别的操作呢？
翻转：没有影响。
赋值：没有影响。
反转：影响大了！

- 还需要维护左起最大值和右起最小值；
- 撞上了赋值标记怎么办？如果反转先来，打翻转和赋值标记；如果赋值先来，直接反转赋值标记；
- 三种操作的顺序如何？先翻转、再反转，最后赋值。

□

Treap

另一种平衡树。
Tree + Heap。
在二叉树的基础上，存储随机的权重，并按照权重形成堆。
可以证明期望树高 $O(\log n)$ 。
还可以证明修改的期望旋转次数为常数。
因此可以方便地可持久化。（为什么Splay不好可持久化？） □

Treap的操作

和二叉搜索树一样。
操作完成用旋转操作后维护堆的性质。 □

无旋转Treap

基于两个操作：split和merge。

merge类似左偏树的合并，不过不发生交换。

split类似BST查找算法，不过会把找到的下边的分支接到上面。

每次操作最多产生 $O(\log n)$ 个节点，可以直接可持久化。而且可以任意切割合并，可以完成Splay的几乎所有操作。 □

Size-Balanced Tree

国人提出的某种数据结构，通过维护某种子树的大小关系来保持平衡。
似乎和AVL一般快，反正比Splay快。
有兴趣的同学可以自学。 □

替罪羊树

Scapegoat Tree。
非常暴力。
定义了平衡因子 α ，一般取 $0.6 \sim 0.7$ 。
一旦发现有节点的较大儿子的大小超过了整棵子树的大小乘以 α ，则将整棵子树重构为完全二叉树。如果有多个这样的节点，则选择深度最小的。
可以证明均摊复杂度为 $O(\log n)$ 。
有兴趣的同学可以自学。 □

Link-Cut Tree

引入

Splay可以非常灵活地维护一个序列。
树可以视为一条一条链。

引入

Splay可以非常灵活地维护一个序列。
树可以视为一条一条链。

考虑有根树，以及一个点到根的链。
按从根到这个点的顺序抽出节点的序列，用Splay维护。
其他的点呢？

引入

Splay可以非常灵活地维护一个序列。
树可以视为一条一条链。

考虑有根树，以及一个点到根的链。
按从根到这个点的顺序抽出节点的序列，用Splay维护。
其他的点呢？

序列左侧的元素是这个点的祖先，可能出现在右边的元素是这个点的子孙。

我们知道Splay也是树。能不能把两棵树结合一下？ □

启发

我们需要Splay做什么？

启发

我们需要Splay做什么？
——我们需要Splay维护上页中提到的序列，即点与点之间的辈分关系。

启发

我们需要Splay做什么？
——我们需要Splay维护上页中提到的序列，即点与点之间的辈分关系。

Splay需要什么？

启发

我们需要Splay做什么？
——我们需要Splay维护上页中提到的序列，即点与点之间的辈分关系。

Splay需要什么？
——Splay只需要这一偏序关系即可。

启发

我们需要Splay做什么？
——我们需要Splay维护上页中提到的序列，即点与点之间的辈分关系。

Splay需要什么？
——Splay只需要这一偏序关系即可。

Splay的操作对于我们需要维护的关系会产生怎样的影响？

启发

我们需要Splay做什么？
——我们需要Splay维护上页中提到的序列，即点与点之间的辈分关系。

Splay需要什么？
——Splay只需要这一偏序关系即可。

Splay的操作对于我们需要维护的关系会产生怎样的影响？
——考虑简单情况：某个点有两个儿子。我们在这个点在Splay中的右儿子上接两个节点，再对其中一个儿子旋转。
——即便是这么旋转之后，偏序关系也不会变。

启发

我们需要Splay做什么？
——我们需要Splay维护上页中提到的序列，即点与点之间的辈分关系。

Splay需要什么？
——Splay只需要这一偏序关系即可。

Splay的操作对于我们需要维护的关系会产生怎样的影响？
——考虑简单情况：某个点有两个儿子。我们在这个点在Splay中的右儿子上接两个节点，再对其中一个儿子旋转。
——即便是这么旋转之后，偏序关系也不会变。

于是我们可以这么做—— □

定义

我们直接按照树的结构建Splay：树中节点的父亲就是自己在Splay中的父亲。

但是初始时，我们让所有节点的左右儿子都为空。

我们再定义**实边**为，父亲的儿子也有自己的点连向父亲的边。反之则是**虚边**。

从Splay的角度来看，初始时相当于 n 条序列。 □

Expose操作

当我们需要一个点到根的路径时，我们需要把这些点在Splay中都串起来。

也就是从这个点出发，一路向父亲走。如果发现某个点的父亲的右儿子不是自己，就改一下。

当然，还得把最开始那个点的右儿子改为空。

朴素的实现的话，最坏可能有 $O(n)$ 。

Expose操作

当我们需要一个点到根的路径时，我们需要把这些点在Splay中都串起来。

也就是从这个点出发，一路向父亲走。如果发现某个点的父亲的右儿子不是自己，就改一下。

当然，还得把最开始那个点的右儿子改为空。

朴素的实现的话，最坏可能有 $O(n)$ 。

于是Splay就登场啦。

我们用splay操作将当前的点提到“实根”，然后直接改父亲就好。

可以证明均摊复杂度为 $O(\log n)$ 。 □

任意两点间的路径

如果我们不是要到根的路径呢？

任意两点间的路径

如果我们不是要到根的路径呢？

只需要先expose一边，再对另一边做类似的操作。但是，我们需要在LCA的位置停住。

因此，如果某次走到的父亲已经是Splay的根了，那么我们就找到LCA了。（是吗？）

此时我们有两条链，拼起来就是整个路径了。 □

改变根节点

有时我们需要以另一个节点为根。不一定是题目要求，可能是别的操作的要求。

改变根节点

有时我们需要以另一个节点为根。不一定是题目要求，可能是别的操作的要求。

改变根节点，其实是颠倒了两个根之间的辈分关系。

而其它点没有影响。

改变根节点

有时我们需要以另一个节点为根。不一定是题目要求，可能是别的操作的要求。

改变根节点，其实是颠倒了两个根之间的辈分关系。

而其它点没有影响。

也就相当于翻转了这一条链。

因此expose后打翻转标记即可。 □

加边

此时的图应当是森林，加边后仍然应当是森林。

加边

此时的图应当是森林，加边后仍然应当是森林。

两个连通块之间的辈分关系是无所谓的。因此其实就是让其中一个点成为另外整棵树的父亲。

而且，边的另一个端点应当是整棵子树的根。

加边

此时的图应当是森林，加边后仍然应当是森林。

两个连通块之间的辈分关系是无所谓的。因此其实就是让其中一个点成为另外整棵树的父亲。

而且，边的另一个端点应当是整棵子树的根。

那么，只要让在一边换根之后，直接接起来就好。 □

删边

我们先假设这两点间一定有边。

删边

我们先假设这两点间一定有边。
首先要解决的问题是，怎么知道谁是父亲？

删边

我们先假设这两点间一定有边。
首先要解决的问题是，怎么知道谁是父亲？
答案是不知道，只能两边都试一试。
我们用类似加边的思考方法，先找到儿子那边的子树的根。

删边

我们先假设这两点间一定有边。

首先要解决的问题是，怎么知道谁是父亲？

答案是不知道，只能两边都试一试。

我们用类似加边的思考方法，先找到儿子那边的子树的根。

但是又不能找到父亲上面去。因此要先把父亲那边断开。

于是先对父亲做expose的第一步：splay后断开右儿子。此时儿子的实树根的父亲应当是父亲了。

不然我们就可以知道：一定是反着来的。 □

删边

如果两点间没边呢？

删边

如果两点间没边呢？

此时上面的方法就错了！仍然会断开一条边，从而将两点分成两个连通块。

删边

如果两点间没边呢？

此时上面的方法就错了！仍然会断开一条边，从而将两点分成两个连通块。

那么我们换一种找法：expose了儿子之后，父亲一定是儿子的前驱。

这个方法对于有边无边都是对的。

当然要注意push。 □

判断连通性

我们可以用类似找两点间路径的方法。

如果连通，那么在碰到Splay根后，此时根的实树中最右侧的节点一定是另一节点。 □

应用

用LCT可以干什么？

应用

用LCT可以干什么？

- 链修改；
- 链查询；
- 改变树的形态。

其他

能使用其他平衡树吗？

其他

能使用其他平衡树吗？
答案是可以，但是复杂度会变成 $O(\log^2 n)$ 。
我也不知道为什么。有兴趣的同学可以看Tarjan的论文。
对，这个也是Tarjan发明的。 □

2012集训队互测 Tree [6]

n 个节点的树，每个点的初始点权为1。有 q 次操作，操作有四种：

- 将路径上的点权 $+k$；
- 将路径上的点权 $\times k$；
- 求路径上点权和；
- 删去一条边，加上另一条边，保证操作后仍为一棵树。

[6] *http://tsinsen.com/A1303*

2012集训队互测 Tree [6]

n 个节点的树，每个点的初始点权为1。有 q 次操作，操作有四种：

- 将路径上的点权 $+k$ ；
- 将路径上的点权 $\times k$ ；
- 求路径上点权和；
- 删去一条边，加上另一条边，保证操作后仍为一棵树。

最基础的应用。

说到底LCT也还是Splay，Splay维护起来和线段树也差不多。

要注意加和乘的标记应用顺序。

□

[6] *http://tsinsen.com/A1303*

HNOI2010 弹飞绵羊 [7]

地上有 n 个格子，每个格子上有数字 k_i 。

当绵羊到达第 i 个格子时，它会被弹到第 $i+k_i$ 个格子，当 $i+k_i>n$ 时，绵羊被弹飞。

有 q 次操作，操作有两种：

- 求绵羊从第 i 个格子出发，弹几次后会被弹飞；
- 修改一个格子的数字。

[7] http://www.lydsy.com/JudgeOnline/problem.php?id=2002

HNOI2010 弹飞绵羊 [7]

地上有 n 个格子，每个格子上有数字 k_i 。

当绵羊到达第 i 个格子时，它会被弹到第 $i+k_i$ 个格子，当 $i+k_i>n$ 时，绵羊被弹飞。

有 q 次操作，操作有两种：

- 求绵羊从第 i 个格子出发，弹几次后会被弹飞；
- 修改一个格子的数字。

不难发现，如果令每个格子指向其弹到的格子，弹飞的格子指向一个虚拟节点，那么就构成了一棵树。

用LCT维护这个树即可。

我们要知道的是链的长度。

也就是Splay左子树的大小。

有没有更简单的办法？

[7] *http://www.lydsy.com/JudgeOnline/problem.php?id=2002*

HNOI2010 弹飞绵羊 [7]

地上有 n 个格子，每个格子上有数字 k_i 。

当绵羊到达第 i 个格子时，它会被弹到第 $i+k_i$ 个格子，当 $i+k_i>n$ 时，绵羊被弹飞。

有 q 次操作，操作有两种：

- 求绵羊从第 i 个格子出发，弹几次后会被弹飞；
- 修改一个格子的数字。

不难发现，如果令每个格子指向其弹到的格子，弹飞的格子指向一个虚拟节点，那么就构成了一棵树。

用LCT维护这个树即可。

我们要知道的是链的长度。

也就是Splay左子树的大小。

有没有更简单的办法？

Splay维护括号序列？

□

[7] http://www.lydsy.com/JudgeOnline/problem.php?id=2002

WC2006 水管局长 [8]

n 个点 m 条边的图，边带权。有 q 次操作，操作有两种：

- 求两点间路径上最大边权的最小值；
- 删除一条边。

[8] *http://www.lydsy.com/JudgeOnline/problem.php?id=2594*

WC2006 水管局长 [8]

n 个点 m 条边的图，边带权。有 q 次操作，操作有两种：

- 求两点间路径上最大边权的最小值；
- 删除一条边。

一个结论是，能成为答案的边一定在最小生成树上。

（已经是常用结论了）

但是删边怎么办呢？

[8] http://www.lydsy.com/JudgeOnline/problem.php?id=2594

WC2006 水管局长 [8]

n 个点 m 条边的图，边带权。有 q 次操作，操作有两种：

- 求两点间路径上最大边权的最小值；
- 删除一条边。

一个结论是，能成为答案的边一定在最小生成树上。

（已经是常用结论了）

但是删边怎么办呢？

倒过来，变成加边！

[8] http://www.lydsy.com/JudgeOnline/problem.php?id=2594

WC2006 水管局长 [8]

n 个点 m 条边的图，边带权。有 q 次操作，操作有两种：

- 求两点间路径上最大边权的最小值；
- 删除一条边。

一个结论是，能成为答案的边一定在最小生成树上。

（已经是常用结论了）

但是删边怎么办呢？

倒过来，变成加边！

加入一条边会得到一个环，删去环上的最大边。

□

[8] http://www.lydsy.com/JudgeOnline/problem.php?id=2594

BZOJ3091 城市旅行 [9]

n 个点的树，每个点有点权。有 q 次操作，操作有四种：

- 加入一条边；
- 删除一条边；
- 将路径上的点权 $+k$；
- 询问在路径上任取两点，得到的子路径上的点权和的期望值。

你还需要判断给定的操作是否合法。每次操作后图应仍为森林。 □

[9] *http://www.lydsy.com/JudgeOnline/problem.php?id=3091*

BZOJ3091 城市旅行 - 题解

这个题最麻烦的地方在于维护期望值。

我们先推一下式子：假设是一条长度为 n 的链，点权为 $a_1, \ldots, a_n$ ，那么

$$S(l,r) = \sum_{i=l}^{r} a_i$$

$$E =$$

BZOJ3091 城市旅行 - 题解

这个题最麻烦的地方在于维护期望值。

我们先推一下式子：假设是一条长度为 n 的链，点权为 $a_1, \ldots, a_n$ ，那么

$$
\begin{aligned}
S(l,r) &= \sum_{i=l}^{r} a_i \\
E &= \frac{2}{n(n+1)} \sum_{l=1}^{n-1} \sum_{r=l}^{n} S(l,r) \\
&= \frac{2}{n(n+1)} \sum_{i=1}^{n} i(n-i+1) \cdot a_i
\end{aligned}
$$

BZOJ3091 城市旅行 - 题解

这个题最麻烦的地方在于维护期望值。

我们先推一下式子：假设是一条长度为 n 的链，点权为 $a_1, \ldots, a_n$ ，那么

$$\begin{aligned} S(l,r) &= \sum_{i=l}^{r} a_i \\ E &= \frac{2}{n(n+1)} \sum_{l=1}^{n-1} \sum_{r=l}^{n} S(l,r) \\ &= \frac{2}{n(n+1)} \sum_{i=1}^{n} i(n-i+1) \cdot a_i \end{aligned}$$

如何在Splay中维护这个值？ □

BZOJ3091 城市旅行 - 题解

维护以下值：

- $size$ 为树的大小；
- $sum = \sum a_i$ ；
- $lm = \sum i \cdot a_i$ ；
- $rm = \sum (size - i + 1) \cdot a_i$ ；
- E 为期望。

当然，还有得增量。

BZOJ3091 城市旅行 - 题解

维护以下值：

- $size$ 为树的大小；
- $sum = \sum a_i$ ；
- $lm = \sum i \cdot a_i$ ；
- $rm = \sum (size - i + 1) \cdot a_i$ ；
- E 为期望。

当然，还有得增量。

另外，由于需要换根，在处理翻转标记的时候还需要交换 lm 和 rm 。 □

LNOI2014 LCA [10]

n 个节点的有根树，1号节点为根。

有 q 次询问，每次需要求出编号在 $[l, r]$ 间的节点各自与点 x 的LCA的深度之和，即：

$$\sum_{i=l}^{r} \mathrm{depth}[\mathrm{LCA}(i, x)]$$

□

[10] *http://www.lydsy.com/JudgeOnline/problem.php?id=3626*

LNOI2014 LCA - 题解

假设固定一个点，如何快速求出另外一个点与这个点的LCA深度？

LNOI2014 LCA - 题解

假设固定一个点，如何快速求出另外一个点与这个点的LCA深度？
和LCT有何关系？

LNOI2014 LCA - 题解

假设固定一个点，如何快速求出另外一个点与这个点的LCA深度？
和LCT有何关系？
如果固定多个点呢？

LNOI2014 LCA - 题解

假设固定一个点，如何快速求出另外一个点与这个点的LCA深度？
和LCT有何关系？
如果固定多个点呢？

将这些点到根的路径 $+1$ ，那么每个点到根的路径和就是要求的式子。

LNOI2014 LCA - 题解

假设固定一个点，如何快速求出另外一个点与这个点的LCA深度？
和LCT有何关系？
如果固定多个点呢？

将这些点到根的路径 $+1$ ，那么每个点到根的路径和就是要求的式子。

区间怎么处理？

LNOI2014 LCA - 题解

假设固定一个点，如何快速求出另外一个点与这个点的LCA深度？
和LCT有何关系？
如果固定多个点呢？

将这些点到根的路径 $+1$ ，那么每个点到根的路径和就是要求的式子。

区间怎么处理？
拆成两个前缀，离线处理。 □

Chef and Graph Queries [11]

给定一个 n 个点 m 条边的图，每条边编号为 $1 \sim m$ 。有 q 次询问，每次给定 l 和 r ，询问当仅保留编号为 $l \sim r$ 的边时，图中有多少个连通块。

$n, m, q \leq 200000$ 。可能有自环与重边。 □

[11] *https://www.codechef.com/MARCH14/problems/GERALD07*

Chef and Graph Queries - 题解

我们考虑按照 $1 \sim m$ 的顺序依次向空图中加边。加入一条边时，只会有两种情况：

1. 合并了两个连通块；
2. 形成了一个环。

Chef and Graph Queries - 题解

我们考虑按照 $1 \sim m$ 的顺序依次向空图中加边。加入一条边时，只会有两种情况：

1. 合并了两个连通块；
2. 形成了一个环。

先看第一种，只有合并两个连通块时才会产生贡献。我们要求的实际上就是一个区间中这类边有多少条。

Chef and Graph Queries - 题解

我们考虑按照 $1 \sim m$ 的顺序依次向空图中加边。加入一条边时，只会有两种情况：

1. 合并了两个连通块；
2. 形成了一个环。

先看第一种，只有合并两个连通块时才会产生贡献。我们要求的实际上就是一个区间中这类边有多少条。

再看第二种，形成环则代表可以删去图中的一条边，并保持当前的连通性。

不妨删去环上加入时间最早（即编号最小）的一条边。我们发现了什么？ □

Chef and Graph Queries - 题解

假设加入的边为 i ，删去的边为 a_i 。只有当 i 加入的时候才能“安全”删除 a_i ，否则会导致一个连通块分成两个。

Chef and Graph Queries - 题解

假设加入的边为 i ，删去的边为 a_i 。只有当 i 加入的时候才能“安全”删除 a_i ，否则会导致一个连通块分成两个。

换句话说，i 会产生贡献 $\iff$ 选择的区间中有 i 且没有 a_i $\square$

Chef and Graph Queries - 题解

假设我们已求得 a 。当我们询问 $[l, r]$ 时，我们实际上需要知道什么？

Chef and Graph Queries - 题解

假设我们已求得 a 。当我们询问 $[l, r]$ 时，我们实际上需要知道什么？

我们需要知道在 $[l, r]$ 中有多少 $a_i < l$ 。设其个数为 x ，连通块的个数就是 $n - x$ 。

Chef and Graph Queries - 题解

假设我们已求得 a 。当我们询问 $[l,r]$ 时，我们实际上需要知道什么？

我们需要知道在 $[l,r]$ 中有多少 $a_i < l$ 。设其个数为 x ，连通块的个数就是 $n-x$ 。

用可持久化线段树可以轻松解决这一询问。 □

Chef and Graph Queries - 题解

至于求 a ，只需维护一棵LCT，按 $1 \sim m$ 的顺序加边。

加入一条边时，判断是否成环。如果构成了环，则找到环上编号最小的边的编号，即为 a_i 。

如果没有成环，则记 $a_i = 0$ 。注意当边为自环时，$a_i = i$ 。

总复杂度 $O((n+m)\log n)$ 。 □

Gangsters of Treeland

给定一棵 n 个点的树，1 号节点为根。初始时每一个点都被染成了一种不同的颜色。如果一条边的两个端点颜色不同，则其费用为 1 ，否则费用为 0 。

有 q 次操作，操作有下面两种：

- 将从点 u 到根的路径上的所有点染成一种新的颜色。
- 询问点 u 子树中所有点走到根的费用的平均数。

$n, q \leq 100000$ 。 □

Gangsters of Treeland - Solution

直接维护每个点的颜色，查询时数链上有多少种不同的颜色？

Gangsters of Treeland - Solution

直接维护每个点的颜色，查询时数链上有多少种不同的颜色？
带修改的树上第 k 大？或者是分块乱搞？无论哪个复杂度都太高。

Gangsters of Treeland - Solution

直接维护每个点的颜色，查询时数链上有多少种不同的颜色？
带修改的树上第 k 大？或者是分块乱搞？无论哪个复杂度都太高。

为什么非得纠结颜色呢？可不可以利用“每次修改的颜色是一种新颜色”这样的性质？ □

Gangsters of Treeland - Solution

我们考虑直接维护边的费用。

Gangsters of Treeland - Solution

我们考虑直接维护边的费用。

初始时所有边的费用均为 1 ，修改一个点时，这个点到根的路径上的所有边的费用变为 0 ，而其他和这条路径上的点相连的费用变为 1 。

Gangsters of Treeland - Solution

我们考虑直接维护边的费用。

初始时所有边的费用均为 1 ，修改一个点时，这个点到根的路径上的所有边的费用变为 0 ，而其他和这条路径上的点相连的费用变为 1 。

一次操作影响的边可能达到 $O(n)$ ，直接操作是肯定不行的。一定还有别的性质。

Gangsters of Treeland - Solution

我们考虑直接维护边的费用。

初始时所有边的费用均为 1 ，修改一个点时，这个点到根的路径上的所有边的费用变为 0 ，而其他和这条路径上的点相连的费用变为 1 。

一次操作影响的边可能达到 $O(n)$ ，直接操作是肯定不行的。一定还有别的性质。

每个点和儿子的连边中，至多一条费用为 0 。

Gangsters of Treeland - Solution

我们考虑直接维护边的费用。

初始时所有边的费用均为 1 ，修改一个点时，这个点到根的路径上的所有边的费用变为 0 ，而其他和这条路径上的点相连的费用变为 1 。

一次操作影响的边可能达到 $O(n)$ ，直接操作是肯定不行的。一定还有别的性质。

每个点和儿子的连边中，至多一条费用为 0 。

这似乎让人想起了什么……特别熟悉的东西……

Gangsters of Treeland - Solution

我们考虑直接维护边的费用。

初始时所有边的费用均为 1 ，修改一个点时，这个点到根的路径上的所有边的费用变为 0 ，而其他和这条路径上的点相连的费用变为 1 。

一次操作影响的边可能达到 $O(n)$ ，直接操作是肯定不行的。一定还有别的性质。

每个点和儿子的连边中，至多一条费用为 0 。

这似乎让人想起了什么……特别熟悉的东西……

这不就是一棵Link-Cut Tree吗？ □

Gangsters of Treeland - Solution

费用为 0 的边就是LCT的实边，费用为 1 的边就是LCT的虚边。每次操作一个点就相当于expose（也称access）一个点。

而一个点走到根的费用就是到根路径上的虚边条数。

容易发现这个和原问题是等价的。

Gangsters of Treeland - Solution

费用为 0 的边就是LCT的实边，费用为 1 的边就是LCT的虚边。每次操作一个点就相当于expose（也称access）一个点。

而一个点走到根的费用就是到根路径上的虚边条数。

容易发现这个和原问题是等价的。

还记得LCT的复杂度吗？expose操作是均摊 $O(\log n)$ 的。

Gangsters of Treeland - Solution

费用为 0 的边就是LCT的实边，费用为 1 的边就是LCT的虚边。每次操作一个点就相当于expose（也称access）一个点。

而一个点走到根的费用就是到根路径上的虚边条数。

容易发现这个和原问题是等价的。

还记得LCT的复杂度吗？expose操作是均摊 $O(\log n)$ 的。

也就是说，expose时的“关键点”，即改变了实边的点，是均摊 $O(\log n)$ 的。

换言之，我们只有至多 $O(n \log n)$ 次实质上的修改！ $\square$

Gangsters of Treeland - Solution

我们实现一棵LCT，修改一个点时进行expose操作，每找到一个关键点就修改一次。

而我们要维护的，就是每个点到根的虚边条数。

Gangsters of Treeland - Solution

我们实现一棵LCT，修改一个点时进行expose操作，每找到一个关键点就修改一次。

而我们要维护的，就是每个点到根的虚边条数。

这个就很简单了。求出DFS序之后建线段树，每次虚边和实边切换的时候就做一次段修改。查询则直接是段查询。

总复杂度 $O(n \log^2 n)$ 。 □

Gangsters of Treeland - Extras

是不是觉得转化略神？

实际上还可以更神。由于问题等价于LCT，我们可以增加其它LCT支持的操作，比如换根。

Gangsters of Treeland - Extras

是不是觉得转化略神？

实际上还可以更神。由于问题等价于LCT，我们可以增加其它LCT支持的操作，比如换根。

回忆换根的实现：先把要提成根的点expose，再splay到根，之后再打翻转标记。那么我们可以给题目增加这么一个操作：换根，同时把新旧根之间的路径染成新的颜色。

Gangsters of Treeland - Extras

是不是觉得转化略神？

实际上还可以更神。由于问题等价于LCT，我们可以增加其它LCT支持的操作，比如换根。

回忆换根的实现：先把要提成根的点expose，再splay到根，之后再打翻转标记。那么我们可以给题目增加这么一个操作：换根，同时把新旧根之间的路径染成新的颜色。

问题在于线段树那部分要怎么实现。我们需要支持子树查询、子树修改，以及换根。

Gangsters of Treeland - Extras

是不是觉得转化略神？

实际上还可以更神。由于问题等价于LCT，我们可以增加其它LCT支持的操作，比如换根。

回忆换根的实现：先把要提成根的点expose，再splay到根，之后再打翻转标记。那么我们可以给题目增加这么一个操作：换根，同时把新旧根之间的路径染成新的颜色。

问题在于线段树那部分要怎么实现。我们需要支持子树查询、子树修改，以及换根。

实际上也是可做的。这里就不说了，有兴趣的同学可以自己思考或者在课后与我交流。

Gangsters of Treeland - Extras

是不是觉得转化略神？

实际上还可以更神。由于问题等价于LCT，我们可以增加其它LCT支持的操作，比如换根。

回忆换根的实现：先把要提成根的点expose，再splay到根，之后再打翻转标记。那么我们可以给题目增加这么一个操作：换根，同时把新旧根之间的路径染成新的颜色。

问题在于线段树那部分要怎么实现。我们需要支持子树查询、子树修改，以及换根。

实际上也是可做的。这里就不说了，有兴趣的同学可以自己思考或者在课后与我交流。

至于LCT的别的操作能不能支持呢？这个我没有细想，同样，有兴趣的同学可以自己思考这个问题。 □

分块方法

SQRT-N Decomposition

朴素分块方法

直接将序列分成 $O\left(\sqrt{n}\right)$ 块。

朴素分块方法

直接将序列分成 $O\left(\sqrt{n}\right)$ 块。
每个区间会对应 $O\left(n/\sqrt{n}\right) = O\left(\sqrt{n}\right)$ 块，以及 $O\left(\sqrt{n}\right)$ 个单独的元素。
因此单次操作的复杂度为 $O\left(\sqrt{n}\right)$ 。配合标记、重构，可以完成大多数操作。

朴素分块方法

直接将序列分成 $O\left(\sqrt{n}\right)$ 块。

每个区间会对应 $O\left(n/\sqrt{n}\right) = O\left(\sqrt{n}\right)$ 块，以及 $O\left(\sqrt{n}\right)$ 个单独的元素。

因此单次操作的复杂度为 $O\left(\sqrt{n}\right)$ 。配合标记、重构，可以完成大多数操作。

根据不同操作的复杂度、出现频率，可以调整块的大小，达到更优的复杂度。 □

按大小分类

有时候我们不一定非要把序列分块：我们可以按照权值、出现次数、参数大小等分类。

说起来比较抽象，通过例题体会一下。 □

CF80D Time to Raid Cowavans[12]

给定长度为 n 的序列 $a[1\sim n]$，有 q 次询问。

每次询问给定 (x,y)，求 $a[x]+a[x+y]+a[x+2y]+\cdots$

[12] *http://codeforces.com/contest/103/problem/D*

CF80D Time to Raid Cowavans [12]

给定长度为 n 的序列 $a[1 \sim n]$，有 q 次询问。

每次询问给定 (x, y)，求 $a[x] + a[x+y] + a[x+2y] + \cdots$

要求的东西不连续，传统的数据结构无用武之地。

y 的取值任意，也不好预处理。

[12] http://codeforces.com/contest/103/problem/D

CF80D Time to Raid Cowavans [12]

给定长度为 n 的序列 $a[1\sim n]$，有 q 次询问。

每次询问给定 (x,y)，求 $a[x]+a[x+y]+a[x+2y]+\cdots$

要求的东西不连续，传统的数据结构无用武之地。

y 的取值任意，也不好预处理。

不妨按照 y 的大小分类：

[12] http://codeforces.com/contest/103/problem/D

CF80D Time to Raid Cowavans [12]

给定长度为 n 的序列 $a[1\sim n]$，有 q 次询问。

每次询问给定 (x,y)，求 $a[x]+a[x+y]+a[x+2y]+\cdots$

要求的东西不连续，传统的数据结构无用武之地。

y 的取值任意，也不好预处理。

不妨按照 y 的大小分类：

- 如果 $y\le\sqrt{n}$，那么预处理出每个位置的答案，用类似后缀求和的方法可以做到 $O\left(n\sqrt{n}\right)$；
- 如果 $y>\sqrt{n}$，那么一次询问设计的元素个数不超过 $n/\sqrt{n}=O\left(\sqrt{n}\right)$，直接统计。

□

[12] *http://codeforces.com/contest/103/problem/D*

经典题目

给定 n 个点 m 条边的无向图，每个点是黑色或者白色。有 q 次操作，操作有两种：

- 将一个点反色；
- 询问与一个点直接相连的黑色点的个数。

经典题目

给定 n 个点 m 条边的无向图，每个点是黑色或者白色。有 q 次操作，操作有两种：

- 将一个点反色；
- 询问与一个点直接相连的黑色点的个数。

直观感受可以得出，绝大多数点的度数都比较小。

但是度数大的点还是可能存在不少。

经典题目

给定 n 个点 m 条边的无向图，每个点是黑色或者白色。有 q 次操作，操作有两种：

- 将一个点反色；
- 询问与一个点直接相连的黑色点的个数。

直观感受可以得出，绝大多数点的度数都比较小。

但是度数大的点还是可能存在不少。

因此按照度数分类：

- 对于度数 $\leq \sqrt{n}$ 的点，修改/询问时枚举相连的点；
- 对于度数 $> \sqrt{n}$ 的点，这类点的个数最多 $O\left(m/\sqrt{n}\right) = O\left(\sqrt{n}\right)$ 个，因此在修改其他点时，同时更新这些点的答案；修改这类点时不管其它点，询问时直接输出答案。

$\square$

莫队算法

莫队算法是处理区间问题的离线算法，相当万能。

它的思路是：我们将询问区间以某种顺序重排，然后依次处理。从前一个区间移到下一个时，**暴力移动**：一个一个删除、一个一个插入。

这么暴力的方法，可以做到怎样的复杂度？

莫队算法

莫队算法是处理区间问题的离线算法，相当万能。

它的思路是：我们将询问区间以某种顺序重排，然后依次处理。从前一个区间移到下一个时，**暴力移动**：一个一个删除、一个一个插入。

这么暴力的方法，可以做到怎样的复杂度？

答案是 $O\left(n\sqrt{n}\right)$ 次插入/删除。

将区间按照左端点分块，同一块内的按照右端点排序。 □

Tsinsen A1206 小Z的袜子 [13]

长度为 n 的序列，有 q 个询问。
每次给定区间 $[l, r]$ ，问在区间中随机抽取两个元素，其值相同的概率。

[13] *http://tsinsen.com/A1206*

Tsinsen A1206 小Z的袜子[13]

长度为 n 的序列，有 q 个询问。每次给定区间 $[l, r]$，问在区间中随机抽取两个元素，其值相同的概率。

传统的数据结构几乎无从下手。但是使用莫队算法，可以轻易做到 $O\left(n\sqrt{n}\right)$。

$\square$

[13] *http://tsinsen.com/A1206*

Paint Pearls

给定一个长度为 n 的序列，每一位有一个目标颜色。初始时每一位都没有颜色。每次可以选择一个区间，将区间内的所有元素改为其**目标颜色**。设区间内不同颜色的数量为 x ，则操作的代价为 x^2 。求最小代价。

$n \leq 50000$ 。 □

Paint Pearls - Solution

朴素的DP为：令 $f[i]$ 表示前 i 个元素变为目标颜色的最小代价，方程为

$$f[i] = \min\{f[j-1] + w(j,i)\}$$

复杂度 $O(n^2)$ 。 $\square$

Paint Pearls - Solution

容易发现答案的上界为 n —每次操作一个元素即可。

Paint Pearls - Solution

容易发现答案的上界为 n —每次操作一个元素即可。

基于上界又可以发现，如果一个区间内有超过 $\sqrt{n}$ 种颜色，我们一定不会去操作它。

Paint Pearls - Solution

容易发现答案的上界为 n —每次操作一个元素即可。

基于上界又可以发现，如果一个区间内有超过 $\sqrt{n}$ 种颜色，我们一定不会去操作它。

换句话说，对于一个 $f[i]$ ，只需要考虑 $\sqrt{n}$ 个不同的 $w(j, i)$ 的取值。

Paint Pearls - Solution

容易发现答案的上界为 n —每次操作一个元素即可。

基于上界又可以发现，如果一个区间内有超过 $\sqrt{n}$ 种颜色，我们一定不会去操作它。

换句话说，对于一个 $f[i]$ ，只需要考虑 $\sqrt{n}$ 个不同的 $w(j,i)$ 的取值。

而 f 显然单调不减，因此 $w(j,i)$ 相同时选择最靠左的 $f[j-1]$ 。

Paint Pearls - Solution

容易发现答案的上界为 n —每次操作一个元素即可。

基于上界又可以发现，如果一个区间内有超过 $\sqrt{n}$ 种颜色，我们一定不会去操作它。

换句话说，对于一个 $f[i]$ ，只需要考虑 $\sqrt{n}$ 个不同的 $w(j,i)$ 的取值。

而 f 显然单调不减，因此 $w(j,i)$ 相同时选择最靠左的 $f[j-1]$ 。

我们只需知道这些 j 的值。 □

Paint Pearls - Solution

只要记录当前位置往左出现的前 $\sqrt{n}$ 种颜色，及对应位置即可。

移动到下一个数时，如果颜色出现在了前 $\sqrt{n}$ 种之中，则暴力删除并移到最前面。

总复杂度 $O\left(n\sqrt{n}\right)$ 。

Paint Pearls - Solution

只要记录当前位置往左出现的前 $\sqrt{n}$ 种颜色，及对应位置即可。

移动到下一个数时，如果颜色出现在了前 $\sqrt{n}$ 种之中，则暴力删除并移到最前面。

总复杂度 $O\left(n\sqrt{n}\right)$ 。

其实，数据中最优解选的每个区间都只有不超过5种颜色…… □

Sereja and Arcs

给定一个长度为 n 的序列 a ，对于任意的 $x \neq y$ ，如果 $a[x] = a[y]$ ，则在 x 和 y 之间画一条弧。称两条弧 x, y 和 l, r 相交当且仅当 $x < l < y < r$ 或者 $l < x < r < y$ 。求有多少条异色弧相交。

$n \leq 100000$ ， $a[i] \leq 100000$ 。时限5s。 □

Sereja and Arcs - 题解

我们把圆弧视为线段，问题就是求有多少对有交且端点形如1212的异色线段。

Sereja and Arcs - 题解

我们把圆弧视为线段，问题就是求有多少对有交且端点形如1212的异色线段。

将颜色按出现次数分类，次数大于 T 的为大块，其他的为小块。
那么我们需要处理三种情况：

1. 大块之间的贡献；
2. 小块与大块之间的贡献；
3. 小块之间的贡献。

□

Sereja and Arcs - 题解

大块之间的贡献

Sereja and Arcs - 题解

大块之间的贡献

我们枚举1212中的第二个1，再枚举另外一种颜色。

可以发现，答案为第二个1右边2的出现次数，乘上左边每个2的左侧的1的个数之和。

Sereja and Arcs - 题解

大块之间的贡献

我们枚举1212中的第二个1，再枚举另外一种颜色。

可以发现，答案为第二个1右边2的出现次数，乘上左边每个2的左侧的1的个数之和。

由于大块个数不超过 $O\left(\frac{n}{T}\right)$，我们可以预处理每个位置左侧每种颜色的出现次数。

用这个就能维护，对于某种颜色 col，和某个位置以左与这个位置颜色相同的所有位置，其左侧的 col 颜色的个数之和。

计算时枚举一个位置和一种颜色。总复杂度 $O\left(\frac{n^2}{T}\right)$。 □

Sereja and Arcs - 题解

小块与大块之间的贡献

Sereja and Arcs - 题解

小块与大块之间的贡献
枚举每个大块，再枚举每个小块。

Sereja and Arcs - 题解

小块与大块之间的贡献

枚举每个大块，再枚举每个小块。

按照小块的每个元素把大块的元素划成若干区间，求出每个区间里的大块元素个数，记这个序列为 a 。

假设我们枚举小块的左右端点，那么大块的两个元素的选择就是

$$(\text{左边}+\text{右边})\times\text{中间}$$

$\square$

Sereja and Arcs - 题解

小块与大块之间的贡献

Sereja and Arcs - 题解

小块与大块之间的贡献

考虑当 a 的某一个元素成为“中间”部分时的贡献，我们需要求出这个元素为“中间”时可能的“左右”之和。

这个元素左边第一块会被算一次，第二块被算两次，以此类推。整个这一部分还要再乘上右端点的可能的个数。

右边也是类似的。这一部分是可以通过一些简单的预处理求出来的。

那么复杂度也是 $O\left(\frac{n^2}{T}\right)$。 □

Sereja and Arcs - 题解

小块之间的贡献

Sereja and Arcs - 题解

小块之间的贡献

先考虑一个 $O(n^3)$ 的算法，枚举1212中2的左右端点，再枚举另外一种颜色。

此时相交的线段数就是该颜色在两个2之间的出现次数，乘上第一个2左边的出现次数。

Sereja and Arcs - 题解

小块之间的贡献

先考虑一个 $O(n^3)$ 的算法，枚举1212中2的左右端点，再枚举另外一种颜色。

此时相交的线段数就是该颜色在两个2之间的出现次数，乘上第一个2左边的出现次数。

如果左端点从右往左扫，则可以在 $O(1)$ 的时间内处理变化，并直接计算对答案的贡献的增量，从而优化到 $O(n^2)$ 。

Sereja and Arcs - 题解

小块之间的贡献

先考虑一个 $O(n^3)$ 的算法，枚举1212中2的左右端点，再枚举另外一种颜色。

此时相交的线段数就是该颜色在两个2之间的出现次数，乘上第一个2左边的出现次数。

如果左端点从右往左扫，则可以在 $O(1)$ 的时间内处理变化，并直接计算对答案的贡献的增量，从而优化到 $O(n^2)$ 。

令 $f[i]$ 代表在当前右端点下，左端点为 i 时相交的线段数。不妨考虑，右端点右移时，f 会如何变化。

如果我们能维护 f ，那么只需要枚举与右端点颜色相同的位置。 □

Sereja and Arcs - 题解

小块之间的贡献

Sereja and Arcs - 题解

小块之间的贡献

我们考虑只枚举与右端点颜色相同的位置，此时对 f 的影响是按照这些位置分段的，每一段内的增量相同。

也就是说，右端点右移会导致若干个段的修改，因此我们可以用树状数组维护答案。

Sereja and Arcs - 题解

小块之间的贡献

我们考虑只枚举与右端点颜色相同的位置，此时对 f 的影响是按照这些位置分段的，每一段内的增量相同。

也就是说，右端点右移会导致若干个段的修改，因此我们可以用树状数组维护答案。

令 x_i 为颜色 i 的出现次数，复杂度应该是 $O\left(\left(\sum x_i^2\right)\log n\right)$ 。

而 $\max\left\{\sum x_i^2\right\}$ 其实是 $O(nT)$ 级别的。

考虑给某个 x_i 加 1 带来的增量 $\Delta = 2x_i + 1$ ，因此一定会选择最大的 x_i 去加1。

换句话说，当有 $\frac{n}{T}$ 个 x_i 为 T 时原式取到最大值。因此复杂度为 $O(nT\log n)$ 。 □

Sereja and Arcs - 题解

综上，取

$$T = \sqrt{\frac{n}{\log n}}$$

可得最优复杂度 $O\left(n\sqrt{n\log n}\right)$ 。时限有5s，毫无压力。

Sereja and Arcs - 题解

综上，取

$$T = \sqrt{\frac{n}{\log n}}$$

可得最优复杂度 $O\left(n\sqrt{n \log n}\right)$ 。时限有5s，毫无压力。

但可以发现，$O\left(\frac{n^2}{T}\right)$ 的有两个部分，$O(nT \log n)$ 的只有一部分，因此应适当调大 T 。

经过实测，取 $T = \left\lfloor 1.8 \cdot \sqrt{\frac{n}{\log n}} \right\rfloor$ 时效果最好，只需要2.48s即可通过全部数据。 □

在结束之前

如何学好数据结构？

如何在考试中做出数据结构题？

真的能在考试中写对数据结构吗？

NOI水平的数据结构题大概是什么样的难度？ □

Fin.

谢谢大家！欢迎课后交流。 $\square$