一些乱七八组的东西

greatinfluence

雅礼中学

December 14, 2018

Background

因为讲课是随机分配的，所以有的东西我也是乱学的，敬请海涵。

kd-tree

K-dimensional tree(简称kd-tree)，是一种用于解决大规模的高维数据空间进行近期邻查找(Nearest Neighbor)和近似近期邻查找(Approximate Nearest Neighbor)问题的数据结构。

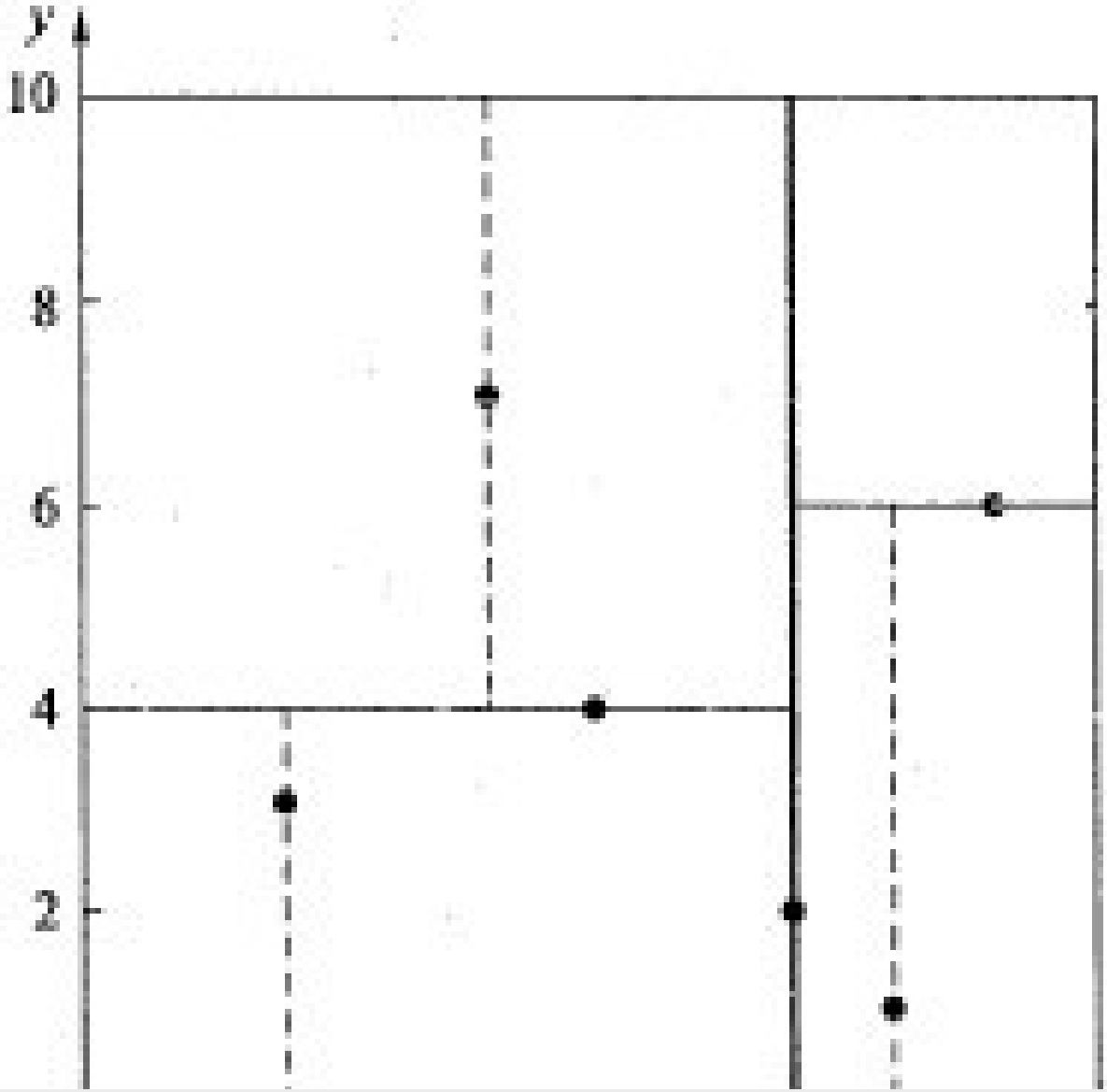
K-dimensional tree(简称kd-tree)，是一种用于解决大规模的高维数据空间进行近期邻查找(Nearest Neighbor)和近似近期邻查找(Approximate Nearest Neighbor)问题的数据结构。

然而在OI中，我们只用这个解决k近邻问题(并且k主要是1)。

K-dimensional tree(简称kd-tree)，是一种用于解决大规模的高维数据空间进行近期邻查找(Nearest Neighbor)和近似近期邻查找(Approximate Nearest Neighbor)问题的数据结构。

然而在OI中，我们只用这个解决k近邻问题(并且k主要是1)。

我们先看看一棵kd-tree会长什么样。


y
10
8
6
4
2

如图，每个点都会选择一个切割的维度，然后将它所在空间切成两部分，然后递归建树直到空间内只有1个点位置。

如图，每个点都会选择一个切割的维度，然后将它所在空间切成两部分，然后递归建树直到空间内只有1个点位置。

因此，kd-tree是一棵二叉树。

现在我们先不管怎么用，来看看如何去构建一棵kd-tree。

kd-tree的构建非常简单。

kd-tree的构建非常简单。

1.对于每个点，观察它代表空间的每个维度，确认极差最大的一维。

kd-tree的构建非常简单。

1.对于每个点，观察它代表空间的每个维度，确认极差最大的一维。

2.将代表空间包含的所有点按照这一维排序，然后找到区间中点。

kd-tree的构建非常简单。

1.对于每个点，观察它代表空间的每个维度，确认极差最大的一维。

2.将代表空间包含的所有点按照这一维排序，然后找到区间中点。

3.从这个点将空间划分成两部分，然后将两部分的点分别递归建树。

kd-tree的构建非常简单。

1.对于每个点，观察它代表空间的每个维度，确认极差最大的一维。

2.将代表空间包含的所有点按照这一维排序，然后找到区间中点。

3.从这个点将空间划分成两部分，然后将两部分的点分别递归建树。

具体怎么打可以看附加文件。

kd-tree的构建非常简单。

1.对于每个点，观察它代表空间的每个维度，确认极差最大的一维。

2.将代表空间包含的所有点按照这一维排序，然后找到区间中点。

3.从这个点将空间划分成两部分，然后将两部分的点分别递归建树。

具体怎么打可以看附加文件。

注意第一步你也可以每一维按顺序依次剖开，但是会被特殊数据构造到复杂度错误。

建完树后，我们可以开始看看这个东西有什么用了。

建完树后，我们可以开始看看这个东西有什么用了。

主要应用就是最近邻查询。

最近邻查询的步骤大概就是一下几步。

最近邻查询的步骤大概就是一下几步。

1.首先访问根节点。

最近邻查询的步骤大概就是一下几步。

1.首先访问根节点。

2.用当前点到查询点的距离更新答案。

最近邻查询的步骤大概就是一下几步。

1.首先访问根节点。

2.用当前点到查询点的距离更新答案。

3.找到查询点被分到了左右儿子的哪一个里面，然后访问那个儿子，并返回2。

最近邻查询的步骤大概就是一下几步。

1.首先访问根节点。

2.用当前点到查询点的距离更新答案。

3.找到查询点被分到了左右儿子的哪一个里面，然后访问那个儿子，并返回2。

4.回溯的时候，如果查询点加上当前最近距离会越过剖分面，则访问另一个儿子。

最近邻查询的步骤大概就是一下几步。

1.首先访问根节点。

2.用当前点到查询点的距离更新答案。

3.找到查询点被分到了左右儿子的哪一个里面，然后访问那个儿子，并返回2。

4.回溯的时候，如果查询点加上当前最近距离会越过剖分面，则访问另一个儿子。

5.实现过于简单请自行YY。

以上的查询算法的时间复杂度为单次$O(kn^{\frac{k-1}{k}})$(k为维度)。

以上的查询算法的时间复杂度为单次$O(kn^{\frac{k-1}{k}})$(k为维度)。

明显当k达到一定大小时，查询效率极低与暴力差不多。

以上的查询算法的时间复杂度为单次$O(kn^{\frac{k-1}{k}})$(k为维度)。

明显当k达到一定大小时，查询效率极低与暴力差不多。

而实际应用中，我们有时并不需要知道最优解，只需要一个近似解就可以了。(然而在OI中没有这样的好事)

以上的查询算法的时间复杂度为单次$O(kn^{\frac{k-1}{k}})$(k为维度)。

明显当k达到一定大小时，查询效率极低与暴力差不多。

而实际应用中，我们有时并不需要知道最优解，只需要一个近似解就可以了。(然而在OI中没有这样的好事)

这时就可以利用近似搜索算法了。

近似搜索算法的流程很简单。

近似搜索算法的流程很简单。

1.确定最大回溯层数B。(一般是手调最优值)

近似搜索算法的流程很简单。

1.确定最大回溯层数B。(一般是手调最优值)

2.像正常算法一样，先从根开始一路搜索查询点。

近似搜索算法的流程很简单。

1.确定最大回溯层数B。(一般是手调最优值)

2.像正常算法一样，先从根开始一路搜索查询点。

3.回溯时，如果另一个儿子可能存在最优解，我们将这个儿子放进优先队列里，按照代表超平面和查询点的距离从小到大排序。

近似搜索算法的流程很简单。

1.确定最大回溯层数B。(一般是手调最优值)

2.像正常算法一样，先从根开始一路搜索查询点。

3.回溯时，如果另一个儿子可能存在最优解，我们将这个儿子放进优先队列里，按照代表超平面和查询点的距离从小到大排序。

4.跑完后，如果未达到最大回溯层数并且优先队列非空，那么每次取出距离最小的节点，作为根继续搜索。

近似搜索算法的流程很简单。

1.确定最大回溯层数B。(一般是手调最优值)

2.像正常算法一样，先从根开始一路搜索查询点。

3.回溯时，如果另一个儿子可能存在最优解，我们将这个儿子放进优先队列里，按照代表超平面和查询点的距离从小到大排序。

4.跑完后，如果未达到最大回溯层数并且优先队列非空，那么每次取出距离最小的节点，作为根继续搜索。

这样的复杂度可以确认为单次$O(Bk \log n)$。

Luogu P4169[Violet]天使玩偶/SJY摆棋子

需要维护一个结构，要求支持以下2个操作。

1.向二位空间内插入一个点。

2.给出一个二维点，询问距离这个点曼哈顿距离最近的点的曼哈顿距离。

$n, m \leq 3 \times 10^5, T = 3s$

本题可以使用cdq分治解决，这个后面会讲(也许吧)。

本题可以使用cdq分治解决，这个后面会讲(也许吧)。

但是与此同时，也可以使用kd-tree做到按题意模拟。

本题可以使用cdq分治解决，这个后面会讲(也许吧)。

但是与此同时，也可以使用kd-tree做到按题意模拟。

不过kd-tree的结构不支持改变，而暴力插入会导致树形结构严重失衡影响复杂度。

本题可以使用cdq分治解决，这个后面会讲(也许吧)。

但是与此同时，也可以使用kd-tree做到按题意模拟。

不过kd-tree的结构不支持改变，而暴力插入会导致树形结构严重失衡影响复杂度。

这时只需要在外面套用一层替罪羊就可以了。

本题可以使用cdq分治解决，这个后面会讲(也许吧)。

但是与此同时，也可以使用kd-tree做到按题意模拟。

不过kd-tree的结构不支持改变，而暴力插入会导致树形结构严重失衡影响复杂度。

这时只需要在外面套用一层替罪羊就可以了。

替罪羊不会的问oyiya大佬。

时间复杂度$O(m\sqrt{n})$。

块状链表

这个东西用处不大。

这个东西用处不大。

能够实现的平衡树都能够实现，且复杂度更优秀。

这个东西用处不大。

能够实现的平衡树都能够实现，且复杂度更优秀。

但是它比平衡树好写，至少代码量更少。

这个东西用处不大。

能够实现的平衡树都能够实现，且复杂度更优秀。

但是它比平衡树好写，至少代码量更少。

可以给懒癌晚期选手用来替代部分平衡树。

首先我们先回忆数组和链表的区别。

首先我们先回忆数组和链表的区别。

我们可以发现数组的访问可以做到$O(1)$，而链表只能做到$O(n)$。

首先我们先回忆数组和链表的区别。

我们可以发现数组的访问可以做到$O(1)$，而链表只能做到$O(n)$。

但是相对的，数组的中部插入删除元素只能做到$O(n)$，而链表可以做到$O(1)$。

首先我们先回忆数组和链表的区别。

我们可以发现数组的访问可以做到$O(1)$，而链表只能做到$O(n)$。

但是相对的，数组的中部插入删除元素只能做到$O(n)$，而链表可以做到$O(1)$。

那么，有没有一种结构，可以综合这二者的优缺点呢?

~~平衡树~~

为了实现二者优缺点的综合，我们将采用一下做法。

为了实现二者优缺点的综合，我们将采用一下做法。

先确定一个常数B。

为了实现二者优缺点的综合，我们将采用一下做法。

先确定一个常数B。

我们使用长度为B的数组来维护每小一部分，数组间采用链表链接。

为了实现二者优缺点的综合，我们将采用一下做法。

先确定一个常数B。

我们使用长度为B的数组来维护每小一部分，数组间采用链表链接。

这样你就得到了块状链表。

~~是不是很简单?~~

我们来考虑怎么用这个东西来支持插入，删除和定位。

我们来考虑怎么用这个东西来支持插入，删除和定位。

首先我们考虑定位。

我们来考虑怎么用这个东西来支持插入，删除和定位。

首先我们考虑定位。

我们从前向后一块块扫，每次观察查询点是否在块内。一旦在块内就从这个块的数组内直接查询。

我们来考虑怎么用这个东西来支持插入，删除和定位。

首先我们考虑定位。

我们从前向后一块块扫，每次观察查询点是否在块内。一旦在块内就从这个块的数组内直接查询。

这个操作明显是$O(\frac{n}{B})$的。

然后考虑插入和删除。

然后考虑插入和删除。

我们可以先定位，然后直接暴力将数字删除/加入，将后面的元素暴力移动。

然后考虑插入和删除。

我们可以先定位，然后直接暴力将数字删除/加入，将后面的元素暴力移动。

这部分复杂度为$O(\frac{n}{B}+B)$。

然后考虑插入和删除。

我们可以先定位，然后直接暴力将数字删除/加入，将后面的元素暴力移动。

这部分复杂度为$O(\frac{n}{B}+B)$。

可以发现，操作会改变链表的结构，因此一旦出现大小超过$2B$的块，我们就将它裂开。出现大小小于$\frac{B}{2}$的块，我们就将这个块和后面一块合并为一个块。

然后考虑插入和删除。

我们可以先定位，然后直接暴力将数字删除/加入，将后面的元素暴力移动。

这部分复杂度为$O(\frac{n}{B}+B)$。

可以发现，操作会改变链表的结构，因此一旦出现大小超过$2B$的块，我们就将它裂开。出现大小小于$\frac{B}{2}$的块，我们就将这个块和后面一块合并为一个块。

对以上操作进行分析，可以知道，当$B=\sqrt{n}$时，复杂度达到下界，为$O(\sqrt{n})$。

[NOI2003]Editor

要求维护一个编辑器，支持以下操作:

1.将光标向后移动k位。特别的，当$k = 0$时，移动至文本开头。

2.在光标处插入长度为n的串S。光标位置不变。

3.删除光标后n个字母。光标位置不变。

4.输出光标后n个字母。光标位置不变。

5.光标前移1位。

6.光标后移1位。

操作1$\leq 50,000$，操作2+操作3$\leq 4,000$，操作5+操作6$\leq 200,000$。插入串长综合不超过$2Mb$。

Example

直接按照之前讲的模拟就可以了。

直接按照之前讲的模拟就可以了。

时间复杂度为$O(n\sqrt{n})$。你也可以用平衡树做到$O(n\log n)$。

树套树

有的时候，我们看到某道题，发现他要用的结构很简单，但是需要额外支持一些其他结构的操作。

有的时候，我们看到某道题，发现他要用的结构很简单，但是需要额外支持一些其他结构的操作。

有的时候，这意味着树套树。

树套树基本就是在某些结构(如线段树，平衡树，主席树，树状数组等)内，再使用其他结构作为底层维护细节信息。

树套树基本就是在某些结构(如线段树，平衡树，主席树，树状数组等)内，再使用其他结构作为底层维护细节信息。

因为是结构之间套起来，所以代码量一般很大。

树套树基本就是在某些结构(如线段树，平衡树，主席树，树状数组等)内，再使用其他结构作为底层维护细节信息。

因为是结构之间套起来，所以代码量一般很大。

但是其实并不是所有的树套树代码量都很大。

树套树基本就是在某些结构(如线段树，平衡树，主席树，树状数组等)内，再使用其他结构作为底层维护细节信息。

因为是结构之间套起来，所以代码量一般很大。

但是其实并不是所有的树套树代码量都很大。

比如树链剖分其实就是一个树套树套树套树结构。

因为树套树之间区别很大，没什么关系，因此在此只简单介绍几种常用的树套树。

因为树套树之间区别很大，没什么关系，因此在此只简单介绍几种常用的树套树。

注意，在此默认你会所有基础数据结构。不会可以问别人(反正别问我)。

树链剖分 and LCT

树链剖分 and LCT

这两个东西用的很多，相信大家都会。

线段树套线段树

线段树套线段树

廖哥教的二维线段树就是这个。不会的可以问廖哥。

线段树套平衡树

线段树套平衡树

洛谷上的模板题(P3380)就是这个。

线段树套平衡树

洛谷上的模板题(P3380)就是这个。

多用来处理无插入强制在线区间查询另一属性问题。

线段树套平衡树

洛谷上的模板题(P3380)就是这个。

多用来处理无插入强制在线区间查询另一属性问题。

非强制在线就没必要写这个了。

线段树套平衡树

洛谷上的模板题(P3380)就是这个。

多用来处理无插入强制在线区间查询另一属性问题。

非强制在线就没必要写这个了。

注意，如果对常数没有什么要求的话，平衡树可以用set或者__gnu_pbds::tree替代。

平衡树套线段树/平衡树

如果上一道题需要支持插入的话，那么就需要使用平衡树套线段树或者平衡树套平衡树了。

平衡树套线段树/平衡树

如果上一道题需要支持插入的话，那么就需要使用平衡树套线段树或者平衡树套平衡树了。

注意，底层操作通常很麻烦，旋转的代价一般非常大。做表层的平衡树一般采用替罪羊树实现。

平衡树套线段树/平衡树

如果上一道题需要支持插入的话，那么就需要使用平衡树套线段树或者平衡树套平衡树了。

注意，底层操作通常很麻烦，旋转的代价一般非常大。做表层的平衡树一般采用替罪羊树实现。

替罪羊不会的问oyiya大佬。

树状数组套可持久化线段树

树状数组套可持久化线段树

后面那个俗称主席树，这个俗称待修改主席树。

树状数组套可持久化线段树

后面那个俗称主席树，这个俗称待修改主席树。

它可以支持无插入待修改区间查询另一属性。

树状数组套可持久化线段树

后面那个俗称主席树，这个俗称待修改主席树。

它可以支持无插入待修改区间查询另一属性。

这个用的很多，可以算是一种基础数据结构(吧)。

平衡树套kd-tree/点分树

平衡树套kd-tree/点分树

当我们需要对一种结构改变代价很大的结构进行改变时，我们可以使用替罪羊树套这种结构来实现。

平衡树套kd-tree/点分树

当我们需要对一种结构改变代价很大的结构进行改变时，我们可以使用替罪羊树套这种结构来实现。

不会的问oyiya大佬。

树链剖分/BST/LCT套动态树形dp

树链剖分/BST/LCT套动态树形dp

当树形dp需要带上修改时，我们可以采用维护矩阵的方法对其进行动态计算。

[HNOI2015]开店

给你一棵n个点树，每个点都有一个权值。

有m次询问，每次询问给你一个点，问这个点到所有权值在L到R之间的点的距离和为多少。强制在线。

$n \leq 1.5 \times 10^5, m \leq 2 \times 10^5, T = 3s$

你可以用动态点分治来实现这道题，但是这和讲课内容无关。

你可以用动态点分治来实现这道题，但是这和讲课内容无关。

先将答案差分，变成查询权值在$1 \to r$的点到询问点的距离和。

你可以用动态点分治来实现这道题，但是这和讲课内容无关。

先将答案差分，变成查询权值在$1 \to r$的点到询问点的距离和。

设询问点为u，则答案表示为:

你可以用动态点分治来实现这道题，但是这和讲课内容无关。

先将答案差分，变成查询权值在$1 \to r$的点到询问点的距离和。

设询问点为u，则答案表示为:

$$\sum_{i=1}^{r} dep_u + dep_i - 2 * dep_{lca} = r * dep_u + \sum_{i=1}^{r} dep_i - 2 * dep_{lca}$$

可以发现前面的部分很好求，但是lca的深度和比较麻烦。

可以发现前面的部分很好求，但是lca的深度和比较麻烦。

这时我们可以将点按照权值排序，然后依次插入。

可以发现前面的部分很好求，但是lca的深度和比较麻烦。

这时我们可以将点按照权值排序，然后依次插入。

可以发现插入后会给每个节点造成它自己深度的贡献，用树链剖分套线段树的区间操作即可。

可以发现前面的部分很好求，但是lca的深度和比较麻烦。

这时我们可以将点按照权值排序，然后依次插入。

可以发现插入后会给每个节点造成它自己深度的贡献，用树链剖分套线段树的区间操作即可。

因为我们需要询问在一段区间内的答案，因此需要将线段树可持久化。此时我们换用待修主席树维护。利用标记永久化后跑得很快。

可以发现前面的部分很好求，但是lca的深度和比较麻烦。

这时我们可以将点按照权值排序，然后依次插入。

可以发现插入后会给每个节点造成它自己深度的贡献，用树链剖分套线段树的区间操作即可。

因为我们需要询问在一段区间内的答案，因此需要将线段树可持久化。此时我们换用待修主席树维护。利用标记永久化后跑得很快。

时间复杂度$O(n \log^2 n)$。

分块

有的时候，有些问题没有什么时间复杂度很优秀的做法。

有的时候，有些问题没有什么时间复杂度很优秀的做法。

但是，这种问题有的时候有一些暴力算法，复杂度可能比较高。

有的时候，有些问题没有什么时间复杂度很优秀的做法。

但是，这种问题有的时候有一些暴力算法，复杂度可能比较高。

这种问题可能启发你将暴力的复杂度均摊，将数据分批暴力预处理或者分批计算。

有的时候，有些问题没有什么时间复杂度很优秀的做法。

但是，这种问题有的时候有一些暴力算法，复杂度可能比较高。

这种问题可能启发你将暴力的复杂度均摊，将数据分批暴力预处理或者分批计算。

这种算法我们称为分块。

分块属于套路，比较简单，我们从例题入手。

[SDOI2009]HH的项链

给你一个序列，多次询问区间不同数字个数。

$n, m \leq 5 \times 10^5$。

这道题属于分块基本应用:统计区间相异元素信息。

这道题属于分块基本应用:统计区间相异元素信息。

我们可以将序列每B个分为一段，一共分成$\frac{n}{B}$段。

这道题属于分块基本应用:统计区间相异元素信息。

我们可以将序列每B个分为一段，一共分成$\frac{n}{B}$段。

然后，我们将询问更改，变成问区间所有上次出现在左端点以前的数字个数。

这道题属于分块基本应用:统计区间相异元素信息。

我们可以将序列每B个分为一段，一共分成$\frac{n}{B}$段。

然后，我们将询问更改，变成问区间所有上次出现在左端点以前的数字个数。

我们分完块后，把每个块按照上次出现位置排序。查询的时候直接在里面二分即可。边角料暴力处理。

这道题属于分块基本应用:统计区间相异元素信息。

我们可以将序列每B个分为一段，一共分成$\frac{n}{B}$段。

然后，我们将询问更改，变成问区间所有上次出现在左端点以前的数字个数。

我们分完块后，把每个块按照上次出现位置排序。查询的时候直接在里面二分即可。边角料暴力处理。

时间复杂度$O[n\log n+m(\frac{n}{B}\log B+B)]$。当$B$取$\sqrt{n\log n}$时取得最优复杂度$O(n\log n+m\sqrt{n\log n})$。

[HNOI2010]弹飞绵羊

维护一个序列，要求支持以下操作:

1.修改一个位置的权值

2.给你一个点。从这个点出发，每次位置向后移这个点的权值步。问最早什么时候到达序列以外。

$n \leq 2 \times 10^5, m \leq 10^5$

这道题可以用LCT做，详情请问oyiya大佬。

这道题可以用LCT做，详情请问oyiya大佬。

LCT明显太难写了，但是我们发现数据范围很小，可以支持$m\sqrt{n}$的算法。

这道题可以用LCT做，详情请问oyiya大佬。

LCT明显太难写了，但是我们发现数据范围很小，可以支持$m\sqrt{n}$的算法。

因此，我们将数据分块，每个位置储存它弹出这个块的最小步数。

这道题可以用LCT做，详情请问oyiya大佬。

LCT明显太难写了，但是我们发现数据范围很小，可以支持$m\sqrt{n}$的算法。

因此，我们将数据分块，每个位置储存它弹出这个块的最小步数。

每次暴力修改即可。时间复杂度$O((n+m)\sqrt{n})$。

[Ynoi2019模拟赛]Yuno loves sqrt technology III

给你一个序列，m次询问区间众数出现次数。强制在线。

$n, m \leq 5 \times 10^5$

我们可以先预处理所有数字出现位置，然后将数据分块并预处理块内答案和块间答案。

我们可以先预处理所有数字出现位置，然后将数据分块并预处理块内答案和块间答案。

然后，我们变更答案的计算方法。假设当前答案为ans，那么只要存在一个数字，它向后第ans次出现的位置仍然在区间内，则答案可以更新为$ans+1$。

我们可以先预处理所有数字出现位置，然后将数据分块并预处理块内答案和块间答案。

然后，我们变更答案的计算方法。假设当前答案为ans，那么只要存在一个数字，它向后第ans次出现的位置仍然在区间内，则答案可以更新为$ans+1$。

因此，我们先将答案赋值为区间最大块的答案，然后向两边扩展扫边角料，顺便更新答案。

时间复杂度$O(n\sqrt{n})$。

如果你对分块算法很感兴趣的话，可以去关注一下[YNOI]。

如果你对分块算法很感兴趣的话，可以去关注一下[YNOI]。

你也可以看看这篇博客，来了解分块的几个强大操作:

http://olddrivertree.blog.uoj.ac/blog/4656

扫描线

扫描线分为几何扫描线和数点式扫描线。

扫描线分为几何扫描线和数点式扫描线。

几何的可以问ShichengXiao。

扫描线分为几何扫描线和数点式扫描线。

几何的可以问ShichengXiao。

数点式扫描线用于数点问题，可以通过离线的方式来使复杂度下降一个log。

因为过于套路，这里直接用例题来说明。

LOJ6276 果树

给你一棵树，每个点有一个颜色。求不经过重复颜色的路径条数。保证每种颜色的点数量不超过20个。

$n \leq 10^5$

发现每种颜色的点数量不超过20个，我们可以考虑通过枚举点对来计算贡献。

发现每种颜色的点数量不超过20个，我们可以考虑通过枚举点对来计算贡献。

可以发现，通过枚举点对的路径的dfn会构成1个或2个矩形。

发现每种颜色的点数量不超过20个，我们可以考虑通过枚举点对来计算贡献。

可以发现，通过枚举点对的路径的dfn会构成1个或2个矩形。

因此我们考虑将它们按照1维排序，然后利用扫描线求矩形并数点即可求出不合法方案。容斥即得到合法方案数。

发现每种颜色的点数量不超过20个，我们可以考虑通过枚举点对来计算贡献。

可以发现，通过枚举点对的路径的dfn会构成1个或2个矩形。

因此我们考虑将它们按照1维排序，然后利用扫描线求矩形并数点即可求出不合法方案。容斥即得到合法方案数。

时间复杂度$O(20n\log n)$。

离线算法

有的时候，有的题目并没有快速算法来回答询问，或者快速算法难以实现，或者在限定时间内无法通过。

有的时候，有的题目并没有快速算法来回答询问，或者快速算法难以实现，或者在限定时间内无法通过。

这时，我们就应该考虑使用离线算法来解决问题。

离线算法，简单来说就是将询问或者是结构存储下来，然后再一起处理。

离线算法，简单来说就是将询问或者是结构存储下来，然后再一起处理。

这种算法往往比在线算法更加好实现，且复杂度不会更劣。

离线算法，简单来说就是将询问或者是结构存储下来，然后再一起处理。

这种算法往往比在线算法更加好实现，且复杂度不会更劣。

限于篇幅在此只介绍一小部分。

[LNOI2014]LCA

给出一个 n 个节点的有根树，有q次询问，每次询问给出l r z，求$\sum_{i=l}^{r} dep[LCA(i,z)]$

$n,q \leq 5 \times 10^4$

这道题是n久以前的开店那道题的弱化版。

这道题是n久以前的开店那道题的弱化版。

做法很简单，直接将询问离线，排序后处理询问即可。

这道题是n久以前的开店那道题的弱化版。

做法很简单，直接将询问离线，排序后处理询问即可。

时间复杂度$O(n \log n)$。

精明的爷爷

维护一个数轴，支持以下操作:

回到历史版本，并在p_i的位置上放一个数字x_i。

每次操作结束后询问当前数轴上满足$i < j$且$w_i < w_j$的数对个数。

$n \leq 5 \times 10^5$

我们考虑将问题转换，变成在该操作之前放过的数字中满足要求的数对对数。

我们考虑将问题转换，变成在该操作之前放过的数字中满足要求的数对对数。

那么考虑将操作树建出来。

我们考虑将问题转换，变成在该操作之前放过的数字中满足要求的数对对数。

那么考虑将操作树建出来。

经过简单观察可以发现，我们可以每次统计这个点插入的数字产生的贡献，然后再从上至下累和。

我们考虑将问题转换，变成在该操作之前放过的数字中满足要求的数对对数。

那么考虑将操作树建出来。

经过简单观察可以发现，我们可以每次统计这个点插入的数字产生的贡献，然后再从上至下累和。

可以发现权值贡献就是满足两维都小于两个值或者都大于两个值的点数。

我们考虑将问题转换，变成在该操作之前放过的数字中满足要求的数对对数。

那么考虑将操作树建出来。

经过简单观察可以发现，我们可以每次统计这个点插入的数字产生的贡献，然后再从上至下累和。

可以发现权值贡献就是满足两维都小于两个值或者都大于两个值的点数。

用点分治在套用二维数点即可。

我们考虑将问题转换，变成在该操作之前放过的数字中满足要求的数对对数。

那么考虑将操作树建出来。

经过简单观察可以发现，我们可以每次统计这个点插入的数字产生的贡献，然后再从上至下累和。

可以发现权值贡献就是满足两维都小于两个值或者都大于两个值的点数。

用点分治在套用二维数点即可。

时间复杂度$O(n\log^2 n)$。

分治

分治是个大内容。

分治是个大内容。

要讲的话可以讲一整天。

分治是个大内容。

要讲的话可以讲一整天。

限于篇幅就简单点了。

这里从题目开始下手。

共点圆

维护一个二维平面，支持以下操作:

1.给定一个点，插入圆心位于该点且过原点的圆。

2.询问某点是否在所有圆内部(含圆周)

$n \leq 5 \times 10^5$

我们可以转换式子，发现每个圆对于询问点的限制可以转换为一个半平面：

$$2x_0 * x + 2y_0 * y \geq x_0^2 + y_0^2$$

我们可以转换式子，发现每个圆对于询问点的限制可以转换为一个半平面：

$$2x_0 * x + 2y_0 * y \geq x_0^2 + y_0^2$$

于是，问题变为动态插入半平面，以及询问一个点是否在所有半平面内。

我们可以转换式子，发现每个圆对于询问点的限制可以转换为一个半平面：

$$2x_0 * x + 2y_0 * y \geq x_0^2 + y_0^2$$

于是，问题变为动态插入半平面，以及询问一个点是否在所有半平面内。

可以用splay直接维护半平面交，但是代码复杂度过高且常数也不小。

我们可以转换式子，发现每个圆对于询问点的限制可以转换为一个半平面：

$$2x_0 * x + 2y_0 * y \geq x_0^2 + y_0^2$$

于是，问题变为动态插入半平面，以及询问一个点是否在所有半平面内。

可以用splay直接维护半平面交，但是代码复杂度过高且常数也不小。

但是，可以发现，每个半平面对询问点的贡献是独立的。因此可以利用cdq分治来处理这个问题。

我们可以转换式子，发现每个圆对于询问点的限制可以转换为一个半平面：

$$2x_0 * x + 2y_0 * y \geq x_0^2 + y_0^2$$

于是，问题变为动态插入半平面，以及询问一个点是否在所有半平面内。

可以用splay直接维护半平面交，但是代码复杂度过高且常数也不小。

但是，可以发现，每个半平面对询问点的贡献是独立的。因此可以利用cdq分治来处理这个问题。

具体来说，就是每次处理出左半边的半平面对右半边询问点的影响，在分别递归两边。

我们可以转换式子，发现每个圆对于询问点的限制可以转换为一个半平面：

$$2x_0 * x + 2y_0 * y \geq x_0^2 + y_0^2$$

于是，问题变为动态插入半平面，以及询问一个点是否在所有半平面内。

可以用splay直接维护半平面交，但是代码复杂度过高且常数也不小。

但是，可以发现，每个半平面对询问点的贡献是独立的。因此可以利用cdq分治来处理这个问题。

具体来说，就是每次处理出左半边的半平面对右半边询问点的影响，在分别递归两边。

时间复杂度$O(n \log^2 n)$。

Luogu P4169[Violet]天使玩偶/SJY摆棋子

需要维护一个结构，要求支持以下2个操作。

1.向二位空间内插入一个点。

2.给出一个二维点，询问距离这个点曼哈顿距离最近的点的曼哈顿距离。

$n, m \leq 3 \times 10^5, T = 3s$

之前介绍过这道题的kd-tree做法，现在介绍分治做法。

之前介绍过这道题的kd-tree做法，现在介绍分治做法。

我们可以将这个距离拆分成为4种不同情况:查询点在询问点的四个象限中。

之前介绍过这道题的kd-tree做法，现在介绍分治做法。

我们可以将这个距离拆分成为4种不同情况:查询点在询问点的四个象限中。

我们可以发现，这时候可以去掉绝对值。

之前介绍过这道题的kd-tree做法，现在介绍分治做法。

我们可以将这个距离拆分成为4种不同情况:查询点在询问点的四个象限中。

我们可以发现，这时候可以去掉绝对值。

然后，题目转为了一个简单的三维偏序问题。

之前介绍过这道题的kd-tree做法，现在介绍分治做法。

我们可以将这个距离拆分成为4种不同情况:查询点在询问点的四个象限中。

我们可以发现，这时候可以去掉绝对值。

然后，题目转为了一个简单的三维偏序问题。

用cdq解决这个偏序即可。

时间复杂度$O(n \log^2 n)$。

[BZOJ4836]二元运算

定义二元运算$\oplus$满足:

$$x \oplus y = \begin{cases} x + y & (x < y) \\ x - y & (otherwise) \end{cases}$$

给定两个序列a和b，q次询问每次给出一个数c，问有多少数对(i, j)满足$a_i \oplus b_j = c$。

$len_a, len_b, q \leq 5 \times 10^4$

首先，我们将a和b使用桶记录每种数字出现次数，设为A和B。

首先，我们将a和b使用桶记录每种数字出现次数，设为A和B。

然后，我们考虑计算出 $w_z = \sum_{i=1}^{n} \sum_{j=1}^{m} [i \oplus j == z] A_i * B_j$ 。

首先，我们将a和b使用桶记录每种数字出现次数，设为A和B。

然后，我们考虑计算出 $w_z = \sum_{i=1}^{n} \sum_{j=1}^{m} [i \oplus j == z] A_i * B_j$ 。

因为答案与数字相对大小有关，我们考虑使用cdq分治来计算点对贡献。

首先，我们将a和b使用桶记录每种数字出现次数，设为A和B。

然后，我们考虑计算出 $w_z = \sum_{i=1}^{n} \sum_{j=1}^{m} [i \oplus j == z] A_i * B_j$ 。

因为答案与数字相对大小有关，我们考虑使用cdq分治来计算点对贡献。

以$a_i < b_j$部分为例。

首先，我们将a和b使用桶记录每种数字出现次数，设为A和B。

然后，我们考虑计算出 $w_z = \sum_{i=1}^{n} \sum_{j=1}^{m} [i \oplus j == z] A_i * B_j$ 。

因为答案与数字相对大小有关，我们考虑使用cdq分治来计算点对贡献。

以$a_i < b_j$部分为例。

我们每次将序列分为2部分，每次计算出左边的序列对右边序列的影响。

首先，我们将a和b使用桶记录每种数字出现次数，设为A和B。

然后，我们考虑计算出 $w_z = \sum_{i=1}^{n} \sum_{j=1}^{m} [i \oplus j == z] A_i * B_j$ 。

因为答案与数字相对大小有关，我们考虑使用cdq分治来计算点对贡献。

以$a_i < b_j$部分为例。

我们每次将序列分为2部分，每次计算出左边的序列对右边序列的影响。

可以发现，答案为$\sum_{i=l}^{mid} \sum_{j=mid+1}^{r} [i + j == z] A_i * B_j$。

首先，我们将a和b使用桶记录每种数字出现次数，设为A和B。

然后，我们考虑计算出 $w_z = \sum_{i=1}^{n} \sum_{j=1}^{m} [i \oplus j == z] A_i * B_j$ 。

因为答案与数字相对大小有关，我们考虑使用cdq分治来计算点对贡献。

以$a_i < b_j$部分为例。

我们每次将序列分为2部分，每次计算出左边的序列对右边序列的影响。

可以发现，答案为$\sum_{i=l}^{mid} \sum_{j=mid+1}^{r} [i + j == z] A_i * B_j$。

这明显是卷积形式，利用FFT优化即可。

首先，我们将a和b使用桶记录每种数字出现次数，设为A和B。

然后，我们考虑计算出 $w_z = \sum_{i=1}^{n} \sum_{j=1}^{m} [i \oplus j == z] A_i * B_j$ 。

因为答案与数字相对大小有关，我们考虑使用cdq分治来计算点对贡献。

以$a_i < b_j$部分为例。

我们每次将序列分为2部分，每次计算出左边的序列对右边序列的影响。

可以发现，答案为$\sum_{i=l}^{mid} \sum_{j=mid+1}^{r} [i + j == z] A_i * B_j$。

这明显是卷积形式，利用FFT优化即可。

时间复杂度$O(n \log^2 n)$。

[51nod 1376]最长递增子序列的数量

统计最长上升子序列数量。

$n \leq 5 \times 10^4$

这道题当然能够用树状数组做了。

这道题当然能够用树状数组做了。

但是可以注意，这道题也可以用cdq分治做。

这道题当然能够用树状数组做了。

但是可以注意，这道题也可以用cdq分治做。

能够发现，能够转移到i的点，它的两维都会比i小。

这道题当然能够用树状数组做了。

但是可以注意，这道题也可以用cdq分治做。

能够发现，能够转移到i的点，它的两维都会比i小。

因此利用cdq分治解决二位偏序即可。

时间复杂度$O(n \log n)$。

[SDOI2016]模式字符串

给你一棵树，树上每个点都有一个大写字母。在给你一个模板串S。求树上有多少条路径组成的串可以用S重复整数次表示出来。

数据组数 $c \leq 10, n, |S| \leq 10^6, T = 2s$

数据强度原因可以通过 $O(10n \log n)$ 。

因为是树上路径问题，因此可以考虑点分治。

因为是树上路径问题，因此可以考虑点分治。

考虑枚举重心，并算出过重心的路径的答案。

因为是树上路径问题，因此可以考虑点分治。

考虑枚举重心，并算出过重心的路径的答案。

显然可以记录每个点作为前缀匹配到第i位的数量和作为后缀匹配到第i位的数量。这个利用哈希可以做到$O(size)$。

因为是树上路径问题，因此可以考虑点分治。

考虑枚举重心，并算出过重心的路径的答案。

显然可以记录每个点作为前缀匹配到第i位的数量和作为后缀匹配到第i位的数量。这个利用哈希可以做到$O(size)$。

然后合并答案统计即可。时间复杂度$O(cn \log n)$。

疫情控制问题

给出一棵 n 个节点的有根树,每个节点都有点权 a_i 。对于每个点,你可以选择花费 a_i 的代价将 i 的子树中所有叶子节点选中。 现在有 m 个操作,每个操作为以下两者中的一种:

1.修改某个点的 a_i

2.输入 i ,询问如果要将 i 子树中所有叶子节点都选中,最少的花费

$n, m \leq 10^6$

属于简单动态dp问题，可以采用链分治的方法解决。

属于简单动态dp问题，可以采用链分治的方法解决。

列出dp后，构造出矩阵即可快速转移。利用BST可以优化复杂度。

属于简单动态dp问题，可以采用链分治的方法解决。

列出dp后，构造出矩阵即可快速转移。利用BST可以优化复杂度。

时间复杂度$O(n \log n)$。

这次讲课讲了一些大家都会的算法。

这次讲课讲了一些大家都会的算法。

这里可以给出几道题，然后帮助你更好地理解这些题。

kd-tree:[APIO2018]Circle selection 选圆圈

块状链表:[BZOJ3337]ORZJRY I

树套树:[TJOI2017]不勤劳的图书管理员,[luogu4278]带插入区间K小值

分块:YNOI基本上所有的题

扫描线:HNOI的好几道题

离线算法:大部分的工业题

分治:[luogu4319]变化的道路,[WC2014]紫荆花之恋,[SDOI2017]切树游戏

Thanks