数据结构基础篇

oyiya

雅礼中学

December 22, 2018

Fenwick Tree 的基础用法

Fenwick Tree 的基础用法

- 单点修改区间查询

Fenwick Tree 的基础用法

- 单点修改区间查询
- 区间修改单点查询

Fenwick Tree 的基础用法

- 单点修改区间查询
- 区间修改单点查询
- 区间修改区间查询

Fenwick Tree 的基础用法

- 单点修改区间查询
- 区间修改单点查询
- 区间修改区间查询

区修区查：

Fenwick Tree 的基础用法

- 单点修改区间查询
- 区间修改单点查询
- 区间修改区间查询

区修区查：

求原序列 a_i 做差分数组 d_i，那么 $a_i = \sum_{j=1}^{i} d_j$

Fenwick Tree 的基础用法

- 单点修改区间查询
- 区间修改单点查询
- 区间修改区间查询

区修区查：

求原序列 a_i 做差分数组 d_i，那么 $a_i = \sum_{j=1}^{i} d_j$

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} \sum_{j=1}^{i} d_j = \sum_{i=1}^{n} (n-i+1) d_i = (n+1) \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} i d_i$$

Fenwick Tree 的基础用法

- 单点修改区间查询
- 区间修改单点查询
- 区间修改区间查询

区修区查：

求原序列 a_i 做差分数组 d_i，那么 $a_i = \sum_{j=1}^{i} d_j$

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} \sum_{j=1}^{i} d_j = \sum_{i=1}^{n} (n-i+1) d_i = (n+1) \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} i d_i$$

$$\sum_{i=l}^{r} f(i) = \sum_{i=1}^{r} f(i) - \sum_{i=1}^{l-1} f(i)$$

Fenwick Tree 的黑科技

Fenwick Tree 的黑科技

- 对于给定维数的BIT，写个dfs

Fenwick Tree 的黑科技

- 对于给定维数的BIT，写个dfs
- 下标从零开始，进来的时候下标加 1 .....

Fenwick Tree 的黑科技

- 对于给定维数的BIT，写个dfs
- 下标从零开始，进来的时候下标加 1 .....
- 要用BIT记后缀和的话，翻转BIT就好了

Fenwick Tree 的黑科技

- 对于给定维数的BIT，写个dfs
- 下标从零开始，进来的时候下标加 1 .....
- 要用BIT记后缀和的话，翻转BIT就好了
- BIT清零也可以用时间戳

Fenwick Tree 的黑科技

- 对于给定维数的BIT，写个dfs
- 下标从零开始，进来的时候下标加 1 .....
- 要用BIT记后缀和的话，翻转BIT就好了
- BIT清零也可以用时间戳
- 接下来要讲的BIT上二分

Fenwick Tree 的黑科技

- 对于给定维数的BIT，写个dfs
- 下标从零开始，进来的时候下标加 1 .....
- 要用BIT记后缀和的话，翻转BIT就好了
- BIT清零也可以用时间戳
- 接下来要讲的BIT上二分
- 接下来会提到的标记永久化BIT

Fenwick Tree 的黑科技

- 对于给定维数的BIT，写个dfs
- 下标从零开始，进来的时候下标加 1 .....
- 要用BIT记后缀和的话，翻转BIT就好了
- BIT清零也可以用时间戳
- 接下来要讲的BIT上二分
- 接下来会提到的标记永久化BIT
- ......

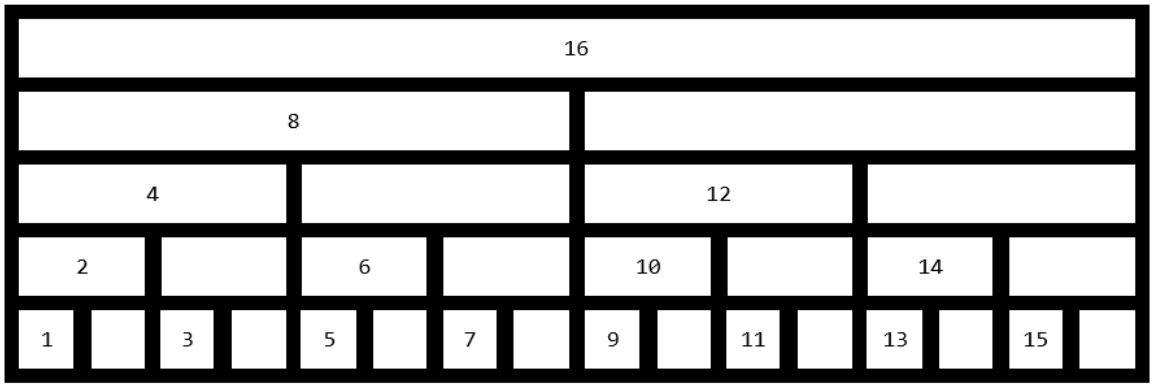
Fenwick Tree 上二分

Fenwick Tree 上二分

首先，可以把BIT看做没有右儿子的线段树，如下图

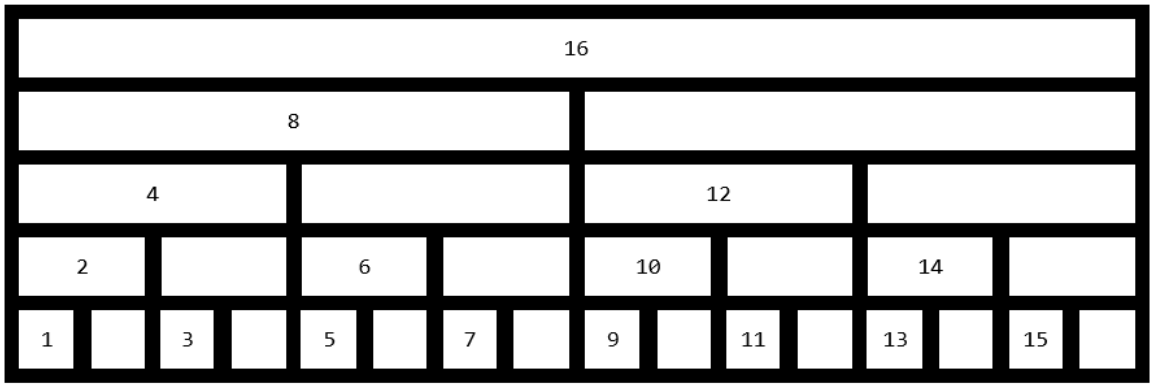
Fenwick Tree 上二分

首先，可以把BIT看做没有右儿子的线段树，如下图



Fenwick Tree 上二分

首先，可以把BIT看做没有右儿子的线段树，如下图



由于没有右儿子的话不方便，所以强行把一个区间右斜下方的区间当做自己的右儿子。例如上图中 8 的右儿子就是 12

于是我们可以知道：

于是我们可以知道：

对于 p 代表的区间，它的左儿子是 $p-(lowbit(p)>>1)$，右儿子是 $p+(lowbit(p)>>1)$

于是我们可以知道：

对于 p 代表的区间，它的左儿子是 $p-(lowbit(p)>>1)$，右儿子是 $p+(lowbit(p)>>1)$

于是类似线段树二分，如果条件大于左儿子贡献，走右儿子；反之亦然

于是我们可以知道：

对于 p 代表的区间，它的左儿子是 $p-(lowbit(p)>>1)$，右儿子是 $p+(lowbit(p)>>1)$

于是类似线段树二分，如果条件大于左儿子贡献，走右儿子；反之亦然

不同的是，树状数组二分的当前位置等价于线段树二分的左儿子位置，所以要比较的贡献即为自己的权值

于是我们可以知道：

对于 p 代表的区间，它的左儿子是 $p-(lowbit(p)>>1)$，右儿子是 $p+(lowbit(p)>>1)$

于是类似线段树二分，如果条件大于左儿子贡献，走右儿子；反之亦然

不同的是，树状数组二分的当前位置等价于线段树二分的左儿子位置，所以要比较的贡献即为自己的权值

边界条件是走到叶子，即 p 为奇数

如果BIT的值域 R 为 2 的次幂，那么初始节点直接为 R

如果BIT的值域 R 为 2 的次幂，那么初始节点直接为 R

否则，初始节点为小于等于 R 的最大的 2 的次幂，即 $1 << ilogb(R)$

如果BIT的值域 R 为 2 的次幂，那么初始节点直接为 R

否则，初始节点为小于等于 R 的最大的 2 的次幂，即 $1 << ilogb(R)$

此时要把空间开到 $\frac{3}{2}2^{ilogb(R)}$

Fenwick Tree 标记永久化

Fenwick Tree 标记永久化

类似之前的过程，再进行标记永久化，就可以用BIT代替简单线段树

Fenwick Tree 标记永久化

类似之前的过程，再进行标记永久化，就可以用BIT代替简单线段树

它可以用来维护支持标记永久化的可减性信息的，写法跟标记永久化线段树一样

Fenwick Tree 标记永久化

类似之前的过程，再进行标记永久化，就可以用BIT代替简单线段树

它可以用来维护支持标记永久化的可减性信息的，写法跟标记永久化线段树一样

区间 $[l,r]$ 的加操作要转化成 $[1,l-1]$ 的减操作和 $[1,r]$ 的加操作

Fenwick Tree 标记永久化

类似之前的过程，再进行标记永久化，就可以用BIT代替简单线段树

它可以用来维护支持标记永久化的可减性信息的，写法跟标记永久化线段树一样

区间 $[l,r]$ 的加操作要转化成 $[1,l-1]$ 的减操作和 $[1,r]$ 的加操作

但它的局限性很大，所以碰见这种问题还是写线段树更稳

Segment Tree 合并

Segment Tree 合并

前置技能是动态开点

Segment Tree 合并

前置技能是动态开点

同时遍历两颗线段树，每次在相同的区间进行讨论

Segment Tree 合并

前置技能是动态开点

同时遍历两颗线段树，每次在相同的区间进行讨论

如果只有一棵线段树有当前区间的节点，直接返回

Segment Tree 合并

前置技能是动态开点

同时遍历两颗线段树，每次在相同的区间进行讨论

如果只有一棵线段树有当前区间的节点，直接返回

否则，递归合并子区间，然后合并当前区间信息

Segment Tree 合并

前置技能是动态开点

同时遍历两颗线段树，每次在相同的区间进行讨论

如果只有一棵线段树有当前区间的节点，直接返回

否则，递归合并子区间，然后合并当前区间信息

复杂度 log

Segment Tree 分裂

Segment Tree 分裂

分裂与二分很像，对当前区间分情况讨论

Segment Tree 分裂

分裂与二分很像，对当前区间分情况讨论

如果要分的部分在右子树，那么递归右子树

Segment Tree 分裂

分裂与二分很像，对当前区间分情况讨论

如果要分的部分在右子树，那么递归右子树

否则交换两棵树当前区间的右儿子（因为全部都属于分出来的树），递归左儿子

```
1  //split t1 to t1&t2 so that s[t1]=k
2  void split(int t1,int &t2,int k)
3  {
4      clear(t2);
5      int ls=s[ch[t1][0]];
6      if(k>ls) split(ch[t1][1],ch[t2][1],k-ls);
7      else swap(ch[t1][1],ch[t2][1]);
8      if(k<ls) split(ch[t1][0],ch[t2][0],k);
9      s[t2]=s[t1]-k; s[t1]=k;
10 }
```

TJOI2016/HEOI2016 排序

TJOI2016/HEOI2016 排序

给出一个 1 到 n 的全排列，现在对这个全排列序列进行 m 次局部排序，排序分为两种：

1:(0,l,r)表示将区间[l,r]的数字升序排序

2:(1,l,r)表示将区间[l,r]的数字降序排序

最后询问第 q 位置上的数字。

TJOI2016/HEOI2016 排序

给出一个 1 到 n 的全排列，现在对这个全排列序列进行 m 次局部排序，排序分为两种：

1:(0,l,r)表示将区间[l,r]的数字升序排序

2:(1,l,r)表示将区间[l,r]的数字降序排序

最后询问第 q 位置上的数字。

$1 \leq q \leq n, 1 \leq n \leq 10^5, 1 \leq m \leq 10^5$

将排序好的一段有序区间当做一个集合

将排序好的一段有序区间当做一个集合

用 set 维护这些集合，每次权值线段树合并加分裂完成排序过程

Segment Tree 分治

Segment Tree 分治

线段树分治主要处理一类可以离线的问题

Segment Tree 分治

线段树分治主要处理一类可以离线的问题

一般有两种操作，查询和修改

Segment Tree 分治

线段树分治主要处理一类可以离线的问题

一般有两种操作，查询和修改

按操作的时间建一棵线段树，即时间线段树

Segment Tree 分治

线段树分治主要处理一类可以离线的问题

一般有两种操作，查询和修改

按操作的时间建一棵线段树，即时间线段树

考虑每个修改操作所影响的区间，然后在线段树上打上标记，每一个操作最多打 log 个标记

Segment Tree 分治

线段树分治主要处理一类可以离线的问题

一般有两种操作，查询和修改

按操作的时间建一棵线段树，即时间线段树

考虑每个修改操作所影响的区间，然后在线段树上打上标记，每一个操作最多打 log 个标记

最后求答案，遍历线段树，进入区间完成区间上标记的操作，离开时撤销操作

Segment Tree 分治

线段树分治主要处理一类可以离线的问题

一般有两种操作，查询和修改

按操作的时间建一棵线段树，即时间线段树

考虑每个修改操作所影响的区间，然后在线段树上打上标记，每一个操作最多打 log 个标记

最后求答案，遍历线段树，进入区间完成区间上标记的操作，离开时撤销操作

到达叶子时，满足询问要求的状态，记下答案就好了

LUOGU 4319 变化的道路

LUOGU 4319 变化的道路

H 国道路都有一个值 w，如果第一次通过这条道路，那么 L 值会减少 w

H 国有 N 个国家，最开始 $N-1$ 条道路，刚好构成一棵树

小 w 将从城市 1 出发，游览所有城市，总共游览 32766 天，对于每一天，他希望游览结束后 L 还是一个正数，那么出发时 L 值至少为多少

H 国的所有边都是无向边，没有一条道路连接相同的一个城市

LUOGU 4319 变化的道路

H 国道路都有一个值 w，如果第一次通过这条道路，那么 L 值会减少 w

H 国有 N 个国家，最开始 $N-1$ 条道路，刚好构成一棵树

小 w 将从城市 1 出发，游览所有城市，总共游览 32766 天，对于每一天，他希望游览结束后 L 还是一个正数，那么出发时 L 值至少为多少

H 国的所有边都是无向边，没有一条道路连接相同的一个城市

$1 \leq N \leq 50000, 1 \leq l \leq r \leq 32766, 1 \leq w \leq 10^9$

时间线段树分治裸题

时间线段树分治裸题

对于当前的一天，如果知道图，那么要求的就是MST边权和

时间线段树分治裸题

对于当前的一天，如果知道图，那么要求的就是MST边权和

用一个LCT维护就好了

可持久化 Segment Tree

可持久化 Segment Tree

可以询问历史版本的线段树

可持久化 Segment Tree

可以询问历史版本的线段树

正常来说要建 n 棵线段树，于是考虑很多无用节点合并起来

可持久化 Segment Tree

可以询问历史版本的线段树

正常来说要建 n 棵线段树，于是考虑很多无用节点合并起来

每个版本基于上个版本建立，查询用前缀和思路

可持久化 Segment Tree

可以询问历史版本的线段树

正常来说要建 n 棵线段树，于是考虑很多无用节点合并起来

每个版本基于上个版本建立，查询用前缀和思路

主席树就是一种常见的可持Segment Tree

ZKW Segment Tree

ZKW Segment Tree

ZKW属于非递归线段树

ZKW Segment Tree

ZKW属于非递归线段树

它的作用就是卡常卡空间缩码量，但局限性特别大，只能完成一些简单操作

ZKW Segment Tree

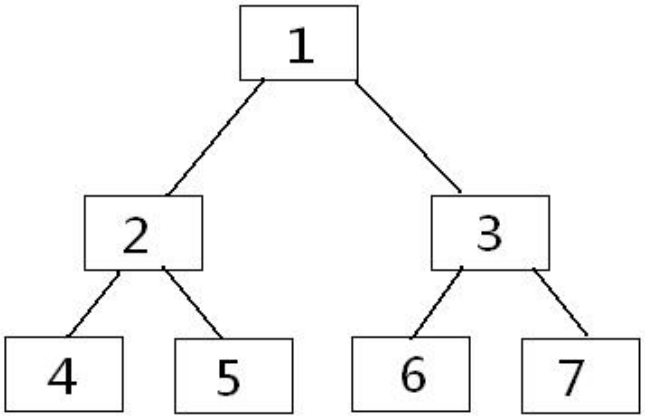
ZKW属于非递归线段树

它的作用就是卡常卡空间缩码量，但局限性特别大，只能完成一些简单操作

所以它还是很少用的

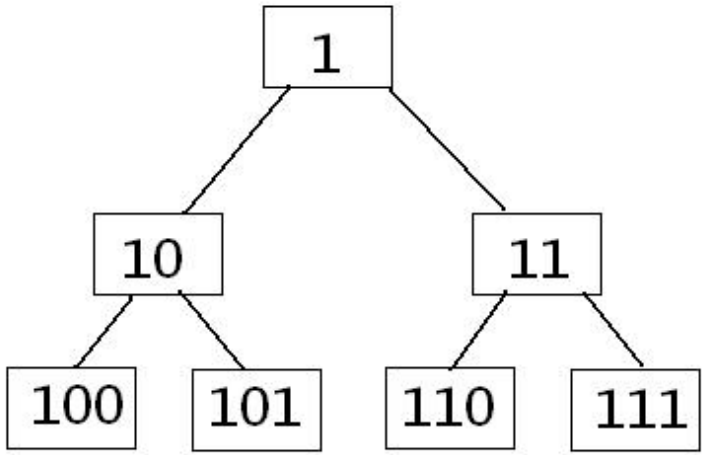
将一棵线段树堆式储存，变成这样

将一棵线段树堆式储存，变成这样

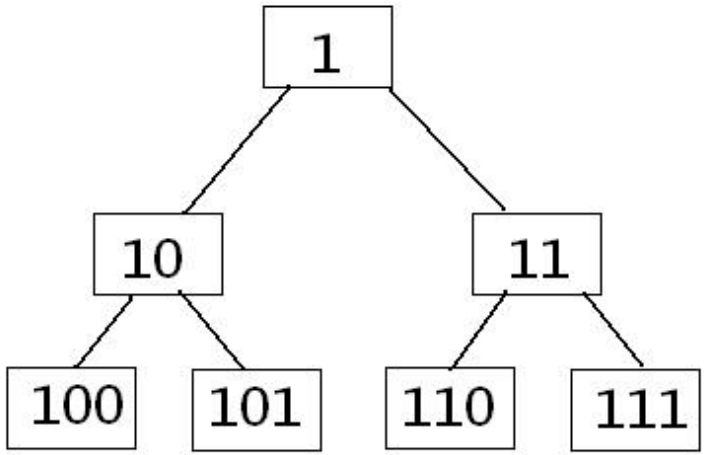


将这些数字变成二进制

将这些数字变成二进制

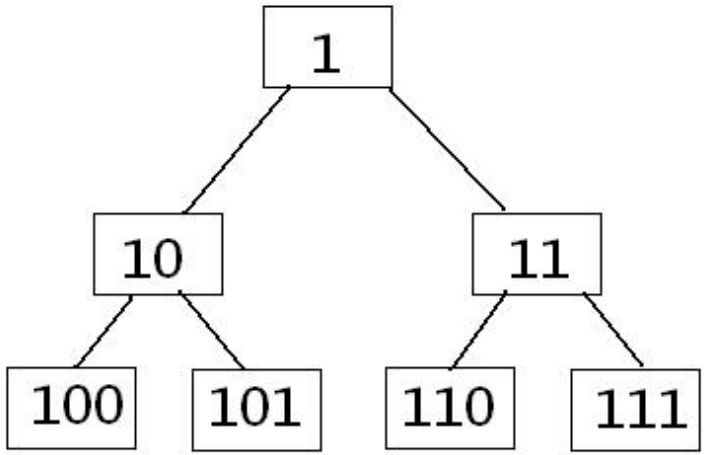


将这些数字变成二进制



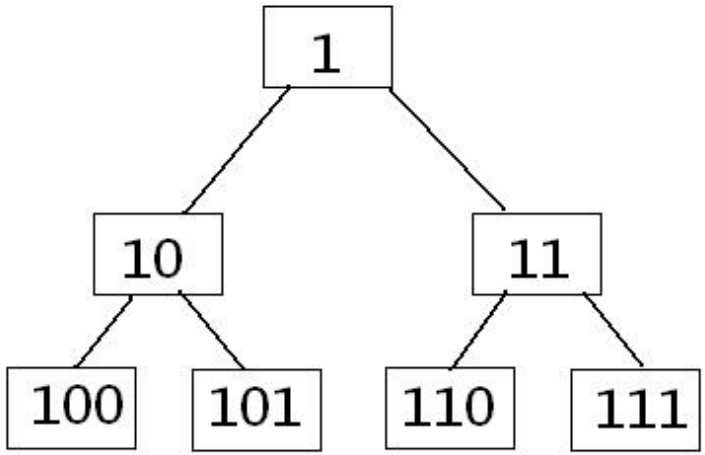
- 一个节点的父节点是这个数右移 1

将这些数字变成二进制



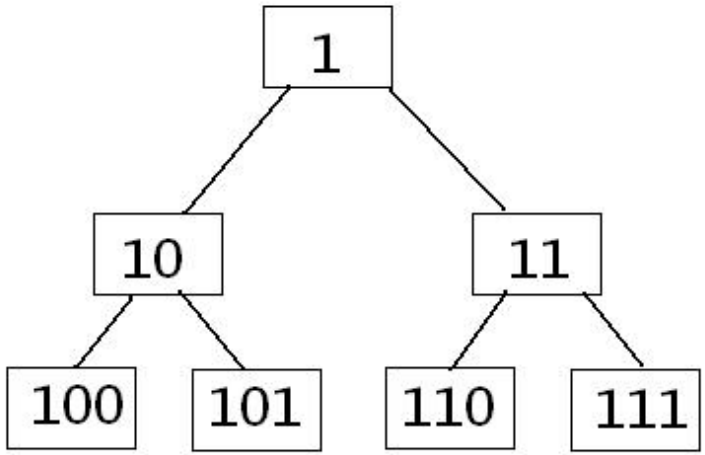
- 一个节点的父节点是这个数右移 1
- 一个节点 p 的左节点是 $p << 1$，右节点是 $p << 1|1$

将这些数字变成二进制



- 一个节点的父节点是这个数右移 1
- 一个节点 p 的左节点是 $p << 1$，右节点是 $p << 1|1$
- 同一层的节点是依次递增的，第 n 层有 $2^{(n-1)}$ 个节点

将这些数字变成二进制



- 一个节点的父节点是这个数右移 1
- 一个节点 p 的左节点是 $p << 1$，右节点是 $p << 1|1$
- 同一层的节点是依次递增的，第 n 层有 $2^{(n-1)}$ 个节点
- 最后一层有多少节点，值域就是多少

这样我们可以 $O(1)$ 定位一个点在线段树中的位置

这样我们可以 $O(1)$ 定位一个点在线段树中的位置

那么建树的时候可以直接输入，然后直接访问父亲，更新父亲就好了

这样我们可以 $O(1)$ 定位一个点在线段树中的位置

那么建树的时候可以直接输入，然后直接访问父亲，更新父亲就好了

然而在一些情况下，ZKW线段树节点上记的值并不是自己的值，而是与父亲的差分

这样我们可以 $O(1)$ 定位一个点在线段树中的位置

那么建树的时候可以直接输入，然后直接访问父亲，更新父亲就好了

然而在一些情况下，ZKW线段树节点上记的值并不是自己的值，而是与父亲的差分

通过这种方式来实现lazy标记的传递

具体实现过程为找区间贡献的时候，从两个边界节点往上走，累加贡献

具体实现过程为找区间贡献的时候，从两个边界节点往上走，累加贡献

当两个指针为兄弟的时候，就只需要从它们的父亲一路向上累计到根就可以了

具体实现过程为找区间贡献的时候，从两个边界节点往上走，累加贡献

当两个指针为兄弟的时候，就只需要从它们的父亲一路向上累计到根就可以了

这样的方法来做区间查询和区间修改

具体实现过程为找区间贡献的时候，从两个边界节点往上走，累加贡献

当两个指针为兄弟的时候，就只需要从它们的父亲一路向上累计到根就可以了

这样的方法来做区间查询和区间修改

注意区间操作要把闭区间变成开区间来算

具体实现过程为找区间贡献的时候，从两个边界节点往上走，累加贡献

当两个指针为兄弟的时候，就只需要从它们的父亲一路向上累计到根就可以了

这样的方法来做区间查询和区间修改

注意区间操作要把闭区间变成开区间来算

代码可以参见这里

Treap

Treap

普通平衡树

Treap

普通平衡树

基于BST，每个节点的 key 满足中序遍历有序

Treap

普通平衡树

基于BST，每个节点的 key 满足中序遍历有序

基于堆，每个节点的 ext 满足儿子的重量大于父亲的重量

Treap

普通平衡树

基于BST，每个节点的 key 满足中序遍历有序

基于堆，每个节点的 ext 满足儿子的重量大于父亲的重量

加入节点，通过BST的性质找到要插入的位置

Treap

普通平衡树

基于BST，每个节点的 key 满足中序遍历有序

基于堆，每个节点的 ext 满足儿子的重量大于父亲的重量

加入节点，通过BST的性质找到要插入的位置

插入后，如果不满足堆的性质，进行旋转

可持久化 Treap

可持久化 Treap

俗称fhq，通过非旋转维护，支持询问历史版本还有区间操作

可持久化 Treap

俗称fhq，通过非旋转维护，支持询问历史版本还有区间操作

它的所有操作不是基于旋转，而是基于 $merge$ 和 $split$

可持久化 Treap

俗称fhq，通过非旋转维护，支持询问历史版本还有区间操作

它的所有操作不是基于旋转，而是基于 $merge$ 和 $split$

$merge(a, b)$ 代表把以 a 和以 b 为根的平衡树和在一起

可持久化 Treap

俗称fhq，通过非旋转维护，支持询问历史版本还有区间操作

它的所有操作不是基于旋转，而是基于 $merge$ 和 $split$

$merge(a, b)$ 代表把以 a 和以 b 为根的平衡树和在一起

$split(a, k)$ 代表把以 a 为根的平衡树拆成前 k 项和后 $size - k$ 项

要把一个数插入到第 k 个位置，把 1 到 $k-1$ 和 $k+1$ 和 n $split$ 开，然后把三棵树 $merge$ 起来

要把一个数插入到第 k 个位置，把 1 到 $k-1$ 和 $k+1$ 和 n $split$ 开，然后把三棵树 $merge$ 起来

要把第 k 个数删除，类似加入，$split$ 成三棵树，然后 $merge$ 其中两棵

要把一个数插入到第 k 个位置，把 1 到 $k-1$ 和 $k+1$ 和 n $split$ 开，然后把三棵树 $merge$ 起来

要把第 k 个数删除，类似加入，$split$ 成三棵树，然后 $merge$ 其中两棵

然后因为可以支持拆树和合树，所以区间操作都可以完成

要把一个数插入到第 k 个位置，把 1 到 $k-1$ 和 $k+1$ 和 n $split$ 开，然后把三棵树 $merge$ 起来

要把第 k 个数删除，类似加入，$split$ 成三棵树，然后 $merge$ 其中两棵

然后因为可以支持拆树和合树，所以区间操作都可以完成

把修改区间拆出来，打标记，然后合并

要把一个数插入到第 k 个位置，把 1 到 $k-1$ 和 $k+1$ 和 n $split$ 开，然后把三棵树 $merge$ 起来

要把第 k 个数删除，类似加入，$split$ 成三棵树，然后 $merge$ 其中两棵

然后因为可以支持拆树和合树，所以区间操作都可以完成

把修改区间拆出来，打标记，然后合并

翻转、平移、求和、修改值等都可以做

由于fhq的节点不需要维护父亲节点的信息，所以可以支持可持

由于fhq的节点不需要维护父亲节点的信息，所以可以支持可持

这个类似于可持线段树做就好了

由于fhq的节点不需要维护父亲节点的信息，所以可以支持可持

这个类似于可持线段树做就好了

代码可参见这里

由于fhq的节点不需要维护父亲节点的信息，所以可以支持可持

这个类似于可持线段树做就好了

代码可参见这里

之后可以去BZOJ1500练板子

CF702F T-Shirts

CF702F T-Shirts

有 n 种T恤，每种有一个价格 c_i 和品质 q_i。有 m 个人要买T恤，第 i 个人有 v_i 块钱，每个人每次都买一件能买得起的 q_i 最大的T恤。

一个人只能买一种T恤一件，所有人互不影响。

求最后每个人买了多少件T恤。

CF702F T-Shirts

有 n 种T恤，每种有一个价格 c_i 和品质 q_i。有 m 个人要买T恤，第 i 个人有 v_i 块钱，每个人每次都买一件能买得起的 q_i 最大的T恤。

一个人只能买一种T恤一件，所有人互不影响。

求最后每个人买了多少件T恤。

$1 \leq n, m \leq 2e5$

先将T恤按品质从大到小排序

先将T恤按品质从大到小排序

用fhq维护所有人的钱的集合

先将T恤按品质从大到小排序

用fhq维护所有人的钱的集合

枚举到一个T恤，把所持现金大于当前T恤的人的集合区间减价格

先将T恤按品质从大到小排序

用fhq维护所有人的钱的集合

枚举到一个T恤，把所持现金大于当前T恤的人的集合区间减价格

但减完之后不能直接 $merge$ ，因为会有人的钱小于另一个集合

先将T恤按品质从大到小排序

用fhq维护所有人的钱的集合

枚举到一个T恤，把所持现金大于当前T恤的人的集合区间减价格

但减完之后不能直接 $merge$ ，因为会有人的钱小于另一个集合

但是考虑到这样的情况，每个人最多不超过 log 次，所以我们直接对这些人暴力合并就好了

先将T恤按品质从大到小排序

用fhq维护所有人的钱的集合

枚举到一个T恤，把所持现金大于当前T恤的人的集合区间减价格

但减完之后不能直接 $merge$ ，因为会有人的钱小于另一个集合

但是考虑到这样的情况，每个人最多不超过 log 次，所以我们直接对这些人暴力合并就好了

最后复杂度 log^2

Splay

文艺平衡树

Splay

文艺平衡树

带双旋功能，但是更好写。它的旋转不由任何值决定，而是完成一个操作后，把当前所在的点旋转到根

Splay

文艺平衡树

带双旋功能，但是更好写。它的旋转不由任何值决定，而是完成一个操作后，把当前所在的点旋转到根

它的复杂度均摊是 log 的

Splay

文艺平衡树

带双旋功能，但是更好写。它的旋转不由任何值决定，而是完成一个操作后，把当前所在的点旋转到根

它的复杂度均摊是 log 的

最重要的是，它是LCT的基础

splay自己本身就可以维护区间操作

splay自己本身就可以维护区间操作

这是因为它的splay可以把一个点旋到指定一个点的下方

splay自己本身就可以维护区间操作

这是因为它的splay可以把一个点旋到指定一个点的下方

这个时候满足BST性质的就不是权值，而是位置的下标

splay自己本身就可以维护区间操作

这是因为它的splay可以把一个点旋到指定一个点的下方

这个时候满足BST性质的就不是权值，而是位置的下标

操作时，把这段区间旋转好，然后打标记之后pushdown就好了

Scapegoat Tree

替罪羊树，似乎没用过?

Scapegoat Tree

替罪羊树，似乎没用过?

一种平衡树，如果不满足平衡条件，那么重构

Scapegoat Tree

替罪羊树，似乎没用过?

一种平衡树，如果不满足平衡条件，那么重构

平衡条件为左子树大小 $< \alpha \times$ 根大小 & 右子树大小 $< \alpha \times$ 根大小

如果某个时刻存在不平衡情况，那么把所有的点，把树拉平成链表

如果某个时刻存在不平衡情况，那么把所有的点，把树拉平成链表

然后递归取链的终点作为根，构造平衡树

如果某个时刻存在不平衡情况，那么把所有的点，把树拉平成链表

然后递归取链的终点作为根，构造平衡树

这样构造出来是一棵几乎完美的二叉树

如果某个时刻存在不平衡情况，那么把所有的点，把树拉平成链表

然后递归取链的终点作为根，构造平衡树

这样构造出来是一棵几乎完美的二叉树

代码参考这里

Link Cut Tree

Link Cut Tree

没有什么好说的了

Link Cut Tree

没有什么好说的了

各种知识已经在上次讲过了

Link Cut Tree

没有什么好说的了

各种知识已经在上次讲过了

子树问题的关键是虚子树信息的维护

Link Cut Tree

没有什么好说的了

各种知识已经在上次讲过了

子树问题的关键是虚子树信息的维护

动态图问题关键是以操作结束时间为权值做MST

树链剖分

树链剖分

树链剖分，常见的就是重链剖分和长链剖分吧

树链剖分

树链剖分，常见的就是重链剖分和长链剖分吧

这个东西能很好地解决一些树上问题

树链剖分

树链剖分，常见的就是重链剖分和长链剖分吧

这个东西能很好地解决一些树上问题

重链剖分以 $size$ 为关键字分配重儿子，访问任意一条链的复杂度 log

树链剖分

树链剖分，常见的就是重链剖分和长链剖分吧

这个东西能很好地解决一些树上问题

重链剖分以 $size$ 为关键字分配重儿子，访问任意一条链的复杂度 log

长链剖分以 dep 为关键字分配重儿子，$O(1)$ 回答 k 级祖先，$O(n)$ 合并与深度有关的子树信息

树链剖分

树链剖分，常见的就是重链剖分和长链剖分吧

这个东西能很好地解决一些树上问题

重链剖分以 $size$ 为关键字分配重儿子，访问任意一条链的复杂度 log

长链剖分以 dep 为关键字分配重儿子，$O(1)$ 回答 k 级祖先，$O(n)$ 合并与深度有关的子树信息

请zy来讲一下毛毛虫剖分（如果他觉得我们足够强可以听懂的话）

启发式合并

启发式合并

数据结构中的启发式合并就是在两个数据结构合并时，把小的数据结构中的元素暴力加进大的数据结构

启发式合并

数据结构中的启发式合并就是在两个数据结构合并时，把小的数据结构中的元素暴力加进大的数据结构

复杂度均摊 log

Thanks