数据结构入门

CF 377D Developing Game

- 给n个人，每个人有三个属性li,vi,ri，
- 选择最多的人的集合，满足任意x,y有vx∈[ly,ry]。

- 合法集合必定存在L∈[max{li},min{vi}]，R∈[max{vi},min{ri}]，考虑(L,R)的坐标，那么每个人就是一个矩形，求最多矩形覆盖的点扫描线即可。

CF 475F Meta-universe

- 无穷大平面被分成Z*Z的格子，现在某些格子里有星球，你可以选取一行或者一列，满足：
- 1.这一行/列是空的
- 2.这一行/列的两边都不是空的
- 然后就可以把一个星球的集合分成两个。
- 现在给你一个大小为n的集合，问你最后能分成多少个不能再分的集合。
- •n<=100000;|xi|,|yi|<=109.

- 考虑按照x或者y排序然后暴力枚举在哪里切割。
- 这样最坏情况下复杂度是O(n^2)的。
- 怎么办呢？
- 我们从x两边同时往中间扫，对于y也是一样。扫到了就把少的那边分出来，然后两边递归。
- 这样复杂度是什么呢？
- 每次的扫描复杂度和分出来的较小的集合是同阶的，这样一个集合每被扫一次，它所在的集合大小至少会翻倍，这样扫描的复杂度就是O(logn)了。
- 使用set分别维护x,y两个顺序，复杂度O(nlog^2n)。

CF 464E

- 一个n个点m条边的带权无向图，求s到t的最短路。输入中u,v,x表示<u,v>间有一条长度为2^x的边，保证无重边和自环。
- 要求输出最短路长度对10^9+7取模的值并输出一条最短路方案。
- n,m,ui,vi,xi<=105.
- 5s,768M.

- 考虑要支持哪些操作：
- 1.快速赋值
- 2.比较两个数
- 找到最高的不同位
- 3.给一个数加上 2^x
- A.找到第一个不小于x且为0的位p
- B.把[x,p-1]赋为0
- C.把第p位赋为1

- 都是可以用可持久化线段树完成的操作

CF 480E

- 给一个n*m的01矩阵，有k次操作，每次把(x,y)修改为1，并求当前矩阵最大全0子正方形。
- n,m,k<=2000.
- 3s,256M.

- 求最大全0子方阵是O(n2)的，但显然不能兹瓷修改。
- 容易支持修改的是分治结构，考虑建立分治结构。
- 假设n>m，我们可以取出第floor((n+1)/2)行，并维护和这一行有交的所有正方形，两边继续分治，当n<m时把矩阵转置一下就能保证规模每次折半了。
- 一行怎么维护呢，我们记录每个元素上/下方向的第一个1的距离，并设两个指针扫描，这样只要用单调队列维护区间最大值即可。
- 单次修改复杂度是T(n)=T(n/2)+O(n)=O(n)
- 总复杂度就是O(nk)。

序列上的数据结构

- 线段树
- 平衡树
- 各种可持久化
- 吉老师线段树

CF 643 G

- 有一个序列a[1…n]，支持区间赋值，然后求区间里面出现超过了p%的数字个数。
- 这里对于询问，可以输出100/p个数字，并且我们只要求把对的输出即可，可以输出多余的不对的结果。
- n<=1.5e5, p>=20。

- 考虑绝对众数，我们存在一个O(n)的做法。
- 这个东西可以推广到出现比例为p%的情况。

HDU 6087

- 给你一个序列，支持区间求和。
- for (i=l..r) a[i]=a[i-k]操作。
- for (i=l..r) a[i]=b[i]操作。
- n<=1e5, q<=1e5

- 裸可持久化平衡树

CF 453E

- 有n个🐴，每个🐴一开始的能量值为si，每个时刻恢复能量的速度为vi，然后有个能量上限mi。
- 现在按时间一次执行操作a[l...r]询问l...r里面的🐴现在的能量和，同时将这些🐴的能量清空。
- n, q <= 1e5

- 首先考虑每次都是清空[1...n]的🐴，那么可以计算出来每个🐴恢复能量所需要的时间。排序之后，一定是后缀的🐴能量都满了，然后前面的🐴能量没满，所以可以分别计算。
- 对于原问题，我们用一个set记录一下，哪些段的🐴同时清空。然后修改的时候暴力for一下每个段，一个段可以用主席树来计算前面的值。
- 时间复杂度O(n log n + q log n)。

CF 1172F

- 给出a[1…n],p，求
- sum(l,r)的值。
- n<=1e6, m<=2e5

Function — a fake way to calculate sum modulo p

```
1:  function ModAdd(x, y, p)
2:      if x + y < p then
3:          return x + y
4:      else
5:          return x + y − p
6:
7:  function Sum(A, l, r, p)
8:      result ← 0
9:      for i ← l to r do
10:         result ← ModAdd(result, A[i], p)
11:     return result
```

- 线段树，对于每个区间求出一个分段线性函数，也就是说左端点从x进去，右端点是f(x)。
- 考虑两个并一起就是两个分段线性函数复合。
- 首先，可以发现x+sum-f(x)其实就是减了多少次p。
- 不难发现这个分段函数最多分成l+1段

- 所以可以对每个区间求出分段函数，合并两个分段函数的时候，就是一个简单的two pointer，可以在线性的时间内算出来。
- 最后查询的时候，就是在O(log n)个区间里面分别二分，就做完了。

北京赛区 2018 E

- 维护一个数组a[0...2^k-1]。
- 支持以下操作：
- 对于l<=i<=r令a[i]=a[i^x]
- 对于l<=i<=r令a[i^x]=a[i]
- 对于l<=i<=r令a[i]=a[i]^x
- 询问l<=i<=r中二进制中1为奇数个的下标的数字的异或和。

- 不难发现xor出来的二进制数奇偶性只和xor的两个数有关。
- 是个区间复制操作。
- 可持久化Trie

CF 896 E

- 给你一个序列，支持：
- [l,r]里面严格大于x的数字减去x。
- 查询[l,r]里有多少=x的数字。
- n,m<=10^5, a[i]<=10^5

- 分块，然后使用链表存f[i][x]表示i里面权值为x的数字，然后记录一下每个的最大值。
- 对于操作1，如果值域>=2x，那么把[1…x]的数字往上合并，否则把[x+1…max]往下合并。
- 这样可以在O(a)的时间复杂度内使得值域-a。
- 对于询问 顺便维护一下链表长度就可以了。

牛客4 F

- 有一个a[1..n]的排列。
- 支持操作
- merge(a[1..m],a[m+1…r])
- 询问a[i]

```
def merge(a,b):
  c=[]
  while a.size() and b.size():
    if a[0]<b[0]:
      c.append(a[0])
      a.pop_front()
    else:
      c.append(b[0])
      b.pop_front()
  return c+a+b
```

- 考虑这个merge，就是先把序列分成一段一段之后然后merge。
- 所以我们用平衡树维护这些开头。
- 注意使用merge的时候我们会把m+1和r+1对应的区间给切开来，这个时候需要重建出这些块，插回到原来的序列中。

- 为啥这样直接模拟的复杂度是对的？
- 考虑一下块头连续上升的段数。这样合并显然复杂度是O(这个区间的段数的)。
- 而这样模拟之后这个区间的块头就会变成连续上升的一段。
- 然后每次切开对这玩意的影响也是O(1)的
- 所以均摊是O(n)的

CF 997 E

- 有一个排列p[1…n]，一个线段是好的当且仅当它是连续段。
- q个询问，给出[l,r]里面，问多少子区间是连续段。
- n<=10^5

- 注意到若区间$[l,r]$满足$max-min=r-l$，则这个区间是好的区间。
- 于是在当前指针r下维护一个$(max-min)-(r-l)$的最小值以及最小值个数。
- 然后维护一个答案sum，代表某个区间最小值的贡献（注意这时候不需要维护是不是0，我们只需要在最后累计答案的时候在树根判断就可以了，这个题根一定为0，所以没有判）
- 但是发现这样不行，我们只是维护了一个右端点为r的区间的贡献，于是维护一个dev贡献数组，维护历史出现的最小值出现的贡献。
- 具体点说，dev也相当于一个tag，表示这个区间当前的答案需要被贡献到sum，不管它之后怎么样。
- 然后区间max,min的相关东西用单调栈额外维护，复杂度是均摊的log n

CF 1148 H

- 有一个序列，要求在线支持在末尾插入一个数。
- 询问l,r之间有多少子区间，满足mex(a[x],a[x+1], … ,a[y])=k
- n<=1e5

- 首先考虑离线怎么做。
- 对于询问[l,r]我们记录一下右端点为r的时候，左端点每个数字最后一一次出现的位置pos。
- 那么mex(a[l],a[l+1],…,a[r])>=k当且仅l<=min(pos[0],pos[1],…,pos[k-1])

- 所以我们需要维护前缀最小值，找到一个时刻x使得

min(pos[0…k])>=x，然后答案就是x时刻到r时刻的min(pos[0…k])-l+1的总和。

- 换句话来说我们需要支持前缀最小值然后求和。

- 如果是离线的话，如果我们逆序操作，那么每次实际上是将一个数字改小，这样会覆盖后面一些连续的段，所以前缀最小值的段的变化量是O(n)的。
- 改成正序操作之后，变化量仍然是O(n)的，但是每次将一个数改大，需要在线段树里面找出接下来的前缀位置。
- 总的时间复杂度O((n+q) log ^2 n)

CF 1034 D

- 有n个区间si=[ai,bi]，令区间[l,r]的价值为s[l…r],这些区间的并的长度。
- 求前k大区间的价值和。
- n<=3e5, k<=1e9

- 首先我们可以二分答案k，然后怎么统计答案呢？
- 我们使用双指针[l,r]使得区间里面的数字大于等于k。
- 这个时候我们记一下每个位置最后被染的时刻，那么[l,r]区间并就是所有>=l的位置之和。
- 同时我们计算总和也可以用这种方法，但这样的时间复杂度是O(n log n)的。

- 我们可以在一开始就预处理出区间的变化情况，这样每次二分就可以在O(1)时间复杂度内解决。
- 所以整个算法的时间复杂度是O(n log n + n log T)。

CF 1083 D

- 有一个数列a[1…n]。
- 求四元组a,b,c,d的个数 (a<b<c<d)。
- 满足[b,c)中元素两两不同，[b,c)中的元素在[a,b), [c,d)中没有出现过。
- n<=10^5

- 对于[b,c)，记pre[i]表示a[i]出现左边一个的位置，nxt[i]表示a[i]出现右边一个的位置。
- 那么答案就是(b-max(pre))*(min(nxt)-c)，用单调栈+线段树维护。

动态树(分治)

常用的树分治方法

- 1.点分治
- 2.链分治
- 3.边分治
- 4.link cut tree
- 5.top tree

Notice

- 目前我见过的类树分治的东西有：
- 点分治
- 边分治
- 链分治 --- 树链剖分 --- 树上启发式合并
- lct维护子树
- 静态lct维护子树
- 类top tree
- Top tree
- 静态top tree

Note

- 其实会点分治/链分治就够用了
- Top tree类的东西现在基本上没人会的，所以不用管
- 事实是top tree是目前我觉得的最通用化的树分治处理数据结构

Explanation

- 大概说一下原理吧
- 比如区间操作问题，这个东西如果线段树来维护就是每次访问的区间严格O(logn)个
- 如果用平衡树维护连续段来维护，且每次访问的区间可以被缩点，那每次访问的区间会变成均摊O(1)个
- 这个对于树同样适用

Explanation

- 树链剖分由于轻-重链切换O(logn)次（或者说树的启发式合并每次访问O(logn)个点），每次访问的区间不连续导致复杂度O(log^2n)
- 而LCT由于链修改时具有缩点的性质，在树上拓展序列上的缩点可以证明缩点个数均摊O(logn)个
- 然后由于splay自身具有Finger search的性质，按照Dynamic Finger Theorem可以证明LCT的复杂度为均摊O(logn)

Explanation

- 这里可以看出LCT的维护方式本质和平衡树缩点维护序列的方式类似，或者说是平衡树缩点维护序列算法的一个拓展，因为树link的过程满足缩点性质，这两个可以高效结合
- LCT可以经过进一步拓展实现LCT维护子树，再进一步拓展成为Top tree，由于Top tree具有维护子树动态DP信息的性质，所以实质上Top tree可以看做是动态化的树链剖分，即动态化的链分治

点分治

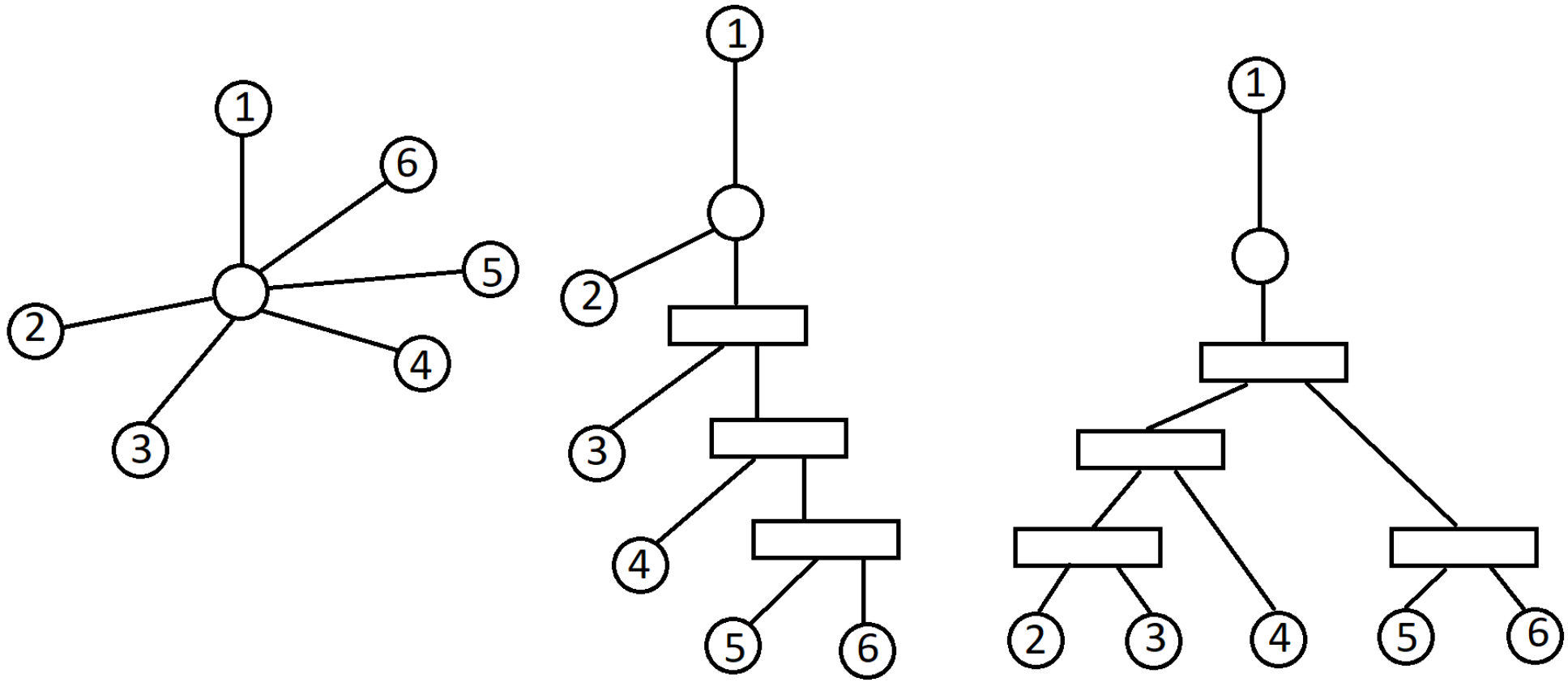
- 可以看作是每次删去一个点之后继续分治下去
- 每次找树的一个重心，然后递归每个子树
- 这些找到的重心构成一个树形结构
- 因为这个分治中心构成的树的深度O(logn)
- 所以还是在每个分治中心维护一个数据结构
- 修改和查询都暴力跳就可以了

边分治

- 可以看作是每次删去一条边之后继续分治下去
- 删去一条边之后树会分成两个子树，删去的边需要保证让两边的子树size尽可能相近
- 直接朴素地在这个树上做可能复杂度不正确，需要进行三度化

三度化

- 对于每个度数>3的节点，可以通过加虚点的方法让该节点度数变为3



三度化

- 注意三度化只是可以解决部分树点度数过大的问题，因为其本质并不是对点度数进行了分治，而仅仅是改变了树的结构

链分治

- 基于HLD的分治结构
- 可以看作是每次删去一条重链之后继续分治下去
- 每个重链头开一个数据结构
- 每次修改的时候跳重链，在每个重链头的数据结构上进行修改
- 查询同理

Luogu2056 [ZJOI2007]捉迷藏

- 树，点权0，1
- 每次修改一个点点权，询问两个最远1的距离

Solution

- 对于传统树分治做法，就是在每个树分治的分治中心上开个堆维护一下：
- 每个子树一个堆维护到其的最大距离
- 再开个堆维护上面那个堆的前2大值
- 再开个堆维护全局的最大值
- 然后每次暴力更新即可，总复杂度O((n + m)log^2n)

Solution

- 还有一种括号序列的做法
- 本质上是ETT在这个题上的特殊化，这里就不介绍了

Bzoj2566 xmastree

- 有一棵含N个结点的树，树上每条边(ai,bi)都有一个权值wi。树上每个结点涂有一个初始颜色ci。现在有很多次修改操作，第i次修改会将结点xi的颜色修改成yi。请在所有修改前和每次修改之后输出一个数，表示对应时刻最近的同色结点对的距离。
- 其中，距离定义为树上两点的最短路的距离。最短路按边的权值wi计算。

Solution

- 和之前那个题差不多，对每个颜色分别维护即可

Bzoj3730 震波&Hdu4918 Query on the subtree

- 树，点权
- 1.修改一个点的点权
- 2.查询距离一个点<=k的所有点的点权和

Solution

- 裸题
- 开一个数据结构，维护离分治中心距离为1…的所有点的点权和
- 为了防止算重，可以再开一个数据结构，维护每个点到分治中心的父亲的负贡献即可差分掉

Bzoj4372 烁烁的游戏

- 树，点权
- 1.修改距离一个点<=k的所有点的点权和
- 2.查询一个点的点权

Solution

- 和前一题有啥区别

[ZJOI2015]幻想乡战略游戏

- 带修改，维护带权重心

[Ynoi2012]D1T3

- 音无彩名给了你一棵n个节点的树，每个节点有一种颜色，有m次查询操作
- 查询操作给定参数l r x，需输出：
- 将树中编号在[l,r]内的所有节点保留，x所在联通块中颜色种类数
- 每次查询操作独立

Solution

- 每次查询的时候
- 考虑在点分树上找到深度最浅的一个点，满足这个点被保留区间点x所在连通块包含
- 然后我们可以把根固定为这个点来进行统计

Solution

- 转换为
- 给定一棵有根树
- 每次查询保留区间点，根所在连通块的颜色数

Solution

- 对每个颜色分别考虑贡献
- 一个点在保留区间颜色的时候和根连通等价于这个点到根路径上[min,max]的区间是区间[l,r]的子区间
- 问题就转化为了，平面上有若干个点，询问某个点右下角的点的颜色数
- 这个可以离线O(logn)解决，前面点分治是O(logn)代价

- 总复杂度O(nlog^2n)

floj307 不可知圆环

Description

纱绫发现，『不可知圆环』不仅可以进行传送，如果在传送中关闭，还可以将物品切断。

纱绫决定试验一下这个能力。她找到了一个 n 个节点的树，对于每条边都决定切断或不切断，这样一共有 2^{n-1} 种方案。纱绫想知道，有多少种方案使得点 1 所在连通块的大小为 k。答案对 1811939329 取模。

Solution

- 处理树的问题通常可以考虑分治，而点分治在这道题中会有一些不便，考虑进行链分治。
- 先对树进行重链剖分，每次处理一条重链。
- 先递归下去求出每个轻儿子的答案数组。对于一个重链上的点，合并其轻儿子可以用分治 FFT 做到 O(nlog^2n)

Solution

- 这样每个重链上的点都有一个答案数组。
- 考虑合并重链上的点。如果按深度从小到大对重链编号，则连通块在重链上的部分一定是一段前缀，可以对重链进行分治 FFT。一个前缀要么是左半的一部分，要么是左半加右半的一部分，所以分治时可以维护两个数组，一个是包含左端点的答案，一个是同时包含左右端点的答案。
- 时间复杂度为 O(nlog^3n)，但常数很小。

Ynoi2012D2T3 ??????

- 给一棵树，支持：
- 1.把一条链上所有点加上k
- 2.查询距离一个点<=1的所有点的点权kth

Solution1

- 暴力

- 总复杂度O(nm)
- 期望得分：20

Solution2

- 对点的度数进行根号分治
- 度数>sqrt(n)的点有sqrt(n)个，其他点度数都<sqrt(n)
- 维护所有大点的数据结构
- 每次修改O(sqrtn * 数据结构复杂度)

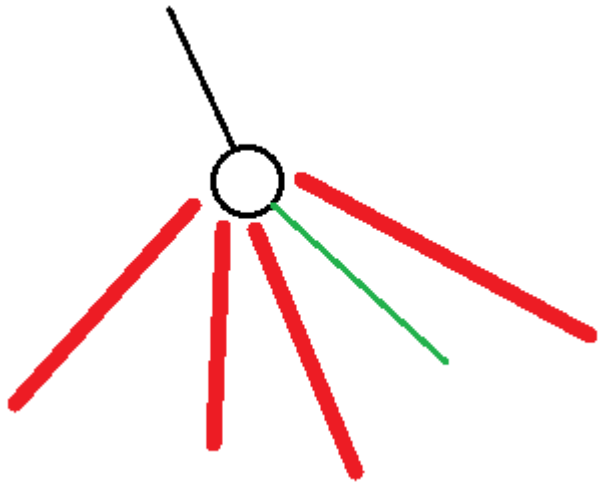
Solution2

- 查询的时候如果是大点就直接查，如果是小点就是暴力查询一圈
- 这个是O(sqrt(n) * 数据结构复杂度)
- 对两个数据结构都用一个O(sqrt(n)) – O(1)平衡的分块
- 于是做到了O(msqrt(n))的复杂度

- 期望得分：20-50

Solution3

- 考虑链分治
- 一个点距离<=1的点里面，只有3个可能不是重链头
- 自己，父亲，轻儿子



Solution3

- 然后每次查询的时候暴力插入这3个（O(1)个）点
- 每次修改的时候把一条链上所有重链头的父亲都修改这个重链头的值
- 于是做到了查询O(logn)修改O(log^2n)的复杂度
- 期望得分：80~100

UOJ#191 Unknown

有一个元素为向量的序列S，下标从1开始，初始时S为空，现在你需要支持三个操作：

1.在S的末尾添加一个元素(x, y)。

2.删除S的末尾元素。

3.询问下标在$[l, r]$区间内的元素中，$(x, y) \times S_i$的最大值。

其中×表示向量的叉积，$(x_1, y_1) \times (x_2, y_2) = x_1 y_2 - x_2 y_1$

- 答案肯定在上凸壳上
- 考虑如何维护上凸壳
- 首先建出操作树之后可以发现询问是树上从顶向下的一条链
- 对于每个询问链，用树剖把它切成log n个dfs序上的区间
- 除了最浅的那个区间之外都是一条重链的前缀
- 最浅的那个区间直接做
- 剩下的那些一个一个插入做就可以了

Link Cut Tree

- 也叫Sleator Tarjan Tree
- 原理是用一棵支持finger search的平衡树（如splay）来维护所有重链，这里的重链是动态的
- 每次访问一个点的时候会将这个点到当前根的路径放在同一个重链中
- 同时支持换根

Link Cut Tree

- 可以证明在树上这样的重链段变化均摊是O(mlogn)次，平衡树复杂度为单次split merge O(logs)，所以理应O(log^2n)单次
- 但是由于使用splay等结构可以和lct的结构很好地match起来，可以证明复杂度为O(mlogn)均摊的（然而比如leafy tree这种结构就很难和lct的结构match起来，因为天生性质冲突）

Link Cut Tree

- Link cut tree可以高效地支持链信息维护
- 如查询x到y的简单路径信息，可以先把x节点设为根节点，然后访问y节点，这样x到y的节点就在同一条重链上了，也就被同一棵平衡树所维护
- 可以发现这样的操作类似于平衡树维护区间的时候提取一段区间

动态DP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，最少的花费是多少？

数据范围 $n, m \leq 1000000$。

时间限制 1 秒

空间限制 256MB。

如果每次询问都可以单独做一次暴力 DP，那么可以记录 $f(i)$ 表示把 i 子树中所有叶子选中的最小费用，则若 i 为叶子则 $f(i) = a_i$，否则

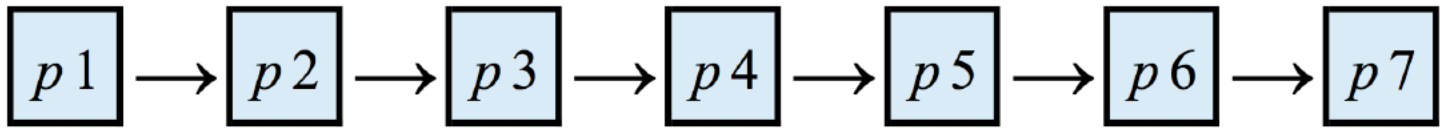
$$f(i) = \min\left(a_i, \sum_{p \in Ch(i)} f(p)\right)$$

时间复杂度为线性。

现在考虑有修改的数据。这题中的树是有根树，其 DP 值的计算有一个严格的顺序，如果用点分治难以维护计算 DP 值的顺序，更难以支持询问。

继续考虑链分治。借鉴前面的思想，考虑按照重链树的顺序计算 DP 值。由于重链树同样是有根树，按照重链树的顺序计算 DP 值只是相当于对每一个节点选择了子节点的转移顺序，并不影响转移的正确性，因此这里采用链分治是完全可行的。

要解决的第一个问题是如何在一条链上转移。设当前重链的链长为 m，我们记录 p_i 表示这条链上从上往下的第 i 个点，c_i 表示节点 i 的所有轻儿子 x 的 $f(x)$ 之和（为了方便，若 i 为叶子，则记 $c_i = \infty$）



由于是按照重链树的顺序进行 DP，同样也只需要用一个变量维护 c_i，和树上最大权独立集的算法类似。修改的时候也只需要将 c_i 加上该轻儿子的 $f(x)$ 的差值。

为了求出这条重链上每个节点的 $f(i)$，可以这样 DP：首先 $f(p_m) = a_{p_m}$，接下来对于任意 $1 \le i < m$ 都有

$$f(p_i) = \min\left(a_{p_i}, f(p_{i+1}) + c_{p_i}\right)$$

分析这个式子。可以把一个位置的 (a_{p_i}, c_{p_i}) 看做一个变换 (a, b) 满足

$$trans_{(a,b)}(x) = \min(a, x + b)$$

则对于当前重链，$f(p_i)$ 就是 $(a_{p_m}, c_{p_m}), (a_{p_{m-1}}, cp_{m-1}), ..., (a_{p_{i+1}}, c_{p_{i+1}}), (a_{p_i}, c_{p_i})$ 这些变换依次作用在 0 上的结果。而两个这样的变换合并之后，仍是原来的形式：

$$
\begin{aligned}
& trans_{(c,d)}(trans_{(a,b)}(x)) \\
= \ & trans_{(c,d)}(\min(a, b + x)) \\
= \ & \min(c, d + \min(a, b + x)) \\
= \ & \min(\min(c, a + d), b + d + x)
\end{aligned}
$$

即对一个数 x 先执行变换 (a, b) 再执行变换 (c, d) 相当于执行了变换 $(\min(c, a + d), b + d)$。

那么就可以直接用线段树来维护区间的变换和。每次询问的时候，查询该重链所在的后缀和即可，复杂度 $O(\log n)$。修改 a_p 的时候，先将 p 所在重链的线段树中 p 所在这一位进行修改，接下来求出新的 $f_{top(p)}$，然后更新 $c_{fa_{top(p)}}$ 并进行对应线段树上的修改，再继续往上更新直至根即可，复杂度 $O(\log^2 n)$。

- 实际上用LCT也可以实现
- 和链分治几乎类似
- 用LCT的Splay代替上面的那个线段树
- 但是会难写一点

一个prefinal题

- 有n个人(编号0~n-1) 每个人手上有个a_i
- 接下来会进行n轮游戏(从0开始)
  - 最开始有个数x = 0
  - 第i轮轮到第i个人时，第i个人可以选择是否把x变成(x + a_i) % n
- 游戏结束后，第x个人获胜
- 每个人会采取如下策略：只有当变了x必胜，且不变不必胜的情况下，他才会在他这轮变动x
  - 并且这是个Common Knowledge ，即所有人到知道大家会采用这个策略。

- Q次操作，每次修改一个a_i，问谁获胜？

- i朝(i-a_i)%n连边
- (i-a_i)%n >= i?毫无意义
- 形成了一棵树
- 是个开枪游戏
- 如果0活下来了就是0
- 否则是开枪杀掉0的编号最小的人
- LCT 动态DP维护这个开枪游戏

Codeforces 545 F

- 给出一棵树，每个点有一个优先级，初始为自己的标号，如果每次删去树上优先级最低的叶子，将得到一个序列，要求维护有三种操作：
  1.将一个点的优先级修改为此时所有点的优先级的最大值+1
  2.询问某个点在序列中的位置

Solution

- 点x在点y之前被删除，当且仅当它不是y的祖先，并且x的子树最大值不超过y的子树最大值
- 子树最大值相同的点形成了若干条链，如果没有修改的话可以直接维护

Solution

- 由于每次只会把一个点改成最大值+1，考虑lct维护access连续段均摊，等价于一个换根操作，并且会整体修改到原来的根上一段路径的子树最大值
- 把子树最大值相同的一段链用一棵splay维护，每次access时直接合并一堆链的splay，然后树状数组就可以了

- 时间复杂度O(nlog^2n)，空间复杂度O(n)

bzoj 4025 二分图

- 有n个点m条边，边会在start时刻出现在end时刻消失，求对于每一段时间，该图是不是一个二分图。

Solution

- 考虑离线
- 用lct维护删除时间的最大生成树。
- 加入一条边时，如果两点不连通则直接link，否则肯定有一条边多余，若形成奇环则将多余的边加入集合。
- 删除一条边时，若这条边是树边则直接删除，否则若在集合中，则从集合中删除。
- 查询时，如果集合中没有边，则为二分图。

bzoj 3514 Codechef MARCH14 GERALD07加强版

- N个点M条边的无向图，询问保留图中编号在[l,r]的边的时候图中的联通块个数，强制在线

Solution

- 先用lct预处理一下，依次加入每一条边
- 当前加入第i条边，如果这个边加入之后成环了，就弹出这个环上面编号最小的边x，并且记录下b[i] = x，默认b[i] = 0
- 之后就是查询n-区间中b值小于于l的数个数
- 因为如果b值小于l，那这个边与[l,r]内边不成环，所以连通块数量-1
- 否则没有贡献
- 时间复杂度O((n+m)logn)，空间复杂度O(nlogn)

[CodeChef Pushflow]

- N个点的图 每个点最多属于一个简单环 边有边权。
- M次操作，询问两个点之间最大流或者修改一条边的边权

- 最大流就是最小割
- 最小割要么是环上两条边，要么是一条环间的边

- 环上的两条边显然包含了最小的那条边，把它删掉并且把它的权值加到这个环的其他边上

[WineDAG’s prevention]

- N个点的树 Q次操作 增加一个点的点权 翻转一条路径 询问路径和/最大值/最小值

- 我们用一棵Splay来维护形态，再用另一棵权值Splay来维护权值。用次序来一一对应。修改就可以在权值树上直接做而不影响树的形态。
- 复杂度O(qlogn)

[ZJOI2016 大森林]

小Y家里有一个大森林，里面有 n 棵树，编号从 1 到 n 。一开始这些树都只是树苗，只有一个节点，标号为 1 。这些树都有一个特殊的节点，我们称之为生长节点，这些节点有生长出子节点的能力。

小Y掌握了一种魔法，能让第 l 棵树到第 r 棵树的生长节点长出一个子节点。同时她还能修改第 l 棵树到第 r 棵树的生长节点。

她告诉了你她使用魔法的记录，你能不能管理她家的森林，并且回答她的询问呢?

输入格式

第一行包含 2 个正整数 n,m，共有 n 棵树和 m 个操作。

接下来 m 行，每行包含若干非负整数表示一个操作，操作格式为：

1. $0\ l\ r$ 表示将第 l 棵树到第 r 棵树的生长节点下面长出一个子节点，子节点的标号为上一个 0 号操作叶子标号加 1（例如，第一个 0 号操作产生的子节点标号为 2），l 到 r 之间的树长出的节点标号都相同。保证 $1 \leq l \leq r \leq n$。
2. $1\ l\ r\ x$ 表示将第 l 棵树到第 r 棵树的生长节点改到标号为 x 的节点。对于区间内的每棵树，如果标号 x 的点不在其中，那么这个操作对该树不产生影响。保证 $1 \leq l \leq r \leq n$，x 不超过当前所有树中节点最大的标号。
3. $2\ x\ u\ v$ 询问第 x 棵树中节点 u 到节点 v 的距离，也就是在第 x 棵树中从节点 u 和节点 v 的最短路上边的数量。保证 $1 \leq x \leq n$，这棵树中节点 u 和节点 v 存在。

- 我们从左到右扫描每棵树。用set维护时间轴。每次考虑一个变化。
- 考虑建虚点。虚点的权值为0，不影响实际的答案。
- 如果新发生了更改生长节点这件事，我们新建一个权值为0的虚点，之后的生长操作都在这个虚点下面做。
- 如果取消了更改生长节点这个事，那么我们就把当前这个生长节点代表的虚点接到上一个有意义的生长节点去。
- 如果新发生了生长这件事 那么就直接在当时那个时间对应生长节点代表的虚点下面新加入一个点
- 如果取消了生长这件事 那么就直接在当时那个时间对应生长节点代表的虚点下面删掉那个点
- 询问的话就直接在LCT上查询两个点之间的点权和就可以了。

- 有三部分点A B C，各有n,n1,n2个点。
- 其中A B 构成一棵树，A C构成一棵树,A之间的点没有连边
- 随机I,j把B1~Bi,C1~Cj都删了，A仍然连通的概率
- N 1e5

- i递增的时候j递减
- 用LCT维护删除时间最小生成树

UR 284

一番战斗之后，程序猿被计算鸡们赶走了。随着垫子计算鸡一声令下：“追！”，于是计算鸡村全村上下开始乘胜追击。计算鸡们希望在新的一年到来之际给程序猿以重创，出掉这一年的恶气。

可是程序猿一追就走，一走就跑，一跑就无影无踪。计算鸡们开始跋山涉水寻找程序猿的踪迹。快乐游戏鸡跟随大部队走着走着，突然说道：“我好像打过类似的游戏”。

快乐游戏鸡玩过的游戏是这样的：给定一棵 n 个结点的树，其中 1 号结点是根。每次玩家可以在树上行走，走过一条边需要 1 秒的时间，但只能往当前所在的点的某个儿子走，不能往父亲走。每次游戏需要从 s 号结点走到 t 号结点去。

玩家有一个总死亡次数，初始为 0。每个结点上有一个程序猿和一个参数 w_i，如果走到结点 i 的时候，当前总的死亡次数小于 w_i，那么玩家就会立刻死亡并回到起点 s，且该死亡过程不需要时间；如果总死亡次数大于等于 w_i，那么玩家就能熟练地对付程序猿从而安然无恙。**注意每次游戏时不需要考虑 s 和 t 上的程序猿。**

该游戏会进行若干轮，每轮会清空总死亡次数并给出一组新的 s, t。现在请你对于每一轮游戏输出走到 t 所需要的最短时间（单位为秒）。

保证每个询问的 s 可以到达 t。

- 你可以发现每次从点 s 出生到撞程序猿死亡跟前几次是怎么死的并没有关系。所以对于每次“从点 s 出生到撞程序猿死亡”的过程都可以贪心选最近的点早死早超生。如果暴力BFS贪心的话复杂度是会炸的，离散化权值也不行。
- 我们换个角度，考虑有多少次从出生到死亡的过程需要一秒，多少次需要两秒等等。
- 具体来说，考虑记录 f[i][j] 表示 i 的子树中和 i 距离小于等于 j 的点的权值最大值。 这样 f[i][j]−f[i][j−1] 就等于有多少次从出生到死亡需要 j 秒。 那么只要从 0 开始枚举 j 把贡献加起来，直到 s 和 t 联通的时候直接从 s 走到 t 就好了。

- 我们把每个询问转化为总和减去大于等于最大权值的部分的后缀和。
- 接着我们按w从大到小加入点，考虑维护**每个点的子树中的最小深度**。
- 这个怎么维护呢，考虑一个点肯定是更新它到根的路径，所以我们用类似LCT的access的方法更新，我们可以保证随时每个链的最小深度都相同，如果当前的链的最小深度小于用来更新的值就直接退出，否则的话我们就更新，最后再把这些更新了的链连起来。

静态lct

- 很多问题并不涉及到树形态的修改，所以用lct维护会常数比较大
- 可以考虑将lct给”静态化”
- 发现HLD的问题是虽然跳轻边的次数是O(logn)的，但是在重链上表现不佳
- 因为HLD是对于每条重链局部平衡的，并不能和全局的结构很好地配合起来

静态lct

- 对于重链我们不建一个平衡的线段树，而是建一个基于biased dictionary problem的bst
- 将每个节点赋予额外权值为其轻儿子子树size和+1
- 对于每条重链我们找该重链的带权重心，以该节点作为bst的根，两边做为左右子树建bst
- 轻边不改变
- 可以证明这样的结构我们每次跳a层，子树size一定会增大1/b，其中a和b是两个常数，好像是bst上跳三层一定增大一倍，所以深度是O(logn)的
- 这个挺好写的，缺点是静态结构无法动态改变树的形态

lct维护子树

- 对于每个节点，维护所有轻儿子的信息的合并
- 每次access的时候对一些点的一个儿子进行轻重边切换，这个时候可以O(1)维护这个切换对该点所有轻儿子信息的合并的影响
- 然后查询一个点子树的时候把其轻儿子的信息加上重儿子的信息
- 由于一个点最多有一个重儿子，所以这里复杂度为O(logn)
- 缺点是必须支持子树信息可差分

静态lct维护子树

- 同理，把lct换成静态lct即可

使用splay维护轻儿子的lct

- 我们对每个lct节点，再开一棵平衡树维护所有轻儿子
- 这样可以维护子树任意信息，只需要处理边标记pushdown到轻边平衡树上的情况，以及轻边平衡树标记pushdown到边上的情况——即原lct上标记以及每个点的轻边平衡树标记互相的影响
- 这样便达成了top tree的全部功能

Thanks for listening