树形结构

编程兔教育

主要内容

- 堆
- 并查集
- 线段树
- 可持久化线段树
- 树状数组
- 倍增求LCA
- 树上差分
- 树链剖分：重链剖分、长链剖分

堆

堆

- 世界上目前有很多种堆，比如二叉堆、斐波那契堆、配对堆、左偏树等等，不过由于各种原因，OI里面提到的堆都是二叉堆，也就是下面要讲的。下文的堆都是指的二叉堆。
- 能做的事情非常有限，可以在单次严格O(logn)的时间复杂度内插入一个数字、删除最小的数字，并严格O(1)询问最小的数字。

关于优先队列(priority_queue)

- C++的STL中关于堆的实现，是一个大根堆。

- 因为别人都写好了所以直接用就行了……所以手写堆其实基本用不到。

删除任意数字

- 除了插入/删除堆顶，还要实现删除堆中任意数字（要保证其一定在堆中）

- 做法很简单，再用一个堆来维护需要删除哪些数字即可。
- 每次取堆顶的时候看看这个元素是否已经被删除了即可。

例1. 堆排序

- 利用堆的特性进行排序。
- 时间复杂度O(nlogn)。

例2. 带插入的中位数(P3871)

- https://www.luogu.com.cn/problem/P3871
- 每次插入一个数字，然后询问所有数字的中位数。
- 中位数是指将一个序列按照从小到大排序后处在中间位置的数。
  （若序列长度为偶数，则指处在中间位置的两个数中较小的那个）

sol

- 做法很简单，维护一个大根堆一个小根堆，使得大根堆维护的是前一半的元素，小根堆维护剩下的。每次插入x视情况放到小根堆/大根堆里面，n是奇数，从大根堆中弹出y，并插入元素min(x,y)，小根堆中插入元素max(x,y)；n是偶数，从小根堆中弹出y，并插入元素max(x,y)，大根堆中插入元素min(x,y)。每次只有常数级别的变化，因此复杂度O(nlogn)。
- 当询问时输出大根堆中的堆顶元素即可。

例3. 行有序数表第k大

- 有一个n*m的数表，每一行数字从小到大。询问这个数表中第k小的数字，要求（忽略读入复杂度）复杂度O((n+k)logn)。

sol

- 从小到大考虑i=1..k。
- 用一个小根堆维护每行有可能成为第i大的答案，每次取堆顶，再扩展新元素（同一行右边的元素）即可。

例4. 第k小点对和

- 给你两个序列a，b，求a(x)+b(y)的第k小。要求O((n+k)logn)

- 做法很简单，a和b都从小到大排序，然后相当于是有个数表，第x行第y个元素是a(x)+b(y)，然后套用刚才的算法即可。

例5. noip2015普及组 T4 推销员(P2672)

- https://www.luogu.com.cn/problem/P2672

- 观察样例可以得到结论：选 X 家的最优解肯定包含选 X-1 家的最优解。
- 那么我们就有一个 $O(n^2)$ 的算法：每次枚举新添加的点，计算添加上去答案会增大多少，取最多的一个就可以了。

sol

- 考虑如果当前最远走到的距离是 S，那么对于某一家，如果 $s_i>S$，选这一家答案就会加上 $2(s_i-S)+a_i$；如果 $s_i\le S$ 答案就会加上a_i。
- 那么我们把所有 $si\le S$ 的拿出来，如果从这里面选就一定会选 a_i 最大的；而如果从 $si>S$ 的里面选就一定会选 $2s_i+a_i$ 最大的（因为答案会加上 $2s_i+a_i-2S$）。
- 用两个大根堆分别维护当前的 S 左边的所有 a_i 以及右边的所有 $2s_i+a_i$，每次选两边增量较大的一个即可。有时候 S 会增大，这时候需要从右边堆里删一些数。这样复杂度就是 O(n log n)。
- 这道题提供了另一种贪心思想：在之前的答案基础上考虑如何添加一个数使得答案最大。它要求最优子结构性质（即这道题里 X 的答案一定是 X-1 的答案添加一个数）。

并查集

并查集

- 能够做的事情很简单：每次合并两个不相交集合，然后询问两个元素是否在同一个集合里。

- 用图论的说法就是每次连接两个点，并询问两点是否连通（无向图）。

考虑一个暴力

- 我们给每个联通块找一个代表元素，每次判断两个点是否连通就只要判断他们的代表元素是否相同即可。

- 这样合并两个集合只要将其中一个代表元素的”父亲”设为另一个代表元素即可。

- 这样每次找代表元素就一直往上跳直到不能跳为止。

- 这看上去太不优秀了，因为有可能每次要跳O(n)步才能找到代表元素。我们希望让跳的次数尽量少。

- 有以下几种优化。

启发式合并

- 我们再对每个集合维护一个大小，每次把小的集合的代表元素的父亲设为大的集合的代表元素。

- 这样每次跳一步，其子树大小就会至少翻倍，这样每个点任意时刻到根的路径的长度都是O(logn)的。

- 这种每次把小的弄到大的上面的操作叫启发式合并。

启发式合并

```
void Union(int x,int y){
        int fx=find(x),fy=find(y);
        if(fx!=fy){
                if(size[fx]>size[fy]) f[fy]=fx,size[fx]+=size[fy];
                else f[fy]=fx,size[fy]+=size[fx];
        }
}
```

按秩合并

- 每个集合维护深度最大值，每次把深度小的挂到深度大的上边。

- 显然新的集合深度变大1当且仅当原先两个集合深度相等，因此可以归纳证明每次最大深度+1，集合大小也会翻倍，复杂度和上一个一样。

按秩合并

```
void Union(int x,int y){
        int fx=find(x),fy=find(y);
        if(fx!=fy){
                if(dep[fx]>dep[fy]) f[fy]=fx;
                else{
                        if(dep[fx]==dep[fy]) dep[fy]++;
                        f[fx]=fy;
                }
        }
}
```

路径压缩

- 前文两种优化看起来不像是人能想到的。

- 不过注意到每次你找到x的代表元素，就意味着x到根的路径上所有点的代表元素都会被确定，因此直接把这些节点的父亲重新设成根即可。
- 可以想象尽管可能某次操作代价很高，但这之后的操作代价就会变低，总代价就不会很高。

路径压缩

```
int find(int x){
        return f[x] == x ? x : f[x] = find(f[x]);
}
```

基于路经压缩的复杂度

- 复杂度是基于均摊的（就是尽管某些单步代价很高但是总代价不大），我不会证明，据说是O(n+log(1+m/n) (n或者m))的，总之也是O(nlogn)级别的。

- Tarjan很牛的证明了当路径压缩和按秩合并一块用的时候复杂度是O(n*alpha(n))的，其中lpha(n)是阿克曼的某个反函数，增长慢到大概你取n是全宇宙的原子总数，这玩意也不超过6大概。

实现

- 实践中从未见过谁真的写了同时路径压缩按秩合并的……
- 一般都只写路径压缩。在某些不能基于均摊的时候会写启发式合并或者按秩合并。总之的确没见过谁实现了两件事同时干的。
- 就算只路径压缩，虽然理论可以卡到O(nlogn)，不过常数很小，所以通常分析复杂度的时候会被看做近似线性。

例6. NOI2015 程序自动分析(P1955)

- https://www.luogu.com.cn/problem/P1955
- 给定 n 个变量，然后给定 m 个约束，每个约束都形如 $x_i = x_j$ 或 $x_i \neq x_j$。
- 判断这些约束条件能不能同时被满足。
- 例如有 4 个变量，4 条约束： $x_1 = x_2$，$x_2 = x_3$，$x_3 = x_4$，$x_1 \neq x_4$，那么这些约束条件就不能同时被满足。
- N，M <= 100000

sol

- 我们首先处理相等关系，用并查集把所有相等的变量划为一个集合。
- 然后再检查所有的不等式，如果某个不等式的两个变量属于同一个集合，那么这两个变量是被之前的等式要求相等，不能与这个不等式同时满足，输出不合法。
- 如果不存在不合法的情况，输出合法。
- 复杂度 $O\left(m\alpha(n)\right)$ 。
- 下标i和j比较大，所以要先离散所有的约束条件。

加权并查集

- 本质上就是并查集中每个点到父亲的边有一个权值，表示在原图中这个点到父节点的某些信息。
- 或者在根节点处维护整个联通块的某些信息。

例7. NOI2002 银河英雄传说(P1196)

- https://www.luogu.org/problemnew/show/P1196
- 有 30000 个战舰，初始时每个战舰都单独在一列。现在有 N 个指令：
- 1.将 x 战舰所在的列作为一个整体接到 y 战舰所在的列尾部。
- 2.询问 x 与 y 是否在同一列，如果是，则输出他们两个之间隔了多少战舰。
- N <= 500000

sol

- 看到合并和查询就首先想到并查集。
- 我们可以简单的实现合并和查询是否连通，但是怎么维护两艘战舰之间隔了多少战舰呢？
- 我们可以额外维护一个数组 dis[]，dis[x] 表示战舰 x 到 fa[x] 中间有多少战舰。
- 可以轻松支持路径压缩，但不支持按秩合并，因为战舰连接的顺序是给定的。
- 复杂度 $O(N\log_2 30000)$，足够通过本题。

例8. JSOI2008 星球大战(P1197)

- https://www.luogu.com.cn/problem/P1197
- 有 N 个星球，被 M 条双向道路连接。现在每次摧毁一个星球，同时连接该星球的道路也被摧毁。问初始时及每次摧毁后剩下的星球构成多少个连通块。
- 两个星球在一个连通块当且仅当它们通过道路可以直接或间接连通。
- N <= 400000 M <= 200000

sol

- 我们发现并查集可以轻松处理合并操作，但是不能处理切断操作。
- 考虑到只有摧毁星球的操作，我们可以倒过来跑一遍。
- 先处理出最终状态，然后再从后往前一个一个加入星球，这样就可以处理摧毁操作了。
- 每合并一次连通块个数会减一，因为是把两个合为一个。
- 复杂度 $O(M\alpha(N))$ 。

例9. NOIP2010 关押罪犯(P1525)

- https://www.luogu.com.cn/problem/P1525

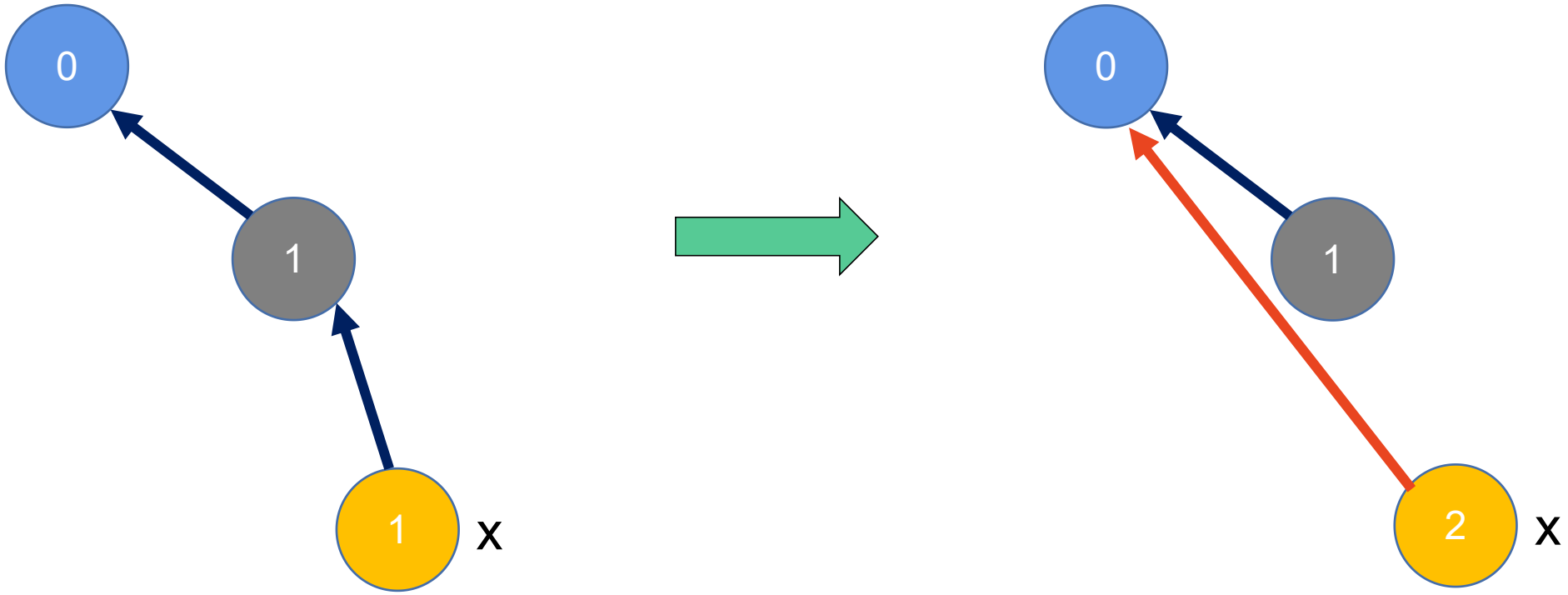
- 一个直观想法是尽可能让边权大的边，两端颜色不同。
- 因此我们从大到小加边，看能否钦定两端的颜色不同。
- 带权并查集维护每个点和他并查集上的父节点是否颜色相同即可。

例10. 食物链(P2024)

- https://www.luogu.com.cn/problem/P2024
- sol1：拓展域并查集。
- sol2：带权并查集维护一个点和其父节点的关系（谁吃谁或者是否同类）。
- 设定不同的距离dis(x,y)来表示不同的关系。在实现距离的时候，设dis[x]表示x到根的距离，dis(x,y)=(dis[y]−dis[x]+3)%3。
- dis(x,y)=0表示x与y是同类，dis(x,y)=1表示x吃y，dis(x,y)=2表示y吃x。

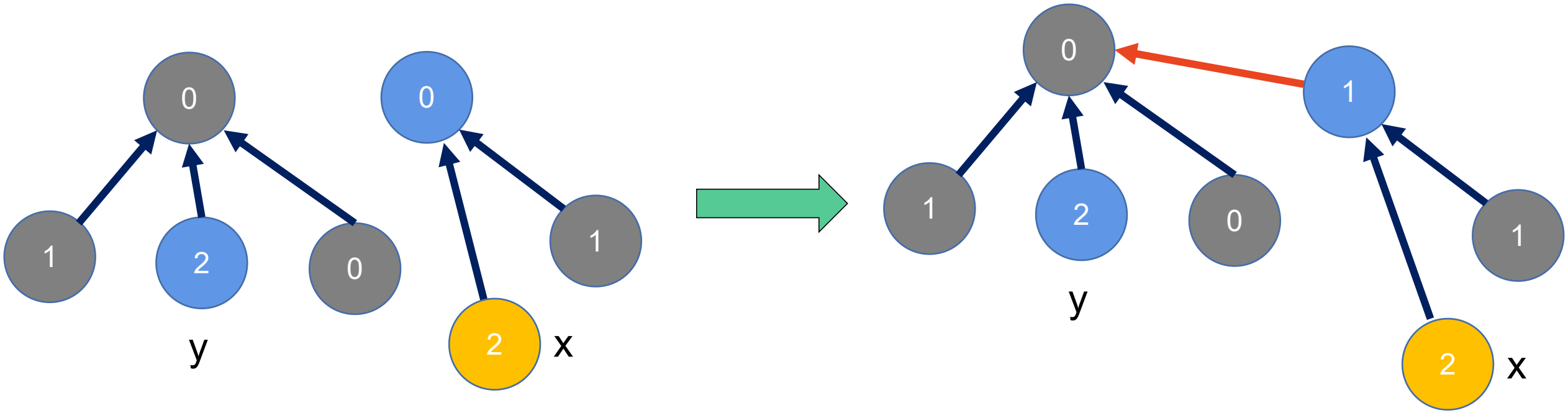
查询

- dis[x]=(dis[x]+dis[fa[x]])%3

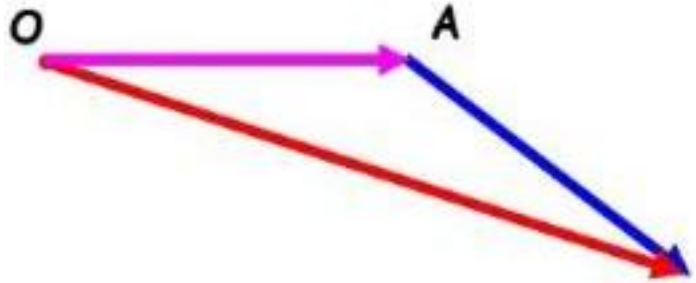


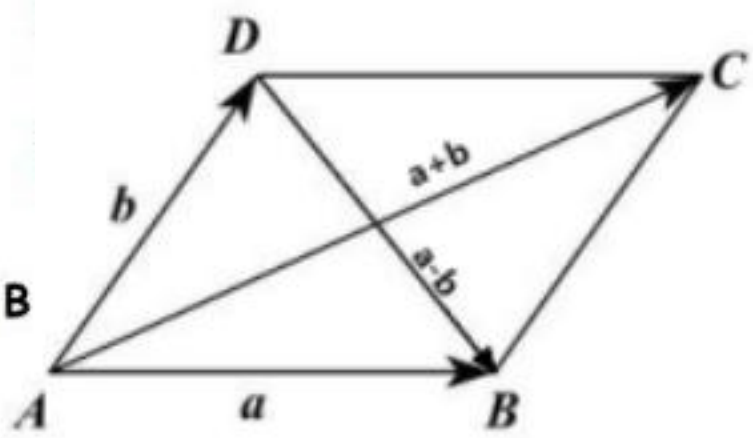
合并

- 假设x吃y



向量加法和减法





- 向量是指具有大小和方向的量。
- 可以形象化地表示为带箭头的线段。
- 箭头所指：代表向量的方向；线段长度：代表向量的大小。
- 三角形法则解决向量加法的方法：将各个向量依次首尾顺次相接，结果为第一个向量的起点指向最后一个向量的终点。
- 平行四边形定则解决向量加法的方法：将两个向量平移至公共起点，以向量的两条边作平行四边形，结果为公共起点的对角线。
- 平行四边形定则解决向量减法的方法：将两个向量平移至公共起点，以向量的两条边作平行四边形，结果由减向量的终点指向被减向量的终点。

向量加法和减法

- 坐标系解向量加减法：在直角坐标系里面，定义原点为向量的起点。两个向量和与差的坐标分别等于这两个向量相应坐标的和与差若向量的表示为(x，y)形式。
- A(X1，Y1)，B(X2，Y2)，则A + B=（X1+X2，Y1+Y2），A - B=（X1-X2，Y1-Y2）。
- 简单地讲：向量的加减就是向量对应分量的加减，类似于物理的正交分解。

例11. HDU 3038

- http://acm.hdu.edu.cn/showproblem.php?pid=3038
- 一类很神奇的题目
- 考虑告诉你一个区间的和本质上就是在讲从一个缝隙走到另一个缝隙的距离。这里的距离满足dis(x,y)+dis(y,x)=0。
- 然后并查集维护一个点和父节点的距离即可。
- 注意给定闭区间[a,b]的和，要转化为(a,b]或者[a,b)。

例12.

- 有一列数字，每次告诉你一个区间的和，或者问能否确定一个区间的和。n<=100000。

例13.

- 每次往集合里面加入一个区间或者询问是否能用集合里面的一些区间异或出某个区间。n,l,r<=100000。

sol

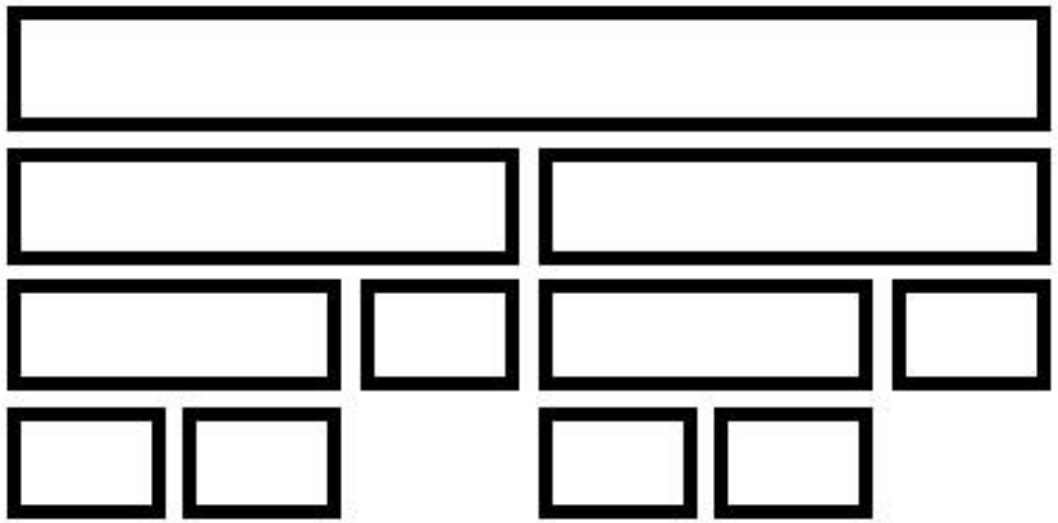
- 和刚刚那个题差不多，本质上一个区间等价于从左边的缝隙走到右边的缝隙的边，然后若询问的时候两端缝隙连通那么就能被表示。

- 将某些区间问题视作关于缝隙的图是一类特殊技巧，有必要掌握一下。

线段树

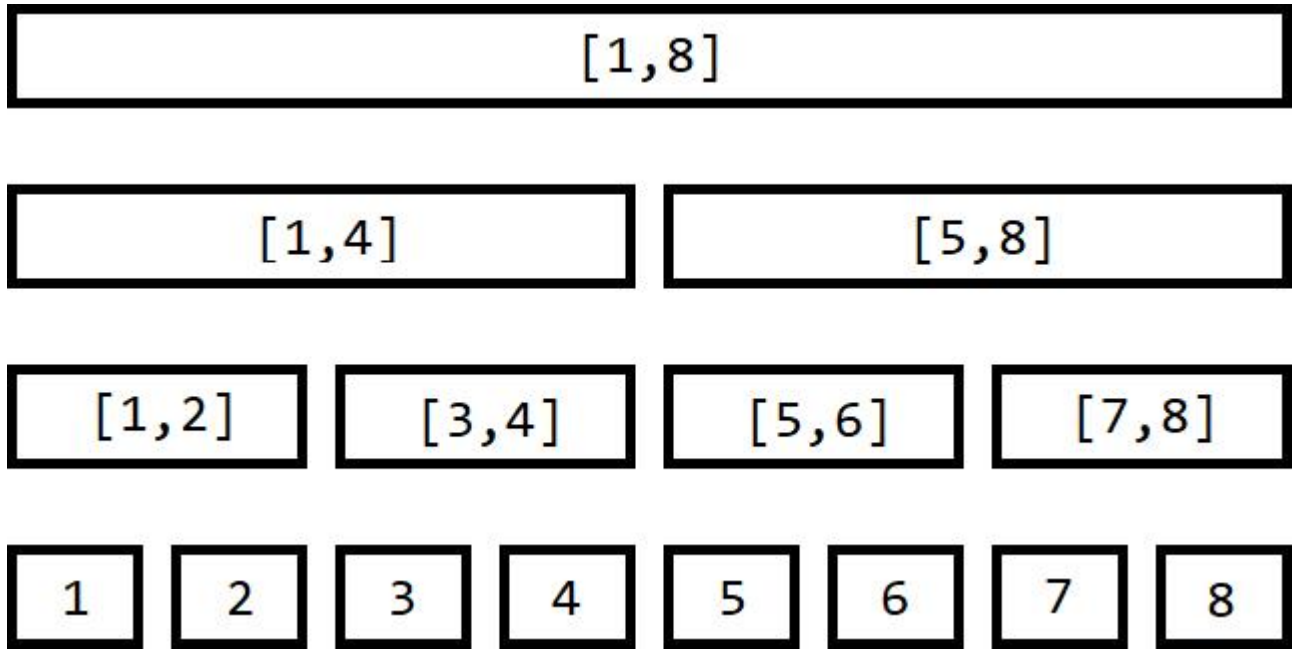
线段树

- 线段树是一种二叉搜索树（实质是平衡二叉树），线段树的每个结点都存储了一个区间，也可以理解成一个线段。
- 用Node[l,r]表示线段树上区间[l,r]的结点，其儿子就是
- Node[l,l+r>>1]
- Node[(l+r>>1)+1,r]
- 当l==r时为叶子，停止。
- 这样尽量等分下去的树形结构。



线段树

- 首先假设n是2的幂，我们在整个数组上建一个如图所示的数据结构。
- 其中每一个位置存储一段区间的和。
- 修改和查询只会访问O(log n)个节点，所以总时间复杂度 为O((n + q) log n)。



线段树能用来干什么

- 线段树的适用范围很广，可以维护修改以及查询区间上的最值、求和。更可以扩充到二维线段树（矩阵树）和三维线段树（空间树）。对于一维线段树来说，每次更新以及查询的时间复杂度为O(logN)。
- 事实上，线段树多用于解决区间问题，但并不是线段树只能解决区间问题。

线段树构建

用数组存线段树并不是真正的完全二叉树，最后一层可能很空，且空节点数量可以达到2n个，因此数组最好开4n大小的长度保证不越界。

- 递归方式建树 build(1,n,1);

```
void build(int l,int r,int x){//x为当前需要建立的结点，l为区间的左端点，r则为右端点
    if(l == r) tree[x].sum = a[l];//左端点等于右端点，即为叶子节点，直接赋值即可
    else{
        int mid = l + ((r-l)>>1);//左儿子的结点区间为[l,mid]，右儿子的结点区间为[mid+1,r]
        build(l,mid, x<<1);     //递归构造左儿子结点
        build(m+1,r, x<<1|1);  //递归构造右儿子结点
        pushup(x);     //更新父节点
    }
}
```

单点修改

- 单点修改就是指只修改线段树的某个叶子节点的值。
- update(1,1,n,p,v);

```
#define lc x<<1
#define rc x<<1|1
#define mid ((l+r)>>1)
void pushup(int x){
  tree[x].sum=tree[lc].sum+tree[rc].sum;
}
void update(int x,int l,int r,int p,int v){
   if(l==r){tree[x].sum=v;return;}
   if(p<=mid)update(lc,l,mid,p,v);
   else update(rc,mid+1,r,p,v);
   pushup(x);
}
```

区间查询/求和

- 递归方式区间查询 query(1,1,n,L,R);

```
int query(int x,int l,int r,int from,int to){
    if(l>=from && r<=to)return tree[x].sum;
    int ans=0;
    if(from<=mid)ans+=query(lc,l,mid,from,to);//左子树和需要查询的区间有交集
    if(to>mid)ans+=query(rc,mid+1,r,from,to); //右子树和需要查询的区间有交集
    return ans;
}
```

例14. POJ2299求逆序对

- 实现用线段树求逆序对个数。

sol

- 值域线段树。
- 对[0,10^9]建树，每次先查询[a[i] +1，10^9]的区间和，加入答案。
- 然后再a[i]的位置上 +1 即可。

- 10^9范围太大，可以考虑对n个数进行离散化。
- 离散化：对n个数排序，最小的数认为是1，第二小的认为是2…这样n个数的值域范围就是 [1,n] 了。

sol

- 离散化模板代码：

```
int cnt=0;
for(int i=1;i<=n;i++) bin[++cnt]=a[i];
sort(bin+1,bin+n+1);
cnt = unique(bin+1,bin+cnt+1)-bin-1;
for(int i=1;i<=n;i++) a[i]=lower_bound(bin+1,bin+cnt+1,a[i])-bin;
```

区间修改

- 以区间加为例。
- 由于区间加不可能一直递归下去，因此我们引入懒标记。 每一个节点设一个add，表示这个区间加了多少。询问到一个点就先把这个点的懒标记下放。

```
void pushdown(int x,int l,int r){
    tree[lc].add+=tree[x].add;
    tree[rc].add+=tree[x].add;
    tree[lc].sum+=tree[x].add*(mid-l+1);
    tree[rc].sum+=tree[x].add*(r-mid);
    tree[x].add=0;
}
```

区间加

```
void add(int x,int l,int r,int from,int to,int v){
    if(l>=from && r<=to){
        tree[x].add+=v, tree[x].sum+=v*(r-l+1);
        return;
    }
    pushdown(x,l,r);
    if(from<=mid)add(lc,l,mid,from,to,v);
    if(to>mid)add(rc,mid+1,r,from,to,v);  pushup(x);
}
```

修改后的区间求和

```
int query(int x,int l,int r,int from,int to){
   if(l>=from && r<=to)return tree[x].sum;
   pushdown(x,l,r);
   int ans=0;
   if(from<=mid)ans+=query(lc,l,mid,from,to);
   if(to>mid)ans+=query(rc,mid+1,r,from,to);
   return ans;
}
```

思考

1. 如果是区间修改怎么操作?

只需要把 += 变成 = 即可。

2. 如果区间查询时查询区间最值怎么操作?

pushup:

tree[x].min = min(tree[lc].min, tree[rc].min);

pushdown:

tree[lc] += tree[x].add, tree[rc] += tree[x].add;

思考

3. 实现区间加和询问区间最小值的个数怎么操作?

新开变量cnt表示当前节点代表区间最小值个数。

pushup:

tree[lc].min == tree[rc].min: tree[x].cnt = tree[lc].cnt + tree[rc].cnt;

tree[lc].min < tree[rc].min: tree[x].cnt = tree[lc].cnt;

tree[rc].min < tree[lc].min: tree[x].cnt = tree[rc].cnt;

pushdown:

cnt 不需要修改。

例15. P2023 [AHOI2009]维护序列

- https://www.luogu.com.cn/problem/P2023
- 实现一个数据结构，支持：
- 区间加
- 区间乘
- 区间求和

sol

- 我们需要同时维护两个懒标记。
- 这时候需要注意两个懒标记之间的影响。也就是在区间乘的 时候，不光需要修改维护的和，还需要修改区间加的懒标记。
- 懒标记下放的时候要注意，先下放乘再下放加。

sol

```
void pushdown(int x,int l,int r){
    int v=tree[x].mul,tree[x].mul=1;
    tree[lc].mul*=v,tree[rc].mul*=v;
    tree[lc].add*=v,tree[rc].add*=v;
    tree[lc].sum*=v,tree[rc].sum*=v;
    v=tree[x].add,tree[x].add=0;
    tree[lc].add+=v,tree[rc].add+=v;
    tree[lc].sum+=v*(mid-l+1);
    tree[rc].sum+=v*(r-mid);
}
```

例16.

- 实现一个数据结构，支持：
- 区间加
- 区间求平方和
- 区间求方差
- $1 \le n, q \le 10^5$

- 例如，洛谷P1471方差、P5142区间方差。

sol

- 设 S_1 为区间和，S_2 为区间平方和，D 为区间方差。 求方差:

$$D=\frac{1}{r-l+1}\sum_{i=l}^{r}\left(a_i-\bar{a}\right)^2$$

$$=\frac{1}{r-l+1}\left(\sum_{i=l}^{r}a_i^2-2\bar{a}\sum_{i=l}^{r}a_i+\left(r-l+1\right)\bar{a}^2\right)$$

$$=\frac{1}{r-l+1}\left(S_2-2\frac{S_1}{r-l+1}S_1+\left(r-l+1\right)\left(\frac{S_1}{r-l+1}\right)^2\right)$$

$$=\frac{S_2}{r-l+1}-\left(\frac{S_1}{r-l+1}\right)^2$$

sol

- 一个懒标记，下放标记的时候必须先修改平方和，因为要用到之前区间和的值。
- 区间加 v 后维护平方和。

$$
\begin{aligned}
S_2' &= \sum_{i=l}^{r} \left(a_i + v\right)^2 \\
&= \sum_{i=l}^{r} \left(a_i^2 + 2va_i + v^2\right) \\
&= S_2 + 2vS_1 + \left(r - l + 1\right)v^2
\end{aligned}
$$

例17.

- 实现一个数据结构，支持：
- 区间加等差数列
- 区间求和
- $1 \le n, q \le 10^5$

- 例如，洛谷P1438无聊的数列

sol

- 维护懒标记的时候维护两个信息：首项和公差。
- 加等差数列对区间和带来的影响可以直接计算，下放懒标记的时候也可以直接计算。

例18. hdu3333 线段树离线查询

- 一个长度为 n 的序列，q 次询问，每次询问 (x, y)：求出 [x, y] 区间中出现过的数字之和。（即重复数字只算一次）
- n<=3e4, q<=1e5

sol

- 考虑直接用线段树维护，发现无法合并左右两个子节点，因为不能统计每个节点出现了哪几种数。

- 解法一：暴力。
- 解法二：设 left[i] 为 a[i] 左边第一次与 a[i] 相等的下标，则 [x, y]区间里若 left[i] < x 则可以累加进答案。
- 预处理 left 数组可以离散化后 left[i]=last[a[i]],last[a[i]]=i;

sol

- 问题变成每次求 [x,y] 中 left<x 的 a 的和。
- 这个时候考虑离线求解。

- 不用每次询问求解，先对left值进行排序，然后顺序枚举 x, 依次把 满足 left[i]<x 的 a[i] 插入线段树中的第 i 个位置。
- 即：对于任意 y, 询问 [x, y] 的话，如果 i 在 [x, y] 中，那 a[i] 统计入答案。
- 时间复杂度O((n+q)logn)

例19. poj 2828 线段树上二分

●有 n 个人来排队，第 i 个人来的时候会排在 p[i] (0<=p[i]<=i) 个人后面，同时它会被分配一个数字 v[i]。
●现在告诉你 n 对（p[i], v[i]），最后按照队伍顺序输出每个人的数字。
●n<=2e5
【样例输入】
4
0 77
1 51
1 33
2 69
【样例输出】
77 33 69 51

sol

- 考虑模拟做法，数组和指针维护都是O（n^2）。
- 正难则反。倒着考虑每个人，就可以确定每个人的位置。
- 比如第 n 个人，一定在 p[n]+1 位置上。
- 然后考虑如何放第 n-1 个人的位置。
- 如果 p[n-1]+1 < p[n]+1, 也就是说第 n-1 个人的位置没有因为第 n 个人的进入而改变，因此可以放在 p[n-1]+1 处。
- 如果 p[n-1]+1 >= p[n]+1, 也就是说第 n-1 个人在第 n 个人进入后向后移了一位，因此第 n-1 个人的位置应在 p[n-1]+2 处。
- 可以总结为第 n-1 个人的位置在除了第 n 个人的位置外的第 p[n-1]+1 处。

sol

- 接着考虑第 n-2 个人的位置，可以分析得应该在除了第 n 个人和第 n-1 个人外的 p[n-2]+1 处。
- 因此我们发现，我们如果只考虑相对位置，这个问题是个规模逐渐减小的递归问题。
- 问题规模为 n 时，确定第 n 个人的位置，删除这个位置后，问题变成 n-1 规模的同样的问题，与n 无关。只是在最后输出需要考虑绝对位置，留出 n 所在的空位。
- 于是问题就变成了： 一开始有 n 个位置，每次查询第 p[i]+1 个空位置，然后将该位置变成非空。

sol

- 一个简单的想法就是用线段树维护一个全 1 的数组 a。
- 查找第 x 个空位置就是找一个下标 i 使得 a[1…i] 的前缀和等于 x。

- 由于 a 是单增的，所以我们可以二分下标 i。
- 用线段树维护区间和，每次二分时查询即可。
- 时间复杂度为 O(n logn logn)

sol

- 注意：线段树本身就有分治的性质。
- 使用分治套分治时我们要思考能否减少冗余计算，只使用一层分治解决问题。
- 于是我们可以直接在线段树上二分。

例20. hdu1540 线段树上找答案

- 有 n 个连在一起的地道，接下来有 m 个操作，每个操作有以下几种：

D x：炸掉 x 号地道

Q x：查询包含 x 的最长地道长度并输出

R：修复上一个被炸的地道

- n,m<=50000

【样例输入】

```
7 9
D 3
D 6
D 5
Q 4
Q 5
R
Q 4
R
Q 4
```

【样例输出】

```
1
0
2
4
```

sol

- 用线段树来维护每个点是否被炸掉，0 表示炸掉，1 表示完好。

- 考虑如何查询包含 x 的最大连续长度。
- 我们先放宽条件，思考如何查询 [1，n] 里的最大长度。

sol

- 假设我们用一个 f[k] 来维护编号为 k 的区间的最长连续长度。
- 考虑怎样合并两个子节点的信息。此时我们发现：
- 不能通过 f[lc]+f[rc] 来更新 f[k]，因为连续段可能没在一起。
- 不能通过 max(f[lc], f[rc]) 来更新 f[k]，因为连续段可能连在一起。
- 因此，对于每个区间，我们需要再维护两个值：
- lmax[k]：区间最左端最长的连续段长度。
- rmax[k]：区间最右端最长的连续段长度。
- 此时，f[k] 可以用 rmax[lc]+lmax[rc] 来更新。更新时 lmax, rmax 也跟着更新。

sol

- 然后回到本问题，现在要求包含 x 的最长连续字段。
- 假设我们现在在编号为 k ， 区间为 [l, r] 。
- 我们首先需要判断包含 x 的最长连续字段是否横跨了 mid。
- 如果 mid-rmax[lc]+1 <= x <=mid, 此时返回 rmax[lc]+lmax[rc]即可。
- 如果 mid+1 <= x <= mid+lmax[rc], 此时返回 rmax[lc]+lmax[rc]即可。
- 否则说明 x 到 mid 中间肯定有一个点被炸掉了。这样另一个子节点对答案没影响。
- 那么递归查询 k 所在的子节点就行。

总结

- 线段树主要用来解决下面这种问题：
- 区间信息满足可加性（比如求和，取 min/max 等）。
- 区间修改对这段区间产生的影响可以 O(1) 更新。

- 线段树维护核心是要考虑怎么合并俩个子区间维护的值。

可持久化线段树

可持久化线段树

- 有的时候，我们不光需要支持修改，还需要支持访问历史版本。
- 这个时候普通的线段树就没法胜任了，因为每次我们都覆盖 了之前的版本。
- 有没有什么方法呢?

方法

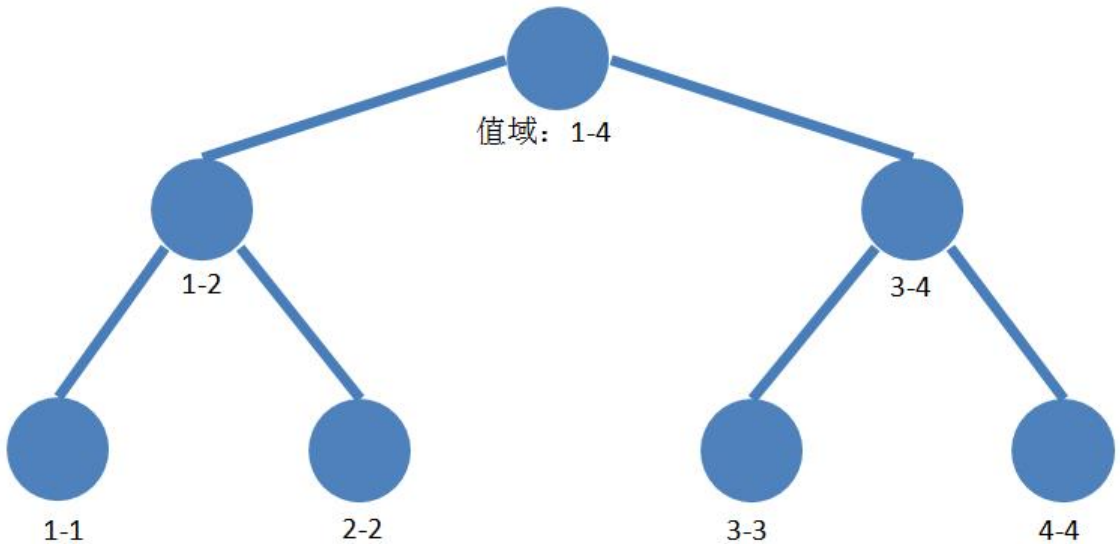
- 方法一：直接记录之前得到的所有的线段树。复杂度 $O(n^2)$。
- 方法二：注意到每次修改的位置都不会很多，所以相同的节点就没必要再记录一遍了。复杂度 $O(n \log n)$。
- 唯一的问题是由于需要每次新建节点，我们没办法再用位运算访问子节点，而需要在每一个点上记录左右儿子的位置。

可持久化线段树

- 概念：可持久化线段树又被称为主席树。可持久化是指更新的同时保留了历史版本，可以获得所有的历史版本。本质上是多颗线段树，不过这些线段树共同使用了一部分枝干。

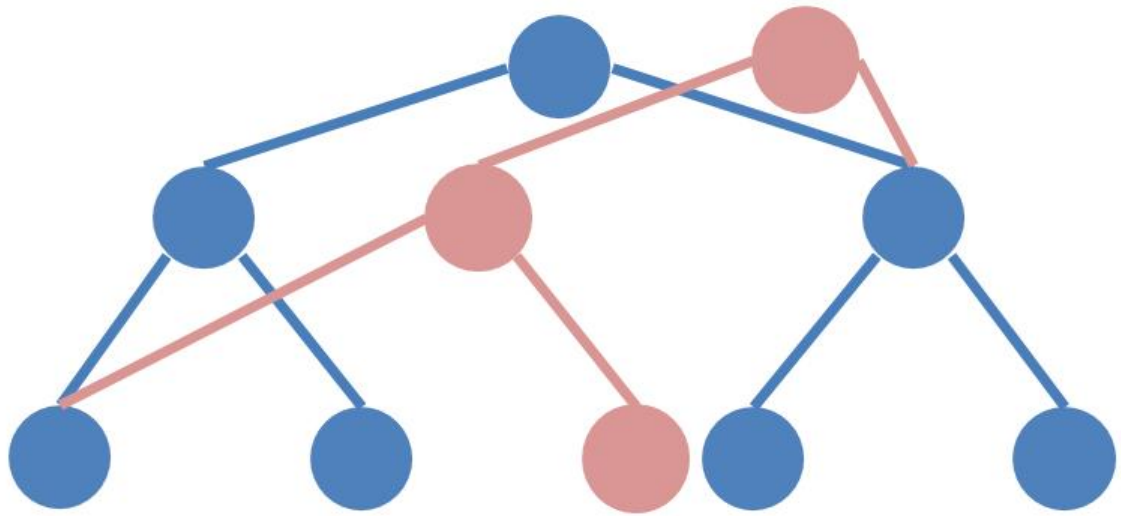
- 实现：可持久化线段树和线段树的实现有很大差别。线段树的left和right表示区间的左右边界，而可持久化线段树的left和right表示该节点的左右儿子。每更新一次，新建log(n)条边。

初始的空线段树



所有结点的cnt域为0，表示空序列的各类数的个数为0

插入第一个元素：2



插入第一个元素后，土红色是新生成的结点，它们是cnt都是1，
其余的蓝色结点的cnt仍是0。每插入一个元素，仅新增log(n)
个结点，耗时log(n)，因此建树的时间、空间均为nlog(n)

例21. P3919 可持久化线段树 1

- https://www.luogu.com.cn/problem/P3919

- 维护这样的一个长度为 N 的数组，支持如下几种操作：
- 在某个历史版本上修改某一个位置上的值。
- 访问某个历史版本上的某一位置的值。

例22. 区间第k大

- 长度为 n 的数组，每次查询一段区间里第 k 大的数。
- $1 \le n, q \le 10^5$

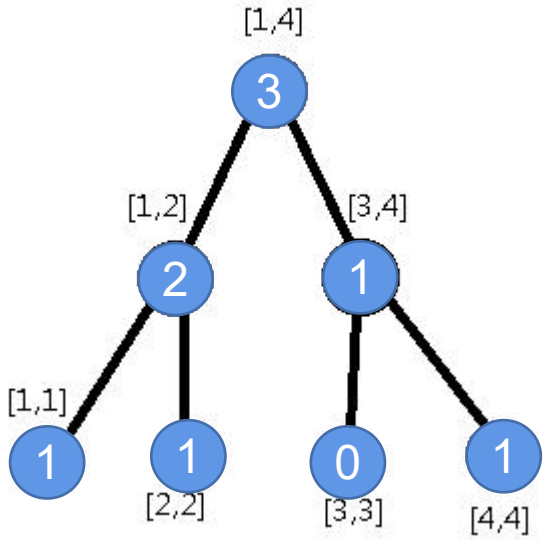
- 例如：洛谷P3834

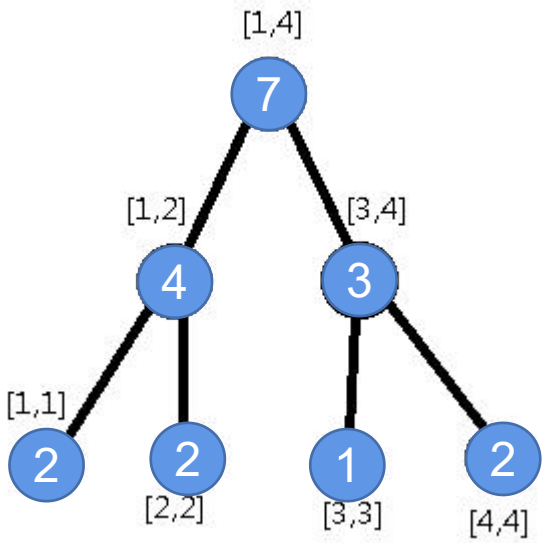
sol

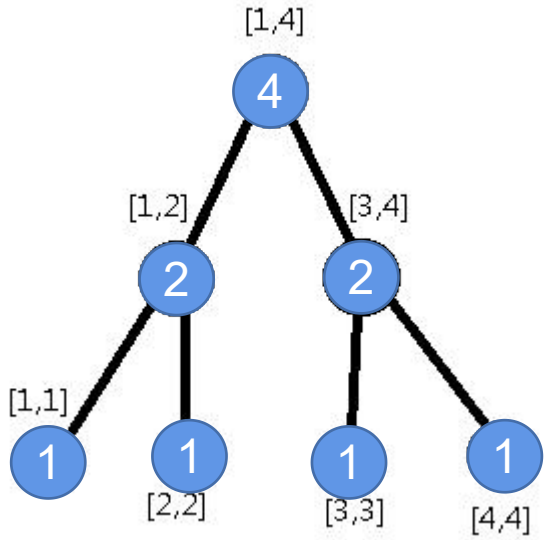
- 我们先考虑一个比较简单的问题：如何维护全局第 k 大?
- 可以建一棵权值线段树，维护一列数中数的个数，然后在线段树上二分解决。
- 维护前缀第 k 大?
- 把这个权值线段树可持久化，这样我们就可以随时拎出来一个前缀。
- 区间第 k 大?
- 相当于两个线段树相减，同样可以用可持久化线段树维护。 时间复杂度 O(nlogn)。

sol

- 假设数列1 2 4 3 2 1 4，求区间 [4,7] 的第 3 小。
- 用 i=7 时的线段树减去 i=3 时的线段树，具体方法是对应节点的权值相减。







可持久化线段树单点加

```
int add(int pre,int o,int l,int r,int p,int v){
  if(l==r){tree[o].sum=tree[pre].sum+v; return o;}
  if(p<=mid){
      rson=tree[pre].child[1],tot+=1;
      lson=add(tree[pre].child[0],tot,l,mid,p,v);
  }
  else{
      lson=tree[pre].child[0],tot+=1;
      rson=add(tree[pre].child[1],tot,mid+1,r,p,v);
  }
  pushup(o);
  return o;
}
```

例23. 树上第k大

- n 个点的一棵树，每次查询一条链 u, v 上第 k 大的数。
- $1 \le n, q \le 10^5$

- 例如：洛谷P2633

sol

- 链剖，然后可持久化，每次拿出来 O(logn) 个线段树进行二分，……
- 恭喜你想到了一个 $O(n \log^3 n)$ 的算法。
- 我们注意到，在可持久化线段树上，我们的 root[x] 不一定去依赖 root[x-1]，完全可以依赖别的位置。
- 所以我们可以每一个点依赖它的父节点。这样每一个点的线段树就是维护的它到根节点的信息。
- 对于一条链 $u \ldots v$，我们设 u 和 v 的 LCA 是 d，那么只需要在 $T(u) + T(v) - T(d) - T(pa(d))$ 上进行二分即可。
- 时间复杂度 $O(n \log n)$。

可持久化并查集

- 实现一个可持久化并查集，不光要支持所有并查集的操作， 还需要支持访问历史版本。
- $1 \le n, 1 \le 10^5$

- 首先科普一个关于并查集的知识点：
- 按秩合并 + 路径压缩：$O(\alpha(n))$（反阿克曼函数）
- 只用按秩合并或只用路径压缩：$O(\log n)$
- 在某些情况下，我们只能用按秩合并不能用路径压缩，比如可持久化。

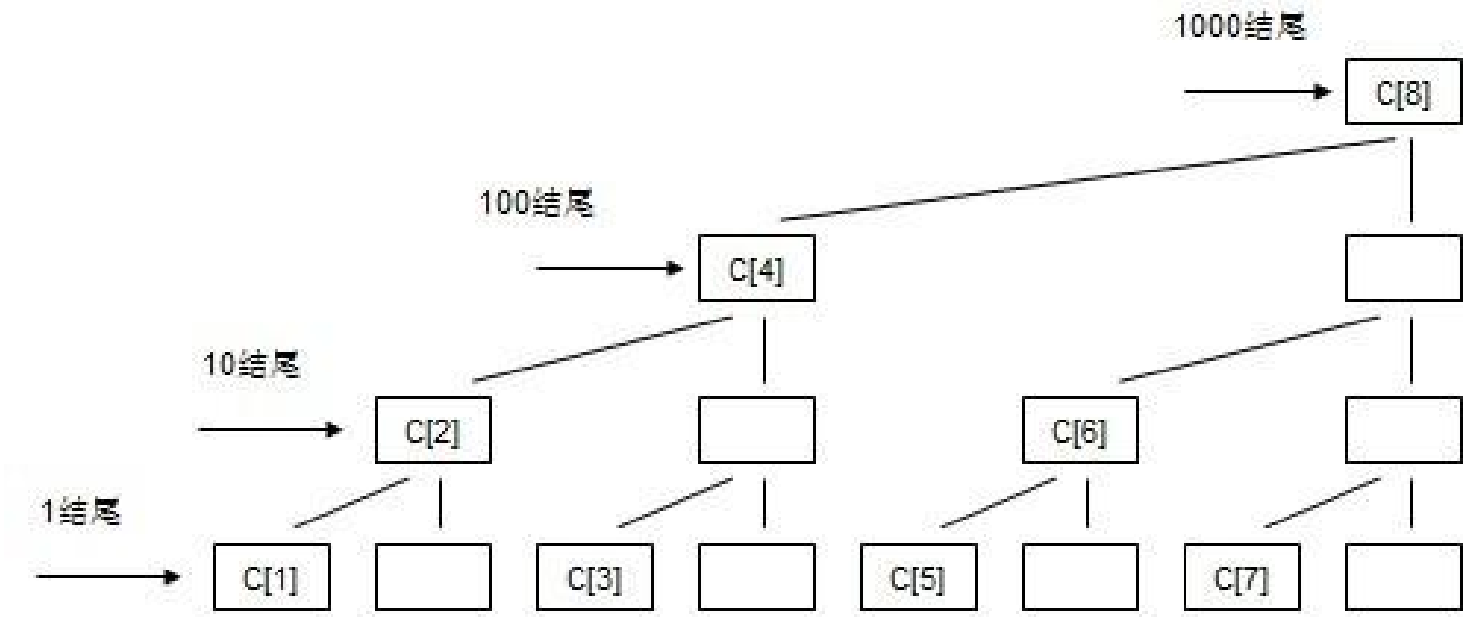
可持久化并查集

- 用可持久化线段树，我们可以实现数组的可持久化。也就是维护一个数组，支持单点修改数组元素和访问历史版本。
- 并查集，其实无非是维护 fa 和按秩合并所需的 dep，将这两个数组都可持久化，我们就可以实现并查集的可持久化。
- 不能路径压缩，因为那样的话 fa 会进行很多修改。
- 时间复杂度 $O(n \log^2 n)$，两个 $\log n$ 一个来自按秩合并并查集，一个来自可持久化线段树。

树状数组

树状数组

- 树状数组是什么东西？就是用数组来模拟树形结构。
- 设序列为A[1]~A[8]，树状数组为C[]
- 图中有一棵满二叉树，满二叉树的每一个叶子结点对应A[]中的一个元素。



观察一下每个C代表了哪些A

- C[1]=A[1];
- C[2]=A[1]+A[2];
- C[3]=A[3];
- C[4]=A[1]+A[2]+A[3]+A[4];
- C[5]=A[5];
- C[6]=A[5]+A[6];
- C[7]=A[7];
- C[8]= A[1]+A[2]+A[3]+A[4]+A[5]+A[6]+A[7]+A[8];
- 这颗树是有规律的:
- C[i] = A[i - 2^k+1] + A[i - 2^k+2] + … + A[i];//k为i的二进制中从最低位到高位连续零的长度

树状数组能用来干什么

- 解决大部分基于区间上的更新以及求和问题，时间复杂度O(logn)
- 通俗来讲：
- 可以动态维护一个序列。
- 支持单点修改，查询前缀和。
- 树状数组可以解决的问题都可以用线段树解决，但是不能解决复杂的区间问题。

结点更新

- 更新a[i]影响哪些位置呢？a[i]包含于C[i]、C[i+2^{k1}]、C[(i+2^{k1})+2^{k2}]...

```
int lowbit(int x){    //求2^k
    return x&(-x);
}
void update(int i,int value){     //在i位置加上value
    while(i <= n){
        c[i] += value;
        i += lowbit(i);
    }
}
```

前缀和

- 前缀和SUM[i] = C[i]+C[i-2^{k1}]+C[(i-2^{k1})-2^{k2}] + ……，如：
  SUM[7]=C[7]+C[6]+C[4]

```
int getsum(int i){    //求A[1 - i]的和
    int res = 0;
    while(i > 0){
        res += c[i];
        i -= lowbit(i);
    }
    return res;
}
```

树状树组的构建

- 最简单的方法是把每个数插入进去。
- 这样是O(nlogn)的。

```
for(int i = 1; i <= n; i++){
        cin>>a[i];
        update(i,a[i]);   //输入初值的时候，也相当于更新了值
}
```

树状树组

- 在信息可减的情况下，可以各种差分：
- 单点加，区间查询
- 区间加，单点查询
- 区间加，区间查询

单点加，区间查询

- 查询区间为[l,r]的时候
- 差分为前r个数的和减去前l-1个数的和
- getsum(r)-getsum(l-1)

区间加，单点查询

- 不妨换个思路，利用差分建树，D[i] = A[i] - A[i-1]，规定A[0]=0
- 把区间[l,r]加k，差分为D[l]加k，D[r+1]减k
- 查询单点的时候只需要查询对应这个点的前缀和，A[i] = $\Sigma_{j=1}^{i}$ D[j]

区间加，区间查询

- 维护原数组的差分数组$D[i] = A[i] - A[i-1]$
- 于是$A[i] = \Sigma^{i}_{j=1} D[j]$
- 前缀和$\sum^{n}_{i=1} A[i] = (n+1) * \sum^{n}_{i=1} D[i] - \sum^{n}_{i=1} (D[i] * i)$
- 需要维护两个树状数组，$sum1[i] = D[i]$，$sum2[i] = D[i]*i$

二维树状数组

- 单点修改，矩形求和
- 直接多一层for即可，因为树状数组自带完美的差分

例24.

- 给出一个序列，求逆序对个数
- 逆序对即i < j且a[i] > a[j]的对

sol

- 按值域开树状树组？如果数值非常大，直接开数组显然是不行了，这时的解决办法就是离散化。
- 按数值由大到小排序，这样得到的下标值即是离散化结果，等效于原序列的数值。把这些下标值当成原序列，按照树状数组求逆序对的原理做。
- 每次把这个数的位置加入到树状数组中，因为是排完序之后，所以之前加入的一定比后加入的大，然后再查询当前这个数前面的位置。

倍増求LCA

树

- 树的存储：邻接表
- 对一般树的存储OI常见有两种实现方式：
- 1）用vector。
- 2）用“链式前向星”（不知道谁起的奇怪的名字，其实就是链表）

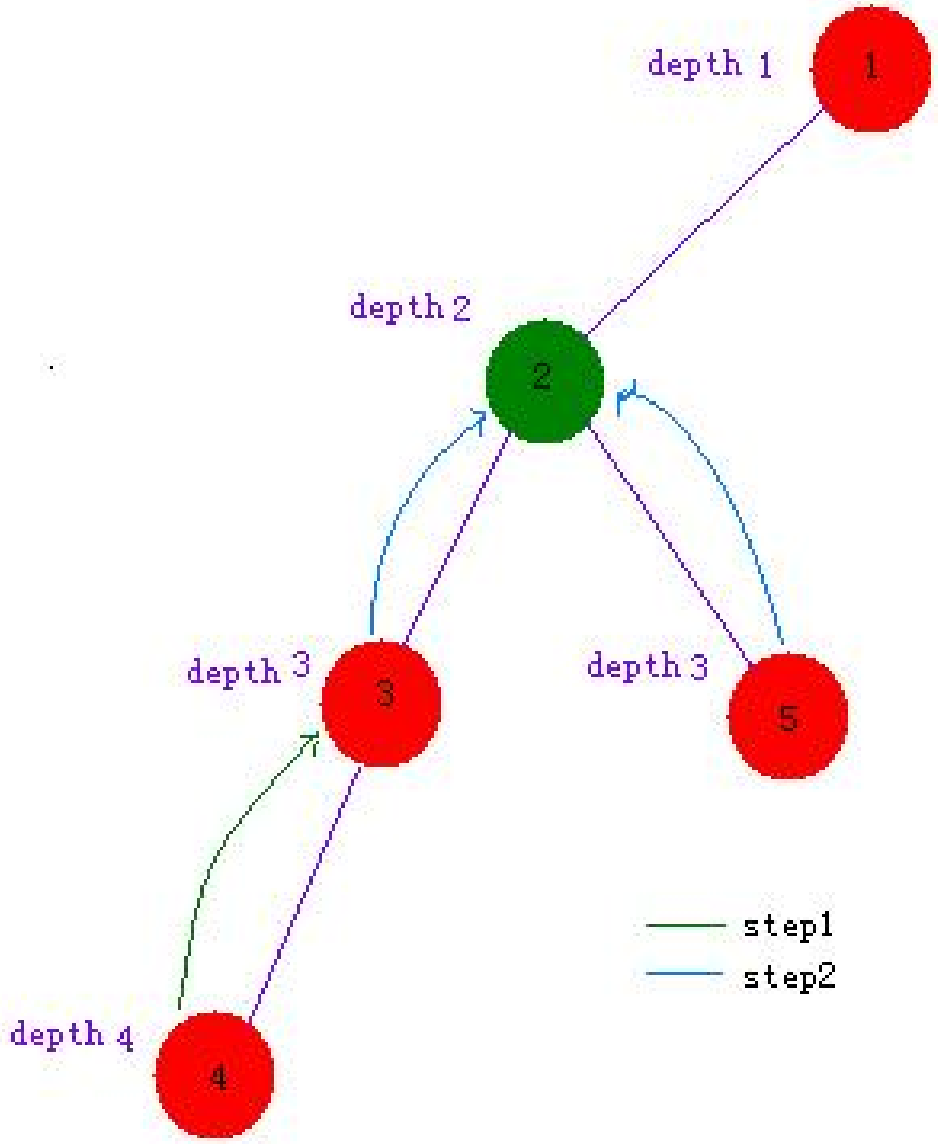
- 有人说vector慢，有人说vector访问内存连续，总之到底哪个更快我也不清楚。我个人更喜欢使用后者。

LCA

- LCA（Lowest/Least Common Ancestors）是两个点的最近公共祖先。
- 通常有四种快速求LCA的办法：tarjan求LCA、RMQ、树上倍增、树剖，其中第一个尽管是OIer比赛场上能写的理论复杂度最优的算法但是只能离线求，RMQ长度要翻倍。实践中通常使用倍增和树剖，二者在随机数据下实现良好的话常数差不多，不过后者空间线性好像优秀一点……
- （顺带LCA是可以O(n)预处理O(1)回答的）
- 不过倍增的思想十分有用，还是有必要通过这个问题了解一下的。
- 时间关系前两种方法就不讲了。

倍增求LCA

- 例如求4和5的LCA。
- 最粗暴的方法就是先dfs一次，处理出每个点的深度，然后把深度更深的那一个点（4）一个点地一个点地往上跳，直到到某个点（3）和另外那个点（5）的深度一样。
- 然后两个点一起一个点地一个点地往上跳，直到到某个点（就是最近公共祖先）两个点“变”成了一个点。



倍增求LCA

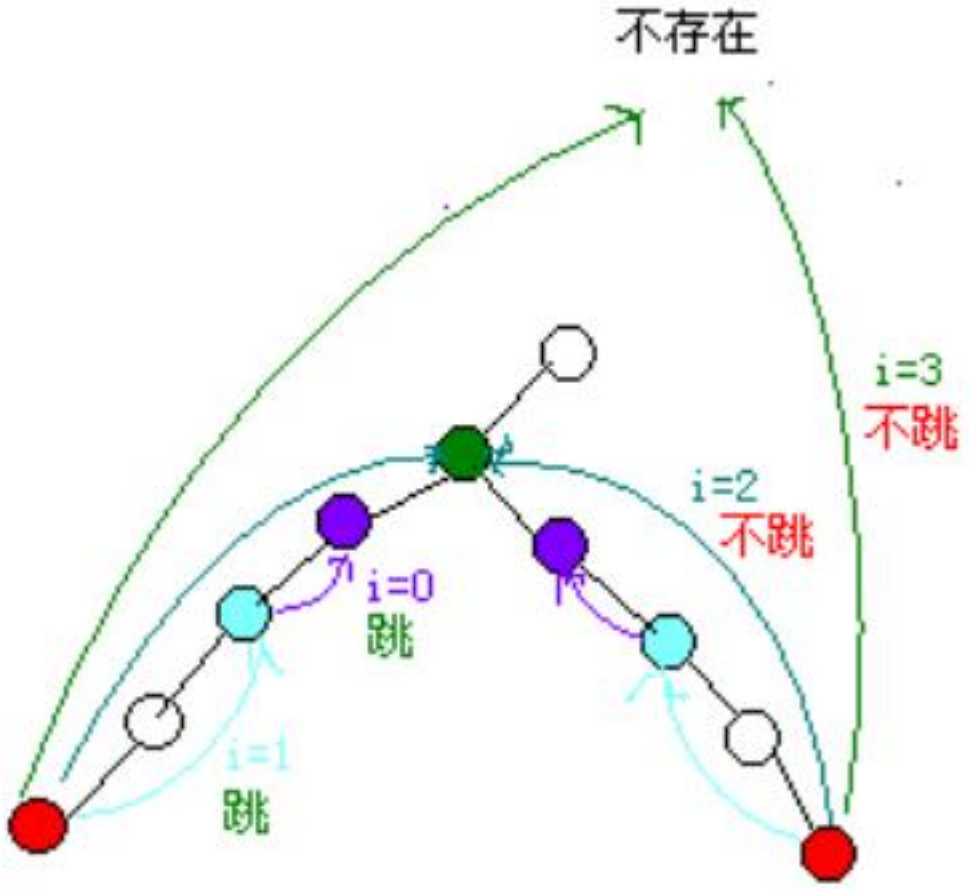
- 倍增的话就是一次跳2^i 个点。
- 我们先预处理up[x][i]表示从x往上跳2^i步后会到哪里，以及每个点的深度。预处理复杂度$O(nlog_2n)$.
- 如何查询呢?
- 分两步走:
  - 将u和v移动到同样的深度。
  - u和v同时向上移动，直到重合。第一个重合的点即为LCA。

倍增求LCA

- 移动到同样深度
  - 令u为深度较大的点。我们从$\log_2 n$到0枚举，令枚举的数字为j。如果从u向上跳2^j步小于了v的深度，不动；否则向上跳2^j步。这样一定能移动到和v同样的深度。
  - 假设一共要跳k步，上面的算法相当于是枚举k的每个二进制位是0还是1。

倍增求LCA

- 从同样的深度移动到同一个点
  - 和上一步类似。很快就会发现一个很严重的问题：两个点按照这样跳，不能保证一定是最近的。
  - 从$\log_2 n$到0枚举，令枚举的数字为j。如果两个点向上跳2^j将要重合，不动；否则向上跳2^j步。
  - 通过这种方法，u和v一定能够到达这样一种状态——它们当前不重合，如果再往上跳一步，就会重合。所以再往上跳一步得到的就是LCA。



- 我们注意到，在整个倍增查找LCA的过程中，从u到v的整条路径都被扫描了一遍。如果我们在倍增数组f[i][j]中再记录一些别的信息，就可以实现树路径信息的维护和查询。

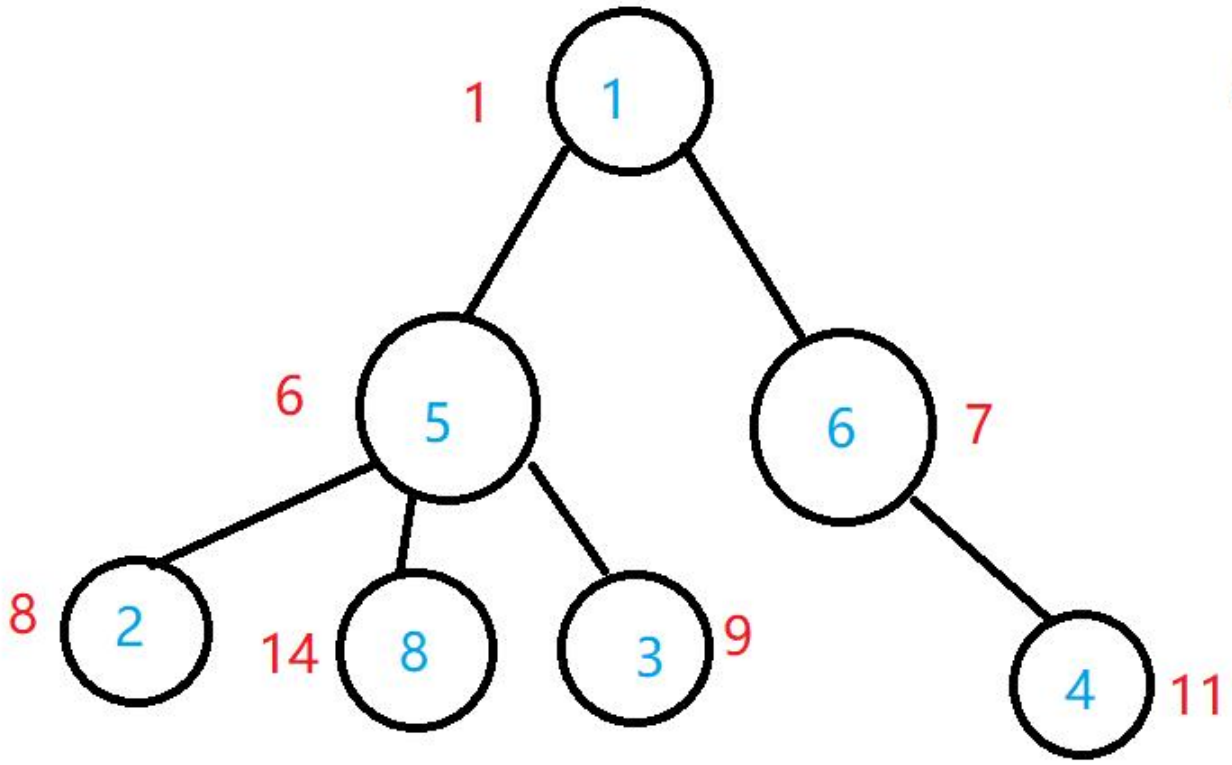
例25. 求树上两个点的距离

- dis(x,y)=dis(root,x)+dis(root,y)-2*dis(root,LCA(x,y))

树上差分

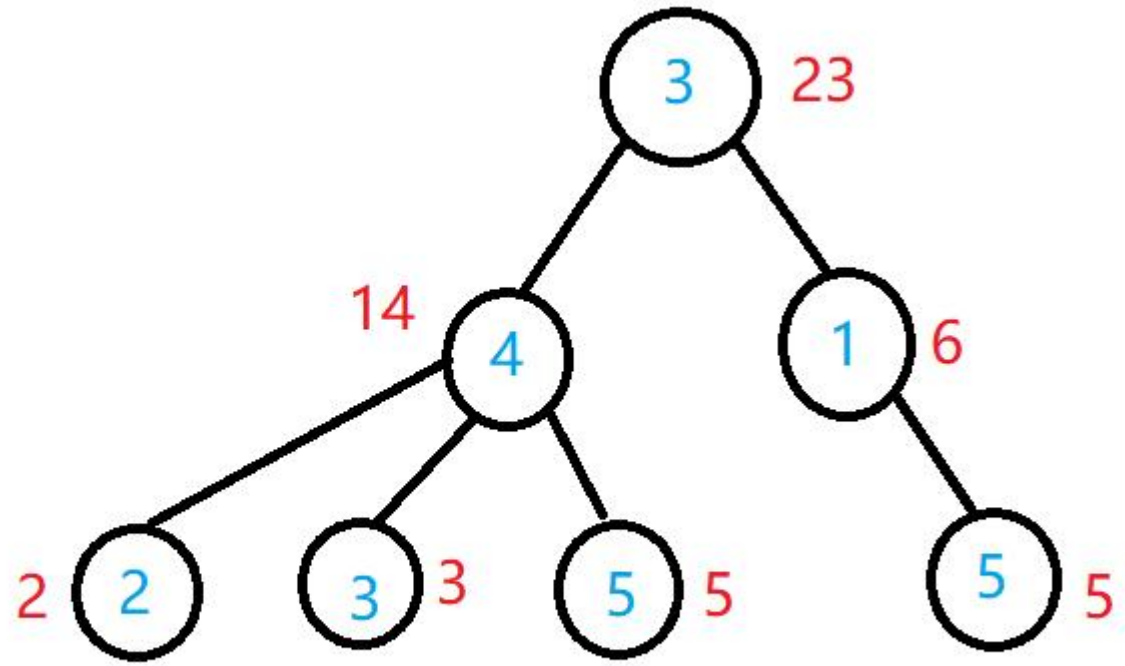
树上前缀和

- 从根向下，到某点的前缀和就是从根到这一点的这条路径上点（或者边）的权值之和。



树上前缀和

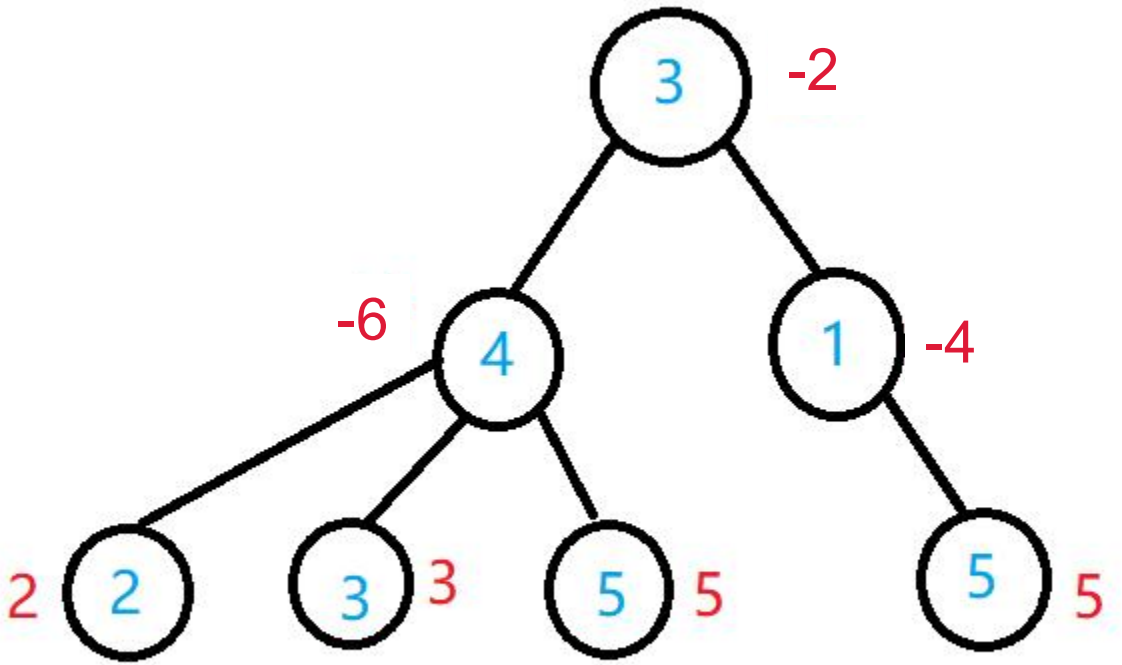
- 从底向上，某一点的前缀和就是以那个点为根的一棵子树上所有点权（或边权）之和。



点上蓝色数字为权值，

点边红色数字为前缀和结果

树上差分



点上蓝色数字为权值，

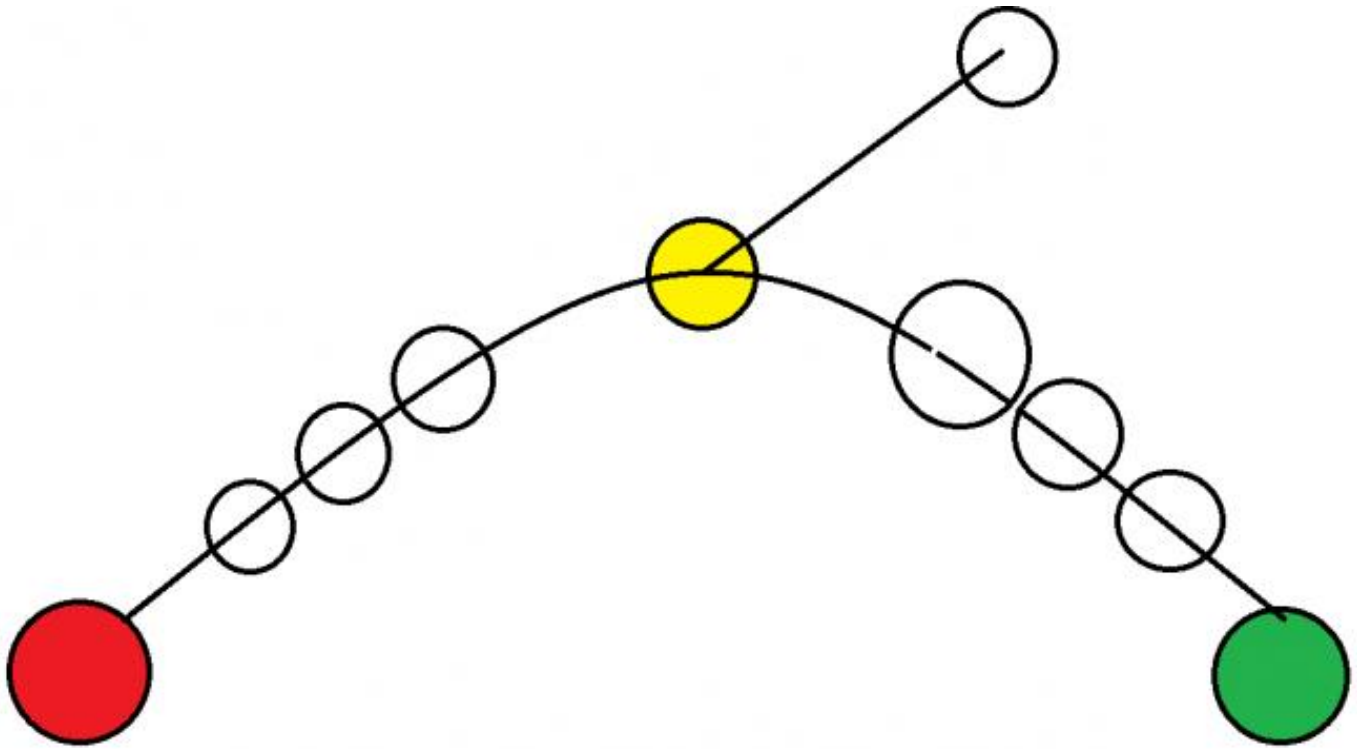
点边红色数字为差分结果

树上差分

- 某个节点对它到根节点的路径上的每个点的贡献。
- 分为点差分和边差分。
- 点差分：点带权
- 边差分：边带权
- 某个节点的权值即为它子树所有节点的差分和。

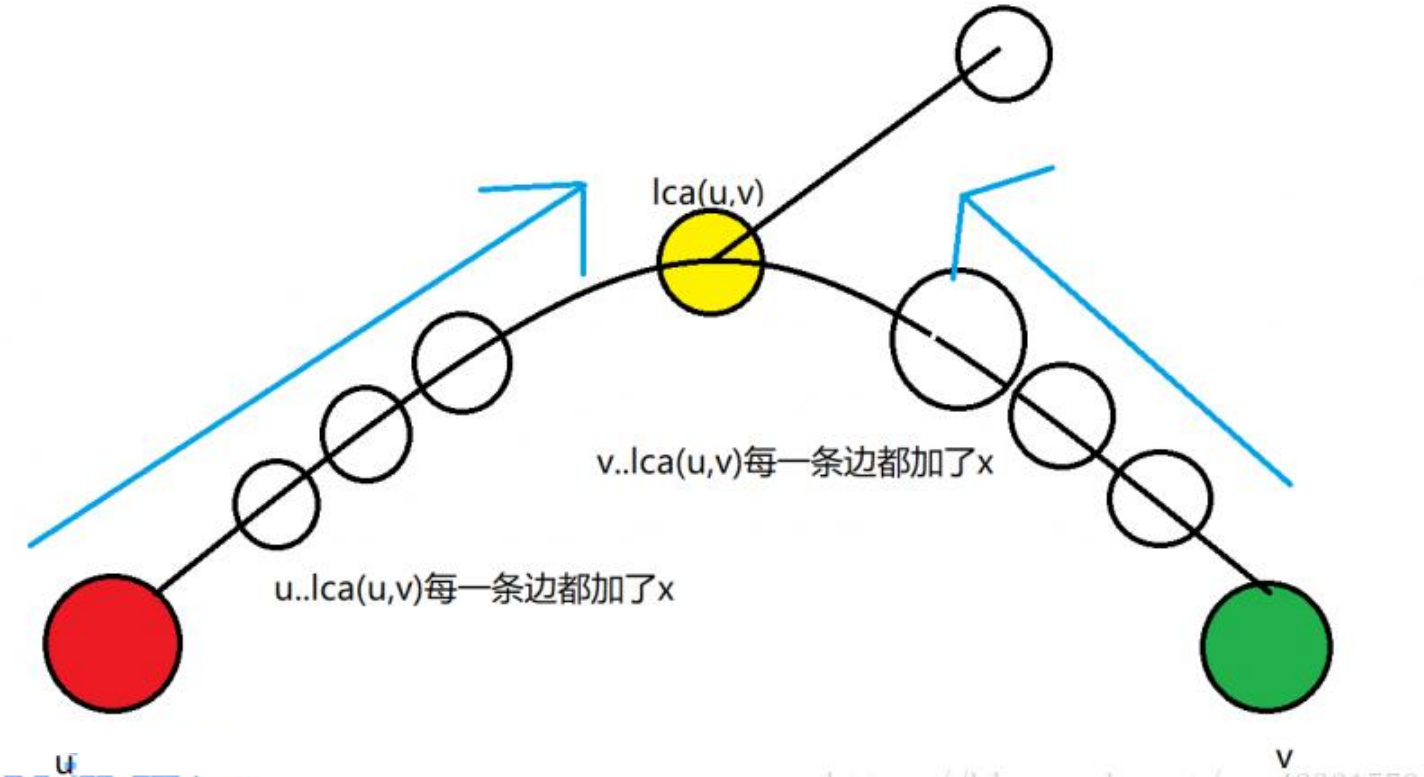
点差分

- 有n次修改操作，每次把u..v的所有点权都加x，最后问点权最大的为多少。
- diff[u]+=x,diff[v]+=x,diff[lca(u,v)]-=x,diff[fa(lca)]-=x;

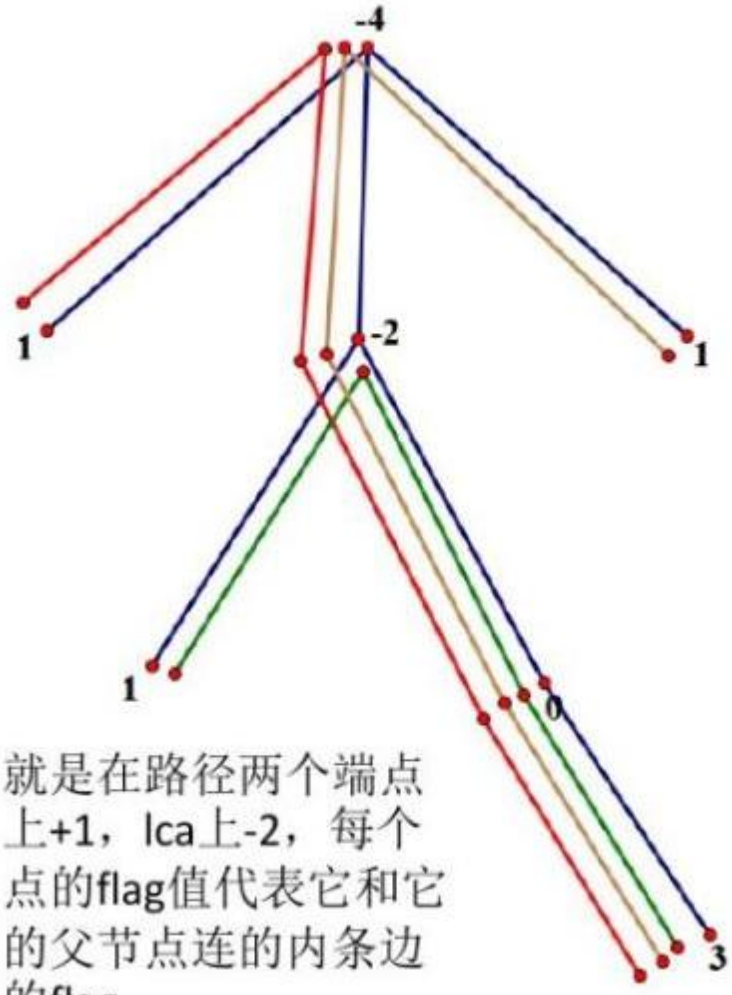


边差分

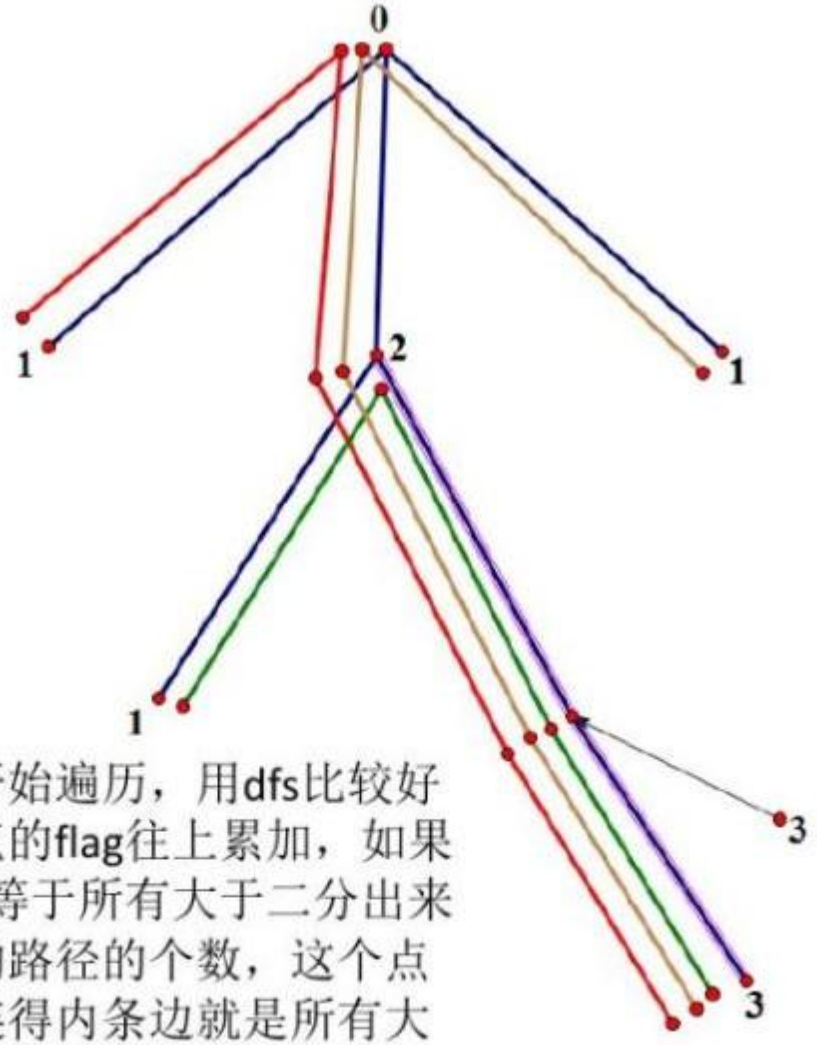
- 给你一棵树，有n次修改操作，每次把u..v的路径权值加x，最后问从ux..uy的路径权值和。
- diff[u]+=x,diff[v]+=x,diff[lca(u,v)]-=2*x;



怎么求节点权值——以边差分为例



就是在路径两个端点上+1，lca上-2，每个点的flag值代表它和它的父节点连的内条边的flag



从叶子节点开始遍历，用dfs比较好写，把子节点的flag往上累加，如果某个点的flag等于所有大于二分出来的内个答案的路径的个数，这个点与它父节点连得内条边就是所有大于二分出来的内个答案的路径的交集中的一条边

例26. P3128 [USACO15DEC]Max Flow P

- https://www.luogu.com.cn/problem/P3128

- LCA+点差分模板题

例26. P2680 [NOIP2015 提高组] 运输计划

- https://www.luogu.com.cn/problem/P2680

- 重链剖分求LCA，用倍增会TLE
- 边差分

树链剖分

关于LCA的一个性质

- 树上三个点x，y，z，满足x的dfs序小于y的，y的小于z的。
- 那么LCA(x,z)=（LCA(x,y)和LCA(y,z)中深度较小的那一个）。

- 也就是dpt[LCA(x,z)]=min(dpt[LCA(x,y)],dpt[LCA(y,z)])

- 也就是说树上给你一些点，这些点两两点的LCA形成的集合，和按照dfs序排序后相邻两个点取LCA形成的集合，是一样的。

思考一个问题

- 将树从x到y结点最短路径上所有节点的值都加上z——树上差分，O(n+m)。
- 求树从x到y结点最短路径上所有节点的值之和——LCA，dfs预处理每个节点的dis（即到根节点的最短路径长度），distance(x,y) = dis(x) + dis(y) - 2*dis(lca)，时间复杂度O(mlogn+n)。

- 如果刚才的两个问题结合起来，成为一道题的两种操作呢?
- 刚才的方法显然就不够优秀了（每次询问之前要跑dfs更新dis）。
- 树链剖分华丽登场，将“树”分割成多条“链”，然后利用数据结构来维护这些链（本质上是一种优化暴力）。

树链剖分

- 树链剖分简称树剖，有很多种，在OI圈里最经典最普及的是重链剖分，一般提到树剖就是专指重链剖分，除此之外最近在全国范围内比较有名的是长链剖分，再其余的用处就不大了。

- 所谓树剖，就是把树剖成若干条关于点的深度连续变化链（也就是从上到下的链），使得任意两条链之间交集为空，所有链并集为全集。

重链剖分

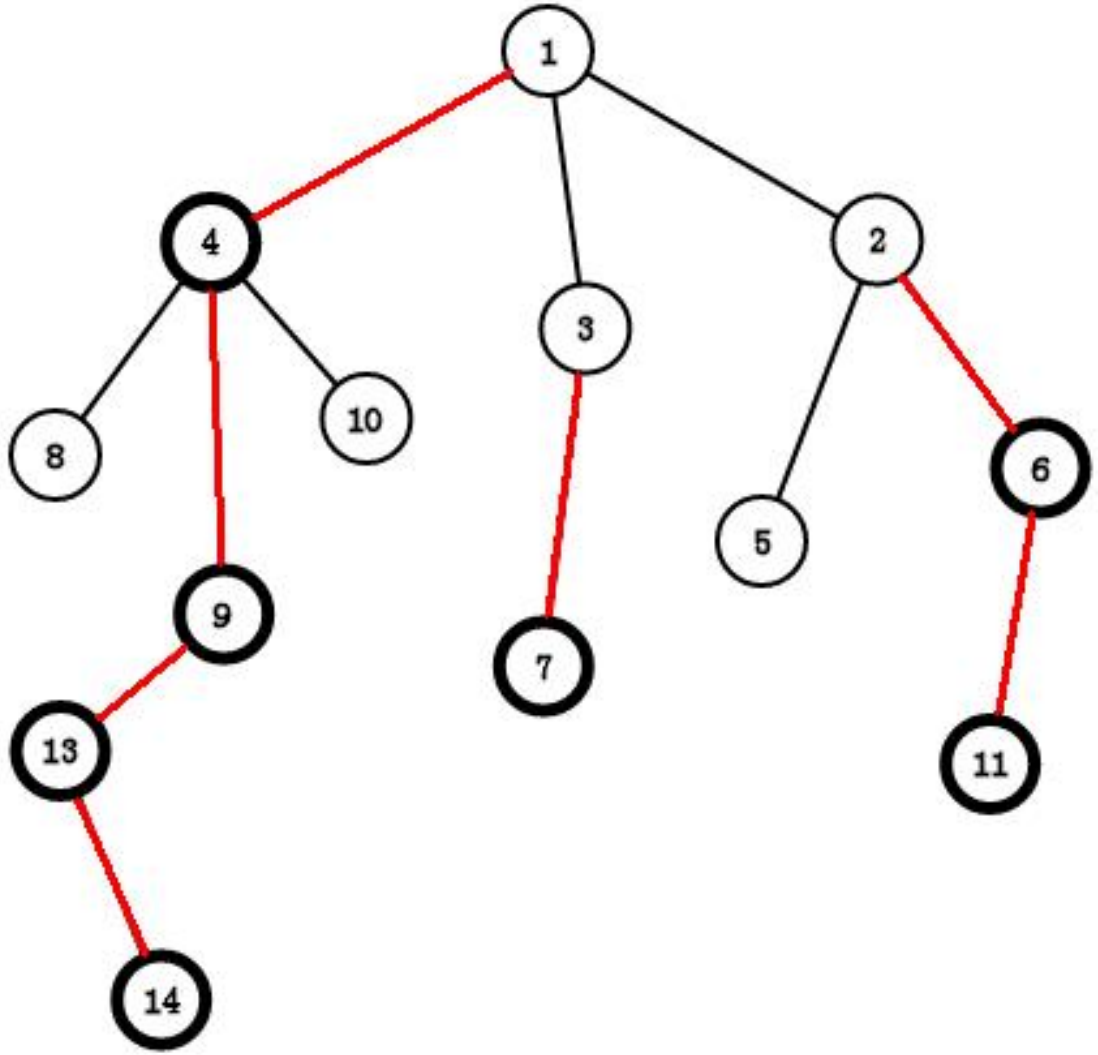
- 重儿子：父亲节点的所有儿子中子树结点数目最多（size最大）的结点。
- 轻儿子：父亲节点中除了重儿子以外的儿子。
- 重边：父亲结点和重儿子连成的边。
- 轻边：父亲节点和轻儿子连成的边。
- 重链：由多条重边连接而成的路径。
- 轻链：由多条轻边连接而成的路径。

重链剖分

- 对每个点求出其大小sz、深度d、父节点fa以及大小最大的儿子son，然后son[x]称为x的重儿子，其余的叫轻儿子。
- 然后每个点和其重儿子（如果有的话）中间的边涂黑，称为重边，有公共点的重边形成链，这些链就形成了一个树剖。和轻儿连的边叫轻边。特殊的，如果一个点不是其父节点的重儿子且这个这个点是叶子节点，那么这个点单独是一条链。
- 然后维护出每个点的链顶。

重链剖分

- 黑色节点是重儿子，其余节点是轻儿子。
- 红边是重边，黑边是轻边。
- 叶节点没有重儿子，非叶节点有且只有1个重儿子。
- 并不是轻儿子之后全是轻儿子，比如2后面就有6和11两个重儿子。
- 可以这样理解：当一个节点选了重儿子之后，并不能保证它的轻儿子就是叶节点，所以就以这个轻儿子为根，再去选这个轻儿子的轻重儿子。



性质

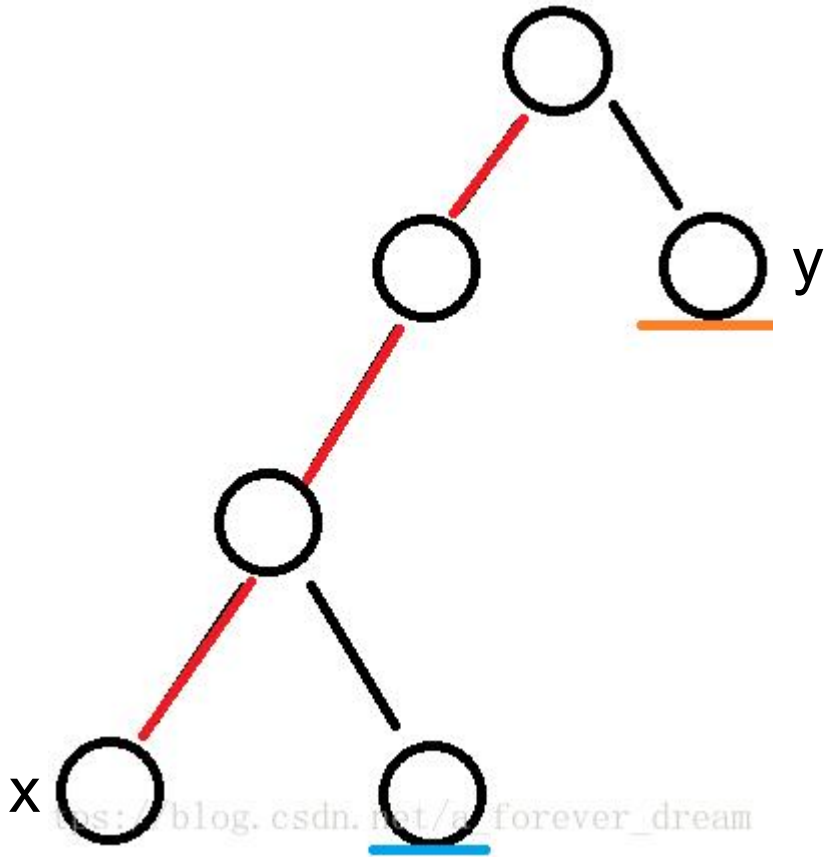
- 任意一个点到根的路径上，不会经过超过log(n)条重链，换句话说，轻边数量不超过logn。
- 这是由于，一个轻儿子的子树大小不会超过其父节点子树大小一半，也就是说，每次向上经过一条轻边，子树大小就要至少翻倍。

- 这句话的另一种表达形式就是：所有点的轻儿子子树大小之和是O(nlogn)的，这一跟重链剖分其实关系不大的结论在做一些问题的时候会有用处。

能做什么？

- 比如求LCA
- 由于每个点到根的路径经过轻边数量是logn的，那么其到根的路径就是logn条重链的前缀拼成的。
- 类似的两个点间的路径就是logn条重链的区间拼成的。
- 如何提取出这些重链的区间？若两个点在同一条链上则结束；否则跳那个链顶深度更大的点，跳到链顶的父节点。
- 你发现这样顺便就求出了LCA……（最后两个点在同一条链上的时候，深度较浅的那个点就是LCA）

- 为什么每次跳的不是深度更大的节点而是所在链顶深度更大的往上跳？
- 因为，如果先跳深度更大的，有可能会跳过头。
- 假如深度大的先跳，那么x会先跳，
- 然后了，x会跳到不知道哪里去……
- 但如果先跳y的话，他会先跳到自己，
- 然后再跳到他的父亲——根节点，
- 然后他们就在同一条重链上了。



- 更常用的操作是用一些数据结构维护这个重链，比如线段树、树状数组、平衡树、主席树、全局二叉平衡树、重量平衡树之类的。

例27. P3384 【模板】轻重链剖分/树链剖分

- https://www.luogu.com.cn/problem/P3384

- 1 x y z，表示将树从 x 到 y 结点最短路径上所有节点的值都加上 z。
- 2 x y，表示求树从 x 到 y 结点最短路径上所有节点的值之和。
- 3 x z，表示将以 x 为根节点的子树内所有节点值都加上 z。
- 4 x 表示求以 x 为根节点的子树内所有节点值之和。

长链剖分

- 和重链剖分类似，只不过选择son[x]的时候，选择那个最长链长度最大的作为son[x]。

性质

- 性质一：对树长链剖分后，树上所有长链的长度和为O(n)。
- 性质二：任意一个点x的k级祖先y所在长链长度一定≥k。
- 性质三：任何一个点向上跳跃重链的次数不会超过sqrt(n)次。

应用

- O(nlogn)-O(1)在线询问一个点的k级祖先。
- O(n)统计每个点子树中以深度为下标的可合并信息。

例28.

- 多次询问x子树中到x距离为k的点有多少。
- 要求线性。

- 直接长剖，然后用f[x,i]表示x子树中距离x为i的有多少，首先初始化f[x,0]=1,f[x,i]=f[son[x],i-1]，后者可以直接指针位移一下O(1)完成。
- 然后依次用O(轻儿子最长链深度)的时间复杂度更新出这个数组即可。这样就线性了。

例29. K级祖先问题

- 多次询问一个点x的第k级祖先。要求O(nlogn)+O(1)。

- 长链剖分，在链顶处维护倍增中的up数组，以及其向上、向下（沿重链）走1<=i<=链长步，到达哪个点；
- 每次对于询问的k，若距离链顶不足k，可以直接算；否则跳到链顶重新算k，找到一个最大的不超过k的2的幂次，比如t，然后通过倍增数组跳t步，然后可以发现此时所在点y，其所在链长肯定大于等于t，因此仍然能够直接求出。

Thanks