数据结构 Day2

马皓然

分治介绍

- 在计算机科学中，分治法是一种很重要的算法。字面上的解释是“分而治之”，就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。这个技巧是很多高效算法的基础，如排序算法(快速排序，归并排序)，傅立叶变换(快速傅立叶变换)……

分治介绍

- 该问题的规模缩小到一定的程度就可以容易地解决
- 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。
- 利用该问题分解出的子问题的解可以合并为该问题的解；
- 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

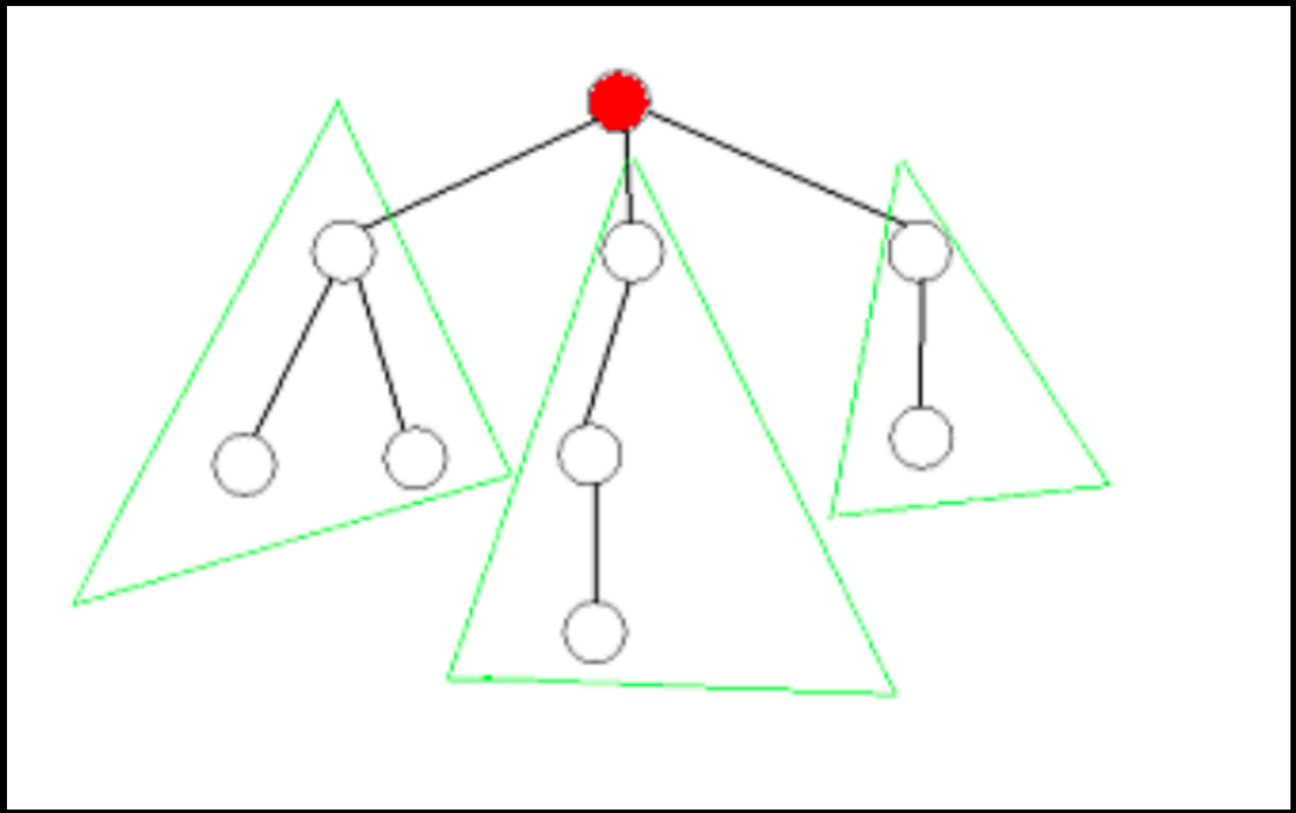
树的定义

- 树被定义为没有圈的连通图，具有以下几个性质:
  - 在树中去掉一条边后所得的图是不连通的。
  - 在树中添加一条边后所得的图一定存在圈。
  - 树的每一对顶点 U 和 V 之间有且仅有一条路径。
- 通常来说我们会在一个线性结构上进行分治，但由于树结构的特殊性，使得分治算法在树上也经常有很好的效果、

内容

- 基于点的分治
- 基于边的分治
- 树的路径剖分（树链剖分）
- 基于询问的分治

点分治

- 基于点的分治方法：首先选取一个点将无根树转为有根树，再递归处理每一颗以根结点的儿子为根的子树。
- 

点分治

- 题目：poj1741
- 题意：求树上距离小于等于K的点对有多少个
- N<=10000

点分治

- 树上两个点之间的路径分为两类：
  - 经过根节点；
  - 不经过根节点（在某一棵子树中）；
- 换一种计算方法：
  - 任意选择一个点为根，把无根树变为有根树；
  - 依次取出以每个点为根节点的子树；
  - 考虑子树中经过根节点的路径，统计符合要求的路径条数。

点分治

- 如果我们已经取出了子树Tree：
  - 设子树Tree的根节点为Root；
  - 利用树形动态规划可以求出Tree中的每一个点在Tree中的深度，记为D[i] (Depth)；
  - 以及它在Root的哪一个子节点的子树中，记为B[i] (Belong)。
  - 特别地，B[Root]=Root。
- 一次树形DP的复杂度是O(n)的。
- 性质：一条路径经过Root，当且仅当路径的两个端点x、y满足B[x]≠B[y]。

点分治

- 把树中的点都取出来放进结构体数组中，按照D[x]递增排序。
- 只需要用两个指针 i, j 一个从前开始一个从后开始扫描数组就可以计算出满足D[x]+D[y]<=k的点对(x,y)的个数，其中x、y分别是i、j指向的节点编号。
- 计算方法如下：
  - 把左指针从左向右扫描，对每个点计算以它为端点的满足要求的路径条数。
  - 左指针向右扫描的过程中，恰好使得D[x]+D[y]<=k的右指针位置是单调递减的。
  - 设F[i]表示在左、右指针中间满足B[x]=i的点x的个数，F[i]很容易维护。
  - 设当前左指针指向点x，则 j-i-F[B[x]] 就是要求的答案。
- 排序是O(nlogn)的，扫描是O(n)的。

点分治

- 对每一个点为根的子树都要做上述计算。
- 所以我们可以从整棵树的根开始，先对整棵树进行上述计算。
- 然后删掉根节点，将原树划分为多个子树，再对这些子树进行上述计算。
- 如此递归下去，直到子树里只包含1个点时结束。

- 可以发现，每递归一层，上述计算涉及到还未删除的所有节点。
- 所以时间复杂度是O(递归层数$*\ nlogn$)。
- 最坏情况出现在链状数据上，时间复杂度达到了O(n^2)。

点分治

- 由于原问题中的树是无根的，所以对于点分治过程中涉及的每一棵子树，可以以它的重心为根进行上述计算。
- 删去点x后，原树分成的若干棵子树中节点最多的那棵子树的节点数记为Cnt。
- 树的重心定义为：使得Cnt的值最小化的点。

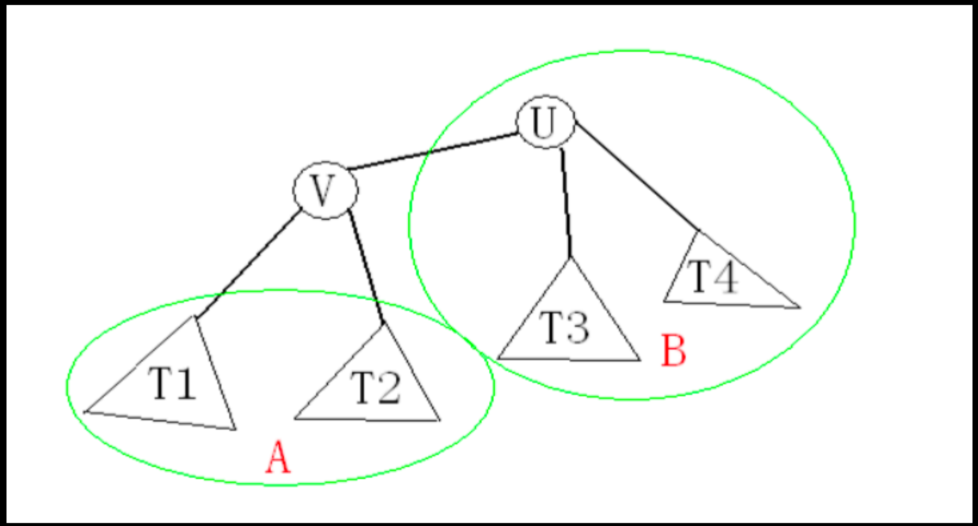
- 这样递归层数最多为O$(logn)$层。
- 所以时间复杂度不会超过O$(nlog^2n)$。
- 算法的瓶颈在于排序，某些题目上可以采用基于非比较的排序算法降低时间复杂度。
- 有些实现方法可以达到O$(nlogn)$，不过本题中提到的方法有较广泛的扩展和适用性。

点分治

- 从上题中可以看出，点分治算法的复杂度与递归层数有很大的关系，所以我们需要对点分治的最大递归层数进行分析。

- 对于基于点的分治，我们通常会选取一个点，要求将其删去后，结点最多的树的结点个数最小，这个点被称为“树的重心”。而对于这个问题，可以使用在树上的动态规划来解决，时间复杂度为 O(n)，n 为树的节点总数。

点分治

- 引理：每棵树存在一个点使得去掉其剩下的子树的结点个数均不大于 N / 2
- 证明：假设U是树的重心，令其为根节点，它与V1，V2，……，Vk相邻，用 Size(X) 表示以 X 为根的子树的结点个数。记 V 为 V1，……，Vk 中 Size 值最大的点。
- 我们采取反证法，即假设 Size(V) > N/2，那么我们考虑如果选取 V 作为整颗树的根节点，记 Size’(X) 表示此时以 X 为根的子树的结点个数。
- 如图，对于 A 部分，显然 Size'(Ti) < Size(V)，对于 B 部分，Size'(U) = N - Size(V) < N/2 < Size(V) ，这与U是树的重心矛盾。
- 定理得证。
- 

点分治

- 由引理可得，在基于点的分治中每次我们都会将树的结点个数减少一半，因此递归深度最坏是O(log N)的，在树是一条链的时候达到上界。

点分治

- 题目：bzoj2599
- 一棵树,每条边有权.求一条路径,权值和等于K,且边的数量最小
- Input：第一行 两个整数 n, k；第2..n行 每行三个整数 表示一条无向边的两端和权值 (注意点的编号从0开始)
- Output：一个整数表示最小边数量，如果不存在这样的路径输出-1

点分治

- 直接在子树上做：

- 对每个节点到根的权值和进行Hash，记录每个权值和对应的最少边数量。

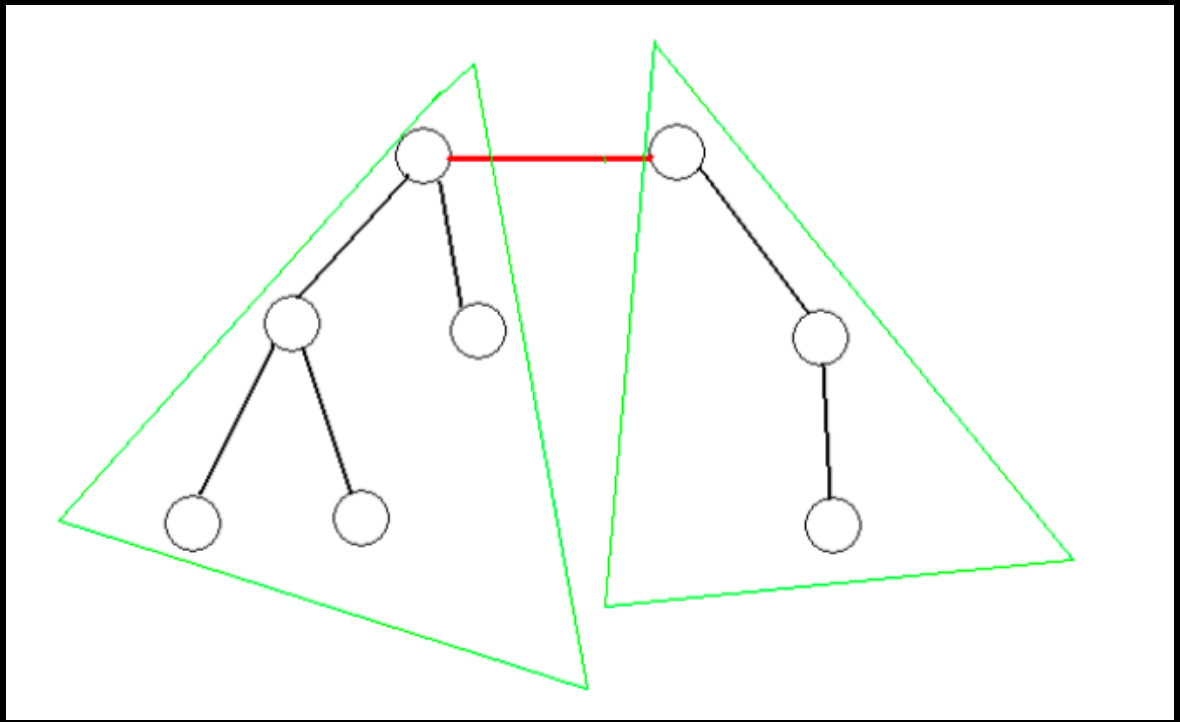
- 设根的子节点为S1、S2……Sk，对于以子节点Si为根的子树中的点为端点的路径，另一端点的取值范围就在以S1~S(i-1)为根的子树中。

- 所以对于每个Si，先以Si为根的子树中所有节点作为端点更新答案，然后把这些节点添加到Hash表中。

点分治

- 变成数组上的线性问题用指针扫描法：
- 扫描的时候做一些改变：两个指针指向的点x、y满足B[x]≠B[y]，并且以x、y为端点的路径权值和等于M时，用该路径的边数更新答案即可。
- 左指针确定后，满足D[x]+D[y]=M的右指针的范围是一个区间。而D[x]相同的左指针也是一段区间。
- 为了避免枚举这些左指针时，对右指针的范围区间重复扫描更新，可以对这个区间记录路径边数的最小值M1和次小值M2，及其对应节点X1、X2，并且B[X1]≠B[X2]。
- 这样对于这一段左指针，可以更新答案的右指针位置不是X1就是X2。

边分治

- 基于边的分治方法：在树中选取一条边，将原树分成两棵不相交的树，递归处理。
- 

边分治

- 题目：spoj QTREE4　http://www.spoj.com/problems/QTREE4/
- N个点构成的树。每个点有黑白两种颜色。M个操作，两种类型。
  - 一个点的颜色取反；
  - 询问树上距离最远的两个白点的距离。
- N，M <= 50000。

边分治

- 边分治算法可以在O($nlog^2n$)的时间解决该问题。
- 对原树T进行边分治，会得到一棵新树T'。
- 新树的叶子节点是原树的节点，其它节点是原树的边。
- 构建T'的方法如下：
- （1）每次找到T中的一条中心边x-y作为T'的根节点。

  中心边：断开后使得分成的两棵树的节点数之差尽可能小的边。
- （2）这条中心边把T分成以x为根的子树T1和以y为根的子树T2两部分。

  断开点：子树T1在点x处与中心边断开，我们称x为子树T1的断开点。
- （3）递归对T1和T2进行边分治建树，把T1'和T2'的根作为T'的子节点。

边分治

- 分治过程中，设当前正在处理子树T，其中心边为x-y。
  - 对以x为根的子树T1进行树状DP，求出T1中所有点到x的距离。
  - 为子树T1建立一个堆H1，堆中存储的是T1中所有白点到x的距离、以及T1中所有黑点到x的距离的相反数（负值）。
  - 对于以y为根的子树T2，进行同样的树状DP和建堆H2操作。
  - 把堆H1关联到新树T1'的根节点（T1的中心边）上。
  - 把堆H2关联到新树T2'的根节点（T2的中心边）上。
- 这样除了整棵新树的根节点外，其它新树中的节点都被关联了一个堆，堆中存储的是子树里每一个白点到断开点的距离、以及每一个黑点到断开点的距离的相反数。

边分治

- 查询树上距离最远的两个白点的距离：
  - 为新树的每个点x关联一个值P：以x为根的子树中两个白点的最远距离。
  - P=Max(P1,P2,堆H1的最大值+堆H2的最大值+中心边权值)。P1和P2是x的子节点关联的值，H1和H2是x 的子节点关联的堆。
- 把点x的颜色取反：
  - 新树x到T’根节点的路径上所有关联的堆中，把点x的值变为相反数，并调整堆。
  - 更新x到T’根节点的路径上所有关联的值。

边分治

- 从上题中可以看出，边分治算法的复杂度与递归层数也有很大的关系，所以我们需要对边分治的最大递归层数进行分析。

- 对于基于点的分治，我们选取的边要满足所分离出来的两棵子树的结点个数尽量平均，这条边称为“中心边[1]”。而对于这个问题，可以使用在树上的动态规划来解决，时间复杂度为 O(n)，n 为树的节点总数。

边分治

- 引理：如果一棵树中每个点的度均不大于 D ，那么存在一条边使得分出的两棵子树的结点个数在 $[N/(D+1), N*D/(D+1)]$ 之间。($N >= 2$)

- 证明：

    - 令 D 为所有点的度的最大值。 当$D=1$时，命题显然。当 $D>1$时，我们设最优方案为边$U-V$，且以U,V 为根的两棵子树的结点个数分别为 S 和 $N - S$ ，不妨设 $S >= N - S$。

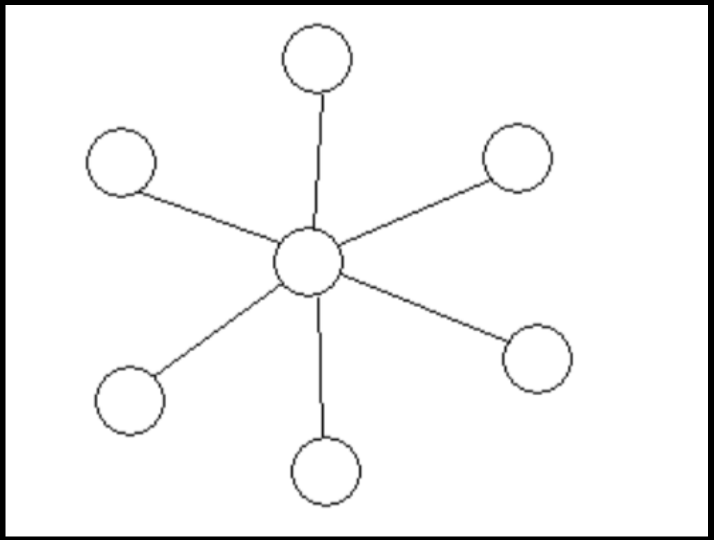
    - 设 X 为 U 的儿子中以 X 为根的子树的节点数最大的一个，我们考虑另一种方案 $X-U$，设除去边 $X-U$ 后以 X 为根的子树结点个数为P。显然 $P>=(S-1)/(D-1)$，由于$P<S$且边 $U-V$ 是最优方案，所以 $N-P >= S$ ，与 $P>=(S-1)/(D-1)$ 联立可得 $S<=((D-1)N+1)/D$，又 $N>=D+1$，所以 $S<=N*D/(D+1)$。

- 证毕。

边分治

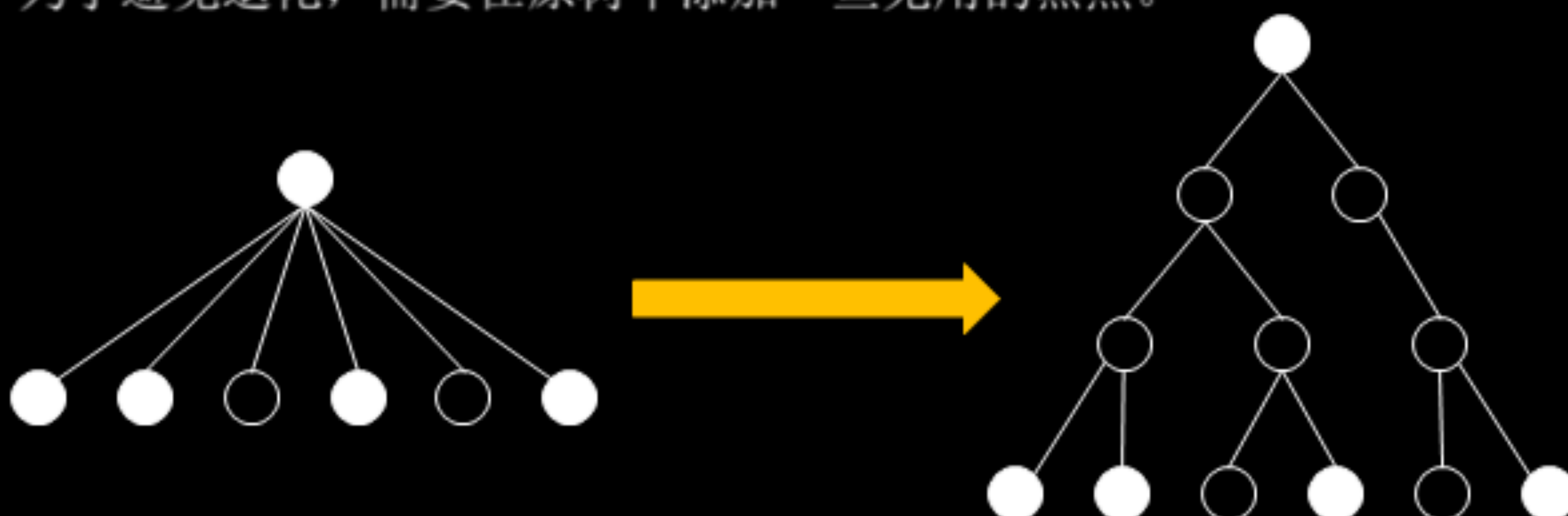
- 由定理 2 我们可以得到在 D 为常数时，基于边的分治递归最坏深度为O(log N)。
- 但是在一般的题目中，D可能较大甚至达到O(N)，这时这个算法的效率十分低。
- 如何改进基于边的分治的时间复杂度?

边分治

- 首先，我们试图改变选择边的标准，可惜这是改变不了算法的最坏时间复杂度的，当树的形态类似与如图所示时，无论选择哪条边， 结果都是一样的。
- 
- 注意到算法的复杂度分析的决定性因素是每个点的度数，我们猜 想是否可以通过等价的转换，使原图转化成一个每个点的度数是常数级别的新图呢？

边分治

- 每递归一层，就要花费O($nlogn$)的时间，总时间复杂度为O(递归层数$* \ nlogn$)。
- 最坏情况出现在菊花形数据上，此时需要递归n层，时间复杂度退化为O(n^2)。
- 为了避免退化，需要在原树中添加一些无用的黑点。



- 这样总点数不会超过$2n$，递归层数就是O($logn$)，时间复杂度O($nlog^2n$)。

边分治

- 这个转化给予了我们原树所没有的性质，那就是每个点的度至多为 3。
- 幸运的是，这个改变树结构的方法是可以推广的。白色结点代表 的是不影响结果的中间结点，我们可以用这个方法来解决一般的问题。

树链剖分

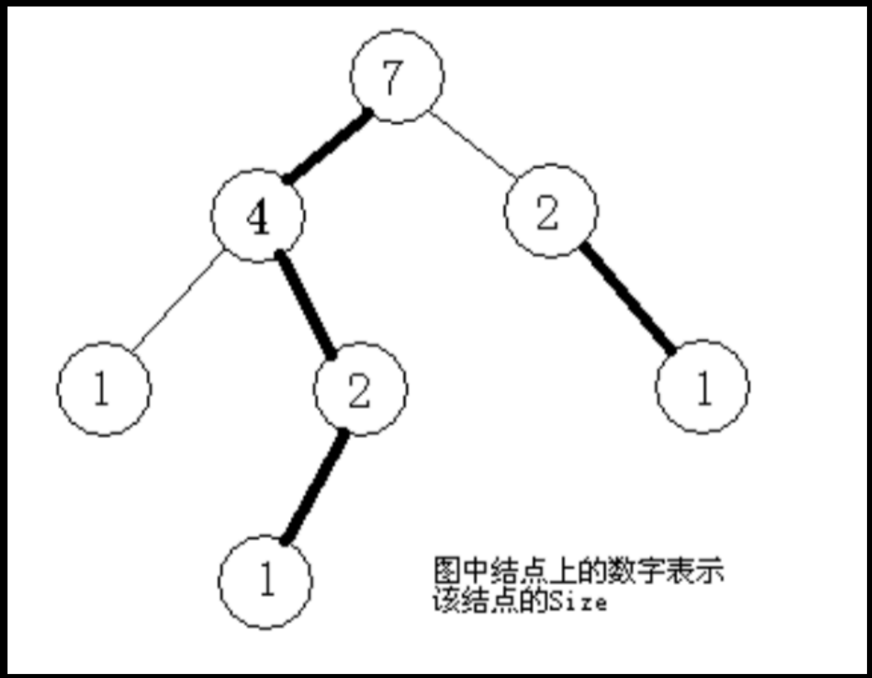
- SPOJ Query On a Tree 4
- 有一棵包含 N 个结点的树，每条边都有一个权值，要求模拟两种 操作:
  - 改变某条边的权值
  - 询问U,V 之间的路径中权值最大的边。 数据范围:
- N <=10000

树链剖分

- 引入路径剖分

- 考虑到虽然这颗树的边权在不断改变着，但树的形态并未改变， 因此考虑将这棵树的路径进行剖分，这里介绍一种在实践中常用的剖分方法: 轻重边路径剖分

树链剖分

- 我们将树中的边分为两类:轻边和重边。
- 

树链剖分

- 记Size(U)表示以U 为根的子树的结点个数，令V 为U 的儿子中Size 最大的一个，那么我们称边 (U ,V) 为重边，其余边为轻边。
- 显然轻重边路径剖分具有如下性质
  - 性质 1：如果(U,V)为轻边，则Size(V)<=Size(U)/2
  - 性质 2：从根到某一点的路径上轻边的个数不大于 O(log N)
  - 性质 3：我们称某条路径为重路径，当且仅当它全部由重边组成。那么对于每个点到根的路径上都不超过O(log N)条轻边和O(logN)条重路径。

树链剖分

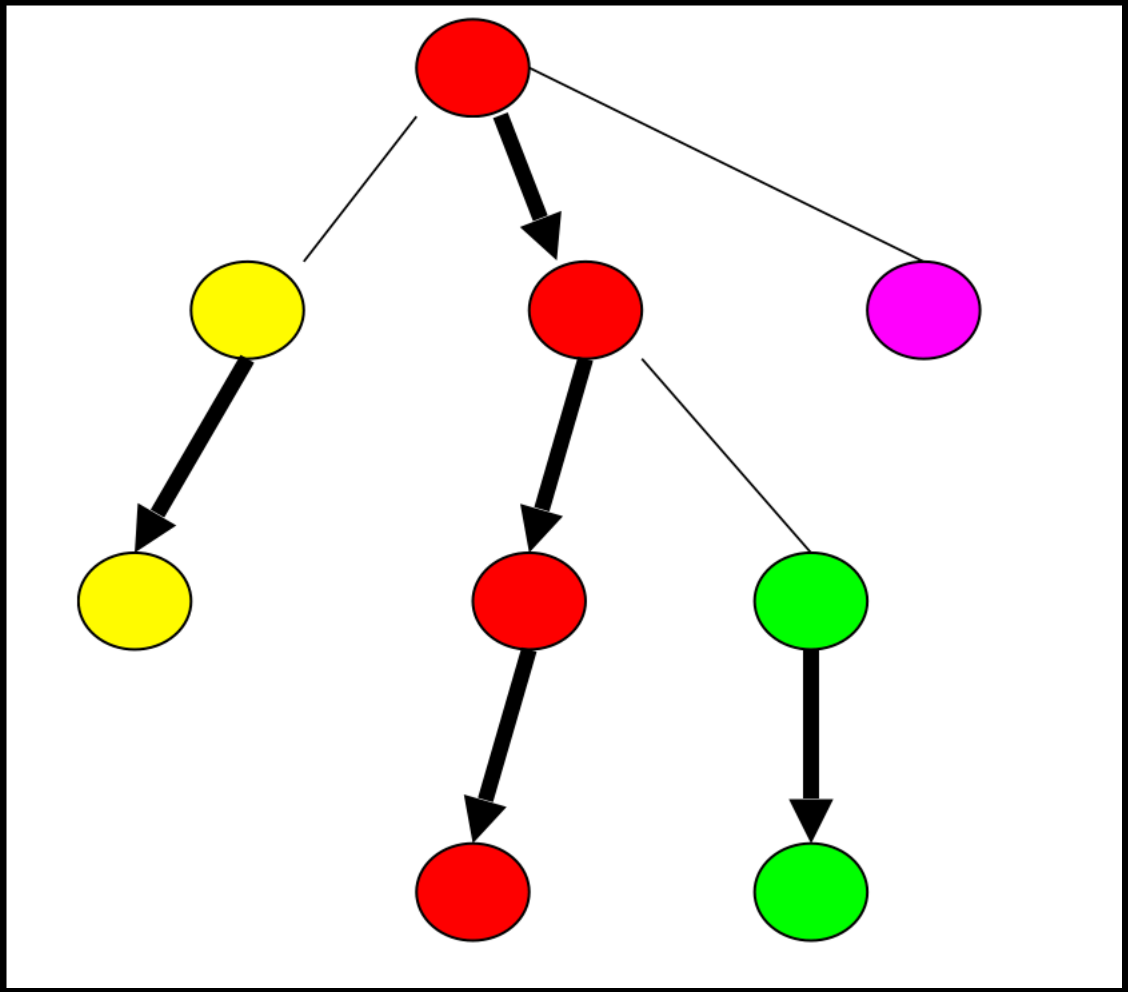
- 现在我们回到原题，对树进行轻重边路径剖分。对于询问操作， 我们可以分别处理两个点到其最近公共祖先的路径。根据性质 3，路径可以分解成最多 O(log N) 条轻边和 O(log N) 条重路径，那么只需考虑如何维护这两种对象。
- 对于轻边，我们直接处理即可。而对于重路径，我们只需用线段树来维护。这个算法对于两种操作的时间复杂度分别为O(logN)，O((logN)^2)，可以在时限内通过本题的所有数据了。

树链剖分

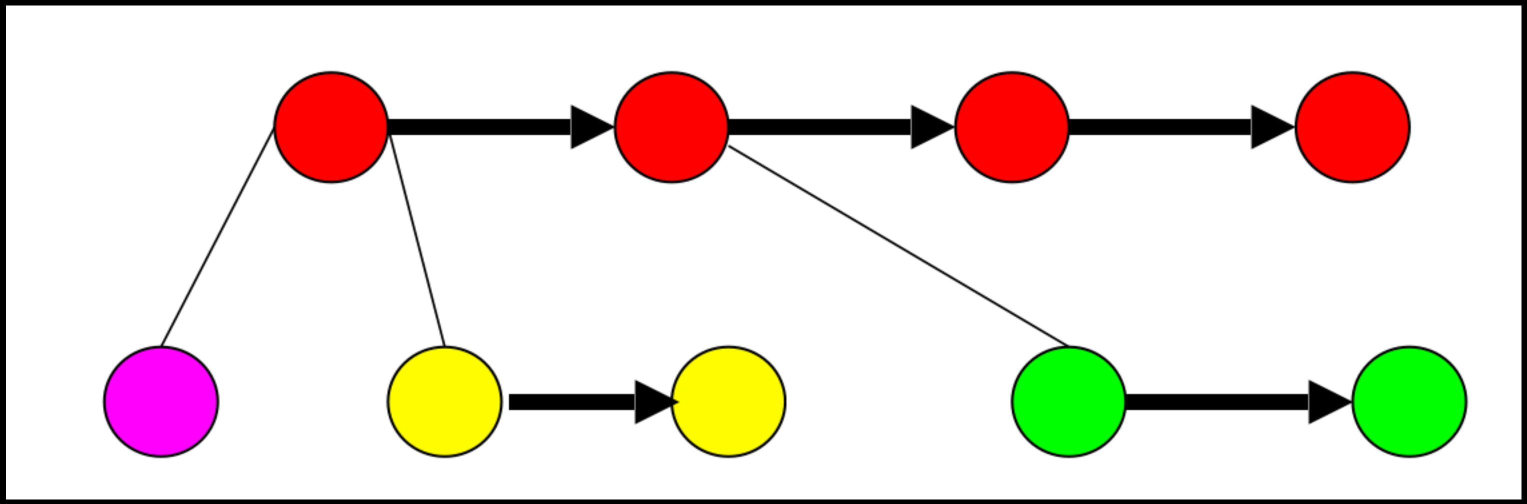
- 记siz[v]表示以v为根的子树的节点数，dep[v]表示v的深度(根深度为1)，top[v]表示v所在的重链的顶端节点，fa[v]表示v的父亲，son[v]表示与v在同一重链上的v的儿子节点（姑且称为重儿子），w[v]表示v与其父亲节点的连边（姑且称为v的父边）在线段树中的位置。只要把这些东西求出来，就能用logn的时间完成原问题中的操作。

  - 重儿子：siz[u]为v的子节点中siz值最大的，那么u就是v的重儿子。
    轻儿子：v的其它子节点。
    重边：点v与其重儿子的连边。
    轻边：点v与其轻儿子的连边。
    重链：由重边连成的路径。
    轻链：轻边。

树链剖分

- 怎样理解树链剖分也是一种树分治呢?
- 

树链剖分

- 按照点到根结点路径上的轻边个数分层摆放。
- 
- 所以路径剖分算法可以看做是基于链的分治
- 深度为 O(logn)

树链剖分

- Astar2008 黑白树
- 你拥有一棵有 N 个结点白色的树——所有节点都是白色的。 接下来，你需要处理 C 条指令:
  - 修改指令：改变一个给定结点的颜色(白变黑，黑变白);
  - 查询指令：询问从结点 1 到一个给定结点的路径上第一个 黑色结点编号。
- 数据范围：N <= 1000000，C <= 1000000

树链剖分

- 初看此题，我们感觉不是很好下手，“到一个给定结点的路径上 第一个黑色结点编号”似乎也没有什么特殊的性质，也很难找到一个 数据组织方式能够直接维护，这使得我们陷入了僵局。
- 由于本题中树的形态没有改变，且我们需要维护的对象是一个点 到根的路径，我们考虑使用路径剖分。
- 与上面一题不同的是，上题我们需要维护的是若干边的最大值，
- 而这题我们需要的维护的对象变成了点。
- 不过，我们仍然可以使用路径剖分，由路径的剖分方式可以知道
- 每个点都属于且仅属于一条重路径，所以我们只需考虑重路径，不需 要考虑轻边，这样比起上一题来说需要考虑的对象变少了。而维护重 路径相当于解决线性结构上的问题，使用堆或线段树都可以达到目 的。

树链剖分

- 回忆QTREE4
- N个点构成的树。每个点有黑白两种颜色。M个操作，两种类型。
- 一个点的颜色取反；
- 询问树上距离最远的两个黑点的距离。
- N，M <= 50000。

树链剖分

- 将路径剖分理解成基于链的分治后，我们可以用类似基于点的分治的方法将路径分类。
  - 与当前链有重合部分
  - 与当前链无重合部分（递归处理即可）

树链剖分

- 我们使用线段树来求解与当前链有重合部分的路径的最大长度

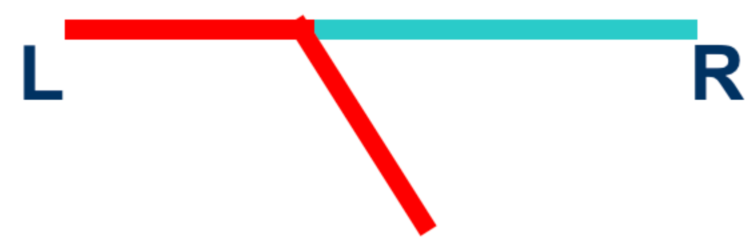
- 记D(i)表示第i个结点至子树内某个黑色结点的路径中长度的最大值，D2(i)表示第i个结点至子树内某个黑色结点的路径中长度的次大值。（两条路径仅在头结点处相交。如果至黑色结点的路径不存在，那么长度记为负无穷）

- Dist(i,j)表示链上的第i个点到第j个点的距离。

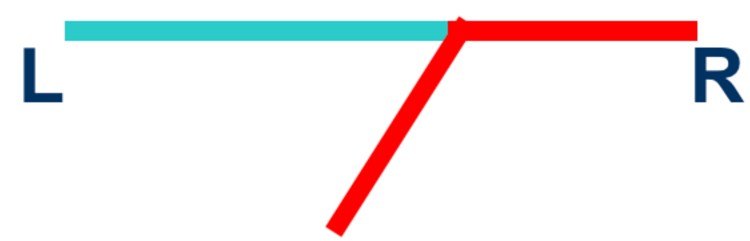
树链剖分

- 对于线段树中的一个区间[L,R],我们需要记录下面三个量

$$MaxL = Max\{Dist(L,i) + D(i)\}$$



$$MaxR = Max\{D(i) + Dist(i,R)\}$$



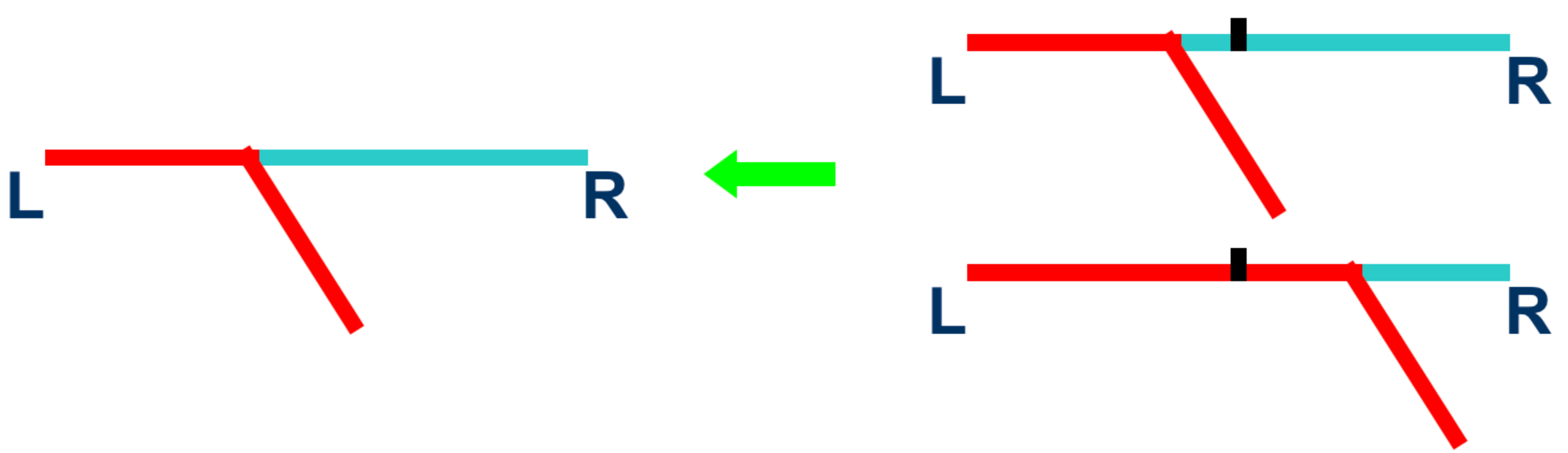
-

Opt = 与此链的重合部分在[L,R]的路径的最大长度

树链剖分

- 设区间[L,R]的结点编号为P，Lc,Rc分别表示P的左右两个儿子，区间[L,Mid]和[Mid+1,R]。我们可以得到如下转移：

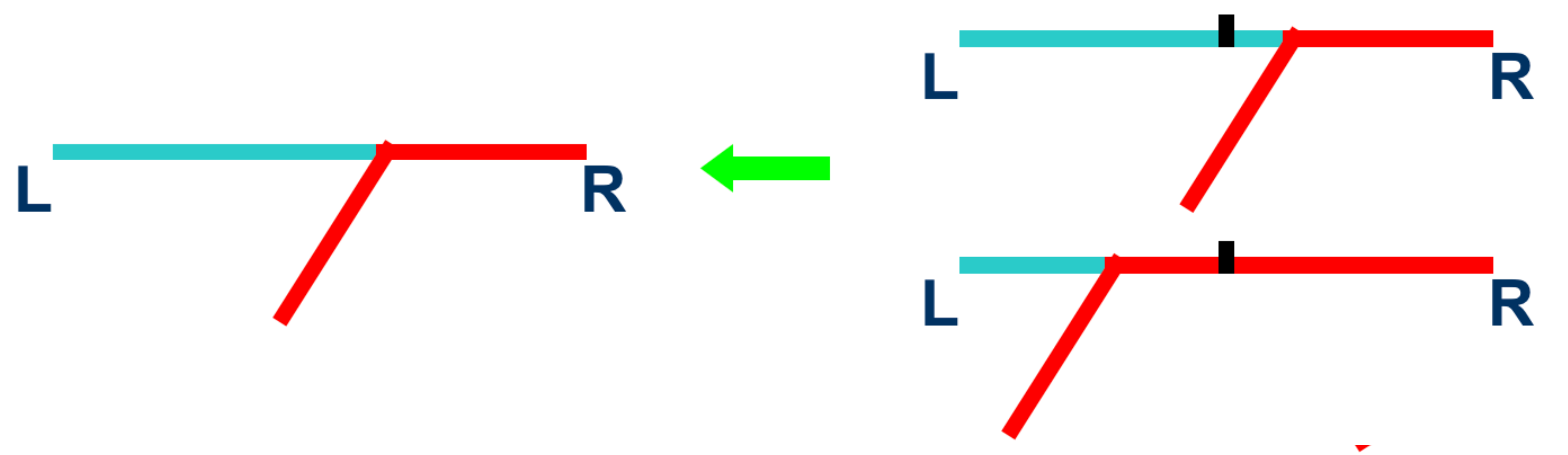
$$MaxL(P) = Max\{MaxL(Lc), Dist(L, Mid+1) + MaxL(Rc)\}$$

- 

树链剖分

- 设区间[L,R]的结点编号为P，Lc,Rc分别表示P的左右两个儿子，区间[L,Mid]和[Mid+1,R]。我们可以得到如下转移：

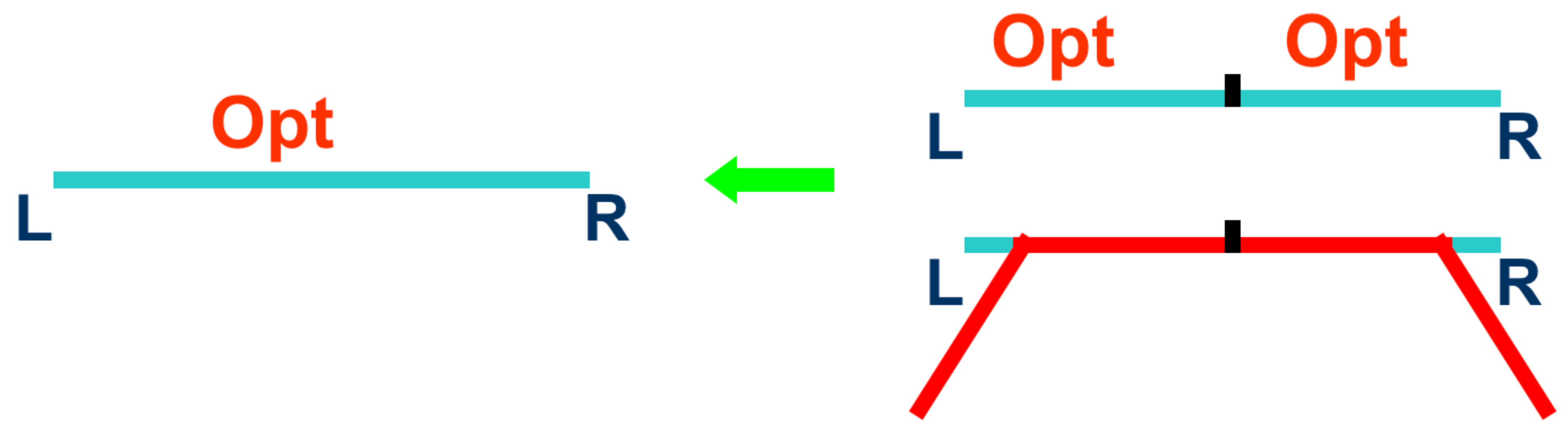
$$MaxR(P) = Max\{MaxR(Rc), MaxR(Lc) + Dist(Mid, R)\}$$

- 

树链剖分

- 设区间[L,R]的结点编号为P，Lc,Rc分别表示P的左右两个儿子，区间[L,Mid]和[Mid+1,R]。我们可以得到如下转移：

$$Opt(P) = Max\left\{ \begin{matrix} Opt(Lc), Opt(Rc), \\ MaxR(Lc) + MaxL(Rc) + Dist(Mid, Mid+1) \end{matrix} \right\}$$

- 

树链剖分

- $Dist(i,j) = Dist(1,j) - Dist(1,i)$
- 对于只有一个点 L 的边界情况
- $MaxL = D(L)$

  $MaxR = D(L)$

  $$Opt = \begin{cases} Max\{D(L)+D2(L), D(L)\} & \text{黑色} \\ D(L)+D2(L) & \text{白色} \end{cases}$$

树链剖分

- 我们记 C1…Ck 表示 x 的 k 个儿子(不包括同层结点)， Li 表示 Ci 所在的链的线段树根结点，Cost(p) 表示 (x, p) 的边权。那么点 x 向下至某个黑色结点的路径的长度集合为:
- $$\begin{array}{ll} \{MaxL(Li)+Cost(Ci),0\} & x\text{ 为黑色结点} \\ \{MaxL(Li)+Cost(Ci)\} & x\text{ 为白色结点} \end{array}$$
- 我们可以用堆来维护这个集合，这样 D(x), D2(x) 的获取就是 O(1) 的了。

树链剖分

- 复杂度分析：
  - 询问操作：我们使用堆来存贮每条链的最优结果
    - O(1)
  - 修改操作：修改一个点最多影响O(logN)条链，对于每条链我们需要修改堆和线段树
    - O((logN)^2)

树链剖分

- 题目：bzoj2243
- Description：给定一棵有n个节点的无根树和m个操作，操作有2类：1、将节点a到节点b路径上所有点都染成颜色c；2、询问节点a到节点b路径上的颜色段数量（连续相同颜色被认为是同一段），如“112221”由3段组成：“11”、“222”和“1”。请你写一个程序依次完成这m个操作。
- Input：第一行包含2个整数n和m，分别表示节点数和操作数；第二行包含n个正整数表示n个节点的初始颜色。下面每行包含两个整数x和y，表示x和y之间有一条无向边。下面每行描述一个操作：“C a b c”表示这是一个染色操作，把节点a到节点b路径上所有点（包括a和b）都染成颜色c；
- “Q a b”表示这是一个询问操作，询问节点a到节点b（包括a和b）路径上的颜色段数量。
- Output
- 对于每个询问操作，输出一行答案。
- 数N<=10^5，操作数M<=10^5，所有的颜色C为整数且在[0, 10^9]之间。