高级数据结构

崔奕凡

天津大学

2023 年 7 月 26 日

目录

1. 树链剖分
2. 主席树
3. k-D Tree
4. 可持久化 trie
5. 树套树
   - 线段树套线段树
   - 树状数组套权值线段树
6. The End

目录

1. 树链剖分
2. 主席树
3. k-D Tree
4. 可持久化 trie
5. 树套树
   - 线段树套线段树
   - 树状数组套权值线段树
6. The End

引入

概念

树链剖分用于将树分割成若干条链的形式，以维护树上路径的信息。

具体来说，将整棵树剖分为若干条链，使它组合成线性结构，然后用其他的数据结构维护信息。

树链剖分（树剖/链剖）有多种形式，如重链剖分，长链剖分和用于 Link/cut Tree 的剖分（有时被称作「实链剖分」），大多数情况下（没有特别说明时），「树链剖分」都指「重链剖分」。

引入

概念

重链剖分可以将树上的任意一条路径划分成不超过 $O(\log n)$ 条连续的链，每条链上的点深度互不相同（即是自底向上的一条链，链上所有点的 LCA 为链的一个端点）。

重链剖分还能保证划分出的每条链上的节点 DFS 序连续，因此可以方便地用一些维护序列的数据结构（如线段树）来维护树上路径的信息。如：

- 1. 修改树上两点之间的路径上所有点的值。
- 2. 查询树上两点之间的路径上节点权值的和/极值/其它（在序列上可以用数据结构维护，便于合并的信息）。

重链剖分

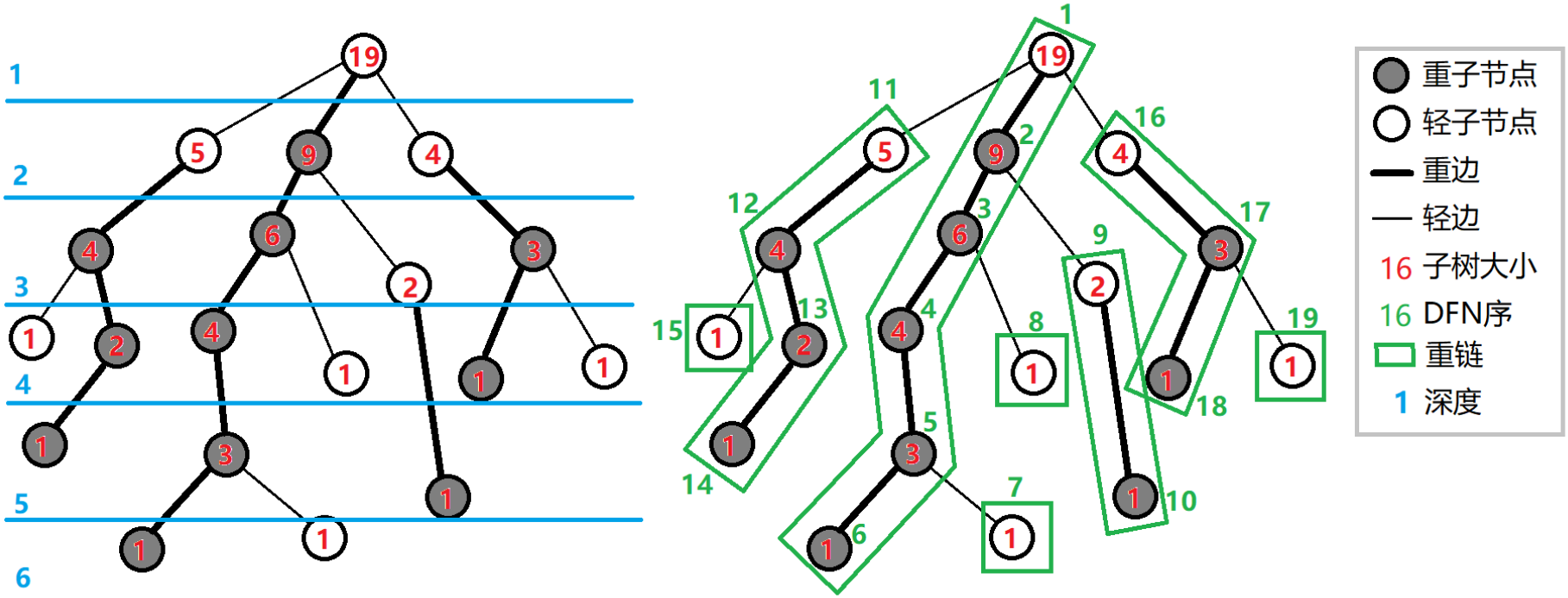
概念

我们给出一些定义：

- **重子节点**：表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。
- **轻子节点**：表示除重子节点剩余的所有子结点。
- **重边**：从点到重子节点的边。
- **轻边**：到其他轻子节点的边为轻边。
- **重链**：若干条首尾衔接的重边构成。

把落单的结点也当作重链，那么整棵树就被剖分成若干条重链。

重链剖分



代码实现两个 DFS

```
void dfs1(int o) {
  son[o] = -1;
  siz[o] = 1;
  for (int j = h[o]; j; j = nxt[j])
    if (!dep[p[j]]) {
      dep[p[j]] = dep[o] + 1;
      fa[p[j]] = o;
      dfs1(p[j]);
      siz[o] += siz[p[j]];
      if (son[o] == -1 || siz[p[j]] > siz[son[o]]) son[o] = p[j];
    }
}

void dfs2(int o, int t) {
  top[o] = t;
  cnt++;
  dfn[o] = cnt;
  rnk[cnt] = o;
  if (son[o] == -1) return;
  dfs2(son[o], t);  // 优先对重儿子进行 DFS，可以保证同一条重链上的点 DFS 序连续
  for (int j = h[o]; j; j = nxt[j])
    if (p[j] != son[o] && p[j] != fa[o]) dfs2(p[j], p[j]);
}
```

变量表示

- top(x) 表示节点 x 所在重链的顶部节点（深度最小）。
- dfn(x) 表示节点 x 的 DFS 序，也是其在线段树中的编号。
- rnk(x) 表示 DFS 序所对应的节点编号，有 rnk(dfn(x))=x。

重链剖分

性质

- 树上每个节点都属于且仅属于一条重链。
- 所有的重链将整棵树完全剖分。
- 在剖分时重边优先遍历，最后树的 DFS 序上，重链内的 DFS 序是连续的。按 DFN 排序后的序列即为剖分后的链。
- 一颗子树内的 DFS 序是连续的。
- 可以发现，当我们向下经过一条轻边时，所在子树的大小至少会除以二。
- 因此，对于树上的任意一条路径，把它拆分成从 LCA 分别向两边往下走，分别最多走 $O(\log n)$ 次，因此，树上的每条路径都可以被拆分成不超过 $O(\log n)$ 条重链。

常见应用

路径上维护

链上的 DFS 序是连续的，可以使用线段树、树状数组维护。每次选择深度较大的链往上跳，直到两点在同一条链上。同样的跳链结构适用于维护、统计路径上的其他信息。

子树维护

有时会要求，维护子树上的信息，譬如将以 x 为根的子树的所有结点的权值增加 v。

在 DFS 搜索的时候，子树中的结点的 DFS 序是连续的。

每一个结点记录 *bottom* 表示所在子树连续区间末端的结点。

这样就把子树信息转化为连续的一段区间信息。

常见应用

```
int lca(int u, int v) {
  while (top[u] != top[v]) {
    if (dep[top[u]] > dep[top[v]])
      u = fa[top[u]];
    else
      v = fa[top[v]];
  }
  return dep[u] > dep[v] ? v : u;
}
```

求最近公共祖先

不断向上跳重链，当跳到同一条重链上时，深度较小的结点即为 LCA。向上跳重链时需要先跳所在重链顶端深度较大的那个。

例题

链接

「ZJOI2008」树的统计
https://loj.ac/problem/10138

Description

对一棵有 n 个节点，节点带权值的静态树，进行三种操作共 q 次：

- 修改单个节点的权值；
- 查询 u 到 v 的路径上的最大权值；
- 查询 u 到 v 的路径上的权值之和。

保证 $1 \leq n \leq 30000$ ， $0 \leq q \leq 200000$ 。

例题

题解

根据题面以及以上的性质，你的线段树需要维护三种操作：单点修改；区间查询最大值；区间查询和。

单点修改很容易实现。

由于子树的 DFS 序连续（无论是否树剖都是如此），修改一个节点的子树只用修改这一段连续的 DFS 序区间。

例题

题解

问题是如何修改/查询两个节点之间的路径。

考虑我们是如何用倍增法求解 LCA 的。首先我们将两个节点提到同一高度，然后将两个节点一起向上跳。对于树链剖分也可以使用这样的思想。

在向上跳的过程中，如果当前节点在重链上，向上跳到重链顶端，如果当前节点不在重链上，向上跳一个节点。如此直到两节点相同。沿途更新/查询区间信息。

对于每个询问，最多经过 $O(\log n)$ 条重链，每条重链上线段树的复杂度为 $O(\log n)$，因此总时间复杂度为 $O(n \log n + q \log^2 n)$。实际上重链个数很难达到 $O(\log n)$（可以用完全二叉树卡满），所以树剖在一般情况下常数较小。

目录

1 树链剖分

2 主席树

3 k-D Tree

4 可持久化 trie

5 树套树
线段树套线段树
树状数组套权值线段树

6 The End

引入

概念

主席树是一种可持久化的数据结构。

想要让线段树可持久化，最朴素的实现方法是每进行一次操作都建一棵新的树，这样做的时间和空间复杂度都是不可接受的。

稍微思考一下会发现，每次修改操作都只会有少数的点被修改，所以大量的点是可以共用的。主席树的核心就是利用共用的点来减少时间和空间复杂度。

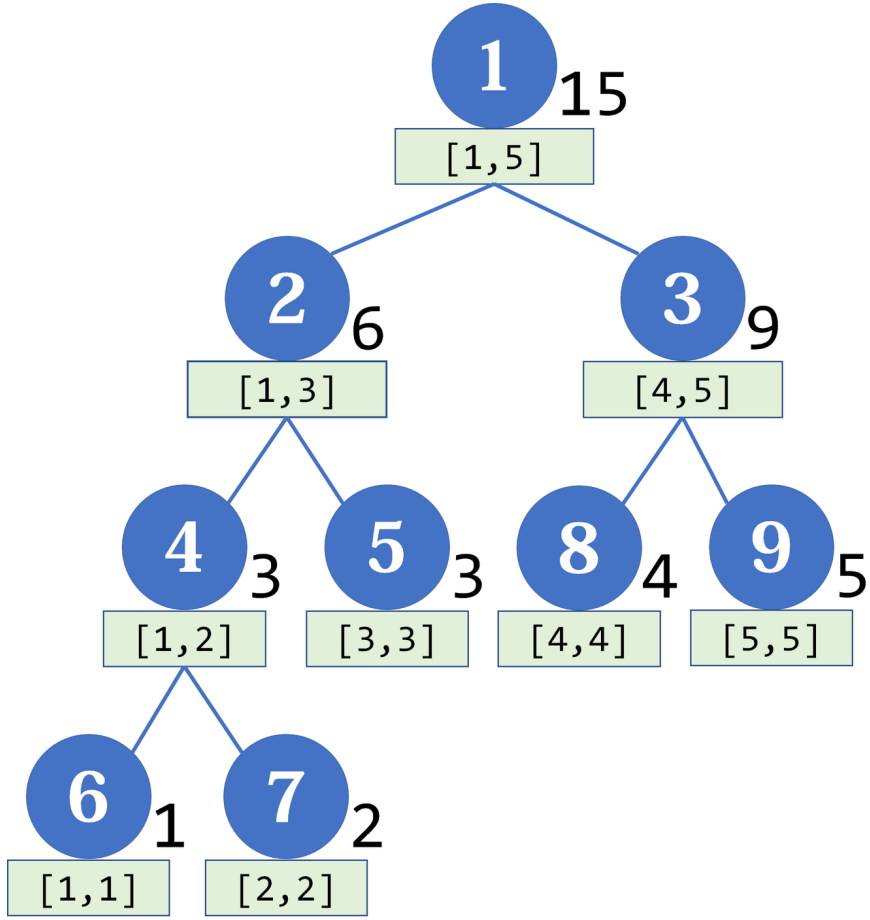
原理实现

建树

我们用动态开点的方法存储每个点的左右子节点。不过，对于最初版本的那棵树，我们应当一次性建好，而不必一个一个插入：

```
void build(int l = 1, int r = n, int p = 1)
{
    if (l == r)
        val(p) = A[l];
    else
    {
        ls(p) = ++cnt, rs(p) = ++cnt; // 新建节点
        int mid = (l + r) / 2;
        build(l, mid, ls(p));
        build(mid + 1, r, rs(p));
        val(p) = val(ls(p)) + val(rs(p));
    }
}
```

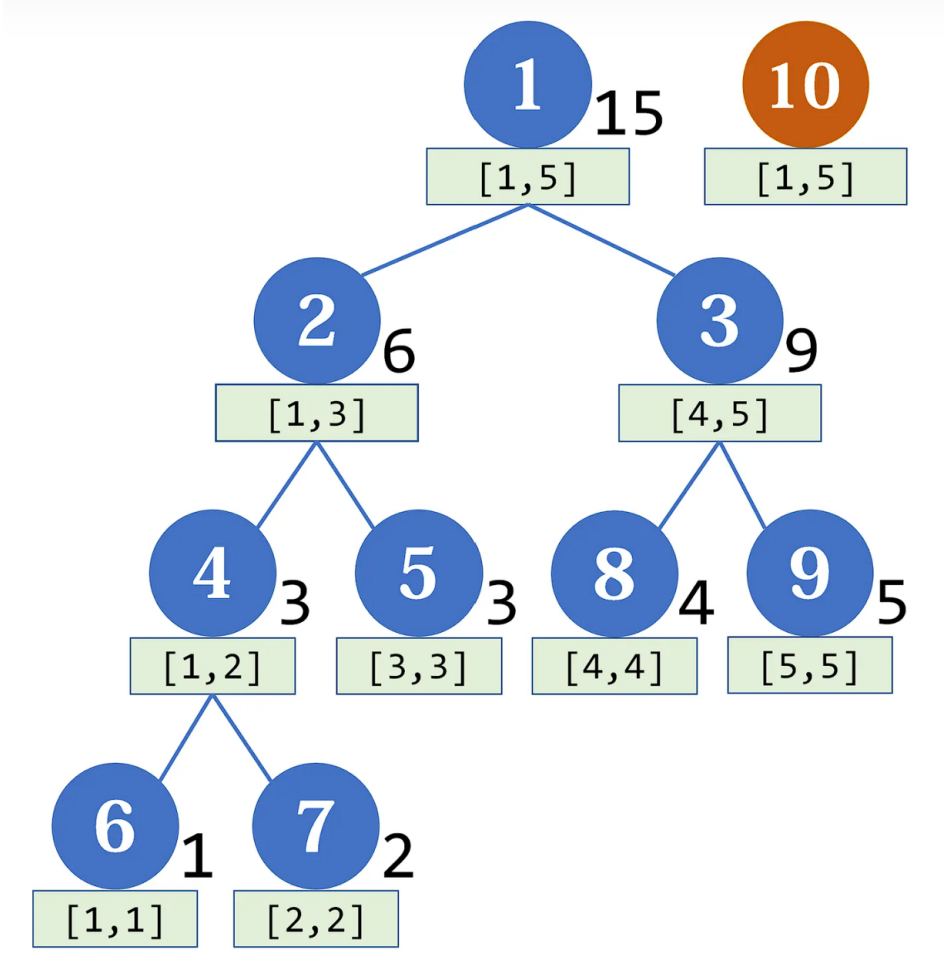
原理实现



示例

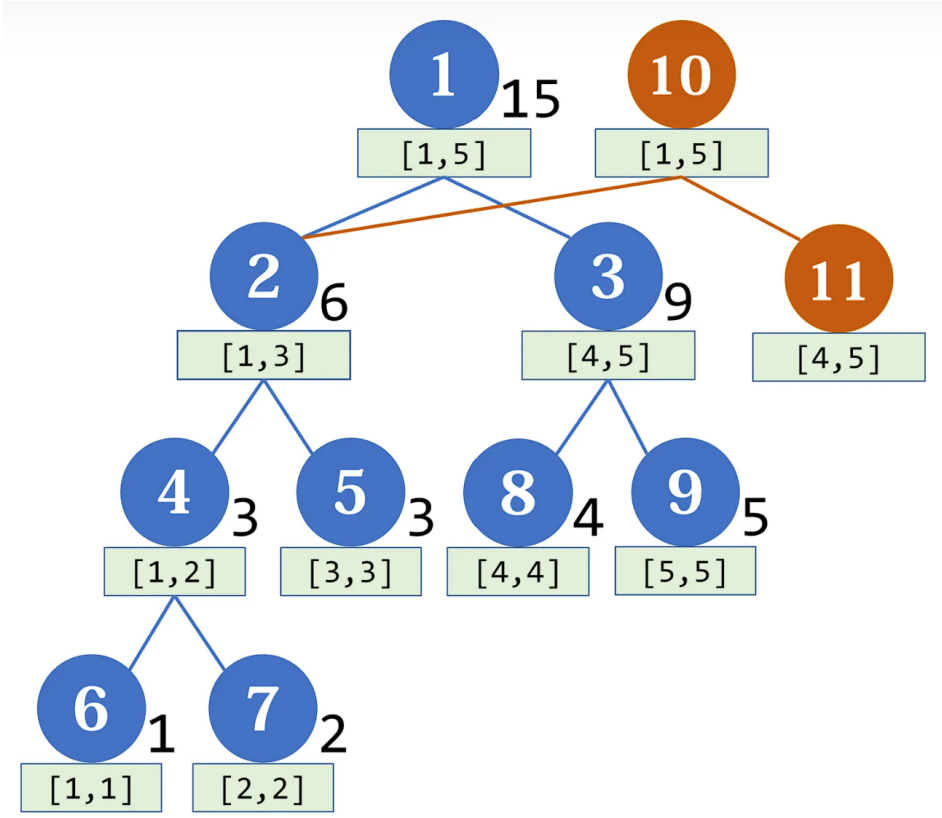
我们维持这棵树不变，添加额外的节点来代替修改操作。
假如原始数据为 [1,2,3,4,5]，现在我们有了形如这样的一棵线段树：

原理实现



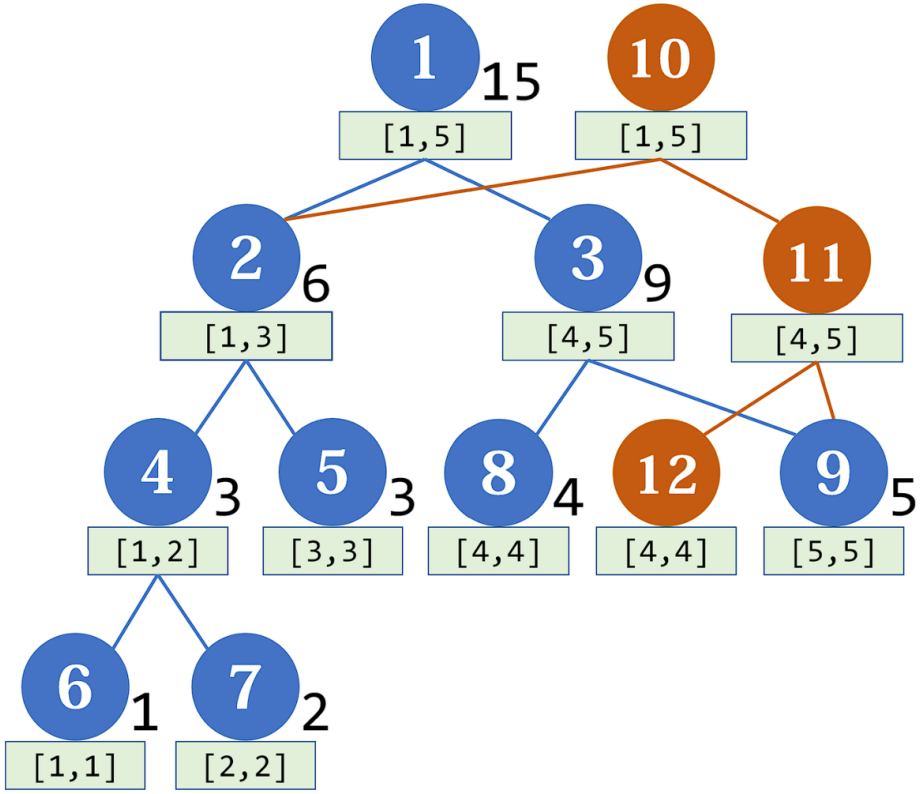
现在进行单点修改操作，我们要令 A_4 加 2，先建一个新的根节点

原理实现



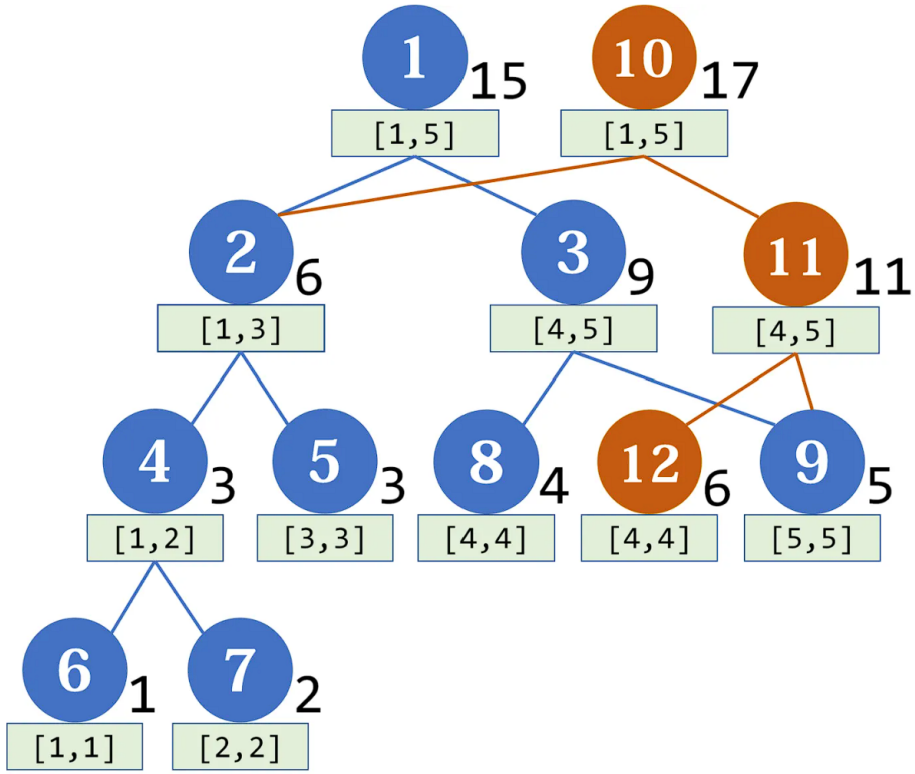
要修改的 A_4 对应的叶子节点在右子树上。所以，这个新节点的左儿子应该与原节点的左儿子相同（即与之共用左子树）。再创建一个新的节点作为右儿子。

原理实现



接下来处理这个新节点（这是一个递归的过程）。显然 A_4 应该在左子树，所以相似地，沿用原树的右儿子，然后创建一个新的节点作为左儿子。

原理实现



接下来给各个节点赋值，和普通的线段树类似，叶子节点直接赋值，其它节点则利用子节点计算值。

原理实现

```
// 令A_x加d，p表示原版本的节点，q表示新版本的节点
void update(int x, int d, int p, int q, int cl = 1, int cr = n)
{
    if (cl == cr)
        val(q) = val(p) + d; // 给叶子节点赋值
    else
    {
        ls(q) = ls(p), rs(q) = rs(p); // 复制节点
        int mid = (cl + cr) / 2;
        if (x <= mid)
            ls(q) = ++cnt, update(x, d, ls(p), ls(q), cl, mid); // 创建新节点作为左儿子，然后往左递归
        else
            rs(q) = ++cnt, update(x, d, rs(p), rs(q), mid + 1, cr); // 创建新节点作为右儿子，然后往右递归
        val(q) = val(ls(q)) + val(rs(q)); // 根据子节点给当前节点赋值
    }
}
```

这样就完成了单点修改。事实上我们没有修改任何东西，我们保留了原来的版本，并新增了一个修改后的版本。查询操作和普通的线段树查询操作是一样的，只不过需要指定从哪个根节点开始查询。

例题

题目大意

给定 n 个整数构成的序列 a，将对于指定的闭区间 $[l, r]$ 查询其区间内的第 k 小值。

链接

P3834 【模板】可持久化线段树 2
https://www.luogu.com.cn/problem/P3834

例题

题解

这是一道经典的主席树问题。

我们在序列 a 值域上建立权值线段树，为序列上的每一个位置动态开点。例如在点 a_i 我们维护的实际上是区间 $[1, i]$ 的权值线段树的和，即前缀和。

每次查询时，取出 $root[l-1]$ 与 $root[r]$ ，查询时每个结点进行 $sum[root[r]] - sum[root[l-1]]$ 的差操作，能得到查询区间 $[l, r]$ 上这个结点的真实 sum 。

然后在这个作差的权值线段树上二分查找第 k 小值。

目录

1 树链剖分

2 主席树

3 k-D Tree

4 可持久化 trie

5 树套树
线段树套线段树
树状数组套权值线段树

6 The End

引入

概念

k-D Tree(KDT，k-Dimension Tree) 是一种可以高效处理 k 维空间信息的数据结构。

在结点数 n 远大于 2^k 时，应用 k-D Tree 的时间效率很好。

在算法竞赛的题目中，一般有 $k = 2$ 。在本页面分析时间复杂度时，将认为 k 是常数。

建树

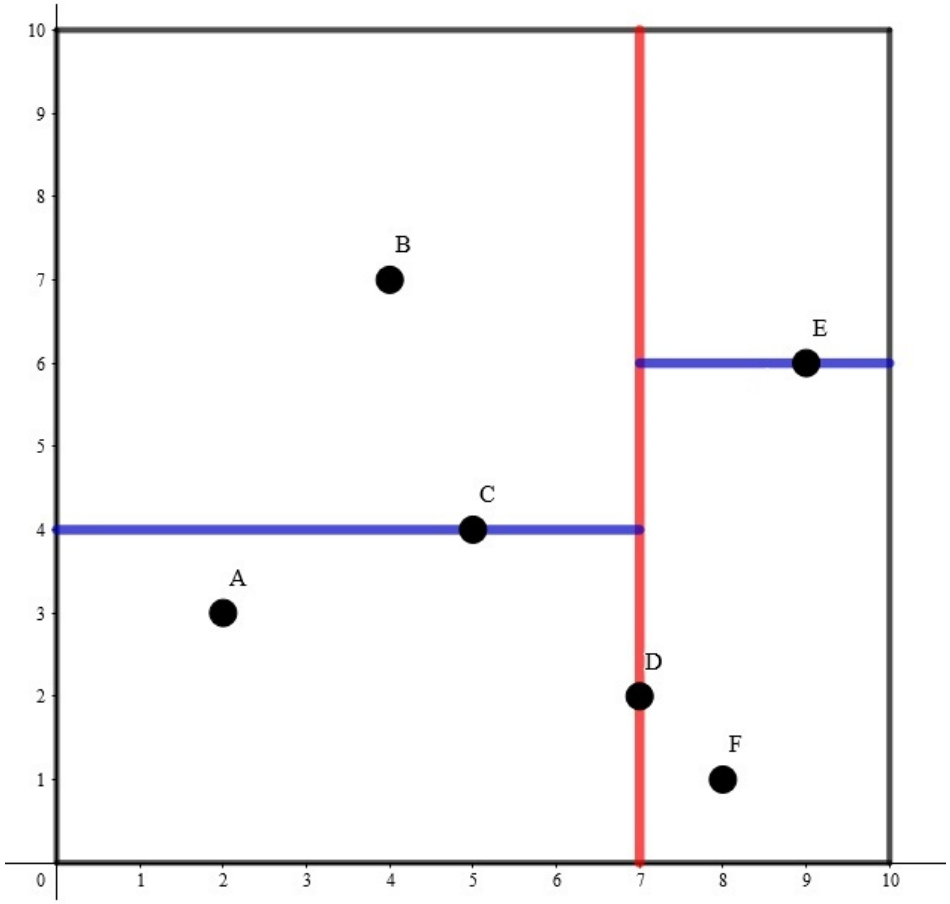
性质

k-D Tree 具有二叉搜索树的形态，二叉搜索树上的每个结点都对应 k 维空间内的一个点。
其每个子树中的点都在一个 k 维的超长方体内，这个超长方体内的所有点也都在这个子树中。

假设我们已经知道了 k 维空间内的 n 个不同的点的坐标，要将其构建成一棵 k-D Tree，步骤如下：

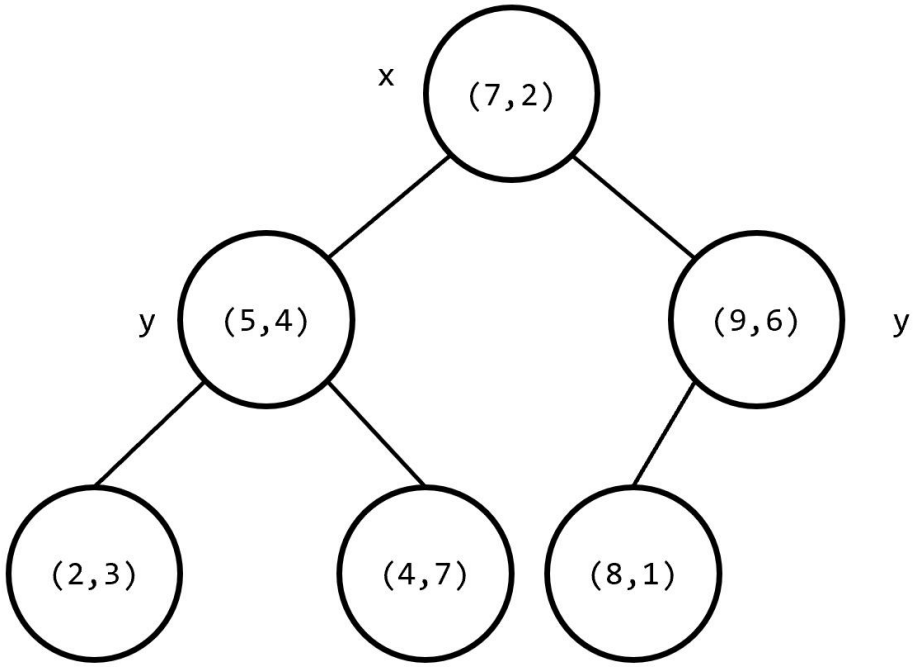
- 若当前超长方体中只有一个点，返回这个点。
- 选择一个维度，将当前超长方体按照这个维度分成两个超长方体。
- 选择切割点：在选择的维度上选择一个点，这一维度上的值小于这个点的归入一个超长方体（左子树），其余的归入另一个超长方体（右子树）。
- 将选择的点作为这棵子树的根节点，递归对分出的两个超长方体构建左右子树，维护子树的信息。

建树



为了方便理解，我们举一个 k=2 时的例子。

建树



其构建出 k-D Tree 的形态可能是这样的：

建树优化和时间复杂度

其中树上每个结点上的坐标是选择的分割点的坐标，非叶子结点旁的 x 或 y 是选择的切割维度。

这样的复杂度无法保证。对于 2,3 两步，我们提出两个优化：

- 选择的维度要满足其内部点的分布的差异度最大，即每次选择的切割维度是方差最大的维度。
- 每次在维度上选择切割点时选择该维度上的中位数，这样可以保证每次分成的左右子树大小尽量相等。

建树优化和时间复杂度

可以发现，使用优化 2 后，构建出的 k-D Tree 的树高最多为 $O(\log n)$。

现在，构建 k-D Tree 时间复杂度的瓶颈在于快速选出一个维度上的中位数，并将在该维度上的值小于该中位数的置于中位数的左边，其余置于右边。

如果每次都使用 sort 函数对该维度进行排序，时间复杂度是 $O(n \log^2 n)$ 的。事实上，单次找出 n 个元素中的中位数并将中位数置于排序后正确的位置的复杂度可以达到 $O(n)$。

建树优化和时间复杂度

由于 k-D Tree 只要求要中位数在排序后正确的位置上，所以我们只需要递归排序包含中位数的一侧。可以证明，这样的期望复杂度是 $O(n)$ 的。

在 algorithm 库中，有一个实现相同功能的函数 $nth\_element()$，只需写 $nth\_element(s+l, s+mid, s+r+1, cmp)$ 就可实现将排序后的 mid 的值放在 $s[mid]$ 处。

借助这种思想，构建 k-D Tree 时间复杂度是 $O(n \log n)$ 的。

插入/删除

如果维护的这个 k 维点集是可变的，即可能会插入或删除一些点，此时 k-D Tree 的平衡性无法保证。

由于 k-D Tree 的构造，不能支持旋转，类似与 FHQ Treap 的随机优先级也不能保证其复杂度，可以保证平衡性的手段只有类似于替罪羊树的重构思想。

我们引入一个重构常数 α。

对于 k-D Tree 上的一个结点 x，若其有一个子树的结点数在以 x 为根的子树的结点数中的占比大于 α，则认为以 x 为根的子树是不平衡的，需要重构。

重构时，先遍历子树求出一个序列，然后用以上描述的方法建出一棵 k-D Tree，代替原来不平衡的子树。

邻域查询

链接

P1429 平面最近点对
https://www.luogu.com.cn/problem/P1429

Description

给定平面上的 n 个点 (x_i, y_i)，
找出平面上最近两个点对之间的欧几里得距离。
$2 \leq n \leq 2 \times 10^5$
$0 \leq x_i, y_i \leq 10^9$

邻域查询

题解

首先建出关于这 n 个点的 2-D Tree。

枚举每个结点，对于每个结点找到不等于该结点且距离最小的点，即可求出答案。每次暴力遍历 2-D Tree 上的每个结点的时间复杂度是 $O(n)$ 的，需要剪枝。

我们可以维护一个子树中的所有结点在每一维上的坐标的最小值和最大值。

邻域查询

题解

假设当前已经找到的最近点对的距离是 ans，如果查询点到子树内所有点都包含在内的长方形的最近距离大于等于 ans，则在这个子树内一定没有答案，搜索时不进入这个子树。

此外，还可以使用一种启发式搜索的方法，即若一个结点的两个子树都有可能包含答案，先在与查询点距离最近的一个子树中搜索答案。

可以认为，查询点到子树对应的长方形的最近距离就是此题的估价函数。

注意：虽然以上使用的种种优化，但是使用 k-D Tree 单次查询最近点的时间复杂度最坏还是 $O(n)$ 的，使用时请注意。

高维空间上的操作

链接

P4148 简单题
https://www.luogu.com.cn/problem/P4148

Description

在一个初始值全为 0 的 $n \times n$ 的二维矩阵上，进行 q 次操作，每次操作为以下两种之一：

- 1. x y A：将坐标 (x,y) 上的数加上 A。
- 2. x1 y1 x2 y2：输出以 (x1,y1) 为左下角，(x2,y2) 为右上角的矩形内（包括矩形边界）的数字和。

强制在线。内存限制 20M。保证答案及所有过程量在 int 范围内。
$1 \leq n \leq 500000, 1 \leq q \leq 200000$

高维空间上的操作

题解

20M 的空间卡掉了所有树套树，强制在线卡掉了 CDQ 分治，只能使用 k-D Tree。

构建 2-D Tree，支持两种操作：添加一个 2 维点；查询矩形区域内的所有点的权值和。可以使用带重构的 k-D Tree 实现。

高维空间上的操作

题解

在查询矩形区域内的所有点的权值和时，仍然需要记录子树内每一维度上的坐标的最大值和最小值。

- 1. 如果当前子树对应的矩形与所求矩形没有交点，则不继续搜索其子树；
- 2. 如果当前子树对应的矩形完全包含在所求矩形内，返回当前子树内所有点的权值和；
- 3. 否则，判断当前点是否在所求矩形内，更新答案并递归在左右子树中查找答案。

如果在 2-D 树上进行矩阵查询操作，已经被完全覆盖的子树不会继续查询，则单次查询时间复杂度是最优 $O(\log n)$ ，最坏 $O(\sqrt{n})$ 的。

将结论扩展到 k 维的情况，则最坏时间复杂度是 $O(n^{1-\frac{1}{k}})$ 的。

目录

1. 树链剖分
2. 主席树
3. k-D Tree
4. 可持久化 trie
5. 树套树
   - 线段树套线段树
   - 树状数组套权值线段树
6. The End

引入

概念

可持久化 Trie 的方式和可持久化线段树的方式是相似的.

即每次只修改被添加或值被修改的节点，而保留没有被改动的节点，在上一个版本的基础上连边，使最后每个版本的 Trie 树的根遍历所能分离出的 Trie 树都是完整且包含全部信息的。

大部分的可持久化 Trie 题中，Trie 都是以 01-Trie 的形式出现的。

例题

链接

P4735 最大异或和
https://www.luogu.com.cn/problem/P4735

Description

对一个长度为 n 的数组 a 维护以下操作：

- 1. 在数组的末尾添加一个数 x，数组的长度 n 自增 1。
- 2. 给出查询区间 [l,r] 和一个值 k，求当 $l \le p \le r$ 时，$k \oplus \bigoplus_{i=p}^{n} a_i$ 的最大值。

求解过程

题解

这个求的值可利用常用的处理连续异或的方法，记 $s_x = \bigoplus_{i=1}^{x} a_i$ ，则原式等价于 $s_{p-1} \oplus s_n \oplus k$ ，

观察到 $s_n \oplus k$ 在查询的过程中是固定的，题目的查询变化为查询在区间 [l-1,r-1] 中异或定值 $s_n \oplus k$ 的最大值。

求解过程

继续按类似于可持久化线段树的思路，考虑每次的查询都查询整个区间。

我们只需把这个区间建一棵 Trie 树，将这个区间中的每个树都加入这棵 Trie 中，查询的时候，尽量往与当前位不相同的地方跳。

查询区间，只需要利用前缀和和差分的思想，用两棵前缀 Trie 树（也就是按顺序添加数的两个历史版本）相减即得到该区间的 Trie 树。

再利用动态开点的思想，不添加没有计算过的点，以减少空间占用。

目录

1. 树链剖分
2. 主席树
3. k-D Tree
4. 可持久化 trie
5. **树套树**
   - **线段树套线段树**
   - **树状数组套权值线段树**
6. The End

实现原理

考虑用树套树如何实现在二维平面上进行单点修改，区域查询。

考虑外层的线段树，最底层的 1 到 n 个节点的子树，分别代表第 1 到第 n 行的线段树。

那么这些底层的节点对应的父节点，就代表其两个子节点的子树所在的一片区域。

性质

空间复杂度

通常情况下，我们不可能对于外层线段树的每一个结点都建立一颗子线段树，空间需求过大。树套树一般采取动态开点的策略。

单次修改，我们会涉及到外层线段树的 $\log n$ 个节点，且对于每个节点的子树涉及 $\log n$ 个节点，所以单次修改产生的空间最多为 $\log^2 n$。

时间复杂度

对于询问操作，我们考虑我们在外层线段树上进行 $\log n$ 次操作，每次操作会在一个内层线段树上进行 $\log n$ 次操作，所以时间复杂度为 $\log^2 n$ 。

修改操作，与询问操作复杂度相同，也为 $\log^2 n$ 。

例题

链接

陌上花开
https://www.luogu.com.cn/problem/P3810

Description

有 n 个元素，第 i 个元素有 a_i, b_i, c_i 三个属性，设 $f(i)$ 表示满足 $a_j \le a_i$ 且 $b_j \le b_i$ 且 $c_j \le c_i$ 且 $j \ne i$ 的 j 的数量。
对于 $d \in [0, n)$，求 $f(i) = d$ 的数量。
$1 \le n \le 10^5$，$1 \le a_i, b_i, c_i \le k \le 2 \times 10^5$。

例题

题解

将第一维排序处理，然后用树套树维护第二维和第三维。

面对多维度信息的题目时，如果题目没有要求强制在线，

我们也还可以考虑 CDQ 分治，或者整体二分等分治算法，来避免使用高级数据结构，减少代码实现难度。

例题

链接

二逼平衡树（树套树）
https://loj.ac/p/106

Description

这是一道模板题。
您需要写一种数据结构（可参考题目标题），来维护一个有序数列，其中需要提供以下操作：

- 查询 x 在区间内的排名；
- 查询区间内排名为 k 的值；
- 修改某一位置上的数值；
- 查询 x 在区间内的前驱（前驱定义为小于 x，且最大的数）；
- 查询 x 在区间内的后继（后继定义为大于 x，且最小的数）。

例题

题解

如果用线段树套平衡树，即对于线段树的每一个节点，对于其所表示的区间维护一个平衡树，然后用二分来查找 k 小值。

由于每次查询操作都要覆盖多个区间，即有多个节点，但是平衡树并不能多个值一起查找，所以时间复杂度是 $O(n \log^3 n)$ ，并不是最优的。

例题

题解

优化的思路是把二分答案的操作和查询小于一个值的数的数量两种操作结合起来，使用线段树套动态开点权值线段树，由于所有线段树的结构是相同的，可以在多棵树上同时进行线段树上二分。

在修改操作进行时，先在线段树上从上往下跳到被修改的点，删除所经过的点所指向的动态开点权值线段树上的原来的值，然后插入新的值，

要经过 $O(\log n)$ 个线段树上的节点，在动态开点权值线段树上一次修改操作是 $O(\log n)$ 的，所以修改操作的时间复杂度为 $O(\log^2 n)$ 。

例题

题解

在查询答案时，先取出该区间覆盖在线段树上的所有点，然后用类似于静态区间 k 小值的方法，将这些点一起向左儿子或向右儿子跳。

如果所有这些点左儿子存储的值大于等于 k，则往左跳，否则往右跳。

由于最多只能覆盖 $O(\log n)$ 个节点，所以最多一次只有这么多个节点向下跳，时间复杂度为 $O(\log^2 n)$ 。

由于线段树的常数较大，在实现中往往使用常数更小且更方便处理前缀和的树状数组实现。另外空间复杂度是 $O(n \log^2 n)$ 的，使用时注意空间限制。

目录

1 树链剖分

2 主席树

3 k-D Tree

4 可持久化 trie

5 树套树
线段树套线段树
树状数组套权值线段树

6 The End

Thank you! Any Question?