点分治 & 动态点分治

性质

一类将问题分治解决的算法。

一般的流程都是：

> - 找重心。
> - 依次遍历每个子树：先统计答案。当某点与维护的状态集满足互补关系（即符合条件）就更新答案。当遍历完一次子树后，将子树的状态添加到状态集中。
> - 遍历子树删除状态集。
> - 递归子树。

动态点分治

动态点分治用来解决 **带点权/边权修改** 的树上路径信息统计问题。

点分树

回顾点分治的计算过程。

对于一个结点 x 来说，其子树中的简单路径包括两种：经过结点 x 的，由一条或两条从 x 出发的路径组成的；和不经过结点 x 的，即已经包含在其所有儿子结点子树中的路径。

对于一个子树中简单路径的计算，我们选择一个分治中心 rt，计算经过该节点的子树中路径的信息，然后对于其每个儿子结点，将删去 rt 后该点所在连通块作为一个子树，递归计算。选择的分治中心点可以构成一个树形结构，称为 **点分树**。我们发现，计算点分树中同一层的结点所代表的连通块（即以该结点为分治中心的连通块）的大小总和是 $O(n)$ 的。这意味着，点分治的时间复杂度是与点分树的深度相关的，若点分树的深度为 h，则点分治的复杂度为 $O(nh)$。

可以证明，当我们每次选择连通块的重心作为分治中心的时候，点分树的深度最小，为 $O(\log n)$ 的。这样，我们就可以在 $O(n \log n)$ 的时间复杂度内统计树上 $O(n^2)$ 条路径的信息了。

由于树的形态在动态点分治的过程中不会改变，所以点分树的形态在动态点分治的过程中也不会改变。

下面给出求点分树的参考代码：

```
void calcsiz(int x, int f) {
  siz[x] = 1;
  maxx[x] = 0;
  for (int j = h[x]; j; j = nxt[j])
    if (p[j] != f && !vis[p[j]]) {
      calcsiz(p[j], x);
      siz[x] += siz[p[j]];
      maxx[x] = max(maxx[x], siz[p[j]]);
    }
  maxx[x] =
      max(maxx[x], sum - siz[x]);  // maxx[x] 表示以 x 为根时的最大子树大小
  if (maxx[x] < maxx[rt])
    rt = x;  // 这里不能写 <= ，保证在第二次 calcsiz 时 rt 不改变
}
```

```
void pre(int x) {
  vis[x] = true;  // 表示在之后的过程中不考虑 x 这个点
  for (int j = h[x]; j; j = nxt[j])
    if (!vis[p[j]]) {
      sum = siz[p[j]];
      rt = 0;
      maxx[rt] = inf;
      calcsiz(p[j], -1);
      calcsiz(rt, -1);  // 计算两次，第二次求出以 rt 为根时的各子树大小
      fa[rt] = x;
      pre(rt);  // 记录点分树上的父亲
    }
}
int main() {
  sum = n;
  rt = 0;
  maxx[rt] = inf;
  calcsiz(1, -1);
  calcsiz(rt, -1);
  pre(rt);
}
```

实现修改

在查询和修改的时候，我们在点分树上暴力跳父亲修改。由于点分树的深度最多是 $O(\log n)$ 的，所以这样做复杂度能得到保证。

在动态点分治的过程中，需要一个结点到其点分树上的祖先的距离等其他信息，由于一个点最多有 $O(\log n)$ 个祖先，我们可以在计算点分树时额外计算深度 $dep[x]$ 或使用 LCA，预处理出这些距离或实现实时查询。**注意**：一个结点到其点分树上的祖先的距离不一定递增，不能累加！

在动态点分治的过程中，一个结点在其点分树上的祖先结点的信息中可能会被重复计算，这是我们需要消去重复部分的影响。一般的方法是对于一个连通块用两种方式记录：一个是其到分治中心的距离信息，另一个是其到点分树上分治中心父亲的距离信息。这一部分内容将在例题中得到展现。

??? note " 例题 [「ZJOI2007」捉迷藏]"
    给定一棵有 n 个结点的树，初始时所有结点都是黑色的。你需要实现以下两种操作：

```
1. 反转一个结点的颜色（白变黑，黑变白）；
2.  询问树上两个最远的黑点的距离。

    $n\le 10^5,m\le 5\times 10^5$
```

求出点分树，对于每个结点 x 维护两个 **可删堆**。$dist[x]$ 存储结点 x 代表的连通块中的所有黑点到 x 的距离信息，$ch[x]$ 表示结点 x 在点分树上的所有儿子和它自己中的黑点到 x 的距离信息，由于本题贪心的求答案方法，且两个来自于同一子树的路径不能成为一条完成的路径，我们只在这个堆中插入其自己的值和其每个子树中的最大值。我们发现，$ch[x]$ 中最大的两个值（如果没有两个就是所有值）的和就是分治时分支中心为 x 时经过结点 x 的最长黑端点路径。我们可以用可删堆 ans 存储所有结点的答案，这个堆中的最大值就是我们所求的答案。

我们可以根据上面的定义维护 $dist[x], ch[x], ans$ 这些可删堆。当 $dist[x]$ 中的值发生变化时，我们也可以在 $O(\log n)$ 的时间复杂度内维护 $ch[x], ans$。

现在我们来看一下，当我们反转一个点的颜色时，$dist[x]$ 值会发生怎样的改变。当结点原来是黑色时，我们要进行的是删除操作；当结点原来是白色时，我们要进行的是插入操作。

假如我们要反转结点 x 的颜色。对于其所有祖先 u，我们在 $dist[u]$ 中插入或删除 $dist(x, u)$，并同时维护 $ch[x], ans$ 的值。特别的，我们要在 $ch[x]$ 中插入或删除值 0。

参考代码：

```
--8<-- "docs/graph/code/dynamic-tree-divide/dynamic-tree-divide_1.cpp"
```

???+note " 例题Luogu P6329 【模板】点分树 | 震波"
    给定一棵有 n 个结点的树，树上每个结点都有一个权值 $v[x]$。实现以下两种操作：

    1. 询问与结点 x 距离不超过 y 的结点权值和；
    2. 修改结点 x 的点权为 y，即 $v[x]=y$。

我们用动态开点权值线段树记录距离信息。

类似于上题的思路，对于每个结点，我们维护线段树 $dist[x]$，表示分治块 x 中的所有结点到结点 x 的距离信息，下标为距离，权值加上点权。线段树 $ch[x]$ 表示分治块 x 中所有结点到结点 x 在分治树上的父亲结点的距离信息。

在本题中，所有查询和修改都需要在点分树上对所有祖先进行修改。

以查询操作为例，如果我们要查询距离结点 x 不超过 y 的结点的权值和，我们要先将答案加上线段树 $dist[x]$ 中下标从 0 到 y 的权值和，然后我们遍历 x 的所有祖先 u，设其低一级祖先为 v，令 $d = dist(x, u)$，如果我们不进入包含 x 的子树，即以 v 为根的子树，那么我们要将答案加上线段树 $dist[u]$ 中下标从 0 到 $y - d$ 的权值和。由于我们重复计算了以 v 为根的部分，我们要将答案减去线段树 $ch[v]$ 中下标从 0 到 $y - d$ 的权值和。

在进行修改操作时，我们要同时维护 $dist[x]$ 和 $ch[x]$。

参考代码：