可持久化数据结构

jiangly

可持久化数据结构

- 可持久化数据结构（Persistent Data Structures）可以保留历史版本的信息，支持对历史版本进行查询甚至修改。
- 如果历史版本只支持访问，只有最新版本可以修改（也就是所有版本线性排列），则称为部分可持久化（Partially Persistent）。
- 如果所有版本支持访问和修改（也就是所有版本形成树形依赖关系），则称为完全可持久化（Fully Persistent）。

可持久化数据结构

- 一些题目要求在线，很可能需要使用可持久化数据结构。
- 除此之外，一些算法需要同时使用多个版本的数据结构，或是需要在线地操作数据结构，则也依赖可持久化数据结构。
- 如果题目有版本的依赖关系，但却没有强制在线，通常不需要可持久化数据结构，只需要在版本树上 DFS，用可回退数据结构维护。
- 设计可持久化数据结构要求时间复杂度不能是均摊的。
- 常见的可持久化数据结构有可持久化线段树、可持久化平衡树、可持久化数组、可持久化堆等。

可持久化技术

- 设计可持久化数据结构通常有两种不同的技术：Path Copy 和 Fat Node。
- Fat Node 适用于设计部分可持久化数据结构，其原理是在每次修改信息时记录时间戳，每次访问时通过记录的时间戳找到对应时间的值。
- Path Copy 适用于树形数据结构，其原理是在每次修改时复制访问的路径，而重复使用未访问到的结点。Path Copy 通常可以做到完全可持久化，并且不会导致复杂度提高。

部分可持久化数组

- 维护数组 $a_1, \ldots, a_n$，支持以下操作：
  - 把 a_i 修改为 x。
  - 查询 a_i 在第 k 次操作后的值。

- 每个位置开一个动态数组，存若干二元组 (t, x) 表示时间 t 时修改成了 x。
- 修改时只需在对应位置最后插入。$O(1)$。
- 查询时二分出最后一次修改。$O(\log q)$。

例题：HDOJ 4348 To the Moon

- 长度为 n 的数组，m 次以下操作：
  - 将 $a_l, ..., a_r$ 加上 d，该操作会将时间加 1。
  - 求 $a_l, ..., a_r$ 的和。
  - 求 t 时刻 $a_l, ..., a_r$ 的和。
  - 回到 t 时刻。
- $1 \leq n, m \leq 10^5$。

例题：HDOJ 4348 To the Moon

- 使用可持久化线段树。
- 可持久化线段树采用 Path Copy 的技术，每次修改时复制路径上的所有结点。
- 必须显式建立树的结构，而非用数组存储。
- 一般有两种实现方式：
  - **void modify(Node *&t, ...)**
  - **Node *modify(Node *t, ...)**
- 标记处理上一般也有两种处理方式：下传标记或标记永久化（特殊情况）。

例题：HDOJ 4348 To the Moon

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int tag = 0;
    int sum = 0;
    int siz = 0;

    Node(Node *t = nullptr) {
        if (t) {
            *this = *t;
        }
    }
};

void add(Node *&t, int v) {
    t = new Node(t);
    t->sum += t->siz * v;
    t->tag += v;
}
```

```
void push(Node *t) {
    add(t->l, t->tag);
    add(t->r, t->tag);
    t->tag = 0;
}

void rangeAdd(Node *&t, int l, int r, int x, int y, int v) {
    if (l ≥ y || r ≤ x) {
        return;
    }
    if (l ≥ x && r ≤ y) {
        add(t, v);
        return;
    }
    t = new Node(t);
    push(t);
    int m = (l + r) / 2;
    rangeAdd(t->l, l, m, x, y, v);
    rangeAdd(t->r, m, r, x, y, v);
    pull(t);
}
```

例题：HDOJ 4348 To the Moon

- 可持久化线段树在实现上要注意的就是一旦对一个点进行了实质上的修改，就要将其复制（$push$ 并没有造成实质上的修改）。
- 使用可持久化数据结构要特别注意空间复杂度和常数。
- 对于可持久化线段树，一般来说，单点修改每次 $\log n$，区间修改标记永久化每次 $2\log n$，区间修改下传标记在上面的实现是每次 $6\log n$，稍加修改可以变成 $4\log n$。
- 一般来说，只有对于任意的修改 a, b 都能找到 c 使得 $ab = ca$ 才能标记永久化。可交换的标记是一个特殊情况。
- 时空复杂度均为 $O(n + m\log n)$。

例题：静态区间第 k 小

- 给定数组 $a_1, ..., a_n$。
- q 次询问 $a_l, ..., a_r$ 当中第 k 小的数。
- 要求 $O(n\log n)$ 预处理，$O(\log n)$ 回答询问。

例题：静态区间第 k 小

- 假设有一个值域线段树 T，求 T 上的第 k 小，可以 $O(\log n)$ 完成。
- 怎么样得到区间 $[l,r]$ 的值域线段树？
- 预处理出每个前缀的值域线段树，这一步用到可持久化。
- $[l,r]$ 的线段树每个结点的值就是 $[1,r]$ 的值减去 $[1,l-1]$ 的值，在线段树上二分的过程中进行这样的计算即可。
- 可以扩展到树上路径，用两点和 LCA 出的线段树进行差分。
- 在 k 棵线段树上进行二分的时间复杂度为 $O(k\log n)$。

可持久化线段树合并

- 只需要在线段树合并的基础上每次新加一个结点即可。

完全可持久化数组

- 这里只介绍在 OI 中最常用的方法：可持久化线段树。
- 即用线段树维护数组，非叶结点不存任何信息，只作为访问的中介。
- 时间复杂度 $O(\log n)$。

可持久化并查集

- 并查集虽然是树形结构，但是每个点存储的不是孩子，反而是父结点。
- 存父节点在可持久化数据结构中是不被允许的。这是因为，当数据结构可持久化之后就不再是树形结构，而是 DAG。
- 并查集基于数组实现，那么可持久化并查集可以基于可持久化数组实现。
- （虽然线段树、二叉堆等数据结构也能用数组实现，但因为可持久化数组并非 $O(1)$，会导致复杂度增加）
- 不再能使用路径压缩，必须按秩合并，时间复杂度 $O(\log^2 n)$。

可持久化并查集

- 并查集虽然是树形结构，但是每个点存储的不是孩子，反而是父结点。
- 存父节点在可持久化数据结构中是不被允许的。这是因为，当数据结构可持久化之后就不再是树形结构，而是 DAG。
- 并查集基于数组实现，那么可持久化并查集可以基于可持久化数组实现。
- （虽然线段树、二叉堆等数据结构也能用数组实现，但因为可持久化数组并非 $O(1)$，会导致复杂度增加）
- 不再能使用路径压缩，必须按秩合并，时间复杂度 $O(\log^2 n)$。

可持久化 Trie

- 绝大部分情况下，Trie 和线段树是等价的。
- 例外的情况也就是 Trie 可以做一些交换左右儿子的操作。

可持久化平衡树

- Splay、替罪羊树等无法因为复杂度均摊无法可持久化。
- 最常用的可持久化平衡树是 Treap，但是其由于随机性有许多劣势，如合并复杂度高、无法复制等。
- 最理想的可持久化平衡树是 AVL 树或者红黑树。
- 如果需要 Treap 实现复制的功能，可以不记录随机种子，只记录每个点的大小，在比较时按照大小加权随机。这个方法的复杂度尚未被证明。

GYM103104K Chtholly and World-End Battle

- 数组 $a_1, \ldots, a_n$。
- m 次询问，每次给出 l, r, v，求对 $i = l, \ldots, r$ 依次执行 $v \leftarrow |a_i - v|$ 之后 v 的值。
- 强制在线。
- $1 \leq n, m, a_i, v \leq 10^5$。

GYM103104K Chtholly and World-End Battle

- 线段树，考虑对每个区间维护 v 经过 $[l,r]$ 之后的结果。
- 从后往前，假设已经得到了 $[l+1,r]$ 的结果 f，则只需令 $f(x) \leftarrow f(|x-a_l|)$，也就是对 f 进行一些截取、复制、翻转、合并的操作，可以用可持久化平衡树维护。
- 询问时按照线段树询问即可。
- $O\big((n+q)\log n\log V\big)$。

例题：动态凸包

- 平面上 n 个点 (i, y_i)。
- q 次修改，每次修改 y_i，然后询问凸包点数。
- 要求每次修改 $O(\log^2 n)$。

例题：动态凸包

- 用线段树维护，则合并信息需要合并两个值域不相交的凸包。
- 用可持久化平衡树维护凸包，则可以在 $O(\log n)$ 的时间内通过在两棵平衡树上同时二分来求出切线，进而得到合并后的凸包。

例题：动态凸包

- 用线段树维护，则合并信息需要合并两个值域不相交的凸包。
- 用可持久化平衡树维护凸包，则可以在 $O(\log n)$ 的时间内通过在两棵平衡树上同时二分来求出切线，进而得到合并后的凸包。