珂朵莉树

名称简介

老司机树，ODT(Old Driver Tree)，又名珂朵莉树（Chtholly Tree)。起源自 CF896C [https://codeforces.com/problemset/problem/896/C] 。

前置知识

会用 STL 的 set 就行。

核心思想

把值相同的区间合并成一个结点保存在 set 里面。

用处

骗分。只要是有区间赋值操作的数据结构题都可以用来骗分。在数据随机的情况下一般效率较高，但在不保证数据随机的场合下，会被精心构造的特殊数据卡到超时。

如果要保证复杂度正确，必须保证数据随机。详见 Codeforces 上关于珂朵莉树的复杂度的证明 [http://codeforces.com/blog/entry/56135?#comment-398940] 。

更详细的严格证明见 珂朵莉树的复杂度分析 [https://zhuanlan.zhihu.com/p/102786071] 。对于 add，assign 和 sum 操作，用 set 实现的珂朵莉树的复杂度为 $O(n \log \log n)$ ，而用链表实现的复杂度为 $O(n \log n)$ 。

正文

首先，结点的保存方式：

```
struct Node_t {
  int l, r;
  mutable int v;
  Node_t(const int &il, const int &ir, const int &iv) : l(il),
r(ir), v(iv) {}
  inline bool operator<(const Node_t &o) const { return l <
o.l; }
};
```

其中， `int v` 是你自己指定的附加数据。

`mutable` 关键字的含义是什么?

`mutable` 的意思是“可变的”，让我们可以在后面的操作中修改 `v` 的值。在 C++ 中，mutable 是为了突破 const 的限制而设置的。被 mutable 修饰的变量（mutable 只能用于修饰类中的非静态数据成员），将永远处于可变的状态，即使在一个 const 函数中。

这意味着，我们可以直接修改已经插入 `set` 的元素的 `v` 值，而不用将该元素取出后重新加入 `set` 。

然后，我们定义一个 `set<Node_t> odt;` 来维护这些结点。为简化代码，可以 `typedef set<Node_t>::iterator iter` ，当然在题目支持 C++11 时也可以使用 `auto` 。

split

`split` 是最核心的操作之一，它用于将原本包含点 x 的区间（设为 $[l,r]$ ）分裂为两个区间 $[l,x)$ 和 $[x,r]$ 并返回指向后者的迭代器。 参考代码如下：

```
auto split(int x) {
  if (x > n) return odt.end();
  auto it = --odt.upper_bound((Node_t){x, 0, 0});
  if (it->l == x) return it;
  int l = it->l, r = it->r, v = it->v;
  odt.erase(it);
  odt.insert(Node_t(l, x - 1, v));
  return odt.insert(Node_t(x, r, v)).first;
}
```

这个玩意有什么用呢？ 任何对于 $[l, r]$ 的区间操作，都可以转换成 set 上 $[split(l), split(r + 1))$ 的操作。

assign

另外一个重要的操作 `assign` 用于对一段区间进行赋值。 对于 ODT 来说，区间操作只有这个比较特殊，也是保证复杂度的关键。 如果 ODT 里全是长度为 1 的区间，就成了暴力，但是有了 `assign` ，可以使 ODT 的大小下降。 参考代码如下：

```
void assign(int l, int r, int v) {
  auto itr = split(r + 1), itl = split(l);
  odt.erase(itl, itr);
  odt.insert(Node_t(l, r, v));
}
```

其他操作

套模板就好了，参考代码如下：

```
void performance(int l, int r) {
  auto itr = split(r + 1), itl = split(l);
  for (; itl != itr; ++itl) {
    // Perform Operations here
  }
}
```

注： **珂朵莉树在进行求取区间左右端点操作时，必须先 split 右端点，再 split 左端点** 。若先 split 左端点，返回的迭代器可能在 split 右端点的时候失效，可能会导致 RE。

习题

- 「SCOI2010」序列操作 [https://www.luogu.com.cn/problem/P2572]
- 「SHOI2015」脑洞治疗仪 [https://loj.ac/problem/2037]
- 「Luogu 2787」理理思维 [https://www.luogu.com.cn/problem/P2787]
- 「Luogu 4979」矿洞：坍塌 [https://www.luogu.com.cn/problem/P4979]