1. 线段树

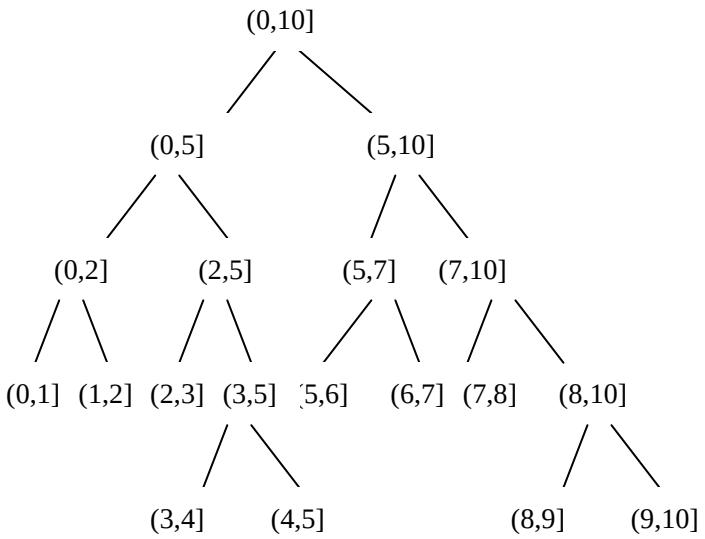
（1）线段树的基本定义和一些概念

线段树是解决数列维护问题的一种常用手段，基本能保证每个操作的复杂度在$O(\log n)$的级别，拥有明显的效率优势。

线段树是一棵二叉树，每个结点表示一个区间$(a,b]$。

如果$b=a+1$，则此结点为叶结点，没有子结点；否则此结点有两个子结点，左子结点表示的区间为$\left(a,\left\lfloor\frac{a+b}{2}\right\rfloor\right]$，右子结点表示的区间为$\left(\left\lfloor\frac{a+b}{2}\right\rfloor,b\right]$。

若根结点表示的区间为$(0,10]$，那么其对应的线段树如下图所示：



图一 线段树的一个例子

可以发现，如果一棵线段树的根结点表示的区间为$(0,n]$，那么这棵线段树将会有n个叶子结点，又因为每个结点要么有两个子结点，要么没有子结点，所以线段树总结点数为$2n-1$，因此线段树的空间复杂度为$O(n)$。然后可以发现一棵线段树除了最后一层外，前面每一层的结点都是满的，因此线段树的深度$h=\lceil\log(2n-1)\rceil=O(\log n)$。

对于线段树中的每个结点，其表示一个区间，我们可以记录和这个区间相关的一些信息（如最大值、最小值、和等），但要满足可二分性，即能直接由其子

结点的相关信息得到。

对于询问区间信息和修改区间信息的操作，线段树一般在能$O(\log n)$的时间内完成，而且常数相对较小（和后面的伸展树比较），算是一种高效实用的数据结构。

（2）线段树的基本操作——查询区间信息

使用线段树的目的就是能高效地查找和修改区间信息，下面先介绍第一个操作——查询操作。

对于当前要查询的区间$(a,b]$，我们从线段树的根结点开始，如果当前结点表示的区间被查询区间完全包含，那么更新结果，否则分别考察左右子结点，如果查询区间与某个子结点有交集（也可能两个都有），那么就递归考察这个子结点。代码框架如下[1]：

```
// node 为线段树的结点类型，其中 Left 和 Right 分别表示区间左右端点
// Lch 和 Rch 分别表示指向左右孩子的指针
void Query(node *p, int a, int b)  // 当前考察结点为 p，查询区间为(a,b]
  {
    if (a <= p->Left && p->Right <= b)
    // 如果当前结点的区间包含在查询区间内
      {
        ......      // 更新结果
        return;
      }
    Push_Down(p);  // 等到下面的修改操作再解释这句
    int mid = (p->Left + p->Right) / 2;  // 计算左右子结点的分隔点
    if (a < mid) Query(p->Lch, a, b);  // 和左孩子有交集，考察左子结点
    if (b > mid) Query(p->Rch, a, b);  // 和右孩子有交集，考察右子结点
  }
```

对于任意一个区间，会被划分成很多在线段树上存在的区间，可以证明，划分出来的区间在线段树的每层最多有两个，又因为线段树的深度为$O(\log n)$，因此查询操作的时间复杂度为$O(\log n)$。

（3）线段树的基本操作——修改区间信息

相对于查询区间信息，修改区间信息显得稍微复杂一些。

[1] 本文中的代码均使用 C++语言描述

如果题目只会对线段树中形如$(a-1,a]$的区间进行修改，那么问题会简单得多。因为这样的区间必然是一个叶子结点，修改完这个结点的信息后，只需向上维护其祖先的信息即可。

但如果要修改任意区间的信息（比如将这个区间全部置为某个数），那么问题就远没那么简单。如果我们还像上面查询的那样，把修改区间划分成线段树中的区间，然后只修改这些结点的信息，那么这些结点的子结点的信息就是错误的，如果某次询问用到了，必然会出错，但又不能将这些子结点的信息全部修改，这必将会使时间复杂降为$O(n)$。

下面我们引入延迟标记的一些概念。每个结点新增加一个标记，记录这个结点是否被进行了某种修改操作，并且这种修改操作会影响其子结点。那么还是像上面的一样，对于任意区间的修改，我们先按照查询的方式将其划分成线段树中的结点，然后修改这些结点的信息，并给这些结点标上代表这种修改操作的标记。

这样的话，在修改和查询的时候，如果我们到了一个结点 p，并且决定考虑其子结点，那么我们就要看看结点 p 有没有标记，如果有，就要按照标记修改其子结点的信息，并且给子结点都标上相同的标记，最后消掉 p 的标记。（查询程序中的 *PushDown* 就是将结点的标记向下传递）。修改区间操作的代码框架：

```
// node 为线段树的结点类型，其中 Left 和 Right 分别表示区间左右端点
// Lch 和 Rch 分别表示指向左右孩子的指针
void Change(node *p, int a, int b)  // 当前考察结点为 p，修改区间为(a,b]
  {
    if (a <= p->Left && p->Right <= b)
    // 如果当前结点的区间包含在修改区间内
      {
        ......     // 修改当前结点的信息，并标上标记
        return;
      }
    Push_Down(p);  // 把当前结点的标记向下传递
    int mid = (p->Left + p->Right) / 2;  // 计算左右子结点的分隔点
    if (a < mid) Change(p->Lch, a, b);  // 和左孩子有交集，考察左子结点
    if (b > mid) Change(p->Rch, a, b);  // 和右孩子有交集，考察右子结点
    Update(p);  // 维护当前结点的信息（因为其子结点的信息可能有更改）
  }
```

（4）线段树特点总结

利用线段树，我们可以高效地询问和修改一个数列中某个区间的信息，并且代码也不算特别复杂。

但是线段树也是有一定的局限性的，其中最明显的就是数列中数的个数必须固定，即不能添加或删除数列中的数。对于这个问题，下面介绍的伸展树就可以完美的解决。