11-0419

数据结构之线段树

Category: 数据结构与算法 View: 149,766 阅 Author: Dong

作者:Dong | 新浪微博：西成懂 | 可以转载，但必须以超链接形式标明文章原始出处和作者信息及版权声明

网址:http://dongxicheng.org/structure/segment-tree/
本博客的文章集合:http://dongxicheng.org/recommend/

中国第一个在线Hadoop教育平台—小象学院，推荐给Hadoop初学者和实践者，网址是： http://www.chinahadoop.cn/

本博客微信公共账号：hadoop123（微信号为：hadoop-123），分享hadoop技术内幕，hadoop最新技术进展，发布hadoop相关职位和求职信息，hadoop技术交流聚会、讲座以及会议等。二维码如下：

1、概述

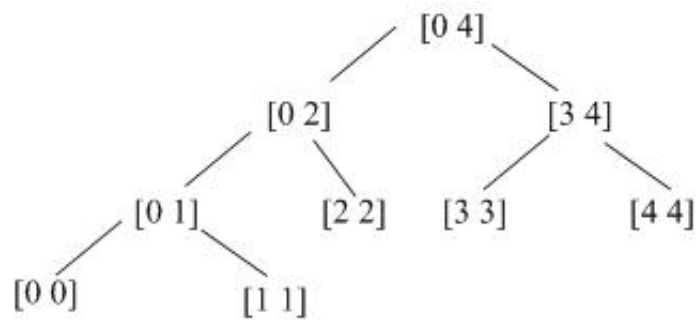
线段树，也叫区间树，是一个完全二叉树，它在各个节点保存一条线段（即“子数组”），因而常用于解决数列维护问题，它基本能保证每个操作的复杂度为O(lgN)。

2、线段树基本操作

线段树的基本操作主要包括构造线段树，区间查询和区间修改。

（1） 线段树构造

首先介绍构造线段树的方法：让根节点表示区间[0,N-1]，即所有N个数所组成的一个区间，然后，把区间分成两半，分别由左右子树表示。不难证明，这样的线段树的节点数只有2N-1个，是O(N)级别的，如图：



显然，构造线段树是一个递归的过程，伪代码如下：

```
1  //构造求解区间最小值的线段树
2
3  function 构造以v为根的子树
4
5    if v所表示的区间内只有一个元素
6
7       v区间的最小值就是这个元素，构造过程结束
8
9    end if
10
11   把v所属的区间一分为二，用w和x两个节点表示。
```

```
    标记v的左儿子是w，右儿子是x

    分别构造以w和以x为根的子树（递归）

    v区间的最小值 <- min(w区间的最小值,x区间的最小值)

end function
```

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 =ceil(log(2n -1))=O(log n)。

（2）　区间查询

区间查询指用户输入一个区间，获取该区间的有关信息，如区间中最大值，最小值，第N大的值等。

比如前面一个图中所示的树，如果询问区间是[0,2]，或者询问的区间是[3,3]，不难直接找到对应的节点回答这一问题。但并不是所有的提问都这么容易回答，比如[0,3]，就没有哪一个节点记录了这个区间的最小值。当然，解决方法也不难找到：把[0,2]和[3,3]两个区间（它们在整数意义上是相连的两个区间）的最小值“合并”起来，也就是求这两个最小值的最小值，就能求出[0,3]范围的最小值。同理，对于其他询问的区间，也都可以找到若干个相连的区间，合并后可以得到询问的区间。

区间查询的伪代码如下：

```
// 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

}
```

可见，这样的过程一定选出了尽量少的区间，它们相连后正好涵盖了整个[l,r]，没有重复也没有遗漏。同时，考虑到线段树上每层的节点最多会被选取2个，一共选取的节点数也是O(log n)的，因此查询的时间复杂度也是O(log n)。

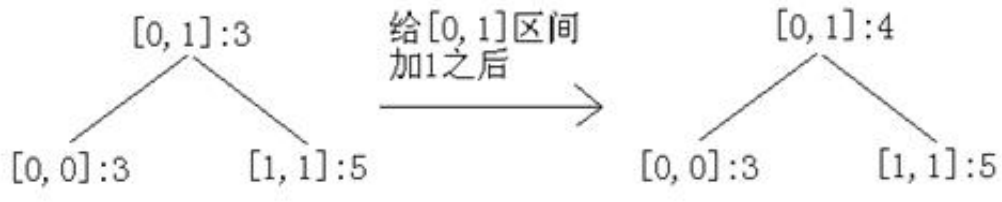
线段树并不适合所有区间查询情况，它的使用条件是“相邻的区间的信息可以被合并成两个区间的并区间的信息”。即问题是可以被分解解决的。

（3）　区间修改

当用户修改一个区间的值时，如果连同其子孙全部修改，则改动的节点数必定会远远超过O(log n)个。因而，如果要想把区间修改操作也控制在O(log n)的时间内，只修改O(log n)个节点的信息就成为必要。

借鉴前一节区间查询用到的思路：区间修改时如果修改了一个节点所表示的区间，也不用去修改它的儿子节点。然而，对于被修改节点的祖先节点，也必须更新它所记录的值，否则查询操作就肯定会出问题（正如修改单个节点的情况一样）。

这些选出的节点的祖先节点直接更新值即可，而选出的节点的子孙却显然不能这么简单地处理：每个节点的值必须能由两个儿子节点的值得到，如这幅图中的例子：



这里，节点[0,1]的值应该是4，但是两个儿子的值又分别是3和5。如果查询[0,0]区间的RMQ，算出来的结果会是3，而正确答案显然是4。

问题显然在于，尽管修改了一个节点以后，不用修改它的儿子节点，但是它的儿子节点的信息事实上已经被改变了。这就需要我们在节点里增设一个域：**标记**。把对节点的修改情况储存在标记里面，这样，当我们自上而下地访问某节点时，就能把一路上所遇到的所有标记都考虑进去。

但是，在一个节点带上标记时，会给更新这个节点的值带来一些麻烦。继续上面的例子，如果我把位置0的数字从4改成了3，区间[0,0]的值应该变回3，但实际上，由于区间[0,1]有一个“添加了1”的标记，如果直接把值修改为3，则查询区间[0,0]的时候我们会得到3+1=4这个错误结果。但是，把这个3改成2，虽然正确，却并不直观，更不利于推广（参见下面的一个例子）。

为此我们引入**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(这种修改操作会影响其子结点)。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。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，同时消掉p 的标记。代码框架为：

```
// 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）

}
```

3、应用

下面给出线段树的几个应用：

（1）有一列数，初始值全部为0。每次可以进行以下三种操作中的一种：

a．给指定区间的每个数加上一个特定值；

b.将指定区间的所有数置成一个统一的值；

c.询问一个区间上的最小值、最大值、所有数的和。

给出一系列a.b.操作后，输出c的结果。

[问题分析]

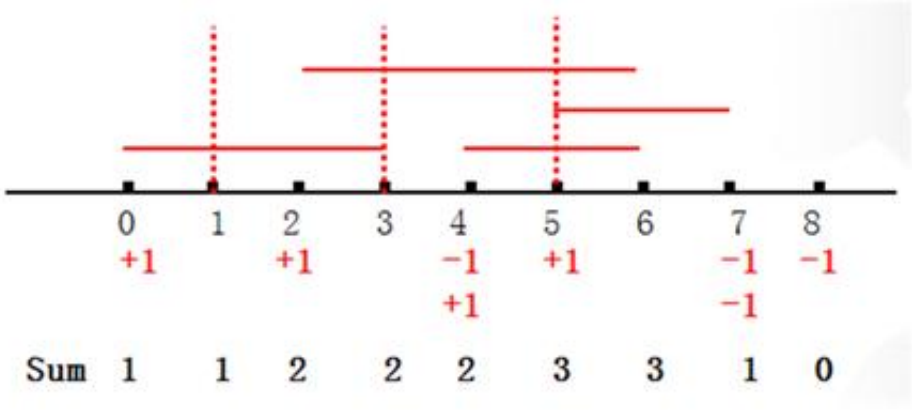
这个是典型的线段树的应用。在每个节点上维护一下几个变量：delta（区间增加值），same（区间被置为某个值），min（区间最小值），max（区间最大值），sum（区间和），其中delta和same属于“延迟标记”。

（2）在所有不大于30000的自然数范围内讨论一个问题：已知n条线段，把端点依次输入给你，然后有m（≤30000）个询问，每个询问输入一个点，要求这个点在多少条线段上出现过。

[问题分析]

在这个问题中，我们可以直接对问题处理的区间建立线段树，在线段树上维护区间被覆盖的次数。将n条线段插入线段树，然后对于询问的每个点，直接查询被覆盖的次数即可。

但是我们在这里用这道题目，更希望能够说明一个问题，那就是这道题目完全可以不用线段树。我们将每个线段拆成(L,+1)，(R+1,-1)的两个事件点，每个询问点也在对应坐标处加上一个询问的事件点，排序之后扫描就可以完成题目的询问。我们这里讨论的问题是一个离线的问题，因此我们也设计出了一个很简单的离线算法。线段树在处理在线问题的时候会更加有效，因为它维护了一个实时的信息。



这个题目也告诉我们，有的题目尽管可以使用线段树处理，但是如果我们能够抓住题目的特点，就可能获得更加优秀的算法。

（3）某次列车途经C个城市，城市编号依次为1到C，列车上共有S个座位，铁路局规定售出的车票只能是坐票，即车上所有的旅客都有座，售票系统是由计算机执行的，每一个售票申请包含三个参数，分别用O、D、N表示，O为起始站，D为目的地站，N为车票张数，售票系统对该售票申请作出受理或不受理的决定，只有在从O到D的区段内列车上都有N个或N个以上的空座位时该售票申请才被受理，请你写一个程序，实现这个自动售票系统。

[问题分析]

这里我们可以把所有的车站顺次放在一个数轴上，在数轴上建立线段树，在线段树上维护区间的delta与max。每次判断一个售票申请是否可行就是查询区间上的最大值；每个插入一个售票请求，就是给一个区间上所有的元素加上购票数。

这道题目在线段树上维护的信息既包括自下至上的递推，也包括了自上至下的传递，能够比较全面地对线段树的基本操作进行训练。

（4）给一个n*n的方格棋盘，初始时每个格子都是白色。现在要刷M次黑色或白色的油漆。每次刷漆的区域都是一个平行棋盘边缘的矩形区域。

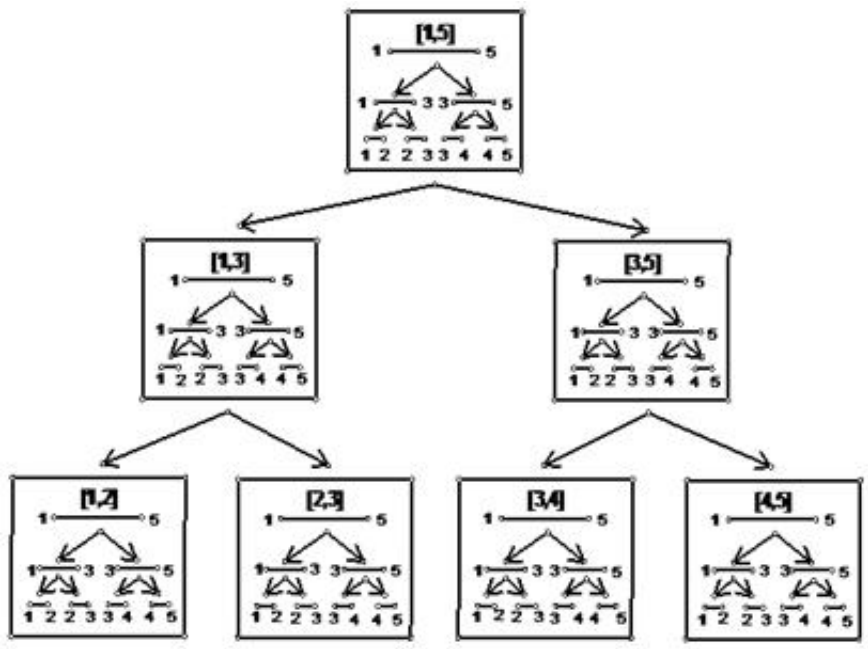
输入n,M,以及每次刷漆的区域和颜色，输出刷了M次之后棋盘上还有多少个棋格是白色。

[问题分析]

首先我们从简单入手，考虑一维的问题。即对于一个长度为n的白色线段，对它进行M次修改（每次更新某一子区域的颜色）。问最后还剩下的白色区域有多长。

对于这个问题，很容易想到建立一棵线段树的模型。复杂度为O(MIgn)。

扩展到二维，需要把线段树进行调整，即首先在横坐标上建立线段树，它的每个节点是一棵建立在纵坐标上的线段树（即树中有树。称为二维线段树）。复杂度为O(M(logn)^2)。



4、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。

5、参考资料

（1） 杨弋文章：《线段树》：

http://download.csdn.net/source/2255479

（2） 林涛文章《线段树的应用》：

http://wenku.baidu.com/view/d65cf31fb7360b4c2e3f64ac.html

（3） 朱全民文章《线段树及其应用》：

http://wenku.baidu.com/view/437ad3bec77da26925c5b0ba.html

（4） 线段树：