**线段树开4N空间证明**

线段树采用数组储存时，无疑，其储存空间利用与其左右子树定义有关

方式一：

左子树：[l,(l+r)/2]

右子树：[(l+r)/2+1,r]

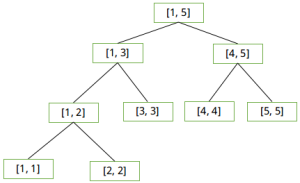
方式二：

左子树：[l,(l+r+1)/2−1]

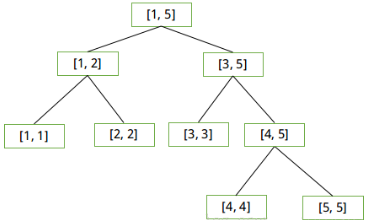
右子树：[(l+r+1)/2,r]

假设定义区间[1,5]的线段树，很容易看出它们的不同

方式一：



方式二：



由此初步看来，采用第二种方式定义，可能会在其左边产生较大的空白区域。

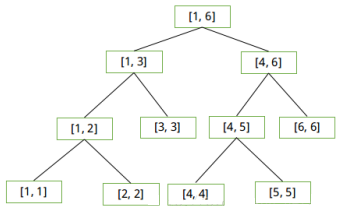
实际上也的确如此，

为了方便我们的习惯，考虑第一种方式定义情况

对于某一区间[l, r]，当区间元素个数为偶数时，这很好，因为这样左右子树分得相同多的叶结点，也即是左右区间长度相等；如果区间为奇数，这意味着其中一边不得不多拿一个叶结点，而究竟谁多拿，依靠你判决左右子树的策略，例如方式一：左边多，方式二，右边多。

思考这样一个问题其实是有利于我们明白当区间长度逐渐增大时，最后两层叶结点它们的变动情况。依靠你的判决策略，当当前结点区间长度为奇数时，多拿的一方会将这种方式逐层传递，例如这里，左边多一个结点，最终会被传递到最下层

好了开始我的问题，假设一开始，给出的区间长度为4，你很容易画出它们的线段树图形（一棵满二叉树），区间增加为5时，图也已经给你了（方式一），如果现在，区间增长为6，你应该也能画出它们的图形



如果我再增加呢？增加到7，然后再增加到8呢？（8是一棵满二叉树）

仔细想想，线段树的形状随着区间长度增加（数组元素增加）实际上遵循这样一个规律，从一颗满二叉树，到另外一课满二叉树，每次递增时，会多出一个叶结点(减少一个，增加两个)，好像是，在原来树的基础上首先判断当前结点左右子树区间长度（即是叶结点个数）是不是一样的，恩，如果不一样把这个结点再安排到叶结点少的那棵子树上，使其左右平衡，如果一样了，再把就它放到左子树上（依据左右子树划分策略，这里为方式一）

从另外一个角度看，好像也是合乎情理的，线段树的区间均分使得它含有这样一个性质，左右两边的叶结点个数（区间长度）之差小于等于1。

这样看来，线段树不就有点平衡二叉树的意味了么？是的，它的确是一颗平衡二叉树，在我的上一片文章中也有简单探讨

好了，明白了这些，我们接下来讨论为什么线段树要开出4N的空间

给你一棵满二叉树，其叶结点个数为N，问，有多少个非叶结点？

方法是很简单多样的，你可以选择求和，也可以根据二叉树的一些性质，这是容易的。

考虑

n0+n1+n2=2n2+n1+1

注意到此时，n0=N,n1=0

所以

n2=N−1

总共有N-1个非叶结点

考虑极端情况，方式二，且最后一层只有两个结点（如上图方式二的图那样）

此时，有N个叶结点，倒数第二层有N-1个结点

因此，除去最后一层的两个结点外，总共有

(N−1)+(N−1−1)=2N−3

由于是数组储存，最后一层有

2×(N−1)=2N−2

个结点（含空结点）总计结点个数为

2N−3+2N−2=4N−5

好了，这就是最终最坏的结果

为什么这是最坏的情况？对于一棵满二叉树（N=2k），无疑，是最好的，此时储存空间达到最小，为2N（注意这里算上了0位置，结点为2N-1），当区间长度增加1时，情况马上变坏，因为你不得不开出近似4N的空间来储存它，剩出多个没有利用的空间了，接下来，你可以松口气了，因为直到填满下一个二树前，这些空间都是足够的。

而对于方式一来说，在大多数情况下，没有方式二来的那么剧烈，一下子增加到4N，恩，先左边加一点，然后隔一段距离，右边加一点，然后左边，然后右边，每次右边增加时左边的储存就不用管了，因为已经足够。

因此可以看出，开出4N，最坏情况下，也可以得到满足

一个有意思的问题是，什么情况下方式一达到最坏呢？是不是方式一最坏的情况下空间也是需要达到4N呢？

好像有点复杂

不过可以设想的是，应该没有方式二那样如此耗费空间才是。

假设数组元素个数为N，最后一层结点为4，如果你明白了我前面的有关线段树叶结点变动的说明，你应该能理解它究竟是怎样一幅图

好了，除去最后一层结点，总个数为

(N−2)+(N−2−1)=2N−5

最后一层结点个数（含空结点）为

((N−2)×2)/2+1=N−1

总个数为

2N−5+N−1=3N−6

注：这只是第一次出现最坏的情况，这有可能并不是最坏的情况，可能存在某些中间态。

不过可以看出的是，相较于第二方式，可能储存上相对较为节省空间

不过，总体来说，从第二种方式来看，线段树开4N，的确是有必要的