线段树的入门级 总结

线段树是一种二叉搜索树，与区间树相似，它将一个区间划分成一些单元区间，每个单元区间对应线段树中的一个叶结点。

对于线段树中的每一个非叶子节点[a,b]，它的左儿子表示的区间为[a,(a+b)/2]，右儿子表示的区间为[(a+b)/2+1,b]。因此线段树是平衡二叉树，最后的子节点数目为 N，即整个线段区间的长度。

使用线段树可以快速的查找某一个节点在若干条线段中出现的次数，时间复杂度为 O(logN)。而未优化的空间复杂度为 2N，因此有时需要离散化让空间压缩。

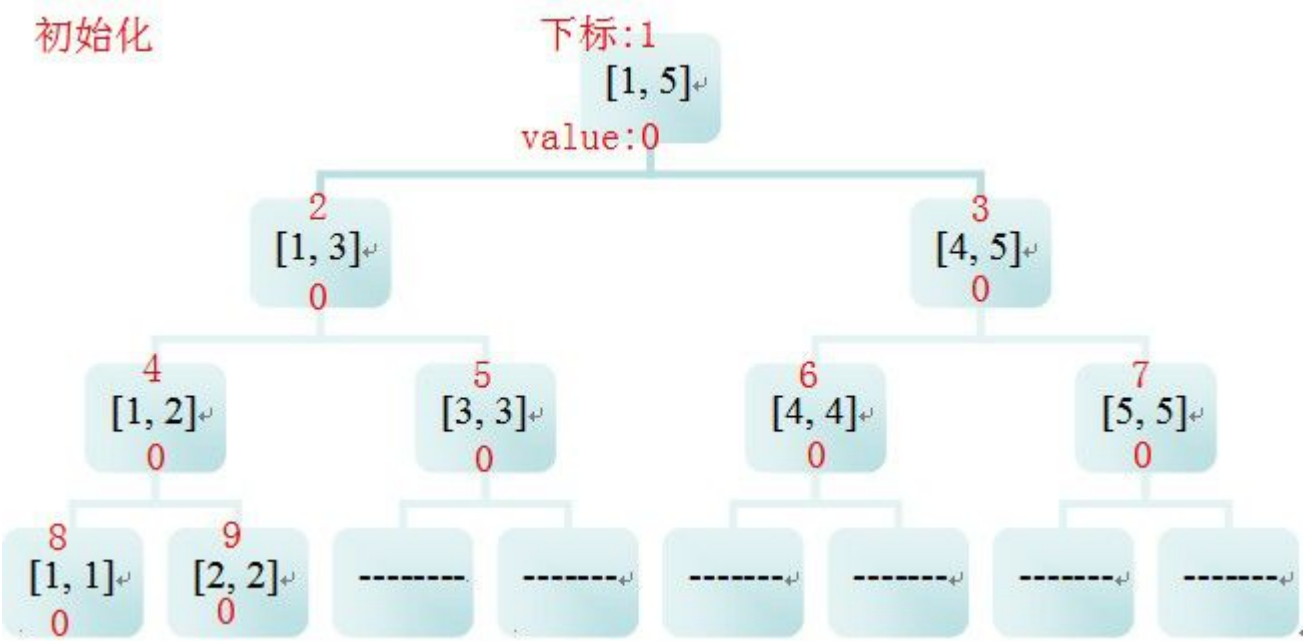
----来自百度百科

【以下以 求区间最大值为例】

先看声明：

[cpp] view plaincopy

```cpp
#include <stdio.h>
#include <math.h>
const int MAXNODE = 2097152;
const int MAX = 1000003;
struct NODE{
    int value;        // 结点对应区间的权值
    int left,right;   // 区间 [left,right]
}node[MAXNODE];
int father[MAX];     // 每个点(当区间长度为 0 时，对应一个点)对应的结构体数组下标
```



【创建线段树（初始化）】：

由于线段树是用二叉树结构储存的，而且是近乎完全二叉树的，所以在这里我使用了数组来代替链表上图中区间上面的红色数字表示了结构体数组中对应的下标。

在完全二叉树中假如一个结点的序号（数组下标）为 I，那么 （二叉树基本关系）

I 的父亲为 I/2，

I 的另一个兄弟为 I/2*2 或 I/2*2+1

I 的两个孩子为 I*2 (左) I*2+1(右)

有了这样的关系之后，我们便能很方便的写出创建线段树的代码了。

[cpp] view plaincopy

```cpp
void BuildTree(int i,int left,int right){ // 为区间[left,right]建立一个以 i 为祖先的线段树，i 为数组下标，我称作结点序号
    node[i].left = left;    // 写入第 i 个结点中的 左区间
    node[i].right = right;  // 写入第 i 个结点中的 右区间
    node[i].value = 0;      // 每个区间初始化为 0
    if (left == right){ // 当区间长度为 0 时，结束递归
        father[left] = i; // 能知道某个点对应的序号，为了更新的时候从下往上一直到顶
        return;
    }
    // 该结点往 左孩子的方向 继续建立线段树，线段的划分是二分思想，如果写过二分查找的话这里很容易接受
    // 这里将 区间[left,right] 一分为二了
```

```
11.     BuildTree(i<<1, left, (int)floor( (right+left) / 2.0));
12.     // 该结点往 右孩子的方向 继续建立线段树
13.     BuildTree((i<<1) + 1, (int)floor( (right+left) / 2.0) + 1, right);
14. }
```

【单点更新线段树】：

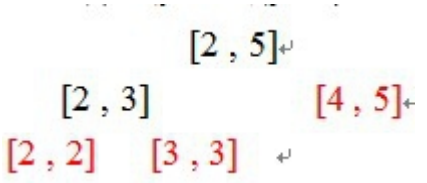
由于我事先用 father[ ] 数组保存过 每单个结点 对应的下标了，因此我只需要知道第几个点，就能知道这个点在结构体中的位置（即下标）了，这样的话，根据之前已知的基本关系，就只需要直接一路更新上去即可。

[cpp] view plaincopy

```
1.  void UpdataTree(int ri){ // 从下往上更新（注：这个点本身已经在函数外更新过了）
2.
3.      if (ri == 1)return; // 向上已经找到了祖先（整个线段树的祖先结点 对应的下标为1）
4.      int fi = ri / 2;        // ri 的父结点
5.      int a = node[fi<<1].value; // 该父结点的两个孩子结点（左）
6.      int b = node[(fi<<1)+1].value; // 右
7.      node[fi].value = (a > b)?(a):(b);    // 更新这个父结点（从两个孩子结点中挑个大的）
8.      UpdataTree(ri/2);       // 递归更新，由父结点往上找
9.  }
```

【查询区间最大值】：

将一段区间按照建立的线段树从上往下一直拆开，直到存在有完全重合的区间停止。对照图例建立的树，假如查询区间为 [2,5]



红色的区间为完全重合的区间，因为在这个具体问题中我们只需要比较这 三个区间的值 找出 最大值 即可。

[cpp] view plaincopy

```
int Max = -1<<20;
void Query(int i,int l,int r){ // i 为区间的序号（对应的区间是最大范围的那个区间，也是第一个图最顶端的区间，一般初始是 1 啦）
    if (node[i].left == l && node[i].right == r){ // 找到了一个完全重合的区间
        Max = (Max < node[i].value)?node[i].value:(Max);
        return ;
    }
    i = i << 1; // get the left child of the tree node
    if (l <= node[i].right){ // 左区间有涉及
        if (r <= node[i].right) // 全包含于左区间，则查询区间形态不变
            Query(i, l, r);
        else // 半包含于左区间，则查询区间拆分，左端点不变，右端点变为左孩子的右区间端点
            Query(i, l, node[i].right);
    }
    i += 1; // right child of the tree
    if (r >= node[i].left){ // 右区间有涉及
        if (l >= node[i].left) // 全包含于右区间，则查询区间形态不变
            Query(i, l, r);
        else // 半包含于左区间，则查询区间拆分，与上同理
            Query(i, node[i].left, r);
    }
}
```