线段树与树状数组

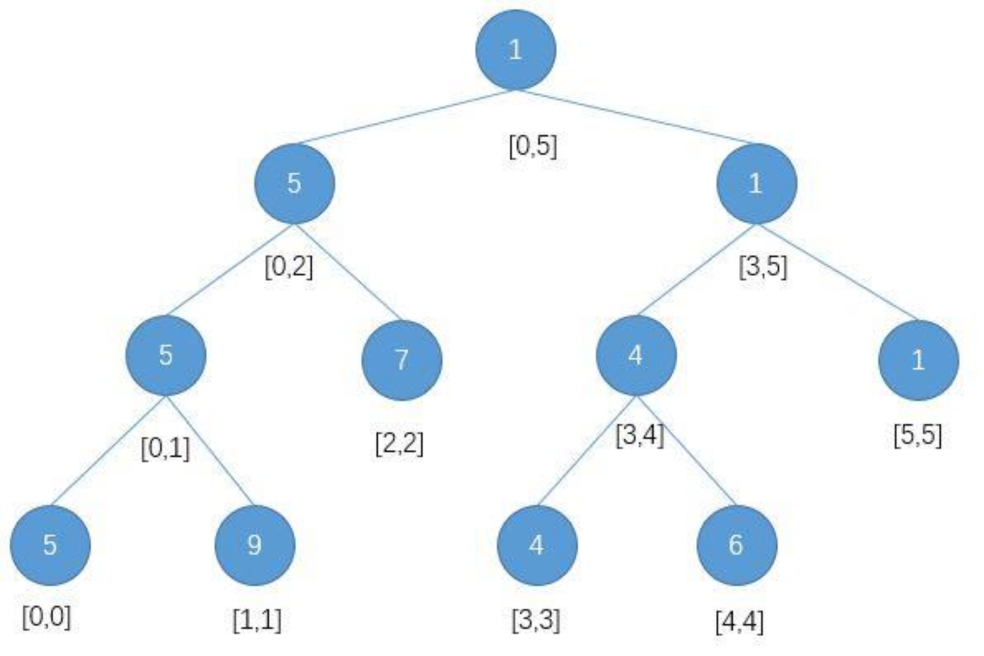
GAREN 2019.03.31

引入

- 先给大家一个问题：
- 给你一个长度为n的数组，需要支持10^5次区间加，如何解决？
- 对每次区间加都弄个for循环显然爆炸，那我们可以怎么解决？
- 您的好友“线段树”已上线

线段树

- 线段树（segment tree）是一颗完全二叉树，每个节点储存一个区间的信息。
- 通过对每个区间信息的维护，可以在$O(\log n)$时间内完成一个区间操作。
- 每个点代表一个区间，并且如果这个区间可分，则分为左右儿子两个子区间。
- 这是它的样子：（以维护区间最小值为例）



线段树存储

- 本人建议用一个struct写一个封装好的线段树，不容易跟别的弄混，并且封装了听说跑得快。
- 我是这么存的：

```
struct segTree {
    int minv[maxn << 2];//表示一个节点代表区间的最小值

} seg;
```

线段树构建

- build操作就是一个简单的递归建树，非常好理解。

```
void build(int root, int l, int r) {
    if(l == r) minv[root] = a[l];
    else {
        int mid = (l + r) >> 1;
        build(lson, l, mid);
        build(rson, mid + 1, r);
        pushup(root);
    }
}
```

线段树单点加

- 单点修改是最简单的修改了。直接从树根开始二分到目标点，修改再上传标记即可。

```
void update1(int root, int l, int r, int pos, int k) {
    if(l == r) minv[root] += k;
    else {
        int mid = (l + r) >> 1;
        if(pos <= mid) update1(lson, l, mid, pos, k);
        else update1(rson, mid + 1, r, pos, k);
        pushup(root);
    }
}
```

线段树区间加

- 因为这里要询问的只有区间最小值，所以区间加暂且还是简单的。

```
void update2(int root, int l, int r, int x, int y, int k) {
    if(r < x || y < l) return;//可能区间不重叠，掐掉
    if(l == r) minv[root] += k;
    else {
        int mid = (l + r) >> 1;
        update2(lson, l, mid, x, y, k);
        update2(rson, mid + 1, r, x, y, k);
        pushup(root);
    }
}
```

线段树单点查询

- 查询是比修改简单的。（我这么认为）

```
int query1(int root, int l, int r, int pos) {
    if(l == r) return minv[root];
    else {
        int mid = (l + r) >> 1;
        if(pos <= mid) return query1(lson, l, mid, pos);
        else return query1(rson, mid + 1, r, pos);
    }
}
```

线段树区间查询

```
int query2(int root, int l, int r, int x, int y) {
    if(r < x || y < l) return INF;
    if(x <= l && r <= y) return minv[root];
    int mid = (l + r) >> 1;
    return std::min(query2(lson, l, mid, x, y), query2(rson, mid + 1, r, x, y));
```

维护区间加的线段树

- 上面写的就是维护区间最值的线段树，还比较简单。
- 但是如果需要维护区间加，就需要另外一种神奇的操作：懒标记(lazytag)。
- 区间查询跟前面差不多，就不啰嗦了。
- 重点就在于维护和更新方面。
- 我们直接看代码：

线段树相关功能

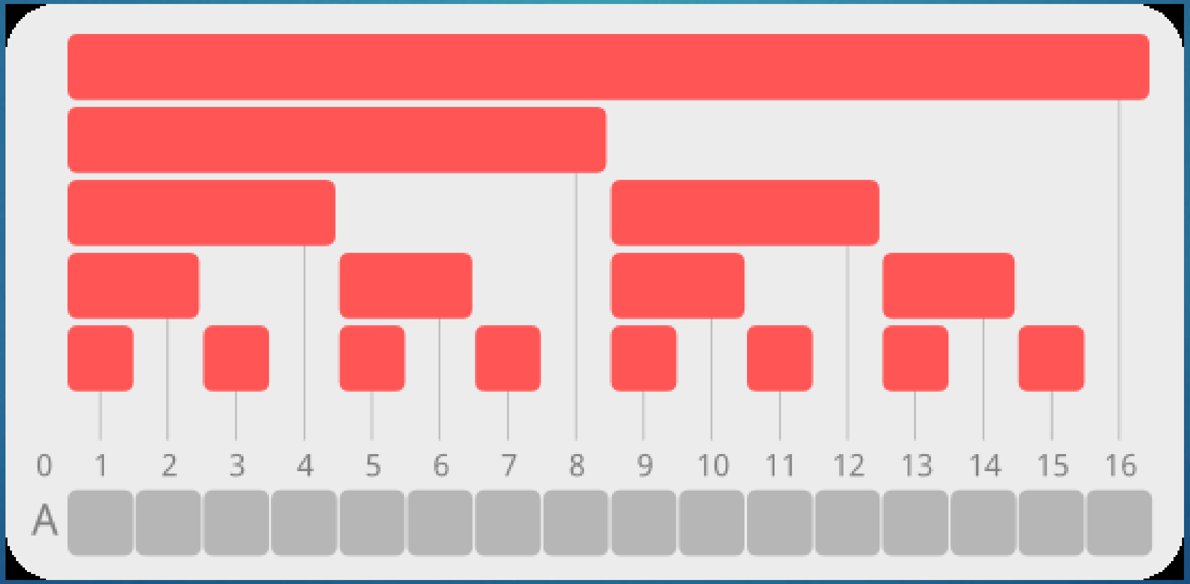
- 区间四则运算、区间开方、询问区间平方和、区间子段和等操作。
- 如果不以下标建树，以值域来建树的话，可以找到区间第k大。（主席树）

线段树作业

- P3372 【模板】线段树 1
- P3373 【模板】线段树 2
- P3368 【模板】 树状数组 2

树状数组

- 树状数组(binary indexed tree, BIT)可以理解为没有左儿子的线段树。
- 每个节点维护的是一段区间的和。
- 具体结构长这个样子：



树状数组规律

- 我们推一推规律啊：
- $s_1 = a_1, s_2 = a_2, s_3 = a_3 + a_2, s_4 = a_4$
- $s_5 = a_5 + a_4, s_6 = a_4 + a_2, s_7 = a_7 + a_6 + a_4, s_8 = a_8$
- ……
- 把这些下标换成二进制看一下：
- $s_{0001} = a_{0001}, s_{0010} = a_{0010}, s_{0011} = a_{0011} + a_{0010}, s_{0100} = a_{0100}$
- $s_{0101} = a_{0101} + a_{0100}, s_{0110} = a_{0110} + a_{0100}$
- $s_{0111} = a_{0111} + a_{0110} + a_{0100}, s_{1000} = a_{1000}$
- 有没有什么规律？

lowbit

- 可以发现，一个从1到n的前缀和，在树状数组中可以由特定的元素相加得到。
- 并且，每一个下标的变化都是有规律的：二进制中的最后一个1被去掉后剩下的数字就是下一个下标。
- 如何快速去掉一个数二进制下的最后一个1？我们可以快速获得二进制下的最后一个1，即lowbit操作。
- 这里直接给出结论：x&-x即可求出。（可通过补码证明）
- 树状数组的精髓也就在于lowbit，使一个原本$O(n)$的前缀和变为$O(\log n)$。

树状数组存储

- 一个数组就完事了。如果愿意的话原数组也保留下来。

树状数组前缀和

- 用lowbit一路减下去，遇到的就加上，得到的就是前缀和了。

树状数组单点修改

- 用lowbit一路向上一路更新，更新与该节点有关的所有节点。

再见