B题

恩，还是个线段树

不过是加入了区间修改

在进行修改时打lazy标记

本题每个节点需要保存6个数据，分别为

0的数量 1的数量 左端连续的数量 右端连续的数量 区间长度

建树与没有区间修改的线段树差不多

但是修改和查询都有一些差别

区间修改

edit修改函数

```
void edit(int node, int l, int r, int el, int er)
{
    int mid = (l + r) / 2;
    if (el <= l && r <= er)//当前区间被修改区间包括后直接打标记返回
    {
        tag(node);//为该节点打上标记
        return;
    }
    if (segTree[node][4] == 1)//标记下传必须写在判断包含的下面，因为有可能打两次标记后标记消失了
        update_lazy(node);//标记下传
    if (el <= mid)//修改左子树
        edit(node * 2, l, mid, el, er);
    if (mid < er)//修改右子树
        edit(node * 2 + 1, mid + 1, r, el, er);
    segTree[node] = update(segTree[node * 2], segTree[node * 2 + 1]);//根据左右节点的值回溯更新上层节点
}
```

整体与查询差不多，也是当前区间被修改区间包括后直接打标记返回，否则二分继续修改子树

tag标记函数

打标记的tag函数如下

```
void tag(int node)
{
    for (int i = 0; i < 4; ++i)
        segTree[node][i] = -segTree[node][i];
    swap(segTree[node][0], segTree[node][1]);//上面都是更新当前节点信息
    segTree[node][4] ^= 1;//打上标记
}
```

tag函数功能主要有两个，更新当前节点信息，以及为当前节点打上标记

update_lazy下传标记

下传标记的update_lazy为

```
void update_lazy(int node)
{
    segTree[node][4] = 0;//清除标记

    if (segTree[node][5] != 1)//不是叶子节点则下传标记
    {
        tag(node * 2);
        tag(node * 2 + 1);
    }
}
```

它的功能为清除标记以及为左右节点打标记

关键

要注意的是如果当前区间不被修改区间包括， 并且当前节点有lazy标记，则下传标记， 顺序不能错， 顺序错的话会带来性能损失。

查询

区间查询整体差不多，不过查询时碰上标记需要下传，同样的下传标记的函数需要卸载返回结果的前面

```
vector<int> query(int node, int l, int r, int ql, int qr)
{
    vector<int> r1(6, 0), r2(6, 0);
    int mid = (l + r) / 2;
    if (ql <= l && r <= qr)//被包含直接返回结果
        return segTree[node];
    if (segTree[node][4] == 1)//同样的下传标记卸载返回结果的前面
        update_lazy(node);
    if (ql <= mid)//查询左子树
        r1 = query(node * 2, l, mid, ql, qr);
    if (mid < qr)//查询右子树
        r2 = query(node * 2 + 1, mid + 1, r, ql, qr);
    return update(r1, r2);//根据r1,  r2计算结果
}
```

以上