线段树

将数组拆分为一整个二叉树，每个节点代表一个区间，每个节点的值为区间内的值的和，这样就可以在`O(logn)`的时间内完成区间修改和区间查询。
线段树的建树过程是`O(n)`的，因为每个节点都要遍历一遍，但是线段树的修改和查询都是`O(logn)`的，因为每次都会将区间拆分为两个子区间，所以最多只会遍历`O(logn)`个节点。
线段树的区间修改和区间查询都是`O(logn)`的，因为每次都会将区间拆分为两个子区间，所以最多只会遍历`O(logn)`个节点。
这里主要实现的是线段树的区间加减，以及查询区间之和，根据题目意思可以有更多的操作，比如区间最大值，区间最小值，区间最大公约数等等。

```
//C++ code

#include<bits/stdc++.h>

using namespace std;
#define ll long long
struct tree {
    int l, r;
    long long sum, lazy_tag;
};

const int maxn = 1e5 + 5;
tree t[4 * maxn + 3];
int a[maxn + 3];

void build(int position, int l, int r) {
    t[position].l = l;
    t[position].r = r;
    if (l == r) {
        t[position].sum = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(position * 2, l, mid);
    build(position * 2 + 1, mid + 1, r);
    t[position].sum = t[position * 2].sum + t[position * 2 + 1].sum;//建树时，当前节点的和为左右子节点的和
}//建树

void spread(int position) {
    if (t[position].lazy_tag) {//有就传递，没有就不传递
        t[position * 2].sum += t[position].lazy_tag * (t[position * 2].r - t[position * 2].l + 1);
        t[position * 2 + 1].sum += t[position].lazy_tag * (t[position * 2 + 1].r - t[position * 2 + 1].l + 1);
        t[position * 2].lazy_tag += t[position].lazy_tag;
        t[position * 2 + 1].lazy_tag += t[position].lazy_tag;
        t[position].lazy_tag = 0;//向下传递后父节点lazy_tag清零
    }
}//传递lazy_tag

void change(int position, int l, int r, int k) {
```

```
    if (l <= t[position].l && r >= t[position].r) {//如果当前区间被包含在修改区间内，直接修改当前区间的和
        t[position].sum += (ll) k * (t[position].r - t[position].l + 1);
        t[position].lazy_tag += k;
        return;
    }
    spread(position);//传递lazy_tag
    int midd = (t[position].l + t[position].r) >> 1;//对于线段树上进行修改时，使用树上的区间来判断
    if (l <= midd) change(position * 2, l, r, k);
    if (r > midd) change(position * 2 + 1, l, r, k);
    t[position].sum = t[position * 2].sum + t[position * 2 + 1].sum;
}//区间更改&更新lazy_tag

long long ask(int position, int l, int r) {
    if (l <= t[position].l && r >= t[position].r) return t[position].sum;//如果当前区间被包含在查询区间内，直接返回当前区间的和
    spread(position);//传递lazy_tag
    int midd = (t[position].l + t[position].r) >> 1;
    long long ans = 0;
    if (l <= midd) ans += ask(position * 2, l, r);
    if (r > midd) ans += ask(position * 2 + 1, l, r);
    return ans;
}//用递归区间查询

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    build(1, 1, n);
    while (m--) {
        int op, x, y, k;
        cin >> op >> x >> y;
        if (op == 1) {
            cin >> k;
            change(1, x, y, k);
        } else {
            cout << ask(1, x, y) << endl;
        }
    }
    return 0;
}
```

树内元素改变：

样例#1:

```
5 5
1 5 4 2 3
2 2 4
1 2 3 2
```

```
    2 3 4
    1 1 5 1
    2 1 4

初始建树后:   15   10   5   6   4   2   3   1   5   0
第一次更改:   19   14   5   8   6   2   3   1   7   0
第二次更改:   24   14   5   8   6   2   3   1   7   0
第三次更改:   24   17   7   8   6   3   4   1   7   0
下标对应区间: 1,5  1,3 4,5 1,2 3,3 4,4 5,5 1,1 2,2  0,0
```