线段树

基本属性

- 作用简介，基本特性。
  - 维护区间信息的数据结构。
    - 区间修改$O(logN)$ 单点修改，区间修改。
    - 区间查询$O(logN)$实现区间求和，区间最大值，区间最小值。
  - 维护信息特点
    - 区间形式的数据。具有明确的前后关系。
    - 很多时候，维护的代数系统是一个半群。
      - 封闭性
      - 结合律
      - 存在幺元
      - 例如整数域上的乘法，加法。max，min运算。

详解

- 建树原理以及过程：

```
1. 思想 ，线段树就是一个二叉树。
2.从最上面的结点开始，第一个节点管理一整棵树
                   第一个的两个子节点分别管理前半段和后半段
                   一直递归下去，直到一个节点的对应的区间长
     度为1.
3.细节实现问题:
 3.1 编号， 根的编号就是管理了全一段的信息的编号为1.
 对于任何一个父节点，其子节点都满足如下关系；
 son1=2*fa  son2=2*fa+1.
 这样递归的同时标记就行。直接考虑递归建树。管理子节点的线程直接   发返回，关于子
节点的信息。。
 3.2 关于树的一些基本属性的研究
    树的节点个数： 1+2+....2(logn)向下取整。
    树的深度  log n. 1 2 4 ... n;2^deepth=n; deepth=(logn)向上取整。
```

$$2^{log(\lceil n\rceil)+1}-1$$

防止出错省事，直接开到最大就行。或者确定具体值的大小是多少。

code

```
void build(int l,int r,int p){
    //对[l,r]区间建立线段树，当前根的编号为p;
    if(l==r){
        d[p]=a[l];
        return ;
    }
    int m=l+((t-s)>>1);
    //注意优先级。移位运算符的优先级小于加减法。
    //如果写成（s+t）>>1也许会超出范围。
    build(l,m,p*2),build(m+1,r,p*2+1);
    d[p]=d[p*2]+d[p*2+1];
}
```

应用以及拓展。

- 区间查询

```
可以发现，线段树已经维护管理了一些区间。
而我们查询的区间可以这直接由这些节点得到或者拼接得到。最多拆分为logn个区间。算法复杂度
O（logn）
具体怎么成一个比较大的节点?
直接模拟一遍区间的划分过程，先碰到的肯定是更大的。
遍历过程中搜索到的节点信息由有三种
1.查询区间完全包含。
  直接返回该贡献。
2.有交集，但是并不全包含。
  继续往下深搜分割该区间。必然会到1
3.完全没有交集，返回贡献0，不再向下递归，查找是否有贡献。
```

code

```
int getsum(int l, int r, int s, int t,int p)
{
    //查询左，区间，右区间，当前节点区间的左右边界，当前节点的标记。
    if(l<=s&&r>=t)return d[p];//当前节点，可以直接返回贡献。
    int m=s+((t-s)>>1),sum=0;//分割子区间。
    if(l<=m)sum+=getsum(l,r,s,m,p*2);//左，区间是否可以贡献
    if(r>m)sum+=getsum(l,r,m+1,t,p*2+1);//右区间是否可贡献。
    return sum;
}
```

线段树的区间修改，和懒惰标记

- 过程详解；
  - 最暴力方法

    区间修改就把所有和该区间有关的节点都遍历一次修改，复杂度达到n；
  - 优化思想（为什么会大幅度提高效率，该角度的本质是什么?
    - 实际上，某些几点的信息在过程中不被使用。更新等于浪费算力。
    - 父节点可以直接的访问，管理子节点。并且，每一次访问一个节点，必然先经历它的父节点。（如果存在。）

      ```
      类比一个管理系统。有若干个小区。每一个小区有若干栋，每栋又有若干层，每一层又有若干房间。
      现在我们是管理一个送水的系统。某一栋楼要增加1桶水。我们只需要在楼管中记录1.表示当前这一栋楼都要送一桶水。
      现在又确定，某一层楼要送1桶水。直接在楼管中打标签即可。
      并不需要，每一次更改的时候。就详细分配到具体的房间中打标记来更新某一个房间的水的情况。
      当我们再次的访问时就顺便将信息下放即可。
      这样管理起来，效率显然更高。
      ```

      - 直接用空间换时间，增加关于一个节点的属性，当前节点所管理的区间中，区间变化的记录。

code

```
void update(int l, int r ,int c, int s, int t, int p){
    //[l,r]为修改区间，c为被修改的元素的变化量，[s,t]为当前节点
    if(l<=s&&t<=r){
        d[p]+=(t-s+1)*c , b[p]+=c;
        return;
    }
    int m=s+((t-s)>>1);
    if(b[p]&&s!=t){
        d[p*2]+=b[p]*(m-s+1),d[p*2+1]+=b[p]*(t-m);
        b[p*2]+=b[p],b[p*2+1]+=b[p];
        b[p]=0;
    }
    if(l<=m) update(l, r, c, s, m, p*2);
    if(r>m) update(l, r, c, m+1, t, p*2+1);
    d[p]=d[p*2]+d[p*2+1];//再次进行更新。
}
```

区间查询（区间求和）：

- 简介
    - 和上面代码进行对比。引入了区间的懒惰标记。及时更新
- **code**

```
int getsum(int l, int r ,int s, int t, int p){
    if(l<=s&& t<=r)return d[p];
    int m=s+((t-s)>>1);
    if(b[p]){
        d[p*2]+=b[p]*(m-s+1),d[p*2+1]+=b[p]*(t-m);
        b[p*2]+=b[p],b[p*2+1]+=b[p];
        b[p]=0;
    }
    int sum=0;
    if(l<=m) sum=getsum(l,r,s,m,p*2);
    if(r>m) sum+=getsum(l,r,m+1,t,p*2+1);
    return sum;
}
```

拓展（区间修改的方式，将区间上的所有的数字都变为同一个值。）

- 直接将懒惰标记和d之间的关系变化一下就行。

```
void update(int l, int r int c, int s, int t, int p){
    //[l,r]为修改区间，c为被修改的元素的变化量，[s,t]为当前节点
    if(l<=s&&t<=r){
        d[p]=(t-s+1)*c , b[p]=c,v[p]=1;
        return;
    }
    int m=s+((t-s)>>1);
    if(v[p]){
        d[p*2]=b[p]*(m-s+1),d[p*2+1]=b[p]*(t-m);
        b[p*2]=b[p],b[p*2+1]=b[p];
        v[p]=0;
        v[p*2]=v[p*2+1]=1;
    }
    if(l<=m) update(l, r, c, m, p*2);
    if(r>m) update(l,r,c,m+1,t,p*2+1);
    d[p]=d[p*2]+d[p*2+1];//再次进行更新。
}

int getsum(int l,int r,int s,int t,int p){
    //[l,r],表示区间，[s,t]表示当前访问节点的区间，p表示此时区间的标记。
    if(l<=s,t<=r)return d[p];//到一个节点时，该节点的信息已经给更新。
    return d[p];
    int m=s((t-s)>>1);
    if(v[p]){
        d[p*2]=b[p]*(m-s+1),d[p*2+1]=d[p]*(t-m);
```

```
            b[p]=b[p*2+1]=b[p];
            v[p*2]=v[p*2+1]=1;
            v[p]=0;
        }
        int sum=0;
        if(l<=m) sum=getsum( l, r, m+1, t, p*2+1);
        if(r>m)  sum+=getsum(l ,r ,m+1 ,t ,p*2+1);
        return sum;
    }
```

剩下的自己解决的问题

- 将该数据结构，用类封装。
    - 注意定义线段树对象为，全局对象，局部对象不能开那么大的数组。

code_first

```
#include <iostream>
using namespace std;

const int maxn = 1e5 + 10;
int a[maxn];
int n;

typedef long long ll;
class segement_tree
{
public:
    //写一个线段树板子并且用类语法做好封装
    ll d[maxn << 2];                //乘4;
    ll b[maxn << 2];                //这里选择标记。
    void _build(int s, int t, int p) //建树函数
    {
        if (s == t)
        {
            d[p] = a[s];
            return;
        }
        int m = s + ((t - s) >> 1); //计算中点。
        _build(s, m, p * 2);
        _build(m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1];
    }
    void update(int l, int r, int c, int s, int t, int p)
    {
```

```
        //[l,r]为修改区间，c为被修改的元素的变化量，[s,t]为当前节点
        if (l <= s && t <= r)
        {
            d[p] += (t - s + 1) * c, b[p] += c;
            return;
        }
        int m = s + ((t - s) >> 1);
        if (b[p] && s != t)
        {
            d[p * 2] += b[p] * (m - s + 1), d[p * 2 + 1] += b[p] * (t -
m);
            b[p * 2] += b[p], b[p * 2 + 1] += b[p];
            b[p] = 0;
        }
        if (l <= m)
            update(l, r, c, s, m, p * 2);
        if (r > m)
            update(l, r, c, m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1]; //再次进行更新。
    }
    ll getsum(int l, int r, int s, int t, int p)
    {
        if (l <= s && t <= r)
            return d[p];
        int m = s + ((t - s) >> 1);
        if (b[p])
        {
            d[p * 2] += b[p] * (m - s + 1), d[p * 2 + 1] += b[p] * (t -
m);
            b[p * 2] += b[p], b[p * 2 + 1] += b[p];
            b[p] = 0;
        }
        ll sum = 0;
        if (l <= m)
            sum = getsum(l, r, s, m, p * 2);
        if (r > m)
            sum += getsum(l, r, m + 1, t, p * 2 + 1);
        return sum;
    }
} tree1;
int main()
{
    int q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    tree1._build(1, n, 1);
    for (int i = 1; i <= q; i++)
    {
```

```
        int choice;
        cin >> choice;
        if (choice == 1)
        {
            int l, r, k;
            cin >> l >> r >> k;
            tree1.update(l, r, k, 1, n, 1);
        }
        else
        {
            int l, r;
            cin >> l >> r;
            cout << tree1.getsum(l, r, 1, n, 1) << '\n';
        }
    }
}
```

进化角度

- 使封装程度更高。
- 函数的调用更加简洁。优化调用使用的参数列表。
- 引入模板。
- 引入一个完全的复结构，不需要递归函数时都去定义一些变量来表示每个节点所管理的区间。

引入模板，以及优化接口：

```
typedef long long ll;
template <class T>
const int maxn=1e5+10;
class segement_tree{
private:
    //写一个线段树板子并且用类语法做好封装
    T d[maxn << 2]; //乘4;
    T b[maxn << 2]; //这里选择标记。
    void _build(int s, int t, int p) //建树函数
    {
        if (s == t){
            d[p] = a[s];
            return;
        }
        int m = s + ((t - s) >> 1); //计算中点。
        _build(s, m, p * 2);
        _build(m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1];
    }
    void _update(int l, int r, int c, int s, int t, int p)
```

```
    {
        //[l,r]为修改区间，c为被修改的元素的变化量，[s,t]为当前节点
        if (l <= s && t <= r)
        {
            d[p] += (t - s + 1) * c, b[p] += c;
            return;
        }
        int m = s + ((t - s) >> 1);
        if (b[p] && s != t)
        {
            d[p * 2] += b[p] * (m - s + 1), d[p * 2 + 1] += b[p] * (t -
m);
            b[p * 2] += b[p], b[p * 2 + 1] += b[p];
            b[p] = 0;
        }
        if (l <= m)
            _update(l, r, c, s, m, p * 2);
        if (r > m)
            _update(l, r, c, m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1]; //再次进行更新。
    }
    T _getsum(int l, int r, int s, int t, int p)
    {
        if (l <= s && t <= r)
            return d[p];
        int m = s + ((t - s) >> 1);
        if (b[p])
        {
            d[p * 2] += b[p] * (m - s + 1), d[p * 2 + 1] += b[p] * (t -
m);
            b[p * 2] += b[p], b[p * 2 + 1] += b[p];
            b[p] = 0;
        }
        T sum = 0;
        if (l <= m)
            sum = _getsum(l, r, s, m, p * 2);
        if (r > m)
            sum += _getsum(l, r, m + 1, t, p * 2 + 1);
        return sum;
    }
public:
    void build() //建树函数的中间层。
    {
        _build(1, n, 1);
    }
    void update(int l, int r, int c)
    {
        _update(l, r, c, 1, n, 1);
    }
```

```
    T getsum(int l, int r)
    {
        return _getsum(l, r, 1, n, 1);
    }
};
//定义一棵树。
segement_tree<ll> tree1;
```

若干折磨后，出错点总在，运用的一些量的身份的准确的定位上。

```
///////////////////////
int a[maxn];
int n;
typedef long long ll;
template <class T>
class segement_tree
{
private:
    //写一个线段树板子并且用类语法做好封装
    struct inter {
        int l;
        int r;
    };
    vector<ll> d;   //乘4;
    vector<ll> b;   //这b里选择标记。
    T l, r, c, ans; //分别表示区间，以及查询结果。
    vector<inter> _lr;
    void _build(int s, int t, int p) {
        _lr[p].l = s,  _lr[p].r = t;
        if (s == t){
            d[p] = a[s];
            return;
        }
        int m = s + ((t - s) >> 1); //计算中点。
        _build(s, m, p * 2);
        _build(m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1];
    }
    void _update(int p){
        //[l,r]为修改区间，c为被修改的元素的变化量，[s,t]为当前节点
        if (l <= _lr[p].l && _lr[p].r <= r){
            d[p] += (_lr[p].r - _lr[p].l + 1) * c, b[p] += c;
            return;
        }
        int m = _lr[p].l + ((_lr[p].r - _lr[p].l) >> 1);
        if (b[p] && _lr[p].l != _lr[p].r){
```

```
            d[p * 2] += b[p] * (m - _lr[p * 2].l + 1), d[p * 2 + 1] += b[p] * (_lr[p * 2 + 1].r - m);
            b[p * 2] += b[p], b[p * 2 + 1] += b[p];
            b[p] = 0;
        }
        if (l <= m) _update(p * 2);
        if (r > m) _update(p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1]; //再次进行更新。
    }
    void _getsum(int p){
        if (l <= _lr[p].l && _lr[p].r <= r){
            ans += d[p];
            return;
        }
        int m = _lr[p].l + ((_lr[p].r - _lr[p].l) >> 1);
        if (b[p]){
            d[p * 2] += b[p] * (m - _lr[p * 2].l + 1), d[p * 2 + 1] += b[p] * (_lr[p * 2 + 1].r - m);
            b[p * 2] += b[p], b[p * 2 + 1] += b[p];
            b[p] = 0;
        }
        if (l <= m) _getsum(p * 2);
        if (r > m) _getsum(p * 2 + 1);
    }

public:
    void build(){
        _lr.resize(n * 4 + 1);
        d.resize(n * 4 + 1);
        b.resize(n * 4 + 1);
        _build(1, n, 1);
    }
    void update(int s, int t, int k){
        l = s, r = t, c = k;
        _update(1);
    }
    T getsum(int s, int t){
        l = s, r = t;
        ans = 0;
        _getsum(1);
        return ans;
    }
};
//定义一棵树。
segement_tree<ll> tree1;
int main()
{
    int q;
    cin >> n >> q;
```

```cpp
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    tree1.build();
    for (int i = 1; i <= q; i++)
    {
        int choice;
        cin >> choice;
        if (choice == 1)
        {
            int l, r, k;
            cin >> l >> r >> k;
            tree1.update(l, r, k);
        }
        else
        {
            int l, r;
            cin >> l >> r;
            cout << tree1.getsum(l, r) << '\n';
        }
    }
}
```