关于模板的打造

- 学习一些更加系统的命名方式：
- 学习dls的代码的可拓展性的概念。
- 数据结构上的模板一般不是直接就用的。像线段树往往要根据题意进行一些拓展。

关于细节问题的理解

1. 为什么一次查询修改的复杂度是$log(n)$

   从每一层最多有多少个节点被访问的角度来证明。

2. 为什么数组最大开4*n

   理想上的二叉树最后一层要达到n个节点。

函数设计，以及函数之间的耦合关系：

1. 建树函数 build()
2. 查询函数 query()
3. 更新函数
    1. 单点加
    2. 区间加
    3. 区间修改
    4. 单点修改
4. 懒惰标记管理。
    1. 懒惰标记下传。
    2.

经典问题

1. 区间修改

2. 区间最小值

3.维护各种区间信息

1. 维护区间和
2. 维护区间最小值（维护一在各种修改下，依然可以用优越的复杂度查询区间信息等等。）
3. 维护区间最小值个数。
4. 维护最大子段和mss

segement tree

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int N = 201000;
const ll mod = 1000000007;
```

```
7
8  int n, q;
9  int a[N];
10
11 struct tag {
12     //维护标记。
13 };
14
15 //标记合并。用于updatetag
16 tag operator + (const tag &t1, const tag &t2) {
17     // (x * t1.mul + t1.add) * t2.mul + t2.add
18     return {t1.mul * t2.mul % mod, (t1.add * t2.mul + t2.add) % mod};
19 }
20
21 //节点的内容。数据项以及内容。
22 //如果维护信息比较复杂。也可以考虑将信息封装。并且如上写一个区间信息合并重载函数。
23 struct node {
24     tag t;
25     ll val;
26     int sz;
27 } seg[N * 4];
28
29 // [l, r]
30
31 //在modify以及建树之后。把两个儿子的信息合并。
32 void update(int id) {
33     seg[id].val = (seg[id * 2].val + seg[id * 2 + 1].val) % mod;
34 }
35
36 //push down。
37 //完成两项东西。合并标记。
38 //修改区间信息。
39 void settag(int id, tag t) {
40     seg[id].t = seg[id].t + t;
41     seg[id].val = (seg[id].val * t.mul + seg[id].sz * t.add) % mod;
42 }
43 //记得将下放后将标记初始化。
44 void pushdown(int id) {
45     if (seg[id].t.mul != 1 || seg[id].t.add != 0) { // 标记非空
46         settag(id * 2, seg[id].t);
47         settag(id * 2 + 1, seg[id].t);
48         seg[id].t.mul = 1;
49         seg[id].t.add = 0;
50     }
51 }
52
53 //建树。记得建完之后ypdate.
54 //以及到达终点时，将节点信息修正。
55 void build(int id, int l, int r) {
56     seg[id].t = {1, 0};
57     seg[id].sz = r - l + 1;
58     if (l == r) {
59         seg[id].val = {a[l]};
60     } else {
61         int mid = (l + r) / 2;
```

```
        build(id * 2, l, mid);
        build(id * 2 + 1, mid + 1, r);
        update(id);
    }
}

// 节点为id，对应的区间为[l, r]，修改a[pos] -> val
//记得update
//正确sertag
void modify(int id, int l, int r, int ql, int qr, tag t) {
    if (l == ql && r == qr) {
        settag(id, t);
        return;
    }
    int mid = (l + r) / 2;
    // 重要!!
    pushdown(id);
    if (qr <= mid) modify(id * 2, l, mid, ql, qr, t);
    else if (ql > mid) modify(id * 2 + 1, mid + 1, r, ql,qr, t);
    else {
        modify(id * 2, l, mid, ql, mid, t);
        modify(id * 2 + 1, mid + 1, r, mid + 1, qr, t);
    }
    // 重要!!
    update(id);
}
// [ql, qr]表示查询的区间
//到达终点时及时返回。
ll query(int id, int l, int r, int ql, int qr) {
    if (l == ql && r == qr) return seg[id].val;
    int mid = (l + r) / 2;
    // 重要!!
    pushdown(id);
    if (qr <= mid) return query(id * 2, l, mid, ql, qr);
    else if (ql > mid) return query(id * 2 + 1, mid + 1, r, ql,qr);
    else {
        // qr > mid, ql <= mid
        // [ql, mid], [mid + 1, qr]
        return (query(id * 2, l, mid, ql, mid) +
            query(id * 2 + 1, mid + 1, r, mid + 1, qr)) % mod;
    }
}

```

使用tips：

1. 该代码处理的问题是： 同时维护区间加，区间改变，区间乘法三种操作。
2. 当遇到一个问题时，一个良好的修正是。

维护信息 - 》 build -》updata - > 区间信息合并函数

tag - > modify - 》 push_down - >settag

query - > 返回类型，修改内容等等。信息合并的需求等等。

http://oj.daimayuan.top/course/15/problem/654

除了一些实现bug。

代码写的很臭；

主要看几份代码

比较以及生长

1. 代码长度上，为什么能够做到两倍：

因为自己没有写多了一些其它的懒惰标记维护。

另外，关于递归后，处理两个儿子时。其实进行的是两个区间信息的合并操作。这个操作在build和modify，change 都存在。

所以可以进行一个封装。加法封装。

一般而言，线段树只是管理者一个区间。因此不太需要引入一个类。面向过程即可。

目标是理解这种封装角度，以及在这种高度封装的模板上完成迁移，信息维护利用等等。

数据结构 应该这样学。用这样一套东西，面对新问题的时候知道该删哪改哪。这样就理解了数据结构的成员，行为。

关于节点的定义：

线段树节点的定义。

```
struct info{
    int minva;
    int micnt;
};

struct node{
    info val;
    type lazy;//懒惰标记
};
```

当前问题dls的代码

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int N = 201000;

```

```cpp
int n, q;
int a[N];

struct info {
    int minv, mincnt;
};

info operator + (const info &l, const info &r) {
    info a;
    a.minv = min(l.minv, r.minv);
    if (l.minv == r.minv) a.mincnt = l.mincnt + r.mincnt;
    else if (l.minv < r.minv) a.mincnt = l.mincnt;
    else a.mincnt = r.mincnt;
    return a;
}

struct node {
    info val;
} seg[N * 4];

// [l, r]

void update(int id) {
    seg[id].val = seg[id * 2].val + seg[id * 2 + 1].val;
}

void build(int id, int l, int r) {
    if (l == r) {
        seg[id].val = {a[l], 1};
    } else {
        int mid = (l + r) / 2;
        build(id * 2, l, mid);
        build(id * 2 + 1, mid + 1, r);
        update(id);
    }
}

// 节点为id，对应的区间为[l, r]，修改a[pos] -> val
void change(int id, int l, int r, int pos, int val) {
    if (l == r) {
        seg[id].val = {val, 1};
    } else {
        int mid = (l + r) / 2;
        if (pos <= mid) change(id * 2, l, mid, pos, val);
        else change(id * 2 + 1, mid + 1, r, pos, val);
        // 重要!!
        update(id);
    }
}
// [ql, qr]表示查询的区间
info query(int id, int l, int r, int ql, int qr) {
    if (l == ql && r == qr) return seg[id].val;
    int mid = (l + r) / 2;
    // [l, mid] , [mid + 1, r]
    if (qr <= mid) return query(id * 2, l, mid, ql, qr);
```

```
    else if (ql > mid) return query(id * 2 + 1, mid + 1, r, ql,qr);
    else {
        // qr > mid, ql <= mid
        // [ql, mid], [mid + 1, qr]
        return query(id * 2, l, mid, ql, mid) +
            query(id * 2 + 1, mid + 1, r, mid + 1, qr);
    }
}

int main() {
    scanf("%d%d", &n, &q);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
    }
    build(1, 1, n);
    for (int i = 0; i < q; i++) {
        int ty;
        scanf("%d", &ty);
        if (ty == 1) {
            int x, d;
            scanf("%d%d", &x, &d);
            change(1, 1, n, x, d);
        } else {
            int l, r;
            scanf("%d%d", &l, &r);
            auto ans = query(1, 1, n, l, r);
            printf("%d %d\n", ans.minv, ans.mincnt);
        }
    }
}
```

典型求区间和的问题：dls code

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int N = 201000;

int n, q;
int a[N];

struct node {
    int minv;
} seg[N * 4];

// [l, r]

void update(int id) {
    seg[id].minv = min(seg[id * 2].minv, seg[id * 2 + 1].minv);
}
```

```
19
20  void build(int id, int l, int r) {
21      if (l == r) {
22          seg[id].minv = a[l];
23      } else {
24          int mid = (l + r) / 2;
25          build(id * 2, l, mid);
26          build(id * 2 + 1, mid + 1, r);
27          update(id);
28      }
29  }
30
31  // 节点为id，对应的区间为[l, r]，修改a[pos] -> val
32  void change(int id, int l, int r, int pos, int val) {
33      if (l == r) {
34          seg[id].minv = val;
35      } else {
36          int mid = (l + r) / 2;
37          if (pos <= mid) change(id * 2, l, mid, pos, val);
38          else change(id * 2 + 1, mid + 1, r, pos, val);
39          // 重要!!
40          update(id);
41      }
42  }
43  // [ql, qr]表示查询的区间
44  int query(int id, int l, int r, int ql, int qr) {
45      if (l == ql && r == qr) return seg[id].minv;
46      int mid = (l + r) / 2;
47      // [l, mid] , [mid + 1, r]
48      if (qr <= mid) return query(id * 2, l, mid, ql, qr);
49      else if (ql > mid) return query(id * 2 + 1, mid + 1, r, ql,qr);
50      else {
51          // qr > mid, ql <= mid
52          // [ql, mid], [mid + 1, qr]
53          return min(query(id * 2, l, mid, ql, mid),
54              query(id * 2 + 1, mid + 1, r, mid + 1, qr));
55      }
56  }
57
58  int main() {
59      scanf("%d%d", &n, &q);
60      for (int i = 1; i <= n; i++) {
61          scanf("%d", &a[i]);
62      }
63      build(1, 1, n);
64      for (int i = 0; i < q; i++) {
65          int ty;
66          scanf("%d", &ty);
67          if (ty == 1) {
68              int x, d;
69              scanf("%d%d", &x, &d);
70              change(1, 1, n, x, d);
71          } else {
72              int l, r;
73              scanf("%d%d", &l, &r);
```

```
            printf("%d\n", query(1, 1, n, l, r));
        }
    }
}
```

我的臭代码

```
//维护区间中最小值出现的次数
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 2E5 + 10;


//信息不够紧凑。
int a[N];
//维护信息
int mi[N << 2]; //维护区间中的最小值。
int c[N << 2]; //维护最小值出现的次数。
//修改管理。

int lz[N << 2];//懒惰标记。


//传递子树的信息。
//收集子树的信息。

//建树函数里面主要完成几种功能。
//一直往下递归。
//返回子区间信息
//整理两个子区间信息。称为合并操作。
void build(int no , int l , int r)
{
    if (l == r) {
        c[no] = 1;
        mi[no] = a[r];
        return;
    }
    int mid = (l + r) >> 1;
    build(no << 1, l, mid );
    build(no << 1 | 1, mid + 1, r);
    if (mi[no << 1] == mi[no << 1 | 1])
    {
        c[no] = c[no << 1] + c[no << 1 | 1];
        mi[no] = mi[no << 1];
    }
    else if (mi[no << 1] < mi[no << 1 | 1])
    {
        c[no] = c[no << 1];
        mi[no] = mi[no << 1];
    }
    else
```

```
    {
        c[no] = c[no << 1 | 1];
        mi[no] = mi[no << 1 | 1];
    }
}

void pd(int no, int l, int r) // 区间信息管理的节点编号。//左右区间。
{
    int mid = (l + r) >> 1;
    lz[no << 1] = lz[no << 1 | 1] = lz[no];
    c[no << 1] = (mid - l + 1);
    c[no << 1 | 1] = (r - mid);
    mi[no << 1] = mi[no << 1 | 1] = lz[no];
    lz[no] = 0;
}

void set_tag(int no , int l, int r, int k)
{
    lz[no] = k;
    mi[no] = k;
    c[no] = r - l + 1;
}
void modify(int no, int l, int r, int ql, int qr, int k)
{
    if (l >= ql && r <= qr)
    {
        set_tag(no, l, r, k);
        return;
    }
    if (lz[no]) {pd(no, l, r);}
    int mid = (l + r) >> 1;

    if (l <= qr && mid >= ql)
        modify(no << 1, l, mid, ql, qr, k);
    if ( mid + 1 <= qr && r >= ql)
        modify(no << 1 | 1, mid + 1, r, ql, qr, k);

    //进行更新：
    if (mi[no << 1] == mi[no << 1 | 1])
    {
        mi[no] = mi[no << 1];
        c[no] = c[no << 1] + c[no << 1 | 1];
    }
    else if (mi[no << 1] < mi[no << 1 | 1])
    {
        mi[no] = mi[no << 1];
        c[no] = c[no << 1];
    }
    else
    {
        mi[no] = mi[no << 1 | 1];
        c[no] = c[no << 1 | 1];
    }
}

```

```
102
103
104 //感觉这个查询不大方便。
105 //其实就是分成若干个块最终会分为若干个块。怎么处理这若干个块的信息呢？
106 //简单return一个int并不可以。因为合并的选择的时候要关注两个量。
107 //直接在全局里面定位两个？
108 struct node {
109     int mi;
110     int sum;
111 };
112 node query(int no  , int l, int r , int ql , int qr)
113 {
114     if (l >= ql && r <= qr)
115     {
116         return {mi[no], c[no]};
117     }
118     if (lz[no])
119         pd(no, l, r);
120     int mid = (l + r) >> 1;
121
122     node res = {int(1E9 + 10), 0};
123
124     if (l <= qr && mid >= ql)
125         res = query(no << 1, l, mid, ql, qr);
126     if (mid + 1 <= qr && r >= ql)
127     {
128         node temp = query (no << 1 | 1, mid + 1, r , ql , qr);
129         if (temp.mi == res.mi)
130             res.sum += temp.sum;
131         else if (temp.mi < res.mi)
132             res = temp;
133     }
134     return res;
135 }
136
137
138 int main()
139 {
140     ios::sync_with_stdio(false);
141     cin.tie(0);
142
143     int n , m;
144     cin >> n >> m;
145     for (int i = 1; i <= n; i++)
146         cin >> a[i];
147
148     build(1, 1, n);
149     for (int i = 1; i <= m; i++)
150     {
151         int ty, x, y;
152         cin >> ty >> x >> y;
153         if (ty == 1)
154         {
155             modify(1, 1, n, x, x, y);
156         }
```

```
        else {
            auto ans = query(1, 1, n, x, y);
            cout << ans.mi << ' ' << ans.sum  << '\n';
        }
    }
}

/* stuff you should look for
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/
```

一个明智地追求快乐的人，除了培养生活赖以支撑的主要兴趣之外，总得设法培养其他许多闲情逸致。