

# 预备役寒训2

## 树状数组

### 作用:

多用于**单点修改**和**区间查询**，树状数组能解决的问题线段树都能解决，但代码量树状数组远小于线段树，不容易写错，且常数较小

也可以用于求解**区间修改**和**单点查询**

### 复杂度

#### 空间复杂度

$O(n)$ ， $n$  为维护信息的最大值得域

#### 时间复杂度

单点修改、区间查询:  $O(\log_2 n)$

### 要求

所维护的信息需要满足**结合律且可差分**，如加法、乘法和异或等

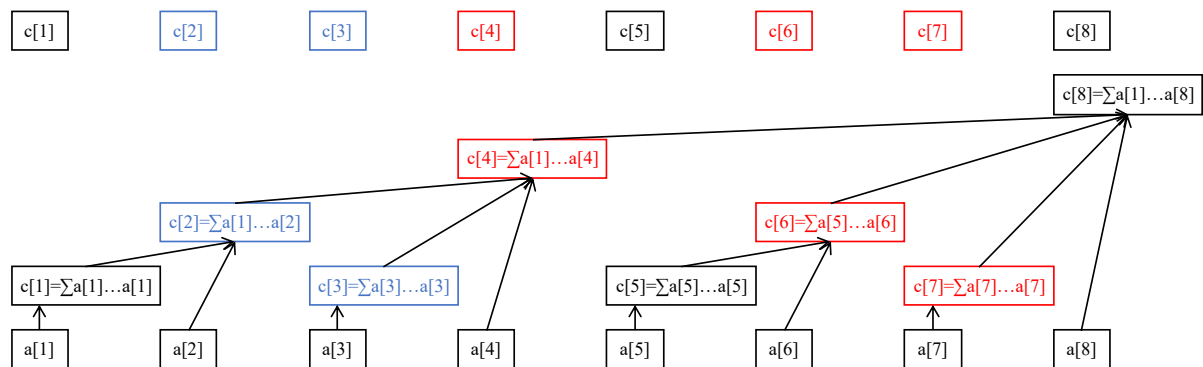
### 树状数组维护的信息（区间和为例）

记  $x$  的二进制最低为 1 所在的位数为  $k$

$lowbit(x)$  表示  $2^k$ ，可以通过位运算知识（反码补码）得到  $lowbit(x) = x \& -x$

树状数组  $tr_x$  表示  $\sum_{i=x-lowbit(x)+1}^x a_i$ ，即区间  $[x - lowbit(x) + 1, x]$  的和，要记录  $tr_1 \sim tr_n$ ，故空间复杂度  $O(n)$

(维护区间乘时需要将  $\sum$  改为  $\prod$  即可，即加法的地方改为乘法)



$$\text{查询 } \sum a[4] \dots a[7] = (\sum a[1] \dots a[7]) - (\sum a[1] \dots a[3])$$

(此树高为  $O(\log_2 n)$ )

当我们要求解  $\sum_{i=1}^n a_i$ ，可以将其拆分成  $\log_2 n$  段区间的和，意义为直接与  $tr_n$  相连的  $tr_x$

例如, 当  $n = (38)_{10} = (100110)_2$  时,

$$\begin{aligned} (38)_{10} &= (100110)_2 \\ \sum_{i=1}^{(38)_{10}} a_i &= \sum_{i=(100100)_2+1}^{(100110)_2} a_i + \sum_{i=(100000)_2+1}^{(100100)_2} a_i + \sum_{i=(000000)_2+1}^{(100000)_2} a_i \\ &= tr_{(100110)_2} + tr_{(100100)_2} + tr_{(100000)_2} \\ &= tr_{38} + tr_{36} + tr_{32} \\ &\quad (36 = 38 - lowbit(38), 32 = 36 - lowbit(38)) \end{aligned}$$

由此可见,  $\forall n \in \mathbb{Z}, \sum_{i=1}^n a_i$  可以拆成的段数等于  $n$  的二进制中 1 的个数

## 区间查询

区间查询使用类似前缀和思路, 求  $\sum_{i=k}^n a_i = \sum_{i=1}^n a_i - \sum_{i=1}^{k-1} a_i$ , 求  $\sum_{i=1}^n a_i$  和  $\sum_{i=1}^{k-1} a_i$  的复杂度均为  $O(\log_2 n)$  故区间查询复杂度为  $O(\log_2 n)$

## 单点修改

考虑如何单点修改  $a_x$ , 我们只需要修改所有管辖了  $a_x$  的  $tr_y$  即可

所有管辖了  $a_x$  的  $tr_y$  即为  $tr_x$  即其所有祖先, 每个祖先可以通过  $y = x + lowbit(x)$  到达

故修改过程为:

- 先令  $x' = x$
- 修改  $tr_{x'}$
- 令  $x' \leftarrow x' + lowbit(x')$ , 若  $x' > n$  跳出循环, 否则返回第二步

由于一直加  $lowbit(x')$ , 最多不会加超过  $\log_2 n$  次, 故单点修改复杂度  $O(\log_2 n)$

(注意 0 一直加  $lowbit(x)$  还是 0, 会死循环, 故建议树状数组从 1 开始建立)

## 代码实现

```
1 vector<int> tr(n + 1);
2
3 auto lowbit = [&](int x) {
4     return x & -x;
5 };
6
7 auto add = [&](int x, int k) {
8     for (; x < tr.size(); x += lowbit(x)) {
9         tr[x] += k;
10    }
11 };
12
13 auto get = [&](int x) {
14     int res = 0;
15     for (; x; x -= lowbit(x)) {
```

```
16     res += tr[x];
17     }
18     return res;
19 };
```

## 离散化

就是当有些数据因为本身很大或者类型不支持，自身无法作为数组的下标来方便地处理，而影响最终结果的只有元素之间的相对大小关系时，我们可以将原来的数据按照排名来处理问题，即离散化，可视作一种哈希

## 复杂度

### 空间复杂度

$O(n)$

### 时间复杂度

$O(n \log_2 n)$

## 原理

先复制原数组记作  $ta$ ，对  $ta$  排序并去重，然后对  $ta$  使用二分得到排名

```
1 auto ta = a;
2
3 sort(ta.begin() + 1, ta.end());
4 ta.erase(unique(ta.begin() + 1, ta.end()), ta.end());
5
6 auto get_rank = [&](int x) {
7     return lower_bound(ta.begin() + 1, ta.end(), x) - ta.begin();
8 };
```

## 线段树

## 作用

常用来维护区间信息，可在  $O(\log_2 n)$  的时间复杂度内实现单点修改，区间修改，区间查询（区间求和，区间最大值，区间最小值）等操作

## 复杂度

### 空间复杂度

$O(n)$

## 时间复杂度

单点修改, 区间修改, 区间查询 (区间求和, 区间最大值, 区间最小值) :  $O(\log_2 n)$

## 线段树的基本结构 (以区间加为例)

使用堆式存储结构, 将每个长度不为 1 的区间划分为左右两个区间递归求解, 并通过合并左右两个子区间得到该区间信息

$d[1]=60$ [1,5]			
$d[2]=33$ [1,3]		$d[3]=27$ [4,5]	
$d[4]=21$ [1,2]	$d[5]=12$ [3,3]	$d[6]=13$ [4,4]	$d[7]=14$ [5,5]
$d[8]=10$ [1,1]	$d[9]=11$ [2,2]		

对于  $d_i$  的左儿子节点就是  $d_{2 \times i}$ , 右儿子节点为  $d_{2 \times i + 1}$ , 用位运算分别为  $d_{i \ll 1}$  和  $d_{i \ll 1 | 1}$

若  $d_i$  管辖区间为  $[l, r]$ , 令  $mid = \lfloor \frac{l+r}{2} \rfloor$ , 则左右儿子管辖区间分别为  $[l, mid]$ ,  $[mid + 1, r]$

最后的叶子节点即为原数组的信息, 由于采用堆式存储, 叶子节点为  $n$  时, 数组范围最大为  $2^{\lceil \log_2 n \rceil + 1} \leq 4n - 5$

(若使用动态开点, 则数组范围最大为  $2n - 1$ )

故空间复杂度为  $O(n)$

## 区间查询

查询区间  $[l, r]$  的信息可以通过将其拆分成最多  $O(\log_2 n)$  个**极大**的区间, 然后合并这些区间得到答案

例如, 求  $\sum_{i=2}^5 a_i = d_3 + d_5 + d_9$

若合并区间信息为  $O(1)$ , 则区间查询复杂度为  $O(\log_2 n)$

## 区间修改与懒标记

若要求修改区间  $[l, r]$ ，要把所有包含区间  $[l, r]$  的区间都修改一次，复杂度太大，我们可以采用懒标记优化

d[1]=60 t[1]=0			
d[2]=33 t[2]=0		d[3]=27 t[3]=0	
d[4]=21 t[4]=0	d[5]=12 t[5]=0	d[6]=13 t[6]=0	d[7]=14 t[7]=0
d[8]=10 t[8]=0	d[9]=11 t[9]=0		

懒标记就是一种**延迟**对节点的修改，每次修改时，仅修改**极大的区间节点**，并在这下节点**增加懒标记**，若在遍历过程中遇到懒标记则将**懒标记下传**至子节点并**修改**子节点，若当前为叶子节点则不下传，回溯过程合并左右儿子信息

例如，当对区间  $[2, 5]$  加上10时， $d_3 = 47$ ， $t_3 = 10$ ， $d_5 = 22$ ， $t_5 = 10$ ， $d_9 = 21$ ， $t_9 = 10$

之后再查询区间  $[4, 4]$  会遍历到  $t_3$  则将  $t_3$  下传操作， $d_3 = 47$ ， $t_3 = 0$ ， $d_6 = 23$ ， $t_6 = 10$ ， $d_7 = 24$ ， $t_7 = 10$ ，查询结果为  $d_6 = 23$

由于每次修改仅更改极大的区间节点，若更新懒标记的复杂度为  $O(1)$  则复杂度为  $O(\log_2 n)$

## 代码实现

### 正常写法 (偷的)

```
1 struct node {
2     int l,r;
3     int lazy,sum;
4 }tr[N*4];
5
6 void pushup(int u){
7     tr[u].sum=tr[u<<1].sum+tr[u<<1|1].sum;
8 }
9
10 void pushdown(int u){
```

```

11     if(tr[u].lazy){
12         tr[u<<1].sum+=(tr[u<<1].r-tr[u<<1].l+1)*tr[u].lazy;
13         tr[u<<1|1].sum+=(tr[u<<1|1].r-tr[u<<1|1].l+1)*tr[u].lazy;
14         tr[u<<1].lazy+=tr[u].lazy;
15         tr[u<<1|1].lazy+=tr[u].lazy;
16         tr[u].lazy=0;
17     }
18 }
19
20 void build(int u,int l,int r){
21     tr[u].l=l;
22     tr[u].r=r;
23     if(l==r){
24         tr[u].sum=a[l];
25         return;
26     }
27     int mid=l+r>>1;
28     build(u<<1,l,mid);
29     build(u<<1|1,mid+1,r);
30     pushup(u);
31 }
32
33 void addtr(int u,int l,int r,int k){
34     if(l<=tr[u].l&&tr[u].r){
35         tr[u].sum+=(tr[u].r-tr[u].l+1)*k;
36         tr[u].lazy+=k;
37         return;
38     }
39     pushdown(u);
40     int mid=tr[u].r+tr[u].l>>1;
41     if(l<=mid) addtr(u<<1,l,r,k);
42     if(r>mid) addtr(u<<1|1,l,r,k);
43     pushup(u);
44 }
45
46 int findtr(int u,int l,int r){
47     if(l<=tr[u].l&&tr[u].r){
48         return tr[u].sum;
49     }
50     pushdown(u);
51     int mid=tr[u].l+tr[u].r>>1;
52     int ans=0;
53     if(l<=mid) ans+=findtr(u<<1,l,r);
54     if(r>mid) ans+=findtr(u<<1|1,l,r);
55     pushup(u);
56     return ans;
57 }

```

## 小封装版

```

1 struct node {
2     int l, r;
3     int sum, add;
4 };
5

```

```

6  vector<node> tr;
7
8  auto merge = [&](node l, node r, node u = {}) { //合并贡献
9      u = {l.l, r.r, l.sum + r.sum};
10     return u;
11 };
12
13 auto pushup = [&](int p) {
14     tr[p] = merge(tr[p << 1], tr[p << 1 | 1], tr[p]); //向上传
15 };
16
17 auto add_down = [&](int k, node & u) { //加法的更新
18     u.add += k;
19     u.sum += (u.r - u.l + 1) * k;
20 };
21
22 auto pushdown = [&](int p) { //标记下传
23     if (tr[p].add) {
24         add_down(tr[p].add, tr[p << 1]), add_down(tr[p].add, tr[p << 1 |
25 1]);
26         tr[p].add = 0;
27     }
28 };
29 auto build = [&](int sz) { //建树
30     tr = vector<node>(sz << 2);
31
32     auto build = [&](auto build, int l, int r, int p) {
33         tr[p] = {l, r};
34         if (l == r) {
35             tr[p].sum = a[r]; //叶子节点等于原数组
36             return;
37         }
38         pushdown(p); //可不用
39         int mid = l + r >> 1;
40         build(build, l, mid, p << 1), build(build, mid + 1, r, p << 1 | 1);
41         pushup(p);
42     };
43
44     build(build, 1, sz, 1);
45 };
46
47 auto update = [&](auto update, int l, int r, int k, int p) { //更新
48     if (l <= tr[p].l && tr[p].r <= r) {
49         add_down(k, tr[p]); //添加标记,更新节点
50         return;
51     }
52     pushdown(p);
53     if (l <= tr[p << 1].r) update(update, l, r, k, p << 1); //左边有交集
54     if (tr[p << 1 | 1].l <= r) update(update, l, r, k, p << 1 | 1); //右边有交
55     pushup(p);
56 };
57
58 auto query = [&](auto query, int l, int r, int p) -> node {
59     if (l <= tr[p].l && tr[p].r <= r) { //被包含

```

```

60     return tr[p];
61 }
62 pushdown(p);
63 if (r <= tr[p << 1].r) return query(query, l, r, p << 1); //只在左边
64 if (tr[p << 1 | 1].l <= l) return query(query, l, r, p << 1 | 1); //只
    在右边
65 return merge(query(query, l, r, p << 1), query(query, l, r, p << 1 |
    1)); //左右都有
66 };

```

## Manacher

### 作用

对于一个长度为  $n$  的字符串（下标从 1 开始，可以找出  $d_i$  分别表示奇回文子串以位置  $i$  开始到最右端位置包含的字符个数，即以  $i$  位置为中心的最长奇回文子串半径（半径包含自己，可能与 oi-wiki 稍有不同）

例如， $s = abababc$ ，以  $s_4 = b$  的奇回文子串半径为  $d_4 = 3$

对于偶回文子串，我们通过对每个字符前后加上 #（任意一个原字符串中未出现过的字符）来转变为奇回文子串，这样我们只需要考虑奇回文子串足以

例如， $s = ababaac$ ，转变后为  $\#a\#b\#a\#b\#a\#a\#c\#$ ， $s_6 = a, d_6 = 6, s_{11} = \#, d_{11} = 3$

此处  $d_i - 1$  意义为以  $i$  位置为中心在**原串中的**最长回文子串长度， $\lfloor \frac{d_i}{2} \rfloor$  为以  $i$  位置为中心在**原串中的**回文子串个数

### 复杂度

#### 空间复杂度

$O(n)$

#### 时间复杂度

$O(n)$

### 算法原理

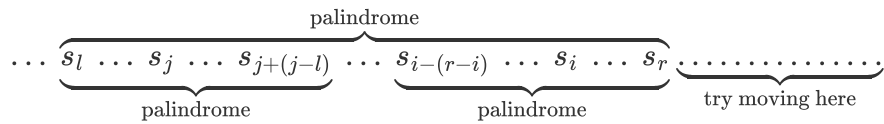
假设  $d_1 \sim d_{i-1}$  已经计算完毕，维护以  $1 \sim i - 1$  为中心的最靠右的回文子串的边界  $[l, r]$  和区间中点  $mid$ ，实际上  $l = 2 \times mid - r$ ，故不记录  $l$

- 若  $i > r$ ，我们暴力计算回文子串的半径，即一步步比较
- 若  $i \leq r$ ，我们考虑从  $[l, r]$  中获取信息
  - 若  $i$  在  $[l, r]$  中能找到反转位置，得到反转位置  $j = l + (r - i) = 2 \times mid - i$ ，故  $d_i = d_j$

$$\dots s_l \dots \underbrace{s_{j-d_1[j]+1} \dots s_j \dots s_{j+d_1[j]-1}}_{\text{palindrome}} \dots \overbrace{s_{i-d_1[j]+1} \dots s_i \dots s_{i+d_1[j]-1}}^{\text{palindrome}} \dots s_r \dots$$

- 若  $i$  在  $[l, r]$  中无法找到反转位置，这种情况我们进行截断操作，先令  $d_i = r - i$ ，因为此长度是最长不会超过  $[l, r]$  的回文子串半径，然后再暴力一步步的增加回文子串半径





- 最后我们更新新的  $[l, r]$ ,  $mid$

注意到时间复杂度瓶颈主要在于暴力比较并计算回文子串半径这里，由于每次暴力比较均会使得  $r$  增加，而  $r$  在算法过程不减小， $r \leq 2 \times n$ ，故时间复杂度为  $O(n)$

## 代码实现

```

1  auto read_string = [&]() {
2      string t, s = "#";
3      cin >> t;
4      for (auto x : t) {
5          s += x, s += '#';
6      }
7      return s;
8  };
9
10 auto check = [&](char a, char b) {
11     if (a == b) {
12         return 1;
13     }
14     return 0;
15 };
16
17 string s = read_string();
18 vector<int> d; //d为包括自己的回文半径
19
20 auto Manacher = [&](string &s) -> void {
21     d = vector<int>(s.size());
22     int mid = 0, r = 0; //mid记录区间中点
23     for (int i = 1; i < s.size(); i++){
24         d[i] = i <= r ? min(d[mid * 2 - i], r - i) : (int)1;
25         while (1 <= i - d[i] && i + d[i] < s.size() && check(s[i - d[i]],
s[i + d[i]])) {
26             d[i] ++;
27         }
28         if (d[i] + i > r) {
29             r = d[i] + i;
30             mid = i;
31         }
32     }
33 };

```

## 练习

1. [P3374 【模板】树状数组 1 - 洛谷 | 计算机科学教育新生态 \(luogu.com.cn\)](https://www.luogu.com.cn/problem/P3374)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define int long long
4
5  vector<int> tr;

```

```

6
7 int lowbit(int x) {
8     return x & -x;
9 }
10
11 void add(int x, int k) {
12     for (int i = x; i < tr.size(); i += lowbit(i)) {
13         tr[i] += k;
14     }
15 }
16
17 int get_sum(int r) {
18     int res = 0;
19     for (int i = r; i; i -= lowbit(i)) {
20         res += tr[i];
21     }
22     return res;
23 }
24
25 void QAQ() {
26     int n, m;
27     cin >> n >> m;
28
29     tr = vector<int>(n + 1);
30
31     for (int i = 1; i <= n; i++) {
32         int x;
33         cin >> x;
34         add(i, x);
35     }
36
37     for (int i = 1; i <= m; i++) {
38         int op, x, y;
39         cin >> op >> x >> y;
40
41         if (op == 2) {
42             cout << get_sum(y) - get_sum(x - 1) << "\n";
43         } else {
44             add(x, y);
45         }
46     }
47 }
48 }
49
50 signed main() {
51     cin.tie(0) -> sync_with_stdio(0);
52     int t = 1;
53     // cin >> t;
54
55     while (t--) {
56         QAQ();
57     }
58 }

```

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define int long long
4
5 void QAQ() {
6     int n, m;
7     cin >> n >> m;
8
9     vector<int> a(n + 1);
10
11     for (int i = 1; i <= n; i++) {
12         cin >> a[i];
13     }
14
15     struct node {
16         int l, r;
17         int sum, add;
18     };
19
20     vector<node> tr;
21
22     auto merge = [&](node l, node r, node u = {}) { //合并贡献
23         u = {l.l, r.r, l.sum + r.sum};
24         return u;
25     };
26
27     auto pushup = [&](int p) {
28         tr[p] = merge(tr[p << 1], tr[p << 1 | 1], tr[p]); //向上传
29     };
30
31     auto add_down = [&](int k, node & u) { //加法的更新
32         u.add += k;
33         u.sum += (u.r - u.l + 1) * k;
34     };
35
36     auto pushdown = [&](int p) { //标记下传
37         if (tr[p].add) {
38             add_down(tr[p].add, tr[p << 1]), add_down(tr[p].add, tr[p
39 << 1 | 1]);
40             tr[p].add = 0;
41         }
42     };
43
44     auto build = [&](int sz) { //建树
45         tr = vector<node>(sz << 2);
46
47         auto build = [&](auto build, int l, int r, int p) {
48             tr[p] = {l, r};
49             if (l == r) {
50                 tr[p].sum = a[r]; //叶子节点等于原数组
51                 return;
52             }
53             pushdown(p); //可不用
54             int mid = l + r >> 1;
55             build(build, l, mid, p << 1), build(build, mid + 1, r, p <<
56 1 | 1);

```

```

55     pushup(p);
56     };
57
58     build(build, 1, sz, 1);
59     };
60
61     auto update = [&](auto update, int l, int r, int k, int p) { //
更新
62         if (l <= tr[p].l && tr[p].r <= r) {
63             add_down(k, tr[p]); //添加标记,更新节点
64             return;
65         }
66         pushdown(p);
67         if (l <= tr[p << 1].r) update(update, l, r, k, p << 1); //左边有
交集
68         if (tr[p << 1 | 1].l <= r) update(update, l, r, k, p << 1 | 1);
//右边有交集
69         pushup(p);
70     };
71
72     auto query = [&](auto query, int l, int r, int p) -> node {
73         if (l <= tr[p].l && tr[p].r <= r) { //被包含
74             return tr[p];
75         }
76         pushdown(p);
77         if (r <= tr[p << 1].r) return query(query, l, r, p << 1); //
只在左边
78         if (tr[p << 1 | 1].l <= l) return query(query, l, r, p << 1 |
1); //只在右边
79         return merge(query(query, l, r, p << 1), query(query, l, r, p
<< 1 | 1)); //左右都有
80     };
81
82     build(n);
83
84     for (int i = 1; i <= m; i++) {
85         int op, x, y;
86         cin >> op >> x >> y;
87
88         if (op == 1) {
89             int k;
90             cin >> k;
91
92             update(update, x, y, k, 1);
93         } else {
94             auto tmp = query(query, x, y, 1);
95
96             cout << tmp.sum << "\n";
97         }
98     }
99 }
100
101 signed main() {
102     cin.tie(0) -> sync_with_stdio(0);
103     int t = 1;
104     // cin >> t;

```

```

105
106     while (t--) {
107         QAQ();
108     }
109 }

```

### 3. P3805 【模板】manacher - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define int long long
4
5  void QAQ() {
6      auto read_string = [&]() {
7          string t, s = " #";
8          cin >> t;
9          for (auto x : t) {
10             s += x, s += '#';
11         }
12         return s;
13     };
14
15     auto check = [&](char a, char b) {
16         if (a == b) {
17             return 1;
18         }
19         return 0;
20     };
21
22     string s = read_string();
23     vector<int> d; //d为包括自己的回文半径
24
25     auto Manacher = [&](string &s) -> void {
26         d = vector<int>(s.size());
27         int mid = 0, r = 0; //mid记录区间中点
28         for (int i = 1; i < s.size(); i++){
29             d[i] = i <= r ? min(d[mid * 2 - i], r - i) : (int)1;
30             while (1 <= i - d[i] && i + d[i] < s.size() && check(s[i -
d[i]], s[i + d[i]])) {
31                 d[i] ++;
32             }
33             if (d[i] + i > r) {
34                 r = d[i] + i;
35                 mid = i;
36             }
37         }
38     };
39
40     Manacher(s);
41     int ans = 0;
42
43     for (int i = 1; i < s.size(); i++) {
44         ans = max(ans, d[i] - 1);
45     }
46
47     cout << ans << "\n";

```

```

48 }
49
50 signed main() {
51     cin.tie(0) -> sync_with_stdio(0);
52     int t = 1;
53     // cin >> t;
54
55     while (t--) {
56         QAQ();
57     }
58 }

```

4. [P3368 【模板】树状数组 2 - 洛谷 | 计算机科学教育新生态 \(luogu.com.cn\)](#)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define int long long
4
5  void QAQ() {
6      int n, m;
7      cin >> n >> m;
8
9      vector<int> tr(n + 1);
10
11     auto lowbit = [&](int x) {
12         return x & -x;
13     };
14
15     auto add = [&](int x, int k) {
16         for ( ; x < tr.size(); x += lowbit(x)) {
17             tr[x] += k;
18         }
19     };
20
21     auto get = [&](int x) {
22         int res = 0;
23         for ( ; x; x -= lowbit(x)) {
24             res += tr[x];
25         }
26         return res;
27     };
28
29     vector<int> a(n + 1);
30
31     for (int i = 1; i <= n; i++) {
32         cin >> a[i];
33         add(i, a[i] - a[i - 1]);
34     }
35
36     for (int i = 1; i <= m; i++) {
37         int op, x;
38         cin >> op >> x;
39
40         if (op == 1) {
41             int y, k;
42             cin >> y >> k;

```

```

43         add(x, k), add(y + 1, -k);
44     } else {
45         cout << get(x) << "\n";
46     }
47 }
48 }
49
50 signed main() {
51     cin.tie(0) -> sync_with_stdio(0);
52     int t = 1;
53     // cin >> t;
54
55     while (t--) {
56         QAQ();
57     }
58 }

```

5. [P1908 逆序对 - 洛谷 | 计算机科学教育新生态 \(luogu.com.cn\)](https://www.luogu.com.cn/problem/P1908)

**归并排序:**

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define int long long
4
5  void QAQ() {
6      int n;
7      cin >> n;
8
9      vector<int> a(n + 1);
10
11     for (int i = 1; i <= n; i++) {
12         cin >> a[i];
13     }
14
15     int ans = 0;
16
17     auto dfs = [&](auto dfs, int s, int t) -> void {
18         if(s >= t) return;
19         int mid = s + t >> 1;
20         dfs(dfs, s, mid), dfs(dfs, mid + 1, t);
21         vector<int> f(a.begin() + s, a.begin() + mid + 1), g(a.begin() +
mid + 1, a.begin() + t + 1);
22         for (int i = 0, j = 0, p = s; i < f.size() || j < g.size(); ) {
23             if(i>=f.size()){
24                 a[p++] = g[j++];
25             } else if (j >= g.size()) {
26                 a[p++] = f[i++];
27             } else if (f[i] <= g[j]) {
28                 a[p++] = f[i++];
29             } else {
30                 a[p++] = g[j++], ans += f.size() - i;
31             }
32         }
33     };
34

```

```

35     dfs(dfs, 1, n);
36
37     cout << ans << "\n";
38 }
39
40 signed main() {
41     cin.tie(0) -> sync_with_stdio(0);
42     int t = 1;
43     // cin >> t;
44
45     while (t--) {
46         QAQ();
47     }
48 }

```

## 树状数组

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define int long long
4
5  void QAQ() {
6      int n;
7      cin >> n;
8
9      vector<int> a(n + 1);
10
11     int ans = 0;
12
13     for (int i = 1; i <= n; i++) {
14         cin >> a[i];
15     }
16
17     auto _ = a;
18
19     sort(_.begin() + 1, _.end());
20     _.erase(unique(_.begin() + 1, _.end()), _.end());
21
22     auto get_rank = [&](int x) {
23         return lower_bound(_.begin() + 1, _.end(), x) - _.begin();
24     };
25
26     vector<int> tr(_.size());
27
28     auto lowbit = [&](int x) {
29         return x & -x;
30     };
31
32     auto add = [&](int x, int k) {
33         for (; x < tr.size(); x += lowbit(x)) {
34             tr[x] += k;
35         }
36     };
37
38     auto get = [&](int x) {

```



```

39     int res = 0;
40     for ( ; x; x -= lowbit(x)) {
41         res += tr[x];
42     }
43     return res;
44 };
45
46 for (int i = 1; i <= n; i++) {
47     add(get_rank(a[i]), 1);
48     ans += get(tr.size() - 1) - get(get_rank(a[i]));
49 }
50
51 cout << ans << "\n";
52 }
53
54 signed main() {
55     cin.tie(0) -> sync_with_stdio(0);
56     int t = 1;
57     // cin >> t;
58
59     while (t--) {
60         QAQ();
61     }
62 }

```

6. [P3373 【模板】线段树 2 - 洛谷 | 计算机科学教育新生态\(luogu.com.cn\)](#)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define int long long
4
5  void QAQ() {
6     int n, q, mod;
7     cin >> n >> q >> mod;
8
9     vector<int> a(n + 1);
10
11     for (int i = 1; i <= n; i++) {
12         cin >> a[i];
13     }
14
15     struct node {
16         int l, r;
17         int sum = 0, add = 0, mul = 1; // 最好赋初始值
18     };
19
20     vector<node> tr;
21
22     auto merge = [&](node l, node r, node u = {}) { //合并贡献
23         u = {l.l, r.r, (l.sum + r.sum) % mod, 0, 1};
24         return u;
25     };
26
27     auto pushup = [&](int p) {
28         tr[p] = merge(tr[p << 1], tr[p << 1 | 1], tr[p]); //向上传
29     };

```

```

30
31     auto add_down = [&](int k, node & u) { //加法的更新
32         (u.add += k) %= mod;
33         (u.sum += (u.r - u.l + 1) * k % mod) %= mod;
34     };
35
36     auto mul_down = [&](int k, node & u) {
37         (u.add *= k) %= mod; //乘法会对加法标记有更新影响
38         (u.mul *= k) %= mod;
39         (u.sum *= k) %= mod;
40     };
41
42     auto pushdown = [&](int p) { //标记下传, 先下传乘法再下传加法
43         mul_down(tr[p].mul, tr[p << 1]), mul_down(tr[p].mul, tr[p << 1
44 | 1]);
45         tr[p].mul = 1;
46
47         add_down(tr[p].add, tr[p << 1]), add_down(tr[p].add, tr[p << 1
48 | 1]);
49         tr[p].add = 0;
50     };
51
52     auto build = [&](int sz) { //建树
53         tr = vector<node>(sz << 2);
54
55         auto build = [&](auto build, int l, int r, int p) {
56             tr[p] = {l, r, 0, 0, 1};
57             if (l == r) {
58                 tr[p].sum = a[r] % mod; //叶子节点等于原数组
59                 return;
60             }
61             pushdown(p); //可不用
62             int mid = l + r >> 1;
63             build(build, l, mid, p << 1), build(build, mid + 1, r, p <<
64 1 | 1);
65             pushup(p);
66         };
67
68         build(build, 1, sz, 1);
69     };
70
71     auto update = [&](auto update, int l, int r, int k, int p) { //
72     更新
73         if (l <= tr[p].l && tr[p].r <= r) {
74             add_down(k, tr[p]); //添加标记,更新节点
75             return;
76         }
77         pushdown(p);
78         if (l <= tr[p << 1].r) update(update, l, r, k, p << 1); //左边有
79     交集
80         if (tr[p << 1 | 1].l <= r) update(update, l, r, k, p << 1 | 1);
81     //右边有交集
82         pushup(p);
83     };
84
85

```

```

79     auto modify = [&](auto modify, int l, int r, int k, int p) { //
更新
80         if (l <= tr[p].l && tr[p].r <= r) {
81             mul_down(k, tr[p]); //添加标记,更新节点
82             return;
83         }
84         pushdown(p);
85         if (l <= tr[p << 1].r) modify(modify, l, r, k, p << 1); //左边有
交集
86         if (tr[p << 1 | 1].l <= r) modify(modify, l, r, k, p << 1 | 1);
//右边有交集
87         pushup(p);
88     };
89
90     auto query = [&](auto query, int l, int r, int p) -> node {
91         if (l <= tr[p].l && tr[p].r <= r) { //被包含
92             return tr[p];
93         }
94         pushdown(p);
95         if (r <= tr[p << 1].r) return query(query, l, r, p << 1); //
只在左边
96         if (tr[p << 1 | 1].l <= l) return query(query, l, r, p << 1 |
1); //只在右边
97         return merge(query(query, l, r, p << 1), query(query, l, r, p
<< 1 | 1)); //左右都有
98     };
99
100     build(n);
101
102     for (int i = 1; i <= q; i++) {
103         int op, x, y;
104         cin >> op >> x >> y;
105
106         if (op == 1) {
107             int k;
108             cin >> k;
109             modify(modify, x, y, k, 1);
110         } else if (op == 2) {
111             int k;
112             cin >> k;
113             update(update, x, y, k, 1);
114         } else {
115             auto tmp = query(query, x, y, 1);
116             cout << tmp.sum << "\n";
117         }
118     }
119 }
120
121 signed main() {
122     cin.tie(0) -> sync_with_stdio(0);
123     int t = 1;
124     // cin >> t;
125
126     while (t--) {
127         QAQ();
128     }

```

