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

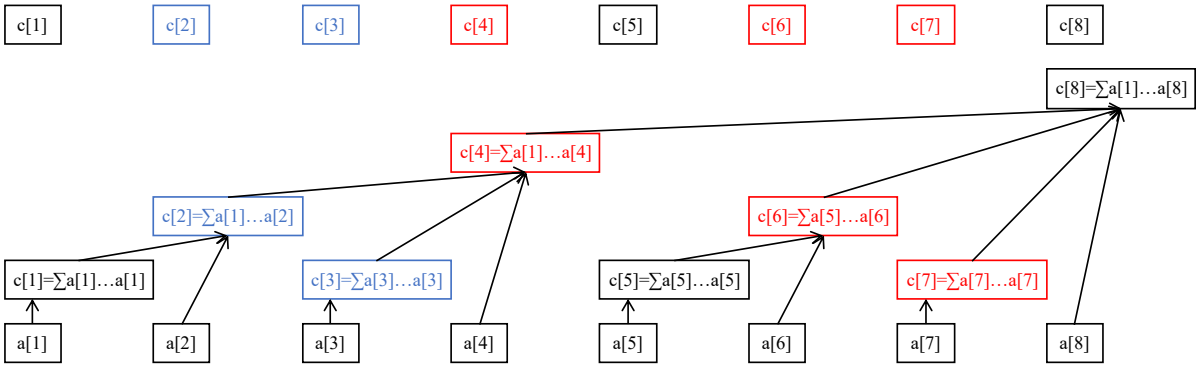
树状数组维护的信息（区间和为例）

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）



查询 ∑a[4]…a[7] = (∑a[1]…a[7]) - (∑a[1]…a[3])

（此树高为 $O(\log_2 n)$ ）

当我们需要求解 $\sum_{i=1}^{n} a_i$，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，

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

$$(36 = 38 - lowbit(38), 32 = 36 - lowbit(38))$$

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

```cpp
vector<int> tr(n + 1);

auto lowbit = [&](int x) {
    return x & -x;
};

auto add = [&](int x, int k) {
    for ( ; x < tr.size(); x += lowbit(x)) {
        tr[x] += k;
    }
};

auto get = [&](int x) {
    int res = 0;
    for ( ; x; x -= lowbit(x)) {
```

```
        res += tr[x];
    }
    return res;
};
```

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

```
auto ta = a;

sort(ta.begin() + 1, ta.end());
ta.erase(unique(ta.begin() + 1, ta.end()), ta.end());

auto get_rank = [&](int x) {
    return lower_bound(ta.begin() + 1, ta.end(), x) - ta.begin();
};
```

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

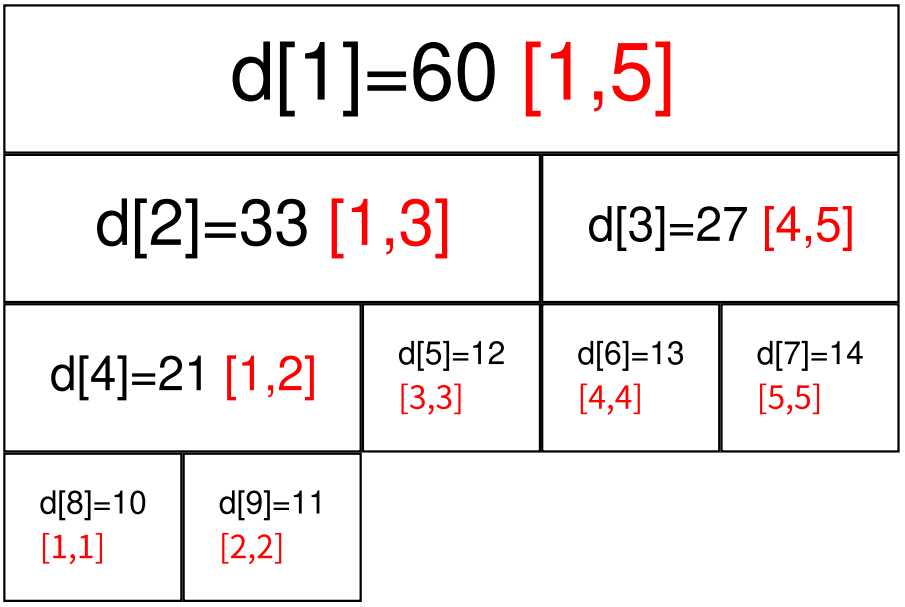
$O(n)$

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 的区间划分为左右俩个区间递归求解，并通过合并左右俩个子区间得到该区间信息



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 $d_{i<<1}$ 和 $d_{i<<1|1}$

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 $2^{\lceil\log_2 n\rceil+1}\leqslant 4n-5$

（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

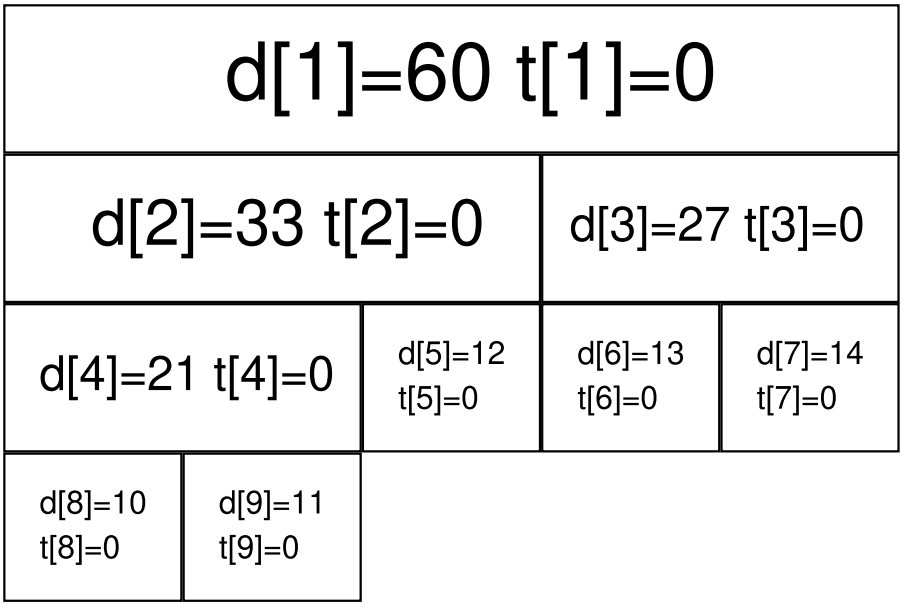
查询区间 $[l,r]$ 的信息可以通过将其拆分成最多 $O(\log_2 n)$ 个**极大**的区间，然后合并这些区间得到答案

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

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



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）

```
struct node {
    int l,r;
    int lazy,sum;
}tr[N*4];

void pushup(int u){
    tr[u].sum=tr[u<<1].sum+tr[u<<1|1].sum;
}

void pushdown(int u){
```

```
    if(tr[u].lazy){
        tr[u<<1].sum+=(tr[u<<1].r-tr[u<<1].l+1)*tr[u].lazy;
        tr[u<<1|1].sum+=(tr[u<<1|1].r-tr[u<<1|1].l+1)*tr[u].lazy;
        tr[u<<1].lazy+=tr[u].lazy;
        tr[u<<1|1].lazy+=tr[u].lazy;
        tr[u].lazy=0;
    }
}

void build(int u,int l,int r){
    tr[u].l=l;
    tr[u].r=r;
    if(l==r){
        tr[u].sum=a[l];
        return;
    }
    int mid=l+r>>1;
    build(u<<1,l,mid);
    build(u<<1|1,mid+1,r);
    pushup(u);
}

void addtr(int u,int l,int r,int k){
    if(l<=tr[u].l&&r>=tr[u].r){
        tr[u].sum+=(tr[u].r-tr[u].l+1)*k;
        tr[u].lazy+=k;
        return;
    }
    pushdown(u);
    int mid=tr[u].r+tr[u].l>>1;
    if(l<=mid) addtr(u<<1,l,r,k);
    if(r>mid) addtr(u<<1|1,l,r,k);
    pushup(u);
}

int findtr(int u,int l,int r){
    if(l<=tr[u].l&&r>=tr[u].r){
        return tr[u].sum;
    }
    pushdown(u);
    int mid=tr[u].l+tr[u].r>>1;
    int ans=0;
    if(l<=mid) ans+=findtr(u<<1,l,r);
    if(r>mid) ans+=findtr(u<<1|1,l,r);
    pushup(u);
    return ans;
}
```

小封装版

```
struct node {
    int l, r;
    int sum, add;
};

```

```
vector<node> tr;

auto merge = [&](node l, node r, node u = {}) { //合并贡献
    u = {l.l, r.r, l.sum + r.sum};
    return u;
};

auto pushup = [&](int p) {
    tr[p] = merge(tr[p << 1], tr[p << 1 | 1], tr[p]);   //向上传
};

auto add_down = [&](int k, node & u) {  //加法的更新
    u.add += k;
    u.sum += (u.r - u.l + 1) * k;
};

auto pushdown = [&](int p) {    //标记下传
    if (tr[p].add) {
        add_down(tr[p].add, tr[p << 1]), add_down(tr[p].add, tr[p << 1 | 1]);
        tr[p].add = 0;
    }
};

auto build = [&](int sz) {  //建树
    tr = vector<node>(sz << 2);

    auto build = [&](auto build, int l, int r, int p) {
        tr[p] = {l, r};
        if (l == r) {
            tr[p].sum = a[r];   //叶子节点等于原数组
            return;
        }
        pushdown(p);    //可不用
        int mid = l + r >> 1;
        build(build, l, mid, p << 1), build(build, mid + 1, r, p << 1 | 1);
        pushup(p);
    };

    build(build, 1, sz, 1);
};

auto update = [&](auto update, int l, int r, int k, int p) {    //更新
    if (l <= tr[p].l && tr[p].r <= r) {
        add_down(k, tr[p]); //添加标记,更新节点
        return;
    }
    pushdown(p);
    if (l <= tr[p << 1].r) update(update, l, r, k, p << 1); //左边有交集
    if (tr[p << 1 | 1].l <= r) update(update, l, r, k, p << 1 | 1); //右边有交集
    pushup(p);
};

auto query = [&](auto query, int l, int r, int p) -> node {
    if (l <= tr[p].l && tr[p].r <= r) { //被包含
```

```
        return tr[p];
    }
    pushdown(p);
    if (r <= tr[p << 1].r) return query(query, l, r, p << 1);   //只在左边
    if (tr[p << 1 | 1].l <= l) return query(query, l, r, p << 1 | 1);   //只在右边
    return merge(query(query, l, r, p << 1), query(query, l, r, p << 1 | 1));   //左右都有
};
```

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， $s = ababaabc$ ，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

$O(n)$

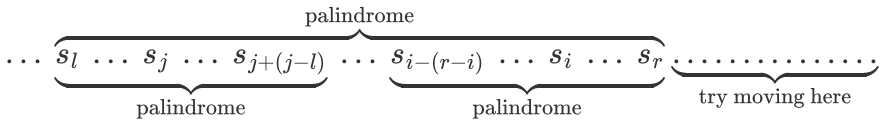
算法原理

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 $i \leqslant r$ ，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 $d_i = d_j$

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

  - 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， $r \leqslant 2 \times n$ ，故时间复杂度为 $O(n)$

代码实现

```
auto read_string = [&]() {
    string t, s = " #";
    cin >> t;
    for (auto x : t) {
        s += x, s += '#';
    }
    return s;
};

auto check = [&](char a, char b) {
    if (a == b) {
        return 1;
    }
    return 0;
};

string s = read_string();
vector<int> d;  //d为包括自己的回文半径

auto Manacher = [&] (string &s) -> void {
    d = vector<int>(s.size());
    int mid = 0, r = 0; //mid记录区间中点
    for (int i = 1; i < s.size(); i ++){
        d[i] = i <= r ? min(d[mid * 2 - i], r - i) : (int)1;
        while (1 <= i - d[i] && i + d[i] < s.size() && check(s[i - d[i]],
s[i + d[i]])) {
            d[i] ++;
        }
        if (d[i] + i > r) {
            r = d[i] + i;
            mid = i;
        }
    }
};
```

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)

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

vector<int> tr;
```

```cpp

int lowbit(int x) {
    return x & -x;
}

void add(int x, int k) {
    for (int i = x; i < tr.size(); i += lowbit(i)) {
        tr[i] += k;
    }
}

int get_sum(int r) {
    int res = 0;
    for (int i = r; i; i -= lowbit(i)) {
        res += tr[i];
    }
    return res;
}

void QAQ() {
    int n, m;
    cin >> n >> m;

    tr = vector<int>(n + 1);

    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        add(i, x);
    }

    for (int i = 1; i <= m; i++) {
        int op, x, y;
        cin >> op >> x >> y;

        if (op == 2) {
            cout << get_sum(y) - get_sum(x - 1) << "\n";
        } else {
            add(x, y);
        }

    }
}

signed main() {
    cin.tie(0) -> sync_with_stdio(0);
    int t = 1;
    // cin >> t;

    while (t--) {
        QAQ();
    }
}
```

2. P3372 【模板】线段树 1 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

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

void QAQ() {
    int n, m;
    cin >> n >> m;

    vector<int> a(n + 1);

    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    struct node {
        int l, r;
        int sum, add;
    };

    vector<node> tr;

    auto merge = [&](node l, node r, node u = {}) { //合并贡献
        u = {l.l, r.r, l.sum + r.sum};
        return u;
    };

    auto pushup = [&](int p) {
        tr[p] = merge(tr[p << 1], tr[p << 1 | 1], tr[p]);   //向上传
    };

    auto add_down = [&](int k, node & u) {  //加法的更新
        u.add += k;
        u.sum += (u.r - u.l + 1) * k;
    };

    auto pushdown = [&](int p) {    //标记下传
        if (tr[p].add) {
            add_down(tr[p].add, tr[p << 1]), add_down(tr[p].add, tr[p
<< 1 | 1]);
            tr[p].add = 0;
        }
    };

    auto build = [&](int sz) {  //建树
        tr = vector<node>(sz << 2);

        auto build = [&](auto build, int l, int r, int p) {
            tr[p] = {l, r};
            if (l == r) {
                tr[p].sum = a[r];   //叶子节点等于原数组
                return;
            }
            pushdown(p);    //可不用
            int mid = l + r >> 1;
            build(build, l, mid, p << 1), build(build, mid + 1, r, p <<
1 | 1);
```

```
            pushup(p);
        };

        build(build, 1, sz, 1);
    };

    auto update = [&](auto update, int l, int r, int k, int p) {    //更新
        if (l <= tr[p].l && tr[p].r <= r) {
            add_down(k, tr[p]); //添加标记,更新节点
            return;
        }
        pushdown(p);
        if (l <= tr[p << 1].r) update(update, l, r, k, p << 1); //左边有交集
        if (tr[p << 1 | 1].l <= r) update(update, l, r, k, p << 1 | 1); //右边有交集
        pushup(p);
    };

    auto query = [&](auto query, int l, int r, int p) -> node {
        if (l <= tr[p].l && tr[p].r <= r) { //被包含
            return tr[p];
        }
        pushdown(p);
        if (r <= tr[p << 1].r) return query(query, l, r, p << 1);   //只在左边
        if (tr[p << 1 | 1].l <= l) return query(query, l, r, p << 1 | 1);   //只在右边
        return merge(query(query, l, r, p << 1), query(query, l, r, p << 1 | 1));   //左右都有
    };

    build(n);

    for (int i = 1; i <= m; i++) {
        int op, x, y;
        cin >> op >> x >> y;

        if (op == 1) {
            int k;
            cin >> k;

            update(update, x, y, k, 1);
        } else {
            auto tmp = query(query, x, y, 1);

            cout << tmp.sum << "\n";
        }
    }
}

signed main() {
    cin.tie(0) -> sync_with_stdio(0);
    int t = 1;
    // 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";
```

```
}

signed main() {
    cin.tie(0) -> sync_with_stdio(0);
    int t = 1;
    // cin >> t;

    while (t--) {
        QAQ();
    }
}
```

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

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

void QAQ() {
    int n, m;
    cin >> n >> m;

    vector<int> tr(n + 1);

    auto lowbit = [&](int x) {
        return x & -x;
    };

    auto add = [&](int x, int k) {
        for ( ; x < tr.size(); x += lowbit(x)) {
            tr[x] += k;
        }
    };

    auto get = [&](int x) {
        int res = 0;
        for ( ; x; x -= lowbit(x)) {
            res += tr[x];
        }
        return res;
    };

    vector<int> a(n + 1);

    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        add(i, a[i] - a[i - 1]);
    }

    for (int i = 1; i <= m; i++) {
        int op, x;
        cin >> op >> x;

        if (op == 1) {
            int y, k;
            cin >> y >> k;
```

```
            add(x, k), add(y + 1, -k);
        } else {
            cout << get(x) << "\n";
        }
    }
}

signed main() {
    cin.tie(0) -> sync_with_stdio(0);
    int t = 1;
    // cin >> t;

    while (t--) {
        QAQ();
    }
}
```

5. P1908 逆序对 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

归并排序：

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

void QAQ() {
    int n;
    cin >> n;

    vector<int> a(n + 1);

    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    int ans = 0;

    auto dfs = [&](auto dfs, int s, int t) -> void {
        if(s >= t) return;
        int mid = s + t >> 1;
        dfs(dfs, s, mid), dfs(dfs, mid + 1, t);
        vector<int> f(a.begin() + s, a.begin() + mid + 1), g(a.begin() +
mid + 1, a.begin() + t + 1);
        for (int i = 0, j = 0, p = s; i < f.size() || j < g.size(); ) {
            if(i>=f.size()){
                a[p++] = g[j++];
            } else if (j >= g.size()) {
                a[p++] = f[i++];
            } else if (f[i] <= g[j]) {
                a[p++] = f[i++];
            } else {
                a[p++] = g[j++], ans += f.size() - i;
            }
        }
    };

```

```
    dfs(dfs, 1, n);

    cout << ans << "\n";
}

signed main() {
    cin.tie(0) -> sync_with_stdio(0);
    int t = 1;
    // cin >> t;

    while (t--) {
        QAQ();
    }
}
```

树状数组

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

void QAQ() {
    int n;
    cin >> n;

    vector<int> a(n + 1);

    int ans = 0;

    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    auto _ = a;

    sort(_.begin() + 1, _.end());
    _.erase(unique(_.begin() + 1, _.end()), _.end());

    auto get_rank = [&](int x) {
        return lower_bound(_.begin() + 1, _.end(), x) - _.begin();
    };

    vector<int> tr(_.size());

    auto lowbit = [&](int x) {
        return x & -x;
    };

    auto add = [&](int x, int k) {
        for ( ; x < tr.size(); x += lowbit(x)) {
            tr[x] += k;
        }
    };

    auto get = [&](int x) {
```

```
        int res = 0;
        for ( ; x; x -= lowbit(x)) {
            res += tr[x];
        }
        return res;
    };

    for (int i = 1; i <= n; i++) {
        add(get_rank(a[i]), 1);
        ans += get(tr.size() - 1) - get(get_rank(a[i]));
    }

    cout << ans << "\n";
}

signed main() {
    cin.tie(0) -> sync_with_stdio(0);
    int t = 1;
    // cin >> t;

    while (t--) {
        QAQ();
    }
}
```

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

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

void QAQ() {
    int n, q, mod;
    cin >> n >> q >> mod;

    vector<int> a(n + 1);

    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    struct node {
        int l, r;
        int sum = 0, add = 0, mul = 1;  // 最好赋初始值
    };

    vector<node> tr;

    auto merge = [&](node l, node r, node u = {}) { //合并贡献
        u = {l.l, r.r, (l.sum + r.sum) % mod, 0, 1};
        return u;
    };

    auto pushup = [&](int p) {
        tr[p] = merge(tr[p << 1], tr[p << 1 | 1], tr[p]);   //向上传
    };
```

```
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
    | 1]);
44          tr[p].mul = 1;
45
46          add_down(tr[p].add, tr[p << 1]), add_down(tr[p].add, tr[p << 1
    | 1]);
47          tr[p].add = 0;
48      };
49
50      auto build = [&](int sz) {  //建树
51          tr = vector<node>(sz << 2);
52
53          auto build = [&](auto build, int l, int r, int p) {
54              tr[p] = {l, r, 0, 0, 1};
55              if (l == r) {
56                  tr[p].sum = a[r] % mod; //叶子节点等于原数组
57                  return;
58              }
59              pushdown(p);    //可不用
60              int mid = l + r >> 1;
61              build(build, l, mid, p << 1), build(build, mid + 1, r, p <<
    1 | 1);
62              pushup(p);
63          };
64
65          build(build, 1, sz, 1);
66      };
67
68      auto update = [&](auto update, int l, int r, int k, int p) {    //
    更新
69          if (l <= tr[p].l && tr[p].r <= r) {
70              add_down(k, tr[p]); //添加标记,更新节点
71              return;
72          }
73          pushdown(p);
74          if (l <= tr[p << 1].r) update(update, l, r, k, p << 1); //左边有
    交集
75          if (tr[p << 1 | 1].l <= r) update(update, l, r, k, p << 1 | 1);
    //右边有交集
76          pushup(p);
77      };
78
```

```cpp
    auto modify = [&](auto modify, int l, int r, int k, int p) {    //更新
        if (l <= tr[p].l && tr[p].r <= r) {
            mul_down(k, tr[p]); //添加标记,更新节点
            return;
        }
        pushdown(p);
        if (l <= tr[p << 1].r) modify(modify, l, r, k, p << 1); //左边有交集
        if (tr[p << 1 | 1].l <= r) modify(modify, l, r, k, p << 1 | 1); //右边有交集
        pushup(p);
    };

    auto query = [&](auto query, int l, int r, int p) -> node {
        if (l <= tr[p].l && tr[p].r <= r) { //被包含
            return tr[p];
        }
        pushdown(p);
        if (r <= tr[p << 1].r) return query(query, l, r, p << 1);   //只在左边
        if (tr[p << 1 | 1].l <= l) return query(query, l, r, p << 1 | 1);   //只在右边
        return merge(query(query, l, r, p << 1), query(query, l, r, p << 1 | 1));   //左右都有
    };

    build(n);

    for (int i = 1; i <= q; i++) {
        int op, x, y;
        cin >> op >> x >> y;

        if (op == 1) {
            int k;
            cin >> k;
            modify(modify, x, y, k, 1);
        } else if (op == 2) {
            int k;
            cin >> k;
            update(update, x, y, k, 1);
        } else {
            auto tmp = query(query, x, y, 1);
            cout << tmp.sum << "\n";
        }
    }
}

signed main() {
    cin.tie(0) -> sync_with_stdio(0);
    int t = 1;
    // cin >> t;

    while (t--) {
        QAQ();
    }
```

```
129 }
```