

# 二分

二分中，需要看答案是否满足单调性，满足单调性即可二分，然后根据二分答案判断是否符合题目条件。

```
1  int bisearch(int a[],int n,int x)
2  {
3      int left = 0,right = n-1;
4      while(left<=right)
5      {
6          int mid = (left+right)>>1;
7          if(a[mid] == x)
8              return mid;
9          if(x > a[mid])
10             left = mid+1;
11         else
12             right = mid- 1;
13     }
14     return -1;
15 }
16
```

```
1  while (l < r)//往左，最大找最小
2  {
3      int mid = l + r >> 1;
4      if (check(mid))
5          r = mid;
6      else
7          l = mid + 1;
8  }
9  while (l < r)//往右，最小找最大
10 {
11     int mid = l + r + 1 >> 1;
12     if (check(mid))
13         l = mid;
14     else
15         r = mid - 1;
16 }
```

记录法

后继(最大找最小)

```

1 while (l<=r)
2 {
3     int mid=(l+r)/2;
4     if (check(mid))
5     {
6         ans=mid;
7         r=mid-1;
8     }
9     else l=mid+1;
10 }printf("%d",ans);

```

前驱（最小找最大）

```

1 while (l<=r)
2 {
3     int mid=(l+r)/2;
4     if (check(mid))
5     {
6         l = mid + 1;
7         ans =mid;
8     }
9     else
10         r= mid - 1;
11 }printf("%d",ans);

```

二分答案常见步骤

- 1、证明问题单调性。
- 2、确定上下界
- 3、设计check()函数。
- 4、上下界之内二分答案。

## 异或线性基

把对 $n$ 个数的组合求异或，缩小到对 $m$ 个数组合求异或。设原定数字的集合为

$A = \{a_1, a_2, a_3, \dots, a_n\}$ ，求得线性基结果为 $P = \{p_1, p_2, p_3, \dots, p_k\}$ 。在A和P上面分别对任意组合求异或结果一样。

## 线性基构造

### 1、基本原理

规则：P中的每个元素的二进制位数均不同。P中元素的最少位数为，最大位数为m，P中的元素个数不会超过m。

设 $A = \{a_1, a_2\}$ ，且两个元素位数相同（即首位都为1）。那么他的一个线性基为 $P = \{a_1, a_1 \oplus a_2\}$ 。

证明： $a_1 \oplus a_2$ 比 $a_1$ 、 $a_2$ 的长度短，因为首位异或之后为0。那么 $\{a_1, a_2\}$ 与 $\{a_1, a_1 \oplus a_2\}$ 异或结果相同。

当A中的位数超过两个时候，连续处理即可。比如  $A = \{1000, 1101, 1111\}$ 。先把1000放入P中，然后把1101和P中的1000异或，结果为101，放入P中，此时P中有1000，101，然后把1111与1000异或的结果与101异或 放入P中，结果为10。P中为1000 101 10。

## 2.高斯消元

把A中的每个数写成n\*m的0/1矩阵，然后简化成阶梯矩阵。

$$\begin{array}{ccc} 1000 & 1000 & 1000 \\ 1101 & \Rightarrow 0101 & \Rightarrow 0101 \\ 1111 & 0111 & 0010 \end{array}$$

# 线性基应用

## 最小异或和

有全0行，则最小为0，否则为P中最小的元素。

## 最大异或和

异或所有的即可。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  long long p[63];
5  bool zero;
6  void Insert(long long x)
7  {
8      for(int i = 63; i >= 0; i--)
9      {
10         if(x>>i == 1)
11         {
12             if(p[i] == 0)
13             {
14                 p[i] = x;
15                 return ;
16             }
17             else
18             {
19                 x ^=p[i];
20             }
21         }
22     }
23     zero = true;
24 }
25 long long qmax()
26 {
27     long long ans = 0;
28     for(int i = 63; i >= 0; i--)
29     {
30         ans = max(ans,ans^p[i]);
31     }
32     return ans;
33 }
34 int main ()
```

```

35 {
36     long long x;
37     int n;
38     cin>>n;
39     for(int i =1;i <= n;i++)
40     {
41         cin>>x;
42         Insert(x);
43     }
44     cout<<qmax()<<endl;
45     return 0;
46 }

```

## 第k大异或和/第k小异或和

$k = 1$ , 取1111,  $k = 2$ ,取1110,  $k = 3$ ,取1101。

所以是选 $2^t - k$ 的二进制对应的那些行。t为P中元素个数。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int n;
4  bool zero;
5  long long a[10100];
6  void Gauss()//高斯消元法求线性基
7  {
8      int i,k = 1;
9      long long j = 1ll<<62;
10     for(;j;j >>=1)
11     {
12         for(i = k;i <= n;i++)
13             if(a[i] & j)
14                 break;
15         if(i > n)
16             continue;
17         swap(a[i],a[k]);
18         for(i = 1;i <= n;i++)
19             if(i != k&&a[i]&j)
20                 a[i]^=a[k];
21         k++;
22     }
23     k--;
24     if(k != n)
25         zero = true;
26     else
27         zero = false;
28     n = k;
29 }
30 long long Query(long long k)
31 {
32     long long ans = 0;
33     if(zero)
34         k--;
35     if(!k)
36         return 0;

```

```

37     for(int i = n;i-->0)
38     {
39         if(k&1)
40             ans ^= a[i];
41         k>>=1;
42     }
43     if(k)
44         return -1;
45     return ans;
46 }
47 int main ()
48 {
49     int cnt = 0;
50     int t;
51     cin>>t;
52     while(t-->0)
53     {
54         printf("Case # %d:\n",++cnt);
55         cin>>n;
56         for(int i = 1;i <= n;i++)
57         {
58             cin>>a[i];
59         }
60         Gauss();
61         int q;
62         cin>>q;
63         while(q-->0)
64         {
65             long long k;
66             cin>>k;
67             cout<<Query(k)<<endl;
68         }
69     }
70 }

```

## Data Structure 数据结构

### 线段树

#### 普通线段树

```

1  /**
2   * @see https://www.luogu.com.cn/problem/P2068
3   */
4  #include <bits/stdc++.h>
5
6  using ll = long long;
7  const int N = 1e5 + 3;
8  ll a[N];
9
10 struct node {
11     int l, r;
12     ll val, sum;
13 } tr[N << 2];

```

```

14
15 inline int left_child(int p) {
16     return p << 1;
17 }
18
19 inline int right_child(int p) {
20     return p << 1 | 1;
21 }
22
23 inline void push_up(int p) {
24     tr[p].val = tr[left_child(p)].val + tr[right_child(p)].val;
25 }
26
27 inline void push_down(int p) {
28     if (tr[p].sum) {
29         tr[left_child(p)].val += tr[p].sum * (tr[left_child(p)].r -
tr[left_child(p)].l + 1);
30         tr[right_child(p)].val += tr[p].sum * (tr[right_child(p)].r -
tr[right_child(p)].l + 1);
31         tr[left_child(p)].sum += tr[p].sum;
32         tr[right_child(p)].sum += tr[p].sum;
33         tr[p].sum = 0;
34     }
35 }
36
37 inline void build(int p, int s, int t) {
38     tr[p] = {s, t, 0, 0};
39     if(s == t) {
40         tr[p].val = a[s];
41         return;
42     }
43
44     int mid = (s + t) >> 1;
45     build(left_child(p), s, mid);
46     build(right_child(p), mid + 1, t);
47     push_up(p);
48 }
49
50 inline void update(int p, int l, int r, ll k) {
51     if(l <= tr[p].l && tr[p].r <= r) {
52         tr[p].val += k * (tr[p].r - tr[p].l + 1);
53         tr[p].sum += k;
54         return;
55     }
56
57     push_down(p);
58     int mid = (tr[p].l + tr[p].r) >> 1;
59     if(l <= mid) {
60         update(left_child(p), l, r, k);
61     }
62     if(mid < r) {
63         update(right_child(p), l, r, k);
64     }
65     push_up(p);
66 }
67

```

```

68 inline ll query(int p, int l, int r) {
69     if(l <= tr[p].l && tr[p].r <= r) {
70         return tr[p].val;
71     }
72
73     push_down(p);
74     int mid = (tr[p].l + tr[p].r) >> 1;
75     ll ans = 0;
76     if(l <= mid) {
77         ans += query(left_child(p), l, r);
78     }
79     if(mid < r) {
80         ans += query(right_child(p), l, r);
81     }
82     return ans;
83 }
84
85 int main() {
86     int n, w;
87     std::cin >> n >> w;
88     build(1, 1, N);
89     while (w--) {
90         char op;
91         int a, b;
92         std::cin >> op >> a >> b;
93         if (op == 'x') {
94             update(1, a, a, b);
95         } else {
96             std::cout << query(1, a, b) << std::endl;
97         }
98     }
99 }

```

## 加乘线段树

```

1  /**
2   * @see https://www.luogu.com.cn/problem/P3373
3   */
4  #include <bits/stdc++.h>
5
6  using ll = long long;
7  const int N = 1e5 + 7;
8  ll a[N];
9  int n, q, m;
10
11 struct node {
12     ll val, mul, add;
13 } t[N << 2];
14
15 inline int left_child(int p) {
16     return p << 1;
17 }
18
19 inline int right_child(int p) {
20     return p << 1 | 1;

```

```

21 }
22
23 void push_up(int p) {
24     t[p].val = (t[left_child(p)].val + t[right_child(p)].val) % m;
25 }
26
27 void push_down(int p, int l, int r) {
28     int mid = (l + r) >> 1;
29     t[left_child(p)].val = (t[left_child(p)].val * t[p].mul + t[p].add *
30 (mid - l + 1)) % m;
31     t[right_child(p)].val = (t[right_child(p)].val * t[p].mul + t[p].add *
32 (r - mid)) % m;
33     t[left_child(p)].add = (t[left_child(p)].add * t[p].mul + t[p].add) % m;
34     t[right_child(p)].add = (t[right_child(p)].add * t[p].mul + t[p].add) %
35 m;
36     t[left_child(p)].mul = (t[left_child(p)].mul * t[p].mul) % m;
37     t[right_child(p)].mul = (t[right_child(p)].mul * t[p].mul) % m;
38     t[p].mul = 1;
39     t[p].add = 0;
40 }
41
42 void build(int p, int l, int r) {
43     t[p] = {0, 1, 0};
44     if(l == r) {
45         t[p].val = a[l] % m;
46         return;
47     }
48     int mid = (l + r) >> 1;
49     build(left_child(p), l, mid);
50     build(right_child(p), mid + 1, r);
51     push_up(p);
52 }
53
54 void update1(int p, int std_l, int std_r, int l, int r, ll k) {
55     if(r < std_l || std_r < l) {
56         return;
57     }
58     if(l <= std_l && std_r <= r) {
59         t[p].val = (t[p].val * k) % m;
60         t[p].add = (t[p].add * k) % m;
61         t[p].mul = (t[p].mul * k) % m;
62         return;
63     }
64     push_down(p, std_l, std_r);
65     int mid = (std_l + std_r) >> 1;
66     update1(left_child(p), std_l, mid, l, r, k);
67     update1(right_child(p), mid + 1, std_r, l, r, k);
68     push_up(p);
69 }
70
71 /**
72  * [l, r] 区间乘以 k
73  */
74 void update1(int l, int r, ll k) {

```



```

74     update1(1, 1, n, 1, r, k);
75 }
76
77 void update2(int p, int std_l, int std_r, int l, int r, ll k) {
78     if(r < std_l || std_r < l) {
79         return;
80     }
81     if(l <= std_l && std_r <= r) {
82         t[p].val = (t[p].val + k * (std_r - std_l + 1)) % m;
83         t[p].add = (t[p].add + k) % m;
84         return;
85     }
86
87     push_down(p, std_l, std_r);
88     int mid = (std_l + std_r) >> 1;
89     update2(left_child(p), std_l, mid, l, r, k);
90     update2(right_child(p), mid + 1, std_r, l, r, k);
91     push_up(p);
92 }
93
94 /**
95  * [l, r] 区间加上 k
96  */
97 void update2(int l, int r, ll k) {
98     update2(1, 1, n, l, r, k);
99 }
100
101 ll rangeSum(int p, int std_l, int std_r, int l, int r) {
102     if(r < std_l || std_r < l) {
103         return 0;
104     }
105     if(l <= std_l && std_r <= r) {
106         return t[p].val;
107     }
108
109     push_down(p, std_l, std_r);
110     int mid = (std_l + std_r) >> 1;
111     return (rangeSum(left_child(p), std_l, mid, l, r) +
112         rangeSum(right_child(p), mid + 1, std_r, l, r)) % m;
113 }
114
115 ll rangeSum(int l, int r) {
116     return rangeSum(1, 1, n, l, r);
117 }
118
119 int main() {
120     std::cin >> n >> q >> m;
121     for (int i = 1; i <= n; ++i) {
122         std::cin >> a[i];
123     }
124     build(1, 1, n);
125     while(q--) {
126         int op, x, y;
127         ll k;
128         std::cin >> op;
129         if(op == 1) {

```

```

129         std::cin >> x >> y >> k;
130         update1(x, y, k);
131     } else if(op == 2) {
132         std::cin >> x >> y >> k;
133         update2(x, y, k);
134     } else {
135         std::cin >> x >> y;
136         std::cout << rangeSum(x, y) << std::endl;
137     }
138 }
139 }

```

## DSU 并查集

### 普通并查集

```

1  const int N = 2e4 + 3;
2  int fa[N];
3
4  void init(int n) {
5      for(int i = 1; i <= n; ++i) {
6          fa[i] = i;
7      }
8  }
9
10 int query(int x) {
11     while(x != fa[x]) {
12         x = fa[x] = fa[fa[x]];
13     }
14     return x;
15 }
16
17 void merge(int x, int y) {
18     int r1 = query(x);
19     int r2 = query(y);
20     if(r1 != r2) {
21         fa[r2] = r1;
22     }
23 }
24
25 bool same(int x, int y) {
26     return query(x) == query(y);
27 }

```

### 带权并查集

#### 1. 封装版

```

1  /**
2   * @see
3   * https://codeforces.com/edu/course/2/lesson/7/1/practice/contest/289390/pr
4   * 封装为结构体
5   */

```

```

5  #include <bits/stdc++.h>
6
7  struct Info {
8      int min, max, cnt;
9
10     Info() {}
11
12     Info(int id)
13         : min(id + 1), max(id + 1), cnt(1) {}
14
15     void apply(const Info& fy) {
16         min = std::min(min, fy.min);
17         max = std::max(max, fy.max);
18         cnt += fy.cnt;
19     }
20 };
21
22 template <typename Info>
23 struct DSU {
24     std::vector<int> fa;
25     std::vector<Info> info;
26
27     DSU() {}
28
29     DSU(int n) {
30         init(n);
31     }
32
33     void init(int n) {
34         fa.resize(n);
35         info.resize(n);
36         for(int i = 0; i < n; ++i) {
37             fa[i] = i;
38             info[i] = Info{i};
39         }
40     }
41
42     int query(int x) {
43         while (x != fa[x]) {
44             x = fa[x] = fa[fa[x]];
45         }
46         return x;
47     }
48
49     bool merge(int x, int y) {
50         int fx = query(x);
51         int fy = query(y);
52         if (fx == fy) {
53             return false;
54         }
55         fa[fy] = fx;
56         info[fx].apply(info[fy]);
57         return true;
58     }
59
60     bool same(int x, int y) {

```

```

61         return query(x) == query(y);
62     }
63
64     Info get(int x) {
65         return info[x];
66     }
67 };
68
69 int main() {
70     int n, m;
71     std::cin >> n >> m;
72     DSU<Info> dsu(n + 1);
73     while (m--) {
74         std::string op;
75         int u, v;
76         std::cin >> op;
77         if (op == "get") {
78             std::cin >> v;
79             Info fv = dsu.get(dsu.query(v - 1));
80             std::cout << fv.min << ' ' << fv.max << ' ' << fv.cnt <<
std::endl;
81         } else {
82             std::cin >> u >> v;
83             dsu.merge(u - 1, v - 1);
84         }
85     }
86 }

```

## 2. 简化版（下标从0开始）

```

1  /**
2   * @see
3   * https://codeforces.com/edu/course/2/lesson/7/1/practice/contest/289390/pr
4   * 下标从 0 开始
5   */
6
7  #include <bits/stdc++.h>
8
9  const int N = 3e5 + 7;
10
11  struct Info {
12      int max, min, cnt;
13  } info[N];
14
15  int fa[N];
16
17  void init(int n) {
18      for (int i = 0; i < n; ++i) {
19          fa[i] = i;
20          info[i] = {i + 1, i + 1, 1};
21      }
22  }
23
24  int query(int x) {
25      while (x != fa[x]) {
26          x = fa[x] = fa[fa[x]];
27      }
28      return x;
29  }

```

```

25     }
26     return x;
27 }
28
29 bool merge(int x, int y) {
30     int fx = query(x);
31     int fy = query(y);
32     if (fx == fy) {
33         return false;
34     }
35     fa[fy] = fx;
36     info[fx].max = std::max(info[fx].max, info[fy].max);
37     info[fx].min = std::min(info[fx].min, info[fy].min);
38     info[fx].cnt += info[fy].cnt;
39     return true;
40 }
41
42 bool same(int x, int y) {
43     return query(x) == query(y);
44 }
45
46 int main() {
47     int n, m;
48     std::cin >> n >> m;
49     init(n);
50     while (m--) {
51         std::string op;
52         int u, v;
53         std::cin >> op;
54         if (op == "get") {
55             std::cin >> v;
56             int fv = query(v - 1);
57             std::cout << info[fv].min << ' ' << info[fv].max << ' ' <<
info[fv].cnt << std::endl;
58         } else {
59             std::cin >> u >> v;
60             merge(u - 1, v - 1);
61         }
62     }
63 }

```

### 3. 简化版（下标从1开始）

```

1  /**
2   * @see
3   * https://codeforces.com/edu/course/2/lesson/7/1/practice/contest/289390/pr
4   * 下标从 1 开始
5   */
6
7  #include <bits/stdc++.h>
8
9  const int N = 3e5 + 7;
10
11 struct Info {
12     int max, min, cnt;
13 } info[N];

```

```

12
13 int fa[N];
14
15 void init(int n) {
16     for (int i = 1; i <= n; ++i) {
17         fa[i] = i;
18         info[i] = {i, i, 1};
19     }
20 }
21
22 int query(int x) {
23     while (x != fa[x]) {
24         x = fa[x] = fa[fa[x]];
25     }
26     return x;
27 }
28
29 bool merge(int x, int y) {
30     int fx = query(x);
31     int fy = query(y);
32     if (fx == fy) {
33         return false;
34     }
35     fa[fy] = fx;
36     info[fx].max = std::max(info[fx].max, info[fy].max);
37     info[fx].min = std::min(info[fx].min, info[fy].min);
38     info[fx].cnt += info[fy].cnt;
39     return true;
40 }
41
42 bool same(int x, int y) {
43     return query(x) == query(y);
44 }
45
46 int main() {
47     int n, m;
48     std::cin >> n >> m;
49     init(n);
50     while (m--) {
51         std::string op;
52         int u, v;
53         std::cin >> op;
54         if (op == "get") {
55             std::cin >> v;
56             int fv = query(v);
57             std::cout << info[fv].min << ' ' << info[fv].max << ' ' <<
info[fv].cnt << std::endl;
58         } else {
59             std::cin >> u >> v;
60             merge(u, v);
61         }
62     }
63 }

```

# heap

```
1 std::priority_queue<int, std::vector<int>, std::less<int>> pq; // less表示按照递减(从大到小)的顺序插入元素
2 std::priority_queue<int, std::vector<int>, std::greater<int>> pq; // greater表示按照递增(从小到大)的顺序插入元素
```

## Graph 图论

### 链式前向星

```
1 const int N = 1e5 + 3;
2
3 int head[N], tot;
4
5 struct node {
6     int to, next;
7 } edge[N << 1];
8
9 void addEdge(int u, int v) {
10     edge[++tot] = {v, head[u]};
11     head[u] = tot;
12 }
13
14 // 遍历x的邻接节点
15 for(int i = head[x]; i; i = edge[i].next) {
16
17 }
```

## 二分图最大权匹配

```
1 /**
2  * @see https://www.luogu.com.cn/problem/B3605
3  */
4 #include <bits/stdc++.h>
5
6 const int N = 2.5e5 + 3;
7
8 int head[N], tot;
9
10 struct node {
11     int to, next;
12 } edge[N << 1];
13
14 void addEdge(int u, int v) {
15     edge[++tot] = {v, head[u]};
16     head[u] = tot;
17 }
```

```

18
19 bool vis[N];
20 int match[N];
21
22 bool dfs(int x) {
23     for(int i = head[x]; i; i = edge[i].next) {
24         int to = edge[i].to;
25         if(!vis[to]) {
26             vis[to] = true;
27             if(!match[to] || dfs(match[to])) {
28                 match[to] = x;
29                 return true;
30             }
31         }
32     }
33     return false;
34 }
35
36 int main() {
37     int n1, nr, m;
38     std::cin >> n1 >> nr >> m;
39     while(m--) {
40         int u, v;
41         std::cin >> u >> v;
42         addEdge(u, v);
43     }
44
45     int ans = 0;
46     for(int i = 1; i <= n1; ++i) {
47         ans += dfs(i);
48         memset(vis, 0, sizeof(vis));
49     }
50     std::cout << ans << std::endl;
51 }

```

## 最近公共祖先LCA

### 树链剖分求LCA

```

1  /**
2   * 树链剖分求LCA
3   * @see https://blog.csdn.net/qq\_41418281/article/details/108220247
4   */
5  #include <bits/stdc++.h>
6
7  struct HLD {
8      int n; // 节点个数
9      std::vector<std::vector<int>> edge; // 邻接矩阵
10     std::vector<int> siz; // siz[u]: 存以u为根的子树的结点数
11     std::vector<int> dep; // dep[u]: 存u的深度
12     std::vector<int> top; // top[u]: 存u所在重链的顶点
13     std::vector<int> son; // son[u]: 存u的重儿子
14     std::vector<int> fa; // fa[u]: 存u的父节点
15 }

```



```

16     HLD(int n) {
17         this->n = n;
18         edge.resize(n + 1);
19         siz.resize(n + 1);
20         dep.resize(n + 1);
21         top.resize(n + 1);
22         son.resize(n + 1);
23         fa.resize(n + 1);
24     }
25
26     /**
27     * 添加边
28     */
29     void addEdge(int u, int v) {
30         edge[u].push_back(v);
31     }
32
33     /**
34     * 遍历u的邻接结点
35     */
36     void forEach(int u, const std::function<void(int)>& func) {
37         for (auto& n : edge[u]) {
38             func(n);
39         }
40     }
41
42     /**
43     * 初始化
44     * O(n)
45     */
46     void init(int root = 1) {
47         dfs1(root, 0);
48         dfs2(root, root);
49     }
50
51     /**
52     * 求解lca
53     * O(log n)
54     */
55     int lca(int u, int v) {
56         while (top[u] != top[v]) {
57             if (dep[top[u]] < dep[top[v]]) {
58                 std::swap(u, v);
59             }
60             u = fa[top[u]];
61         }
62         return dep[u] < dep[v] ? u : v;
63     }
64
65     /**
66     * 查询两点间的距离
67     */
68     int distance(int u, int v) {
69         return dep[u] + dep[v] - (dep[lca(u, v)] << 1);
70     }
71

```

```

72     /**
73      * 第一次dfs
74      */
75     void dfs1(int u, int father) {
76         fa[u] = father;
77         dep[u] = dep[fa[u]] + 1;
78         siz[u] = 1;
79         for (const auto& neighbor : edge[u]) {
80             if (neighbor == fa[u]) {
81                 continue;
82             }
83
84             dfs1(neighbor, u);
85             // 更新当前节点的子树大小
86             siz[u] += siz[neighbor];
87             // 寻找重儿子
88             if (siz[neighbor] > siz[son[u]]) {
89                 son[u] = neighbor;
90             }
91         }
92     }
93
94     /**
95      * 第二次dfs
96      */
97     void dfs2(int u, int t) {
98         top[u] = t;
99         if (!son[u]) {
100             return;
101         }
102         dfs2(son[u], t);
103         for(const auto& neighbor : edge[u]) {
104             if (neighbor == fa[u] || neighbor == son[u]) {
105                 continue;
106             }
107             dfs2(neighbor, neighbor);
108         }
109     }
110 };
111
112 int main() {
113     int n, m, s;
114     std::cin >> n >> m >> s;
115
116     HLD h(n);
117     for (int i = 1; i <= n - 1; ++i) {
118         int u, v;
119         std::cin >> u >> v;
120         h.addEdge(u, v);
121         h.addEdge(v, u);
122     }
123
124     h.init(s);
125
126     while (m--) {
127         int a, b;

```

```

128         std::cin >> a >> b;
129         std::cout << h.lca(a, b) << std::endl;
130     }
131 }

```

## 树链剖分HLD + 树上修改 (HLD + 线段树)

```

1  /**
2   * @see https://www.luogu.com.cn/problem/P3384
3   * 树上修改与查询
4   * 闭区间，下标从1开始
5   */
6  #include <bits/stdc++.h>
7
8  // 树链剖分
9  template <typename T>
10 struct HLD {
11     // 线段树节点
12     struct Node {
13         int l, r;
14         T val, sum;
15     };
16
17     // 树链剖分相关数据
18     int n; // 节点个数
19     std::vector<T> w; // 节点权值
20     std::vector<std::vector<int>> edge; // 树边
21     std::vector<int> fa; // fa[u]: 存u的父节点
22     std::vector<int> dep; // dep[u]: 存u的深度
23     std::vector<int> son; // son[u]: 存u的重儿子
24     std::vector<int> siz; // siz[u]: 存以u为根的子树的结点数
25     std::vector<int> top; // top[u]: 存u所在重链的顶点
26
27     // 树上修改相关数据
28     int cnt; // 新编号计数
29     std::vector<int> id; // id[u]: 存u剖分后的新编号
30     std::vector<T> nw; // 存新编号在树中所对应节点的权值
31     std::vector<Node> tree; // 线段树
32
33     HLD(int n, const std::vector<T>& w) {
34         this->n = n;
35         this->w = w;
36         edge.resize(n + 1);
37         fa.resize(n + 1);
38         dep.resize(n + 1);
39         son.resize(n + 1);
40         siz.resize(n + 1);
41         top.resize(n + 1);
42
43         this->cnt = 0;
44         id.resize(n + 1);
45         nw.resize(n + 1);
46         tree.resize(n << 2); // 4n
47     }
48

```

```

49  /**
50   * 添加边
51   */
52  void addEdge(int u, int v) {
53      edge[u].push_back(v);
54  }
55
56  /**
57   * 初始化
58   * O(n)
59   */
60  void init(int root = 1) {
61      dfs1(root, 0);
62      dfs2(root, root);
63      build(1, 1, n);
64  }
65
66  /**
67   * 求解lca
68   * O(log n)
69   */
70  int lca(int u, int v) {
71      while (top[u] != top[v]) {
72          if (dep[top[u]] < dep[top[v]]) {
73              std::swap(u, v);
74          }
75          u = fa[top[u]];
76      }
77      return dep[u] < dep[v] ? u : v;
78  }
79
80  /**
81   * 查询两点间的距离
82   */
83  int distance(int u, int v) {
84      return dep[u] + dep[v] - (dep[lca(u, v)] << 1);
85  }
86
87  /**
88   * 第一次dfs
89   * 搞出 fa dep siz son
90   */
91  void dfs1(int u, int father) {
92      fa[u] = father;
93      dep[u] = dep[fa[u]] + 1;
94      siz[u] = 1;
95      for (const auto& neighbor : edge[u]) {
96          if (neighbor == father) { // 只准往下走
97              continue;
98          }
99
100         dfs1(neighbor, u);
101         siz[u] += siz[neighbor]; // 更新当前节点的子树大小
102         if (siz[son[u]] < siz[neighbor]) { // 寻找重儿子
103             son[u] = neighbor;
104         }

```

```

105     }
106 }
107
108 /**
109  * 第二次dfs
110  * 搞出 top id nw
111  */
112 void dfs2(int u, int t) {
113     top[u] = t;
114     id[u] = ++cnt;
115     nw[cnt] = w[u];
116     if (!son[u]) {
117         return;
118     }
119     dfs2(son[u], t);
120     for (const auto& neighbor : edge[u]) {
121         if (neighbor == fa[u] || neighbor == son[u]) { // 只准往下走 &&
不能选刚才走的重儿子
                continue;
            }
            dfs2(neighbor, neighbor);
        }
    }
}
126
127
128 inline int lc(int p) {
129     return p << 1;
130 }
131
132 inline int rc(int p) {
133     return p << 1 | 1;
134 }
135
136 void pushUp(int p) {
137     tree[p].sum = tree[lc(p)].sum + tree[rc(p)].sum;
138 }
139
140 void pushDown(int p) {
141     if (tree[p].val) {
142         tree[lc(p)].sum += tree[p].val * (tree[lc(p)].r - tree[lc(p)].l
+ 1);
143         tree[rc(p)].sum += tree[p].val * (tree[rc(p)].r - tree[rc(p)].l
+ 1);
144         tree[lc(p)].val += tree[p].val;
145         tree[rc(p)].val += tree[p].val;
146         tree[p].val = 0;
147     }
148 }
149
150 void build(int p, int s, int t) {
151     tree[p] = {s, t, 0, nw[t]};
152     if (s == t) {
153         return;
154     }
155     int mid = s + t >> 1;
156     build(lc(p), s, mid);
157     build(rc(p), mid + 1, t);

```

```

158     pushUp(p);
159 }
160
161 /**
162  * 区间加
163  */
164 void update(int p, int l, int r, const T& k) {
165     if (l <= tree[p].l && tree[p].r <= r) {
166         tree[p].val += k;
167         tree[p].sum += k * (tree[p].r - tree[p].l + 1);
168         return;
169     }
170     pushDown(p);
171     int mid = tree[p].l + tree[p].r >> 1;
172     if (l <= mid) {
173         update(lc(p), l, r, k);
174     }
175     if (r > mid) {
176         update(rc(p), l, r, k);
177     }
178     pushUp(p);
179 }
180
181 /**
182  * 将树从u到v结点最短路径上所有节点的值都加上k
183  */
184 void updatePath(int u, int v, const T& k) {
185     while (top[u] != top[v]) {
186         if (dep[top[u]] < dep[top[v]]) {
187             std::swap(u, v);
188         }
189         update(1, id[top[u]], id[u], k);
190         u = fa[top[u]];
191     }
192     if (dep[u] < dep[v]) {
193         std::swap(u, v);
194     }
195     update(1, id[v], id[u], k); // 最后一段
196 }
197
198 /**
199  * 将以u为根节点的子树内所有节点值都加上k
200  */
201 void updateTree(int u, const T& k) {
202     update(1, id[u], id[u] + siz[u] - 1, k);
203 }
204
205 /**
206  * 查询区间和
207  */
208 T query(int p, int l, int r) {
209     if (l <= tree[p].l && tree[p].r <= r) {
210         return tree[p].sum;
211     }
212     pushDown(p);
213     int mid = tree[p].l + tree[p].r >> 1;

```

```

214     T ans{};
215     if (l <= mid) {
216         ans += query(lc(p), l, r);
217     }
218     if (r > mid) {
219         ans += query(rc(p), l, r);
220     }
221     return ans;
222 }
223
224 /**
225  * 求树从u到v结点最短路径上所有节点的值之和
226  */
227 T queryPath(int u, int v) {
228     T ans{};
229     while (top[u] != top[v]) {
230         if (dep[top[u]] < dep[top[v]]) {
231             std::swap(u, v);
232         }
233         ans += query(1, id[top[u]], id[u]);
234         u = fa[top[u]];
235     }
236     if (dep[u] < dep[v]) {
237         std::swap(u, v);
238     }
239     ans += query(1, id[v], id[u]); // 最后一段
240     return ans;
241 }
242
243 /**
244  * 以u为根节点的子树内所有节点值之和
245  */
246 T queryTree(int u) {
247     return query(1, id[u], id[u] + siz[u] - 1);
248 }
249 };
250
251 using ll = long long;
252
253 int main() {
254     int n, m, r;
255     ll p;
256     std::cin >> n >> m >> r >> p;
257     std::vector<ll> w(n + 1);
258     for (int i = 1; i <= n; ++i) {
259         std::cin >> w[i];
260     }
261     HLD<ll> hld(n, w);
262     for (int i = 1; i < n; ++i) {
263         int u, v;
264         std::cin >> u >> v;
265         hld.addEdge(u, v);
266         hld.addEdge(v, u);
267     }
268
269     hld.init(r);

```

```

270
271     while (m--) {
272         int op, x, y;
273         ll z;
274         std::cin >> op;
275         if (op == 1) {
276             std::cin >> x >> y >> z;
277             hld.updatePath(x, y, z);
278         } else if (op == 2) {
279             std::cin >> x >> y;
280             std::cout << hld.queryPath(x, y) % p << std::endl;
281         } else if (op == 3) {
282             std::cin >> x >> z;
283             hld.updateTree(x, z);
284         } else if (op == 4) {
285             std::cin >> x;
286             std::cout << hld.queryTree(x) % p << std::endl;
287         }
288     }
289 }

```

## Set

有序性，所有操作是 $\log^n$ ，红黑树实现。只能使用迭代器访问

作用：去重，升序排序。

```

1  set<int> s; //声明
2  s.clear(); //清空
3  s.insert(x); //插入元素，如果之前没有，则插入后排序，否则不插入。
4  int hav = s.count(x); //查找是否有x，返回0或1
5  set<int>::iterator it = s.find(x); //查找x并返回迭代器
6  bool isempty = s.empty(); //判断空集
7  int n = s.size(); //元素个数
8  s.erase(x); //删除

```

## 访问

使用迭代器访问，注意set不支持 `it < st.end()` 的写法

```

1  set<int> st;
2  for(set<int>::iterator it = st.begin(); it != st.end(); it++)
3  {
4      cout << *it;
5  }
6
7  // since c++ 11
8  std::set<int> st = {1, 2, 3, 4, 5};
9  for(const auto& s : st) {
10     std::cout << s << ' ';
11 }

```



## 降序排列

```
1 set<int,greater<int> > st;
2 set<int,greater<int> >::iterator it;
3 st.insert(1);
4 for(it = st.begin();it != st.end();it++)
5 {
6     cout<<*it;
7 }
```

如果是结构体类型，需要在结构体中重载小于运算符。

- 仿函数定义

```
1 template<class T>
2 struct Less {
3     bool operator()(const T& x, const T& y) const {
4         return x < y;
5     }
6 };
7
8 template<class T>
9 struct Greater {
10     bool operator()(const T& x, const T& y) const {
11         return x > y;
12     }
13 };
14
15 int main() {
16     std::set<int, Greater<int>> st = {1, 2, 3, 4, 5};
17
18     for (const auto& s : st) {
19         std::cout << s << ' ';
20     }
21     std::cout << std::endl;
22
23     return 0;
24 }
```

## unordered\_set

底层实现 hash table，存储唯一对象集合

### 遍历

```
1 /**
2  * 遍历 since c++ 11
3  * 可以看出 unordered_set 是无序，不重复的
4  */
5 template <class Os, class K>
6 Os& operator<<(Os& os, const std::unordered_set<K>& v) {
7     os << '[' << v.size() << "]" {}";
8     bool o{};
9     for (const auto& e : v) {
```

```

10         os << (o ? ", " : (o = 1, " ")) << e;
11     }
12     return os << " }";
13 }
14
15 std::unordered_set<int> us = {2, 7, 1, 8, 2, 8};
16 std::cout << us << std::endl; // [4] { 8, 1, 7, 2 }

```

## 插入元素

$O(1)$

若重复则不做什么事

如果操作后新的元素数量大于原 `max_load_factor() * bucket_count()` 则会发生重散列。如果（因插入而）发生了重散列，索引迭代器均会失效。否则（未发生重散列），则迭代器不会失效。

```

1 us.insert(9);
2 std::cout << us << std::endl; // [5] { 9, 8, 1, 7, 2 }

```

## 移除元素

$O(1)$

```

1 us.erase(us.begin()); // 移除开头
2 us.erase(std::next(us.begin())); // 移除第二个元素
3 std::cout << us << std::endl; // [3] { 8, 7, 2 }
4
5 us.erase(7); // 移除具体值
6 std::cout << us << std::endl; // [2] { 8, 2 }
7
8 // iterator erase( const_iterator first, const_iterator last );

```

## 合并两个集合

$O(N)$

```

1 std::unordered_set<char>
2     p{'C', 'B', 'B', 'A'},
3     q{'E', 'D', 'E', 'C'};
4
5 p.merge(q);
6 std::cout << "p: " << p << std::endl; // p: [5] { E, D, A, B, C }
7 std::cout << "q: " << q << std::endl; // q: [1] { C }

```

## 返回匹配特定键的元素数量

只要set中存在这个元素，count就返回1，否则返回0

```

1  std::unordered_set set{2, 7, 1, 8, 2, 8, 1, 8, 2, 8};
2  std::cout << set << std::endl; // [4] { 8, 1, 7, 2 }
3
4  const auto [min, max] = std::ranges::minmax(set); // since c++ 20
5  for (int i = min; i <= max; ++i) {
6      if (set.count(i) == 1) {
7          std::cout << i << ' ';
8      }
9  }
10 std::cout << std::endl;

```

## 查找指定元素

找得到返回迭代器，找不到返回 end()

```

1  if (auto res = set.find(2); res != set.end()) {
2      std::cout << "Find: " << *res << std::endl;
3  } else {
4      std::cout << "Not found!" << std::endl;
5  }

```

## 判断集合中是否存在该元素 *since c++ 20*

```

1  for (int x : {2, 5}) {
2      if (set.contains(x)) {
3          std::cout << "Find: " << x << std::endl; // Find: 2
4      } else {
5          std::cout << "Not found: " << x << std::endl; // Not found: 5
6      }
7  }

```

## 清空集合

## 清空集合

```

1  set.clear();
2  std::cout << set << std::endl; // [0] { }

```

## multiset

底层实现：红黑树

允许元素重复

基本操作 $O(\log n)$ ，证明：摊还分析/势函数

```

1  std::multiset<int> ms = {1, 2, 2, 3, 3};
2  assert(ms.size() == 5);
3  for(const auto& it : ms) {
4      std::cout << it << ' ';
5  }
6  std::cout << std::endl;
7

```

```

8 // insert
9 ms.insert(2);
10 assert(ms.size() == 6);
11 for(const auto& it : ms) {
12     std::cout << it << ' ';
13 }
14 std::cout << std::endl;
15
16 // merge since c++ 17
17 std::multiset<int> ms2 = {2, 2, 3, 3, 4};
18 ms.merge(ms2);
19 assert(ms.size() == 11);
20 for(const auto& it : ms) {
21     std::cout << it << ' ';
22 }
23 std::cout << std::endl;
24
25 // count
26 // ms: 1 2 2 2 2 2 3 3 3 3 4
27 assert(ms.count(2) == 5);
28 assert(ms.count(3) == 4);
29
30 // find
31 assert(ms.find(1) == ms.begin());
32 assert(ms.find(100) == ms.end());
33
34 // contains since c++ 20
35 assert(ms.contains(1));
36 assert(ms.contains(2));
37 assert(ms.contains(3));
38 assert(ms.contains(4));
39 assert(!ms.contains(100));
40
41 // ==
42 // ms: 1 2 2 2 2 2 3 3 3 3 4
43 std::multiset<int> ms3 = {1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4};
44 assert(ms == ms3);
45
46 // erase
47 ms.erase(2);
48 assert(ms.size() == 6);
49 // ms: 1 3 3 3 3 4
50 for(const auto& it : ms) {
51     std::cout << it << ' ';
52 }
53 std::cout << std::endl;
54
55 ms.erase(std::next(ms.begin()));
56 assert(ms.size() == 5);
57 // ms: 1 3 3 3 4
58 for(const auto& it : ms) {
59     std::cout << it << ' ';
60 }
61 std::cout << std::endl;

```

## 自定义结构体

### 自定义结构体

```
1  #include <bits/stdc++.h>
2
3  struct X {
4      int a, b;
5  };
6
7  // 按b逆序
8  struct Cmp {
9      bool operator()(const X& o1, const X& o2) const {
10         return o1.b > o2.b;
11     }
12 };
13
14 int main() {
15     std::multiset<X, Cmp> ms = {{1, 1}, {1, 2}, {1, 3}};
16     /*
17     1 3
18     1 2
19     1 1
20     */
21     for(const auto& it : ms) {
22         std::cout << it.a << ' ' << it.b << std::endl;
23     }
24 }
```

## Map

### 红黑树实现

键值对(key/value)容器，迭代器可以修改value，不能修改key。Map会根据key自动排序。key不一定是int类型，只要是重载的<操作符的类型均可

```
1  map<int, string> m;
2  m.count(k); // 返回m中键值等于k的元素个数
3  m.find(k); // 存在返回指向元素的迭代器，否则返回end()
4  m.erase(k); // 删除m中键为k的元素，返回删除元素的个数。
5  m.erase(p); // 删除迭代器p所指向的元素
6  m.insert(e); // e是一个用在m上的一个pair，如果e.first不在m中，则插入一个值为e.second的新元素；如果该键在m中存在，不做任何操作。
7  m.clear(); // 清空
8  m.empty(); // 判空
```

### 访问

### 访问

```

1 map<int,string>::iterator iter;
2 for(iter = mp.begin();iter != mp.end();iter++)
3 {
4     cout<<iter->first<<" "<<iter->second<<endl;
5 }

```

```

1 auto it = mp.begin();
2 for(auto &[k,v]:mp)
3 {
4
5 }

```

自定义结构体：只能对键自定义排序

```

1 #include <bits/stdc++.h>
2
3 struct X {
4     int a, b;
5 };
6
7 // 按b逆序
8 struct Cmp {
9     bool operator()(const X& o1, const X& o2) const {
10         return o1.b > o2.b;
11     }
12 };
13
14 int main() {
15     std::map<X, int, Cmp> mp = {{{1, 1}, 1}, {{1, 2}, 1}, {{1, 3}, 1}};
16     // 1-3:1 1-2:1 1-1:1
17     for(const auto& [k, v] : mp) {
18         std::cout << k.a << '-' << k.b << ':' << v << ' ';
19     }
20     std::cout << std::endl;
21 }

```

统计出现次数

统计出现次数

```

1 #include <bits/stdc++.h>
2
3 int main() {
4     int n = 10;
5     int a[n] = {1, 5, 5, 5, 4, 4, 4, 4, 2, 3};
6
7     std::map<int, int> mp; // k-num v-cnt
8     for(int i = 0; i < n; ++i) {
9         mp[a[i]]++;
10    }
11
12    for(const auto& [num, cnt] : mp) {
13        std::cout << num << ':' << cnt << std::endl;
14    }

```

## unordered\_map

哈希表实现，操作时间复杂度 $O(1)$ ，证明：摊还分析/势函数

```

1  std::unordered_map<int, int> mp = {{1, 1}, {2, 1}, {3, 1}};
2  assert(mp.size() == 3);
3  for(const auto&[k, v] : mp) {
4      std::cout << k << ':' << v << ' ';
5  }
6  std::cout << std::endl;
7
8  // insert
9  mp.insert({4, 1});
10 assert(mp.size() == 4);
11 mp.insert({4, 2}); // 键相同，不会插进去
12 // mp: 4:1 3:1 2:1 1:1
13 assert(mp.size() == 4);
14 assert(mp[4] == 1);
15
16 // []
17 mp[3] = 2; // 修改
18 assert(mp.size() == 4);
19 assert(mp[3] == 2);
20
21 // erase
22 mp.erase(4);
23 assert(mp.size() == 3);
24 // mp: 3:2 2:1 1:1
25 for(const auto&[k, v] : mp) {
26     std::cout << k << ':' << v << ' ';
27 }
28 std::cout << std::endl;
29
30 // find
31 assert(mp.find(3)->second == 2);
32 assert(mp.find(4) == mp.end());
33
34 // count
35 assert(mp.count(3) == 1);
36 assert(mp.count(2) == 1);
37 assert(mp.count(1) == 1);
38 assert(mp.count(4) == 0);
39
40 // contains since c++ 20
41 assert(mp.contains(3));
42 assert(mp.contains(2));
43 assert(mp.contains(1));
44 assert(!mp.contains(4));
45
46 // iterator
47 // 3:2 2:1 1:1
48 for(auto it = mp.begin(); it != mp.end(); ++it) {
49     std::cout << it->first << ':' << it->second << ' ';

```

```
50 }
51 std::cout << std::endl;
```

## multimap

红黑树实现，摊还时间 $O(\log n)$

允许键重复

```
1  std::multimap<int, int> mp = {{1, 1}, {1, 2}, {1, 3}};
2  assert(mp.size() == 3);
3  // 1:1 1:2 1:3
4  for (const auto &[k, v] : mp) {
5      std::cout << k << ':' << v << ' ';
6  }
7  std::cout << std::endl;
8
9  // insert
10 mp.insert({1, 4});
11 assert(mp.size() == 4);
12 mp.insert({2, 1});
13 assert(mp.size() == 5);
14 // 1:1 1:2 1:3 1:4 2:1
15 for (const auto &[k, v] : mp) {
16     std::cout << k << ':' << v << ' ';
17 }
18 std::cout << std::endl;
19
20 // erase
21 mp.erase(1);
22 assert(mp.size() == 1);
23 // 2:1
24 for (const auto &[k, v] : mp) {
25     std::cout << k << ':' << v << ' ';
26 }
27 std::cout << std::endl;
28
29 mp.insert({1, 1});
30 mp.insert({1, 2});
31 mp.insert({1, 3});
32
33 // count
34 // 1:1 1:2 1:3 2:1
35 assert(mp.count(1) == 3);
36 assert(mp.count(2) == 1);
37
38 // find
39 // 寻找键等于 key 的元素。若容器中有数个拥有所请求的键的元素，则可能返回任意一个
40 std::cout << mp.find(1)->second << std::endl;
41
42 // contains
43 assert(mp.contains(1));
44 assert(mp.contains(2));
45 assert(!mp.contains(100));
46
```



```

47 // 1:1 1:2 1:3 2:1
48 // lower_bound 返回指向首个不小于（即大于或等于）key 的元素的迭代器。
49 assert(mp.lower_bound(1)->first == 1);
50 assert(mp.lower_bound(1)->second == 1);
51
52 // upper_bound 返回指向首个大于 key 的元素的迭代器。
53 assert(mp.upper_bound(1)->first == 2);
54 assert(mp.upper_bound(1)->second == 1);

```

自定义结构体：只能对键自定义排序

自定义结构体：只能对键自定义排序

```

1  #include <bits/stdc++.h>
2
3  struct X {
4      int a, b;
5  };
6
7  // 按b逆序
8  struct Cmp {
9      bool operator()(const X& o1, const X& o2) const {
10         return o1.b > o2.b;
11     }
12 };
13
14 int main() {
15     std::multimap<X, int, Cmp> mp = {{{1, 1}, 1}, {{1, 2}, 1}, {{1, 3}, 1}};
16     // 1-3:1 1-2:1 1-1:1
17     for(const auto&[k, v] : mp) {
18         std::cout << k.a << '-' << k.b << ':' << v << ' ';
19     }
20     std::cout << std::endl;
21 }

```