BEIJING NORMAL UNIVERSITY
SCHOOL OF MATHEMATICS

Template

app1eDog

2023 年 5 月 12 日

目录

1   头文件

1.1   模板

```cpp
// created on Lucian Xu's Laptop

#include <bits/stdc++.h>

// using namespace std;

typedef unsigned int UI;
typedef unsigned long long ULL;
typedef long long LL;
typedef unsigned long long ULL;
typedef __int128 i128;
typedef std::pair<int, int> PII;
typedef std::pair<int, LL> PIL;
typedef std::pair<LL, int> PLI;
typedef std::pair<LL, LL> PLL;
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<LL> vl;
typedef std::vector<vl> vvl;
typedef std::vector<PII> vpi;

#define typet typename T
#define typeu typename U
#define types typename... Ts
#define tempt template <typet>
#define tempu template <typeu>
#define temps template <types>
#define tandu template <typet, typeu>

#define ff first
#define ss second
#define endl '\n'
#define all(v) v.begin(), v.end()
#define rall(v) v.rbegin(), v.rend()

#ifdef LOCAL
#include "debug.h"
#else
#define debug(...) \
    do {           \
    } while (false)
#endif

constexpr int mod = 998244353;
constexpr int inv2 = (mod + 1) / 2;
constexpr int inf = 0x3f3f3f3f;
constexpr LL INF = 1e18;
constexpr double pi = 3.141592653589793;
constexpr double eps = 1e-6;

constexpr int lowbit(int x) { return x & -x; }
constexpr int add(int x, int y) { return x + y < mod ? x + y : x - mod + y; }
constexpr int sub(int x, int y) { return x < y ? mod + x - y : x - y; }
constexpr int mul(LL x, int y) { return x * y % mod; }
constexpr void Add(int& x, int y) { x = add(x, y); }
constexpr void Sub(int& x, int y) { x = sub(x, y); }
constexpr void Mul(int& x, int y) { x = mul(x, y); }
constexpr int pow(int x, int y, int z = 1) {
    for (; y; y /= 2) {
        if (y & 1) Mul(z, x);
        Mul(x, x);
    }
    return z;
}
temps constexpr int add(Ts... x) {
    int y = 0;
    (..., Add(y, x));
    return y;
}
temps constexpr int mul(Ts... x) {
    int y = 1;
    (..., Mul(y, x));
    return y;
}

tandu bool Max(T& x, const U& y) { return x < y ? x = y, true : false; }
tandu bool Min(T& x, const U& y) { return x > y ? x = y, true : false; }

int main() {
```

```
80      std::ios::sync_with_stdio(false);
81      std::cin.tie(0);
82      std::cout.tie(0);
83
84      int t = 1;
85      std::cin >> t;
86      while (t--) {
87
88      }
89      return 0;
90 }
```

1.2 debug.h 文件

```
1  tandu std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p) {
2      return os << '<' << p.ff << ',' << p.ss << endl;
3  }
4
5  template <
6      typet, typename = decltype(std::begin(std::declval<T>())),
7      typename = std::enable_if_t<!std::is_same_v<T, std::string>>>
8  std::ostream& operator<<(std::ostream& os, const T& c) {
9      auto it = std::begin(c);
10     if (it == std::end(c)) return os << "{}";
11     for (os << '{' << *it; ++it != std::end(c); os << ',' << *it)
12         ;
13     return os << '}';
14 }
15
16 #define debug(arg...)              \
17     do {                           \
18         std::cerr << "[" #arg "] :"; \
19         dbg(arg);                  \
20     } while (false)
21
22 temps void dbg(Ts... args) {
23     (..., (std::cerr << ' ' << args));
24     std::cerr << endl;
25 }
```

md5: c29c0bb4ac2d5e2bb3fbd3ea57ecdadb

By MAOoo.

```
1  #include <bits/stdc++.h>
2
3  #define debug(arg...)               \
4      do {                            \
5          std::cerr << "[" #arg "] :"; \
6          dbg(arg);                   \
7      } while (false)
8
9  template <typename T, typename U>
10 std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p);
11 template <typename T, typename, typename>
12 std::ostream& operator<<(std::ostream& os, const T& a);
13 template <typename... Ts>
14 std::ostream& operator<<(std::ostream& os, const std::tuple<Ts...>& t);
15
16 template <typename T, typename U>
17 std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p) {
18     return os << '<' << p.first << ',' << p.second << '>';
19 }
20
21 template <
22     typename T, typename = std::enable_if_t<!std::is_same_v<T, std::string>>,
23     typename It = decltype(std::begin(std::declval<T>()))>
24 std::ostream& operator<<(std::ostream& os, const T& a) {
25     constexpr bool flag = std::is_same_v<
26         typename std::iterator_traits<It>::iterator_category, std::random_access_iterator_tag>;
27     constexpr char L = flag ? '[' : '{';
28     constexpr char R = flag ? ']' : '}';
29     auto it = std::begin(a);
30     if (it == std::end(a)) return os << L << R;
31     for (os << L << *it++; it != std::end(a); it++) os << ',' << *it;
32     return os << R;
33 }
34
35 template <typename T>
36 std::ostream& operator<<(std::ostream& os, std::priority_queue<T> a) {
37     std::vector<T> b;
```

```
38          for (b.reserve(a.size()); not a.empty(); a.pop()) {
39              b.push_back(a.top());
40          }
41          return os << b;
42      }
43
44      template <typename Tuple, std::size_t... Is>
45      void print_tuple_impl(std::ostream& os, const Tuple& t, std::index_sequence<Is...>) {
46          ((os << (Is == 0 ? '<' : ',') << std::get<Is>(t)), ...);
47          os << '>';
48      }
49
50      template <typename... Ts>
51      std::ostream& operator<<(std::ostream& os, const std::tuple<Ts...>& t) {
52          print_tuple_impl(os, t, std::index_sequence_for<Ts...>{});
53          return os;
54      }
55
56      template <typename... Ts>
57      void dbg(Ts... args) {
58          (..., (std::cerr << ' ' << args));
59          std::cerr << endl;
60      }
```

2   数据结构

2.1   栈

2.1.1   单调栈

维护单调下降序列.

```
for (int i = 1; i <= n; i++){
    while (!stk.empty() and stk.back() > a[i]) {
      stk.pop_back();
    }
    stk.pop_back(a[i]);
}
```

2.2   队列

2.2.1   单调队列 (滑动窗口)

维护长度不超过 k 的单调下降的序列.

```
std::deque<int> q;
for (int i = 1; i <= n; i++) {
    while (!q.emprty and a[q.back()] >= a[i]) p.pop_back();
    if (!q.emprty() and i - q.front() >= k) q.pop_front();
    q.push_back(i);
}
```

2.3   DSU

```
vi fa(n + 1);
std::iota(all(fa), 0);
std::function<void(int)> find = [&] (int x) -> int{
    return x == fa[x] ? x : fa[x] = find(fa[x]);
};
auto merge = [&] (int x, int y) -> void{
    x = find(x), y = find(y);
    if (x == y) return;
    // operations //
    fa[y] = x;
};
```

2.4   ST 表

用于解决可重复贡献问题的数据结构.

可重复问题是指对运算 opt, 满足 $x\ opt\ x = x$.

2.4.1   一维 ST 表

以最大值为例.

```
// ST //
vvi f(n + 1, vi(30));
vi Log2(n + 1);
auto ST_init = [&]() -> void {
    Log2[1] = 0;
    for (int i = 1; i <= n; i++) {
        f[i][0] = a[i];
        if (i > 1) Log2[i] = Log2[i / 2] + 1;
    };
```

```cpp
10      int t = Log2[n];
11      for (int j = 1; j <= t; j++) {
12          for (int i = 1; i <= n - (1 << j) + 1) {
13              f[i][j] = std::max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
14          }
15      }
16  };
17
18  auto ST_query = [&](int l, int r) -> int {
19      int t = Log2[r - l + 1];
20      return std::max(f[l][t], f[r - (1 << t) + 1][t]);
21  };
```

2.4.2   二维 ST 表

```cpp
1   // ST //
2   std::vector f(n + 1, std::vector<std::array<std::array<int, 30>, 30>>(m + 1));
3   vi Log2(n + 1);
4   auto ST_init = [&]() -> void {
5       for (int i = 2; i <= std::max(n, m); i++) {
6           Log2[i] = Log2[i / 2] + 1;
7       }
8       for (int i = 2; i <= n; i++) {
9           for (int j = 2; j <= m; j++) {
10              f[i][j][0][0] = a[i][j];
11          }
12      }
13      for (int ki = 0; ki <= Log2[n]; ki++) {
14          for (int kj = 0; kj <= Log2[n]; kj++) {
15              if (!ki && !kj) continue;
16              for (int i = 1; i <= n - (1 << ki) + 1; i++) {
17                  for (int j = 1; j <= m - (1 << kj) + 1; j++) {
18                      if (ki) {
19                          f[i][j][ki][kj] =
20                              std::max(f[i][j][ki - 1][kj], f[i + (1 << (ki - 1))][j][ki - 1][kj]);
21                      } else {
22                          f[i][j][ki][kj] =
23                              std::max(f[i][j][ki][kj - 1], f[i][j + (1 << (kj - 1))][ki][kj - 1]);
24                      }
25                  }
26              }
27          }
28      }
29  };
30  auto ST_query = [&](int x1, int y1, int x2, int y2) -> int {
31      int ki = Log2[x2 - x1 + 1], kj = Log2[y2 - y1 + 1];
32      int t1 = f[x1][y1][ki][kj];
33      int t2 = f[x2 - (1 << ki) + 1][y1][ki][kj];
34      int t3 = f[x1][y2 - (1 << kj) + 1][ki][kj];
35      int t4 = f[x2 - (1 << ki) + 1][y2 - (1 << kj) + 1][ki][kj];
36      return std::max({t1, t2, t3, t4});
37  };
```

2.5   树状数组

2.5.1   单点修改, 区间查询

单点修改: a_x 加上 k. 区间查询: a_1 至 a_x 的和.

```cpp
1   // BIT //
2   vi tr(n + 1);
3   auto add = [&] (int x, int k) -> void {
4       while(x <= n){
5           tr[x] += k;
6           x += lowbit(x);
7       }
8   };
9
10  auto query = [&] (int x) -> int {
11      int ans = 0;
12      while(x){
13          ans += tr[x];
14          x -= lowbit(x);
15      }
16      return ans;
17  };
```

2.5.2   区间修改, 单点查询

设数组 b 为数组 a 的差分数组, 维护数组 b.

区间修改: a_l 至 a_r 每个数加 k. 单点查询: 查询 s_n 的值.

```
1  add(l, k);
2  add(r + 1, -k);
3  query(n)
```

2.5.3   区间修改, 区间查询

设数组 b 为数组 a 的差分数组, c_1 维护 b_i, c_2 维护 $i \times b_i$.

区间修改: a_l 至 a_r 每个数加 k. 区间查询: a_1 至 a_x 的和.

```
1  add(l, k);
2  add(r + 1, -k);
3  add(l, l * k);
4  add(r + 1, -(r + 1) * k)
5  ans = query(x) * (x + 1) - query(x);
```

2.6   线段树

包括 $build$, $push\_up$, $push\_down$, $modify$, $query$ 五个函数.

2.6.1   单点修改

求满足 $i < j < k$ 且 $a_i < a_j < a_k$ 的三元组的个数.

```
1  // Problem: 洛谷: P1637 三元上升子序列
2
3  int n, m, a[N], per[N], suf[N];
4  LL ans;
5
6  struct node{
7      int l, r, sum;
8  }tree[N * 4];
9
10 void push_up(int u){
11     tree[u].sum = tree[u << 1].sum + tree[u << 1 | 1].sum;
12 }
13
14 void build(int u, int l, int r){
15     tree[u] = {l, r, 0};
16     if(l != r){
17         int mid = (l + r) >> 1;
18         build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
19         push_up(u);
20     }
21 }
22
23 void modify(int u, int x, int v){
24     if(tree[u].l == x && tree[u].r == x){
25         tree[u].sum += v;
26         return;
27     }
28     int mid = (tree[u].l + tree[u].r) >> 1;
29     if(x <= mid) modify(u << 1, x, v);
30     else modify(u << 1 | 1, x, v);
31     push_up(u);
32 }
33
34 int query(int u, int l, int r){
35     int L = tree[u].l, R = tree[u].r;
36     if(l <= L && R <= r) return tree[u].sum;
37     int mid = (L + R) >> 1, ans = 0;
38     if(l <= mid) ans = query(u << 1, l, r);
39     if(r > mid) ans += query(u << 1 | 1, l, r);
40     return ans;
41 }
```

```
42
43 | int main(){
44 |
45 |     ios::sync_with_stdio(false);
46 |     cin.tie(0);
47 |     cout.tie(0);
48 |
49 |     build(1, 1, 100000);
50 |     cin >> n;
51 |     rep(i, 1, n){
52 |         cin >> a[i];
53 |         modify(1, a[i], 1);
54 |         if(a[i] > 1) per[i] = query(1, 1, a[i] - 1);
55 |     }
56 |     build(1, 1, 100000);
57 |     per(i, n, 1){
58 |         modify(1, a[i], 1);
59 |         if(a[i] < 100000) suf[i] = query(1, a[i] + 1, 100000);
60 |     }
61 |     rep(i, 1, n) ans += (LL)per[i] * suf[i];
62 |     cout << ans << endl;
63 |
64 |     return 0;
65 | }
```

2.6.2   区间修改 (带 add 的 $lazy\_tag$)

n 个数, m 次操作, 操作分为:

- $1\ x\ y\ k$: 将区间 $[x,\ y]$ 中的数每个加上 k.

- $2\ x\ y$: 输出区间 $[x,\ y]$ 中数的和.

```
 1 | // Problem: 洛谷: P3372 【模板】线段树 1
 2 |
 3 | int n, m, op, l, r;
 4 | LL a[N], k;
 5 |
 6 | struct node{
 7 |     int l, r;
 8 |     LL sum, add;
 9 | }tree[N * 4];
10 |
11 | void push_up(int u){
12 |     tree[u].sum = tree[u << 1].sum + tree[u << 1 | 1].sum;
13 | }
14 |
15 | void push_down(int u){
16 |     auto &root = tree[u], &left = tree[u << 1], &right = tree[u << 1 | 1];
17 |     left.sum += (LL)(left.r - left.l + 1) * root.add;
18 |     right.sum += (LL)(right.r - right.l + 1) * root.add;
19 |     left.add += root.add, right.add += root.add;
20 |     root.add = 0;
21 | }
22 |
23 | void build(int u, int l, int r){
24 |     if(l == r) tree[u] = {l, l, a[l], 0};
25 |     else{
26 |         tree[u] = {l, r, 0, 0};
27 |         int mid = (l + r) >> 1;
28 |         build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
29 |         push_up(u);
30 |     }
31 | }
32 |
33 | void modify(int u, int l, int r, LL k){
34 |     int L = tree[u].l, R = tree[u].r;
35 |     if(l <= L && R <= r){
36 |         tree[u].sum += (LL)(R - L + 1) * k;
37 |         tree[u].add += k;
38 |     }
39 |     else{
40 |         push_down(u);
41 |         int mid = (L + R) >> 1;
42 |         if(l <= mid) modify(u << 1, l, r, k);
43 |         if(r > mid) modify(u << 1 | 1, l, r, k);
44 |         push_up(u);
45 |     }
46 | }
47 |
48 | LL query(int u, int l, int r){
```

```
49          int L = tree[u].l, R = tree[u].r;
50          if(l <= L && R <= r){
51              return tree[u].sum;
52          }
53          else{
54              push_down(u);
55              int mid = (L + R) >> 1;
56              LL ans= 0;
57              if(l <= mid) ans = query(u << 1, l, r);
58              if(r > mid) ans += query(u << 1 | 1, l, r);
59              return ans;
60          }
61 }
62
63 int main(){
64
65      ios::sync_with_stdio(false);
66      cin.tie(0);
67      cout.tie(0);
68
69      cin >> n >> m;
70      rep(i, 1, n){
71          cin >> a[i];
72      }
73      build(1, 1, n);
74      rep(i, 1, m){
75          cin >> op >> l >> r;
76          if(op == 2) cout << query(1, l, r) << endl;
77          else{
78              cin >> k;
79              modify(1, l, r, k);
80          }
81      }
82
83      return 0;
84 }
```

2.6.3 区间修改 (带 add 和 mul 的 $lazy\_tag$)

n 个数, m 次操作, 操作分为:

- 1 x y k: 将区间 $[x, y]$ 中的数每个乘以 k.

- 2 x y k : 将区间 $[x, y]$ 中的数每个加上 k.

- 3 x y: 输出区间 $[x, y]$ 中数的和. (对 p 取模)

```
1  // Problem: 洛谷: P3373 【模板】线段树 2
2
3  int n, m, l, r, op;
4  LL a[N], p, k;
5
6  struct node{
7      int l, r;
8      LL sum, add, mul;
9  }tree[N * 4];
10
11 void push_up(int u){
12     tree[u].sum = (tree[u << 1].sum + tree[u << 1 | 1].sum) % p;
13 }
14
15 void operate(node &root, LL add, LL mul){
16     root.sum = (root.sum * mul + (LL)(root.r - root.l + 1) * add) % p;
17     root.add = (root.add * mul + add) % p;
18     root.mul = root.mul * mul % p;
19 }
20
21 void push_down(int u){
22     operate(tree[u << 1], tree[u].add, tree[u].mul);
23     operate(tree[u << 1 | 1], tree[u].add, tree[u].mul);
24     tree[u].add = 0, tree[u].mul = 1;
25 }
26
27 void build(int u, int l, int r){
28     if(l == r){
29         tree[u] = {l, l, a[l], 0, 1};
30     }
31     else{
32         tree[u] = {l, r, 0, 0, 1};
33         int mid = (l + r) >> 1;
34         build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
```

```
35          push_up(u);
36      }
37  }
38
39  void modify(int u, int l, int r, LL add, LL mul){
40      int L = tree[u].l, R = tree[u].r;
41      if(l <= L && R <= r){
42          operate(tree[u], add, mul);
43      }
44      else{
45          push_down(u);
46          int mid = (L + R) >> 1;
47          if(l <= mid) modify(u << 1, l, r, add, mul);
48          if(r > mid) modify(u << 1 | 1, l, r, add, mul);
49          push_up(u);
50      }
51  }
52
53  LL query(int u, int l, int r){
54      int L = tree[u].l, R = tree[u].r;
55      if(l <= L && R <= r){
56          return tree[u].sum;
57      }
58      else{
59          push_down(u);
60          int mid = (L + R) >> 1;
61          LL ans = 0;
62          if(l <= mid) ans = query(u << 1, l, r);
63          if(r > mid) ans = (ans + query(u << 1 | 1, l, r)) % p;
64          return ans;
65      }
66  }
67
68  int main(){
69
70      ios::sync_with_stdio(false);
71      cin.tie(0);
72      cout.tie(0);
73
74      cin >> n >> m >> p;
75      rep(i, 1, n) cin >> a[i];
76      build(1, 1, n);
77      rep(i, 1, m){
78          cin >> op >> l >> r;
79          if(op == 1){
80              cin >> k;
81              modify(1, l, r, 0, k);
82          }
83          else if(op == 2){
84              cin >> k;
85              modify(1, l, r, k, 1);
86          }
87          else cout << query(1, l, r) << endl;
88      }
89
90      return 0;
91  }
```

2.7   线段树 2.0 (23.05.12)

以维护区间最大值和带加法修改的区间和为例.

需要修改的内容: Info 和 Tag 两个 struct 以及 infoApply 和 tagApple 两个函数.

```
1   struct Info {
2       LL sum = 0, maxx = 0;
3
4       Info(LL _sum = 0, LL _maxx = 0) : sum(_sum), maxx(_maxx) {}
5
6       Info operator+(const Info& b) const { return Info(sum + b.sum, std::max(maxx, b.maxx)); }
7   };
8
9   struct Tag {
10      LL add = 0;
11
12      Tag(LL _add = 0) : add(_add) {}
13
14      bool operator==(const Tag& b) const { return add == b.add; }
15  };
16
17  void infoApply(Info& a, int l, int r, const Tag& tag) {
18      a.sum += 1ll * (r - l + 1) * tag.add;
```

```cpp
19        a.maxx += tag.add;
20  }
21
22  void tagApply(Tag& a, int l, int r, const Tag& tag) { a.add += tag.add; }
23
24  template <class Info, class Tag>
25  class segTree {
26  #define ls i << 1
27  #define rs i << 1 | 1
28  #define mid ((l + r) >> 1)
29  #define lson ls, l, mid
30  #define rson rs, mid + 1, r
31
32      int n;
33      std::vector<Info> info;
34      std::vector<Tag> tag;
35
36    public:
37      segTree(const std::vector<Info>& init) : n(init.size() - 1) {
38          assert(n > 0);
39          info.resize(4 << std::__lg(n));
40          tag.resize(4 << std::__lg(n));
41          auto build = [&](auto dfs, int i, int l, int r) {
42              if (l == r) {
43                  info[i] = init[l];
44                  return;
45              }
46              dfs(dfs, lson);
47              dfs(dfs, rson);
48              push_up(i);
49          };
50          build(build, 1, 1, n);
51      }
52
53
54    private:
55      void push_up(int i) { info[i] = info[ls] + info[rs]; }
56
57
58      template <class... T>
59      void apply(int i, int l, int r, const T&... val) {
60          ::infoApply(info[i], l, r, val...);
61          ::tagApply(tag[i], l, r, val...);
62      }
63
64      void push_down(int i, int l, int r) {
65          if (tag[i] == Tag{}) return;
66          apply(lson, tag[i]);
67          apply(rson, tag[i]);
68          tag[i] = {};
69      }
70
71    public:
72      template <class... T>
73      void rangeApply(int ql, int qr, const T&... val) {
74          auto dfs = [&](auto dfs, int i, int l, int r) {
75              if (qr < l or r < ql) return;
76              if (ql <= l and r <= qr) {
77                  apply(i, l, r, val...);
78                  return;
79              }
80              push_down(i, l, r);
81              dfs(dfs, lson);
82              dfs(dfs, rson);
83              push_up(i);
84          };
85          dfs(dfs, 1, 1, n);
86      }
87
88      Info rangeAsk(int ql, int qr) {
89          Info res{};
90          auto dfs = [&](auto dfs, int i, int l, int r) {
91              if (qr < l or r < ql) return;
92              if (ql <= l and r <= qr) {
93                  res = res + info[i];
94                  return;
95              }
96              push_down(i, l, r);
97              dfs(dfs, lson);
98              dfs(dfs, rson);
99          };
100         dfs(dfs, 1, 1, n);
101         return res;
102     }
103
104 #undef rson
105 #undef lson
```

```
106    #undef mid
107    #undef rs
108    #undef ls
109    };
```

2.7.1   动态开点权值线段树

如果要实现 $push_up$ 函数, 记得先开点再操作.

```cpp
1    // Problem: 洛谷: P3369 【模板】普通平衡树
2
3    struct node {
4        int id, l, r;
5        int ls, rs;
6        int sum;
7
8        node(int _id, int _l, int _r) : id(_id), l(_l), r(_r) {
9            ls = rs = 0;
10           sum = 0;
11       }
12   };
13
14
15   // Segment tree //
16   int idx = 1;
17   std::vector<node> tree = {node{0, 0, 0}};
18
19   auto new_node = [&](int l, int r) -> int {
20       tree.push_back(node(idx, l, r));
21       return idx++;
22   };
23
24   auto push_up = [&](int u) -> void {
25       tree[u].sum = 0;
26       if (tree[u].ls) tree[u].sum += tree[tree[u].ls].sum;
27       if (tree[u].rs) tree[u].sum += tree[tree[u].rs].sum;
28   };
29
30   auto build = [&]() { new_node(-10000000, 10000000); };
31
32   std::function<void(int, int, int, int)> insert = [&](int u, int l, int r, int x) {
33       if (l == r) {
34           tree[u].sum++;
35           return;
36       }
37       int mid = (l + r - 1) / 2;
38       if (x <= mid) {
39           if (!tree[u].ls) tree[u].ls = new_node(l, mid);
40           insert(tree[u].ls, l, mid, x);
41       } else {
42           if (!tree[u].rs) tree[u].rs = new_node(mid + 1, r);
43           insert(tree[u].rs, mid + 1, r, x);
44       }
45       push_up(u);
46   };
47
48   std::function<void(int, int, int, int)> remove = [&](int u, int l, int r, int x) {
49       if (l == r) {
50           if (tree[u].sum) tree[u].sum--;
51           return;
52       }
53       int mid = (l + r - 1) / 2;
54       if (x <= mid) {
55           if (!tree[u].ls) return;
56           remove(tree[u].ls, l, mid, x);
57       } else {
58           if (!tree[u].rs) return;
59           remove(tree[u].rs, mid + 1, r, x);
60       }
61       push_up(u);
62   };
63
64   std::function<int(int, int, int, int)> get_rank_by_key = [&](int u, int l, int r, int x) -> int {
65       if (l == r) {
66           return 1;
67       }
68       int mid = (l + r - 1) / 2;
69       int ans = 0;
70       if (x <= mid) {
71           if (!tree[u].ls) return 1;
72           ans = get_rank_by_key(tree[u].ls, l, mid, x);
73       } else {
```

```
74          if (!tree[u].rs) return tree[tree[u].ls].sum + 1;
75          if (!tree[u].ls) {
76              ans = get_rank_by_key(tree[u].rs, mid + 1, r, x);
77          } else {
78              ans = get_rank_by_key(tree[u].rs, mid + 1, r, x) + tree[tree[u].ls].sum;
79          }
80      }
81      return ans;
82  };
83
84  std::function<int(int, int, int, int)> get_key_by_rank = [&](int u, int l, int r, int x) -> int {
85      if (l == r) {
86          return l;
87      }
88      int mid = (l + r - 1) / 2;
89      if (tree[u].ls) {
90          if (x <= tree[tree[u].ls].sum) {
91              return get_key_by_rank(tree[u].ls, l, mid, x);
92          } else {
93              return get_key_by_rank(tree[u].rs, mid + 1, r, x - tree[tree[u].ls].sum);
94          }
95      } else {
96          return get_key_by_rank(tree[u].rs, mid + 1, r, x);
97      }
98  };
99
100 std::function<int(int)> get_prev = [&](int x) -> int {
101     int rank = get_rank_by_key(1, -10000000, 10000000, x) - 1;
102     debug(rank);
103     return get_key_by_rank(1, -10000000, 10000000, rank);
104 };
105
106 std::function<int(int)> get_next = [&](int x) -> int {
107     debug(x + 1);
108     int rank = get_rank_by_key(1, -10000000, 10000000, x + 1);
109     debug(rank);
110     return get_key_by_rank(1, -10000000, 10000000, rank);
111 };
```

2.7.2 （权值）线段树合并

```
1   // Problem: 洛谷: P4556 [Vani有约会]雨天的尾巴 /【模板】线段树合并
2
3   struct node {
4       int l, r, id;
5       int ls, rs;
6       int cnt, ans;
7
8       node(int _id, int _l, int _r) : id(_id), l(_l), r(_r) {
9           ls = rs = 0;
10          cnt = ans = 0;
11      }
12  };
13
14  int main() {
15      std::ios::sync_with_stdio(false);
16      std::cin.tie(0);
17      std::cout.tie(0);
18
19      int n, m;
20      std::cin >> n >> m;
21      vvi e(n + 1);
22      vi ans(n + 1);
23      for (int i = 1; i < n; i++) {
24          int u, v;
25          std::cin >> u >> v;
26          e[u].push_back(v);
27          e[v].push_back(u);
28      }
29
30      // Segment tree //
31      int idx = 1;
32      vi rt(n + 1);
33      std::vector<node> tree = {node{0, 0, 0}};
34
35      auto new_node = [&](int l, int r) -> int {
36          tree.push_back(node(idx, l, r));
37          return idx++;
38      };
39
40      auto push_up = [&](int u) -> void {
41          if (!tree[u].ls) {
42              tree[u].cnt = tree[tree[u].rs].cnt;
```

```cpp
            tree[u].ans = tree[tree[u].rs].ans;
        } else if (!tree[u].rs) {
            tree[u].cnt = tree[tree[u].ls].cnt;
            tree[u].ans = tree[tree[u].ls].ans;
        } else {
            if (tree[tree[u].rs].cnt > tree[tree[u].ls].cnt) {
                tree[u].cnt = tree[tree[u].rs].cnt;
                tree[u].ans = tree[tree[u].rs].ans;
            } else {
                tree[u].cnt = tree[tree[u].ls].cnt;
                tree[u].ans = tree[tree[u].ls].ans;
            }
        }
    };

    std::function<void(int, int, int, int, int)> modify = [&](int u, int l, int r, int x, int k) {
        if (l == r) {
            tree[u].cnt += k;
            tree[u].ans = l;
            return;
        }
        int mid = (l + r) >> 1;
        if (x <= mid) {
            if (!tree[u].ls) tree[u].ls = new_node(l, mid);
            modify(tree[u].ls, l, mid, x, k);
        } else {
            if (!tree[u].rs) tree[u].rs = new_node(mid + 1, r);
            modify(tree[u].rs, mid + 1, r, x, k);
        }
        push_up(u);
    };

    std::function<int(int, int, int, int)> merge = [&](int u, int v, int l, int r) -> int {
        // v 的信息传递给 u //
        if (!u) return v;
        if (!v) return u;
        if (l == r) {
            tree[u].cnt += tree[v].cnt;
            return u;
        }
        int mid = (l + r) >> 1;
        tree[u].ls = merge(tree[u].ls, tree[v].ls, l, mid);
        tree[u].rs = merge(tree[u].rs, tree[v].rs, mid + 1, r);
        push_up(u);
        return u;
    };

    // LCA //

    for (int i = 1; i <= n; i++) {
        rt[i] = idx;
        new_node(1, 100000);
    }

    for (int i = 1; i <= m; i++) {
        int u, v, w;
        std::cin >> u >> v >> w;
        int lca = LCA(u, v);
        modify(rt[u], 1, 100000, w, 1);
        modify(rt[v], 1, 100000, w, 1);
        modify(rt[lca], 1, 100000, w, -1);
        if (father[lca][0]) {
            modify(rt[father[lca][0]], 1, 100000, w, -1);
        }
    }

    // dfs //
    std::function<void(int, int)> Dfs = [&](int u, int fa) {
        for (auto v : e[u]) {
            if (v == fa) continue;
            Dfs(v, u);
            merge(rt[u], rt[v], 1, 100000);
        }
        ans[u] = tree[rt[u]].ans;
        if (tree[rt[u]].cnt == 0) ans[u] = 0;
    };

    Dfs(1, 0);

    for (int i = 1; i <= n; i++) {
        std::cout << ans[i] << endl;
    }

    return 0;
}
```

2.8   划分树

n 个数, q 次查询, 每次查询区间 $[l, r]$ 中的第 k 大数.

```cpp
int n, q, k, l, r;
int tree[20][N], toleft[20][N], sorted[N];

void build(int dep, int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    int cnt = mid - l + 1;
    for (int i = l; i <= r; i++) {
        if (tree[dep][i] < sorted[mid]) cnt--;
    }
    int ls = l, rs = mid + 1;
    for (int i = l; i <= r; i++) {
        int flag = 0;
        if (tree[dep][i] < sorted[mid] || (tree[dep][i] == sorted[mid] && cnt > 0)) {
            flag = 1;
            tree[dep + 1][ls++] = tree[dep][i];
            if (tree[dep][i] == sorted[mid]) cnt--;
        } else
            tree[dep + 1][rs++] = tree[dep][i];
        toleft[dep][i] = toleft[dep][i - 1] + flag;
    }
    build(dep + 1, l, mid), build(dep + 1, mid + 1, r);
}

int query(int dep, int ql, int qr, int l, int r, int k) {
    if (l == r) return tree[dep][l];
    int mid = (l + r) >> 1;
    int x = toleft[dep][ql - 1] - toleft[dep][l - 1];
    int y = toleft[dep][qr] - toleft[dep][l - 1];
    int rx = ql - l - x, ry = qr - l - y, len = y - x;
    if (len >= k)
        return query(dep + 1, l + x, l + y - 1, l, mid, k);
    else
        return query(dep + 1, mid + rx + 1, mid + ry + 1, mid + 1, r, k - len);
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    std::cin >> n >> q;
    rep(i, 1, n) std::cin >> sorted[i], tree[1][i] = sorted[i];
    std::sort(sorted + 1, sorted + n + 1);
    build(1, 1, n);
    while (q--) {
        std::cin >> l >> r >> k;
        std::cout << query(1, l, r, 1, n, k) << endl;
    }
    return 0;
}
```

2.9   可持久化线段树

2.9.1   第 1 个例题

n 个数, m 次操作, 操作分别为:

- v_i 1 loc_i $value_i$: 将第 v_i 个版本的 $a[loc_i]$ 修改为 $value_i$

- v_i 2 loc_i: 拷贝第 v_i 个版本, 并查询该版本的 $a[loc_i]$

```cpp
// 洛谷 P3919 【模板】可持久化线段树 1( 可持久化数组 )

struct node {
    int l, r, key;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    int n, m;
```

```
13    std::cin >> n >> m;
14    vi a(n + 1);
15    for (int i = 1; i <= n; i++) {
16        std::cin >> a[i];
17    }
18
19    // hjt segment tree //
20    int idx = 0;
21    vi root(m + 1);
22    std::vector<node> tr(n * 25);
23
24    std::function<int(int, int)> build = [&](int l, int r) -> int {
25        int p = ++idx;
26        if (l == r) {
27            tr[p].key = a[l];
28            return p;
29        }
30        int mid = (l + r) >> 1;
31        tr[p].l = build(l, mid);
32        tr[p].r = build(mid + 1, r);
33        return p;
34    };
35
36    std::function<int(int, int, int, int, int)> modify = [&](int p, int l, int r, int k,
37                                                              int x) -> int {
38        int q = ++idx;
39        tr[q].l = tr[p].l, tr[q].r = tr[p].r;
40        if (tr[q].l == tr[q].r) {
41            tr[q].key = x;
42            return q;
43        }
44        int mid = (l + r) >> 1;
45        if (k <= mid) {
46            tr[q].l = modify(tr[q].l, l, mid, k, x);
47        } else {
48            tr[q].r = modify(tr[q].r, mid + 1, r, k, x);
49        }
50        return q;
51    };
52
53    std::function<int(int, int, int, int)> query = [&](int p, int l, int r, int k) -> int {
54        if (tr[p].l == tr[p].r) {
55            return tr[p].key;
56        }
57        int mid = (l + r) >> 1;
58        if (k <= mid) {
59            return query(tr[p].l, l, mid, k);
60        } else {
61            return query(tr[p].r, mid + 1, r, k);
62        }
63    };
64
65    root[0] = build(1, n);
66
67    for (int i = 1; i <= m; i++) {
68        int op, ver, k, x;
69        std::cin >> ver >> op;
70        if (op == 1) {
71            std::cin >> k >> x;
72            root[i] = modify(root[ver], 1, n, k, x);
73        } else {
74            std::cin >> k;
75            root[i] = root[ver];
76            std::cout << query(root[ver], 1, n, k) << endl;
77        }
78    }
79
80    return 0;
81 }
```

指针写法 (可惜洛谷上 #2 点会 MLE, 更新数据后变成 TLE 了)

```
1  int n, m, k, x, vi, op, a[N];
2
3  struct node {
4      node *ch[2];
5      int key;
6
7      node() {
8          key = 0;
9          ch[0] = ch[1] = nullptr;
10     }
11
12     node(node *_node) {
13         key = _node->key;
14         ch[0] = _node->ch[0], ch[1] = _node->ch[1];
```

```
15         }
16 };
17
18 struct segment_tree {
19     node *root[N];
20
21     node *build(int l, int r) {
22         node *new_node;
23         new_node = new node();
24         if (l == r) {
25             new_node->key = a[l];
26             return new_node;
27         }
28         int mid = (l + r) >> 1;
29         new_node->ch[0] = build(l, mid);
30         new_node->ch[1] = build(mid + 1, r);
31         return new_node;
32     }
33
34     // a[k] 改成 x //
35     node *modify(node *p, int l, int r, int k, int x) {
36         node *new_node;
37         new_node = new node(p);
38         if (l == r) {
39             new_node->key = x;
40             return new_node;
41         }
42         int mid = (l + r) >> 1;
43         if (k <= mid)
44             new_node->ch[0] = modify(new_node->ch[0], l, mid, k, x);
45         else
46             new_node->ch[1] = modify(new_node->ch[1], mid + 1, r, k, x);
47         return new_node;
48     }
49
50     // 询问 p 为根节点的版本的 a[k] //
51     int query(node *p, int l, int r, int k) {
52         if (l == r) {
53             return p->key;
54         }
55         int mid = (l + r) >> 1;
56         if (k <= mid)
57             return query(p->ch[0], l, mid, k);
58         else
59             return query(p->ch[1], mid + 1, r, k);
60     }
61 };
62
63 segment_tree tr;
64
65 int main() {
66     ios::sync_with_stdio(false);
67     cin.tie(0);
68     cout.tie(0);
69
70     cin >> n >> m;
71     rep(i, 1, n) cin >> a[i];
72     tr.root[0] = tr.build(1, n);
73     rep(i, 1, m) {
74         cin >> vi >> op;
75         if (op == 1) {
76             cin >> k >> x;
77             tr.root[i] = tr.modify(tr.root[vi], 1, n, k, x);
78         } else {
79             cin >> k;
80             tr.root[i] = tr.root[vi];
81             cout << tr.query(tr.root[vi], 1, n, k) << endl;
82         }
83     }
84     return 0;
85 }
```

2.9.2　第 2 个例题

长度为 n 的序列 a, m 次查询, 每次查询 $[l, r]$ 中的第 k 小值.

```
1 // 洛谷P3834 【模板】可持久化线段树 2
2
3 struct node {
4     int l, r, cnt;
5 };
6
```

```
7  int main() {
8      std::ios::sync_with_stdio(false);
9      std::cin.tie(0);
10     std::cout.tie(0);
11
12     int n, m;
13     std::cin >> n >> m;
14     vi a(n + 1), v;
15     for (int i = 1; i <= n; i++) {
16         std::cin >> a[i];
17         v.push_back(a[i]);
18     }
19     std::sort(all(v));
20     v.erase(unique(all(v)), v.end());
21     auto find = [&](int x) -> int { return std::lower_bound(all(v), x) - v.begin() + 1; };
22
23     // hjt segment tree //
24     std::vector<node>(n * 25);
25     vi root(n + 1);
26     int idx = 0;
27
28     std::function<int(int, int)> build = [&](int l, int r) -> int {
29         int p = ++idx;
30         if (l == r) return p;
31         int mid = (l + r) >> 1;
32         tr[p].l = build(l, mid), tr[p].r = build(mid + 1, r);
33         return p;
34     };
35
36     std::function<int(int, int, int, int)> modify = [&](int p, int l, int r, int x) -> int {
37         int q = ++idx;
38         tr[q] = tr[p];
39         if (tr[q].l == tr[q].r) {
40             tr[q].cnt++;
41             return q;
42         }
43         int mid = (l + r) >> 1;
44         if (x <= mid) {
45             tr[q].l = modify(tr[q].l, l, mid, x);
46         } else {
47             tr[q].r = modify(tr[q].r, mid + 1, r, x);
48         }
49         tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
50         return q;
51     };
52
53     std::function<int(int, int, int, int, int)> query = [&](int p, int q, int l, int r,
54                                                             int x) -> int {
55         if (l == r) return l;
56         int cnt = tr[tr[p].l].cnt - tr[tr[q].l].cnt;
57         int mid = (l + r) >> 1;
58         if (x <= cnt) {
59             return query(tr[p].l, tr[q].l, l, mid, x);
60         } else {
61             return query(tr[p].r, tr[q].r, mid + 1, r, x - cnt);
62         }
63     };
64
65     root[0] = build(1, v.size());
66
67
68     for (int i = 1; i <= n; i++) {
69         root[i] = modify(root[i - 1], 1, v.size(), find(a[i]));
70     }
71     for (int i = 1; i <= m; i++) {
72         int l, r, k;
73         std::cin >> l >> r >> k;
74         std::cout << v[query(root[r], root[l - 1], 1, v.size(), k) - 1] << endl;
75     }
76
77     return 0;
78 }
```

指针写法

```
1  int n, m, a[N];
2  vector<int> v;
3
4  int find(int x) { return lower_bound(all(v), x) - v.begin() + 1; }
5
6  struct node {
7      node *ch[2];
8      int cnt;
9
10     node() {
11         cnt = 0;
```

```
12              ch[0] = ch[1] = nullptr;
13          }
14
15      node(node *_node) {
16              cnt = _node->cnt;
17              ch[0] = _node->ch[0], ch[1] = _node->ch[1];
18          }
19  };
20
21  struct segment_tree {
22      node *root[N];
23
24      node *build(int l, int r) {
25          node *new_node;
26          new_node = new node();
27          if (l == r) {
28              return new_node;
29          }
30          int mid = (l + r) >> 1;
31          new_node->ch[0] = build(l, mid);
32          new_node->ch[1] = build(mid + 1, r);
33          return new_node;
34      }
35
36      node *modify(node *p, int l, int r, int x) {
37          node *new_node;
38          new_node = new node(p);
39          if (l == r) {
40              new_node->cnt++;
41              return new_node;
42          }
43          int mid = (l + r) >> 1;
44          if (x <= mid)
45              new_node->ch[0] = modify(new_node->ch[0], l, mid, x);
46          else
47              new_node->ch[1] = modify(new_node->ch[1], mid + 1, r, x);
48          new_node->cnt = new_node->ch[0]->cnt + new_node->ch[1]->cnt;
49          return new_node;
50      }
51
52      int query(node *p, node *q, int l, int r, int x) {
53          if (l == r) {
54              return l;
55          }
56          int cnt = p->ch[0]->cnt - q->ch[0]->cnt;
57          int mid = (l + r) >> 1;
58          if (x <= cnt)
59              return query(p->ch[0], q->ch[0], l, mid, x);
60          else
61              return query(p->ch[1], q->ch[1], mid + 1, r, x - cnt);
62      }
63  };
64
65  segment_tree tr;
66
67  int main() {
68      ios::sync_with_stdio(false);
69      cin.tie(0);
70      cout.tie(0);
71
72      cin >> n >> m;
73      rep(i, 1, n) {
74          cin >> a[i];
75          v.p_b(a[i]);
76      }
77
78      sort(all(v));
79      v.erase(unique(all(v)), v.end());
80
81      tr.root[0] = tr.build(1, v.size());
82      rep(i, 1, n) { tr.root[i] = tr.modify(tr.root[i - 1], 1, v.size(), find(a[i])); }
83      rep(i, 1, m) {
84          int l, r, k;
85          cin >> l >> r >> k;
86          cout << v[tr.query(tr.root[r], tr.root[l - 1], 1, v.size(), k) - 1] << endl;
87      }
88      return 0;
89  }
```

2.10   笛卡尔树

一种特殊的平衡树, 用元素的值作为平衡点节点的 val, 元素的下标作为 key.
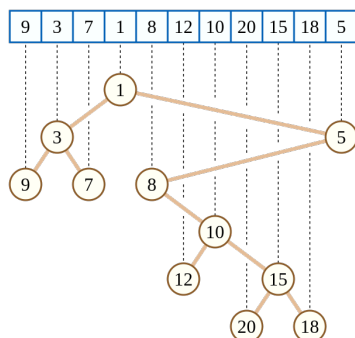
```
1  // cartesian tree //
2  vi ls(n + 1), rs(n + 1), stk(n + 1);
3  int top = 1;
4  for (int i = 1; i <= n; i++) {
5      int k = top;
6      while (k and a[stk[k]] > a[i]) k--;
7      if (k) rs[stk[k]] = i;
8      if (k < top) ls[i] = stk[k + 1];
9      stk[++k] = i;
10     top = k;
11 }
```

2.11　Treap

n 次操作, 操作分为如下 6 种:

- 插入数 x

- 删除数 x(若有多个相同的数, 只删除一个)

- 查询数 x 的排名 (排名定义为小于 x 的数的个数 $+ 1$)

- 查询数 x 的排名

- 求 x 的前驱 (前驱定义为小于 x 的最大数)

- 求 x 的后继 (后继定义为大于 x 的最小数)

2.11.1　旋转 Treap

```
1  // Problem: 洛谷: P3369 【模板】普通平衡树
2
3  int n, root, idx;
4
5  struct node {
6      int l, r;
7      int key, val;
8      int cnt, size;
9  } treap[N];
10
11 void push_up(int p) {
12     treap[p].size = treap[treap[p].l].size + treap[treap[p].r].size + treap[p].cnt;
13 }
14
15 int get_node(int key) {
16     treap[++idx].key = key;
17     treap[idx].val = rand();
18     treap[idx].cnt = treap[idx].size = 1;
19     return idx;
20 }
21
22 void zig(int &p) {
23     // 右旋 //
24     int q = treap[p].l;
25     treap[p].l = treap[q].r, treap[q].r = p, p = q;
26     push_up(treap[p].r), push_up(p);
27 }
28
```

```
29  void zag(int &p) {
30      // 左旋 //
31      int q = treap[p].r;
32      treap[p].r = treap[q].l, treap[q].l = p, p = q;
33      push_up(treap[p].l), push_up(p);
34  }
35
36  void build() {
37      get_node(-inf), get_node(inf);
38      root = 1, treap[1].r = 2;
39      push_up(root);
40      if (treap[1].val < treap[2].val) zag(root);
41  }
42
43  void insert(int &p, int key) {
44      if (!p) {
45          p = get_node(key);
46      } else if (treap[p].key == key) {
47          treap[p].cnt++;
48      } else if (treap[p].key > key) {
49          insert(treap[p].l, key);
50          if (treap[treap[p].l].val > treap[p].val) zig(p);
51      } else {
52          insert(treap[p].r, key);
53          if (treap[treap[p].r].val > treap[p].val) zag(p);
54      }
55      push_up(p);
56  }
57
58  void remove(int &p, int key) {
59      if (!p) return;
60      if (treap[p].key == key) {
61          if (treap[p].cnt > 1) {
62              treap[p].cnt--;
63          } else if (treap[p].l || treap[p].r) {
64              if (!treap[p].r || treap[treap[p].l].val > treap[treap[p].r].val) {
65                  zig(p);
66                  remove(treap[p].r, key);
67              } else {
68                  zag(p);
69                  remove(treap[p].l, key);
70              }
71          } else {
72              p = 0;
73          }
74      } else if {
75          (treap[p].key > key) remove(treap[p].l, key);
76      } else {
77          remove(treap[p].r, key);
78      }
79      push_up(p);
80  }
81
82  int get_rank_by_key(int p, int key) {
83      // 通过数值找排名 //
84      if (!p) return 0;
85      if (treap[p].key == key) return treap[treap[p].l].size;
86      if (treap[p].key > key) return get_rank_by_key(treap[p].l, key);
87      return treap[treap[p].l].size + treap[p].cnt + get_rank_by_key(treap[p].r, key);
88  }
89
90  int get_key_by_rank(int p, int rank) {
91      // 通过排名找数值 //
92      if (!p) return inf;
93      if (treap[treap[p].l].size >= rank) return get_key_by_rank(treap[p].l, rank);
94      if (treap[treap[p].l].size + treap[p].cnt >= rank) return treap[p].key;
95      return get_key_by_rank(treap[p].r, rank - treap[treap[p].l].size - treap[p].cnt);
96  }
97
98  int get_prev(int p, int key) {
99      // 找前驱 //
100     if (!p) return -inf;
101     if (treap[p].key >= key) return get_prev(treap[p].l, key);
102     return max(treap[p].key, get_prev(treap[p].r, key));
103 }
104
105 int get_next(int p, int key) {
106     // 找后继 //
107     if (!p) return inf;
108     if (treap[p].key <= key) return get_next(treap[p].r, key);
109     return min(treap[p].key, get_next(treap[p].l, key));
110 }
111
112 int main() {
113     ios::sync_with_stdio(false);
114     cin.tie(0);
```

```
115        cout.tie(0);
116
117    cin >> n;
118    build();
119    rep(i, 1, n) {
120        int op, x;
121        cin >> op >> x;
122        if (op == 1) {
123            insert(root, x);
124        } else if (op == 2) {
125            remove(root, x);
126        } else if (op == 3) {
127            cout << get_rank_by_key(root, x) << endl;
128        } else if (op == 4) {
129            cout << get_key_by_rank(root, x + 1) << endl;
130        } else if (op == 5) {
131            cout << get_prev(root, x) << endl;
132        } else {
133            cout << get_next(root, x) << endl;
134        }
135    }
136    return 0;
137 }
```

2.11.2   无旋 Treap

```
1  // created on Laptop of Lucian Xu
2
3  struct node {
4      node *ch[2];
5      int key, val;
6      int cnt, size;
7
8      node(int _key) : key(_key), cnt(1), size(1) {
9          ch[0] = ch[1] = nullptr;
10         val = rand();
11     }
12
13     // node(node *_node) {
14     // key = _node->key, val = _node->val, cnt = _node->cnt, size = _node->size;
15     // }
16
17     inline void push_up() {
18         size = cnt;
19         if (ch[0] != nullptr) size += ch[0]->size;
20         if (ch[1] != nullptr) size += ch[1]->size;
21     }
22 };
23
24 struct treap {
25 #define _2 second.first
26 #define _3 second.second
27
28     node *root;
29
30     pair<node *, node *> split(node *p, int key) {
31         if (p == nullptr) return {nullptr, nullptr};
32         if (p->key <= key) {
33             auto temp = split(p->ch[1], key);
34             p->ch[1] = temp.first;
35             p->push_up();
36             return {p, temp.second};
37         } else {
38             auto temp = split(p->ch[0], key);
39             p->ch[0] = temp.second;
40             p->push_up();
41             return {temp.first, p};
42         }
43     }
44
45     pair<node *, pair<node *, node *> > split_by_rank(node *p, int rank) {
46         if (p == nullptr) return {nullptr, {nullptr, nullptr}};
47         int ls_size = p->ch[0] == nullptr ? 0 : p->ch[0]->size;
48         if (rank <= ls_size) {
49             auto temp = split_by_rank(p->ch[0], rank);
50             p->ch[0] = temp._3;
51             p->push_up();
52             return {temp.first, {temp._2, p}};
53         } else if (rank <= ls_size + p->cnt) {
54             node *lt = p->ch[0];
55             node *rt = p->ch[1];
56             p->ch[0] = p->ch[1] = nullptr;
57             return {lt, {p, rt}};
```

```
58              } else {
59                  auto temp = split_by_rank(p->ch[1], rank - ls_size - p->cnt);
60                  p->ch[1] = temp.first;
61                  p->push_up();
62                  return {p, {temp._2, temp._3}};
63              }
64          }
65
66      node *merge(node *u, node *v) {
67          if (u == nullptr && v == nullptr) return nullptr;
68          if (u != nullptr && v == nullptr) return u;
69          if (v != nullptr && u == nullptr) return v;
70          if (u->val < v->val) {
71              u->ch[1] = merge(u->ch[1], v);
72              u->push_up();
73              return u;
74          } else {
75              v->ch[0] = merge(u, v->ch[0]);
76              v->push_up();
77              return v;
78          }
79      }
80
81      void insert(int key) {
82          auto temp = split(root, key);
83          auto l_tr = split(temp.first, key - 1);
84          node *new_node;
85          if (l_tr.second == nullptr) {
86              new_node = new node(key);
87          } else {
88              l_tr.second->cnt++;
89              l_tr.second->push_up();
90          }
91          node *l_tr_combined = merge(l_tr.first, l_tr.second == nullptr ? new_node : l_tr.second);
92          root = merge(l_tr_combined, temp.second);
93      }
94
95      void remove(int key) {
96          auto temp = split(root, key);
97          auto l_tr = split(temp.first, key - 1);
98          if (l_tr.second->cnt > 1) {
99              l_tr.second->cnt--;
100             l_tr.second->push_up();
101             l_tr.first = merge(l_tr.first, l_tr.second);
102         } else {
103             if (temp.first == l_tr.second) temp.first = nullptr;
104             delete l_tr.second;
105             l_tr.second = nullptr;
106         }
107         root = merge(l_tr.first, temp.second);
108     }
109
110     int get_rank_by_key(node *p, int key) {
111         auto temp = split(p, key - 1);
112         int ret = (temp.first == nullptr ? 0 : temp.first->size) + 1;
113         root = merge(temp.first, temp.second);
114         return ret;
115     }
116
117     int get_key_by_rank(node *p, int rank) {
118         auto temp = split_by_rank(p, rank);
119         int ret = temp._2->key;
120         root = merge(temp.first, merge(temp._2, temp._3));
121         return ret;
122     }
123
124     int get_prev(int key) {
125         auto temp = split(root, key - 1);
126         int ret = get_key_by_rank(temp.first, temp.first->size);
127         root = merge(temp.first, temp.second);
128         return ret;
129     }
130
131     int get_nex(int key) {
132         auto temp = split(root, key);
133         int ret = get_key_by_rank(temp.second, 1);
134         root = merge(temp.first, temp.second);
135         return ret;
136     }
137 };
138
139 treap tr;
140
141 int main() {
142     ios::sync_with_stdio(false);
143     cin.tie(0);
144     cout.tie(0);
```

```
145
146     srand(time(0));
147
148     int n;
149     cin >> n;
150     while (n--) {
151         int op, x;
152         cin >> op >> x;
153         if (op == 1) {
154             tr.insert(x);
155         } else if (op == 2) {
156             tr.remove(x);
157         } else if (op == 3) {
158             cout << tr.get_rank_by_key(tr.root, x) << endl;
159         } else if (op == 4) {
160             cout << tr.get_key_by_rank(tr.root, x) << endl;
161         } else if (op == 5) {
162             cout << tr.get_prev(x) << endl;
163         } else {
164             cout << tr.get_nex(x) << endl;
165         }
166     }
167     return 0;
168 }
```

2.11.3 用 01 Trie 实现

使用 01 Trie 只能存在非负数.

速度能快不少, 但只能单点操作, 而且有点费空间.

```
1   // 洛谷 P3369 【模板】普通平衡树
2
3   struct Treap {
4       int id = 1, maxlog = 25;
5       int ch[N * 25][2], siz[N * 25];
6
7       int newnode() {
8           id++;
9           ch[id][0] = ch[id][1] = siz[id] = 0;
10          return id;
11      }
12
13      void merge(int key, int cnt) {
14          int u = 1;
15          for (int i = maxlog - 1; i >= 0; i--) {
16              int v = (key >> i) & 1;
17              if (!ch[u][v]) ch[u][v] = newnode();
18              u = ch[u][v];
19              siz[u] += cnt;
20          }
21      }
22
23      int get_key_by_rank(int rank) {
24          int u = 1, key = 0;
25          for (int i = maxlog - 1; i >= 0; i--) {
26              if (siz[ch[u][0]] >= rank) {
27                  u = ch[u][0];
28              } else {
29                  key |= (1 << i);
30                  rank -= siz[ch[u][0]];
31                  u = ch[u][1];
32              }
33          }
34          return key;
35      }
36
37      int get_rank_by_key(int rank) {
38          int key = 0;
39          int u = 1;
40          for (int i = maxlog - 1; i >= 0; i--) {
41              if ((rank >> i) & 1) {
42                  key += siz[ch[u][0]];
43                  u = ch[u][1];
44              } else {
45                  u = ch[u][0];
46              }
47              if (!u) break;
48          }
49          return key;
50      }
51
```

```
52    int get_prev(int x) { return get_key_by_rank(get_rank_by_key(x)); }
53    int get_next(int x) { return get_key_by_rank(get_rank_by_key(x + 1) + 1); }
54  } treap;
55
56  const int num = 1e7;
57  int n, op, x;
58
59  int main() {
60      std::ios::sync_with_stdio(false);
61      std::cin.tie(0);
62      std::cout.tie(0);
63
64      std::cin >> n;
65      for (int i = 1; i <= n; i++) {
66          std::cin >> op >> x;
67          if (op == 1) {
68              treap.merge(x + num, 1);
69          } else if (op == 2) {
70              treap.merge(x + num, -1);
71          } else if (op == 3) {
72              std::cout << treap.get_rank_by_key(x + num) + 1 << endl;
73          } else if (op == 4) {
74              std::cout << treap.get_key_by_rank(x) - num << endl;
75          } else if (op == 5) {
76              std::cout << treap.get_prev(x + num) - num << endl;
77          } else if (op == 6) {
78              std::cout << treap.get_next(x + num) - num << endl;
79          }
80      }
81      return 0;
82  }
```

2.12   Splay

2.12.1   文艺平衡树

初始为 1 到 n 的序列, m 次操作, 每次将序列下标为 $[l \sim r]$ 的区间翻转.

```
1   // 洛谷 P3391  【模板】文艺平衡树
2
3   struct node {
4       int ch[2], fa, key;
5       int siz, flag;
6
7       void init(int _fa, int _key) { fa = _fa, key = _key, siz = 1; }
8   };
9
10  struct splay {
11      node tr[N];
12      int n, root, idx;
13
14      bool get(int u) { return u == tr[tr[u].fa].ch[1]; }
15
16      void pushup(int u) { tr[u].siz = tr[tr[u].ch[0]].siz + tr[tr[u].ch[1]].siz + 1; }
17
18      void pushdown(int u) {
19          if (tr[u].flag) {
20              std::swap(tr[u].ch[0], tr[u].ch[1]);
21              tr[tr[u].ch[0]].flag ^= 1, tr[tr[u].ch[1]].flag ^= 1;
22              tr[u].flag = 0;
23          }
24      }
25
26      void rotate(int x) {
27          int y = tr[x].fa, z = tr[y].fa;
28          int op = get(x);
29          tr[y].ch[op] = tr[x].ch[op ^ 1];
30          if (tr[x].ch[op ^ 1]) tr[tr[x].ch[op ^ 1]].fa = y;
31          tr[x].ch[op ^ 1] = y;
32          tr[y].fa = x, tr[x].fa = z;
33          if (z) tr[z].ch[y == tr[z].ch[1]] = x;
34          pushup(y), pushup(x);
35      }
36
37      void opt(int u, int k) {
38          for (int f = tr[u].fa; f = tr[u].fa, f != k; rotate(u)) {
39              if (tr[f].fa != k) rotate(get(u) == get(f) ? f : u);
40          }
41          if (k == 0) root = u;
42      }
43
```

```
44 │    void output(int u) {
45 │        pushdown(u);
46 │        if (tr[u].ch[0]) output(tr[u].ch[0]);
47 │        if (tr[u].key >= 1 && tr[u].key <= n) {
48 │            std::cout << tr[u].key << ' ';
49 │        }
50 │        if (tr[u].ch[1]) output(tr[u].ch[1]);
51 │    }
52 │
53 │    void insert(int key) {
54 │        idx++;
55 │        tr[idx].ch[0] = root;
56 │        tr[idx].init(0, key);
57 │        tr[root].fa = idx;
58 │        root = idx;
59 │        pushup(idx);
60 │    }
61 │
62 │    int kth(int k) {
63 │        int u = root;
64 │        while (1) {
65 │            pushdown(u);
66 │            if (tr[u].ch[0] && k <= tr[tr[u].ch[0]].siz) {
67 │                u = tr[u].ch[0];
68 │            } else {
69 │                k -= tr[tr[u].ch[0]].siz + 1;
70 │                if (k <= 0) {
71 │                    opt(u, 0);
72 │                    return u;
73 │                } else {
74 │                    u = tr[u].ch[1];
75 │                }
76 │            }
77 │        }
78 │    }
79 │
80 │} splay;
81 │
82 │int n, m, l, r;
83 │
84 │int main() {
85 │    std::ios::sync_with_stdio(false);
86 │    std::cin.tie(0);
87 │    std::cout.tie(0);
88 │
89 │    std::cin >> n >> m;
90 │    splay.n = n;
91 │    splay.insert(-inf);
92 │    rep(i, 1, n) splay.insert(i);
93 │    splay.insert(inf);
94 │    rep(i, 1, m) {
95 │        std::cin >> l >> r;
96 │        l = splay.kth(l), r = splay.kth(r + 2);
97 │        splay.opt(l, 0), splay.opt(r, l);
98 │        splay.tr[splay.tr[r].ch[0]].flag ^= 1;
99 │    }
100│    splay.output(splay.root);
101│
102│    return 0;
103│}
```

2.12.2　普通平衡树

n 次操作, 操作分为如下 6 种:

- 插入数 x

- 删除数 x(若有多个相同的数, 只删除一个)

- 查询数 x 的排名（排名定义为小于 x 的数的个数 $+1$）

- 查询排名为 x 的数

- 求 x 的前驱（前驱定义为小于 x 的最大数)

- 求 x 的后继（后继定义为大于 x 的最小数)

```
1 │// 洛谷 P3369 【模板】普通平衡树
2 │
```

```cpp
struct node {
    int ch[2], fa, key, siz, cnt;

    void init(int _fa, int _key) { fa = _fa, key = _key, siz = cnt = 1; }

    void clear() { ch[0] = ch[1] = fa = key = siz = cnt = 0; }
};

struct splay {
    node tr[N];
    int n, root, idx;

    bool get(int u) { return u == tr[tr[u].fa].ch[1]; }

    void pushup(int u) { tr[u].siz = tr[tr[u].ch[0]].siz + tr[tr[u].ch[1]].siz + tr[u].cnt; }

    void rotate(int x) {
        int y = tr[x].fa, z = tr[y].fa;
        int op = get(x);
        tr[y].ch[op] = tr[x].ch[op ^ 1];
        if (tr[x].ch[op ^ 1]) tr[tr[x].ch[op ^ 1]].fa = y;
        tr[x].ch[op ^ 1] = y;
        tr[y].fa = x, tr[x].fa = z;
        if (z) tr[z].ch[y == tr[z].ch[1]] = x;
        pushup(y), pushup(x);
    }

    void opt(int u, int k) {
        for (int f = tr[u].fa; f = tr[u].fa, f != k; rotate(u)) {
            if (tr[f].fa != k) {
                rotate(get(u) == get(f) ? f : u);
            }
        }
        if (k == 0) root = u;
    }

    void insert(int key) {
        if (!root) {
            idx++;
            tr[idx].init(0, key);
            root = idx;
            return;
        }
        int u = root, f = 0;
        while (1) {
            if (tr[u].key == key) {
                tr[u].cnt++;
                pushup(u), pushup(f);
                opt(u, 0);
                break;
            }
            f = u, u = tr[u].ch[tr[u].key < key];
            if (!u) {
                idx++;
                tr[idx].init(f, key);
                tr[f].ch[tr[f].key < key] = idx;
                pushup(idx), pushup(f);
                opt(idx, 0);
                break;
            }
        }
    }

    // 返回节点编号 //
    int kth(int rank) {
        int u = root;
        while (1) {
            if (tr[u].ch[0] && rank <= tr[tr[u].ch[0]].siz) {
                u = tr[u].ch[0];
            } else {
                rank -= tr[tr[u].ch[0]].siz + tr[u].cnt;
                if (rank <= 0) {
                    opt(u, 0);
                    return u;
                } else {
                    u = tr[u].ch[1];
                }
            }
        }
    }

    // 返回排名 //
    int nlt(int key) {
        int rank = 0, u = root;
        while (1) {
            if (tr[u].key > key) {
                u = tr[u].ch[0];
```

```
 90 |             } else {
 91 |                 rank += tr[tr[u].ch[0]].siz;
 92 |                 if (tr[u].key == key) {
 93 |                     opt(u, 0);
 94 |                     return rank + 1;
 95 |                 }
 96 |                 rank += tr[u].cnt;
 97 |                 if (tr[u].ch[1]) {
 98 |                     u = tr[u].ch[1];
 99 |                 } else {
100 |                     return rank + 1;
101 |                 }
102 |             }
103 |         }
104 |     }
105 |
106 |     int get_prev(int key) { return kth(nlt(key) - 1); }
107 |
108 |     int get_next(int key) { return kth(nlt(key + 1)); }
109 |
110 |     void remove(int key) {
111 |         nlt(key);
112 |         if (tr[root].cnt > 1) {
113 |             tr[root].cnt--;
114 |             pushup(root);
115 |             return;
116 |         }
117 |         int u = root, l = get_prev(key);
118 |         tr[tr[u].ch[1]].fa = l;
119 |         tr[l].ch[1] = tr[u].ch[1];
120 |         tr[u].clear();
121 |         pushup(root);
122 |     }
123 |
124 |     void output(int u) {
125 |         if (tr[u].ch[0]) output(tr[u].ch[0]);
126 |         std::cout << tr[u].key << ' ';
127 |         if (tr[u].ch[1]) output(tr[u].ch[1]);
128 |     }
129 |
130 | } splay;
131 |
132 | int n, op, x;
133 |
134 | int main() {
135 |     std::ios::sync_with_stdio(false);
136 |     std::cin.tie(0);
137 |     std::cout.tie(0);
138 |
139 |     splay.insert(-inf), splay.insert(inf);
140 |
141 |     std::cin >> n;
142 |     for (int i = 1; i <= n; i++) {
143 |         std::cin >> op >> x;
144 |         if (op == 1) {
145 |             splay.insert(x);
146 |         } else if (op == 2) {
147 |             splay.remove(x);
148 |         } else if (op == 3) {
149 |             std::cout << splay.nlt(x) - 1 << endl;
150 |         } else if (op == 4) {
151 |             std::cout << splay.tr[splay.kth(x + 1)].key << endl;
152 |         } else if (op == 5) {
153 |             std::cout << splay.tr[splay.get_prev(x)].key << endl;
154 |         } else if (op == 6) {
155 |             std::cout << splay.tr[splay.get_next(x)].key << endl;
156 |         }
157 |     }
158 |
159 |     return 0;
160 | }
```

2.13 树套树

2.13.1 线段树套线段树

n 个三维数对 (a_i, b_i, c_i), 设 $f(i)$ 表示 $a_j \leqslant a_i$ 且 $b_j \leqslant b_i$ 且 $c_j \leqslant c_i$ 且 $i \neq j$ 的个数.

输出 $f(i)$ $(0 \leqslant i \leqslant n - 1)$ 的值.

```cpp
// 洛谷 P3810 【模板】三维偏序（ 陌上花开 ）

struct node1 {
    int l, r, root;
} tr1[N << 2];

struct node2 {
    int ch[2], cnt;
} tr2[N << 7];

struct node {
    int x, y, z, cnt;

    bool operator==(const node& a) { return (x == a.x && y == a.y && z == a.z); }

} data[N];

bool cmp(node a, node b) {
    if (a.x != b.x) return a.x < b.x;
    if (a.y != b.y) return a.y < b.y;
    return a.z < b.z;
}

int root_tot, n, m, ans[N], anss[N];

void build(int u, int l, int r) {
    tr1[u].l = l, tr1[u].r = r;
    if (l != r) {
        int mid = (l + r) >> 1;
        build(u << 1, l, mid);
        build(u << 1 | 1, mid + 1, r);
    }
}

void modify_2(int& u, int l, int r, int pos) {
    if (u == 0) u = ++root_tot;
    tr2[u].cnt++;
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (pos <= mid) {
        modify_2(tr2[u].ch[0], l, mid, pos);
    } else {
        modify_2(tr2[u].ch[1], mid + 1, r, pos);
    }
}

int query_2(int& u, int l, int r, int x, int y) {
    if (u == 0) return 0;
    if (x <= l && r <= y) return tr2[u].cnt;
    int mid = (l + r) >> 1, ans = 0;
    if (x <= mid) ans += query_2(tr2[u].ch[0], l, mid, x, y);
    if (mid < y) ans += query_2(tr2[u].ch[1], mid + 1, r, x, y);
    return ans;
}

void modify_1(int u, int l, int r, int t) {
    modify_2(tr1[u].root, 1, m, data[t].z);
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (data[t].y <= mid) {
        modify_1(u << 1, l, mid, t);
    } else {
        modify_1(u << 1 | 1, mid + 1, r, t);
    }
}

int query_1(int u, int l, int r, int t) {
    if (1 <= l && r <= data[t].y) return query_2(tr1[u].root, 1, m, 1, data[t].z);
    int mid = (l + r) >> 1, ans = 0;
    if (1 <= mid) ans += query_1(u << 1, l, mid, t);
    if (mid < data[t].y) ans += query_1(u << 1 | 1, mid + 1, r, t);
    return ans;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    std::cin >> n >> m;
    rep(i, 1, n) {
        int x, y, z;
        std::cin >> x >> y >> z;
        data[i] = {x, y, z};
    }
    std::sort(data + 1, data + n + 1, cmp);
    build(1, 1, m);
```

```
88 |     rep(i, 1, n) {
89 |         modify_1(1, 1, m, i);
90 |         ans[i] = query_1(1, 1, m, i);
91 |     }
92 |     per(i, n - 1, 1) {
93 |         if (data[i] == data[i + 1]) ans[i] = ans[i + 1];
94 |     }
95 |     rep(i, 1, n) anss[ans[i]]++;
96 |     rep(i, 1, n) std::cout << anss[i] << endl;
97 |
98 |     return 0;
99 | }
```

2.13.2 线段树套平衡树

长度为 n 的序列和 m 此操作, 包含 5 种操作:

- l r k: 询问区间 $[l \sim r]$ 中数 k 的排名.

- l r k: 询问区间 $[l \sim r]$ 中排名为 k 的数.

- pos k: 将序列中 pos 位置的数修改为 k .

- l r k: 询问区间 $[l \sim r]$ 中数 k 的前驱.

- l r k: 询问区间 $[l \sim r]$ 中数 k 的后继.

Treap 实现

```
 1 | // 洛谷 P3380 【模板】二逼平衡树( 树套树 )
 2 |
 3 | int n, m, op, l, r, pos, key, root_tot;
 4 | int a[N];
 5 |
 6 | struct node2 {
 7 |     node2 *ch[2];
 8 |     int key, val;
 9 |     int cnt, size;
10 |
11 |     node2(int _key) : key(_key), cnt(1), size(1) {
12 |         ch[0] = ch[1] = nullptr;
13 |         val = rand();
14 |     }
15 |
16 |     // node2(node2 *_node2) {
17 |     // key = _node2->key, val = _node2->val, cnt = _node2->cnt, size = _node2->size;
18 |     // }
19 |
20 |     inline void push_up() {
21 |         size = cnt;
22 |         if (ch[0] != nullptr) size += ch[0]->size;
23 |         if (ch[1] != nullptr) size += ch[1]->size;
24 |     }
25 | };
26 |
27 | struct treap {
28 |     ...
29 | };
30 |
31 | treap tr2[N << 4];
32 |
33 | struct node1 {
34 |     int l, r, root;
35 | } tr1[N << 4];
36 |
37 | void build(int u, int l, int r) {
38 |     tr1[u] = {l, r, u};
39 |     root_tot = std::max(root_tot, u);
40 |     if (l == r) return;
41 |     int mid = (l + r) >> 1;
42 |     build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
43 | }
44 |
45 | void modify(int u, int pos, int key) {
46 |     tr2[u].insert(key);
47 |     if (tr1[u].l == tr1[u].r) return;
48 |     int mid = (tr1[u].l + tr1[u].r) >> 1;
49 |     if (pos <= mid){
50 |         modify(u << 1, pos, key);
```

```
 51          }
 52          else{
 53              modify(u << 1 | 1, pos, key);
 54          }
 55      }
 56
 57      int get_rank_by_key_in_interval(int u, int l, int r, int key) {
 58          if (l <= tr1[u].l && tr1[u].r <= r) return tr2[u].get_rank_by_key(tr2[u].root, key) - 2;
 59          int mid = (tr1[u].l + tr1[u].r) >> 1, ans = 0;
 60          if (l <= mid) ans += get_rank_by_key_in_interval(u << 1, l, r, key);
 61          if (mid < r) ans += get_rank_by_key_in_interval(u << 1 | 1, l, r, key);
 62          return ans;
 63      }
 64
 65      int get_key_by_rank_in_interval(int u, int l, int r, int rank) {
 66          int L = 0, R = 1e8;
 67          while (L < R) {
 68              int mid = (L + R + 1) / 2;
 69              if (get_rank_by_key_in_interval(1, l, r, mid) < rank){
 70                  L = mid;
 71              }
 72              else{
 73                  R = mid - 1;
 74              }
 75          }
 76          return L;
 77      }
 78
 79      void change(int u, int pos, int pre_key, int key) {
 80          tr2[u].remove(pre_key);
 81          tr2[u].insert(key);
 82          if (tr1[u].l == tr1[u].r) return;
 83          int mid = (tr1[u].l + tr1[u].r) >> 1;
 84          if (pos <= mid){
 85              change(u << 1, pos, pre_key, key);
 86          }
 87          else{
 88              change(u << 1 | 1, pos, pre_key, key);
 89          }
 90      }
 91
 92      int get_prev_in_interval(int u, int l, int r, int key) {
 93          if (l <= tr1[u].l && tr1[u].r <= r) return tr2[u].get_prev(key);
 94          int mid = (tr1[u].l + tr1[u].r) >> 1, ans = -inf;
 95          if (l <= mid) ans = std::max(ans, get_prev_in_interval(u << 1, l, r, key));
 96          if (mid < r) ans = std::max(ans, get_prev_in_interval(u << 1 | 1, l, r, key));
 97          return ans;
 98      }
 99
100      int get_nex_in_interval(int u, int l, int r, int key) {
101          if (l <= tr1[u].l && tr1[u].r <= r) return tr2[u].get_nex(key);
102          int mid = (tr1[u].l + tr1[u].r) >> 1, ans = inf;
103          if (l <= mid) ans = std::min(ans, get_nex_in_interval(u << 1, l, r, key));
104          if (mid < r) ans = std::min(ans, get_nex_in_interval(u << 1 | 1, l, r, key));
105          return ans;
106      }
107
108      int main() {
109          std::ios::sync_with_stdio(false);
110          std::cin.tie(0);
111          std::cout.tie(0);
112
113          srand(time(0));
114
115          std::cin >> n >> m;
116          build(1, 1, n);
117          rep(i, 1, n) {
118              std::cin >> a[i];
119              modify(1, i, a[i]);
120          }
121          rep(i, 1, root_tot) { tr2[i].insert(inf), tr2[i].insert(-inf); }
122          rep(i, 1, m) {
123              std::cin >> op;
124              if (op == 1) {
125                  std::cin >> l >> r >> key;
126                  std::cout << get_rank_by_key_in_interval(1, l, r, key) + 1 << endl;
127              } else if (op == 2) {
128                  std::cin >> l >> r >> key;
129                  std::cout << get_key_by_rank_in_interval(1, l, r, key) << endl;
130              } else if (op == 3) {
131                  std::cin >> pos >> key;
132                  change(1, pos, a[pos], key);
133                  a[pos] = key;
134              } else if (op == 4) {
135                  std::cin >> l >> r >> key;
136                  std::cout << get_prev_in_interval(1, l, r, key) << endl;
137              } else if (op == 5) {
```

```
138                 std::cin >> l >> r >> key;
139                 std::cout << get_nex_in_interval(1, l, r, key) << endl;
140
141         }
142
143         return 0;
144 }
```

然而洛谷上的会 T 两个点, Loj 和 ACwing 上的能过.

Splay 实现

```
1  // 洛谷 P3380 【模板】二逼平衡树( 树套树 )
2
3  int n, m, op, l, r, pos, key, root_tot;
4  int a[N];
5
6  struct node{
7      int ch[2], fa, key, siz, cnt;
8
9      void init(int _fa, int _key){
10         fa = _fa, key = _key, siz = cnt = 1;
11     }
12
13     void clear(){
14         ch[0] = ch[1] = fa = key = siz = cnt = 0;
15     }
16 }tr[N * 30];
17
18 struct splay{
19
20     int idx;
21
22     bool get(int u){
23         return u == tr[tr[u].fa].ch[1];
24     }
25
26     void pushup(int u){
27         tr[u].siz = tr[tr[u].ch[0]].siz + tr[tr[u].ch[1]].siz + tr[u].cnt;
28     }
29
30     void rotate(int x){
31         int y = tr[x].fa, z = tr[y].fa;
32         int op = get(x);
33         tr[y].ch[op] = tr[x].ch[op ^ 1];
34         if(tr[x].ch[op ^ 1]) tr[tr[x].ch[op ^ 1]].fa = y;
35         tr[x].ch[op ^ 1] = y;
36         tr[y].fa = x, tr[x].fa = z;
37         if(z) tr[z].ch[y == tr[z].ch[1]] = x;
38         pushup(y), pushup(x);
39     }
40
41     void opt(int& root, int u, int k){
42         for(int f = tr[u].fa; f = tr[u].fa, f != k; rotate(u)){
43             if(tr[f].fa != k) rotate(get(u) == get(f) ? f : u);
44         }
45         if(k == 0) root = u;
46     }
47
48     void insert(int& root, int key){
49         if(tr[root].siz == 0){
50             idx++;
51             tr[idx].init(0, key);
52             root = idx;
53             return;
54         }
55         int u = root, f = 0;
56         while(1){
57             if(tr[u].key == key){
58                 tr[u].cnt++;
59                 pushup(u), pushup(f);
60                 opt(root, u, 0);
61                 break;
62             }
63             f = u, u = tr[u].ch[tr[u].key < key];
64             if(!u){
65                 idx++;
66                 tr[idx].init(f, key);
67                 tr[f].ch[tr[f].key < key] = idx;
68                 pushup(idx), pushup(f);
69                 opt(root, idx, 0);
70                 break;
71             }
72         }
73     }
```

```
74
75    int kth(int& root, int rank){
76        int u = root;
77        while(1){
78            if(tr[u].ch[0] && rank <= tr[tr[u].ch[0]].siz) u = tr[u].ch[0];
79            else{
80                rank -= tr[tr[u].ch[0]].siz + tr[u].cnt;
81                if(rank <= 0){
82                    opt(root, u, 0);
83                    return u;
84                }
85                else u = tr[u].ch[1];
86            }
87        }
88    }
89
90    int nlt(int& root, int key){
91        int rank = 0, u = root;
92        while(1){
93            if(tr[u].key > key) u = tr[u].ch[0];
94            else{
95                rank += tr[tr[u].ch[0]].siz;
96                if(tr[u].key == key){
97                    opt(root, u, 0);
98                    return rank + 1;
99                }
100                rank += tr[u].cnt;
101                if(tr[u].ch[1]) u = tr[u].ch[1];
102                else return rank + 1;
103            }
104        }
105    }
106
107    int get_prev(int& root, int key){
108        return kth(root, nlt(root, key) - 1);
109    }
110
111    int get_next(int& root, int key){
112        return kth(root, nlt(root, key + 1));
113    }
114
115    void remove(int& root, int key){
116        nlt(root, key);
117        if(tr[root].cnt > 1){
118            tr[root].cnt--;
119            pushup(root);
120            return;
121        }
122        int u = root, l = get_prev(root, key);
123        tr[tr[u].ch[1]].fa = l;
124        tr[l].ch[1] = tr[u].ch[1];
125        tr[u].clear();
126        pushup(root);
127    }
128
129    void output(int u){
130        if(tr[u].ch[0]) output(tr[u].ch[0]);
131        std::cout << tr[u].key << ' ';
132        if(tr[u].ch[1]) output(tr[u].ch[1]);
133    }
134
135 }splay;
136
137 struct node1{
138    int l, r, root;
139 }tr1[N * 4];
140
141 void build(int u, int l, int r){
142    tr1[u] = {l, r, u};
143    root_tot = splay.idx = std::max(root_tot, u);
144    if(l == r) return;
145    int mid = (l + r) >> 1;
146    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
147 }
148
149 void modify(int u, int pos, int key){
150    splay.insert(tr1[u].root, key);
151    if(tr1[u].l == tr1[u].r) return;
152    int mid = (tr1[u].l + tr1[u].r) >> 1;
153    if(pos <= mid) modify(u << 1, pos, key);
154    else modify(u << 1 | 1, pos, key);
155 }
156
157 int get_rank_by_key_in_interval(int u, int l, int r, int key){
158    if(l <= tr1[u].l && tr1[u].r <= r)
159        return splay.nlt(tr1[u].root, key) - 2;
160    int mid = (tr1[u].l + tr1[u].r) >> 1, ans = 0;
```

```cpp
161         if(l <= mid) ans += get_rank_by_key_in_interval(u << 1, l, r, key);
162         if(mid < r) ans += get_rank_by_key_in_interval(u << 1 | 1, l, r, key);
163         return ans;
164 }
165
166 int get_key_by_rank_in_interval(int u, int l, int r, int rank){
167     int L = 0, R = 1e8;
168     while(L < R){
169         int mid = (L + R + 1) / 2;
170         if(get_rank_by_key_in_interval(1, l, r, mid) < rank) L = mid;
171         else R = mid - 1;
172     }
173     return L;
174 }
175
176 void change(int u, int pos, int pre_key, int key){
177     splay.remove(tr1[u].root, pre_key);
178     splay.insert(tr1[u].root, key);
179     if(tr1[u].l == tr1[u].r) return;
180     int mid = (tr1[u].l + tr1[u].r) >> 1;
181     if(pos <= mid) change(u << 1, pos, pre_key, key);
182     else change(u << 1 | 1, pos, pre_key, key);
183 }
184
185 int get_prev_in_interval(int u, int l, int r, int key){
186     if(l <= tr1[u].l && tr1[u].r <= r)
187         return tr[splay.get_prev(tr1[u].root, key)].key;
188     int mid = (tr1[u].l + tr1[u].r) >> 1, ans = -inf;
189     if(l <= mid) ans = std::max(ans, get_prev_in_interval(u << 1, l, r, key));
190     if(mid < r) ans = std::max(ans, get_prev_in_interval(u << 1 | 1, l, r, key));
191     return ans;
192
193 }
194
195 int get_next_in_interval(int u, int l, int r, int key){
196     if(l <= tr1[u].l && tr1[u].r <= r)
197         return tr[splay.get_next(tr1[u].root, key)].key;
198     int mid = (tr1[u].l + tr1[u].r) >> 1, ans = inf;
199     if(l <= mid) ans = std::min(ans, get_next_in_interval(u << 1, l, r, key));
200     if(mid < r) ans = std::min(ans, get_next_in_interval(u << 1 | 1, l, r, key));
201     return ans;
202 }
203
204 int main(){
205
206     std::ios::sync_with_stdio(false);
207     std::cin.tie(0);
208     std::cout.tie(0);
209
210     srand(time(0));
211
212     std::cin >> n >> m;
213     build(1, 1, n);
214     rep(i, 1, n){
215         std::cin >> a[i];
216         modify(1, i, a[i]);
217     }
218     rep(i, 1, root_tot){
219         splay.insert(tr1[i].root, inf), splay.insert(tr1[i].root, -inf);
220     }
221     rep(i, 1, m){
222         std::cin >> op;
223         if(op == 1){
224             std::cin >> l >> r >> key;
225             std::cout << get_rank_by_key_in_interval(1, l, r, key) + 1 << endl;
226         }
227         else if(op == 2){
228             std::cin >> l >> r >> key;
229             std::cout << get_key_by_rank_in_interval(1, l, r, key) << endl;
230         }
231         else if(op == 3){
232             std::cin >> pos >> key;
233             change(1, pos, a[pos], key);
234             a[pos] = key;
235         }
236         else if(op == 4){
237             std::cin >> l >> r >> key;
238             std::cout << get_prev_in_interval(1, l, r, key) << endl;
239         }
240         else if(op == 5){
241             std::cin >> l >> r >> key;
242             std::cout << get_next_in_interval(1, l, r, key) << endl;
243         }
244     }
245
246     return 0;
247 }
```

然而洛谷吸氧能过, ACwing 能过, Loj T 一堆.

然而洛谷吸氧能过, ACwing 能过, Loj T 一堆.

3 字符串

3.1 字典树

3.1.1 普通字典树 (单词匹配)

```cpp
// trie //
int cnt;
std::vector<std::array<int, 26>> trie(n + 1);
vi exist(n + 1);

auto insert = [&](string s) -> void {
    int p = 0;
    for (int i = 0; i < s.size() - 1; i++) {
        int c = s[i] - 'a';
        if (!trie[p][c]) trie[p][c] = ++cnt;
        p = trie[p][c];
    }
    exist[p] = true;
};

auto find = [&](string s) -> bool {
    int p = 0;
    for (int i = 0; i < s.size() - 1; i++) {
        int c = s[i] - 'a';
        if (!trie[p][c]) return false;
        p = trie[p][c];
    }
    return exist[p];
};
```

3.1.2 01 字典树 (求最大异或值)

给定 n 个数, 取两个数进行异或运算, 求最大异或值.

```cpp
// trie //
int cnt = 0;
std::vector<std::array<int, 2>> trie(N);

auto insert = [&](int x) -> void {
    int p = 0;
    for (int i = 30; i >= 0; i--) {
        int c = (x >> i) & 1;
        if (!trie[p][c]) trie[p][c] = ++cnt;
        p = trie[p][c];
    }
};

auto find = [&](int x) -> int {
    int sum = 0, p = 0;
    for (int i = 30; i >= 0; i--) {
        int c = (x >> i) & 1;
        if (trie[p][c ^ 1]) {
            p = trie[p][c ^ 1];
            sum += (1 << i);
        } else {
            p = trie[p][c];
        }
    }
    return sum;
};
```

3.1.3 字典树合并

来自浙大城市学院 2023 校赛 E 题.

给定一棵根为 1 的树, 每个点的点权为 w_i. 一共 q 次询问, 每次给出一对 u, v, 询问以 v 为根的子树上的点与 u 的权值最大异或值.

```cpp
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    int n, m;
    std::cin >> n;
    vi w(n + 1);
    for (int i = 1; i <= n; i++) {
        std::cin >> w[i];
    }

    vvi e(n + 1);
    for (int i = 1; i < n; i++) {
        int u, v;
        std::cin >> u >> v;
        e[u].push_back(v);
        e[v].push_back(u);
    }

    // 离线询问 //
    std::cin >> m;
    std::vector<vpi> q(n + 1);
    vi ans(m + 1);
    for (int i = 1; i <= m; i++) {
        int u, v;
        std::cin >> u >> v;
        q[v].emplace_back(u, i);
    }

    // 01 trie //
    std::vector<std::array<int, 2>> tr(1);

    auto new_node = [&]() -> int {
        tr.emplace_back();
        return tr.size() - 1;
    };

    vi id(n + 1);

    auto insert = [&](int root, int x) {
        int p = root;
        for (int i = 29; i >= 0; i--) {
            int c = x >> i & 1;
            if (!tr[p][c]) tr[p][c] = new_node();
            p = tr[p][c];
        }
    };

    auto query = [&](int root, int x) -> int {
        int ans = 0, p = root;
        for (int i = 29; i >= 0; i--) {
            int c = x >> i & 1;
            if (tr[p][c ^ 1]) {
                p = tr[p][c ^ 1];
                ans += (1 << i);
            } else {
                p = tr[p][c];
            }
        }
        return ans;
    };

    std::function<int(int, int)> merge = [&](int a, int b) -> int {
        // b 的信息挪到 a 上 //
        if (!a) return b;
        if (!b) return a;
        tr[a][0] = merge(tr[a][0], tr[b][0]);
        tr[a][1] = merge(tr[a][1], tr[b][1]);
        return a;
    };

    std::function<void(int, int)> dfs = [&](int u, int fa) {
        id[u] = new_node();
        insert(id[u], w[u]);
        for (auto v : e[u]) {
            if (v == fa) continue;
            dfs(v, u);
            id[u] = merge(id[u], id[v]);
        }
        for (auto [v, i] : q[u]) {
            ans[i] = query(id[u], w[v]);
        }
    };
    dfs(1, 0);

    for (int i = 1; i <= m; i++) std::cout << ans[i] << endl;
```

```
88
89        return 0;
90  }
```

3.2   KMP

这一节的 *string* 都是从 0 开始计数.

3.2.1   计算 *next* 数组

```
1   auto get_next = [&](string s) -> vi {
2       int n = s.length();
3       vi next(n);
4       for (int i = 1; i < n; i++) {
5           int j = next[i - 1];
6           while (j > 0 and s[i] != s[j]) j = next[j - 1];
7           if (s[i] == s[j]) j++;
8           next[i] = j;
9       }
10      return next;
11  };
```

3.2.2   在文本串中匹配模式串

求出 s 在 t 中所有出现的位置.

用脏字符连接文本串与模式串跑 KMP 即可.

3.2.3   字符串的最小周期

如果周期大于 1, $n - next[n - 1]$ 是最小周期.

如果周期为 1, 满足条件:

1. $next[n - 1] = n$;

2. $next[n - 1] \neq n$, 但计算出来的并不是循环节, 暴力判断一下.

4   数学 - 多项式

4.1   FFT

```
const int sz = 1 << 23;
int rev[sz];
int rev_n;
void set_rev(int n) {
    if (n == rev_n) return;
    for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) * n) / 2;
    rev_n = n;
}
tempt void butterfly(T* a, int n) {
    set_rev(n);
    for (int i = 0; i < n; i++) {
        if (i < rev[i]) std::swap(a[i], a[rev[i]]);
    }
}

namespace Comp {

long double pi = 3.141592653589793238;

tempt struct complex {
    T x, y;
    complex(T x = 0, T y = 0) : x(x), y(y) {}
    complex operator+(const complex& b) const { return complex(x + b.x, y + b.y); }

    complex operator-(const complex& b) const { return complex(x - b.x, y - b.y); }

    complex operator*(const complex& b) const {
        return complex<T>(x * b.x - y * b.y, x * b.y + y * b.x);
    }
    complex operator~() const { return complex(x, -y); }
    static complex unit(long double rad) { return complex(std::cos(rad), std::sin(rad)); }
};

}    // namespace Comp

struct fft_t {
    typedef Comp::complex<double> complex;
    complex wn[sz];

    fft_t() {
        for (int i = 0; i < sz / 2; i++) {
            wn[sz / 2 + i] = complex::unit(2 * Comp::pi * i / sz);
        }
        for (int i = sz / 2 - 1; i; i--) wn[i] = wn[i * 2];
    }

    void operator()(complex* a, int n, int type) {
        if (type == -1) std::reverse(a + 1, a + n);
        butterfly(a, n);
        for (int i = 1; i < n; i *= 2) {
            const complex* w = wn + i;
            for (complex *b = a, t; b != a + n; b += i + 1) {
                t = b[i];
                b[i] = *b - t;
                *b = *b + t;
                for (int j = 1; j < i; j++) {
                    t = (++b)[i] * w[j];
                    b[i] = *b - t;
                    *b = *b + t;
                }
            }
        }
        if (type == 1) return;
        for (int i = 0; i < n * 2; i++) ((double*) a)[i] /= n;
    }
} FFT;

typedef decltype(FFT)::complex complex;
```

4.1.1   FFT

```
vi fft(const vi& f, const vi& g) {
    static complex ff[sz];
    int n = f.size(), m = g.size();
```

```
 4        vi h(n + m - 1);
 5        if (std::min(n, m) <= 50) {
 6            for (int i = 0; i < n; i++) {
 7                for (int j = 0; j < m; ++j) {
 8                    h[i + j] += f[i] * g[j];
 9                }
10            }
11            return h;
12        }
13        int c = 1;
14        while (c + 1 < n + m) c *= 2;
15        std::memset(ff, 0, sizeof(decltype(*(ff))) * (c));
16        for (int i = 0; i < n; i++) ff[i].x = f[i];
17        for (int i = 0; i < m; i++) ff[i].y = g[i];
18        FFT(ff, c, 1);
19        for (int i = 0; i < c; i++) ff[i] = ff[i] * ff[i];
20        FFT(ff, c, -1);
21        for (int i = 0; i + 1 < n + m; i++) h[i] = std::llround(ff[i].y / 2);
22        return h;
23    }
```

4.1.2 拆系数 FFT

注意改头文件模板的 mod 数.

```
 1  vi mtt(const vi& f, const vi& g) {
 2      static complex ff[3][sz], gg[2][sz];
 3      static int s[3] = {1, 31623, 31623 * 31623};
 4      int n = f.size(), m = g.size();
 5      vi h(n + m - 1);
 6      if (std::min(n, m) <= 50) {
 7          for (int i = 0; i < n; ++i) {
 8              for (int j = 0; j < m; ++j) {
 9                  Add(h[i + j], mul(f[i], g[j]));
10              }
11          }
12          return h;
13      }
14      int c = 1;
15      while (c + 1 < n + m) c *= 2;
16      for (int i = 0; i < 2; ++i) {
17          std::memset(ff[i], 0, sizeof(decltype(*(ff[i]))) * (c));
18          std::memset(gg[i], 0, sizeof(decltype(*(ff[i]))) * (c));
19          for (int j = 0; j < n; ++j) ff[i][j].x = f[j] / s[i] % s[1];
20          for (int j = 0; j < m; ++j) gg[i][j].x = g[j] / s[i] % s[1];
21          FFT(ff[i], c, 1);
22          FFT(gg[i], c, 1);
23      }
24      for (int i = 0; i < c; ++i) {
25          ff[2][i] = ff[1][i] * gg[1][i];
26          ff[1][i] = ff[1][i] * gg[0][i];
27          gg[1][i] = ff[0][i] * gg[1][i];
28          ff[0][i] = ff[0][i] * gg[0][i];
29      }
30      for (int i = 0; i < 3; ++i) {
31          FFT(ff[i], c, -1);
32          for (int j = 0; j + 1 < n + m; ++j) {
33              Add(h[j], mul(std::llround(ff[i][j].x) % mod, s[i]));
34          }
35      }
36      FFT(gg[1], c, -1);
37      for (int i = 0; i + 1 < n + m; ++i) {
38          Add(h[i], mul(std::llround(gg[1][i].x) % mod, s[1]));
39      }
40      return h;
41  }
```

4.2   NTT 全家桶

```
 1  class polynomial : public vi {
 2      public:
 3      polynomial() = default;
 4      polynomial(const vi& v) : vi(v) {}
 5      polynomial(vi&& v) : vi(std::move(v)) {}
 6
 7      int degree() { return size() - 1; }
 8
 9      void clearzero() {
```

```
10            while (size() && !back()) pop_back();
11        }
12 };
13
14
15 polynomial& operator+=(polynomial& a, const polynomial& b) {
16     a.resize(std::max(a.size(), b.size()), 0);
17     for (int i = 0; i < b.size(); i++) {
18         Add(a[i], b[i]);
19     }
20     a.clearzero();
21     return a;
22 }
23
24 polynomial operator+(const polynomial& a, const polynomial& b) {
25     polynomial ans = a;
26     return ans += b;
27 }
28
29 polynomial& operator-=(polynomial& a, const polynomial& b) {
30     a.resize(std::max(a.size(), b.size()), 0);
31     for (int i = 0; i < b.size(); i++) {
32         Sub(a[i], b[i]);
33     }
34     a.clearzero();
35     return a;
36 }
37
38 polynomial operator-(const polynomial& a, const polynomial& b) {
39     polynomial ans = a;
40     return ans -= b;
41 }
42
43 class ntt_t {
44     public:
45     static const int maxbit = 22;
46     static const int sz = 1 << maxbit;
47     static const int mod = 998244353;
48     static const int g = 3;
49
50     std::array<int, sz + 10> w;
51     std::array<int, maxbit + 10> len_inv;
52
53     ntt_t() {
54         int wn = pow(g, (mod - 1) >> maxbit);
55         w[0] = 1;
56         for (int i = 1; i <= sz; i++) {
57             w[i] = mul(w[i - 1], wn);
58         }
59         len_inv[maxbit] = pow(sz, mod - 2);
60         for (int i = maxbit - 1; ~i; i--) {
61             len_inv[i] = add(len_inv[i + 1], len_inv[i + 1]);
62         }
63     }
64
65     void operator()(vi& v, int& n, int type) {
66         int bit = 0;
67         while ((1 << bit) < n) bit++;
68         int tot = (1 << bit);
69         v.resize(tot, 0);
70         vi rev(tot);
71         n = tot;
72         for (int i = 0; i < tot; i++) {
73             rev[i] = rev[i >> 1] >> 1;
74             if (i & 1) {
75                 rev[i] |= tot >> 1;
76             }
77         }
78         for (int i = 0; i < tot; i++) {
79             if (i < rev[i]) {
80                 std::swap(v[i], v[rev[i]]);
81             }
82         }
83         for (int midd = 0; (1 << midd) < tot; midd++) {
84             int mid = 1 << midd;
85             int len = mid << 1;
86             for (int i = 0; i < tot; i += len) {
87                 for (int j = 0; j < mid; j++) {
88                     int w0 = v[i + j];
89                     int w1 = mul(
90                         w[type == 1 ? (j << maxbit - midd - 1) : (len - j << maxbit - midd - 1)],
91                         v[i + j + mid]);
92                     v[i + j] = add(w0, w1);
93                     v[i + j + mid] = sub(w0, w1);
94                 }
95             }
96         }
```

```
 97 |         if (type == -1) {
 98 |             for (int i = 0; i < tot; i++) {
 99 |                 v[i] = mul(v[i], len_inv[bit]);
100 |             }
101 |         }
102 |     }
103 | } NTT;
```

4.2.1 乘法

```
 1 | polynomial& operator*=(polynomial& a, const polynomial& b) {
 2 |     if (!a.size() || !b.size()) {
 3 |         a.resize(0);
 4 |         return a;
 5 |     }
 6 |     polynomial tmp = b;
 7 |     int deg = a.size() + b.size() - 1;
 8 |     int temp = deg;
 9 |
10 |     // 项数较小直接硬算
11 |
12 |     if ((LL) a.size() * (LL) b.size() <= (LL) deg * 50LL) {
13 |         tmp.resize(0);
14 |         tmp.resize(deg, 0);
15 |         for (int i = 0; i < a.size(); i++) {
16 |             for (int j = 0; j < b.size(); j++) {
17 |                 tmp[i + j] = add(tmp[i + j], mul(a[i], b[j]));
18 |             }
19 |         }
20 |         a = tmp;
21 |         return a;
22 |     }
23 |
24 |     // 项数较多跑 NTT
25 |
26 |     NTT(a, deg, 1);
27 |     NTT(tmp, deg, 1);
28 |     for (int i = 0; i < deg; i++) {
29 |         Mul(a[i], tmp[i]);
30 |     }
31 |     NTT(a, deg, -1);
32 |     a.resize(temp);
33 |     return a;
34 | }
35 |
36 | polynomial operator*(const polynomial& a, const polynomial& b) {
37 |     polynomial ans = a;
38 |     return ans *= b;
39 | }
```

4.2.2 逆

```
 1 | polynomial inverse(const polynomial& a) {
 2 |     polynomial ans({pow(a[0], mod - 2)});
 3 |     polynomial temp;
 4 |     polynomial tempa;
 5 |     int deg = a.size();
 6 |     for (int i = 0; (1 << i) < deg; i++) {
 7 |         tempa.resize(0);
 8 |         tempa.resize(1 << i << 1, 0);
 9 |         for (int j = 0; j != tempa.size() and j != deg; j++) {
10 |             tempa[j] = a[j];
11 |         }
12 |         temp = ans * (polynomial({2}) - tempa * ans);
13 |         if (temp.size() > (1 << i << 1)) {
14 |             temp.resize(1 << i << 1, 0);
15 |         }
16 |         temp.clearzero();
17 |         std::swap(temp, ans);
18 |     }
19 |     ans.resize(deg);
20 |     return ans;
21 | }
```

4.2.3 log

```
1  polynomial diffrential(const polynomial& a) {
2      if (!a.size()) {
3          return a;
4      }
5      polynomial ans(vi(a.size() - 1));
6      for (int i = 1; i < a.size(); i++) {
7          ans[i - 1] = mul(a[i], i);
8      }
9      return ans;
10 }
11
12 polynomial integral(const polynomial& a) {
13     polynomial ans(vi(a.size() + 1));
14     for (int i = 0; i < a.size(); i++) {
15         ans[i + 1] = mul(a[i], pow(i + 1, mod - 2));
16     }
17     return ans;
18 }
19
20 polynomial ln(const polynomial& a) {
21     int deg = a.size();
22     polynomial da = diffrential(a);
23     polynomial inva = inverse(a);
24     polynomial ans = integral(da * inva);
25     ans.resize(deg);
26     return ans;
27 }
```

4.2.4 exp

```
1  polynomial exp(const polynomial& a) {
2      polynomial ans({1});
3      polynomial temp;
4      polynomial tempa;
5      polynomial tempaa;
6      int deg = a.size();
7      for (int i = 0; (1 << i) < deg; i++) {
8          tempa.resize(0);
9          tempa.resize(1 << i << 1, 0);
10         for (int j = 0; j != tempa.size() and j != deg; j++) {
11             tempa[j] = a[j];
12         }
13         tempaa = ans;
14         tempaa.resize(1 << i << 1);
15         temp = ans * (tempa + polynomial({1}) - ln(tempaa));
16         if (temp.size() > (1 << i << 1)) {
17             temp.resize(1 << i << 1, 0);
18         }
19         temp.clearzero();
20         std::swap(temp, ans);
21     }
22     ans.resize(deg);
23     return ans;
24 }
```

4.2.5 sqrt

```
1  polynomial sqrt(polynomial& a) {
2      polynomial ans({cipolla(a[0])});
3      if (ans[0] == -1) return ans;
4      polynomial temp;
5      polynomial tempa;
6      polynomial tempaa;
7      int deg = a.size();
8      for (int i = 0; (1 << i) < deg; i++) {
9          tempa.resize(0);
10         tempa.resize(1 << i << 1, 0);
11         for (int j = 0; j != tempa.size() and j != deg; j++) {
12             tempa[j] = a[j];
13         }
14         tempaa = ans;
15         tempaa.resize(1 << i << 1);
16         temp = (tempa * inverse(tempaa) + ans) * inv2;
17         if (temp.size() > (1 << i << 1)) {
18             temp.resize(1 << i << 1, 0);
```

```
19          }
20          temp.clearzero();
21          std::swap(temp, ans);
22      }
23      ans.resize(deg);
24      return ans;
25  }
26
27  // 特判 //
28
29  int cnt = 0;
30  for (int i = 0; i < a.size(); i++) {
31      if (a[i] == 0) {
32          cnt++;
33      } else {
34          break;
35      }
36  }
37  if (cnt) {
38      if (cnt == n) {
39          for (int i = 0; i < n; i++) {
40              std::cout << "0 ";
41          }
42          std::cout << endl;
43          return 0;
44      }
45      if (cnt & 1) {
46          std::cout << "-1" << endl;
47          return 0;
48      }
49      polynomial b(vi(a.size() - cnt));
50      for (int i = cnt; i < a.size(); i++) {
51          b[i - cnt] = a[i];
52      }
53      a = b;
54  }
55  a.resize(n - cnt / 2);
56  a = sqrt(a);
57  if (a[0] == -1) {
58      std::cout << "-1" << endl;
59      return 0;
60  }
61  reverse(all(a));
62  a.resize(n);
63  reverse(all(a));
```

4.3   FWT

4.3.1   与

$$C_i = \sum_{i=j\&k} A_j B_k$$

分治过程

$$\text{FWT}[A] = merge(\text{FWT}[A_0] + \text{FWT}[A_1], \text{FWT}[A_1]),$$

$$\text{UFWT}[A'] = merge(\text{UFWT}[A'_0] - \text{UFWT}[A'_1], \text{UFWT}[A'_1]).$$

```
1  // mod 998244353 //
2  auto FWT_and = [&](vi v, int type) -> vi {
3      int n = v.size();
4      for (int mid = 1; mid < n; mid <<= 1) {
5          for (int block = mid << 1, j = 0; j < n; j += block) {
6              for (int i = j; i < j + mid; i++) {
7                  LL x = v[i], y = v[i + mid];
8                  if (type == 1) {
9                      v[i] = add(x, y);
10                 } else {
11                     v[i] = sub(x, y);
12                 }
13             }
14         }
15     }
16     return v;
17 };
```

4.3.2 或

$$C_i = \sum_{i=j|k} A_j B_k$$

分治过程

$$\text{FWT[A]} = merge(\text{FWT[A}_0], \text{FWT[A}_0] + \text{FWT[A}_1]),$$

$$\text{UFWT[A']} = merge(\text{UFWT[A'}_0], -\text{UFWT[A'}_0] + \text{UFWT[A'}_1]).$$

```cpp
// mod 998244353 //
auto FWT_or = [&](vi v, int type) -> vi {
    int n = v.size();
    for (int mid = 1; mid < n; mid <<= 1) {
        for (int block = mid << 1, j = 0; j < n; j += block) {
            for (int i = j; i < j + mid; i++) {
                LL x = v[i], y = v[i + mid];
                if (type == 1) {
                    v[i + mid] = add(x, y);
                } else {
                    v[i + mid] = sub(y, x);
                }
            }
        }
    }
    return v;
};
```

4.3.3 异或

$$C_i = \sum_{i=j\,\text{xor}\,k} A_j B_k$$

分治过程

$$\text{FWT[A]} = merge(\text{FWT[A}_0] + \text{FWT[A}_1], \text{FWT[A}_0] - \text{FWT[A}_1]),$$

$$\text{UFWT[A']} = merge(\frac{\text{UFWT[A'}_0] + \text{UFWT[A'}_1]}{2}, \frac{\text{UFWT[A'}_0] - \text{UFWT[A'}_1]}{2}).$$

```cpp
// mod 998244353 //
auto FWT_xor = [&](vi v, int type) -> vi {
    int n = v.size();
    for (int mid = 1; mid < n; mid <<= 1) {
        for (int block = mid << 1, j = 0; j < n; j += block) {
            for (int i = j; i < j + mid; i++) {
                LL x = v[i], y = v[i + mid];
                v[i] = add(x, y);
                v[i + mid] = sub(x, y);
                if (type == -1) {
                    Mul(v[i], inv2);
                    Mul(v[i + mid], inv2);
                }
            }
        }
    }
    return v;
};
```

统一地,

```cpp
a = FWT(a, 1),  b = FWT(b, 1);
for (int i = 0; i < (1 << n); i++) {
    c[i] = mul(a[i], b[i]);
}
c = FWT(c, -1);
```

4.4 拉格朗日插值

4.4.1 一般的插值

给出一个多项式 $f(x)$ 上的 n 个点 (x_i, y_i), 求 $f(k)$.

拉格朗日插值的结果是

$$f(x) = \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j},$$

直接带入 k 并且取模即可, 时间复杂度 $O(n^2)$.

```
int ans = 0;
for (int i = 1; i <= n; i++) {
    LL s1 = y[i] % mod, s2 = 1LL;
    for (int j = 1; j <= n; j++) {
        if (i != j) {
            Mul(s1, (k - x[j]));
            Mul(s2, (x[i] - x[j]));
        }
    }
    Add(ans, mul(s1, pow(s2, mod - 2)));
}
```

4.4.2 坐标连续的插值

给出的点是 (i, y_i).

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \\
&= \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{x - j}{i - j} \\
&= \sum_{i=1}^{n} y_i \cdot \frac{\prod\limits_{j=1}^{n}(x - j)}{(x - i)(-1)^{n+1-i}(i - 1)!(n + 1 - i)!} \\
&= \left(\prod_{j=1}^{n}(x - j) \right) \left(\sum_{i=1}^{n} \frac{(-1)^{n+1-i} y_i}{(x - i)(i - 1)!(n + 1 - i)!} \right),
\end{aligned}
$$

时间复杂度为 $O(n)$.

5 数学 - 数论

5.1 欧几里得算法

5.1.1 欧几里得算法

5.1.2 扩展欧几里得算法

```cpp
std::function<void(LL, LL, LL&, LL&)> exgcd = [&](LL a, LL b, LL& x, LL& y) -> void {
    LL x1 = 1, x2 = 0, x3 = 0, x4 = 1;
    while (b != 0) {
        LL c = a / b;
        std::tie(x1, x2, x3, x4, a, b) =
            std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
    }
    x = x1, y = x2;
};
```

5.1.3 类欧几里得算法

一般形式: 求 $f(a,b,c,n) = \sum\limits_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$.

$f(a,b,c,n)$ 可以单独求.

$$f(a,b,c,n) = nm - f(c, c - b - 1, a, m - 1)$$

```cpp
LL f(LL a, LL b, LL c, LL n) {
    if (a == 0) return ((b / c) * (n + 1));
    if (a >= c || b >= c)
        return f(a % c, b % c, c, n) + (a / c) * n * (n + 1) / 2 + (b / c) * (n + 1);
    LL m = (a * n + b) / c;
    LL v = f(c, c - b - 1, a, m - 1);
    return n * m - v;
}
```

更进一步, 求: $g(a,b,c,n) = \sum\limits_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor$ 以及 $h(a,b,c,n) = \sum\limits_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$

直接记吧.

$$g(a,b,c,n) = \lfloor \frac{mn(n+1) - f(c,c-b-1,a,m-1) - h(c,c-b-1,a,m-1)}{2} \rfloor$$

$$h(a,b,c,n) = nm(m+1) - 2f(c,c-b-1,a,m-1) - 2g(c,c-b-1,a,m-1) - f(a,b,c,n)$$

```cpp
const int inv2 = 499122177, inv6 = 166374059;    // 2和6的逆元 //

LL f(LL a, LL b, LL c, LL n);
LL g(LL a, LL b, LL c, LL n);
LL h(LL a, LL b, LL c, LL n);

struct data {
    LL f, g, h;
};

data calc(LL a, LL b, LL c, LL n) {
    LL ac = a / c, bc = b / c, m = (a * n + b) / c, n1 = n + 1, n21 = n * 2 + 1;
    data d;
    if (a == 0) {
        d.f = bc * n1 % mod;
        d.g = bc * n % mod * n1 % mod * inv2 % mod;
        d.h = bc * bc % mod * n1 % mod;
        return d;
    }
    if (a >= c || b >= c) {
        d.f = n * n1 % mod * inv2 % mod * ac % mod + bc * n1 % mod;
        d.g =
            ac * n % mod * n1 % mod * n21 % mod * inv6 % mod + bc * n % mod * n1 % mod * inv2 % mod;
        d.h = ac * ac % mod * n % mod * n1 % mod * n21 % mod * inv6 % mod +
            bc * bc % mod * n1 % mod + ac * bc % mod * n % mod * n1 % mod;
        d.f %= mod, d.g %= mod, d.h %= mod;
        data e = calc(a % c, b % c, c, n);
```

```
28          d.h += e.h + 2 * bc % mod * e.f % mod + 2 * ac % mod * e.g % mod;
29          d.g += e.g, d.f += e.f;
30          d.f %= mod, d.g %= mod, d.h %= mod;
31          return d;
32      }
33      data e = calc(c, c - b - 1, a, m - 1);
34      d.f = n * m % mod - e.f, d.f = (d.f % mod + mod) % mod;
35      d.g = m * n % mod * n1 % mod - e.h - e.f, d.g = (d.g * inv2 % mod + mod) % mod;
36      d.h = n * m % mod * (m + 1) % mod - 2 * e.g - 2 * e.f - d.f;
37      d.h = (d.h % mod + mod) % mod;
38      return d;
39  }
```

5.2   快速幂

```
1   auto quick_power = [&](LL a, LL n, LL mod) -> LL {
2       LL ans = 1;
3       while (n != 0) {
4           if (n & 1) ans = ans * a % mod;
5           a = a * a % mod;
6           n >>= 1;
7       }
8       return ans;
9   };
```

5.3   逆元

5.3.1   费马小定理

p 为素数, 有 $a^{-1} \equiv a^{p-2} \bmod p$.

5.3.2   扩展欧几里得

```
1   auto inv = [&](LL a, LL p) -> LL {
2       if (std::gcd(a, p) != 1) return -1;
3       LL x, y;
4       exgcd(a, p, x, y);
5       return (x % p + p) % p;
6   }
```

5.3.3   线性递推

$a^{-1} \equiv -\lfloor \frac{p}{a} \rfloor \times (p\%a)^{-1}$.

```
1   vi inv(n + 1);
2   auto sieve_inv = [&](int n) -> void {
3       inv[1] = 1;
4       for (int i = 2; i <= n; i++) {
5           inv[i] = mul(sub(p, p / i), inv[p % i]);
6       }
7   }
```

5.4   欧拉函数

设 $n = \prod_{i=1}^{s} p_i{}^{k_i}$, 则 $\varphi(n) = n \cdot \prod_{i=1}^{s}(1 - \frac{1}{p_i})$.

5.4.1   某个数的欧拉函数值

```
1   auto phi = [&](int n) -> int {
2       int ans = n;
```

```
 3      for (int i = 2; i * i <= n; i++) {
 4          if (n % i != 0) continue;
 5          ans = ans / i * (i - 1);
 6          while (n % i == 0) n /= i;
 7      }
 8      if (n > 1) ans = ans / n * (n - 1);
 9      return ans;
10 };
```

5.4.2 欧拉定理

若 $\gcd(a, p) = 1$, 则 $a^{\varphi(p)} \equiv 1 \pmod{p}$.

5.4.3 扩展欧拉定理

若 $\gcd(a, p) \neq 1$, 则 $a^b = \begin{cases} a^b & b \leqslant \varphi(p) \\ a^{b \% \varphi(p) + \varphi(p)} \bmod p & b > \varphi(p) \end{cases}$.

5.5 中国剩余定理

求解
$$\begin{cases} N \equiv a_1 \mod m_1 \\ N \equiv a_2 \mod m_2 \\ \cdots \\ N \equiv a_n \mod m_n \end{cases}$$

有 $N \equiv \sum\limits_{i=1}^{k} a_i \times \mathrm{inv}\left(\frac{M}{m_i}, m_i\right) \times \left(\frac{M}{m_i}\right) \bmod M$

前提: 模数为两两不同的素数

```
1 auto crt = [&](int n, const vi& a, const vi& m) -> int{
2     LL ans = 0, M = 1;
3     for(int i = 1; i <= n; i++) M *= m[i];
4     for(int i = 1; i <= n; i++){
5         ans = (ans + a[i] * inv(M / m[i], m[i]) * (M / m[i])) % M;
6     }
7     return (ans % M + M) % M;
8 };
```

5.5.1 扩展中国剩余定理

```
 1 auto excrt = [&](int n, const vi& a, const vi& m) -> LL{
 2     LL A = a[1], M = m[1];
 3     for (int i = 2; i <= n; i++) {
 4         LL x, y, d = std::gcd(M, m[i]);
 5         exgcd(M, m[i], x, y);
 6         LL mod = M / d * m[i];
 7         x = x * (a[i] - A) / d % (m[i] / d);
 8         A = ((M * x + A) % mod + mod) % mod;
 9         M = mod;
10     }
11     return A;
12 };
```

5.6　数论分块

5.6.1　分块的逻辑

下取整 $\lfloor \frac{n}{g} \rfloor = k$ 的分块 ($g \leqslant n$).

```cpp
for(int l = 1, r, k; l <= n; l = r + 1){
    k = n / l;
    r = n / (n / l);
    debug(l, r, k);
}
```

$k = \lfloor \frac{n}{g} \rfloor$ 从大到小遍历 $\lfloor \frac{n}{g} \rfloor$ 的所有取值, $[l, r]$ 对应的是 g 取值的区间.

下面是 debug 结果.

```cpp
// n = 11
[l, r, k] : 1 1 11
[l, r, k] : 2 2 5
[l, r, k] : 3 3 3
[l, r, k] : 4 5 2
[l, r, k] : 6 11 1
```

上取整 $\lceil \frac{n}{g} \rceil = k$ 的分块 ($g < n$).

```cpp
for(int l = 1, r, k; l < n; l = r + 1){
    k = (n + l - 1) / l;
    r = (n + k - 2) / (k - 1) - 1;
    debug(l, r, k);
}
```

$k = \lceil \frac{n}{g} \rceil$ 从大到小遍历 $\lceil \frac{n}{g} \rceil$ 的所有取值, $[l, r]$ 对应的是 g 取值的区间.

下面是 debug 结果.

```cpp
// n = 11
[l, r, k] : 1 1 11
[l, r, k] : 2 2 6
[l, r, k] : 3 3 4
[l, r, k] : 4 5 3
[l, r, k] : 6 10 2
```

5.6.2　一般形式

设 s 为 f 的前缀.

$\sum_{i=1}^{n} f(i) \lfloor \frac{n}{i} \rfloor$.

```cpp
for (int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    ans += (s[r] - s[l - 1]) * (n / l);
}
```

$\sum_{i=1}^{n} f(i) \lfloor \frac{a}{i} \rfloor \lfloor \frac{b}{i} \rfloor$.

```cpp
for (int l = 1, r, r1, r2; l <= n; l = r + 1) {
    if (a / l) {
        r1 = a / (a / l);
    } else {
        r1 = n;
    }
    if (b / l) {
        r2 = b / (b / l);
    } else {
        r2 = n;
    }
    r = min(min(r1, r2), n);
    ans += (s[r] - s[l - 1]) * (a / l) * (b / l);
}
```

5.7 威尔逊定理

5.8 卢卡斯定理

5.8.1 卢卡斯定理

用于求大组合数, 并且模数是一个不大的素数.

$$\left(\begin{array}{c} n \\ m \end{array} \right) \bmod p = \left(\begin{array}{c} \lfloor n/p \rfloor \\ \lfloor m/p \rfloor \end{array} \right) \cdot \left(\begin{array}{c} n \bmod p \\ m \bmod p \end{array} \right) \bmod p.$$

其中 $\left(\begin{array}{c} \lfloor n/p \rfloor \\ \lfloor m/p \rfloor \end{array} \right)$ 可以继续用卢卡斯定理计算, $\left(\begin{array}{c} n \bmod p \\ m \bmod p \end{array} \right)$ 可以直接计算.

当 $m = 0$ 的时候, 返回 1.

p 不会太大, 一般在 10^5 左右.

```
auto C = [&](LL n, LL m, LL p) -> LL {
    if (n < m) return 0;
    if (m == 0) return 1;
    return ((fac[n] * inv_fac[m]) % p * inv_fac[n - m]) % p;
};

auto lucas = [&](auto&& self, LL n, LL m, LL p) -> LL {
    if (n < m) return 0;
    if (m == 0) return 1;
    return (C(n % p, m % p, p) * self(self, n / p, m / p, p)) % p;
}
```

5.8.2 素数在组合数中的次数

Legengre 给出一种 $n!$ 中素数 p 的幂次的计算方式为: $\sum_{1 \leqslant j} \lfloor \frac{n}{p^j} \rfloor$.

另一种计算方式利用 p 进制下各位数字和: $v_p(n!) = \frac{n - S_p(n)}{p-1}$.

则有 $v_p(C_m^n) = \frac{S_p(n) + S_p(m-n) - S_p(m)}{p-1}$.

5.8.3 扩展卢卡斯定理

计算 $\left(\begin{array}{c} n \\ m \end{array} \right) \bmod M$, 其中 M 可能为合数, 分为三步:

第一部分: CRT.

原问题变成求:

$$\begin{cases} \left(\begin{array}{c} n \\ m \end{array} \right) \equiv a_1 \bmod p_1^{\alpha_1} \\ \left(\begin{array}{c} n \\ m \end{array} \right) \equiv a_2 \bmod p_2^{\alpha_2} \\ \cdots \\ \left(\begin{array}{c} n \\ m \end{array} \right) \equiv a_k \bmod p_k^{\alpha_k} \end{cases}$$

,

在求出 a_i 之后就可以利用 CRT 求出答案.

第二部分: 移除分子分母中的素数.

问题转换成求解 $\begin{pmatrix} n \\ m \end{pmatrix} \bmod q^k$, 等价于求

$$\frac{\frac{n!}{q^x}}{\frac{m!}{q^y} \frac{(n-m)!}{q^z}} q^{x-y-z} \bmod q^k$$

其中 x 表示 $n!$ 中 q 的次数, y, z 同理.

第三部分: 威尔逊定理的推论.

问题转换为求 $\frac{n!}{q^x} \bmod q^k$, 可以利用威尔逊定理的推论.

```cpp
// Problem: 洛谷: P4720 【模板】扩展卢卡斯定理/exLucas

LL n, m, p;
LL fac[N], inv_fac[N];

LL quick_power(LL a, LL n, LL p){
    LL ans = 1;
    while(n != 0){
        if(n & 1) ans = (ans * a) % p;
        a = (a * a) % p;
        n >>= 1;
    }
    return ans;
}

void exgcd(LL a, LL b, LL &x, LL &y) {
    LL x1 = 1, x2 = 0, x3 = 0, x4 = 1;
    while(b != 0){
        LL c = a / b;
        tie(x1, x2, x3, x4, a, b) =
            make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
    }
    x = x1, y = x2;
}

LL mul_inv(LL a, LL p){
    LL x, y;
    exgcd(a, p, x, y);
    return (x % p + p) % p;
}

LL func(LL n, LL pi, LL pk){
    if(!n) return 1;
    LL ans = 1;
    for(LL i = 2; i <= pk; i++){
        if(i % pi) ans = ans * i % p;
    }
    ans = quick_power(ans, n / pk, pk);
    for(LL i = 2; i <= n % pk; i++){
        if(i % pi) ans = ans * i % pk;
    }
    return ans * func(n / pi, pi, pk) % pk;
}

LL multiLucas(LL n, LL m, LL pi, LL pk){
    int cnt = 0;
    for(LL i = n; i; i /= pi) cnt += i / pi;
    for(LL i = m; i; i /= pi) cnt -= i / pi;
    for(LL i = n - m; i; i /= pi) cnt -= i / pi;
    return quick_power(pi, cnt, pk) * func(n, pi, pk) % pk
            * mul_inv(func(m, pi, pk), pk) % pk * mul_inv(func(n - m, pi, pk), pk) % pk;
}

LL CRT(LL a[], LL m[], LL k){
    LL ans = 0;
    for(int i = 1; i <= k; i++){
        ans = (ans + a[i] * mul_inv(p / m[i], m[i]) * (p / m[i])) % p;
    }
    return (ans % p + p) % p;
}

LL exLucas(LL n, LL m, LL p){
    int cnt = 0;
    LL prime[20], a[20];
    for(LL i = 2; i * i <= p; i++){
        if(p % i == 0){
            prime[++cnt] = 1;
            while(p % i == 0) prime[cnt] = prime[cnt] * i, p /= i;
            a[cnt] = multiLucas(n, m, i, prime[cnt]);
        }
    }
```

```
72      if(p > 1) prime[++cnt] = p, a[cnt] = multiLucas(n, m, p, p);
73      return CRT(a, prime, cnt);
74  }
75
76  int main(){
77
78      ios::sync_with_stdio(false);
79      cin.tie(0);
80      cout.tie(0);
81
82      cin >> n >> m >> p;
83      cout << exLucas(n, m, p) << endl;
84
85      return 0;
86  }
```

5.9   裴蜀定理

5.9.1   裴蜀定理

设 x, y 是不全为零的整数, 则存在整数 a, b 使得 $ax + by = \gcd(x, y)$.

5.9.2   推论

若 $gcd(a, b) = 1, x, y \in \mathbb{N}, ax + by = n$, 则称 a, b 可以表示 n.

记 $C = ad - a - b$, 则 n 与 $C - n$ 中有且仅有一个可以被 a, b 表示.

当 $n < ab$ 时, 不大于 n 的能被表示的非负整数的个数是 $\sum\limits_{i=1}^{\left[\frac{n}{a}\right]} \left[\frac{n-ia}{b}\right]$, 可以用类欧几里得算法可求解.

5.10   升幂定理

简记为 LTE, 分为模为奇素数和模为 2 两部分, 简记为 LTF_p 和 LTF_2.

将素数 p 在整数 n 中的个数记为 $v_p(n)$.

5.10.1   模为奇素数

如果 $n \in \mathbb{N}_+, a, b \nmid p, a \equiv b \bmod p$,

则 $v_p\left(a^n - b^n\right) = v_p(a - b) + v_p(n)$

5.10.2   模为 2

如果 $n \in \mathbb{Z}_+, a, b$ 为奇数,

则 $v_2\left(a^n - b^n\right) = \begin{cases} v_2(a - b) & \text{n is odd,} \\ v_2(a - b) + v_2(a + b) +_2(n) - 1 & \text{n is even.} \end{cases}$

5.11   筛法汇总

5.11.1   素数筛

```
1  int n;
2  vi prime;
3  std::vector<bool> is_prime(n + 1);
4  void Euler_sieve(int n){
```

```
 5 |    for(int i = 2; i <= n; i++){
 6 |        if(!is_prime[i]) prime.push_back(i);
 7 |        for(auto p : prime){
 8 |            if(i * p > n) break;
 9 |            is_prime[i * p] = 1;
10 |            if(i % p == 0) break;
11 |        }
12 |    }
13 | }
14 | // is_prime 为 true 的时候是合数 //
```

5.11.2 欧拉函数 $\varphi(n)$

```
 1 | int n;
 2 | vi phi(n + 1), prime;
 3 | std::vector<bool> is_prime(n + 1);
 4 | void phi_sieve(int n){
 5 |    for(int i = 2; i <= n; i++){
 6 |        if(!is_prime[i]){
 7 |            prime.push_back(i);
 8 |            phi[i] = i - 1;
 9 |        }
10 |        for(auto p : prime){
11 |            if(i * p > n) break;
12 |            is_prime[i * p] = 1;
13 |            if(i % p){
14 |                phi[i * p] = phi[i] * phi[p];
15 |            }
16 |            else{
17 |                phi[i * p] = phi[i] * p;
18 |                break;
19 |            }
20 |        }
21 |    }
22 | }
23 | // is_prime 为 true 的时候是合数 //
```

5.11.3 莫比乌斯函数 $\mu(n)$

```
 1 | int n;
 2 | vi mu(n + 1), prime;
 3 | std::vector<bool> is_prime(n + 1);
 4 | void mu_sieve(int n){
 5 |    mu[1] = 1;
 6 |    for(int i = 2; i <= n; i++){
 7 |        if(!is_prime[i]){
 8 |            prime.push_back(i);
 9 |            mu[i] = -1;
10 |        }
11 |        for(auto p : prime){
12 |            if(i * p > n) break;
13 |            is_prime[i * p] = 1;
14 |            if(i % p){
15 |                mu[i * p] = -mu[i];
16 |            }
17 |            else{
18 |                mu[i * p] = 0;
19 |                break;
20 |            }
21 |        }
22 |    }
23 |    // is_prime 为 true 的时候是合数 //
24 | }
```

5.11.4 因数求和 $d(n)$

$$d(n) = \sum_{k|n} k$$

```
 1 | int n;
 2 | vi d(n + 1), g(n + 1), prime;
 3 | std::vector<bool> is_prime(n + 1);
 4 | void d_sieve(int n){
 5 |    d[1] = g[1] = 1;
```

```
 6        for(int i = 2; i <= n; i++){
 7            if(!is_prime[i]){
 8                prime.push_back(i);
 9                d[i] = g[i] = i + 1;
10            }
11            for(auto p : prime){
12                if(i * p > n) break;
13                is_prime[i * p] = 1;
14                if(i % p){
15                    g[i * p] = p + 1;
16                    d[i * p] = d[i] * d[p];
17                }
18                else{
19                    g[i * p] = d[i] * p + 1;
20                    d[i * p] = d[i] / g[i] * g[i * p];
21                    break;
22                }
23            }
24        }
25        // is_prime 为 true 的时候是合数 //
26 }
```

5.12   莫比乌斯反演

5.12.1   莫比乌斯函数

$$\mu(n) = \begin{cases} 1 & n = 1, \\ 0 & n含有平方因子, \\ (-1)^k & k为n的本质不同素因子个数. \end{cases}$$

几个性质:

- $\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases}$,

- $\varphi(n) = \sum_{d|n} d \cdot \mu(\frac{n}{d})$.

一个简单易写的 $O(n \log n)$ 求法.

```
1 mu[1] = 1;
2 for(int i = 1; i <= N; i++){
3     for(int j = i + i; j <= N; j += i){
4         mu[j] -= mu[i];
5     }
6 }
```

5.12.2   莫比乌斯反演

设 $f(n)$, $F(n)$.

- $F(n) = \sum_{d|n} f(d)$, 则 $f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$.

- $F(n) = \sum_{n|d} f(d)$, 则 $f(n) = \sum_{n|d} \mu(\frac{d}{n}) F(d)$.

5.12.3   例子

$$\sum_{i=1}^{n} \sum_{j=1}^{m} [gcd(i,j) = k] = \sum_{d=1}^{\min\{\lfloor \frac{n}{k} \rfloor, \lfloor \frac{m}{k} \rfloor\}} \mu(d) \lfloor \frac{n}{kd} \rfloor \lfloor \frac{m}{kd} \rfloor.$$

5.13   BSGS

5.13.1   BSGS

在 $\gcd(a, p) = 1$ 的前提下求解满足 $a^x \equiv b \bmod p$ 的 x.

时间复杂度 $O(\sqrt{p})$.

```
auto BSGS = [&](LL a, LL b, LL p) -> LL {
    if (1 % p == b % p) return 0;
    LL k = sqrt(p) + 1;
    unordered_map<LL, LL> hash(2 * k);
    for (LL i = 0, j = b % p; i < k; i++) {
        hash[j] = i;
        j = j * a % p;
    }
    LL ak = 1;
    for (int i = 1; i <= k; i++) ak = ak * a % p;
    for (int i = 1, j = ak; i <= k; i++) {
        if (hash.count(j)) return (LL) i * k - hash[j];
        j = (LL) j * ak % p;
    }
    return -inf;
};
```

5.13.2   扩展 BSGS

$(a, p) \neq 1$ 的情形.

```
std::function<LL(LL, LL, LL)> exBSGS = [&](LL a, LL b, LL p) -> LL {
    b = (b % p + p) % p;
    if ((LL) 1 % p == b % p) return 0;
    LL x, y, d;
    exgcd(a, p, x, y, d);
    if (d > 1) {
        if (b % d != 0) return -inf;
        LL d1;
        exgcd(a / d, p / d, x, y, d1);
        return exBSGS(a, b / d * x % (p / d), p / d) + 1;
    }
    return BSGS(a, b, p);
}
```

5.14   Miller-Rabin 素数检验

```
vector<int> test = {2, 7, 61};
// vector<LL> test = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

auto miller_rabin = [&](LL n) -> bool {
    if (n <= 3) return n == 2 || n == 3;
    LL a = n - 1, b = 0;
    while (!(a & 1)) a >>= 1, b++;
    for (auto x : test) {
        v = quick_power(x, a, n);
        if (v == 1 || v == n - 1) continue;
        for (j = 0; j < b; j++) {
            if (v == n - 1) break;
            v = (i28) v * v % n;
        }
        if (j >= b) return false;
    }
    return true;
};
```

```
// by 林泽瀚 //
vl vv = {2, 3, 5, 7, 11, 13, 17, 23, 29};

auto miller_rabin = [&](LL n) -> bool {
    auto test = [&](LL n, int a) {
        if (n == a) return true;
        if (n % 2 == 0) return false;
        LL d = (n - 1) >> __builtin_ctzll(n - 1);
```

```
 9          LL r = quick_power(a, d, n);
10          while (d < n - 1 and r != 1 and r != n - 1) {
11              d <<= 1;
12              r = (i128) r * r % n;
13          }
14          return r == n - 1 or d & 1;
15      };
16      if (n == 2 or n == 3) return true;
17      for (auto p : vv) {
18          if (test(n, p) == 0) return false;
19      }
20      return true;
21  }
```

5.15 Pollard-Rho 算法

能在 $O(n^{\frac{1}{4}})$ 的时间内随机出一个 n 的非平凡因数.

5.15.1 倍增实现

```
 1  auto pollard_rho = [&](LL x) -> LL{
 2      LL s = 0, t = 0, val = 1;
 3      LL c = rand() % (x - 1) + 1;
 4      for(int goal = 1;; goal <<= 1, s = t, val = 1){
 5          for(int step = 1; step <= goal; step++){
 6              t = ((i128) t * t + c) % x;
 7              val = (i128) val * abs(t - s) % x;
 8              if(step % 127 == 0){
 9                  LL d = std::gcd(val, x);
10                  if(d > 1) return d;
11              }
12          }
13          LL d = std::gcd(val, x);
14          if(d > 1) return d;
15      }
16  };
```

5.15.2 利用 Miller-Rabin 和 Pollard-Rho 进行素因数分解

```
 1  auto factorize = [&](LL a) -> vl{
 2      vl ans, stk;
 3      for (auto p : prime) {
 4          if (p > 1000) break;
 5          while (a % p == 0) {
 6              ans.push_back(p);
 7              a /= p;
 8          }
 9          if (a == 1) return ans;
10      }
11      // 先筛小素数，再跑 Pollard-Rho //
12      stk.push_back(a);
13      while (!stk.empty()) {
14          LL b = stk.back();
15          stk.pop_back();
16          if (miller_rabin(b)) {
17              ans.push_back(b);
18              continue;
19          }
20          LL c = b;
21          while (c >= b) c = pollard_rho(b);
22          stk.push_back(c);
23          stk.push_back(b / c);
24      }
25      return ans;
26  };
```

5.16 二次剩余

5.16.1 Cipolla 算法

```cpp
int cipolla(int x) {
    std::srand(time(0));
    auto check = [&](int x) -> bool { return pow(x, (mod - 1) / 2) == 1; };
    if (!x) return 0;
    if (!check(x)) return -1;
    int a, b;
    while (1) {
        a = rand() % mod;
        b = sub(mul(a, a), x);
        if (!check(b)) break;
    }
    PII t = {a, 1};
    PII ans = {1, 0};
    auto mulp = [&](PII x, PII y) -> PII {
        auto [x1, x2] = x;
        auto [y1, y2] = y;
        int c = add(mul(x1, y1), mul(x2, y2, b));
        int d = add(mul(x1, y2), mul(x2, y1));
        return {c, d};
    };
    for (int i = (mod + 1) / 2; i; i >>= 1) {
        if (i & 1) ans = mulp(ans, t);
        t = mulp(t, t);
    }
    return std::min(ans.ff, mod - ans.ff);
}
```

6   数学 - 组合数学

6.1   斯特林数

6.1.1   第一类 Stirling 数

记作 $s(n,k)$ 或者 $\left[\frac{n}{k}\right]$.

表示将 n 个两两不同的元素划分成 k 个圆排列的方案数.

递推式
$$s(n,k) = s(n-1,k-1) + (n-1)\,s(n-1,k), where\ s(n,0) = [n=0].$$

6.1.2   第二类 Stirling 数

记作 $S(n,k)$ 或者 $\left\{\frac{n}{k}\right\}$.

表示将 n 个两两不同的元素划分为 k 个互不相交的非空子集的方案数.

递推式
$$S(n,k) = S(n-1,k-1) + k\,S(n-1,k), where\ S(n,0) = [n=0].$$

7 数学 - 复数

```
tandu struct Comp {
    T a, b;

    Comp(T _a = 0, T _b = 0) { a = _a, b = _b; }

    Comp operator+(const Comp& x) const { return Comp(a + x.a, b + x.b); }

    Comp operator-(const Comp& x) const { return Comp(a - x.a, b - x.b); }

    Comp operator*(const Comp& x) const { return Comp(a * x.a - b * x.b, a * x.b + b * x.a); }

    bool operator==(const Comp& x) const { return a == x.a and b == x.b; }

    T real() { return a; }

    T imag() { return b; }

    U norm() { return (U) a * a + (U) b * b; }

    Comp conj() { return Comp(a, -b); }

    Comp operator/(const Comp& x) const {
        Comp y = x;
        Comp c = Comp(a, b) * y.conj();
        T d = y.norm();
        return Comp(c.a / d, c.b / d);
    }
};

typedef Comp<LL, LL> complex;

complex gcd(complex a, complex b) {
    LL d = b.norm();
    if (d == 0) return a;
    std::vector<complex> v(4);
    complex c = a * b.conj();
    auto fdiv = [&](LL a, LL b) -> LL { return a / b - ((a ^ b) < 0 && (a % b)); };
    v[0] = complex(fdiv(c.real(), d), fdiv(c.imag(), d));
    v[1] = v[0] + complex(1, 0);
    v[2] = v[0] + complex(0, 1);
    v[3] = v[0] + complex(1, 1);
    for (auto& x : v) {
        x = a - x * b;
    }
    std::sort(all(v), [&](complex a, complex b) { return a.norm() < b.norm(); });
    return gcd(b, v[0]);
};
```

8   数学 - 线性代数

8.1   行列式

模 998244353.

```cpp
auto det = [&](int n, vvi e) -> int {
    int ans = 1;
    for (int i = 1; i <= n; i++) {
        if (a[i][i] == 0) {
            for (int j = i + 1; j <= n; j++) {
                if (a[j][i] != 0) {
                    for (int k = i; k <= n; k++) {
                        std::swap(a[i][k], a[j][k]);
                    }
                    ans = sub(mod, ans);
                    break;
                }
            }
        }
        if (a[i][i] == 0) return 0;
        Mul(ans, a[i][i]);
        int x = pow(a[i][i], mod - 2);
        for (int k = i; k <= n; k++) {
            Mul(a[i][k], x);
        }
        for (int j = i + 1; j <= n; j++) {
            int x = a[j][i];
            for (int k = i; k <= n; k++) {
                Sub(a[j][k], mul(a[i][k], x));
            }
        }
    }
    return ans;
};
```

8.2   矩阵乘法

$A_{n \times m}$ 乘 $B_{m \times k}$ 并模 998244353.

```cpp
auto matrix_mul = [&](int n, int m, int k, vvi a, vvi b) -> vvi {
    vvi c(n + 1, vi(k + 1));
    for (int i = 1; i <= n; i++) {
        for (int l = 1; l <= m; l++) {
            int x = a[i][l];
            for (int j = 1; j <= k; j++) {
                Add(c[i][j], mul(x, b[l][j]));
            }
        }
    }
    return c;
};
```

9 博弈论

9.1 Nim 游戏

若 Nim 和为 0 , 则先手必败.

暴力打表.

```cpp
vi SG(100, -1); /* 记忆化 */
std::function<int(int)> sg = [&](int x) -> int {
    if (/* 为最终态 */) return SG[x] = 0;
    if (SG[x] != -1) return SG[x];
    vi st;
    for (/* 枚举所有可到达的状态 y */) {
        st.push_back(sg(y));
    }
    std::sort(all(st));
    st.erase(unique(all(st)), st.end());
    for (int i = 0; i < st.size(); i++) {
        if (st[i] != i) return SG[x] = i;
    }
    return SG[x] = st.size();
};
```

9.2 anti-Nim 游戏

若

- 所有堆的石子均为一个, 且 Nim 和不为 0,

- 至少有一堆石子超过一个, 且 Nim 和为 0,

则先手必败.

10   线性规划

10.1   单纯形算法

```cpp
// by jiangly //
std::vector<double> solve(const std::vector<std::vector<double> > &a,
                          const std::vector<double> &b, const std::vector<double> &c) {
    int n = (int)a.size(), m = (int)a[0].size() + 1;
    std::vector<std::vector<double> > value(n + 2, std::vector<double>(m + 1));
    std::vector<int> index(n + m);
    int r = n, s = m - 1;
    for (int i = 0; i < n + m; ++i) {
        index[i] = i;
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j) {
            value[i][j] = -a[i][j];
        }
        value[i][m - 1] = 1;
        value[i][m] = b[i];
        if (value[r][m] > value[i][m]) {
            r = i;
        }
    }
    for (int j = 0; j < m - 1; ++j) {
        value[n][j] = c[j];
    }
    value[n + 1][m - 1] = -1;
    for (double number; ; ) {
        if (r < n) {
            std::swap(index[s], index[r + m]);
            value[r][s] = 1 / value[r][s];
            for (int j = 0; j <= m; ++j) {
                if (j != s) {
                    value[r][j] *= -value[r][s];
                }
            }
            for (int i = 0; i <= n + 1; ++i) {
                if (i != r) {
                    for (int j = 0; j <= m; ++j) {
                        if (j != s) {
                            value[i][j] += value[r][j] * value[i][s];
                        }
                    }
                    value[i][s] *= value[r][s];
                }
            }
        }
        r = s = -1;
        for (int j = 0; j < m; ++j) {
            if (s < 0 || index[s] > index[j]) {
                if (value[n + 1][j] > eps || value[n + 1][j] > -eps && value[n][j] > eps) {
                    s = j;
                }
            }
        }
        if (s < 0) {
            break;
        }
        for (int i = 0; i < n; ++i) {
            if (value[i][s] < -eps) {
                if (r < 0
                || (number = value[r][m] / value[r][s] - value[i][m] / value[i][s]) < -eps
                || number < eps && index[r + m] > index[i + m]) {
                    r = i;
                }
            }
        }
        if (r < 0) {
            //    Solution is unbounded.
            return std::vector<double>();
        }
    }
    if (value[n + 1][m] < -eps) {
        //    No solution.
        return std::vector<double>();
    }
    std::vector<double> answer(m - 1);
    for (int i = m; i < n + m; ++i) {
        if (index[i] < m - 1) {
            answer[index[i]] = value[i - m][m];
        }
    }
```

```
79        }
80        return answer;
81 }
```

11　　图论

11.1　　拓扑排序

```cpp
vi top;
auto top_sort = [&]() -> bool {
    vi d(n + 1);
    std::queue<int> q;
    for (int i = 1; i <= n; i++) {
        d[i] = e[i].size();
        if (!d[i]) q.push(i);
    }
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        top.push_back(u);
        for (auto v : e[u]) {
            d[v]--;
            if (!d[v]) q.push(v);
        }
    }
    if (top.size() != n) return false;
    return true;
};
```

11.2　　最短路

11.2.1　　最短路

Floyd

```cpp
auto floyd = [&]() -> vvi {
    vvi dist(n + 1, vi(n + 1, inf));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            Min(dist[i][j], e[i][j]);
        }
        dist[i][i] = 0;
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                Min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
    return dist;
};
```

Dijkstra

```cpp
auto dijkstra = [&](int s) -> vl {
    vl dist(n + 1, INF);
    vi vis(n + 1, 0);
    dist[s] = 0;
    std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> q;
    q.emplace(0LL, s);
    while (!q.empty()) {
        auto [dis, u] = q.top();
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (auto [v, w] : e[u]) {
            if (dist[v] > dis + w) {
                dist[v] = dis + w;
                q.emplace(dist[v], v);
            }
        }
    }
    return dist;
};
```

11.2.2 最短路计数

Dijkstra

```cpp
auto dijkstra = [&](int s) -> std::pair<vl, vi> {
    vl dist(n + 1, INF);
    vi cnt(n + 1), vis(n + 1);
    dist[s] = 0;
    cnt[s] = 1;
    std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> q;
    q.emplace(0LL, s);
    while (!q.empty()) {
        auto [dis, u] = q.top();
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (auto [v, w] : e[u]) {
            if (dist[v] > dis + w) {
                dist[v] = dis + w;
                cnt[v] = cnt[u];
                q.push({dist[v], v});
            } else if (dist[v] == dis + w) {
                // cnt[v] += cnt[u];
                cnt[v] += cnt[u];
                cnt[v] %= 100003;
            }
        }
    }
    return {dist, cnt};
};
```

Floyd

```cpp
auto floyd() = [&] -> std::pair<vvi, vvi> {
    vvi dist(n + 1, vi(n + 1, inf));
    vvi cnt(n + 1, vi(n + 1));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            Min(dist[i][j], e[i][j]);
        }
        dist[i][i] = 0;
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (dist[i][j] == dist[i][k] + dist[k][j]) {
                    cnt[i][j] += cnt[i][k] * cnt[k][j];
                } else if (dist[i][j] > dist[i][k] + dist[k][j]) {
                    cnt[i][j] = cnt[i][k] * cnt[k][j];
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    return {dist, cnt};
}
```

11.2.3 负环

```cpp
auto spfa = [&]() -> bool {
    std::queue<int> q;
    vi vis(n + 1), cnt(n + 1);
    for (int i = 1; i <= n; i++) {
        q.push(i);
        vis[i] = 1;
    }
    while (!q.empty()) {
        auto u = q.front();
        q.pop();
        vis[u] = 0;
        for (auto [v, w] : e[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                cnt[v] = cnt[u] + 1;
                if (cnt[v] >= n) return true;
                if (!vis[v]) {
                    q.push(v);
                    vis[v] = 1;
                }
```

```
21            }
22        }
23    }
24    return false;
25 }
```

11.2.4   分层最短路

有一个 n 个点 m 条边的无向图, 你可以选择 k 条道路以零代价通行, 求 s 到 t 的最小花费.

```cpp
1  // Problem: 洛谷: P4568 [JLOI2011] 飞行路线
2
3  int main() {
4      std::ios::sync_with_stdio(false);
5      std::cin.tie(0);
6      std::cout.tie(0);
7
8      int n, m, k, s, t;
9      std::cin >> n >> m >> k;
10     std::cin >> s >> t;
11     std::vector<std::vector<PIL>> e(n * (k + 1) + 1);
12     for (int i = 1; i <= m; i++) {
13         int a, b, c;
14         std::cin >> a >> b >> c;
15         e[a].emplace_back(b, c);
16         e[b].emplace_back(a, c);
17         for (int j = 1; j <= k; j++) {
18             e[a + (j - 1) * n].emplace_back(b + j * n, 0);
19             e[b + (j - 1) * n].emplace_back(a + j * n, 0);
20             e[a + j * n].emplace_back(b + j * n, c);
21             e[b + j * n].emplace_back(a + j * n, c);
22         }
23     }
24
25     auto dijkstra = [&](int s) -> vl {};
26
27     vl dist = dijkstra(s);
28     LL ans = INF;
29     for (int i = t; i <= n * (k + 1); i += n) {
30         Min(ans, dist[i]);
31     }
32
33     std::cout << ans << endl;
34
35     return 0;
36 }
```

11.3   差分约束

对于不等式 $a_i - a_j \leqslant c$, 建立一条节点 j 指向 i 边权为 c 的有向边. 再连接从 0 指向 i 边权为 0 有向边, 接着跑 0 为起点的单源最短路, 如果有负环则无解, 否则 $a_i = dist_i$ 为一组解.

11.4   最小生成树

11.4.1   最小生成树

Kruskal

```cpp
1  std::vector<std::tuple<int, int, int>> edge;
2
3  // DSU //
4
5  auto kruskal = [&]() -> int {
6      std::sort(all(edge), [&](std::tuple<int, int, int> a, std::tuple<int, int, int> b) {
7          auto [x1, y1, w1] = a;
8          auto [x2, y2, w2] = b;
9          return w1 < w2;
10     });
11     int res = 0, cnt = 0;
12     for (int i = 0; i < m; i++) {
13         auto [a, b, w] = edge[i];
```

```
14          a = find(a), b = find(b);
15          if (a != b) {
16              fa[a] = b;
17              res += w;
18              // res = std::max(res, w);
19              cnt++;
20          }
21      }
22      if (cnt < n - 1) return -1;
23      return res;
24  }
```

11.5 强连通分量

11.5.1 强连通分量

Tarjan 算法

```
1   vi dfn(n + 1), low(n + 1), stk(n + 1), belong(n + 1);
2   int timestamp = 0, top = 0, scc_cnt = 0;
3   std::vector<bool> in_stk(n + 1);
4
5   auto tarjan = [&](auto&& self, int u) -> void {
6       dfn[u] = low[u] = ++timestamp;
7       stk[++top] = u;
8       in_stk[u] = true;
9       for (auto v : e[u]) {
10          if (!dfn[v]) {
11              self(self, v);
12              Min(low[u], low[v]);
13          } else if (in_stk[v]) {
14              Min(low[u], dfn[v]);
15          }
16      }
17      if (dfn[u] == low[u]) {
18          scc_cnt++;
19          int v;
20          do {
21              v = stk[top--];
22              in_stk[v] = false;
23              belong[v] = scc_cnt;
24          } while (v != u);
25      }
26  };
```

11.6 双连通分量

11.6.1 点双连通分量

求点双连通分量.

```
1   vvi e(n + 1);
2   vi dfn(n + 1), low(n + 1), is_bcc(n + 1), stk;
3   int timestamp = 0, bcc_cnt = 0, root = 0;
4   vvi bcc(2 * n + 1);
5   std::function<void(int, int)> tarjan = [&](int u, int fa) {
6       dfn[u] = low[u] = ++timestamp;
7       int child = 0;
8       stk.push_back(u);
9       if (u == root and e[u].empty()) {
10          bcc_cnt++;
11          bcc[bcc_cnt].push_back(u);
12          return;
13      }
14      for (auto v : e[u]) {
15          if (!dfn[v]) {
16              tarjan(v, u);
17              low[u] = std::min(low[u], low[v]);
18              if (low[v] >= dfn[u]) {
19                  child++;
20                  if (u != root or child > 1) {
21                      is_bcc[u] = 1;
22                  }
23                  bcc_cnt++;
```

```
24          int z;
25          do {
26              z = stk.back();
27              stk.pop_back();
28              bcc[bcc_cnt].push_back(z);
29          } while (z != v);
30          bcc[bcc_cnt].push_back(u);
31      }
32  } else if (v != fa) {
33      low[u] = std::min(low[u], dfn[v]);
34  }
35  }
36 };
37 for (int i = 1; i <= n; i++) {
38     if (!dfn[i]) {
39         root = i;
40         tarjan(i, i);
41     }
42 }
```

求割点.

```
1  vi dfn(n + 1), low(n + 1), is_bcc(n + 1);
2  int timestamp = 0, bcc = 0, root = 0;
3  std::function<void(int, int)> tarjan = [&](int u, int fa) {
4      dfn[u] = low[u] = ++timestamp;
5      int child = 0;
6      for (auto v : e[u]) {
7          if (!dfn[v]) {
8              tarjan(v, u);
9              low[u] = std::min(low[u], low[v]);
10             if (low[v] >= dfn[u]) {
11                 child++;
12                 if ((u != root or child > 1) and !is_bcc[u]) {
13                     bcc++;
14                     is_bcc[u] = 1;
15                 }
16             }
17         } else if (v != fa) {
18             low[u] = std::min(low[u], dfn[v]);
19         }
20     }
21 };
22 for (int i = 1; i <= n; i++) {
23     if (!dfn[i]) {
24         root = i;
25         tarjan(i, i);
26     }
27 }
```

11.6.2　边双连通分量

求边双连通分量.

```
1  std::vector<vpi> e(n + 1);
2  for (int i = 1; i <= m; i++) {
3      int u, v;
4      std::cin >> u >> v;
5      e[u].emplace_back(v, i);
6      e[v].emplace_back(u, i);
7  }
8  vi dfn(n + 1), low(n + 1), is_ecc(n + 1), fa(n + 1), stk;
9  int timestamp = 0, ecc_cnt = 0;
10 vvi ecc(2 * n + 1);
11 std::function<void(int, int)> tarjan = [&](int u, int id) {
12     low[u] = dfn[u] = ++timestamp;
13     stk.push_back(u);
14     for (auto [v, idx] : e[u]) {
15         if (!dfn[v]) {
16             tarjan(v, idx);
17             low[u] = std::min(low[u], low[v]);
18         } else if (idx != id) {
19             low[u] = std::min(low[u], dfn[v]);
20         }
21     }
22     if (dfn[u] == low[u]) {
23         ecc_cnt++;
24         int v;
25         do {
26             v = stk.back();
27             stk.pop_back();
```

```
28            ecc[ecc_cnt].push_back(v);
29        } while (v != u);
30    }
31 };
32 for (int i = 1; i <= n; i++) {
33    if (!dfn[i]) {
34        tarjan(i, 0);
35    }
36 }
```

求桥. (可能有诈)

```
1  vvi e(n + 1);
2  vi dfn(n + 1), low(n + 1), is_ecc(n + 1), fa(n + 1);
3  int timestamp = 0, ecc = 0;
4  std::function<void(int, int)> tarjan = [&](int u, int faa) {
5      fa[u] = faa;
6      low[u] = dfn[u] = ++timestamp;
7      for (auto v : e[u]) {
8          if (!dfn[v]) {
9              tarjan(v, u);
10             low[u] = std::min(low[u], low[v]);
11             if (low[v] > dfn[u]) {
12                 is_ecc[v] = 1;
13                 ecc++;
14             }
15         } else if (dfn[v] < dfn[u] && v != faa) {
16             low[u] = std::min(low[u], dfn[v]);
17         }
18     }
19 };
20 for (int i = 1; i <= n; i++) {
21     if (!dfn[i]) {
22         tarjan(i, i);
23     }
24 }
```

11.7   树上问题 - 树的直径

如果要找到直径上的的点, 只能用两次 DFS.

如果边权为负, 只能用树形 DP.

11.7.1   两次 DFS

```
1  vvi e(n + 1);
2  vi d(n + 1);
3  int ans, id;
4  void dfs(int u, int fa){
5      // f[u] = fa; //
6      for(auto v : e[u]){
7          if(v == fa) continue;
8          d[v] = d[u] + 1;
9          if(d[v] > d[id]) id = i;
10         dfs(v, u);
11     }
12 }
13 int main(){
14     dfs(1, 0);
15     d[id] = 0;
16     dfs(id, 0);
17     cout << d[id] << endl;
18     // for(int i = id; i; i = f[i]) cout << i << ' '; //
19     return 0;
20 }
```

11.7.2   树形 DP

```
1  vvi e(n + 1);
2  vi d1(n + 1), d2(n + 1);
3  int ans;
4  void dfs(int u, int fa){
```

```
5      d1[u] = d2[u] = 0;
6      for(int v : e[u]){
7          if(v == fa) continue;
8          dfs(v, u);
9          int t = d1[v] + 1; // t = d1[v] + w; //
10         if(t > d1[u]){
11             d2[u] = d1[u];
12             d1[u] = t;
13         }
14         else if(t > d2[u]){
15             d2[u] = t;
16         }
17     }
18     Max(ans, d1[u] + d2[u]);
19 }
20 int main(){
21     dfs(1, 0);
22     cout << ans << endl;
23     return 0;
24 }
```

11.8   树上问题 - 树的重心

只考虑点带权值

```
1  int sum;    // 点权和 //
2  vi size(n + 1), weight(n + 1), w(n + 1), depth(n + 1);
3  std::array<int, 2> centroid = {0, 0};
4  auto get_centroid = [&](auto&& self, int u, int fa) -> void {
5      size[u] = w[u];
6      weight[u] = 0;
7      for (auto v : e[u]) {
8          if (v == fa) continue;
9          self(self, v, u);
10         size[u] += size[v];
11         Max(weight[u], size[v]);
12     }
13     Max(weight[u], sum - size[u]);
14     if (weight[u] <= sum / 2) {
15         centroid[centroid[0] != 0] = u;
16     }
17 };
```

11.9   树上问题 - DSU on tree

给出一课 n 个节点以 1 为根的树, 每个节点染上一种颜色, 询问以 u 为节点的子树中有多少种颜色.

```
1  // Problem: 洛谷: U41492 树上数颜色
2
3  int main() {
4      std::ios::sync_with_stdio(false);
5      std::cin.tie(0);
6      std::cout.tie(0);
7
8      int n, m, dfn = 0, cnttot = 0;
9      std::cin >> n;
10     vvi e(n + 1);
11     vi siz(n + 1), col(n + 1), son(n + 1), dfnl(n + 1), dfnr(n + 1), rank(n + 1);
12     vi ans(n + 1), cnt(n + 1);
13
14     for (int i = 1; i < n; i++) {
15         int u, v;
16         std::cin >> u >> v;
17         e[u].push_back(v);
18         e[v].push_back(u);
19     }
20     for (int i = 1; i <= n; i++) {
21         std::cin >> col[i];
22     }
23     auto add = [&](int u) -> void {
24         if (cnt[col[u]] == 0) cnttot++;
25         cnt[col[u]]++;
26     };
27     auto del = [&](int u) -> void {
28         cnt[col[u]]--;
29         if (cnt[col[u]] == 0) cnttot--;
30     };
```

```
31        auto dfs1 = [&](auto&& self, int u, int fa) -> void {
32            dfnl[u] = ++dfn;
33            rank[dfn] = u;
34            siz[u] = 1;
35            for (auto v : e[u]) {
36                if (v == fa) continue;
37                self(self, v, u);
38                siz[u] += siz[v];
39                if (!son[u] or siz[son[u]] < siz[v]) son[u] = v;
40            }
41            dfnr[u] = dfn;
42        };
43        auto dfs2 = [&](auto&& self, int u, int fa, bool op) -> void {
44            for (auto v : e[u]) {
45                if (v == fa or v == son[u]) continue;
46                self(self, v, u, false);
47            }
48            if (son[u]) self(self, son[u], u, true);
49            for (auto v : e[u]) {
50                if (v == fa or v == son[u]) continue;
51                rep(i, dfnl[v], dfnr[v]) { add(rank[i]); }
52            }
53            add(u);
54            ans[u] = cnttot;
55            if (op == false) {
56                rep(i, dfnl[u], dfnr[u]) { del(rank[i]); }
57            }
58        };
59        dfs1(dfs1, 1, 0);
60        dfs2(dfs2, 1, 0, false);
61        std::cin >> m;
62        for (int i = 1; i <= m; i++) {
63            int u;
64            std::cin >> u;
65            std::cout << ans[u] << endl;
66        }
67        return 0;
68    }
```

11.10 树上问题 - LCA

11.10.1 倍增算法

```
1  // Problem: 洛谷: P3379 【模板】最近公共祖先( LCA )
2
3  // LCA //
4  vvi e(n + 1), fa(n + 1, vi(50));
5  vi dep(n + 1);
6
7  auto dfs = [&](auto&& self, int u) -> void {
8      for (auto v : e[u]) {
9          if (v == fa[u][0]) continue;
10         dep[v] = dep[u] + 1;
11         fa[v][0] = u;
12         self(self, v);
13     }
14 };
15
16 auto init = [&]() -> void {
17     dep[root] = 1;
18     dfs(dfs, root);
19     for (int j = 1; j <= 30; j++) {
20         for (int i = 1; i <= n; i++) {
21             fa[i][j] = fa[fa[i][j - 1]][j - 1];
22         }
23     }
24 };
25 init();
26
27 auto LCA = [&](int a, int b) -> int {
28     if (dep[a] > dep[b]) std::swap(a, b);
29     int d = dep[b] - dep[a];
30     for (int i = 0; (1 << i) <= d; i++) {
31         if (d & (1 << i)) b = fa[b][i];
32     }
33     if (a == b) return a;
34     for (int i = 30; i >= 0 and a != b; i--) {
35         if (fa[a][i] == fa[b][i]) continue;
36         a = fa[a][i];
37         b = fa[b][i];
38     }
```

```
39 |     return fa[a][0];
40 | };
41 |
42 | auto dist = [&](int a, int b) -> int { return dep[a] + dep[b] - dep[LCA(a, b)] * 2; };
```

11.11 树上问题 - 树链剖分

11.11.1 轻重链剖分

对一棵有根树进行如下 4 种操作:

- 1 x y z: 将节点 x 到节点 y 的最短路径上所有节点的值加上 z.

- 2 x y: 查询节点 x 到节点 y 的最短路径上所有节点的值的和.

- 3 x z: 将以节点 x 为根的子树上所有节点的值加上 z.

- 4 x: 查询以节点 x 为根的子树上所有节点的值的和.

```cpp
1  | // Problem: 洛谷: P3384 【模板】重链剖分/树链剖分
2  |
3  | // HLD //
4  | int cnt = 0;
5  | vi son(n + 1), fa(n + 1), siz(n + 1), depth(n + 1);
6  | vi dfn(n + 1), rank(n + 1), top(n + 1), botton(n + 1);
7  |
8  | auto dfs1 = [&](auto&& self, int u) -> void {
9  |     son[u] = -1, siz[u] = 1;
10 |     for (auto v : e[u]) {
11 |         if (depth[v] != 0) continue;
12 |         depth[v] = depth[u] + 1;
13 |         fa[v] = u;
14 |         self(self, v);
15 |         siz[u] += siz[v];
16 |         if (son[u] == -1 or siz[v] > siz[son[u]]) son[u] = v;
17 |     }
18 | };
19 |
20 | auto dfs2 = [&](auto&& self, int u, int t) -> void {
21 |     top[u] = t;
22 |     dfn[u] = ++cnt;
23 |     rank[cnt] = u;
24 |     botton[u] = dfn[u];
25 |     if (son[u] == -1) return;
26 |     self(self, son[u], t);
27 |     Max(botton[u], botton[son[u]]);
28 |     for (auto v : e[u]) {
29 |         if (v != son[u] and v != fa[u]) {
30 |             self(self, v, v);
31 |             Max(botton[u], botton[v]);
32 |         }
33 |     }
34 | };
35 |
36 | depth[root] = 1;
37 | dfs1(dfs1, root);
38 | dfs2(dfs2, root, root);
39 |
40 | /*
41 |
42 | // 求 LCA //
43 | auto LCA = [&](int a, int b) -> int {
44 |     while (top[a] != top[b]) {
45 |         if (depth[top[a]] < depth[top[b]]) std::swap(a, b);
46 |         a = fa[top[a]];
47 |     }
48 |     return (depth[a] > depth[b] ? b : a);
49 | };
50 |
51 | // 维护 u 到 v 的路径 //
52 | while (top[u] != top[v]) {
53 |     if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
54 |     opt(dfn[top[u]], dfn[u]);
55 |     u = fa[top[u]];
56 | }
57 | if (dfn[u] > dfn[v]) std::swap(u, v);
58 | opt(dfn[u], dfn[v]);
59 |
```

```
60  // 维护 u 为根的子树 //
61  opt(dfn[u], botton[u]);
62
63  */
64
65  // segment tree //
66
67  /*
68  build() 函数中
69  if(l == r) tree[u] = {l, l, w[rank[l]], 0};
70  */
71  build(1, 1, n);
72
73  for (int i = 1; i <= m; i++) {
74      int op, u, v;
75      LL k;
76      std::cin >> op;
77      if (op == 1) {
78          std::cin >> u >> v >> k;
79          while (top[u] != top[v]) {
80              if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
81              modify(1, dfn[top[u]], dfn[u], k);
82              u = fa[top[u]];
83          }
84          if (dfn[u] > dfn[v]) std::swap(u, v);
85          modify(1, dfn[u], dfn[v], k);
86      } else if (op == 2) {
87          std::cin >> u >> v;
88          LL ans = 0;
89          while (top[u] != top[v]) {
90              if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
91              ans = (ans + query(1, dfn[top[u]], dfn[u])) % p;
92              u = fa[top[u]];
93          }
94          if (dfn[u] > dfn[v]) std::swap(u, v);
95          ans = (ans + query(1, dfn[u], dfn[v])) % p;
96          std::cout << ans << endl;
97      } else if (op == 3) {
98          std::cin >> u >> k;
99          modify(1, dfn[u], botton[u], k);
100     } else {
101         std::cin >> u;
102         std::cout << query(1, dfn[u], botton[u]) % p << endl;
103     }
104 }
```

11.12 树上问题 - 树分治

11.12.1 点分治

第一个题

一棵 $n \leqslant 10^4$ 个点的树, 边权 $w \leqslant 10^4$. $m \leqslant 100$ 次询问树上是否存在长度为 $k \leqslant 10^7$ 的路径.

```
1   // 洛谷 P3806 【模板】点分治1
2
3   int main() {
4       std::ios::sync_with_stdio(false);
5       std::cin.tie(0);
6       std::cout.tie(0);
7
8       int n, m, k;
9       std::cin >> n >> m;
10
11      std::vector<vpi> e(n + 1);
12      std::map<int, PII> mp;
13
14      for (int i = 1; i < n; i++) {
15          int u, v, w;
16          std::cin >> u >> v >> w;
17          e[u].emplace_back(v, w);
18          e[v].emplace_back(u, w);
19      }
20      for (int i = 1; i <= m; i++) {
21          std::cin >> k;
22          mp[i] = {k, 0};
23      }
24
25      // centroid decomposition //
26      int top1 = 0, top2 = 0, root;
```

```
27      vi len1(n + 1), len2(n + 1), vis(n + 1);
28      static std::array<int, 20000010> cnt;
29
30      std::function<int(int, int)> get_size = [&](int u, int fa) -> int {
31          if (vis[u]) return 0;
32          int ans = 1;
33          for (auto [v, w] : e[u]) {
34              if (v == fa) continue;
35              ans += get_size(v, u);
36          }
37          return ans;
38      };
39
40      std::function<int(int, int, int, int&)> get_root = [&](int u, int fa, int tot,
41                                                             int& root) -> int {
42          if (vis[u]) return 0;
43          int sum = 1, maxx = 0;
44          for (auto [v, w] : e[u]) {
45              if (v == fa) continue;
46              int tmp = get_root(v, u, tot, root);
47              Max(maxx, tmp);
48              sum += tmp;
49          }
50          Max(maxx, tot - sum);
51          if (2 * maxx <= tot) root = u;
52          return sum;
53      };
54
55      std::function<void(int, int, int)> get_dist = [&](int u, int fa, int dist) -> void {
56          if (dist <= 10000000) len1[++top1] = dist;
57          for (auto [v, w] : e[u]) {
58              if (v == fa or vis[v]) continue;
59              get_dist(v, u, dist + w);
60          }
61      };
62
63      auto solve = [&](int u, int dist) -> void {
64          top2 = 0;
65          for (auto [v, w] : e[u]) {
66              if (vis[v]) continue;
67              top1 = 0;
68              get_dist(v, u, w);
69              for (int i = 1; i <= top1; i++) {
70                  for (int tt = 1; tt <= m; tt++) {
71                      int k = mp[tt].ff;
72                      if (k >= len1[i]) mp[tt].ss |= cnt[k - len1[i]];
73                  }
74              }
75              for (int i = 1; i <= top1; i++) {
76                  len2[++top2] = len1[i];
77                  cnt[len1[i]] = 1;
78              }
79          }
80          for (int i = 1; i <= top2; i++) cnt[len2[i]] = 0;
81      };
82
83      std::function<void(int)> divide = [&](int u) -> void {
84          vis[u] = cnt[0] = 1;
85          solve(u, 0);
86          for (auto [v, w] : e[u]) {
87              if (vis[v]) continue;
88              get_root(v, u, get_size(v, u), root);
89              divide(root);
90          }
91      };
92
93      get_root(1, 0, get_size(1, 0), root);
94      divide(root);
95
96      for (int i = 1; i <= m; i++) {
97          if (mp[i].ss == 0) {
98              std::cout << "NAY" << endl;
99          } else {
100             std::cout << "AYE" << endl;
101         }
102     }
103
104     return 0;
105 }
```

第二个题

一棵 $n \leqslant 4 \times 10^4$ 个点的树, 边权 $w \leqslant 10^3$. 询问树上长度不超过 $k \leqslant 2 \times 10^4$ 的路径的数量.

```
1 // 洛谷 P4178 Tree
```

```
2
3   int main() {
4       std::ios::sync_with_stdio(false);
5       std::cin.tie(0);
6       std::cout.tie(0);
7
8       int n, k;
9       std::cin >> n;
10      std::vector<vpi> e(n + 1);
11      for (int i = 1; i < n; i++) {
12          int u, v, w;
13          std::cin >> u >> v >> w;
14          e[u].emplace_back(v, w);
15          e[v].emplace_back(u, w);
16      }
17      std::cin >> k;
18
19      // centroid decomposition //
20      int root;
21      vi len, vis(n + 1);
22
23      std::function<int(int, int)> get_size = [&](int u, int fa) -> int {
24          if (vis[u]) return 0;
25          int ans = 1;
26          for (auto [v, w] : e[u]) {
27              if (v == fa) continue;
28              ans += get_size(v, u);
29          }
30          return ans;
31      };
32
33      std::function<int(int, int, int, int&)> get_root = [&](int u, int fa, int tot,
34                                                             int& root) -> int {
35          if (vis[u]) return 0;
36          int sum = 1, maxx = 0;
37          for (auto [v, w] : e[u]) {
38              if (v == fa) continue;
39              int tmp = get_root(v, u, tot, root);
40              maxx = std::max(maxx, tmp);
41              sum += tmp;
42          }
43          maxx = std::max(maxx, tot - sum);
44          if (2 * maxx <= tot) root = u;
45          return sum;
46      };
47
48      std::function<void(int, int, int)> get_dist = [&](int u, int fa, int dist) -> void {
49          len.push_back(dist);
50          for (auto [v, w] : e[u]) {
51              if (v == fa || vis[v]) continue;
52              get_dist(v, u, dist + w);
53          }
54      };
55
56      auto solve = [&](int u, int dist) -> int {
57          len.clear();
58          get_dist(u, 0, dist);
59          std::sort(all(len));
60          int ans = 0;
61          for (int l = 0, r = len.size() - 1; l < r;) {
62              if (len[l] + len[r] <= k) {
63                  ans += r - l++;
64              } else {
65                  r--;
66              }
67          }
68          return ans;
69      };
70
71      std::function<int(int)> divide = [&](int u) -> int {
72          vis[u] = true;
73          int ans = solve(u, 0);
74          for (auto [v, w] : e[u]) {
75              if (vis[v]) continue;
76              ans -= solve(v, w);
77              get_root(v, u, get_size(v, u), root);
78              ans += divide(root);
79          }
80          return ans;
81      };
82
83      get_root(1, 0, get_size(1, 0), root);
84      std::cout << divide(root) << endl;
85
86      return 0;
87  }
```

11.13　基环树

11.13.1　找环

```cpp
// Pseudotree //

vi roots, vis(n + 1), tmp;
int found = 0;
std::function<void(int, int)> find_ring = [&](int u, int fa) -> void {
    if (found) return;
    tmp.push_back(u);
    vis[u] = true;
    for (auto v : e[u]) {
        if (v == fa) continue;
        if (!vis[v]) {
            find_ring(v, u);
        } else {
            int flag = 0;
            for (auto x : tmp) {
                if (x == v) flag = 1;
                if (flag) roots.push_back(x);
            }
            found = 1;
            return;
        }
    }
    tmp.pop_back();
};
find_ring(1, 0);
```

11.14　树上问题 - AHU 算法

```cpp
std::map<vi, int> mapple;
std::function<int(vvi&, int, int)> tree_hash = [&](vvi& e, int u, int fa) -> int {
    vi code;
    if (u == 0) code.push_back(-1);
    for (auto v : e[u]) {
        if (v == fa) continue;
        code.push_back(tree_hash(e, v, u));
    }
    std::sort(all(code));
    int id = mapple.size();
    auto it = mapple.find(code);
    if (it == mapple.end()) {
        mapple[code] = id;
    } else {
        id = it->ss;
    }
    return id;
};
```

11.15　虚树

```cpp
// virtual tree //

auto build_vtree = [&](vi ver) -> void {
    std::sort(all(ver), [&](int x, int y) { return dfn[x] < dfn[y]; });
    vi stk = {1};
    for (auto v : ver) {
        int u = stk.back();
        int lca = LCA(v, u);
        if (lca != u) {
            while (dfn[lca] < dfn[stk.end()[-2]]) {
                g[stk.end()[-2]].push_back(stk.back());
                stk.pop_back();
            }
            u = stk.back();
            if (dfn[lca] != dfn[stk.end()[-2]]) {
                g[lca].push_back(u);
                stk.pop_back();
                stk.push_back(lca);
            } else {
                g[lca].push_back(u);
                stk.pop_back();
```

```
22            }
23        }
24        stk.push_back(v);
25    }
26    while (stk.size() > 1) {
27        int u = stk.end()[-2];
28        int v = stk.back();
29        g[u].push_back(v);
30        stk.pop_back();
31    }
32 };
```

11.16   2 - SAT

给出 n 个集合, 每个集合有 2 个元素, 已知若干个数对 (a, b), 表示 a 与 b 矛盾. 要从每个集合各选择一个元素, 判断能否一共选 n 个两两不矛盾的元素.

设集合 $\{a1, a2\}$, $\{b1, b2\}$, 如果 $a1$ 与 $b2$ 矛盾, 为了自洽, 建立由 $a1- > b1$, $a2- > b2$ 这两条有向边. 表示选了 $a1$ 则必须选 $b1$, 选了 $b2$ 则必须选 $a2$ 才能够自洽.

然后跑一遍 Tarjan 判断是否有一个集合中的两个元素在同一个 SCC 中, 若有则无解, 否则有解. 构造方案只需要把几个不矛盾的 SCC 拼起来.

```cpp
1  // Problem: 洛谷: P5782 [POI2001] 和平委员会
2
3  int main() {
4      std::ios::sync_with_stdio(false);
5      std::cin.tie(0);
6      std::cout.tie(0);
7
8      int n, m;
9      std::cin >> n >> m;
10     n *= 2;
11     vvi e(n + 1);
12     for (int i = 1; i <= m; i++) {
13         int u, v;
14         std::cin >> u >> v;
15         e[u].push_back(v & 1 ? v + 1 : v - 1);
16         e[v].push_back(u & 1 ? u + 1 : u - 1);
17     }
18
19     // tarjan //
20
21     vi ans;
22     for (int i = 1; i <= n; i += 2) {
23         if (belong[i] == belong[i + 1]) {
24             std::cout << "NIE" << endl;
25             return 0;
26         } else {
27             ans.push_back(belong[i] > belong[i + 1] ? i + 1 : i);
28         }
29     }
30     for (auto x : ans) {
31         std::cout << x << endl;
32     }
33
34     return 0;
35 }
```

11.17   欧拉图

Hierholzer 算法

11.17.1   有向图

```cpp
1  struct node{
2      int to;
3      bool exist;
4  };
5  vector<node> edge[N];
```

```
 6  vector<int> ans;
 7  int n, m, flag1, flag2;
 8  int d[N], last[N];
 9  bool cmp(node a, node b){
10      return a.to < b.to;
11  }
12  void hierholzer(int u){
13      for(int i = 0; i < edge[u].size(); i = max(i, last[u]) + 1){
14          // 比i++能加速 //
15          if(edge[u][i].exist){
16              edge[u][i].exist = 0;
17              last[u] = i;
18              hierholzer(edge[u][i].to);
19          }
20      }
21      ans.push_back(u);
22  }
23  bool check(){
24      for(int i = 1; i <= n; i++){
25          if(d[i] > 1 || d[i] < -1) return 0;
26          if(d[i] == 1) flag1++;
27          else if(d[i] == -1) flag2++;
28      }
29      if(flag1 > 1 || flag2 > 1) return 0;
30      return 1;
31  }
32  int main(){
33      /* 边: a -> b
34          scanf("%d%d", &a, &b);
35          edge[a].push_back((node){b, 1});
36          d[a]++;
37          d[b]--;
38      */
39      for(int i = 1; i <= n; i++){
40          sort(edge[i].begin(), edge[i].end(), cmp);
41      }
42      // 要求字典序最下，对边排序 //
43      if(!check()){
44          cout << "No" << endl;
45          return 0;
46      }
47      int id = 1;
48      for(int i = 1; i <= n; i++){
49          if(d[i] == 1){
50              id = i;
51              break;
52          }
53      }
54      hierholzer(id);
55      for(int i = ans.size() - 1; i >= 0; i--){
56          printf("%d ", ans[i]);
57      }
58      return 0;
59  }
```

11.17.2　无向图

```
 1  struct node{
 2      int to, revref;
 3      bool exist;
 4  };
 5  vector<node> edge[N];
 6  vector<int> ans;
 7  int n, m, flag;
 8  int d[N], reftop[N], last[N];
 9  bool cmp(node a, node b){
10      return a.to < b.to;
11  }
12  void hierholzer(int u){
13      for (int i = 0; i < edge[u].size(); i = max(i, last[u]) + 1){
14          // 比i++能加速 //
15          if(edge[u][i].exist){
16              auto t = edge[u][i];
17              t.exist = 0;
18              edge[t.to][t.revref].exist = 0;
19              last[u] = i;
20              hierholzer(t.to);
21          }
22      }
23      ans.push_back(u);
24  }
25  bool check(){
```

```
26        for(int i = 1; i <= n; i++){
27            if(d[i] % 2 == 1) flag++;
28        }
29        if(flag == 0 || flag == 2) return 1;
30        return 0;
31 }
32 int main(){
33     /* 边: a -> b
34            scanf("%d%d", &a, &b);
35            edge[a].push_back((node){b, 0, 1});
36            edge[b].push_back((node){a, 0, 1});
37            d[a]++;
38            d[b]++;
39     */
40        for(int i = 1; i <= n; i++){
41            sort(edge[i].begin(), edge[i].end(), cmp);
42        }
43        for(int i = 1; i <= n; i++){
44            for(int j = 0; j < edge[i].size(); j++){
45                edge[i][j].revref = reftop[edge[i][j].to]++;
46            }
47        }
48        if(!check()){
49            cout << "No" << endl;
50            return 0;
51        }
52        int id = 0;
53        for(int i = 1; i <= n; i++){
54            if(!d[id] && d[i]) id = i;
55            else if(!(d[id] & 1) && (d[i] & 1)) id = i;
56        }
57        hierholzer(id);
58        for(int i = ans.size() - 1; i >= 0; i--){
59            cout << ans[i] << endl;
60        }
61        return 0;
62 }
```

11.18   最小环

11.18.1   Dijkstra

枚举所有边, 每一次求删除一条边之后对这条边的起点跑一次 Dijkstra.

总复杂度 $O(m(n + m) \log n)$

11.18.2   floyd

```
1  auto min_circle = [&]() -> int {
2      vvi dist(n + 1, vi(n + 1, inf));
3      for (int i = 1; i <= n; i++) {
4          for (int j = 1; j <= n; j++) {
5              Min(dist[i][j], g[i][j]);
6          }
7          dist[i][i] = 0;
8      }
9      for (int k = 1; k <= n; k++) {
10         for (int i = 1; i < k; i++) {
11             for (int j = 1; j < i; j++) {
12                 Min(ans, dist[i][j] + g[i][k] + g[k][j]);
13             }
14         }
15         for (int i = 1; i <= n; i++) {
16             for (int j = 1; j <= n; j++) {
17                 Min(dist[i][j], dist[i][k] + dist[k][j]);
18             }
19         }
20     }
21     return ans;
22 };
```

总复杂度 $O(n^3)$

11.19   网络流 - 最大流

11.19.1   Dinic

时间复杂度为 $O(n^2m)$, 单位流量是 $O(m \cdot \min\{m^{\frac{1}{2}}, n^{\frac{2}{3}}\})$.

```cpp
struct edge {
    int from, to;
    LL cap, flow;

    edge(int u, int v, LL c, LL f) : from(u), to(v), cap(c), flow(f) {}
};

struct Dinic {
    int n, m = 0, s, t;
    std::vector<edge> e;
    vi g[N];
    int d[N], cur[N], vis[N];

    void init(int n) {
        for (int i = 0; i < n; i++) g[i].clear();
        e.clear();
        m = 0;
    }

    void add(int from, int to, LL cap) {
        e.push_back(edge(from, to, cap, 0));
        e.push_back(edge(to, from, 0, 0));
        g[from].push_back(m++);
        g[to].push_back(m++);
    }

    bool bfs() {
        for (int i = 1; i <= n; i++) {
            vis[i] = 0;
        }
        std::queue<int> q;
        q.push(s), d[s] = 0, vis[s] = 1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = 0; i < g[u].size(); i++) {
                edge& ee = e[g[u][i]];
                if (!vis[ee.to] and ee.cap > ee.flow) {
                    vis[ee.to] = 1;
                    d[ee.to] = d[u] + 1;
                    q.push(ee.to);
                }
            }
        }
        return vis[t];
    }

    LL dfs(int u, LL now) {
        if (u == t || now == 0) return now;
        LL flow = 0, f;
        for (int& i = cur[u]; i < g[u].size(); i++) {
            edge& ee = e[g[u][i]];
            edge& er = e[g[u][i] ^ 1];
            if (d[u] + 1 == d[ee.to] and (f = dfs(ee.to, std::min(now, ee.cap - ee.flow))) > 0) {
                ee.flow += f, er.flow -= f;
                flow += f, now -= f;
                if (now == 0) break;
            }
        }
        return flow;
    }

    LL dinic() {
        LL ans = 0;
        while (bfs()) {
            for (int i = 1; i <= n; i++) cur[i] = 0;
            ans += dfs(s, INF);
        }
        return ans;
    }
} maxf;
```

11.19.2   HLPP

时间复杂度上界为 $O(n^2\sqrt{m})$. 使用记得先跑 init().
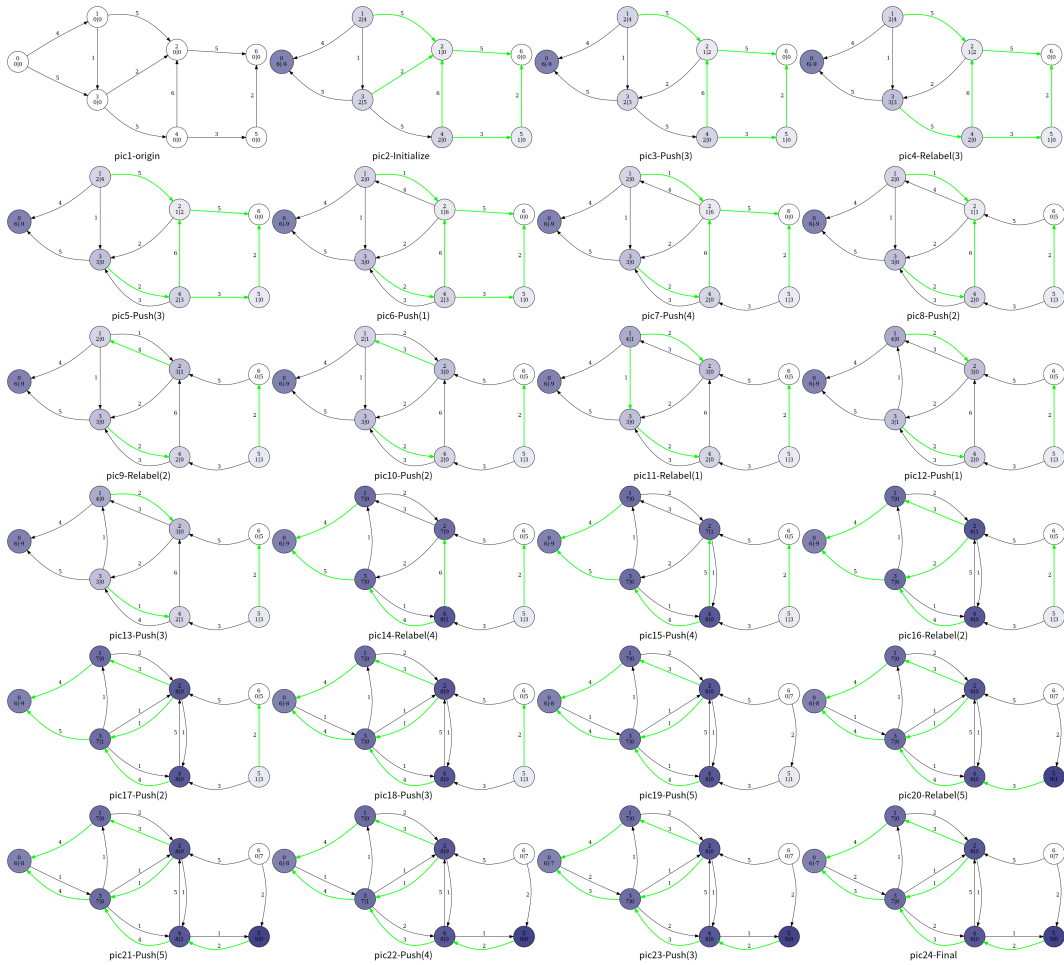

```
1   struct HLPP {
2       int n, m = 0, s, t;
3       std::vector<edge> e;        // 边 //
4       std::vector<node> nd;       // 点 //
5       std::vector<int> g[N];      // 点的连边编号 //
6       std::priority_queue<node> q;
7       std::queue<int> qq;
8       bool vis[N];
9       int cnt[N];
10
11      void init() {
12          e.clear();
13          nd.clear();
14          for (int i = 0; i <= n + 1; i++) {
15              nd.push_back(node(inf, i, 0));
16              g[i].clear();
17              vis[i] = false;
18          }
19      }
20
21      void add(int u, int v, LL w) {
22          e.push_back(edge(u, v, w));
23          e.push_back(edge(v, u, 0));
24          g[u].push_back(m++);
25          g[v].push_back(m++);
26      }
27
28      void bfs() {
29          nd[t].hight = 0;
30          qq.push(t);
31          while (!qq.empty()) {
32              int u = qq.front();
33              qq.pop();
34              vis[u] = false;
35              for (auto j : g[u]) {
36                  int v = e[j].to;
37                  if (e[j].cap == 0 && nd[v].hight > nd[u].hight + 1) {
```

```cpp
                        nd[v].hight = nd[u].hight + 1;
                        if (vis[v] == false) {
                            qq.push(v);
                            vis[v] = true;
                        }
                    }
                }
            }
        }
        return;
    }

    void _push(int u) {
        for (auto j : g[u]) {
            edge &ee = e[j], &er = e[j ^ 1];
            int v = ee.to;
            node &nu = nd[u], &nv = nd[v];
            if (ee.cap && nv.hight + 1 == nu.hight) {
                // 推流 //
                LL flow = std::min(ee.cap, nu.flow);
                ee.cap -= flow, er.cap += flow;
                nu.flow -= flow, nv.flow += flow;
                if (vis[v] == false && v != t && v != s) {
                    q.push(nv);
                    vis[v] = true;
                }
                if (nu.flow == 0) break;
            }
        }
    }

    void relabel(int u) {
        nd[u].hight = inf;
        for (auto j : g[u]) {
            int v = e[j].to;
            if (e[j].cap && nd[v].hight + 1 < nd[u].hight) {
                nd[u].hight = nd[v].hight + 1;
            }
        }
    }

    LL hlpp() {
        bfs();
        if (nd[s].hight == inf) return 0;
        nd[s].hight = n;
        for (int i = 1; i <= n; i++) {
            if (nd[i].hight < inf) cnt[nd[i].hight]++;
        }
        for (auto j : g[s]) {
            int v = e[j].to;
            int flow = e[j].cap;
            if (flow) {
                e[j].cap -= flow, e[j ^ 1].cap += flow;
                nd[s].flow -= flow, nd[v].flow += flow;
                if (vis[v] == false && v != s && v != t) {
                    q.push(nd[v]);
                    vis[v] = true;
                }
            }
        }
        while (!q.empty()) {
            int u = q.top().id;
            q.pop();
            vis[u] = false;
            _push(u);
            if (nd[u].flow) {
                cnt[nd[u].hight]--;
                if (cnt[nd[u].hight] == 0) {
                    for (int i = 1; i <= n; i++) {
                        if (i != s && i != t && nd[i].hight > nd[u].hight && nd[i].hight < n + 1) {
                            nd[i].hight = n + 1;
                        }
                    }
                }
                // 上面为 gap 优化 //
                relabel(u);
                cnt[nd[u].hight]++;
                q.push(nd[u]);
                vis[u] = true;
            }
        }
        return nd[t].flow;
    }
} maxf;
```

11.20 网络流 - 费用流

11.20.1 Dinic + SPFA

处理无负环的网络.

```cpp
struct edge {
    int from, to;
    LL cap, cost;

    edge(int u, int v, LL c, LL w) : from(u), to(v), cap(c), cost(w) {}
};

struct MCMF {
    int n, m = 0, s, t;
    std::vector<edge> e;
    vi g[N];
    int cur[N], vis[N];
    LL dist[N], minc;

    void init(int n) {
        for (int i = 0; i < n; i++) g[i].clear();
        e.clear();
        minc = m = 0;
    }

    void add(int from, int to, LL cap, LL cost) {
        e.push_back(edge(from, to, cap, cost));
        e.push_back(edge(to, from, 0, -cost));
        g[from].push_back(m++);
        g[to].push_back(m++);
    }

    bool spfa() {
        rep(i, 1, n) { dist[i] = INF, cur[i] = 0; }
        std::queue<int> q;
        q.push(s), dist[s] = 0, vis[s] = 1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            vis[u] = 0;
            for (int j = cur[u]; j < g[u].size(); j++) {
                edge& ee = e[g[u][j]];
                int v = ee.to;
                if (ee.cap && dist[v] > dist[u] + ee.cost) {
                    dist[v] = dist[u] + ee.cost;
                    if (!vis[v]) {
                        q.push(v);
                        vis[v] = 1;
                    }
                }
            }
        }
        return dist[t] != INF;
    }

    LL dfs(int u, LL now) {
        if (u == t) return now;
        vis[u] = 1;
        LL ans = 0;
        for (int& i = cur[u]; i < g[u].size() && ans < now; i++) {
            edge &ee = e[g[u][i]], &er = e[g[u][i] ^ 1];
            int v = ee.to;
            if (!vis[v] && ee.cap && dist[v] == dist[u] + ee.cost) {
                LL f = dfs(v, std::min(ee.cap, now - ans));
                if (f) {
                    minc += f * ee.cost, ans += f;
                    ee.cap -= f;
                    er.cap += f;
                }
            }
        }
        vis[u] = 0;
        return ans;
    }

    PLL mcmf() {
        LL maxf = 0;
        while (spfa()) {
            LL tmp;
            while ((tmp = dfs(s, INF))) maxf += tmp;
        }
        return std::makepair(maxf, minc);
    }
```

```
79 | } minc_maxf;
```

11.20.2   Primal-Dual 原始对偶算法

处理无负环的网络.

```
 1 | struct edge {
 2 |     int from, to;
 3 |     LL cap, cost;
 4 |
 5 |     edge(int u, int v, LL c, LL w) : from(u), to(v), cap(c), cost(w) {}
 6 | };
 7 |
 8 | struct node {
 9 |     int v, e;
10 |
11 |     node(int _v = 0, int _e = 0) : v(_v), e(_e) {}
12 | };
13 |
14 | const int maxn = 5000 + 10;
15 |
16 | struct MCMF {
17 |     int n, m = 0, s, t;
18 |     std::vector<edge> e;
19 |     vi g[maxn];
20 |     int dis[maxn], vis[maxn], h[maxn];
21 |     node p[maxn * 2];
22 |
23 |     void add(int from, int to, LL cap, LL cost) {
24 |         e.push_back(edge(from, to, cap, cost));
25 |         e.push_back(edge(to, from, 0, -cost));
26 |         g[from].push_back(m++);
27 |         g[to].push_back(m++);
28 |     }
29 |
30 |     bool dijkstra() {
31 |         std::priority_queue<PII, std::vector<PII>, std::greater<PII>> q;
32 |         for (int i = 1; i <= n; i++) {
33 |             dis[i] = inf;
34 |             vis[i] = 0;
35 |         }
36 |         dis[s] = 0;
37 |         q.push({0, s});
38 |         while (!q.empty()) {
39 |             int u = q.top().ss;
40 |             q.pop();
41 |             if (vis[u]) continue;
42 |             vis[u] = 1;
43 |             for (auto i : g[u]) {
44 |                 edge ee = e[i];
45 |                 int v = ee.to, nc = ee.cost + h[u] - h[v];
46 |                 if (ee.cap and dis[v] > dis[u] + nc) {
47 |                     dis[v] = dis[u] + nc;
48 |                     p[v] = node(u, i);
49 |                     if (!vis[v]) q.push({dis[v], v});
50 |                 }
51 |             }
52 |         }
53 |         return dis[t] != inf;
54 |     }
55 |
56 |     void spfa() {
57 |         std::queue<int> q;
58 |         for (int i = 1; i <= n; i++) h[i] = inf;
59 |         h[s] = 0, vis[s] = 1;
60 |         q.push(s);
61 |         while (!q.empty()) {
62 |             int u = q.front();
63 |             q.pop();
64 |             vis[u] = 0;
65 |             for (auto i : g[u]) {
66 |                 edge ee = e[i];
67 |                 int v = ee.to;
68 |                 if (ee.cap and h[v] > h[u] + ee.cost) {
69 |                     h[v] = h[u] + ee.cost;
70 |                     if (!vis[v]) {
71 |                         vis[v] = 1;
72 |                         q.push(v);
73 |                     }
74 |                 }
75 |             }
76 |         }
```

```
77          }
78
79      PLL mcmf() {
80          LL maxf = 0, minc = 0;
81          spfa();
82          while (dijkstra()) {
83              LL minf = INF;
84              for (int i = 1; i <= n; i++) h[i] += dis[i];
85              for (int i = t; i != s; i = p[i].v) minf = std::min(minf, e[p[i].e].cap);
86              for (int i = t; i != s; i = p[i].v) {
87                  e[p[i].e].cap -= minf;
88                  e[p[i].e ^ 1].cap += minf;
89              }
90              maxf += minf;
91              minc += minf * h[t];
92          }
93          return std::makepair(maxf, minc);
94      }
95  } minc_maxf;
```

11.21 网络流 - 最小割

最小割解决的问题是将图中的点集 V 划分成 S 与 T, 使得 S 与 T 之间的连边的容量总和最小.

11.21.1 最大流最小割定理

网络中 s 到 t 的最大流流量的值等于所要求的最小割的值. 所以求最小割只需要跑 Dinic 即可.

11.21.2 获取 S 中的点

在 Dinic 的 bfs 函数中, 每次将所有点的 d 数组值改为无穷大, 最后跑完最大流之后 d 数组不为无穷大的就是和源点一起在 S 集合中的点.

11.21.3 例子

最小割的本质是对图中点集进行 2-划分, 网络流只是求解答案的手段.

1. 在图中花费最小的代价断开一些边使得源点 s 无法流到汇点 t

   直接跑最大流就得到了答案.

2. 在图中删除最少的点使得源点 s 无法流到汇点 t

   对每个点进行拆点, 在 i 与 i' 之间建立容量为 1 的有向边.

11.22 图匹配 - 二分图最大匹配

11.22.1 Kuhn-Munkres 算法

时间复杂度: $O(n^3)$.

```
1   auto KM = [&](int n1, int n2, vvi e) -> std::pair<vi, vi> {
2       vi vis(n2 + 1);
3       vi l(n1 + 1, -1), r(n2 + 1, -1);
4       std::function<bool(int)> dfs = [&](int u) -> bool {
5           for (auto v : e[u]) {
6               if (!vis[v]) {
7                   vis[v] = 1;
8                   if (r[v] == -1 or dfs(r[v])) {
9                       r[v] = u;
10                      return true;
11                  }
12              }
```

```
13 |         }
14 |         return false;
15 |     };
16 |     for (int i = 1; i <= n1; i++) {
17 |         std::fill(all(vis), 0);
18 |         dfs(i);
19 |     }
20 |     for (int i = 1; i <= n2; i++) {
21 |         if (r[i] == -1) continue;
22 |         l[r[i]] = i;
23 |     }
24 |     return {l, r};
25 | };
26 | auto [mchl, mchr] = KM(n1, n2, e);
27 | std::cout << mchl.size() - std::count(all(mchl), -1) << endl;
```

11.22.2   Hopcroft-Karp 算法

据说时间复杂度是 $O(m\sqrt{n})$ 的, 但是快的飞起.

```
 1 | vpi e(m);
 2 | auto hopcroft_karp = [&](int n, int m, vpi& e) -> std::pair<vi, vi> {
 3 |     vi g(e.size()), l(n + 1, -1), r(m + 1, -1), d(n + 2);
 4 |     for (auto [u, v] : e) d[u]++;
 5 |     std::partial_sum(all(d), d.begin());
 6 |     for (auto [u, v] : e) g[--d[u]] = v;
 7 |     for (vi a, p, q(n + 1);;) {
 8 |         a.assign(n + 1, -1);
 9 |         p.assign(n + 1, -1);
10 |         int t = 1;
11 |         for (int i = 1; i <= n; i++) {
12 |             if (l[i] == -1) {
13 |                 q[t++] = a[i] = p[i] = i;
14 |             }
15 |         }
16 |         bool match = false;
17 |         for (int i = 1; i < t; i++) {
18 |             int u = q[i];
19 |             if (l[a[u]] != -1) continue;
20 |             for (int j = d[u]; j < d[u + 1]; j++) {
21 |                 int v = g[j];
22 |                 if (r[v] == -1) {
23 |                     while (v != -1) {
24 |                         r[v] = u;
25 |                         std::swap(l[u], v);
26 |                         u = p[u];
27 |                     }
28 |                     match = true;
29 |                     break;
30 |                 }
31 |                 if (p[r[v]] == -1) {
32 |                     q[t++] = v = r[v];
33 |                     p[v] = u;
34 |                     a[v] = a[u];
35 |                 }
36 |             }
37 |         }
38 |         if (!match) break;
39 |     }
40 |     return {l, r};
41 | };
42 | auto [mchl, mchr] = hopcroft_karp(n1, n2, e);
43 | std::cout << mchl.size() - std::count(all(mchl), -1) << endl;
```

11.23   图匹配 - 二分图最大权匹配

11.23.1   Kuhn-Munkres

注意是否为完美匹配, 非完美选 0, 完美选 $-INF$.

```
 1 | auto KM = [&](int n, vvl e) -> std::tuple<LL, vi, vi> {
 2 |     vl la(n + 1), lb(n + 1), pp(n + 1), vx(n + 1);
 3 |     vi l(n + 1, -1), r(n + 1, -1);
 4 |     vi va(n + 1), vb(n + 1);
 5 |     LL delta;
 6 |     auto bfs = [&](int x) -> void {
```

```cpp
        int a, y = 0, y1 = 0;
        std::fill(all(pp), 0);
        std::fill(all(vx), INF);
        r[y] = x;
        do {
            a = r[y], delta = INF, vb[y] = 1;
            for (int b = 1; b <= n; b++) {
                if (!vb[b]) {
                    if (vx[b] > la[a] + lb[b] - e[a][b]) {
                        vx[b] = la[a] + lb[b] - e[a][b];
                        pp[b] = y;
                    }
                    if (vx[b] < delta) {
                        delta = vx[b];
                        y1 = b;
                    }
                }
            }
            for (int b = 0; b <= n; b++) {
                if (vb[b]) {
                    la[r[b]] -= delta;
                    lb[b] += delta;
                } else
                    vx[b] -= delta;
            }
            y = y1;
        } while (r[y] != -1);
        while (y) {
            r[y] = r[pp[y]];
            y = pp[y];
        }
    };
    for (int i = 1; i <= n; i++) {
        std::fill(all(vb), 0);
        bfs(i);
    }
    LL ans = 0;
    for (int i = 1; i <= n; i++) {
        if (r[i] == -1) continue;
        l[r[i]] = i;
        ans += e[r[i]][i];
    }
    return {ans, l, r};
};

auto [ans, mchl, mchr] = KM(n, e);
```

12 计算几何

12.1 二维基础

12.1.1 向量计算

```
tandu struct pnt {
    T x, y;

    pnt(T _x = 0, T _y = 0) { x = _x, y = _y; }

    pnt operator+(const pnt& a) const { return pnt(x + a.x, y + a.y); }

    pnt operator-(const pnt& a) const { return pnt(x - a.x, y - a.y); }
    /*
    bool operator<(const pnt& a) const {
        if (fabs(x - a.x) < eps) return y < a.y;
        return x < a.x;
    }
    */
    // 注意数乘会不会爆 int //
    pnt operator*(const T k) const { return pnt(k * x, k * y); }

    U operator*(const pnt& a) const { return (U) x * a.x + (U) y * a.y; }

    U operator^(const pnt& a) const { return (U) x * a.y - (U) y * a.x; }

    U dist(const pnt a) { return ((U) a.x - x) * ((U) a.x - x) + ((U) a.y - y) * ((U) a.y - y); }

    U len() { return dist(pnt(0, 0)); }

    // 两向量夹角，返回 cos 值 //
    double get_angle(pnt a) {
        return (double) (pnt(x, y) * a) / sqrt((double) pnt(x, y).len() * (double) a.len());
    }
};

typedef pnt<LL, LL> point;
```

12.2 凸包

12.2.1 二维凸包

```
// convex hull //
auto andrew = [&]() -> std::vector<point> {
    std::sort(all(v));
    std::vector<point> stk;
    for (int i = 0; i < n; i++) {
        point x = v[i];
        while (stk.size() > 1 and ((stk.end()[-1] - stk.end()[-2]) ^ (x - stk.end()[-2])) <= 0) {
            stk.pop_back();
        }
        stk.push_back(x);
    }
    int tmp = stk.size();
    for (int i = n - 2; i >= 0; i--) {
        point x = v[i];
        while (stk.size() > tmp and ((stk.end()[-1] - stk.end()[-2]) ^ (x - stk.end()[-2])) <= 0) {
            stk.pop_back();
        }
        stk.push_back(x);
    }
    return stk;
};
auto convex = andrew();
```

13 离线算法

13.1 莫队

13.1.1 普通莫队

```cpp
int block = n / sqrt(2 * m / 3);
std::sort(all(q), [&](node a, node b) {
    return a.l / block == b.l / block ? (a.r == b.r ? 0 : ((a.l / block) & 1) ^ (a.r < b.r))
                                      : a.l < b.l;
});

auto move = [&](int x, int op) -> void {
    if (op == 1) {
        ...
    } else {
        ...
    }
};

for (int k = 1, l = 1, r = 0; k <= m; k++) {
    node Q = q[k];
    while (l > Q.l) {
        move(--l, 1);
    }
    while (r < Q.r) {
        move(++r, 1);
    }
    while (l < Q.l) {
        move(l++, -1);
    }
    while (r > Q.r) {
        move(r--, -1);
    }
}
```

13.1.2 带修改莫队

13.1.3 树上莫队

13.2 离散化

```cpp
std::sort(all(a));
a.erase(unique(all(a)), a.end());
auto get_id = [&](const int& x) -> int { return lower_bound(all(a), x) - a.begin() + 1; };
```

13.3 CDQ 分治

n 个三维数对 (a_i, b_i, c_i), 设 $f(i)$ 表示 $a_j \leqslant a_i$ 且 $b_j \leqslant b_i$ 且 $c_j \leqslant c_i$ 且 $i \neq j$ 的个数.

输出 $f(i)$ $(0 \leqslant i \leqslant n-1)$ 的值.

```cpp
// 洛谷 P3810 【模板】三维偏序( 陌上花开 )

struct data {
    int a, b, c, cnt, ans;

    data(int _a = 0, int _b = 0, int _c = 0, int _cnt = 0, int _ans = 0) {
        a = _a, b = _b, c = _c, cnt = _cnt, ans = _ans;
    }

    bool operator!=(data x) {
        if (a != x.a) return true;
        if (b != x.b) return true;
        if (c != x.c) return true;
        return false;
    }
};
```

```cpp
17
18 int main() {
19     std::ios::sync_with_stdio(false);
20     std::cin.tie(0);
21     std::cout.tie(0);
22
23     int n, k;
24     std::cin >> n >> k;
25     static data v1[N], v2[N];
26     for (int i = 1; i <= n; i++) {
27         std::cin >> v1[i].a >> v1[i].b >> v1[i].c;
28     }
29
30     std::sort(v1 + 1, v1 + n + 1, [&](data x, data y) {
31         if (x.a != y.a) return x.a < y.a;
32         if (x.b != y.b) return x.b < y.b;
33         return x.c < y.c;
34     });
35
36     int t = 0, top = 0;
37     for (int i = 1; i <= n; i++) {
38         t++;
39         if (v1[i] != v1[i + 1]) {
40             v2[++top] = v1[i];
41             v2[top].cnt = t;
42             t = 0;
43         }
44     }
45
46     // BIT //
47
48     // CDQ //
49     std::function<void(int, int)> CDQ = [&](int l, int r) -> void {
50         if (l == r) return;
51         int mid = (l + r) >> 1;
52         CDQ(l, mid), CDQ(mid + 1, r);
53         std::sort(v2 + l, v2 + mid + 1, [&](data x, data y) {
54             if (x.b != y.b) return x.b < y.b;
55             return x.c < y.c;
56         });
57         std::sort(v2 + mid + 1, v2 + r + 1, [&](data x, data y) {
58             if (x.b != y.b) return x.b < y.b;
59             return x.c < y.c;
60         });
61         int i = l, j = mid + 1;
62         while (j <= r) {
63             while (i <= mid && v2[i].b <= v2[j].b) {
64                 add(v2[i].c, v2[i].cnt);
65                 i++;
66             }
67             v2[j].ans += query(v2[j].c);
68             j++;
69         }
70         for (int ii = l; ii < i; ii++) {
71             add(v2[ii].c, -v2[ii].cnt);
72         }
73         return;
74     };
75
76     CDQ(1, top);
77     vi ans(n + 1);
78     for (int i = 1; i <= top; i++) {
79         ans[v2[i].ans + v2[i].cnt] += v2[i].cnt;
80     }
81     for (int i = 1; i <= n; i++) {
82         std::cout << ans[i] << endl;
83     }
84
85     return 0;
86 }
```