

BELJING NORMAL UNIVERSITY  
SCHOOL OF MATHEMATICS

---

# Template

---

appleDog

2023 年 11 月 10 日

# 目录

<b>1</b>	<b>hpp</b>	<b>5</b>
1.1	heading . . . . .	5
1.2	debug.h . . . . .	6
1.3	$F_p$ . . . . .	6
<b>2</b>	<b>shell scripts</b>	<b>8</b>
2.1	md5er.sh . . . . .	8
2.2	formater.sh . . . . .	8
2.3	checker.sh . . . . .	8
<b>3</b>	<b>data structure</b>	<b>9</b>
3.1	stack . . . . .	9
3.2	queue . . . . .	9
3.3	DSU . . . . .	9
3.4	ST . . . . .	9
3.5	cartesian tree . . . . .	10
3.6	segment tree . . . . .	10
3.7	hjt segment tree . . . . .	16
3.8	treap . . . . .	19
3.9	splay . . . . .	24
3.10	tree in tree . . . . .	27
<b>4</b>	<b>string</b>	<b>33</b>
4.1	kmp . . . . .	33
4.2	z function . . . . .	33
4.3	trie . . . . .	34
<b>5</b>	<b>math - number theory</b>	<b>37</b>
5.1	Eculid . . . . .	37
5.2	inverse . . . . .	38
5.3	sieve . . . . .	39
5.4	block . . . . .	41
5.5	CRT & exCRT . . . . .	42
5.6	BSGS & exBSGS . . . . .	42
5.7	Miller Rabin . . . . .	43

目录	3
5.8 Pollard Rho . . . . .	43
5.9 quadratic residu . . . . .	44
5.10 Lucas . . . . .	44
5.11 Wilson . . . . .	47
5.12 LTE . . . . .	47
5.13 Mobius inversion . . . . .	48
<b>6 math - polynomial</b>	<b>49</b>
6.1 FTT . . . . .	49
6.2 FWT . . . . .	50
6.3 class polynomial . . . . .	52
6.4 wsy poly . . . . .	55
<b>7 math - game theory</b>	<b>63</b>
7.1 nim game . . . . .	63
7.2 anti - nim game . . . . .	63
<b>8 math - linear algebra</b>	<b>64</b>
8.1 matrix . . . . .	64
8.2 linear basis . . . . .	64
8.3 linear programming . . . . .	64
<b>9 complex number</b>	<b>65</b>
<b>10 graph</b>	<b>66</b>
10.1 topsort . . . . .	66
10.2 shortest path . . . . .	66
10.3 minimum spanning tree . . . . .	70
10.4 SCC . . . . .	70
10.4.1 缩点 . . . . .	71
10.5 DCC . . . . .	71
10.6 two set . . . . .	72
10.7 minimum ring . . . . .	73
10.8 tree - center of gravity . . . . .	73
10.9 tree - DSU on tree . . . . .	74
10.10 tree - AHU . . . . .	75
10.11 tree - LCA . . . . .	75
10.12 tree - HLD . . . . .	76

10.13	tree - virtual tree . . . . .	77
10.14	tree - pseudo tree . . . . .	77
10.15	tree - divide and conquer on tree . . . . .	78
10.16	network flow - maximal flow . . . . .	81
10.17	network flow - minimum cost flow . . . . .	84
10.18	network flow - minimal cut . . . . .	86
10.19	matching - matching on bipartite graph . . . . .	87
10.20	matching - matching on general graph . . . . .	88
<b>11</b>	<b>geometry</b>	<b>89</b>
11.1	two demention . . . . .	89
11.2	convex . . . . .	90
<b>12</b>	<b>offline algorithm</b>	<b>91</b>
12.1	discretization . . . . .	91
12.2	Mo algorithm . . . . .	91
12.3	CDQ . . . . .	91

# 1 hpp

## 1.1 heading

```

1  #include <bits/stdc++.h>
2
3  // using namespace std;
4
5  #define typet typename T
6  #define typeu typename U
7  #define types typename... Ts
8  #define tempt template <typet>
9  #define tempu template <typeu>
10 #define temps template <types>
11 #define tandu template <typet, typeu>
12
13 using LL = long long;
14 using i128 = __int128;
15 using PII = std::pair<int, int>;
16 /*
17 using UI = unsigned int;
18 using ULL = unsigned long long;
19 using ULL = unsigned long long;
20 using PIL = std::pair<int, LL>;
21 using PLI = std::pair<LL, int>;
22 using PLL = std::pair<LL, LL>;
23 */
24 using vi = std::vector<int>;
25 using vvi = std::vector<vi>;
26 using vl = std::vector<LL>;
27 using vvl = std::vector<vl>;
28 using vpi = std::vector<PII>;
29
30 #define ff first
31 #define ss second
32 #define all(v) v.begin(), v.end()
33 #define rall(v) v.rbegin(), v.rend()
34
35 #ifdef LOCAL
36 #include "debug.h"
37 #else
38 #define debug(...) \
39     do { \
40         } while (false)
41 #endif
42
43 constexpr int mod = 998244353;
44 constexpr int inv2 = (mod + 1) / 2;
45 constexpr int inf = 0x3f3f3f3f;
46 constexpr LL INF = 1e18;
47 constexpr double pi = 3.141592653589793;
48 constexpr double eps = 1e-6;
49
50 constexpr int lowbit(int x) { return x & -x; }
51 /*
52 constexpr int add(int x, int y) { return x + y < mod ? x + y : x - mod + y; }
53 constexpr int sub(int x, int y) { return x < y ? mod + x - y : x - y; }
54 constexpr int mul(LL x, int y) { return x * y % mod; }
55 constexpr void Add(int& x, int y) { x = add(x, y); }
56 constexpr void Sub(int& x, int y) { x = sub(x, y); }
57 constexpr void Mul(int& x, int y) { x = mul(x, y); }
58 constexpr int pow(int x, int y, int z = 1) {
59     for (; y; y /= 2) {
60         if (y & 1) Mul(z, x);
61         Mul(x, x);
62     }
63     return z;
64 }
65 temps constexpr int add(Ts... x) {
66     int y = 0;
67     (... , Add(y, x));
68     return y;
69 }
70 temps constexpr int mul(Ts... x) {
71     int y = 1;
72     (... , Mul(y, x));
73     return y;
74 }
75 */
76
77 tandu bool Max(T& x, const U& y) { return x < y ? x = y, true : false; }
78 tandu bool Min(T& x, const U& y) { return x > y ? x = y, true : false; }
79

```

```

80 void solut() {
81 }
82 }
83
84 int main() {
85     std::ios::sync_with_stdio(false);
86     std::cin.tie(0);
87     std::cout.tie(0);
88
89     int t = 1;
90     std::cin >> t;
91     while (t-- > 0) {
92         solut();
93     }
94     return 0;
95 }

```

## 1.2 debug.h

md5 为:

```

1  tandu std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p) {
2      return os << '<' << p.ff << ',' << p.ss << '>';
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 << std::endl;
25 }

```

## 1.3 $F_p$

```

1  template <int P>
2  struct Mint {
3      int v = 0;
4
5      // reflection
6      template <typet = int>
7      constexpr operator T() const {
8          return v;
9      }
10
11     // constructor //
12     constexpr Mint() = default;
13     template <typet>
14     constexpr Mint(T x) : v(x % P) {}
15     constexpr int val() const { return v; }
16     constexpr int mod() { return P; }
17
18     // io //
19     friend std::istream& operator>>(std::istream& is, Mint& x) {
20         LL y;
21         is >> y;
22         x = y;
23         return is;
24     }
25     friend std::ostream& operator<<(std::ostream& os, Mint x) { return os << x.v; }
26
27     // comparision //
28     friend constexpr bool operator==(const Mint& lhs, const Mint& rhs) { return lhs.v == rhs.v; }
29     friend constexpr bool operator!=(const Mint& lhs, const Mint& rhs) { return lhs.v != rhs.v; }

```

```

30 friend constexpr bool operator<(const Mint& lhs, const Mint& rhs) { return lhs.v < rhs.v; }
31 friend constexpr bool operator<=(const Mint& lhs, const Mint& rhs) { return lhs.v <= rhs.v; }
32 friend constexpr bool operator>(const Mint& lhs, const Mint& rhs) { return lhs.v > rhs.v; }
33 friend constexpr bool operator>=(const Mint& lhs, const Mint& rhs) { return lhs.v >= rhs.v; }
34
35 // arithmetic //
36 template <typet>
37 friend constexpr Mint power(Mint a, T n) {
38     Mint ans = 1;
39     while (n) {
40         if (n & 1) ans *= a;
41         a *= a;
42         n >>= 1;
43     }
44     return ans;
45 }
46 friend constexpr Mint inv(const Mint& rhs) { return power(rhs, P - 2); }
47 friend constexpr Mint operator+(const Mint& lhs, const Mint& rhs) {
48     return lhs.val() + rhs.val() < P ? lhs.val() + rhs.val() : lhs.val() - P + rhs.val();
49 }
50 friend constexpr Mint operator-(const Mint& lhs, const Mint& rhs) {
51     return lhs.val() < rhs.val() ? lhs.val() + P - rhs.val() : lhs.val() - rhs.val();
52 }
53 friend constexpr Mint operator*(const Mint& lhs, const Mint& rhs) {
54     return static_cast<LL>(lhs.val()) * rhs.val() % P;
55 }
56 friend constexpr Mint operator/(const Mint& lhs, const Mint& rhs) { return lhs * inv(rhs); }
57 Mint operator+() const { return *this; }
58 Mint operator-() const { return Mint() - *this; }
59 constexpr Mint& operator++() {
60     v++;
61     if (v == P) v = 0;
62     return *this;
63 }
64 constexpr Mint& operator--() {
65     if (v == 0) v = P;
66     v--;
67     return *this;
68 }
69 constexpr Mint& operator++(int) {
70     Mint ans = *this;
71     ++*this;
72     return ans;
73 }
74 constexpr Mint& operator--(int) {
75     Mint ans = *this;
76     --*this;
77     return ans;
78 }
79 constexpr Mint& operator+=(const Mint& rhs) {
80     v = v + rhs;
81     return *this;
82 }
83 constexpr Mint& operator-=(const Mint& rhs) {
84     v = v - rhs;
85     return *this;
86 }
87 constexpr Mint& operator*=(const Mint& rhs) {
88     v = v * rhs;
89     return *this;
90 }
91 constexpr Mint& operator/=(const Mint& rhs) {
92     v = v / rhs;
93     return *this;
94 }
95 };
96 using Z = Mint<998244353>;

```

## 2 shell scripts

### 2.1 md5er.sh

得到一份 cpp 代码的 MD5 码.

```
#!/bin/bash

hash=$(md5sum <(tr -d '[:space:]' < "$1") | awk '{print $1}')
```

### 2.2 formater.sh

修改.out 以及.ans 的格式:

```
#!/bin/bash

if false; then
    if [ ! -f "$1" ]; then
        echo "File not found!"
        exit 1
    fi
fi

# The code above is to ensure the stability of the program

sed -i 's/[[:space:]]*$/ ' "$1"
sed -i -e 's/{/^$/!G;}' "$1"
```

### 2.3 checker.sh

对一份代码跑所有测试样例并比对.

```
#!/bin/bash

# current=$(pwd)
cd "$1"

g++ -o main -O2 -std=c++17 -DLOCAL main.cpp

for input in *.in; do
    output=${input%.*}.out
    answer=${input%.*}.ans

    ./main < $input > $output

    echo "case ${input%.*}: "
    echo "My: "
    cat $output
    echo "Answer: "
    cat $answer

    # if you want to check by yourself, then you don't need the code below
    if false; then
        $("current"/formater.sh $output)
        $("current"/formater.sh $answer)

        if diff $output $answer > /dev/null; then
            echo "${input%.*}: Accepted"
        else
            echo "${input%.*}: Wrong answer"
            cat $output
            cat $answer
        fi
    fi
done
```



## 3 data structure

### 3.1 stack

```

1 vi stk;
2 for (int i = 1; i <= n; i++){
3     while (!stk.empty() and stk.back() > a[i]) {
4         stk.pop_back();
5     }
6     stk.push_back(a[i]);
7 }

```

### 3.2 queue

```

1 std::deque<int> q;
2 for (int i = 1; i <= n; i++) {
3     while (!q.empty() and a[q.back()] >= a[i]) q.pop_back();
4     if (!q.empty() and i - q.front() >= k) q.pop_front();
5     q.push_back(i);
6 }

```

### 3.3 DSU

```

1 vi fa(n + 1);
2 std::iota(all(fa), 0);
3 std::function<void(int)> find = [&] (int x) -> int{
4     return x == fa[x] ? x : fa[x] = find(fa[x]);
5 };
6 auto merge = [&] (int x, int y) -> void{
7     x = find(x), y = find(y);
8     if (x == y) return;
9     /* operations */
10    fa[y] = x;
11 };

```

### 3.4 ST

用于解决可重复问题的数据结构。

可重复问题是指对运算  $opt$ ，满足  $x \ opt \ x = x$ 。

一维

```

1 vvi f(n + 1, vi(30));
2 vi Log2(n + 1);
3 auto ST_init = [&]() -> void {
4     for (int i = 1; i <= n; i++) {
5         f[i][0] = a[i];
6         if (i > 1) Log2[i] = Log2[i / 2] + 1;
7     }
8     int t = Log2[n];
9     for (int j = 1; j <= t; j++) {
10        for (int i = 1; i <= n - (1 << j) + 1; i++) {
11            f[i][j] = std::max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
12        }
13    }
14 };
15
16 auto ST_query = [&](int l, int r) -> int {
17     int t = Log2[r - l + 1];
18     return std::max(f[l][t], f[r - (1 << t) + 1][t]);
19 };

```

## 二维

```

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

```

## 3.5 cartesian tree

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 }

```

## 3.6 segment tree

## 维护半群

```

1  struct Info {
2      /* 重载 operator+ */
3  };
4
5  struct Tag {
6      /* 重载 operator== */
7  };
8
9  void infoApply(Info& a, int l, int r, const Tag& tag) {}
10
11 void tagApply(Tag& a, int l, int r, const Tag& tag) {}
12
13 template <class Info, class Tag>
14 class segTree {
15 #define ls i << 1

```

```

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

```

## 区间修改 (带 add 和 mul 的 lazy tag)

$n$  个数,  $m$  次操作, 操作分为

1.  $1\ x\ y\ k$ : 将区间  $[x, y]$  中的数每个乘以  $k$ . 2.  $2\ x\ y\ k$ : 将区间  $[x, y]$  中的数每个加上  $k$ . 3.  $3\ x\ y$ : 输出区间  $[x, y]$  中数的和. (对  $p$  取模)

```

1 // Problem: P3373 【模板】线段树 2
2
3 struct Info {
4     LL sum = 0;
5
6     Info(LL _sum = 0) : sum(_sum) {}
7
8     Info operator+(const Info& b) const { return Info(add(sum + b.sum)); }
9 };
10
11 struct Tag {
12     LL add = 0, mul = 1;
13
14     Tag(LL _add = 0, LL _mul = 1) : add(_add), mul(_mul) {}
15
16     bool operator==(const Tag& b) const { return add == b.add and mul == b.mul; }
17 };
18
19 void infoApply(Info& a, int l, int r, const Tag& tag) {
20     a.sum = add(mul(a.sum, tag.mul), mul((r - l + 1), tag.add));
21 }
22
23 void tagApply(Tag& a, int l, int r, const Tag& tag) {
24     a.add = add(mul(a.add, tag.mul), tag.add);
25     a.mul = mul(a.mul, tag.mul);
26 }
27
28 template <class Info, class Tag>
29 class segTree {
30 #define ls i << 1
31 #define rs i << 1 | 1
32 #define mid ((l + r) >> 1)
33 #define lson ls, l, mid
34 #define rson rs, mid + 1, r
35
36     int n;
37     std::vector<Info> info;
38     std::vector<Tag> tag;
39
40 public:
41     segTree(const std::vector<Info>& init) : n(init.size() - 1) {
42         assert(n > 0);
43         info.resize(4 << std::lg(n));
44         tag.resize(4 << std::lg(n));
45         auto build = [&](auto dfs, int i, int l, int r) {
46             if (l == r) {
47                 info[i] = init[l];
48                 return;
49             }
50             dfs(dfs, lson);
51             dfs(dfs, rson);
52             push_up(i);
53         };
54         build(build, 1, 1, n);
55     }
56
57 private:
58     void push_up(int i) { info[i] = info[ls] + info[rs]; }
59
60
61     template <class... T>
62     void apply(int i, int l, int r, const T&... val) {
63         ::infoApply(info[i], l, r, val...);
64         ::tagApply(tag[i], l, r, val...);
65     }
66
67     void push_down(int i, int l, int r) {
68         if (tag[i] == Tag{}) return;
69         apply(lson, tag[i]);
70         apply(rson, tag[i]);
71         tag[i] = {};
72     }
73
74 public:
75     template <class... T>
76     void rangeMerge(int ql, int qr, const T&... val) {

```

```

78     auto dfs = [&](auto dfs, int i, int l, int r) {
79         if (qr < l or r < ql) return;
80         if (ql <= l and r <= qr) {
81             apply(i, l, r, val...);
82             return;
83         }
84         push_down(i, l, r);
85         dfs(dfs, lson);
86         dfs(dfs, rson);
87         push_up(i);
88     };
89     dfs(dfs, 1, 1, n);
90 }
91
92 Info rangeQuery(int ql, int qr) {
93     Info res{};
94     auto dfs = [&](auto dfs, int i, int l, int r) {
95         if (qr < l or r < ql) return;
96         if (ql <= l and r <= qr) {
97             res = res + info[i];
98             return;
99         }
100        push_down(i, l, r);
101        dfs(dfs, lson);
102        dfs(dfs, rson);
103    };
104    dfs(dfs, 1, 1, n);
105    return res;
106 }
107
108 #undef rson
109 #undef lson
110 #undef mid
111 #undef rs
112 #undef ls
113 };
114
115 int main() {
116     std::ios::sync_with_stdio(false);
117     std::cin.tie(0);
118     std::cout.tie(0);
119
120     int n, m, p;
121     std::cin >> n >> m >> p;
122     std::vector<Info> a(n + 1);
123     for (int i = 1; i <= n; i++) std::cin >> a[i].sum;
124     static segTree<Info, Tag> tr(a);
125
126     while (m--) {
127         int op, k, l, r;
128         std::cin >> op >> l >> r;
129         if (op == 1) {
130             std::cin >> k;
131             tr.rangeMerge(l, r, Tag(0, k));
132         } else if (op == 2) {
133             std::cin >> k;
134             tr.rangeMerge(l, r, Tag(k, 1));
135         } else {
136             std::cout << tr.rangeQuery(l, r).sum << '\n';
137         }
138     }
139
140     return 0;
141 }

```

### 动态开点权值线段树

如果实现 push up 记得先开点再 push.

```

1 // Problem: 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 };

```

### (权值) 线段树合并

首先村落里的一共有  $n$  座房屋, 并形成一个树状结构. 然后救济粮分  $m$  次发放, 每次选择两个房屋  $(x, y)$ , 然后对于  $x$  到  $y$  的路径上每座房子里发放一袋  $z$  类型的救济粮. 查询所有的救济粮发放完毕后, 每座房子里存放的最多的是哪种救济粮.

```

1 // Problem: 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;
43             tree[u].ans = tree[tree[u].rs].ans;
44         } else if (!tree[u].rs) {
45             tree[u].cnt = tree[tree[u].ls].cnt;
46             tree[u].ans = tree[tree[u].ls].ans;
47         } else {
48             if (tree[tree[u].rs].cnt > tree[tree[u].ls].cnt) {
49                 tree[u].cnt = tree[tree[u].rs].cnt;
50                 tree[u].ans = tree[tree[u].rs].ans;
51             } else {
52                 tree[u].cnt = tree[tree[u].ls].cnt;
53                 tree[u].ans = tree[tree[u].ls].ans;
54             }
55         }
56     };
57
58     std::function<void(int, int, int, int, int)> modify = [&](int u, int l, int r, int x, int k) {
59         if (l == r) {
60             tree[u].cnt += k;
61             tree[u].ans = l;
62             return;
63         }
64         int mid = (l + r) >> 1;

```

```

65     if (x <= mid) {
66         if (!tree[u].ls) tree[u].ls = new_node(l, mid);
67         modify(tree[u].ls, l, mid, x, k);
68     } else {
69         if (!tree[u].rs) tree[u].rs = new_node(mid + 1, r);
70         modify(tree[u].rs, mid + 1, r, x, k);
71     }
72     push_up(u);
73 };
74
75 std::function<int(int, int, int, int)> merge = [&](int u, int v, int l, int r) -> int {
76     /* v 的信息传递给 u */
77     if (!u) return v;
78     if (!v) return u;
79     if (l == r) {
80         tree[u].cnt += tree[v].cnt;
81         return u;
82     }
83     int mid = (l + r) >> 1;
84     tree[u].ls = merge(tree[u].ls, tree[v].ls, l, mid);
85     tree[u].rs = merge(tree[u].rs, tree[v].rs, mid + 1, r);
86     push_up(u);
87     return u;
88 };
89
90 /* LCA */
91
92 for (int i = 1; i <= n; i++) {
93     rt[i] = idx;
94     new_node(1, 100000);
95 }
96
97 for (int i = 1; i <= m; i++) {
98     int u, v, w;
99     std::cin >> u >> v >> w;
100     int lca = LCA(u, v);
101     modify(rt[u], 1, 100000, w, 1);
102     modify(rt[v], 1, 100000, w, 1);
103     modify(rt[lca], 1, 100000, w, -1);
104     if (father[lca][0]) {
105         modify(rt[father[lca][0]], 1, 100000, w, -1);
106     }
107 }
108
109 /* dfs */
110 std::function<void(int, int)> Dfs = [&](int u, int fa) {
111     for (auto v : e[u]) {
112         if (v == fa) continue;
113         Dfs(v, u);
114         merge(rt[u], rt[v], 1, 100000);
115     }
116     ans[u] = tree[rt[u]].ans;
117     if (tree[rt[u]].cnt == 0) ans[u] = 0;
118 };
119
120 Dfs(1, 0);
121
122 for (int i = 1; i <= n; i++) {
123     std::cout << ans[i] << '\n';
124 }
125
126 return 0;
127 }

```

### 3.7 hjt segment tree

#### 第 1 个例题

$n$  个数,  $m$  次操作, 操作分别为

1.  $v_i$  1  $loc_i$   $value_i$ : 将第  $v_i$  个版本的  $a[loc_i]$  修改为  $value_i$ ,
2.  $v_i$  2  $loc_i$ : 拷贝第  $v_i$  个版本, 并查询该版本的  $a[loc_i]$ .

```

1 // 洛谷 P3919 【模板】可持久化线段树 1 (可持久化数组)
2
3 struct node {

```



```

4   int l, r, key;
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);
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) << '\n';
77          }
78      }
79
80      return 0;
81  }

```

## 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    for (int i = 1; i <= n; i++) {
68        root[i] = modify(root[i - 1], 1, v.size(), find(a[i]));
69    }
70    for (int i = 1; i <= m; i++) {
71        int l, r, k;
72        std::cin >> l >> r >> k;
73        std::cout << v[query(root[r], root[l - 1], 1, v.size(), k) - 1] << '\n';
74    }
75
76    return 0;
77 }
78

```

## 3.8 treap

### 旋转 treap

$n$  次操作, 操作分为如下 6 种:

1. 插入数  $x$ ,
2. 删除数  $x$  (若有多个相同的数, 只删除一个),
3. 查询数  $x$  的排名 (排名定义为小于  $x$  的数的个数 + 1),
4. 查询排名为  $x$  的数,
5. 求  $x$  的前驱 (前驱定义为小于  $x$  的最大数),
6. 求  $x$  的后继 (后继定义为大于  $x$  的最小数).

```

1 // Problem: 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) << '\n';
128         } else if (op == 4) {
129             cout << get_key_by_rank(root, x + 1) << '\n';
130         } else if (op == 5) {
131             cout << get_prev(root, x) << '\n';
132         } else {
133             cout << get_next(root, x) << '\n';
134         }
135     }
136     return 0;
137 }

```

## 无旋 treap

与旋转 Treap 同一个题目

```

1 struct node {
2     node *ch[2];
3     int key, val;
4     int cnt, size;
5
6     node(int _key) : key(_key), cnt(1), size(1) {
7         ch[0] = ch[1] = nullptr;
8         val = rand();
9     }
10
11     // node(node *_node) {
12     //     key = _node->key, val = _node->val, cnt = _node->cnt, size = _node->size;
13     // }
14
15     inline void push_up() {
16         size = cnt;
17         if (ch[0] != nullptr) size += ch[0]->size;
18         if (ch[1] != nullptr) size += ch[1]->size;
19     }
20 };
21
22 struct treap {
23     #define _2 second.first
24     #define _3 second.second
25
26     node *root;
27
28     pair<node *, node *> split(node *p, int key) {
29         if (p == nullptr) return {nullptr, nullptr};
30         if (p->key <= key) {
31             auto temp = split(p->ch[1], key);
32             p->ch[1] = temp.first;
33             p->push_up();
34             return {p, temp.second};
35         } else {
36             auto temp = split(p->ch[0], key);
37             p->ch[0] = temp.second;
38             p->push_up();
39             return {temp.first, p};
40         }
41     }
42
43     pair<node *, pair<node *, node *> > split_by_rank(node *p, int rank) {
44         if (p == nullptr) return {nullptr, {nullptr, nullptr}};
45         int ls_size = p->ch[0] == nullptr ? 0 : p->ch[0]->size;
46         if (rank <= ls_size) {
47             auto temp = split_by_rank(p->ch[0], rank);
48             p->ch[0] = temp._3;
49             p->push_up();
50             return {temp.first, {temp._2, p}};
51         } else if (rank <= ls_size + p->cnt) {
52             node *lt = p->ch[0];
53             node *rt = p->ch[1];
54             p->ch[0] = p->ch[1] = nullptr;
55             return {lt, {p, rt}};
56         } else {
57             auto temp = split_by_rank(p->ch[1], rank - ls_size - p->cnt);
58             p->ch[1] = temp.first;
59             p->push_up();
60             return {p, {temp._2, temp._3}};
61         }
62     }
63
64     node *merge(node *u, node *v) {
65         if (u == nullptr && v == nullptr) return nullptr;
66         if (u != nullptr && v == nullptr) return u;
67         if (v != nullptr && u == nullptr) return v;
68         if (u->val < v->val) {
69             u->ch[1] = merge(u->ch[1], v);
70             u->push_up();
71             return u;
72         } else {
73             v->ch[0] = merge(u, v->ch[0]);
74             v->push_up();
75             return v;
76         }
77     }
78
79     void insert(int key) {
80         auto temp = split(root, key);
81         auto l_tr = split(temp.first, key - 1);
82         node *new_node;
83         if (l_tr.second == nullptr) {
84             new_node = new node(key);
85         } else {
86             l_tr.second->cnt++;

```

```

87         l_tr.second->push_up();
88     }
89     node *l_tr_combined = merge(l_tr.first, l_tr.second == nullptr ? new_node : l_tr.second);
90     root = merge(l_tr_combined, temp.second);
91 }
92
93 void remove(int key) {
94     auto temp = split(root, key);
95     auto l_tr = split(temp.first, key - 1);
96     if (l_tr.second->cnt > 1) {
97         l_tr.second->cnt--;
98         l_tr.second->push_up();
99         l_tr.first = merge(l_tr.first, l_tr.second);
100     } else {
101         if (temp.first == l_tr.second) temp.first = nullptr;
102         delete l_tr.second;
103         l_tr.second = nullptr;
104     }
105     root = merge(l_tr.first, temp.second);
106 }
107
108 int get_rank_by_key(node *p, int key) {
109     auto temp = split(p, key - 1);
110     int ret = (temp.first == nullptr ? 0 : temp.first->size) + 1;
111     root = merge(temp.first, temp.second);
112     return ret;
113 }
114
115 int get_key_by_rank(node *p, int rank) {
116     auto temp = split_by_rank(p, rank);
117     int ret = temp._2->key;
118     root = merge(temp.first, merge(temp._2, temp._3));
119     return ret;
120 }
121
122 int get_prev(int key) {
123     auto temp = split(root, key - 1);
124     int ret = get_key_by_rank(temp.first, temp.first->size);
125     root = merge(temp.first, temp.second);
126     return ret;
127 }
128
129 int get_nex(int key) {
130     auto temp = split(root, key);
131     int ret = get_key_by_rank(temp.second, 1);
132     root = merge(temp.first, temp.second);
133     return ret;
134 }
135 };
136
137 treap tr;
138
139 int main() {
140     ios::sync_with_stdio(false);
141     cin.tie(0);
142     cout.tie(0);
143
144     srand(time(0));
145
146     int n;
147     cin >> n;
148     while (n-- > 0) {
149         int op, x;
150         cin >> op >> x;
151         if (op == 1) {
152             tr.insert(x);
153         } else if (op == 2) {
154             tr.remove(x);
155         } else if (op == 3) {
156             cout << tr.get_rank_by_key(tr.root, x) << '\n';
157         } else if (op == 4) {
158             cout << tr.get_key_by_rank(tr.root, x) << '\n';
159         } else if (op == 5) {
160             cout << tr.get_prev(x) << '\n';
161         } else {
162             cout << tr.get_nex(x) << '\n';
163         }
164     }
165     return 0;
166 }

```

## 用 01 trie 实现的一种方式

同样的题目, 注意使用 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 << '\n';
73         } else if (op == 4) {
74             std::cout << treap.get_key_by_rank(x) - num << '\n';
75         } else if (op == 5) {
76             std::cout << treap.get_prev(x + num) - num << '\n';
77         } else if (op == 6) {
78             std::cout << treap.get_next(x + num) - num << '\n';
79         }
80     }
81 }

```

```

80     }
81     return 0;
82 }

```

### 3.9 splay

#### 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[f].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, 1);
98         splay.tr[splay.tr[r].ch[0]].flag ^= 1;
99     }
100     splay.output(splay.root);
101
102     return 0;
103 }

```

### 普通平衡树

$n$  次操作, 操作分为如下 6 种:

1. 插入数  $x$
2. 删除数  $x$  (若有多个相同的数, 只删除一个)
3. 查询数  $x$  的排名 (排名定义为小于  $x$  的数的个数 + 1)
4. 查询排名为  $x$  的数
5. 求  $x$  的前驱 (前驱定义为小于  $x$  的最大数)
6. 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
3 struct node {
4     int ch[2], fa, key, siz, cnt;
5
6     void init(int _fa, int _key) { fa = _fa, key = _key, siz = cnt = 1; }
7
8     void clear() { ch[0] = ch[1] = fa = key = siz = cnt = 0; }
9 };
10
11 struct splay {
12     node tr[N];
13     int n, root, idx;
14
15     bool get(int u) { return u == tr[tr[u].fa].ch[1]; }
16
17     void pushup(int u) { tr[u].siz = tr[tr[u].ch[0]].siz + tr[tr[u].ch[1]].siz + tr[u].cnt; }
18
19     void rotate(int x) {
20         int y = tr[x].fa, z = tr[y].fa;
21         int op = get(x);
22         tr[y].ch[op] = tr[x].ch[op ^ 1];
23         if (tr[x].ch[op ^ 1]) tr[tr[x].ch[op ^ 1]].fa = y;
24         tr[x].ch[op ^ 1] = y;
25         tr[y].fa = x, tr[x].fa = z;
26         if (z) tr[z].ch[y == tr[z].ch[1]] = x;
27         pushup(y), pushup(x);
28     }
29
30     void opt(int u, int k) {
31         for (int f = tr[u].fa; f = tr[u].fa, f != k; rotate(u)) {
32             if (tr[f].fa != k) {
33                 rotate(get(u) == get(f) ? f : u);
34             }
35         }
36         if (k == 0) root = u;
37     }
38
39     void insert(int key) {

```

```

40     if (!root) {
41         idx++;
42         tr[idx].init(0, key);
43         root = idx;
44         return;
45     }
46     int u = root, f = 0;
47     while (1) {
48         if (tr[u].key == key) {
49             tr[u].cnt++;
50             pushup(u), pushup(f);
51             opt(u, 0);
52             break;
53         }
54         f = u, u = tr[u].ch[tr[u].key < key];
55         if (!u) {
56             idx++;
57             tr[idx].init(f, key);
58             tr[f].ch[tr[f].key < key] = idx;
59             pushup(idx), pushup(f);
60             opt(idx, 0);
61             break;
62         }
63     }
64 }
65
66 // 返回节点编号 //
67 int kth(int rank) {
68     int u = root;
69     while (1) {
70         if (tr[u].ch[0] && rank <= tr[tr[u].ch[0]].siz) {
71             u = tr[u].ch[0];
72         } else {
73             rank -= tr[tr[u].ch[0]].siz + tr[u].cnt;
74             if (rank <= 0) {
75                 opt(u, 0);
76                 return u;
77             } else {
78                 u = tr[u].ch[1];
79             }
80         }
81     }
82 }
83
84 // 返回排名 //
85 int nlt(int key) {
86     int rank = 0, u = root;
87     while (1) {
88         if (tr[u].key > key) {
89             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 } splay;
130
131 int n, op, x;
132
133 int main() {
134     std::ios::sync_with_stdio(false);
135     std::cin.tie(0);
136     std::cout.tie(0);
137
138     splay.insert(-inf), splay.insert(inf);
139
140     std::cin >> n;
141     for (int i = 1; i <= n; i++) {
142         std::cin >> op >> x;
143         if (op == 1) {
144             splay.insert(x);
145         } else if (op == 2) {
146             splay.remove(x);
147         } else if (op == 3) {
148             std::cout << splay.nlt(x) - 1 << endl;
149         } else if (op == 4) {
150             std::cout << splay.tr[splay.kth(x + 1)].key << endl;
151         } else if (op == 5) {
152             std::cout << splay.tr[splay.get_prev(x)].key << endl;
153         } else if (op == 6) {
154             std::cout << splay.tr[splay.get_next(x)].key << endl;
155         }
156     }
157 }
158
159 return 0;
160 }

```

### 3.10 tree in tree

#### 线段树套线段树

$n$  个三维数对  $(a_i, b_i, c_i)$ , 设  $f(i)$  表示  $a_j \leq a_i$  且  $b_j \leq b_i$  且  $c_j \leq c_i$  且  $i \neq j$  的个数. 输出  $f(i)$  ( $0 \leq i \leq n-1$ ) 的值.

```

1 // 洛谷 P3810 【模板】三维偏序（陌上花开）
2
3 struct node1 {
4     int l, r, root;
5 } tr1[N << 2];
6
7 struct node2 {
8     int ch[2], cnt;
9 } tr2[N << 7];
10
11 struct node {
12     int x, y, z, cnt;
13
14     bool operator==(const node& a) { return (x == a.x && y == a.y && z == a.z); }
15 } data[N];
16
17 bool cmp(node a, node b) {
18     if (a.x != b.x) return a.x < b.x;
19     if (a.y != b.y) return a.y < b.y;
20     return a.z < b.z;
21 }
22
23 int root_tot, n, m, ans[N], anss[N];
24
25 void build(int u, int l, int r) {
26     tr1[u].l = l, tr1[u].r = r;
27     if (l != r) {
28         int mid = (l + r) >> 1;
29         build(u << 1, l, mid);
30         build(u << 1 | 1, mid + 1, r);
31     }
32 }
33
34 void modify_2(int& u, int l, int r, int pos) {
35     if (u == 0) u = ++root_tot;
36     tr2[u].cnt++;
37     if (l == r) return;

```

```

39     int mid = (l + r) >> 1;
40     if (pos <= mid) {
41         modify_2(tr2[u].ch[0], 1, mid, pos);
42     } else {
43         modify_2(tr2[u].ch[1], mid + 1, r, pos);
44     }
45 }
46
47 int query_2(int& u, int l, int r, int x, int y) {
48     if (u == 0) return 0;
49     if (x <= l && r <= y) return tr2[u].cnt;
50     int mid = (l + r) >> 1, ans = 0;
51     if (x <= mid) ans += query_2(tr2[u].ch[0], 1, mid, x, y);
52     if (mid < y) ans += query_2(tr2[u].ch[1], mid + 1, r, x, y);
53     return ans;
54 }
55
56 void modify_1(int u, int l, int r, int t) {
57     modify_2(tr1[u].root, 1, m, data[t].z);
58     if (l == r) return;
59     int mid = (l + r) >> 1;
60     if (data[t].y <= mid) {
61         modify_1(u << 1, l, mid, t);
62     } else {
63         modify_1(u << 1 | 1, mid + 1, r, t);
64     }
65 }
66
67 int query_1(int u, int l, int r, int t) {
68     if (l <= 1 && r <= data[t].y) return query_2(tr1[u].root, 1, m, 1, data[t].z);
69     int mid = (l + r) >> 1, ans = 0;
70     if (l <= mid) ans += query_1(u << 1, l, mid, t);
71     if (mid < data[t].y) ans += query_1(u << 1 | 1, mid + 1, r, t);
72     return ans;
73 }
74
75 int main() {
76     std::ios::sync_with_stdio(false);
77     std::cin.tie(0);
78     std::cout.tie(0);
79
80     std::cin >> n >> m;
81     rep(i, 1, n) {
82         int x, y, z;
83         std::cin >> x >> y >> z;
84         data[i] = {x, y, z};
85     }
86     std::sort(data + 1, data + n + 1, cmp);
87     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 }

```

## 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:

1.  $l\ r\ k$ : 询问区间  $[l \sim r]$  中数  $k$  的排名.
2.  $l\ r\ k$ : 询问区间  $[l \sim r]$  中排名为  $k$  的数.
3.  $pos\ k$ : 将序列中  $pos$  位置的数修改为  $k$ .
4.  $l\ r\ k$ : 询问区间  $[l \sim r]$  中数  $k$  的前驱.
5.  $l\ r\ k$ : 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;
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     std::ios::sync_with_stdio(false);
206     std::cin.tie(0);
207     std::cout.tie(0);
208
209     srand(time(0));
210
211     std::cin >> n >> m;
212     build(1, 1, n);
213     rep(i, 1, n){
214         std::cin >> a[i];
215         modify(1, i, a[i]);
216     }
217     rep(i, 1, root_tot){
218         splay.insert(tr1[i].root, inf), splay.insert(tr1[i].root, -inf);
219     }
220     rep(i, 1, m){
221         std::cin >> op;
222         if(op == 1){
223             std::cin >> l >> r >> key;
224             std::cout << get_rank_by_key_in_interval(1, l, r, key) + 1 << endl;
225         }
226         else if(op == 2){
227             std::cin >> l >> r >> key;
228             std::cout << get_key_by_rank_in_interval(1, l, r, key) << endl;
229         }
230         else if(op == 3){
231             std::cin >> pos >> key;
232             change(1, pos, a[pos], key);
233             a[pos] = key;
234         }
235         else if(op == 4){
236             std::cin >> l >> r >> key;
237             std::cout << get_prev_in_interval(1, l, r, key) << endl;
238         }
239         else if(op == 5){
240             std::cin >> l >> r >> key;
241             std::cout << get_next_in_interval(1, l, r, key) << endl;
242         }
243     }
244 }
245
246 return 0;
247 }

```

然而洛谷, ACwing 能过, Loj T 一堆。

## 4 string

### 4.1 kmp

```

1 auto get_next = [&](const std::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 };

```

### 4.2 z function

```

1 auto z_function = [&](const std::string& s) -> vi {
2     int n = s.size();

```

```

3   vi z(n);
4   for (int i = 1, l = 0, r = 0; i < n; i++) {
5       if (i <= r and z[i - 1] < r - i + 1) {
6           z[i] = z[i - 1];
7       } else {
8           z[i] = std::max(0, r - i + 1);
9           while (z[i] + i < n and s[z[i]] == s[z[i] + i]) z[i]++;
10        }
11        if (z[i] + i - 1 > r) {
12            l = i;
13            r = z[i] + i - 1;
14        }
15    }
16    return z;
17 };

```

### 4.3 trie

#### 普通字典树 (单词匹配)

```

1   int cnt;
2   std::vector<std::array<int, 26>> trie(n + 1);
3   vi exist(n + 1);
4
5   auto insert = [&](const std::string& s) -> void {
6       int p = 0;
7       for (const auto ch : s) {
8           int c = ch - 'a';
9           if (!trie[p][c]) trie[p][c] = ++cnt;
10          p = trie[p][c];
11      }
12      exist[p] = true;
13  };
14
15  auto find = [&](const string& s) -> bool {
16      int p = 0;
17      for (const auto ch : s) {
18          int c = ch - 'a';
19          if (!trie[p][c]) return false;
20          p = trie[p][c];
21      }
22      return exist[p];
23  };

```

#### 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.

```

1   // trie //
2   int cnt = 0;
3   std::vector<std::array<int, 2>> trie(N);
4
5   auto insert = [&](int x) -> void {
6       int p = 0;
7       for (int i = 30; i >= 0; i--) {
8           int c = (x >> i) & 1;
9           if (!trie[p][c]) trie[p][c] = ++cnt;
10          p = trie[p][c];
11      }
12  };
13
14  auto find = [&](int x) -> int {
15      int sum = 0, p = 0;
16      for (int i = 30; i >= 0; i--) {
17          int c = (x >> i) & 1;
18          if (trie[p][c ^ 1]) {
19              p = trie[p][c ^ 1];
20              sum += (1 << i);
21          } else {
22              p = trie[p][c];
23          }
24      }
25      return sum;
26  };

```

## 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.

```

1  int main() {
2      std::ios::sync_with_stdio(false);
3      std::cin.tie(0);
4      std::cout.tie(0);
5
6      int n, m;
7      std::cin >> n;
8      vi w(n + 1);
9      for (int i = 1; i <= n; i++) {
10         std::cin >> w[i];
11     }
12
13     vvi e(n + 1);
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
21     /* 离线询问 */
22     std::cin >> m;
23     std::vector<vpi> q(n + 1);
24     vi ans(m + 1);
25     for (int i = 1; i <= m; i++) {
26         int u, v;
27         std::cin >> u >> v;
28         q[v].emplace_back(u, i);
29     }
30
31     /* 01 trie */
32     std::vector<std::array<int, 2>> tr(1);
33
34     auto new_node = [&]() -> int {
35         tr.emplace_back();
36         return tr.size() - 1;
37     };
38
39     vi id(n + 1);
40
41     auto insert = [&](int root, int x) {
42         int p = root;
43         for (int i = 29; i >= 0; i--) {
44             int c = x >> i & 1;
45             if (!tr[p][c]) tr[p][c] = new_node();
46             p = tr[p][c];
47         }
48     };
49
50     auto query = [&](int root, int x) -> int {
51         int ans = 0, p = root;
52         for (int i = 29; i >= 0; i--) {
53             int c = x >> i & 1;
54             if (tr[p][c ^ 1]) {
55                 p = tr[p][c ^ 1];
56                 ans += (1 << i);
57             } else {
58                 p = tr[p][c];
59             }
60         }
61         return ans;
62     };
63
64     std::function<int(int, int)> merge = [&](int a, int b) -> int {
65         // b 的信息挪到 a 上 //
66         if (!a) return b;
67         if (!b) return a;
68         tr[a][0] = merge(tr[a][0], tr[b][0]);
69         tr[a][1] = merge(tr[a][1], tr[b][1]);
70         return a;
71     };
72
73     std::function<void(int, int)> dfs = [&](int u, int fa) {
74         id[u] = new_node();
75         insert(id[u], w[u]);
76         for (auto v : e[u]) {
77             if (v == fa) continue;

```

```
78         dfs(v, u);
79         id[u] = merge(id[u], id[v]);
80     }
81     for (auto [v, i] : q[u]) {
82         ans[i] = query(id[u], w[v]);
83     }
84 };
85 dfs(1, 0);
86
87 for (int i = 1; i <= m; i++) std::cout << ans[i] << endl;
88
89 return 0;
90 }
```

## 5 math - number theory

### 5.1 Eculid

#### 欧几里得算法

```
1 std::gcd(a, b)
```

#### 扩展欧几里得算法

```
1 auto exgcd = [&](LL a, LL b, LL& x, LL& y) {
2     LL x1 = 1, x2 = 0, x3 = 0, x4 = 1;
3     while (b != 0) {
4         LL c = a / b;
5         std::tie(x1, x2, x3, x4, a, b) =
6             std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
7     }
8     x = x1, y = x2;
9 };
```

```
1 auto exgcd = [&](auto&& self, LL a, LL b, LL& x, LL& y) {
2     if (!b) {
3         x = 1, y = 0;
4         return;
5     }
6     self(self, b, a % b, y, x);
7     y -= a / b * x;
8 };
```

```
1 auto exgcd = [&](auto&& self, LL a, LL b, LL& x, LL& y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     LL d = self(self, b, a % b, y, x);
7     y -= a / b * x;
8     return d;
9 };
```

#### 类欧几里得算法

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

$f(a, b, c, n)$  可以单独求.

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

```
1 LL f(LL a, LL b, LL c, LL n) {
2     if (a == 0) return ((b / c) * (n + 1));
3     if (a >= c || b >= c)
4         return f(a % c, b % c, c, n) + (a / c) * n * (n + 1) / 2 + (b / c) * (n + 1);
5     LL m = (a * n + b) / c;
6     LL v = f(c, c - b - 1, a, m - 1);
7     return n * m - v;
8 }
```

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

直接记吧.

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

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

```

1  const int inv2 = 499122177;
2  const int inv6 = 166374059;
3
4  LL f(LL a, LL b, LL c, LL n);
5  LL g(LL a, LL b, LL c, LL n);
6  LL h(LL a, LL b, LL c, LL n);
7
8  struct data {
9      LL f, g, h;
10 };
11
12 data calc(LL a, LL b, LL c, LL n) {
13     LL ac = a / c, bc = b / c, m = (a * n + b) / c, n1 = n + 1, n21 = n * 2 + 1;
14     data d;
15     if (a == 0) {
16         d.f = bc * n1 % mod;
17         d.g = bc * n % mod * n1 % mod * inv2 % mod;
18         d.h = bc * bc % mod * n1 % mod;
19         return d;
20     }
21     if (a >= c || b >= c) {
22         d.f = n * n1 % mod * inv2 % mod * ac % mod + bc * n1 % mod;
23         d.g =
24             ac * n % mod * n1 % mod * n21 % mod * inv6 % mod + bc * n % mod * n1 % mod * inv2 % mod;
25         d.h = ac * ac % mod * n % mod * n1 % mod * n21 % mod * inv6 % mod +
26             bc * bc % mod * n1 % mod + ac * bc % mod * n % mod * n1 % mod;
27         d.f %= mod, d.g %= mod, d.h %= mod;
28         data e = calc(a % c, b % c, c, n);
29         d.h += e.h + 2 * bc % mod * e.f % mod + 2 * ac % mod * e.g % mod;
30         d.g += e.g, d.f += e.f;
31         d.f %= mod, d.g %= mod, d.h %= mod;
32         return d;
33     }
34     data e = calc(c, c - b - 1, a, m - 1);
35     d.f = n * m % mod - e.f, d.f = (d.f % mod + mod) % mod;
36     d.g = m * n % mod * n1 % mod - e.h - e.f, d.g = (d.g * inv2 % mod + mod) % mod;
37     d.h = n * m % mod * (m + 1) % mod - 2 * e.g - 2 * e.f - d.f;
38     d.h = (d.h % mod + mod) % mod;
39     return d;
40 }

```

## 5.2 inverse

### 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) {
3      inv[1] = 1;
4      for (int i = 2; i <= n; i++) {
5          inv[i] = 1ll * (p - p / i) * inv[p % i] % p;
6      }
7  };

```

### 求 $n$ 个数的逆元

```

1  auto get_inv = [&](const vi& a) {
2      int n = a.size();
3      vi b(n), f(n), ivf(n);
4      f[0] = a[0];
5      for (int i = 1; i < n; i++) {
6          f[i] = 1ll * f[i - 1] * a[i] % p;
7      }
8      ivf.back() = quick_power(f.back(), p - 2, p);
9      for (int i = n - 1; i; i--) {
10         ivf[i - 1] = 1ll * ivf[i] * a[i] % p;
11     }
12     b[0] = ivf[0];
13     for (int i = 1; i < n; i++) {
14         b[i] = 1ll * ivf[i] * f[i - 1] % p;
15     }
16     return b;
17 };

```

## 5.3 sieve

### 素数

```

1 vi prime, is_prime(n + 1, 1);
2 auto Euler_sieve = [&](int n){
3     for (int i = 2; i <= n; i++) {
4         if (is_prime[i]) prime.push_back(i);
5         for (auto p : prime) {
6             if (i * p > n) break;
7             is_prime[i * p] = 0;
8             if (i % p == 0) break;
9         }
10    }
11 };

```

### 欧拉函数

```

1 vi phi(n + 1), prime;
2 vi is_prime(n + 1, 1);
3 auto get_phi = [&](int n) {
4     int cnt = 0;
5     phi[1] = 1;
6     for (int i = 2; i <= n; i++) {
7         if (is_prime[i]) {
8             prime.push_back(i);
9             phi[i] = i - 1;
10        }
11        for (auto p : prime) {
12            if (i * p > n) break;
13            is_prime[i * p] = 0;
14            if (i % p) {
15                phi[i * p] = phi[i] * phi[p];
16            } else {
17                phi[i * p] = phi[i] * p;
18                break;
19            }
20        }
21    }
22 };

```

### 约数和

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

```

1 vi g(n + 1), d(n + 1), prime;
2 vi is_prime(n + 1, 1);
3 auto get_d = [&](int n) {
4     int tot = 0;
5     g[1] = d[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] = 0;
14            if (i % p == 0) {
15                g[i * p] = g[i] * p + 1;
16                d[i * p] = d[i] / g[i] * g[i * p];
17                break;
18            } else {
19                d[i * p] = d[i] * d[p];
20                g[i * p] = 1 + p;
21            }
22        }
23    }
24 };

```

## 莫比乌斯函数

```

1 vi mu(n + 1), prime;
2 vi is_prime(n + 1, 1);
3 auto get_mu = [&](int n) {
4     mu[1] = 1;
5     for (int i = 2; i <= n; i++) {
6         if (is_prime[i]) {
7             prime.push_back(i);
8             mu[i] = -1;
9         }
10        for (auto p : prime) {
11            if (i * p > n) break;
12            is_prime[i * p] = 0;
13            if (i % p == 0) {
14                mu[i * p] = 0;
15                break;
16            }
17            mu[i * p] = -mu[i];
18        }
19    }
20 };

```

## 杜教筛

```

1 const int N = 1e7;
2 vi mu(N + 1), phi(N + 1), prime;
3 vl sum_phi(N + 1), sum_mu(N + 1);
4 vi is_prime(N + 1, 1);
5 std::map<LL, LL> mp_mu;
6
7 /* 计算 1 ~ 10^7 的 mu */
8 auto get_mu = [&](int n) {
9     phi[1] = mu[1] = 1;
10    for (int i = 2; i <= n; i++) {
11        if (is_prime[i]) {
12            prime.push_back(i);
13            phi[i] = i - 1;
14            mu[i] = -1;
15        }
16        for (auto p : prime) {
17            if (i * p > n) break;
18            is_prime[i * p] = 0;
19            if (i % p == 0) {
20                phi[i * p] = phi[i] * p;
21                mu[i * p] = 0;
22                break;
23            }
24            phi[i * p] = phi[i] * phi[p];
25            mu[i * p] = -mu[i];
26        }
27    }
28 };
29 get_mu(N);
30 for (int i = 1; i <= N; i++) {
31     sum_phi[i] = sum_phi[i - 1] + phi[i];
32     sum_mu[i] = sum_mu[i - 1] + mu[i];
33 }
34
35 /* 杜教筛：求 mu 的前缀和 */
36 std::function<LL(LL)> S_mu = [&](LL x) -> LL {
37     if (x <= N) return sum_mu[x];
38     auto it = mp_mu.find(x);
39     if (it != mp_mu.end()) return mp_mu[x];
40     LL ans = 1;
41     for (LL i = 2, j; i <= x; i = j + 1) {
42         j = x / (x / i);
43         ans -= S_mu(x / i) * (j - i + 1);
44     }
45     return mp_mu[x] = ans;
46 };
47
48 /* 杜教筛：求 phi 的前缀和 */
49 auto S_phi = [&](LL x) -> LL {
50     if (x <= N) return sum_phi[x];
51     LL ans = 0;
52     for (LL i = 1, j; i <= x; i = j + 1) {
53         j = x / (x / i);
54         ans += 1ll * (S_mu(j) - S_mu(i - 1)) * (x / i) * (x / i);
55     }
56     return (ans - 1) / 2 + 1;

```



57 };

## 5.4 block

### 分块的逻辑

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

```
1 for(int l = 1, r, k; l <= n; l = r + 1){
2     k = n / l;
3     r = n / (n / l);
4     debug(l, r, k);
5 }
```

$k = \lfloor \frac{n}{g} \rfloor$  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.

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

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

```
1 for(int l = 1, r, k; l < n; l = r + 1){
2     k = (n + l - 1) / l;
3     r = (n + k - 2) / (k - 1) - 1;
4     debug(l, r, k);
5 }
```

$k = \lceil \frac{n}{g} \rceil$  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.

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

### 一般形式

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

设  $s(i)$  为  $f(i)$  的前缀和。

```
1 for (int l = 1, r; l <= n; l = r + 1) {
2     r = n / (n / l);
3     ans += (s[r] - s[l - 1]) * (n / l);
4 }
```

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

```
1 for (int l = 1, r, r1, r2; l <= n; l = r + 1) {
2     if (a / l) {
3         r1 = a / (a / l);
4     } else {
5         r1 = n;
6     }
7     if (b / l) {
8         r2 = b / (b / l);
9     } else {
10        r2 = n;
11    }
12    r = min(min(r1, r2), n);
13    ans += (s[r] - s[l - 1]) * (a / l) * (b / l);
14 }
```

## 5.5 CRT & exCRT

求解

$$\begin{cases} N \equiv a_1 \pmod{m_1} \\ N \equiv a_2 \pmod{m_2} \\ \dots \\ N \equiv a_n \pmod{m_n} \end{cases}$$

有  $N \equiv \sum_{i=1}^k a_i \times \text{inv}\left(\frac{M}{m_i}, m_i\right) \times \left(\frac{M}{m_i}\right) \pmod{M}$

```

1 auto crt = [&](int n, const vi& a, const vi& m) -> LL{
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 };

```

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 BSGS & exBSGS

求解满足  $a^x \equiv b \pmod{p}$  的  $x$

```

1 /* return value = -1e18 means no solution */
2 auto BSGS = [&](LL a, LL b, LL p) {
3     if (1 % p == b % p) return 0ll;
4     LL k = std::sqrt(p) + 1;
5     std::unordered_map<LL, LL> hash;
6     for (LL i = 0, j = b % p; i < k; i++) {
7         hash[j] = i;
8         j = j * a % p;
9     }
10    LL ak = 1;
11    for (int i = 1; i <= k; i++) ak = ak * a % p;
12    for (int i = 1, j = ak; i <= k; i++) {
13        if (hash.count(j)) return 1ll * i * k - hash[j];
14        j = 1ll * j * ak % p;
15    }
16    return -INF;
17 };

```

$(a, p) \neq 1$  的情形

```

1 /* return value < 0 means no solution */
2 auto exBSGS = [&](auto&& self, LL a, LL b, LL p) {
3     b = (b % p + p) % p;
4     if (1ll % p == b % p) return 0ll;
5     LL x, y, d = std::gcd(a, p);
6     exgcd(exgcd, a, p, x, y);
7     if (d > 1) {
8         if (b % d != 0) return -INF;
9         exgcd(exgcd, a / d, p / d, x, y);
10        return self(self, a, b / d * x % (p / d), p / d) + 1;
11    }
12    return BSGS(a, b, p);
13 };

```

## 5.7 Miller Rabin

原理基于：对奇素数  $p$ ,  $a^2 \equiv 1 \pmod p$  的解为  $x \equiv 1 \pmod p$  或  $x \equiv p-1 \pmod p$ , 以及费马小定理.

随机一个底数  $x$ , 将  $a^{p-1} \pmod p$  的指数  $p-1$  分解为  $a \times 2^b$ , 计算出  $x^a$ , 之后进行最多  $b$  次平方操作, 若发现非平凡平方根时即可判断出其不是素数, 否则通过此轮测试.

`test_time` 为测试次数, 建议设为不小于 8 的整数以保证正确率, 但也不宜过大, 否则会影响效率.

```
1 auto miller_rabin = [&](LL n) -> bool {
2     if (n <= 3) return n == 2 || n == 3;
3     LL a = n - 1, b = 0;
4     while (!(a & 1)) a >>= 1, b++;
5     for (int i = 1, j; i <= 10; i++) { /* test time = 10 */
6         LL x = rand() % (n - 2) + 2, v = quick_power(x, a, n);
7         if (v == 1 || v == n - 1) continue;
8         for (j = 0; j < b; j++) {
9             if (v == n - 1) break;
10            v = (i128) v * v % n;
11        }
12        if (j >= b) return false;
13    }
14    return true;
15 };
```

事实上底数没必要随机 10 次, 检验如下数即可. 快速幂记得要 i128.

1. int 范围: 2, 7, 61.

2. LL 范围: 2, 325, 9375, 28178, 450775, 9780504, 1795265022.

```
1 vl vv = {2, 3, 5, 7, 11, 13, 17, 23, 29};
2 auto miller_rabin = [&](LL n) -> bool {
3     auto test = [&](LL n, int a) {
4         if (n == a) return true;
5         if (n % 2 == 0) return false;
6         LL d = (n - 1) >> __builtin_ctzll(n - 1);
7         LL r = quick_power(a, d, n);
8         while (d < n - 1 and r != 1 and r != n - 1) {
9             d <<= 1;
10            r = (i128) r * r % n;
11        }
12        return r == n - 1 or d & 1;
13    };
14    if (n == 2 or n == 3) return true;
15    for (auto p : vv) {
16        if (test(n, p) == 0) return false;
17    }
18    return true;
19 };
```

## 5.8 Pollard Rho

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

```
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 };
```

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 };

```

## 5.9 quadratic residu

Cipolla 算法

```

1 auto cipolla = [&](int x) {
2     std::srand(time(0));
3     auto check = [&](int x) -> bool { return pow(x, (mod - 1) / 2) == 1; };
4     if (!x) return 0;
5     if (!check(x)) return -1;
6     int a, b;
7     while (1) {
8         a = rand() % mod;
9         b = sub(mul(a, a), x);
10        if (!check(b)) break;
11    }
12    PII t = {a, 1};
13    PII ans = {1, 0};
14    auto mulp = [&](PII x, PII y) -> PII {
15        auto [x1, x2] = x;
16        auto [y1, y2] = y;
17        int c = add(mul(x1, y1), mul(x2, y2, b));
18        int d = add(mul(x1, y2), mul(x2, y1));
19        return {c, d};
20    };
21    for (int i = (mod + 1) / 2; i; i >>= 1) {
22        if (i & 1) ans = mulp(ans, t);
23        t = mulp(t, t);
24    }
25    return std::min(ans.ff, mod - ans.ff);
26 }

```

## 5.10 Lucas

卢卡斯定理

用于求大组合数, 并且模数是一个不大的素数.

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

$\binom{n \bmod p}{m \bmod p}$  可以直接计算,  $\binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor}$  可以继续使用卢卡斯计算.

递归至  $m = 0$  的时候, 返回 1.

$p$  不太大, 一般在  $10^5$  左右.

```

1 auto C = [&](LL n, LL m, LL p) -> LL {
2     if (n < m) return 0;
3     if (m == 0) return 1;
4     return fac[n] * inv_fac[m] % p * inv_fac[n - m] % p;
5 };
6
7 auto lucas = [&](auto&& self, LL n, LL m, LL p) -> LL {
8     if (n < m) return 0;
9     if (m == 0) return 1;
10    return C(n % p, m % p, p) * self(self, n / p, m / p, p) % p;
11 }

```

### 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:

$$\sum_{1 \leq j} \left\lfloor \frac{n}{p^j} \right\rfloor.$$

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

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

### 扩展卢卡斯定理

计算

$$\binom{n}{m} \bmod p,$$

$p$  可能为合数.

第一部分: CRT.

原问题变成求

$$\left\{ \begin{array}{l} \binom{n}{m} \equiv a_1 \bmod p_1^{\alpha_1} \\ \binom{n}{m} \equiv a_2 \bmod p_2^{\alpha_2} \\ \dots \\ \binom{n}{m} \equiv a_k \bmod p_k^{\alpha_k} \end{array} \right.$$

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

$$\binom{n}{m} \bmod q^k.$$

等价于

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

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.

```

1  auto exLucas = [&](LL n, LL m, LL p) {
2      auto inv = [&](LL a, LL p) {
3          LL x, y;
4          exgcd(a, p, x, y);
5          return (x % p + p) % p;
6      };
7
8      auto func = [&](auto&& self, LL n, LL pi, LL pk) {
9          if (!n) return 1ll;
10         LL ans = 1;
11         for (LL i = 2; i <= pk; i++) {
12             if (i % pi) ans = ans * i % p;
13         }
14         ans = quick_power(ans, n / pk, pk);
15         for (LL i = 2; i <= n % pk; i++) {
16             if (i % pi) ans = ans * i % pk;
17         }
18         ans = ans * self(self, n / pi, pi, pk) % pk;
19         return ans;
20     };
21
22     auto multiLucas = [&](LL n, LL m, LL pi, LL pk) {
23         LL cnt = 0;
24         for (LL i = n; i; i /= pi) cnt += i / pi;
25         for (LL i = m; i; i /= pi) cnt -= i / pi;
26         for (LL i = n - m; i; i /= pi) cnt -= i / pi;
27         LL ans = quick_power(pi, cnt, pk) * func(func, n, pi, pk) % pk;
28         ans = ans * inv(func(func, m, pi, pk), pk) % pk;
29         ans = ans * inv(func(func, n - m, pi, pk), pk) % pk;
30         return ans;
31     };
32
33     auto crt = [&](const vl& a, const vl& m, int k) {
34         LL ans = 0;
35
36         for (int i = 0; i < k; i++) {
37             ans = (ans + a[i] * inv(p / m[i], m[i]) * (p / m[i])) % p;
38         }
39         return (ans % p + p) % p;
40     };
41
42     vl a, prime;
43     LL pp = p;
44     for (int i = 2; i * i <= pp; i++) {
45         if (pp % i) continue;
46         prime.push_back(i);
47         while (pp % i == 0) {
48             prime.back() *= i;
49             pp /= i;
50         }
51         a.push_back(multiLucas(n, m, i, prime.back()));
52     }
53     if (pp > 1) {
54         prime.push_back(pp);
55         a.push_back(multiLucas(n, m, pp, pp));
56     }
57     return crt(a, prime, a.size());
58 };

```

## 5.11 Wilson

### 简单结论

对于素数  $p$  有

$$(p-1)! \equiv -1 \pmod{p}.$$

### 推论

令  $(n!)_p$  表示不大于  $n$  且不被  $p$  整除的正整数的乘积.

特殊情形:  $n$  为素数  $p$  时即为上述结论.

一般结论: 对素数  $p$  和正整数  $q$  有

$$((p^q)!)_p \equiv \pm 1 \pmod{p^q}.$$

详细定义:

$$((p^q)!)_p = \begin{cases} 1 & \text{if } p = 2 \text{ and } q \geq 3, \\ -1 & \text{other wise.} \end{cases}$$

### 更进一步的推论

## 5.12 LTE

将素数  $p$  在整数  $n$  中的个数记为  $v_p(n)$ .

$$(n, p) = 1$$

对所有素数  $p$  和满足  $(n, p) = 1$  的整数  $n$ , 有

1. 若  $p \mid x - y$ , 则有

$$v_p(x^n - y^n) = v_p(x - y).$$

2. 若  $p \mid x - y$ , 则对奇数  $n$  有

$$v_p(x^n + y^n) = v_p(x + y).$$

### $p$ 是奇素数

对所有奇素数  $p$  有

1. 若  $p \mid x - y$ , 则有

$$v_p(x^n - y^n) = v_p(x - y) + v_p(n).$$

2. 若  $p \mid x - y$ , 则对奇数  $n$  有

$$v_p(x^n + y^n) = v_p(x + y) + v_p(n).$$

$$p = 2$$

对  $p = 2$  且  $p \mid x - y$  有

1. 对奇数  $n$  有

$$v_2(x^n - y^n) = v_2(x - y).$$

2. 对偶数  $n$  有

$$v_2(x^n - y^n) = v_2(x - y) + v_2(x + y) + v_2(n) - 1.$$

除此之外, 对上述  $x, y, n$ , 若  $4 \mid x - y$ , 有

1.  $v_2(x + y) = 1$ .

2.  $v_2(x^n - y^n) = v_2(x - y) + v_2(n)$ .

### 5.13 Mobius inversion

莫比乌斯函数

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

性质

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

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

反演结论

$$[\gcd(i, j) = 1] = \sum_{d|\gcd(i, j)} \mu(d).$$

$O(n \log n)$  求莫比乌斯函数

```

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 }
```

莫比乌斯变换

设  $f(n), F(n)$ .

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

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



## 6 math - polynomial

### 6.1 FTT

#### FFT 与拆系数 FFT

```

1  const int sz = 1 << 23;
2  int rev[sz];
3  int rev_n;
4  void set_rev(int n) {
5      if (n == rev_n) return;
6      for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) * n) / 2;
7      rev_n = n;
8  }
9  template void butterfly(T* a, int n) {
10     set_rev(n);
11     for (int i = 0; i < n; i++) {
12         if (i < rev[i]) std::swap(a[i], a[rev[i]]);
13     }
14 }
15
16 namespace Comp {
17
18     long double pi = 3.141592653589793238;
19
20     template struct complex {
21         T x, y;
22         complex(T x = 0, T y = 0) : x(x), y(y) {}
23         complex operator+(const complex& b) const { return complex(x + b.x, y + b.y); }
24
25         complex operator-(const complex& b) const { return complex(x - b.x, y - b.y); }
26
27         complex operator*(const complex& b) const {
28             return complex<T>(x * b.x - y * b.y, x * b.y + y * b.x);
29         }
30         complex operator~() const { return complex(x, -y); }
31         static complex unit(long double rad) { return complex(std::cos(rad), std::sin(rad)); }
32     };
33
34 } // namespace Comp
35
36 struct fft_t {
37     typedef Comp::complex<double> complex;
38     complex wn[sz];
39
40     fft_t() {
41         for (int i = 0; i < sz / 2; i++) {
42             wn[sz / 2 + i] = complex::unit(2 * Comp::pi * i / sz);
43         }
44         for (int i = sz / 2 - 1; i; i--) wn[i] = wn[i * 2];
45     }
46
47     void operator()(complex* a, int n, int type) {
48         if (type == -1) std::reverse(a + 1, a + n);
49         butterfly(a, n);
50         for (int i = 1; i < n; i *= 2) {
51             const complex* w = wn + i;
52             for (complex* b = a, t; b != a + n; b += i + 1) {
53                 t = b[i];
54                 b[i] = *b - t;
55                 *b = *b + t;
56                 for (int j = 1; j < i; j++) {
57                     t = (++b)[i] * w[j];
58                     b[i] = *b - t;
59                     *b = *b + t;
60                 }
61             }
62         }
63         if (type == 1) return;
64         for (int i = 0; i < n * 2; i++) ((double*) a)[i] /= n;
65     }
66 } FFT;
67
68 typedef decltype(FFT)::complex complex;
69
70 vi fft(const vi& f, const vi& g) {
71     static complex ff[sz];
72     int n = f.size(), m = g.size();
73     vi h(n + m - 1);
74     if (std::min(n, m) <= 50) {
75         for (int i = 0; i < n; i++) {

```

```

76         for (int j = 0; j < m; ++j) {
77             h[i + j] += f[i] * g[j];
78         }
79     }
80     return h;
81 }
82 int c = 1;
83 while (c + 1 < n + m) c *= 2;
84 std::memset(ff, 0, sizeof(decltype(*(ff))) * (c));
85 for (int i = 0; i < n; i++) ff[i].x = f[i];
86 for (int i = 0; i < m; i++) ff[i].y = g[i];
87 FFT(ff, c, 1);
88 for (int i = 0; i < c; i++) ff[i] = ff[i] * ff[i];
89 FFT(ff, c, -1);
90 for (int i = 0; i + 1 < n + m; i++) h[i] = std::llround(ff[i].y / 2);
91 return h;
92 }
93
94 vi mtt(const vi& f, const vi& g) {
95     static complex ff[3][sz], gg[2][sz];
96     static int s[3] = {1, 31623, 31623 * 31623};
97     int n = f.size(), m = g.size();
98     vi h(n + m - 1);
99     if (std::min(n, m) <= 50) {
100         for (int i = 0; i < n; ++i) {
101             for (int j = 0; j < m; ++j) {
102                 Add(h[i + j], mul(f[i], g[j]));
103             }
104         }
105         return h;
106     }
107     int c = 1;
108     while (c + 1 < n + m) c *= 2;
109     for (int i = 0; i < 2; ++i) {
110         std::memset(ff[i], 0, sizeof(decltype(*(ff[i]))) * (c));
111         std::memset(gg[i], 0, sizeof(decltype(*(ff[i]))) * (c));
112         for (int j = 0; j < n; ++j) ff[i][j].x = f[j] / s[i] % s[1];
113         for (int j = 0; j < m; ++j) gg[i][j].x = g[j] / s[i] % s[1];
114         FFT(ff[i], c, 1);
115         FFT(gg[i], c, 1);
116     }
117     for (int i = 0; i < c; ++i) {
118         ff[2][i] = ff[1][i] * gg[1][i];
119         ff[1][i] = ff[1][i] * gg[0][i];
120         gg[1][i] = ff[0][i] * gg[1][i];
121         ff[0][i] = ff[0][i] * gg[0][i];
122     }
123     for (int i = 0; i < 3; ++i) {
124         FFT(ff[i], c, -1);
125         for (int j = 0; j + 1 < n + m; ++j) {
126             Add(h[j], mul(std::llround(ff[i][j].x) % mod, s[i]));
127         }
128     }
129     FFT(gg[1], c, -1);
130     for (int i = 0; i + 1 < n + m; ++i) {
131         Add(h[i], mul(std::llround(gg[1][i].x) % mod, s[1]));
132     }
133     return h;
134 }

```

## 6.2 FWT

and

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

分治过程

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

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

```

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 };

```

or

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

分治过程

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

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

```

1  /* mod 998244353 */
2  auto FWT_or = [&](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 + mid] = add(x, y);
10                 } else {
11                     v[i + mid] = sub(y, x);
12                 }
13             }
14         }
15     }
16     return v;
17 };

```

xor

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

分治过程

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

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

```

1  /* mod 998244353 */
2  auto FWT_xor = [&](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                  v[i] = add(x, y);
9                  v[i + mid] = sub(x, y);
10                 if (type == -1) {
11                     Mul(v[i], inv2);
12                     Mul(v[i + mid], inv2);
13                 }
14             }
15         }
16     }
17     return v;
18 };

```

统一地,

```

1 a = FWT(a, 1), b = FWT(b, 1);
2 for (int i = 0; i < (1 << n); i++) {
3     c[i] = mul(a[i], b[i]);
4 }
5 c = FWT(c, -1);

```

### 6.3 class polynomial

```

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;

```

## 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 }

```

## 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 }

```

## 对数

```

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  }

```

## 指数

```

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  }

```

## 根号

```

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));

```

## 6.4 wsy poly

```

1 #include <bits/stdc++.h>
2
3 using ul = std::uint32_t;
4 using li = std::int32_t;
5 using ll = std::int64_t;
6 using ull = std::uint64_t;
7 using llf = long double;
8 using lf = double;
9 using vul = std::vector<ul>;
10 using vvul = std::vector<vul>;
11 using pulb = std::pair<ul, bool>;
12 using vpulb = std::vector<pulb>;
13 using vvpulb = std::vector<vpulb>;

```

```

14 using vb = std::vector<bool>;
15
16 const ul base = 998244353;
17
18 std::mt19937 rnd;
19
20 ul plus(ul a, ul b) { return a + b < base ? a + b : a + b - base; }
21
22 ul minus(ul a, ul b) { return a < b ? a + base - b : a - b; }
23
24 ul mul(ul a, ul b) { return ull(a) * ull(b) % base; }
25
26 void exgcd(li a, li b, li& x, li& y) {
27     if (b) {
28         exgcd(b, a % b, y, x);
29         y -= x * (a / b);
30     } else {
31         x = 1;
32         y = 0;
33     }
34 }
35
36 ul inverse(ul a) {
37     li x, y;
38     exgcd(a, base, x, y);
39     return x < 0 ? x + li(base) : x;
40 }
41
42 ul pow(ul a, ul b) {
43     ul ret = 1;
44     ul temp = a;
45     while (b) {
46         if (b & 1) {
47             ret = mul(ret, temp);
48         }
49         temp = mul(temp, temp);
50         b >>= 1;
51     }
52     return ret;
53 }
54
55
56 ul sqrt(ul x) {
57     ul a;
58     ul w2;
59     while (true) {
60         a = rnd() % base;
61         w2 = minus(mul(a, a), x);
62         if (pow(w2, base - 1 >> 1) == base - 1) {
63             break;
64         }
65     }
66     ul b = base + 1 >> 1;
67     ul rs = 1, rt = 0;
68     ul as = a, at = 1;
69     ul qs, qt;
70     while (b) {
71         if (b & 1) {
72             qs = plus(mul(rs, as), mul(mul(rt, at), w2));
73             qt = plus(mul(rs, at), mul(rt, as));
74             rs = qs;
75             rt = qt;
76         }
77         b >>= 1;
78         qs = plus(mul(as, as), mul(mul(at, at), w2));
79         qt = plus(mul(as, at), mul(as, at));
80         as = qs;
81         at = qt;
82     }
83     return rs + rs < base ? rs : base - rs;
84 }
85
86 ul log(ul x, ul y, bool initd = false) {
87     static std::map<ul, ul> bs;
88     const ul d = std::round(std::sqrt(1l(base - 1)));
89     if (!initd) {
90         bs.clear();
91         for (ul i = 0, j = 1; i != d; ++i, j = mul(j, x)) {
92             bs[j] = i;
93         }
94     }
95     ul temp = inverse(pow(x, d));
96     for (ul i = 0, j = 1; i += d, j = mul(j, temp)) {
97         auto it = bs.find(mul(y, j));
98         if (it != bs.end()) {
99             return it->second + i;
100         }

```



```

101     }
102 }
103
104 ul powroot(ul x, ul y, bool initd = false) {
105     const ul g = 3;
106     ul lgx = log(g, x, initd);
107     li s, t;
108     exgcd(y, base - 1, s, t);
109     if (s < 0) {
110         s += base - 1;
111     }
112     return pow(g, ull(s) * ull(lgx) % (base - 1));
113 }
114
115 class polynomial : public vul {
116 public:
117     void clearzero() {
118         while (size() && !back()) {
119             pop_back();
120         }
121     }
122     polynomial() = default;
123     polynomial(const vul& a) : vul(a) {}
124     polynomial(vul&& a) : vul(std::move(a)) {}
125     ul degree() const { return size() - 1; }
126     ul operator()(ul x) const {
127         ul ret = 0;
128         for (ul i = size() - 1; ~i; --i) {
129             ret = mul(ret, x);
130             ret = plus(ret, vul::operator[](i));
131         }
132         return ret;
133     }
134 };
135
136 polynomial& operator+=(polynomial& a, const polynomial& b) {
137     a.resize(std::max(a.size(), b.size()), 0);
138     for (ul i = 0; i != b.size(); ++i) {
139         a[i] = plus(a[i], b[i]);
140     }
141     a.clearzero();
142     return a;
143 }
144
145 polynomial operator+(const polynomial& a, const polynomial& b) {
146     polynomial ret = a;
147     return ret += b;
148 }
149
150 polynomial& operator-=(polynomial& a, const polynomial& b) {
151     a.resize(std::max(a.size(), b.size()), 0);
152     for (ul i = 0; i != b.size(); ++i) {
153         a[i] = minus(a[i], b[i]);
154     }
155     a.clearzero();
156     return a;
157 }
158
159 polynomial operator-(const polynomial& a, const polynomial& b) {
160     polynomial ret = a;
161     return ret -= b;
162 }
163
164 class ntt_t {
165 public:
166     static const ul lgysz = 20;
167     static const ul sz = 1 << lgysz;
168     static const ul g = 3;
169     ul w[sz + 1];
170     ul leninv[lgysz + 1];
171     ntt_t() {
172         ul wn = pow(g, (base - 1) >> lgysz);
173         w[0] = 1;
174         for (ul i = 1; i <= sz; ++i) {
175             w[i] = mul(w[i - 1], wn);
176         }
177         leninv[lgysz] = inverse(sz);
178         for (ul i = lgysz - 1; ~i; --i) {
179             leninv[i] = plus(leninv[i + 1], leninv[i + 1]);
180         }
181     }
182     void operator()(vul& v, ul& n, bool inv) {
183         ul lgn = 0;
184         while ((1 << lgn) < n) {
185             ++lgn;
186         }
187         n = 1 << lgn;

```

```

188     v.resize(n, 0);
189     for (ul i = 0, j = 0; i != n; ++i) {
190         if (i < j) {
191             std::swap(v[i], v[j]);
192         }
193         ul k = n >> 1;
194         while (k & j) {
195             j &= ~k;
196             k >>= 1;
197         }
198         j |= k;
199     }
200     for (ul lgmid = 0; (1 << lgmid) != n; ++lgmid) {
201         ul mid = 1 << lgmid;
202         ul len = mid << 1;
203         for (ul i = 0; i != n; i += len) {
204             for (ul j = 0; j != mid; ++j) {
205                 ul t0 = v[i + j];
206                 ul t1 =
207                     mul(w[inv ? (len - j << lgsz - lgmid - 1) : (j << lgsz - lgmid - 1)],
208                        v[i + j + mid]);
209                 v[i + j] = plus(t0, t1);
210                 v[i + j + mid] = minus(t0, t1);
211             }
212         }
213     }
214     if (inv) {
215         for (ul i = 0; i != n; ++i) {
216             v[i] = mul(v[i], leninv[lgn]);
217         }
218     }
219 }
220 } ntt;
221
222 polynomial& operator*=(polynomial& a, const polynomial& b) {
223     if (!b.size() || !a.size()) {
224         a.resize(0);
225         return a;
226     }
227     polynomial temp = b;
228     ul npmp1 = a.size() + b.size() - 1;
229     if (ull(a.size()) * ull(b.size()) <= ull(npmp1) * ull(50)) {
230         temp.resize(0);
231         temp.resize(npmp1, 0);
232         for (ul i = 0; i != a.size(); ++i) {
233             for (ul j = 0; j != b.size(); ++j) {
234                 temp[i + j] = plus(temp[i + j], mul(a[i], b[j]));
235             }
236         }
237         a = temp;
238         a.clearzero();
239         return a;
240     }
241     ntt(a, npmp1, false);
242     ntt(temp, npmp1, false);
243     for (ul i = 0; i != npmp1; ++i) {
244         a[i] = mul(a[i], temp[i]);
245     }
246     ntt(a, npmp1, true);
247     a.clearzero();
248     return a;
249 }
250
251 polynomial operator*(const polynomial& a, const polynomial& b) {
252     polynomial ret = a;
253     return ret *= b;
254 }
255
256 polynomial& operator*=(polynomial& a, ul b) {
257     if (!b) {
258         a.resize(0);
259         return a;
260     }
261     for (ul i = 0; i != a.size(); ++i) {
262         a[i] = mul(a[i], b);
263     }
264     return a;
265 }
266
267 polynomial operator*(const polynomial& a, ul b) {
268     polynomial ret = a;
269     return ret *= b;
270 }
271
272 polynomial inverse(const polynomial& a, ul lgdeg) {
273     polynomial ret({inverse(a[0])});
274     polynomial temp;

```

```

275 polynomial tempa;
276 for (ul i = 0; i != lgdeg; ++i) {
277     tempa.resize(0);
278     tempa.resize(1 << i << 1, 0);
279     for (ul j = 0; j != tempa.size() && j != a.size(); ++j) {
280         tempa[j] = a[j];
281     }
282     temp = ret * (polynomial({2}) - tempa * ret);
283     if (temp.size() > (1 << i << 1)) {
284         temp.resize(1 << i << 1, 0);
285     }
286     temp.clearzero();
287     std::swap(temp, ret);
288 }
289 return ret;
290 }
291
292 void quotientremain(const polynomial& a, polynomial b, polynomial& q, polynomial& r) {
293     if (a.size() < b.size()) {
294         q = polynomial();
295         r = std::move(a);
296         return;
297     }
298     std::reverse(b.begin(), b.end());
299     auto ta = a;
300     std::reverse(ta.begin(), ta.end());
301     ul n = a.size() - 1;
302     ul m = b.size() - 1;
303     ta.resize(n - m + 1);
304     ul lgnmmp1 = 0;
305     while ((1 << lgnmmp1) < n - m + 1) {
306         ++lgnmmp1;
307     }
308     q = ta * inverse(b, lgnmmp1);
309     q.resize(n - m + 1);
310     std::reverse(b.begin(), b.end());
311     std::reverse(q.begin(), q.end());
312     r = a - b * q;
313 }
314
315 polynomial mod(const polynomial& a, const polynomial& b) {
316     polynomial q, r;
317     quotientremain(a, b, q, r);
318     return r;
319 }
320
321 polynomial quotient(const polynomial& a, const polynomial& b) {
322     polynomial q, r;
323     quotientremain(a, b, q, r);
324     return q;
325 }
326
327 polynomial sqrt(const polynomial& a, ul lgdeg) {
328     polynomial ret({sqrt(a[0])});
329     polynomial temp;
330     polynomial tempa;
331     for (ul i = 0; i != lgdeg; ++i) {
332         tempa.resize(0);
333         tempa.resize(1 << i << 1, 0);
334         for (ul j = 0; j != tempa.size() && j != a.size(); ++j) {
335             tempa[j] = a[j];
336         }
337         temp = (tempa * inverse(ret, i + 1) + ret) * (base + 1 >> 1);
338         if (temp.size() > (1 << i << 1)) {
339             temp.resize(1 << i << 1, 0);
340         }
341         temp.clearzero();
342         std::swap(temp, ret);
343     }
344     return ret;
345 }
346
347 polynomial differential(const polynomial& a) {
348     if (!a.size()) {
349         return a;
350     }
351     polynomial ret(vul(a.size() - 1, 0));
352     for (ul i = 1; i != a.size(); ++i) {
353         ret[i - 1] = mul(a[i], i);
354     }
355     return ret;
356 }
357
358 polynomial integral(const polynomial& a) {
359     polynomial ret(vul(a.size() + 1, 0));
360     for (ul i = 0; i != a.size(); ++i) {
361         ret[i + 1] = mul(a[i], inverse(i + 1));

```

```

362     }
363     return ret;
364 }
365
366 polynomial ln(const polynomial& a, ul lgdeg) {
367     polynomial da = differential(a);
368     polynomial inva = inverse(a, lgdeg);
369     polynomial ret = integral(da * inva);
370     if (ret.size() > (1 << lgdeg)) {
371         ret.resize(1 << lgdeg);
372         ret.clearzero();
373     }
374     return ret;
375 }
376
377 polynomial exp(const polynomial& a, ul lgdeg) {
378     polynomial ret({1});
379     polynomial temp;
380     polynomial tempa;
381     for (ul i = 0; i != lgdeg; ++i) {
382         tempa.resize(0);
383         tempa.resize(1 << i << 1, 0);
384         for (ul j = 0; j != tempa.size() && j != a.size(); ++j) {
385             tempa[j] = a[j];
386         }
387         temp = ret * (polynomial({1}) - ln(ret, i + 1) + tempa);
388         if (temp.size() > (1 << i << 1)) {
389             temp.resize(1 << i << 1, 0);
390         }
391         temp.clearzero();
392         std::swap(temp, ret);
393     }
394     return ret;
395 }
396
397 polynomial pow(const polynomial& a, ul k, ul lgdeg) { return exp(ln(a, lgdeg) * k, lgdeg); }
398
399 polynomial alpi[1 << 16][17];
400
401 polynomial getalpi(const ul x[], ul l, ul lgrml) {
402     if (lgrml == 0) {
403         return alpi[l][lgrml] = vul({minus(0, x[l]), 1});
404     }
405     return alpi[l][lgrml] = getalpi(x, l, lgrml - 1) * getalpi(x, l + (1 << lgrml - 1), lgrml - 1);
406 }
407
408 void multians(const polynomial& f, const ul x[], ul y[], ul l, ul lgrml) {
409     if (f.size() <= 700) {
410         for (ul i = l; i != l + (1 << lgrml); ++i) {
411             y[i] = f(x[i]);
412         }
413         return;
414     }
415     if (lgrml == 0) {
416         y[l] = f(x[l]);
417         return;
418     }
419     multians(mod(f, alpi[l][lgrml - 1]), x, y, l, lgrml - 1);
420     multians(mod(f, alpi[l + (1 << lgrml - 1)][lgrml - 1]), x, y, l + (1 << lgrml - 1), lgrml - 1);
421 }
422
423 ul sqrt(ul x) {
424     ul a;
425     ul w2;
426     while (true) {
427         a = rnd() % base;
428         w2 = minus(mul(a, a), x);
429         if (pow(w2, base - 1 >> 1) == base - 1) {
430             break;
431         }
432     }
433     ul b = base + 1 >> 1;
434     ul rs = 1, rt = 0;
435     ul as = a, at = 1;
436     ul qs, qt;
437     while (b) {
438         if (b & 1) {
439             qs = plus(mul(rs, as), mul(mul(rt, at), w2));
440             qt = plus(mul(rs, at), mul(rt, as));
441             rs = qs;
442             rt = qt;
443         }
444         b >>= 1;
445         qs = plus(mul(as, as), mul(mul(at, at), w2));
446         qt = plus(mul(as, at), mul(as, at));
447         as = qs;
448         at = qt;

```

```

449     }
450     return rs + rs < base ? rs : base - rs;
451 }
452
453 ul log(ul x, ul y, bool initied = false) {
454     static std::map<ul, ul> bs;
455     const ul d = std::round(std::sqrt(1f(base - 1)));
456     if (!initied) {
457         bs.clear();
458         for (ul i = 0, j = 1; i != d; ++i, j = mul(j, x)) {
459             bs[j] = i;
460         }
461     }
462     ul temp = inverse(pow(x, d));
463     for (ul i = 0, j = 1; i += d, j = mul(j, temp)) {
464         auto it = bs.find(mul(y, j));
465         if (it != bs.end()) {
466             return it->second + i;
467         }
468     }
469 }
470
471 ul powroot(ul x, ul y, bool initied = false) {
472     const ul g = 3;
473     ul lgx = log(g, x, initied);
474     li s, t;
475     exgcd(y, base - 1, s, t);
476     if (s < 0) {
477         s += base - 1;
478     }
479     return pow(g, ull(s) * ull(lgx) % (base - 1));
480 }
481
482 ul n;
483
484 int main() {
485     std::scanf("%u", &n);
486     polynomial f;
487     for (ul i = 0; i <= n; ++i) {
488         ul t;
489         std::scanf("%u", &t);
490         f.push_back(t % base);
491     }
492     polynomial g = exp(ln(f * inverse(f[0]), 17) * inverse(3), 17) * powroot(f[0], 3);
493     while (g.size() <= n) {
494         g.push_back(0);
495     }
496     for (ul i = 0; i <= n; ++i) {
497         if (i) {
498             std::putchar(' ');
499         }
500         std::printf("%u", g[i]);
501     }
502     std::putchar('\n');
503     return 0;
504 }

```

## Lagrange interpolation

### 一般的插值

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

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  $O(n^2)$ .

```

1 auto lagrange = (const vi& x, const vi& y, int n, int k) {
2     for (int i = 1; i <= n; i++) {
3         LL s1 = y[i] % mod, s2 = 1ll;
4         for (int j = 1; j <= n; j++) {
5             if (i != j) {
6                 s1 = s1 * (k - x[j]) % mod;
7                 s2 = s2 * (x[i] - x[j]) % mod;
8             }
9         }

```

```

10 |         Add(ans, mul(s1, quick_power(s2, mod - 2, mod)));
11 |     }
12 |     return ans;
13 | };

```

### 坐标连续的插值

给出的点是  $(i, y_i)$ .

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

时间复杂度为  $O(n)$ .

## 7 math - game theory

### 7.1 nim game

若 nim 和为 0, 则先手必败.

暴力打表.

```

1 vi SG(21, -1); /* 记忆化 */
2 std::function<int(int, int)> sg = [&](int x) -> int {
3     if (/* 为最终态 */) return SG[x] = 0;
4     if (SG[x] != -1) return SG[x];
5     vi st;
6     for (/* 枚举所有可到达的状态 y */) {
7         st.push_back(sg(y));
8     }
9     std::sort(all(st));
10    for (int i = 0; i < st.size(); i++) {
11        if (st[i] != i) return SG[x] = i;
12    }
13    return SG[x] = st.size();
14 };

```

### 7.2 anti - nim game

若

1. 所有堆的石子均为一个, 且 nim 和不为 0,
2. 至少有一堆石子超过一个, 且 nim 和为 0,

则先手必败.

## 8 math - linear algebra

### 8.1 matrix

#### determinant mod 998244353

```

1 auto det = [&](int n, vvi e) -> int {
2     int ans = 1;
3     for (int i = 1; i <= n; i++) {
4         if (a[i][i] == 0) {
5             for (int j = i + 1; j <= n; j++) {
6                 if (a[j][i] != 0) {
7                     for (int k = i; k <= n; k++) {
8                         std::swap(a[i][k], a[j][k]);
9                     }
10                    ans = sub(mod, ans);
11                    break;
12                }
13            }
14        }
15        if (a[i][i] == 0) return 0;
16        Mul(ans, a[i][i]);
17        int x = pow(a[i][i], mod - 2);
18        for (int k = i; k <= n; k++) {
19            Mul(a[i][k], x);
20        }
21        for (int j = i + 1; j <= n; j++) {
22            int x = a[j][i];
23            for (int k = i; k <= n; k++) {
24                Sub(a[j][k], mul(a[i][k], x));
25            }
26        }
27    }
28    return ans;
29 };

```

#### matrix multiplication

$A_{n \times m}$  与  $B_{m \times k}$  相乘并模 998244353.

```

1 auto matrix_mul = [&](int n, int m, int k, const vvi& a, const vvi& b) -> vvi {
2     vvi c(n + 1, vi(k + 1));
3     for (int i = 1; i <= n; i++) {
4         for (int l = 1; l <= m; l++) {
5             int x = a[i][l];
6             for (int j = 1; j <= k; j++) {
7                 Add(c[i][j], mul(x, b[l][j]));
8             }
9         }
10    }
11    return c;
12 };

```

### 8.2 linear basis

```

1 vi p(35);
2 auto add_basis = [&](int x) {
3     for (int i = 31; i >= 0; i--) {
4         if ((x >> i) & 1) continue;
5         if (!p[i]) {
6             p[i] = x;
7             break;
8         }
9         x ^= p[i];
10    }
11 };

```

### 8.3 linear programming



## 9 complex number

```

1  tandu struct Comp {
2      T a, b;
3
4      Comp(T _a = 0, T _b = 0) { a = _a, b = _b; }
5
6      Comp operator+(const Comp& x) const { return Comp(a + x.a, b + x.b); }
7
8      Comp operator-(const Comp& x) const { return Comp(a - x.a, b - x.b); }
9
10     Comp operator*(const Comp& x) const { return Comp(a * x.a - b * x.b, a * x.b + b * x.a); }
11
12     bool operator==(const Comp& x) const { return a == x.a and b == x.b; }
13
14     T real() { return a; }
15
16     T imag() { return b; }
17
18     U norm() { return (U) a * a + (U) b * b; }
19
20     Comp conj() { return Comp(a, -b); }
21
22     Comp operator/(const Comp& x) const {
23         Comp y = x;
24         Comp c = Comp(a, b) * y.conj();
25         T d = y.norm();
26         return Comp(c.a / d, c.b / d);
27     }
28 };
29
30 typedef Comp<LL, LL> complex;
31
32 complex gcd(complex a, complex b) {
33     LL d = b.norm();
34     if (d == 0) return a;
35     std::vector<complex> v(4);
36     complex c = a * b.conj();
37     auto fdiv = [&](LL a, LL b) -> LL { return a / b - ((a ^ b) < 0 && (a % b)); };
38     v[0] = complex(fdiv(c.real(), d), fdiv(c.imag(), d));
39     v[1] = v[0] + complex(1, 0);
40     v[2] = v[0] + complex(0, 1);
41     v[3] = v[0] + complex(1, 1);
42     for (auto& x : v) {
43         x = a - x * b;
44     }
45     std::sort(all(v), [&](complex a, complex b) { return a.norm() < b.norm(); });
46     return gcd(b, v[0]);
47 };

```

## 10 graph

### 10.1 topsort

```

1 vi top;
2 auto top_sort = [&]() -> bool {
3     vi d(n + 1);
4     std::queue<int> q;
5     for (int i = 1; i <= n; i++) {
6         d[i] = e[i].size();
7         if (!d[i]) q.push(i);
8     }
9     while (!q.empty()) {
10        int u = q.front();
11        q.pop();
12        top.push_back(u);
13        for (auto v : e[u]) {
14            d[v]--;
15            if (!d[v]) q.push(v);
16        }
17    }
18    if (top.size() != n) return false;
19    return true;
20 };

```

### 10.2 shortest path

#### Floyd

```

1 auto floyd = [&]() -> vvi {
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], e[i][j]);
6         }
7         dist[i][i] = 0;
8     }
9     for (int k = 1; k <= n; k++) {
10        for (int i = 1; i <= n; i++) {
11            for (int j = 1; j <= n; j++) {
12                Min(dist[i][j], dist[i][k] + dist[k][j]);
13            }
14        }
15    }
16    return dist;
17 };

```

#### Dijkstra

```

1 auto dijkstra = [&](int s) -> vl {
2     vl dist(n + 1, INF);
3     vi vis(n + 1, 0);
4     dist[s] = 0;
5     std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> q;
6     q.emplace(0LL, s);
7     while (!q.empty()) {
8         auto [dis, u] = q.top();
9         q.pop();
10        if (vis[u]) continue;
11        vis[u] = 1;
12        for (const auto& [v, w] : e[u]) {
13            if (dist[v] > dis + w) {
14                dist[v] = dis + w;
15                q.emplace(dist[v], v);
16            }
17        }
18    }
19    return dist;
20 };

```

## Bellman - Fold

```

1  int n, m, s;
2  int dist[N];
3  struct node{
4      int from, to, w;
5  }edge[M];
6  void bellman_fold(int s){
7      memset(dist, 0x3f, sizeof(dist));
8      dist[s] = 0;
9      for(int i = 1; i <= n; i++){
10         bool flag = true;
11         for(int j = 1; j <= m; j++){
12             int a = edge[j].from, b = edge[j].to, w = edge[j].w;
13             if(dist[a] == 0x3f3f3f3f) continue;
14             if(dist[b] > dist[a] + w){
15                 dist[b] = dist[a] + w;
16                 flag = false;
17             }
18         }
19         if(flag) break;
20     }
21 }

```

## SPFA

```

1  int n, m, s;
2  vl dist(n + 1, INF);
3  std::vector<bool> vis(n + 1);
4  std::vector<PLI > e(n + 1);
5
6  void spfa(int s){
7      rep(i, 1, n) dist[i] = INF;
8      dist[s] = 0;
9      std::queue<int> q;
10     q.push(s);
11     vis[s] = true;
12     while(q.size()){
13         auto u = q.front();
14         q.pop();
15         vis[u] = false;
16         for(auto j : e[u]){
17             int v = j.ff; LL w = j.ss;
18             if(dist[v] > dist[u] + w){
19                 dist[v] = dist[u] + w;
20                 if(!vis[v]){
21                     q.push(v);
22                     vis[v] = true;
23                 }
24             }
25         }
26     }
27 }

```

## Johnson

```

1  auto johnson = [&]() -> vl {
2      /* 负环 */
3      vl dist1(n + 1);
4      vl vis(n + 1), cnt(n + 1);
5      auto spfa = [&]() -> bool {
6          std::queue<int> q;
7          for (int u = 1; u <= n; u++) {
8              q.push(u);
9              vis[u] = false;
10          }
11          while (!q.empty()) {
12              auto u = q.front();
13              q.pop();
14              vis[u] = false;
15              for (auto [v, w] : e[u]) {
16                  if (dist1[v] > dist1[u] + w) {
17                      dist1[v] = dist1[u] + w;
18                      Max(cnt[v], cnt[u] + 1);
19                      if (cnt[v] >= n) return true;
20                      if (!vis[v]) {
21                          q.push(v);

```

```

22         vis[v] = true;
23     }
24 }
25 }
26 }
27 return false;
28 };
29
30 /* dijkstra */
31 vl dist2(n + 1);
32 auto dijkstra = [&](int s) {
33     for (int u = 1; u <= n; u++) {
34         dist2[u] = 1e9;
35         vis[u] = false;
36     }
37     dist2[s] = 0;
38     std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> q;
39     q.emplace(0, s);
40     while (!q.empty()) {
41         auto [d, u] = q.top();
42         q.pop();
43         if (vis[u]) continue;
44         vis[u] = true;
45         for (const auto& [v, w] : e[u]) {
46             if (dist2[v] > d + w) {
47                 dist2[v] = d + w;
48                 q.emplace(dist2[v], v);
49             }
50         }
51     }
52 };
53
54 if (spfa()) return vvl{};
55 for (int u = 1; u <= n; u++) {
56     for (auto& [v, w] : e[u]) {
57         w += dist1[u] - dist1[v];
58     }
59 }
60 vvl dist(n + 1, vl(n + 1));
61 for (int u; u <= n; u++) {
62     dijkstra(u);
63     for (int v = 1; v <= n; v++) {
64         if (dist2[v] == 1e9) {
65             dist[u][v] = INF;
66         } else {
67             dist[u][v] = dist2[v] + dist1[v] - dist1[u];
68         }
69     }
70 }
71 return dist;
72 };

```

### 最短路计数 - Dijkstra

```

1 auto dijkstra = [&](int s) -> std::pair<vl, vi> {
2     vl dist(n + 1, INF);
3     vi cnt(n + 1), vis(n + 1);
4     dist[s] = 0;
5     cnt[s] = 1;
6     std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> q;
7     q.emplace(0LL, s);
8     while (!q.empty()) {
9         auto [dis, u] = q.top();
10        q.pop();
11        if (vis[u]) continue;
12        vis[u] = 1;
13        for (const auto& [v, w] : e[u]) {
14            if (dist[v] > dis + w) {
15                dist[v] = dis + w;
16                cnt[v] = cnt[u];
17                q.push({dist[v], v});
18            } else if (dist[v] == dis + w) {
19                // cnt[v] += cnt[u];
20                cnt[v] += cnt[u];
21                cnt[v] %= 100003;
22            }
23        }
24    }
25    return {dist, cnt};
26 };

```

## 最短路计数 - Floyd

```

1 auto floyd() = [&] -> std::pair<vvi, vvi> {
2     vvi dist(n + 1, vi(n + 1, inf));
3     vvi cnt(n + 1, vi(n + 1, 1));
4     for (int i = 1; i <= n; i++) {
5         for (int j = 1; j <= n; j++) {
6             Min(dist[i][j], e[i][j]);
7         }
8         dist[i][i] = 0;
9     }
10    for (int k = 1; k <= n; k++) {
11        for (int i = 1; i <= n; i++) {
12            for (int j = 1; j <= n; j++) {
13                if (dist[i][j] == dist[i][k] + dist[k][j]) {
14                    cnt[i][j] += cnt[i][k] * cnt[k][j];
15                } else if (dist[i][j] > dist[i][k] + dist[k][j]) {
16                    cnt[i][j] = cnt[i][k] * cnt[k][j];
17                    dist[i][j] = dist[i][k] + dist[k][j];
18                }
19            }
20        }
21    }
22    return {dist, cnt};
23 };

```

## 负环

判断的是最短路长度.

```

1 auto spfa = [&]() -> bool {
2     std::queue<int> q;
3     vi vis(n + 1), cnt(n + 1);
4     for (int i = 1; i <= n; i++) {
5         q.push(i);
6         vis[i] = true;
7     }
8     while (!q.empty()) {
9         auto u = q.front();
10        q.pop();
11        vis[u] = false;
12        for (const auto& [v, w] : e[u]) {
13            if (dist[v] > dist[u] + w) {
14                dist[v] = dist[u] + w;
15                cnt[v] = cnt[u] + 1;
16                if (cnt[v] >= n) return true;
17                if (!vis[v]) {
18                    q.push(v);
19                    vis[v] = true;
20                }
21            }
22        }
23    }
24    return false;
25 }

```

## 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。

```

1 int main() {
2     std::ios::sync_with_stdio(false);
3     std::cin.tie(0);
4     std::cout.tie(0);
5
6     int n, m, k, s, t;
7     std::cin >> n >> m >> k;
8     std::cin >> s >> t;
9     std::vector<std::vector<PIL>> e(n * (k + 1) + 1);
10    for (int i = 1; i <= m; i++) {
11        int a, b, c;
12        std::cin >> a >> b >> c;
13        e[a].emplace_back(b, c);
14        e[b].emplace_back(a, c);
15        for (int j = 1; j <= k; j++) {
16            e[a + (j - 1) * n].emplace_back(b + j * n, 0);

```

```

17         e[b + (j - 1) * n].emplace_back(a + j * n, 0);
18         e[a + j * n].emplace_back(b + j * n, c);
19         e[b + j * n].emplace_back(a + j * n, c);
20     }
21 }
22
23 auto dijkstra = [&](int s) -> vl {};
24
25 vl dist = dijkstra(s);
26 LL ans = INF;
27 for (int i = t; i <= n * (k + 1); i += n) {
28     Min(ans, dist[i]);
29 }
30
31 std::cout << ans << endl;
32
33 return 0;
34 }

```

## 10.3 minimum spanning tree

### Kruskal

```

1  std::vector<std::tuple<int, int, int>> edge;
2  auto kruskal = [&]() -> int {
3      std::sort(all(edge), [&](std::tuple<int, int, int> a, std::tuple<int, int, int> b) {
4          auto [x1, y1, w1] = a;
5          auto [x2, y2, w2] = b;
6          return w1 < w2;
7      });
8      int res = 0, cnt = 0;
9      for (int i = 0; i < m; i++) {
10         auto [a, b, w] = edge[i];
11         a = find(a), b = find(b);
12         if (a != b) {
13             fa[a] = b;
14             res += w;
15             /* res = std::max(res, w); */
16             cnt++;
17         }
18     }
19     if (cnt < n - 1) return -1;
20     return res;
21 }

```

## 10.4 SCC

### Tarjan

```

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

```

## 10.4.1 缩点

## 10.5 DCC

## 点双连通分量

求点双连通分量.

```

1 vi dfn(n + 1), low(n + 1), is_bcc(n + 1), stk;
2 int timestamp = 0, bcc_cnt = 0, root = 0;
3 vvi bcc(2 * n + 1);
4 std::function<void(int, int)> tarjan = [&](int u, int fa) {
5     dfn[u] = low[u] = ++timestamp;
6     int child = 0;
7     stk.push_back(u);
8     if (u == root and e[u].empty()) {
9         bcc_cnt++;
10        bcc[bcc_cnt].push_back(u);
11        return;
12    }
13    for (auto v : e[u]) {
14        if (!dfn[v]) {
15            tarjan(v, u);
16            low[u] = std::min(low[u], low[v]);
17            if (low[v] >= dfn[u]) {
18                child++;
19                if (u != root or child > 1) {
20                    is_bcc[u] = 1;
21                }
22                bcc_cnt++;
23                int z;
24                do {
25                    z = stk.back();
26                    stk.pop_back();
27                    bcc[bcc_cnt].push_back(z);
28                } while (z != v);
29                bcc[bcc_cnt].push_back(u);
30            }
31        } else if (v != fa) {
32            low[u] = std::min(low[u], dfn[v]);
33        }
34    }
35 };
36 for (int i = 1; i <= n; i++) {
37     if (!dfn[i]) {
38         root = i;
39         tarjan(i, i);
40     }
41 }

```

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 }

```

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

```

## 10.6 two set

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.

```

1  auto twoSat = [&](int n, const vpi& v) -> vi {
2      /* tarjan */
3      vvi e(2 * n);
4      vi dfn(2 * n), low(2 * n), stk(2 * n), belong(2 * n);
5      int timestamp = 0, top = 0, scc_cnt = 0;

```



```

6      std::vector<bool> in_stk(2 * n);
7
8      auto tarjan = [&](auto&& self, int u) -> void {
9          dfn[u] = low[u] = ++timestamp;
10         stk[++top] = u;
11         in_stk[u] = true;
12         for (const auto& v : e[u]) {
13             if (!dfn[v]) {
14                 self(self, v);
15                 Min(low[u], low[v]);
16             } else if (in_stk[v]) {
17                 Min(low[u], dfn[v]);
18             }
19         }
20         if (dfn[u] == low[u]) {
21             scc_cnt++;
22             int v;
23             do {
24                 v = stk[top--];
25                 in_stk[v] = false;
26                 belong[v] = scc_cnt;
27             } while (v != u);
28         }
29     };
30     /* end tarjan */
31
32     for (const auto& [a, b] : v) {
33         e[a].push_back(b ^ 1);
34         e[b].push_back(a ^ 1);
35     }
36     for (int i = 0; i < 2 * n; i++) {
37         if (!dfn[i]) tarjan(tarjan, i);
38     }
39     vi ans;
40     for (int i = 0; i < 2 * n; i += 2) {
41         if (belong[i] == belong[i + 1]) return vi{};
42         ans.push_back(belong[i] > belong[i + 1] ? i + 1 : i);
43     }
44     return ans;
45 };

```

上述将  $i$  与  $i + 1$  作为一个集合里的元素, 编号为 0 至  $2n - 1$ .

## 10.7 minimum ring

### 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     };

```

### tree - diameter

## 10.8 tree - center of gravity

```

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 };

```

## 10.9 tree - DSU on tree

给出一棵  $n$  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: 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 }

```

## 10.10 tree - AHU

```

1  std::map<vi, int> mapple;
2  std::function<int(vvi&, int, int)> tree_hash = [&](vvi& e, int u, int fa) -> int {
3      vi code;
4      if (u == 0) code.push_back(-1);
5      for (auto v : e[u]) {
6          if (v == fa) continue;
7          code.push_back(tree_hash(e, v, u));
8      }
9      std::sort(all(code));
10     int id = mapple.size();
11     auto it = mapple.find(code);
12     if (it == mapple.end()) {
13         mapple[code] = id;
14     } else {
15         id = it->ss;
16     }
17     return id;
18 };

```

## 10.11 tree - LCA

```

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

```

## 10.12 tree - HLD

对一棵有根树进行如下 4 种操作:

1.  $1\ x\ y\ z$ : 将节点  $x$  到节点  $y$  的最短路径上所有节点的值加上  $z$ .
2.  $2\ x\ y$ : 查询节点  $x$  到节点  $y$  的最短路径上所有节点的值和.
3.  $3\ x\ z$ : 将以节点  $x$  为根的子树上所有节点的值加上  $z$ .
4.  $4\ x$ : 查询以节点  $x$  为根的子树上所有节点的值和.

```

1  /* HLD */
2  int cnt = 0;
3  vi son(n + 1), fa(n + 1), siz(n + 1), depth(n + 1);
4  vi dfn(n + 1), rank(n + 1), top(n + 1), botton(n + 1);
5
6  auto dfs1 = [&](auto&& self, int u) -> void {
7      son[u] = -1, siz[u] = 1;
8      for (auto v : e[u]) {
9          if (depth[v] != 0) continue;
10         depth[v] = depth[u] + 1;
11         fa[v] = u;
12         self(self, v);
13         siz[u] += siz[v];
14         if (son[u] == -1 or siz[v] > siz[son[u]]) son[u] = v;
15     }
16 };
17
18 auto dfs2 = [&](auto&& self, int u, int t) -> void {
19     top[u] = t;
20     dfn[u] = ++cnt;
21     rank[cnt] = u;
22     botton[u] = dfn[u];
23     if (son[u] == -1) return;
24     self(self, son[u], t);
25     Max(botton[u], botton[son[u]]);
26     for (auto v : e[u]) {
27         if (v != son[u] and v != fa[u]) {
28             self(self, v, v);
29             Max(botton[u], botton[v]);
30         }
31     }
32 };
33
34 depth[root] = 1;
35 dfs1(dfs1, root);
36 dfs2(dfs2, root, root);
37
38 /*
39
40 /* 求 LCA */
41 auto LCA = [&](int a, int b) -> int {
42     while (top[a] != top[b]) {
43         if (depth[top[a]] < depth[top[b]]) std::swap(a, b);
44         a = fa[top[a]];
45     }
46     return (depth[a] > depth[b] ? b : a);
47 };
48
49 /* 维护 u 到 v 的路径 */
50 while (top[u] != top[v]) {
51     if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
52     opt(dfn[top[u]], dfn[u]);
53     u = fa[top[u]];
54 }
55 if (dfn[u] > dfn[v]) std::swap(u, v);
56 opt(dfn[u], dfn[v]);
57
58 /* 维护 u 为根的子树 */
59 opt(dfn[u], botton[u]);
60
61 */
62
63 /*
64 线段树的 build() 函数中
65 if (l == r) tree[u] = {1, 1, w[rank[l]], 0};
66 */
67
68 build(1, 1, n);

```

```

69
70 for (int i = 1; i <= m; i++) {
71     int op, u, v;
72     LL k;
73     std::cin >> op;
74     if (op == 1) {
75         std::cin >> u >> v >> k;
76         while (top[u] != top[v]) {
77             if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
78             modify(1, dfn[top[u]], dfn[u], k);
79             u = fa[top[u]];
80         }
81         if (dfn[u] > dfn[v]) std::swap(u, v);
82         modify(1, dfn[u], dfn[v], k);
83     } else if (op == 2) {
84         std::cin >> u >> v;
85         LL ans = 0;
86         while (top[u] != top[v]) {
87             if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
88             ans = (ans + query(1, dfn[top[u]], dfn[u])) % p;
89             u = fa[top[u]];
90         }
91         if (dfn[u] > dfn[v]) std::swap(u, v);
92         ans = (ans + query(1, dfn[u], dfn[v])) % p;
93         std::cout << ans << endl;
94     } else if (op == 3) {
95         std::cin >> u >> k;
96         modify(1, dfn[u], botton[u], k);
97     } else {
98         std::cin >> u;
99         std::cout << query(1, dfn[u], botton[u]) % p << endl;
100     }
101 }

```

## 10.13 tree - virtual tree

```

1 auto build_vtree = [&](vi ver) -> void {
2     std::sort(all(ver), [&](int x, int y) { return dfn[x] < dfn[y]; });
3     vi stk = {1};
4     for (auto v : ver) {
5         int u = stk.back();
6         int lca = LCA(v, u);
7         if (lca != u) {
8             while (dfn[lca] < dfn[stk.end()[-2]]) {
9                 g[stk.end()[-2]].push_back(stk.back());
10                stk.pop_back();
11            }
12            u = stk.back();
13            if (dfn[lca] != dfn[stk.end()[-2]]) {
14                g[lca].push_back(u);
15                stk.pop_back();
16                stk.push_back(lca);
17            } else {
18                g[lca].push_back(u);
19                stk.pop_back();
20            }
21        }
22        stk.push_back(v);
23    }
24    while (stk.size() > 1) {
25        int u = stk.end()[-2];
26        int v = stk.back();
27        g[u].push_back(v);
28        stk.pop_back();
29    }
30 };

```

## 10.14 tree - pseudo tree

```

1 /* ring detection (directed) */
2 vi vis(n + 1), fa(n + 1), ring;
3 auto dfs = [&](auto&& self, int u) -> bool {
4     vis[u] = 1;
5     for (const auto& v : e[u]) {
6         if (!vis[v]) {
7             fa[v] = u;
8             if (self(self, v)) {
9                 return true;

```

```

10     }
11     } else if (vis[v] == 1) {
12         ring.push_back(v);
13         for (auto x = u; x != v; x = fa[x]) {
14             ring.push_back(x);
15         }
16         reverse(all(ring));
17         return true;
18     }
19 }
20 vis[u] = 2;
21 return false;
22 };
23 for (int i = 1; i <= n; i++) {
24     if (!vis[i]) {
25         if (dfs(dfs, i)) {
26             // operations //
27         }
28     }
29 }
30
31 /* cycle detection (undirected) */
32 vi vis(n + 1), ring;
33 vpi fa(n + 1);
34 auto dfs = [&](auto&& self, int u, int from) -> bool {
35     vis[u] = 1;
36     for (const auto& [v, id] : e[u]) {
37         if (id == from) continue;
38         if (!vis[v]) {
39             fa[v] = {u, id};
40             if (self(self, v, id)) {
41                 return true;
42             }
43         } else if (vis[v] == 1) {
44             ring.push_back(v);
45             for (auto x = u; x != v; x = fa[x].ff) {
46                 ring.push_back(x);
47             }
48             return true;
49         }
50     }
51     vis[u] = 2;
52     return false;
53 };
54 for (int i = 1; i <= n; i++) {
55     if (!vis[i]) {
56         if (dfs(dfs, i, 0)) {
57             // operations //
58         }
59     }
60 }

```

## 10.15 tree - divide and conquer on tree

### 点分治

#### 第一个题

一棵  $n \leq 10^4$  个点的树，边权  $w \leq 10^4$ 。  $m \leq 100$  次询问树上是否存在长度为  $k \leq 10^7$  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  $n \leq 4 \times 10^4$  个点的树, 边权  $w \leq 10^3$ . 询问树上长度不超过  $k \leq 2 \times 10^4$  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 }

```

## 10.16 network flow - maximal flow

### Dinic

理论

通过 BFS 将网络根据点到原点的距离 (每条边长度定义为 1) 分层, 然后通过 DFS 暴力地在有效的网络中寻找增广路, 不断循环上述步骤直至图中不存在增广路.

BFS 逻辑:

$u \rightarrow v$  的条件满足下面两条:

1.  $v$  未必走过;
2.  $e: u \rightarrow v$  上还有残余流量, 即当前  $e$  的流量未达到其上限.

DFS 逻辑:

维护两个值:  $u$ : 当前搜索到哪个点;  $now$ : 可以增加的流量.  $u \rightarrow v$  的条件:

1. 在上一次 BFS 时,  $v$  在  $u$  下面一层, 即  $d_v = d_u + 1$ .
2. 递归  $\text{dfs}(v, now)$ , 这时可增加的流量上限要与  $e: u \rightarrow v$  中可增加的流量上限取最小值, 递归结果大于零才意味着可以增加流量.

优化:

1. 一次可以处理多条增广路.
2. 每一条有向边事实上只会增加一次流量, 引入  $cur$  记录处理到了每个点的哪一条边以加快 DFS.

```

1 struct edge {
2     int from, to;
3     LL cap, flow;
4
5     edge(int u, int v, LL c, LL f) : from(u), to(v), cap(c), flow(f) {}
6 };
7
8 struct Dinic {
9     int n, m = 0, s, t;
10    std::vector<edge> e;
11    vi g[N];
12    int d[N], cur[N], vis[N];
13
14    void init(int n) {
15        for (int i = 0; i < n; i++) g[i].clear();
16        e.clear();
17        m = 0;
18    }
19
20    void add(int from, int to, LL cap) {
21        e.push_back(edge(from, to, cap, 0));
22        e.push_back(edge(to, from, 0, 0));
23        g[from].push_back(m++);
24        g[to].push_back(m++);
25    }
26
27    bool bfs() {
28        for (int i = 1; i <= n; i++) {
29            vis[i] = 0;

```

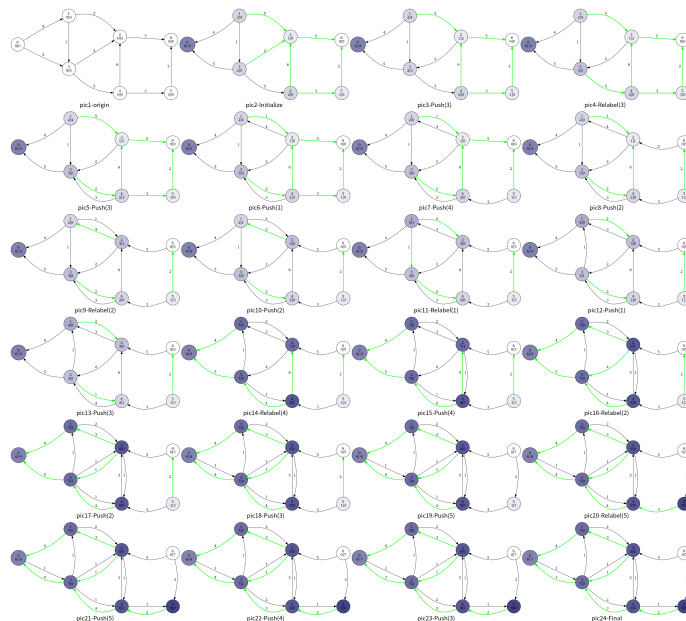
```

30     }
31     std::queue<int> q;
32     q.push(s), d[s] = 0, vis[s] = 1;
33     while (!q.empty()) {
34         int u = q.front();
35         q.pop();
36         for (int i = 0; i < g[u].size(); i++) {
37             edge& ee = e[g[u][i]];
38             if (!vis[ee.to] and ee.cap > ee.flow) {
39                 vis[ee.to] = 1;
40                 d[ee.to] = d[u] + 1;
41                 q.push(ee.to);
42             }
43         }
44     }
45     return vis[t];
46 }
47
48 LL dfs(int u, LL now) {
49     if (u == t || now == 0) return now;
50     LL flow = 0, f;
51     for (int& i = cur[u]; i < g[u].size(); i++) {
52         edge& ee = e[g[u][i]];
53         edge& er = e[g[u][i] ^ 1];
54         if (d[u] + 1 == d[ee.to] and (f = dfs(ee.to, std::min(now, ee.cap - ee.flow))) > 0) {
55             ee.flow += f, er.flow -= f;
56             flow += f, now -= f;
57             if (now == 0) break;
58         }
59     }
60     return flow;
61 }
62
63 LL dinic() {
64     LL ans = 0;
65     while (bfs()) {
66         for (int i = 1; i <= n; i++) cur[i] = 0;
67         ans += dfs(s, INF);
68     }
69     return ans;
70 }
71 } maxf;

```

## HLPP

抄板子吧，别管原理了，留一个图吧.



```

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.pushback(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.pushback(edge(u, v, w));
23     e.pushback(edge(v, u, 0));
24     g[u].pushback(m++);
25     g[v].pushback(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) {
38                 nd[v].hight = nd[u].hight + 1;
39                 if (vis[v] == false) {
40                     qq.push(v);
41                     vis[v] = true;
42                 }
43             }
44         }
45     }
46     return;
47 }
48
49 void _push(int u) {
50     for (auto j : g[u]) {
51         edge &ee = e[j], &er = e[j ^ 1];
52         int v = ee.to;
53         node &nu = nd[u], &nv = nd[v];
54         if (ee.cap && nv.hight + 1 == nu.hight) {
55             LL flow = std::min(ee.cap, nu.flow);
56             ee.cap -= flow, er.cap += flow;
57             nu.flow -= flow, nv.flow += flow;
58             if (vis[v] == false && v != t && v != s) {
59                 q.push(nv);
60                 vis[v] = true;
61             }
62             if (nu.flow == 0) break;
63         }
64     }
65 }
66
67 void relabel(int u) {
68     nd[u].hight = inf;
69     for (auto j : g[u]) {
70         int v = e[j].to;
71         if (e[j].cap && nd[v].hight + 1 < nd[u].hight) {
72             nd[u].hight = nd[v].hight + 1;
73         }
74     }
75 }
76
77 LL hlpp() {
78     bfs();
79     if (nd[s].hight == inf) return 0;
80     nd[s].hight = n;
81     for (int i = 1; i <= n; i++) {
82         if (nd[i].hight < inf) cnt[nd[i].hight]++;
83     }
84     for (auto j : g[s]) {
85         int v = e[j].to;
86         int flow = e[j].cap;
87         if (flow) {
88             e[j].cap -= flow, e[j ^ 1].cap += flow;
89             nd[s].flow -= flow, nd[v].flow += flow;
90             if (vis[v] == false && v != s && v != t) {
91                 q.push(nd[v]);
92                 vis[v] = true;

```

```

93     }
94 }
95 }
96 while (!q.empty()) {
97     int u = q.top().id;
98     q.pop();
99     vis[u] = false;
100     _push(u);
101     if (nd[u].flow) {
102         cnt[nd[u].hight]--;
103         if (cnt[nd[u].hight] == 0) {
104             for (int i = 1; i <= n; i++) {
105                 if (i != s && i != t && nd[i].hight > nd[u].hight && nd[i].hight < n + 1) {
106                     nd[i].hight = n + 1;
107                 }
108             }
109         }
110         relabel(u);
111         cnt[nd[u].hight]++;
112         q.push(nd[u]);
113         vis[u] = true;
114     }
115 }
116 return nd[t].flow;
117 }
118 } maxf;

```

## 10.17 network flow - minimum cost flow

在网络中获得最大流的同时要求费用最小。

### Dinic + SPFA

```

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 MCMF {
9     int n, m = 0, s, t;
10    std::vector<edge> e;
11    vi g[N];
12    int cur[N], vis[N];
13    LL dist[N], minc;
14
15    void init(int n) {
16        for (int i = 0; i < n; i++) g[i].clear();
17        e.clear();
18        minc = m = 0;
19    }
20
21    void add(int from, int to, LL cap, LL cost) {
22        e.push_back(edge(from, to, cap, cost));
23        e.push_back(edge(to, from, 0, -cost));
24        g[from].push_back(m++);
25        g[to].push_back(m++);
26    }
27
28    bool spfa() {
29        rep(i, 1, n) { dist[i] = INF, cur[i] = 0; }
30        std::queue<int> q;
31        q.push(s), dist[s] = 0, vis[s] = 1;
32        while (!q.empty()) {
33            int u = q.front();
34            q.pop();
35            vis[u] = 0;
36            for (int j = cur[u]; j < g[u].size(); j++) {
37                edge& ee = e[g[u][j]];
38                int v = ee.to;
39                if (ee.cap && dist[v] > dist[u] + ee.cost) {
40                    dist[v] = dist[u] + ee.cost;
41                    if (!vis[v]) {
42                        q.push(v);
43                        vis[v] = 1;
44                    }
45                }
46            }
47        }
48    }

```

```

47     }
48     return dist[t] != INF;
49 }
50
51 LL dfs(int u, LL now) {
52     if (u == t) return now;
53     vis[u] = 1;
54     LL ans = 0;
55     for (int& i = cur[u]; i < g[u].size() && ans < now; i++) {
56         edge &ee = e[g[u][i]], &er = e[g[u][i] ^ 1];
57         int v = ee.to;
58         if (!vis[v] && ee.cap && dist[v] == dist[u] + ee.cost) {
59             LL f = dfs(v, std::min(ee.cap, now - ans));
60             if (f) {
61                 minc += f * ee.cost, ans += f;
62                 ee.cap -= f;
63                 er.cap += f;
64             }
65         }
66     }
67     vis[u] = 0;
68     return ans;
69 }
70
71 PLL mcmf() {
72     LL maxf = 0;
73     while (spfa()) {
74         LL tmp;
75         while ((tmp = dfs(s, INF))) maxf += tmp;
76     }
77     return std::makepair(maxf, minc);
78 }
79 } minc_maxf;

```

## Primal-Dual 原始对偶算法

```

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 && 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::make_pair(maxf, minc);
94 }
95 } minc_maxf;

```

存在负环的网络

## 10.18 network flow - minimal cut

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.

### 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.

### 获得 $S$ 中的所有点

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  $S$  集中的点.

### 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.

## 10.19 matching - matching on bipartite graph

### 二分图最大匹配

#### Kuhn-Munkres

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;

```

#### Hopcroft-Karp

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 };

```

## 二分图最大权匹配

### Kuhn-Munkres

注意是否为完美匹配, 非完美选 0, 完美选  $-INF$ . (存疑)

```

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 {
7         int a, y = 0, y1 = 0;
8         std::fill(all(pp), 0);
9         std::fill(all(vx), INF);
10        r[y] = x;
11        do {
12            a = r[y], delta = INF, vb[y] = 1;
13            for (int b = 1; b <= n; b++) {
14                if (!vb[b]) {
15                    if (vx[b] > la[a] + lb[b] - e[a][b]) {
16                        vx[b] = la[a] + lb[b] - e[a][b];
17                        pp[b] = y;
18                    }
19                    if (vx[b] < delta) {
20                        delta = vx[b];
21                        y1 = b;
22                    }
23                }
24            }
25            for (int b = 0; b <= n; b++) {
26                if (vb[b]) {
27                    la[r[b]] -= delta;
28                    lb[b] += delta;
29                } else {
30                    vx[b] -= delta;
31                }
32            }
33            y = y1;
34        } while (r[y] != -1);
35        while (y) {
36            r[y] = r[pp[y]];
37            y = pp[y];
38        }
39    };
40    for (int i = 1; i <= n; i++) {
41        std::fill(all(vb), 0);
42        bfs(i);
43    }
44    LL ans = 0;
45    for (int i = 1; i <= n; i++) {
46        if (r[i] == -1) continue;
47        l[r[i]] = i;
48        ans += e[r[i]][i];
49    }
50    return {ans, l, r};
51 };
52 auto [ans, mchl, mchr] = KM(n, e);

```

## 10.20 matching - matching on general graph



## 11 geometry

### 11.1 two demention

#### 点与向量

```

1  tandu struct pnt {
2      T x, y;
3
4      pnt(T _x = 0, T _y = 0) { x = _x, y = _y; }
5
6      pnt operator+(const pnt& a) const { return pnt(x + a.x, y + a.y); }
7
8      pnt operator-(const pnt& a) const { return pnt(x - a.x, y - a.y); }
9
10     /*
11     bool operator<(const pnt& a) const {
12         if (std::is_same<T, double>::value) {
13             if (fabs(x - a.x) < eps) return y < a.y;
14         } else {
15             if (x == a.x) return y < a.y;
16         }
17         return x < a.x;
18     }
19     */
20
21     /* 注意数乘会不会爆 int */
22     pnt operator*(const T k) const { return pnt(k * x, k * y); }
23
24     U operator*(const pnt& a) const { return (U) x * a.x + (U) y * a.y; }
25
26     U operator^(const pnt& a) const { return (U) x * a.y - (U) y * a.x; }
27
28     U dist(const pnt a) { return ((U) a.x - x) * ((U) a.x - x) + ((U) a.y - y) * ((U) a.y - y); }
29
30     U len() { return dist(pnt(0, 0)); }
31
32     /* a, b, c 成逆时针 */
33     friend U area(pnt a, pnt b, pnt c) { return (b - a) ^ (c - a); }
34
35     /* 两向量夹角, 返回 cos 值 */
36     double get_angle(pnt a) {
37         return (double) (pnt(x, y) * a) / sqrt((double) pnt(x, y).len() * (double) a.len());
38     }
39 };

```

#### 线段

```

1  struct line {
2      point a, b;
3
4      line(point _a = {}, point _b = {}) { a = _a, b = _b; }
5
6      /* 交点类型为 double */
7      friend point iPoint(line p, line q) {
8          point v1 = p.b - p.a;
9          point v2 = q.b - q.a;
10         point u = q.a - p.a;
11         return q.a + (q.b - q.a) * ((u ^ v1) * 1. / (v1 ^ v2));
12     }
13
14     /* 极角排序 */
15     bool operator<(const line& p) const {
16         double t1 = std::atan2((b - a).y, (b - a).x);
17         double t2 = std::atan2((p.b - p.a).y, (p.b - p.a).x);
18         if (fabs(t1 - t2) > eps) {
19             return t1 < t2;
20         }
21         return ((p.a - a) ^ (p.b - a)) > eps;
22     }
23 };

```

## 11.2 convex

### 2D

```

1  auto andrew = [&](std::vector<point>& v) -> std::vector<point> {
2      std::sort(all(v));
3      std::vector<point> stk;
4      for (int i = 0; i < n; i++) {
5          point x = v[i];
6          while (stk.size() > 1 and ((stk.end()[-1] - stk.end()[-2]) ^ (x - stk.end()[-2])) <= 0) {
7              stk.pop_back();
8          }
9          stk.push_back(x);
10     }
11     int tmp = stk.size();
12     for (int i = n - 2; i >= 0; i--) {
13         point x = v[i];
14         while (stk.size() > tmp and ((stk.end()[-1] - stk.end()[-2]) ^ (x - stk.end()[-2])) <= 0) {
15             stk.pop_back();
16         }
17         stk.push_back(x);
18     }
19     return stk;
20 };

```

### half plane

```

1  auto halfPlane = [&](std::vector<line>& ln) -> std::vector<point> {
2      std::sort(all(ln));
3      ln.erase(
4          unique(
5              all(ln),
6              [](line& p, line& q) {
7                  double t1 = std::atan2((p.b - p.a).y, (p.b - p.a).x);
8                  double t2 = std::atan2((q.b - q.a).y, (q.b - q.a).x);
9                  return fabs((t1 - t2)) < eps;
10             }
11         ), ln.end());
12     auto check = [&](line p, line q, line r) -> bool {
13         point a = iPoint(p, q);
14         return ((r.b - r.a) ^ (a - r.a)) < -eps;
15     };
16     line q[ln.size() + 2];
17     int hh = 1, tt = 0;
18     q[+tt] = ln[0];
19     q[+tt] = ln[1];
20     for (int i = 2; i < (int) ln.size(); i++) {
21         while (hh < tt and check(q[tt - 1], q[tt], ln[i])) tt--;
22         while (hh < tt and check(q[hh + 1], q[hh], ln[i])) hh++;
23         q[+tt] = ln[i];
24     }
25     while (hh < tt and check(q[tt - 1], q[tt], q[hh])) tt--;
26     while (hh < tt and check(q[hh + 1], q[hh], q[tt])) hh++;
27     q[tt + 1] = q[hh];
28     std::vector<point> ans;
29     for (int i = hh; i <= tt; i++) {
30         ans.push_back(iPoint(q[i], q[i + 1]));
31     }
32     return ans;
33 };

```

## 12 offline algorithm

### 12.1 discretization

```

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

```

### 12.2 Mo algorithm

普通莫队

```

1 int block = n / sqrt(2 * m / 3);
2 std::sort(all(q), [&](node a, node b) {
3     return a.l / block == b.l / block ? (a.r == b.r ? 0 : ((a.l / block) & 1) ^ (a.r < b.r))
4         : a.l < b.l;
5 });
6
7 auto move = [&](int x, int op) -> void {
8     if (op == 1) {
9         /* operations */
10    } else {
11        /* operations */
12    }
13 };
14
15 for (int k = 1, l = 1, r = 0; k <= m; k++) {
16     node Q = q[k];
17     while (l > Q.l) {
18         move(--l, 1);
19     }
20     while (r < Q.r) {
21         move(++r, 1);
22     }
23     while (l < Q.l) {
24         move(l++, -1);
25     }
26     while (r > Q.r) {
27         move(r--, -1);
28     }
29 }

```

### 12.3 CDQ

$n$  个三维数对  $(a_i, b_i, c_i)$ , 设  $f(i)$  表示  $a_j \leq a_i, b_j \leq b_i, c_j \leq c_i (i \neq j)$  的个数. 输出  $f(i)$  ( $0 \leq i \leq n-1$ ) 的值.

```

1 // 洛谷 P3810 【模板】三维偏序（陌上花开）
2
3 struct data {
4     int a, b, c, cnt, ans;
5
6     data(int _a = 0, int _b = 0, int _c = 0, int _cnt = 0, int _ans = 0) {
7         a = _a, b = _b, c = _c, cnt = _cnt, ans = _ans;
8     }
9
10    bool operator!=(data x) {
11        if (a != x.a) return true;
12        if (b != x.b) return true;
13        if (c != x.c) return true;
14        return false;
15    }
16 };
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;

```

```

25     std::cin >> n >> k;
26     static data v1[N], v2[N];
27     for (int i = 1; i <= n; i++) {
28         std::cin >> v1[i].a >> v1[i].b >> v1[i].c;
29     }
30
31     std::sort(v1 + 1, v1 + n + 1, [&](data x, data y) {
32         if (x.a != y.a) return x.a < y.a;
33         if (x.b != y.b) return x.b < y.b;
34         return x.c < y.c;
35     });
36
37     int t = 0, top = 0;
38     for (int i = 1; i <= n; i++) {
39         t++;
40         if (v1[i] != v1[i + 1]) {
41             v2[++top] = v1[i];
42             v2[top].cnt = t;
43             t = 0;
44         }
45     }
46
47     vi tr(N);
48
49     auto add = [&](int pos, int val) -> void {
50         while (pos <= k) {
51             tr[pos] += val;
52             pos += lowbit(pos);
53         }
54     };
55
56     auto query = [&](int pos) -> int {
57         int ans = 0;
58         while (pos > 0) {
59             ans += tr[pos];
60             pos -= lowbit(pos);
61         }
62         return ans;
63     };
64
65     std::function<void(int, int)> CDQ = [&](int l, int r) -> void {
66         if (l == r) return;
67         int mid = (l + r) >> 1;
68         CDQ(l, mid), CDQ(mid + 1, r);
69         std::sort(v2 + 1, v2 + mid + 1, [&](data x, data y) {
70             if (x.b != y.b) return x.b < y.b;
71             return x.c < y.c;
72         });
73         std::sort(v2 + mid + 1, v2 + r + 1, [&](data x, data y) {
74             if (x.b != y.b) return x.b < y.b;
75             return x.c < y.c;
76         });
77         int i = l, j = mid + 1;
78         while (j <= r) {
79             while (i <= mid && v2[i].b <= v2[j].b) {
80                 add(v2[i].c, v2[i].cnt);
81                 i++;
82             }
83             v2[j].ans += query(v2[j].c);
84             j++;
85         }
86         for (int ii = l; ii < i; ii++) {
87             add(v2[ii].c, -v2[ii].cnt);
88         }
89         return;
90     };
91
92     CDQ(1, top);
93     vi ans(n + 1);
94     for (int i = 1; i <= top; i++) {
95         ans[v2[i].ans + v2[i].cnt] += v2[i].cnt;
96     }
97     for (int i = 1; i <= n; i++) {
98         std::cout << ans[i] << endl;
99     }
100
101     return 0;
102 }

```