



ACM/ICPC Template The Last Dance

浙江工商大学

最后一舞

May 18, 2023

Contents

0	头文件	1
0.1	header	1
1	字符串	2
1.1	hash	2
1.2	KMP	2
1.3	automaton	3
1.4	manacher	4
1.5	Z-function	4
1.6	suffix-array	5
1.7	suffixAutomaton	7
2	动态规划	9
2.1	01knapSack	9
2.2	simpleBoundedKnapSack	9
2.3	binaryBoundedKnapSack	9
2.4	monotonousQueueMultipleKnapSack	10
2.5	unboundedKnapSack	10
2.6	hybridKnapSack	10
2.7	groupKnapSack	11
2.8	KnapSack2d	11
2.9	treeKnapSack	12
2.10	LIS	12
2.11	digitDP	13
2.12	bitmaskingDP	13
2.13	quadrilateralOptimization	14
2.14	baseRingTreeDP	14
2.15	segmentTreeDP	15
2.16	LCS	17
3	数据结构	18
3.1	BTree	18
3.2	pbds-tree	20
3.3	fenwick	20
3.4	fenwick2d	21
3.5	SegmentTree	22
3.6	SegmentTree2d	23
3.7	ValueSegmentTree	24
3.8	线段树动态开点合并分裂	25
3.9	线段树维护 LIS 方案数	26
3.10	线段树维护最小字典序 LIS	28
3.11	线段树维护插队问题	30
3.12	线段树维护连续区间异或值	31
3.13	线段树维护区间异或	32
3.14	LazySegmentTree	33
3.15	SparseTable	35
3.16	CartesianTree	37
3.17	DancingLinks	37
3.18	ChthollyTree	39
3.19	monotonousQueue	40
3.20	monotonousStack	41
3.21	difference	42
3.22	trie	43
3.23	HashTable	43
3.24	2-4 维前缀和	44
3.25	simpleDSU	45
3.26	valueDSU	46
3.27	modifyDSU	46

3.28	varietyDSU	47
3.29	KD 求矩阵权值和	48
3.30	KD 求最近点对距离	50
3.31	CDQ 分治	51
3.32	cdq 处理逆序数	51
3.33	cdq 处理二维偏序	52
3.34	cdq 套 cdq 处理三维偏序	53
3.35	cdq 套树状数组处理三维偏序	54
3.36	cdq 维护矩阵内二维数点	55
3.37	advancedO1trie	56
3.38	advancedArray	57
3.39	advancedDSU	58
3.40	扫描线求面积并	60
3.41	扫描线求周长并	61
3.42	区间第 k 小	63
3.43	区间前 k 大	64
3.44	树套树维护三维偏序	65
3.45	线段树套主席树-二维区间不同数	67
3.46	Scapegoat	69
3.47	Splay	72
3.48	O1trie	77
3.49	Treap	78
4	博弈论	80
4.1	bash	80
4.2	fibonacci	80
4.3	nim	80
4.4	wythoff	80
4.5	sgFunction	81
5	树与森林	82
5.1	forest	82
5.2	dfs-forest	82
5.3	lca-forest	84
5.4	重链剖分	85
5.5	tree-diameter	87
5.6	树的重心	89
5.7	树的最大匹配	89
5.8	树分治-点分治	90
5.9	树上 dsu-维护路径信息	93
5.10	树上 dsu-维护子树信息	94
5.11	树上 K 祖先	95
5.12	virtualTree	96
5.13	LCT	98
6	数学	100
6.1	Mint	100
6.2	Z	103
6.3	exgcd	105
6.4	factorizer	105
6.5	comb	110
6.6	算术基本定理	110
6.7	筛 phi	111
6.8	筛 mobius	111
6.9	筛积性函数	112
6.10	欧拉函数	112
6.11	原根	113
6.12	原根表	115
6.13	阶乘逆元	116

6.14 常见积性函数	116
6.15 Miller-Rabin	116
6.16 quadraticResidue	117
6.17 bags	119
6.18 EX-BSGS	119
6.19 CRT	120
6.20 EX-CRT	121
6.21 EX-Lucas	123
6.22 min25	125
6.23 BM	125
6.24 duSieve	127
6.25 反演相关	128
6.26 simpson	129
6.27 Bell	129
6.28 Catalan	130
6.29 Lucas	131
6.30 伯努利数	132
6.31 步移-组合数前缀和	134
6.32 康托展开	134
6.33 模数非质数的组合	137
6.34 k 次最小置换复原	138
6.35 Striling	139
6.36 第二类斯特林数-行	139
6.37 第二类斯特林数-列	140
6.38 第一类斯特林数-行	141
6.39 第一类斯特林数-列	143
6.40 整数拆分多项式求逆	145
6.41 普通型母函数	146
6.42 指数型母函数	147
6.43 伯努利数求和	148
6.44 fibonacciBigInteger	150
6.45 fibonacciCircular	151
6.46 Matrix	155
6.47 拉格朗日插值求和	157
6.48 SolveLinearSystem	159
6.49 矩阵求逆	160
6.50 eraseLinearBasis	161
6.51 simpleLinearBasis	162
6.52 intervalModifyLinearBasis	164
6.53 noIntervalModifyLinearBasis	165
6.54 FWT	166
6.55 cdq 分治 FFT	167
6.56 求逆分治 FFT	168
6.57 多项式求逆	169
6.58 多项式快速幂	170
6.59 多项式除法、取模	172
6.60 多项式 \ln -Exp-Pow	173
6.61 任意模数 MTT-拆系数法	175
6.62 任意模数 NTT-三模数法	177
6.63 多项式优化常系数齐次线性递推	178
6.64 FFT 加速带有通配符字符串匹配	180
6.65 FFT 加速朴素字符串匹配	181
6.66 \times 不连续、暴力插值	183
6.67 \times 连续、前缀优化	183
6.68 多项式 \ln -exp-pow 处理边界为 1	184
6.69 二次剩余处理边界不为 1	185
6.70 二维几何	187
6.71 三维几何	193
6.72 Poly-Z	199

6.73	多项式快速幂	204
6.74	Geometry	206
7	图论	209
7.1	graph	209
7.2	isBipartiteGraph	210
7.3	hungry	210
7.4	KM	211
7.5	galeShapley	213
7.6	bellman-ford	214
7.7	dijkstra	215
7.8	floyd	215
7.9	spfa	216
7.10	Kruskal	216
7.11	prim	216
7.12	topSort	217
7.13	scc	218
7.14	cycles	218
7.15	ternaryRingCount	220
7.16	eulerian-path	220
7.17	twoSat	222
7.18	maxAssignment	222
7.19	tarjan	225
7.20	biconnected-components	226
7.21	differenceConstraints	226
7.22	AHU	227
7.23	Astar	229
7.24	dinic	231
7.25	ISAP	233
7.26	mcmf	235
7.27	conclusion.md	236
8	其他	237
8.1	simpleMO	237
8.2	modifyMO	238
8.3	rollbackMO	239
8.4	treeMO	241
8.5	fastIO	243
8.6	simulatedAnnealing	246
8.7	Cantor	247
8.8	BigInteger	248
8.9	debug	250
8.10	int128	252

0 头文件

0.1 header

```

1  #include "bits/stdc++.h"
2  using namespace std;
3
4  typedef long long ll;
5  typedef long double ld;
6  typedef unsigned long long ull;
7  typedef vector<ll> VI;
8  typedef pair<int, int> pii;
9  typedef pair<double, double> pdd;
10 typedef pair<ll, ll> pll;
11
12 #define endl "\n"
13 #define fi first
14 #define se second
15 #define eb emplace_back
16 #define mem(a, b) memset(a, b, sizeof(a))
17
18 const ll INF = 0x3f3f3f3f;
19 const ll mod = 998244353;
20 const double eps = 1e-6;
21 const double PI = acos(-1);
22 const double R = 0.57721566490153286060651209;
23
24 void solve() {
25
26 }
27
28 signed main() {
29     ios_base::sync_with_stdio(false);
30     // cin.tie(nullptr);
31     // cout.tie(nullptr);
32 #ifdef FZT_ACM_LOCAL
33     freopen("in.txt", "r", stdin);
34     freopen("out.txt", "w", stdout);
35     signed test_index_for_debug = 1;
36     char acm_local_for_debug = 0;
37     do {
38         if (acm_local_for_debug == '$') exit(0);
39         if (test_index_for_debug > 20)
40             throw runtime_error("Check the stdin!!!");
41         auto start_clock_for_debug = clock();
42         solve();
43         auto end_clock_for_debug = clock();
44         cout << "Test " << test_index_for_debug << " successful" << endl;
45         cerr << "Test " << test_index_for_debug++ << " Run Time: "
46              << double(end_clock_for_debug - start_clock_for_debug) / CLOCKS_PER_SEC <<
47              "s" << endl;
48         cout << "-----" << endl;
49     } while (cin >> acm_local_for_debug && cin.putback(acm_local_for_debug));
50 #else
51     solve();
52 #endif
53     return 0;
54 }

```

1 字符串

1.1 hash

```

1 struct Hash {
2     using ui64 = unsigned long long;
3     static constexpr int P = 1331;
4     std::vector<ui64> h, p;
5
6     Hash(std::string s) : h(s.size() + 1), p(s.size() + 1) {
7         p[0] = 1;
8         for (int i = 1; i <= (int) s.size(); i++) {
9             p[i] = p[i - 1] * P;
10            h[i] = h[i - 1] * P + s[i - 1] - '0';
11        }
12    }
13
14    ui64 rangeSum(int l, int r) {
15        return h[r] - h[l - 1] * p[r - l + 1];
16    }
17 };

```

1.2 KMP

```

1 template <typename T>
2 std::vector<int> kmp_table(int n, const T &s) {
3     std::vector<int> p(n, 0);
4     int k = 0;
5     for (int i = 1; i < n; i++) {
6         while (k > 0 && !(s[i] == s[k])) {
7             k = p[k - 1];
8         }
9         if (s[i] == s[k]) {
10            k++;
11        }
12        p[i] = k;
13    }
14    return p;
15 }
16
17 template <typename T>
18 std::vector<int> kmp_table(const T &s) {
19     return kmp_table((int) s.size(), s);
20 }
21
22 template <typename T>
23 std::vector<int> kmp_search(int n, const T &s, int m, const T &w, const std::vector<int>
24 > &p) {
25     assert(n >= 1 && (int) p.size() == n);
26     std::vector<int> res;
27     int k = 0;
28     for (int i = 0; i < m; i++) {
29         while (k > 0 && (k == n || !(w[i] == s[k]))) {
30             k = p[k - 1];
31         }
32         if (w[i] == s[k]) {
33             k++;
34         }
35     }
36 }

```

```

34         if (k == n) {
35             res.push_back(i - n + 1);
36         }
37     }
38     return res;
39     // returns 0-indexed positions of occurrences of s in w
40 }
41
42 template <typename T>
43 std::vector<int> kmp_search(const T &s, const T &w, const std::vector<int> &p) {
44     return kmp_search((int) s.size(), s, (int) w.size(), w, p);
45 }

```

1.3 automaton

```

1 struct Automaton {
2     static constexpr int ALPHABET_SIZE = 26;
3     std::vector<std::vector<int>>> tr;
4     std::vector<int> e;
5     std::vector<int> fail;
6     int tot;
7
8     Automaton(int n) : tr(n, std::vector<int>(ALPHABET_SIZE)), e(n), fail(n), tot(0) {}
9     Automaton(int m, std::vector<std::string> s) : Automaton(m) {
10         for(int i = 0; i < (int) s.size(); i++) {
11             insert(s[i]);
12         }
13         build();
14     }
15
16     void insert(std::string s) {
17         int u = 0;
18         for(int i = 0; i < (int) s.size(); i++) {
19             if(!tr[u][s[i] - 'a']) tr[u][s[i] - 'a'] = ++tot;
20             u = tr[u][s[i] - 'a'];
21         }
22         e[u]++;
23     }
24
25     void build() {
26         std::queue<int> q;
27         for(int i = 0; i < 26; i++) {
28             if(tr[0][i]) q.push(tr[0][i]);
29         }
30         while(q.size()) {
31             int u = q.front();
32             q.pop();
33             for(int i = 0; i < 26; i++) {
34                 if(tr[u][i]) {
35                     fail[tr[u][i]] = tr[fail[u]][i];
36                     q.push(tr[u][i]);
37                 } else {
38                     tr[u][i] = tr[fail[u]][i];
39                 }
40             }
41         }
42     }
43 }

```



```

44     int query(std::string t) {
45         int u = 0, res = 0;
46         for(int i = 0; i < (int) t.size(); i++) {
47             u = tr[u][t[i] - 'a'];
48             for(int j = u; j && e[j] != -1; j = fail[j]) {
49                 res += e[j], e[j] = -1;
50             }
51         }
52         return res;
53     }
54 };

```

1.4 manacher

```

1  template <typename T>
2  std::vector<int> manacher(int n, const T &s) {
3      if (n == 0) {
4          return std::vector<int>();
5      }
6      std::vector<int> res(2 * n - 1, 0);
7      int l = -1, r = -1;
8      for (int z = 0; z < 2 * n - 1; z++) {
9          int i = (z + 1) >> 1;
10         int j = z >> 1;
11         int p = (i >= r ? 0 : std::min(r - i, res[2 * (l + r) - z]));
12         while (j + p + 1 < n && i - p - 1 >= 0) {
13             if (!(s[j + p + 1] == s[i - p - 1])) {
14                 break;
15             }
16             p++;
17         }
18         if (j + p > r) {
19             l = i - p;
20             r = j + p;
21         }
22         res[z] = p;
23     }
24     return res;
25 }
26
27 template <typename T>
28 std::vector<int> manacher(const T &s) {
29     return manacher((int) s.size(), s);
30 }

```

1.5 Z-function

```

1  template <typename T>
2  std::vector<int> z_function(int n, const T &s) {
3      std::vector<int> z(n, 0);
4      int l = 0, r = 0;
5      for (int i = 1; i < n; i++) {
6          z[i] = (i > r ? 0 : std::min(r - i + 1, z[i - l]));
7          while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
8              z[i]++;
9          }
10         if (i + z[i] - 1 > r) {

```

```

11         l = i;
12         r = i + z[i] - 1;
13     }
14 }
15 return z;
16 }
17
18 template <typename T>
19 std::vector<int> z_function(const T &s) {
20     return z_function((int) s.size(), s);
21 }
22
23 template <typename T>
24 std::vector<int> z_search(int n, const T &s, int m, const T &w, std::vector<int> &z) {
25     assert(n >= 1 && (int) z.size() == n);
26     std::vector<int> p(m);
27     int l = 0, r = -1;
28     for (int i = 0; i < m; i++) {
29         p[i] = (i > r ? 0 : std::min(r - i + 1, z[i - l]));
30         while (i + p[i] < m && p[i] < n && s[p[i]] == w[i + p[i]]) {
31             p[i]++;
32         }
33         if (i + p[i] - 1 > r) {
34             l = i;
35             r = i + p[i] - 1;
36         }
37     }
38     return p;
39 };
40
41 template <typename T>
42 std::vector<int> z_search(const T &s, const T &w, std::vector<int> &z) {
43     return z_search((int) s.size(), s, (int) w.size(), w, z);
44 };

```

1.6 suffix-array

```

1 template <typename T>
2 std::vector<int> suffix_array(int n, const T &s, int char_bound) {
3     std::vector<int> a(n);
4     if (n == 0) {
5         return a;
6     }
7     if (char_bound != -1) {
8         std::vector<int> aux(char_bound, 0);
9         for (int i = 0; i < n; i++) {
10             aux[s[i]]++;
11         }
12         int sum = 0;
13         for (int i = 0; i < char_bound; i++) {
14             int add = aux[i];
15             aux[i] = sum;
16             sum += add;
17         }
18         for (int i = 0; i < n; i++) {
19             a[aux[s[i]]++] = i;
20         }
21     } else {

```

```

22     iota(a.begin(), a.end(), 0);
23     sort(a.begin(), a.end(), [&s](int i, int j) { return s[i] < s[j]; });
24 }
25 std::vector<int> sorted_by_second(n);
26 std::vector<int> ptr_group(n);
27 std::vector<int> new_group(n);
28 std::vector<int> group(n);
29 group[a[0]] = 0;
30 for (int i = 1; i < n; i++) {
31     group[a[i]] = group[a[i - 1]] + (!s[a[i]] == s[a[i - 1]]));
32 }
33 int cnt = group[a[n - 1]] + 1;
34 int step = 1;
35 while (cnt < n) {
36     int at = 0;
37     for (int i = n - step; i < n; i++) {
38         sorted_by_second[at++] = i;
39     }
40     for (int i = 0; i < n; i++) {
41         if (a[i] - step >= 0) {
42             sorted_by_second[at++] = a[i] - step;
43         }
44     }
45     for (int i = n - 1; i >= 0; i--) {
46         ptr_group[group[a[i]]] = i;
47     }
48     for (int i = 0; i < n; i++) {
49         int x = sorted_by_second[i];
50         a[ptr_group[group[x]]++] = x;
51     }
52     new_group[a[0]] = 0;
53     for (int i = 1; i < n; i++) {
54         if (group[a[i]] != group[a[i - 1]]) {
55             new_group[a[i]] = new_group[a[i - 1]] + 1;
56         } else {
57             int pre = (a[i - 1] + step >= n ? -1 : group[a[i - 1] + step]);
58             int cur = (a[i] + step >= n ? -1 : group[a[i] + step]);
59             new_group[a[i]] = new_group[a[i - 1]] + (pre != cur);
60         }
61     }
62     swap(group, new_group);
63     cnt = group[a[n - 1]] + 1;
64     step <<= 1;
65 }
66 return a;
67 }
68
69 template <typename T>
70 std::vector<int> suffix_array(const T &s, int char_bound) {
71     return suffix_array((int) s.size(), s, char_bound);
72 }
73
74 template <typename T>
75 std::vector<int> build_lcp(int n, const T &s, const std::vector<int> &sa) {
76     assert((int) sa.size() == n);
77     std::vector<int> pos(n);
78     for (int i = 0; i < n; i++) {
79         pos[sa[i]] = i;
80     }

```

```

81     std::vector<int> lcp(std::max(n - 1, 0));
82     int k = 0;
83     for (int i = 0; i < n; i++) {
84         k = std::max(k - 1, 0);
85         if (pos[i] == n - 1) {
86             k = 0;
87         } else {
88             int j = sa[pos[i] + 1];
89             while (i + k < n && j + k < n && s[i + k] == s[j + k]) {
90                 k++;
91             }
92             lcp[pos[i]] = k;
93         }
94     }
95     return lcp;
96 }
97
98 template <typename T>
99 std::vector<int> build_lcp(const T &s, const std::vector<int> &sa) {
100     return build_lcp((int) s.size(), s, sa);
101 }

```

1.7 suffixAutomaton

```

1  struct SuffixAutomaton {
2      static constexpr int ALPHABET_SIZE = 26, N = 1e4;
3      struct Node {
4          int len;
5          int link;
6          int next[ALPHABET_SIZE];
7          int siz;
8          Node() : len(0), link(0), next{} {}
9      } t[2 * N];
10     int cntNodes;
11     SuffixAutomaton() {
12         cntNodes = 1;
13         std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
14         t[0].len = -1;
15     }
16     int extend(int p, int c) {
17         if (t[p].next[c]) {
18             int q = t[p].next[c];
19             if (t[q].len == t[p].len + 1)
20                 return q;
21             int r = ++cntNodes;
22             t[r].len = t[p].len + 1;
23             t[r].link = t[q].link;
24             std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
25             t[q].link = r;
26             while (t[p].next[c] == q) {
27                 t[p].next[c] = r;
28                 p = t[p].link;
29             }
30             return r;
31         }
32         int cur = ++cntNodes;
33         t[cur].len = t[p].len + 1; t[cur].siz++;
34         while (!t[p].next[c]) {

```

```
35         t[p].next[c] = cur;
36         p = t[p].link;
37     }
38     t[cur].link = extend(p, c);
39     return cur;
40 }
41 };
```

2 动态规划

2.1 01knapSack

```

1 template <typename T>
2 T knapSack(const std::vector<int>& v, const std::vector<T>& w, int V) {
3     int n = (int) v.size();
4     std::vector<T> dp(V + 1);
5     for (int i = 0; i < n; i++) {
6         for (int j = V; j >= v[i]; j--) {
7             dp[j] = std::max(dp[j], dp[j - v[i]] + w[i]);
8         }
9     }
10    return dp.back();
11 }

```

2.2 simpleBoundedKnapSack

```

1 template <typename T>
2 T simpleMultipleKnapSack(const std::vector<int>& v, const std::vector<T>& w, const std
   ::vector<int>& cnt, int V) {
3     int n = (int) v.size();
4     std::vector<T> dp(V + 1);
5     for (int i = 0; i < n; i++) {
6         for (int j = V; j >= 0; j--) {
7             for (int k = 1; k <= cnt[i]; k++) {
8                 if (j - k * v[i] < 0) break;
9                 dp[j] = std::max(dp[j], dp[j - k * v[i]] + k * w[i]);
10            }
11        }
12    }
13    return dp.back();
14 }

```

2.3 binaryBoundedKnapSack

```

1 template <typename T>
2 T binaryMultipleKnapSack(const std::vector<int>& v, const std::vector<T>& w, const std
   ::vector<int>& cnt, int V) {
3     int n = (int) v.size();
4     std::vector<T> dp(V + 1);
5     std::vector<std::array<T, 2>> bags;
6     for (int i = 0; i < n; i++) {
7         for (int k = 1; k <= cnt[i]; k <= 1) {
8             cnt[i] -= k;
9             bags.push_back({v[i] * k, w[i] * k});
10        }
11        if (cnt[i] > 0) {
12            bags.push_back({v[i] * cnt[i], w[i] * cnt[i]});
13        }
14    }
15    for (auto& [nv, nw] : bags) {
16        for (int j = V; j >= nv; j--) {
17            dp[j] = std::max(dp[j], dp[j - nv] + nw);
18        }
19    }
20    return dp.back();

```

21 }

2.4 monotonousQueueMultipleKnapSack

```

1 // no correct
2 template <typename T>
3 T monotonousQueueMultipleKnapSack(const std::vector<int>& v, const std::vector<T>& w,
4   const std::vector<int>& cnt, int V) {
5     int n = (int) v.size();
6     std::vector<T> dp(V + 1);
7     std::vector<int> que(V);
8     for (int i = 0; i < n; i++) {
9       std::vector<T> ndp(V + 1);
10      for (int j = 0; j < v[i]; j++) {
11        int head = 1, tail = 0;
12        for (int k = j; k <= V; k += v[i]) {
13          while (head <= tail && que[head] < k - cnt[i] * v[i]) head++;
14          if (head <= tail) ndp[k] = std::max(dp[k], dp[que[head]] + (k - que[
15            head]) / v[i] * w[i]);
16          while (head <= tail && dp[k] >= dp[que[tail]] + (k - que[tail]) / v[i]
17            * w[i]) tail--;
18          que[++tail] = k;
19        }
20      }
21      dp = std::move(ndp);
22    }
23    return dp.back();
24  }

```

2.5 unboundedKnapSack

```

1 template <typename T>
2 T entireKnapSack(const std::vector<int>& v, const std::vector<T>& w, int V) {
3   int n = (int) v.size();
4   std::vector<T> dp(V + 1);
5   for (int i = 0; i < n; i++) {
6     for (int j = v[i]; j <= V; j++) {
7       dp[j] = std::max(dp[j], dp[j - v[i]] + w[i]);
8     }
9   }
10  return dp.back();
11 }

```

2.6 hybridKnapSack

```

1 template <typename T>
2 T hybridKnapSack(const std::vector<int>& v, const std::vector<T>& w, const std::vector<
3   int>& cnt, int V) {
4   int n = (int) v.size();
5   std::vector<T> dp(V + 1);
6   std::vector<std::tuple<int, T, int>> bags;
7   for (int i = 0; i < n; i++) {
8     if (cnt[i] < 0) {
9       bags.push_back(std::make_tuple(v[i], w[i], -1));
10    } else if (cnt[i] == 0) {
11      bags.push_back(std::make_tuple(v[i], w[i], 0));
12    }
13  }
14  return dp.back();
15 }

```

```

11     } else {
12         for (int k = 1; k <= cnt[i]; k <= 1) {
13             cnt[i] -= k;
14             bags.push_back(std::make_tuple(v[i] * k, w[i] * k, -1));
15         }
16         if (cnt[i] > 0) {
17             bags.push_back(std::make_tuple(v[i] * cnt[i], w[i] * cnt[i], -1));
18         }
19     }
20 }
21 for (auto& bag : bags) {
22     int v, op; T w;
23     std::tie(v, w, op) = bag;
24     if (op == -1) {
25         for (int j = V; j >= v; j--) {
26             dp[j] = std::max(dp[j], dp[j - v] + w);
27         }
28     } else {
29         for (int j = v; j <= V; j++) {
30             dp[j] = std::max(dp[j], dp[j - v] + w);
31         }
32     }
33 }
34 return dp.back();
35 }

```

2.7 groupKnapSack

```

1 template <typename T>
2 T groupKnapSack(const std::vector<std::vector<std::array<int, 2>>& groups, int V) {
3     std::vector<T> dp(V + 1);
4     for (auto& group : groups) {
5         for (int j = V; j >= 0; j--) {
6             for (auto& [v, w] : group) {
7                 if (j >= v) {
8                     dp[j] = std::max(dp[j], dp[j - v] + w);
9                 }
10                /*
11                 if (j >= v) {
12                     // from groups[i], ensure dp[i][j-v] has at least one item because
13                     dp[i][j-v] is not -1
14                     if (~dp[i][j - v]) dp[i][j] = std::max(dp[i][j], dp[i][j - v] + w);
15                     if (~dp[i - 1][j - v]) dp[i][j] = std::max(dp[i][j], dp[i - 1][j -
16                     v] + w);
17                     // from groups[i - 1], ensure dp[i-1][j-v] has at least one item
18                     because dp[i-1][j-v] is not -1
19                     // the order is not swap
20                 }
21                */
22            }
23        }
24    }
25    return dp.back();
26 }

```

2.8 KnapSack2d


```

1 template <typename T>
2 T twoDimensionalKnapSack(const std::vector<int>& v, const std::vector<int>& m, const
   std::vector<T>& w, int V, int M) {
3     int n = (int) v.size();
4     std::vector<std::vector<T>> dp(V + 1, std::vector<T>(M + 1));
5     for (int i = 0; i < n; i++) {
6         for (int j = V; j >= v[i]; j--) {
7             for (int k = M; k >= m[i]; k--) {
8                 dp[j][k] = std::max(dp[j][k], dp[j - v[i]][k - m[i]] + w[i]);
9             }
10        }
11    }
12    return dp.back().back();
13 }

```

2.9 treeKnapSack

```

1 template <typename T>
2 T treeKnapSack(const forest<T>& g) {
3     std::vector<int> siz(n);
4     std::vector<std::vector<T>> dp(n);
5
6     std::function<void(int, int)> dfs = [&](int u, int fa) {
7         siz[u] = 1;
8         for (int id : g.g[u]) {
9             auto& e = g.edges[id];
10            int to = e.from ^ e.to ^ u;
11            if (to == fa) continue;
12            dfs(to, u);
13            int now = min(siz[u] + siz[v] + 1, M);
14            int t[MAX_M]; for (int i = 0; i <= M; i++) t[i] = INF/-INF; //初始化
15            for (int i = 0; i <= siz[u]; i++)
16                for (int j = 0; j <= siz[v] && i + j <= M; j++) {
17                    //...转移方程
18                }
19            for (int i = 0; i <= now; i++) f[u][i] = min/max(f[u][i], t[i]);
20            siz[u] = now;
21        }
22    };
23 }

```

2.10 LIS

```

1 template <typename T>
2 int lis(const std::vector<T>& a) {
3     std::vector<T> u;
4     for (const T& x : a) {
5         auto it = upper_bound(u.begin(), u.end(), x);
6         if (it == u.end()) {
7             u.push_back(x);
8         } else {
9             *it = x;
10        }
11    }
12    return (int) u.size();
13 }

```

2.11 digitDP

```

1 // the numbers of exist 49 in 0~n
2 template <typename T>
3 T digitDP(T n) {
4     T x = n;
5     std::vector<int> digit;
6     while (x) {
7         digit.push_back(x % 10);
8         x /= 10;
9     }
10
11     std::vector<std::vector<T>> dp(digit.size(), std::vector<T>(10));
12     std::function<T(int, int, bool limit)> dfs = [&](int pos, int pre, bool limit) ->
13     int {
14         if (pos == -1) return 1;
15         if (!limit && dp[pos][pre]) return dp[pos][pre];
16         int up = limit ? digit[pos] : 9;
17         T ans = 0;
18         for (int i = 0; i <= up; i++) {
19             if (pre == 4 && i == 9) {
20                 continue;
21             }
22             ans += dfs(pos - 1, i, limit && i == digit[pos]);
23         }
24         if (!limit) dp[pos][pre] = ans;
25     };
26     return dfs(digit.size() - 1, 0, 1);
27 }

```

2.12 bitmaskingDP

```

1 // 杭电1565
2
3 int n;
4 int a[22][22];
5 int dp[22][1 << 18]; // 第一维是行数, 第二位是该行的方案数, 继承了前面所有行数的方案数
6 int tot[1 << 18]; // 方案数
7
8 int calc(int i, int k)
9 {
10     int cnt = 1, res = 0;
11     while(k)
12     {
13         if(k & 1) res += a[i][cnt];
14         k >>= 1;
15         cnt++;
16     }
17     return res;
18 }
19
20 void solve()
21 {
22     while(cin >> n) {
23         mem(dp, 0);
24         int cnt = 0;
25     }

```

```

26     for (int i = 0; i <= (1 << n) - 1; i++) { // 预处理
27         if ((i & (i >> 1)) == 0) // 判断i这个二进制是否满足相邻没有两个1的条件
28             tot[++cnt] = i;
29     }
30
31     for (int i = 1; i <= n; i++)
32         for (int j = 1; j <= n; j++)
33             cin >> a[i][j];
34
35     for (int i = 1; i <= n; i++) { // 行遍历
36         for (int k = 1; k <= cnt; k++) { // 第i行k的二进制排列的数, 与下面的j进行&
37             int val = calc(i, tot[k]); // 计算k的二进制中1所在a数组里的权值
38             for (int j = 1; j <= cnt; j++) { // 第i-1行j的二进制排列的数, 与上面的k进行&
// 并进行状态转移
39                 if ((tot[j] & tot[k]) == 0)
40                     dp[i][k] = max(dp[i][k], dp[i - 1][j] + val);
41             }
42         }
43     }
44
45     int ans = -1;
46     for (int j = 1; j <= cnt; j++)
47         ans = max(ans, dp[n][j]);
48
49     cout << ans << endl;
50 }
51 }

```

2.13 quadrilateralOptimization

```

1 // 四边形优化区间dp( $n^3 \rightarrow n^2$ )
2 //  $a < b < c < d$ ,  $f[l][r] = \min(f[l][k] + f[k + 1][r] + \text{cost}(l, r))$ 
3 // 1.  $\text{cost}(b, c) \leq \text{cost}(a, d)$ 
4 // 2.  $\text{cost}(a, c) + \text{cost}(b, d) \leq \text{cost}(a, d) + \text{cost}(b, c)$ , 即交叉小于包含
5
6 template <typename T>
7 void quadrilateralOptimization() {
8     for (int len = 2; len <= n; len++) {
9         for (int l = 1, r; l + len - 1 <= n; l++) {
10             r = l + len - 1;
11             mn[l][r] = 0x3f3f3f3f;
12             for (int k = m[l][r - 1]; k <= m[l + 1][r]; k++)
13                 if (mn[l][k] + mn[k + 1][r] + cost(l, r) < mn[l][r]) {
14                     mn[l][r] = mn[l][k] + mn[k + 1][r] + cost(l, r);
15                     m[l][r] = k;
16                 }
17         }
18     }
19 }

```

2.14 baseRingTreeDP

```

1 int flag, S, E; // flag是否找到环, SE为环上两个点
2
3 void findCircle(int u, int fa) {
4     vis[u] = 1;
5     for (int i = head[u], v; i; i = e[i].nxt)

```

```

6         if ((v = e[i].to) != fa) {
7             if (vis[v]) flag = 1, S = u, E = v;
8             else findCircle(v, u);
9         }
10    }
11
12    void dp(int u, int fa) {
13        //dp过程
14
15        for (int i = head[u], v; i; i = e[i].nxt)
16            if ((v = e[i].to) != fa && v) {
17                dp(v, u);
18            }
19    }
20 }
21
22 ll calc(int u) {
23     flag = 0;
24     findCircle(u, 0);
25     if (flag) {
26         for (int i = head[S], v; i; i = e[i].nxt)
27             if ((v = e[i].to) == E) {
28                 e[i].to = e[i ^ 1].to = 0; //删边操作, 注意e[tot]中tot从2开始
29                 break;
30             }
31         ll res = 0;
32         dp(S, 0); res = max(res, ...);
33         dp(E, 0); res = max(res, ...);
34         return res;
35     }
36     else {
37         dp(u, 0);
38         return ...;
39     }
40 }

```

2.15 segmentTreeDP

```

1  #define lc u << 1
2  #define rc u << 1 | 1
3  #define mid (t[u].l + t[u].r) / 2
4
5  const int N = 3e5 + 10;
6  const int K = 100 + 10;
7  int n, k;
8  int dp[N][K];
9
10 struct Tree {
11     int l, r;
12     int mx;
13     int tag;
14 } t[N << 2];
15
16 inline void push_up(int u) {
17     t[u].mx = max(t[lc].mx, t[rc].mx);
18 }
19
20 inline void push_down(int u) {

```

```

21     if(!t[u].tag) return ;
22     t[lc].tag += t[u].tag;
23     t[rc].tag += t[u].tag;
24     t[lc].mx  += t[u].tag;
25     t[rc].mx  += t[u].tag;
26     t[u].tag = 0;
27 }
28
29 void build(int u, int l, int r, int k) {
30     t[u].l = l; t[u].r = r;
31     t[u].tag = t[u].mx = 0;
32     if(l == r) {
33         t[u].mx = dp[l][k];
34         return ;
35     }
36     int m = (l + r) / 2;
37     build(lc, l, m, k);
38     build(rc, m + 1, r, k);
39     push_up(u);
40 }
41
42 void modify(int u, int ql, int qr, int val) {
43     if(ql <= t[u].l && t[u].r <= qr) {
44         t[u].mx += val;
45         t[u].tag += val;
46         return ;
47     }
48     int ans = -INF;
49     push_down(u);
50     if(ql <= mid) modify(lc, ql, qr, val);
51     if(qr > mid) modify(rc, ql, qr, val);
52     push_up(u);
53 }
54
55 int query(int u, int ql, int qr) {
56     if(ql <= t[u].l && t[u].r <= qr) return t[u].mx;
57     push_down(u);
58     int ans = 0;
59     if(ql <= mid) ans = max(ans, query(lc, ql, qr));
60     if(qr > mid) ans = max(ans, query(rc, ql, qr));
61     return ans;
62 }
63
64 void solve() {
65     cin >> n >> k;
66     vector<int> a(n + 1), pre(n + 1), from(n + 1);
67     set<int> s;
68     for(int i = 1; i <= n; i++) {
69         cin >> a[i];
70         s.insert(a[i]);
71         from[i] = pre[a[i]]; pre[a[i]] = i;
72         dp[i][1] = (int)s.size();
73     }
74     for(int i = 2; i <= k; i++) {
75         build(1, 1, n, i - 1);
76         for(int j = i; j <= n; j++) {
77             modify(1, from[j], j - 1, 1);
78             dp[j][i] = max(dp[j - 1][i - 1] + 1, query(1, i - 1, j - 1));
79         }
80     }

```

```
80     }
81     cout << dp[n][k] << endl;
82 }
```

2.16 LCS

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int main() {
5      int n; cin >> n;
6      vector<int> p(n + 1), q(n + 1);
7      for(int i = 1; i <= n; i++) cin >> p[i];
8      for(int i = 1; i <= n; i++) cin >> q[i];
9
10     vector<int> id(n + 1);
11     for(int i = 1; i <= n; i++) id[q[i]] = i;
12     vector<int> lis;
13     // 将b中与a中的元素相关的位置, 从大到小push进lis中
14     for(int i = 1; i <= n; i++) {
15         vector<int> d;
16         for(int j = p[i]; j <= n; j += p[i]) {
17             d.emplace_back(id[j]);
18         }
19         sort(d.begin(), d.end(), greater<int>());
20         for(auto x : d) lis.emplace_back(x);
21     }
22
23     vector<int> f;
24     f.push_back(lis[0]);
25     for(int i = 1; i < (int)lis.size(); i++) {
26         if(lis[i] > f.back()) f.push_back(lis[i]);
27         else {
28             int pos = lower_bound(f.begin(), f.end(), lis[i]) - f.begin();
29             f[pos] = lis[i];
30         }
31     }
32
33     cout << f.size() << endl;
34 }
```

3 数据结构

3.1 BTree

```

1  template<class T>
2
3  struct TreeNode {
4      T value;
5      TreeNode *left;
6      TreeNode *right;
7  };
8
9  template<class T>
10 TreeNode<T> *createTree(const T *pre, const T *in, const int len) {
11     TreeNode<T> *t = NULL;
12     if (len > 0) {
13         t = new TreeNode<T>;
14         t->value = pre[0];
15         int index;
16         for (index = 0; index < len; index++) {
17             if (in[index] == pre[0]) {
18                 break;
19             }
20         }
21         if (index == len) {
22             index = -1;
23         }
24         t->left = createTree(pre + 1, in, index);
25         t->right = createTree(pre + index + 1, in + index + 1, len - index - 1);
26     }
27     return t;
28 }
29
30 template<class T>
31 int preOrder(TreeNode<T> *root, queue<T> &out) {
32     if (root) {
33         int count = 1;
34         out.push(root->value);
35         count += preOrder(root->left, out);
36         count += preOrder(root->right, out);
37         return count;
38     } else {
39         return 0;
40     }
41 }
42
43 template<class T>
44 int inOrder(TreeNode<T> *root, queue<T> &out) {
45     if (root) {
46         int count = 1;
47         count += inOrder(root->left, out);
48         out.push(root->value);
49         count += inOrder(root->right, out);
50         return count;
51     } else {
52         return 0;
53     }
54 }
55

```

```

56 template<class T>
57 void postOrder(TreeNode<T> *root, queue<T> &out) {
58     if (root) {
59         postOrder(root->left, out);
60         postOrder(root->right, out);
61         out.push(root->value);
62     } else {
63         return;
64     }
65 }
66
67 template<class T>
68 T *convertQueueToArray(queue<T> &out, int len) {
69     T *list = new T[len];
70     int now = 0;
71     while (!out.empty() && now < len) {
72         list[now] = out.front();
73         out.pop();
74         now++;
75     }
76     return list;
77 }
78
79 template<class T>
80 void destroyTree(TreeNode<T> *root) {
81     if (root) {
82         destroyTree(root->left);
83         destroyTree(root->right);
84         delete root;
85     } else return;
86 }
87
88 template<class T>
89 void insertIntoBSTree(TreeNode<T> *root, const T &value) {
90     if (!root) {
91         return;
92     }
93     if (value < root->value) {
94         if (root->left) {
95             insertIntoTree(root->left, value);
96         } else {
97             root->left = new TreeNode<T>;
98             root->left->value = value;
99             root->left->left = NULL;
100             root->left->right = NULL;
101         }
102     } else if (value > root->value) {
103         if (root->right) {
104             insertIntoTree(root->right, value);
105         } else {
106             root->right = new TreeNode<T>;
107             root->right->value = value;
108             root->right->left = NULL;
109             root->right->right = NULL;
110         }
111     }
112 }
113
114 template<class T>

```



```

115 TreeNode<T> *createBSTree(T *list, int len) {
116     if (len < 1) {
117         return NULL;
118     }
119     TreeNode<T> *root = new TreeNode<char>;
120     root->value = list[0];
121     root->left = NULL;
122     root->right = NULL;
123     for (int i = 1; i < len; i++) {
124         insertIntoBSTree(root, list[i]);
125     }
126     return root;
127 }

```

3.2 pbds-tree

```

1 // RBTREE 红黑树
2 #include <ext/pb_ds/tree_policy.hpp>
3 #include <ext/pb_ds/assoc_container.hpp>
4 // 红黑树
5 __gnu_pbds::tree<int, null_type, less<int>, rb_tree_tag,
6     tree_order_statistics_node_update> t;
7 // null_type无映射(低版本g++为null_mapped_type)
8 // 类似multiset
9 __gnu_pbds::tree<int, null_type, less_equal<int>, rb_tree_tag,
10     tree_order_statistics_node_update> t;
11 find_by_order(size_t order);
12 // 结点更新
13 tree_order_statistics_node_update
14 insert(p);
15 erase(it);
16 // 求k在树中是第几小,假设插入当前值判断当前值是第几小,最小为第0小
17 order_of_key(p);
18 // 找到第order小的迭代器
19 find_by_order(order);
20 // 前驱
21 lower_bound(p);
22 // 后驱
23 upper_bound(p);
24 // 合并
25 a.join(b);
26 // 分割 key小于等于v的元素属于a, 其余的属于b
27 a.split(v, b);
28 // 优先队列
29 #include <ext/pb_ds/priority_queue.hpp>
30 #include <ext/pb_ds/assoc_container.hpp>
31 // 操作类似于stl的优先队列
32 typedef __gnu_pbds::priority_queue<node, greater<node>, __gnu_pbds::thin_heap_tag> heap
33 ;
34 heap::point_iterator; // 指向元素的指针

```

3.3 fenwick

```

1 template <typename T>
2 class fenwick {
3 public:

```

```

4     std::vector<T> fenw;
5     int n;
6
7     fenwick(int _n) : n(_n) {
8         fenw.resize(n);
9     }
10
11    void modify(int x, T v) {
12        while (x < n) {
13            fenw[x] += v;
14            x |= (x + 1);
15        }
16    }
17
18    T get(int x) {
19        T v{};
20        while (x >= 0) {
21            v += fenw[x];
22            x = (x & (x + 1)) - 1;
23        }
24        return v;
25    }
26 };

```

3.4 fenwick2d

```

1  template <typename T>
2  class Fenwick2d {
3  public:
4      std::vector<std::vector<T>> fenw;
5      const int n, m;
6
7      Fenwick2d(int _n, int _m) : n(_n), m(_m) {
8          fenw.resize(n);
9          for (int i = 0; i < n; i++) {
10             fenw[i].resize(m);
11         }
12     }
13
14     inline void modify(int i, int j, T v) {
15         int x = i;
16         while (x < n) {
17             int y = j;
18             while (y < m) {
19                 fenw[x][y] += v;
20                 y |= (y + 1);
21             }
22             x |= (x + 1);
23         }
24     }
25
26     inline T get(int i, int j) {
27         T v{};
28         int x = i;
29         while (x >= 0) {
30             int y = j;
31             while (y >= 0) {
32                 v += fenw[x][y];

```

```
33         y = (y & (y + 1)) - 1;
34     }
35     x = (x & (x + 1)) - 1;
36 }
37 return v;
38 }
39 };
```

3.5 SegmentTree

```
1 struct Info {
2
3 };
4
5 Info operator+(const Info& a, const Info& b) {
6
7 }
8
9 template<class Info,
10     class Merge = std::plus<Info>>
11 struct SegmentTree {
12     const int n;
13     const Merge merge;
14     std::vector<Info> info;
15     SegmentTree(int n) : n(n), merge(Merge()), info(4 * n + 10) {}
16     SegmentTree(std::vector<Info> init) : SegmentTree(init.size()) {
17         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
18             if (r - l == 1) {
19                 info[p] = init[l];
20                 return;
21             }
22             int m = (l + r) / 2;
23             build(2 * p, l, m);
24             build(2 * p + 1, m, r);
25             pull(p);
26         };
27         build(1, 0, n);
28     }
29     void pull(int p) {
30         info[p] = merge(info[2 * p], info[2 * p + 1]);
31     }
32     void modify(int p, int l, int r, int x, const Info &v) {
33         if (r - l == 1) {
34             info[p] = v;
35             return;
36         }
37         int m = (l + r) / 2;
38         if (x < m) {
39             modify(2 * p, l, m, x, v);
40         } else {
41             modify(2 * p + 1, m, r, x, v);
42         }
43         pull(p);
44     }
45     void modify(int p, const Info &v) {
46         modify(1, 0, n, p, v);
47     }
48     Info rangeQuery(int p, int l, int r, int x, int y) {
```

```

49     if (l >= y || r <= x) {
50         return Info();
51     }
52     if (l >= x && r <= y) {
53         return info[p];
54     }
55     int m = (l + r) / 2;
56     return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
57 }
58 Info rangeQuery(int l, int r) {
59     return rangeQuery(1, 0, n, l, r);
60 }
61 };

```

3.6 SegmentTree2d

```

1  #define lc u << 1
2  #define rc u << 1 | 1
3  #define m (l + r) / 2
4
5  const int N = 1e3 + 10;
6
7  struct Tree_y {
8      int mx, mn;
9
10     Tree_y operator + (const Tree_y &rhs) {
11         Tree_y ans;
12         ans.mx = max(mx, rhs.mx);
13         ans.mn = min(mn, rhs.mn);
14         return ans;
15     }
16 };
17
18 int leafx[N], leafy[N];
19
20 struct Tree_x {
21     Tree_y ty[N << 2];
22
23     void build(int u, int l, int r) {
24         ty[u].mx = -INF; ty[u].mn = INF;
25         if (l == r) {
26             leafy[l] = u;
27             return;
28         }
29         build(lc, l, m);
30         build(rc, m + 1, r);
31     }
32
33     Tree_y query(int u, int l, int r, int ql, int qr) {
34         if (qr < l || r < ql) return (Tree_y) {-INF, INF};
35         if (ql <= l && r <= qr) return ty[u];
36         return query(lc, l, m, ql, qr) + query(rc, m + 1, r, ql, qr);
37     }
38 }tx[N << 2];
39
40 int n;
41
42 void build(int u, int l, int r) {

```

```

43     tx[u].build(1, 1, n);
44     if(l == r) {
45         leafx[l] = u;
46         return ;
47     }
48     build(lc, l, m);
49     build(rc, m + 1, r);
50 }
51
52 // (x,y)单点更新, 首先更新叶子节点, 然后向上合并父亲节点
53 void modify(int x, int y, int val) {
54     int valx = leafx[x];
55     int valy = leafy[y];
56     tx[valx].ty[valy].mn = tx[valx].ty[valy].mx = val;
57     for(int i = valx; i; i >>= 1) {
58         for(int j = valy; j; j >>= 1) {
59             if(i == valx && j == valy) continue ;
60             if(j == valy) {
61                 // 如果当前更新的列就是需要更新的叶子节点, 那么由当前行的两个儿子节点来更新
62                 tx[i].ty[j] = tx[i << 1].ty[j] + tx[i << 1 | 1].ty[j];
63             }
64             else {
65                 tx[i].ty[j] = tx[i].ty[j << 1] + tx[i].ty[j << 1 | 1];
66             }
67         }
68     }
69 }
70
71 Tree_y query(int u, int l, int r, int ql, int qr, int qx, int qy) {
72     if(qr < l || r < ql) return (Tree_y) {-INF, INF};
73     if(ql <= l && r <= qr) return tx[u].query(1, 1, n, qx, qy);
74     return query(lc, l, m, ql, qr, qx, qy) + query(rc, m + 1, r, ql, qr, qx, qy);
75 }

```

3.7 ValueSegmentTree

```

1  // 权值线段树, 相当于一个桶, 每个节点用来表示一个区间的数***出现的次数***。
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  #define lc u << 1
8  #define rc u << 1 | 1
9  #define m (l + r) / 2
10 #define mid (t[u].l + t[u].r) / 2
11
12 const int N = 1e5 + 10;
13
14 int t[N << 2];
15
16 int a[N];
17
18 void push_up(int u) {
19     t[u] = t[lc] + t[rc];
20 }
21
22 void build(int u, int l, int r) {

```

```

23     if(l == r) {
24         t[u] = a[l]; // a[l]表示数为l的个数
25         return ;
26     }
27     build(lc, l, m);
28     build(rc, m + 1, r);
29     push_up(u);
30 }
31
32 void update(int u, int l, int r, int k, int cnt) { // k这个数的个数增加cnt
33     if(l == r) {
34         t[u] += cnt;
35         return ;
36     }
37     if(k <= m)
38         update(lc, l, m, k, cnt);
39     else
40         update(rc, m + 1, r, k, cnt);
41     push_up(u);
42 }
43
44 int query(int u, int l, int r, int k) { // 查询k这个数的个数
45     if(l == r) {
46         return t[u];
47     }
48     if(k <= m)
49         return query(lc, l, m, k);
50     else
51         return query(rc, m + 1, r, k);
52 }
53
54 int k_max_th(int u, int l, int r, int k) { // 查询第k大的值
55     if(l == r) {
56         return l;
57     }
58     if(t[lc] >= k) return k_max_th(lc, l, m, k);
59     else return k_max_th(rc, m + 1, r, k - t[lc]);
60 }

```

3.8 线段树动态开点合并分裂

```

1  #define mid (l+r)/2
2  static const int MAX_N = N * 40;
3  int rt[N], now;
4  int lc[MAX_N], rc[MAX_N];
5  ll sum[MAX_N];
6  int tot, rub[MAX_N];
7  int newNode() { return rub[0] ? rub[rub[0]--] : ++tot; }
8  void remove(int &u) {
9      lc[u] = rc[u] = sum[u] = 0;
10     rub[++rub[0]] = u;
11     u = 0;
12 }
13 void push_up(int u) { sum[u] = sum[lc[u]] + sum[rc[u]]; }
14 void build(int &u, int l, int r) {
15     u = newNode();
16     if (l == r) {
17         sum[u] = cnt[l];

```

```

18         return;
19     }
20     build(lc[u], l, mid); build(rc[u], mid + 1, r);
21     push_up(u);
22 }
23 void update(int &u, int l, int r, int p, ll k) {
24     if (!u) u = newNode();
25     if (l == r) {
26         sum[u] += k;
27         return;
28     }
29     if (p <= mid) update(lc[u], l, mid, p, k);
30     else update(rc[u], mid + 1, r, p, k);
31     push_up(u);
32 }
33 ll querySum(int u, int l, int r, int ql, int qr) {
34     if (!u) return 0;
35     if (ql <= l && r <= qr) return sum[u];
36     ll res = 0;
37     if (ql <= mid) res += querySum(lc[u], l, mid, ql, qr);
38     if (qr > mid) res += querySum(rc[u], mid + 1, r, ql, qr);
39     return res;
40 }
41 int queryKth(int u, int l, int r, ll k) {
42     if (l == r) return l;
43     if (k <= sum[lc[u]]) return queryKth(lc[u], l, mid, k);
44     else return queryKth(rc[u], mid + 1, r, k - sum[lc[u]]);
45 }
46 void merge(int u, int v, int l, int r) {
47     if (l == r) {
48         sum[u] += sum[v];
49         return;
50     }
51     if (lc[u] && lc[v]) merge(lc[u], lc[v], l, mid), remove(lc[v]);
52     else if (lc[v]) lc[u] = lc[v], lc[v] = 0;
53     if (rc[u] && rc[v]) merge(rc[u], rc[v], mid + 1, r), remove(rc[v]);
54     else if (rc[v]) rc[u] = rc[v], rc[v] = 0;
55     push_up(u);
56 }
57 void split(int &newp, int &u, int l, int r, int ql, int qr) { //分裂出[ql, qr]间的点
58     if (!u) return;
59     if (ql <= l && r <= qr) {
60         newp = u;
61         u = 0;
62         return;
63     }
64     if (!newp) newp = newNode();
65     if (ql <= mid) split(lc[newp], lc[u], l, mid, ql, qr);
66     if (qr > mid) split(rc[newp], rc[u], mid + 1, r, ql, qr);
67     push_up(u);
68     push_up(newp);
69 }
70 }
71 #undef mid

```

3.9 线段树维护 LIS 方案数

```

1 // 线段树维护序列总LIS的长度mx.fi和方案数mx.se

```

```

2 // 以及对于每个点, 可以存在于多少个LIS种
3
4 // https://nanti.jisuanke.com/t/39611
5
6 namespace Tree_LIS {
7     const int N = 1e6 + 10;
8
9     #define lc t[u].l
10    #define rc t[u].r
11    #define mid (l + r) / 2
12
13    struct Tree {
14        int l, r;
15        ll len; // 长度
16        ll sum; // 个数
17    }t[N << 2];
18
19    int root, cnt;
20    void init() {
21        mem(t, 0);
22        cnt = root = 0;
23    }
24
25    void push_up(int u) {
26        if(t[lc].len == t[rc].len) {
27            t[u].len = t[lc].len;
28            t[u].sum = (t[lc].sum + t[rc].sum) % mod;
29        }
30        else if(t[lc].len < t[rc].len) {
31            t[u].len = t[rc].len;
32            t[u].sum = t[rc].sum;
33        }
34        else {
35            t[u].len = t[lc].len;
36            t[u].sum = t[lc].sum;
37        }
38    }
39
40    void modify(int &u, int l, int r, int p, int le, int su) {
41        if(!u) u = ++cnt;
42        if(l == r) {
43            if(t[u].len == le) t[u].sum = (t[u].sum + su) % mod;
44            else if(t[u].len < le) {
45                t[u].len = le;
46                t[u].sum = su;
47            }
48            return ;
49        }
50        if(!lc) lc = ++cnt;
51        if(!rc) rc = ++cnt;
52        if(p <= mid) modify(lc, l, mid, p, le, su);
53        else modify(rc, mid + 1, r, p, le, su);
54        push_up(u);
55    }
56
57    pll query(int u, int l, int r, int ql, int qr) {
58        if(ql <= l && r <= qr) return pll{t[u].len, t[u].sum};
59        pll lson = {0, 0}, rson = {0, 0};
60        if(!lc) lc = ++cnt;

```



```

61         if(!rc) rc = ++cnt;
62         if(ql <= mid) lson = query(lc, l, mid, ql, qr);
63         if(qr > mid) rson = query(rc, mid + 1, r, ql, qr);
64         if(lson.fi == rson.fi) return pll{lson.fi, (lson.se + rson.se) % mod};
65         else if(lson.fi < rson.fi) return rson;
66         else return lson;
67     }
68 };
69
70 using namespace Tree_LIS;
71
72 ll quick_pow(ll a, ll b) {
73     ll ans = 1;
74     while(b) {
75         if(b & 1) ans = ans * a % mod;
76         a = a * a % mod;
77         b >>= 1;
78     }
79     return ans % mod;
80 }
81
82 void solve() {
83     int n; cin >> n;
84     int L = 0, R = 1e9 + 7;
85     vector<int> a(n + 1);
86     vector<pll> l(n + 1), r(n + 1);
87     for(int i = 1; i <= n; i++) cin >> a[i];
88     init();
89     modify(root, L, R, a[1], 1, 1);
90     l[1] = {1, 1};
91     for(int i = 2; i <= n; i++) {
92         pll temp = query(root, L, R, 0, a[i] - 1);
93         if(temp.fi == 0) temp = {0, 1};
94         modify(root, L, R, a[i], temp.fi + 1, temp.se);
95         l[i] = {temp.fi + 1, temp.se};
96     }
97     pll mx = query(root, L, R, 0, R);
98     init();
99     modify(root, L, R, R - a[n], 1, 1);
100    r[n] = {1, 1};
101    for(int i = n - 1; i >= 1; i--) {
102        pll temp = query(root, L, R, 0, R - a[i] - 1);
103        if(temp.fi == 0) temp = {0, 1};
104        modify(root, L, R, R - a[i], temp.fi + 1, temp.se);
105        r[i] = {temp.fi + 1, temp.se};
106    }
107    for(int i = 1; i <= n; i++) {
108        if(r[i].fi + l[i].fi - 1 == mx.fi) {
109            cout << (r[i].se * l[i].se % mod * quick_pow(mx.se, mod - 2) % mod + mod) %
110            mod << " ";
111        }
112        else cout << 0 << " ";
113    }
114    cout << endl;
115 }

```

3.10 线段树维护最小字典序 LIS

```

1 // 线段树维护LIS输出字典序最小的路径
2 const int N = 1e5 + 10;
3
4 #define lc u << 1
5 #define rc u << 1 | 1
6 #define mid (t[u].l + t[u].r) / 2
7
8 struct Tree {
9     int l, r;
10    int mx;
11    int id;
12 }t[N << 2];
13
14 inline void push_up(int u) {
15     if (t[lc].mx > t[rc].mx)
16         t[u].mx = t[lc].mx, t[u].id = t[lc].id;
17     else if (t[lc].mx < t[rc].mx)
18         t[u].mx = t[rc].mx, t[u].id = t[rc].id;
19     else
20         t[u].mx = t[lc].mx, t[u].id = min(t[lc].id, t[rc].id);
21 }
22
23 void build(int u, int l, int r) {
24     t[u].l = l;
25     t[u].r = r;
26     t[u].mx = t[u].id = 0;
27     if (l == r)
28         return;
29     int m = (l + r) >> 1;
30     build(lc, l, m);
31     build(rc, m + 1, r);
32     push_up(u);
33 }
34
35 void modify(int u, int ql, int qr, int val, int id) {
36     if (ql <= t[u].l && t[u].r <= qr) {
37         if (t[u].mx < val || (t[u].mx == val && t[u].id > id)) {
38             t[u].mx = val;
39             t[u].id = id;
40         }
41         return;
42     }
43     if (ql <= mid)
44         modify(lc, ql, qr, val, id);
45     if (qr > mid)
46         modify(rc, ql, qr, val, id);
47     push_up(u);
48 }
49
50 pii query(int u, int ql, int qr) {
51     if (ql <= t[u].l && t[u].r <= qr)
52         return pii{t[u].mx, t[u].id};
53     pii lson = {-1, -1}, rson = {-1, -1};
54     if (ql <= mid)
55         lson = query(lc, ql, qr);
56     if (qr > mid)
57         rson = query(rc, ql, qr);
58     if (lson.x > rson.x)
59         return lson;

```

```

60     else if (lson.x < rson.x)
61         return rson;
62     else
63         return {lson.x, min(lson.y, rson.y)};
64 }
65
66 void solve() {
67     int n;
68     cin >> n;
69     assert(1 <= n && n <= 1e5);
70     build(1, 1, 1e5);
71     vector<int> a(n + 1), ans(n + 1), fa(n + 1);
72     pii res = {0, 0};
73     for (int i = 1; i <= n; i++) {
74         cin >> a[i];
75         assert(1 <= a[i] && a[i] <= 1e5);
76         if (a[i] == 1) {
77             fa[i] = 0;
78             ans[i] = 1;
79             modify(1, a[i], a[i], 1, i);
80             continue;
81         }
82         pii temp = query(1, 1, a[i] - 1);
83         ans[i] = temp.x + 1;
84         fa[i] = temp.y;
85         modify(1, a[i], a[i], ans[i], i);
86         if (res.x < ans[i])
87             res = pii{ans[i], i};
88     }
89     vector<int> v;
90     int tt = res.second;
91     while (tt) {
92         v.push_back(tt);
93         tt = fa[tt];
94     }
95     cout << v.size() << endl;
96     for (int i = v.size() - 1; i >= 0; i--) {
97         cout << v[i] << (i == 0 ? endl : " ");
98     }
99 }
100
101
102 // 线段树维护LIS方案数

```

3.11 线段树维护插队问题

```

1 // n个人, 每个人a_i要顺序坐在pos_i, 问最终的序列如何
2 // 最后一个人一定坐在自己喜欢坐的位置, 去掉该位置, 倒数第二个人成为最后一个人, 所以就是查找空位置的第
   pos位置
3
4 const int N = 4e5 + 10;
5
6 #define lc u << 1
7 #define rc u << 1 | 1
8 #define mid (l + r) / 2
9 int sum[N << 2], ans[N];
10
11 void push_up(int u) {

```

```

12     sum[u] = sum[lc] + sum[rc];
13 }
14
15 void build(int u, int l, int r) {
16     if(l == r) {
17         sum[u] = 1;
18         ans[l] = 0;
19         return ;
20     }
21     build(lc, l, mid);
22     build(rc, mid + 1, r);
23     push_up(u);
24 }
25
26 void modify(int u, int l, int r, int k, int val) {
27     if(l == r) {
28         ans[l] = val;
29         sum[u] = 0;
30         return ;
31     }
32     if(sum[lc] >= k) modify(lc, l, mid, k, val);
33     else modify(rc, mid + 1, r, k - sum[lc], val);
34     push_up(u);
35 }
36
37 void solve() {
38     int n;
39     while(~scanf("%d", &n)) {
40         vector<pii> p(n + 1);
41         for(int i = 1; i <= n; i++) {
42             scanf("%d%d", &p[i].fi, &p[i].se);
43         }
44         build(1, 1, n);
45         for(int i = n; i >= 1; i--) {
46             modify(1, 1, n, p[i].fi + 1, p[i].se);
47         }
48         for(int i = 1; i <= n; i++) {
49             printf("%d ", ans[i]);
50         }
51         printf("\n");
52     }
53 }

```

3.12 线段树维护连续区间异或值

```

1 // 当 $[l, r]^x$ 时, 很不幸异或出来的结果不一定连续, 而是分成多个连续区间, 所以需要用线段树来构造一个区间异
   // 或 $x$ 之后还是连续区间
2
3 //  $[0, 7]$ 可以分成 $[0, 3]$ 和 $[4, 7]$ , 这样区间异或 $x$ 还是连续区间
4
5 // 主要操作:
6
7 /*
8 把低pos位全为0
9 int ql = (l ^ val) & (((1 << 30) - 1) ^ (1 << pos) - 1);
10 int qr = ql + (1 << pos) - 1;
11 把低pos位全为1
12

```

```

13 高pos不管
14 */
15
16 vector<pii> g[N], len;
17 int l[N], r[N];
18
19 void modify(int pos, int l, int r, int L, int R, int val) {
20     if(L <= l && r <= R) {
21         // 把低pos设置为0
22         int ql = (l ^ val) & (((1 << 30) - 1) ^ (1 << pos) - 1);
23         int qr = ql + (1 << pos) - 1;
24         len.push_back(pii{ql, qr});
25         return ;
26     }
27     int mid = (l + r) / 2;
28     if(L <= mid) modify(pos - 1, l, mid, L, R, val);
29     if(R > mid) modify(pos - 1, mid + 1, r, L, R, val);
30 }
31
32 void dfs(int u, int fa, int w) {
33     modify(30, 0, (1 << 30) - 1, l[u], r[u], w);
34     for(auto e : g[u]) {
35         if(e.fi == fa) continue ;
36         dfs(e.fi, u, e.se ^ w);
37     }
38 }

```

3.13 线段树维护区间异或

```

1  const int N = 2e5 + 10;
2
3  #define lc u << 1
4  #define rc u << 1 | 1
5
6  struct Tree {
7      int sum, tag;
8  } t[21][N << 2];
9  int a[N];
10
11 void push_up(int id, int u) {
12     t[id][u].sum = t[id][lc].sum + t[id][rc].sum;
13 }
14
15 void push_down(int id, int u, int l, int r) {
16     if(!t[id][u].tag) return ;
17     int m = (l + r) / 2;
18     t[id][lc].sum = (m - l + 1) - t[id][lc].sum;
19     t[id][rc].sum = (r - m) - t[id][rc].sum;
20     t[id][lc].tag ^= 1;
21     t[id][rc].tag ^= 1;
22     t[id][u].tag = 0;
23 }
24
25 void build(int id, int u, int l, int r) {
26     if(l == r) {
27         t[id][u].sum = (a[l] >> id) & 1;
28         t[id][u].tag = 0;
29         return ;

```

```

30     }
31     int m = (l + r) / 2;
32     build(id, lc, l, m);
33     build(id, rc, m + 1, r);
34     push_up(id, u);
35 }
36
37 void modify(int id, int u, int l, int r, int ql, int qr) {
38     if(ql <= l && r <= qr) {
39         t[id][u].sum = (r - l + 1) - t[id][u].sum;
40         t[id][u].tag ^= 1;
41         return ;
42     }
43     push_down(id, u, l, r);
44     int m = (l + r) / 2;
45     if(ql <= m) modify(id, lc, l, m, ql, qr);
46     if(qr > m) modify(id, rc, m + 1, r, ql, qr);
47     push_up(id, u);
48 }
49
50 int query(int id, int u, int l, int r, int ql, int qr) {
51     if(ql <= l && r <= qr) return t[id][u].sum;
52     push_down(id, u, l, r);
53     int ans = 0;
54     int m = (l + r) / 2;
55     if(ql <= m) ans += query(id, lc, l, m, ql, qr);
56     if(qr > m) ans += query(id, rc, m + 1, r, ql, qr);
57     return ans;
58 }
59
60 void solve() {
61     int n, m; cin >> n >> m;
62     for(int i = 1; i <= n; i++) cin >> a[i];
63     for(int i = 0; i <= 20; i++) {
64         build(i, 1, 1, n);
65     }
66     while(m--) {
67         int opt; cin >> opt;
68         if(opt == 1) {
69             int l, r; cin >> l >> r;
70             ll ans = 0;
71             for(int i = 0; i <= 20; i++) {
72                 ans += query(i, 1, 1, n, l, r) * (1ll << i);
73             }
74             cout << ans << endl;
75         }
76         else {
77             int l, r, k; cin >> l >> r >> k;
78             for(int i = 0; i <= 20; i++) {
79                 if((k >> i) & 1) modify(i, 1, 1, n, l, r);
80             }
81         }
82     }
83 }

```

3.14 LazySegmentTree

```
2 struct Info {
3
4 };
5
6 struct Tag {
7
8 };
9
10 Info operator+(const Info& a, const Info& b) {
11
12 }
13
14 void apply(Info &a, const Tag &b) {
15
16 }
17
18 void apply(Tag &a, const Tag &b) {
19
20 }
21
22 template<class Info, class Tag,
23         class Merge = std::plus<Info>>
24 struct LazySegmentTree {
25     const int n;
26     const Merge merge;
27     std::vector<Info> info;
28     std::vector<Tag> tag;
29     LazySegmentTree(int n) : n(n), merge(Merge()), info(4 * n + 10), tag(4 * n + 10) {}
30     LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {
31         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
32             if (r - l == 1) {
33                 info[p] = init[l];
34                 return;
35             }
36             int m = (l + r) / 2;
37             build(2 * p, l, m);
38             build(2 * p + 1, m, r);
39             pull(p);
40         };
41         build(1, 0, n);
42     }
43     void pull(int p) {
44         info[p] = merge(info[2 * p], info[2 * p + 1]);
45     }
46     void apply(int p, const Tag &v) {
47         ::apply(info[p], v);
48         ::apply(tag[p], v);
49     }
50     void push(int p) {
51         apply(2 * p, tag[p]);
52         apply(2 * p + 1, tag[p]);
53         tag[p] = Tag();
54     }
55     void modify(int p, int l, int r, int x, const Info &v) {
56         if (r - l == 1) {
57             info[p] = info[p] + v;
58             return;
59         }
60         int m = (l + r) / 2;
```

```

61     push(p);
62     if (x < m) {
63         modify(2 * p, l, m, x, v);
64     } else {
65         modify(2 * p + 1, m, r, x, v);
66     }
67     pull(p);
68 }
69 void modify(int p, const Info &v) {
70     modify(1, 0, n, p, v);
71 }
72 Info rangeQuery(int p, int l, int r, int x, int y) {
73     if (l >= y || r <= x) {
74         return Info();
75     }
76     if (l >= x && r <= y) {
77         return info[p];
78     }
79     int m = (l + r) / 2;
80     push(p);
81     return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
82 }
83 Info rangeQuery(int l, int r) {
84     return rangeQuery(1, 0, n, l, r);
85 }
86 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
87     if (l >= y || r <= x) {
88         return;
89     }
90     if (l >= x && r <= y) {
91         apply(p, v);
92         return;
93     }
94     int m = (l + r) / 2;
95     push(p);
96     rangeApply(2 * p, l, m, x, y, v);
97     rangeApply(2 * p + 1, m, r, x, y, v);
98     pull(p);
99 }
100 void rangeApply(int l, int r, const Tag &v) {
101     return rangeApply(1, 0, n, l, r, v);
102 }
103 };

```

3.15 SparseTable

```

1  // 倍增思想加DP优化
2
3  #include <bits/stdc++.h>
4  using namespace std;
5
6  const int N = 1e5 + 10;
7
8  int a[N];
9
10
11 template <typename T, class F = std::function<T(const T&, const T&)>>
12 class SparseTable {

```



```

13 public:
14     int n;
15     std::vector<std::vector<T>> mat;
16     F func;
17
18     SparseTable(const std::vector<T>& a, const F& f) : func(f) {
19         n = static_cast<int>(a.size());
20         int max_log = 32 - __builtin_clz(n);
21         mat.resize(max_log);
22         mat[0] = a;
23         for (int j = 1; j < max_log; j++) {
24             mat[j].resize(n - (1 << j) + 1);
25             for (int i = 0; i <= n - (1 << j); i++) {
26                 mat[j][i] = func(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
27             }
28         }
29     }
30
31     T get(int from, int to) const {
32         assert(0 <= from && from <= to && to <= n - 1);
33         int lg = 32 - __builtin_clz(to - from + 1) - 1;
34         return func(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
35     }
36 };
37
38 // 二维ST表
39 int f[105][105][105];
40
41 void Two_ST() {
42     int n, m;
43     cin >> n >> m;
44     for(int i = 1; i <= n; i++)
45         for(int j = 1; j <= m; j++){
46             cin >> a[i];
47             f[i][j][0] = a[i];
48         }
49
50 // 大矩阵分成四个小矩阵求最值
51
52     for(int k = 1; k < log2(n); k++) {
53         for(int i = 1; i <= n; i++) {
54             for(int j = 1; j <= m; j++) {
55                 if((i + (1 << (k - 1)) <= n) && (j + (1 << (k - 1)) <= m))
56                     f[i][j][k] = max(max(f[i][j + (1 << (k - 1))][k - 1], f[i + (1 << (k - 1))][j][k - 1]),
57                                     max(f[i][j][k - 1], f[i + (1 << (k - 1))][j + (1 << (k - 1))][k - 1]));
58             }
59         }
60
61         int T;
62         cin >> T;
63         while(T--) {
64             int l, r, k;
65             cin >> l >> r >> k;
66             int len = log2(k);
67             int s = max(max(f[l][r][len], f[l + k - (1 << len)][r + k - (1 << len)][len]),
68                         max(f[l + k - (1 << len)][r][len], f[l][r + k - (1 << len)][len]));
69         }
70     }

```

```
69 }
```

3.16 CartesianTree

```
1 // 笛卡尔树是一种由数列构造的特殊二叉搜索树, 每个节点都有两个键值, first为下标, second为权值
2 // 笛卡尔树满足两个性质, 在下标递增的情况下就是一个大/小根堆
3
4 // 笛卡尔树, 静态建树, 区间最值跳转
5 struct CartesianTree {
6     int rt; // 根节点
7     pii ch[N]; // 左右儿子
8     int st[N]; // 单调栈
9
10    void build(int n, int p[]) {
11        rt = 0;
12        int t = 0;
13        for (int i = 1; i <= n; i++) {
14            ch[i] = {0, 0};
15            // 决定了大于还是小于
16            while (t && p[st[t]] > p[i]) --t;
17            if (t) {
18                // 上一个点的右儿子作为自己的左儿子
19                // 成为上一个点的右儿子
20                ch[i].first = ch[st[t]].second;
21                ch[st[t]].second = i;
22            } else { // 自己作为根节点
23                ch[i].first = rt;
24                rt = i;
25            }
26            st[++t] = i;
27        }
28    }
29 } dika;
```

3.17 DancingLinks

```
1 // Dancing Links
2 struct DLX {
3     int n, m, size;
4     int U[MaxNode], D[MaxNode], L[MaxNode], R[MaxNode], Row[MaxNode], Col[MaxNode];
5     int H[MaxN], S[MaxM];
6     int ansd, ans[MaxN];
7
8     void init(int _n, int _m) {
9         n = _n;
10        m = _m;
11        for (int i = 0; i <= m; i++) {
12            S[i] = 0;
13            U[i] = D[i] = i;
14            L[i] = i - 1;
15            R[i] = i + 1;
16        }
17        R[m] = 0;
18        L[0] = m;
19        size = m;
20        for (int i = 0; i <= n; i++) {
21            H[i] = -1;
```

```

22     }
23 }
24
25 void Link(int r, int c) {
26     ++S[Col[++size] = c];
27     Row[size] = r;
28     D[size] = D[c];
29     U[D[c]] = size;
30     U[size] = c;
31     D[c] = size;
32     if (H[r] < 0) {
33         H[r] = L[size] = R[size] = size;
34     } else {
35         R[size] = R[H[r]];
36         L[R[H[r]]] = size;
37         L[size] = H[r];
38         R[H[r]] = size;
39     }
40 }
41
42 void remove(int c) {
43     L[R[c]] = L[c];
44     R[L[c]] = R[c];
45     for (int i = D[c]; i != c; i = D[i]) {
46         for (int j = R[i]; j != i; j = R[j]) {
47             U[D[j]] = U[j];
48             D[U[j]] = D[j];
49             --S[Col[j]];
50         }
51     }
52 };
53
54 void resume(int c) {
55     for (int i = U[c]; i != c; i = U[i])
56         for (int j = L[i]; j != i; j = L[j])
57             ++S[Col[U[D[j]] = D[U[j]] = j]];
58     L[R[c]] = R[L[c]] = c;
59 }
60
61 bool Dance(int d) {
62     if (R[0] == 0) {
63         for (int i = 0; i < d; i++) {
64             printf("%d%c", ans[i], " \n"[i == d - 1]);
65         }
66         return true;
67     }
68     int c = R[0];
69     for (int i = R[0]; i != 0; i = R[i]) if (S[i] < S[c]) c = i;
70     remove(c);
71     for (int i = D[c]; i != c; i = D[i]) {
72         ans[d] = Row[i];
73         for (int j = R[i]; j != i; j = R[j]) remove(Col[j]);
74         if (Dance(d + 1)) return true;
75         for (int j = L[i]; j != i; j = L[j]) resume(Col[j]);
76     }
77     resume(c);
78     return false;
79 }
80 };

```

3.18 ChthollyTree

```

1 // 要先Split右端点(r+1), 在Split左端点(l)
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 typedef long long ll;
8
9 struct node {
10     int l, r;
11     mutable ll v;
12     node (int L, int R = -1, ll V = 0) : l(L), r(R), v(V) {}
13     bool operator < (const node &rhs) const {
14         return l < rhs.l;
15     }
16 };
17
18 set<node> s;
19
20 auto Split(int pos) {
21     auto it = s.lower_bound(node(pos));
22     if(it != s.end() && it->l == pos) return it;
23     --it;
24     int L = it->l, R = it->r;
25     ll V = it->v;
26     s.erase(it);
27     s.insert(node(L, pos - 1, V));
28     return s.insert(node(pos, R, V)).first;
29 }
30
31 void assign_val(int l, int r, ll val) { // 推平操作
32     auto itr = Split(r + 1);
33     auto itl = Split(l);
34     s.erase(itl, itr);
35     s.insert(node(l, r, val));
36 }
37
38 void add(int l, int r, ll val) { // 区间加
39     auto itr = Split(r + 1);
40     auto itl = Split(l);
41     for( ; itl != itr; ++itl) {
42         itl->v += val;
43     }
44 }
45
46 ll kth(int l, int r, int k) { // 区间第k小
47     vector<pair<ll, int>> v;
48     auto itr = Split(r + 1);
49     auto itl = Split(l);
50     for( ; itl != itr; ++itl) {
51         v.push_back(pair<ll, int>{itl->v, itl->r - itl->l + 1});
52     }
53     sort(v.begin(), v.end());
54     for(auto it = v.begin(); it != v.end(); ++it) {
55         k -= it->second;
56         if(k <= 0) return it->first;
57     }

```

```

58 }
59
60 ll quick_pow(ll a, ll b, ll p) ;
61
62 ll qpow(int l, int r, int ex, int p) {
63     auto itr = Split(r + 1);
64     auto itl = Split(l);
65     ll ans = 0;
66     for( ;itl != itr; ++itl)
67         ans = (ans + ll(itl -> r - itl -> l + 1) * quick_pow(itl -> v, ll(ex), ll(p)) %
68             ll(p)) % ll(p);
69     return ans % ll(p);
70 }
71 int main() {
72     int n, m;
73     cin >> n >> m;
74     for (int i = 1; i <= n; ++i){
75         ll x;
76         cin >> x;
77         s.insert(node(i,i,x));
78     }
79     s.insert(node(n + 1, n + 1, 0));
80     while(m--) {
81         int opt, l, r, x;
82         cin >> opt >> l >> r >> x;
83         if(opt == 1) add(l, r, x);
84         else if(opt == 2) assign_val(l, r, x);
85         else if(opt == 3) cout << kth(l, r, x) << endl;
86         else {
87             int y;
88             cin >> y;
89             cout << qpow(l, r, x, y) << endl;
90         }
91     }
92 }

```

3.19 monotonusQueue

```

1  template <typename T>
2  struct monotonusQueue {
3      std::vector<T> a;
4      monotonusQueue(const std::vector<T>& init) : a(init) {}
5
6      std::vector<T> Max(int k) {
7          int n = (int) a.size();
8          int head = 0, tail = -1;
9          std::queue<int> que(n);
10         std::vector<T> ans;
11         for (int i = 0; i < n; i++) {
12             while (head <= tail && a[que[tail]] <= a[i]) tail--;
13             que[++tail] = i;
14             while (que[head] + k <= i) head++;
15             if (i >= k - 1) ans.push_back(a[que[head]]);
16         }
17         return ans;
18     }
19
20     std::vector<T> Min(int k) {

```

```

21     int n = (int) a.size();
22     int head = 0, tail = -1;
23     std::queue<int> que(n);
24     std::vector<T> ans;
25     for (int i = 0; i < n; i++) {
26         while (head <= tail && a[que[tail]] >= a[i]) tail--;
27         que[++tail] = i;
28         while (que[head] + k <= i) head++;
29         if (i >= k - 1) ans.push_back(a[que[head]]);
30     }
31     return ans;
32 }
33 };

```

3.20 monotonusStack

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct Monotone_stack {
5      static const int N = 1e5 + 10;
6      int a[N];
7      stack<int> s;
8      int n;
9
10     void read() {
11         cin >> n;
12         for(int i = 1; i <= n; i++) cin >> a[i];
13     }
14
15     void Monotone_min() {
16         for(int i = 1; i <= n; i++) {
17             if(s.empty() || s.top() >= a[i])
18                 s.push(a[i]);
19             else {
20                 while(!s.empty() && s.top() < a[i]) {
21                     cout << s.top() << endl;
22                     s.pop();
23                 }
24                 s.push(a[i]);
25             }
26         }
27         while(!s.empty()) {
28             cout << s.top() << endl;
29             s.pop();
30         }
31     }
32
33     void Monotone_max() {
34         for(int i = 1; i <= n; i++) {
35             if(s.empty() || s.top() <= a[i])
36                 s.push(a[i]);
37             else {
38                 while(!s.empty() && s.top() > a[i]) {
39                     cout << s.top() << endl;
40                     s.pop();
41                 }
42                 s.push(a[i]);

```

```

43     }
44     }
45     while(!s.empty()) {
46         cout << s.top() << endl;
47         s.pop();
48     }
49 }
50
51 }Worker;

```

3.21 difference

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  template <typename T>
6  struct difference {
7      int n;
8      std::vector<T> d;
9
10     difference(int _n) : n(_n), d(_n + 1) {}
11     difference(std::vector<T>& init) : difference(init.size()) {
12         d[0] = init[0];
13         for (int i = 1; i < n; i++) {
14             d[i] = init[i] - init[i - 1];
15         }
16     }
17
18     void modify(int l, int r, T v) {
19         assert(0 <= l && l <= n - 1 && 0 <= r && r <= n - 1);
20         d[l] += v;
21         d[r + 1] -= v;
22     }
23
24     void solve() {
25         for (int i = 1; i < n; i++) {
26             d[i] += d[i - 1];
27         }
28     }
29 };
30
31 template <typename T>
32 struct difference2d {
33     int n, m;
34     std::vector<std::vector<T>> d;
35
36     difference2d(int _n, int _m) : n(_n), m(_m), d(_n + 1, std::vector<T>(_m + 1)) {}
37     difference2d(std::vector<std::vector<T>>& init) : difference2d(init.size(), init
38         [0].size()) {
39         for (int i = 0; i < n; i++) {
40             for (int j = 0; j < m; j++) {
41                 if (i == 0 || j == 0) d[i][j] = init[i][j];
42                 else d[i][j] = init[i][j] - init[i - 1][j] - init[i][j - 1] + init[i -
43                     1][j - 1];
44             }
45         }
46     }
47 }

```

```

45
46 void modify(int x1, int y1, int x2, int y2, T v) {
47     assert(0 <= x1 <= n - 1 && 0 <= y1 <= n - 1 && 0 <= x2 <= n - 1 && 0 <= y2 <= n
    - 1);
48     d[x1][y1] += v;
49     d[x1][y2 + 1] -= v;
50     d[x2 + 1][y1] -= v;
51     d[x2 + 1][y2 + 1] += v;
52 }
53 };

```

3.22 trie

```

1 class Trie {
2 private :
3     Trie* next[26] = {nullptr};
4     int val;
5 public :
6     Trie() {}
7
8     void insert(std::string& s) {
9         Trie* root = this;
10        for (char &c : s) {
11            if (root -> next[c] == nullptr) {
12                root -> next[c] = new Trie();
13            }
14            root = root -> next[c];
15        }
16        root -> val ++;
17    }
18
19    void del(std::string& s) {
20        Trie* root = this;
21        for (char &c : s) {
22            root = root -> next[c];
23        }
24        root -> val --;
25    }
26
27    int search(std::string& s) {
28        int ans = 0;
29        Trie* root = this;
30        for (char& c : s) {
31            if (!root -> next[c]) break ;
32            root = root -> next[c];
33            ans += root -> val;
34        }
35        return ans;
36    }
37 };

```

3.23 HashTable

```

1 template<typename T>
2 class HashTable{
3 private :
4     const int maxn;

```



```

5     std::vector<std::vector<T>> key, val;
6
7 public :
8     HashTable(int n) : maxn(n), key(n), val(n) {}
9
10    int hash(int x){
11        return (((long long)x * (x + 1)) ^ x) % maxn;
12    }
13    void insert(int x){
14        int u = hash(x);
15        for(int v = 0; v < (int)key[u].size(); ++v)
16            if(key[u][v] == x){
17                ++val[u][v];
18                return;
19            }
20        key[u].push_back(x), val[u].push_back(1);
21    }
22    T query(int x){
23        int u = hash(x);
24        for(int v = 0; v < (int)key[u].size(); ++v)
25            if(key[u][v] == x)
26                return val[u][v];
27        return 0;
28    }
29 };

```

3.24 2-4 维前缀和

```

1 // 统计(a,b)到(c,d)这个矩阵中的所有0子矩阵
2
3 const int N = 50 + 10;
4 int sum[N][N];
5 int Q[N][N][N][N];
6
7 void solve() {
8     int n, m, q; cin >> n >> m >> q;
9     for(int i = 1; i <= n; i++) {
10         string s; cin >> s;
11         for(int j = 1; j <= m; j++) {
12             sum[i][j] = (s[j - 1] - '0') + sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1];
13         }
14     }
15
16     for(int a = 1; a <= n; a++) {
17         for(int b = 1; b <= m; b++) {
18             for(int c = a; c <= n; c++) {
19                 for(int d = b; d <= m; d++) {
20                     if(sum[c][d] - sum[a - 1][d] - sum[c][b - 1] + sum[a - 1][b - 1] ==
21                        0) {
22                         Q[a][b][c][d]++;
23                     }
24                 }
25             }
26         }
27     }
28     for(int a = n; a >= 1; a--) {

```

```

29     for(int b = m; b >= 1; b--) {
30         for(int c = 1; c <= n; c++) {
31             for(int d = 1; d <= m; d++) {
32                 Q[a][b][c][d] += Q[a + 1][b][c][d];
33             }
34         }
35     }
36 }
37
38 for(int a = n; a >= 1; a--) {
39     for(int b = m; b >= 1; b--) {
40         for(int c = 1; c <= n; c++) {
41             for(int d = 1; d <= m; d++) {
42                 Q[a][b][c][d] += Q[a][b + 1][c][d];
43             }
44         }
45     }
46 }
47
48 for(int a = n; a >= 1; a--) {
49     for(int b = m; b >= 1; b--) {
50         for(int c = 1; c <= n; c++) {
51             for(int d = 1; d <= m; d++) {
52                 Q[a][b][c][d] += Q[a][b][c - 1][d];
53             }
54         }
55     }
56 }
57
58 for(int a = n; a >= 1; a--) {
59     for(int b = m; b >= 1; b--) {
60         for(int c = 1; c <= n; c++) {
61             for(int d = 1; d <= m; d++) {
62                 Q[a][b][c][d] += Q[a][b][c][d - 1];
63             }
64         }
65     }
66 }
67
68 while(q--) {
69     int a, b, c, d; cin >> a >> b >> c >> d;
70     cout << Q[a][b][c][d] << endl;
71 }
72 }

```

3.25 simpleDSU

```

1  class DSU {
2  private :
3      std::vector<int> f, siz;
4      std::vector<int> dep;
5  public :
6      DSU(int n) : f(n), dep(n), siz(n, 1) { iota(f.begin(), f.end(), 0); }
7      int find(int x) {
8          while(x != f[x]) x = f[x] = f[f[x]];
9          return x;
10     }
11     bool same(int x, int y) { return find(x) == find(y); }

```

```

12     bool merge(int x, int y) {
13         x = find(x);
14         y = find(y);
15         if(x == y) return false;
16         if (dep[x] > dep[y]) std::swap(x, y);
17         siz[y] += siz[x];
18         f[x] = y;
19         dep[y] = std::max(dep[y], dep[x] + 1);
20         return true;
21     }
22     int size(int x) { return siz[find(x)]; }
23 };

```

3.26 valueDSU

```

1  class DSU {
2  private :
3      std::vector<int> f, siz, val;
4
5  public :
6      DSU(int n) : f(n), val(n), siz(n, 1) { iota(f.begin(), f.end(), 0); }
7
8      int find(int x) {
9          if (x != f[x]) {
10             int fa = f[x];
11             f[x] = find(f[x]);
12             val[x] += val[fa];
13         }
14         return f[x];
15     }
16     bool same(int x, int y) { return find(x) == find(y); }
17     bool merge(int x, int y, int v) {
18         int nx = find(x);
19         int ny = find(y);
20         if(nx == ny) return false;
21         siz[nx] += siz[ny];
22         f[ny] = nx;
23         val[ny] = val[x] + v - val[y];
24         return true;
25     }
26     int size(int x) { return siz[find(x)]; }
27 };

```

3.27 modifyDSU

```

1  struct node {
2      int x, y, z;
3  };
4
5  struct UnionFind {
6  private:
7      int rk[N], pre[N], siz[N], totNode; //N为最大点数
8      stack<node> st; //node记录上次修改的内容
9  public:
10     void init(int tot) {
11         totNode = tot;
12         for (int i = 1; i <= totNode; i++)

```

```

13         pre[i] = i, siz[i] = rk[i] = 1;
14     }
15     int find(int x) { while (x ^ pre[x]) x = pre[x]; return x; }
16     void merge(int x, int y) { //按秩合并
17         x = find(x), y = find(y);
18         if (x == y) return;
19         if (rk[x] < rk[y]) swap(x, y);
20         st.push(node{ y, rk[x], siz[y] });
21         pre[y] = x, rk[x] += rk[y] == rk[y], siz[x] += siz[y];
22     }
23     int start() { return st.size(); }
24     void end(int last) { //撤回merge操作
25         while (st.size() > last) {
26             node tp = st.top();
27             rk[pre[tp.x]] -= tp.y, siz[pre[tp.x]] -= tp.z;
28             pre[tp.x] = tp.x;
29             st.pop();
30         }
31     }
32     bool judge() { return siz[find(1)] == totNode; }
33 };

```

3.28 varietyDSU

```

1  class DSU {
2  private :
3      std::vector<int> f, siz;
4
5  public :
6      DSU(int n) : f(n), siz(n, 1) { iota(f.begin(), f.end(), 0); }
7      int find(int x) {
8          while (x != f[x]) x = f[x] = f[f[x]];
9          return x;
10     }
11     bool same(int x, int y) { return find(x) == find(y); }
12     bool merge(int x, int y) {
13         x = find(x);
14         y = find(y);
15         if (x == y) return false;
16         siz[x] += siz[y];
17         f[y] = x;
18         return true;
19     }
20     int size(int x) { return siz[find(x)]; }
21 };
22
23 int main() {
24     int n, q;
25     cin >> n >> q;
26     for (int i = 1; i <= 2 * n; i++) f[i] = i; // **
27
28     while (q--) {
29         int flag, x, y;
30         cin >> flag >> x >> y;
31         if (flag) { // 敌人
32             merge(x + n, y);
33             merge(y + n, x);
34         }

```

```

35         else {
36             merge(x, y); // 同伴
37         }
38     }
39     int ans = 0;
40     for(int i = 1; i <= n; i++) {
41         if(f[i] == i) ans ++;
42     }
43     cout << ans << endl;
44 }

```

3.29 KD 求矩阵权值和

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int MAX = 200005;
5  const double alpha = 0.75;
6  //查包含在x1,y1,x2,y2为左下角和右上角的矩形里面权值之和
7  //K-D Tree 二维划分树
8  int n, ans, rt, WD, tot, top, rub[MAX];
9
10 struct node {
11     int x[2], w;
12 } p[MAX];
13
14 struct K_D_tree {
15     int ls, rs, siz, mn[2], mx[2], sum;
16     //mn[0], mx[0] -> x的取值范围
17     //mn[1], mx[1] -> y的取值范围
18     node tmp;
19 } t[MAX];
20
21 int operator < (const node &a, const node &b) { return a.x[WD] < b.x[WD]; }
22
23 int newnode() {
24     if (top) return rub[top--];
25     else return ++tot;
26 }
27
28 void push_up(int u) {
29     for (int i = 0; i <= 1; i++) { //更新x, y的取值范围
30         t[u].mn[i] = t[u].mx[i] = t[u].tmp.x[i];
31         if (t[u].ls) { //左子树的最大最小值
32             t[u].mn[i] = min(t[u].mn[i], t[t[u].ls].mn[i]);
33             t[u].mx[i] = max(t[u].mx[i], t[t[u].ls].mx[i]);
34         }
35         if (t[u].rs) { //右子树的最大最小值
36             t[u].mn[i] = min(t[u].mn[i], t[t[u].rs].mn[i]);
37             t[u].mx[i] = max(t[u].mx[i], t[t[u].rs].mx[i]);
38         }
39     }
40     t[u].sum = t[t[u].ls].sum + t[t[u].rs].sum + t[u].tmp.w;
41     t[u].siz = t[t[u].ls].siz + t[t[u].rs].siz + 1;
42 }
43
44 int build(int l, int r, int wd) {
45     if (l > r) return 0;

```

```

46     int u = newnode();
47     int m = (l + r) >> 1;
48     WD = wd; nth_element(p + l, p + m, p + r + 1);
49     t[u].tmp = p[m];
50     t[u].ls = build(l, m - 1, wd ^ 1);
51     t[u].rs = build(m + 1, r, wd ^ 1);
52     push_up(u);
53     return u;
54 }
55
56 void pia(int u, int num) { //拍扁回炉重做
57     if (t[u].ls) pia(t[u].ls, num);
58     p[t[t[u].ls].siz + num + 1] = t[u].tmp, rub[++top] = u;
59     if (t[u].rs) pia(t[u].rs, t[t[u].ls].siz + num + 1);
60 }
61
62 void check(int &u, int wd) { //检查是否平衡, 不平衡则需要重建
63     if (t[u].siz * alpha < t[t[u].ls].siz || t[u].siz * alpha < t[t[u].rs].siz) pia(u,
64         0), u = build(1, t[u].siz, wd);
65 }
66
67 void insert(int &u, node tp, int wd) { //插入点
68     if (!u) {
69         u = newnode();
70         t[u].ls = t[u].rs = 0, t[u].tmp = tp;
71         push_up(u);
72         return;
73     }
74     if (tp.x[wd] <= t[u].tmp.x[wd]) insert(t[u].ls, tp, wd ^ 1);
75     else insert(t[u].rs, tp, wd ^ 1);
76     push_up(u);
77     check(u, wd);
78 }
79
80 bool in(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) { //完全被包含
81     return (x1 <= X1 && X2 <= x2 && y1 <= Y1 && Y2 <= y2);
82 }
83
84 bool out(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) { //完全无交集
85     return (x1 > X2 || x2 < X1 || y1 > Y2 || y2 < Y1);
86 }
87
88 int query(int u, int x1, int y1, int x2, int y2) {
89     if (!u) return 0;
90     int res = 0;
91     if (in(x1, y1, x2, y2, t[u].mn[0], t[u].mn[1], t[u].mx[0], t[u].mx[1])) return t[u].sum;
92     if (out(x1, y1, x2, y2, t[u].mn[0], t[u].mn[1], t[u].mx[0], t[u].mx[1])) return 0;
93     if (in(x1, y1, x2, y2, t[u].tmp.x[0], t[u].tmp.x[1], t[u].tmp.x[0], t[u].tmp.x[1]))
94         res += t[u].tmp.w;
95     res += query(t[u].ls, x1, y1, x2, y2) + query(t[u].rs, x1, y1, x2, y2);
96     return res;
97 }
98
99 void init() {
100     ans = rt = top = tot = 0;
101 }

```

```

102 void solve() {
103     scanf("%d", &n);
104     while(1) {
105         int opt; scanf("%d",&opt);
106         if(opt == 1) {
107             int x, y, w; scanf("%d%d%d",&x,&y,&w);
108             insert(rt, node{x ^ ans, y ^ ans, w ^ ans}, 0);
109         }
110         else if(opt == 2) {
111             int x1, y1, x2, y2; scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
112             ans = query(rt, x1 ^ ans, y1 ^ ans, x2 ^ ans, y2 ^ ans);
113             printf("%d\n",ans);
114         }
115         else break;
116     }
117 }

```

3.30 KD 求最近点对距离

```

1 // 二维平面里最近点对距离
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 typedef long long ll;
6 const int N = 1e5 + 10;
7 struct node {
8     ll x[2];
9 }a[N], b[N];
10
11 int now, n;
12 ll ans;
13 map<pair<ll, ll>, int> mp;
14
15 bool cmp(node a, node b) { return a.x[now] < b.x[now]; }
16
17 ll sqr(int x) { return (ll)x * x; }
18 ll dis(node a, node b) { return sqr(a.x[0] - b.x[0]) + sqr(a.x[1] - b.x[1]); }
19
20 void build(int l, int r, int d) {
21     if(l >= r) return ;
22     int m = (l + r) >> 1;
23     now = d;
24     nth_element(a + l, a + m, a + r, cmp);
25     build(l, m, d ^ 1);
26     build(m + 1, r, d ^ 1);
27 }
28
29 void query(int l, int r, int d, node p) {
30     if(l >= r) return ;
31     int m = (l + r) >> 1;
32     int ql = l, qr = m;
33     ll res = dis(a[m], p);
34     if(ans == 0 || res && ans > res) ans = res;
35     if(p.x[d] > a[m].x[d]) ql = m + 1, qr = r;
36     query(ql, qr, d ^ 1, p);
37     if(ans > sqr(a[m].x[d] - p.x[d]))
38         query(l + m - ql + 1, m + r - qr, d ^ 1, p);
39 }

```

```

40
41 void solve() {
42     scanf("%d",&n);
43     ll sum = 5e18;
44     for(int i = 0; i < n; i++) {
45         scanf("%lld %lld",&a[i].x[0],&a[i].x[1]);
46         if(mp[{a[i].x[0], a[i].x[1]}]) sum = 0;
47         else mp[{a[i].x[0], a[i].x[1]}]++;
48         b[i] = a[i];
49     }
50     build(0, n, 0);
51     for(int i = 0; i < n; i++) {
52         ans = 0;
53         query(0, n, 0, b[i]);
54         sum = min(ans, sum);
55     }
56     printf("%.4lf\n",sqrt(1.0 * sum));
57 }

```

3.31 CDQ 分治

```

1 // 原问题：“任意两个元素之间的贡献”
2
3 // 降维成的子问题：“左段元素对右段每一个元素的贡献”
4
5 // 分而治之，最基础的应用为归并排序
6 // 将一个区间分成两个区间[l,mid]和[mid+1,r]，然后排序使其有序
7 // 注意事项：
8 // 1、离散化
9 // 2、如果有多个点的参量完全相同，由于顺着添加，查出来的最后一个答案才是正确的
10 // 对于y排序，我们可以直接对两段区间Sort，然而归并排序本身就是分治，我们可以在CDQ的过程中进行归并排
    序，要比Sort少一个log
11
12 // 千万不要忘记离散化!!!!!!!!!!!!!!!!!!!!!!

```

3.32 cdq 处理逆序数

```

1 const int N = 1e5 + 10;
2 int ans[N], cnt[N];
3
4 struct star {
5     int x, y, id;
6 }a[N], tmp[N];
7
8 bool cmp(star a, star b) {
9     if(a.y == b.y) return a.x < b.x;
10    return a.y < b.y;
11 }
12
13 void cdq(int l, int r) {
14     if(l == r) return ;
15     int m = (l + r) / 2;
16     cdq(l, m);
17     cdq(m + 1, r);
18     int p = l, q = m + 1;
19     for(int i = l; i <= r; i++) {
20         if((p <= m && a[p].x <= a[q].x) || q > r) {

```



```

21         tmp[i] = a[p++];
22     }
23     else {
24         ans[a[q].id] += p - l;
25         tmp[i] = a[q++];
26     }
27 }
28 for(int i = l; i <= r; i++) a[i] = tmp[i];
29 }
30
31 void solve() {
32     int n; cin >> n;
33     for(int i = 1; i <= n; i++) cin >> a[i].x >> a[i].y, a[i].id = i;
34     sort(a + 1, a + n + 1, cmp);
35     cdq(1, n);
36     for(int i = 1; i <= n; i++) cnt[ans[i]]++;
37     for(int i = 0; i < n; i++) cout << cnt[i] << endl;
38 }

```

3.33 cdq 处理二维偏序

```

1
2 const int N = 1e5 + 10;
3 int ans[N], cnt[N];
4
5 struct star {
6     int x, y, id;
7 } a[N], tmp[N];
8
9 bool cmp(star a, star b) {
10     if(a.x == b.x) return a.y < b.y;
11     return a.x < b.x;
12 }
13
14 void cdq(int l, int r) {
15     if(l == r) return;
16     int m = (l + r) / 2;
17     cdq(l, m);
18     cdq(m + 1, r);
19     int p = l, q = m + 1;
20     for(int i = l; i <= r; i++) {
21         if((p <= m && a[p].x <= a[q].x) || q > r) {
22             tmp[i] = a[p++];
23         }
24         else {
25             ans[a[q].id] += i - l;
26             tmp[i] = a[q++];
27         }
28     }
29     for(int i = l; i <= r; i++) a[i] = tmp[i];
30 }
31
32 void solve() {
33     int n; cin >> n;
34     for(int i = 1; i <= n; i++) cin >> a[i].x >> a[i].y;
35     sort(a + 1, a + n + 1, cmp);
36     cdq(1, n);
37     for(int i = 1; i <= n; i++) cnt[ans[i]]++;

```

```

38     for(int i = 0; i < n; i++) cout << cnt[i] << endl;
39 }

```

3.34 cdq 套 cdq 处理三维偏序

```

1  // 一维排序、二维cdq、三维树状数组
2
3  const int N = 1e5 + 10;
4  struct node {
5      int x, y, z;
6      int id;
7      int tag;
8      bool operator < (const node &a) const {
9          if(x != a.x) return x < a.x;
10         if(y != a.y) return y < a.y;
11         return z < a.z;
12     }
13     bool operator == (const node &a) const {
14         return x == a.x && y == a.y && z == a.z;
15     }
16 } a[N], b[N], tmp[N];
17
18 int ans[N];
19 int n;
20
21 void cdq2(int l, int r) {
22     if(l == r) return;
23     int mid = (l + r) / 2;
24     cdq2(l, mid); cdq2(mid + 1, r);
25     int p = l, q = mid + 1, cnt = 0;
26     for(int i = l; i <= r; i++) {
27         if(q > r || (p <= mid && b[p].z <= b[q].z)) {
28             if(b[p].tag == 0) cnt++;
29             tmp[i] = b[p++];
30         }
31         else {
32             if(b[q].tag == 1) ans[b[q].id] += cnt;
33             tmp[i] = b[q++];
34         }
35     }
36     for(int i = l; i <= r; i++) b[i] = tmp[i];
37 }
38
39 void cdq1(int l, int r) {
40     if(l == r) return;
41     int mid = (l + r) / 2;
42     cdq1(l, mid); cdq1(mid + 1, r);
43     int p = l, q = mid + 1;
44     /* 因为是计算左端元素对右端元素的影响，所以需要打个标记tag来记录他是左端还是右端元素 */
45     for(int i = l; i <= r; i++) {
46         if(q > r || (p <= mid && a[p].y <= a[q].y)) {
47             a[p].tag = 0;
48             b[i] = a[p++];
49         }
50         else {
51             a[q].tag = 1;
52             b[i] = a[q++];
53         }
54     }
55 }

```

```

54     }
55     for(int i = l; i <= r; i++) a[i] = b[i];
56     cdq2(l, r);
57 }
58
59 void solve() {
60     n = read();
61     for(int i = 1; i <= n; i++) {
62         a[i].x = read(), a[i].y = read(), a[i].z = read();
63         a[i].id = i;
64     }
65     sort(a + 1, a + n + 1);
66     for(int i = n - 1; i >= 1; i--) {
67         if(a[i + 1] == a[i]) ans[a[i].id] = ans[a[i + 1].id] + 1;
68     }
69     cdq1(1, n);
70     for(int i = 1; i <= n; i++) cout << ans[i] << endl;
71 }

```

3.35 cdq 套树状数组处理三维偏序

```

1  // 一维排序、二维cdq、三维树状数组
2
3  const int N = 1e5 + 10;
4  struct node {
5      int x, y, z;
6      int id;
7      bool operator < (const node &a) const {
8          if(x != a.x) return x < a.x;
9          if(y != a.y) return y < a.y;
10         return z < a.z;
11     }
12     bool operator == (const node &a) const {
13         return x == a.x && y == a.y && z == a.z;
14     }
15 } a[N], b[N];
16 int n;
17
18
19 int ans[N];
20
21 struct BIT {
22     #define lowbit(x) (x & (-x))
23     int n;
24     int t[N];
25
26     void init(int _n) {
27         mem(t, 0);
28         n = _n;
29     }
30
31     void update(int x, int val) {
32         while (x <= n) {
33             t[x] += val;
34             x += lowbit(x);
35         }
36     }
37

```

```

38     int query(int x) {
39         int ans = 0;
40         while (x) {
41             ans += t[x];
42             x -= lowbit(x);
43         }
44         return ans;
45     }
46 }bit;
47
48 void cdq(int l, int r) {
49     if(l == r) return ;
50     int mid = (l + r) / 2;
51     cdq(l, mid);
52     cdq(mid + 1, r);
53     int p = l, q = mid + 1;
54     for(int i = l; i <= r; i++) {
55         if(q > r || (p <= mid && a[p].y <= a[q].y)) {
56             bit.update(a[p].z, 1);
57             b[i] = a[p++];
58         }
59         else {
60             ans[a[q].id] += bit.query(a[q].z);
61             b[i] = a[q++];
62         }
63     }
64     for(int i = l; i <= mid; i++) bit.update(a[i].z, -1);
65     for(int i = l; i <= r; i++) a[i] = b[i];
66 }
67
68 void solve() {
69     n = read();
70     int mx = 0;
71     for(int i = 1; i <= n; i++) {
72         a[i].x = read(), a[i].y = read(), a[i].z = read();
73         a[i].id = i;
74         mx = max(mx, a[i].z);
75     }
76     bit.init(mx);
77     sort(a + 1, a + n + 1);
78     for(int i = n - 1; i >= 1; i--) {
79         if(a[i] == a[i + 1]) ans[a[i].id] = ans[a[i + 1].id] + 1;
80     }
81     cdq(1, n);
82     for(int i = 1; i <= n; i++) cout << ans[i] << endl;
83 }

```

3.36 cdq 维护矩阵内二维数点

```

1 // 求二维平面上(x1,y1)到(x2,y2)的矩阵中数点
2
3 // 利用前缀和思想, 把问题划分成[x2,y2] - [x1-1,y2] - [x2,y2-1] + [x1-1,y1-1]
4
5 // 所有要建立4个虚点为查询点, 而原本实点为修改点
6
7 const int N = 3e6 + 10;
8
9 struct node {

```

```

10     int x, y, opt, id;
11     // opt为操作类型, 1为修改, 0为查询
12     bool operator < (const node& o) const {
13         return x == o.x ? (y == o.y ? opt : y < o.y) : x < o.x;
14     }
15     // 注意排序顺序, 坐标相同时, 要使opt放前面, 因为要先修改
16
17     bool operator == (const node &o) const {
18         return x == o.x && y == o.y;
19     }
20 }a[N], tmp[N];
21
22 int ans[N];
23
24 void cdq(int l, int r) {
25     if(l == r) return ;
26     int mid = (l + r) / 2;
27     cdq(l, mid); cdq(mid + 1, r);
28     int p = l, q = mid + 1, cnt = 0;
29     for(int i = l; i <= r; i++) {
30         if(q > r || (p <= mid && a[p].y <= a[q].y)) {
31             cnt += a[p].opt;
32             tmp[i] = a[p++];
33         }
34         else {
35             ans[a[q].id] += cnt;
36             tmp[i] = a[q++];
37         }
38     }
39     for(int i = l; i <= r; i++) a[i] = tmp[i];
40 }
41
42 void solve() {
43     int n = read(), m = read();
44     for(int i = 1; i <= n; i++) {
45         a[i].x = read(), a[i].y = read(), a[i].opt = 1;
46     }
47     int _n = 0;
48     for(int i = 1; i <= m; i++) {
49         int x1 = read(), y1 = read(), x2 = read(), y2 = read();
50         a[++n] = (node){x2, y2, 0, ++_n};
51         a[++n] = (node){x2, y1 - 1, 0, ++_n};
52         a[++n] = (node){x1 - 1, y2, 0, ++_n};
53         a[++n] = (node){x1 - 1, y1 - 1, 0, ++_n};
54     }
55     sort(a + 1, a + n + 1);
56     cdq(1, n);
57     for(int i = 1; i + 3 <= _n; i += 4) {
58         cout << ans[i] - ans[i + 1] - ans[i + 2] + ans[i + 3] << endl;
59     }
60 }

```

3.37 advanced01trie

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 2e7 + 10;

```

```

5
6 int t[N][2];
7 int cnt, root[N], sz[N][2];
8 int a[N];
9
10 void insert(int pre, int &now, int i, int x) {
11     if(i < 0) return ;
12     now = ++cnt;
13     int d = x >> i & 1;
14     t[now][d ^ 1] = t[pre][d ^ 1];
15     sz[now][d ^ 1] = sz[pre][d ^ 1]; sz[now][d] = sz[pre][d] + 1;
16     insert(t[pre][d], t[now][d], i - 1, x);
17 }
18
19 int query(int l, int r, int i, int x) {
20     if(i < 0) return 0;
21     int d = x >> i & 1;
22     int tmp = sz[r][d ^ 1] - sz[l][d ^ 1];
23     if(tmp > 0) return query(t[l][d ^ 1], t[r][d ^ 1], i - 1, x) + (1 << i);
24     else return query(t[l][d], t[r][d], i - 1, x);
25 }
26
27 int main() {
28     int n, m;
29     cin >> n >> m;
30     for(int i = 1; i <= n; i++) {
31         int x;
32         cin >> x;
33         insert(root[i - 1], root[i], 30, x);
34     }
35     while(m--) {
36         int l, r, x;
37         cin >> l >> r >> x;
38         cout << query(root[l - 1], root[r], 30, x) << endl;
39     }
40 }

```

3.38 advancedArray

```

1
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int N = 1e5 + 10;
6
7 struct Node {
8     int l, r, val;
9 }hjt[N * 40];
10 int cnt, root[N];
11 int a[N];
12
13 inline void build(int &now, int l, int r) {
14     now = ++cnt;
15     if(l == r) {
16         hjt[now].val = a[l];
17         return ;
18     }
19     int m = (l + r) >> 1;

```

```

20     build(hjt[now].l, l, m);
21     build(hjt[now].r, m + 1, r);
22 }
23
24 inline void modify(int ver, int &now, int l, int r, int pos, int value) {
25     hjt[now = ++cnt] = hjt[ver];
26     if(l == r) {
27         hjt[now].val = value;
28         return ;
29     }
30     int m = (l + r) >> 1;
31     if(pos <= m) modify(hjt[ver].l, hjt[now].l, l, m, pos, value);
32     else modify(hjt[ver].r, hjt[now].r, m + 1, r, pos, value);
33 }
34
35 inline int query(int now, int l, int r, int pos) {
36     if(l == r) return hjt[now].val;
37     int m = (l + r) >> 1;
38     if(pos <= m) return query(hjt[now].l, l, m, pos);
39     else return query(hjt[now].r, m + 1, r, pos);
40 }
41
42 int main() {
43     int n, m;
44     cin >> n >> m;
45     for(int i = 1; i <= n; i++) cin >> a[i];
46     build(root[0], 1, n);
47     for(int i = 1; i <= m; i++) {
48         int ver, opt;
49         cin >> ver >> opt;
50         if(opt == 1) {
51             int pos, value;
52             cin >> pos >> value;
53             modify(root[ver], root[i], 1, n, pos, value);
54         }
55         else {
56             int pos;
57             cin >> pos;
58             root[i] = root[ver];
59             cout << query(root[i], 1, n, pos) << endl;
60         }
61     }
62 }

```

3.39 advancedDSU

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e5 + 10;
5
6  struct Node {
7      int l, r, val;
8  }hjt[N * 40 * 2];
9
10 int cnt, rootfa[N], rootdep[N], tot;
11 int n;
12

```

```

13 inline void build(int &now, int l, int r) {
14     now = ++cnt;
15     if(l == r) {
16         hjt[now].val = ++tot;
17         return ;
18     }
19     int m = (l + r) >> 1;
20     build(hjt[now].l, l, m);
21     build(hjt[now].r, m + 1, r);
22 }
23
24 inline void modify(int ver, int &now, int l, int r, int pos, int value) {
25     hjt[now = ++cnt] = hjt[ver];
26     if(l == r) {
27         hjt[now].val = value;
28         return ;
29     }
30     int m = (l + r) >> 1;
31     if(pos <= m) modify(hjt[ver].l, hjt[now].l, l, m, pos, value);
32     else modify(hjt[ver].r, hjt[now].r, m + 1, r, pos, value);
33 }
34
35 inline int query(int now, int l, int r, int pos) {
36     if(l == r) return hjt[now].val;
37     int m = (l + r) >> 1;
38     if(pos <= m) return query(hjt[now].l, l, m, pos);
39     else return query(hjt[now].r, m + 1, r, pos);
40 }
41
42 inline int find(int ver, int x) {
43     int fx = query(rootfa[ver], 1, n, x);
44     return fx == x ? x : find(ver, fx);
45 }
46
47 inline void merge(int ver, int x, int y) {
48     x = find(ver - 1, x);
49     y = find(ver - 1, y);
50     if(x == y) {
51         rootfa[ver] = rootfa[ver - 1];
52         rootdep[ver] = rootdep[ver - 1];
53     }
54     else {
55         int depx = query(rootdep[ver - 1], 1, n, x);
56         int depy = query(rootdep[ver - 1], 1, n, y);
57         if(depx < depy) {
58             modify(rootfa[ver - 1], rootfa[ver], 1, n, x, y);
59             rootdep[ver] = rootdep[ver - 1];
60         }
61         else if(depx > depy) {
62             modify(rootfa[ver - 1], rootfa[ver], 1, n, y, x);
63             rootdep[ver] = rootdep[ver - 1];
64         }
65         else {
66             modify(rootfa[ver - 1], rootfa[ver], 1, n, x, y);
67             modify(rootdep[ver - 1], rootdep[ver], 1, n, y, depy + 1);
68         }
69     }
70 }
71

```



```

72 int main() {
73     int m;
74     cin >> n >> m;
75     build(rootfa[0], 1, n);
76     for(int ver = 1; ver <= m; ver++) {
77         int opt, x, y;
78         cin >> opt;
79         if(opt == 1) {
80             cin >> x >> y;
81             merge(ver, x, y);
82         }
83         else if(opt == 2) {
84             cin >> x;
85             rootfa[ver] = rootfa[x];
86             rootdep[ver] = rootdep[x];
87         }
88         else {
89             cin >> x >> y;
90             rootfa[ver] = rootfa[ver - 1];
91             rootdep[ver] = rootdep[ver - 1];
92             int u = find(ver, x);
93             int v = find(ver, y);
94             if(u == v) cout << 1 << endl;
95             else cout << 0 << endl;
96         }
97     }
98 }

```

3.40 扫描线求面积并

```

1  // 横向扫描
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  const int N = 2e5 + 10;
6
7  #define lc u << 1
8  #define rc u << 1 | 1
9  #define mid (t[u].l + t[u].r) >> 1
10
11 int n, cnt;
12
13 double v[N];
14 struct L {
15     double x, y1, y2;
16     int state;
17     bool operator < (L rhs) {return x < rhs.x; }
18 }line[N << 2];
19
20 struct Node {
21     int l, r, cover;
22     double len;
23 }t[N << 2];
24
25 inline void push_up(int u) {
26     if(t[u].cover) t[u].len = v[t[u].r + 1] - v[t[u].l];
27     else if(t[u].l == t[u].r) t[u].len = 0;
28     else t[u].len = t[lc].len + t[rc].len;

```

```

29 }
30
31 void build(int u, int l, int r) {
32     t[u].l = l; t[u].r = r;
33     if(l == r) {
34         t[l].cover = t[l].len = 0;
35         return ;
36     }
37     int m = (l + r) >> 1;
38     build(lc, l, m);
39     build(rc, m + 1, r);
40 }
41
42 void modify(int u, int ql, int qr, int state) {
43     if(ql <= t[u].l && t[u].r <= qr) {
44         t[u].cover += state;
45         push_up(u);
46         return ;
47     }
48     if(ql <= mid) modify(lc, ql, qr, state);
49     if(qr > mid) modify(rc, ql, qr, state);
50     push_up(u);
51 }
52
53 void init() {
54     cin >> n;
55     for(int i = 1; i <= n; i++) {
56         double x1, y1, x2, y2;
57         scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
58         line[i] = L{x1, y1, y2, 1}; v[i] = y1;
59         line[n + i] = L{x2, y1, y2, -1}; v[n + i] = y2;
60     }
61     n <= 1;
62     sort(line + 1, line + n + 1);
63     sort(v + 1, v + n + 1);
64     cnt = unique(v + 1, v + n + 1) - (v + 1);
65     build(1, 1, cnt);
66 }
67
68 void solve() {
69     double ans = 0;
70     for(int i = 1; i <= n; i++) {
71         int ql = lower_bound(v + 1, v + cnt + 1, line[i].y1) - v;
72         int qr = lower_bound(v + 1, v + cnt + 1, line[i].y2) - v - 1;
73         modify(1, ql, qr, line[i].state);
74         ans += t[1].len * (line[i + 1].x - line[i].x);
75     }
76     cout << ans << endl;
77 }
78
79 int main() {
80     init();
81     solve();
82 }

```

3.41 扫描线求周长并

```
1 // 纵向扫描
```

```

2  #include <bits/stdc++.h>
3  using namespace std;
4
5  const int N = 2e5 + 10;
6
7  #define INF 0x3ffffff
8  #define lc u << 1
9  #define rc u << 1 | 1
10 #define mid (t[u].l + t[u].r) >> 1
11
12 int n;
13
14 struct L {
15     int y, x1, x2;
16     int state;
17     bool operator < (L rhs) {return y < rhs.y; }
18 }line[N << 2];
19
20 struct Node {
21     int l, r, cover;
22     bool ls, rs;
23     int num;
24     int len;
25 }t[N << 2];
26
27 inline void push_up(int u) {
28     if(t[u].cover) {
29         t[u].len = t[u].r - t[u].l + 1;
30         t[u].ls = t[u].rs = 1;
31         t[u].num = 1;
32     }
33     else if(t[u].l == t[u].r) {
34         t[u].ls = t[u].rs = 0;
35         t[u].len = t[u].num = 0;
36     }
37     else {
38         t[u].len = t[lc].len + t[rc].len;
39         t[u].ls = t[lc].ls; t[u].rs = t[rc].rs;
40         t[u].num = t[lc].num + t[rc].num - (t[lc].rs & t[rc].ls);
41     }
42 }
43
44 void build(int u, int l, int r) {
45     t[u].l = l; t[u].r = r;
46     if(l == r) {
47         t[u].len = t[u].cover = t[u].ls = t[u].rs = t[u].num = 0;
48         return ;
49     }
50     int m = (l + r) >> 1;
51     build(lc, l, m);
52     build(rc, m + 1, r);
53 }
54
55 void modify(int u, int ql, int qr, int state) {
56     if(ql <= t[u].l && t[u].r <= qr) {
57         t[u].cover += state;
58         push_up(u);
59         return ;
60     }

```

```

61     if(ql <= mid) modify(lc, ql, qr, state);
62     if(qr > mid)  modify(rc, ql, qr, state);
63     push_up(u);
64 }
65
66 void init() {
67     cin >> n;
68     int mx = -INF, mn = INF;
69     for (int i = 1; i <= n; i++) {
70         int x1, x2, y1, y2;
71         cin >> x1 >> y1 >> x2 >> y2;
72         mx = max(mx, max(x1, x2));
73         mn = min(mn, min(x1, x2));
74         line[i] = L{y1, x1, x2, 1};
75         line[n + i] = L{y2, x1, x2, -1};
76     }
77     n <= 1;
78     sort(line + 1, line + n + 1);
79     build(1, mn, mx);
80 }
81
82 void solve() {
83     int ans = 0;
84     int last = 0;
85     for(int i = 1; i <= n; i++) {
86         modify(1, line[i].x1, line[i].x2 - 1, line[i].state);
87         ans += abs(t[1].len - last); // 横线
88         ans += (line[i + 1].y - line[i].y) * 2 * t[1].num; // 竖线
89         last = t[1].len;
90     }
91     printf("%d\n", ans);
92 }
93
94 int main() {
95     init();
96     solve();
97 }

```

3.42 区间第 k 小

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e5 + 10;
5
6  vector<int> v;
7  struct Node {
8      int l, r;
9      int val;
10 }hjt[N * 40];
11 int root[N], cnt;
12
13 int get_id(int x) { return lower_bound(v.begin(), v.end(), x) - v.begin() + 1; }
14
15 void insert(int pre, int &now, int l, int r, int p) {
16     hjt[++cnt] = hjt[pre];
17     now = cnt;
18     hjt[now].val++;

```

```

19     if(l == r) return ;
20     int m = (l + r) >> 1;
21     if(p <= m) insert(hjt[pre].l, hjt[now].l, l, m, p);
22     else insert(hjt[pre].r, hjt[now].r, m + 1, r, p);
23 }
24
25 int query(int L, int R, int l, int r, int k) {
26     if(l == r) return l;
27     int m = (l + r) >> 1;
28     int tmp = hjt[hjt[R].l].val - hjt[hjt[L].l].val;
29     if(k <= tmp) return query(hjt[L].l, hjt[R].l, l, m, k);
30     else return query(hjt[L].r, hjt[R].r, m + 1, r, k - tmp);
31 }
32
33 int main() {
34     int n, q;
35     cin >> n >> q;
36     vector<int> a(n + 1);
37     for(int i = 1; i <= n; i++) { cin >> a[i]; v.push_back(a[i]); }
38     sort(v.begin(), v.end());
39     v.erase(unique(v.begin(), v.end()), v.end());
40
41     for(int i = 1; i <= n; i++) {
42         insert(root[i - 1], root[i], 1, n, get_id(a[i]));
43     }
44
45     while(q--) {
46         int l, r, k;
47         cin >> l >> r >> k;
48         cout << v[query(root[l - 1], root[r], 1, n, k) - 1] << endl;
49     }
50 }

```

3.43 区间前 k 大

```

1
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 typedef long long ll;
6 const int N = 1e6 + 10;
7 int a[N];
8 vector<int> v;
9 int cnt, root[N];
10
11 struct Node {
12     int l, r;
13     ll sum;
14     int num;
15     int val;
16 }hjt[N * 40];
17
18 int getid(int x) { return lower_bound(v.begin(), v.end(), x) - v.begin() + 1; }
19
20 void insert(int pre, int &now, int l, int r, int p, int val) {
21     now = ++cnt;
22     hjt[now] = hjt[pre];
23     hjt[now].num++; hjt[now].sum += val;

```

```

24     if(l == r) {
25         hjt[now].val = val;
26         return ;
27     }
28     int m = (l + r) >> 1;
29     if(p <= m) insert(hjt[pre].l, hjt[now].l, l, m, p, val);
30     else insert(hjt[pre].r, hjt[now].r, m + 1, r, p, val);
31 }
32
33 ll query(int L, int R, int l, int r, int k) {
34     if(l == r) return hjt[R].val * k;
35     int m = (l + r) >> 1;
36     int tmp = hjt[hjt[R].r].num - hjt[hjt[L].r].num;
37     if(k <= tmp) return query(hjt[L].r, hjt[R].r, m + 1, r, k);
38     else return hjt[hjt[R].r].sum - hjt[hjt[L].r].sum + query(hjt[L].l, hjt[R].l, l, m,
39         k - tmp);
40 }
41 void init(int n) {
42     v.clear();
43     cnt = 0;
44     for(int i = 1; i <= n; i++) {
45         scanf("%d", &a[i]);
46         v.push_back(a[i]); root[i] = 0;
47     }
48     sort(v.begin(), v.end());
49     v.erase(unique(v.begin(), v.end()), v.end());
50 }
51
52 int main() {
53     int _;
54     scanf("%d", &_);
55     while(_--) {
56         int n;
57         scanf("%d", &n);
58         init(n);
59         for(int i = 1; i <= n; i++) {
60             insert(root[i - 1], root[i], 1, n, getid(a[i]), a[i]);
61         }
62         int q;
63         scanf("%d", &q);
64         while(q--) {
65             int l, r, k;
66             scanf("%d%d%d", &l, &r, &k);
67             int t = r - l + 1;
68             ll ans = query(root[l - 1], root[r], 1, n, k);
69             printf("%lld\n", 1ll * t * (t + 1) * (2 * t + 1) / 6 + ans);
70         }
71     }
72 }

```

3.44 树套树维护三维偏序

```

1  const int N = 4e6 + 10;
2
3  int n, k;
4  struct node {
5      int a, b, c;

```

```

6     int operator < (const node &o) const {
7         return a != o.a ? (a < o.a) : (b != o.b ? (b < o.b) : (c < o.c));
8     }
9
10    int operator == (const node &o) const {
11        return a == o.a && b == o.b && c == o.c;
12    }
13 }p[N];
14
15
16 struct Tree1 {
17     int l, r;
18 }t1[N << 2];
19
20 struct Tree2 {
21     int l, r;
22     int num;
23 }t2[N << 2];
24
25 int root, root2[N];
26 int cnt1, cnt2;
27
28 void vec_insert(int &u, int l, int r, int pos, int val) {
29     if(!u) u = ++cnt2;
30     t2[u].num += val;
31     if(l == r) return ;
32     int m = (l + r) / 2;
33     if(pos <= m) vec_insert(t2[u].l, l, m, pos, val);
34     else vec_insert(t2[u].r, m + 1, r, pos, val);
35 }
36
37 int vec_query(int u, int l, int r, int ql, int qr) {
38     if(!u) return 0;
39     if(ql <= l && r <= qr) return t2[u].num;
40     int ans = 0;
41     int mid = (l + r) / 2;
42     if(ql <= mid) ans += vec_query(t2[u].l, l, mid, ql, qr);
43     if(qr > mid) ans += vec_query(t2[u].r, mid + 1, r, ql, qr);
44     return ans;
45 }
46
47 // 在第一维权值线段树在[1,k]根据p[x].b插入, 第二维权值线段树在[1,k]根据p[x].c插入
48 void tree_insert(int &u, int l, int r, int x, int val) {
49     if(!u) u = ++cnt1;
50     vec_insert(root2[u], 1, k, p[x].c, val);
51     if(l == r) return ;
52     int m = (l + r) / 2;
53     if(p[x].b <= m) tree_insert(t1[u].l, l, m, x, val);
54     else tree_insert(t1[u].r, m + 1, r, x, val);
55 }
56
57 int tree_query(int u, int l, int r, int x) {
58     if(!u) return 0;
59     if(1 <= l && r <= p[x].b) return vec_query(root2[u], 1, k, 1, p[x].c);
60     int mid = (l + r) / 2;
61     int ans = 0;
62     if(1 <= mid) ans += tree_query(t1[u].l, l, mid, x);
63     if(p[x].b > mid) ans += tree_query(t1[u].r, mid + 1, r, x);
64     return ans;

```

```

65 }
66
67 void solve() {
68     cin >> n >> k;
69     for(int i = 1; i <= n; i++) cin >> p[i].a >> p[i].b >> p[i].c;
70     sort(p + 1, p + n + 1);
71     vector<int> ans(n + 1);
72     int sum = 1;
73     for(int i = 1; i <= n; i++) {
74         // 因为这些个都一样, 如果不这样操作, 会使后面的不会对前面的有贡献
75         if(p[i + 1] == p[i]) {
76             sum++;
77             continue;
78         }
79         tree_insert(root, 1, k, i, sum);
80         int res = tree_query(root, 1, k, i);
81         ans[res] += sum;
82         sum = 1;
83     }
84     for(int i = 1; i <= n; i++) cout << ans[i] << endl;
85 }

```

3.45 线段树套主席树-二维区间不同数

```

1  const int N = 2e5 + 10;
2
3  int n, m, l, r, a, b, num[N], Last[N], pre[N], Hash[N], ans[N][2];
4  int cnt, root[N], sum[N * 20], lc[N * 20], rc[N * 20];
5  struct Query {
6      int a, b, l, id;
7  };
8
9  struct data {
10     int a, v;
11
12     bool operator<(const data &b) const {
13         return a < b.a;
14     }
15 } d[N];
16
17 vector<Query> q[N * 4];
18
19 void addquery(int o, int l, int r, int L, int R, Query qry) {
20     if (L <= l && R >= r) {
21         q[o].push_back(qry);
22         return;
23     }
24     int mid = (l + r) / 2;
25     if (L <= mid) {
26         addquery(o * 2, l, mid, L, R, qry);
27     }
28     if (R > mid) {
29         addquery(o * 2 + 1, mid + 1, r, L, R, qry);
30     }
31 }
32
33 void build(int y, int &x, int l, int r, int k) {
34     x = ++cnt;

```



```

35     sum[x] = sum[y] + 1;
36     lc[x] = lc[y];
37     rc[x] = rc[y];
38     if (l == r) {
39         return;
40     }
41     int mid = (l + r) / 2;
42     if (k <= mid) {
43         build(lc[y], lc[x], l, mid, k);
44     } else {
45         build(rc[y], rc[x], mid + 1, r, k);
46     }
47 }
48
49 int query(int y, int x, int l, int r, int k) {
50     if (!x || l == r) {
51         return 0;
52     }
53     int mid = (l + r) / 2;
54     if (k <= mid) {
55         return query(lc[y], lc[x], l, mid, k);
56     } else {
57         return sum[lc[x]] - sum[lc[y]] + query(rc[y], rc[x], mid + 1, r, k);
58     }
59 }
60
61 void insert(int o, int l, int r) {
62     if (q[o].size()) {
63         Hash[0] = 0;
64         for (int i = l; i <= r; i++) {
65             Hash[++Hash[0]] = num[i];
66         }
67         sort(Hash + 1, Hash + Hash[0] + 1);
68         int s = 0;
69         for (int i = l; i <= r; i++) {
70             d[++s].a = lower_bound(Hash + 1, Hash + Hash[0] + 1, num[i]) - Hash;
71             d[s].v = pre[i];
72         }
73         sort(d + 1, d + s + 1);
74         for (int i = 1; i <= s; i++) {
75             build(root[i - 1], root[i], 0, n, d[i].v);
76         }
77         int a, b;
78         for (int i = 0; i < q[o].size(); i++) {
79             a = lower_bound(Hash + 1, Hash + Hash[0] + 1, q[o][i].a) - Hash;
80             b = upper_bound(Hash + 1, Hash + Hash[0] + 1, q[o][i].b) - Hash - 1;
81             ans[q[o][i].id][0] += sum[root[b]] - sum[root[a - 1]];
82             ans[q[o][i].id][1] += query(root[a - 1], root[b], 0, n, q[o][i].l);
83         }
84         memset(root, 0, sizeof(int) * (Hash[0] + 1));
85         memset(sum, 0, sizeof(int) * (cnt + 1));
86         memset(lc, 0, sizeof(int) * (cnt + 1));
87         memset(rc, 0, sizeof(int) * (cnt + 1));
88         cnt = 0;
89     }
90     if (l == r) {
91         return;
92     }
93     int mid = (l + r) / 2;

```

```

94     insert(o * 2, l, mid);
95     insert(o * 2 + 1, mid + 1, r);
96 }
97
98 void solve() {
99     int _;
100    cin >> _;
101    while (_--) {
102        for (int i = 0; i < N * 4; i++) q[i].clear();
103        cnt = 0;
104        for (int i = 0; i < N; i++) ans[i][0] = ans[i][1] = pre[i] = Last[i] = 0;
105        cin >> n >> m;
106        for (int i = 1; i <= n; i++) {
107            cin >> num[i];
108            num[i]++;
109            pre[i] = Last[num[i]];
110            Last[num[i]] = i;
111        }
112        for (int i = 1; i <= m; i++) {
113            cin >> l >> a >> r >> b;
114            a++, b++;
115            addquery(1, 1, 1e5 + 1, l, r, (Query) {a, b, l, i});
116        }
117        insert(1, 1, 1e5 + 1);
118        for (int i = 1; i <= m; i++) {
119            cout << ans[i][1] << endl;
120        }
121    }
122 }

```

3.46 Scapegoat

```

1  // 无旋转平衡，暴力拍扁重构
2
3  #include <bits/stdc++.h>
4  using namespace std;
5
6  namespace Scapegoat_Tree {
7  #define MAXN (100000 + 10)
8      const double alpha = 0.75;
9      struct Node {
10         Node * ch[2];
11         int key, size, cover; // size为有效节点的数量，cover为节点总数量
12         bool exist; // 是否存在 (即是否被删除)
13         void PushUp(void) {
14             size = ch[0]->size + ch[1]->size + (int)exist;
15             cover = ch[0]->cover + ch[1]->cover + 1;
16         }
17         bool isBad(void) { // 判断是否需要重构
18             return ((ch[0]->cover > cover * alpha + 5) ||
19                     (ch[1]->cover > cover * alpha + 5));
20         }
21     };
22     struct STree {
23     protected:
24         Node mem_poor[MAXN]; // 内存池，直接分配好避免动态分配内存占用时间
25         Node *tail, *root, *null; // 用null表示NULL的指针更方便，tail为内存分配指针，root为根
26         Node *bc[MAXN]; int bc_top; // 储存被删除的节点的内存地址，分配时可以再利用这些地址

```

```

27
28 Node * NewNode(int key) {
29     Node * p = bc_top ? bc[--bc_top] : tail++;
30     p->ch[0] = p->ch[1] = null;
31     p->size = p->cover = 1; p->exist = true;
32     p->key = key;
33     return p;
34 }
35 void Travel(Node * p, vector<Node *>&v) {
36     if (p == null) return;
37     Travel(p->ch[0], v);
38     if (p->exist) v.push_back(p); // 构建序列
39     else bc[bc_top++] = p; // 回收
40     Travel(p->ch[1], v);
41 }
42 Node * Divide(vector<Node *>&v, int l, int r) {
43     if (l >= r) return null;
44     int mid = (l + r) >> 1;
45     Node * p = v[mid];
46     p->ch[0] = Divide(v, l, mid);
47     p->ch[1] = Divide(v, mid + 1, r);
48     p->PushUp(); // 自底向上维护, 先维护子树
49     return p;
50 }
51 void Rebuild(Node * &p) {
52     static vector<Node *>v; v.clear();
53     Travel(p, v); p = Divide(v, 0, v.size());
54 }
55 Node ** Insert(Node *&p, int val) {
56     if (p == null) {
57         p = NewNode(val);
58         return &null;
59     }
60     else {
61         p->size++; p->cover++;
62
63         // 返回值储存需要重构的位置, 若子树也需要重构, 本节点开始也需要重构, 以本节点为根重构
64         Node ** res = Insert(p->ch[val >= p->key], val);
65         if (p->isBad()) res = &p;
66         return res;
67     }
68 }
69 void Erase(Node *p, int id) {
70     p->size--;
71     int offset = p->ch[0]->size + p->exist;
72     if (p->exist && id == offset) {
73         p->exist = false;
74         return;
75     }
76     else {
77         if (id <= offset) Erase(p->ch[0], id);
78         else Erase(p->ch[1], id - offset);
79     }
80 }
81 public:
82     void Init(void) {
83         tail = mem_poor;
84         null = tail++;
85         null->ch[0] = null->ch[1] = null;

```

```

86         null->cover = null->size = null->key = 0;
87         root = null; bc_top = 0;
88     }
89     STree(void) { Init(); }
90
91     void Insert(int val) {
92         Node ** p = Insert(root, val);
93         if (*p != null) Rebuild(*p);
94     }
95     int Rank(int val) {
96         Node * now = root;
97         int ans = 1;
98         while (now != null) { // 非递归求排名
99             if (now->key >= val) now = now->ch[0];
100             else {
101                 ans += now->ch[0]->size + now->exist;
102                 now = now->ch[1];
103             }
104         }
105         return ans;
106     }
107     int Kth(int k) {
108         Node * now = root;
109         while (now != null) { // 非递归求第K大
110             if (now->ch[0]->size + 1 == k && now->exist) return now->key;
111             else if (now->ch[0]->size >= k) now = now->ch[0];
112             else k -= now->ch[0]->size + now->exist, now = now->ch[1];
113         }
114     }
115     void Erase(int k) {
116         Erase(root, Rank(k));
117         if (root->size < alpha * root->cover) Rebuild(root);
118     }
119     void Erase_kth(int k) {
120         Erase(root, k);
121         if (root->size < alpha * root->cover) Rebuild(root);
122     }
123 }sTree;
124 #undef MAXN
125
126 }
127
128 int main() {
129     Scapegoat_Tree::sTree.Init();
130     int _; cin >> _;
131     while(_--) {
132         int opt, x;
133         cin >> opt >> x;
134         if(opt == 1) Scapegoat_Tree::sTree.Insert(x);
135         else if(opt == 2) Scapegoat_Tree::sTree.Erase(x);
136         else if(opt == 3) cout << Scapegoat_Tree::sTree.Rank(x) << endl;
137         else if(opt == 4) cout << Scapegoat_Tree::sTree.Kth(x) << endl;
138         else if(opt == 5) cout << Scapegoat_Tree::sTree.Kth(Scapegoat_Tree::sTree.Rank(
139             x) - 1) << endl;
140         else if(opt == 6) cout << Scapegoat_Tree::sTree.Kth(Scapegoat_Tree::sTree.Rank(
141             x + 1)) << endl;
142     }
143 }

```

3.47 Splay

```

1  class node {
2  public:
3      int id;
4      node *l;
5      node *r;
6      node *p;
7      bool rev;
8      int sz;
9      // declare extra variables
10
11     node(int _id) {
12         id = _id;
13         l = r = p = nullptr;
14         rev = false;
15         sz = 1;
16         // init extra variables
17     }
18
19     void unsafe_reverse() {
20         rev ^= 1;
21         std::swap(l, r);
22         pull();
23     }
24
25     // apply changes:
26     void unsafe_apply() {
27     }
28
29     void push() {
30         if (rev) {
31             if (l != nullptr) {
32                 l -> unsafe_reverse();
33             }
34             if (r != nullptr) {
35                 r -> unsafe_reverse();
36             }
37             rev = 0;
38         }
39         // now push everything else:
40     }
41
42     void pull() {
43         sz = 1;
44         // now init from self:
45
46         if (l != nullptr) {
47             l -> p = this;
48             sz += l -> sz;
49             // now pull from l:
50
51         }
52         if (r != nullptr) {
53             r -> p = this;
54             sz += r -> sz;
55             // now pull from r:
56
57         }

```

```

58     }
59 };
60
61 void debug_node(node* v, std::string pref = "") {
62 #ifdef LOCAL
63     if (v != nullptr) {
64         debug_node(v -> r, pref + " ");
65         std::cerr << pref << "-" << " " << v -> id << '\n';
66         debug_node(v -> l, pref + " ");
67     } else {
68         std::cerr << pref << "-" << " " << "NULL" << '\n';
69     }
70 #endif
71 }
72
73 namespace splay_tree {
74
75 bool is_bst_root(node* v) {
76     if (v == nullptr) {
77         return false;
78     }
79     return (v -> p == nullptr || (v -> p -> l != v && v -> p -> r != v));
80 }
81
82 void rotate(node* v) {
83     node* u = v -> p;
84     assert(u != nullptr);
85     u -> push();
86     v -> push();
87     v -> p = u -> p;
88     if (v -> p != nullptr) {
89         if (v -> p -> l == u) {
90             v -> p -> l = v;
91         }
92         if (v -> p -> r == u) {
93             v -> p -> r = v;
94         }
95     }
96     if (v == u -> l) {
97         u -> l = v -> r;
98         v -> r = u;
99     } else {
100         u -> r = v -> l;
101         v -> l = u;
102     }
103     u -> pull();
104     v -> pull();
105 }
106
107 void splay(node* v) {
108     if (v == nullptr) {
109         return ;
110     }
111     while (!is_bst_root(v)) {
112         node* u = v -> p;
113         if (!is_bst_root(u)) {
114             if ((u -> l == v) ^ (u -> p -> l == u)) {
115                 rotate(v);
116             } else {
117                 rotate(u);

```

```

117         }
118     }
119     rotate(v);
120 }
121 }
122
123 std::pair<node*, int> find(node* v, const std::function<int(node*)> &go_to) {
124     // go_to returns: 0 -- found; -1 -- go left; 1 -- go right
125     // find returns the last vertex on the descent and its go_to
126     if (v == nullptr) {
127         return {nullptr, 0};
128     }
129     splay(v);
130     int dir;
131     while (true) {
132         v -> push();
133         dir = go_to(v);
134         if (dir == 0) {
135             break ;
136         }
137         node* u = (dir == -1 ? v -> l : v -> r);
138         if (u == nullptr) {
139             break ;
140         }
141         v = u;
142     }
143     splay(v);
144     return {v, dir};
145 }
146
147 node* get_leftmost(node* v) {
148     return find(v, [&](node*) { return -1; }).first;
149 }
150
151 node* get_rightmost(node* v) {
152     return find(v, [&](node*) { return 1; }).first;
153 }
154
155 node* get_kth(node* v, int k) { // 0-indexed
156     std::pair<node*, int> p = find(v, [&](node* u) {
157         if (u -> l != nullptr) {
158             if (u -> l -> sz > k) {
159                 return -1;
160             }
161             k -= u -> l -> sz;
162         }
163         if (k == 0) {
164             return 0;
165         }
166         k--;
167         return 1;
168     });
169     return (p.second == 0 ? p.first : nullptr);
170 }
171
172 int get_position(node* v) { // 0-indexed
173     splay(v);
174     return (v -> l != nullptr ? v -> l -> sz : 0);
175 }

```

```

176
177 node* get_bst_root(node* v) {
178     splay(v);
179     return v;
180 }
181
182 std::pair<node*, node*> split(node* v, const std::function<bool(node*)> &is_right) {
183     if (v == nullptr) {
184         return {nullptr, nullptr};
185     }
186     std::pair<node*, int> p = find(v, [&](node* u) { return is_right(u) ? -1 : 1; });
187     v = p.first;
188     v -> push();
189     if (p.second == -1) {
190         node* u = v -> l;
191         if (u == nullptr) {
192             return {nullptr, v};
193         }
194         v -> l = nullptr;
195         u -> p = v -> p;
196         u = get_rightmost(u);
197         v -> p = u;
198         v -> pull();
199         return {u, v};
200     } else {
201         node* u = v -> r;
202         if (u == nullptr) {
203             return {v, nullptr};
204         }
205         v -> r = nullptr;
206         v -> pull();
207         return {v, u};
208     }
209 }
210
211 std::pair<node*, node*> split_leftmost_k(node* v, int k) {
212     return split(v, [&](node* u) {
213         int left_and_me = (u -> l != nullptr ? u -> l -> sz : 0) + 1;
214         if (k >= left_and_me) {
215             k -= left_and_me;
216             return false;
217         }
218         return true;
219     });
220 }
221
222 node* merge(node* v, node* u) {
223     if (v == nullptr) {
224         return u;
225     }
226     if (u == nullptr) {
227         return v;
228     }
229     v = get_rightmost(v);
230     assert(v -> r == nullptr);
231     splay(u);
232     v -> push();
233     v -> r = u;
234     v -> pull();

```



```

235     return v;
236 }
237
238 int count_left(node* v, const std::function<bool(node*)> &is_right) {
239     if (v == nullptr) {
240         return 0;
241     }
242     std::pair<node*, int> p = find(v, [&](node* u) { return is_right(u) ? -1 : 1; });
243     node* u = p.first;
244     return (u -> l != nullptr ? u -> l -> sz : 0) + (p.second == 1);
245 }
246
247 node* add(node* r, node* v, const std::function<bool(node*)> &go_left) {
248     std::pair<node*, node*> p = split(r, go_left);
249     return merge(p.first, merge(v, p.second));
250 }
251
252 node* remove(node* v) { // returns the new root
253     splay(v);
254     v -> push();
255     node* x = v -> l;
256     node* y = v -> r;
257     v -> l = v -> r = nullptr;
258     node* z = merge(x, y);
259     if (z != nullptr) {
260         z -> p = v -> p;
261     }
262     v -> p = nullptr;
263     v -> push();
264     v -> pull(); // now v might be reusable
265     return z;
266 }
267
268 node* next(node* v) {
269     splay(v);
270     v -> push();
271     if (v -> r == nullptr) {
272         return nullptr;
273     }
274     v = v -> r;
275     while (v -> l != nullptr) {
276         v -> push();
277         v = v -> l;
278     }
279     splay(v);
280     return v;
281 }
282
283 node* prev(node* v) {
284     splay(v);
285     v -> push();
286     if (v -> l == nullptr) {
287         return nullptr;
288     }
289     v = v -> l;
290     while (v -> r != nullptr) {
291         v -> push();
292         v = v -> r;
293     }

```

```

294     splay(v);
295     return v;
296 }
297
298 int get_size(node* v) {
299     splay(v);
300     return (v != nullptr ? v -> sz : 0);
301 }
302
303 template<typename... T>
304 void apply(node* v, T... args) {
305     splay(v);
306     v -> unsafe_apply(args...);
307 }
308
309 void reverse(node* v) {
310     splay(v);
311     v -> unsafe_reverse();
312 }
313
314 } // namespace splay_tree

```

3.48 01trie

```

1  template<typename T>
2  class Trie {
3  private :
4      Trie* next[2] = {nullptr};
5      int val;
6      const int maxl = 32;
7  public :
8      Trie() {}
9
10     void insert(T x) {
11         Trie* root = this;
12         for(int i = maxl; i >= 0; i--) {
13             int u = x >> i & 1;
14             if(root -> next[u] == nullptr) root -> next[u] = new Trie();
15             root = root -> next[u];
16             root -> val ++;
17         }
18     }
19
20     void del(T x) {
21         Trie* root = this;
22         for (int i = maxl; i >= 0; i--) {
23             root = root -> next[x >> i & 1];
24             root -> val --;
25         }
26     }
27
28     T search(T x) {
29         T ans = 0;
30         Trie* root = this;
31         for (int i = maxl; i >= 0; i--) {
32             int u = x >> i & 1;
33             if (root -> next[!u] && root -> next[!u] -> val) {
34                 ans += 1 << i;

```

```

35         root = root -> next[!u];
36     } else {
37         root = root -> next[u];
38     }
39 }
40 return ans;
41 }
42 };

```

3.49 Treap

```

1  struct Tree {
2      Tree *l;
3      Tree *r;
4      int x;
5      int siz;
6      Tree(Tree *v) { *this = *v; }
7      Tree(int x = 0) : l(nullptr), r(nullptr), x(x), siz(1) {}
8      void pull() {
9          siz = 1;
10         if (l != nullptr) {
11             siz += l->siz;
12         }
13         if (r != nullptr) {
14             siz += r->siz;
15         }
16     }
17 };
18
19 int cnt = 0;
20 constexpr int N = 1e7;
21 Tree pool[N];
22
23 std::mt19937 rnd(std::chrono::steady_clock::now().time_since_epoch().count());
24
25 template<class... T>
26 Tree *newTree(T... x) {
27     Tree *t = &pool[cnt++];
28     *t = Tree(x...);
29     return t;
30 }
31
32 Tree *merge(Tree *a, Tree *b) {
33     if (a == nullptr) {
34         return b;
35     }
36     if (b == nullptr) {
37         return a;
38     }
39     Tree *t;
40     if (int(rnd() % (a->siz + b->siz)) < a->siz) {
41         t = newTree(a);
42         t->r = merge(t->r, b);
43     } else {
44         t = newTree(b);
45         t->l = merge(a, t->l);
46     }
47     t->pull();

```

```
48     return t;
49 }
50
51 std::pair<Tree *, Tree *> split(Tree *t, int k) {
52     if (t == nullptr || k == 0) {
53         return {nullptr, t};
54     }
55     if (t->siz == k) {
56         return {t, nullptr};
57     }
58     int szl = t->l == nullptr ? 0 : t->l->siz;
59     Tree *u = newTree(t);
60     if (k <= szl) {
61         auto [a, b] = split(t->l, k);
62         u->l = b;
63         u->pull();
64         return {a, u};
65     } else {
66         auto [a, b] = split(t->r, k - 1 - szl);
67         u->r = a;
68         u->pull();
69         return {u, b};
70     }
71 }
```

4 博弈论

4.1 bash

```
1 template <typename T>
2 bool bash(const T& a, const T& b) {
3     return a % (b + 1);
4 }
```

4.2 fibonacci

```
1 template <typename T>
2 bool Fibonacci(const T& x) {
3     std::unordered_map<int, bool> was;
4     std::vector<int> fib(51);
5     fib[1] = 1; fib[2] = 2;
6     for (int i = 3; i <= 50; i++) {
7         fib[i] = fib[i - 1] + fib[i - 2];
8         was[fib[i]] = true;
9     }
10    return !was[fib[x]];
11 }
```

4.3 nim

```
1 template <typename T>
2 bool nimGame(const std::vector<T>& stones) {
3     int res = 0;
4     for (int &x : stones) {
5         res ^= x;
6     }
7     return res;
8 }
```

4.4 wythoff

```
1 template <typename T>
2 bool wythoff(T& a, T& b) {
3     if (a > b) std::swap(a, b);
4     T delta = b - a;
5     T res = delta * (1.0 + sqrt(5.0)) / 2;
6     return !(res == a);
7 }
8
9 template <typename T>
10 bool wythoff_exp(T& a, T& b, T& k) {
11     k++;
12     if (a > b) std::swap(a, b);
13     T delta = (b - a) / k;
14     T res1 = delta * (2 - k + sqrt(4.0 + k * k)) / 2;
15     T res2 = delta * (2 + k + sqrt(4.0 + k * k)) / 2;
16     return !(res1 == a && res2 == b);
17 }
```

4.5 sgFunction

```
1 // SG函数
2 #define N 1001
3 //f[]: 可以取走的石子个数
4 //sg[]: 0~n的SG函数值
5 int f[N], sg[N], mex[N];
6
7 void getSG(int n) {
8     int i, j;
9     memset(sg, 0, sizeof(sg));
10    for (i = 1; i <= n; i++) {
11        memset(mex, 0, sizeof(mex));
12        for (j = 1; f[j] <= i; j++)
13            mex[sg[i - f[j]]] = 1;
14        for (j = 0; j <= n; j++) { //求mex{}中未出现的最小的非负整数
15            if (mex[j] == 0) {
16                sg[i] = j;
17                break;
18            }
19        }
20    }
21 }
```

5 树与森林

5.1 forest

```

1  template <typename T>
2  class forest : public graph<T> {
3  public:
4      using graph<T>::edges;
5      using graph<T>::g;
6      using graph<T>::n;
7
8      forest(int _n) : graph<T>(_n) {}
9
10     int add (int from, int to, T cost = 1) {
11         assert(0 <= from && from < n && 0 <= to && to < n);
12         int id = (int) edges.size();
13         assert(id < n - 1);
14         g[from].push_back(id);
15         g[to].push_back(id);
16         edges.push_back({from, to, cost});
17         return id;
18     }
19 };

```

5.2 dfs-forest

```

1  template <typename T>
2  class dfs_forest : public forest<T> {
3  public:
4      using forest<T>::edges;
5      using forest<T>::g;
6      using forest<T>::n;
7
8      std::vector<int> pv;
9      std::vector<int> pe;
10     std::vector<int> order;
11     std::vector<int> pos;
12     std::vector<int> end;
13     std::vector<int> sz;
14     std::vector<int> root;
15     std::vector<int> depth;
16     std::vector<T> dist;
17
18     dfs_forest(int _n) : forest<T>(_n) {}
19
20     void init() {
21         pv = std::vector<int>(n, -1);
22         pe = std::vector<int>(n, -1);
23         order.clear();
24         pos = std::vector<int>(n, -1);
25         end = std::vector<int>(n, -1);
26         sz = std::vector<int>(n, 0);
27         root = std::vector<int>(n, -1);
28         depth = std::vector<int>(n, -1);
29         dist = std::vector<T>(n);
30     }
31
32     void clear() {

```

```

33     pv.clear();
34     pe.clear();
35     order.clear();
36     pos.clear();
37     end.clear();
38     sz.clear();
39     root.clear();
40     depth.clear();
41     dist.clear();
42 }
43
44 private:
45     void do_dfs(int v) {
46         pos[v] = (int) order.size();
47         order.push_back(v);
48         sz[v] = 1;
49         for (int id : g[v]) {
50             if (id == pe[v]) {
51                 continue;
52             }
53             auto &e = edges[id];
54             int to = e.from ^ e.to ^ v;
55             depth[to] = depth[v] + 1;
56             dist[to] = dist[v] + e.cost;
57             pv[to] = v;
58             pe[to] = id;
59             root[to] = (root[v] != -1 ? root[v] : to);
60             do_dfs(to);
61             sz[v] += sz[to];
62         }
63         end[v] = (int) order.size() - 1;
64     }
65
66     void do_dfs_from(int v) {
67         depth[v] = 0;
68         dist[v] = T{};
69         root[v] = v;
70         pv[v] = pe[v] = -1;
71         do_dfs(v);
72     }
73
74 public:
75     void dfs(int v, bool clear_order = true) {
76         if (pv.empty()) {
77             init();
78         } else {
79             if (clear_order) {
80                 order.clear();
81             }
82         }
83         do_dfs_from(v);
84     }
85
86     void dfs_all() {
87         init();
88         for (int v = 0; v < n; v++) {
89             if (depth[v] == -1) {
90                 do_dfs_from(v);
91             }
92         }
93     }

```



```

92     }
93     assert((int) order.size() == n);
94 }
95 };

```

5.3 lca-forest

```

1  template <typename T>
2  class lca_forest : public dfs_forest<T> {
3  public:
4      using dfs_forest<T>::edges;
5      using dfs_forest<T>::g;
6      using dfs_forest<T>::n;
7      using dfs_forest<T>::pv;
8      using dfs_forest<T>::pos;
9      using dfs_forest<T>::end;
10     using dfs_forest<T>::depth;
11
12     int h;
13     std::vector<std::vector<int>>> pr;
14
15     lca_forest(int _n) : dfs_forest<T>(_n) {}
16
17     inline void build_lca() {
18         assert(!pv.empty());
19         int max_depth = 0;
20         for (int i = 0; i < n; i++) {
21             max_depth = std::max(max_depth, depth[i]);
22         }
23         h = 1;
24         while ((1 << h) <= max_depth) {
25             h++;
26         }
27         pr.resize(n);
28         for (int i = 0; i < n; i++) {
29             pr[i].resize(h);
30             pr[i][0] = pv[i];
31         }
32         for (int j = 1; j < h; j++) {
33             for (int i = 0; i < n; i++) {
34                 pr[i][j] = (pr[i][j - 1] == -1 ? -1 : pr[pr[i][j - 1]][j - 1]);
35             }
36         }
37     }
38
39     inline bool anc(int x, int y) {
40         return (pos[x] <= pos[y] && end[y] <= end[x]);
41     }
42
43     inline int go_up(int x, int up) {
44         assert(!pr.empty());
45         up = std::min(up, (1 << h) - 1);
46         for (int j = h - 1; j >= 0; j--) {
47             if (up & (1 << j)) {
48                 x = pr[x][j];
49                 if (x == -1) {
50                     break;
51                 }

```

```

52     }
53 }
54 return x;
55 }
56
57 inline int lca(int x, int y) {
58     assert(!pr.empty());
59     if (anc(x, y)) {
60         return x;
61     }
62     if (anc(y, x)) {
63         return y;
64     }
65     for (int j = h - 1; j >= 0; j--) {
66         if (pr[x][j] != -1 && !anc(pr[x][j], y)) {
67             x = pr[x][j];
68         }
69     }
70     return pr[x][0];
71 }
72
73 inline int dist(int x, int y) {
74     return depth[x] + depth[y] - depth[lca(x, y)] * 2;
75 }
76 };

```

5.4 重链剖分

```

1  const int maxn = 4e5 + 10;
2
3  struct Edge {
4      int v, next;
5  } e[maxn << 1];
6
7  int head[maxn * 2], cnt;
8
9  inline void add(int u, int v) {
10     e[++cnt].v = v;
11     e[cnt].next = head[u];
12     head[u] = cnt;
13 }
14
15 int fa[maxn], dep[maxn], siz[maxn], son[maxn];
16
17 void dfs1(int u, int par) {
18     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
19     for(int i = head[u]; ~i; i = e[i].next) {
20         int v = e[i].v;
21         if(v == par) continue;
22         dfs1(v, u);
23         siz[u] += siz[v];
24         if(!son[u] || siz[v] > siz[son[u]])
25             son[u] = v;
26     }
27 }
28
29 int dfn[maxn], top[maxn], nodeof[maxn], tim;
30

```

```

31 void dfs2(int u, int topf) {
32     nodeof[dfn[u] = ++tim] = u;
33     top[u] = topf;
34     if(!son[u]) return ;
35     dfs2(son[u], topf);
36     for(int i = head[u]; ~i; i = e[i].next) {
37         int v = e[i].v;
38         if(v == fa[u] || v == son[u]) continue;
39         dfs2(v, v);
40     }
41 }
42
43 int w[maxn];
44
45 #define lc u << 1
46 #define rc u << 1 | 1
47 #define mid (t[u].l + t[u].r) / 2
48 struct Tree {
49     int l, r, sum, tag;
50 }t[maxn << 2];
51 inline void push_up(int u) ;
52 inline void push_down(int u) ;
53 void build(int u, int l, int r) ;
54 void modify(int u, int ql, int qr, int v) ;
55 int query(int u, int ql, int qr) ;
56
57 void modify_chain(int x, int y, int val) {
58     while(top[x] != top[y]) {
59         if(dep[top[x]] < dep[top[y]]) swap(x, y);
60         modify(1, dfn[top[x]], dfn[x], val);
61         x = fa[top[x]];
62     }
63     if(dep[x] > dep[y]) swap(x, y);
64     modify(1, dfn[x], dfn[y], val);
65 }
66
67 int query_chain(int x, int y) {
68     int ans = 0;
69     while(top[x] != top[y]) {
70         if(dep[top[x]] < dep[top[y]]) swap(x, y);
71         ans += query(1, dfn[top[x]], dfn[x]);
72         x = fa[top[x]];
73     }
74     if(dep[x] > dep[y]) swap(x, y);
75     ans += query(1, dfn[x], dfn[y]);
76     return ans;
77 }
78
79 signed main() {
80     memset(head, -1, sizeof(head));
81     int n; cin >> n;
82     for(int i = 1; i <= n; i++) cin >> w[i];
83     for(int i = 1; i <= n - 1; i++) {
84         int u, v; cin >> u >> v;
85         add(u, v);
86         add(v, u);
87     }
88     dfs1(1, 0);
89     dfs2(1, 1);

```

```

90     build(1, 1, n);
91     int m; cin >> m;
92     while(m--) {
93         int opt; cin >> opt;
94         if(opt == 1) {
95             int x, y, val; cin >> x >> y >> val;
96             modify_chain(x, y, val);
97         }
98         else if(opt == 2) {
99             int x, val; cin >> x >> val;
100             modify(1, dfn[x], dfn[x] + siz[x] - 1, val);
101         }
102         else if(opt == 3) {
103             int x, y; cin >> x >> y;
104             cout << query_chain(x, y) << endl;
105         }
106         else if(opt == 4) {
107             int x; cin >> x;
108             cout << query(1, dfn[x], dfn[x] + siz[x] - 1) << endl;
109         }
110     }
111 }

```

5.5 tree-diameter

```

1  template <typename T>
2  std::vector<int> find_tree_diameter(const forest<T>& g, T& diameter) {
3      diameter = 0;
4      int st = 0, ed = 0;
5      std::vector<T> dis(g.n);
6      std::vector<int> pre(g.n);
7
8      std::function<void(int, int)> dfs1 = [&](int u, int parent) {
9          if (dis[u] > dis[st]) st = u;
10         for (int id : g.g[u]) {
11             auto& e = g.edges[id];
12             int to = e.from ^ e.to ^ u;
13             if (to == parent) continue;
14             dis[to] = dis[u] + e.cost;
15             dfs1(to, u);
16         }
17     };
18
19     std::function<void(int, int)> dfs2 = [&](int u, int parent) {
20         if (dis[u] > dis[ed]) ed = u;
21         for (int id : g.g[u]) {
22             auto& e = g.edges[id];
23             int to = e.from ^ e.to ^ u;
24             if (to == parent) continue;
25             pre[to] = u;
26             dis[to] = dis[u] + e.cost;
27             dfs2(to, u);
28         }
29     };
30
31     dfs1(0, -1);
32     dis.assign(g.n, 0);
33     dfs2(st, -1);

```

```

34     std::vector<int> vertexs{ed};
35     int now = ed;
36     do {
37         vertexs.push_back(pre[now]);
38         now = pre[now];
39     } while (now != st);
40     reverse(vertexs.begin(), vertexs.end());
41     return vertexs;
42 }
43
44 template <typename T>
45 std::vector<int> find_tree_all_diameters(const forest<T>& g, T& diameter) {
46     diameter = 0;
47     std::vector<std::array<T, 2>> dp(g.n);
48     std::vector<int> down(g.n);
49     std::vector<T> up(g.n);
50
51     std::function<void(int, int)> dfs1 = [&](int u, int parent) {
52         for (int id : g.g[u]) {
53             auto& e = g.edges[id];
54             int to = e.from ^ e.to ^ u;
55             if (to == parent) continue;
56             dfs1(to, u);
57             if (dp[to][0] + e.cost > dp[u][0]) {
58                 dp[u][1] = dp[u][0];
59                 dp[u][0] = dp[to][0] + e.cost;
60                 down[u] = to;
61             } else if (dp[to][0] + e.cost > dp[u][1]) {
62                 dp[u][1] = dp[to][0] + e.cost;
63             }
64             diameter = std::max(diameter, dp[u][1] + dp[u][0]);
65         }
66     };
67
68     std::function<void(int, int)> dfs2 = [&](int u, int parent) {
69         for (int id : g.g[u]) {
70             auto& e = g.edges[id];
71             int to = e.from ^ e.to ^ u;
72             if (to == parent) continue;
73             up[to] = up[u] + e.cost;
74             if (down[u] == to) up[to] = std::max(up[to], dp[u][1] + e.cost);
75             else up[to] = std::max(up[to], dp[u][0] + e.cost);
76             dfs2(to, u);
77         }
78     };
79
80     dfs1(0, -1);
81     dfs2(0, -1);
82     std::vector<int> vertexs;
83     for (int i = 0; i < g.n; i++) {
84         std::vector<int> dis{ dp[i][1], dp[i][0], up[i] };
85         sort(dis.begin(), dis.end());
86         if (dis[1] + dis[2] == diameter) vertexs.push_back(i);
87     }
88     return vertexs;
89 };

```

5.6 树的重心

```

1  const int N = 1e5 + 10;
2
3  struct Edge {
4      int v, next;
5  }e[N * 2];
6
7  int cnt, head[N * 2];
8
9  int d[N], R[2], root;
10 int n;
11
12 int balance;
13
14 inline void add(int u, int v) {
15     e[++cnt].v = v;
16     e[cnt].next = head[u];
17     head[u] = cnt;
18 }
19
20 void DFS(int u, int fa) {
21     d[u] = 1;
22     int res = 0;
23     for(int i = head[u] ; i != -1 ; i = e[i].next) {
24         int v = e[i].v;
25         if(v == fa) continue;
26         DFS(v, u);
27         d[u] += d[v];
28         res = max(res, d[v]);
29     }
30     res = max(res, n - d[u]);
31     if(res < balance) {
32         R[root++] = u;
33         balance = res;
34     }
35     else if(res == balance) {
36         R[root++] = u;
37     }
38 }
39
40 int main() {
41     cin >> n;
42     balance = n / 2;
43     for(int i = 1; i < n; i++) {
44         int u, v;
45         cin >> u >> v;
46         add(u, v);
47         add(v, u);
48     }
49     DFS(1, 0);
50     if(R[0]) cout << R[0] << endl;
51     if(R[1]) cout << R[1] << endl;
52 }

```

5.7 树的最大匹配

```

1  // 设状态为f[u][1/0]表示以u为根的子树与儿子连边/不连边的最大匹配

```

```

2
3 const int N = 1e5 + 10;
4 vector<int> g[N];
5 int f[N][2];
6
7 void dfs(int u, int fa) {
8     int mn = INF;
9     for(auto v : g[u]) {
10         if(v == fa) continue ;
11         dfs(v, u);
12         f[u][0] += f[v][1]; // u不与儿子连边, 即加上所有与儿子连边的v
13         f[u][1] += f[v][1]; // u与儿子连边, 即加上一个不与儿子连边的v和其他所有与儿子连边的v
14         mn = min(mn, f[v][1] - f[v][0]);
15     }
16     if(mn != INF) f[u][1] = dp[u][1] - mn + 1;
17 }

```

5.8 树分治-点分治

```

1 // 题意: n个节点的树, 存在边权, 范围1e18
2 // 求任意两点之间点集的子集中两点之间路径异或和为0的个数
3 // u<v,u'<v',(u',v') ⊆ path(u,v),求path(u', v')异或和==0
4
5 struct Edge {
6     int to, nxt;
7     ll w;
8 };
9 const int N = int(1e5 + 10);
10 const int M = N << 1;
11
12 struct Graph {
13     int head[N];
14     Edge eg[M];
15     int tot;
16
17     void init(int n) {
18         memset(head, -1, sizeof(int) * ++n);
19     }
20
21     inline void addEdge(int u, int v, ll w) {
22         eg[tot] = {v, head[u], w};
23         head[u] = tot++;
24     }
25 } gh;
26
27 bool vis[N];
28 // q队列, fa祖先, sz是子树大小, smx是子树最大
29 int q[N], fa[N], sz[N], smx[N];
30
31 int froot(int s) {
32     int l, r, mn = N, rt = 0;
33     q[l = r = 1] = s;
34     while (l <= r) {
35         int u = q[l++];
36         sz[u] = 1;
37         smx[u] = 0;
38         for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
39             int v = gh.eg[i].to;

```

```

40         if (v == fa[u] || vis[v]) continue;
41         fa[v] = u;
42         q[++r] = v;
43     }
44 }
45 // 反向遍历所有点算size
46 while (--l) {
47     int u = q[l];
48     int mx = max(smx[u], r - sz[u]);
49     if (mx < mn) mn = mx, rt = u;
50     if (l == 1) break; // 根节点没有fa
51     sz[fa[u]] += sz[u];
52     smx[fa[u]] = max(smx[fa[u]], sz[u]);
53 }
54 return rt;
55 }
56
57 // sons子树方向节点个数, val根到该节点异或和, gc边后继方向的节点个数
58 int sons[N], gc[M];
59 ll val[N];
60 ll ans = 0;
61 int n;
62
63 const int MOD = int(1e9 + 7);
64
65 ll nums[N];
66 int cnt[N];
67
68 void go(int s, int rt) {
69     fa[s] = rt;
70     val[s] = 0;
71     int l, r;
72     // 不计算s
73     q[l = r = 0] = s;
74     int m = 0;
75     while (l <= r) {
76         int u = q[l++];
77         nums[m++] = val[u];
78         for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
79             int v = gh.eg[i].to;
80             if (v == fa[u] || vis[v]) continue;
81             fa[v] = u;
82             q[++r] = v;
83             val[v] = val[u] ^ gh.eg[i].w;
84             // 这个点方向后面有多少点
85             sons[v] = gc[i];
86         }
87     }
88     sort(nums, nums + m);
89     m = unique(nums, nums + m) - nums;
90     mst(cnt, 0, m);
91     // 遍历分支
92     for (int j = gh.head[s]; ~j; j = gh.eg[j].nxt) {
93         // 分支的根
94         int du = gh.eg[j].to;
95         if (vis[du]) continue;
96         q[l = r = 1] = du;
97         while (l <= r) {
98             int u = q[l++];

```



```

99         int k = lower_bound(nums, nums + m, val[u]) - nums;
100         (ans += 1ll * sons[u] * cnt[k] % MOD) %= MOD;
101         if (val[u] == 0) {
102             (ans += 1ll * sons[u] * (n - gc[j]) % MOD) %= MOD;
103         }
104         for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
105             int v = gh.eg[i].to;
106             if (v == fa[u] || vis[v]) continue;
107             q[++r] = v;
108         }
109     }
110     // 增加这个方向的值
111     while (--l) {
112         int u = q[l];
113         int k = lower_bound(nums, nums + m, val[u]) - nums;
114         (cnt[k] += sons[u]) %= MOD;
115     }
116 }
117 }
118
119 void work(int u) {
120     // 换根
121     u = froot(u);
122     vis[u] = true;
123     go(u, 0);
124     for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
125         int v = gh.eg[i].to;
126         if (vis[v]) continue;
127         work(v);
128     }
129 }
130
131 // 预处理边后继节点个数
132 int pdfs(int u, int f) {
133     int fg_id = -1;
134     int s = 1;
135     for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
136         int v = gh.eg[i].to;
137         if (v == f) { // 记录父边ID
138             fg_id = i;
139             continue;
140         }
141         int c = pdfs(v, u);
142         gc[i] = c;
143         s += c;
144     }
145     // 存在父边
146     if (~fg_id) gc[fg_id] = n - s;
147     return s;
148 }
149
150 void solve() {
151     while (cin >> n) {
152         gh.init(n);
153         for (int i = 2; i <= n; i++) {
154             int u, v;
155             ll w;
156             u = i;
157             cin >> v >> w;

```

```

158         gh.addEdge(u, v, w);
159         gh.addEdge(v, u, w);
160     }
161     mst(vis, false, n + 1);
162     pdfs(1, 0);
163     ans = 0;
164     work(1);
165     cout << ans << endl;
166 }
167 }

```

5.9 树上 dsu-维护路径信息

```

1  const int N = 1e5 + 10;
2
3  vector<int> g[N];
4  int siz[N], dep[N], son[N], dfn[N], nodeof[N], tim;
5
6  void calc(int u, int w) {
7      // ....对u这一节点进行单独处理
8      if(w > 0) // ....计算贡献
9          else // ....撤销影响
10 }
11
12 void dfs1(int u, int fa) {
13     dep[u] = dep[fa] + (siz[u] = 1);
14     nodeof[dfn[u] = ++tim] = u;
15     for(auto v : g[u]) {
16         if(v == fa) continue ;
17         dfs1(v, u);
18         siz[u] += siz[v];
19         if(!son[u] || siz[v] > siz[son[u]]) son[u] = v;
20     }
21 }
22
23 void dfs2(int u, int fa, bool keep) {
24     for(auto v : g[u]) {
25         if(v == fa || v == son[u]) continue ;
26         dfs2(v, u, 0);
27     }
28     if(son[u]) {
29         dfs2(son[u], u, 1);
30     }
31     for(auto v : g[u]) {
32         if(v == fa || v == son[u]) continue ;
33         for(int j = 0; j < siz[v]; j++) {
34             // ....更新答案
35         }
36         for(int j = 0; j < siz[v]; j++) {
37             calc(nodeof[dfn[v] + j], 1);
38         }
39     }
40     calc(u, 1);
41     // ....更新答案
42     if(!keep) {
43         for(int i = 0; i < siz[u]; i++) calc(nodeof[dfn[u] + i], -1);
44     }
45 }

```

```

46
47 int main() {
48     int n; cin >> n;
49     for(int i = 1; i < n; i++) {
50         int u, v;
51         g[u].push_back(v);
52         g[v].push_back(u);
53     }
54     dfs1(1, 0);
55     dfs2(1, 0, 0);
56 }

```

5.10 树上 dsu-维护子树信息

```

1  const int N = 2e5 + 10;
2
3  vector<int> g[N];
4
5  int siz[N], son[N], col[N];
6  int ans[N], cnt[N];
7  bool vis[N];
8  int maxx, sum;
9  // maxx为每棵子树里出现最多的颜色, sum为编号和
10
11
12 void calc(int u, int fa, int val) {
13     /*
14     针对不同问题, 采取的操作
15     */
16     else if(val > 0 && cnt[col[u]] == maxx) sum += col[u];
17     for(auto v : g[u]) {
18         if(v != fa && !vis[v]) calc(v, u, w);
19     }
20 }
21
22 void dfs1(int u, int fa) {
23     siz[u] = 1;
24     for(auto v : g[u]) {
25         if(v == fa) continue ;
26         dfs1(v, u);
27         siz[u] += siz[v];
28         if(!son[u] || siz[v] > siz[son[u]]) son[u] = v;
29     }
30 }
31
32 void dfs2(int u, int fa, bool keep) {
33     for(auto v : g[u]) {
34         if(v != fa && v != son[u]) {
35             dfs2(v, u, 0);
36         }
37     }
38     if(son[u]) {
39         dfs2(son[u], u, 1);
40         vis[son[u]] = 1;
41     }
42     calc(u, fa, 1);
43     ans[u] = sum;
44     if(son[u]) vis[son[u]] = 0;

```

```

45     if(!keep) {
46         calc(u, fa, -1);
47         maxx = sum = 0;
48     }
49 }
50
51 int main() {
52     int n; cin >> n;
53     for(int i = 1; i <= n; i++) cin >> col[i];
54     for(int i = 1; i < n; i++) {
55         int u, v; cin >> u >> v;
56         g[u].push_back(v);
57         g[v].push_back(u);
58     }
59     dfs0(1, 0);
60     dfs1(1, 0, false);
61     for(int i = 1; i <= n; i++) cout << ans[i] << endl;
62 }

```

5.11 树上 K 祖先

```

1  //倍增KFA, 空间大点, 但是好写
2  vector<int> g[N];
3
4  int anc[N][20];
5  void dfs(int u, int fa) {
6      anc[u][0] = fa;
7      for (int i = 1; i <= 19; i++) anc[u][i] = anc[anc[u][i - 1]][i - 1];
8      for (auto &v: g[u])
9          if (v != fa) dfs(v, u);
10 }
11
12 int kthFa(int u, int k) {
13     int bit = 0;
14     while (k) {
15         if (k & 1) u = anc[u][bit];
16         k >>= 1;
17         bit++;
18     }
19     return u;
20 }
21
22
23 //树剖KFA
24 int siz[N], son[N], dep[N], fa[N], top[N];
25 int id[N], nodeOf[N], cnt;
26 void dfs(int u, int par) {
27     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
28     for (auto &v: g[u])
29         if (v != par) {
30             dfs(v, u);
31             siz[u] += siz[v];
32             if (!son[u] || siz[v] > siz[son[u]])
33                 son[u] = v;
34         }
35 }
36
37 void dfs2(int u, int topf) {

```

```

38     nodeOf[id[u] = ++cnt] = u, top[u] = topf;
39     if (!son[u]) return;
40     dfs2(son[u], topf);
41     for (auto &v: g[u])
42         if (v != fa[u] && v != son[u]) dfs2(v, v);
43 }
44
45 int kthFa(int u, int k) {
46     while (k >= id[u] - id[top[u]] + 1 && u) {
47         k -= id[u] - id[top[u]] + 1;
48         u = fa[top[u]];
49     }
50     return nodeOf[id[u] - k];
51 }

```

5.12 virtualTree

```

1  //虚树可以处理多次询问，并且每次询问只需要树上的K个关键点
2  //建立的虚树能保证点数 < 2 * K
3  //如果对虚树做dp，总体复杂度和ΣK有关
4  //考虑dp的时候，需要同时考虑非关键点对答案的影响
5
6  int n;
7
8  struct edge {
9      int nxt, to;
10 } e[N << 1];
11 int head[N], tot;
12 void add(int u, int v) { e[++tot] = edge{ head[u], v }, head[u] = tot; }
13
14 int dep[N], fa[N], topfa[N], siz[N], son[N], dfn[N], cnt;
15 void dfs(int u, int par) {
16     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
17     int max_son = -1;
18     for (int i = head[u], v; i; i = e[i].nxt)
19         if ((v = e[i].to) != par) {
20             dfs(v, u);
21             siz[u] += siz[v];
22             if (max_son < siz[v]) son[u] = v, max_son = siz[v];
23         }
24 }
25 void dfs2(int u, int topf) {
26     topfa[u] = topf, dfn[u] = ++cnt;
27     if (!son[u]) return;
28     dfs2(son[u], topf);
29     for (int i = head[u], v; i; i = e[i].nxt)
30         if ((v = e[i].to) != fa[u] && v != son[u]) dfs2(v, v);
31 }
32 int LCA(int x, int y) {
33     while (topfa[x] != topfa[y]) {
34         if (dep[topfa[x]] < dep[topfa[y]]) swap(x, y);
35         x = fa[topfa[x]];
36     }
37     return dep[x] < dep[y] ? x : y;
38 }
39 int getDis(int x, int y) { return dep[x] + dep[y] - 2 * dep[LCA(x, y)]; }
40
41 //建立虚树

```

```

42 int tag[N]; //tag[u] = 1 <=> 关键点
43 vector<int> g[N]; //虚树边
44 void add_edge(int u, int v) { g[u].push_back(v); }
45 int st[N], top, rt; //rt为虚树根
46 void insert(int u) {
47     if (top == 1) {
48         st[++top] = u;
49         return;
50     }
51     int lca = LCA(u, st[top]);
52     if (lca != st[top]) {
53         while (top > 1 && dfn[st[top - 1]] >= dfn[lca])
54             add_edge(st[top - 1], st[top]), top--;
55         if (lca != st[top]) add_edge(lca, st[top]), st[top] = lca;
56     }
57     st[++top] = u;
58 }
59 bool cmp(const int &x, const int &y) { return dfn[x] < dfn[y]; }
60 void build(vector<int> &v) {
61     st[top = 1] = rt;
62     sort(v.begin(), v.end(), cmp);
63     for (auto &i: v) {
64         tag[i] = 1;
65         if (i != rt) insert(i);
66     }
67     while (top > 1) add_edge(st[top - 1], st[top]), top--;
68 }
69
70
71 void dp(int u) {
72     //...
73 }
74 void clear(int u) { //清空虚树边和标记, 也可以和dp合并
75     for (auto &v: g[u]) clear(v);
76     g[u].clear(); tag[u] = 0;
77 }
78 void solve() {
79     //...
80     dp(rt); clear(rt);
81     //...
82 }
83
84 int main() {
85     scanf("%d", &n);
86     for (int i = 1; i < n; i++) {
87         int u, v; scanf("%d%d", &u, &v);
88         add(u, v); add(v, u);
89     }
90     //此处距离为1, 所以用dep替代dis, dis[fa[rt] = 0] = -1
91     dep[0] = -1, rt = 1;
92     dfs(rt, 0); dfs2(rt, rt);
93
94
95     int Q; scanf("%d", &Q);
96     while (Q--) {
97         int K; scanf("%d", &K); //读取关键点
98         for (int i = 1; i <= K; i++) scanf("%d", &a[i]);
99         //构建虚树
100        build(a);

```

```

101     solve();
102 }
103
104     return 0;
105 }

```

5.13 LCT

```

1  int ch[N][2], fa[N], rev[N], siz[N]; //基本内容
2  int sum[N], val[N], tag[N]; //另外要维护的
3  #define lc ch[u][0]
4  #define rc ch[u][1]
5  #define identify(u) (ch[fa[u]][1] == u)
6  #define isRoot(u) (u != ch[fa[u]][0] && u != ch[fa[u]][1])
7  void flip(int u) { swap(lc, rc); rev[u] ^= 1; }
8  void push_up(int u) {
9      siz[u] = siz[lc] + siz[rc] + 1;
10     //...
11 }
12 void push_down(int u) {
13     if (rev[u]) {
14         if (lc) flip(lc);
15         if (rc) flip(rc);
16         rev[u] = 0;
17     }
18     //...
19 }
20 void update(int u) { //当前点之上的所有点都push_down
21     if (!isRoot(u)) update(fa[u]);
22     push_down(u);
23 }
24 void rotate(int u) {
25     int f = fa[u], fc = identify(u);
26     int g = fa[f], gc = identify(f);
27     int uc = fc ^ 1, c = ch[u][uc];
28     if (!isRoot(f))
29         ch[g][gc] = u; fa[u] = g;
30     ch[f][fc] = c; fa[c] = f;
31     ch[u][uc] = f; fa[f] = u;
32     push_up(f); push_up(u);
33 }
34 void splay(int u) { //将u变为u所在的Splay的根
35     update(u);
36     for (int f; f = fa[u], !isRoot(u); rotate(u))
37         if (!isRoot(f)) rotate(identify(f) ^ identify(u) ? u : f);
38 }
39 int access(int u) { //将(rt, u)之间的路径变为实链
40     int pre = 0;
41     for (; u; u = fa[pre = u])
42         splay(u), rc = pre, push_up(u);
43     return pre;
44 }
45 void makeRoot(int u) { //将u变为整棵树的根(注意:不一定是当前splay的根)
46     u = access(u);
47     flip(u);
48 }
49 int findRoot(int u) {
50     access(u), splay(u);

```

```
51     while (lc) push_down(u), u = lc;
52     splay(u);
53     return u;
54 }
55 void link(int u, int v) {
56     makeRoot(u); splay(u);
57     if (findRoot(v) != u) fa[u] = v;
58 }
59 void split(int u, int v) {
60     makeRoot(u);
61     access(v); splay(v); //加了这个就将v变为splay的根
62 }
63 void cut(int u, int v) {
64     makeRoot(u); splay(u);
65     if (findRoot(v) == u && fa[v] == u && !ch[v][0]) {
66         fa[v] = ch[u][1] = 0;
67         push_up(u);
68     }
69 }
70 void fix(int u, int k) {
71     splay(u); val[u] = k;
72 }
```


6 数学

6.1 Mint

```

1  template <typename T>
2  T inverse(T a, T m) {
3      T u = 0, v = 1;
4      while (a != 0) {
5          T t = m / a;
6          m -= t * a; std::swap(a, m);
7          u -= t * v; std::swap(u, v);
8      }
9      assert(m == 1);
10     return u;
11 }
12
13 template <typename T>
14 class Modular {
15 public:
16     using Type = typename std::decay<decltype(T::value)>::type;
17
18     constexpr Modular() : value() {}
19     template <typename U>
20     Modular(const U& x) {
21         value = normalize(x);
22     }
23
24     template <typename U>
25     static Type normalize(const U& x) {
26         Type v;
27         if (-mod() <= x && x < mod()) v = static_cast<Type>(x);
28         else v = static_cast<Type>(x % mod());
29         if (v < 0) v += mod();
30         return v;
31     }
32
33     const Type& operator()() const { return value; }
34     template <typename U>
35     explicit operator U() const { return static_cast<U>(value); }
36     constexpr static Type mod() { return T::value; }
37
38     template <typename U> Modular& operator*=(const U& other) { return *this *= Modular(other); }
39     template <typename U> Modular& operator/=(const U& other) { return *this /= Modular(other); }
40     Modular& operator+=(const Modular& other) { if ((value += other.value) >= mod()) value -= mod(); return *this; }
41     Modular& operator-=(const Modular& other) { if ((value -= other.value) < 0) value += mod(); return *this; }
42     template <typename U> Modular& operator+=(const U& other) { return *this += Modular(other); }
43     template <typename U> Modular& operator-=(const U& other) { return *this -= Modular(other); }
44     Modular& operator++() { return *this += 1; }
45     Modular& operator--() { return *this -= 1; }
46     Modular operator++(int) { Modular result(*this); *this += 1; return result; }
47     Modular operator--(int) { Modular result(*this); *this -= 1; return result; }
48     Modular operator-(const) { return Modular(-value); }
49

```

```

50     template <typename U = T>
51     typename std::enable_if<std::is_same<typename Modular<U>::Type, int>::value,
Modular>::type& operator*=(const Modular& rhs) {
52 #ifdef _WIN32
53     uint64_t x = static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value);
54     uint32_t xh = static_cast<uint32_t>(x >> 32), xl = static_cast<uint32_t>(x), d, m;
55     asm(
56         "divl %4; \n\t"
57         : "=a" (d), "=d" (m)
58         : "d" (xh), "a" (xl), "r" (mod()))
59     );
60     value = m;
61 #else
62     value = normalize(static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value));
63 #endif
64     return *this;
65     }
66     template <typename U = T>
67     typename std::enable_if<std::is_same<typename Modular<U>::Type, long long>::value,
Modular>::type& operator*=(const Modular& rhs) {
68         long long q = static_cast<long long>(static_cast<long double>(value) * rhs.
value / mod());
69         value = normalize(value * rhs.value - q * mod());
70         return *this;
71     }
72     template <typename U = T>
73     typename std::enable_if<!std::is_integral<typename Modular<U>::Type>::value,
Modular>::type& operator*=(const Modular& rhs) {
74         value = normalize(value * rhs.value);
75         return *this;
76     }
77
78     Modular& operator/=(const Modular& other) { return *this *= Modular(inverse(other.
value, mod())); }
79
80     friend const Type& abs(const Modular& x) { return x.value; }
81
82     template <typename U>
83     friend bool operator==(const Modular<U>& lhs, const Modular<U>& rhs);
84
85     template <typename U>
86     friend bool operator<(const Modular<U>& lhs, const Modular<U>& rhs);
87
88     template <typename V, typename U>
89     friend V& operator>>(V& stream, Modular<U>& number);
90
91 private:
92     Type value;
93 };
94
95 template <typename T> bool operator==(const Modular<T>& lhs, const Modular<T>& rhs) {
    return lhs.value == rhs.value; }
96 template <typename T, typename U> bool operator==(const Modular<T>& lhs, U rhs) {
    return lhs == Modular<T>(rhs); }
97 template <typename T, typename U> bool operator==(U lhs, const Modular<T>& rhs) {
    return Modular<T>(lhs) == rhs; }
98
99 template <typename T> bool operator!=(const Modular<T>& lhs, const Modular<T>& rhs) {
    return !(lhs == rhs); }

```

```

100 template <typename T, typename U> bool operator!=(const Modular<T>& lhs, U rhs) {
    return !(lhs == rhs); }
101 template <typename T, typename U> bool operator!=(U lhs, const Modular<T>& rhs) {
    return !(lhs == rhs); }
102
103 template <typename T> bool operator<(const Modular<T>& lhs, const Modular<T>& rhs) {
    return lhs.value < rhs.value; }
104
105 template <typename T> Modular<T> operator+(const Modular<T>& lhs, const Modular<T>& rhs
    ) { return Modular<T>(lhs) += rhs; }
106 template <typename T, typename U> Modular<T> operator+(const Modular<T>& lhs, U rhs) {
    return Modular<T>(lhs) += rhs; }
107 template <typename T, typename U> Modular<T> operator+(U lhs, const Modular<T>& rhs) {
    return Modular<T>(lhs) += rhs; }
108
109 template <typename T> Modular<T> operator-(const Modular<T>& lhs, const Modular<T>& rhs
    ) { return Modular<T>(lhs) -= rhs; }
110 template <typename T, typename U> Modular<T> operator-(const Modular<T>& lhs, U rhs) {
    return Modular<T>(lhs) -= rhs; }
111 template <typename T, typename U> Modular<T> operator-(U lhs, const Modular<T>& rhs) {
    return Modular<T>(lhs) -= rhs; }
112
113 template <typename T> Modular<T> operator*(const Modular<T>& lhs, const Modular<T>& rhs
    ) { return Modular<T>(lhs) *= rhs; }
114 template <typename T, typename U> Modular<T> operator*(const Modular<T>& lhs, U rhs) {
    return Modular<T>(lhs) *= rhs; }
115 template <typename T, typename U> Modular<T> operator*(U lhs, const Modular<T>& rhs) {
    return Modular<T>(lhs) *= rhs; }
116
117 template <typename T> Modular<T> operator/(const Modular<T>& lhs, const Modular<T>& rhs
    ) { return Modular<T>(lhs) /= rhs; }
118 template <typename T, typename U> Modular<T> operator/(const Modular<T>& lhs, U rhs) {
    return Modular<T>(lhs) /= rhs; }
119 template <typename T, typename U> Modular<T> operator/(U lhs, const Modular<T>& rhs) {
    return Modular<T>(lhs) /= rhs; }
120
121 template<typename T, typename U>
122 Modular<T> power(const Modular<T>& a, const U& b) {
123     assert(b >= 0);
124     Modular<T> x = a, res = 1;
125     U p = b;
126     while (p > 0) {
127         if (p & 1) res *= x;
128         x *= x;
129         p >>= 1;
130     }
131     return res;
132 }
133
134 template <typename T>
135 bool IsZero(const Modular<T>& number) {
136     return number() == 0;
137 }
138
139 template <typename T>
140 std::string to_string(const Modular<T>& number) {
141     return to_string(number());
142 }
143

```

```

144 // U == std::ostream? but done this way because of fastoutput
145 template <typename U, typename T>
146 U& operator<<(U& stream, const Modular<T>& number) {
147     return stream << number();
148 }
149
150 // U == std::istream? but done this way because of fastinput
151 template <typename U, typename T>
152 U& operator>>(U& stream, Modular<T>& number) {
153     typename std::common_type<typename Modular<T>::Type, long long>::type x;
154     stream >> x;
155     number.value = Modular<T>::normalize(x);
156     return stream;
157 }
158
159 /*
160 using ModType = int;
161
162 struct VarMod { static ModType value; };
163 ModType VarMod::value;
164 ModType& md = VarMod::value;
165 using Mint = Modular<VarMod>;
166 */
167
168 constexpr int md = 998244353;
169 using Mint = Modular<std::integral_constant<std::decay<decltype(md)>::type, md>>;
170
171 std::vector<Mint> fact(1, 1);
172 std::vector<Mint> inv_fact(1, 1);
173
174 /*Mint C(int n, int k) {
175     if (k < 0 || k > n) {
176         return 0;
177     }
178     while ((int) fact.size() < n + 1) {
179         fact.push_back(fact.back() * (int) fact.size());
180         inv_fact.push_back(1 / fact.back());
181     }
182     return fact[n] * inv_fact[k] * inv_fact[n - k];
183 }*/

```

6.2 Z

```

1  constexpr int P = 998244353;
2  using i64 = long long;
3  // assume -P <= x < 2P
4  int norm(int x) {
5      if (x < 0) {
6          x += P;
7      }
8      if (x >= P) {
9          x -= P;
10     }
11     return x;
12 }
13 template<class T>
14 T power(T a, i64 b) {
15     T res = 1;

```

```

16     for (; b; b /= 2, a *= a) {
17         if (b % 2) {
18             res *= a;
19         }
20     }
21     return res;
22 }
23 struct Z {
24     int x;
25     Z(int x = 0) : x(norm(x)) {}
26     Z(i64 x) : x(norm(x % P)) {}
27     int val() const {
28         return x;
29     }
30     Z operator-() const {
31         return Z(norm(P - x));
32     }
33     Z inv() const {
34         assert(x != 0);
35         return power(*this, P - 2);
36     }
37     Z &operator*=(const Z &rhs) {
38         x = i64(x) * rhs.x % P;
39         return *this;
40     }
41     Z &operator+=(const Z &rhs) {
42         x = norm(x + rhs.x);
43         return *this;
44     }
45     Z &operator--(const Z &rhs) {
46         x = norm(x - rhs.x);
47         return *this;
48     }
49     Z &operator/=(const Z &rhs) {
50         return *this *= rhs.inv();
51     }
52     friend Z operator*(const Z &lhs, const Z &rhs) {
53         Z res = lhs;
54         res *= rhs;
55         return res;
56     }
57     friend Z operator+(const Z &lhs, const Z &rhs) {
58         Z res = lhs;
59         res += rhs;
60         return res;
61     }
62     friend Z operator-(const Z &lhs, const Z &rhs) {
63         Z res = lhs;
64         res -= rhs;
65         return res;
66     }
67     friend Z operator/(const Z &lhs, const Z &rhs) {
68         Z res = lhs;
69         res /= rhs;
70         return res;
71     }
72     friend std::istream &operator>>(std::istream &is, Z &a) {
73         i64 v;
74         is >> v;

```

```

75     a = Z(v);
76     return is;
77 }
78 friend std::ostream &operator<<(std::ostream &os, const Z &a) {
79     return os << a.val();
80 }
81 };

```

6.3 exgcd

```

1  template <typename T>
2  std::array<T, 3> exgcd(T a, T b) {
3      if (b == 0) {
4          return {a, 1, 0};
5      }
6      auto [g, x, y] = exgcd(b, a % b);
7      return {g, y, x - a / b * y};
8      /*
9      auto [g, x, y] = exgcd<long long>(a, b);
10     assert(1LL * a * x + 1LL * b * y == g);
11     if (c % g != 0) {
12         std::cout << -1 << std::endl;
13         continue ;
14     }
15     */
16 }

```

6.4 factorizer

```

1  namespace factorizer {
2
3  template <typename T>
4  struct FactorizerVarMod { static T value; };
5  template <typename T>
6  T FactorizerVarMod<T>::value;
7
8  template <typename T>
9  bool IsPrime(T n, const std::vector<T>& bases) {
10     if (n < 2) {
11         return false;
12     }
13     std::vector<T> small_primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
14     for (const T& x : small_primes) {
15         if (n % x == 0) {
16             return n == x;
17         }
18     }
19     if (n < 31 * 31) {
20         return true;
21     }
22     int s = 0;
23     T d = n - 1;
24     while ((d & 1) == 0) {
25         d >>= 1;
26         s++;
27     }
28     FactorizerVarMod<T>::value = n;

```

```

29     for (const T& a : bases) {
30         if (a % n == 0) {
31             continue;
32         }
33         Modular<FactorizerVarMod<T>> cur = a;
34         cur = power(cur, d);
35         if (cur == 1) {
36             continue;
37         }
38         bool witness = true;
39         for (int r = 0; r < s; r++) {
40             if (cur == n - 1) {
41                 witness = false;
42                 break;
43             }
44             cur *= cur;
45         }
46         if (witness) {
47             return false;
48         }
49     }
50     return true;
51 }
52
53 bool IsPrime(int64_t n) {
54     return IsPrime(n, {2, 325, 9375, 28178, 450775, 9780504, 1795265022});
55 }
56
57 bool IsPrime(int32_t n) {
58     return IsPrime(n, {2, 7, 61});
59 }
60
61 // but if you really need uint64_t version...
62 /*
63 bool IsPrime(uint64_t n) {
64     if (n < 2) {
65         return false;
66     }
67     std::vector<uint32_t> small_primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
68     for (uint32_t x : small_primes) {
69         if (n == x) {
70             return true;
71         }
72         if (n % x == 0) {
73             return false;
74         }
75     }
76     if (n < 31 * 31) {
77         return true;
78     }
79     uint32_t s = __builtin_ctzll(n - 1);
80     uint64_t d = (n - 1) >> s;
81     function<bool(uint64_t)> witness = [&n, &s, &d](uint64_t a) {
82         uint64_t cur = 1, p = d;
83         while (p > 0) {
84             if (p & 1) {
85                 cur = (__uint128_t) cur * a % n;
86             }
87             a = (__uint128_t) a * a % n;

```

```

88         p >= 1;
89     }
90     if (cur == 1) {
91         return false;
92     }
93     for (uint32_t r = 0; r < s; r++) {
94         if (cur == n - 1) {
95             return false;
96         }
97         cur = (__uint128_t) cur * cur % n;
98     }
99     return true;
100 };
101 std::vector<uint64_t> bases_64bit = {2, 325, 9375, 28178, 450775, 9780504,
1795265022};
102 for (uint64_t a : bases_64bit) {
103     if (a % n == 0) {
104         return true;
105     }
106     if (witness(a)) {
107         return false;
108     }
109 }
110 return true;
111 }
112 */
113
114 std::vector<int> least = {0, 1};
115 std::vector<int> primes;
116 int precalculated = 1;
117
118 void RunLinearSieve(int n) {
119     n = std::max(n, 1);
120     least.assign(n + 1, 0);
121     primes.clear();
122     for (int i = 2; i <= n; i++) {
123         if (least[i] == 0) {
124             least[i] = i;
125             primes.push_back(i);
126         }
127         for (int x : primes) {
128             if (x > least[i] || i * x > n) {
129                 break;
130             }
131             least[i * x] = x;
132         }
133     }
134     precalculated = n;
135 }
136
137 void RunSlowSieve(int n) {
138     n = std::max(n, 1);
139     least.assign(n + 1, 0);
140     for (int i = 2; i * i <= n; i++) {
141         if (least[i] == 0) {
142             for (int j = i * i; j <= n; j += i) {
143                 if (least[j] == 0) {
144                     least[j] = i;
145                 }
146             }
147         }
148     }

```



```

146         }
147     }
148 }
149 primes.clear();
150 for (int i = 2; i <= n; i++) {
151     if (least[i] == 0) {
152         least[i] = i;
153         primes.push_back(i);
154     }
155 }
156 precalculated = n;
157 }
158
159 void RunSieve(int n) {
160     RunLinearSieve(n);
161 }
162
163 template <typename T>
164 std::vector<std::pair<T, int>> MergeFactors(const std::vector<std::pair<T, int>>& a,
165     const std::vector<std::pair<T, int>>& b) {
166     std::vector<std::pair<T, int>> c;
167     int i = 0;
168     int j = 0;
169     while (i < (int) a.size() || j < (int) b.size()) {
170         if (i < (int) a.size() && j < (int) b.size() && a[i].first == b[j].first) {
171             c.emplace_back(a[i].first, a[i].second + b[j].second);
172             ++i;
173             ++j;
174             continue;
175         }
176         if (j == (int) b.size() || (i < (int) a.size() && a[i].first < b[j].first)) {
177             c.push_back(a[i++]);
178         } else {
179             c.push_back(b[j++]);
180         }
181     }
182     return c;
183 }
184
185 template <typename T>
186 std::vector<std::pair<T, int>> RhoC(const T& n, const T& c) {
187     if (n <= 1) {
188         return {};
189     }
190     if ((n & 1) == 0) {
191         return MergeFactors({{2, 1}}, RhoC(n / 2, c));
192     }
193     if (IsPrime(n)) {
194         return {{n, 1}};
195     }
196     FactorizerVarMod<T>::value = n;
197     Modular<FactorizerVarMod<T>> x = 2;
198     Modular<FactorizerVarMod<T>> saved = 2;
199     T power = 1;
200     T lam = 1;
201     while (true) {
202         x = x * x + c;
203         T g = __gcd((x - saved), n);
204         if (g != 1) {

```

```

204         return MergeFactors(RhoC(g, c + 1), RhoC(n / g, c + 1));
205     }
206     if (power == lam) {
207         saved = x;
208         power <= 1;
209         lam = 0;
210     }
211     lam++;
212 }
213 return {};
214 }
215
216 template <typename T>
217 std::vector<std::pair<T, int>> Rho(const T& n) {
218     return RhoC(n, static_cast<T>(1));
219 }
220
221 template <typename T>
222 std::vector<std::pair<T, int>> Factorize(T x) {
223     if (x <= 1) {
224         return {};
225     }
226     if (x <= precalculated) {
227         std::vector<std::pair<T, int>> ret;
228         while (x > 1) {
229             if (!ret.empty() && ret.back().first == least[x]) {
230                 ret.back().second++;
231             } else {
232                 ret.emplace_back(least[x], 1);
233             }
234             x /= least[x];
235         }
236         return ret;
237     }
238     if (x <= static_cast<int64_t>(precalculated) * precalculated) {
239         std::vector<std::pair<T, int>> ret;
240         if (!IsPrime(x)) {
241             for (T i : primes) {
242                 T t = x / i;
243                 if (i > t) {
244                     break;
245                 }
246                 if (x == t * i) {
247                     int cnt = 0;
248                     while (x % i == 0) {
249                         x /= i;
250                         cnt++;
251                     }
252                     ret.emplace_back(i, cnt);
253                     if (IsPrime(x)) {
254                         break;
255                     }
256                 }
257             }
258         }
259         if (x > 1) {
260             ret.emplace_back(x, 1);
261         }
262         return ret;

```

```

263     }
264     return Rho(x);
265 }
266
267 template <typename T>
268 std::vector<T> BuildDivisorsFromFactors(const std::vector<std::pair<T, int>>& factors)
269 {
270     std::vector<T> divisors = {1};
271     for (auto& p : factors) {
272         int sz = (int) divisors.size();
273         for (int i = 0; i < sz; i++) {
274             T cur = divisors[i];
275             for (int j = 0; j < p.second; j++) {
276                 cur *= p.first;
277                 divisors.push_back(cur);
278             }
279         }
280     }
281     sort(divisors.begin(), divisors.end());
282     return divisors;
283 }
284 } // namespace factorizer

```

6.5 comb

```

1 class Comb {
2 public :
3     const int n;
4     std::vector<Z> fac, inv, ifac;
5     Comb(int n) : n(n), fac(n), inv(n), ifac(n) {
6         fac[0] = fac[1] = inv[0] = inv[1] = ifac[0] = ifac[1] = 1;
7         for(int i = 2; i < n; i++) {
8             fac[i] = fac[i - 1] * i;
9             inv[i] = (P - P / i) * inv[P % i];
10            ifac[i] = ifac[i - 1] * inv[i];
11        }
12    }
13    Z C(int n, int m) {
14        if(m < 0 || n < 0 || m > n) return 0;
15        return fac[n] * ifac[m] * ifac[n - m];
16    }
17    Z Lucas(long long m, long long n) { return n ? Lucas(m / P, n / P) * C(m % P, n % P) : 1; }
18 };

```

6.6 算术基本定理

```

1 ll get_Count(ll n) {
2     ll ans = 1;
3     for(int i = 2; i * i <= n; i++) {
4         if(n % i == 0) {
5             int a = 0;
6             while(n % i == 0) {
7                 a++;
8                 n /= i;
9             }

```

```

10         ans *= (a + 1);
11     }
12 }
13 if(n > 1) ans *= 2;
14 return ans;
15 }
16
17 ll get_Sum(ll n) {
18     ll ans = 1;
19     for(int i = 2; i * i <= n; i++) {
20         if(n % i == 0) {
21             ll a = 1;
22             while(n % i == 0) {
23                 n /= i;
24                 a *= i;
25             }
26             ans = ans * (a * i - 1) / (i - 1);
27         }
28     }
29     if(n > 1) ans *= (n + 1);
30     return ans;
31 }

```

6.7 筛 phi

```

1 int is_prime[N], prime[N], cnt, phi[N];
2 void makePhi() {
3     phi[1] = 1, cnt = 0;
4     for (int i = 2; i < N; i++) {
5         if (!is_prime[i]) prime[++cnt] = i, phi[i] = i - 1;
6         for (int j = 1; j <= cnt && i * prime[j] < N; j++) {
7             is_prime[i * prime[j]] = 1;
8             if (i % prime[j] == 0) {
9                 phi[i * prime[j]] = phi[i] * prime[j];
10                break;
11            }
12            else phi[i * prime[j]] = phi[i] * phi[prime[j]];
13        }
14    }
15 }

```

6.8 筛 mobius

```

1 const int N = 1e5 + 10;
2 bool is_prime[N];
3 int prime[N], mu[N], cnt;
4
5 void makeMobius() {
6     mu[1] = 1; is_prime[0] = is_prime[1] = true;
7     for(int i = 2; i < N; i++) {
8         if (!is_prime[i]) {
9             mu[i] = -1;
10            prime[++cnt] = i;
11        }
12        for (int j = 1; j <= cnt && i * prime[j] < N; j++) {
13            is_prime[i * prime[j]] = true;
14            if (i % prime[j] == 0) {

```

```

15         mu[i * prime[j]] = 0;
16         break;
17     }
18     mu[i * prime[j]] = -mu[i];
19 }
20 }
21 }

```

6.9 筛积性函数

```

1 //只需要计算f(p ^ k)即可
2 //其余的都可以通过积性函数的性质来计算
3
4 int vis[N], prime[N], num;
5 int f[N], low[N];
6
7 void makeF(int siz) { //f为积性函数
8     num = 0, low[1] = f[1] = 1;
9     for (int i = 2; i <= siz; i++) {
10         if (!vis[i]) prime[++num] = i, low[i] = i, f[i] = ...; //这里是f(p)的答案
11         for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
12             vis[i * prime[j]] = 1;
13             if (i % prime[j] == 0) {
14                 low[i * prime[j]] = low[i] * prime[j];
15                 if (low[i] == i) { //i = prime[j] ^ k
16                     //只需要这里算一下
17                     //考虑 p ^ 1, p ^ 2, p ^ 3...
18                 }
19                 else f[i * prime[j]] = 1ll * f[i / low[i]] * f[prime[j] * low[i]] % mod;
20             }
21             break;
22             low[i * prime[j]] = prime[j];
23             f[i * prime[j]] = 1ll * f[i] * f[prime[j]] % mod;
24         }
25     }
26 }

```

6.10 欧拉函数

```

1 // 求解单个正整数的欧拉函数
2 int Get_phi(int n) {
3     int ans = n;
4     for (int i = 2; i * i <= n; i++) {
5         if (n % i == 0) {
6             ans = ans - ans / i;
7             while (n % i == 0)
8                 n /= i;
9         }
10    }
11    if (n > 1)
12        ans = ans - ans / n;
13    return ans;
14 }
15
16 // 埃拉托斯特尼筛求欧拉函数
17 int phi[10005];

```

```

18
19 void Euler_sieve(int n) {
20     phi[1] = 1;
21     for(int i = 2; i <= n; i++) {
22         if(!phi[i]) {
23             for(int j = i; j <= n; j += i) {
24                 if(!phi[j])
25                     phi[j] = j;
26                 phi[j] = phi[j] / i * (i - 1);
27             }
28         }
29     }
30 }
31
32 // 欧拉筛求欧拉函数
33
34 const int N = 5e6 + 10;
35 bool is_prime[N];
36 int prime[N], phi[N], tot;
37
38 void Euler() {
39     phi[1] = 1; is_prime[1] = true;
40     for(int i = 2; i < N; i++){
41         if(!is_prime[i]) {
42             phi[i] = i - 1;
43             prime[++tot] = i;
44         }
45         for(int j = 1; j <= tot && i * prime[j] < N; j++){
46             is_prime[i * prime[j]] = true;
47             if(i % prime[j] == 0) {
48                 phi[i * prime[j]] = phi[i] * (prime[j] - 1);
49             }
50             else{
51                 phi[i * prime[j]] = phi[i] * prime[j];
52                 break;
53             }
54         }
55     }
56 }

```

6.11 原根

```

1 typedef long long ll;
2
3 vector<ll> YG;
4 ll p, n; // p是模数, n是p的欧拉函数值
5
6 ll gcd(ll a, ll b) {
7     return b ? gcd(b, a % b) : a;
8 }
9
10 ll quick_pow(ll a, ll b, ll p) ;
11
12 ll phi(ll n) {
13     ll ans = n;
14     for(int i = 2; i * i <= n; i++) {
15         if(n % i == 0) {
16             ans = ans - ans / i;

```

```

17         while(n % i == 0) {
18             n /= i;
19         }
20     }
21 }
22 if(n > 1)
23     ans = ans - ans / n;
24 return ans;
25 }
26
27 vector<ll> PrimeFac(ll n) { // n的素因子
28     vector<ll> fac;
29     fac.clear();
30     for(ll i = 2; i * i <= n; i++) {
31         if(n % i == 0) {
32             fac.push_back(i);
33             while(n % i == 0)
34                 n /= i;
35         }
36     }
37     if(n > 1)
38         fac.push_back(n);
39     return fac;
40 }
41
42 bool is_Protogen(ll p) { // 原根p = 2、4、p^k、2*p^k(p为非2的质数, k为任意数)
43     if(p == 2 || p == 4) return true;
44     if(p <= 1 || p % 4 == 0) return false;
45     ll num = 0;
46     while(p % 2 == 0) // 2的倍数先筛掉
47         p /= 2;
48     for(int i = 3; i * i <= p; i++) { // p只能是一个非2的素数的倍数构成, 否则没有原根
49         if(p % i == 0) {
50             num++;
51             while(p % i == 0)
52                 p /= i;
53         }
54     }
55     if(p > 1) num++;
56     if(num == 1) return true;
57     return false;
58 }
59
60 ll Protogen(ll p) {
61     if(!is_Protogen(p)) // 先判断是否存在原根
62         return -1;
63     n = phi(p);
64     if(p == 2) return 1;
65     if(p == 3) return 2;
66     if(p == 4) return 3;
67     vector<ll> fac = PrimeFac(n); // f(p)的素因子
68     for(int i = 2; i <= p - 1; i++) {
69         if(gcd(i, p) != 1) // n是模p的欧拉函数值, i要和n互质
70             continue;
71         bool flag = true;
72         for(ll j = 0; j < fac.size(); j++) {
73             if(quick_pow(i, n / fac[j], p) == 1)
74                 flag = 0;
75         }

```

```

76         if(flag) // i就是原根
77             return i;
78     }
79     return -1;
80 }
81
82 void Sum_Protogen(ll k) { // 找出n的所有原根
83     YG.push_back(k);
84     for(int i = 2; i < n; i++) {
85         if(gcd(i, n) == 1) // i要与f(n)互质
86             YG.push_back(quick_pow(k, i, p));
87     }
88 }
89
90 int main() {
91     cin >> p;
92     ll k = Protogen(p); // p的原根
93     cout << k << endl;
94     Sum_Protogen(k);
95     for(int i = 0; i < YG.size(); i++) {
96         cout << YG[i] << " ";
97     }
98     cout << endl;
99     return 0;
100 }

```

6.12 原根表

1	mod					原根
2	$r \cdot 2^{k+1}$	r	k	g		
3	3	1	1	2		
4	5	1	2	2		
5	17	1	4	3		
6	97	3	5	5		
7	193	3	6	5		
8	257	1	8	3		
9	7681	15	9	17		
10	12289	3	12	11		
11	40961	5	13	3		
12	65537	1	16	3		
13	786433	3	18	10		
14	5767169	11	19	3		
15	7340033	7	20	3		
16	23068673	11	21	3		
17	104857601	25	22	3		
18	167772161	5	25	3		
19	469762049	7	26	3		
20	998244353	119	23	3	这个数常用	
21	1004535809	479	21	3	加起来不会爆int	
22	2013265921	15	27	31		
23	2281701377	17	27	3	这个数平方刚好不会爆ll	
24	3221225473	3	30	5		
25	75161927681	35	31	3		
26	77309411329	9	33	7		
27	206158430209	3	36	22		
28	2061584302081	15	37	7		
29	2748779069441	5	39	3		
30	6597069766657	3	41	5		


```

31 39582418599937 9 42 5
32 79164837199873 9 43 5
33 263882790666241 15 44 7
34 1231453023109121 35 45 3
35 1337006139375617 19 46 3
36 3799912185593857 27 47 5
37 4222124650659841 15 48 19
38 7881299347898369 7 50 6
39 31525197391593473 7 52 3
40 180143985094819841 5 55 6
41 1945555039024054273 27 56 5
42 4179340454199820289 29 57 3

```

6.13 阶乘逆元

```

1  const int N = 5e6 + 10;
2  const ll mod = 1e9 + 7;
3
4  ll F[N], invn[N], invF[N];
5
6  void Init() {
7      F[0] = F[1] = invn[0] = invn[1] = invF[0] = invF[1] = 1;
8      for(int i = 2; i < N; i++){
9          F[i] = F[i - 1] * i % mod;
10         invn[i] = (mod - mod / i) * invn[mod % i] % mod;
11         invF[i] = invF[i - 1] * invn[i] % mod;
12     }
13 }

```

6.14 常见积性函数

```

1  //phi
2  //phi[i * j] = phi[i] * phi[j] * gcd(i, j) / phi[gcd(i, j)]
3
4  // d
5  // d[i * j] = \sum_{x|i} * \sum_{y|j} * [gcd(x, y) = 1]

```

6.15 Miller-Rabin

```

1  // 二次探测定理: 对素数p, 满足x^2≡1(modp)的小于p的正整数解x只有1或p-1.
2
3  #include <bits/stdc++.h>
4  using namespace std;
5  typedef long long ll;
6  const int N = 1e5 + 7;
7  const int times = 10;
8
9  ll ksc(ll a, ll b, ll mod) {
10     ll ans = 0;
11     while(b > 0) {
12         if(b & 1) {
13             ans = (ans + a) % mod;
14         }
15         a = (a << 1) % mod;
16         b >>= 1;
17     }

```

```

18     return ans;
19 }
20
21 ll quick_pow(ll a, ll b, ll mod) {
22     ll ans = 1, base = a;
23     while(b != 0) {
24         if(b & 1) {
25             ans = ans * base % mod;
26         }
27         base = base * base % mod;
28         b >>= 1;
29     }
30     return ans;
31 }
32
33 bool Miller_Pabin(ll n)//Miller测试的主体结构
34 {
35     if(n < 2) return false;
36     if(n == 2) return true;
37     if(n & 1 == 0) return false;//对于偶数的优化
38     ll k = 0, u = n - 1;//p为Miller测试的k, u为Miller测试的m
39
40     while(u & 1 == 0){ // 把x拆成u*2^k
41         u >>= 1;
42         k++;
43     }
44     srand(time(NULL));
45
46     ll x, pre; // pre为上次探测的x的值
47
48     for(int i = 1; i <= times; i++) {
49         x = rand() % (n - 1) + 1;
50         x = quick_pow(x, u, n); // 先求出x^u(mod n)
51         pre = x;
52         for(int j = 1; j <= k; j++) {
53             x = ksc(x, x, n);
54             if(x == 1 && pre != 1 && pre != n - 1)
55                 return false;
56             pre = x;
57         }
58         if(x != -1)
59             return false;
60     }
61     return true;
62 }
63
64 int main() {
65     ll n; cin >> n;
66     cout << (Miller_Pabin(n) ? "Prime" : "Not a Prime") << endl;
67 }

```

6.16 quadraticResidue

```

1 typedef long long ll;
2
3 typedef struct{
4     ll x, y; // 把求出来的w作为虚部, 则为a + bw
5 }num;

```

```

6
7 ll quick_pow(ll a, ll b, ll p) {
8     ll ans = 1;
9     while(b) {
10         if(b & 1) ans = ans * a % p;
11         a = a * a % p;
12         b >>= 1;
13     }
14     return ans % p;
15 }
16
17
18 num num_mul(num a, num b, ll w, ll p) { // 复数乘法
19     num ans = {0, 0};
20     ans.x = (a.x * b.x % p + a.y * b.y % p * w % p + p) % p;
21     ans.y = (a.x * b.y % p + a.y * b.x % p + p) % p;
22     return ans;
23 }
24
25 ll num_pow(num a, ll b, ll w, ll p) { // 复数快速幂
26     num ans = {1, 0};
27     while(b) {
28         if(b & 1)
29             ans = num_mul(ans, a, w, p);
30         a = num_mul(a, a, w, p);
31         b >>= 1;
32     }
33     return ans.x % p;
34 }
35
36 ll legendre(ll a, ll p) { // 勒让德符号 = {1, -1, 0}
37     return quick_pow(a, (p - 1) >> 1, p);
38 }
39
40 ll Cipolla(ll n, ll p) { // 输入a和p, 是否存在x使得x^2 = a (mod p), 存在二次剩余返回x, 存在二次
    // 非剩余返回-1      注意: p是奇质数
41     n %= p;
42     if(n == 0)
43         return 0;
44     if(p == 2)
45         return 1;
46     if(legendre(n, p) + 1 == p) // 二次非剩余
47         return -1;
48
49     ll a, w;
50
51     while(true) { // 找出a, 求出w, 随机成功的概率是50%, 所以数学期望是2
52         a = rand() % p;
53         w = ((a * a - n) % p + p) % p;
54         if(legendre(w, p) + 1 == p) // 找到w, 非二次剩余条件
55             break;
56     }
57     num x = {a, 1};
58     return num_pow(x, (p + 1) >> 1, w, p) % p; // 计算x, 一个解是x, 另一个解是p-x, 这里的w其实
    // 要开方, 但是由拉格朗日定理可知虚部为0, 所以最终答案就是对x的实部用快速幂求解
59 }
60
61 int main()
62 {

```

```

63     ll n, p;
64     cin >> n >> p;
65     srand((unsigned)time(NULL));
66     cout << Cipolla(n, p) << endl;
67     return 0;
68 }

```

6.17 bags

```

1  // 求解  $a^x = b \pmod c$ , 要求  $\gcd(a, c) = 1$ , 不要求  $p$  为素数,  $x$  的范围是  $0 \leq x \leq p-1$ 
2
3  template <typename T>
4  struct Hash {
5      int n;
6      int cnt;
7      std::vector<int> head, next, hash, id;
8      Hash(int _n) : n(_n), head(_n, -1), next(_n), id(_n), hash(_n), cnt(0) {}
9
10     void insert(T x, T y) {
11         T k = x % n;
12         hash[cnt] = x;
13         id[cnt] = y;
14         next[cnt] = head[k];
15         head[k] = cnt++;
16     }
17
18     T query(T x) {
19         for(int i = head[x % n]; i != -1 ; i = next[i]){
20             if(hash[i] == x)
21                 return id[i];
22         }
23         return -1;
24     }
25 };
26
27 template <typename T>
28 T bsgs(T& a, T& b, T& c) {
29     a %= c; b %= c;
30     int cnt = 1;
31     if (b == 1) return 0;
32     Hash<long long> hs(100005);
33     T m = std::ceil(sqrt((double) c));
34     T x = 1, p = 1;
35     for (T j = 0; j < m; j++, p = p * a % c) {
36         hs.insert(p * b % c, j);
37     }
38     for (T i = 1, j; i <= m; i++) {
39         x = x * p % c;
40         if ((j = hs.query(x)) != -1) {
41             return i * m - j;
42         }
43     }
44     return -1;
45 }

```

6.18 EX-BSGS

```

1 // a和c不互质
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 typedef long long ll;
6
7 ll gcd(ll a, ll b) {
8     return b ? gcd(b, a % b) : a;
9 }
10
11 template <typename T>
12 T ex_bsgs(T& a, T& b, T& c) {
13     a %= c; b %= c;
14     if (b == 1) return 0;
15     T k = 0, tmp = 1, d;
16     while (true) {
17         d = __gcd(a, c);
18         if (d == 1) {
19             break ;
20         }
21         if (b % d) {
22             return -1;
23         }
24         b /= d; c /= d;
25         tmp = tmp * (a / d) % c;
26         k++;
27         if (tmp == b) {
28             return k;
29         }
30     }
31     std::unordered_map<T, T> mp;
32     T m = std::ceil(sqrt((double) c));
33     T x = 1, p = 1;
34     for(T j = 0; j < m; j++, p = p * a % c) {
35         mp[p * b % c] = j;
36     }
37     x = tmp % c;
38     for(T i = 1; i <= m; i++) { // 枚举a^im
39         x = x * p % c;
40         if(mp[x]) {
41             return k + i * m - mp[x];
42         }
43     }
44     return -1;
45 }

```

6.19 CRT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 typedef long long ll;
5 ll m[10005], a[10005], n; //a是余数, b是模数
6
7 void exgcd(ll a, ll b, ll &x, ll &y) {
8     if(b == 0) {
9         x = 1;
10        y = 0;

```

```

11     return;
12 }
13 exgcd(b, a % b , y , x);
14 y -= a / b * x;
15 }
16
17 ll INV(ll a, ll mod) {
18     ll x, y;
19     exgcd(a , mod , x , y);
20     x = (x % mod + mod) % mod;
21     return x;
22 }
23
24 ll CRT() {
25     ll ans = 0, M = 1;
26     for(ll i = 1; i <= n; i++) {
27         M *= m[i]; // M是所有除数的乘积
28     }
29     for(ll i = 1; i <= n; i++) {
30         ll mm = M / m[i];
31         ll ret = INV(mm , m[i]); // 先求逆元
32         ans = (ans + a[i] * mm % M * ret % M) % M;
33     /*
34     ans = (ans + quick_mul(quick_mul(m , ret , M) , b[i] , M)) % M;
35     利用快速乘防止爆longlong
36     */
37     }
38     return (ans + M) % M;
39 }
40
41 int main() {
42     ll ans = 0;
43     scanf("%lld",&n);
44     for(ll i = 1; i <= n; i++) {
45         scanf("%lld%lld",&m[i],&a[i]);
46         a[i] = (a[i] % m[i] + m[i]) % m[i]; // 防止b[i]为负
47     }
48     ans = CRT(); // 精髓
49     printf("%lld",ans);
50     return 0;
51 }

```

6.20 EX-CRT

```

1  #include <iostream>
2
3  using namespace std;
4
5  typedef long long ll;
6
7  ll c[100005], m[100005], n;
8
9  ll ksc(ll a, ll b, ll mod) {
10     ll ans = 0;
11     while(b > 0) {
12         if(b & 1) {
13             ans = (ans + a) % mod;
14         }

```

```

15     a = (a << 1) % mod;
16     b >>= 1;
17 }
18 return ans;
19 }
20
21 ll gcd(ll a, ll b) {
22     return b ? gcd(b, a % b) : a;
23 }
24
25 ll ex_gcd(ll a, ll b, ll &x, ll &y) {
26     ll res, t;
27     if(!b) {
28         x = 1;
29         y = 0;
30         return a;
31     }
32     res = ex_gcd(b, a % b, x, y);
33     t = x;
34     x = y;
35     y = t - (a / b) * y;
36     return res;
37 }
38
39 ll INV(ll a, ll mod) {
40     ll x, y;
41     ll d = ex_gcd(a, mod, x, y);
42     return d ? (x % mod + mod) % mod : -1;
43 }
44
45 ll EX_CRT() {
46     ll x, y;
47     ll ans = c[1];
48     ll M = m[1];
49     for(int i = 2; i <= n; i++) {
50         ll C = ((c[i] - ans) % m[i] + m[i]) % m[i];
51         ll T = ex_gcd(M, m[i], x, y);
52         if((c[i] - ans) % T)
53             return -1;
54         x = ksc(x, C / T, m[i] / T);
55         ans += M * x;
56         M *= (m[i] / T);
57         ans = (ans % M + M) % M;
58     }
59     return ans;
60 }
61
62 /*
63 ll EX_CRT() // 便于理解
64 {
65     for(int i = 2; i <= n; i++)
66     {
67         ll M1 = m[i - 1], M2 = m[i], C1 = c[i - 1], C2 = c[i];
68         ll T = gcd(M1, M2); // gcd(M1, M2)
69         if((C2 - C1) % T) // 无解
70             return -1;
71         m[i] = (M1 * M2) / T; // 合并后新同余方程的模
72         c[i] = INV(M1 / T, M2 / T) * (C2 - C1) / T % (M2 / T) * M1 + C1; // 可快速乘优化
73         c[i] = (c[i] % m[i] + m[i]) % m[i]; // 合并后新同余方程的余

```

```

74     }
75     return c[n];
76 }
77 */
78
79 int main()
80 {
81     cin >> n;
82     for(int i = 1; i <= n; i++)
83         cin >> c[i] >> m[i];
84     cout << EX_CRT() << endl;
85 }

```

6.21 EX-Lucas

```

1  // p不为质数，利用中国剩余定理结合求解
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  typedef long long ll;
6
7  const int N = 1e5 + 10;
8
9  ll quick_pow(ll a, ll b, ll P) {
10     ll ans = 1;
11     while(b) {
12         if(b & 1)
13             ans = ans * a % P;
14         a = a * a % P;
15         b >>= 1;
16     }
17     return ans % P;
18 }
19
20 ll ex_gcd(ll a, ll b, ll &x, ll &y) {
21     ll res, t;
22     if(!b) {
23         x = 1;
24         y = 0;
25         return a;
26     }
27     res = ex_gcd(b, a % b, x, y);
28     t = x;
29     x = y;
30     y = t - (a / b) * y;
31     return res;
32 }
33
34 ll INV(ll a, ll mod) {
35     ll x, y;
36     ll d = ex_gcd(a, mod, x, y);
37     return d ? (x % mod + mod) % mod : -1;
38 }
39
40 ll fac(ll n, ll P, ll pk) { // 阶乘除去质因子后模质数幂 (n / p^a) % pk
41     if(!n) return 1;
42     ll ans = 1;
43     for(int i = 1; i < pk; i++) { // 第三部分: n!与p互质的乘积

```



```

44     if(i % P)
45         ans = ans * i % pk;
46     }
47     ans = quick_pow(ans, n / pk, pk) % pk; // 第三部分: n!与p互质的乘积,ans循环的次数为n/pk
48     for(int i = 1; i <= n % pk; i++) { // 第四部分: 循环过后n!剩下的部分
49         if(i % P) ans = ans * i % pk;
50     }
51     return ans * fac(n / P, P, pk) % pk; // 第一部分, p的幂, 个数为n/p; 第二部分: (n/p)!
52 }
53
54 ll C(ll m, ll n, ll P, ll pk) { // 组合数模质数幂
55     if(n < 0 || m < 0 || n > m) return 0;
56     ll f1 = fac(m, P, pk), f2 = fac(n, P, pk), f3 = fac(m - n, P, pk), tmp = 0; // tmp
    = pk1 - pk2 - pk3
57     for(ll i = m; i ; i /= P) tmp += i / P;
58     for(ll i = n; i ; i /= P) tmp -= i / P;
59     for(ll i = m - n; i ; i /= P) tmp -= i / P;
60     return f1 * INV(f2, pk) % pk * INV(f3, pk) * quick_pow(P, tmp, pk) % pk;
61 }
62
63 ll p[N], a[N];
64 int cnt;
65
66 ll CRT() {
67     ll M = 1, ans = 0;
68     for(int i = 1; i <= cnt; i++) M *= p[i];
69     for(int i = 1; i <= cnt; i++) {
70         ll m = M / p[i];
71         ans = (ans + a[i] * m % M * INV(m, p[i]) % M) % M;
72     }
73     return (ans % M + M) % M;
74 }
75
76 ll EX_Lucas(ll m, ll n, ll P) {
77     for(int i = 2; i * i <= P; i++) {
78         if(P % i == 0) {
79             ll tmp = 1;
80             while(P % i == 0) {
81                 tmp *= i;
82                 P /= i;
83             }
84             p[++cnt] = tmp;
85             a[cnt] = C(m, n, i, tmp);
86         }
87     }
88     if(P > 1) {
89         p[++cnt] = P;
90         a[cnt] = C(m, n, P, P);
91     }
92     return CRT();
93 }
94 int main() {
95     ll m, n, P;
96     cin >> m >> n >> P;
97     cnt = 0;
98     cout << EX_Lucas(m, n, P) << endl;
99 }

```

6.22 min25

```

1  typedef long long ll;
2
3  const int N = 1e5 + 10;
4
5
6  namespace Min25 {
7      int prime[N], id1[N], id2[N], flag[N], ncnt, m;
8
9      ll g[N], sum[N], a[N], T, n;
10
11     inline int ID(ll x) {
12         return x <= T ? id1[x] : id2[n / x];
13     }
14
15     inline ll calc(ll x) {
16         return x * (x + 1) / 2 - 1;
17     }
18
19     inline ll f(ll x) {
20         return x;
21     }
22
23     inline void init() {
24         ncnt = 0, m = 0;
25         T = sqrt(n + 0.5);
26         for (int i = 2; i <= T; i++) {
27             if (!flag[i]) prime[++ncnt] = i, sum[ncnt] = sum[ncnt - 1] + i;
28             for (int j = 1; j <= ncnt && i * prime[j] <= T; j++) {
29                 flag[i * prime[j]] = 1;
30                 if (i % prime[j] == 0) break;
31             }
32         }
33         for (ll l = 1; l <= n; l = n / (n / l) + 1) {
34             a[++m] = n / l;
35             if (a[m] <= T) id1[a[m]] = m; else id2[n / a[m]] = m;
36             g[m] = calc(a[m]);
37         }
38         for (int i = 1; i <= ncnt; i++)
39             for (int j = 1; j <= m && (ll)prime[i] * prime[i] <= a[j]; j++)
40                 g[j] = g[j] - (ll)prime[i] * (g[ID(a[j] / prime[i])] - sum[i - 1]);
41     }
42
43     inline ll Solve(ll x) {
44         if (x <= 1) return x;
45         return n = x, init(), g[ID(n)];
46     }
47
48 }

```

6.23 BM

```

1  typedef long long ll;
2  const ll mod = 1e9 + 7;
3
4  typedef vector<ll> VI;
5

```

```

6 ll quick_pow(ll a, ll b) ;
7
8 namespace linear_seq {
9     const ll N = 1e5 + 10;
10    ll res[N], base[N], _c[N], _md[N];
11
12    vector<ll> Md;
13    void mul(ll *a, ll *b, ll k) {
14        for (ll i = 0; i < 2 * k; i++)
15            _c[i] = 0;
16        for (ll i = 0; i < k; i++) {
17            if (a[i]) {
18                for (int j = 0; j < k; j++) {
19                    _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
20                }
21            }
22        }
23        for (ll i = 2 * k - 1; i >= k; i--) {
24            if (_c[i]) {
25                for (ll j = 0; j < Md.size(); j++) {
26                    _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j]]) % mod;
27                }
28            }
29        }
30        for (ll i = 0; i < k; i++)
31            a[i] = _c[i];
32    }
33
34    ll solve(ll n, VI a, VI b) {
35        // a 系数 b 初值 b[n + 1] = a[0] * b[n] + ...
36        // cout << b.size() << endl;
37        ll ans = 0, pnt = 0;
38        ll k = a.size();
39        assert(a.size() == b.size());
40        for (ll i = 0; i < k; i++)
41            _md[k - i - 1] = -a[i];
42        _md[k] = 1;
43        Md.clear();
44        for (ll i = 0; i < k; i++) {
45            if (_md[i] != 0)
46                Md.push_back(i);
47        }
48        for (ll i = 0; i < k; i++)
49            res[i] = base[i] = 0;
50        res[0] = 1;
51        while ((1ll << pnt) <= n)
52            pnt++;
53        for (ll p = pnt; p >= 0; p--) {
54            mul(res, res, k);
55            if ((n >> p) & 1) {
56                for (ll i = k - 1; i >= 0; i--)
57                    res[i + 1] = res[i];
58                res[0] = 0;
59                for (ll i = 0; i < Md.size(); i++)
60                    res[Md[i]] = (res[Md[i]] - res[k] * _md[Md[i]]) % mod;
61            }
62        }
63        for (ll i = 0; i < k; i++)
64            ans = (ans + res[i] * b[i]) % mod;

```

```

65     return ans;
66 }
67
68 VI BM(VI s) {
69     VI C(1, 1), B(1, 1);
70     ll L = 0, m = 1, b = 1;
71     for (ll n = 0; n < s.size(); n++) {
72         ll d = 0;
73         for (ll i = 0; i < L + 1; i++)
74             d = (d + (ll)C[i] * s[n - i]) % mod;
75         if (d == 0)
76             m++;
77         else if (2 * L <= n) {
78             VI T = C;
79             ll c = mod - d * quick_pow(b, mod - 2) % mod;
80             while (C.size() < B.size() + m)
81                 C.push_back(0);
82             for (int i = 0; i < B.size(); i++)
83                 C[i + m] = (C[i + m] + c * B[i]) % mod;
84             L = n + 1 - L;
85             B = T;
86             b = d;
87             m = 1;
88         }
89         else {
90             ll c = mod - d * quick_pow(b, mod - 2) % mod;
91             while (C.size() < B.size() + m)
92                 C.push_back(0);
93             for (ll i = 0; i < B.size(); i++)
94                 C[i + m] = (C[i + m] + c * B[i]) % mod;
95             m++;
96         }
97     }
98     return C;
99 }
100
101 ll gao(VI a, ll n) {
102     VI c = BM(a);
103     c.erase(c.begin());
104     for (ll i = 0; i < c.size(); i++)
105         c[i] = (mod - c[i]) % mod;
106     return solve(n, c, VI(a.begin(), a.begin() + c.size()));
107 }
108 }
109
110 void solve() {
111     int n;
112     while (~scanf("%d", &n)) {
113         VI v = VI{1,2,4,7,13,24};
114         printf("%d\n", linear_seq::gao(v, n - 1));
115     }
116 }

```

6.24 duSieve

```

1 #include <bits/stdc++.h>
2
3 using namespace std;

```

```

4
5 typedef long long ll;
6 const int N = 1e6 + 10;
7
8 unordered_map<int, ll> smu, sphi;
9 bool isPrime[N];
10 int prime[N], num;
11 ll mu[N], phi[N];
12
13 void makeMobiusAndEuler(int siz) {
14     mu[1] = phi[1] = 1;
15     for (int i = 2; i <= siz; i++) {
16         if (!isPrime[i]) prime[++num] = i, mu[i] = -1, phi[i] = i - 1;
17         for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
18             isPrime[i * prime[j]] = 1;
19             if (i % prime[j] == 0) {
20                 mu[i * prime[j]] = 0;
21                 phi[i * prime[j]] = phi[i] * prime[j];
22                 break;
23             }
24             else {
25                 phi[i * prime[j]] = phi[prime[j]] * phi[i];
26                 mu[i * prime[j]] = -mu[i];
27             }
28         }
29     }
30     for (int i = 1; i <= siz; i++) mu[i] += mu[i - 1], phi[i] += phi[i - 1];
31 }
32
33 ll getSmu(int n) {
34     if (n < N) return mu[n];
35     if (smu[n]) return smu[n];
36     ll res = 1;
37     for (unsigned int l = 2, r = 0; l <= n; l = r + 1) {
38         r = n / (n / l);
39         res -= 1ll * (r - l + 1) * getSmu(n / l);
40     }
41     return smu[n] = res;
42 }
43
44 ll getSphi(int n) {
45     if (n < N) return phi[n];
46     if (sphi[n]) return sphi[n];
47     ll res = 1ll * n * (n + 1) / 2;
48     for (unsigned int l = 2, r = 0; l <= n; l = r + 1) {
49         r = n / (n / l);
50         res -= 1ll * (r - l + 1) * getSphi(n / l);
51     }
52     return sphi[n] = res;
53 }

```

6.25 反演相关

```

1 /*
2 莫比乌斯反演
3  $g[n] = \sum_{d|n} f[d]$ 
4  $f[d] = \sum_{d|n} g[d] * \mu[n/d]$ 
5 二项式反演

```

```

6  g[n] = \sum_{i = 1}^n C(n, i) * f[i]
7  f[n] = \sum_{i = 1}^n C(n, i) * g[i] * (-1)^{n - i}
8  子集反演
9  f(S) = \sum_{T \in S} g(T)
10 g(S) = \sum_{T \in S} f(T) * (-1)^{|S| - |T|}
11 */

```

6.26 simpson

```

1  // 求一个函数在一个区间上的数值积分
2
3  double f(double x) { // 题目中要求的辛普森积分函数，这里简单写一下f(x)=x*x
4      return x * x;
5  }
6
7  double Simpson(double a, double b) {
8      double mid = (a + b) / 2.0;
9      return (b - a) * (f(a) + f(b) + 4.0 * f(mid)) / 6.0;
10 }
11
12 double DFS(double a, double b, double eps)
13 {
14     double mid = (a + b) / 2.0;
15     double SA = Simpson(a, mid), SM = Simpson(a, b), SB = Simpson(mid, b);
16     if(fabs(SA + SB - SM) <= 15.0 * eps)
17         return SA + SB + (SA + SB - SM) / 15.0;
18     return DFS(a, mid, eps / 2.0) + DFS(mid, b, eps / 2.0);
19 }
20
21 // 求一个函数在0~无穷的上数值积分，若收敛输出答案，若发散输出orz

```

6.27 Bell

```

1
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  typedef long long ll;
6
7  const int N = 20;
8
9  ll S2[N][N];
10 ll B[N];
11
12 void Stirling2() {
13     S2[0][0] = 1;
14
15     for(int i = 1; i < N; i++) {
16         for(int j = 1; j <= i; j++) {
17             S2[i][j] = S2[i - 1][j - 1] + j * S2[i - 1][j];
18         }
19     }
20 }
21
22
23 // 根据第二类斯特林数
24

```

```

25 void Bell1() {
26     for(int i = 0; i < N; i++) {
27         for(int j = 0; j <= i; j++) {
28             B[i] += S2[i][j];
29         }
30     }
31 }
32
33 // Bell三角形递推
34 ll b[N][N];
35
36 void Bell2() {
37     b[1][1] = 1;
38     for(int i = 2; i < N; i++) {
39         b[i][1] = b[i - 1][i - 1];
40
41         for(int j = 2; j < N; j++) {
42             b[i][j] = b[i][j - 1] + b[i - 1][j - 1];
43         }
44     }
45 }
46
47 // 自身递推
48 ll fac[N];
49
50 ll C(ll m, ll n) {
51     return fac[m] / (fac[n] * fac[m - n]);
52 }
53
54 void Bell3() {
55     fac[1] = 1;
56     for(int i = 2; i < N; i++)
57         fac[i] = fac[i - 1] * i;
58
59     B[0] = 1;
60
61     for(int i = 1; i < N; i++) {
62         for(int k = 0; k <= i; k++) {
63             B[i] += C(i, k) * B[k];
64         }
65     }
66 }

```

6.28 Catalan

```

1
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 typedef long long ll;
6
7 const int N = 1e5 + 10;
8
9 int C[N];

```

```

10
11 // 线性递推
12
13 void Calc1() {
14     C[0] = 1;
15     for(int i = 1; i < N; i++) {
16         C[i] = C[i - 1] * (4 * i - 2) / (i + 1);
17     }
18 }
19
20 // 组合数求解
21
22 int f[N];
23
24 void fac() {
25     f[0] = 1;
26     for(int i = 1; i < N; i++) {
27         f[i] = f[i - 1] * i;
28     }
29 }
30
31 void Calc2(int n) {
32     C[n] = f[2 * n] / f[n + 1];
33 }
34
35 // 多项式求解
36
37 void Calc3(int n) {
38     if(n == 1)
39         C[n] = 1;
40
41     for(int i = 1; i <= n; i++) {
42         C[n] += C[n - i] * C[i - 1];
43     }
44 }

```

6.29 Lucas

```

1 // mod一定为质数
2
3 namespace Comb {
4     ll mod;
5     const int N = 1e6 + 10;
6     ll F[N], invF[N], inv[N];
7
8     void init() {
9         F[0] = F[1] = invF[0] = invF[1] = inv[0] = inv[1] = 1;
10        for (int i = 2; i < N; i++) {
11            F[i] = F[i - 1] * i % mod;
12            inv[i] = (mod - mod / i) * inv[mod % i] % mod;
13            invF[i] = invF[i - 1] * inv[i] % mod;
14        }
15    }
16
17    ll C(ll m, ll n) {
18        if (m < 0 || n < 0 || n > m) return 0;
19        ll ans = F[m];
20        ans = ans * invF[n] % mod;

```



```

21     ans = ans * invF[m - n] % mod;
22     return ans;
23 }
24
25 ll Lucas(ll m, ll n) {
26     return n ? Lucas(m / mod, n / mod) * C(m % mod, n % mod) % mod : 1;
27 }
28
29 }
30
31 // Comb::Lucas(m, n)

```

6.30 伯努利数

```

1 namespace BNL {
2     const int N = 1e7 + 10, M = 1e6 + 10;
3     struct Complex {
4         double x, y;
5         Complex(double a = 0, double b = 0): x(a), y(b) {}
6         Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y);
7     }
8         Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y);
9     }
10        Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y,
11            x * rhs.y + y * rhs.x); }
12        Complex conj() { return Complex(x, -y); }
13    } w[N];
14
15    int tr[N];
16    ll F[N], G[N];
17
18    ll quick_pow(ll a, ll b, ll p) {
19        ll ans = 1;
20        while(b) {
21            if(b & 1) ans = ans * a % p;
22            a = a * a % p;
23            b >>= 1;
24        }
25        return ans % p;
26    }
27
28    void FFT(Complex *A, int len) {
29        for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
30        for (int i = 2, lcy = len >> 1; i <= len; i <= 1, lcy >>= 1)
31            for (int j = 0; j < len; j += i) {
32                Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
33                for (int k = 0; k < i >> 1; k++) {
34                    Complex tmp = *r * *p;
35                    *r = *l - tmp, *l = *l + tmp;
36                    ++l, ++r, p += lcy;
37                }
38            }
39    }
40
41    inline void MTT(ll *x, ll *y, ll *z, int n) {
42        int len = 1; while (len <= n) len <= 1;
43        for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 :
44            0);
45    }
46 }

```

```

41     for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin
(2 * PI * i / len));
42
43     for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
44     static Complex a[N], b[N];
45     static Complex dfta[N], dftb[N], dftc[N], dftd[N];
46
47     for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
48     for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
49     FFT(a, len), FFT(b, len);
50     for (int i = 0; i < len; i++) {
51         int j = (len - i) & (len - 1);
52         static Complex da, db, dc, dd;
53         da = (a[i] + a[j].conj()) * Complex(0.5, 0);
54         db = (a[i] - a[j].conj()) * Complex(0, -0.5);
55         dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
56         dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
57         dfta[j] = da * dc;
58         dftb[j] = da * dd;
59         dftc[j] = db * dc;
60         dftd[j] = db * dd;
61     }
62     for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
63     for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
64     FFT(a, len), FFT(b, len);
65     for (int i = 0; i < len; i++) {
66         int da = (ll)(a[i].x / len + 0.5) % mod;
67         int db = (ll)(a[i].y / len + 0.5) % mod;
68         int dc = (ll)(b[i].x / len + 0.5) % mod;
69         int dd = (ll)(b[i].y / len + 0.5) % mod;
70         z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
71     }
72 }
73
74 int getlen(int n) {
75     int len = 1; while (len < (n << 1)) len <<= 1;
76     for (int i = 0; i < len; i++) tr[i] = (tr[i] >> 1) >> 1 | (i & 1 ? len >> 1 :
0);
77     for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin
(2 * PI * i / len));
78     return len;
79 }
80
81 void Get_Inv(ll *f, ll *g, int n) {
82     if(n == 1) { g[0] = quick_pow(f[0], mod - 2, mod); return ; }
83     Get_Inv(f, g, (n + 1) >> 1);
84     int len = getlen(n);
85     static ll c[N];
86     for(int i = 0; i < len; i++) c[i] = i < n ? f[i] : 0;
87     MTT(c, g, c, len); MTT(c, g, c, len);
88     for(int i = 0; i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
89     for(int i = n; i < len; i++) g[i] = 0;
90     for(int i = 0; i < len; i++) c[i] = 0;
91 }
92
93 ll f[N], invff[N], inv[N];
94 ll B[N];
95
96 ll C(ll m, ll n) {

```

```

97     if(m < 0 || n < 0 || n > m)
98         return 0;
99     ll ans = ff[m];
100     ans = ans * invff[n] % mod;
101     ans = ans * invff[m - n] % mod;
102     return ans;
103 }
104
105 void init(int m) {
106     ff[0] = ff[1] = inv[0] = inv[1] = invff[0] = invff[1] = 1;
107     for(int i = 2; i < M; i++)
108     {
109         ff[i] = ff[i - 1] * i % mod;
110         inv[i] = mod - (mod / i) * inv[mod % i] % mod;
111         invff[i] = invff[i - 1] * inv[i] % mod;
112     }
113
114     for(int i = 0; i <= m + 10; i++) F[i] = invff[i + 1];
115     Get_Inv(F, G, m + 10);
116     for(int i = 0; i <= m + 10; i++) B[i] = G[i] * ff[i] % mod;
117 }
118
119 ll solve(ll n, int k) {
120     init(k);
121     ll ans = 0, prod = n % mod;
122     for(int i = k; ~i; i--) {
123         ans = (ans + prod * B[i] % mod * C(k + 1, i) % mod) % mod;
124         prod = prod * n % mod;
125     }
126     ans = ans * quick_pow(k + 1, mod - 2, mod) % mod;
127     return ans;
128 }
129 }
130
131 void solve() {
132     ll n; int k; cin >> n >> k;
133     cout << BNL::solve(n + 1, k) << endl;
134 }

```

6.31 步移-组合数前缀和

```

1  //  $S(n, m) = \sum_{i=0}^m C(n, i)$ 
2
3  int x, y;
4  ll s;
5  ll S(int n, int m) {
6      while(y < m) (s = s + C(x, ++y)) %= mod;
7      while(y > m) (s = s - C(x, y--)) %= mod;
8      while(x < n) (s = s * 2 - C(x++, y)) %= mod;
9      while(x > n) (s = (s + C(--x, y)) * inv2) %= mod;
10     return s;
11 }

```

6.32 康托展开

```

1  #include <iostream>
2  #include <vector>

```

```

3  #include <algorithm>
4
5  using namespace std;
6
7  typedef long long ll;
8  const int mod = 1e9 + 7;
9  const int N = 1e5 + 10;
10
11 ll fac[N];
12 int a[N]; // 排列, 康托展开求解
13 int n;
14 ll x; // 逆康托展开求解
15
16 void Get_FC() {
17     fac[0] = 1;
18     for(int i = 1; i < N; i++)
19         fac[i] = fac[i - 1] * i % mod;
20 }
21
22 ll CanTor() {
23     ll ans = 0;
24     for(int i = 1; i <= n; i++) {
25         ll smaller = 0;
26         for(int j = i + 1; j <= n; j++) {
27             if(a[j] < a[i])
28                 smaller++;
29         }
30         ans = (ans + fac[n - i] * smaller % mod) % mod;
31     }
32     return ans + 1;
33 }
34
35 void DeCantor() {
36     vector<int> v; // 存放当前可选数
37     vector<int> a; // 所求的排列组合序
38     for(int i = 1; i <= n; i++) {
39         v.push_back(i);
40     }
41     for(int i = n; i >= 1; i--) {
42         int r = x % fac[i - 1];
43         int t = x / fac[i - 1];
44         x = r;
45         sort(v.begin(), v.end());
46         a.push_back(v[t]);
47         v.erase(v.begin() + t);
48     }
49     for(int i = 0; i < a.size(); i++)
50         cout << a[i] << " ";
51     cout << endl;
52 }
53
54 // 线段树优化
55
56 const int N = 1000010;
57
58 ll fac[N];
59 int a[N]; // 排列, 康托展开求解
60 int n;
61

```

```

62 struct SegmentTree {
63     int ls, rs;
64     int sum;
65 }t[N << 2];
66
67 int cnt, root;
68
69 void push_up(int u) {
70     t[u].sum = (t[lc].sum + t[rc].sum) % mod;
71 }
72
73 void build(int &u, int l, int r) {
74     if(!u) u = ++cnt;
75     if(l == r) {
76         t[u].sum = 1;
77         return ;
78     }
79     build(lc, l, m);
80     build(rc, m + 1, r);
81     push_up(u);
82 }
83
84 void update(int &u, int l, int r, int k) {
85     if(!u) u = ++cnt;
86     if(l == r) {
87         t[u].sum = 0;
88         return ;
89     }
90     if(k <= m) update(lc, l, m, k);
91     else update(rc, m + 1, r, k);
92     push_up(u);
93 }
94
95 ll query(int u, int l, int r, int ql, int qr) {
96     if(ql > qr) return 0;
97     if(ql == l && qr == r) {
98         return t[u].sum;
99     }
100     if(qr <= m) return query(lc, l, m, ql, qr) % mod;
101     else if(ql > m) return query(rc, m + 1, r, ql, qr) % mod;
102     else return (query(lc, l, m, ql, m) + query(rc, m + 1, r, m + 1, qr)) % mod;
103 }
104
105 void Get_F() {
106     fac[0] = 1;
107     for(int i = 1; i < N; i++)
108         fac[i] = fac[i - 1] * i % mod;
109 }
110
111 void solve()
112 {
113     Get_F();
114     cin >> n;
115     build(root, 1, n);
116     ll ans = 0;
117     for(int i = 1; i <= n; i++) {
118         cin >> a[i];
119         update(root, 1, n, a[i]);
120         ans = (ans + query(root, 1, n, 1, a[i] - 1) * fac[n - i]) % mod;

```

```

121     }
122     cout << (ans + 1) % mod << endl;
123 }

```

6.33 模数非质数的组合

```

1  // 模数非质数情况下的组合问题
2  // one way, use CRT merge ans
3  // https://ac.nowcoder.com/discuss/655940?type=101&order=0&pos=2&page=1&channel=-1&
   source_id=discuss_tag_nctrack
4  // another way
5  // https://ac.nowcoder.com/acm/contest/view-submission?submissionId=47754622
6
7  #include <bits/stdc++.h>
8
9  using namespace std;
10 typedef long long ll;
11 const int N = 1e6 + 10;
12
13 ll qpow(ll a, ll b, ll mod) {
14     ll res = 1;
15     while (b) {
16         if (b & 1) res = res * a % mod;
17         a = a * a % mod;
18         b >>= 1;
19     }
20     return res;
21 }
22
23 ll exgcd(ll a, ll b, ll &x, ll &y) {
24     if (!b) {
25         x = 1, y = 0;
26         return a;
27     }
28     ll res = exgcd(b, a % b, x, y);
29     ll t = y;
30     y = x - a / b * y;
31     x = t;
32     return res;
33 }
34
35 ll inv(ll a, ll b) {
36     ll x = 0, y = 0;
37     exgcd(a, b, x, y);
38     return x = (x % b + b) % b;
39 }
40
41 //r[]为余数, m为模数, 其中模数互质
42 //M = pi(mi), Mi = M / mi, invMi = Mi % mi
43 //ni满足是除了mi之外的倍数, 且模mi为ri
44 //利用逆元性质, 即ri * Mi * invMi = ri (mod mi)
45 //res = (sigma(ri * Mi * invMi)) % M
46
47 ll china(ll r[], ll m[], int n) {
48     ll M = 1, res = 0;
49     for (int i = 1; i <= n; i++) M *= m[i];
50     for (int i = 1; i <= n; i++) {
51         ll Mi = M / m[i], invMi = inv(Mi, m[i]);

```

```

52     res = (res + r[i] * Mi % M * invMi % M) % M;
53     //res = (res + mul(mul(r[i], Mi, M), invMi, M)) % M;按位乘
54 }
55 return (res % M + M) % M;
56 }
57
58 int f[N], g[N], F[N], G[N], invF[N];
59
60 int calc(int n, int p, int k) {
61     ll mod = qpow(p, k, LONG_LONG_MAX);
62     F[0] = 1, G[0] = 0;
63     for (int i = 1; i <= n; i++) {
64         g[i] = 0, f[i] = i;
65         while (f[i] % p == 0) f[i] /= p, g[i]++;
66         F[i] = 1ll * F[i - 1] * f[i] % mod;
67         G[i] = G[i - 1] + g[i];
68     }
69     invF[n] = inv(F[n], mod);
70     for (int i = n; i >= 1; i--) invF[i - 1] = 1ll * invF[i] * f[i] % mod;
71     int ans = 0;
72     for (int i = 0; i <= n / 2; i++) {
73         int t = 1ll * F[n] * invF[n - 2 * i] % mod * invF[i] % mod * invF[i] % mod *
74             qpow(p, G[n] - G[n - 2 * i] - 2 * G[i], LONG_LONG_MAX) % mod;
75         ans = (ans + 1ll * t) % mod;
76     }
77     return ans;
78 }
79
80 ll r[20], m[20];
81
82 int main() {
83 #ifdef ACM_LOCAL
84     freopen("input.in", "r", stdin);
85     freopen("output.out", "w", stdout);
86 #endif
87     int n, p;
88     scanf("%d%d", &n, &p);
89     int num = 0;
90     for (int i = 2; i * i <= p; i++)
91         if (p % i == 0) {
92             int k = 0;
93             m[++num] = 1;
94             while (p % i == 0) p /= i, k++, m[num] *= i;
95             r[num] = calc(n, i, k);
96         }
97     if (p > 1) {
98         m[++num] = p;
99         r[num] = calc(n, p, 1);
100     }
101     printf("%lld\n", china(r, m, num));
102     return 0;
103 }
104 }

```

6.34 k 次最小置换复原

```

1
2 void solve() {

```

```

3     int n; cin >> n;
4     vector<int> a(n + 1), vis(n + 1);
5     for(int i = 1; i <= n; i++) cin >> a[i];
6     ll ans = 1;
7     for(int i = 1; i <= n; i++) {
8         if(!vis[i]) {
9             int cnt = 0;
10            int x = i;
11            while(!vis[x]) {
12                vis[x] = 1;
13                cnt++;
14                x = a[x];
15            }
16            ans = lcm(ans, cnt);
17        }
18    }
19    cout << ans << endl;
20 }

```

6.35 Stirling

```

1  typedef long long ll;
2  const int N = 20;
3
4  // 第一类斯特林数
5  ll S1[N][N];
6  void Stirling1() {
7      S1[0][0] = 1;
8      for(int i = 1; i < N; i++) {
9          for(int j = 1; j <= i; j++) {
10             S1[i][j] = S1[i - 1][j - 1] + (i - 1) * S1[i - 1][j];
11         }
12     }
13 }
14
15 // 第二类斯特林数
16 ll S2[N][N];
17 void Stirling2() {
18     S2[0][0] = 1;
19     for(int i = 1; i < N; i++) {
20         for(int j = 1; j <= i; j++) {
21             S2[i][j] = S2[i - 1][j - 1] + j * S2[i - 1][j];
22         }
23     }
24 }

```

6.36 第二类斯特林数-行

```

1  // {n,m} -> n个不同元素划分成m个相同的集合中（不能有空集）的方案数。
2
3  // {n,m} = {n-1,m-1} + k{n-1,m}
4
5  // {n,m} = \sum_{i=0}^n \frac{i^n}{i!} * \{(-1)^{m-i}\} \{(m-i)!\}
6
7  const int N = 1e6 + 10;
8  const ll mod = 167772161;
9

```



```

10 ll F[N], invF[N];
11 void init() ;
12
13 ll qpow(ll a, ll b, ll mod) ;
14
15 const ll G = 3;
16 const ll invG = qpow(G, mod - 2, mod);
17 int tr[N];
18
19 void NTT(ll *A, int len, int type) ;
20 void mul(ll *a, ll *b, int n) ;
21
22 ll a[N], b[N];
23
24 void solve() {
25     init();
26     int n; cin >> n;
27     for(int i = 0; i <= n; i++) {
28         a[i] = qpow(i, n, mod) * invF[i] % mod;
29         if(i & 1) b[i] = mod - invF[i];
30         else b[i] = invF[i];
31     }
32     mul(a, b, 2 * n);
33     for(int i = 0; i <= n; i++) cout << a[i] << (i == n ? endl : " ");
34 }

```

6.37 第二类斯特林数-列

```

1 // 把n个不同元素划分成m个相同的集合（不能有空集）的方案数。
2
3 //  $k! \sum_{i=0}^n \frac{\{i, k\} x^i}{i!} = (e^x - 1)^k$ 
4
5 const int N = 6e5 + 10;
6 const ll mod = 167772161;
7
8 ll quick_pow(ll a, ll b) ;
9
10 const ll G = 3;
11 const ll invG = quick_pow(G, mod - 2);
12
13 int tr[N];
14 bool flag;
15
16 void NTT(ll *A, int len, int type) ;
17 void mul(ll *a, ll *b, int len) ;
18 void Get_Der(ll *f, ll *g, int len) ;
19 void Get_Int(ll *f, ll *g, int len) ;
20 void Get_Inv(ll *f, ll *g, int n) ;
21 void Get_Ln(ll *f, ll *g, int n) ;
22 void Get_Exp(ll *f, ll *g, int n) ;
23 void Get_Pow(ll *f, ll *g, int n, ll k1, ll k2);
24
25 ll a[N], ans[N];
26
27 ll F[N], invF[N], inv[N];
28 void init() ;
29
30

```

```

31 void solve() {
32     init();
33     int n; ll k; cin >> n >> k; n++;
34     if(k >= mod) flag = 1;
35     for(int i = 1; i < n; i++) a[i] = invF[i];
36     Get_Pow(a, ans, n, k % mod, k % (mod - 1));
37     for(int i = 0; i < n; i++) {
38         cout << ans[i] * invF[k] % mod * F[i] % mod << (i == n - 1 ? endl : " ");
39     }
40 }

```

6.38 第一类斯特林数-行

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <cstring>
4
5  typedef long long LL;
6  const int N = 550050;
7  const int mod = 167772161;
8
9  LL pow_mod(LL a, LL b) {
10     LL ans = 1;
11     for (; b >= 1, a = a * a % mod)
12         if (b & 1) ans = ans * a % mod;
13     return ans;
14 }
15
16 int L, rev[N];
17 LL w[N], inv[N], fac[N], ifac[N];
18
19 void Init(int n) {
20     L = 1;
21     while (L <= n) L <<= 1;
22     for (int i = 1; i < L; ++i)
23         rev[i] = (rev[i >> 1] >> 1) | ((i & 1) * L / 2);
24     LL wn = pow_mod(3, (mod - 1) / L);
25     w[L >> 1] = 1;
26     for (int i = L >> 1; i < L; ++i) w[i + 1] = w[i] * wn % mod;
27     for (int i = (L >> 1) - 1; i; --i) w[i] = w[i << 1];
28 }
29
30 void DFT(LL *A, int len) {
31     int k = __builtin_ctz(L) - __builtin_ctz(len);
32     for (int i = 1; i < len; ++i) {
33         int j = rev[i] >> k;
34         if (j > i) std::swap(A[i], A[j]);
35     }
36     for (int h = 1; h < len; h <<= 1)
37         for (int i = 0; i < len; i += (h << 1))
38             for (int j = 0; j < h; ++j) {
39                 LL t = A[i + j + h] * w[j + h] % mod;
40                 A[i + j + h] = A[i + j] - t;
41                 A[i + j] += t;
42             }
43     for (int i = 0; i < len; ++i) A[i] %= mod;
44 }
45

```

```

46 void IDFT(LL *A, int len) {
47     std::reverse(A + 1, A + len);
48     DFT(A, len);
49     int v = mod - (mod - 1) / len;
50     for (int i = 0; i < len; ++i) A[i] = A[i] * v % mod;
51 }
52
53 void offset(const LL *f, int n, LL c, LL *g) {
54     //  $g(x) = f(x + c)$ 
55     //  $g[i] = 1/i! \sum_{j=i}^n j! f[j] c^{j-i} / (j-i)!$ 
56     static LL tA[N], tB[N];
57     int l = 1; while (l <= n + n) l <<= 1;
58     for (int i = 0; i < n; ++i) tA[n - i - 1] = f[i] * fac[i] % mod;
59     LL pc = 1;
60     for (int i = 0; i < n; ++i, pc = pc * c % mod) tB[i] = pc * ifac[i] % mod;
61     for (int i = n; i < l; ++i) tA[i] = tB[i] = 0;
62     DFT(tA, l); DFT(tB, l);
63     for (int i = 0; i < l; ++i) tA[i] = tA[i] * tB[i] % mod;
64     IDFT(tA, l);
65     for (int i = 0; i < n; ++i)
66         g[i] = tA[n - i - 1] * ifac[i] % mod;
67 }
68
69 void Solve(int n, LL *f) {
70     if (n == 0) return void(f[0] = 1);
71     static LL tA[N], tB[N];
72     int m = n / 2;
73     Solve(m, f);
74     int l = 1; while (l <= n) l <<= 1;
75     offset(f, m + 1, m, tA);
76     for (int i = 0; i <= m; ++i) tB[i] = f[i];
77     for (int i = m + 1; i < l; ++i) tA[i] = tB[i] = 0;
78     DFT(tA, l); DFT(tB, l);
79     for (int i = 0; i < l; ++i) tA[i] = tA[i] * tB[i] % mod;
80     IDFT(tA, l);
81     if (n & 1)
82         for (int i = 0; i <= n; ++i)
83             f[i] = ((i ? tA[i - 1] : 0) + (n - 1) * tA[i]) % mod;
84     else
85         for (int i = 0; i <= n; ++i)
86             f[i] = tA[i];
87 }
88
89 LL f[N];
90
91 int main() {
92     int n;
93     scanf("%d", &n);
94     Init(n * 2);
95     inv[1] = 1;
96     for (int i = 2; i <= n; ++i) inv[i] = -(mod / i) * inv[mod % i] % mod;
97     fac[0] = ifac[0] = 1;
98     for (int i = 1; i <= n; ++i) {
99         fac[i] = fac[i - 1] * i % mod;
100         ifac[i] = ifac[i - 1] * inv[i] % mod;
101     }
102     Solve(n, f);
103     for (int i = 0; i <= n; ++i)
104         printf("%lld ", (f[i] + mod) % mod);

```

```

105     return 0;
106 }

```

6.39 第一类斯特林数-列

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define Int register int
5  #define mod 167772161
6  #define MAXN 531072
7  #define Gi 3
8
9  int quick_pow (int a,int b,int c)
10 {
11     int res = 1;
12     while (b){
13         if (b & 1) res = 1ll * res * a % c;
14         a = 1ll * a * a % c;
15         b >>= 1;
16     }
17     return res;
18 }
19
20 int limit = 1,l,r[MAXN];
21
22 void NTT (int *a,int type)
23 {
24     for (Int i = 0;i < limit;++ i) if (i < r[i]) swap (a[i],a[r[i]]);
25     for (Int mid = 1;mid < limit;mid <= 1){
26         int Wn = quick_pow (Gi,(mod - 1) / (mid < 1),mod);
27         if (type == -1) Wn = quick_pow (Wn,mod - 2,mod);
28         for (Int R = mid < 1,j = 0;j < limit;j += R){
29             for (Int k = 0,w = 1;k < mid;++ k,w = 1ll * w * Wn % mod)
30             {
31                 int x = a[j + k],y = 1ll * w * a[j + k + mid] % mod;
32                 a[j + k] = (x + y) % mod,a[j + k + mid] = (x + mod - y) % mod;
33             }
34         }
35     }
36     if (type == 1) return ;
37     int Inv = quick_pow (limit,mod - 2,mod);
38     for (Int i = 0;i < limit;++ i) a[i] = 1ll * a[i] * Inv % mod;
39 }
40
41 int c[MAXN];
42
43 void Solve (int len,int *a,int *b)
44 {
45     if (len == 1) return b[0] = quick_pow (a[0],mod - 2,mod),void ();
46     Solve ((len + 1) >> 1,a,b);
47     limit = 1,l = 0;
48     while (limit < (len < 1)) limit <= 1,l ++;
49     for (Int i = 0;i < limit;++ i) r[i] = (r[i] >> 1) >> 1 | ((i & 1) << (l - 1));
50     for (Int i = 0;i < len;++ i) c[i] = a[i];
51     for (Int i = len;i < limit;++ i) c[i] = 0;
52     NTT (c,1);NTT (b,1);

```

```

53     for (Int i = 0; i < limit; ++ i) b[i] = 1ll * b[i] * (2 + mod - 1ll * c[i] * b[i] %
mod) % mod;
54     NTT (b, -1);
55     for (Int i = len; i < limit; ++ i) b[i] = 0;
56 }
57
58 void deravitive (int *a, int n){
59     for (Int i = 1; i <= n; ++ i) a[i - 1] = 1ll * a[i] * i % mod;
60     a[n] = 0;
61 }
62
63 void inter (int *a, int n){
64     for (Int i = n; i >= 1; -- i) a[i] = 1ll * a[i - 1] * quick_pow (i, mod - 2, mod) % mod
;
65     a[0] = 0;
66 }
67
68 int b[MAXN];
69
70 void Ln (int *a, int n){
71     memset (b, 0, sizeof (b));
72     Solve (n, a, b); deravitive (a, n);
73     while (limit <= n) limit <<= 1, l ++;
74     for (Int i = 0; i < limit; ++ i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
75     NTT (a, 1), NTT (b, 1);
76     for (Int i = 0; i < limit; ++ i) a[i] = 1ll * a[i] * b[i] % mod;
77     NTT (a, -1);
78     inter (a, n);
79     for (Int i = n + 1; i < limit; ++ i) a[i] = 0;
80 }
81
82 int F0[MAXN];
83
84 void Exp (int *a, int *B, int n){
85     if (n == 1) return B[0] = 1, void ();
86     Exp (a, B, (n + 1) >> 1);
87     for (Int i = 0; i < limit; ++ i) F0[i] = B[i];
88     Ln (F0, n);
89     F0[0] = (a[0] + 1 + mod - F0[0]) % mod;
90     for (Int i = 1; i < n; ++ i) F0[i] = (a[i] + mod - F0[i]) % mod;
91     NTT (F0, 1); NTT (B, 1);
92     for (Int i = 0; i < limit; ++ i) B[i] = 1ll * F0[i] * B[i] % mod;
93     NTT (B, -1);
94     for (Int i = n; i < limit; ++ i) B[i] = 0;
95 }
96
97 int read ()
98 {
99     int x = 0; char c = getchar(); int f = 1;
100     while (c < '0' || c > '9'){if (c == '-') f = -f; c = getchar();}
101     while (c >= '0' && c <= '9'){x = (int)((int)(x << 3) % mod + (int)(x << 1) % mod +
c - '0') % mod; c = getchar();}
102     return x * f;
103 }
104
105 void write (int x)
106 {
107     if (x < 0){x = -x; putchar ('-');}
108     if (x > 9) write (x / 10);

```

```

109     putchar (x % 10 + '0');
110 }
111
112 int n,k;
113 int fac[MAXN],A[MAXN],B[MAXN];
114
115 signed main()
116 {
117     n = read (),k = read ();
118     for (Int i = 0;i < n;++ i) A[i] = quick_pow (i + 1,mod - 2,mod);
119     Ln (A,n);
120     for (Int i = 0;i < n;++ i) A[i] = 1ll * A[i] * k % mod;
121     Exp (A,B,n);fac[0] = 1;
122     for (Int i = 1;i <= max (n,k;++ i) fac[i] = 1ll * fac[i - 1] * i % mod;
123     for (Int i = n;i >= k;-- i) B[i] = B[i - k];
124     for (Int i = 0;i < k;++ i) B[i] = 0;int Inv = quick_pow (fac[k],mod - 2,mod);
125     for (Int i = 0;i <= n;++ i) write (1ll * B[i] * fac[i] % mod * Inv % mod),putchar (
126         ' ');
127     putchar ('\n');
128     return 0;
129 }

```

6.40 整数拆分多项式求逆

```

1
2 // NTT求法, 任意模数复杂度较高
3 #include <bits/stdc++.h>
4 using namespace std;
5 typedef long long ll;
6 const double PI = acos(-1);
7 const int N = 1e5 + 10;
8
9 struct Complex {
10     double x, y;
11     Complex(double a = 0, double b = 0): x(a), y(b) {}
12     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
13     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
14     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
15         rhs.y + y * rhs.x); }
16     Complex conj() { return Complex(x, -y); }
17 } w[N];
18
19 ll mod, g;
20 int tr[N];
21 ll F[N], G[N];
22
23 ll quick_pow(ll a, ll b) ;
24
25 int getLen(int n) ;
26
27 void NTT(ll *a, int len, int opt) ;
28
29 void mul(ll *a, ll *b, ll *z, int n) ;
30
31 void Get_Inv(ll *f, ll *g, int n) ;
32
33 void ChaiFen(ll *f, ll *g, int n) {
34     int len = getLen(n);

```

```

34     for(int i = 0; i < len; i++) {
35         ll a = 1ll * i * (3 * i - 1) / 2; ll b = 1ll * i * (3 * i + 1) / 2;
36         if(a > len && b > len) break;
37         ll tmp;
38         if(i & 1) tmp = mod - 1;
39         else tmp = 1;
40         if(a < len) f[a] = tmp;
41         if(b < len) f[b] = tmp;
42     }
43
44     Get_Inv(f, g, n);
45 }
46
47 int main() {
48     int n;
49     cin >> n;
50     ChaiFen(F, G, n);
51     for(int i = 1; i <= n; i++) cout << G[i] << endl;
52 }

```

6.41 普通型母函数

```

1  // 普通型母函数: (1+x^1+x^2+...) (1+x^2+x^4)(1+x^3+x^6...)(...)(...)... 类似整数拆分
2
3  // a_n=1,1,1,1... = \frac{1}{1-x}
4  // a_n=1,0,1,0... = \frac{1}{1-x^2}
5  // a_n=1,2,3,4... = \frac{1}{(1-x)^2}
6  // a_n=C(m,n) = (1+x)^m
7  // a_n=C(m+n,n) = \frac{1}{(1-x)^{m+1}}
8
9  #include <bits/stdc++.h>
10 using namespace std;
11
12 typedef long long ll;
13
14 // 求解硬币等普通问题
15
16 const int N = 1e5 + 10;
17
18 int a[N]; // 权重为i的组合数, a[P]为答案
19 int b[N]; // 辅助数组
20 int P; // 需要被分解的数
21 int k; // 物品个数
22 int v[N]; // 每个物品的权重
23 int n1[N]; // 对于每种物品起始的因子 (所需要的每个物品最小个数), 最小为0
24 int n2[N]; // 对于每种物品最终的因子 (所需要的每个物品最大个数), 最大为INF
25
26 // 模板一(标准)
27
28 void Calc1() {
29     memset(a, 0, sizeof(a));
30     a[0] = 1;
31
32     for(int i = 1; i <= k; i++) { // 枚举每个物品因子
33         memset(b, 0, sizeof(b));
34         for(int j = n1[i]; j <= n2[i] && j * v[i] <= P; j++) { // 每个物品从最小因子到最大因子循环, 如果n2是无穷的, 则j<=n2[i]可以删去
35             for(int m = 0; m + j * v[i] <= P; m++) { // 循环a的每个项

```

```

36         b[m + j * v[i]] += a[m]; // 把结果加到对应项里, 有点dp的味道
37     }
38 }
39 memcpy(a, b, sizeof(b));
40 }
41 }
42
43 // 模板二 (数据量大的时候可以用, 快速)
44
45 void Calc2() {
46     memset(a, 0, sizeof(a));
47     a[0] = 1;
48     int last = 0;
49     for(int i = 1; i <= k; i++) {
50         int last2 = min(last + n2[i] * v[i], P); // 计算下一次的last
51         memset(b, 0, sizeof(int) * (last2 + 1)); // 只清空b[0..last2]
52         for(int j = n1[i]; j <= n2[i] && j * v[i] <= last2; j++) // last2
53             for(int m = 0; m <= last && m + j * v[i] <= last2; m++) // 一个是last, 一个是
last2
54                 b[m + j * v[i]] += a[m];
55         memcpy(a, b, sizeof(int) * (last2 + 1)); // b赋值给a, 只赋值0..last2
56         last = last2; // 更新last
57     }
58 }

```

6.42 指数型母函数

```

1
2 // 需要借助 $e^x$ 的泰勒展开, 一般求解多重排列数, 即有  $k_1, k_2, \dots, k_n$  个, 求
   从中选出  $m$  件物品的排列数。
3
4 // 对  $n$  个元素全排列, 方案数为  $n!/(n_1!n_2!\dots n_k!)$ , 对  $n$  个中的  $r$  个元素进行全排列, 这里就用到了指数型母函
   数, 即  $G(x) = (1+x/1!+x^2/2!+\dots+x^{k_1}/k_1!)(1+x/1!+x^2/2!+\dots+x^{k_2}/k_2!)\dots(1+x/1!+x^{k_n}/k_n!)$ 
5
6 // 化简得  $G(x) = a_0 + a_1x + a_2x^2/2! + \dots + a_px^p/p!$  ( $p = k_1+k_2+k_3+\dots$ )  $a_i$  为选出  $i$  个物品的排列
   方案数
7
8 // 若题目有规定条件, 比如需要物品  $i$  出现非0的偶数次, 即原式为  $(x^2/2!+x^4/4!+\dots+x^{k_i}/k_i!)$ 
9
10 #include <bits/stdc++.h>
11 using namespace std;
12
13 typedef long double ld;
14
15 double num[15]; // 每种物品的数量, 第  $i$  个物品有  $num[i]$  个
16
17 double a[15], b[15];
18
19 double f[120]; // 阶乘
20
21 void fac()
22 {
23     f[0] = 1;
24     for(int i = 1; i <= 105; i++)
25         f[i] = f[i - 1] * i;
26 }
27

```



```

28 void Calc() {
29     int n, m;
30     cin >> n >> m;
31     for(int i = 1; i <= n; i++)
32         cin >> num[i];
33
34     memset(a, 0, sizeof(a));
35     memset(b, 0, sizeof(b));
36
37     for(int i = 0; i <= num[1]; i++) {
38         a[i] = 1.0 / f[i];
39     }
40
41     for(int i = 2; i <= n; i++) {
42         for(int j = 0; j <= m; j++) {
43             for(int k = 0; k <= num[i] && j + k <= m; k++) {
44                 b[j + k] += a[j] / f[k];
45             }
46         }
47
48         for(int j = 0; j <= m; j++) {
49             a[j] = b[j];
50             b[j] = 0;
51         }
52     }
53
54     cout << a[m] * f[m] << endl;
55 }

```

6.43 伯努利数求和

```

1 namespace BNL {
2     const int N = 1e7 + 10, M = 1e6 + 10;
3     struct Complex {
4         double x, y;
5         Complex(double a = 0, double b = 0): x(a), y(b) {}
6         Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
7         Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
8         Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y,
9             x * rhs.y + y * rhs.x); }
9         Complex conj() { return Complex(x, -y); }
10    } w[N];
11
12    int tr[N];
13    ll F[N], G[N];
14
15    ll quick_pow(ll a, ll b, ll p) {
16        ll ans = 1;
17        while(b) {
18            if(b & 1) ans = ans * a % p;
19            a = a * a % p;
20            b >>= 1;
21        }
22        return ans % p;
23    }
24

```

```

25 void FFT(Complex *A, int len) {
26     for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
27     for (int i = 2, lyc = len >> 1; i <= len; i <= 1, lyc >= 1)
28         for (int j = 0; j < len; j += i) {
29             Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
30             for (int k = 0; k < i >> 1; k++) {
31                 Complex tmp = *r * *p;
32                 *r = *l - tmp, *l = *l + tmp;
33                 ++l, ++r, p += lyc;
34             }
35         }
36 }
37
38 inline void MTT(ll *x, ll *y, ll *z, int n) {
39     int len = 1; while (len <= n) len <= 1;
40     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 :
41 0);
42     for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin
43 (2 * PI * i / len));
44
45     for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
46     static Complex a[N], b[N];
47     static Complex dfta[N], dftb[N], dftc[N], dftd[N];
48
49     for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
50     for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
51     FFT(a, len), FFT(b, len);
52     for (int i = 0; i < len; i++) {
53         int j = (len - i) & (len - 1);
54         static Complex da, db, dc, dd;
55         da = (a[i] + a[j].conj()) * Complex(0.5, 0);
56         db = (a[i] - a[j].conj()) * Complex(0, -0.5);
57         dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
58         dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
59         dfta[j] = da * dc;
60         dftb[j] = da * dd;
61         dftc[j] = db * dc;
62         dftd[j] = db * dd;
63     }
64     for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
65     for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
66     FFT(a, len), FFT(b, len);
67     for (int i = 0; i < len; i++) {
68         int da = (ll)(a[i].x / len + 0.5) % mod;
69         int db = (ll)(a[i].y / len + 0.5) % mod;
70         int dc = (ll)(b[i].x / len + 0.5) % mod;
71         int dd = (ll)(b[i].y / len + 0.5) % mod;
72         z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
73     }
74 }
75
76 int getLen(int n) {
77     int len = 1; while (len < (n << 1)) len <= 1;
78     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 :
79 0);
80     for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin
81 (2 * PI * i / len));
82     return len;
83 }

```

```

80
81 void Get_Inv(ll *f, ll *g, int n) {
82     if(n == 1) { g[0] = quick_pow(f[0], mod - 2, mod); return ; }
83     Get_Inv(f, g, (n + 1) >> 1);
84     int len = getLen(n);
85     static ll c[N];
86     for(int i = 0; i < len; i++) c[i] = i < n ? f[i] : 0;
87     MTT(c, g, c, len); MTT(c, g, c, len);
88     for(int i = 0; i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
89     for(int i = n; i < len; i++) g[i] = 0;
90     for(int i = 0; i < len; i++) c[i] = 0;
91 }
92
93 ll ff[N], invff[N], inv[N];
94 ll B[N];
95
96 ll C(ll m, ll n) {
97     if(m < 0 || n < 0 || n > m)
98         return 0;
99     ll ans = ff[m];
100    ans = ans * invff[n] % mod;
101    ans = ans * invff[m - n] % mod;
102    return ans;
103 }
104
105 void init(int m) {
106     ff[0] = ff[1] = inv[0] = inv[1] = invff[0] = invff[1] = 1;
107     for(int i = 2; i < M; i++)
108     {
109         ff[i] = ff[i - 1] * i % mod;
110         inv[i] = mod - (mod / i) * inv[mod % i] % mod;
111         invff[i] = invff[i - 1] * inv[i] % mod;
112     }
113
114     for(int i = 0; i <= m + 10; i++) F[i] = invff[i + 1];
115     Get_Inv(F, G, m + 10);
116     for(int i = 0; i <= m + 10; i++) B[i] = G[i] * ff[i] % mod;
117 }
118
119 ll solve(ll n, int k) {
120     init(k);
121     ll ans = 0, prod = n % mod;
122     for(int i = k; ~i; i--) {
123         ans = (ans + prod * B[i] % mod * C(k + 1, i) % mod) % mod;
124         prod = prod * n % mod;
125     }
126     ans = ans * quick_pow(k + 1, mod - 2, mod) % mod;
127     return ans;
128 }
129 }
130
131 void solve() {
132     ll n; int k; cin >> n >> k;
133     cout << BNL::solve(n + 1, k) << endl;
134 }

```

6.44 fibonacciBigInteger

```

1 class Fib {
2     int n;
3     std::vector<std::vector<int>> f;
4     Fib(int _n) : n(_n), f(_n, std::vector<int>(_n)) {
5         f[1][1] = 1;
6         f[2][1] = 1;
7         int s, add = 0;
8         for (int i = 3; i < _n; i++) {
9             for (int j = 1; j < _n; j++) {
10                 s = f[i - 2][j] + f[i - 1][j] + add;
11                 f[i][j] = s % 10;
12                 add = s / 10;
13             }
14         }
15     }
16
17     std::string get(const int m) {
18         int k = n - 1;
19         while (!f[m][k]) k--;
20         std::string s = "";
21         s.push_back(f[n][k] + '0');
22         return s;
23     }
24 };

```

6.45 fibonacciCircular

```

1 // 斐波那契循环节
2
3 typedef long long ll;
4 typedef long double ld;
5 typedef pair<int, int> pdd;
6
7 #define INF 0x7f7f7f
8 #define mem(a,b) memset(a, b, sizeof(a))
9 #define FOR(i, x, n) for(int i = x; i <= n; i++)
10
11 const ll mod = 1e9 + 7;
12 const int maxn = 5e6 + 10;
13
14 bool is_prime[maxn];
15 ll prime[maxn];
16 int p;
17
18 void sieve() // 素数筛
19 {
20     p = 0;
21     mem(is_prime, true);
22     is_prime[0] = is_prime[1] = false;
23     for(int i = 2; i < maxn; i++)
24     {
25         if(is_prime[i])
26         {
27             prime[++p] = i;
28             for(int j = i + i; j < maxn; j += i)
29             {
30                 is_prime[j] = false;
31             }
32         }
33     }
34 }

```

```

32     }
33 }
34 }
35
36 ll gcd(ll a, ll b)
37 {
38     return b ? gcd(b, a % b) : a;
39 }
40
41 ll quick_pow(ll a, ll b)
42 {
43     a %= mod;
44     ll ans = 1;
45     ll base = a;
46     while(b)
47     {
48         if(b&1)
49         {
50             ans = ans * base % mod;
51         }
52         base = base * base % mod;
53         b >>= 1;
54     }
55     return ans % mod;
56 }
57
58 ll num[maxn]; // 所有质数的循环节
59 ll f[maxn]; // 斐波那契数列
60
61 void Fib_Cyclic_node()
62 {
63     num[1] = 3;
64     for(int i = 2; i <= p; i++) // 找每个素数的循环节num[1~p]
65     {
66         f[1] = 1;
67         f[2] = 2;
68         int x = 3;
69         while(true)
70         {
71             f[x] = f[x - 1] + f[x - 2];
72             f[x] %= prime[i];
73             if(f[x] == 1 && f[x - 1] == 0) // f[x] % prime[i] == f[1]
74                 break;
75             x++;
76         }
77         num[i] = x;
78     }
79
80     ll n;
81     cin >> n; // 如果是质数，那循环节就是num[n]； 如果是合数，那循环节就是n的素因子的最小公倍数
82     ll ans = 1;
83     ll x;
84     for(int i = 1; i <= p; i++)
85     {
86         if(n % prime[i] == 0)
87         {
88             x = 0;
89             while(n % prime[i] == 0)
90                 {

```

```

91         n /= prime[i];
92         x++;
93     }
94 }
95 ll k = num[i] * quick_pow(prime[i], x - 1);
96 ans = ans * k / gcd(ans, k); // 最小公倍数
97 }
98 cout << ans << endl; // 最小循环节
99 }
100
101 // 广义斐波那契循环节
102
103 // fib(n) = a * fib(n - 1) + b * fib(n - 2)
104 // fib(1) = c    fib(2) = d
105 // 求f(n) mod p的循环节
106 // c = a * a - 4b是模p的二次剩余时枚举p-1的因子, 否则枚举(p+1)(p-1)的因子
107
108 typedef long long ll;
109 typedef long double ld;
110 typedef pair<int, int> pdd;
111
112 #define INF 0x7f7f7f
113 #define mem(a,b) memset(a , b , sizeof(a))
114 #define FOR(i, x, n) for(int i = x; i <= n; i++)
115
116 ll fac[2][505];
117 ll cnt, ct;
118
119 ll pri[6] = {2, 3, 7, 109, 167, 500000003};
120 ll num[6] = {4, 2, 1, 2, 1, 1};
121
122 const ll mod = 1e9 + 7;
123 const int maxn = 5e6 + 10;
124
125 struct Matrix{
126     ll m[2][2];
127 };
128
129 Matrix A;
130
131 Matrix I = {1, 0, 0, 1}; // 单位矩阵
132
133 Matrix multi(Matrix a, Matrix b) // 矩阵乘法
134 {
135     Matrix C;
136     for(int i = 0; i < 2; i++)
137     {
138         for(int j = 0; j < 2; j++)
139         {
140             C.m[i][j] = 0;
141             for(int k = 0; k < 2; k++)
142             {
143                 C.m[i][j] = (C.m[i][j] + a.m[i][k] * b.m[k][j] % mod) % mod;
144             }
145             C.m[i][j] %= mod;
146         }
147     }
148     return C;
149 }

```

```

150
151 Matrix quick_Matrix(Matrix a, ll b) // 矩阵快速幂
152 {
153     Matrix ans = I, base = a;
154     while(b)
155     {
156         if(b & 1)
157         {
158             ans = multi(a, base);
159         }
160         base = multi(base, base);
161         b >>= 1;
162     }
163     return ans;
164 }
165
166 ll quick_pow(ll a, ll b) ;
167
168 ll legendre(ll a, ll p) // 勒让德符号 = {1, -1, 0}
169 {
170     ll k = quick_pow(a, (p - 1) >> 1);
171     if(k == 1)
172         return 1;
173     else
174         return -1;
175 }
176
177 void DFS(int dept, ll product = 1)
178 {
179     if(dept == cnt)
180     {
181         fac[1][ct++] = product;
182         return;
183     }
184     for(int i=0; i<=num[dept]; i++)
185     {
186         DFS(dept+1, product);
187         product *= pri[dept];
188     }
189 }
190
191 bool Fib_node(Matrix a, ll n) // n是否为循环节
192 {
193     Matrix ans = quick_Matrix(a, n);
194     return (ans.m[0][0] == 1 && ans.m[0][1] == 0 && ans.m[1][0] == 0 && ans.m[1][1] ==
195     1); // 是否为单位矩阵I
196
197 ll Fib_Cyclic_node(ll a, ll b, ll c, ll d) // 广义斐波那契循环节斐波那契循环节
198 {
199     fac[0][0] = 1;
200     fac[0][1] = 2;
201     fac[0][3] = 500000003;
202     fac[0][3] = 1000000006;
203     ll c = a * a - 4 * b;
204     A.m[0][0] = a;
205     A.m[0][1] = b;
206     A.m[1][0] = 1;
207     A.m[1][1] = 0;

```

```

208     if(legendre(c, mod) == 1) // c是否为1e9+7的二次剩余
209     {
210         for(int i = 0; i < 4; i++)
211         {
212             if(Fib_node(A, fac[0][i]))
213                 return fac[0][i];
214         }
215     }
216     else
217     {
218         ct = 0;
219         cnt = 6;
220         DFS(0, 1);
221         sort(fac[1], fac[1] + ct);
222         for(int i = 0; i < ct; i++)
223         {
224             if(Fib_node(A, fac[1][i]))
225                 return fac[1][i];
226         }
227     }
228 }
229 }
230
231 int main()
232 {
233     ll a, b, c, d;
234     cin >> a >> b >> c >> d;
235     ll n = Fib_Cyclic_node(a, b, c, d); // 广义斐波那契循环节循环节长度
236     cout << n << endl;
237 }

```

6.46 Matrix

```

1  template <typename T>
2  struct Matrix {
3      const int n;
4      std::vector<std::vector<T>> mat;
5
6      Matrix(int n = 0) : n(n), mat(n, std::vector<T>(n)) {}
7      Matrix(const std::vector<std::vector<T>> &rhs) : n(n) {
8          mat = std::move(rhs);
9      }
10     Matrix(const Matrix<T> &rhs) : n(n) {
11         mat = std::move(rhs.mat);
12     }
13     void identify() {
14         for (int i = 0; i < n; i++) {
15             for (int j = 0; j < n; j++) {
16                 mat[i][j] = (i == j);
17             }
18         }
19     }
20     T getVal(const int& i, const int& j) { return mat[i][j]; }
21     T size() { return n; }
22     const Matrix operator+(const Matrix& rhs) {
23         Matrix ret(n);
24         for (int i = 0; i < n; i++) {
25             for (int j = 0; j < n; j++) {

```



```

26         ret.mat[i][j] = mat[i][j] + rhs.mat[i][j];
27     }
28 }
29 return ret;
30 }
31 const Matrix operator-(const Matrix& rhs) {
32     Matrix ret(n);
33     for (int i = 0; i < n; i++) {
34         for (int j = 0; j < n; j++) {
35             ret[i][j] = mat[i][j] - rhs[i][j];
36         }
37     }
38     return ret;
39 }
40 const Matrix operator*(const Matrix &rhs) {
41     Matrix ret(n);
42     for (int i = 0; i < n; i++) {
43         for (int j = 0; j < n; j++) {
44             for (int k = 0; k < n; k++) {
45                 ret.mat[i][j] += mat[i][k] * rhs.mat[k][j];
46             }
47         }
48     }
49     return ret;
50 }
51 const Matrix operator+() { return *this; }
52 const Matrix operator-() {
53     Matrix ret(n);
54     for (int i = 0; i < n; i++) {
55         for (int j = 0; j < n; j++) {
56             ret.mat[i][j] = -mat[i][j];
57         }
58     }
59     return ret;
60 }
61 Matrix &operator+=(const Matrix &rhs) {
62     for (int i = 0; i < n; i++) {
63         for (int j = 0; j < n; j++) {
64             mat[i][j] += rhs.mat[i][j];
65         }
66     }
67     return *this;
68 }
69 Matrix &operator-=(const Matrix &rhs) {
70     for (int i = 0; i < n; i++) {
71         for (int j = 0; j < n; j++) {
72             mat[i][j] -= rhs.mat[i][j];
73         }
74     }
75     return *this;
76 }
77 const Matrix operator*=(const Matrix &rhs) {
78     Matrix ret(n);
79     for (int i = 0; i < n; i++) {
80         for (int j = 0; j < n; j++) {
81             for (int k = 0; k < n; k++) {
82                 ret.mat[i][j] += mat[i][k] * rhs.mat[k][j];
83             }
84         }

```

```

85     }
86     return ret;
87 }
88 };
89
90 template <typename T>
91 Matrix power(Matrix<T> a, T b) {
92     Matrix<T> res(2);
93     res.mat[0][0] = res.mat[1][1] = 1;
94     for (; b /= 2; a *= a) {
95         if (b % 2) {
96             res *= a;
97         }
98     }
99     return res;
100 }

```

6.47 拉格朗日插值求和

```

1 namespace polysum {
2 #define rep(i, a, n) for (int i=a;i<n;i++)
3 #define per(i, a, n) for (int i=n-1;i>=a;i--)
4 const int D = 1010000; //可能需要用到的最高次
5 ll a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], c[D], num[D];
6
7 ll powmod(ll a, ll b) {
8     ll res = 1;
9     a %= mod;
10    assert(b >= 0);
11    for (; b; b >>= 1) {
12        if (b & 1) res = res * a % mod;
13        a = a * a % mod;
14    }
15    return res;
16 }
17
18 //函数用途: 给出数列的 (d+1) 项, 其中d为最高次方项
19 //求出数列的第n项, 数组下标从0开始
20 ll calcn(int d, ll *a, ll n) { // a[0].. a[d] a[n]
21     if (n <= d) return a[n];
22     p1[0] = p2[0] = 1;
23     rep(i, 0, d + 1) {
24         ll t = (n - i + mod) % mod;
25         p1[i + 1] = p1[i] * t % mod;
26     }
27     rep(i, 0, d + 1) {
28         ll t = (n - d + i + mod) % mod;
29         p2[i + 1] = p2[i] * t % mod;
30     }
31     ll ans = 0;
32     rep(i, 0, d + 1) {
33         ll t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
34         if ((d - i) & 1) ans = (ans - t + mod) % mod;
35         else ans = (ans + t) % mod;
36     }
37     return ans;
38 }
39

```

```

40 void init(int M) {///用到的最高次
41     f[0] = f[1] = g[0] = g[1] = 1;
42     rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
43     g[M + 4] = powmod(f[M + 4], mod - 2);
44     per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod;///费马小定理筛逆元
45 }
46
47 ///函数用途: 给出数列的 (m+1) 项, 其中m为最高次方
48 ///求出数列的前 (n-1) 项的和 (从第0项开始)
49 ll polysum(ll m, ll *a, ll n) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]
50     for (int i = 0; i <= m; i++) b[i] = a[i];
51
52     ///前n项和, 其最高次幂加1
53     b[m + 1] = calcn(m, b, m + 1);
54     rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
55     return calcn(m + 1, b, n - 1);
56 }
57
58 ll qpolysum(ll R, ll n, ll *a, ll m) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
59     if (R == 1) return polysum(n, a, m);
60     a[m + 1] = calcn(m, a, m + 1);
61     ll r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;
62     h[0][0] = 0;
63     h[0][1] = 1;
64     rep(i, 1, m + 2) {
65         h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
66         h[i][1] = h[i - 1][1] * r % mod;
67     }
68     rep(i, 0, m + 2) {
69         ll t = g[i] * g[m + 1 - i] % mod;
70         if (i & 1) p3 = ((p3 - h[i][0] * t) % mod + mod) % mod, p4 = ((p4 - h[i][1]
71 * t) % mod + mod) % mod;
72         else p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
73     }
74     c = powmod(p4, mod - 2) * (mod - p3) % mod;
75     rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
76     rep(i, 0, m + 2) C[i] = h[i][0];
77     ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
78     if (ans < 0) ans += mod;
79     return ans;
80 }
81
82 ll solve(ll n, int k) {
83     init(k + 10);
84     for (int i = 0; i <= k + 1; i++) num[i] = powmod((ll) i + 1, k);
85     ll ans = polysum(k + 1, num, n) % mod;
86     return ans;
87 }
88
89 void solve() {
90     ll n;
91     int k;
92     cin >> n >> k;
93     cout << polysum::solve(n, k) << endl;
94 }

```

6.48 SolveLinearSystem

```

1  bool IsZero(double v) {
2      return abs(v) < 1e-9;
3  }
4
5  enum GAUSS_MODE {
6      DEGREE, ABS
7  };
8
9  template <typename T>
10 void GaussianElimination(std::vector<std::vector<T>>& a, int limit, GAUSS_MODE mode =
    ABS) {
11     if (a.empty() || a[0].empty()) {
12         return;
13     }
14     int h = static_cast<int>(a.size());
15     int w = static_cast<int>(a[0].size());
16     for (int i = 0; i < h; i++) {
17         assert(w == static_cast<int>(a[i].size()));
18     }
19     assert(limit <= w);
20     std::vector<int> deg(h);
21     for (int i = 0; i < h; i++) {
22         for (int j = 0; j < w; j++) {
23             deg[i] += !IsZero(a[i][j]);
24         }
25     }
26     int r = 0;
27     for (int c = 0; c < limit; c++) {
28         int id = -1;
29         for (int i = r; i < h; i++) {
30             if (!IsZero(a[i][c]) && (id == -1 || (mode == DEGREE && deg[i] < deg[id])
|| (mode == ABS && abs(a[id][c]) < abs(a[i][c])))) {
31                 id = i;
32             }
33         }
34         if (id == -1) {
35             continue;
36         }
37         if (id > r) {
38             std::swap(a[r], a[id]);
39             std::swap(deg[r], deg[id]);
40             for (int j = c; j < w; j++) {
41                 a[id][j] = -a[id][j];
42             }
43         }
44         std::vector<int> nonzero;
45         for (int j = c; j < w; j++) {
46             if (!IsZero(a[r][j])) {
47                 nonzero.push_back(j);
48             }
49         }
50         T inv_a = 1 / a[r][c];
51         for (int i = r + 1; i < h; i++) {
52             if (IsZero(a[i][c])) {
53                 continue;
54             }
55             T coeff = -a[i][c] * inv_a;

```

```

56         for (int j : nonzero) {
57             if (!IsZero(a[i][j])) deg[i]--;
58             a[i][j] += coeff * a[r][j];
59             if (!IsZero(a[i][j])) deg[i]++;
60         }
61     }
62     ++r;
63 }
64 for (r = h - 1; r >= 0; r--) {
65     for (int c = 0; c < limit; c++) {
66         if (!IsZero(a[r][c])) {
67             T inv_a = 1 / a[r][c];
68             for (int i = r - 1; i >= 0; i--) {
69                 if (IsZero(a[i][c])) {
70                     continue;
71                 }
72                 T coeff = -a[i][c] * inv_a;
73                 for (int j = c; j < w; j++) {
74                     a[i][j] += coeff * a[r][j];
75                 }
76             }
77             break;
78         }
79     }
80 }
81 }
82
83 template <typename T>
84 std::vector<T> SolveLinearSystem(std::vector<std::vector<T>> a, const std::vector<T>& b
85     , int w) {
86     int h = static_cast<int>(a.size());
87     assert(h == static_cast<int>(b.size()));
88     if (h > 0) {
89         assert(w == static_cast<int>(a[0].size()));
90     }
91     for (int i = 0; i < h; i++) {
92         a[i].push_back(b[i]);
93     }
94     GaussianElimination(a, w);
95     std::vector<T> x(w, 0);
96     for (int i = 0; i < h; i++) {
97         for (int j = 0; j < w; j++) {
98             // when IsZero(a[i][j]) is no solution
99             if (!IsZero(a[i][j])) {
100                 x[j] = a[i][w] / a[i][j];
101                 break;
102             }
103         }
104     }
105     return x;
106 }

```

6.49 矩阵求逆

```

1 //原始矩阵A[0, n - 1][0, n - 1]
2 //右边一个单位阵I, 在a[0, n - 1][n, (n << 1) - 1]
3 //将左边A变成单位阵时, 右边的I变为A^-1
4

```

```

5 ll a[MAX][MAX];
6 bool Gauss(int n) {
7     for (int i = 0, r; i < n; i++) {
8         r = i;
9         for (int j = i + 1; j < n; j++)
10             if (a[j][i] > a[r][i]) r = j;
11         if (r != i) swap(a[i], a[r]);
12         if (!a[i][i]) return false; //无解
13
14         ll inv = qpow(a[i][i], mod - 2);
15         for (int k = 0; k < n; k++) {
16             if (k == i) continue;
17             ll t = a[k][i] * inv % mod;
18             for (int j = i; j < (n << 1); j++)
19                 a[k][j] = (a[k][j] - t * a[i][j] % mod + mod) % mod;
20         }
21         for (int j = 0; j < (n << 1); j++) a[i][j] = a[i][j] * inv % mod;
22     }
23     return true;
24 }
25
26 int main() {
27     scanf("%d", &n);
28     for (int i = 0; i < n; i++) {
29         a[i][i + n] = 1;
30         for (int j = 0; j < n; j++)
31             scanf("%lld", &a[i][j]);
32     }
33     if(Gauss(n)) {
34         for(int i = 0; i < n; i++) {
35             for(int j = n; j < n * 2; j++) {
36                 cout << a[i][j] << " ";
37             }
38             cout << endl;
39         }
40     }
41     else cout << "No Solution" << endl;
42 }

```

6.50 eraseLinearBasis

```

1 // 离线删除操作，维护线性基中每个元素的最晚删除时间。
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 typedef long long ll;
7 const int maxl = 60;
8
9 struct LinearBasis {
10     ll a[maxl + 10], tim[maxl + 10];
11     int n, size; // 每个相同异或值有 $2^{\{n-size\}}$ 个
12     vector<ll> v;
13
14     LinearBasis() {
15         memset(a, 0, sizeof(a));
16         size = n = 0;
17         v.clear();
18     }
19 }

```

```

18     }
19
20     void insert(ll x, ll t) {
21         n++;
22         for(int i = maxl; i >= 0; i--) {
23             if(!(x >> i & 1)) continue;
24             if(a[i]) {
25                 if(t > tim[i]) swap(t, tim[i]), swap(x, a[i]);
26                 x ^= a[i];
27             }
28             else {
29                 ++size;
30                 a[i] = x; tim[i] = t;
31                 return;
32             }
33         }
34     }
35
36     void erase(ll t) {
37         for(int i = maxl; i >= 0; i--) {
38             if(tim[i] == t) {
39                 a[i] = tim[i] = 0; --size;
40                 return;
41             }
42         }
43     }
44 };
45
46 int main() {
47     LinearBasis lb;
48     int n, m; cin >> n >> m;
49     vector<ll> opt(n + 10), a(n + 10), del(n + 10), pre(n + 10);
50     for(int i = 1; i <= m; i++) {
51         cin >> opt[i] >> a[i];
52         if(opt[i] == 1) pre[a[i]] = i, del[i] = m + 1;
53         else del[pre[a[i]]] = i;
54     }
55     ll ans = 0;
56     for(int i = 1; i <= m; i++) {
57         if(opt[i] == 1) lb.insert(a[i], del[i]);
58         else lb.erase(i);
59         ans ^= 1ll << (lb.n - lb.size);
60     }
61     cout << ans << endl;
62 }

```

6.51 simpleLinearBasis

```

1  template<class Info>
2  struct LinearBasis {
3      const int n;
4      int size;
5      long long num;
6      // 每个异或值都相同的个数都为 $2^{n-r}$ , 所以不同的异或值有 $2^r$ 个.
7      const int maxl = 61;
8      std::vector<long long> a, v;
9      LinearBasis(int n) : n(n), size(0), a(maxl) {}
10     LinearBasis(std::vector<Info> init) : LinearBasis(init.size()) {

```

```

11     auto insert = [&](long long t) {
12         for (int i = maxl - 1; i >= 0; --i) {
13             if (!(t >> i & 1)) continue;
14             if (a[i]) t ^= a[i];
15             else {
16                 ++size;
17                 // Rebuild
18                 for (int j = i - 1; j >= 0; --j) if (t >> j & 1) t ^= a[j];
19                 for (int j = i + 1; j < maxl; ++j) if (a[j] >> i & 1) a[j] ^= t;
20                 //
21                 a[i] = t;
22                 return;
23             }
24         }
25     };
26     for(int i = 0; i < n; i++) insert(init[i]);
27     auto basis = [&]() {
28         for (int i = 0; i < maxl; ++i) if (a[i]) v.push_back(i);
29     };
30     basis();
31     num = 1LL << size;
32 }
33
34 // 查询能否xor出x这个数
35 bool find(long long x) {
36     for(int i = maxl - 1; i >= 0; i--) {
37         if(x >> i & 1) {
38             if(!a[i]) return false;
39             x ^= a[i];
40         }
41     }
42     return true;
43 }
44
45 // 查询异或最大值
46 long long askmax() {
47     long long ans = 0;
48     for(int i = maxl - 1; i >= 0; i--) ans = max(ans, ans ^ a[i]);
49     return ans;
50 }
51
52 // 查询异或最小值
53 long long askmin() {
54     if((int) v.size() < n) return 0;
55     for(int i = 0; i < maxl; i++) if(a[i]) return a[i];
56     return 0;
57 }
58
59 // 查询异或第k小
60 long long askmink(long long x) {
61     if(v.size() != n) x--;
62     if(!x) return 0;
63     if(x >= (1LL << v.size())) return -1;
64     long long ans = 0;
65     for(int i = 0; i < (int) v.size(); i++) {
66         if(x >> i & 1) ans ^= a[v[i]];
67     }
68     return ans;
69 }

```



```

70
71     long long rank(long long x) {
72         long long ret = 0;
73         for (int i = 0; i < (int) v.size(); ++i) if (x >> v[i] & 1) ret += 1LL << i;
74         return ret;
75     }
76 };

```

6.52 intervalModifyLinearBasis

```

1  #include<bits/stdc++.h>
2  #define maxn 200005
3  using namespace std;
4  struct Base{
5      int a[31],cnt;
6      void clear(){memset(a,0,sizeof a),cnt=0;}
7      void ins(int x){
8          if(cnt==30) return;
9          for(int i=29;i>=0&&x;i--) if(x>>i&1){
10             if(a[i]) x^=a[i];
11             else {a[i]=x,cnt++;return;}
12         }
13     }
14     void merge(const Base &B){
15         for(int i=29;i>=0;i--) if(B.a[i]) ins(B.a[i]);
16     }
17 }t[maxn<<2];
18 int n,m,a[maxn],b[maxn];
19 int arr[maxn];
20 void upd(int i,int v){for(;i<=n;i+=i&-i) arr[i]^=v;}
21 int qxor(int i){int s=0;for(;i-=i&-i) s^=arr[i];return s;}
22 void upd(int i){t[i]=t[i<<1],t[i].merge(t[i<<1|1]);}
23 void build(int i,int l,int r){
24     if(l==r) return t[i].ins(b[l]);
25     int mid=(l+r)>>1;
26     build(i<<1,l,mid),build(i<<1|1,mid+1,r);
27     upd(i);
28 }
29 void mdf(int i,int l,int r,int x){
30     if(l==r) {t[i].clear(),t[i].ins(b[x]);return;}
31     int mid=(l+r)>>1;
32     x<=mid?mdf(i<<1,l,mid,x):mdf(i<<1|1,mid+1,r,x);
33     upd(i);
34 }
35 void qry(int i,int l,int r,int x,int y){
36     if(x<=l&&r<=y) return t[0].merge(t[i]);
37     int mid=(l+r)>>1;
38     if(x<=mid) qry(i<<1,l,mid,x,y);
39     if(y>mid) qry(i<<1|1,mid+1,r,x,y);
40 }
41 int main()
42 {
43     scanf("%d%d",&n,&m);
44     for(int i=1;i<=n;i++) scanf("%d",&a[i]),b[i]=a[i]^a[i-1],upd(i,b[i]);
45     build(1,1,n);
46     for(int op,l,r,x;m--;>){
47         scanf("%d%d%d",&op,&l,&r);
48         if(op==1){

```

```

49         scanf("%d",&x);
50         upd(l,x),upd(r+1,x),b[l]^=x,b[r+1]^=x;
51         mdf(1,1,n,l); if(r<n) mdf(1,1,n,r+1);
52     }
53     else{
54         t[0].clear(),t[0].ins(qxor(l)); if(l<r) qry(1,1,n,l+1,r);
55         printf("%d\n",1<t[0].cnt);
56     }
57 }
58 }

```

6.53 noIntervalModifyLinearBasis

```

1  // 扫描r, 维护线性基中每个元素的最大左端点l。与删除操作类似。
2  // 这个可以强制在线, 把每个r的线性基存下来即可。
3  #include <bits/stdc++.h>
4  using namespace std;
5
6  const int N = 2e5 + 10;
7  struct node {
8      int l, r, id;
9      bool operator < (const node &p) const {
10         return r < p.r;
11     }
12 }q[N];
13
14 const int maxl = 60;
15
16 struct LinearBasis {
17     ll a[maxl + 10], pos[maxl + 10];
18     int n, size; // 每个相同异或值有 $2^{n-size}$ 个
19     vector<ll> v;
20
21     LinearBasis() {
22         memset(a, 0, sizeof(a));
23         size = n = 0;
24         v.clear();
25     }
26
27     void insert(ll t, ll id) {
28         n++;
29         for(int i = maxl; i >= 0; i--) {
30             if(!(t >> i & 1)) continue;
31             if(a[i]) {
32                 if(id > pos[i]) swap(id, pos[i]), swap(t, a[i]);
33                 t ^= a[i];
34             }
35             else {
36                 a[i] = t;
37                 pos[i] = id;
38                 return;
39             }
40         }
41     }
42
43     int askmax(ll x) {
44         ll ans = 0;
45         for(int i = maxl; i >= 0; i--) {

```

```

46 //         if(pos[i] >= x && !(ans >> i & 1)) ans ^= a[i];
47         if(pos[i] >= x) ans = max(ans, ans ^ a[i]);
48     }
49     return ans;
50 }
51 };
52
53 // 给你n个数, 每次查询 [公式] 这个区间, 问着个区间的最大异或值。
54
55 void solve() {
56     LinearBasis lb;
57     int n; cin >> n;
58     VI a(n + 1);
59     for(int i = 1; i <= n; i++) cin >> a[i];
60     int m; cin >> m;
61     VI ans(m + 1);
62     for(int i = 1; i <= m; i++) cin >> q[i].l >> q[i].r, q[i].id = i;
63     sort(q + 1, q + m + 1);
64     for(int i = 1, j = 1; i <= n; i++) {
65         lb.insert(a[i], i);
66         for(; j <= m && q[j].r <= i; j++) ans[q[j].id] = lb.askmax(q[j].l);
67     }
68     for(int i = 1; i <= m; i++) cout << ans[i] << endl;
69 }

```

6.54 FWT

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const ll mod = 998244353;
5
6  const int N = 1e5 + 10;
7  int a[N], b[N];
8
9  inline void FWT_OR(int *f, int n, int opt) {
10     for (int o = 2, k = 1; o <= n; o <= 1, k <= 1)
11         for (int i = 0; i < n; i += o)
12             for (int j = 0; j < k; j++)
13                 f[i + j + k] = (f[i + j + k] + 1ll * f[i + j] * opt % mod + mod) % mod;
14 }
15
16 inline void FWT_AND(int *f, int n, int opt) {
17     for (int o = 2, k = 1; o <= n; o <= 1, k <= 1)
18         for (int i = 0; i < n; i += o)
19             for (int j = 0; j < k; j++)
20                 f[i + j] = (f[i + j] + 1ll * f[i + j + k] * opt % mod + mod) % mod;
21 }
22
23 inline void FWT_XOR(int *f, int n, int opt) {
24     for (int o = 2, k = 1; o <= n; o <= 1, k <= 1)
25         for (int i = 0; i < n; i += o)
26             for (int j = 0; j < k; j++) {
27                 ll a0 = f[i + j], a1 = f[i + j + k];
28                 f[i + j] = (a0 + a1) % mod * opt % mod;
29                 f[i + j + k] = (a0 - a1 + mod) % mod * opt % mod;
30             }
31 }

```

```

32
33 inline void mul_OR(int *a, int *b, int n) {
34     FWT_OR(a, n, 1); FWT_OR(b, n, 1);
35     for(int i = 0; i < n; i++) a[i] = a[i] * b[i] % mod;
36     FWT_OR(a, n, -1);
37 }
38
39 inline void mul_AND(int *a, int *b, int n) {
40     FWT_AND(a, n, 1); FWT_AND(b, n, 1);
41     for(int i = 0; i < n; i++) a[i] = a[i] * b[i] % mod;
42     FWT_AND(a, n, -1);
43 }
44
45 ll quick_pow(ll a, ll b) ;
46
47 inline void mul_XOR(int *a, int *b, int n) {
48     ll inv2 = quick_pow(mod, mod - 2);
49     FWT_XOR(a, n, 1); FWT_XOR(b, n, 1);
50     for(int i = 0; i < n; i++) a[i] = a[i] * b[i] % mod;
51     FWT_XOR(a, n, inv2);
52 }
53
54 int main() {
55     int n;
56     cin >> n;
57     n = 1 << n;
58     for(int i = 0; i < n; i++) cin >> a[i];
59     for(int i = 0; i < n; i++) cin >> b[i];
60
61     mul_OR(a, b, n);
62     mul_AND(a, b, n);
63     mul_XOR(a, b, n);
64
65
66 }

```

6.55 cdq 分治 FFT

```

1 // hdu 7054
2 // 求解 $(1+x^{a_1})(1+x^{a_2})\dots(1+x^{a_n})$ 
3 //  $\sum_{i=1}^n a_i \leq 1e6$ .
4
5 // 可以 $f[i][j]$ , 前 $i$ 个数的和为 $j$ 的方案数, 可以用生成函数转换, 并用多项式求解, 同时分治FFT优化。
6
7 const int N = 1e5 + 10;
8 int tr[N];
9 int getLen(int n) ;
10
11 void FFT(Complex *A, int len) ;
12
13 inline void MTT(ll *x, ll *y, ll *z, int len) ;
14
15 struct Poly {
16     ll *p;
17     int len;
18     void init(int len) {
19         p = a + cnt;
20         this->len = len;

```

```

21     for(int i = 0; i <= len; i++) p[i] = read();
22     cnt += len + 1;
23 }
24
25 void mul(const Poly b) {
26     static ll x[N], y[N];
27     int LEN = getLen(len + b.len);
28     for(int i = 0; i <= len; i++) x[i] = p[i];
29     for(int i = 0; i <= b.len; i++) y[i] = b.p[i];
30     for(int i = len + 1; i <= LEN; i++) x[i] = 0;
31     for(int i = b.len + 1; i <= LEN; i++) y[i] = 0;
32     MTT(x, y, p, LEN);
33     this->len += b.len;
34     // 不知道为啥要两倍, 可能会有不为0的情况, 管他呢
35     for(int i = len + 1; i <= 2 * LEN; i++) p[i] = 0;
36     for(int i = 0; i <= LEN; i++) x[i] = y[i] = 0;
37 }
38 };
39
40 Poly cdq(int l, int r) {
41     Poly res;
42     if(l == r) res.init(len); // 长度
43     else {
44         int mid = (l + r) / 2;
45         res = cdq(l, mid);
46         res.mul(cdq(mid + 1, r));
47     }
48     return res;
49 }
50
51 void solve() {
52     mem(a, 0);
53     int n = read();
54     cnt = 0;
55     ll ans = 1;
56     Poly res = cdq(1, n);
57     for(int i = 0; i < n; i++) cout << res.p[i] << " ";
58 }

```

6.56 求逆分治 FFT

```

1 // f[i] = \sum_{j=1}^i f[i-j] * g[j]
2 // g相同, 可以用多项式求逆
3 #include <bits/stdc++.h>
4 using namespace std;
5 typedef long long ll;
6 const double PI = acos(-1);
7 const int N = 1e5 + 10;
8
9 struct Complex {
10     double x, y;
11     Complex(double a = 0, double b = 0): x(a), y(b) {}
12     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
13     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
14     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
        rhs.y + y * rhs.x); }
15     Complex conj() { return Complex(x, -y); }
16 } w[N];

```

```

17
18 ll mod;
19 int tr[N];
20 ll F[N], G[N];
21
22 ll quick_pow(ll a, ll b) ;
23
24 int getLen(int n) ;
25
26 void FFT(Complex *A, int len) ;
27
28 inline void MTT(ll *x, ll *y, ll *z, int len) ;
29
30 void Get_Inv(ll *f, ll *g, int n) ;
31
32 void fenziFFT(ll *f, ll *g, int n) {
33     static ll a[N];
34     for(int i = 1; i < n; i++) a[i] = (mod - f[i]) % mod;
35     a[0] = 1;
36     Get_Inv(a, g, n);
37
38     for(int i = 0; i < n; i++) {
39         a[i] = 0;
40     }
41 }
42
43 int main() {
44     int n;
45     cin >> n;
46     for(int i = 1; i < n; i++) cin >> G[i];
47     fenziFFT(G, F, n);
48
49     for(int i = 0; i < n; i++) cout << F[i] << " ";
50 }

```

6.57 多项式求逆

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const double PI = acos(-1);
5 const int N = 1e5 + 10;
6
7 struct Complex {
8     double x, y;
9     Complex(double a = 0, double b = 0): x(a), y(b) {}
10     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
11     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
12     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
        rhs.y + y * rhs.x); }
13     Complex conj() { return Complex(x, -y); }
14 } w[N];
15
16 ll mod;
17 int tr[N];
18 ll F[N], G[N];
19
20 ll quick_pow(ll a, ll b) ;

```

```

21
22 int getLen(int n) {
23     int len = 1; while (len < (n << 1)) len <= 1;
24     for (int i = 0; i < len; i++) tr[i] = (tr[i] >> 1) >> 1 | (i & 1 ? len >> 1 : 0);
25     for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 *
        PI * i / len));
26     return len;
27 }
28
29 void FFT(Complex *A, int len) ;
30
31 inline void MTT(ll *x, ll *y, ll *z, int len) ;
32
33 void Get_Inv(ll *f, ll *g, int n) {
34     if(n == 1) { g[0] = quick_pow(f[0], mod - 2); return ; }
35     Get_Inv(f, g, (n + 1) >> 1);
36
37     int len = getLen(n);
38     static ll c[N];
39     for(int i = 0; i < len; i++) c[i] = i < n ? f[i] : 0;
40     MTT(c, g, c, len); MTT(c, g, c, len);
41     for(int i = 0; i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
42     for(int i = n; i < len; i++) g[i] = 0;
43     for(int i = 0; i < len; i++) c[i] = 0;
44 }
45
46 int main() {
47     int n;
48     cin >> n;
49     for(int i = 0; i < n; i++) cin >> F[i];
50     Get_Inv(F, G, n);
51     for(int i = 0; i < n; i++) cout << G[i] << " ";
52 }

```

6.58 多项式快速幂

```

1 // f(x)^k, k较小时, 可取, 每次FFT之后长度*2
2 #define maxfft 1048576+5
3
4 struct cp {
5     double a, b;
6     cp operator+(const cp &o) const { return (cp) {a + o.a, b + o.b}; }
7     cp operator-(const cp &o) const { return (cp) {a - o.a, b - o.b}; }
8     cp operator*(const cp &o) const { return (cp) {a * o.a - b * o.b, b * o.a + a * o.b}; }
9     cp operator*(const double &o) const { return (cp) {a * o, b * o}; }
10    cp operator!() const { return (cp) {a, -b}; }
11 } w[maxfft];
12
13 int pos[maxfft];
14
15 void fft_init(int len) {
16     int j = 0;
17     while ((1 << j) < len) j++;
18     j--;
19     for (int i = 0; i < len; i++)
20         pos[i] = pos[i >> 1] >> 1 | ((i & 1) << j);
21 }

```

```

22
23 void fft(cp *x, int len, int sta) {
24     for (int i = 0; i < len; i++)
25         if (i < pos[i]) swap(x[i], x[pos[i]]);
26     w[0] = (cp) {1, 0};
27     for (unsigned i = 2; i <= len; i <= 1) {
28         cp g = (cp) {cos(2 * PI / i), sin(2 * PI / i) * sta};
29         for (int j = i >> 1; j >= 0; j -= 2) w[j] = w[j >> 1];
30         for (int j = 1; j < i >> 1; j += 2) w[j] = w[j - 1] * g;
31         for (int j = 0; j < len; j += i) {
32             cp *a = x + j, *b = a + (i >> 1);
33             for (int l = 0; l < i >> 1; l++) {
34                 cp o = b[l] * w[l];
35                 b[l] = a[l] - o;
36                 a[l] = a[l] + o;
37             }
38         }
39     }
40     if (sta == -1) for (int i = 0; i < len; i++) x[i].a /= len, x[i].b /= len;
41 }
42
43 cp x[maxfft], y[maxfft], z[maxfft];
44
45 void FFT(int *a, int *b, int n, int m, int *c) {
46     int len = 1;
47     while (len < (n + m) >> 1) len <= 1;
48     fft_init(len);
49     for (int i = n / 2; i < len; i++) x[i].a = x[i].b = 0;
50     for (int i = m / 2; i < len; i++) y[i].a = y[i].b = 0;
51     for (int i = 0; i < n; i++) (i & 1 ? x[i >> 1].b : x[i >> 1].a) = a[i];
52     for (int i = 0; i < m; i++) (i & 1 ? y[i >> 1].b : y[i >> 1].a) = b[i];
53     fft(x, len, 1), fft(y, len, 1);
54     for (int i = 0; i < len / 2; i++) {
55         int j = len - 1 & len - i;
56         z[i] = x[i] * y[i] - (x[i] - !x[j]) * (y[i] - !y[j]) * (w[i] + (cp) {1, 0}) *
0.25;
57     }
58     for (int i = len / 2; i < len; i++) {
59         int j = len - 1 & len - i;
60         z[i] = x[i] * y[i] - (x[i] - !x[j]) * (y[i] - !y[j]) * ((cp) {1, 0} - w[i ^ len
>> 1]) * 0.25;
61     }
62     fft(z, len, -1);
63     for (int i = 0; i < n + m; i++)
64         if (i & 1) c[i] = (int) (z[i >> 1].b + 0.5) ? 1 : 0;
65         else c[i] = (int) (z[i >> 1].a + 0.5) ? 1 : 0;
66 }
67
68 int n, k, f[maxfft], g[maxfft];
69
70 void Pow(int *f, int len, int k, int *g) {
71     g[0] = 1;
72     while (k) {
73         if (k & 1) FFT(g, f, len, len, g);
74         FFT(f, f, len, len, f);
75         k >>= 1;
76         len <= 1;
77     }
78 }

```


6.59 多项式除法、取模

```

1
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 const double PI = acos(-1);
6 const int N = 3e5 + 10;
7 ll mod;
8
9 struct Complex {
10     double x, y;
11     Complex(double a = 0, double b = 0): x(a), y(b) {}
12     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
13     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
14     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
        rhs.y + y * rhs.x); }
15     Complex conj() { return Complex(x, -y); }
16 } w[N];
17
18 int tr[N];
19 ll F[N], G[N], D[N], R[N];
20
21 ll quick_pow(ll a, ll b) ;
22
23 int getLen(int n) ;
24
25 void FFT(Complex *A, int len) ;
26
27 inline void MTT(ll *x, ll *y, ll *z, int len) ;
28
29 void Get_Inv(ll *f, ll *g, int n) ;
30
31 void rever(ll *f, int n) { for(int i = 0, j = n - 1; i < j; i++, j--) swap(f[i], f[j]); }
32
33 void Get_Div(ll *f, ll *g, ll *d, ll *r, int n, int m) {
34     static ll a[N], b[N], invb[N];
35     for(int i = 0; i < n; i++) a[i] = f[i];
36     for(int i = 0; i < m; i++) b[i] = g[i];
37     rever(a, n); rever(b, m);
38     //for(int i = 0; i < n - m + 1; i++) b[i] = i < m ? b[i] : 0;
39     Get_Inv(b, invb, n - m + 1);
40
41     int len = getLen(n);
42     MTT(a, invb, a, len);
43     rever(a, n - m + 1);
44     for(int i = 0; i < len; i++) d[i] = i < n - m + 1 ? a[i] : 0;
45     MTT(g, d, b, len);
46     for(int i = 0; i < m; i++) { r[i] = (f[i] - b[i] + mod) % mod; }
47
48     for(int i = m; i < len; i++) r[i] = 0;
49     for(int i = 0; i < len; i++) a[i] = b[i] = invb[i] = 0;
50 }
51
52 int main() {
53     int n, m;
54     cin >> n >> m;
55     for(int i = 0; i < n; i++) { cin >> F[i]; }

```

```

56     for(int i = 0; i < m; i++) { cin >> G[i]; }
57     Get_Div(F, G, D, R, n, m);
58
59     for(int i = 0; i < n - m + 1; i++) cout << D[i] << " ";
60     cout << endl;
61     for(int i = 0; i < m - 1; i++) cout << R[i] << " ";
62     cout << endl;
63 }

```

6.60 多项式 ln-Exp-Pow

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const double PI = acos(-1);
5
6  const int N = 1e6 + 10;
7
8  ll quick_pow(ll a, ll b) {
9      ll ans = 1;
10     while(b) {
11         if(b & 1) ans = ans * a % mod;
12         a = a * a % mod;
13         b >>= 1;
14     }
15     return ans % mod;
16 }
17
18 const ll G = 3;
19 const ll invG = quick_pow(G, mod - 2);
20
21 int tr[N];
22 bool flag;
23
24 void NTT(ll *A, int len, int type) {
25     for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
26     for (int i = 2; i <= len; i <= 1) {
27         int mid = i / 2;
28         ll Wn = quick_pow(type == 1 ? G : invG, (mod - 1) / i);
29         for (int k = 0; k < len; k += i) {
30             ll w = 1;
31             for (int l = k; l < k + mid; l++) {
32                 ll t = w * A[l + mid] % mod;
33                 A[l + mid] = (A[l] - t + mod) % mod;
34                 A[l] = (A[l] + t) % mod;
35                 w = w * Wn % mod;
36             }
37         }
38     }
39     if (type == -1) {
40         ll invn = quick_pow(len, mod - 2);
41         for (int i = 0; i < len; i++)
42             A[i] = A[i] * invn % mod;
43     }
44 }
45
46 void mul(ll *a, ll *b, int len) {
47     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);

```

```

48     NTT(a, len, 1), NTT(b, len, 1);
49     for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
50     NTT(a, len, -1); NTT(b, len, -1);
51 }
52
53 int getLen(int n) {
54     int len = 1; while (len <= (n << 1)) len <<= 1;
55     return len;
56 }
57
58 void Get_Der(ll *f, ll *g, int len) { for(int i = 1; i < len; i++) g[i - 1] = f[i] * i %
    mod; g[len - 1] = 0; }
59
60 void Get_Int(ll *f, ll *g, int len) { for(int i = 1; i < len; i++) g[i] = f[i - 1] *
    quick_pow(i, mod - 2) % mod; g[0] = 0; }
61
62 void Get_Inv(ll *f, ll *g, int n) {
63     if(n == 1) { g[0] = quick_pow(f[0], mod - 2); return ; }
64     Get_Inv(f, g, (n + 1) >> 1);
65
66     int len = getLen(n);
67     static ll c[N];
68     for(int i = 0; i < len; i++) c[i] = i < n ? f[i] : 0;
69     mul(c, g, len);
70     mul(c, g, len);
71     for(int i = 0; i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
72     for(int i = n; i < len; i++) g[i] = 0;
73     for(int i = 0; i < len; i++) c[i] = 0;
74 }
75
76 void Get_Ln(ll *f, ll *g, int n) {
77     static ll a[N], b[N];
78     Get_Der(f, a, n);
79     Get_Inv(f, b, n);
80     int len = getLen(n);
81     mul(a, b, len);
82     Get_Int(a, g, len);
83     for(int i = n; i < len; i++) g[i] = 0;
84     for(int i = 0; i < len; i++) a[i] = b[i] = 0;
85 }
86
87 void Get_Exp(ll *f, ll *g, int n) {
88     if(n == 1) return (void)(g[0] = 1);
89     Get_Exp(f, g, (n + 1) >> 1);
90
91     static ll a[N];
92     Get_Ln(g, a, n);
93     a[0] = (f[0] + 1 - a[0] + mod) % mod;
94     for(int i = 1; i < n; i++) a[i] = (f[i] - a[i] + mod) % mod;
95     int len = getLen(n);
96     mul(g, a, len);
97     for(int i = n; i < len; i++) g[i] = 0;
98     for(int i = 0; i < len; i++) a[i] = 0;
99 }
100
101 void Get_Pow(ll *f, ll *g, int n, ll k1, ll k2) {
102     static ll a[N], b[N], c[N];
103     ll deg = 0; for(int i = 0; i < n && f[i] == 0; i++) ++ deg;
104     if(deg * k1 > n || (flag && deg)) return ;

```

```

105     ll f0 = f[deg], f0k = quick_pow(f0, k2), inv0 = quick_pow(f0, mod - 2);
106     for(int i = deg; i < n; i++) a[i - deg] = f[i] * inv0 % mod;
107     Get_Ln(a, b, n);
108     for(int i = 0; i < n - deg * k1; i++) b[i] = b[i] * k1 % mod;
109     Get_Exp(b, c, n);
110     deg *= k1;
111     for(int i = deg; i < n; i++) g[i] = (c[i - deg] * f0k % mod + mod) % mod;
112     for(int i = 0; i < deg; i++) g[i] = 0;
113     int len = getLen(n);
114     for(int i = n; i < len; i++) g[i] = 0;
115     for(int i = 0; i < len; i++) a[i] = b[i] = c[i] = 0;
116 }
117
118
119 ll a[N], ans[N];
120
121 void solve() {
122     int n; string s; cin >> n >> s;
123     ll k1 = 0, k2 = 0;
124     for(int i = 0; i < s.length(); i++) {
125         k1 = (k1 * 10 + s[i] - '0');
126         flag |= (k1 >= mod);
127         k1 %= mod;
128         k2 = (k2 * 10 + s[i] - '0') % (mod - 1);
129     }
130     for(int i = 0; i < n; i++) cin >> a[i];
131     Get_Pow(a, ans, n, k1, k2); // k1是底 % mod, k2是指数 % mod-1
132     for(int i = 0; i < n; i++) cout << ans[i] << (i == n - 1 ? endl : " ");
133 }

```

6.61 任意模数 MTT-拆系数法

```

1 //将多项式拆成(a1 * mod + a2) * (b1 * mod + b2)的形式
2 //=>a1 * b1 * mod ^ 2 + (a2 * b1 + a1 * b2) * mod + a2 * b2
3 //在利用DFT合并、IDFT合并, 最终只需要4次DFT即可
4 //精度10^14
5 //4倍空间
6
7 #include <bits/stdc++.h>
8 using namespace std;
9 typedef long long ll;
10 const double PI = acos(-1);
11 const int N = 1e5 + 10;
12
13 struct Complex {
14     double x, y;
15     Complex(double a = 0, double b = 0): x(a), y(b) {}
16     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
17     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
18     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
19         rhs.y + y * rhs.x); }
19     Complex conj() { return Complex(x, -y); }
20 } w[N];
21
22 int tr[N];
23 ll a[N], b[N], ans[N];
24
25 int getLen(int n) {

```

```

26     int len = 1; while (len <= n) len <<= 1;
27     for (int i = 0; i < len; i++) tr[i] = (tr[i] >> 1) >> 1 | (i & 1 ? len >> 1 : 0);
28     for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 *
    PI * i / len));
29     return len;
30 }
31
32 void FFT(Complex *A, int len) {
33     for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
34     for (int i = 2, lyc = len >> 1; i <= len; i <<= 1, lyc >>= 1)
35         for (int j = 0; j < len; j += i) {
36             Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
37             for (int k = 0; k < i >> 1; k++) {
38                 Complex tmp = *r * *p;
39                 *r = *l - tmp, *l = *l + tmp;
40                 ++l, ++r, p += lyc;
41             }
42         }
43 }
44
45 inline void MTT(ll *x, ll *y, ll *z, int len) {
46     for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
47     static Complex a[N], b[N];
48     static Complex dfta[N], dftb[N], dftc[N], dftd[N];
49
50     for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
51     for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
52     FFT(a, len), FFT(b, len);
53     for (int i = 0; i < len; i++) {
54         int j = (len - i) & (len - 1);
55         static Complex da, db, dc, dd;
56         da = (a[i] + a[j].conj()) * Complex(0.5, 0);
57         db = (a[i] - a[j].conj()) * Complex(0, -0.5);
58         dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
59         dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
60         dfta[j] = da * dc;
61         dftb[j] = da * dd;
62         dftc[j] = db * dc;
63         dftd[j] = db * dd;
64     }
65     for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
66     for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
67     FFT(a, len), FFT(b, len);
68     for (int i = 0; i < len; i++) {
69         ll da = (ll)(a[i].x / len + 0.5) % mod;
70         ll db = (ll)(a[i].y / len + 0.5) % mod;
71         ll dc = (ll)(b[i].x / len + 0.5) % mod;
72         ll dd = (ll)(b[i].y / len + 0.5) % mod;
73         z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
74     }
75 }
76
77 int main() {
78
79     int n, m;
80     scanf("%d%d%lld", &n, &m, &mod);
81     for (int i = 0; i <= n; i++) scanf("%d", &a[i]);
82     for (int i = 0; i <= m; i++) scanf("%d", &b[i]);
83

```

```

84     MTT(a, b, ans, n + m);
85     for (int i = 0; i <= n + m; i++)
86         printf("%s%d", i == 0 ? "" : " ", (ans[i] + mod) % mod);
87
88     return 0;
89 }

```

6.62 任意模数 NTT-三模数法

```

1  //要求选取的三个模数mod1 * mod2 * mod3 >= p^2*n
2  //优点是精度高, 可达10^26
3  //缺点是常数大(9次NTT), 并且还使用了龟速乘
4  //4倍空间
5
6  #include <bits/stdc++.h>
7  using namespace std;
8  typedef long long ll;
9  const int MAX = 4e5 + 10;
10
11 ll qmul(ll a, ll b, ll mod) {
12     ll res = 0;
13     while (b) {
14         if (b & 1)
15             res = (res + a) % mod;
16         a = (a << 1) % mod;
17         b >>= 1;
18     }
19     return res;
20 }
21
22 ll qpow(ll a, ll b, ll mod) {
23     ll res = 1;
24     while (b) {
25         if (b & 1) res = qmul(res, a, mod);
26         a = qmul(a, a, mod);
27         b >>= 1;
28     }
29     return res;
30 }
31
32 const ll mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, mod4 = mod1 * mod2;
33 const ll G = 3;
34 ll a[3][MAX], b[3][MAX], ans[MAX], p;
35 int tr[MAX];
36
37 void NTT(ll *A, int len, int type, ll mod) {
38     for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
39     for (int i = 2; i <= len; i <= 1) {
40         int mid = i / 2;
41         ll Wn = qpow(type == 1 ? G : qpow(G, mod - 2, mod), (mod - 1) / i, mod);
42         for (int k = 0; k < len; k += i) {
43             ll w = 1;
44             for (int l = k; l < k + mid; l++) {
45                 ll t = w * A[l + mid] % mod;
46                 A[l + mid] = (A[l] - t + mod) % mod;
47                 A[l] = (A[l] + t) % mod;
48                 w = w * Wn % mod;
49             }

```

```

50     }
51 }
52 if (type != 1) {
53     ll invn = qpow(len, mod - 2, mod);
54     for (int i = 0; i < len; i++) A[i] = A[i] * invn % mod;
55 }
56 }
57
58 void mul(int i, int len, ll mod) {
59     NTT(a[i], len, 1, mod), NTT(b[i], len, 1, mod);
60     for (int j = 0; j < len; j++) a[i][j] = a[i][j] * b[i][j] % mod;
61     NTT(a[i], len, -1, mod);
62 }
63
64 void CRT(int len) {
65     ll inv1 = qpow(mod2, mod1 - 2, mod1);
66     ll inv2 = qpow(mod1, mod2 - 2, mod2);
67     ll inv3 = qpow(mod4 % mod3, mod3 - 2, mod3);
68     for (int i = 0; i < len; i++) {
69         ll t = 0;
70         t = (t + qmul(a[0][i] * mod2 % mod4, inv1, mod4)) % mod4;
71         t = (t + qmul(a[1][i] * mod1 % mod4, inv2, mod4)) % mod4;
72         a[1][i] = t;
73         t = (a[2][i] - a[1][i] % mod3 + mod3) % mod3 * inv3 % mod3;
74         ans[i] = (mod4 % p * t % p + a[1][i] % p) % p;
75     }
76 }
77
78 void doNTT(int n) {
79     int len = 1; while (len <= n) len <<= 1;
80     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
81     mul(0, len, mod1), mul(1, len, mod2), mul(2, len, mod3);
82     CRT(len);
83 }
84
85 int main() {
86
87     int n, m;
88     scanf("%d%d%lld", &n, &m, &p);
89     for (int i = 0; i <= n; i++) {
90         ll x; scanf("%lld", &x);
91         a[0][i] = a[1][i] = a[2][i] = x % p;
92     }
93     for (int i = 0; i <= m; i++) {
94         ll x; scanf("%lld", &x);
95         b[0][i] = b[1][i] = b[2][i] = x % p;
96     }
97     doNTT(n + m);
98     for (int i = 0; i <= n + m; i++) printf("%lld ", ans[i]);
99
100     return 0;
101 }

```

6.63 多项式优化常系数齐次线性递推

```

1
2 #include <bits/stdc++.h>
3 using namespace std;

```

```

4  typedef long long ll;
5  const double PI = acos(-1);
6  const int N = 3e5 + 10;
7
8  struct Complex {
9      double x, y;
10     Complex(double a = 0, double b = 0): x(a), y(b) {}
11     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
12     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
13     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
        rhs.y + y * rhs.x); }
14     Complex conj() { return Complex(x, -y); }
15 } w[N];
16
17 ll mod;
18 int n, k, len, tr[N];
19 ll a[N], h[N], ans[N], s[N], invG[N], G[N];
20
21 ll quick_pow(ll a, ll b) ;
22
23 int getLen(int n) ;
24
25 void rever(ll *f, int n) ;
26
27 void FFT(Complex *A, int len) ;
28
29 inline void MTT(ll *x, ll *y, ll *z, int len) ;
30
31 void Get_Inv(ll *f, ll *g, int n) ;
32
33
34 void Mod(ll *f, ll *g) {
35     static ll tmp[N];
36     rever(f, k + k - 1);
37     for(int i = 0; i < k; i++) tmp[i] = f[i];
38     MTT(tmp, invG, tmp, len);
39     for(int i = k - 1; i < len; i++) tmp[i] = 0;
40     rever(f, k + k - 1); rever(tmp, k - 1);
41     MTT(tmp, G, tmp, len);
42     for(int i = 0; i < k; i++) g[i] = (f[i] + mod - tmp[i]) % mod;
43     for(int i = k; i < len; i++) g[i] = 0;
44     for(int i = 0; i < len; i++) tmp[i] = 0;
45 }
46 void fpow(int b) {
47     s[1] = 1; ans[0] = 1;
48     while(b) {
49         if(b & 1) { MTT(ans, s, ans, len); Mod(ans, ans); }
50         MTT(s, s, s, len); Mod(s, s);
51         b >>= 1;
52     }
53 }
54
55 ll DITI(ll *a, ll *h, ll *ans, int n, int k) {
56     G[k] = 1; for(int i = 1; i <= k; i++) G[k - i] = (mod - a[i]) % mod;
57     rever(G, k + 1);
58     len = getLen(k + 1);
59     Get_Inv(G, invG, k + 1);
60     for(int i = k + 1; i < len; i++) invG[i] = 0;
61     rever(G, k + 1);

```



```

62     fpow(n);
63     ll Ans = 0;
64     for(int i = 0; i < k; i++) Ans = (Ans + 1ll * h[i] * ans[i] % mod) % mod;
65     return Ans;
66 }
67
68 int main() {
69     int n, k;
70     cin >> n >> k;
71     for(int i = 1; i <= k; i++){ cin >> a[i]; a[i] = a[i] < 0 ? a[i] + mod : a[i]; }
72     for(int i = 0; i < k; i++) { cin >> h[i]; h[i] = h[i] < 0 ? h[i] + mod : h[i]; }
73
74     ll Ans = DITI(a, h, ans, n, k);
75     cout << Ans << endl;
76 }

```

6.64 FFT 加速带有通配符字符串匹配

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  //  $p[x] = \sum_{i=0}^{m-1} A[i]^3 * B[x-m+i+1] + \sum_{i=0}^{m-1} A[i] * B[x-m+i+1]^3 -$   

6  //  $2 * \sum_{i=0}^{m-1} A[i]^2 * B[x-m+i+1]^2$ 
7
8  const int N = 1e6 + 1e5;
9
10 ll qpow(ll a, ll b, ll mod) {
11     ll ans = 1;
12     while(b) {
13         if(b & 1) ans = ans * a % mod;
14         a = a * a % mod;
15         b >>= 1;
16     }
17     return ans % mod;
18 }
19
20 const ll G = 3;
21 const ll invG = qpow(G, mod - 2, mod);
22 int tr[N];
23
24 void NTT(ll *A, int len, int type) {
25     for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
26     for (int i = 2; i <= len; i <= 1) {
27         int mid = i / 2;
28         ll Wn = qpow(type == 1 ? G : invG, (mod - 1) / i, mod);
29         for (int k = 0; k < len; k += i) {
30             ll w = 1;
31             for (int l = k; l < k + mid; l++) {
32                 ll t = w * A[l + mid] % mod;
33                 A[l + mid] = (A[l] - t + mod) % mod;
34                 A[l] = (A[l] + t) % mod;
35                 w = w * Wn % mod;
36             }
37         }
38     }
39     if (type == -1) {
40         ll invn = qpow(len, mod - 2, mod);

```

```

40         for (int i = 0; i < len; i++)
41             A[i] = A[i] * invn % mod;
42     }
43 }
44
45 void mul(ll *a, ll *b, int n) {
46     int len = 1; while (len <= n) len <= 1;
47     for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
48     NTT(a, len, 1), NTT(b, len, 1);
49     for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
50     NTT(a, len, -1);
51 }
52
53 ll a1[N], a2[N], a3[N], b1[N], b2[N], b3[N];
54
55 void solve() {
56     int m, n; cin >> m >> n;
57     string s, t; cin >> t >> s;
58     for(int i = 0; i < m; i++) {
59         if(t[i] == '*') continue;
60         int temp = t[i] - 'a' + 1;
61         a1[i] = temp;
62         a2[i] = temp * temp;
63         a3[i] = temp * temp * temp;
64     }
65     for(int i = 0; i < n; i++) {
66         if(s[i] == '*') continue;
67         int temp = s[i] - 'a' + 1;
68         b1[i] = temp;
69         b2[i] = temp * temp;
70         b3[i] = temp * temp * temp;
71     }
72     reverse(a1, a1 + m);
73     reverse(a2, a2 + m);
74     reverse(a3, a3 + m);
75     mul(a1, b3, n + m);
76     mul(a2, b2, n + m);
77     mul(a3, b1, n + m);
78     vector<int> ans;
79     for(int x = m - 1; x < n; x++) {
80         ll res = a1[x] + a3[x] - a2[x] * 2;
81         if(!res) ans.push_back(x - m + 2);
82     }
83     cout << ans.size() << endl;
84     for(int i = 0; i < ans.size(); i++) cout << ans[i] << (i == ans.size() - 1 ? endl : " ");
85 }

```

6.65 FFT 加速朴素字符串匹配

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 4e5 + 10;
4
5 //  $P[x] = \sum_{i=0}^{m-1} A[i] + \sum_{i=0}^{m-1} B[x - m + i + 1] - 2 * \sum_{i=0}^{m-1} A[i] * B[x - m + i + 1]$ 
6
7 // reverse(a)

```

```

8
9 // 当串中的字符集较少时, 可以针对每个字符进行FFT, 计算每个字符对整个串的贡献
10
11 ll qpow(ll a, ll b, ll mod) ;
12
13 const ll mod = 998244353;
14 const ll G = 3;
15 const ll invG = qpow(G, mod - 2, mod);
16 int tr[N];
17
18 void NTT(ll *A, int len, int type) {
19     for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
20     for (int i = 2; i <= len; i <= 1) {
21         int mid = i / 2;
22         ll Wn = qpow(type == 1 ? G : invG, (mod - 1) / i, mod);
23         for (int k = 0; k < len; k += i) {
24             ll w = 1;
25             for (int l = k; l < k + mid; l++) {
26                 ll t = w * A[l + mid] % mod;
27                 A[l + mid] = (A[l] - t + mod) % mod;
28                 A[l] = (A[l] + t) % mod;
29                 w = w * Wn % mod;
30             }
31         }
32     }
33     if (type == -1) {
34         ll invn = qpow(len, mod - 2, mod);
35         for (int i = 0; i < len; i++)
36             A[i] = A[i] * invn % mod;
37     }
38 }
39
40 void mul(ll *a, ll *b, int n) {
41     int len = 1; while (len <= n) len <= 1;
42     for (int i = 0; i < len; i++) tr[i] = (tr[i] >> 1) >> 1 | (i & 1 ? len >> 1 : 0);
43     NTT(a, len, 1), NTT(b, len, 1);
44     for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
45     NTT(a, len, -1);
46 }
47
48 ll a[N], b[N];
49
50 void solve() {
51     string s, t; cin >> s >> t;
52     int n = s.length(), m = t.length();
53     for(int i = 0; i < n; i++) a[i] = s[i] - 'a' + 1;
54     for(int i = 0; i < m; i++) b[i] = t[i] - 'a' + 1;
55     reverse(b, b + m);
56     mul(a, b, n + m - 2);
57     double P = 0;
58     for(int i = 0; i < m; i++) {
59         P += (t[i] - 'a' + 1) * (t[i] - 'a' + 1);
60     }
61     vector<int> f(n + 1);
62     for(int i = 1; i < n; i++) {
63         f[i] = f[i - 1] + (s[i] - 'a' + 1) * (s[i] - 'a' + 1);
64     }
65     for(int x = m - 1; x < n; x++) {
66         double res;

```

```

67         if(x == m - 1) res = P + f[x] - a[x] * 2;
68         else res = P + f[x] - f[x - m] - a[x] * 2;
69         if(!res) cout << x - m + 2 << endl;
70     }
71 }

```

6.66 x 不连续、暴力插值

```

1
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 const double PI = acos(-1);
6 const int N = 3e5 + 10;
7
8 ll mod;
9 ll X[N], Y[N];
10
11 ll quick_pow(ll a, ll b) ;
12
13 ll Lagrange(ll *x, ll *y, int n, int k) {
14     ll ans = 0;
15     for(int i = 0; i < n; i++) {
16         ll s1 = 1, s2 = 1;
17         for(int j = 0; j < n; j++) {
18             if(i == j) continue;
19             s1 = s1 * (k - x[j] + mod) % mod;
20             s2 = s2 * (x[i] - x[j] + mod) % mod;
21         }
22         ans = (ans + 1ll * y[i] * s1 % mod * quick_pow(s2, mod - 2) % mod) % mod;
23     }
24     return ans;
25 }
26
27 int main() {
28     int n, k;
29     cin >> n >> k;
30     for(int i = 0; i < n; i++) cin >> X[i] >> Y[i];
31     cout << Lagrange(X, Y, n, k) << endl;
32 }

```

6.67 x 连续、前缀优化

```

1
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 const int N = 1e5 + 10;
6
7 ll mod;
8 ll F[N];
9 ll pre[N], suf[N];
10 ll fac[N], invf[N];
11
12
13 ll quick_pow(ll a, ll b) ;
14

```

```

15 void init() {
16     fac[0] = 1;
17     for(int i = 1; i < N; i++) fac[i] = fac[i - 1] * i % mod;
18     invf[N - 1] = quick_pow(fac[N - 1], mod - 2);
19     for(int i = N - 1; i >= 1; i--) invf[i - 1] = invf[i] * i % mod;
20 }
21
22 ll Lagrange(ll *f, int k, int n) {
23     if(k <= n) return f[k];
24     pre[0] = suf[n] = 1;
25     for(int i = 1; i <= n; i++) pre[i] = pre[i - 1] * (k - i + 1) % mod;
26     for(int i = n; i >= 1; i--) suf[i - 1] = suf[i] * (k - i) % mod;
27     ll ans = 0;
28     for(int i = 0; i <= n; i++) {
29         int opt = (n - i) & 1 ? -1 : 1;
30         ans = (ans + 1ll * opt * pre[i] % mod * suf[i] % mod * invf[i] % mod * invf[n -
31             i] % mod * f[i] % mod + mod) % mod;
32     }
33     return f[k] = ans;
34 }
35
36 int main() {
37     init();
38     int n, k;
39     cin >> n >> k;
40     for(int i = 0; i <= n; i++) cin >> F[i];
41     cout << Lagrange(F, k, n) << endl;
42 }

```

6.68 多项式 ln-exp-pow 处理边界为 1

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const double PI = acos(-1);
5 const int N = 1e5 + 10;
6
7 struct Complex {
8     double x, y;
9     Complex(double a = 0, double b = 0): x(a), y(b) {}
10     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
11     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
12     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
13         rhs.y + y * rhs.x); }
14     Complex conj() { return Complex(x, -y); }
15 } w[N];
16
17 ll mod, inv2;
18 int tr[N];
19 ll F[N], G[N];
20
21 ll quick_pow(ll a, ll b) ;
22
23 int getLen(int n) ;
24
25 void FFT(Complex *A, int len) ;

```

```

26 inline void MTT(ll *x, ll *y, ll *z, int len) ;
27
28 void Get_Inv(ll *f, ll *g, int n) ;
29
30 void Get_Der(ll *f, ll *g, int len) { for(int i = 1; i < len; i++) g[i - 1] = f[i] * i %
    mod; g[len - 1] = 0; }
31
32 void Get_Int(ll *f, ll *g, int len) { for(int i = 1; i < len; i++) g[i] = f[i - 1] *
    quick_pow(i, mod - 2) % mod; g[0] = 0; }
33
34 void Get_Ln(ll *f, ll *g, int n) ;
35
36 void Get_Exp(ll *f, ll *g, int n) ;
37
38 void Get_Pow(ll *f, ll *g, int n, ll k) ;
39
40 void Get_Sqrt(ll *f, ll *g, int n) {
41     static ll a[N];
42     Get_Ln(f, a, n);
43     for(int i = 0; i < n; i++) a[i] = a[i] * inv2 % mod;
44     Get_Exp(a, g, n);
45     int len = getLen(n);
46     for(int i = n; i < len; i++) g[i] = 0;
47     for(int i = 0; i < len; i++) a[i] = 0;
48 }

```

6.69 二次剩余处理边界不为 1

```

1
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 const double PI = acos(-1);
6 const int N = 1e5 + 10;
7
8
9 struct Complex {
10     double x, y;
11     Complex(double a = 0, double b = 0): x(a), y(b) {}
12     Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
13     Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
14     Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
        rhs.y + y * rhs.x); }
15     Complex conj() { return Complex(x, -y); }
16 } w[N];
17
18 ll mod, inv2;
19 int tr[N];
20 ll F[N], G[N];
21
22 ll quick_pow(ll a, ll b) ;
23
24 typedef struct{
25     ll x, y; // 把求出来的w作为虚部, 则为a + bw
26 }num;
27
28 num num_mul(num a, num b, ll w, ll p) { // 复数乘法
29     num ans = {0, 0};

```

```

30     ans.x = (a.x * b.x % p + a.y * b.y % p * w % p + p) % p;
31     ans.y = (a.x * b.y % p + a.y * b.x % p + p) % p;
32     return ans;
33 }
34
35 ll num_pow(num a, ll b, ll w, ll p) { // 复数快速幂
36     num ans = {1, 0};
37     while(b) {
38         if(b & 1)
39             ans = num_mul(ans, a, w, p);
40         a = num_mul(a, a, w, p);
41         b >>= 1;
42     }
43     return ans.x % p;
44 }
45
46 ll legendre(ll a, ll p) { // 勒让德符号 = {1, -1, 0}
47     return quick_pow(a, (p - 1) >> 1);
48 }
49
50 ll Cipolla(ll n, ll p) { // 输入a和p, 是否存在x使得x^2 = a (mod p), 存在二次剩余返回x, 存在二次
    // 非剩余返回-1      注意: p是奇质数
51     n %= p;
52     if(n == 0)
53         return 0;
54     if(p == 2)
55         return 1;
56     ll a, w;
57
58     while(true) { // 找出a, 求出w, 随机成功的概率是50%, 所以数学期望是2
59         a = rand() % p;
60         w = ((a * a - n) % p + p) % p;
61         if(legendre(w, p) + 1 == p) // 找到w, 非二次剩余条件
62             break;
63     }
64     num x = {a, 1};
65     return num_pow(x, (p + 1) >> 1, w, p) % p; // 计算x, 一个解是x, 另一个解是p-x, 这里的w其实
    // 要开方, 但是由拉格朗日定理可知虚部为0, 所以最终答案就是对x的实部用快速幂求解
66 }
67
68 int getLen(int n) ;
69
70 void FFT(Complex *A, int len) ;
71
72 inline void MTT(ll *x, ll *y, ll *z, int len) ;
73
74 void Get_Inv(ll *f, ll *g, int n) ;
75
76 void Get_Sqrt(ll *f, ll *g, int n) {
77     if(n == 1) { ll t = Cipolla(f[0], mod); g[0] = min(mod - t, t); return ; }
78     Get_Sqrt(f, g, (n + 1) >> 1);
79
80     int len = getLen(n);
81     static ll c[N], invg[N];
82     for(int i = 0; i < len; i++) c[i] = i < n ? f[i] : 0;
83     Get_Inv(g, invg, n);
84     MTT(c, invg, c, len);
85     for(int i = 0; i < n; i++) g[i] = inv2 * (c[i] + g[i]) % mod;
86     for(int i = n; i < len; i++) g[i] = 0;

```

```

87     for(int i = 0; i < len; i++) c[i] = invg[i] = 0;
88 }
89
90 int main() {
91     inv2 = quick_pow(2, mod - 2);
92     int n;
93     cin >> n;
94     for(int i = 0; i < n; i++) cin >> F[i];
95     Get_Sqrt(F, G, n);
96     for(int i = 0; i < n; i++) cout << G[i] << " ";
97 }

```

6.70 二维几何

```

1  #include <iostream>
2  #include <cmath>
3
4  using namespace std;
5
6  const double eps = 1e-6;
7  const double pi = acos(-1);
8
9  #define zero(x) ((x) > 0 ? (x) : -(x)) < eps)
10
11 int sgn(double d) {
12     if(fabs(d) < eps)
13         return 0;
14     if(d > 0)
15         return 1;
16     else
17         return -1;
18 }
19
20 int dcmp(double x, double y) {
21     if(fabs(x - y) < eps)
22         return 0;
23     if(x > y)
24         return 1;
25     else
26         return -1;
27 }
28
29 struct Point{ // 点
30     double x, y;
31     Point(double x = 0, double y = 0) : x(x), y(y) {}
32 };
33
34 struct line{
35     Point a, b;
36 };
37
38 typedef Point Vector; // 向量
39
40 // 运算(向量之间)
41
42 Vector operator + (Vector A, Vector B) { // AB
43     return Vector(A.x + B.x, A.y + B.y);
44 }

```



```

45
46 Vector operator - (Point A, Point B) { // BA
47     return Vector(A.x - B.x, A.y - B.y);
48 }
49
50 Vector operator * (Vector A, double p) { // A * p
51     return Vector(A.x * p, A.y * p);
52 }
53
54 Vector operator / (Vector A, double p) { // A / p
55     return Vector(A.x / p, A.y / p);
56 }
57
58 bool operator < (const Point& a, const Point& b) { // 将点升序排列
59     if(a.x == b.x)
60         return a.y < b.y;
61     return a.x < b.x;
62 }
63
64 bool operator == (const Point& a, const Point& b) { // 判断是否为同一点
65     if(dcmp(a.x, b.x) == 0 && dcmp(a.y, b.y) == 0)
66         return true;
67     else
68         return false;
69 }
70
71 /*
-----
    */
72 // 向量
73
74 double Dot(Vector A, Vector B) { // 内积
75     return A.x * B.x + A.y * B.y;
76 }
77
78 double Cross(Vector A, Vector B) { // 外积
79     return A.x * B.y - A.y * B.x;
80 }
81
82 double Length(Vector A) { // 向量取模
83     return sqrt(Dot(A, A));
84 }
85
86 double Angle(Vector A, Vector B) { // 向量夹角
87     return acos(Dot(A, B) / Length(A) / Length(B));
88 }
89
90 double Area(Point A, Point B, Point C) { // 计算两向量构成的平行四边形有向面积
91     return Cross(B - A, C - A);
92 }
93
94 Vector Rotate(Vector A, double rad) { // 计算向量逆时针旋转后的向量
95     return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
96 }
97
98 Vector Normal(Vector A) { // 计算向量逆时针转90度后的单位法向量
99     double L = Length(A);
100     return Vector(-A.y / L, A.x / L);
101 }

```

```

102
103 bool ToLeftTest(Point a, Point b, Point c) { // 判断bc是不是向ab的逆时针方向转向
104     return Cross(b - a, c - b) > 0;
105 }
106
107 /*
-----
*/
108 // 直线与线段
109
110 double Pow(double x) {
111     return x * x;
112 }
113
114 double distance (Point p1, Point p2) { // 两点距离
115     return sqrt(Pow(p1.x - p2.x) + Pow(p1.y - p2.y));
116 }
117
118 int dots_inline(Point p1, Point p2, Point p3) { // 判断三点共线
119     return Cross(p2 - p1, p3 - p1);
120 }
121
122 int dot_online_in(Point p, line l) { // 判断点在线段上 (包含端点)
123     return zero(Cross(l.b - p, l.a - p) && ((l.a.x - p.x) * (l.b.x - p.x) < eps) && ((l
        .a.y - p.y) * (l.b.y - p.y) < eps));
124 }
125
126 int dot_online_ex(Point p, line l) { // 判断点在线段上 (不包含端点)
127     return dot_online_in(p, l) && (!zero(p.x - l.a.x) || !zero(p.y - l.a.y)) && (!zero(
        p.x - l.b.x) || !zero(p.y - l.b.y));
128 }
129
130 int same_side(Point p1, Point p2, line l) { // 判断两点在线段同侧, 点在线段上返回0
131     return Cross(l.a - l.b, p1 - l.b) * Cross(l.a - l.b, p2 - l.b) > eps;
132 }
133
134 int opposite_side(Point p1, Point p2, line l) { // 判断两点在线段异侧, 点在线段上返回0
135     return Cross(l.a - l.b, p1 - l.b) * Cross(l.a - l.b, p2 - l.b) < -eps;
136 }
137
138 int parallel(line u, line v) { // 判断两直线平行
139     return zero((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
140 }
141
142 int perpendicular(line u, line v) { // 判断两直线垂直
143     return zero((u.a.x - u.b.x) * (v.a.x - v.b.x) + (u.a.y - u.b.y) * (v.a.y - v.b.y));
144 }
145
146 int intersect_in(line u, line v) { // 判断两线段相交, 包括端点和部分重合
147     if(!dots_inline(u.a, u.b, v.a) || !dots_inline(u.a, u.b, v.b)) {
148         return !same_side(u.a, u.b, v) && !same_side(v.a, v.b, u);
149     }
150     return dot_online_in(u.a, v) || dot_online_in(u.b, v) || dot_online_in(v.a, u) ||
        dot_online_in(v.b, u);
151 }
152
153 int intersect_ex(line u, line v) { // 判断两线段相交, 不包括端点和部分重合
154     return opposite_side(u.a, u.b, v) && opposite_side(v.a, v.b, u);
155 }

```

```

156
157 // 计算两直线交点, 注意事先判断直线是否相交
158 // 计算两线段交点, 注意事先判断线段相交和平行
159 Point intersection(line u, line v) {
160     Point ret = u.a;
161     double t = ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) * (v.a.x - v.b.x))
162     / ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
163     ret.x += (u.b.x - u.a.x) * t;
164     ret.y += (u.b.y - u.a.y) * t;
165     return ret;
166 }
167 Point ptoline(Point p, line l) { // 点到直线最近点
168     Point t = p;
169     t.x += l.a.y - l.b.y;
170     t.y += l.b.x - l.a.x;
171     line u = {p, t};
172     return intersection(u, l);
173 }
174
175 double disptoline(Point p, line l) { // 点到直线距离
176     return fabs(Cross(p - l.b, l.a - l.b) / distance(l.a, l.b));
177 }
178
179 Point ptoseg(Point p, line l) { // 点到线段最近点
180     Point t = p;
181     t.x += l.a.y - l.b.y;
182     t.y += l.b.x - l.a.x;
183     if(Cross(l.a - p, t - p) * Cross(l.b - p, t - p) > eps)
184         return distance(p, l.a) < distance(p, l.b) ? l.a : l.b;
185     line u = {p, t};
186     return intersection(u, l);
187 }
188
189 double disptoseg(Point p, line l) { // 点到线段距离
190     Point t = p;
191     t.x += l.a.y - l.b.y;
192     t.y += l.b.x - l.a.x;
193     if(Cross(l.a - p, t - p) * Cross(l.b - p, t - p) > eps) {
194         double dis1 = distance(p, l.a);
195         double dis2 = distance(p, l.b);
196         return dis1 < dis2 ? dis1 : dis2;
197     }
198     return fabs(Cross(p - l.b, l.a - l.b) / distance(l.a, l.b));
199 }
200
201 /*
-----
*/
202 // 面积
203
204 double area_triangle(Point p1, Point p2, Point p3) { // 三角形面积 (输入三顶点)
205     return fabs(Cross(p1 - p3, p2 - p3)) / 2;
206 }
207
208 double area_triangle(double a, double b, double c) { // 三角形面积 (输入三边长)
209     double s = (a + b + c) / 2;
210     return sqrt(s * (s - a) * (s - b) * (s - c));
211 }

```

```

212
213 double area_polygon(int n, Point *p) { // 计算多边形面积, 顶点按顺时针或逆时针输入
214     double s1 = 0, s2 = 0;
215     for(int i = 0; i < n; i++) {
216         s1 += p[(i + 1) % n].y * p[i].x;
217         s2 += p[(i + 1) % n].y * p[(i + 2) % n].x;
218     }
219     return fabs(s1 - s2) / 2;
220 }
221
222 /*
-----
    */
223 // 球面
224
225 // 计算圆心角 lat 表示纬度, -90<=w<=90, lng 表示经度
226 // 返回两点所在大圆劣弧对应圆心角, 0<=angle<=pi
227
228 double angle(double lng1, double lat1, double lng2, double lat2) {
229     double dlng = fabs(lng1 - lng2) * pi / 180;
230     while(dlng >= pi + pi) {
231         dlng -= pi + pi;
232     }
233     if(dlng > pi)
234         dlng = pi + pi - dlng;
235     lat1 *= pi / 180;
236     lat2 *= pi / 180;
237     return acos(cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2));
238 }
239
240 // 计算两点距离
241
242 double line_dist(double r, double lng1, double lat1, double lng2, double lat2) {
243     double dlng = fabs(lng1 - lng2) * pi / 180;
244     while(dlng >= pi + pi) {
245         dlng -= pi + pi;
246     }
247     if(dlng > pi)
248         dlng = pi + pi - dlng;
249     lat1 *= pi / 180;
250     lat2 *= pi / 180;
251     return r * sqrt(2 - 2 * (cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2))
252 );
253 }
254 // 计算球面距离
255
256 inline double sphere_dist(double r, double lng1, double lat1, double lng2, double lat2)
257 {
258     return r * angle(lng1, lat1, lng2, lat2);
259 }
260 /*
-----
    */
261 // 三角形
262
263 // 外心
264

```

```

265 Point circumcenter(Point a, Point b, Point c) {
266     line u, v;
267     u.a.x = (a.x + b.x) / 2;
268     u.a.y = (a.y + b.y) / 2;
269     u.b.x = u.a.x - a.y + b.y;
270     u.b.y = u.a.y + a.x - b.x;
271     v.a.x = (a.x + c.x) / 2;
272     v.a.y = (a.y + c.y) / 2;
273     v.b.x = v.a.x - a.y + c.y;
274     v.b.y = v.a.y + a.x - c.y;
275     return intersection(u, v);
276 }
277
278 // 内心
279
280 Point incenter(Point a, Point b, Point c) {
281     line u, v;
282     double m, n;
283     u.a = a;
284     m = atan2(b.y - a.y, b.x - a.x);
285     n = atan2(c.y - a.y, c.x - a.x);
286     u.b.x = u.a.x + cos((m + n) / 2);
287     u.b.y = u.a.y + sin((m + n) / 2);
288     v.a = b;
289     m = atan2(a.y - b.y, a.x - b.x);
290     n = atan2(c.y - b.y, c.x - b.x);
291     v.b.x = v.a.x + cos((m + n) / 2);
292     v.b.y = v.a.y + sin((m + n) / 2);
293     return intersection(u, v);
294 }
295
296 // 垂心
297
298 Point perppcenter(Point a, Point b, Point c) {
299     line u, v;
300     u.a = c;
301     u.b.x = u.a.x - a.y + b.y;
302     u.b.y = u.a.y + a.x - b.x;
303     v.a = b;
304     v.b.x = v.a.x - a.y + c.y;
305     v.b.y = v.a.y + a.x - c.x;
306     return intersection(u, v);
307 }
308
309 // 重心
310 //到三角形三顶点距离的平方和最小的点
311 //三角形内到三边距离之积最大的点
312
313 Point barycenter(Point a, Point b, Point c) {
314     line u, v;
315     u.a.x = (a.x + b.x) / 2;
316     u.a.y = (a.y + b.y) / 2;
317     u.b = c;
318     v.a.x = (a.x + c.x) / 2;
319     v.a.y = (a.y + c.y) / 2;
320     v.b = b;
321     return intersection(u, v);
322 }
323

```

```

324 //费马点
325 //到三角形三顶点距离之和最小的点
326 Point fermentpoint(Point a, Point b, Point c) {
327     Point u,v;
328     double step = fabs(a.x) + fabs(a.y) + fabs(b.x) + fabs(b.y) + fabs(c.x) + fabs(c.y)
329     ;
330     int i, j, k;
331     u.x = (a.x + b.x + c.x) / 3;
332     u.y = (a.y + b.y + c.y) / 3;
333     while(step > 1e-10)
334         for(k = 0; k < 10; step /= 2, k++)
335             for (i = -1; i <= 1; i++)
336                 for (j = -1; j <= 1; j++){
337                     v.x = u.x + step * i;
338                     v.y = u.y + step * j;
339                     if(distance(u,a) + distance(u,b) + distance(u,c) > distance(v,a) + distance
340                        (v,b) + distance(v,c))
341                         u = v;
342                 }
343     return u;
344 }

```

6.71 三维几何

```

1  #include <math.h>
2  #define eps 1e-8
3  #define zero(x) (((x)>0?(x):-x))<eps)
4  struct point3{double x,y,z;};
5  struct line3{point3 a,b;};
6  struct plane3{point3 a,b,c;};
7  //计算 cross product U x V
8  point3 Cross(point3 u,point3 v){
9      point3 ret;
10     ret.x=u.y*v.z-v.y*u.z;
11     ret.y=u.z*v.x-u.x*v.z;
12     ret.z=u.x*v.y-u.y*v.x;
13     return ret;
14 }
15 //计算 dot product U . V
16 double Dot(point3 u,point3 v){
17     return u.x*v.x+u.y*v.y+u.z*v.z;
18 }
19 //矢量差 U - V
20 point3 subtr(point3 u,point3 v){
21     point3 ret;
22     ret.x=u.x-v.x;
23     ret.y=u.y-v.y;
24     ret.z=u.z-v.z;
25     return ret;
26 }
27 //取平面法向量
28 point3 pvec(plane3 s){
29     return Cross(subtr(s.a,s.b),subtr(s.b,s.c));
30 }
31 point3 pvec(point3 s1,point3 s2,point3 s3){
32     return Cross(subtr(s1,s2),subtr(s2,s3));
33 }
34 //两点距离,单参数取向向量大小

```

```

35 double distance(point3 p1,point3 p2){
36     return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-p2.z)
37 );
38 //向量大小
39 double vlen(point3 p){
40     return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
41 }
42 //判三点共线
43 int dots_inline(point3 p1,point3 p2,point3 p3){
44     return vlen(Cross(subt(p1,p2),subt(p2,p3)))<eps;
45 }
46 //判四点共面
47 int dots_onplane(point3 a,point3 b,point3 c,point3 d){
48     return zero(Dot(pvec(a,b,c),subt(d,a)));
49 }
50 //判点是否在线段上,包括端点和共线
51 int dot_online_in(point3 p,line3 l){
52     return zero(vlen(Cross(subt(p,l.a),subt(p,l.b))))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&
53         (l.a.y-p.y)*(l.b.y-p.y)<eps&&(l.a.z-p.z)*(l.b.z-p.z)<eps;
54 }
55 int dot_online_in(point3 p,point3 l1,point3 l2){
56     return zero(vlen(Cross(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)*(l2.x-p.x)<eps&&
57         (l1.y-p.y)*(l2.y-p.y)<eps&&(l1.z-p.z)*(l2.z-p.z)<eps;
58 }
59 //判点是否在线段上,不包括端点
60 int dot_online_ex(point3 p,line3 l){
61     return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y)||!zero(p.z-l.a.z))&&
62         (!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));
63 }
64 int dot_online_ex(point3 p,point3 l1,point3 l2){
65     return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y)||!zero(p.z-l1.z))
66         &&(!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
67 }
68 //判点是否在空间三角形上,包括边界,三点共线无意义
69 int dot_inplane_in(point3 p,plane3 s){
70     return zero(vlen(Cross(subt(s.a,s.b),subt(s.a,s.c)))-vlen(Cross(subt(p,s.a),subt(p,
71 s.b))))-
72         vlen(Cross(subt(p,s.b),subt(p,s.c)))-vlen(Cross(subt(p,s.c),subt(p,s.a)
73 )));
74 }
75 int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3){
76     return zero(vlen(Cross(subt(s1,s2),subt(s1,s3)))-vlen(Cross(subt(p,s1),subt(p,s2)))
77 -
78     vlen(Cross(subt(p,s2),subt(p,s3)))-vlen(Cross(subt(p,s3),subt(p,s1)))));
79 }
80 //判点是否在空间三角形上,不包括边界,三点共线无意义
81 int dot_inplane_ex(point3 p,plane3 s){
82     return dot_inplane_in(p,s)&&vlen(Cross(subt(p,s.a),subt(p,s.b)))>eps&&
83         vlen(Cross(subt(p,s.b),subt(p,s.c)))>eps&&vlen(Cross(subt(p,s.c),subt(p,s.a)
84 ))>eps;
85 }
86 int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3){
87     return dot_inplane_in(p,s1,s2,s3)&&vlen(Cross(subt(p,s1),subt(p,s2)))>eps&&
88         vlen(Cross(subt(p,s2),subt(p,s3)))>eps&&vlen(Cross(subt(p,s3),subt(p,s1)))>
89         eps;
90 }
91 //判两点在线段同侧,点在线段上返回 0,不共面无意义

```

```

87 int same_side(point3 p1,point3 p2,line3 l){
88     return Dot(Cross(subt(l.a,l.b),subt(p1,l.b)),Cross(subt(l.a,l.b),subt(p2,l.b)))>eps;
89 }
90 int same_side(point3 p1,point3 p2,point3 l1,point3 l2){
91     return Dot(Cross(subt(l1,l2),subt(p1,l2)),Cross(subt(l1,l2),subt(p2,l2)))>eps;
92 }
93 //判两点在线段异侧,点在线段上返回 0,不共面无意义
94 int opposite_side(point3 p1,point3 p2,line3 l){
95     return Dot(Cross(subt(l.a,l.b),subt(p1,l.b)),Cross(subt(l.a,l.b),subt(p2,l.b)))<-eps;
96 }
97 int opposite_side(point3 p1,point3 p2,point3 l1,point3 l2){
98     return Dot(Cross(subt(l1,l2),subt(p1,l2)),Cross(subt(l1,l2),subt(p2,l2)))<-eps;
99 }
100 //判两点在平面同侧,点在平面上返回 0
101 int same_side(point3 p1,point3 p2,plane3 s){
102     return Dot(pvec(s),subt(p1,s.a))*Dot(pvec(s),subt(p2,s.a))>eps;
103 }
104 int same_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
105     return Dot(pvec(s1,s2,s3),subt(p1,s1))*Dot(pvec(s1,s2,s3),subt(p2,s1))>eps;
106 }
107 //判两点在平面异侧,点在平面上返回 0
108 int opposite_side(point3 p1,point3 p2,plane3 s){
109     return Dot(pvec(s),subt(p1,s.a))*Dot(pvec(s),subt(p2,s.a))<-eps;
110 }
111 int opposite_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
112     return Dot(pvec(s1,s2,s3),subt(p1,s1))*Dot(pvec(s1,s2,s3),subt(p2,s1))<-eps;
113 }
114 //判两直线平行
115 int parallel(line3 u,line3 v){
116     return vlen(Cross(subt(u.a,u.b),subt(v.a,v.b)))<eps;
117 }
118 int parallel(point3 u1,point3 u2,point3 v1,point3 v2){
119     return vlen(Cross(subt(u1,u2),subt(v1,v2)))<eps;
120 }
121 //判两平面平行
122 int parallel(plane3 u,plane3 v){
123     return vlen(Cross(pvec(u),pvec(v)))<eps;
124 }
125 int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
126     return vlen(Cross(pvec(u1,u2,u3),pvec(v1,v2,v3)))<eps;
127 }
128 //判直线与平面平行
129 int parallel(line3 l,plane3 s){
130     return zero(Dot(subt(l.a,l.b),pvec(s)));
131 }
132 int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
133     return zero(Dot(subt(l1,l2),pvec(s1,s2,s3)));
134 }
135 //判两直线垂直
136 int perpendicular(line3 u,line3 v){
137     return zero(Dot(subt(u.a,u.b),subt(v.a,v.b)));
138 }
139 int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2){
140     return zero(Dot(subt(u1,u2),subt(v1,v2)));
141 }
142 //判两平面垂直
143 int perpendicular(plane3 u,plane3 v){

```



```

144     return zero(Dot(pvec(u),pvec(v)));
145 }
146 int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
147     return zero(Dot(pvec(u1,u2,u3),pvec(v1,v2,v3)));
148 }
149 //判直线与平面平行
150 int perpendicular(line3 l,plane3 s){
151     return vlen(Cross(subt(l.a,l.b),pvec(s)))<eps;
152 }
153 int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
154     return vlen(Cross(subt(l1,l2),pvec(s1,s2,s3)))<eps;
155 }
156 //判两线段相交,包括端点和部分重合
157 int intersect_in(line3 u,line3 v){
158     if (!dots_onplane(u.a,u.b,v.a,v.b))
159         return 0;
160     if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
161         return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
162     return dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||
        dot_online_in(v.b,u);
163 }
164 int intersect_in(point3 u1,point3 u2,point3 v1,point3 v2){
165     if (!dots_onplane(u1,u2,v1,v2))
166         return 0;
167     if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
168         return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
169     return
170         dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||
        dot_online_in(v2,u1,u2);
171 }
172 //判两线段相交,不包括端点和部分重合
173 int intersect_ex(line3 u,line3 v){
174     return dots_onplane(u.a,u.b,v.a,v.b)&&opposite_side(u.a,u.b,v)&&opposite_side(v.a,v
        .b,u);
175 }
176 int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2){
177     return dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,
        u1,u2);
178 }
179 //判线段与空间三角形相交,包括交于边界和(部分)包含
180 int intersect_in(line3 l,plane3 s){
181     return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&
182         !same_side(s.b,s.c,l.a,l.b,s.a)&&!same_side(s.c,s.a,l.a,l.b,s.b);
183 }
184 int intersect_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
185     return !same_side(l1,l2,s1,s2,s3)&&!same_side(s1,s2,l1,l2,s3)&&
186         !same_side(s2,s3,l1,l2,s1)&&!same_side(s3,s1,l1,l2,s2);
187 }
188 //判线段与空间三角形相交,不包括交于边界和(部分)包含
189 int intersect_ex(line3 l,plane3 s){
190     return opposite_side(l.a,l.b,s)&&opposite_side(s.a,s.b,l.a,l.b,s.c)&&
191         opposite_side(s.b,s.c,l.a,l.b,s.a)&&opposite_side(s.c,s.a,l.a,l.b,s.b);
192 }
193 int intersect_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
194     return opposite_side(l1,l2,s1,s2,s3)&&opposite_side(s1,s2,l1,l2,s3)&&
195         opposite_side(s2,s3,l1,l2,s1)&&opposite_side(s3,s1,l1,l2,s2);
196 }
197 //计算两直线交点,注意事先判断直线是否共面和平行!
198 //线段交点请另外判线段相交(同时还是要判断是否平行!)

```

```

199 point3 intersection(line3 u,line3 v){
200     point3 ret=u.a;
201     double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))/
202             ((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
203     ret.x+=(u.b.x-u.a.x)*t;
204     ret.y+=(u.b.y-u.a.y)*t;
205     ret.z+=(u.b.z-u.a.z)*t;
206     return ret;
207 }
208 point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2){
209     point3 ret=u1;
210     double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))/
211             ((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
212     ret.x+=(u2.x-u1.x)*t;
213     ret.y+=(u2.y-u1.y)*t;
214     ret.z+=(u2.z-u1.z)*t;
215     return ret;
216 }
217 //计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!
218 //线段和空间三角形交点请另外判断
219 point3 intersection(line3 l,plane3 s){
220     point3 ret=pvec(s);
221     double t=(ret.x*(s.a.x-l.a.x)+ret.y*(s.a.y-l.a.y)+ret.z*(s.a.z-l.a.z))/
222             (ret.x*(l.b.x-l.a.x)+ret.y*(l.b.y-l.a.y)+ret.z*(l.b.z-l.a.z));
223     ret.x=l.a.x+(l.b.x-l.a.x)*t;
224     ret.y=l.a.y+(l.b.y-l.a.y)*t;
225     ret.z=l.a.z+(l.b.z-l.a.z)*t;
226     return ret;
227 }
228 point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
229     point3 ret=pvec(s1,s2,s3);
230     double t=(ret.x*(s1.x-l1.x)+ret.y*(s1.y-l1.y)+ret.z*(s1.z-l1.z))/
231             (ret.x*(l2.x-l1.x)+ret.y*(l2.y-l1.y)+ret.z*(l2.z-l1.z));
232     ret.x=l1.x+(l2.x-l1.x)*t;
233     ret.y=l1.y+(l2.y-l1.y)*t;
234     ret.z=l1.z+(l2.z-l1.z)*t;
235     return ret;
236 }
237 //计算两平面交线,注意事先判断是否平行,并保证三点不共线!
238 line3 intersection(plane3 u,plane3 v){
239     line3 ret;
240     ret.a=parallel(v.a,v.b,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersection(
241         v.a,v.b,u.a,u.b,u.
242         c);
243     ret.b=parallel(v.c,v.a,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersection(
244         v.c,v.a,u.a,u.b,u.
245         c);
246     return ret;
247 }
248 line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
249     line3 ret;
250     ret.a=parallel(v1,v2,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v1,v2,u1,
251         u2,u3);
252     ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v3,v1,u1,
253         u2,u3);
254     return ret;
255 }
256 //点到直线距离
257 double ptoline(point3 p,line3 l){

```

```

254     return vlen(Cross(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
255 }
256 double ptoline(point3 p,point3 l1,point3 l2){
257     return vlen(Cross(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
258 }
259 //点到平面距离
260 double ptoplane(point3 p,plane3 s){
261     return fabs(Dot(pvec(s),subt(p,s.a)))/vlen(pvec(s));
262 }
263 double ptoplane(point3 p,point3 s1,point3 s2,point3 s3){
264     return fabs(Dot(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
265 }
266 //直线到直线距离
267 double linetoline(line3 u,line3 v){
268     point3 n=Cross(subt(u.a,u.b),subt(v.a,v.b));
269     return fabs(Dot(subt(u.a,v.a),n))/vlen(n);
270 }
271 double linetoline(point3 u1,point3 u2,point3 v1,point3 v2){
272     point3 n=Cross(subt(u1,u2),subt(v1,v2));
273     return fabs(Dot(subt(u1,v1),n))/vlen(n);
274 }
275 //两直线夹角 cos 值
276 double angle_cos(line3 u,line3 v){
277     return Dot(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.b));
278 }
279 double angle_cos(point3 u1,point3 u2,point3 v1,point3 v2){
280     return Dot(subt(u1,u2),subt(v1,v2))/vlen(subt(u1,u2))/vlen(subt(v1,v2));
281 }
282 //两平面夹角 cos 值
283 double angle_cos(plane3 u,plane3 v){
284     return Dot(pvec(u),pvec(v))/vlen(pvec(u))/vlen(pvec(v));
285 }
286 double angle_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
287     return Dot(pvec(u1,u2,u3),pvec(v1,v2,v3))/vlen(pvec(u1,u2,u3))/vlen(pvec(v1,v2,v3))
288     ;
289 }
289 //直线平面夹角 sin 值
290 double angle_sin(line3 l,plane3 s){
291     return Dot(subt(l.a,l.b),pvec(s))/vlen(subt(l.a,l.b))/vlen(pvec(s));
292 }
293 double angle_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
294     return Dot(subt(l1,l2),pvec(s1,s2,s3))/vlen(subt(l1,l2))/vlen(pvec(s1,s2,s3));
295 }
296
297 // 球体相交
298 double vol_ints(double x1, double y1, double z1, double r1, double x2, double y2,
299     double z2, double r2) {
300     double sum = 4.00 / 3.00 * PI * r1 * r1 * r1 + 4.00 / 3.00 * PI * r2 * r2 * r2;
301     double ans = 0;
302     double dis = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2) + (z1 - z2) * (z1 -
303     z2)); //球心距离
304     if (dis >= r1 + r2) //没有交到的地方
305     {
306         ans = 0;
307     } else if (dis + r1 <= r2)//重合
308     {
309         ans = (4.00 / 3.00) * PI * r1 * r1 * r1;
310     } else if (dis + r2 <= r1) {
311         ans = (4.00 / 3.00) * PI * r2 * r2 * r2;
312     }
313 }

```

```

310     } else //相交
311     {
312         double cal = (r1 * r1 + dis * dis - r2 * r2) / (2.00 * dis * r1);
313         double h = r1 * (1 - cal);
314         ans += (1.00 / 3.00) * PI * (3.00 * r1 - h) * h * h;
315         cal = (r2 * r2 + dis * dis - r1 * r1) / (2.00 * dis * r2);
316         h = r2 * (1.00 - cal);
317         ans += (1.00 / 3.00) * PI * (3.00 * r2 - h) * h * h;
318     }
319     return ans;
320 }

```

6.72 Poly-Z

```

1  constexpr int P = 998244353;
2  using i64 = long long;
3  // assume -P <= x < 2P
4  int norm(int x) {
5      if (x < 0) {
6          x += P;
7      }
8      if (x >= P) {
9          x -= P;
10     }
11     return x;
12 }
13 template<class T>
14 T power(T a, int b) {
15     T res = 1;
16     for (; b; b /= 2, a *= a) {
17         if (b % 2) {
18             res *= a;
19         }
20     }
21     return res;
22 }
23 struct Z {
24     int x;
25     Z(int x = 0) : x(norm(x)) {}
26     int val() const {
27         return x;
28     }
29     Z operator-() const {
30         return Z(norm(P - x));
31     }
32     Z inv() const {
33         assert(x != 0);
34         return power(*this, P - 2);
35     }
36     Z &operator*=(const Z &rhs) {
37         x = i64(x) * rhs.x % P;
38         return *this;
39     }
40     Z &operator+=(const Z &rhs) {
41         x = norm(x + rhs.x);
42         return *this;
43     }
44     Z &operator-=(const Z &rhs) {

```

```

45     x = norm(x - rhs.x);
46     return *this;
47 }
48 Z &operator/=(const Z &rhs) {
49     return *this *= rhs.inv();
50 }
51 friend Z operator*(const Z &lhs, const Z &rhs) {
52     Z res = lhs;
53     res *= rhs;
54     return res;
55 }
56 friend Z operator+(const Z &lhs, const Z &rhs) {
57     Z res = lhs;
58     res += rhs;
59     return res;
60 }
61 friend Z operator-(const Z &lhs, const Z &rhs) {
62     Z res = lhs;
63     res -= rhs;
64     return res;
65 }
66 friend Z operator/(const Z &lhs, const Z &rhs) {
67     Z res = lhs;
68     res /= rhs;
69     return res;
70 }
71 };
72
73 std::vector<int> rev;
74 std::vector<Z> roots{0, 1};
75 void dft(std::vector<Z> &a) {
76     int n = a.size();
77
78     if (int(rev.size()) != n) {
79         int k = __builtin_ctz(n) - 1;
80         rev.resize(n);
81         for (int i = 0; i < n; i++) {
82             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
83         }
84     }
85
86     for (int i = 0; i < n; i++) {
87         if (rev[i] < i) {
88             std::swap(a[i], a[rev[i]]);
89         }
90     }
91     if (int(roots.size()) < n) {
92         int k = __builtin_ctz(roots.size());
93         roots.resize(n);
94         while ((1 << k) < n) {
95             Z e = power(Z(3), (P - 1) >> (k + 1));
96             for (int i = 1 << (k - 1); i < (1 << k); i++) {
97                 roots[2 * i] = roots[i];
98                 roots[2 * i + 1] = roots[i] * e;
99             }
100             k++;
101         }
102     }
103     for (int k = 1; k < n; k *= 2) {

```

```

104     for (int i = 0; i < n; i += 2 * k) {
105         for (int j = 0; j < k; j++) {
106             Z u = a[i + j];
107             Z v = a[i + j + k] * roots[k + j];
108             a[i + j] = u + v;
109             a[i + j + k] = u - v;
110         }
111     }
112 }
113 }
114 void idft(std::vector<Z> &a) {
115     int n = a.size();
116     std::reverse(a.begin() + 1, a.end());
117     dft(a);
118     Z inv = (1 - P) / n;
119     for (int i = 0; i < n; i++) {
120         a[i] *= inv;
121     }
122 }
123 struct Poly {
124     std::vector<Z> a;
125     Poly() {}
126     Poly(const std::vector<Z> &a) : a(a) {}
127     int size() const {
128         return a.size();
129     }
130     void resize(int n) {
131         a.resize(n);
132     }
133     Z operator[](int idx) const {
134         if (idx < 0 || idx >= size()) {
135             return 0;
136         }
137         return a[idx];
138     }
139     Z &operator[](int idx) {
140         return a[idx];
141     }
142     Poly mulxk(int k) const {
143         auto b = a;
144         b.insert(b.begin(), k, 0);
145         return Poly(b);
146     }
147     Poly modxk(int k) const {
148         k = std::min(k, size());
149         return Poly(std::vector<Z>(a.begin(), a.begin() + k));
150     }
151     Poly divxk(int k) const {
152         if (size() <= k) {
153             return Poly();
154         }
155         return Poly(std::vector<Z>(a.begin() + k, a.end()));
156     }
157     friend Poly operator+(const Poly &a, const Poly &b) {
158         std::vector<Z> res(std::max(a.size(), b.size()));
159         for (int i = 0; i < int(res.size()); i++) {
160             res[i] = a[i] + b[i];
161         }
162         return Poly(res);

```

```

163     }
164     friend Poly operator-(const Poly &a, const Poly &b) {
165         std::vector<Z> res(std::max(a.size(), b.size()));
166         for (int i = 0; i < int(res.size()); i++) {
167             res[i] = a[i] - b[i];
168         }
169         return Poly(res);
170     }
171     friend Poly operator*(Poly a, Poly b) {
172         if (a.size() == 0 || b.size() == 0) {
173             return Poly();
174         }
175         int sz = 1, tot = a.size() + b.size() - 1;
176         while (sz < tot)
177             sz *= 2;
178         a.a.resize(sz);
179         b.a.resize(sz);
180         dft(a.a);
181         dft(b.a);
182         for (int i = 0; i < sz; ++i) {
183             a.a[i] = a[i] * b[i];
184         }
185         idft(a.a);
186         a.resize(tot);
187         return a;
188     }
189     friend Poly operator*(Z a, Poly b) {
190         for (int i = 0; i < int(b.size()); i++) {
191             b[i] *= a;
192         }
193         return b;
194     }
195     friend Poly operator*(Poly a, Z b) {
196         for (int i = 0; i < int(a.size()); i++) {
197             a[i] *= b;
198         }
199         return a;
200     }
201     Poly &operator+=(Poly b) {
202         return (*this) = (*this) + b;
203     }
204     Poly &operator-=(Poly b) {
205         return (*this) = (*this) - b;
206     }
207     Poly &operator*=(Poly b) {
208         return (*this) = (*this) * b;
209     }
210     Poly deriv() const {
211         if (a.empty()) {
212             return Poly();
213         }
214         std::vector<Z> res(size() - 1);
215         for (int i = 0; i < size() - 1; ++i) {
216             res[i] = (i + 1) * a[i + 1];
217         }
218         return Poly(res);
219     }
220     Poly integr() const {
221         std::vector<Z> res(size() + 1);

```

```

222     for (int i = 0; i < size(); ++i) {
223         res[i + 1] = a[i] / (i + 1);
224     }
225     return Poly(res);
226 }
227 Poly inv(int m) const {
228     Poly x({a[0].inv()});
229     int k = 1;
230     while (k < m) {
231         k *= 2;
232         x = (x * (Poly({2}) - modxk(k) * x)).modxk(k);
233     }
234     return x.modxk(m);
235 }
236 Poly log(int m) const {
237     return (deriv() * inv(m)).integr().modxk(m);
238 }
239 Poly exp(int m) const {
240     Poly x({1});
241     int k = 1;
242     while (k < m) {
243         k *= 2;
244         x = (x * (Poly({1}) - x.log(k) + modxk(k))).modxk(k);
245     }
246     return x.modxk(m);
247 }
248 Poly sqrt(int m) const {
249     Poly x({1});
250     int k = 1;
251     while (k < m) {
252         k *= 2;
253         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
254     }
255     return x.modxk(m);
256 }
257 Poly mult(Poly b) const {
258     if (b.size() == 0) {
259         return Poly();
260     }
261     int n = b.size();
262     std::reverse(b.a.begin(), b.a.end());
263     return ((*this) * b).divxk(n - 1);
264 }
265 std::vector<Z> eval(std::vector<Z> x) const {
266     if (size() == 0) {
267         return std::vector<Z>(x.size(), 0);
268     }
269     const int n = std::max(int(x.size()), size());
270     std::vector<Poly> q(4 * n);
271     std::vector<Z> ans(x.size());
272     x.resize(n);
273     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
274         if (r - l == 1) {
275             q[p] = Poly({1, -x[l]});
276         } else {
277             int m = (l + r) / 2;
278             build(2 * p, l, m);
279             build(2 * p + 1, m, r);
280             q[p] = q[2 * p] * q[2 * p + 1];

```



```

281     }
282     };
283     build(1, 0, n);
284     std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r
, const Poly &num) {
285         if (r - l == 1) {
286             if (l < int(ans.size())) {
287                 ans[l] = num[0];
288             }
289         } else {
290             int m = (l + r) / 2;
291             work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
292             work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
293         }
294     };
295     work(1, 0, n, mulT(q[1].inv(n)));
296     return ans;
297 }
298 };

```

6.73 多项式快速幂

```

1  #include "bits/stdc++.h"
2  using namespace std;
3
4  const int N = 2e6 + 10;
5
6  const int P = 998244353, g3 = (P + 1) / 3;
7
8  int pow(int a, int b) {
9      int r = 1;
10     while(b) {
11         if(b & 1) r = (ll)r * a % P;
12         a = (ll)a * a % P;
13         b >>= 1;
14     }
15     return r;
16 }
17 namespace poly {
18     int rev[N];
19     void NTT(int *A, int n, int inv) {
20         for(int i = 0; i < n; ++i)
21             if(i < rev[i]) swap(A[i], A[rev[i]]);
22         for(int mid = 1; mid < n; mid <= 1) {
23             int tmp = pow(inv == 1 ? 3 : g3, (P - 1) / (mid << 1));
24             for(int j = 0; j < n; j += (mid << 1)) {
25                 int omega = 1;
26                 for(int k = 0; k < mid; ++k, omega = (ll)omega * tmp % P) {
27                     int x = A[j + k], y = (ll)omega * A[j + k + mid] % P;
28                     A[j + k] = (x + y) % P;
29                     A[j + k + mid] = (ll)(x - y + P) % P;
30                 }
31             }
32         }
33         if(inv == 1) return;
34         int invn = pow(n, P - 2);
35         for(int i = 0; i < n; ++i)
36             A[i] = (ll)A[i] * invn % P;

```

```

37     }
38     void Inv(int *a, int *b, int n) {
39         static int B[N], A[N];
40         b[0] = pow(a[0], P - 2);
41         int len, lim;
42         for(len = 1; len < (n << 1); len <= 1) {
43             lim = len << 1;
44             for(int i = 0; i < len; i++)
45                 A[i] = a[i], B[i] = b[i];
46             for(int i = 0; i < lim; i++)
47                 rev[i] = (rev[i >> 1] >> 1) | ((i & 1) ? len : 0);
48             NTT(A, lim, 1), NTT(B, lim, 1);
49             for(int i = 0; i < lim; i++)
50                 b[i] = ((2LL - 1LL * A[i] * B[i] % P) * B[i] % P + P) % P;
51             NTT(b, lim, -1);
52             for(int i = len; i < lim; i++)
53                 b[i] = 0;
54         }
55         for(int i = 0; i < len; i++)
56             A[i] = B[i] = 0;
57         for(int i = n; i < len; i++)
58             b[i] = 0;
59     }
60     void derivative(int *a, int *b, int n) {
61         b[n - 1] = 0;
62         for(int i = 1; i < n; ++i)
63             b[i - 1] = (1LL)a[i] * i % P;
64     }
65     void inter(int *a, int *b, int n) {
66         *b = 0;
67         for(int i = n - 1; i >= 0; --i)
68             b[i + 1] = a[i] * (1LL)pow(i + 1, P - 2) % P;
69     }
70     void ln(int *a, int *b, int n) {
71         static int F[N];
72         derivative(a, F, n);
73         Inv(a, b, n);
74         int lim = 1;
75         while(lim < (n << 1)) lim <= 1;
76         for(int i = 1; i < lim; i++)
77             rev[i] = (rev[i >> 1] >> 1) | ((i & 1) ? (lim >> 1) : 0);
78         for(int i = n; i < lim; ++i)
79             b[i] = F[i] = 0;
80         NTT(F, lim, 1), NTT(b, lim, 1);
81         for(int i = 0; i < lim; ++i)
82             F[i] = (1LL)b[i] * F[i] % P;
83         NTT(F, lim, 0);
84         inter(F, b, n);
85         for(int i = n; i < lim; ++i)
86             b[i] = 0;
87     }
88     void exp(int*a, int*F, int n) {
89         if(n == 1)
90             *F = 1;
91         else {
92             exp(a, F, n + 1 >> 1);
93             static int F0[N], A[N];
94             for(int i = 0; i <= (n << 1); ++i)
95                 F0[i] = 0, A[i] = a[i];

```

```

96         ln(F, F0, n);
97         int lim = 1;
98         while(lim < (n << 1)) lim <= 1;
99         for(int i = 1; i < lim; i++)
100             rev[i] = (rev[i >> 1] >> 1) | ((i & 1) ? (lim >> 1) : 0);
101         for(int i = n; i < lim; ++i)
102             A[i] = 0;
103         NTT(A, lim, 1), NTT(F0, lim, 1), NTT(F, lim, 1);
104         for(int i = 0; i < lim; ++i)
105             F[i] = F[i] * (A[i] + 1LL - F0[i] + P) % P;
106         NTT(F, lim, 0);
107         for(int i = n; i < lim; ++i)
108             F[i] = 0;
109     }
110 }
111 }
112 using namespace poly;
113
114 int a[N], b[N];
115
116 void solve() {
117     int n, m, k; cin >> n >> m >> k;
118     for(int i = 0; i < n; i++) cin >> a[i];
119     ln(a, b, m);
120     for(int i = 0; i < m; i++) b[i] = b[i] * k % mod;
121     exp(b, a, m);
122     for(int i = 0; i < m; i++) cout << a[i] << " ";
123 }

```

6.74 Geometry

```

1  using Point = std::complex<double>;
2
3  #define x real
4  #define y imag
5
6  double dot(const Point &a, const Point &b) {
7      return (std::conj(a) * b).x();
8  }
9
10 double cross(const Point &a, const Point &b) {
11     return (std::conj(a) * b).y();
12 }
13
14 struct Line {
15     Point a;
16     Point b;
17     Line(const Point &a, const Point &b) : a(a), b(b) {}
18 };
19
20 Point rotate(const Point &a) {
21     return Point(-a.y(), a.x());
22 }
23
24 int sgn(const Point &a) {
25     return a.y() > 0 || (a.y() == 0 && a.x() > 0) ? 1 : -1;
26 }
27

```

```
28 bool onLeft(const Point &a, const Line &l) {
29     return cross(l.b - l.a, a - l.a) > 0;
30 }
31
32 Point intersection(const Line &l1, const Line &l2) {
33     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a,
34         l1.a - l1.b));
35 }
36
37 std::vector<Point> hp(std::vector<Line> lines) {
38     std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
39         auto d1 = l1.b - l1.a;
40         auto d2 = l2.b - l2.a;
41
42         if (sgn(d1) != sgn(d2)) {
43             return sgn(d1) == 1;
44         }
45
46         return cross(d1, d2) > 0;
47     });
48
49     std::deque<Line> ls;
50     std::deque<Point> ps;
51     for (auto l : lines) {
52         if (ls.empty()) {
53             ls.push_back(l);
54             continue;
55         }
56
57         while (!ps.empty() && !onLeft(ps.back(), l)) {
58             ps.pop_back();
59             ls.pop_back();
60         }
61
62         while (!ps.empty() && !onLeft(ps[0], l)) {
63             ps.pop_front();
64             ls.pop_front();
65         }
66
67         if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
68             if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
69                 if (!onLeft(ls.back().a, l)) {
70                     assert(ls.size() == 1);
71                     ls[0] = l;
72                 }
73                 continue;
74             }
75             return {};
76         }
77
78         ps.push_back(intersection(ls.back(), l));
79         ls.push_back(l);
80     }
81
82     while (!ps.empty() && !onLeft(ps.back(), ls[0])) {
83         ps.pop_back();
84         ls.pop_back();
85     }
```

```
86     if (ls.size() <= 2) {
87         return {};
88     }
89     ps.push_back(intersection(ls[0], ls.back()));
90
91     return std::vector<Point>(ps.begin(), ps.end());
92 }
```

7 图论

7.1 graph

```

1  template <typename T>
2  class graph {
3  public:
4      struct edge {
5          int from;
6          int to;
7          T cost;
8      };
9
10     const int n;
11     std::vector<edge> edges;
12     std::vector<std::vector<int>>> g;
13
14     graph(int _n) : n(_n), g(n) {}
15
16     virtual int add(int from, int to, T cost) = 0;
17 };
18
19 template <typename T>
20 class digraph : public graph<T> {
21 public:
22     using graph<T>::edges;
23     using graph<T>::g;
24     using graph<T>::n;
25
26     digraph(int _n) : graph<T>(_n) {}
27
28     int add(int from, int to, T cost = 1) {
29         assert(0 <= from && from < n && 0 <= to && to < n);
30         int id = (int) edges.size();
31         g[from].push_back(id);
32         edges.push_back({from, to, cost});
33         return id;
34     }
35
36     digraph<T> reverse() const {
37         digraph<T> rev(n);
38         for (auto &e : edges) {
39             rev.add(e.to, e.from, e.cost);
40         }
41         return rev;
42     }
43 };
44
45
46 template <typename T>
47 class undigraph : public graph<T> {
48 public:
49     using graph<T>::edges;
50     using graph<T>::g;
51     using graph<T>::n;
52
53     undigraph(int _n) : graph<T>(_n) {}
54
55     int add(int from, int to, T cost = 1) {

```

```

56     assert(0 <= from && from < n && 0 <= to && to < n);
57     int id = (int) edges.size();
58     g[from].push_back(id);
59     g[to].push_back(id);
60     edges.push_back({from, to, cost});
61     return id;
62 }
63 };

```

7.2 isBipartiteGraph

```

1  template <typename T>
2  bool isBipartiteGraph(const graph<T>& g) {
3      std::vector<int> color(g.n);
4      bool flag = true;
5      std::function<bool(int, int)> dfs = [&](int u, int x) -> bool {
6          for (int id : g.g[u]) {
7              auto& e = g.edges[id];
8              int to = e.from ^ e.to ^ u;
9              if (!color[to]) {
10                 dfs(to, 3 - x);
11             }
12             if (color[to] == color[u]) {
13                 flag = false;
14             }
15         }
16     };
17     for (int i = 0; i < g.n; i++) {
18         if (!color[i]) {
19             dfs(i, 1);
20         }
21     }
22     return flag;
23 }

```

7.3 hungry

```

1  template <typename T>
2  int hungry(const digraph<T>& g) {
3      std::vector<bool> was(g.n);
4      std::vector<int> match(g.n, -1);
5      std::function<bool(int)> dfs = [&](int u) -> bool {
6          for (int id : g.g[u]) {
7              auto& e = g.edges[id];
8              int to = e.to;
9              if (!was[to]) {
10                 was[to] = true;
11                 if (match[to] == -1 || dfs(match[to])) {
12                     match[to] = u;
13                     return true;
14                 }
15             }
16         }
17         return false;
18     };
19
20     int ans = 0;

```

```

21     for (int i = 0; i < n; i++) {
22         vis.assign(g.n, false);
23         if (dfs(i)) ans++;
24     }
25     return ans;
26 }

```

7.4 KM

```

1
2 template <typename T>
3 class hungarian { // km
4 public :
5     int n;
6     std::vector<int> matchx; // 左集合对应的匹配点
7     std::vector<int> matchy; // 右集合对应的匹配点
8     std::vector<int> pre;    // 连接右集合的左点
9     std::vector<bool> visx;  // 拜访数组 左
10    std::vector<bool> visy;  // 拜访数组 右
11    std::vector<T> lx;
12    std::vector<T> ly;
13    std::vector<vector<T> > g;
14    std::vector<T> slack;
15    T inf;
16    T res;
17    std::queue<int> q;
18    int org_n;
19    int org_m;
20
21    hungarian(int _n, int _m) {
22        org_n = _n;
23        org_m = _m;
24        n = max(_n, _m);
25        inf = numeric_limits<T>::max();
26        res = 0;
27        g = vector<vector<T> >(n, vector<T>(n));
28        matchx = vector<int>(n, -1);
29        matchy = vector<int>(n, -1);
30        pre = vector<int>(n);
31        visx = vector<bool>(n);
32        visy = vector<bool>(n);
33        lx = vector<T>(n, -inf);
34        ly = vector<T>(n);
35        slack = vector<T>(n);
36    }
37
38    void addEdge(int u, int v, int w) {
39        g[u][v] = max(w, 0); // 负值还不如不匹配 因此设为0不影响
40    }
41
42    bool check(int v) {
43        visy[v] = true;
44        if (matchy[v] != -1) {
45            q.push(matchy[v]);
46            visx[matchy[v]] = true; // in S
47            return false;
48        }
49        // 找到新的未匹配点 更新匹配点 pre 数组记录着"非匹配边"上与之相连的点

```



```

50     while (v != -1) {
51         matchy[v] = pre[v];
52         swap(v, matchx[pre[v]]);
53     }
54     return true;
55 }
56
57 void bfs(int i) {
58     while (!q.empty()) {
59         q.pop();
60     }
61     q.push(i);
62     visx[i] = true;
63     while (true) {
64         while (!q.empty()) {
65             int u = q.front();
66             q.pop();
67             for (int v = 0; v < n; v++) {
68                 if (!visy[v]) {
69                     T delta = lx[u] + ly[v] - g[u][v];
70                     if (slack[v] >= delta) {
71                         pre[v] = u;
72                         if (delta) {
73                             slack[v] = delta;
74                         } else if (check(v)) { // delta=0 代表有机会加入相等子图 找增广路
75                                                 // 找到就return 重建交错树
76                             return;
77                         }
78                     }
79                 }
80             }
81         }
82         // 没有增广路 修改顶标
83         T a = inf;
84         for (int j = 0; j < n; j++) {
85             if (!visy[j]) {
86                 a = min(a, slack[j]);
87             }
88         }
89         for (int j = 0; j < n; j++) {
90             if (visx[j]) { // S
91                 lx[j] -= a;
92             }
93             if (visy[j]) { // T
94                 ly[j] += a;
95             } else { // T'
96                 slack[j] -= a;
97             }
98         }
99         for (int j = 0; j < n; j++) {
100             if (!visy[j] && slack[j] == 0 && check(j)) {
101                 return;
102             }
103         }
104     }
105 }
106
107 void solve() {
108     // 初始顶标

```

```

109     for (int i = 0; i < n; i++) {
110         for (int j = 0; j < n; j++) {
111             lx[i] = max(lx[i], g[i][j]);
112         }
113     }
114
115     for (int i = 0; i < n; i++) {
116         fill(slack.begin(), slack.end(), inf);
117         fill(visx.begin(), visx.end(), false);
118         fill(visy.begin(), visy.end(), false);
119         bfs(i);
120     }
121
122     // custom
123     for (int i = 0; i < n; i++) {
124         if (g[i][matchx[i]] > 0) {
125             res += g[i][matchx[i]];
126         } else {
127             matchx[i] = -1;
128         }
129     }
130     // cout << res << "\n";
131     // for (int i = 0; i < org_n; i++) {
132     //     cout << matchx[i] + 1 << " ";
133     // }
134     // cout << "\n";
135 }
136 };

```

7.5 galeShapley

```

1  #include<iostream>
2  using namespace std;
3
4  const int N=4;
5
6  void GaleShapley(const int (&man)[MAX][MAX], const int (&woman)[MAX][MAX], int (&match)
    [MAX]) {
7      int wm[MAX][MAX];    // wm[i][j]: rank from girl i to boy j
8      int choose[MAX];    // choose[i]: current boyfriend of girl i
9      int manIndex[MAX];  // manIndex[i]: how many girls that have rejected boy i
10     int i, j;
11     int w, m;
12     for (i = 0; i < N; i++) {
13         match[i] = -1;
14         choose[i] = -1;
15         manIndex[i] = 0;
16         for (j = 0; j < N; j++)
17             wm[i][woman[i][j]] = j;
18     }
19
20     bool bSingle = false;
21     while (!bSingle) {
22         bSingle = true;
23         for (i = 0; i < N; i++) {
24             if (match[i] != -1) // boy i already have a girlfriend
25                 continue;
26             bSingle = false;

```

```

27         j = manIndex[i]++; // the jth girl that boy i like most
28         w = man[i][j];
29         m = choose[w]; // current girl w's boyfriend
30         if (m == -1 || wm[w][i] < wm[w][m]) { // if girl w prefer boy i
31             match[i] = w;
32             choose[w] = i;
33             if (m != -1)
34                 match[m] = -1;
35         }
36     }
37 }
38 }
39
40
41 void Print(const int(&match)[MAX], int N) {
42     for (int i = 0; i < N; i++)
43         cout << i << " " << match[i] << endl;
44 }
45
46
47 int main(){
48     int man[N][N]={
49         {2,3,1,0},
50         {2,1,3,0},
51         {0,2,3,1},
52         {1,3,2,0},
53     };
54     int woman[N][N]={
55         {0,3,2,1},
56         {0,1,2,3},
57         {0,2,3,1},
58         {1,0,3,2},
59     };
60
61     int match[N];
62     GaleShapley(man,woman,match);
63     Print(match,N);
64
65     return 0;
66 }

```

7.6 bellman-ford

```

1  template <typename T>
2  std::vector<std::vector<T>> bellman_ford(const graph<T>& g, int st) {
3      std::vector<T> dist(g.n, std::numeric_limits<T>::max());
4      dist[st] = 0;
5
6      // Relax all edges |V| - 1 times. A simple
7      // shortest path from src to any other vertex can have
8      // at-most |V| - 1 edges
9      for (int i = 1; i < g.n; i++) {
10         for (auto& e : g.edges) {
11             int from = e.from, to = e.to, cost = e.cost;
12             if (dist[from] != std::numeric_limits<T>::max() && dist[from] + cost < dist
13                 [to]) {
14                 dist[to] = dist[from] + e.cost;
15             }
16         }
17     }
18 }

```

```

15     }
16 }
17
18 for (auto& e : g.edges) {
19     int from = e.from, to = e.to, cost = e.cost;
20     if (dist[from] != std::numeric_limits<T>::max() && dist[from] + cost < dist[to]
21 ) {
22         // Graph contains negative weight cycle
23         return {};
24     }
25 }
26 return dist;
27 }

```

7.7 dijkstra

```

1 template <typename T>
2 std::vector<T> dijkstra(const graph<T>& g, int st) {
3     assert(0 <= st && st < g.n);
4     std::vector<T> dist(g.n, std::numeric_limits<T>::max());
5     std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T, int>>> > q;
6     dist[st] = 0;
7     q.emplace(dist[st], st);
8     while (!q.empty()) {
9         T expected = q.top().first;
10        int u = q.top().second; q.pop();
11        if (dist[u] != expected) {
12            continue;
13        }
14        for (int id : g.g[u]) {
15            auto& e = g.edges[id];
16            int to = e.from ^ e.to ^ u;
17            if (dist[to] > dist[u] + e.cost) {
18                dist[to] = dist[u] + e.cost;
19                q.emplace(dist[to], to);
20            }
21        }
22    }
23    return dist;
24    // returns numeric_limits<T>::max() if there's no path
25 }

```

7.8 floyd

```

1 template <typename T>
2 void floyd(std::vector<std::vector<T>>& dist) {
3     for (int k = 0; k < dist.size(); k++) {
4         for (int i = 0; i < dist.size(); i++) {
5             for (int j = 0; j < dist.size(); j++) {
6                 dist[i][j] = std::min(dist[i][j], dist[i][k], dist[k][j]);
7             }
8         }
9     }
10 }

```

7.9 spfa

```

1  template <typename T>
2  std::vector<T> spfa(const graph<T>& g, int st) {
3      std::vector<T> dist(g.n, std::numeric_limits<T>::max());
4      std::vector<bool> vis(g.n);
5      std::vector<int> cnt(g.n);
6      std::vector<int> x(1, st);
7      dist[st] = 0; vis[st] = true;
8      for (int ptr = 0; ptr < x.size(); ptr++) {
9          int u = x[ptr];
10         vis[u] = false;
11         for (int id : g.g[u]) {
12             auto& e = g.edges[id];
13             int to = e.from ^ e.to ^ u;
14             if (dist[to] > dist[u] + e.cost) {
15                 dist[to] = dist[u] + e.cost;
16                 if (!vis[to]) {
17                     cnt[to]++;
18                     vis[to] = true;
19                     if (cnt[to] >= g.n) {
20                         return std::vector<T>();
21                     }
22                     x.push_back(to);
23                 }
24             }
25         }
26     }
27     return dist;
28 }

```

7.10 Kruskal

```

1  template <typename T>
2  std::vector<int> find_mst(const undigraph<T> &g, T& ans) {
3      std::vector<int> order(g.edges.size());
4      iota(order.begin(), order.end(), 0);
5      sort(order.begin(), order.end(), [&g](int a, int b) {
6          return g.edges[a].cost < g.edges[b].cost;
7      });
8      DSU d(g.n);
9      std::vector<int> ans_list;
10     ans = 0;
11     for (int id : order) {
12         auto &e = g.edges[id];
13         if (!d.same(e.from, e.to)) {
14             d.merge(e.from, e.to);
15             ans_list.push_back(id);
16             ans += e.cost;
17         }
18     }
19     return ans_list;
20     // returns edge ids of minimum "spanning" forest
21 }

```

7.11 prim

```

1  template <typename T>
2  bool find_mst(const undigraph<T> &g, T& ans) {
3      std::vector<bool> vis(g.n);
4      std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T, int>>> > q;
5      q.push({0, 0});
6      int cnt = 0; ans = 0;
7      while (!q.empty() && cnt < g.n) {
8          T expected = q.top().first;
9          int u = q.top().second; q.pop();
10         if (vis[u]) continue;
11         vis[u] = true;
12         ans += expected; cnt++;
13         for (int id : g.g[u]) {
14             auto &e = g.edges[id];
15             int to = e.from ^ e.to ^ u;
16             if (!vis[to]) {
17                 q.push({e.cost, to});
18             }
19         }
20     }
21     return cnt == g.n;
22     // returns false if there's not connected
23 }

```

7.12 topSort

```

1  template <typename T>
2  std::vector<int> find_topsort(const digraph<T> &g) {
3      std::vector<int> deg(g.n, 0);
4      for (int id = 0; id < (int) g.edges.size(); id++) {
5          deg[g.edges[id].to]++;
6      }
7      std::vector<int> x;
8      for (int i = 0; i < g.n; i++) {
9          if (deg[i] == 0) {
10             x.push_back(i);
11         }
12     }
13
14     for (int ptr = 0; ptr < (int) x.size(); ptr++) {
15         int i = x[ptr];
16         for (int id : g.g[i]) {
17             auto &e = g.edges[id];
18             int to = e.to;
19             if (--deg[to] == 0) {
20                 x.push_back(to);
21             }
22         }
23     }
24
25     if ((int) x.size() != g.n) {
26         return std::vector<int>();
27     }
28     return x;
29 }

```

7.13 scc

```

1  template <typename T>
2  std::vector<int> find_scc(const digraph<T> &g, int &cnt) {
3      digraph<T> g_rev = g.reverse();
4      std::vector<int> order;
5      std::vector<bool> was(g.n, false);
6      std::function<void(int)> dfs1 = [&](int v) {
7          was[v] = true;
8          for (int id : g.g[v]) {
9              auto &e = g.edges[id];
10             int to = e.to;
11             if (!was[to]) {
12                 dfs1(to);
13             }
14         }
15         order.push_back(v);
16     };
17     for (int i = 0; i < g.n; i++) {
18         if (!was[i]) {
19             dfs1(i);
20         }
21     }
22     std::vector<int> c(g.n, -1);
23     std::function<void(int)> dfs2 = [&](int v) {
24         for (int id : g_rev.g[v]) {
25             auto &e = g_rev.edges[id];
26             int to = e.to;
27             if (c[to] == -1) {
28                 c[to] = c[v];
29                 dfs2(to);
30             }
31         }
32     };
33     cnt = 0;
34     for (int id = g.n - 1; id >= 0; id--) {
35         int i = order[id];
36         if (c[i] != -1) {
37             continue;
38         }
39         c[i] = cnt++;
40         dfs2(i);
41     }
42     return c;
43 }

```

7.14 cycles

```

1  template <typename T>
2  std::vector<std::vector<int>> find_cycles(const graph<T> &g, int bound_cnt = 1 << 30,
3      int bound_size = 1 << 30) {
4      std::vector<int> was(g.n, -1);
5      std::vector<int> st;
6      std::vector<std::vector<int>> cycles;
7      int total_size = 0;
8      std::function<void(int, int)> dfs = [&](int v, int pe) {
9          if ((int) cycles.size() >= bound_cnt || total_size >= bound_size) {
10             return;
11         }
12     };

```

```

10     }
11     was[v] = (int) st.size();
12     for (int id : g.g[v]) {
13         if (id == pe) {
14             continue ;
15         }
16         auto &e = g.edges[id];
17         int to = e.from ^ e.to ^ v;
18         if (was[to] >= 0) {
19             std::vector<int> cycle(1, id);
20             for (int j = was[to]; j < (int) st.size(); j++) {
21                 cycle.push_back(st[j]);
22             }
23             cycles.push_back(cycle);
24             total_size += (int) cycle.size();
25             if ((int) cycles.size() >= bound_cnt || total_size >= bound_size) {
26                 return;
27             }
28             continue;
29         }
30         if (was[to] == -1) {
31             st.push_back(id);
32             dfs(to, id);
33             st.pop_back();
34         }
35     }
36     was[v] = -2;
37 };
38 for (int i = 0; i < g.n; i++) {
39     if (was[i] == -1) {
40         dfs(i, -1);
41     }
42 }
43 return cycles;
44 // cycles are given by edge ids, all cycles are simple
45 // breaks after getting bound_cnt cycles or total_size >= bound_size
46 // digraph: finds at least one cycle in every connected component (if not broken)
47 // undigraph: finds cycle basis
48 }
49
50 template <typename T>
51 std::vector<int> edges_to_vertices(const graph<T> &g, const std::vector<int> &
    edge_cycle) {
52     int sz = (int) edge_cycle.size();
53     std::vector<int> vertex_cycle;
54     if (sz <= 2) {
55         vertex_cycle.push_back(g.edges[edge_cycle[0]].from);
56         if (sz == 2) {
57             vertex_cycle.push_back(g.edges[edge_cycle[0]].to);
58         }
59     } else {
60         for (int i = 0; i < sz; i++) {
61             int j = (i + 1) % sz;
62             auto &e = g.edges[edge_cycle[i]];
63             auto &other = g.edges[edge_cycle[j]];
64             if (other.from == e.from || other.to == e.from) {
65                 vertex_cycle.push_back(e.to);
66             } else {
67                 vertex_cycle.push_back(e.from);

```



```

68         }
69     }
70 }
71 return vertex_cycle;
72 // only for simple cycles!
73 }

```

7.15 ternaryRingCount

```

1  template <typename T>
2  int ternaryRingCount(const digraph<T>& g) {
3      std::vector<int> d(g.n);
4      for (auto& e : g.edges) {
5          int from = e.from, to = e.to;
6          d[from]++;
7          d[to]++;
8      }
9      digraph<T>& ng(g.n);
10     for (auto& e : g.edges) {
11         int from = e.from, to = e.to;
12         if (d[from] < d[to] || (d[from] == d[to] && from > to)) std::swap(from, to);
13         ng.add(from, to);
14     }
15     int ans = 0;
16     std::vector<bool> was(ng.n);
17     for (int u = 0; u < ng.n; u++) {
18         for (int id : ng.g[u]) {
19             auto& e = ng.edges[id];
20             int to = e.to;
21             was[to] = u;
22         }
23         for (int id1 : ng.g[u]) {
24             auto& e1 = ng.edges[id1];
25             int to1 = e1.to;
26             for (int id2 : ng.g[to1]) {
27                 auto& e2 = ng.g[id2];
28                 int to2 = e2.to;
29                 if (was[to2] == u) ans++;
30             }
31         }
32     }
33     return ans;
34 }

```

7.16 eulerian-path

```

1  template <typename T>
2  std::vector<int> find_eulerian_path(const graph<T> &g, int &root) {
3      // in_deg and out_deg are fake for undigraph!
4      std::vector<int> in_deg(g.n, 0);
5      std::vector<int> out_deg(g.n, 0);
6      int cnt_edges = 0;
7      for (int id = 0; id < (int) g.edges.size(); id++) {
8          cnt_edges++;
9          auto &e = g.edges[id];
10         out_deg[e.from]++;
11         in_deg[e.to]++;

```

```

12     }
13     root = -1;
14     int odd = 0;
15     for (int i = 0; i < g.n; i++) {
16         if ((in_deg[i] + out_deg[i]) % 2 == 1) {
17             odd++;
18             if (root == -1 || out_deg[i] - in_deg[i] > out_deg[root] - in_deg[root]) {
19                 root = i;
20             }
21         }
22     }
23     if (odd > 2) {
24         root = -1;
25         return std::vector<int>();
26     }
27     if (root == -1) {
28         root = 0;
29         while (root < g.n && in_deg[root] + out_deg[root] == 0) {
30             root++;
31         }
32         if (root == g.n) {
33             // an empty path
34             root = 0;
35             return std::vector<int>();
36         }
37     }
38     std::vector<bool> used(g.edges.size(), false);
39     std::vector<int> ptr(g.n, 0);
40     std::vector<int> balance(g.n, 0);
41     std::vector<int> res(cnt_edges);
42     int stack_ptr = 0;
43     int write_ptr = cnt_edges;
44     int v = root;
45     while (true) {
46         bool found = false;
47         while (ptr[v] < (int) g.g[v].size()) {
48             int id = g.g[v][ptr[v]++];
49             if (used[id]) {
50                 continue;
51             }
52             used[id] = true;
53             res[stack_ptr++] = id;
54             auto &e = g.edges[id];
55             balance[v]++;
56             v ^= e.from ^ e.to;
57             balance[v]--;
58             found = true;
59             break;
60         }
61         if (!found) {
62             if (stack_ptr == 0) {
63                 break;
64             }
65             int id = res[--stack_ptr];
66             res[--write_ptr] = id;
67             auto &e = g.edges[id];
68             v ^= e.from ^ e.to;
69         }
70     }

```

```

71     int disbalance = 0;
72     for (int i = 0; i < g.n; i++) {
73         disbalance += abs(balance[i]);
74     }
75     if (write_ptr != 0 || disbalance > 2) {
76         root = -1;
77         return std::vector<int>();
78     }
79     return res;
80 }

```

7.17 twoSat

```

1  class twosat {
2  public:
3      digraph<int> g;
4      int n;
5
6      twosat(int _n) : g(digraph<int>(2 * _n)), n(_n) {}
7
8      inline void add(int x, int value_x) {
9          assert(0 <= x && x < n);
10         assert(0 <= value_x && value_x <= 1);
11         g.add(2 * x + (value_x ^ 1), 2 * x + value_x);
12     }
13
14     inline void add(int x, int value_x, int y, int value_y) {
15         assert(0 <= x && x < n && 0 <= y && y < n);
16         assert(0 <= value_x && value_x <= 1 && 0 <= value_y && value_y <= 1);
17         g.add(2 * x + (value_x ^ 1), 2 * y + value_y);
18         g.add(2 * y + (value_y ^ 1), 2 * x + value_x);
19     }
20
21     inline std::vector<int> solve() {
22         int cnt;
23         std::vector<int> c = find_scc(g, cnt);
24         std::vector<int> res(n);
25         for (int i = 0; i < n; i++) {
26             if (c[2 * i] == c[2 * i + 1]) {
27                 return std::vector<int>();
28             }
29             res[i] = (c[2 * i] < c[2 * i + 1]);
30         }
31         return res;
32     }
33 };

```

7.18 maxAssignment

```

1  template<class T>
2  struct MaxAssignment {
3  public:
4      T solve(int nx, int ny, std::vector<std::vector<T>> a) {
5          assert(0 <= nx && nx <= ny);
6          assert(int(a.size()) == nx);
7          for (int i = 0; i < nx; ++i) {
8              assert(int(a[i].size()) == ny);

```

```

9         for (auto x : a[i])
10             assert(x >= 0);
11     }
12
13     auto update = [&](int x) {
14         for (int y = 0; y < ny; ++y) {
15             if (lx[x] + ly[y] - a[x][y] < slack[y]) {
16                 slack[y] = lx[x] + ly[y] - a[x][y];
17                 slackx[y] = x;
18             }
19         }
20     };
21
22     costs.resize(nx + 1);
23     costs[0] = 0;
24     lx.assign(nx, std::numeric_limits<T>::max());
25     ly.assign(ny, 0);
26     xy.assign(nx, -1);
27     yx.assign(ny, -1);
28     slackx.resize(ny);
29     for (int cur = 0; cur < nx; ++cur) {
30         std::queue<int> que;
31         visx.assign(nx, false);
32         visy.assign(ny, false);
33         slack.assign(ny, std::numeric_limits<T>::max());
34         p.assign(nx, -1);
35
36         for (int x = 0; x < nx; ++x) {
37             if (xy[x] == -1) {
38                 que.push(x);
39                 visx[x] = true;
40                 update(x);
41             }
42         }
43
44         int ex, ey;
45         bool found = false;
46         while (!found) {
47             while (!que.empty() && !found) {
48                 auto x = que.front();
49                 que.pop();
50                 for (int y = 0; y < ny; ++y) {
51                     if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
52                         if (yx[y] == -1) {
53                             ex = x;
54                             ey = y;
55                             found = true;
56                             break;
57                         }
58                         que.push(yx[y]);
59                         p[yx[y]] = x;
60                         visy[y] = visx[yx[y]] = true;
61                         update(yx[y]);
62                     }
63                 }
64             }
65             if (found)
66                 break;
67

```

```

68         T delta = std::numeric_limits<T>::max();
69         for (int y = 0; y < ny; ++y)
70             if (!visy[y])
71                 delta = std::min(delta, slack[y]);
72         for (int x = 0; x < nx; ++x)
73             if (visx[x])
74                 lx[x] -= delta;
75         for (int y = 0; y < ny; ++y) {
76             if (visy[y]) {
77                 ly[y] += delta;
78             } else {
79                 slack[y] -= delta;
80             }
81         }
82         for (int y = 0; y < ny; ++y) {
83             if (!visy[y] && slack[y] == 0) {
84                 if (yx[y] == -1) {
85                     ex = slackx[y];
86                     ey = y;
87                     found = true;
88                     break;
89                 }
90                 que.push(yx[y]);
91                 p[yx[y]] = slackx[y];
92                 visy[y] = visx[yx[y]] = true;
93                 update(yx[y]);
94             }
95         }
96     }
97
98     costs[cur + 1] = costs[cur];
99     for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
100         costs[cur + 1] += a[x][y];
101         if (xy[x] != -1)
102             costs[cur + 1] -= a[x][xy[x]];
103         ty = xy[x];
104         xy[x] = y;
105         yx[y] = x;
106     }
107 }
108 return costs[nx];
109 }
110 std::vector<int> assignment() {
111     return xy;
112 }
113 std::pair<std::vector<T>, std::vector<T>> labels() {
114     return std::make_pair(lx, ly);
115 }
116 std::vector<T> weights() {
117     return costs;
118 }
119 private:
120     std::vector<T> lx, ly, slack, costs;
121     std::vector<int> xy, yx, p, slackx;
122     std::vector<bool> visx, visy;
123 };

```

7.19 tarjan

```

1  template <typename T>
2  class tarjan {
3  public:
4      digraph<T> &g;
5      int tim;
6      std::vector<int> dfn;
7      std::vector<int> low;
8      std::vector<bool> vis;
9      std::stack<int> s;
10     std::set<int> cutVertexs;
11     std::vector<std::array<int, 2>> cutBridges;
12     std::vector<std::vector<int>> cycles;
13     int root;
14     tarjan(digraph<T> &_g) : g(_g) {
15         dfn.resize(g.n);
16         low.resize(g.n);
17         vis.resize(g.n);
18         tim = 1;
19     }
20
21     void dfs(int u) {
22         s.push(u);
23         dfn[u] = low[u] = tim++;
24         vis[u] = true;
25         int child = 0;
26         for (int id : g.g[u]) {
27             auto &e = g.edges[id];
28             int to = e.to;
29             if (!dfn[to]) {
30                 dfs(to);
31                 low[u] = std::min(low[u], low[to]);
32                 // if u is not root and low value of one of its child is more than dfn
33                 // value of u // if u is root of DFS tree and has two or more children
34                 if (low[to] >= dfn[u]) {
35                     child++;
36                     if (u != root || child >= 2) {
37                         cutVertexs.insert(u);
38                     }
39                 }
40                 // if the lowest vertex reachable from subtree under v is below u is
41                 // DFS tree, the u-b is a bridge
42                 if (low[to] > dfn[u]) {
43                     cutBridges.push_back({std::min(u, to), std::max(u, to)});
44                 }
45                 } else if (vis[to]) {
46                     low[u] = std::min(low[u], dfn[to]);
47                 }
48             }
49
50             // find a strongly connected component
51             if (dfn[u] == low[u]) {
52                 int from, cnt = 0;
53                 std::vector<int> cycle;
54                 do {
55                     from = s.top();
56                     cycle.push_back(from);

```

```

56         vis[from] = false;
57         s.pop();
58         cnt++;
59         // TODO ... num of connected component = cnt
60     } while (from != u);
61     cycles.push_back(cycle);
62 }
63 }
64
65 void solve() {
66     std::vector<std::vector<int>> cycles;
67     for (int i = 0; i < g.n; i++) {
68         if (!dfn[i]) {
69             root = i;
70             dfs(i);
71         }
72     }
73 }
74 };

```

7.20 biconnected-components

```

1  template <typename T>
2  std::vector<int> find_bicone(dfs_undigraph<T> &g, int &cnt) {
3      g.dfs_all();
4      std::vector<int> vertex_comp(g.n);
5      cnt = 0;
6      for (int i : g.order) {
7          if (g.pv[i] == -1 || g.min_depth[i] == g.depth[i]) {
8              vertex_comp[i] = cnt++;
9          } else {
10             vertex_comp[i] = vertex_comp[g.pv[i]];
11         }
12     }
13     return vertex_comp;
14 }

```

7.21 differenceConstraints

```

1
2  /*
3  差分约束是解决这样一类问题
4  给出n个形如 $x[j] - x[i] \leq k$ 的式子, 求 $x[n] - x[1]$ 的最大/最小值
5  最大值→把所有式子整理为 $x[j] - x[i] \leq k$ , 从i向j连一条边权为k的边, 跑最短路
6  最小值→把所有式子整理为 $x[j] - x[i] \geq k$ , 从i向j连一条边权为k的边, 跑最长路
7  注意初始化 有时候需要超级源点0
8  */
9
10 //dfs跑差分约束最短路
11 template <typename T>
12 bool differenceConstraints(const graph<T>& g) {
13     std::vector<bool> was(g.n);
14     std::vector<T> dist(g.n, std::numeric_limits<T>::max());
15     std::function<bool(int)> spfa = [&](int u) -> bool {
16         was[u] = true;
17         for (int id : g.g[u]) {
18             auto& e = g.edges[id];

```

```

19         int to = e.from ^ e.to ^ to;
20         if (dist[u] + e.cost < dist[to]) {
21             if (was[to]) return false;
22             dist[to] = dist[u] + e.cost;
23             if (!spfa(to)) return false;
24         }
25     }
26     was[u] = false;
27     return true;
28 };
29 return spfa(0);
30 }

```

7.22 AHU

```

1 //用来判断两棵树是否同构
2 // AHU :判断两棵树是否是同构
3 //同构:在更换节点的标号之后两棵树能完全相同
4
5 const int N = 1e5 + 5;
6 const int maxn = N << 1;
7
8 int n;
9 struct Edge {
10     int v, nxt;
11 } e[maxn << 1];
12 int head[maxn], sz[maxn], f[maxn], maxv[maxn], tag[maxn], tot, Max;
13 vector<int> center[2], L[maxn], subtree_tags[maxn];
14 void addedge(int u, int v) {
15     e[tot].v = v;
16     e[tot].nxt = head[u];
17     head[u] = tot++;
18     e[tot].v = u;
19     e[tot].nxt = head[v];
20     head[v] = tot++;
21 }
22
23 void dfs_size(int u, int fa) {
24     sz[u] = 1;
25     maxv[u] = 0;
26     for (int i = head[u]; i; i = e[i].nxt) {
27         int v = e[i].v;
28         if (v == fa) continue;
29         dfs_size(v, u);
30         sz[u] += sz[v];
31         maxv[u] = max(maxv[u], sz[v]);
32     }
33 }
34
35 void dfs_center(int rt, int u, int fa, int id) {
36     maxv[u] = max(maxv[u], sz[rt] - sz[u]);
37     if (Max > maxv[u]) {
38         center[id].clear();
39         Max = maxv[u];
40     }
41     if (Max == maxv[u]) center[id].push_back(u);
42     for (int i = head[u]; i; i = e[i].nxt) {
43         int v = e[i].v;

```



```

44         if (v == fa) continue;
45         dfs_center(rt, v, u, id);
46     }
47 }
48
49 int dfs_height(int u, int fa, int depth) {
50     L[depth].push_back(u);
51     f[u] = fa;
52     int h = 0;
53     for (int i = head[u]; i; i = e[i].nxt) {
54         int v = e[i].v;
55         if (v == fa) continue;
56         h = max(h, dfs_height(v, u, depth + 1));
57     }
58     return h + 1;
59 }
60
61 void init(int n) {
62     for (int i = 1; i <= 2 * n; i++) head[i] = 0;
63     tot = 1;
64     center[0].clear();
65     center[1].clear();
66
67     int u, v;
68     for (int i = 1; i <= n - 1; i++) { //在这里输入第一棵树的边
69         scanf("%d %d", &u, &v);
70         addedge(u, v);
71     }
72     dfs_size(1, -1);
73     Max = n;
74     dfs_center(1, 1, -1, 0);
75
76     for (int i = 1; i <= n - 1; i++) { //在这里输入第二棵树的边
77         scanf("%d %d", &u, &v);
78         addedge(u + n, v + n);
79     }
80     dfs_size(1 + n, -1);
81     Max = n;
82     dfs_center(1 + n, 1 + n, -1, 1);
83 }
84
85 bool cmp(int u, int v) { return subtree_tags[u] < subtree_tags[v]; }
86
87 bool rootedTreeIsomorphism(int rt1, int rt2) {
88     for (int i = 0; i <= 2 * n + 1; i++) L[i].clear(), subtree_tags[i].clear();
89     int h1 = dfs_height(rt1, -1, 0);
90     int h2 = dfs_height(rt2, -1, 0);
91     if (h1 != h2) return false;
92     int h = h1 - 1;
93     for (int j = 0; j < (int)L[h].size(); j++) tag[L[h][j]] = 0;
94     for (int i = h - 1; i >= 0; i--) {
95         for (int j = 0; j < (int)L[i + 1].size(); j++) {
96             int v = L[i + 1][j];
97             subtree_tags[f[v]].push_back(tag[v]);
98         }
99     }
100     sort(L[i].begin(), L[i].end(), cmp);
101
102     for (int j = 0, cnt = 0; j < (int)L[i].size(); j++) {

```

```

103         if (j && subtree_tags[L[i][j]] != subtree_tags[L[i][j - 1]]) ++cnt;
104         tag[L[i][j]] = cnt;
105     }
106 }
107 return subtree_tags[rt1] == subtree_tags[rt2];
108 }
109
110 bool treeIsomorphism() {
111     if (center[0].size() == center[1].size()) {
112         if (rootedTreeIsomorphism(center[0][0], center[1][0])) return true;
113         if (center[0].size() > 1)
114             return rootedTreeIsomorphism(center[0][0], center[1][1]);
115     }
116     return false;
117 }
118
119 int main() {
120     int T;
121     scanf("%d", &T);
122     while (T--) {
123         scanf("%d", &n);
124         init(n);
125         puts(treeIsomorphism() ? "YES" : "NO");
126     }
127     return 0;
128 }

```

7.23 Astar

```

1  #include "bits/stdc++.h"
2  using namespace std;
3  //A*
4  //用来计算点A到点B的第k短的路径
5
6  const int MAXN = 55;
7  const int MAXM = MAXN * MAXN;
8
9  int dis[MAXN];
10 int n, m, k, a, b, cnt;
11 bool hav = false;
12
13 namespace G1{//反图
14     int to[MAXM], val[MAXM], head[MAXN], nxt[MAXM], cnt;
15     bool vis[MAXN];
16
17     void AddEdge(int u, int v, int w) {
18         cnt++;
19         to[cnt] = v;
20         val[cnt] = w;
21         nxt[cnt] = head[u];
22         head[u] = cnt;
23     }
24
25     void Spfa(int s, int t) { //SPFA+SLF跑最短路
26         memset(dis, 0x7f, sizeof(dis)); dis[s] = 0;
27         deque<int> q; q.push_back(s); vis[s] = true;
28         while (!q.empty()) {
29             int u = q.front(); q.pop_front(); vis[u] = false;

```

```

30     for (int i = head[u]; i; i = nxt[i]) {
31         int v = to[i];
32         if (dis[v] > dis[u] + val[i]) {
33             dis[v] = dis[u] + val[i];
34             if (!vis[v]) {
35                 vis[v] = true;
36                 if (!q.empty() && dis[v] < dis[q.front()]) {
37                     q.push_front(v);
38                 } else {
39                     q.push_back(v);
40                 }
41             }
42         }
43     }
44 }
45 }
46 }
47
48 namespace G2{//原图
49     int to[MAXM], val[MAXM], nxt[MAXM], head[MAXN], cnt;
50
51     void AddEdge(int u, int v, int w) {
52         cnt++;
53         to[cnt] = v;
54         val[cnt] = w;
55         nxt[cnt] = head[u];
56         head[u] = cnt;
57     }
58
59     struct Data{//当前位置, 走过的距离, s->now->t总距离, 走的步骤
60         int now, pas, val;
61         vector<int> route;
62         /*
63         bool operator < (const Data &b) const {return val > b.val;}
64         */
65         bool operator < (const Data &b) const {//重载
66             if (val != b.val) return val > b.val;
67             int sz = min(route.size(), b.route.size());
68             for (int i = 0; i < sz; i++) {
69                 if (route[i] != b.route[i]) return route[i] > b.route[i];
70             }
71             return route.size() > b.route.size();
72         }
73     };
74
75     void Astar(int s, int t) {//A*
76         priority_queue<Data> q;
77         Data st;
78         st.now = s; st.pas = 0; st.val = dis[s]; st.route = vector<int>{s};
79         q.push(st);
80         vector<int> vec;
81         while (!q.empty()) {
82             Data u = q.top(); q.pop();
83             if (u.now == t) {//更新路径数
84                 :: cnt++;
85                 if (:: cnt == k) {//最终答案
86                     cout << u.route[0];
87                     for (int i = 1, sz = u.route.size(); i < sz; i++)
88                         cout << '-' << u.route[i];

```

```

89         hav = true;
90         return;
91     }
92 }
93 for (int i = head[u.now]; i; i = nxt[i]) { //广搜
94     int v = to[i];
95     vec = u.route;
96     bool visit = false;
97     for (int j = 0, sz = vec.size(); j < sz; j++) { //记录是否重复经过
98         if (vec[j] == v) {
99             visit = true;
100             break;
101         }
102     }
103     if (visit) continue;
104     Data nx = u;
105     nx.now = v;
106     nx.pas = u.pas + val[i];
107     nx.val = dis[v] + nx.pas;
108     nx.route.push_back(v);
109     q.push(nx);
110 }
111 }
112 }
113 }
114
115 int main() {
116     cin >> n >> m >> k >> a >> b;
117     for (int i = 1; i <= m; i++) {
118         int u, v, w;
119         cin >> u >> v >> w;
120         G1 :: AddEdge(v, u, w);
121         G2 :: AddEdge(u, v, w);
122     }
123     G1 :: Spfa(b, a);
124     G2 :: Astar(a, b);
125     if (!hav) cout << "No" << endl;
126     return 0;
127 }

```

7.24 dinic

```

1  template <typename T>
2  class flow_graph {
3  public:
4      static constexpr T eps = (T) 1e-9;
5
6      struct edge {
7          int from;
8          int to;
9          T c;
10         T f;
11     };
12
13     const int n;
14     std::vector<edge> edges;
15     std::vector<std::vector<int>>> g;
16     int st;

```

```

17     int fin;
18     T flow;
19
20     flow_graph(int _n, int _st, int _fin) : n(n), st(_st), fin(_fin) {
21         assert(0 <= st && st < n && 0 <= fin && fin < n && st != fin);
22         g.resize(n);
23         flow = 0;
24     }
25
26     void clear_flow() {
27         for (const edge& e : edges) {
28             e.f = 0;
29         }
30         flow = 0;
31     }
32
33     int add(int from, int to, T forward_cap, T backward_cap) {
34         assert(0 <= from && from < n && 0 <= to && to < n);
35         int id = (int) edges.size();
36         g[from].push_back(id);
37         edges.push_back({from, to, forward_cap, 0});
38         g[to].push_back(id + 1);
39         edges.push_back({to, from, backward_cap, 0});
40         return id;
41     }
42 };
43
44 template <typename T>
45 class dinic {
46 public:
47     flow_graph<T> &g;
48     std::vector<int> ptr;
49     std::vector<int> d;
50     std::vector<int> q;
51
52     dinic(flow_graph<T> &_g) : g(_g) {
53         ptr.resize(g.n);
54         d.resize(g.n);
55         q.resize(g.n);
56     }
57
58     bool expath() {
59         fill(d.begin(), d.end(), -1);
60         q[0] = g.fin;
61         d[g.fin] = 0;
62         int beg = 0, end = 1;
63         while (beg < end) {
64             int i = q[beg++];
65             for (int id : g.g[i]) {
66                 const auto &e = g.edges[id];
67                 const auto &back = g.edges[id ^ 1];
68                 if (back.c - back.f > g.eps && d[e.to] != -1) {
69                     d[e.to] = d[i] + 1;
70                     if (e.to == g.st) {
71                         return true;
72                     }
73                     q[end++] = e.to;
74                 }
75             }

```

```

76     }
77     return false;
78 }
79
80 T dfs(int v, int w) {
81     if (v == g.fin) {
82         return w;
83     }
84     int &j = ptr[v];
85     while (j >= 0) {
86         int id = g.g[v][j];
87         const auto& e = g.edges[id];
88         if (e.c - e.f > g.eps && d[e.to] == d[v] - 1) {
89             T t = dfs(e.to, std::min(e.c - e.f), w);
90             if (t > g.eps) {
91                 g.edges[id].f += t;
92                 g.edges[id ^ 1].f -= t;
93                 return t;
94             }
95         }
96         j--;
97     }
98     return 0;
99 }
100
101 T max_flow() {
102     while (expath()) {
103         for (int i = 0; i < g.n; i++) {
104             ptr[i] = (int) g.g[i].size() - 1;
105         }
106         T big_add = 0;
107         while (true) {
108             T add = dfs(g.st, std::numeric_limits<T>::max());
109             if (add <= g.eps) {
110                 break;
111             }
112             big_add += add;
113         }
114         if (big_add <= g.eps) {
115             break;
116         }
117         g.flow += big_add;
118     }
119     return g.flow;
120 }
121
122 std::vector<bool> min_cut() {
123     max_flow();
124     std::vector<bool> ret(g.n);
125     for (int i = 0; i < g.n; i++) {
126         ret[i] = (d[i] != -1);
127     }
128     return ret;
129 }
130 };

```

7.25 ISAP

```

1 struct ISAP {
2     const static int N = ...; //node size
3     struct Edge {
4         int from, to, cap, flow;
5         bool operator < (const Edge &rhs) const {
6             return from < rhs.from || (from == rhs.from && to < rhs.to);
7         }
8     };
9     int n, m, s, t;
10    vector<Edge> edges;
11    vector<int> g[N];
12    bool vis[N];
13    int dep[N], cur[N], p[N], num[N];
14
15    void addEdge(int from, int to, int cap) {
16        edges.push_back(Edge{from, to, cap, 0});
17        edges.push_back(Edge{to, from, 0, 0});
18        m = edges.size();
19        g[from].push_back(m - 2);
20        g[to].push_back(m - 1);
21    }
22
23    bool bfs() {
24        memset(vis, 0, sizeof(vis));
25        queue<int> q; q.push(t); vis[t] = 1, dep[t] = 0;
26        while (!q.empty()) {
27            int u = q.front(); q.pop();
28            for (auto &v: g[u]) {
29                Edge &e = edges[v ^ 1];
30                if (!vis[e.from] && e.cap > e.flow) {
31                    dep[e.from] = dep[u] + (vis[e.from] = 1);
32                    q.push(e.from);
33                }
34            }
35        }
36        return vis[s];
37    }
38
39    void init(int siz) {
40        n = siz;
41        for (int i = 0; i < siz; i++) g[i].clear();
42        edges.clear();
43    }
44
45    int augment() {
46        int u = t, a = INF;
47        while (u != s) {
48            Edge &e = edges[p[u]];
49            a = min(a, e.cap - e.flow);
50            u = edges[p[u]].from;
51        }
52        u = t;
53        while (u != s) {
54            edges[p[u]].flow += a;
55            edges[p[u] ^ 1].flow -= a;
56            u = edges[p[u]].from;
57        }
58        return a;
59    }

```

```

60
61 int maxFlow(int S, int T) {
62     s = S, t = T;
63     int flow = 0; bfs();
64     memset(num, 0, sizeof(num));
65     for (int i = 0; i < n; i++) num[dep[i]]++;
66     int u = S;
67     memset(cur, 0, sizeof(cur));
68     while (dep[S] < n) {
69         if (u == T) {
70             flow += augment();
71             u = S;
72         }
73         int ok = 0;
74         for (int i = cur[u]; i < g[u].size(); i++) {
75             Edge &e = edges[g[u][i]];
76             if (e.cap > e.flow && dep[u] == dep[e.to] + 1) {
77                 ok = 1;
78                 p[e.to] = g[u][i];
79                 cur[u] = i;
80                 u = e.to;
81                 break;
82             }
83         }
84         if (!ok) {
85             int mn = n - 1;
86             for (int i = 0; i < g[u].size(); i++) {
87                 Edge &e = edges[g[u][i]];
88                 if (e.cap > e.flow) mn = min(mn, dep[e.to]);
89             }
90             if (--num[dep[u]] == 0) break;
91             num[dep[u] = mn + 1]++;
92             cur[u] = 0;
93             if (u != S) u = edges[p[u]].from;
94         }
95     }
96     return flow;
97 }
98
99 } flow;

```

7.26 mcmf

```

1 struct MCMF {
2     struct Edge {
3         int from, to, cap, flow, cost;
4         Edge(int u, int v, int c, int f, int cc)
5             : from(u), to(v), cap(c), flow(f), cost(cc) {}
6     };
7     static constexpr int INF = 1e9;
8     int n, m;
9
10    std::vector<Edge> edges;
11    std::vector<std::vector<int>>> G;
12    std::vector<int> inq;
13    std::vector<int> d;
14    std::vector<int> p;
15    std::vector<int> a;

```



```

16
17
18 MCMF(int n) : n(n), G(n), inq(n), d(n), p(n), a(n) {}
19
20 void add(int from, int to, int cap, int cost) {
21     edges.emplace_back(Edge(from, to, cap, 0, cost));
22     edges.emplace_back(Edge(to, from, 0, 0, -cost));
23     m = int(edges.size());
24     G[from].emplace_back(m - 2);
25     G[to].emplace_back(m - 1);
26 }
27
28 bool spfa(int s, int t, int &flow, int &cost) {
29     for (int i = 1; i < n; ++i) d[i] = INF;
30     inq.assign(n, 0);
31     d[s] = 0;
32     inq[s] = 1;
33     p[s] = 0;
34     std::queue<int> q;
35     a[s] = INF;
36     q.push(s);
37     while (!q.empty()) {
38         int u = q.front();
39         q.pop();
40         inq[u] = 0;
41         for (int i = 0; i < int(G[u].size()); ++i) {
42             Edge &e = edges[G[u][i]];
43             if (e.cap > e.flow && d[e.to] > d[u] + e.cost) {
44                 d[e.to] = d[u] + e.cost;
45                 p[e.to] = G[u][i];
46                 a[e.to] = std::min(a[u], e.cap - e.flow);
47                 if (!inq[e.to]) {
48                     q.push(e.to);
49                     inq[e.to] = 1;
50                 }
51             }
52         }
53     }
54     if (d[t] == INF) return false;
55     flow += a[t];
56     cost += d[t] * a[t];
57     for (int u = t; u != s; u = edges[p[u]].from) {
58         edges[p[u]].flow += a[t];
59         edges[p[u] ^ 1].flow -= a[t];
60     }
61     return true;
62 }
63
64 int MincostMaxflow(int s, int t, int &cost) {
65     int flow = 0;
66     cost = 0;
67     while (spfa(s, t, flow, cost));
68     return flow;
69 }
70 };

```

7.27 conclusion.md

8 其他

8.1 simpleMO

```

1  // O(n*sqrt(n))
2  const int N = 1e5 + 10;
3
4  int a[N], cnt[N], ans[N];
5  int belong[N];
6
7  struct Q {
8      int l, r, id;
9  } q[N];
10
11 int Size, bnum;
12
13 bool cmp(Q a, Q b) {
14     return (belong[a.l] ^ belong[b.l]) ? belong[a.l] < belong[b.l] : belong[a.l] & 1 ?
        a.r < b.r : a.r > b.r;
15 }
16
17 int now = 0;
18
19 inline void add(int pos) {
20     if(!cnt[a[pos]]) now++;
21     ++cnt[a[pos]];
22 }
23
24 inline void del(int pos) {
25     --cnt[a[pos]];
26     if(!cnt[a[pos]]) --now;
27 }
28
29 int main() {
30     int n, m;
31     cin >> n >> m;
32     Size = sqrt(n);
33     bnum = ceil((double)n / Size);
34     for(int i = 1; i <= bnum; i++) {
35         for(int j = (i - 1) * Size + 1; j <= i * Size; j++) {
36             belong[j] = i;
37         }
38     }
39     for(int i = 1; i <= n; i++) cin >> a[i];
40     for(int i = 1; i <= m; i++) {
41         cin >> q[i].l >> q[i].r;
42         q[i].id = i;
43     }
44     sort(q + 1, q + m + 1, cmp);
45     int l = 1, r = 0;
46     for(int i = 1; i <= m; i++) {
47         int ql = q[i].l, qr = q[i].r;
48         while(r < qr) add(++r);
49         while(r > qr) del(r--);
50         while(l < ql) del(l++);
51         while(l > ql) add(--l);
52         ans[q[i].id] = now;
53     }
54     for(int i = 1; i <= m; i++) cout << ans[i] << endl;

```

55 }

8.2 modifyMO

```

1  //带修莫队模板题
2  //查询[ql, qr]间不同颜色数量, 带修改
3
4  //  $O(n^{5/3})$ 
5
6  int n, m;
7  int c[N], cnt[N];
8  int belong[N], size, totq, totm;
9  int ans[N], res;
10
11 struct Query {
12     int l, r, t, id;
13     bool operator < (const Query &rhs) const {
14         return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] :
15             (belong[r] ^ belong[rhs.r] ? belong[r] < belong[rhs.r] : t < rhs.t);
16     }
17 } q[N];
18
19 struct Modify {
20     int pos, val;
21 } modify[N];
22
23 void add(int x) {
24     if (!cnt[c[x]]) res++;
25     cnt[c[x]]++;
26 }
27
28 void del(int x) {
29     cnt[c[x]]--;
30     if (!cnt[c[x]]) res--;
31 }
32
33 void upd(int x, int ql, int qr) {
34     int pos = modify[x].pos;
35     if (ql <= pos && pos <= qr) {
36         cnt[c[pos]]--; if (!cnt[c[pos]]) res--;
37         if (!cnt[modify[x].val]) res++; cnt[modify[x].val]++;
38     }
39     swap(modify[x].val, c[pos]); //0 0 0 j0 0 0 0 '0 0 L0 z
40 }
41
42 int main() {
43 #ifdef ACM_LOCAL
44     freopen("input.in", "r", stdin);
45     freopen("output.out", "w", stdout);
46 #endif
47     scanf("%d%d", &n, &m);
48     for (int i = 1; i <= n; i++) scanf("%d", &c[i]);
49     for (int i = 1; i <= m; i++) {
50         char op[10]; scanf("%s", op);
51         if (op[0] == 'Q') {
52             int ql, qr; scanf("%d%d", &ql, &qr); totq++;
53             q[totq] = Query{ql, qr, totm, totq};
54         }

```

```

55     else {
56         int pos, val; scanf("%d%d", &pos, &val); totm++;
57         modify[totm] = Modify{pos, val};
58     }
59 }
60
61 //size = N ^ (2 / 3), (N * totm) ^ (1 / 3)
62 size = ceil(pow(n, (long double)2.0 / 3)); int num = ceil((long double)n / size);
63 for (int i = 1, j = 1; i <= num; i++)
64     while (j <= i * size && j <= n)
65         belong[j++] = i;
66
67 sort(q + 1, q + 1 + totq);
68
69 int l = 1, r = 0, t = 0;
70 for (int i = 1; i <= totq; i++) {
71     int ql = q[i].l, qr = q[i].r, qt = q[i].t;
72     while (l < ql) del(l++);
73     while (l > ql) add(--l);
74     while (r < qr) add(++r);
75     while (r > qr) del(r--);
76     while (t < qt) upd(++t, ql, qr);
77     while (t > qt) upd(t--, ql, qr);
78     ans[q[i].id] = res;
79 }
80
81 for (int i = 1; i <= totq; i++) printf("%d\n", ans[i]);
82
83 return 0;
84 }

```

8.3 rollbackMO

```

1 //问题可以莫队（询问可以离线，不带修改）
2 //区间伸长的时候很好维护信息
3 //区间缩短的时候不太好维护信息（如最大值，删除以后不知道次大值是多少）
4 // O(nsqrt(n))
5
6 struct Hash {
7     int b[N], tot;
8     void init() { tot = 0; }
9     void insert(int x) { b[++tot] = x; }
10    void build() {
11        sort(b + 1, b + 1 + tot);
12        tot = unique(b + 1, b + 1 + tot) - (b + 1);
13    }
14    int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
15 } ha;
16
17 int n, m;
18 int c[N], pos[N], cnt[N], cntt[N];
19 int belong[N], sizz;
20 ll ans[N], res;
21
22 struct Query {
23     int l, r, id;
24     bool operator < (const Query &rhs) const {
25         return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] : r < rhs.r;

```

```

26     }
27 } q[N];
28
29 ll bruteForce(int ql, int qr) {
30     ll result = 0;
31     for (int i = ql; i <= qr; i++) {
32         cntt[pos[i]]++;
33         result = max(result, 1ll * c[i] * cntt[pos[i]]);
34     }
35     for (int i = ql; i <= qr; i++) cntt[pos[i]]--;
36     return result;
37 }
38
39 void add(int x) {
40     cnt[pos[x]]++;
41     res = max(res, 1ll * c[x] * cnt[pos[x]]);
42 }
43
44 void del(int x) {
45     cnt[pos[x]]--;
46 }
47
48 int main() {
49
50
51     scanf("%d%d", &n, &m);
52     for (int i = 1; i <= n; i++) scanf("%d", &c[i]), ha.insert(c[i]);
53     ha.build();
54     for (int i = 1; i <= n; i++) pos[i] = ha.pos(c[i]);
55
56     sizz = sqrt(n); int num = ceil((long double)n / sizz);
57     for (int i = 1, j = 1; i <= num; i++)
58         while (j <= i * sizz && j <= n)
59             belong[j++] = i;
60
61     for (int i = 1; i <= m; i++) scanf("%d%d", &q[i].l, &q[i].r), q[i].id = i;
62     sort(q + 1, q + 1 + m);
63
64     for (int i = 1, j = 1; i <= num; i++) {
65         memset(cnt, 0, sizeof(cnt));
66         int right = min(i * sizz, n);
67         res = 0;
68         for (int l = right + 1, r = right; j <= m && belong[q[j].l] == i; j++, l =
right + 1) {
69             int ql = q[j].l, qr = q[j].r;
70             if (qr - ql + 1 <= sizz) {
71                 ans[q[j].id] = bruteForce(ql, qr);
72                 continue;
73             }
74             while (r < qr) add(++r);
75             ll tmp = res;
76             while (l > ql) add(--l);
77             ans[q[j].id] = res;
78             res = tmp;
79             while (l < right + 1) del(l++);
80         }
81     }
82
83     for (int i = 1; i <= m; i++) printf("%lld\n", ans[i]);

```

```

84
85     return 0;
86 }

```

8.4 treeMO

```

1  const int N = 1e5 + 10;
2
3  struct Hash {
4      int b[N], tot;
5      void init() { tot = 0; }
6      void insert(int x) { b[++tot] = x; }
7      void build() {
8          sort(b + 1, b + 1 + tot);
9          tot = unique(b + 1, b + 1 + tot) - (b + 1);
10     }
11     int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
12 } ha;
13
14 int n, m;
15 int c[N], cnt[N];
16 vector<int> g[N];
17 int st[N], ed[N], dfnt, nodeOf[N << 1], tag[N];
18 int belong[N], sizz;
19 int ans[N], res;
20
21 struct Query {
22     int l, r, id, k;
23     bool operator < (const Query &rhs) const {
24         return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] : r < rhs.r;
25     }
26 } q[N];
27
28 int son[N], siz[N], top[N], fa[N], dep[N];
29 void dfs(int u, int par) {
30     dep[u] = dep[fa[u] = par] + (siz[u] = 1);
31     int max_son = -1; nodeOf[st[u] = ++dfnt] = u;
32     for (auto &v: g[u])
33         if (v != par) {
34             dfs(v, u);
35             siz[u] += siz[v];
36             if (max_son < siz[v])
37                 son[u] = v, max_son = siz[v];
38         }
39     nodeOf[ed[u] = ++dfnt] = u;
40 }
41 void dfs2(int u, int topf) {
42     top[u] = topf;
43     if (!son[u]) return;
44     dfs2(son[u], topf);
45     for (auto &v: g[u])
46         if (v != fa[u] && v != son[u]) dfs2(v, v);
47 }
48 int lca(int x, int y) {
49     while (top[x] != top[y]) {
50         if (dep[top[x]] < dep[top[y]]) swap(x, y);
51         x = fa[top[x]];
52     }

```

```

53     return dep[x] < dep[y] ? x : y;
54 }
55
56 void upd(int x) {
57     x = nodeOf[x];
58     if (tag[x]) {
59         cnt[c[x]]--;
60         if (!cnt[c[x]]) res--;
61     }
62     else {
63         if (!cnt[c[x]]) res++;
64         cnt[c[x]]++;
65     }
66     tag[x] ^= 1;
67 }
68
69 int main() {
70     #ifdef ACM_LOCAL
71         freopen("input.in", "r", stdin);
72         freopen("output.out", "w", stdout);
73     #endif
74     scanf("%d%d", &n, &m);
75     for (int i = 1; i <= n; i++) scanf("%d", &c[i]), ha.insert(c[i]);
76     ha.build();
77     for (int i = 1; i <= n; i++) c[i] = ha.pos(c[i]);
78
79     for (int i = 1; i < n; i++) {
80         int u, v; scanf("%d%d", &u, &v);
81         g[u].push_back(v); g[v].push_back(u);
82     }
83     int rt = 1; dfs(rt, 0); dfs2(rt, rt);
84
85     sizz = sqrt(dfnt); int num = ceil((long double)dfnt / sizz);
86     for (int i = 1, j = 1; i <= num; i++)
87         while (j <= i * sizz && j <= dfnt)
88             belong[j++] = i;
89     for (int i = 1; i <= m; i++) {
90         int u, v; scanf("%d%d", &u, &v);
91         int tlca = lca(u, v);
92         if (st[u] > st[v]) swap(u, v);
93         if (u == tlca) q[i] = Query{st[u], st[v], i, 0};
94         else q[i] = Query{ed[u], st[v], i, tlca};
95     }
96     sort(q + 1, q + 1 + m);
97
98     int l = 1, r = 0;
99     for (int i = 1; i <= m; i++) {
100         int ql = q[i].l, qr = q[i].r;
101         while (l < ql) upd(l++);
102         while (l > ql) upd(--l);
103         while (r < qr) upd(++r);
104         while (r > qr) upd(r--);
105         ans[q[i].id] = res + (q[i].k ? (cnt[c[q[i].k]] == 0) : 0);
106     }
107
108     for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
109
110     return 0;

```

112 }

8.5 fastIO

```

1 namespace FastIO {
2     char gc(void) {
3         const int S = 1 << 17;
4         static char buf[S], *s = buf, *t = buf;
5         if (s == t) t = buf + fread(s = buf, 1, S, stdin);
6         if (s == t) return EOF;
7         return *s++;
8     }
9
10    int read(void) {
11        int a = 0, b = 1, c = gc();
12        for (; !isdigit(c); c = gc()) b ^= (c == '-');
13        for (; isdigit(c); c = gc()) a = a * 10 + c - '0';
14        return b ? a : -a;
15    }
16 }
17 using namespace FastIO;
18
19
20 static struct FastInput {
21     static constexpr int BUF_SIZE = 1 << 20;
22     char buf[BUF_SIZE];
23     size_t chars_read = 0;
24     size_t buf_pos = 0;
25     FILE *in = stdin;
26     char cur = 0;
27
28     inline char get_char() {
29         if (buf_pos >= chars_read) {
30             chars_read = fread(buf, 1, BUF_SIZE, in);
31             buf_pos = 0;
32             buf[0] = (chars_read == 0 ? -1 : buf[0]);
33         }
34         return cur = buf[buf_pos++];
35     }
36
37     inline void tie(int) {}
38
39     inline explicit operator bool() {
40         return cur != -1;
41     }
42
43     inline static bool is_blank(char c) {
44         return c <= ' ';
45     }
46
47     inline bool skip_blanks() {
48         while (is_blank(cur) && cur != -1) {
49             get_char();
50         }
51         return cur != -1;
52     }
53
54     inline FastInput& operator>>(char& c) {

```



```

55     skip_blanks();
56     c = cur;
57     return *this;
58 }
59
60 inline FastInput& operator>>(std::string& s) {
61     if (skip_blanks()) {
62         s.clear();
63         do {
64             s += cur;
65         } while (!is_blank(get_char()));
66     }
67     return *this;
68 }
69
70 template <typename T>
71 inline FastInput& read_integer(T& n) {
72     // unsafe, doesn't check that characters are actually digits
73     n = 0;
74     if (skip_blanks()) {
75         int sign = +1;
76         if (cur == '-') {
77             sign = -1;
78             get_char();
79         }
80         do {
81             n += n + (n << 3) + cur - '0';
82         } while (!is_blank(get_char()));
83         n *= sign;
84     }
85     return *this;
86 }
87
88 template <typename T>
89 inline typename std::enable_if<std::is_integral<T>::value, FastInput&>::type
operator>>(T& n) {
90     return read_integer(n);
91 }
92
93 #if !defined(_WIN32) || defined(_WIN64)
94 inline FastInput& operator>>(__int128& n) {
95     return read_integer(n);
96 }
97 #endif
98
99 template <typename T>
100 inline typename std::enable_if<std::is_floating_point<T>::value, FastInput&>::type
operator>>(T& n) {
101     // not sure if really fast, for compatibility only
102     n = 0;
103     if (skip_blanks()) {
104         std::string s;
105         (*this) >> s;
106         sscanf(s.c_str(), "%lf", &n);
107     }
108     return *this;
109 }
110 } fast_input;
111

```

```

112 #define cin fast_input
113
114 static struct FastOutput {
115     static constexpr int BUF_SIZE = 1 << 20;
116     char buf[BUF_SIZE];
117     size_t buf_pos = 0;
118     static constexpr int TMP_SIZE = 1 << 20;
119     char tmp[TMP_SIZE];
120     FILE *out = stdout;
121
122     inline void put_char(char c) {
123         buf[buf_pos++] = c;
124         if (buf_pos == BUF_SIZE) {
125             fwrite(buf, 1, buf_pos, out);
126             buf_pos = 0;
127         }
128     }
129
130     ~FastOutput() {
131         fwrite(buf, 1, buf_pos, out);
132     }
133
134     inline FastOutput& operator<<(char c) {
135         put_char(c);
136         return *this;
137     }
138
139     inline FastOutput& operator<<(const char* s) {
140         while (*s) {
141             put_char(*s++);
142         }
143         return *this;
144     }
145
146     inline FastOutput& operator<<(const std::string& s) {
147         for (int i = 0; i < (int) s.size(); i++) {
148             put_char(s[i]);
149         }
150         return *this;
151     }
152
153     template <typename T>
154     inline char* integer_to_string(T n) {
155         // beware of TMP_SIZE
156         char* p = tmp + TMP_SIZE - 1;
157         if (n == 0) {
158             *--p = '0';
159         } else {
160             bool is_negative = false;
161             if (n < 0) {
162                 is_negative = true;
163                 n = -n;
164             }
165             while (n > 0) {
166                 *--p = (char) ('0' + n % 10);
167                 n /= 10;
168             }
169             if (is_negative) {
170                 *--p = '-';

```

```

171     }
172 }
173     return p;
174 }
175
176 template <typename T>
177 inline typename std::enable_if<std::is_integral<T>::value, char*>::type stringify(T
    n) {
178     return integer_to_string(n);
179 }
180
181 #if !defined(_WIN32) || defined(_WIN64)
182 inline char* stringify(__int128 n) {
183     return integer_to_string(n);
184 }
185 #endif
186
187 template <typename T>
188 inline typename std::enable_if<std::is_floating_point<T>::value, char*>::type
    stringify(T n) {
189     sprintf(tmp, "%.17f", n);
190     return tmp;
191 }
192
193 template <typename T>
194 inline FastOutput& operator<<(const T& n) {
195     auto p = stringify(n);
196     for (; *p != 0; p++) {
197         put_char(*p);
198     }
199     return *this;
200 }
201 } fast_output;
202
203 #define cout fast_output

```

8.6 simulatedAnnealing

```

1  const double DOWN = 0.996;
2  const double START_T = 5000;
3  double ansx, ansy, ansz, anse;
4
5  void initAns() {
6      //初始化一个答案点(可以选任意点)
7  }
8
9  double getEnergy(double x, double y, double z) {
10     //具体分析题目
11 }
12
13 void SA() {
14     double T = START_T;
15     while (T > 1e-15) {
16         double newx = ansx + (rand() * 2 - RAND_MAX) * T;
17         double newy = ansy + (rand() * 2 - RAND_MAX) * T;
18         double newz = ansz + (rand() * 2 - RAND_MAX) * T;
19         double newe = getEnergy(newx, newy, newz);
20         double delta = newe - anse;

```

```

21         if (delta < 0) ansx = newx, ansy = newy, ansz = newz, anse = newe;
22         else if (exp(-delta / T) * RAND_MAX > rand())
23             ansx = newx, ansy = newy, ansz = newz;
24         T *= DOWN;
25     }
26 }
27
28 void solve() {
29     initAns();
30     while ((double) clock() / CLOCKS_PER_SEC < 2.0) SA();
31 }

```

8.7 Cantor

```

1
2 //主要应用为将N维的排列状态压缩成数字id
3 //然后需要知道具体状态时用逆Cantor得到
4
5 int N;
6 int a[MAX], c[MAX];
7
8 void upd(int p, int k) { for (; p <= N; p += lowbit(p)) c[p] += k; }
9 int query(int p) {
10     int res = 0;
11     for (; p; p -= lowbit(p)) res += c[p];
12     return res;
13 }
14
15 int cantor() {
16     //ans = 1 + \sum_{i=1}^N fac[N-i] * (\sum_{j=i+1}^N x[j] > x[i])
17     int res = 0, fac = 1;
18     for (int i = N; i >= 1; i--) {
19         upd(a[i], 1);
20         res = (res + 1ll * fac * query(a[i] - 1) % mod) % mod;
21         fac = 1ll * fac * (N - i + 1) % mod;
22     }
23     return res + 1;
24 }
25
26 //逆Cantor
27 #define lc u<<1
28 #define rc u<<1|1
29 #define mid (l+r)/2
30 int sum[MAX << 4];
31 void push_up(int u) { sum[u] = sum[lc] + sum[rc]; }
32 void build(int u, int l, int r) {
33     if (l == r) {
34         sum[u] = 1;
35         return;
36     }
37     build(lc, l, mid); build(rc, mid + 1, r);
38     push_up(u);
39 }
40 int query(int u, int l, int r, int k) { //查找第k大并且删除该数
41     sum[u]--;
42     if (l == r) return l;
43     if (k <= sum[lc]) return query(lc, l, mid, k);
44     else return query(rc, mid + 1, r, k - sum[lc]);

```

```

45 }
46
47 vector<int> inCantor(int x, int n) {
48     x--;
49     vector<int> res;
50     ll fac = 1;
51     build(1, 1, n);
52     for (int i = 1; i <= n; i++) fac = fac * i;
53     for (int i = 1; i <= n; i++) {
54         fac = fac / (n - i + 1);
55         int k = x / fac + 1; //比当前这位大的有x / fac位
56         res.push_back(query(1, 1, n, k)); //找到没被选的第k大
57         x %= fac;
58     }
59     return res;
60 }

```

8.8 BigInteger

```

1 struct BigInteger {
2     typedef unsigned long long LL;
3
4     static const int BASE = 1000000000;
5     static const int WIDTH = 8;
6     vector<int> s;
7
8     BigInteger& clean() { while (!s.back() && s.size() > 1) s.pop_back(); return *this; }
9     BigInteger(LL num = 0) { *this = num; }
10    BigInteger(string s) { *this = s; }
11    BigInteger& operator = (long long num) {
12        s.clear();
13        do {
14            s.push_back(num % BASE);
15            num /= BASE;
16        } while (num > 0);
17        return *this;
18    }
19    BigInteger& operator = (const string& str) {
20        s.clear();
21        int x, len = (str.length() - 1) / WIDTH + 1;
22        for (int i = 0; i < len; i++) {
23            int end = str.length() - i * WIDTH;
24            int start = max(0, end - WIDTH);
25            sscanf(str.substr(start, end - start).c_str(), "%d", &x);
26            s.push_back(x);
27        }
28        return (*this).clean();
29    }
30
31    BigInteger operator + (const BigInteger& b) const {
32        BigInteger c; c.s.clear();
33        for (int i = 0, g = 0; ; i++) {
34            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
35            int x = g;
36            if (i < s.size()) x += s[i];
37            if (i < b.s.size()) x += b.s[i];
38            c.s.push_back(x % BASE);
39            g = x / BASE;

```

```

40     }
41     return c;
42 }
43 BigInteger operator - (const BigInteger& b) const {
44     assert(b <= *this); // 减数不能大于被减数
45     BigInteger c; c.s.clear();
46     for (int i = 0, g = 0; ; i++) {
47         if (g == 0 && i >= s.size() && i >= b.s.size()) break;
48         int x = s[i] + g;
49         if (i < b.s.size()) x -= b.s[i];
50         if (x < 0) { g = -1; x += BASE; }
51         else g = 0;
52         c.s.push_back(x);
53     }
54     return c.clean();
55 }
56 BigInteger operator * (const BigInteger& b) const {
57     int i, j; LL g;
58     vector<LL> v(s.size() + b.s.size(), 0);
59     BigInteger c; c.s.clear();
60     for (i = 0; i < s.size(); i++) for (j = 0; j < b.s.size(); j++) v[i + j] += LL(s[i]
61 ) * b.s[j];
62     for (i = 0, g = 0; ; i++) {
63         if (g == 0 && i >= v.size()) break;
64         LL x = v[i] + g;
65         c.s.push_back(x % BASE);
66         g = x / BASE;
67     }
68     return c.clean();
69 }
70 BigInteger operator / (const BigInteger& b) const {
71     assert(b > 0); // 除数必须大于0
72     BigInteger c = *this; // 商:主要是让c.s和(*this).s的vector一样大
73     BigInteger m; // 余数:初始化为0
74     for (int i = s.size() - 1; i >= 0; i--) {
75         m = m * BASE + s[i];
76         c.s[i] = bsearch(b, m);
77         m -= b * c.s[i];
78     }
79     return c.clean();
80 }
81 BigInteger operator % (const BigInteger& b) const { //方法与除法相同
82     BigInteger c = *this;
83     BigInteger m;
84     for (int i = s.size() - 1; i >= 0; i--) {
85         m = m * BASE + s[i];
86         c.s[i] = bsearch(b, m);
87         m -= b * c.s[i];
88     }
89     return m;
90 }
91 // 二分法找出满足bx<=m的最大的x
92 int bsearch(const BigInteger& b, const BigInteger& m) const {
93     int L = 0, R = BASE - 1, x;
94     while (1) {
95         x = (L + R) >> 1;
96         if (b * x <= m) { if (b * (x + 1) > m) return x; else L = x; }
97         else R = x;
98     }
99 }

```

```

98     }
99     BigInteger& operator += (const BigInteger& b) { *this = *this + b; return *this; }
100    BigInteger& operator -= (const BigInteger& b) { *this = *this - b; return *this; }
101    BigInteger& operator *= (const BigInteger& b) { *this = *this * b; return *this; }
102    BigInteger& operator /= (const BigInteger& b) { *this = *this / b; return *this; }
103    BigInteger& operator %= (const BigInteger& b) { *this = *this % b; return *this; }
104
105    bool operator < (const BigInteger& b) const {
106        if (s.size() != b.s.size()) return s.size() < b.s.size();
107        for (int i = s.size() - 1; i >= 0; i--)
108            if (s[i] != b.s[i]) return s[i] < b.s[i];
109        return false;
110    }
111    bool operator > (const BigInteger& b) const { return b < *this; }
112    bool operator <= (const BigInteger& b) const { return !(b < *this); }
113    bool operator >= (const BigInteger& b) const { return !(*this < b); }
114    bool operator != (const BigInteger& b) const { return b < *this || *this < b; }
115    bool operator == (const BigInteger& b) const { return !(b < *this) && !(b > *this); }
116 };
117
118 ostream& operator << (ostream& out, const BigInteger& x) {
119     out << x.s.back();
120     for (int i = x.s.size() - 2; i >= 0; i--) {
121         char buf[20];
122         sprintf(buf, "%08d", x.s[i]);
123         for (int j = 0; j < strlen(buf); j++) out << buf[j];
124     }
125     return out;
126 }
127
128 istream& operator >> (istream& in, BigInteger& x) {
129     string s;
130     if (!(in >> s)) return in;
131     x = s;
132     return in;
133 }
134
135 int main()
136 {
137     int t;
138     scanf("%d", &t);
139     while (t--)
140     {
141         BigInteger a, b;
142         cin >> a >> b;
143         cout << a + b << endl;
144     }
145 }

```

8.9 debug

```

1  using std::string;
2  using std::to_string;
3  template <typename A, typename B>
4  std::string to_string(std::pair<A, B> p);
5
6  template <typename A, typename B, typename C>
7  std::string to_string(std::tuple<A, B, C> p);

```

```

8
9  template <typename A, typename B, typename C, typename D>
10  std::string to_string(std::tuple<A, B, C, D> p);
11
12  std::string to_string(const std::string& s) {
13      return "'" + s + "'";
14  }
15
16  std::string to_string(const char* s) {
17      return to_string((std::string) s);
18  }
19
20  std::string to_string(bool b) {
21      return (b ? "true" : "false");
22  }
23
24  std::string to_string(std::vector<bool> v) {
25      bool first = true;
26      std::string res = "{";
27      for (int i = 0; i < static_cast<int>(v.size()); i++) {
28          if (!first) {
29              res += ", ";
30          }
31          first = false;
32          res += to_string(v[i]);
33      }
34      res += "}";
35      return res;
36  }
37
38  template <size_t N>
39  std::string to_string(std::bitset<N> v) {
40      std::string res = "";
41      for (size_t i = 0; i < N; i++) {
42          res += static_cast<char>('0' + v[i]);
43      }
44      return res;
45  }
46
47  template <typename A>
48  std::string to_string(A v) {
49      bool first = true;
50      std::string res = "{";
51      for (const auto &x : v) {
52          if (!first) {
53              res += ", ";
54          }
55          first = false;
56          res += to_string(x);
57      }
58      res += "}";
59      return res;
60  }
61
62
63  template <typename A, typename B>
64  std::string to_string(std::pair<A, B> p) {
65      return "(" + to_string(p.first) + ", " + to_string(p.second) + ")";
66  }

```



```
67
68 template <typename A, typename B, typename C>
69 std::string to_string(std::tuple<A, B, C> p) {
70     return "(" + to_string(std::get<0>(p)) + ", " + to_string(std::get<1>(p)) + ", " +
        to_string(std::get<2>(p)) + ")";
71 }
72
73 template <typename A, typename B, typename C, typename D>
74 std::string to_string(std::tuple<A, B, C, D> p) {
75     return "(" + to_string(std::get<0>(p)) + ", " + to_string(std::get<1>(p)) + ", " +
        to_string(std::get<2>(p)) + ", " + to_string(std::get<3>(p)) + ")";
76 }
77
78 void debug_out() { std::cerr << std::endl; }
79
80 template <typename Head, typename... Tail>
81 void debug_out(Head H, Tail... T) {
82     std::cerr << " " << to_string(H);
83     debug_out(T...);
84 }
85
86 #ifdef LOCAL
87 #define debug(...) std::cerr << "[" << #__VA_ARGS__ << "]:", debug_out(__VA_ARGS__)
88 #else
89 #define debug(...) 42
90 #endif
```

8.10 int128

```
1 using i128 = __int128;
2
3 std::istream &operator>>(std::istream &is, i128 &n) {
4     n = 0;
5     std::string s;
6     is >> s;
7     for (auto c : s) {
8         n = 10 * n + c - '0';
9     }
10    return is;
11 }
12
13 std::ostream &operator<<(std::ostream &os, i128 n) {
14     if (n == 0) {
15         return os << 0;
16     }
17     std::string s;
18     while (n > 0) {
19         s += '0' + n % 10;
20         n /= 10;
21     }
22     std::reverse(s.begin(), s.end());
23     return os << s;
24 }
```