

ACM ICPC Team Reference - NTHU SplayTreap

Contents

1 Basic	2
1.1 vimrc	2
2 Data-Structure	2
2.1 CDQ	2
2.2 CHT	2
2.3 DLX	2
2.4 fast-set	2
2.5 lazysegtree	3
2.6 LCT	3
2.7 LiChao	4
2.8 rect-add-rect-sum	4
2.9 rollback-dsu	4
2.10 segtree-beats	4

2.11 segtree	5
2.12 sparse-table	5
2.13 static-range-inversion	5
2.14 static-range-lis	6
2.15 treap	6
2.16 union-of-rectangles	6
2.17 wavelet-tree	7
3 Flow-Matching	7
3.1 bipartite-matching	7
3.2 Dinic-LowerBound	7
3.3 Dinic	7
3.4 Flow 建模	8
3.5 general-matching	8
3.6 general-weighted-max- matching	8
3.7 KM	9
3.8 max-clique	9
3.9 MCMF	10
3.10 minimum-general-weighted- perfect-matching	10
4 Geometry	10
4.1 closest-pair	10
4.2 convex-hull	10
4.3 point-in-convex-hull	11
4.4 point	11
4.5 polar-angle-sort	11

5 Graph	11
5.1 2-SAT	11
5.2 centroid-tree	11
5.3 chromatic-number	11
5.4 HLD	11
5.5 lowlink	12
5.6 manhattan-mst	12
5.7 SCC	12
5.8 triangle-sum	12
6 Math	13
6.1 Aliens	13
6.2 Berlekamp-Massey	13
6.3 Chinese-Remainder	13
6.4 Combination	13
6.5 Determinant	13
6.6 Discrete-Log	13
6.7 extgcd	13
6.8 Floor-Sum	14
6.9 FWHT	14
6.10 Gauss-Jordan	14
6.11 Int-Div	14
6.12 Linear-Sieve	14
6.13 Miller-Rabin	14
6.14 Min-of-Mod-of-Linear	14
6.15 Mod-Inv	14
6.16 Pollard-Rho	14
6.17 Primes	15

6.18 Xor-Basis	15
6.19 估計值	15
6.20 定理	15
6.21 數字	16
6.22 歐幾里得類算法	16
6.23 生成函數	16
7 Misc	16
7.1 fast	16
7.2 for	16
7.3 next-combination	16
7.4 PBDS	16
7.5 python	16
7.6 rng	16
7.7 rotate90	16
7.8 timer	16
8 String	17
8.1 AC	17
8.2 KMP	17
8.3 LCP	17
8.4 manacher	17
8.5 rolling-hash	17
8.6 SAIS	17
8.7 SAM	17
8.8 smallest-rotation	17
8.9 Z	17

1 Basic

1.1 vimrc

```
1 se nu ai hls et ru ic is sc cul
2 se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
3 syntax on
4 hi cursorline cterm=none ctermbg=89
5 set bg=dark
6 inoremap {<CR> {<CR><Esc>ko<tab>
```

2 Data-Structure

2.1 CDQ

```
1 void CDQ(int l, int r) {
2     if(l + 1 == r) return;
3     int mid = (l + r) / 2;
4     CDQ(l, mid), CDQ(mid, r);
5     int i = l;
6     for(int j = mid; j < r; j++) {
7         const Q& q = qry[j];
8         while(i < mid && qry[i].x >= q.x) {
9             if(qry[i].id == -1) fenw.add(qry[i].y,
10                qry[i].w);
11             i++;
12         }
13         if(q.id >= 0) ans[q.id] += q.w * fenw.
14             sum(q.y, sz - 1);
15     }
16     for(int p = l; p < i; p++) if(qry[p].id ==
17         -1) fenw.add(qry[p].y, -qry[p].w);
18     inplace_merge(qry.begin() + l, qry.begin()
19         + mid, qry.begin() + r, [](const Q& a
20         , const Q& b) {
21         return a.x > b.x;
22     });
23 }
```

2.2 CHT

```
1 struct line_t {
2     mutable ll k, m, p;
3     bool operator<(const line_t& o) const {
4         return k < o.k; }
5     bool operator<(ll x) const { return p < x;
6         }
7 };
8 template<bool MAX>
9 struct CHT : multiset<line_t, less<>> {
10     const ll INF = 1e18;
11     bool isect(iterator x, iterator y) {
12         if(y == end()) return x->p = INF, 0;
13         if(x->k == y->k) {
14             x->p = (x->m > y->m ? INF : -INF);
```

```
13     } else {
14         x->p = floor_div(y->m - x->m, x->k - y
15             ->k); // see Math
16     }
17     return x->p >= y->p;
18 }
19 void add_line(ll k, ll m) {
20     if(!MAX) k = -k, m = -m;
21     auto z = insert({k, m, 0}), y = z++, x =
22         y;
23     while(isect(y, z)) z = erase(z);
24     if(x != begin() && isect(--x, y)) isect(
25         x, y = erase(y));
26     while((y = x) != begin() && (--x)->p >=
27         y->p) isect(x, erase(y));
28 }
29 ll get(ll x) {
30     assert(!empty());
31     auto l = *lower_bound(x);
32     return (l.k * x + l.m) * (MAX ? +1 : -1)
33     ;
34 }
```

2.3 DLX

```
1 struct DLX {
2     int n, m, tot, ans;
3     vi first, siz, L, R, U, D, col, row, stk;
4     DLX(int _n, int _m) : n(_n), m(_m), tot(_m
5         ) {
6         int sz = n * m;
7         first = siz = L = R = U = D = col = row
8             = stk = vi(sz);
9         REP(i, m + 1) {
10             L[i] = i - 1, R[i] = i + 1;
11             U[i] = D[i] = i;
12         }
13         L[0] = m, R[m] = 0;
14     }
15     void insert(int r, int c) {
16         r++, c++;
17         col[++tot] = c, row[tot] = r, ++siz[c];
18         D[tot] = D[c], U[D[c]] = tot, U[tot] = c
19             , D[c] = tot;
20         if(!first[r]) first[r] = L[tot] = R[tot]
21             = tot;
22         else {
23             L[R[tot]] = R[first[r]] = tot;
24             R[L[tot]] = first[r] = tot;
25         }
26     }
27     #define TRAV(i, X, j) for(i = X[j]; i != j
28         ; i = X[i])
29     void remove(int c) {
30         int i, j;
31         L[R[c]] = L[c], R[L[c]] = R[c];
32         TRAV(i, D, c) TRAV(j, R, i) {
33             D[U[D[j]]] = U[j] = D[j];
34             siz[col[j]]--;
35         }
36     }
37     void recover(int c) {
```

```
33     int i, j;
34     TRAV(i, U, c) TRAV(j, L, i) {
35         U[D[j]] = D[U[j]] = j;
36         siz[col[j]]++;
37     }
38     L[R[c]] = R[L[c]] = c;
39 }
40 bool dance(int dep) {
41     if(!R[0]) return ans = dep, true;
42     int i, j, c = R[0];
43     TRAV(i, R, 0) if(siz[i] < siz[c]) c = i;
44     remove(c);
45     TRAV(i, D, c) {
46         stk[dep] = row[i];
47         TRAV(j, R, i) remove(col[j]);
48         if(dance(dep + 1)) return true;
49         TRAV(j, L, i) recover(col[j]);
50     }
51     recover(c);
52     return false;
53 }
54 vi solve() {
55     if(!dance(1)) return {};
56     return vi(stk.begin() + 1, stk.begin() +
57         ans);
58 }
```

2.4 fast-set

```
1 // Can correctly work with numbers in range
2 // [0; MAXN]
3 // Supports all std::set operations in O(1)
4 // on random queries / dense arrays, O(
5 // log64(N)) in worst case (sparse array).
6 // Count operation works in O(1) always.
7 template<uint MAXN>
8 class fast_set {
9 private:
10     static const uint PREF = (MAXN <= 64 ? 0 :
11         MAXN <= 4096 ? 1 :
12         MAXN <= 262144 ? 1 + 64 :
13         MAXN <= 16777216 ? 1 + 64 +
14             4096 :
15         MAXN <= 1073741824 ? 1 + 64
16             + 4096 + 262144 : 227) +
17         1;
18     static constexpr ull lb(int x) {
19         if(x == 64) return ULLONG_MAX;
20         return (1ULL << x) - 1;
21     };
22     static const uint SZ = PREF + (MAXN + 63)
23         / 64 + 1;
24     ull m[SZ] = {0};
25     inline uint left(uint v) const { return (v
26         - 62) * 64; }
27     inline uint parent(uint v) const { return
28         v / 64 + 62; }
29     inline void setbit(uint v) { m[v >> 6] |=
30         1ULL << (v & 63); }
31     inline void resetbit(uint v) { m[v >> 6]
32         &= ~(1ULL << (v & 63)); }
```

```
22 inline uint getbit(uint v) const { return
23     m[v >> 6] >> (v & 63) & 1; }
24 inline ull childs_value(uint v) const {
25     return m[left(v) >> 6]; }
26 inline int left_go(uint x, const uint c)
27     const {
28     const ull rem = x & 63;
29     uint bt = PREF * 64 + x;
30     ull num = m[bt >> 6] & lb(rem + c);
31     if(num) return (x ^ rem) | __lg(num);
32     for(bt = parent(bt); bt > 62; bt =
33         parent(bt)) {
34         const ull rem = bt & 63;
35         num = m[bt >> 6] & lb(rem);
36         if(num) {
37             bt = (bt ^ rem) | __lg(num);
38             break;
39         }
40     }
41     if(bt == 62) return -1;
42     while(bt < PREF * 64) bt = left(bt) |
43         __lg(m[bt - 62]);
44     return bt - PREF * 64;
45 }
46 inline int right_go(uint x, const uint c)
47     const {
48     const ull rem = x & 63;
49     uint bt = PREF * 64 + x;
50     ull num = m[bt >> 6] & ~lb(rem + c);
51     if(num) return (x ^ rem) |
52         __builtin_ctzll(num);
53     for(bt = parent(bt); bt > 62; bt =
54         parent(bt)) {
55         const ull rem = bt & 63;
56         num = m[bt >> 6] & ~lb(rem + 1);
57         if(num) {
58             bt = (bt ^ rem) | __builtin_ctzll(
59                 num);
60             break;
61         }
62     }
63     if(bt == 62) return -1;
64     while(bt < PREF * 64) bt = left(bt) |
65         __builtin_ctzll(m[bt - 62]);
66     return bt - PREF * 64;
67 }
68 public:
69 fast_set() { assert(PREF != 228); setbit
70     (62); }
71 bool empty() const {return getbit(63);}
72 void clear() { fill(m, m + SZ, 0); setbit
73     (62); }
74 bool count(uint x) const { return m[PREF +
75     (x >> 6)] >> (x & 63) & 1; }
76 void insert(uint x) { for(uint v = PREF *
77     64 + x; !getbit(v); v = parent(v))
78     setbit(v); }
79 void erase(uint x) {
80     if(!getbit(PREF * 64 + x)) return;
81     resetbit(PREF * 64 + x);
82     for(uint v = parent(PREF * 64 + x); v >
83         62 && !childs_value(v); v = parent(v)
84         ) resetbit(v);
85 }
```

```

70 int find_next(uint x) const { return
    right_go(x, 0); } // >=
71 int find_prev(uint x) const { return
    left_go(x, 1); } // <=
72 };

```

2.5 lazysegtree

```

1 template<class S,
2     S (*e)(),
3     S (*op)(S, S),
4     class F,
5     F (*id)(),
6     S (*mapping)(F, S),
7     F (*composition)(F, F)>
8 struct lazy_segtree {
9     int n, size, log;
10    vector<S> d; vector<F> lz;
11    void update(int k) { d[k] = op(d[k << 1],
12    d[k << 1 | 1]); }
13    void all_apply(int k, F f) {
14    d[k] = mapping(f, d[k]);
15    if(k < size) lz[k] = composition(f, lz[k]
16    );}
17    void push(int k) {
18    all_apply(k << 1, lz[k]);
19    all_apply(k << 1 | 1, lz[k]);
20    lz[k] = id();
21    }
22    lazy_segtree(int _n) : lazy_segtree(vector
23    <S>(_n, e())) {}
24    lazy_segtree(const vector<S>& v) : n(sz(v)
25    ) {
26    log = __lg(2 * n - 1), size = 1 << log;
27    d.resize(size * 2, e());
28    lz.resize(size, id());
29    REP(i, n) d[size + i] = v[i];
30    for(int i = size - 1; i; i--) update(i);
31    }
32    void set(int p, S x) {
33    p += size;
34    for(int i = log; i; --i) push(p >> i);
35    d[p] = x;
36    for(int i = 1; i <= log; ++i) update(p
37    >> i);
38    }
39    S get(int p) {
40    p += size;
41    for(int i = log; i; i--) push(p >> i);
42    return d[p];
43    }
44    S prod(int l, int r) {
45    if(l == r) return e();
46    l += size; r += size;
47    for(int i = log; i; i--) {
48    if(((l >> i) << i) != 1) push(l >> i);
49    if(((r >> i) << i) != 1) push(r >> i);
50    }
51    S sm1 = e(), smr = e();
52    while(l < r) {
53    if(l & 1) sm1 = op(sm1, d[l++]);
54    if(r & 1) smr = op(d[--r], smr);
55    l >>= 1;
56    r >>= 1;
57    }
58    return op(sm1, smr);
59    }
60    S all_prod() const { return d[1]; }
61    void apply(int p, F f) {
62    p += size;
63    for(int i = log; i; i--) push(p >> i);
64    d[p] = mapping(f, d[p]);
65    for(int i = 1; i <= log; i++) update(p
66    >> i);
67    }
68    void apply(int l, int r, F f) {
69    if(l == r) return;
70    l += size; r += size;
71    for(int i = log; i; i--) {
72    if(((l >> i) << i) != 1) push(l >> i);
73    if(((r >> i) << i) != 1) push((r - 1)
74    >> i);
75    }
76    {
77    int l2 = l, r2 = r;
78    while(l < r) {
79    if(l & 1) all_apply(l++, f);
80    if(r & 1) all_apply(--r, f);
81    l >>= 1;
82    r >>= 1;
83    }
84    l = l2;
85    r = r2;
86    }
87    for(int i = 1; i <= log; i++) {
88    if(((l >> i) << i) != 1) update(l >> i
89    );
90    if(((r >> i) << i) != 1) update((r -
91    1) >> i);
92    }
93    }
94    template<class G> int max_right(int l, G g)
95    {
96    assert(0 <= l && l <= n && g(e()));
97    if(l == n) return n;
98    l += size;
99    for(int i = log; i; i--) push(l >> i);
100    S sm = e();
101    do {
102    while(!(l & 1)) l >>= 1;
103    if(!g(op(sm, d[l]))) {
104    while(l < size) {
105    push(l);
106    l <<= 1;
107    if(g(op(sm, d[l]))) sm = op(sm, d[
108    l++]);
109    }
110    return l - size;
111    }
112    sm = op(sm, d[l++]);
113    } while((l & -1) != 1);
114    return n;
115    }
116    template<class G> int min_left(int r, G g)
117    {
118    assert(0 <= r && r <= n && g(e()));
119    if(r == 0) return 0;
120    r += size;

```

```

110 for(int i = log; i >= 1; i--) push((r -
111 1) >> i);
112 S sm = e();
113 do {
114    r--;
115    while(r > 1 && (r & 1)) r >>= 1;
116    if(!g(op(d[r], sm))) {
117    while(r < size) {
118    push(r);
119    r = r << 1 | 1;
120    if(g(op(d[r], sm))) sm = op(d[r
121    --], sm);
122    }
123    return r + 1 - size;
124    }
125    sm = op(d[r], sm);
126    } while((r & -r) != r);
127    return 0;
128    }
129    }

```

2.6 LCT

```

1 template<class S,
2     S (*e)(),
3     S (*op)(S, S),
4     S (*reversal)(S),
5     class F,
6     F (*id)(),
7     S (*mapping)(F, S),
8     F (*composition)(F, F)>
9 struct lazy_lct {
10    struct Node {
11    S val = e(), sum = e();
12    F lz = id();
13    bool rev = false;
14    int sz = 1;
15    Node *l = nullptr, *r = nullptr, *p =
16    nullptr;
17    Node() {}
18    Node(const S& s) : val(s), sum(s) {}
19    bool is_root() const { return p ==
20    nullptr || (p->l != this && p->r !=
21    this); }
22    };
23    int n;
24    vector<Node> a;
25    lazy_lct() : n(0) {}
26    explicit lazy_lct(int _n) : lazy_lct(
27    vector<S>(_n, e())) {}
28    explicit lazy_lct(const vector<S>& v) : n(
29    sz(v)) { REP(i, n) a.eb(v[i]); }
30    Node* access(int u) {
31    Node* v = &a[u];
32    Node* last = nullptr;
33    for(Node* p = v; p != nullptr; p = p->p)
34    splay(p, p->r = last, pull(last =
35    p));
36    splay(v);
37    return last;
38    }
39    void make_root(int u) { access(u), a[u].
40    rev ^= 1, push(&a[u]); }

```

```

33 void link(int u, int v) { make_root(v), a[
34 v].p = &a[u]; }
35 void cut(int u) {
36 access(u);
37 if(a[u].l != nullptr) a[u].l->p =
38 nullptr, a[u].l = nullptr, pull(&a[u
39 ]]);
40 }
41 void cut(int u, int v) { make_root(u), cut
42 (v); }
43 bool is_connected(int u, int v) {
44 if(u == v) return true;
45 return access(u), access(v), a[u].p !=
46 nullptr;
47 }
48 int get_lca(int u, int v) { return access(
49 u), access(v) - &a[0]; }
50 void set(int u, const S& s) { access(u), a
51 [u].val = s, pull(&a[u]); }
52 S get(int u) { return access(u), a[u].val;
53 }
54 void apply(int u, int v, const F& f) {
55 make_root(u), access(v), all_apply(&a[
56 v], f), push(&a[v]); }
57 S prod(int u, int v) { return make_root(u)
58 , access(v), a[v].sum; }
59 void rotate(Node* v) {
60 auto attach = [&](Node* p, bool side,
61 Node* c) {
62 (side ? p->r : p->l) = c;
63 pull(p);
64 if(c != nullptr) c->p = p;
65 };
66 Node *p = v->p, *g = p->p;
67 bool rgt = (p->r == v);
68 bool rt = p->is_root();
69 attach(p, rgt, (rgt ? v->l : v->r));
70 attach(v, !rgt, p);
71 if(!rt) attach(g, (g->r == p), v);
72 else v->p = g;
73 }
74 void splay(Node* v) {
75 push(v);
76 while(!v->is_root()) {
77 auto p = v->p;
78 auto g = p->p;
79 if(!p->is_root()) push(g);
80 push(p), push(v);
81 if(!p->is_root()) rotate((g->r == p)
82 == (p->r == v) ? p : v);
83 rotate(v);
84 }
85 }
86 void all_apply(Node* v, F f) {
87 v->val = mapping(f, v->val), v->sum =
88 mapping(f, v->sum);
89 v->lz = composition(f, v->lz);
90 }
91 void push(Node* v) {
92 if(v->lz != id()) {
93 if(v->l != nullptr) all_apply(v->l, v
94 ->lz);
95 if(v->r != nullptr) all_apply(v->r, v
96 ->lz);
97 v->lz = id();
98 }
99 }

```

```

83 if(v->rev) {
84     swap(v->l, v->r);
85     if(v->l != nullptr) v->l->rev ^= 1;
86     if(v->r != nullptr) v->r->rev ^= 1;
87     v->sum = reversal(v->sum);
88     v->rev = false;
89 }
90 void pull(Node* v) {
91     v->sz = 1;
92     v->sum = v->val;
93     if(v->l != nullptr) {
94         push(v->l);
95         v->sum = op(v->l->sum, v->sum);
96         v->sz += v->l->sz;
97     }
98     if(v->r != nullptr) {
99         push(v->r);
100        v->sum = op(v->sum, v->r->sum);
101        v->sz += v->r->sz;
102    }
103 }
104 }
105 };

```

2.7 LiChao

```

1 template<class T>
2 struct LiChao {
3     static constexpr T INF = numeric_limits<T>
4         ::max();
5     struct Line {
6         T a, b;
7         Line(T a, T b) : a(a), b(b) {}
8         T operator()(T x) const { return a * x + b; }
9     };
10    int n;
11    vector<Line> fs;
12    vector<T> xs;
13    LiChao(const vector<T>& xs_) {
14        xs = sort_unique(xs_);
15        n = SZ(xs);
16        fs.assign(2 * n, Line(T(0), INF));
17    }
18    int index(T x) const { return lower_bound(
19        ALL(xs), x) - xs.begin(); }
20    void add_line(T a, T b) { update(a, b, 0,
21        n); }
22    // [xl, xr) ax+b
23    void add_segment(T xl, T xr, T a, T b) {
24        int l = index(xl), r = index(xr);
25        update(a, b, l, r);
26    }
27    void update(T a, T b, int l, int r) {
28        Line g(a, b);
29        for(l += n, r += n; l < r; l >>= 1, r
30            >>= 1) {
31            if(l & 1) descend(g, l++);
32            if(r & 1) descend(g, --r);
33        }
34    }
35    void descend(Line g, int i) {
36        int l = i, r = i + 1;

```

```

33 while(l < n) l <= 1, r <= 1;
34 while(l < r) {
35     int c = (l + r) / 2;
36     T xl = xs[l - n], xr = xs[r - 1 - n],
37       xc = xs[c - n];
38     Line& f = fs[i];
39     if(f(xl) <= g(xl) && f(xr) <= g(xr))
40         return;
41     if(f(xl) >= g(xl) && f(xr) >= g(xr)) {
42         f = g; return; }
43     if(f(xc) > g(xc)) swap(f, g);
44     if(f(xl) > g(xl)) i = 2 * i, r = c;
45     else i = 2 * i + 1, l = c;
46 }
47 T get(T x) {
48     int i = index(x);
49     T res = INF;
50     for(i += n; i >= 1; i >>= 1) res = min(res,
51         fs[i](x));
52     return res;
53 }
54 }
55 };

```

2.8 rect-add-rect-sum

```

1 template<class Int, class T>
2 struct RectangleAddRectangleSum {
3     struct AQ { Int xl, xr, yl, yr; T val; };
4     struct SQ { Int xl, xr, yl, yr; };
5     vector<AQ> add_qry;
6     vector<SQ> sum_qry;
7     // A[x][y] += val for(x, y) in [xl, xr) *
8     // [yl, yr)
9     void add_rectangle(Int xl, Int xr, Int yl,
10        Int yr, T val) { add_qry.pb({xl, xr,
11        yl, yr, val}); }
12    // Get sum of A[x][y] for(x, y) in [xl, xr)
13    // * [yl, yr)
14    void add_query(Int xl, Int xr, Int yl, Int
15        yr) { sum_qry.pb({xl, xr, yl, yr}); }
16    vector<T> solve() {
17        vector<Int> ys;
18        for(auto &a : add_qry) ys.pb(a.yl), ys.
19            pb(a.yr);
20        ys = sort_unique(ys);
21        const int Y = SZ(ys);
22        vector<tuple<Int, int, int>> ops;
23        REP(q, SZ(sum_qry)) {
24            ops.pb(sum_qry[q].xl, 0, q);
25            ops.pb(sum_qry[q].xr, 1, q);
26        }
27        REP(q, SZ(add_qry)) {
28            ops.pb(add_qry[q].xl, 2, q);
29            ops.pb(add_qry[q].xr, 3, q);
30        }
31        sort(ALL(ops));
32        fenwick<T> b00(Y), b01(Y), b10(Y), b11(Y);
33        vector<T> ret(SZ(sum_qry));
34        for(auto o : ops) {
35            int qtype = get<1>(o), q = get<2>(o);
36            if(qtype >= 2) {

```

```

31 const auto& query = add_qry[q];
32 int i = lower_bound(ALL(ys), query.
33     yl) - ys.begin();
34 int j = lower_bound(ALL(ys), query.
35     yr) - ys.begin();
36 T x = get<0>(o);
37 T yi = query.yl, yj = query.yr;
38 if(qtype & 1) swap(i, j), swap(yi,
39     yj);
40 b00.add(i, x * yi * query.val);
41 b01.add(i, -x * query.val);
42 b10.add(i, -yi * query.val);
43 b11.add(i, query.val);
44 b00.add(j, -x * yj * query.val);
45 b01.add(j, x * query.val);
46 b10.add(j, yj * query.val);
47 b11.add(j, -query.val);
48 } else {
49     const auto& query = sum_qry[q];
50     int i = lower_bound(ALL(ys), query.
51         yl) - ys.begin();
52     int j = lower_bound(ALL(ys), query.
53         yr) - ys.begin();
54     T x = get<0>(o);
55     T yi = query.yl, yj = query.yr;
56     if(qtype & 1) swap(i, j), swap(yi,
57         yj);
58     ret[q] += b00.get(i - 1) + b01.get(i
59         - 1) * yi + b10.get(i - 1) * x
60         + b11.get(i - 1) * x * yi;
61     ret[q] -= b00.get(j - 1) + b01.get(j
62         - 1) * yj + b10.get(j - 1) * x
63         + b11.get(j - 1) * x * yj;
64 }
65 return ret;
66 }
67 }
68 };

```

2.9 rollback-dsu

```

1 struct RollbackDSU {
2     int n; vi sz, tag;
3     vector<tuple<int, int, int, int>> ops;
4     void init(int _n) {
5         n = _n;
6         sz.assign(n, -1);
7         tag.clear();
8     }
9     int leader(int x) {
10        while(sz[x] >= 0) x = sz[x];
11        return x;
12    }
13    bool merge(int x, int y) {
14        x = leader(x), y = leader(y);
15        if(x == y) return false;
16        if(-sz[x] < -sz[y]) swap(x, y);
17        op.pb(x, sz[x], y, sz[y]);
18        sz[x] += sz[y]; sz[y] = x;
19        return true;
20    }
21    int size(int x) { return -sz[leader(x)]; }
22    void add_tag() { tag.pb(sz(op)); }

```

```

23 void rollback() {
24     int z = tag.back(); tag.ppb();
25     while(sz(op) > z) {
26         auto [x, sx, y, sy] = op.back(); op.
27             ppb();
28         sz[x] = sx;
29         sz[y] = sy;
30     }
31 }
32 };

```

2.10 segtree-beats

```

1 struct segtree_beats {
2     static constexpr ll INF = numeric_limits<
3         ll>::max() / 2.1;
4     struct alignas(32) Node {
5         ll sum = 0, g1 = 0, l1 = 0;
6         ll g2 = -INF, gc = 1, l2 = INF, lc = 1,
7             add = 0;
8     };
9     ll n, log;
10    vector<Node> v;
11    segtree_beats() {}
12    segtree_beats(int _n) : segtree_beats(
13        vector<ll>(_n)) {}
14    segtree_beats(const vector<ll>& vc) {
15        n = 1, log = 0;
16        while(n < SZ(vc)) n <= 1, log++;
17        v.resize(2 * n);
18        REP(i, SZ(vc)) v[i + n].sum = v[i + n].
19            g1 = v[i + n].l1 = vc[i];
20        for(ll i = n - 1; i; --i) update(i);
21    }
22    void range_chmin(int l, int r, ll x) {
23        inner_apply<1>(l, r, x); }
24    void range_chmax(int l, int r, ll x) {
25        inner_apply<2>(l, r, x); }
26    void range_add(int l, int r, ll x) {
27        inner_apply<3>(l, r, x); }
28    void range_update(int l, int r, ll x) {
29        inner_apply<4>(l, r, x); }
30    ll range_min(int l, int r) { return
31        inner_fold<1>(l, r); }
32    ll range_max(int l, int r) { return
33        inner_fold<2>(l, r); }
34    ll range_sum(int l, int r) { return
35        inner_fold<3>(l, r); }
36    void update(int k) {
37        Node& p = v[k];
38        Node& l = v[k * 2];
39        Node& r = v[k * 2 + 1];
40        p.sum = l.sum + r.sum;
41        if(l.g1 == r.g1) {
42            p.g1 = l.g1;
43            p.g2 = max(l.g2, r.g2);
44            p.gc = l.gc + r.gc;
45        } else {
46            bool f = l.g1 > r.g1;
47            p.g1 = f ? l.g1 : r.g1;
48            p.gc = f ? l.gc : r.gc;
49            p.g2 = max(f ? r.g1 : l.g1, f ? l.g2 :
50                r.g2);

```

```

39 }
40 if(l.l1 == r.l1) {
41     p.l1 = l.l1;
42     p.l2 = min(l.l2, r.l2);
43     p.lc = l.lc + r.lc;
44 } else {
45     bool f = l.l1 < r.l1;
46     p.l1 = f ? l.l1 : r.l1;
47     p.lc = f ? l.lc : r.lc;
48     p.l2 = min(f ? r.l1 : l.l1, f ? l.l2 :
49                 r.l2);
50 }
51 void push_add(int k, ll x) {
52     Node& p = v[k];
53     p.sum += x << (log + __builtin_clz(k) -
54                 31);
55     p.g1 += x, p.l1 += x;
56     if(p.g2 != -INF) p.g2 += x;
57     if(p.l2 != INF) p.l2 += x;
58     p.add += x;
59 }
60 void push_min(int k, ll x) {
61     Node& p = v[k];
62     p.sum += (x - p.g1) * p.gc;
63     if(p.l1 == p.g1) p.l1 = x;
64     if(p.l2 == p.g1) p.l2 = x;
65     p.g1 = x;
66 }
67 void push_max(int k, ll x) {
68     Node& p = v[k];
69     p.sum += (x - p.l1) * p.lc;
70     if(p.g1 == p.l1) p.g1 = x;
71     if(p.g2 == p.l1) p.g2 = x;
72     p.l1 = x;
73 }
74 void push(int k) {
75     Node& p = v[k];
76     if(p.add != 0) {
77         push_add(k * 2, p.add);
78         push_add(k * 2 + 1, p.add);
79         p.add = 0;
80     }
81     if(p.g1 < v[k * 2].g1) push_min(k * 2, p
82                                     .g1);
83     if(p.l1 > v[k * 2].l1) push_max(k * 2, p
84                                     .l1);
85     if(p.g1 < v[k * 2 + 1].g1) push_min(k *
86                                     2 + 1, p.g1);
87     if(p.l1 > v[k * 2 + 1].l1) push_max(k *
88                                     2 + 1, p.l1);
89 }
90 void subtree_chmin(int k, ll x) {
91     if(v[k].g1 <= x) return;
92     if(v[k].g2 < x) {
93         push_min(k, x);
94         return;
95     }
96     push(k);
97     subtree_chmin(k * 2, x), subtree_chmin(k
98                 * 2 + 1, x);
99     update(k);
100 }
101 void subtree_chmax(int k, ll x) {
102     if(v[k].g1 >= x) return;
103     if(v[k].g2 > x) {
104         push_max(k, x);
105         return;
106     }
107     push(k);
108     subtree_chmax(k * 2, x), subtree_chmax(k
109                 * 2 + 1, x);
110     update(k);
111 }

```

```

98 if(x <= v[k].l1) return;
99 if(x < v[k].l2) {
100     push_max(k, x);
101     return;
102 }
103 push(k);
104 subtree_chmax(k * 2, x), subtree_chmax(k
105                 * 2 + 1, x);
106 update(k);
107 }
108 template<int cmd>
109 inline void _apply(int k, ll x) {
110     if constexpr(cmd == 1) subtree_chmin(k,
111                 x);
112     if constexpr(cmd == 2) subtree_chmax(k,
113                 x);
114     if constexpr(cmd == 3) push_add(k, x);
115     if constexpr(cmd == 4) subtree_chmin(k,
116                 x), subtree_chmax(k, x);
117 }
118 template<int cmd>
119 void inner_apply(int l, int r, ll x) {
120     if(l == r) return;
121     l += n, r += n;
122     for(int i = log; i >= 1; i--) {
123         if(((l >> i) << i) != 1) push(l >> i);
124         if(((r >> i) << i) != r) push((r - 1)
125                                     >> i);
126     }
127     {
128         int l2 = l, r2 = r;
129         while (l < r) {
130             if(l & 1) _apply<cmd>(l++, x);
131             if(r & 1) _apply<cmd>(--r, x);
132             l >>= 1, r >>= 1;
133         }
134         l = l2, r = r2;
135     }
136     for(int i = 1; i <= log; i++) {
137         if(((l >> i) << i) != 1) update(l >> i
138                                     );
139         if(((r >> i) << i) != r) update((r -
140                                     1) >> i);
141     }
142 }
143 template<int cmd>
144 inline ll e() {
145     if constexpr(cmd == 1) return INF;
146     if constexpr(cmd == 2) return -INF;
147     return 0;
148 }
149 template<int cmd>
150 inline void op(ll& a, const Node& b) {
151     if constexpr(cmd == 1) a = min(a, b.l1);
152     if constexpr(cmd == 2) a = max(a, b.g1);
153     if constexpr(cmd == 3) a += b.sum;
154 }
155 template<int cmd>
156 ll inner_fold(int l, int r) {
157     if(l == r) return e<cmd>();
158     l += n, r += n;
159     for(int i = log; i >= 1; i--) {
160         if(((l >> i) << i) != 1) push(l >> i);
161         if(((r >> i) << i) != r) push((r - 1)
162                                     >> i);
163     }

```

```

156 }
157 ll lx = e<cmd>(), rx = e<cmd>();
158 while (l < r) {
159     if(l & 1) op<cmd>(lx, v[l++]);
160     if(r & 1) op<cmd>(rx, v[--r]);
161     l >>= 1, r >>= 1;
162 }
163 if constexpr(cmd == 1) lx = min(lx, rx);
164 if constexpr(cmd == 2) lx = max(lx, rx);
165 if constexpr(cmd == 3) lx += rx;
166 return lx;
167 }
168 };

```

2.11 segtree

```

1 template<class S, S (*e)(), S (*op)(S, S)>
2 struct segtree {
3     int n, size, log;
4     vector<S> st;
5     void update(int v) { st[v] = op(st[v <<
6         1], st[v << 1 | 1]); }
7     segtree(int _n) : segtree(vector<S>(_n, e
8         ())) {}
9     segtree(const vector<S>& a) : n(sz(a)) {
10         log = __lg(2 * n - 1), size = 1 << log;
11         st.resize(size << 1, e());
12         REP(i, n) st[size + i] = a[i];
13         for(int i = size - 1; i; i--) update(i);
14     }
15     void set(int p, S val) {
16         st[p += size] = val;
17         for(int i = 1; i <= log; ++i) update(p
18             >> i);
19     }
20     S get(int p) const {
21         return st[p + size];
22     }
23     S prod(int l, int r) const {
24         assert(0 <= l && l <= r && r <= n);
25         S sml = e(), smr = e();
26         l += size, r += size;
27         while(l < r) {
28             if(l & 1) sml = op(sml, st[l++]);
29             if(r & 1) smr = op(st[--r], smr);
30             l >>= 1;
31             r >>= 1;
32         }
33         return op(sml, smr);
34     }
35     S all_prod() const { return st[1]; }
36     template<class F> int max_right(int l, F f
37         ) const {
38         assert(0 <= l && l <= n && f(e()));
39         if(l == n) return n;
40         l += size;
41         S sm = e();
42         do {
43             while(~l & 1) l >>= 1;
44             if(!f(op(sm, st[l]))) {
45                 while(l < size) {
46                     l <<= 1;

```

```

43         if(f(op(sm, st[l]))) sm = op(sm,
44             st[l++]);
45         }
46         return l - size;
47     }
48     sm = op(sm, st[l++]);
49     } while((l & -1) != 1);
50     return n;
51 }
52 template<class F> int min_left(int r, F f)
53 {
54     const {
55         assert(0 <= r && r <= n && f(e()));
56         if(r == 0) return 0;
57         r += size;
58         S sm = e();
59         do {
60             r--;
61             while(r > 1 && (r & 1)) r >>= 1;
62             if(!f(op(st[r], sm))) {
63                 while(r < size) {
64                     r = r << 1 | 1;
65                     if(f(op(st[r], sm))) sm = op(st[r]
66                         --, sm);
67                 }
68                 return r + 1 - size;
69             }
70             sm = op(st[r], sm);
71             } while((r & -r) != r);
72     }
73     return 0;
74 }

```

2.12 sparse-table

```

1 template<class T, T (*op)(T, T)>
2 struct sparse_table {
3     int n;
4     vector<vector<T>> b;
5     sparse_table(const vector<T>& a) : n(sz(a))
6     {
7         int lg = __lg(n) + 1;
8         b.resize(lg); b[0] = a;
9         for(int j = 1; j < lg; ++j) {
10             b[j].resize(n - (1 << j) + 1);
11             REP(i, n - (1 << j) + 1) b[j][i] = op(
12                 b[j - 1][i], b[j - 1][i + (1 << j)
13                     - 1]);
14         }
15     }
16     T prod(int from, int to) {
17         int lg = __lg(to - from + 1);
18         return op(b[lg][from], b[lg][to - (1 <<
19             lg) + 1]);
20     }
21 };

```

2.13 static-range-inversion

```

1 struct static_range_inversion {
2     int sz;

```

```

3 vi a, L, R;
4 vector<ll> ans;
5 static_range_inversion(vi _a) : a(_a) {
6     _a = sort_unique(_a);
7     REP(i, SZ(a)) a[i] = lower_bound(ALL(_a),
8         , a[i]) - _a.begin();
9     sz = SZ(_a);
10 }
11 void add_query(int l, int r) { L.push_back
12     (l), R.push_back(r); }
13 vector<ll> solve() {
14     const int q = SZ(L);
15     const int B = max(1.0, SZ(a) / sqrt(q));
16     vi ord(q);
17     iota(ALL(ord), 0);
18     sort(ALL(ord), [&](int i, int j) {
19         if(L[i] / B == L[j] / B) {
20             return L[i] / B & 1 ? R[i] > R[j] :
21                 R[i] < R[j];
22         }
23         return L[i] < L[j];
24     });
25     ans.resize(q);
26     fenwick<ll> fenw(sz + 1);
27     ll cnt = 0;
28     auto AL = [&](int i) {
29         cnt += fenw.sum(0, a[i] - 1);
30         fenw.add(a[i], +1);
31     };
32     auto AR = [&](int i) {
33         cnt += fenw.sum(a[i] + 1, sz);
34         fenw.add(a[i], +1);
35     };
36     auto DL = [&](int i) {
37         cnt -= fenw.sum(0, a[i] - 1);
38         fenw.add(a[i], -1);
39     };
40     auto DR = [&](int i) {
41         cnt -= fenw.sum(a[i] + 1, sz);
42         fenw.add(a[i], -1);
43     };
44     int l = 0, r = 0;
45     for(int i = 0; i < q; i++) {
46         int id = ord[i], ql = L[id], qr = R[id];
47         while(l > ql) AL(--l);
48         while(r < qr) AR(r++);
49         while(l < ql) DL(l--);
50         while(r > qr) DR(r--);
51         ans[id] = cnt;
52     }
53     return ans;
54 }
55 };

```

2.14 static-range-lis

```

1 struct static_range_lis {
2     int n;
3     vector<vector<pii>> qry;
4     vi invp, res_monge, ans;
5     static_range_lis(vi a) : n(SZ(a)), qry(n +
6         1, invp(n), res_monge(n) {

```

```

6     // a must be permutation of [0, n)
7     REP(i, n) invp[a[i]] = i;
8 }
9 void add_query(int l, int r) { qry[l].pb({
10     r, SZ(ans)}), ans.pb(r - 1); }
11 void unit_monge_mult(vi& a, vi& b, vi& r) {
12     {
13         int n = SZ(a);
14         if(n == 2) {
15             if(!a[0] && !b[0]) r[0] = 0, r[1] = 1;
16             else r[0] = 1, r[1] = 0;
17             return;
18         }
19         if(n == 1) return r[0] = 0, void();
20         int d = n / 2;
21         vi a1(d), a2(n - d), b1(d), b2(n - d);
22         vi mpa1(d), mpa2(n - d), mpb1(d), mpb2(n - d);
23         int p[2] = {};
24         REP(i, n) {
25             if(a[i] < d) a1[p[0]] = a[i], mpa1[p[0]++] = i;
26             else a2[p[1]] = a[i] - d, mpa2[p[1]++] = i;
27         }
28         p[0] = p[1] = 0;
29         REP(i, n) {
30             if(b[i] < d) b1[p[0]] = b[i], mpb1[p[0]++] = i;
31             else b2[p[1]] = b[i] - d, mpb2[p[1]++] = i;
32         }
33         vi c1(d), c2(n - d);
34         unit_monge_mult(a1, b1, c1);
35         unit_monge_mult(a2, b2, c2);
36         vi cpx(n), cpy(n), cqx(n), cqy(n);
37         REP(i, d) cpx[mpa1[i]] = mpb1[c1[i]],
38             cpy[mpa1[i]] = 0;
39         REP(i, n - d) cpx[mpa2[i]] = mpb2[c2[i]],
40             cpy[mpa2[i]] = 1;
41         REP(i, n) r[i] = cpx[i];
42         REP(i, n) cqx[cpx[i]] = i, cqy[cpx[i]] = cpy[i];
43         int hi = n, lo = n, his = 0, los = 0;
44         REP(i, n) {
45             if(cqy[i] ^ (cqx[i] >= hi)) his--;
46             while(hi > 0 && his < 0) {
47                 hi--;
48                 if(cpy[hi] ^ (cpx[hi] > i)) his++;
49             }
50             while(lo > 0 && los <= 0) {
51                 lo--;
52                 if(cpy[lo] ^ (cpx[lo] >= i)) los++;
53             }
54             if(los > 0 && hi == lo) r[lo] = i;
55             if(cqy[i] ^ (cqx[i] >= lo)) los--;
56         }
57     }
58     void subunit_monge_mult(vi& a, vi& b, vi& c) {
59         int n = SZ(a);
60         vi za(n), zb(n), res(n), vis(n), mpa(n, -1), mpb(n, -1);
61         int ca = n;
62         IREP(i, n) if(a[i] != -1) vis[a[i]] = 1,
63             za[--ca] = a[i], mpa[ca] = i;

```

```

64 IREP(i, n) if(!vis[i]) za[--ca] = i;
65 fill(ALL(vis), -1);
66 REP(i, n) if(b[i] != -1) vis[b[i]] = i;
67 ca = 0;
68 REP(i, n) if(vis[i] != -1) mpb[ca] = i,
69     rb[vis[i]] = ca++;
70 REP(i, n) if(rb[i] == -1) rb[i] = ca++;
71 REP(i, n) zb[rb[i]] = i;
72 unit_monge_mult(za, zb, res);
73 fill(ALL(c), -1);
74 REP(i, n) if(mpa[i] != -1 && mpb[res[i]]
75     != -1) c[mpa[i]] = mpb[res[i]];
76 }
77 void solve(vi& p, vi& ret) {
78     int n = SZ(p);
79     if(n == 1) return ret[0] = -1, void();
80     int d = n / 2;
81     vi pl(d), pr(n - d);
82     REP(i, d) pl[i] = p[i];
83     REP(i, n - d) pr[i] = p[i + d];
84     vi vis(n, -1);
85     REP(i, d) vis[pl[i]] = i;
86     vi tl(d), tr(n - d), mpl(d), mpr(n - d);
87     int ca = 0;
88     REP(i, n) if(vis[i] != -1) mpl[ca] = i,
89         tl[vis[i]] = ca++;
90     ca = 0;
91     fill(ALL(vis), -1);
92     REP(i, n - d) vis[pr[i]] = i;
93     REP(i, n) if(vis[i] != -1) mpr[ca] = i,
94         tr[vis[i]] = ca++;
95     vi vl(d), vr(n - d);
96     solve(tl, vl), solve(tr, vr);
97     vi sl(n), sr(n);
98     iota(ALL(sl), 0), iota(ALL(sr), 0);
99     REP(i, d) sl[mpl[i]] = (vl[i] == -1 ? -1 : mpl[vl[i]]);
100     REP(i, n - d) sr[mpr[i]] = (vr[i] == -1 ? -1 : mpr[vr[i]]);
101     subunit_monge_mult(sl, sr, ret);
102 }
103 vi solve() {
104     solve(invp, res_monge);
105     vi fenw(n + 1);
106     IREP(i, n) {
107         if(res_monge[i] != -1) {
108             for(int p = res_monge[i] + 1; p <= n; p += p & -p) fenw[p]++;
109         }
110         for(auto& z : qry[i]) {
111             auto [id, c] = z;
112             for(int p = id; p; p -= p & -p) ans[id] += fenw[p];
113         }
114     }
115     return ans;
116 }
117 };

```

2.15 treap

```

1 struct Node {
2     bool rev = false;

```

```

3     int sz = 1, pri = rng();
4     Node *l = NULL, *r = NULL, *p = NULL;
5     // TODO
6 }
7 void pull(Node& v) {
8     v->sz = 1 + size(v->l) + size(v->r);
9     // TODO
10 }
11 void push(Node& v) {
12     if(v->rev) {
13         swap(v->l, v->r);
14         if(v->l) v->l->rev ^= 1;
15         if(v->r) v->r->rev ^= 1;
16         v->rev = false;
17     }
18 }
19 Node* merge(Node* a, Node* b) {
20     if(!a || !b) return (a ? a : b);
21     push(a), push(b);
22     if(a->pri > b->pri) {
23         a->r = merge(a->r, b);
24         pull(a); return a;
25     } else {
26         b->l = merge(a, b->l);
27         pull(b); return b;
28     }
29 }
30 pair<Node*, Node*> split(Node* v, int k) {
31     if(!v) return {NULL, NULL};
32     push(v);
33     if(size(v->l) >= k) {
34         auto p = split(v->l, k);
35         if(p.first) p.first->p = NULL;
36         v->l = p.second;
37         pull(v); return {p.first, v};
38     } else {
39         auto p = split(v->r, k - size(v->l) - 1);
40         if(p.second) p.second->p = NULL;
41         v->r = p.first;
42         pull(v); return {v, p.second};
43     }
44 }
45 int get_position(Node* v) { // 0-indexed
46     int k = (v->l != NULL ? v->l->sz : 0);
47     while(v->p != NULL) {
48         if(v == v->p->r) {
49             k++;
50             if(v->p->l != NULL) k += v->p->l->sz;
51         }
52         v = v->p;
53     }
54     return k;
55 }

```

2.16 union-of-rectangles

```

1 // 2
2 // 1 10 1 10
3 // 0 2 0 2
4 // ans = 84
5 vector<int> vx, vy;
6 struct q { int piv, s, e, x; };

```



```

7 struct tree {
8     vector<int> seg, tag;
9     tree(int _n) : seg(_n * 16), tag(_n * 16)
10     {}
11 void add(int ql, int qr, int x, int v, int
12         1, int r) {
13     if(qr <= 1 || r <= ql) return;
14     if(ql <= 1 && r <= qr) {
15         tag[v] += x;
16         if(tag[v] == 0) {
17             if(1 != r) seg[v] = seg[2 * v] + seg
18             [2 * v + 1];
19             else seg[v] = 0;
20         } else seg[v] = vx[r] - vx[1];
21     } else {
22         int mid = (1 + r) / 2;
23         add(ql, qr, x, 2 * v, 1, mid);
24         add(ql, qr, x, 2 * v + 1, mid, r);
25         if(tag[v] == 0 && 1 != r) seg[v] = seg
26         [2 * v] + seg[2 * v + 1];
27     }
28 }
29 int q() { return seg[1]; }
30 };
31 int main() {
32     int n; cin >> n;
33     vector<int> x1(n), x2(n), y_(n), y2(n);
34     for (int i = 0; i < n; i++) {
35         cin >> x1[i] >> x2[i] >> y_[i] >> y2[i];
36         // L R D U
37         vx.pb(x1[i]), vx.pb(x2[i]);
38         vy.pb(y_[i]), vy.pb(y2[i]);
39     }
40     vx = sort_unique(vx);
41     vy = sort_unique(vy);
42     vector<q> a(2 * n);
43     REP(i, n) {
44         x1[i] = lower_bound(ALL(vx), x1[i]) - vx
45         .begin();
46         x2[i] = lower_bound(ALL(vx), x2[i]) - vx
47         .begin();
48         y_[i] = lower_bound(ALL(vy), y_[i]) - vy
49         .begin();
50         y2[i] = lower_bound(ALL(vy), y2[i]) - vy
51         .begin();
52         a[2 * i] = {y_[i], x1[i], x2[i], +1};
53         a[2 * i + 1] = {y2[i], x1[i], x2[i],
54         -1};
55     }
56     sort(ALL(a), [](q a, q b) { return a.piv <
57         b.piv; });
58     tree seg(n);
59     ll ans = 0;
60     REP(i, 2 * n) {
61         int j = i;
62         while(j < 2 * n && a[i].piv == a[j].piv)
63             seg.add(a[j].s, a[j].e, a[j].x, 1, 0,
64             vx.size());
65         j++;
66         if(a[i].piv + 1 != SZ(vy)) ans += 1LL *
67         seg.q() * (vy[a[i].piv + 1] - vy[a[i]
68         ].piv]);
69         i = j - 1;
70     }

```

2.17 wavelet-tree

```

1 template<class T>
2 struct wavelet_tree {
3     int n, log;
4     vector<T> vals;
5     vi sums;
6     vector<ull> bits;
7     inline void set_bit(int i, ull v) { bits[i]
8     >> 6] |= (v << (i & 63)); }
9     inline int get_sum(int i) const { return
10     sums[i >> 6] + __builtin_popcountll(
11     bits[i >> 6] & ((1ULL << (i & 63)) -
12     1)); }
13     wavelet_tree(const vector<T>& _v) : n(SZ(
14     _v)) {
15         vals = sort_unique(_v);
16         log = __lg(2 * vals.size() - 1);
17         bits.resize((log * n + 64) >> 6, 0ULL);
18         sums.resize(SZ(bits), 0);
19         vi v(SZ(_v)), cnt(SZ(vals) + 1);
20         REP(i, SZ(v)) {
21             v[i] = lower_bound(ALL(vals), _v[i]) -
22             vals.begin();
23             cnt[v[i] + 1] += 1;
24         }
25         partial_sum(ALL(cnt) - 1, cnt.begin());
26         REP(j, log) {
27             for(int i : v) {
28                 int tmp = i >> (log - 1 - j);
29                 int pos = (tmp >> 1) << (log - j);
30                 set_bit(j * n + cnt[pos], tmp & 1);
31                 cnt[pos]++;
32             }
33             for(int i : v) cnt[(i >> (log - j)) <<
34             (log - j)]--;
35         }
36         for(int i = 1; i < (int) sums.size(); i
37         ++){ sums[i] = sums[i - 1] +
38         __builtin_popcountll(bits[i - 1]); }
39     }
40     T get_kth(int a, int b, int k) {
41         for(int j = 0, ia = 0, ib = n, res = 0;
42         j < log; j++) {
43             if(j == log) return vals[res];
44             int cnt_ia = get_sum(n * j + ia);
45             int cnt_a = get_sum(n * j + a);
46             int cnt_b = get_sum(n * j + b);
47             int cnt_ib = get_sum(n * j + ib);
48             int ab_zeros = (b - a) - (cnt_b -
49             cnt_a);
50             if(ab_zeros > k) {
51                 res <= 1;
52                 ib -= cnt_ib - cnt_ia;
53                 a -= cnt_a - cnt_ia;
54                 b -= cnt_b - cnt_ia;
55             } else {
56                 res = (res << 1) | 1;
57                 k -= ab_zeros;

```

```

48         ia += (ib - ia) - (cnt_ib - cnt_ia);
49         a += (ib - a) - (cnt_ib - cnt_a);
50         b += (ib - b) - (cnt_ib - cnt_b);
51     }
52 }
53 };
54 };

```

3 Flow-Matching

3.1 bipartite-matching

```

1 struct bipartite_matching {
2     int n, m; // 二分圖左右人數 (0 ~ n-1), (0
3     ~ m-1)
4     vector<vi> g;
5     vi lhs, rhs, dist; // i 與 lhs[i] 配對 (
6     lhs[i] == -1 代表沒有配對)
7     bipartite_matching(int _n, int _m) : n(_n)
8     , m(_m), g(_n), lhs(_n, -1), rhs(_m,
9     -1), dist(_n) {}
10    void add_edge(int u, int v) { g[u].pb(v);
11    }
12    void bfs() {
13        queue<int> q;
14        REP(i, n) {
15            if(lhs[i] == -1) {
16                q.push(i);
17                dist[i] = 0;
18            } else {
19                dist[i] = -1;
20            }
21        }
22        while(!q.empty()) {
23            int u = q.front(); q.pop();
24            for(auto v : g[u]) {
25                if(rhs[v] != -1 && dist[rhs[v]] ==
26                -1) {
27                    dist[rhs[v]] = dist[u] + 1;
28                    q.push(rhs[v]);
29                }
30            }
31        }
32    }
33    bool dfs(int u) {
34        for(auto v : g[u]) {
35            if(rhs[v] == -1) {
36                rhs[lhs[u] = v] = u;
37                return true;
38            }
39        }
40        for(auto v : g[u]) {
41            if(dist[rhs[v]] == dist[u] + 1 && dfs(
42            rhs[v])) {
43                rhs[lhs[u] = v] = u;
44                return true;
45            }
46        }
47        return false;
48    }
49 }

```

```

42 int solve() {
43     int ans = 0;
44     while(true) {
45         bfs();
46         int aug = 0;
47         REP(i, n) if(lhs[i] == -1) aug += dfs(
48         i);
49         if(!aug) break;
50         ans += aug;
51     }
52     return ans;
53 }

```

3.2 Dinic-LowerBound

```

1 template<class T>
2 struct DinicLowerBound {
3     using Maxflow = Dinic<T>;
4     int n;
5     Maxflow d;
6     vector<T> in;
7     DinicLowerBound(int _n) : n(_n), d(_n + 2)
8     , in(_n) {}
9     int add_edge(int from, int to, T low, T
10     high) {
11         assert(0 <= low && low <= high);
12         in[from] -= low, in[to] += low;
13         return d.add_edge(from, to, high - low);
14     }
15     T flow(int s, int t) {
16         T sum = 0;
17         REP(i, n) {
18             if(in[i] > 0) {
19                 d.add_edge(n, i, in[i]);
20                 sum += in[i];
21             }
22             if(in[i] < 0) d.add_edge(i, n + 1, -in
23             [i]);
24         }
25         d.add_edge(t, s, numeric_limits<T>::max
26         ());
27         if(d.flow(n, n + 1) < sum) return -1;
28         return d.flow(s, t);
29     }
30 }

```

3.3 Dinic

```

1 template<class T>
2 class Dinic {
3 public:
4     struct Edge {
5         int from, to;
6         T cap;
7         Edge(int x, int y, T z) : from(x), to(y)
8         , cap(z) {}
9     };
10    constexpr T INF = 1e9;
11    int n;

```

```

11 vector<Edge> edges;
12 vector<vi> g;
13 vi cur, h; // h : Level graph
14 Dinic(int _n) : n(_n), g(_n) {}
15 void add_edge(int u, int v, T c) {
16     g[u].pb(sz(edges));
17     edges.eb(u, v, c);
18     g[v].pb(sz(edges));
19     edges.eb(v, u, 0);
20 }
21 bool bfs(int s, int t) {
22     h.assign(n, -1);
23     queue<int> q;
24     h[s] = 0;
25     q.push(s);
26     while(!q.empty()) {
27         int u = q.front(); q.pop();
28         for(int i : g[u]) {
29             const auto& e = edges[i];
30             int v = e.to;
31             if(e.cap > 0 && h[v] == -1) {
32                 h[v] = h[u] + 1;
33                 if(v == t) return true;
34                 q.push(v);
35             }
36         }
37     }
38     return false;
39 }
40 T dfs(int u, int t, T f) {
41     if(u == t) return f;
42     T r = f;
43     for(int& i = cur[u]; i < sz(g[u]); ++i)
44     {
45         int j = g[u][i];
46         const auto& e = edges[j];
47         int v = e.to;
48         T c = e.cap;
49         if(c > 0 && h[v] == h[u] + 1) {
50             T a = dfs(v, t, min(r, c));
51             edges[j].cap -= a;
52             edges[j ^ 1].cap += a;
53             if((r -= a) == 0) return f;
54         }
55     }
56     return f - r;
57 }
58 T flow(int s, int t, T f = INF) {
59     T ans = 0;
60     while(f > 0 && bfs(s, t)) {
61         cur.assign(n, 0);
62         T cur = dfs(s, t, f);
63         ans += cur;
64         f -= cur;
65     }
66     return ans;
67 }

```

3.4 Flow 建模

- Maximum/Minimum flow with lower bound / Circulation problem
 - Construct super source S and sink T .

- For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
- For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
- If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.
 - To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
 - To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
- The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.

- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
 - Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 - DFS from unmatched vertices in X .
 - $x \in X$ is chosen iff x is unvisited.
 - $y \in Y$ is chosen iff y is visited.

- Minimum cost cyclic flow

- Construct super source S and sink T
- For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$
- For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
- For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$
- For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$
- Flow from S to T , the answer is the cost of the flow $C + K$

- Maximum density induced subgraph

- Binary search on answer, suppose we're checking answer T
- Construct a max flow model, let K be the sum of all weights
- Connect source $s \rightarrow v, v \in G$ with capacity K
- For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
- For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
- T is a valid answer if the maximum flow $f < K|V|$

- Minimum weight edge cover

- For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
- Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
- Find the minimum weight perfect matching on G' .

- Project selection problem
 - If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 - Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 - The mincut is equivalent to the maximum profit of a subset of projects.

- 0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x \bar{y} + x' \bar{y}')$$

can be minimized by the mincut of the following graph:

- Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y .
- Create edge (x, y) with capacity c_{xy} .
- Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3.5 general-matching

```

1 struct GeneralMaxMatch {
2     int n;
3     vector<pii> es;
4     vi g, vis, mate; // i 與 mate[i] 配對 (
5     // mate[i] == -1 代表沒有匹配)
6     GeneralMaxMatch(int n) : n(n), g(n, -1),
7     mate(n, -1) {}
8     bool dfs(int u) {
9         if(vis[u]) return false;
10        vis[u] = true;
11        for(int ei = g[u]; ei != -1; ) {
12            auto [x, y] = es[ei]; ei = y;
13            if(mate[x] == -1) {
14                mate[mate[u] = x] = u;
15                return true;
16            }
17        }
18        for(int ei = g[u]; ei != -1; ) {
19            auto [x, y] = es[ei]; ei = y;
20            int nu = mate[x];
21            mate[mate[u] = x] = u;
22            mate[nu] = -1;
23            if(dfs(nu)) return true;
24            mate[mate[nu] = x] = nu;
25            mate[u] = -1;
26        }
27        return false;
28    }
29    void add_edge(int a, int b) {
30        auto f = [&](int a, int b) {
31            es.eb(b, g[a]);
32            g[a] = sz(es) - 1;
33        };
34        f(a, b); f(b, a);
35    }
36    int solve() {
37        vi o(n);
38        iota(all(o), 0);
39        int ans = 0;
40        REP(it, 100) {
41            shuffle(all(o), rng);

```

3.6 general-weighted-max-matching

```

1 // 1-based QQ
2 struct WeightGraph {
3     static const int inf = INT_MAX;
4     static const int maxn = 514;
5     struct edge {
6         int u, v, w;
7         edge() {}
8         edge(int u, int v, int w) : u(u), v(v), w
9         (w) {}
10    };
11    int n, n_x;
12    edge g[maxn * 2][maxn * 2];
13    int lab[maxn * 2];
14    int match[maxn * 2], slack[maxn * 2], st[
15    maxn * 2], pa[maxn * 2];
16    int flo_from[maxn * 2][maxn + 1], S[maxn *
17    2], vis[maxn * 2];
18    vector<int> flo[maxn * 2];
19    queue<int> q;
20    int e_delta(const edge &e) { return lab[e.
21    u] + lab[e.v] - g[e.u][e.v].w * 2; }
22    void update_slack(int u, int x) { if(!
23    slack[x] || e_delta(g[u][x]) < e_delta
24    (g[slack[x]][x])) slack[x] = u; }
25    void set_slack(int x) {
26        slack[x] = 0;
27        REP(u, n) if(g[u + 1][x].w > 0 && st[u +
28        1] != x && S[st[u + 1]] == 0)
29        update_slack(u + 1, x);
30    }
31    void q_push(int x) {
32        if(x <= n) q.push(x);
33        else REP(i, SZ(flo[x])) q_push(flo[x][i
34        ]);
35    }
36    void set_st(int x, int b) {
37        st[x] = b;
38        if(x > n) REP(i, SZ(flo[x])) set_st(flo[
39        x][i], b);
40    }
41    int get_pr(int b, int xr) {
42        int pr = find(ALL(flo[b]), xr) - flo[b].
43        begin();
44        if(pr % 2 == 1) {
45            reverse(1 + ALL(flo[b]));
46            return SZ(flo[b]) - pr;
47        }
48        return pr;
49    }
50    void set_match(int u, int v) {
51        match[u] = g[u][v].v;
52        if(u <= n) return;
53        edge e = g[u][v];

```



```

43 int xr = flo_from[u][e.u], pr = get_pr(u
    , xr);
44 for(int i = 0; i < pr; ++i) set_match(
    flo[u][i], flo[u][i ^ 1]);
45 set_match(xr, v);
46 rotate(flo[u].begin(), flo[u].begin() +
    pr, flo[u].end());
47 }
48 void augment(int u, int v) {
49     while(true) {
50         int xnv = st[match[u]];
51         set_match(u, v);
52         if(!xnv) return;
53         set_match(xnv, st[pa[xnv]]);
54         u = st[pa[xnv]], v = xnv;
55     }
56 }
57 int get_lca(int u, int v) {
58     static int t = 0;
59     for(++t; u || v; swap(u, v)) {
60         if(u == 0) continue;
61         if(vis[u] == t) return u;
62         vis[u] = t;
63         if(u = st[match[u]]) u = st[pa[u]];
64     }
65     return 0;
66 }
67 void add_blossom(int u, int lca, int v) {
68     int b = n + 1;
69     while(b <= n_x && st[b]) ++b;
70     if(b > n_x) n_x++;
71     lab[b] = S[b] = 0;
72     match[b] = match[lca];
73     flo[b].clear(); flo[b].pb(lca);
74     for(int x = u, y; x != lca; x = st[pa[y]
        ]) flo[b].pb(x), flo[b].pb(y = st[
        match[x]]), q_push(y);
75     reverse(1 + ALL(flo[b]));
76     for(int x = v, y; x != lca; x = st[pa[y]
        ]) flo[b].pb(x), flo[b].pb(y = st[
        match[x]]), q_push(y);
77     set_st(b, b);
78     REP(x, n_x) g[b][x + 1].w = g[x + 1][b].
        w = 0;
79     REP(x, n) flo_from[b][x + 1] = 0;
80     REP(i, SZ(flo[b])) {
81         int xs = flo[b][i];
82         REP(x, n_x) if(g[b][x + 1].w == 0 ||
            e_delta(g[xs][x + 1]) < e_delta(g[
            b][x + 1])) g[b][x + 1] = g[xs][x
            + 1], g[x + 1][b] = g[x + 1][xs];
83         REP(x, n) if(flo_from[xs][x + 1])
            flo_from[b][x + 1] = xs;
84     }
85     set_slack(b);
86 }
87 void expand_blossom(int b) {
88     REP(i, SZ(flo[b])) set_st(flo[b][i], flo
        [b][i]);
89     int xr = flo_from[b][g[b][pa[b]].u], pr
        = get_pr(b, xr);
90     for(int i = 0; i < pr; i += 2) {
91         int xs = flo[b][i], xns = flo[b][i +
            1];
92         pa[xs] = g[xns][xs].u;
93         S[xs] = 1, S[xns] = 0;
94         slack[xs] = 0, set_slack(xns);
95         q_push(xns);
96     }
97     S[xr] = 1, pa[xr] = pa[b];
98     for(size_t i = pr + 1; i < SZ(flo[b]);
        ++i) {
99         int xs = flo[b][i];
100         S[xs] = -1, set_slack(xs);
101     }
102     st[b] = 0;
103 }
104 bool on_found_edge(const edge &e) {
105     int u = st[e.u], v = st[e.v];
106     if(S[v] == -1) {
107         pa[v] = e.u, S[v] = 1;
108         int nu = st[match[v]];
109         slack[v] = slack[nu] = 0;
110         S[nu] = 0, q_push(nu);
111     } else if(S[v] == 0) {
112         int lca = get_lca(u, v);
113         if(!lca) return augment(u, v), augment(
            v, u), true;
114         else add_blossom(u, lca, v);
115     }
116     return false;
117 }
118 bool matching() {
119     memset(S + 1, -1, sizeof(int) * n_x);
120     memset(slack + 1, 0, sizeof(int) * n_x);
121     q = queue<int>();
122     REP(x, n_x) if(st[x + 1] == x + 1 && !
        match[x + 1]) pa[x + 1] = 0, S[x +
        1] = 0, q_push(x + 1);
123     if(q.empty()) return false;
124     while(true) {
125         while(!q.empty()) {
126             int u = q.front(); q.pop();
127             if(S[st[u]] == 1) continue;
128             for(int v = 1; v <= n; ++v)
129                 if(g[u][v].w > 0 && st[u] != st[v]
                    ) {
130                     if(e_delta(g[u][v]) == 0) {
131                         if(on_found_edge(g[u][v]))
132                             return true;
133                     } else update_slack(u, st[v]);
134                 }
135             int d = inf;
136             for(int b = n + 1; b <= n_x; ++b) if(
                st[b] == b && S[b] == 1) d = min(d
                    , lab[b] / 2);
137             for(int x = 1; x <= n_x; ++x) {
138                 if(st[x] == x && slack[x]) {
139                     if(S[x] == -1) d = min(d, e_delta(
                        g[slack[x]][x]));
140                     else if(S[x] == 0) d = min(d,
                        e_delta(g[slack[x]][x]) / 2);
141                 }
142             }
143             REP(u, n) {
144                 if(S[st[u + 1]] == 0) {
145                     if(lab[u + 1] <= d) return 0;
146                     lab[u + 1] -= d;
147                 } else if(S[st[u + 1]] == 1) lab[u +
                    1] += d;
148             }
149         }
150         for(int b = n + 1; b <= n_x; ++b)
151             if(st[b] == b) {
152                 if(S[st[b]] == 0) lab[b] += d * 2;
153                 else if(S[st[b]] == 1) lab[b] -= d
                    * 2;
154             }
155         q = queue<int>();
156         for(int x = 1; x <= n_x; ++x)
157             if(st[x] == x && slack[x] && st[
                slack[x]] != x && e_delta(g[
                slack[x]][x]) == 0)
158                 if(on_found_edge(g[slack[x]][x]))
159                     return true;
160         for(int b = n + 1; b <= n_x; ++b)
161             if(st[b] == b && S[b] == 1 && lab[b]
                == 0) expand_blossom(b);
162     }
163     return false;
164 }
165 pair<ll, int> solve() {
166     memset(match + 1, 0, sizeof(int) * n);
167     n_x = n;
168     int n_matches = 0;
169     ll tot_weight = 0;
170     for(int u = 0; u <= n; ++u) st[u] = u,
        flo[u].clear();
171     int w_max = 0;
172     for(int u = 1; u <= n; ++u)
173         for(int v = 1; v <= n; ++v) {
174             flo_from[u][v] = (u == v ? u : 0);
175             w_max = max(w_max, g[u][v].w);
176         }
177     for(int u = 1; u <= n; ++u) lab[u] =
        w_max;
178     while(matching()) ++n_matches;
179     for(int u = 1; u <= n; ++u)
180         if(match[u] && match[u] < u)
181             tot_weight += g[u][match[u]].w;
182     return make_pair(tot_weight, n_matches);
183 }
184 void add_edge(int u, int v, int w) { g[u][
    v].w = g[v][u].w = w; }
185 void init(int n) : n_(n) {
186     REP(u, n) REP(v, n) g[u + 1][v + 1] =
        edge(u + 1, v + 1, 0);
187 }
188 };

```

3.7 KM

```

1 template<class T>
2 struct KM {
3     static constexpr T INF = numeric_limits<T>
        >::max();
4     int n, ql, qr;
5     vector<vector<T>>> w;
6     vector<T> hl, hr, slk;
7     vi fl, fr, pre, qu;
8     vector<bool> vl, vr;
9     KM(int n) : n(n), w(n, vector<T>(n, -INF))
        , hl(n), hr(n), slk(n), fl(n), fr(n),
        pre(n), qu(n), vl(n), vr(n) {}

```

3.8 max-clique

```

1 template<int V>
2 struct max_clique {
3     using B = bitset<V>;
4     int n = 0;
5     vector<B> g, buf;
6     struct P {

```

```

7   int idx, col, deg;
8   P(int a, int b, int b) : idx(a), col(b),
    deg(c) {}
9   };
10  max_clique(int _n) : n(_n), g(_n), buf(_n)
    {}
11  void add_edge(int a, int b) {
12      assert(a != b);
13      g[a][b] = g[b][a] = 1;
14  }
15  vector<int> now, clique;
16  void dfs(vector<P>& rem){
17      if(SZ(clique) < SZ(now)) clique = now;
18      sort(ALL(rem), [](P a, P b) { return a.
    deg > b.deg; });
19      int max_c = 1;
20      for(auto& p : rem){
21          p.col = 0;
22          while((g[p.idx] & buf[p.col]).any()) p
    .col++;
23          max_c = max(max_c, p.idx + 1);
24          buf[p.col][p.idx] = 1;
25      }
26      REP(i, max_c) buf[i].reset();
27      sort(ALL(rem), [](P a, P b) { return a.
    col < b.col; });
28      for(; !rem.empty(); rem.pop_back()){
29          auto& p = rem.back();
30          if(now.size() + p.col + 1 <= clique.
    size()) break;
31          vector<P> nrem;
32          B bs;
33          for(auto& q : rem){
34              if(g[p.idx][q.idx]){
35                  nrem.emplace_back(q.idx, -1, 0);
36                  bs[q.idx] = 1;
37              }
38          }
39          for(auto& q : nrem) q.deg = (bs & g[q.
    idx]).count();
40          now.emplace_back(p.idx);
41          dfs(nrem);
42          now.pop_back();
43      }
44  }
45  vector<int> solve(){
46      vector<P> remark;
47      REP(i, n) remark.emplace_back(i, -1, SZ(
    g[i]));
48      dfs(remark);
49      return clique;
50  }
51  };

```

3.9 MCMF

```

1  template<class S, class T>
2  class MCMF {
3  public:
4      struct Edge {
5          int from, to;
6          S cap;
7          T cost;

```

```

8          Edge(int u, int v, S x, T y) : from(u),
    to(v), cap(x), cost(y) {}
9      };
10     const ll INF = 1e18L;
11     int n;
12     vector<Edge> edges;
13     vector<vi> g;
14     vector<T> d;
15     vector<bool> inq;
16     vi pedge;
17     MCMF(int _n) : n(_n), g(_n), d(_n), inq(_n)
    {}, pedge(_n) {}
18     void add_edge(int u, int v, S cap, T cost)
    {
19         g[u].pb(sz(edges));
20         edges.pb(u, v, cap, cost);
21         g[v].pb(sz(edges));
22         edges.pb(v, u, 0, -cost);
23     }
24     bool spfa(int s, int t) {
25         bool found = false;
26         fill(all(d), INF);
27         d[s] = 0;
28         inq[s] = true;
29         queue<int> q;
30         q.push(s);
31         while(!q.empty()) {
32             int u = q.front(); q.pop();
33             if(u == t) found = true;
34             inq[u] = false;
35             for(auto& id : g[u]) {
36                 const auto& e = edges[id];
37                 if(e.cap > 0 && d[u] + e.cost < d[e.
    to]) {
38                     d[e.to] = d[u] + e.cost;
39                     pedge[e.to] = id;
40                     if(!inq[e.to]) {
41                         q.push(e.to);
42                         inq[e.to] = true;
43                     }
44                 }
45             }
46         }
47         return found;
48     }
49     pair<S, T> flow(int s, int t, S f = INF) {
50         S cap = 0;
51         T cost = 0;
52         while(f > 0 && spfa(s, t)) {
53             S send = f;
54             int u = t;
55             while(u != s) {
56                 const Edge& e = edges[pedge[u]];
57                 send = min(send, e.cap);
58                 u = e.from;
59             }
60             u = t;
61             while(u != s) {
62                 Edge& e = edges[pedge[u]];
63                 e.cap -= send;
64                 Edge& b = edges[pedge[u] ^ 1];
65                 b.cap += send;
66                 u = e.from;
67             }
68             cap += send;
69             f -= send;

```

```

70         cost += send * d[t];
71     }
72     return {cap, cost};
73 }
74 };

```

3.10 minimum-general-weighted-perfect-matching

```

1  struct Graph {
2      // Minimum General Weighted Matching (
    Perfect Match) 0-base
3      static const int MXN = 105;
4      int n, edge[MXN][MXN];
5      int match[MXN], dis[MXN], onstk[MXN];
6      vector<int> stk;
7      void init(int _n) {
8          n = _n;
9          for(int i=0; i<n; i++)
10             for(int j=0; j<n; j++)
11                 edge[i][j] = 0;
12     }
13     void add_edge(int u, int v, int w) { edge[
    u][v] = edge[v][u] = w; }
14     bool SPFA(int u){
15         if(onstk[u]) return true;
16         stk.push_back(u);
17         onstk[u] = 1;
18         for(int v=0; v<n; v++){
19             if(u != v && match[u] != v && !onstk[v]
    ){
20                 int m = match[v];
21                 if(dis[m] > dis[u] - edge[v][m] +
    edge[u][v]){
22                     dis[m] = dis[u] - edge[v][m] +
    edge[u][v];
23                     onstk[v] = 1;
24                     stk.push_back(v);
25                     if(SPFA(m)) return true;
26                     stk.pop_back();
27                     onstk[v] = 0;
28                 }
29             }
30         }
31         onstk[u] = 0;
32         stk.pop_back();
33         return false;
34     }
35 }
36
37 int solve() {
38     for(int i = 0; i < n; i += 2) match[i] =
    i + 1, match[i+1] = i;
39     while(true) {
40         int found = 0;
41         for(int i=0; i<n; i++) dis[i] = onstk[
    i] = 0;
42         for(int i=0; i<n; i++){
43             stk.clear();
44             if(!onstk[i] && SPFA(i)){
45                 found = 1;
46                 while(stk.size()>=2){
47                     int u = stk.back(); stk.pop_
    back();

```

```

46             int v = stk.back(); stk.pop_back
    ();
47             match[u] = v;
48             match[v] = u;
49         }
50     }
51     }
52     if(!found) break;
53 }
54 int ans = 0;
55 for(int i=0; i<n; i++) ans += edge[i][
    match[i]];
56 return ans / 2;
57 }
58 }graph;

```

4 Geometry

4.1 closest-pair

```

1  const ll INF = 9e18L + 5;
2  vector<P> a;
3  sort(all(a), [](P a, P b) { return a.x < b.x
    ; });
4  ll SQ(ll x) { return x * x; }
5  ll solve(int l, int r) {
6      if(l + 1 == r) return INF;
7      int m = (l + r) / 2;
8      ll midx = a[m].x;
9      ll d = min(solve(l, m), solve(m, r));
10     inplace_merge(a.begin() + l, a.begin() + m
    , a.begin() + r, [](P a, P b) {
11         return a.y < b.y;
12     });
13     vector<P> p;
14     for(int i = l; i < r; ++i) if(SQ(a[i].x -
    midx) < d) p.pb(a[i]);
15     REP(i, sz(p)) {
16         for(int j = i + 1; j < sz(p); ++j) {
17             d = min(d, SQ(p[i].x - p[j].x) + SQ(
    p[i].y - p[j].y));
18             if(SQ(p[i].y - p[j].y) > d) break;
19         }
20     }
21     return d; // 距離平方
22 }

```

4.2 convex-hull

```

1  void convex_hull(vector<P>& dots) {
2      sort(all(dots));
3      vector<P> ans(1, dots[0]);
4      for(int it = 0; it < 2; it++, reverse(all(
    dots))) {
5          for(int i = 1, t = sz(ans); i < sz(dots)
    ; i++) {
6              while(sz(ans) > t && ori(ans[sz(ans) -
    2], ans.back(), dots[i]) < 0) {

```

```

7   ans.ppb();
8   }
9   }
10  }
11  ans.ppb();
12  swap(ans, dots);
13 }

```

4.3 point-in-convex-hull

```

1  int point_in_convex_hull(const vector<P>& a,
2   P p) {
3   // -1 ON, 0 OUT, +1 IN
4   // 要先逆時針排序
5   int n = sz(a);
6   if(btw(a[0], a[1], p) || btw(a[0], a[n -
7   1], p)) return -1;
8   int l = 0, r = n - 1;
9   while(l <= r) {
10    int m = (l + r) / 2;
11    auto a1 = cross(a[m] - a[0], p - a[0]);
12    auto a2 = cross(a[(m + 1) % n] - a[0], p
13    - a[0]);
14    if(a1 >= 0 && a2 <= 0) {
15      auto res = cross(a[(m + 1) % n] - a[m
16      ], p - a[m]);
17      return res > 0 ? 1 : (res >= 0 ? -1 :
18      0);
19    }
20    if(a1 < 0) r = m - 1;
21    else l = m + 1;
22  }
23  return 0;
24 }

```

4.4 point

```

1  using P = pair<ll, ll>;
2  P operator+(P a, P b) { return P{a.X + b.X,
3   a.Y + b.Y}; }
4  P operator-(P a, P b) { return P{a.X - b.X,
5   a.Y - b.Y}; }
6  P operator*(P a, ll b) { return P{a.X * b, a
7   .Y * b}; }
8  P operator/(P a, ll b) { return P{a.X / b, a
9   .Y / b}; }
10 ll dot(P a, P b) { return a.X * b.X + a.Y *
11 b.Y; }
12 ll cross(P a, P b) { return a.X * b.Y - a.Y
13 * b.X; }
14 ll abs2(P a) { return dot(a, a); }
15 double abs(P a) { return sqrt(abs2(a)); }
16 int sign(ll x) { return x < 0 ? -1 : (x == 0
17 ? 0 : 1); }
18 int ori(P a, P b, P c) { return sign(cross(b
19 - a, c - a)); }
20 bool collinear(P a, P b, P c) { return sign(
21 cross(a - c, b - c)) == 0; }
22 bool btw(P a, P b, P c) {

```

```

14 if(!collinear(a, b, c)) return 0;
15 return sign(dot(a - c, b - c)) <= 0;
16 }
17 bool seg_intersect(P a, P b, P c, P d) {
18 int a123 = ori(a, b, c);
19 int a124 = ori(a, b, d);
20 int a341 = ori(c, d, a);
21 int a342 = ori(c, d, b);
22 if(a123 == 0 && a124 == 0) {
23 return btw(a, b, c) || btw(a, b, d) ||
24 btw(c, d, a) || btw(c, d, b);
25 }
26 return a123 * a124 <= 0 && a341 * a342 <=
27 0;
28 }
29 P intersect(P a, P b, P c, P d) {
30 int a123 = cross(b - a, c - a);
31 int a124 = cross(b - a, d - a);
32 return (d * a123 - c * a124) / (a123 -
33 a124);
34 }
35 struct line { P A, B; };
36 P vec(line L) { return L.B - L.A; }
37 P projection(P p, line L) { return L.A + vec
38 (L) / abs(vec(L)) * dot(p - L.A, vec(L))
39 / abs(vec(L)); }

```

4.5 polar-angle-sort

```

1 bool cmp(P a, P b) {
2 #define ng(k) (sign(k.Y) < 0 || (sign(k.Y)
3 == 0 && sign(k.X) < 0))
4 int A = ng(a), B = ng(b);
5 if(A != B) return A < B;
6 if(sign(cross(a, b)) == 0) return abs2(a)
7 < abs2(b);
8 return sign(cross(a, b)) > 0;
9 }

```

5 Graph

5.1 2-SAT

```

1 struct two_sat {
2 int n; SCC g;
3 vector<bool> ans;
4 two_sat(int _n) : n(_n), g(_n * 2) {}
5 void add_or(int u, bool x, int v, bool y) {
6 {
7 g.add_edge(2 * u + !x, 2 * v + y);
8 g.add_edge(2 * v + !y, 2 * u + x);
9 }
10 bool solve() {
11 ans.resize(n);
12 auto id = g.solve();
13 REP(i, n) {

```

```

13 if(id[2 * i] == id[2 * i + 1]) return
14 false;
15 ans[i] = (id[2 * i] < id[2 * i + 1]);
16 }
17 return true;
18 }

```

5.2 centroid-tree

```

1 pair<int, vector<vi>> centroid_tree(const
2 vector<vi>& g) {
3 int n = sz(g);
4 vi siz(n);
5 vector<bool> vis(n);
6 auto dfs_sz = [&](auto f, int u, int p) ->
7 void {
8 siz[u] = 1;
9 for(auto v : g[u]) {
10 if(v == p || vis[v]) continue;
11 f(f, v, u);
12 siz[u] += siz[v];
13 }
14 };
15 auto find_cd = [&](auto f, int u, int p,
16 int all) -> int {
17 for(auto v : g[u]) {
18 if(v == p || vis[v]) continue;
19 if(siz[v] * 2 > all) return f(f, v, u,
20 all);
21 }
22 return u;
23 };
24 vector<vi> h(n);
25 auto build = [&](auto f, int u) -> int {
26 dfs_sz(dfs_sz, u, -1);
27 int cd = find_cd(find_cd, u, -1, siz[u]);
28 ;
29 vis[cd] = true;
30 for(auto v : g[cd]) {
31 if(vis[v]) continue;
32 int child = f(f, v);
33 h[cd].pb(child);
34 }
35 return cd;
36 };
37 int root = build(build, 0);
38 return {root, h};
39 }

```

5.3 chromatic-number

```

1 // vi to(n);
2 // to[u] |= 1 << v;
3 // to[v] |= 1 << u;
4 int chromatic_number(vi g) {
5 constexpr int MOD = 998244353;
6 int n = SZ(g);
7 vector<int> I(1 << n); I[0] = 1;
8 for(int s = 1; s < (1 << n); s++) {

```

```

9 int v = __builtin_ctz(s), t = s ^ (1 <<
10 v);
11 I[s] = (I[t] + I[t & ~g[v]]) % MOD;
12 }
13 auto f = I;
14 for(int k = 1; k <= n; k++) {
15 int sum = 0;
16 REP(s, 1 << n) {
17 if((__builtin_popcount(s) ^ n) & 1)
18 sum -= f[s];
19 else sum += f[s];
20 sum = ((sum % MOD) + MOD) % MOD;
21 f[s] = 1LL * f[s] * I[s] % MOD;
22 }
23 if(sum != 0) return k;
24 }
25 return 48763;
26 }

```

5.4 HLD

```

1 struct HLD {
2 int n;
3 vector<vi> g;
4 vi siz, par, depth, top, tour, fi, id;
5 sparse_table<pii, min> st;
6 HLD(int _n) : n(_n), g(_n), siz(_n), par(
7 _n), depth(_n), top(_n), fi(_n), id(_n) {
8 tour.reserve(n);
9 }
10 void add_edge(int u, int v) {
11 g[u].push_back(v);
12 g[v].push_back(u);
13 }
14 void build(int root = 0) {
15 par[root] = -1;
16 top[root] = root;
17 vector<pii> euler_tour;
18 euler_tour.reserve(2 * n - 1);
19 dfs_sz(root);
20 dfs_link(euler_tour, root);
21 st = sparse_table<pii, min>(euler_tour);
22 }
23 int get_lca(int u, int v) {
24 int L = fi[u], R = fi[v];
25 if(L > R) swap(L, R);
26 return st.prod(L, R).second;
27 }
28 bool is_anc(int u, int v) {
29 return id[u] <= id[v] && id[v] < id[u] +
30 siz[u];
31 }
32 bool on_path(int a, int b, int x) {
33 return (is_ancestor(x, a) || is_ancestor
34 (x, b)) && is_ancestor(get_lca(a, b),
35 x);
36 }
37 int get_dist(int u, int v) {
38 return depth[u] + depth[v] - 2 * depth[
39 get_lca(u, v)];
40 }
41 int kth_anc(int u, int k) {

```

```

37 if(depth[u] < k) return -1;
38 int d = depth[u] - k;
39 while(depth[top[u]] > d) u = par[top[u]
40 ];
41 return tour[id[u] + d - depth[u]];
42 }
43 int kth_node_on_path(int a, int b, int k)
44 {
45     int z = get_lca(a, b);
46     int fi = depth[a] - depth[z];
47     int se = depth[b] - depth[z];
48     if(k < 0 || k > fi + se) return -1;
49     if(k < fi) return kth_anc(a, k);
50     return kth_anc(b, fi + se - k);
51 }
52 vector<pii> get_path(int u, int v, bool
53 include_lca = true) {
54     if(u == v && !include_lca) return {};
55     vector<pii> seg;
56     while(top[u] != top[v]) {
57         if(depth[top[u]] > depth[top[v]]) swap
58         (u, v);
59         seg.pb(id[top[v]], id[v]);
60         v = par[top[v]];
61     }
62     if(depth[u] > depth[v]) swap(u, v); // u
63     is_lca
64     if(u != v || include_lca) seg.pb(id[u] +
65 !include_lca, id[v]);
66     return seg;
67 }
68 void dfs_sz(int u) {
69     if(par[u] != -1) g[u].erase(find(all(g[u]
70 ), par[u]));
71     siz[u] = 1;
72     for(auto& v : g[u]) {
73         par[v] = u;
74         depth[v] = depth[u] + 1;
75         dfs_sz(v);
76         siz[u] += siz[v];
77         if(siz[v] > siz[g[u][0]]) swap(v, g[u]
78         ][0]);
79     }
80 }
81 void dfs_link(vector<pii>& euler_tour, int
82 u) {
83     fi[u] = sz(euler_tour);
84     id[u] = sz(tour);
85     euler_tour.pb(depth[u], u);
86     tour.pb(u);
87     for(auto v : g[u]) {
88         top[v] = (v == g[u][0] ? top[u] : v);
89         dfs_link(euler_tour, v);
90         euler_tour.pb(depth[u], u);
91     }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

5.5 lowlink

```

1 struct lowlink {
2     int n, cnt = 0, tecc_cnt = 0, tvcc_cnt =
3     0;

```

```

4     vector<vector<pii>> g;
5     vector<pii> edges;
6     vi roots, id, low, tecc_id, tvcc_id;
7     vector<bool> is_bridge, is_cut,
8     is_tree_edge;
9     lowlink(int _n) : n(_n), g(_n), is_cut(_n,
10     false), id(_n, -1), low(_n, -1) {}
11     void add_edge(int u, int v) {
12         g[u].eb(v, sz(edges));
13         g[v].eb(u, sz(edges));
14         edges.pb(u, v);
15         is_bridge.pb(false);
16         is_tree_edge.pb(false);
17         tvcc_id.pb(-1);
18     }
19     void dfs(int u, int peid = -1) {
20         static vi stk;
21         static int rid;
22         if(peid < 0) rid = cnt;
23         if(peid == -1) roots.pb(u);
24         id[u] = low[u] = cnt++;
25         for(auto [v, eid] : g[u]) {
26             if(eid == peid) continue;
27             if(id[v] < id[u]) stk.pb(eid);
28             if(id[v] >= 0) {
29                 low[u] = min(low[u], id[v]);
30             } else {
31                 is_tree_edge[eid] = true;
32                 dfs(v, eid);
33                 low[u] = min(low[u], low[v]);
34                 if((id[u] == rid && id[v] != rid +
35                 1) || (id[u] != rid && low[v] >=
36                 id[u])) {
37                     is_cut[u] = true;
38                 }
39                 if(low[v] >= id[u]) {
40                     while(true) {
41                         int e = stk.back();
42                         stk.pop_back();
43                         tvcc_id[e] = tvcc_cnt;
44                         if(e == eid) break;
45                     }
46                     tvcc_cnt++;
47                 }
48             }
49         }
50     }
51     void build() {
52         REP(i, n) if(id[i] < 0) dfs(i);
53         REP(i, sz(edges)) {
54             auto [u, v] = edges[i];
55             if(id[u] > id[v]) swap(u, v);
56             is_bridge[i] = (id[u] < low[v]);
57         }
58     }
59     vector<vi> two_ecc() { // 邊雙
60         tecc_cnt = 0;
61         tecc_id.assign(n, -1);
62         vi stk;
63         REP(i, n) {
64             if(tecc_id[i] != -1) continue;
65             tecc_id[i] = tecc_cnt;
66             stk.pb(i);
67             while(sz(stk)) {
68                 int u = stk.back(); stk.pop_back();
69                 for(auto [v, eid] : g[u]) {
70                     if(eid == tecc_id[u]) continue;
71                     if(id[v] < id[u]) stk.pb(v);
72                     if(id[v] >= 0) tecc_id[v] = tecc_id[u];
73                     else {
74                         dfs(v, eid);
75                         tecc_id[v] = tecc_id[u];
76                     }
77                 }
78             }
79         }
80     }
81     vector<vi> bcc_vertices() { // 點雙
82         vector<vi> comp(tvcc_cnt);
83         REP(i, sz(edges)) {
84             comp[tvcc_id[i]].pb(edges[i].first);
85             comp[tvcc_id[i]].pb(edges[i].second);
86         }
87         for(auto& v : comp) {
88             sort(all(v));
89             v.erase(unique(all(v)), v.end());
90         }
91         REP(i, n) if(g[i].empty()) comp.pb({});
92         return comp;
93     }
94     vector<vi> bcc_edges() {
95         vector<vi> ret(tvcc_cnt);
96         REP(i, sz(edges)) ret[tvcc_id[i]].pb(i);
97         return ret;
98     }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

64     for(auto [v, eid] : g[u]) {
65         if(tecc_id[v] >= 0 || is_bridge[
66             eid]) {
67             continue;
68         }
69         tecc_id[v] = tecc_cnt;
70         stk.pb(v);
71     }
72     tecc_cnt++;
73     vector<vi> comp(tecc_cnt);
74     REP(i, n) comp[tecc_id[i]].pb(i);
75     return comp;
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299
```

```

3  int n = SZ(x);
4  vi deg(n);
5  vector<vector<int>> g(n);
6  for(auto& [u, v] : edges) {
7      if(u > v) swap(u, v);
8      deg[u]++, deg[v]++;
9  }
10 REP(i, n) g[i].reserve(deg[i]);
11 for(auto [u, v] : edges) {
12     if(deg[u] > deg[v]) swap(u, v);
13     g[u].pb(v);
14 }
15 vi val(n);
16 __int128 ans = 0;
17 REP(a, n) {
18     for(auto b : g[a]) val[b] = x[b];
19     for(auto b : g[a]) {
20         ll tmp = 0;
21         for(auto c : g[b]) tmp += val[c];
22         ans += __int128(tmp) * x[a] * x[b];
23     }
24     for(auto b : g[a]) val[b] = 0;
25 }
26 return ans % mod;
27 }

```

6 Math

6.1 Aliens

```

1  template<class Func, bool MAX>
2  ll Aliens(ll l, ll r, int k, Func f) {
3      while(l < r) {
4          ll m = l + (r - l) / 2;
5          auto [score, op] = f(m);
6          if(op == k) return score + m * k * (MAX
7              ? +1 : -1);
8          if(op < k) r = m;
9          else l = m + 1;
10     }
11     return f(l).first + l * k * (MAX ? +1 :
12         -1);

```

6.2 Berlekamp-Massey

```

1  // - [1, 2, 4, 8, 16] -> (1, [1, -2])
2  // - [1, 1, 2, 3, 5, 8] -> (2, [1, -1, -1])
3  // - [0, 0, 0, 0, 1] -> (5, [1, 0, 0, 0, 0,
4      998244352]) (mod 998244353)
5  // - [] -> (0, [1])
6  // - [0, 0, 0] -> (0, [1])
7  // - [-2] -> (1, [1, 2])
8  template<class T>
9  pair<int, vector<T>> BM(const vector<T>& S)
10 {
11     using poly = vector<T>;
12     int N = SZ(S);

```

```

11 poly C_rev{1}, B{1};
12 int L = 0, m = 1;
13 T b = 1;
14 auto adjust = [](poly C, const poly &B, T
15     d, T b, int m) -> poly {
16     C.resize(max(SZ(C), SZ(B) + m));
17     T a = d / b;
18     REP(i, SZ(B)) C[i + m] -= a * B[i];
19     return C;
20 };
21 REP(n, N) {
22     T d = S[n];
23     REP(i, L) d += C_rev[i + 1] * S[n - 1 -
24         i];
25     if(d == 0) m++;
26     else if (2 * L <= n) {
27         poly Q = C_rev;
28         C_rev = adjust(C_rev, B, d, b, m);
29         L = n + 1 - L, B = Q, b = d, m = 1;
30     } else C_rev = adjust(C_rev, B, d, b, m
31         ++);
32 }
33 return {L, C_rev};
34 }
35 // Calculate  $x^N \bmod f(x)$ 
36 // Complexity:  $\mathcal{O}(K^2 \log N)$  ( $K$ : deg. of
37      $f$ )
38 // (4, [1, -1, -1]) -> [2, 3]
39 // (  $x^4 = (x^2 + x + 2)(x^2 - x - 1) + 3x + 2$  )
40 template<class T>
41 vector<T> monomial_mod_polynomial(long long
42     N, const vector<T> &f_rev) {
43     assert(!f_rev.empty() && f_rev[0] == 1);
44     int K = SZ(f_rev) - 1;
45     if(!K) return {};
46     int D = 64 - __builtin_clzll(N);
47     vector<T> ret(K, 0);
48     ret[0] = 1;
49     auto self_conv = [](vector<T> x) -> vector
50         <T> {
51         int d = SZ(x);
52         vector<T> ret(d * 2 - 1);
53         REP(i, d) {
54             ret[i * 2] += x[i] * x[i];
55             REP(j, i) ret[i + j] += x[i] * x[j] *
56                 2;
57         }
58         return ret;
59     };
60     for(int d = D; d--;) {
61         ret = self_conv(ret);
62         for(int i = 2 * K - 2; i >= K; i--) {
63             REP(j, k) ret[i - j - 1] -= ret[i] *
64                 f_rev[j + 1];
65         }
66     }
67     ret.resize(K);
68     if (N >> d & 1) {
69         vector<T> c(K);
70         c[0] = -ret[K - 1] * f_rev[K];
71         for(int i = 1; i < K; i++) c[i] = ret[
72             i - 1] - ret[K - 1] * f_rev[K - i
73             ];
74         ret = c;
75     }

```

```

66 }
67 return ret;
68 }
69 // Guess k-th element of the sequence,
70     assuming linear recurrence
71 template<class T>
72 T guess_kth_term(const vector<T>& a, long
73     long k) {
74     assert(k >= 0);
75     if(k < 1LL * SZ(a)) return a[k];
76     auto f = BM<T>(a).second;
77     auto g = monomial_mod_polynomial<T>(k, f);
78     T ret = 0;
79     REP(i, SZ(g)) ret += g[i] * a[i];
80     return ret;

```

6.3 Chinese-Remainder

```

1  // (rem, mod) {0, 0} for no solution
2  pair<ll, ll> crt(ll r0, ll m0, ll r1, ll m1)
3  {
4      r0 = (r0 % m0 + m0) % m0;
5      r1 = (r1 % m1 + m1) % m1;
6      if(m0 < m1) swap(r0, r1), swap(m0, m1);
7      if(m0 % m1 == 0) {
8          if(r0 % m1 != r1) return {0, 0};
9      }
10     ll g, im, qq;
11     g = ext_gcd(m0, m1, im, qq);
12     ll u1 = (m1 / g);
13     if((r1 - r0) % g) return {0, 0};
14     ll x = (r1 - r0) / g % u1 * im % u1;
15     r0 += x * m0;
16     m0 *= u1;
17     if(r0 < 0) r0 += m0;
18     return {r0, m0};

```

6.4 Combination

```

1  mint binom(int n, int k) {
2      if(k < 0 || k > n) return 0;
3      return fact[n] * inv_fact[k] * inv_fact[n
4          - k];
5  }
6  //  $a_1 + a_2 + \dots + a_n = k, a_i \geq 0$ 
7  mint stars_and_bars(int n, int k) { return
8      binom(k + n - 1, n - 1); }
9  // number of ways from (0, 0) to (n, m)
10 mint paths(int n, int m) { return binom(n +
11     m, n); }
12 mint catalan(int n) { return binom(2 * n, n)
13     - binom(2 * n, n + 1); }

```

6.5 Determinant

```

1  T det(vector<vector<T>> a) {
2      int n = SZ(a);
3      T ret = 1;
4      REP(i, n) {
5          int idx = -1;
6          for(int j = i; j < n; j++) {
7              if(a[j][i] != 0) {
8                  idx = j;
9                  break;
10             }
11         }
12         if(idx == -1) return 0;
13         if(i != idx) {
14             ret *= T(-1);
15             swap(a[i], a[idx]);
16         }
17         ret *= a[i][i];
18         T inv = T(1) / a[i][i];
19         REP(j, n) a[i][j] *= inv;
20         for(int j = i + 1; j < n; j++) {
21             T x = a[j][i];
22             if(x == 0) continue;
23             for(int k = i; k < n; k++) {
24                 a[j][k] -= a[i][k] * x;
25             }
26         }
27     }
28     return ret;
29 }

```

6.6 Discrete-Log

```

1  int discrete_log(int a, int b, int m) {
2      if(b == 1 || m == 1) return 0;
3      int n = sqrt(m) + 2, e = 1, f = 1, j = 1;
4      unordered_map<int, int> A; // becareful
5      while(j <= n && (e = f * 1LL * e * a % m)
6          != b) A[1LL * e * b % m] = j++;
7      if(e == b) return j;
8      if(__gcd(m, e) == __gcd(m, b)) {
9          for(int i = 2; i < n + 2; ++i) {
10             e = 1LL * e * f % m;
11             if(A.find(e) != A.end()) return n * i
12                 - A[e];
13         }
14     }
15     return -1;

```

6.7 extgcd

```

1  //  $ax + by = \gcd(a, b)$ 
2  ll ext_gcd(ll a, ll b, ll& x, ll& y) {
3      if(b == 0) {
4          x = 1, y = 0;
5          return a;
6      }

```



```

7 | ll x1, y1;
8 | ll g = ext_gcd(b, a % b, x1, y1);
9 | x = y1, y = x1 - (a / b) * y1;
10 | return g;
11 | }

```

6.8 Floor-Sum

```

1 | // sum_{i=0}^{n-1} floor((a * i + b) / m) in Log
   | (n + m + a + b)
2 | ll floor_sum(ll n, ll m, ll a, ll b) {
3 |     ll ans = 0;
4 |     if(a >= m) ans += (n - 1) * n * (a / m) /
   |         2, a %= m;
5 |     if(b >= m) ans += n * (b / m), b %= m;
6 |     ll y_max = (a * n + b) / m, x_max = (y_max
   |         * m - b);
7 |     if(y_max == 0) return ans;
8 |     ans += (n - (x_max + a - 1) / a) * y_max;
9 |     return ans + floor_sum(y_max, a, m, (a -
   |         x_max % a) % a);
10 | }

```

6.9 FWHT

```

1 | #define ppc __builtin_popcount
2 | template<class T, class F>
3 | void fwht(vector<T>& a, F f) {
4 |     int n = SZ(a);
5 |     assert(ppc(n) == 1);
6 |     for(int i = 1; i < n; i <= 1) {
7 |         for(int j = 0; j < n; j += i < 1) {
8 |             REP(k, i) f(a[j + k], a[i + j + k]);
9 |         }
10 |    }
11 | }
12 | template<class T>
13 | void or_transform(vector<T>& a, bool inv) {
14 |     fwht(a, [&](T& x, T& y) { y += x * (inv
   |         ? -1 : +1); }); }
15 | template<class T>
16 | void and_transform(vector<T>& a, bool inv) {
17 |     fwht(a, [&](T& x, T& y) { x += y * (inv
   |         ? -1 : +1); }); }
18 | template<class T>
19 | void xor_transform(vector<T>& a, bool inv) {
20 |     fwht(a, [&](T& x, T& y) {
21 |         T z = x + y;
22 |         y = x - y;
23 |         x = z;
24 |     });
25 |     if(inv) {
26 |         T z = T(1) / T(SZ(a));
27 |         for(auto& x : a) x *= z;
28 |     }
29 | }
30 | template<class T>
31 | vector<T> convolution(vector<T> a, vector<T>
   |     b) {
32 |     assert(SZ(a) == SZ(b));

```

```

31 | transform(a, false), transform(b, false);
32 | REP(i, SZ(a)) a[i] *= b[i];
33 | transform(a, true);
34 | return a;
35 | }
36 | template<class T>
37 | vector<T> subset_convolution(const vector<T>
   |     &f, const vector<T>& g) {
38 |     assert(SZ(f) == SZ(g));
39 |     int n = SZ(f);
40 |     assert(ppc(n) == 1);
41 |     const int lg = __lg(n);
42 |     vector<vector<T>> fhat(lg + 1, vector<T>(n
   |         )), ghat(fhat);
43 |     REP(i, n) fhat[ppc(i)][i] = f[i], ghat[ppc
   |         (i)][i] = g[i];
44 |     REP(i, lg + 1) or_transform(fhat[i], false
   |         ), or_transform(ghat[i], false);
45 |     vector<vector<T>> h(lg + 1, vector<T>(n));
46 |     REP(m, n) REP(i, lg + 1) REP(j, i + 1) h[i
   |         ][m] += fhat[j][m] * ghat[i - j][m];
47 |     REP(i, lg + 1) or_transform(h[i], true);
48 |     vector<T> res(n);
49 |     REP(i, n) res[i] = h[ppc(i)][i];
50 |     return res;
51 | }

```

6.10 Gauss-Jordan

```

1 | int GaussJordan(vector<vector<ld>>& a) {
2 |     // -1 no sol, 0 inf sol
3 |     int n = SZ(a);
4 |     REP(i, n) assert(SZ(a[i]) == n + 1);
5 |     REP(i, n) {
6 |         int p = i;
7 |         REP(j, n) {
8 |             if(j < i && abs(a[j][j]) > EPS)
   |                 continue;
9 |             if(abs(a[j][i]) > abs(a[p][i])) p = j;
10 |        }
11 |        REP(j, n + 1) swap(a[i][j], a[p][j]);
12 |        if(abs(a[i][i]) <= EPS) continue;
13 |        REP(j, n) {
14 |            if(i == j) continue;
15 |            ld delta = a[j][i] / a[i][i];
16 |            FOR(k, i, n + 1) a[j][k] -= delta * a
   |                [i][k];
17 |        }
18 |    }
19 |    bool ok = true;
20 |    REP(i, n) {
21 |        if(abs(a[i][i]) <= EPS) {
22 |            if(abs(a[i][n]) > EPS) return -1;
23 |            ok = false;
24 |        }
25 |    }
26 |    return ok;
27 | }

```

6.11 Int-Div

```

1 | ll floor_div(ll a, ll b) {
2 |     return a/b - ((a^b) < 0 && a%b != 0);
3 | }
4 | ll ceil_div(ll a, ll b) {
5 |     return a/b + ((a^b) > 0 && a%b != 0);
6 | }

```

6.12 Linear-Sieve

```

1 | vi primes, least = {0, 1}, phi, mobius;
2 | void linearSieve(int n) {
3 |     least = phi = mobius = vi(n + 1);
4 |     for(int i = 2; i <= n; i++) {
5 |         if(!least[i]) {
6 |             least[i] = i;
7 |             primes.pb(i);
8 |             phi[i] = i - 1;
9 |             mobius[i] = -1;
10 |        }
11 |        for(auto j : primes) {
12 |            if(i * j > n) break;
13 |            least[i * j] = j;
14 |            if(i % j == 0) {
15 |                mobius[i * j] = 0;
16 |                phi[i * j] = phi[i] * j;
17 |                break;
18 |            } else {
19 |                mobius[i * j] = -mobius[i];
20 |                phi[i * j] = phi[i] * phi[j];
21 |            }
22 |        }
23 |    }
24 | }

```

6.13 Miller-Rabin

```

1 | bool is_prime(ll n, vector<ll> x) {
2 |     ll d = n - 1;
3 |     d >>= __builtin_ctzll(d);
4 |     for(auto a : x) {
5 |         if(n <= a) break;
6 |         ll t = d, y = 1, b = t;
7 |         while(b) {
8 |             if(b & 1) y = i128(y) * a % n;
9 |             a = i128(a) * a % n;
10 |            b >>= 1;
11 |        }
12 |        while(t != n - 1 && y != 1 && y != n -
   |            1) {
13 |            y = i128(y) * y % n;
14 |            t <<= 1;
15 |        }
16 |        if(y != n - 1 && t % 2 == 0) return
   |            false;
17 |    }
18 |    return true;
19 | }
20 | bool is_prime(ll n) {
21 |     if(n <= 1) return false;
22 |     if(n % 2 == 0) return n == 2;

```

6.14 Min-of-Mod-of-Linear

```

1 | // \min{Ax + B (mod M) | 0 <= x < N}
2 | int min_of_mod_of_linear(int n, int m, int a
   |     , int b) {
3 |     ll v = floor_sum(n, m, a, b);
4 |     int l = -1, r = m - 1;
5 |     while(r - l > 1) {
6 |         int k = (l + r) / 2;
7 |         if(floor_sum(n, m, a, b + (m - 1 - k)) <
   |             v + n) r = k;
8 |         else l = k;
9 |     }
10 |    return r;
11 | }

```

6.15 Mod-Inv

```

1 | int inv(int a) {
2 |     if(a < N) return inv[a];
3 |     if(a == 1) 1;
4 |     return (MOD - 1LL * (MOD / a) * inv(MOD %
   |         a) % MOD) % MOD;
5 | }
6 | vi mod_inverse(int m, int n = -1) {
7 |     assert(n < m);
8 |     if(n == -1) n = m - 1;
9 |     vi inv(n + 1);
10 |    inv[0] = inv[1] = 1;
11 |    for(int i = 2; i <= n; i++) inv[i] = m - 1
   |        LL * (m / i) * inv[m % i] % m;
12 |    return inv;
13 | }

```

6.16 Pollard-Rho

```

1 | void PollardRho(map<ll, int>& mp, ll n) {
2 |     if(n == 1) return;
3 |     if(is_prime(n)) return mp[n]++, void();
4 |     if(n % 2 == 0) {
5 |         mp[2] += 1;
6 |         PollardRho(mp, n / 2);
7 |         return;
8 |     }
9 |     ll x = 2, y = 2, d = 1, p = 1;
10 |    #define f(x, n, p) ((i128(x) * x % n + p)
   |        % n)
11 |    while(true) {
12 |        if(d != 1 && d != n) {
13 |            PollardRho(mp, d);

```

```

14 PollardRho(mp, n / d);
15 return;
16 }
17 p += (d == n);
18 x = f(x, n, p), y = f(f(y, n, p), n, p);
19 d = __gcd(abs(x - y), n);
20 }
21 #undef f
22 }
23
24 vector<ll> get_divisors(ll n) {
25     if(n == 0) return {};
26     map<ll, int> mp;
27     PollardRho(mp, n);
28     vector<pair<ll, int>> v(all(mp));
29     vector<ll> res;
30     auto f = [&](auto f, int i, ll x) -> void
31     {
32         if(i == sz(v)) {
33             res.pb(x);
34             return;
35         }
36         for(int j = v[i].second; ; j++) {
37             f(f, i + 1, x);
38             if(j == 0) break;
39             x *= v[i].first;
40         }
41     };
42     f(f, 0, 1);
43     sort(all(res));
44     return res;

```

6.17 Primes

```

1 /* 12721 13331 14341 75577 123457 222557
2    556679 999983 1097774749 1076767633
3    100102021 999997771 1001010013
4    1000512343 987654361 999991231 999888733
5    98789101 987777733 999991921 1010101333
6    1010102101 1000000000039
7    100000000000037 2305843009213693951
8    4611686018427387847 9223372036854775783
9    18446744073709551557 */

```

6.18 Xor-Basis

```

1 template<int B>
2 struct xor_basis {
3     using T = long long;
4     bool zero = false, change = false;
5     int cnt = 0;
6     array<T, B> p = {};
7     vector<T> d;
8     void insert(T x) {
9         IREP(i, B) {
10             if(x >> i & 1) {
11                 if(!p[i]) {
12                     p[i] = x, cnt++;
13                     change = true;

```

```

14         return;
15     } else x ^= p[i];
16 }
17 }
18 if(!zero) zero = change = true;
19 }
20 T get_min() {
21     if(zero) return 0;
22     REP(i, B) if(p[i]) return p[i];
23 }
24 T get_max() {
25     T ans = 0;
26     IREP(i, B) ans = max(ans, ans ^ p[i]);
27     return ans;
28 }
29 T get_kth(long long k) {
30     k++;
31     if(k == 1 && zero) return 0;
32     k -= zero;
33     if(k >= (1LL << cnt)) return -1;
34     update();
35     T ans = 0;
36     REP(i, SZ(d)) if(k >> i & 1) ans ^= d[i];
37     return ans;
38 }
39 bool contains(T x) {
40     if(x == 0) return zero;
41     IREP(i, B) if(x >> i & 1) x ^= p[i];
42     return x == 0;
43 }
44 void merge(const xor_basis& other) { REP(i, B) if(other.p[i]) insert(other.p[i]); }
45 void update() {
46     if(!change) return;
47     change = false;
48     d.clear();
49     REP(j, B) IREP(i, j) if(p[j] >> i & 1) p[j] ^= p[i];
50     REP(i, B) if(p[i]) d.pb(p[i]);
51 }
52 };

```

6.19 估計值

- Estimation

- The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200000 for $n < 1e19$.
- The number of ways of writing n as a sum of positive integers, disregarding the order of the summands. 1, 1, 2, 3, 5, 7, 11, 15, 22, 30 for $n = 0 \sim 9$, 627 for $n = 20$, $\sim 2e5$ for $n = 50$, $\sim 2e8$ for $n = 100$.
- Total number of partitions of n distinct elements: $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, \dots$

6.20 定理

- Cramer's rule

$$\begin{matrix} ax + by = e \\ cx + dy = f \end{matrix} \Rightarrow \begin{matrix} x = \frac{ed - bf}{ad - bc} \\ y = \frac{af - ec}{ad - bc} \end{matrix}$$

- Vandermonde's Identity

$$C(n + m, k) = \sum_{i=0}^k C(n, i)C(m, k - i)$$

- Burnside's Lemma

Let us calculate the number of necklaces of n pearls, where each pearl has m possible colors. Two necklaces are symmetric if they are similar after rotating them. There are n ways to change the position of a necklace, because we can rotate it $0, 1, \dots, n-1$ steps clockwise. If the number of steps is 0, all m^n necklaces remain the same, and if the number of steps is 1, only the m necklaces where each pearl has the same color remain the same. More generally, when the number of steps is k , a total of $m^{\gcd(k, n)}$ necklaces remain the same. The reason for this is that blocks of pearls of size $\gcd(k, n)$ will replace each other. Thus, according to Burnside's lemma, the number of necklaces is $\sum_{i=0}^{n-1} \frac{m^{\gcd(i, n)}}{n}$. For example, the number of necklaces of length 4 with 3 colors is $\frac{3^4 + 3 + 3^2 + 3}{4} = 24$

- Lindström–Gessel–Viennot Lemma

定義

$\omega(P)$ 表示 P 這條路徑上所有邊的邊權之積。(路徑計數時，可以將邊權都設為 1)(事實上，邊權可以為生成函數) $e(u, v)$ 表示 u 到 v 的每一條路徑 P 的 $\omega(P)$ 之和，即 $e(u, v) = \sum_{P: u \rightarrow v} \omega(P)$ 。起點

集合 A 是有向無環圖點集的一個子集，大小為 n 。終點集合 B 也是有向無環圖點集的一個子集，大小也為 n 。一組 $A \rightarrow B$ 的不相交路徑 $S: S_i$ 是一條從 A_i 到 $B_{\sigma(S)_i}$ 的路徑 ($\sigma(S)$ 是一個排列)。對於任何 $i \neq j$ ， S_i 和 S_j 沒有公共頂點。 $t(\sigma)$ 表示排列 σ 的逆序對個數。

$$M = \begin{bmatrix} e(A_1, B_1) & e(A_1, B_2) & \cdots & e(A_1, B_n) \\ e(A_2, B_1) & e(A_2, B_2) & \cdots & e(A_2, B_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(A_n, B_1) & e(A_n, B_2) & \cdots & e(A_n, B_n) \end{bmatrix}$$

$$\det(M) = \sum_{S: A \rightarrow B} (-1)^{t(\sigma(S))} \prod_{i=1}^n \omega(S_i)$$

其中 $\sum_{S: A \rightarrow B}$ 表示滿足上文要求的 $A \rightarrow B$ 的每一組不相交路徑 S 。

- Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

- Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{r \cdot \text{rank}(D)}{2}$ is the maximum matching on G .

- Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each labeled vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees.
- Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

- Erdős–Gallai theorem

A sequence of nonnegative integers $d_1 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \dots + d_n$ is even

and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ holds for every $1 \leq k \leq n$.

- Gale–Ryser theorem

A pair of sequences of nonnegative integers $a_1 \geq \dots \geq a_n$ and b_1, \dots, b_n is bigraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Fulkerson–Chen–Anstee theorem

A sequence $(a_1, b_1), \dots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \dots \geq a_n$ is digraphic if and only

if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Möbius inversion formula

$$\begin{aligned} f(n) &= \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) \\ f(n) &= \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d) \end{aligned}$$

- Spherical cap

- A portion of a sphere cut off by a plane.
- r : sphere radius, a : radius of the base of the cap, h : height of the cap, θ : $\arcsin(a/r)$.
- Volume = $\pi h^2(3r - h)/3 = \pi h(3a^2 + h^2)/6 = \pi r^3(2 + \cos \theta)(1 - \cos \theta)^2/3$.
- Area = $2\pi rh = \pi(a^2 + h^2) = 2\pi r^2(1 - \cos \theta)$.

6.21 數字

- Bernoulli numbers

$$B_0 = 1, B_1^\pm = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$$

$$\sum_{j=0}^m \binom{m+1}{j} B_j = 0, \text{EGF is } B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

$$S_m(n) = \sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k^+ n^{m+1-k}$$

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k), S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$$

$$x^n = \sum_{i=0}^n S(n, i) (x)_i$$

- Pentagonal number theorem

$$\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2} \right)$$

- Catalan numbers

$$C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$$

$$C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$$

- Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j 's s.t. $\pi(j) > \pi(j+1)$, $k+1$ j 's s.t. $\pi(j) \geq j$, k j 's s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

- 次方和

$$\sum_{k=1}^n k^3 = \left(\frac{n(n+1)}{2} \right)^2$$

$$\sum_{k=1}^n k^4 = \frac{1}{30} (6n^5 + 15n^4 + 10n^3 - n)$$

$$\sum_{k=1}^n k^5 = \frac{1}{12} (2n^6 + 6n^5 + 5n^4 - n^2)$$

$$\sum_{k=1}^n k^6 = \frac{1}{42} (6n^7 + 21n^6 + 21n^5 - 7n^3 + n)$$

General form:

$$\sum_{k=1}^n k^p = \frac{1}{p+1} (n \sum_{i=1}^p (n+1)^i - \sum_{i=2}^p \binom{i}{p+1} \sum_{k=1}^n k^{p+1-i})$$

6.22 歐幾里得類算法

- $m = \lfloor \frac{an+b}{c} \rfloor$
- Time complexity: $O(\log n)$

$$f(a, b, c, n) = \sum_{i=0}^n \left\lfloor \frac{ai+b}{c} \right\rfloor$$

$$= \begin{cases} \left\lfloor \frac{a}{c} \right\rfloor \cdot \frac{n(n+1)}{2} + \left\lfloor \frac{b}{c} \right\rfloor \cdot (n+1) \\ + f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee b < 0 \\ nm - f(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$

$$g(a, b, c, n) = \sum_{i=0}^n i \left\lfloor \frac{ai+b}{c} \right\rfloor$$

$$= \begin{cases} \left\lfloor \frac{a}{c} \right\rfloor \cdot \frac{n(n+1)(2n+1)}{6} + \left\lfloor \frac{b}{c} \right\rfloor \cdot \frac{n(n+1)}{2} \\ + g(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee b < 0 \\ \frac{1}{2} \cdot (n(n+1)m - f(c, c-b-1, a, m-1) \\ - h(c, c-b-1, a, m-1)), & \text{otherwise} \end{cases}$$

$$h(a, b, c, n) = \sum_{i=0}^n \left\lfloor \frac{ai+b}{c} \right\rfloor^2$$

$$= \begin{cases} \left\lfloor \frac{a}{c} \right\rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \left\lfloor \frac{b}{c} \right\rfloor^2 \cdot (n+1) \\ + \left\lfloor \frac{a}{c} \right\rfloor \cdot \left\lfloor \frac{b}{c} \right\rfloor \cdot n(n+1) \\ + h(a \bmod c, b \bmod c, c, n) \\ + 2 \left\lfloor \frac{a}{c} \right\rfloor \cdot g(a \bmod c, b \bmod c, c, n) \\ + 2 \left\lfloor \frac{b}{c} \right\rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee b < 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) \\ - 2f(c, c-b-1, a, m-1) - f(a, b, c, n), & \text{otherwise} \end{cases}$$

$$h(a, b, c, n) = \sum_{i=0}^n \left\lfloor \frac{ai+b}{c} \right\rfloor^2$$

$$= \begin{cases} \left\lfloor \frac{a}{c} \right\rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \left\lfloor \frac{b}{c} \right\rfloor^2 \cdot (n+1) \\ + \left\lfloor \frac{a}{c} \right\rfloor \cdot \left\lfloor \frac{b}{c} \right\rfloor \cdot n(n+1) \\ + h(a \bmod c, b \bmod c, c, n) \\ + 2 \left\lfloor \frac{a}{c} \right\rfloor \cdot g(a \bmod c, b \bmod c, c, n) \\ + 2 \left\lfloor \frac{b}{c} \right\rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee b < 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) \\ - 2f(c, c-b-1, a, m-1) - f(a, b, c, n), & \text{otherwise} \end{cases}$$

6.23 生成函數

- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$

$$\begin{aligned} - A(rx) &\Rightarrow r^n a_n \\ - A(x) + B(x) &\Rightarrow a_n + b_n \\ - A(x)B(x) &\Rightarrow \sum_{i=0}^n a_i b_{n-i} \\ - A(x)^k &\Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k} \end{aligned}$$

$$\begin{aligned} - xA(x)' &\Rightarrow na_n \\ - \frac{A(x)}{1-x} &\Rightarrow \sum_{i=0}^n a_i \end{aligned}$$

- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$

$$\begin{aligned} - A(x) + B(x) &\Rightarrow a_n + b_n \\ - A^{(k)}(x) &\Rightarrow a_{n+k} \\ - A(x)B(x) &\Rightarrow \sum_{i=0}^n \binom{n}{i} a_i b_{n-i} \\ - A(x)^k &\Rightarrow \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k} \\ - xA(x) &\Rightarrow na_n \end{aligned}$$

- Special Generating Function

$$\begin{aligned} - (1+x)^n &= \sum_{i \geq 0} \binom{n}{i} x^i \\ - \frac{1}{(1-x)^n} &= \sum_{i \geq 0} \binom{i+n-1}{n-1} x^i \end{aligned}$$

7 Misc

7.1 fast

```
#pragma GCC optimize("Ofast,no-stack-protector,unroll-loops,fast-math,inline")
inline char gc() {
    static const size_t sz = 65536;
    static char buf[sz];
    static char *p = buf, *end = buf;
    if(p == end) end = buf + fread(buf, 1, sz, stdin), p = buf;
    return *p++;
}
```

otherwise

7.2 for

```
#define FOR(i, begin, end) for(int i = (begin), i##_end_ = (end); i < i##_end_; i++)
#define IFOR(i, begin, end) for(int i = (end) - 1, i##_begin_ = (begin); i >= i##_begin_; i--)
#define REP(i, n) FOR(i, 0, n)
#define IREP(i, n) IFOR(i, 0, n)
```

7.3 next-combination

```
// Example: 1 -> 2, 4 -> 8, 12(1100) -> 17(1001)
ll next_combination(ll comb) {
    ll x = comb & -comb, y = comb + x;
    return ((comb & ~y) / x >> 1) | y;
}
```

7.4 PBDS

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<ll, null_type, less<ll>, rb_tree_tag, tree_order_statistics_node_update> st;
// find_by_order order_of_key
// __float128_t
for(int i = bs._Find_first(); i < bs.size(); i = bs._Find_next(i));
```

7.5 python

```
from decimal import Decimal, getcontext
getcontext().prec = 1000000000
getcontext().Emax = 9999999999
a = pow(Decimal(2), 82589933) - 1
```

7.6 rng

```
inline ull rng() {
    static ull Q = 48763;
    Q ^= Q << 7;
    Q ^= Q >> 9;
    return Q & 0xFFFFFFFFFULL;
}
```

7.7 rotate90

```
vector<vector<T>> rotate90(const vector<vector<T>>& a) {
    int n = sz(a), m = sz(a[0]);
    vector<vector<T>> b(m, vector<T>(n));
    REP(i, n) REP(j, m) b[j][i] = a[i][m-1-j];
    return b;
}
```

7.8 timer

```
clock_t T1 = clock();
double getCurrentTime() { return (double) (clock() - T1) / CLOCKS_PER_SEC; }
```

8 String

8.1 AC

```

1 template<int ALPHABET = 26, char MIN_CHAR =
  'a'>
2 struct ac_automaton {
3     struct Node {
4         int fail = 0, cnt = 0;
5         array<int, ALPHABET> go{};
6     };
7     vector<Node> node;
8     vi que;
9     int new_node() { return node.eb(), SZ(node)
10         - 1; }
11     Node& operator[](int i) { return node[i];
12     }
13     ac_automaton() { new_node(); }
14     int insert(const string& s) {
15         int p = 0;
16         for(char c : s) {
17             int v = c - MIN_CHAR;
18             if(node[p].go[v] == 0) node[p].go[v] =
19                 new_node();
20             p = node[p].go[v];
21         }
22         node[p].cnt++;
23         return p;
24     }
25     void build() {
26         que.reserve(SZ(node)); que.pb(0);
27         REP(i, SZ(que)) {
28             int u = que[i];
29             REP(j, ALPHABET) {
30                 if(node[u].go[j] == 0) node[u].go[j]
31                     = node[node[u].fail].go[j];
32                 else {
33                     int v = node[u].go[j];
34                     node[v].fail = (u == 0 ? u : node[
35                         node[u].fail].go[j]);
36                     que.pb(v);
37                 }
38             }
39         }
40     }
41 };

```

8.2 KMP

```

1 // abacbabab -> [0, 0, 1, 0, 0, 1, 2, 3]
2 vi KMP(const vi& a) {
3     int n = SZ(a);
4     vi k(n);
5     for(int i = 1; i < n; ++i) {
6         int j = k[i - 1];
7         while(j > 0 && a[i] != a[j]) j = k[j -
8             1];
9         j += (a[i] == a[j]);
10        k[i] = j;
11    }
12 }

```

```

11 return k;
12 }

```

8.3 LCP

```

1 vi lcp(const vi& s, const vi& sa) {
2     int n = SZ(s), h = 0;
3     vi rnk(n), lcp(n - 1);
4     REP(i, n) rnk[sa[i]] = i;
5     REP(i, n) {
6         h -= (h > 0);
7         if(rnk[i] == 0) continue;
8         int j = sa[rnk[i] - 1];
9         for(; j + h < n && i + h < n; h++) if(s[
10             j + h] != s[i + h]) break;
11         lcp[rnk[i] - 1] = h;
12     }
13     return lcp;
14 }

```

8.4 manacher

```

1 // Length: (z[i] - (i & 1)) / 2 * 2 + (i &
  1)
2 vi manacher(string t) {
3     string s = "&";
4     for(char c : t) s.pb(c), s.pb('&');
5     int l = 0, r = 0;
6     vi z(sz(s));
7     REP(i, sz(s)) {
8         z[i] = r > i ? min(z[2 * l - i], r - i)
9             : 1;
10        while(s[i + z[i]] == s[i - z[i]]) z[i]
11            ++;
12        if(z[i] + i > r) r = z[i] + 1, l = i;
13    }
14    return z;
15 }

```

8.5 rolling-hash

```

1 const ll M = 911382323, mod = 972663749;
2 ll Get(vector<ll>& h, int l, int r) {
3     if(!l) return h[r]; // p[i] = M^i % mod
4     ll ans = (h[r] - h[l - 1] * p[r - l + 1])
5         % mod;
6     return (ans + mod) % mod;
7 }
8 vector<ll> Hash(string s) {
9     vector<ll> ans(sz(s));
10    ans[0] = s[0];
11    for(int i = 1; i < sz(s); ++i) ans[i] = (
12        ans[i - 1] * M + s[i]) % mod;
13    return ans;
14 }

```

8.6 SAIS

```

1 // mississippi
2 // 10 7 4 1 0 9 8 6 3 5 2
3 vi SAIS(string a) {
4     #define QQ(i, n) for(int i = (n); i >= 0;
5         i--)
6     int n = sz(a), m = *max_element(all(a)) +
7         1;
8     vi pos(m + 1), x(m), sa(n), val(n), lms;
9     for(auto c : a) pos[c + 1]++;
10    REP(i, m) pos[i + 1] += pos[i];
11    vector<bool> s(n);
12    QQ(i, n - 2) s[i] = a[i] != a[i + 1] ? a[i]
13        < a[i + 1] : s[i + 1];
14    auto ind = [&](const vi& ls) {
15        fill(all(sa), -1);
16        auto L = [&](int i) { if(i >= 0 && !s[i]
17            ) sa[x[a[i]]++] = i; };
18        auto S = [&](int i) { if(i >= 0 && s[i])
19            sa[--x[a[i]]] = i; };
20        REP(i, m) x[i] = pos[i + 1];
21        QQ(i, m) x[i] = pos[i];
22        L(n - 1);
23        REP(i, n) L(sa[i] - 1);
24        REP(i, m) x[i] = pos[i + 1];
25        QQ(i, n - 1) S(sa[i] - 1);
26    };
27    auto ok = [&](int i) { return i == n || (!
28        s[i - 1] && s[i]); };
29    auto same = [&](int i, int j) {
30        do {
31            if(a[i++] != a[j++]) return false;
32        } while(!ok(i) && !ok(j));
33        return ok(i) && ok(j);
34    };
35    for(int i = 1; i < n; i++) if(ok(i)) lms.
36        pb(i);
37    ind(lms);
38    if(sz(lms)) {
39        int p = -1, w = 0;
40        for(auto v : sa) if(v && ok(v)) {
41            if(p != -1 && same(p, v)) w--;
42            val[p = v] = w++;
43        }
44        auto b = lms;
45        for(auto& v : b) v = val[v];
46        b = SAIS(b);
47        for(auto& v : b) v = lms[v];
48        ind(b);
49    }
50    return sa;
51 }

```

8.7 SAM

```

1 // cnt 要先用 bfs 往回推, 第一次出現的位置是
  state.first_pos - |S| + 1
2 struct Node { int go[26], len, link, cnt,
3     first_pos; };
4 Node SA[N]; int sz;

```

```

4 void sa_init() { SA[0].link = -1, SA[0].len
  = 0, sz = 1; }
5 int sa_extend(int p, int c) {
6     int u = sz++;
7     SA[u].first_pos = SA[u].len = SA[p].len +
8         1;
9     SA[u].cnt = 1;
10    while(p != -1 && SA[p].go[c] == 0) {
11        SA[p].go[c] = u;
12        p = SA[p].link;
13    }
14    if(p == -1) {
15        SA[u].link = 0;
16        return u;
17    }
18    int q = SA[p].go[c];
19    if(SA[p].len + 1 == SA[q].len) {
20        SA[u].link = q;
21        return u;
22    }
23    int x = sz++;
24    SA[x] = SA[q];
25    SA[x].cnt = 0;
26    SA[x].len = SA[p].len + 1;
27    SA[q].link = SA[u].link = x;
28    while(p != -1 && SA[p].go[c] == q) {
29        SA[p].go[c] = x;
30        p = SA[p].link;
31    }
32    return u;
33 }

```

8.8 smallest-rotation

```

1 string small_rot(string s) {
2     int n = sz(s), i = 0, j = 1;
3     s += s;
4     while(i < n && j < n) {
5         int k = 0;
6         while(k < n && s[i + k] == s[j + k]) k
7             ++;
8         if(s[i + k] <= s[j + k]) j += k + 1;
9         else i += k + 1;
10        if(i == j) j++;
11    }
12    int ans = i < n ? i : j;
13    return s.substr(ans, n);
14 }

```

8.9 Z

```

1 // abacbabab -> [0, 0, 1, 0, 0, 3, 0, 1]
2 vi z_algorithm(const vi& a) {
3     int n = sz(a);
4     vi z(n);
5     for(int i = 1, j = 0; i < n; ++i) {
6         if(i <= j + z[j]) z[i] = min(z[i - j],
7             j + z[j] - i);
8         while(i + z[i] < n && a[i + z[i]] == a[
9             z[i]]) z[i]++;
10    }
11 }

```

```
8 |         if(i + z[i] > j + z[j]) j = i;  
9 |     }  
10 |     return z;  
11 | }
```


ACM ICPC Judge Test - NTHU SplayTreap

C++ Resource Test

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 namespace system_test {
5
6 const size_t KB = 1024;
7 const size_t MB = KB * 1024;
8 const size_t GB = MB * 1024;
```

```
9 size_t block_size, bound;
10 void stack_size_dfs(size_t depth = 1) {
11     if (depth >= bound)
12         return;
13     int8_t ptr[block_size]; // 若無法編譯將
14                             // block_size 改成常數
15     memset(ptr, 'a', block_size);
16     cout << depth << endl;
17     stack_size_dfs(depth + 1);
18 }
19
20 void stack_size_and_runtime_error(size_t
21     block_size, size_t bound = 1024) {
22     system_test::block_size = block_size;
23     system_test::bound = bound;
24     stack_size_dfs();
25 }
26
27 double speed(int iter_num) {
28     const int block_size = 1024;
29     volatile int A[block_size];
30     auto begin = chrono::high_resolution_clock
31         ::now();
32     while (iter_num--)
33         for (int j = 0; j < block_size; ++j)
34             A[j] += j;
35     auto end = chrono::high_resolution_clock::
36         now();
```

```
37 chrono::duration<double> diff = end -
38     begin;
39     return diff.count();
40 }
41
42 void runtime_error_1() {
43     // Segmentation fault
44     int *ptr = nullptr;
45     *(ptr + 7122) = 7122;
46 }
47
48 void runtime_error_2() {
49     // Segmentation fault
50     int *ptr = (int *)memset;
51     *ptr = 7122;
52 }
53
54 void runtime_error_3() {
55     // munmap_chunk(): invalid pointer
56     int *ptr = (int *)memset;
57     delete ptr;
58 }
59
60 void runtime_error_4() {
61     // free(): invalid pointer
62     int *ptr = new int[7122];
63     ptr += 1;
64     delete[] ptr;
65 }
```

```
66
67 void runtime_error_5() {
68     // maybe illegal instruction
69     int a = 7122, b = 0;
70     cout << (a / b) << endl;
71 }
72
73 void runtime_error_6() {
74     // floating point exception
75     volatile int a = 7122, b = 0;
76     cout << (a / b) << endl;
77 }
78
79 void runtime_error_7() {
80     // call to abort.
81     assert(false);
82 }
83
84 // namespace system_test
85
86 #include <sys/resource.h>
87 void print_stack_limit() { // only work in
88     Linux
89     struct rlimit l;
90     getrlimit(RLIMIT_STACK, &l);
91     cout << "stack_size = " << l.rlim_cur << "
92         byte" << endl;
93 }
```