

ACM ICPC Team Reference - NTHU SplayTreap

Contents

1 Basic	2				
1.1 vimrc	2				
2 Data-Structure	2				
2.1 CHT	2				
2.2 DLX	2				
2.3 fast-set	2				
2.4 lazysegtree	2				
2.5 rect-add-rect-sum	3				
2.6 rollback-dsu	3				
2.7 segtree	3				
		2.8 sparse-table	4		
		2.9 static-range-inversion	4		
		2.10 static-range-lis	4		
		2.11 treap	5		
		2.12 union-of-rectangles	5		
		3 Flow-Matching	5		
		3.1 bipartite-matching	5		
		3.2 Dinic-LowerBound	5		
		3.3 Dinic	6		
		3.4 Flow 建模	6		
		3.5 general-matching	6		
		3.6 general-weighted-max-matching	6		
		3.7 KM	7		
		3.8 max-clique	8		
		3.9 MCMF	8		
		3.10 minimum-general-weighted-perfect-matching	8		
		4 Geometry	8		
		4.1 closest-pair	8		
		4.2 convex-hull	9		
		4.3 point-in-convex-hull	9		
		4.4 point	9		
		4.5 polar-angle-sort	9		
		5 Graph	9		
		5.1 2-SAT	9		
		5.2 centroid-tree	9		
		5.3 HLD	9		
		5.4 lowlink	10		
		5.5 SCC	10		
		6 Math	10		
		6.1 Aliens	10		
		6.2 Berlekamp-Massey	10		
		6.3 Chinese-Remainder	11		
		6.4 Combination	11		
		6.5 Determinant	11		
		6.6 Discrete-Log	11		
		6.7 extgcd	11		
		6.8 Floor-Sum	11		
		6.9 FWHT	11		
		6.10 Gauss-Jordan	12		
		6.11 Int-Div	12		
		6.12 Linear-Sieve	12		
		6.13 Miller-Rabin	12		
		6.14 Min-of-Mod-of-Linear	12		
		6.15 Mod-Inv	12		
		6.16 Pollard-Rho	12		
		6.17 Primes	12		
		6.18 Xor-Basis	12		
		6.19 估計值	13		
		6.20 定理	13		
		6.21 數字	13		
		6.22 歐幾里得類算法	14		
		6.23 生成函數	14		
		7 Misc	14		
		7.1 fast	14		
		7.2 next-combination	14		
		7.3 PBDS	14		
		7.4 python	14		
		7.5 rng	14		
		7.6 rotate90	14		
		7.7 timer	14		
		8 String	14		
		8.1 AC	14		
		8.2 KMP	14		
		8.3 LCP	14		
		8.4 manacher	14		
		8.5 rolling-hash	15		
		8.6 SAIS	15		
		8.7 SAM	15		
		8.8 smallest-rotation	15		
		8.9 Z	15		

1 Basic

1.1 vimrc

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

2 Data-Structure

2.1 CHT

```
struct line_t {
    mutable ll k, m, p;
    bool operator<(const line_t& o) const {
        return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};
template<bool MAX>
struct CHT : multiset<line_t, less<>> {
    const ll INF = 1e18L;
    bool isect(iterator x, iterator y) {
        if(y == end()) return x->p = INF, 0;
        if(x->k == y->k) {
            x->p = (x->m > y->m ? INF : -INF);
        } else {
            x->p = floor_div(y->m - x->m, x->k - y->k); // see Math
        }
        return x->p >= y->p;
    }
    void add_line(ll k, ll m) {
        if(!MAX) k = -k, m = -m;
        auto z = insert({k, m, 0}), y = z++, x = y;
        while(isect(y, z)) z = erase(z);
        if(x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
    }
    ll get(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return (1.k * x + 1.m) * (MAX ? +1 : -1)
            ;
    }
};
```

2.2 DLX

```
struct DLX {
    int n, m, tot, ans;
    vi first, siz, L, R, U, D, col, row, stk;
    DLX(int n, int m) : n(n), m(m), tot(m) {
        int sz = n * m;
        first = siz = L = R = U = D = col = row = stk = vi(sz);
        REP(i, m + 1) {
            L[i] = i - 1, R[i] = i + 1;
            U[i] = D[i] = i;
        }
        L[0] = m, R[m] = 0;
    }
    void insert(int r, int c) {
        r++, c++;
        col[++tot] = c, row[tot] = r, ++siz[c];
        D[tot] = D[c], U[D[c]] = tot, U[tot] = c,
            D[c] = tot;
        if(!first[r]) first[r] = L[tot] = R[tot] = tot;
        else {
            L[R[tot]] = R[first[r]] = tot;
            R[L[tot]] = first[r] = tot;
        }
    }
#define TRAV(i, X, j) for(i = X[j]; i != j; i = X[i])
    void remove(int c) {
        int i, j;
        L[R[c]] = L[c], R[L[c]] = R[c];
        TRAV(i, D, c) TRAV(j, R, i) {
            D[U[D[j]]] = U[j] = D[j];
            siz[col[j]]--;
        }
    }
    void recover(int c) {
        int i, j;
        TRAV(i, U, c) TRAV(j, L, i) {
            U[D[j]] = D[U[j]] = j;
            siz[col[j]]++;
        }
        L[R[c]] = R[L[c]] = c;
    }
    bool dance(int dep) {
        if(!R[0]) return ans = dep, true;
        int i, j, c = R[0];
        TRAV(i, R, 0) if(siz[i] < siz[c]) c = i;
        remove(c);
        TRAV(i, D, c) {
            stk[dep] = row[i];
            TRAV(j, R, i) remove(col[j]);
            if(dance(dep + 1)) return true;
            TRAV(j, L, i) recover(col[j]);
        }
        recover(c);
        return false;
    }
    vi solve() {
        if(!dance(1)) return {};
        return vi(stk.begin() + 1, stk.begin() + ans);
    }
};
```

2.3 fast-set

```
// Can correctly work with numbers in range [0; MAXN]
// Supports all std::set operations in O(1) on random queries / dense arrays, O(Log64(N)) in worst case (sparse array).
// Count operation works in O(1) always.
template<uint MAXN>
class fast_set {
private:
    static const uint PREF = (MAXN <= 64 ? 0 : MAXN <= 4096 ? 1 : MAXN <= 262144 ? 1 + 64 : MAXN <= 16777216 ? 1 + 64 + 4096 : MAXN <= 1073741824 ? 1 + 64 + 4096 + 262144 : 227) + 1;
    static constexpr ull lb(int x) { if(x == 64) return ULLONG_MAX; return (1ULL << x) - 1; };
    static const uint SZ = PREF + (MAXN + 63) / 64 + 1;
    ull m[SZ] = {0};
    inline uint left(uint v) const { return (v - 62) * 64; }
    inline uint parent(uint v) const { return v / 64 + 62; }
    inline void setbit(uint v) { m[v >> 6] |= 1ULL << (v & 63); }
    inline void resetbit(uint v) { m[v >> 6] &= ~(1ULL << (v & 63)); }
    inline uint getbit(uint v) const { return m[v >> 6] >> (v & 63) & 1; }
    inline uint childs_value(uint v) const { return m[left(v) >> 6]; }
    inline int left_go(uint x, const uint c) const { const ull rem = x & 63; uint bt = PREF * 64 + x; ull num = m[bt >> 6] & lb(rem + c); if(num) return (x ^ rem) | __lg(num); for(bt = parent(bt); bt > 62; bt = parent(bt)) { const ull rem = bt & 63; num = m[bt >> 6] & lb(rem); if(num) { if(id()) { bt = (bt ^ rem) | __lg(num); break; } } } if(bt == 62) return -1; while(bt < PREF * 64) bt = left(bt) | __lg(m[bt - 62]); return bt - PREF * 64; }
    inline int right_go(uint x, const uint c) const { const ull rem = x & 63; uint bt = PREF * 64 + x; ull num = m[bt >> 6] & ~lb(rem + c); if(num) return (x ^ rem) | __builtin_ctzll(num); }
```

```
for(bt = parent(bt); bt > 62; bt = parent(bt)) { const ull rem = bt & 63; num = m[bt >> 6] & ~lb(rem + 1); if(num) { bt = (bt ^ rem) | __builtin_ctzll(num); break; } }
if(bt == 62) return -1;
while(bt < PREF * 64) bt = left(bt) | __builtin_ctzll(m[bt - 62]);
return bt - PREF * 64; }

public:
fast_set() { assert(PREF != 228); setbit(62); }
bool empty() const { return getbit(63); }
void clear() { fill(m, m + SZ, 0); setbit(62); }
bool count(uint x) const { return m[PREF + (x >> 6)] >> (x & 63) & 1; }
void insert(uint x) { for(uint v = PREF * 64 + x; !getbit(v); v = parent(v)) setbit(v); }
void erase(uint x) { if(!getbit(PREF * 64 + x)) return; resetbit(PREF * 64 + x); for(uint v = parent(PREF * 64 + x); v > 62 && !childs_value(v); v = parent(v)) resetbit(v); }
int find_next(uint x) const { return right_go(x, 0); } // >=
int find_prev(uint x) const { return left_go(x, 1); } // <=
};
```

2.4 lazysegtree

```
template<class S,
        S (*e)(),
        S (*op)(S, S),
        class F,
        F (*id)(),
        S (*mapping)(F, S),
        F (*composition)(F, F)>
struct lazy_segtree {
    int n, size, log;
    vector<S> d; vector<F> lz;
    void update(int k) { d[k] = op(d[k << 1], d[k << 1 | 1]); }
    void all_apply(int k, F f) { d[k] = mapping(f, d[k]); if(k < size) lz[k] = composition(f, lz[k]); }
}
void push(int k) { all_apply(k << 1, lz[k]); all_apply(k << 1 | 1, lz[k]); lz[k] = id(); }
```

```

20 }
21 lazy_segtree(int _n) : lazy_segtree(vector<S>(_n, e())) {}
22 lazy_segtree(const vector<S>& v) : n(sz(v)) {
23     log = __lg(2 * n - 1), size = 1 << log;
24     d.resize(size * 2, e());
25     lz.resize(size, id());
26     REP(i, n) d[size + i] = v[i];
27     for(int i = size - 1; i; i--) update(i);
28 }
29 void set(int p, S x) {
30     p += size;
31     for(int i = log; i; i--) push(p >> i);
32     d[p] = x;
33     for(int i = 1; i <= log; ++i) update(p >> i);
34 }
35 S get(int p) {
36     p += size;
37     for(int i = log; i; i--) push(p >> i);
38     return d[p];
39 }
40 S prod(int l, int r) {
41     if(l == r) return e();
42     l += size; r += size;
43     for(int i = log; i; i--) {
44         if(((l >> i) << i) != 1) push(l >> i);
45         if(((r >> i) << i) != 1) push(r >> i);
46     }
47     S sm1 = e(), smr = e();
48     while(l < r) {
49         if(l & 1) sm1 = op(sm1, d[l++]);
50         if(r & 1) smr = op(d[--r], smr);
51         l >>= 1;
52         r >>= 1;
53     }
54     return op(sm1, smr);
55 }
56 S all_prod() const { return d[1]; }
57 void apply(int p, F f) {
58     p += size;
59     for(int i = log; i; i--) push(p >> i);
60     d[p] = mapping(f, d[p]);
61     for(int i = 1; i <= log; i++) update(p >> i);
62 }
63 void apply(int l, int r, F f) {
64     if(l == r) return;
65     l += size; r += size;
66     for(int i = log; i; i--) {
67         if(((l >> i) << i) != 1) push(l >> i);
68         if(((r >> i) << i) != 1) push((r - 1) >> i);
69     }
70     {
71         int l2 = l, r2 = r;
72         while(l < r) {
73             if(l & 1) all_apply(l++, f);
74             if(r & 1) all_apply(--r, f);
75             l >>= 1;
76             r >>= 1;
77         }
78         l = l2;
79         r = r2;
80     }

```

2.5 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;

```

```

81     for(int i = 1; i <= log; i++) {
82         if(((l >> i) << i) != 1) update(l >> i);
83         if(((r >> i) << i) != 1) update((r - 1) >> i);
84     }
85 }
86 template<class G> int max_right(int l, G g) {
87     assert(0 <= l && l <= n && g(e()));
88     if(l == n) return n;
89     l += size;
90     for(int i = log; i; i--) push(l >> i);
91     S sm = e();
92     do {
93         while(!(l & 1)) l >>= 1;
94         if(!g(op(sm, d[l]))) {
95             while(l < size) {
96                 push(l);
97                 l <<= 1;
98                 if(g(op(sm, d[l]))) sm = op(sm, d[l++]);
99             }
100             return l - size;
101         }
102         sm = op(sm, d[l++]);
103     } while((l & -1) != 1);
104     return n;
105 }
106 template<class G> int min_left(int r, G g) {
107     assert(0 <= r && r <= n && g(e()));
108     if(r == 0) return 0;
109     r += size;
110     for(int i = log; i >= 1; i--) push((r - 1) >> i);
111     S sm = e();
112     do {
113         r--;
114         while(r > 1 && (r & 1)) r >>= 1;
115         if(!g(op(d[r], sm))) {
116             while(r < size) {
117                 push(r);
118                 r <<= 1;
119                 if(g(op(d[r], sm))) sm = op(d[r--], sm);
120             }
121             return r + 1 - size;
122         }
123         sm = op(d[r], sm);
124     } while((r & -r) != r);
125     return 0;
126 }
127 }

```

```

7 // A[x][y] += val for(x, y) in [xl, xr) * [yl, yr)
8 void add_rectangle(Int xl, Int xr, Int yl, Int yr, T val) { add_qry.pb({xl, xr, yl, yr, val}); }
9 // Get sum of A[x][y] for(x, y) in [xl, xr) * [yl, yr)
10 void add_query(Int xl, Int xr, Int yl, Int yr) { sum_qry.pb({xl, xr, yl, yr}); }
11 vector<T> solve() {
12     vector<Int> ys;
13     for(auto &a : add_qry) ys.pb(a.yl), ys.pb(a.yr);
14     ys = sort_unique(ys);
15     const int Y = SZ(ys);
16     vector<tuple<Int, int, int>> ops;
17     REP(q, SZ(sum_qry)) {
18         ops.pb(sum_qry[q].xl, 0, q);
19         ops.pb(sum_qry[q].xr, 1, q);
20     }
21     REP(q, SZ(add_qry)) {
22         ops.pb(add_qry[q].xl, 2, q);
23         ops.pb(add_qry[q].xr, 3, q);
24     }
25     sort(ALL(ops));
26     fenwick<T> b00(Y), b01(Y), b10(Y), b11(Y);
27     vector<T> ret(SZ(sum_qry));
28     for(auto o : ops) {
29         int qtype = get<1>(o), q = get<2>(o);
30         if(qtype >= 2) {
31             const auto& query = add_qry[q];
32             int i = lower_bound(ALL(ys), query.yl) - ys.begin();
33             int j = lower_bound(ALL(ys), query.yr) - ys.begin();
34             T x = get<0>(o);
35             T yi = query.yl, yj = query.yr;
36             if(qtype & 1) swap(i, j), swap(yi, yj);
37             b00.add(i, x * yi * query.val);
38             b01.add(i, -x * query.val);
39             b10.add(i, -yi * query.val);
40             b11.add(i, query.val);
41             b00.add(j, -x * yj * query.val);
42             b01.add(j, x * query.val);
43             b10.add(j, yj * query.val);
44             b11.add(j, -query.val);
45         } else {
46             const auto& query = sum_qry[q];
47             int i = lower_bound(ALL(ys), query.yl) - ys.begin();
48             int j = lower_bound(ALL(ys), query.yr) - ys.begin();
49             T x = get<0>(o);
50             T yi = query.yl, yj = query.yr;
51             if(qtype & 1) swap(i, j), swap(yi, yj);
52             ret[q] += b00.get(i - 1) + b01.get(i - 1) * x + b11.get(i - 1) * x * yi;
53             ret[q] -= b00.get(j - 1) + b01.get(j - 1) * x + b11.get(j - 1) * x * yj;
54         }
55     }

```

2.6 rollback-dsu

```

1 struct RollbackDSU {
2     int n; vi sz, tag;
3     vector<tuple<int, int, int, int>> op;
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.ppb();
27             sz[x] = sx;
28             sz[y] = sy;
29         }
30     }
31 };

```

2.7 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 << 1], st[v << 1 | 1]); }
6     segtree(int _n) : segtree(vector<S>(_n, e())) {}
7     segtree(const vector<S>& a) : n(sz(a)) {
8         log = __lg(2 * n - 1), size = 1 << log;
9         st.resize(size << 1, e());
10        REP(i, n) st[size + i] = a[i];
11        for(int i = size - 1; i; i--) update(i);
12    }
13    void set(int p, S val) {
14        st[p += size] = val;
15        for(int i = 1; i <= log; ++i) update(p >> i);
16    }

```

```

17 S get(int p) const {
18     return st[p + size];
19 }
20 S prod(int l, int r) const {
21     assert(0 <= l && l <= r && r <= n);
22     S sm1 = e(), smr = e();
23     l += size, r += size;
24     while(l < r) {
25         if(l & 1) sm1 = op(sm1, st[l++]);
26         if(r & 1) smr = op(st[--r], smr);
27         l >>= 1;
28         r >>= 1;
29     }
30     return op(sm1, smr);
31 }
32 S all_prod() const { return st[1]; }
33 template<class F> int max_right(int l, F f)
34     const {
35     assert(0 <= l && l <= n && f(e()));
36     if(l == n) return n;
37     l += size;
38     S sm = e();
39     do {
40         while(~l & 1) l >>= 1;
41         if(!f(op(sm, st[l]))) {
42             while(l < size) {
43                 l <<= 1;
44                 if(f(op(sm, st[l]))) sm = op(sm,
45                     st[l++]);
46             }
47             return l - size;
48         }
49         sm = op(sm, st[l++]);
50     } while((l & -1) != 1);
51     return n;
52 }
53 template<class F> int min_left(int r, F f)
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     return 0;
73 }

```

2.8 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 };

```

2.9 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);

```

```

39         fenw.add(a[i], -1);
40     };
41     int l = 0, r = 0;
42     for(int i = 0; i < q; i++) {
43         int id = ord[i], ql = L[id], qr = R[id]
44             ;
45         while(l > ql) AL(--l);
46         while(r < qr) AR(++r);
47         while(l < ql) DL(l++);
48         while(r > qr) DR(--r);
49         ans[id] = cnt;
50     }
51     return ans;
52 };

```

2.10 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) {
7         // a must be permutation of [0, n)
8         REP(i, n) invp[a[i]] = i;
9     }
10    void add_query(int l, int r) { qry[l].pb({
11        r, SZ(ans)}), ans.pb(r - 1); }
12    void unit_monge_mult(vi& a, vi& b, vi& r)
13    {
14        int n = SZ(a);
15        if(n == 2) {
16            if(!a[0] && !b[0]) r[0] = 0, r[1] = 1;
17            else r[0] = 1, r[1] = 0;
18            return;
19        }
20        if(n == 1) return r[0] = 0, void();
21        int d = n / 2;
22        vi a1(d), a2(n - d), b1(d), b2(n - d);
23        vi mpa1(d), mpa2(n - d), mpb1(d), mpb2(n
24            - d);
25        int p[2] = {};
26        REP(i, n) {
27            if(a[i] < d) a1[p[0]] = a[i], mpa1[p
28                [0]++] = i;
29            else a2[p[1]] = a[i] - d, mpa2[p[1]++]
30                = i;
31        }
32        p[0] = p[1] = 0;
33        REP(i, n) {
34            if(b[i] < d) b1[p[0]] = b[i], mpb1[p
35                [0]++] = i;
36            else b2[p[1]] = b[i] - d, mpb2[p[1]++]
37                = i;
38        }
39        vi c1(d), c2(n - d);
40        unit_monge_mult(a1, b1, c1),
41        unit_monge_mult(a2, b2, c2);
42        vi cpx(n), cpy(n), cqx(n), cqy(n);
43        REP(i, d) cpx[mpa1[i]] = mpb1[c1[i]],
44            cpy[mpa1[i]] = 0;

```

```

35        REP(i, n - d) cpx[mpa2[i]] = mpb2[c2[i]
36            ], cpy[mpa2[i]] = 1;
37        REP(i, n) r[i] = cpx[i];
38        REP(i, n) cqx[cpx[i]] = i, cqy[cpx[i]] =
39            cpy[i];
40        int hi = n, lo = n, his = 0, los = 0;
41        REP(i, n) {
42            if(cqy[i] ^ (cqx[i] >= hi)) his--;
43            while(hi > 0 && his < 0) {
44                hi--;
45                if(cpy[hi] ^ (cpx[hi] > i)) his++;
46            }
47            while(lo > 0 && los <= 0) {
48                lo--;
49                if(cpy[lo] ^ (cpx[lo] >= i)) los++;
50            }
51            if(los > 0 && hi == lo) r[lo] = i;
52            if(cqy[i] ^ (cqx[i] >= lo)) los--;
53        }
54    }
55    void subunit_monge_mult(vi& a, vi& b, vi&
56        c) {
57        int n = SZ(a);
58        vi za(n), zb(n), res(n), vis(n), mpa(n,
59            -1), mpb(n, -1), rb(n, -1);
60        int ca = n;
61        IREP(i, n) if(a[i] != -1) vis[a[i]] = 1,
62            za[--ca] = a[i], mpa[ca] = i;
63        IREP(i, n) if(!vis[i]) za[--ca] = i;
64        fill(ALL(vis), -1);
65        REP(i, n) if(b[i] != -1) vis[b[i]] = i;
66        ca = 0;
67        REP(i, n) if(vis[i] != -1) mpb[ca] = i,
68            rb[vis[i]] = ca++;
69        REP(i, n) if(rb[i] == -1) rb[i] = ca++;
70        REP(i, n) zb[rb[i]] = i;
71        unit_monge_mult(za, zb, res);
72        fill(ALL(c), -1);
73        REP(i, n) if(mpa[i] != -1 && mpb[res[i]]
74            != -1) c[mpa[i]] = mpb[res[i]];
75    }
76    void solve(vi& p, vi& ret) {
77        int n = SZ(p);
78        if(n == 1) return ret[0] = -1, void();
79        int d = n / 2;
80        vi pl(d), pr(n - d);
81        REP(i, d) pl[i] = p[i];
82        REP(i, n - d) pr[i] = p[i + d];
83        vi vis(n, -1);
84        REP(i, d) vis[pl[i]] = i;
85        vi tl(d), tr(n - d), mpl(d), mpr(n - d);
86        int ca = 0;
87        REP(i, n) if(vis[i] != -1) mpl[ca] = i,
88            tl[vis[i]] = ca++;
89        ca = 0;
90        fill(ALL(vis), -1);
91        REP(i, n - d) vis[pr[i]] = i;
92        REP(i, n) if(vis[i] != -1) mpr[ca] = i,
93            tr[vis[i]] = ca++;
94        vi vl(d), vr(n - d);
95        solve(tl, vl), solve(tr, vr);
96        vi sl(n), sr(n);
97        iota(ALL(sl), 0), iota(ALL(sr), 0);
98        REP(i, d) sl[mpl[i]] = (vl[i] == -1 ? -1
99            : mpl[vl[i]]);

```

```

90     REP(i, n - d) sr[mpr[i]] = (vr[i] == -1
91         ? -1 : mpr[vr[i]]);
92     subunit_monge_mult(sl, sr, ret);
93 }
94 vi solve() {
95     solve(invp, res_monge);
96     vi fenw(n + 1);
97     IREP(i, n) {
98         if(res_monge[i] != -1) {
99             for(int p = res_monge[i] + 1; p <= n
100                 ; p += p & -p) fenw[p]++;
101         }
102         for(auto& z : qry[i]){
103             auto [id, c] = z;
104             for(int p = id; p; p -= p & -p) ans[
105                 c] -= fenw[p];
106         }
107     };
108     return ans;
109 }

```

2.11 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         ;
41     if(p.second) p.second->p = NULL;
42     v->r = p.first;
43     pull(v); return {v, p.second};
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 }

```

2.12 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), y1(n), y2(n);
34     for(int i = 0; i < n; i++) {
35         cin >> x1[i] >> x2[i] >> y1[i] >> y2[i];
36         // L R D U
37         vx.pb(x1[i]), vx.pb(x2[i]);
38         vy.pb(y1[i]), vy.pb(y2[i]);
39     }
40     vx = sort_unique(vx);
41     vy = sort_unique(vy);
42     vector<q> a(2 * n);

```

```

38 REP(i, n) {
39     x1[i] = lower_bound(ALL(vx), x1[i]) - vx
40         .begin();
41     x2[i] = lower_bound(ALL(vx), x2[i]) - vx
42         .begin();
43     y1[i] = lower_bound(ALL(vy), y1[i]) - vy
44         .begin();
45     y2[i] = lower_bound(ALL(vy), y2[i]) - vy
46         .begin();
47     a[2 * i] = {y1[i], x1[i], x2[i], +1};
48     a[2 * i + 1] = {y2[i], x1[i], x2[i],
49         -1};
50 }
51 sort(ALL(a), [](q a, q b) { return a.piv <
52     b.piv; });
53 tree seg(n);
54 ll ans = 0;
55 REP(i, 2 * n) {
56     int j = i;
57     while(j < 2 * n && a[j].piv == a[i].piv)
58         seg.add(a[j].s, a[j].e, a[j].x, 1, 0,
59             vx.size());
60     j++;
61     if(a[i].piv + 1 != SZ(vy)) ans += 1LL *
62         seg.q() * (vy[a[i].piv + 1] - vy[a[i]
63             ].piv));
64     i = j - 1;
65 }
66 cout << ans << "\n";
67 }

```

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     bool dfs(int u) {
33         for(auto v : g[u]) {
34             if(rhs[v] == -1) {
35                 rhs[lhs[u] = v] = u;
36                 return true;
37             }
38         }
39         return false;
40     }
41 }
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         }
23     }

```



```

20     if(in[i] < 0) d.add_edge(i, n + 1, -in
21         [i]);
22     }
23     d.add_edge(t, s, numeric_limits<T>::max
24         ());
25     if(d.flow(n, n + 1) < sum) return -1;
26     return d.flow(s, t);
27 }
28 };

```

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;
12     vector<Edge> edges;
13     vector<vi> g;
14     vi cur, h; // h : Level graph
15     Dinic(int _n) : n(_n), g(_n) {}
16     void add_edge(int u, int v, T c) {
17         g[u].pb(sz(edges));
18         edges.eb(u, v, c);
19         g[v].pb(sz(edges));
20         edges.eb(v, u, 0);
21     }
22     bool bfs(int s, int t) {
23         h.assign(n, -1);
24         queue<int> q;
25         h[s] = 0;
26         q.push(s);
27         while(!q.empty()) {
28             int u = q.front(); q.pop();
29             for(int i : g[u]) {
30                 const auto& e = edges[i];
31                 int v = e.to;
32                 if(e.cap > 0 && h[v] == -1) {
33                     h[v] = h[u] + 1;
34                     if(v == t) return true;
35                     q.push(v);
36                 }
37             }
38         }
39         return false;
40     }
41     T dfs(int u, int t, T f) {
42         if(u == t) return f;
43         T r = f;
44         for(int& i = cur[u]; i < sz(g[u]); ++i)
45             {
46                 int j = g[u][i];
47                 const auto& e = edges[j];
48                 int v = e.to;
49                 T c = e.cap;
50                 if(c > 0 && h[v] == h[u] + 1) {
51                     T a = dfs(v, t, min(r, c));
52                     r -= a;
53                     edges[j].cap -= a;
54                     edges[j ^ 1].cap += a;
55                     if((r -= a) == 0) return f;
56                 }
57             }
58         return r;
59     }
60     T flow(int s, int t, T f = INF) {
61         T ans = 0;
62         while(f > 0 && bfs(s, t)) {
63             cur.assign(n, 0);
64             T cur = dfs(s, t, f);
65             ans += cur;
66             f -= cur;
67         }
68         return ans;
69     }
70 };

```

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     GeneralMaxMatch(int n) : n(n), g(n, -1),
6         mate(n, -1) {}
7     bool dfs(int u) {
8         if(vis[u]) return false;
9         vis[u] = true;
10        for(int ei = g[u]; ei != -1; ) {
11            auto [x, y] = es[ei]; ei = y;

```

```

11        if(mate[x] == -1) {
12            mate[mate[u] = x] = u;
13            return true;
14        }
15    }
16    for(int ei = g[u]; ei != -1; ) {
17        auto [x, y] = es[ei]; ei = y;
18        int nu = mate[x];
19        mate[mate[u] = x] = u;
20        mate[nu] = -1;
21        if(dfs(nu)) return true;
22        mate[mate[nu] = x] = nu;
23        mate[u] = -1;
24    }
25    return false;
26 }
27 void add_edge(int a, int b) {
28     auto f = [&](int a, int b) {
29         es.eb(b, g[a]);
30         g[a] = sz(es) - 1;
31     };
32     f(a, b); f(b, a);
33 }
34 int solve() {
35     vi o(n);
36     iota(all(o), 0);
37     int ans = 0;
38     REP(it, 100) {
39         shuffle(all(o), rng);
40         vis.assign(n, false);
41         for(auto i : o) if(mate[i] == -1) ans
42             += dfs(i);
43     }
44     return ans;
45 }

```

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; }

```

```

19 void set_slack(int x) {
20     slack[x] = 0;
21     REP(u, n) if(g[u + 1][x].w > 0 && st[u +
22         1] != x && S[st[u + 1]] == 0)
23         update_slack(u + 1, x);
24 }
25 void q_push(int x) {
26     if(x <= n) q.push(x);
27     else REP(i, SZ(flo[x])) q_push(flo[x][i
28         ]);
29 }
30 void set_st(int x, int b) {
31     st[x] = b;
32     if(x > n) REP(i, SZ(flo[x])) set_st(flo[
33         x][i], b);
34 }
35 int get_pr(int b, int xr) {
36     int pr = find(ALL(flo[b]), xr) - flo[b].
37         begin();
38     if(pr % 2 == 1) {
39         reverse(1 + ALL(flo[b]));
40         return SZ(flo[b]) - pr;
41     }
42     return pr;
43 }
44 void set_match(int u, int v) {
45     match[u] = g[u][v].v;
46     if(u <= n) return;
47     edge e = g[u][v];
48     int xr = flo_from[u][e.u], pr = get_pr(u
49         , xr);
50     for(int i = 0; i < pr; ++i) set_match(
51         flo[u][i], flo[u][i ^ 1]);
52     set_match(xr, v);
53     rotate(flo[u].begin(), flo[u].begin() +
54         pr, flo[u].end());
55 }
56 void augment(int u, int v) {
57     while(true) {
58         int xnv = st[match[u]];
59         set_match(u, v);
60         if(!xnv) return;
61         set_match(xnv, st[pa[xnv]]);
62         u = st[pa[xnv]], v = xnv;
63     }
64 }
65 int get_lca(int u, int v) {
66     static int t = 0;
67     for(++t; u || v; swap(u, v)) {
68         if(u == 0) continue;
69         if(vis[u] == t) return u;
70         vis[u] = t;
71         if(u = st[match[u]]) u = st[pa[u]];
72     }
73     return 0;
74 }
75 void add_blossom(int u, int lca, int v) {
76     int b = n + 1;
77     while(b <= n_x && st[b]) ++b;
78     if(b > n_x) n_x++;
79     lab[b] = S[b] = 0;
80     match[b] = match[lca];
81     flo[b].clear(); flo[b].pb(lca);
82     for(int x = u, y; x != lca; x = st[pa[y
83         ]]) flo[b].pb(x), flo[b].pb(y = st[
84         match[x]]), q_push(y);
85     reverse(1 + ALL(flo[b]));
86     for(int x = v, y; x != lca; x = st[pa[y
87         ]]) flo[b].pb(x), flo[b].pb(y = st[
88         match[x]]), q_push(y);
89     set_st(b, b);
90     REP(x, n_x) g[b][x + 1].w = g[x + 1][b].
91         w = 0;
92     REP(x, n) flo_from[b][x + 1] = 0;
93     REP(i, SZ(flo[b])) {
94         int xs = flo[b][i];
95         REP(x, n_x) if(g[b][x + 1].w == 0 ||
96             e_delta(g[xs][x + 1]) < e_delta(g[
97                 b][x + 1])) g[b][x + 1] = g[xs][x
98                 + 1], g[x + 1][b] = g[x + 1][xs];
99         REP(x, n) if(flo_from[xs][x + 1])
100             flo_from[b][x + 1] = xs;
101     }
102     set_slack(b);
103 }
104 void expand_blossom(int b) {
105     REP(i, SZ(flo[b])) set_st(flo[b][i], flo
106         [b][i]);
107     int xr = flo_from[b][g[b][pa[b]].u], pr
108         = get_pr(b, xr);
109     for(int i = 0; i < pr; i += 2) {
110         int xs = flo[b][i], xns = flo[b][i +
111             1];
112         pa[xs] = g[xns][xs].u;
113         S[xs] = 1, S[xns] = 0;
114         slack[xs] = 0, set_slack(xns);
115         q_push(xns);
116     }
117     S[xr] = 1, pa[xr] = pa[b];
118     for(size_t i = pr + 1; i < SZ(flo[b]);
119         ++i) {
120         int xs = flo[b][i];
121         S[xs] = -1, set_slack(xs);
122     }
123     st[b] = 0;
124 }
125 bool on_found_edge(const edge &e) {
126     int u = st[e.u], v = st[e.v];
127     if(S[v] == -1) {
128         pa[v] = e.u, S[v] = 1;
129         int nu = st[match[v]];
130         slack[v] = slack[nu] = 0;
131         S[nu] = 0, q_push(nu);
132     } else if(S[v] == 0) {
133         int lca = get_lca(u, v);
134         if(!lca) return augment(u, v), augment(
135             v, u), true;
136         else add_blossom(u, lca, v);
137     }
138     return false;
139 }
140 bool matching() {
141     memset(S + 1, -1, sizeof(int) * n_x);
142     memset(slack + 1, 0, sizeof(int) * n_x);
143     q = queue<int>();
144     REP(x, n_x) if(st[x + 1] == x + 1 && !
145         match[x + 1]) pa[x + 1] = 0, S[x +
146             1] = 0, q_push(x + 1);
147     if(q.empty()) return false;
148     while(true) {
149         while(!q.empty()) {
150             int u = q.front(); q.pop();
151             if(S[st[u]] == 1) continue;
152             for(int v = 1; v <= n; ++v)
153                 if(g[u][v].w > 0 && st[u] != st[v
154                     ]) {
155                     if(e_delta(g[u][v]) == 0) {
156                         if(on_found_edge(g[u][v]))
157                             return true;
158                     } else update_slack(u, st[v]);
159                 }
160             int d = inf;
161             for(int b = n + 1; b <= n_x; ++b) if(
162                 st[b] == b && S[b] == 1) d = min(d
163                     , lab[b] / 2);
164             for(int x = 1; x <= n_x; ++x) {
165                 if(st[x] == x && slack[x]) {
166                     if(S[x] == -1) d = min(d, e_delta(
167                         g[slack[x]][x]));
168                     else if(S[x] == 0) d = min(d,
169                         e_delta(g[slack[x]][x]) / 2);
170                 }
171             }
172             REP(u, n) {
173                 if(S[st[u + 1]] == 0) {
174                     if(lab[u + 1] <= d) return 0;
175                     lab[u + 1] -= d;
176                 } else if(S[st[u + 1]] == 1) lab[u +
177                     1] += d;
178             }
179             q = queue<int>();
180             for(int x = 1; x <= n_x; ++x)
181                 if(st[x] == x && slack[x] && st[
182                     slack[x]] != x && e_delta(g[
183                         slack[x]][x]) == 0)
184                     if(on_found_edge(g[slack[x]][x]))
185                         return true;
186             for(int b = n + 1; b <= n_x; ++b)
187                 if(st[b] == b && S[b] == 1 && lab[b]
188                     == 0) expand_blossom(b);
189             return false;
190         }
191     }
192     pair<ll, int> solve() {
193         memset(match + 1, 0, sizeof(int) * n);
194         n_x = n;
195         int n_matches = 0;
196         ll tot_weight = 0;
197         for(int u = 0; u <= n; ++u) st[u] = u,
198             flo[u].clear();
199         int w_max = 0;
200         for(int u = 1; u <= n; ++u)
201             for(int v = 1; v <= n; ++v) {
202                 flo_from[u][v] = (u == v ? u : 0);
203                 w_max = max(w_max, g[u][v].w);
204             }
205         for(int u = 1; u <= n; ++u) lab[u] =
206             w_max;
207         while(matching()) ++n_matches;
208         for(int u = 1; u <= n; ++u)
209             if(match[u] && match[u] < u)
210                 tot_weight += g[u][match[u]].w;
211         return make_pair(tot_weight, n_matches);
212     }
213 }
214 void add_edge(int u, int v, int w) { g[u][
215     v].w = g[v][u].w = w; }
216 void init(int _n) : n(_n) {
217     REP(u, n) REP(v, n) g[u + 1][v + 1] =
218         edge(u + 1, v + 1, 0);
219 }
220 }
221 };

```

3.7 KM

```

1 template<class T>
2 struct KM {
3     static constexpr T INF = numeric_limits<T>
4         >::max();
5     int n, ql, qr;
6     vector<vector<T>>> w;
7     vector<T> hl, hr, slk;
8     vi fl, fr, pre, qu;
9     vector<bool> vl, vr;
10    KM(int n) : n(n), w(n, vector<T>(n, -INF))
11        , hl(n), hr(n), slk(n), fl(n), fr(n),
12        pre(n), qu(n), vl(n), vr(n) {}
13    void add_edge(int u, int v, int x) { w[u][
14        v] = x; } // 最小值要加負號
15    bool check(int x) {
16        vl[x] = 1;
17        if(fl[x] != -1) return vr[qu[qr++] = fl[
18            x]] = 1;
19        while(x != -1) swap(x, fr[fl[x] = pre[x
20            ]]);
21        return 0;
22    }
23    void bfs(int s) {
24        fill(all(slk), INF);
25        fill(all(vl), 0);
26        fill(all(vr), 0);
27        ql = qr = 0, qu[qr++] = s, vr[s] = 1;
28        while(true) {
29            T d;
30            while(ql < qr) {
31                for(int x = 0, y = qu[ql++]; x < n;
32                    ++x) {
33                    if(!vl[x] && slk[x] >= (d = hl[x]
34                        + hr[y] - w[x][y])) {
35                        pre[x] = y;
36                        if(d) slk[x] = d;
37                        else if(!check(x)) return;
38                    }
39                }
40            }
41            d = INF;
42            REP(x, n) if(!vl[x] && d > slk[x]) d =
43                slk[x];
44            REP(x, n) {
45                if(vl[x]) hl[x] += d;
46                else slk[x] -= d;
47                if(vr[x]) hr[x] -= d;
48            }
49            REP(x, n) if(!vl[x] && !slk[x] && !
50                check(x)) return;
51        }
52    }

```

```

41 }
42 }
43 T solve() {
44     fill(all(fl), -1);
45     fill(all(fr), -1);
46     fill(all(hr), 0);
47     REP(i, n) hl[i] = *max_element(all(w[i])
48 );
49     REP(i, n) bfs(i);
50     T ans = 0;
51     REP(i, n) ans += w[i][fl[i]]; // i 跟 fl
52     [i] 配對
53 }
54 };

```

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

```

```

39 for(auto& q : nrem) q.deg = (bs & g[q.
40     idx]).count();
41     now.emplace_back(p.idx);
42     dfs(nrem);
43     now.pop_back();
44 }
45 vector<int> solve() {
46     vector<P> remark;
47     REP(i, n) remark.emplace_back(i, -1, SZ(
48         g[i]));
49     dfs(remark);
50     return clique;
51 }
52 };

```

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

```

```

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 (
3     // Perfect Match) 0-base
4     static const int MXN = 105;
5     int n, edge[MXN][MXN];
6     int match[MXN], dis[MXN], onstk[MXN];
7     vector<int> stk;
8     void init(int _n) {
9         n = _n;
10        for(int i=0; i<n; i++)
11            for(int j=0; j<n; j++)
12                edge[i][j] = 0;
13    }
14    void add_edge(int u, int v, int w) { edge[
15        u][v] = edge[v][u] = w; }
16    bool SPFA(int u) {
17        if(onstk[u]) return true;
18        stk.push_back(u);
19        onstk[u] = 1;
20        for(int v=0; v<n; v++) {
21            if(u != v && match[u] != v && !onstk[v]
22                && edge[u][v] < edge[u][match[u]]) {
23                int m = match[v];
24                if(dis[m] > dis[u] - edge[v][m] +
25                    edge[u][v]) {
26                    dis[m] = dis[u] - edge[v][m] +
27                        edge[u][v];
28                    match[v] = u;
29                    onstk[v] = 1;
30                    SPFA(v);
31                }
32            }
33        }
34        onstk[u] = 0;
35        return false;
36    }
37 }

```

```

22 dis[m] = dis[u] - edge[v][m] +
23     edge[u][v];
24     onstk[v] = 1;
25     stk.push_back(v);
26     if(SPFA(m)) return true;
27     stk.pop_back();
28     onstk[v] = 0;
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }

```

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     ; });
5 ll SQ(ll x) { return x * x; }
6 ll solve(int l, int r) {
7     if(1 + 1 == r) return INF;
8     int m = (l + r) / 2;
9     ll midx = a[m].x;
10    ll d = min(solve(l, m), solve(m, r));
11    inplace_merge(a.begin() + l, a.begin() + m
12        , a.begin() + r, [](P a, P b) {
13            return a.y < b.y;
14        });
15 }

```



```

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

```

4.4 point

```

1 using P = pair<ll, ll>;
2 P operator+(P a, P b) { return P{a.X + b.X,
    a.Y + b.Y}; }
3 P operator-(P a, P b) { return P{a.X - b.X,
    a.Y - b.Y}; }
4 P operator*(P a, ll b) { return P{a.X * b, a
    .Y * b}; }
5 P operator/(P a, ll b) { return P{a.X / b, a
    .Y / b}; }
6 ll dot(P a, P b) { return a.X * b.X + a.Y *
    b.Y; }
7 ll cross(P a, P b) { return a.X * b.Y - a.Y
    * b.X; }
8 ll abs2(P a) { return dot(a, a); }
9 double abs(P a) { return sqrt(abs2(a)); }
10 int sign(ll x) { return x < 0 ? -1 : (x == 0
    ? 0 : 1); }
11 int ori(P a, P b, P c) { return sign(cross(b
    - a, c - a)); }
12 bool collinear(P a, P b, P c) { return sign(
    cross(a - c, b - c)) == 0; }
13 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) ||
            btw(c, d, a) || btw(c, d, b);
24     }
25     return a123 * a124 <= 0 && a341 * a342 <=
        0;
26 }

```

```

27 P intersect(P a, P b, P c, P d) {
28     int a123 = cross(b - a, c - a);
29     int a124 = cross(b - a, d - a);
30     return (d * a123 - c * a124) / (a123 -
        a124);
31 }
32 }
33 struct line { P A, B; };
34 P vec(line L) { return L.B - L.A; }
35 P projection(P p, line L) { return L.A + vec
    (L) / abs(vec(L)) * dot(p - L.A, vec(L))
        / 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)
        == 0 && sign(k.X) < 0))
3     int A = ng(a), B = ng(b);
4     if(A != B) return A < B;
5     if(sign(cross(a, b)) == 0) return abs2(a)
        < abs2(b);
6     return sign(cross(a, b)) > 0;

```

```

7 }

```

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         g.add_edge(2 * u + !x, 2 * v + y);
7         g.add_edge(2 * v + !y, 2 * u + x);
8     }
9     bool solve() {
10        ans.resize(n);
11        auto id = g.solve();
12        REP(i, n) {
13            if(id[2 * i] == id[2 * i + 1]) return
                false;
14            ans[i] = (id[2 * i] < id[2 * i + 1]);
15        }
16        return true;
17    }
18 };

```

5.2 centroid-tree

```

1 pair<int, vector<vi>> centroid_tree(const
    vector<vi>& g) {
2     int n = sz(g);
3     vi siz(n);
4     vector<bool> vis(n);
5     auto dfs_sz = [&](auto f, int u, int p) ->
        void {
6         siz[u] = 1;
7         for(auto v : g[u]) {
8             if(v == p || vis[v]) continue;
9             f(f, v, u);
10            siz[u] += siz[v];
11        }
12    };
13    auto find_cd = [&](auto f, int u, int p,
        int all) -> int {
14        for(auto v : g[u]) {
15            if(v == p || vis[v]) continue;
16            if(siz[v] * 2 > all) return f(f, v, u,
                all);
17        }
18        return u;
19    };
20    vector<vi> h(n);
21    auto build = [&](auto f, int u) -> int {
22        dfs_sz(dfs_sz, u, -1);
23        int cd = find_cd(find_cd, u, -1, siz[u]);
24        vis[cd] = true;

```

```

25     for(auto v : g[cd]) {
26         if(vis[v]) continue;
27         int child = f(f, v);
28         h[cd].pb(child);
29     }
30     return cd;
31 };
32 int root = build(build, 0);
33 return {root, h};
34 }

```

5.3 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(
        _n), depth(_n), top(_n), fi(_n), id(_n)
        {
7         tour.reserve(n);
8     }
9     void add_edge(int u, int v) {
10        g[u].push_back(v);
11        g[v].push_back(u);
12    }
13    void build(int root = 0) {
14        par[root] = -1;
15        top[root] = root;
16        vector<pii> euler_tour;
17        euler_tour.reserve(2 * n - 1);
18        dfs_sz(root);
19        dfs_link(euler_tour, root);
20        st = sparse_table<pii, min>(euler_tour);
21    }
22    int get_lca(int u, int v) {
23        int L = fi[u], R = fi[v];
24        if(L > R) swap(L, R);
25        return st.prod(L, R).second;
26    }
27    bool is_anc(int u, int v) {
28        return id[u] <= id[v] && id[v] < id[u] +
            siz[u];
29    }
30    bool on_path(int a, int b, int x) {
31        return (is_ancestor(x, a) || is_ancestor
            (x, b)) && is_ancestor(get_lca(a, b),
                x);
32    }
33    int get_dist(int u, int v) {
34        return depth[u] + depth[v] - 2 * depth[
            get_lca(u, v)];
35    }
36    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        return tour[id[u] + d - depth[u]];
41    }
42    int kth_node_on_path(int a, int b, int k)
        {

```

```

43 int z = get_lca(a, b);
44 int fi = depth[a] - depth[z];
45 int se = depth[b] - depth[z];
46 if(k < 0 || k > fi + se) return -1;
47 if(k < fi) return kth_anc(a, k);
48 return kth_anc(b, fi + se - k);
49 }
50 vector<pii> get_path(int u, int v, bool
    include_lca = true) {
51     if(u == v && !include_lca) return {};
52     vector<pii> seg;
53     while(top[u] != top[v]) {
54         if(depth[top[u]] > depth[top[v]]) swap
            (u, v);
55         seg.eb(id[top[v]], id[v]);
56         v = par[top[v]];
57     }
58     if(depth[u] > depth[v]) swap(u, v); // u
        is lca
59     if(u != v || include_lca) seg.eb(id[u] +
        !include_lca, id[v]);
60     return seg;
61 }
62 void dfs_sz(int u) {
63     if(par[u] != -1) g[u].erase(find(all(g[u]
        )), par[u]));
64     siz[u] = 1;
65     for(auto& v : g[u]) {
66         par[v] = u;
67         depth[v] = depth[u] + 1;
68         dfs_sz(v);
69         siz[u] += siz[v];
70         if(siz[v] > siz[g[u][0]]) swap(v, g[u]
            [0]);
71     }
72 }
73 void dfs_link(vector<pii>& euler_tour, int
    u) {
74     fi[u] = sz(euler_tour);
75     id[u] = sz(tour);
76     euler_tour.eb(depth[u], u);
77     tour.pb(u);
78     for(auto v : g[u]) {
79         top[v] = (v == g[u][0] ? top[u] : v);
80         dfs_link(euler_tour, v);
81         euler_tour.eb(depth[u], u);
82     }
83 }
84 };

```

5.4 lowlink

```

1 struct lowlink {
2     int n, cnt = 0, tecc_cnt = 0, tvcc_cnt =
        0;
3     vector<vector<pii>> g;
4     vector<pii> edges;
5     vi roots, id, low, tecc_id, tvcc_id;
6     vector<bool> is_bridge, is_cut,
        is_tree_edge;
7     lowlink(int _n) : n(_n), g(_n), is_cut(_n,
        false), id(_n, -1), low(_n, -1) {}
8     void add_edge(int u, int v) {

```

```

9         g[u].eb(v, sz(edges));
10        g[v].eb(u, sz(edges));
11        edges.eb(u, v);
12        is_bridge.pb(false);
13        is_tree_edge.pb(false);
14        tvcc_id.pb(-1);
15    }
16    void dfs(int u, int peid = -1) {
17        static vi stk;
18        static int rid;
19        if(peid < 0) rid = cnt;
20        if(peid == -1) roots.pb(u);
21        id[u] = low[u] = cnt++;
22        for(auto [v, eid] : g[u]) {
23            if(eid == peid) continue;
24            if(id[v] < id[u]) stk.pb(eid);
25            if(id[v] >= 0) {
26                low[u] = min(low[u], id[v]);
27            } else {
28                is_tree_edge[eid] = true;
29                dfs(v, eid);
30                low[u] = min(low[u], low[v]);
31                if((id[u] == rid && id[v] != rid +
                    1) || (id[u] != rid && low[v] >=
                        id[u])) {
32                    is_cut[u] = true;
33                }
34                if(low[v] >= id[u]) {
35                    while(true) {
36                        int e = stk.back();
37                        stk.pop_back();
38                        tvcc_id[e] = tvcc_cnt;
39                        if(e == eid) break;
40                    }
41                    tvcc_cnt++;
42                }
43            }
44        }
45    }
46    void build() {
47        REP(i, n) if(id[i] < 0) dfs(i);
48        REP(i, sz(edges)) {
49            auto [u, v] = edges[i];
50            if(id[u] > id[v]) swap(u, v);
51            is_bridge[i] = (id[u] < low[v]);
52        }
53    }
54    vector<vi> two_ecc() { // 邊雙
55        tecc_cnt = 0;
56        tecc_id.assign(n, -1);
57        vi stk;
58        REP(i, n) {
59            if(tecc_id[i] != -1) continue;
60            tecc_id[i] = tecc_cnt;
61            stk.pb(i);
62            while(sz(stk)) {
63                int u = stk.back(); stk.pop_back();
64                for(auto [v, eid] : g[u]) {
65                    if(tecc_id[v] >= 0 || is_bridge[
                        eid]) {
66                        continue;
67                    }
68                    tecc_id[v] = tecc_cnt;
69                    stk.pb(v);
70                }

```

```

71        }
72        tecc_cnt++;
73    }
74    vector<vi> comp(tecc_cnt);
75    REP(i, n) comp[tecc_id[i]].pb(i);
76    return comp;
77 }
78 vector<vi> bcc_vertices() { // 點雙
79     vector<vi> comp(tvcc_cnt);
80     REP(i, sz(edges)) {
81         comp[tecc_id[i]].pb(edges[i].first);
82         comp[tecc_id[i]].pb(edges[i].second);
83     }
84     for(auto& v : comp) {
85         sort(all(v));
86         v.erase(unique(all(v)), v.end());
87     }
88     REP(i, n) if(g[i].empty()) comp.pb({i});
89     return comp;
90 }
91 vector<vi> bcc_edges() {
92     vector<vi> ret(tvcc_cnt);
93     REP(i, sz(edges)) ret[tecc_id[i]].pb(i);
94     return ret;
95 }
96 };

```

5.5 SCC

```

1 struct SCC {
2     int n;
3     vector<vi> g, h;
4     SCC(int _n) : n(_n), g(_n), h(_n) {}
5     void add_edge(int u, int v) {
6         g[u].pb(v);
7         h[v].pb(u);
8     }
9     vi solve() { // 回傳縮點的編號
10        vi id(n), top;
11        top.reserve(n);
12        #define GO if(id[v] == 0) dfs1(v);
13        function<void(int)> dfs1 = [&](int u) {
14            id[u] = 1;
15            for(auto v : g[u]) GO;
16            top.pb(u);
17        };
18        REP(v, n) GO;
19        fill(all(id), -1);
20        function<void(int, int)> dfs2 = [&](int
            u, int x) {
21            id[u] = x;
22            for(auto v : h[u]) {
23                if(id[v] == -1) {
24                    dfs2(v, x);
25                }
26            }
27        };
28        for(int i = n - 1, cnt = 0; i >= 0; --i)
29            {
30                int u = top[i];
31                if(id[u] == -1) {
32                    dfs2(u, cnt);

```

```

32        cnt += 1;
33    }
34    }
35    return id;
36 }
37 };

```

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
            ? +1 : -1);
7         if(op < k) r = m;
8         else l = m + 1;
9     }
10    return f(l).first + l * k * (MAX ? +1 :
        -1);
11 }

```

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,
    998244352]) (mod 998244353)
4 // - [] -> (0, [1])
5 // - [0, 0, 0] -> (0, [1])
6 // - [-2] -> (1, [1, 2])
7 template<class T>
8 pair<int, vector<T>> BM(const vector<T>& S)
    {
9     using poly = vector<T>;
10    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
        d, T b, int m) -> poly {
15        C.resize(max(SZ(C), SZ(B) + m));
16        T a = d / b;
17        REP(i, SZ(B)) C[i + m] -= a * B[i];
18        return C;
19    };
20    REP(n, N) {
21        T d = S[n];
22        REP(i, L) d += C_rev[i + 1] * S[n - 1 -
            i];
23        if(d == 0) m++;
24        else if (2 * L <= n) {
25            poly Q = C_rev;
26            C_rev = adjust(C_rev, B, d, b, m);
27            L = n + 1 - L, B = Q, b = d, m = 1;

```

```

28 } else C_rev = adjust(C_rev, B, d, b, m
    ++);
29 }
30 return {L, C_rev};
31 }
32 // Calculate  $x^N \bmod f(x)$ 
33 // Complexity:  $O(K^2 \log N)$  ( $K$ : deg. of  $f$ )
34 // (4, [1, -1, -1]) -> [2, 3]
35 // (  $x^4 = (x^2 + x + 2)(x^2 - x - 1) + 3x + 2$  )
36 template<class T>
37 vector<T> monomial_mod_polynomial(long long
    N, const vector<T> &f_rev) {
38     assert(!f_rev.empty() && f_rev[0] == 1);
39     int K = SZ(f_rev) - 1;
40     if(!K) return {};
41     int D = 64 - __builtin_clzll(N);
42     vector<T> ret(K, 0);
43     ret[0] = 1;
44     auto self_conv = [](vector<T> x) -> vector
        <T> {
45         int d = SZ(x);
46         vector<T> ret(d * 2 - 1);
47         REP(i, d) {
48             ret[i * 2] += x[i] * x[i];
49             REP(j, i) ret[i + j] += x[i] * x[j] *
                2;
50         }
51         return ret;
52     };
53     for(int d = D; d--;) {
54         ret = self_conv(ret);
55         for(int i = 2 * K - 2; i >= K; i--) {
56             REP(j, k) ret[i - j - 1] -= ret[i] *
                f_rev[j + 1];
57         }
58         ret.resize(K);
59         if(N >> d & 1) {
60             vector<T> c(K);
61             c[0] = -ret[K - 1] * f_rev[K];
62             for(int i = 1; i < K; i++) c[i] = ret[
                i - 1] - ret[K - 1] * f_rev[K - i
                ];
63             ret = c;
64         }
65     }
66     return ret;
67 }
68 }
69 // Guess k-th element of the sequence,
    assuming linear recurrence
70 template<class T>
71 T guess_kth_term(const vector<T> &a, long
    long k) {
72     assert(k >= 0);
73     if(k < 1LL * SZ(a)) return a[k];
74     auto f = BM<T>(a).second;
75     auto g = monomial_mod_polynomial<T>(k, f);
76     T ret = 0;
77     REP(i, SZ(g)) ret += g[i] * a[i];
78     return ret;
79 }
80 }

```

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     r0 = (r0 % m0 + m0) % m0;
4     r1 = (r1 % m1 + m1) % m1;
5     if(m0 < m1) swap(r0, r1), swap(m0, m1);
6     if(m0 % m1 == 0) {
7         if(r0 % m1 != r1) return {0, 0};
8     }
9     ll g, im, qq;
10    g = ext_gcd(m0, m1, im, qq);
11    ll u1 = (m1 / g);
12    if((r1 - r0) % g) return {0, 0};
13    ll x = (r1 - r0) / g % u1 * im % u1;
14    r0 += x * m0;
15    m0 *= u1;
16    if(r0 < 0) r0 += m0;
17    return {r0, m0};
18 }

```

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
        - k];
4 }
5 // a_1 + a_2 + ... + a_n = k, a_i >= 0
6 mint stars_and_bars(int n, int k) { return
    binom(k + n - 1, n - 1); }
7 // number of ways from (0, 0) to (n, m)
8 mint paths(int n, int m) { return binom(n +
    m, n); }
9 mint catalan(int n) { return binom(2 * n, n)
    - 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)
        != b) A[1LL * e * b % m] = j++;
6     if(e == b) return j;
7     if(__gcd(m, e) == __gcd(m, b)) {
8         for(int i = 2; i < n + 2; ++i) {
9             e = 1LL * e * f % m;
10            if(A.find(e) != A.end()) return n * i
                - A[e];
11        }
12    }
13    return -1;
14 }

```

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 }
16 template<class T>
17 void and_transform(vector<T> &a, bool inv) {
18     fwht(a, [&](T& x, T& y) { x += y * (inv
        ? -1 : +1); });
19 }
20 template<class T>
21 void xor_transform(vector<T> &a, bool inv) {
22     fwht(a, [&](T& x, T& y) {
23         T z = x + y;
24         y = x - y;
25         x = z;
26     });
27     if(inv) {
28         T z = T(1) / T(SZ(a));
29         for(auto& x : a) x *= z;
30     }
31 }
32 template<class T>
33 vector<T> convolution(vector<T> a, vector<T>
    b) {
34     assert(SZ(a) == SZ(b));
35     transform(a, false), transform(b, false);
36     REP(i, SZ(a)) a[i] *= b[i];
37     transform(a, true);
38     return a;
39 }
40 template<class T>
41 vector<T> subset_convolution(const vector<T>
    &f, const vector<T> &g) {
42     assert(SZ(f) == SZ(g));
43     int n = SZ(f);
44     assert(ppc(n) == 1);
45     const int lg = __lg(n);
46     vector<vector<T>> fhat(lg + 1, vector<T>(n
        )), ghat(fhat);
47     REP(i, n) fhat[ppc(i)][i] = f[i], ghat[ppc
        (i)][i] = g[i];
48     REP(i, lg + 1) or_transform(fhat[i], false
        ), or_transform(ghat[i], false);
49     vector<vector<T>> h(lg + 1, vector<T>(n));
50     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)
9                 continue;
10            if(abs(a[j][i]) > abs(a[p][i])) p = j;
11        }
12        REP(j, n + 1) swap(a[i][j], a[p][j]);
13        if(abs(a[i][i]) <= EPS) continue;
14        REP(j, n) {
15            if(i == j) continue;
16            ld delta = a[j][i] / a[i][i];
17            FOR(k, i, n + 1) a[j][k] -= delta * a[i][k];
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;
23     if(n < (1LL << 30)) return is_prime(n, {2, 7, 61});
24     return is_prime(n, {2, 325, 9375, 28178, 450775, 9780504, 1795265022});
25 }

```

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)) <
8             v + n) r = k;
9         else l = k;
10    }
11    return r;

```

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 - 1LL * (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 vector<ll> get_divisors(ll n) {
24     if(n == 0) return {};
25     map<ll, int> mp;
26     PollardRho(mp, n);
27     vector<pair<ll, int>> v(all(mp));
28     vector<ll> res;
29     auto f = [&](auto f, int i, ll x) -> void {
30         {
31             if(i == sz(v)) {

```

```

32         res.pb(x);
33         return;
34     }
35     for(int j = v[i].second; ; j--) {
36         f(f, i + 1, x);
37         if(j == 0) break;
38         x *= v[i].first;
39     }
40 };
41 f(f, 0, 1);
42 sort(all(res));
43 return res;
44 }

```

6.17 Primes

```

1 /* 12721 13331 14341 75577 123457 222557
556679 999983 1097774749 1076767633
100102021 999997771 1001010013
1000512343 987654361 999991231 999888733
98789101 987777733 999991921 1010101333
1010102101 1000000000039
1000000000000037 2305843009213693951
4611686018427387847 9223372036854775783
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

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

• 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 $\sum_{i=0}^{n-1} m^{\gcd(i, n)}$. 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 $d_{ij} = -d_{ji}$. $\frac{rank(D)}{2}$ 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 $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k)$ 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.t. $\pi(j) > \pi(j+1)$, $k+1, j$: s.t. $\pi(j) \geq j$, k, j : 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:

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

6.22 歐幾里得類算法

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

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

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

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

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

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

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

6.23 生成函數

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

$$\begin{aligned} - A(rx) &\Rightarrow r^n a_n \\ - A(x) + B(x) &\Rightarrow a_n + b_n \\ - A(x)B(x) &\Rightarrow \sum_{i=0}^n a_i b_{n-i} \\ - A(x)^k &\Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k} \\ - xA(x)' &\Rightarrow n a_n \\ - \frac{A(x)}{1-x} &\Rightarrow \sum_{i=0}^n a_i \end{aligned}$$

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

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

- Special Generating Function

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

7 Misc

7.1 fast

```
1 #pragma GCC optimize("Ofast,no-stack-
   protector,unroll-loops,fast-math,inline")
2 inline char gc() {
3     static const size_t sz = 65536;
4     static char buf[sz];
5     static char *p = buf, *end = buf;
6     if(p == end) end = buf + fread(buf, 1, sz,
7         stdin), p = buf;
8     return *p++;
9 }
```

7.2 next-combination

```
1 a ≥ c ∨ b ≥ c
2 n < 0 ∨ a = 0
3 // Example: 1 -> 2, 4 -> 8, 12(1100) ->
4 // 17(10001)
5 ll next_combination(ll comb) {
6     ll x = comb & -comb, y = comb + x;
7     return ((comb & ~y) / x >> 1) | y;
8 }
```

7.3 PBDS

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

7.4 python

```
1 from decimal import Decimal, getcontext
2 getcontext().prec = 1000000000
3 getcontext().Emax = 999999999
4 a = pow(Decimal(2), 82589933) - 1
```

7.5 rng

```
1 inline ull rng() {
2     static ull Q = 48763;
3     Q ^= Q << 7;
4     Q ^= Q >> 9;
5     return Q & 0xFFFFFFFFFULL;
6 }
```

7.6 rotate90

```
1 vector<vector<T>> rotate90(const vector<
   vector<T>>& a) {
2     int n = sz(a), m = sz(a[0]);
3     vector<vector<T>> b(m, vector<T>(n));
4     REP(i, n) REP(j, m) b[j][i] = a[i][m - 1
5         - j];
6     return b;
7 }
```

7.7 timer

```
1 clock_t T1 = clock();
2 double getCurrentTime() { return (double) (
3     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             }
33             else {
34                 int v = node[u].go[j];
35                 node[v].fail = (u == 0 ? u : node[
36                     u].fail).go[j];
37             }
38         }
39     }
```

```
31     que.pb(v);
32 }
33 }
34 }
35 }
36 }
```

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    return k;
13 }
```

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 }
```

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

```
37 }
38 auto b = lms;
39 for(auto& v : b) v = val[v];
40 b = SAIS(b);
41 for(auto& v : b) v = lms[v];
42 ind(b);
43 }
44 return sa;
45 }
```

8.7 SAM

```
1 // cnt 要先用 bfs 往回推, 第一次出現的位置是
2 // state.first_pos - |S| + 1
3 struct Node { int go[26], len, link, cnt,
4     first_pos; };
5 Node SA[N]; int sz;
6 void sa_init() { SA[0].link = -1, SA[0].len
7     = 0, sz = 1; }
8 int sa_extend(int p, int c) {
9     int u = sz++;
10    SA[u].first_pos = SA[p].len = SA[p].len +
11        1;
12    SA[u].cnt = 1;
13    while(p != -1 && SA[p].go[c] == 0) {
14        SA[p].go[c] = u;
15        p = SA[p].link;
16    }
17    if(p == -1) {
18        SA[u].link = 0;
19        return u;
20    }
21    int q = SA[p].go[c];
22    if(SA[p].len + 1 == SA[q].len) {
23        SA[u].link = q;
24        return u;
25    }
26    int x = sz++;
27    SA[x] = SA[q];
28    SA[x].cnt = 0;
29    SA[x].len = SA[p].len + 1;
30    SA[q].link = SA[u].link = x;
31    while(p != -1 && SA[p].go[c] == q) {
32        SA[p].go[c] = x;
33        p = SA[p].link;
34    }
35    return u;
36 }
```

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    }
```

```
9     if(i == j) j++;
10 }
11 int ans = i < n ? i : j;
12 return s.substr(ans, n);
13 }
```

8.9 Z

```
1 // abacababa -> [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], j
7             + z[j] - i);
8         while(i + z[i] < n && a[i + z[i]] == a[z
9             [i]]) z[i]++;
10        if(i + z[i] > j + z[j]) j = i;
11    }
12    return z;
13 }
```

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

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

```
62
63 void runtime_error_5() {
64     // maybe illegal instruction
65     int a = 7122, b = 0;
66     cout << (a / b) << endl;
67 }
68 void runtime_error_6() {
69     // floating point exception
70     volatile int a = 7122, b = 0;
71     cout << (a / b) << endl;
72 }
73 void runtime_error_7() {
74     // call to abort.
75     assert(false);
76 }
77 // namespace system_test
78
79 #include <sys/resource.h>
80 void print_stack_limit() { // only work in
81     Linux
82     struct rlimit l;
83     getrlimit(RLIMIT_STACK, &l);
84     cout << "stack_size = " << l.rlim_cur << "
85         byte" << endl;
86 }
87 }
```