

ACM ICPC Team Reference - NTHU LinkCutTreap

Contents

1 Basic	2				
1.1 template	2				
1.2 vimrc	2				
2 Data-Structure	2				
2.1 wavelet-tree	2				
2.2 lazysegtree	2				
2.3 LiChao	2				
2.4 DLX	3				
2.5 sparse-table	3				
2.6 static-range-lis	3				
2.7 rollback-dsu	4				
2.8 static-range-inversion	4				
2.9 LCT	4				
2.10 segtree-beats	4				
2.11 union-of-rectangles	5				
2.12 CHT	6				
2.13 treap	6				
		2.14 VEB	6		
		2.15 rect-add-rect-sum	6		
		2.16 CDQ	7		
		2.17 segtree	7		
3 Flow-Matching	7				
3.1 KM	7				
3.2 bipartite-matching	7				
3.3 Dinic-LowerBound	8				
3.4 MCMF	8				
3.5 minimum-general-weighted-perfect-matching	8				
3.6 Flow 建模	8				
3.7 general-weighted-max-matching	9				
3.8 general-matching	10				
3.9 Dinic	10				
3.10 max-clique	10				
4 Geometry	10				
4.1 point-in-convex-hull	10				
4.2 half-plane	10				
4.3 point	11				
4.4 定理	11				
4.5 min-enclosing-circle	11				
4.6 convex-hull	11				
4.7 polar-angle-sort	11				
4.8 closest-pair	11				
5 Graph	11				
5.1 centroid-tree	11				
5.2 chromatic-number	12				
		5.3 count-bridge-online	12		
		5.4 2-SAT	12		
		5.5 lowlink	12		
		5.6 manhattan-mst	13		
		5.7 SCC	13		
		5.8 HLD	13		
		5.9 planar	13		
		5.10 BCC-tree	14		
		5.11 triangle-sum	14		
6 Math	14				
6.1 Min-of-Mod-of-Linear	14				
6.2 Gauss-Jordan	14				
6.3 Miller-Rabin	15				
6.4 Floor-Sum	15				
6.5 Discrete-Log	15				
6.6 Xor-Basis	15				
6.7 數字	15				
6.8 Primes	15				
6.9 Determinant	16				
6.10 extgcd	16				
6.11 NTT	16				
6.12 Poly	16				
6.13 Simplex	17				
6.14 Triangle	17				
6.15 Chinese-Remainder	17				
6.16 Pollard-Rho	17				
6.17 Mod-Sqrt	18				
6.18 Combination	18				
6.19 Mod-Inv	18				
6.20 FWHT	18				
6.21 Aliens	18				
		6.22 Berlekamp-Massey	18		
		6.23 定理	19		
		6.24 Int-Div	19		
		6.25 生成函數	19		
		6.26 GCD-Convolution	19		
		6.27 歐幾里得類算法	20		
		6.28 Linear-Sieve	20		
		6.29 估計值	20		
7 Misc	20				
7.1 PBDS	20				
7.2 python	20				
7.3 timer	20				
7.4 next-combination	20				
7.5 rng	20				
7.6 gc	20				
7.7 rotate90	20				
8 String	20				
8.1 smallest-rotation	20				
8.2 AC	20				
8.3 Z	20				
8.4 rolling-hash	21				
8.5 hash61	21				
8.6 LCP	21				
8.7 SAIS	21				
8.8 KMP	21				
8.9 wildcard-pattern-matching	21				
8.10 SAM	21				
8.11 manacher	21				

1 Basic

1.1 template

```
1 #pragma GCC optimize("Ofast,no-stack-
  protector,unroll-loops,fast-math,inline"
  )
2 #define FOR(i, begin, end) for(int i = (
  begin), i##_end_ = (end); i < i##_end_;
  i++)
3 #define IFOR(i, begin, end) for(int i = (end
  ) - 1, i##_begin_ = (begin); i >= i##_
  _begin_; i--)
4 #define REP(i, n) FOR(i, 0, n)
5 #define IREP(i, n) IFOR(i, 0, n)
```

1.2 vimrc

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

2 Data-Structure

2.1 wavelet-tree

```
1 template<class T>
2 struct wavelet_tree {
3     int n, log;
4     vector<T> vals;
5     vi sums;
6     vector<ull> bits;
7     void set_bit(int i, ull v) { bits[i >> 6]
8         |= (v << (i & 63)); }
9     int get_sum(int i) const { return sums[i
10         >> 6] + __builtin_popcountll(bits[i >>
11         6] & ((1ULL << (i & 63)) - 1)); }
12     wavelet_tree(const vector<T>& _v) : n(SZ(
13         _v)) {
14         vals = sort_unique(_v);
15         log = __lg(2 * vals.size() - 1);
16         bits.resize((log * n + 64) >> 6, 0ULL);
17         sums.resize(SZ(bits), 0);
18         vi v(SZ(_v), cnt(SZ(vals) + 1);
19         REP(i, SZ(v)) {
20             v[i] = lower_bound(ALL(vals), _v[i]) -
21                 vals.begin();
22             cnt[v[i] + 1] += 1;
23         }
24         partial_sum(ALL(cnt) - 1, cnt.begin());
25         REP(j, log) {
26             for(int i : v) {
```

```
22         int tmp = i >> (log - 1 - j);
23         int pos = (tmp >> 1) << (log - j);
24         set_bit(j * n + cnt[pos], tmp & 1);
25         cnt[pos]++;
26     }
27     for(int i : v) cnt[(i >> (log - j)) <<
28         (log - j)]--;
29     FOR(i, 1, SZ(sums)) sums[i] = sums[i -
30         1] + __builtin_popcountll(bits[i -
31         1]);
32     }
33     T get_kth(int a, int b, int k) {
34         for(int j = 0, ia = 0, ib = n, res = 0;;
35             j++) {
36             if(j == log) return vals[res];
37             int cnt_ia = get_sum(n * j + ia);
38             int cnt_a = get_sum(n * j + a);
39             int cnt_b = get_sum(n * j + b);
40             int cnt_ib = get_sum(n * j + ib);
41             int ab_zeros = (b - a) - (cnt_b -
42                 cnt_a);
43             if(ab_zeros > k) {
44                 res <<= 1;
45                 ib -= cnt_ib - cnt_ia;
46                 a -= cnt_a - cnt_ia;
47                 b -= cnt_b - cnt_ia;
48             } else {
49                 res = (res << 1) | 1;
50                 k -= ab_zeros;
51                 ia += (ib - ia) - (cnt_ib - cnt_ia);
52                 a += (ib - a) - (cnt_ib - cnt_a);
53                 b += (ib - b) - (cnt_ib - cnt_b);
54             }
55         }
56     }
57 }
```

2.2 lazysegtree

```
1 template<class S,
2     S (*e)(),
3     S (*op)(S, S),
4     class F,
5     F (*id)(),
6     S (*mapping)(F, S),
7     F (*composition)(F, F)>
8 struct lazy_segtree {
9     int n, size, log;
10    vector<S> d; vector<F> lz;
11    void update(int k) { d[k] = op(d[k << 1],
12        d[k << 1 | 1]); }
13    void all_apply(int k, F f) {
14        d[k] = mapping(f, d[k]);
15        if(k < size) lz[k] = composition(f, lz[k]);
16    }
17    void push(int k) {
18        all_apply(k << 1, lz[k]);
19        all_apply(k << 1 | 1, lz[k]);
20        lz[k] = id();
21    }
```

```
21 lazy_segtree(int _n) : lazy_segtree(vector
22    <S>(_n, e())) {}
23 lazy_segtree(const vector<S>& v) : n(SZ(v))
24    {}
25     log = __lg(2 * n - 1), size = 1 << log;
26     d.resize(size * 2, e());
27     lz.resize(size, id());
28     REP(i, n) d[size + i] = v[i];
29     for(int i = size - 1; i; i--) update(i);
30 }
31 void set(int p, S x) {
32     p += size;
33     for(int i = log; i; --i) push(p >> i);
34     d[p] = x;
35     for(int i = 1; i <= log; ++i) update(p
36         >> i);
37 }
38 S get(int p) {
39     p += size;
40     for(int i = log; i; i--) push(p >> i);
41     return d[p];
42 }
43 S prod(int l, int r) {
44     if(l == r) return e();
45     l += size; r += size;
46     for(int i = log; i; i--) {
47         if(((l >> i) << i) != 1) push(l >> i);
48         if(((r >> i) << i) != 1) push(r >> i);
49     }
50     S sm1 = e(), smr = e();
51     while(l < r) {
52         if(l & 1) sm1 = op(sm1, d[l++]);
53         if(r & 1) smr = op(d[--r], smr);
54         l >>= 1, r >>= 1;
55     }
56     return op(sm1, smr);
57 }
58 S all_prod() const { return d[1]; }
59 void apply(int p, F f) {
60     p += size;
61     for(int i = log; i; i--) push(p >> i);
62     d[p] = mapping(f, d[p]);
63     for(int i = 1; i <= log; i++) update(p
64         >> i);
65 }
66 void apply(int l, int r, F f) {
67     if(l == r) return;
68     if(l & 1) return;
69     l += size; r += size;
70     for(int i = log; i; i--) {
71         if(((l >> i) << i) != 1) push(l >> i);
72         if(((r >> i) << i) != 1) push((r - 1)
73             >> i);
74     }
75     }
76     int l2 = l, r2 = r;
77     while(l < r) {
78         if(l & 1) all_apply(l++, f);
79         if(r & 1) all_apply(--r, f);
80         l >>= 1, r >>= 1;
81     }
82     l = l2;
83     r = r2;
84 }
85 for(int i = 1; i <= log; i++) {
86     if(((l >> i) << i) != 1) update(l >> i
87         );
88 }
```

```
81     if(((r >> i) << i) != 1) update((r -
82         1) >> i);
83 }
84 template<class G> int max_right(int l, G g
85     ) {
86     assert(0 <= l && l <= n && g(e()));
87     if(l == n) return n;
88     l += size;
89     for(int i = log; i; i--) push(l >> i);
90     S sm = e();
91     do {
92         while(!(l & 1)) l >>= 1;
93         if(!g(op(sm, d[l]))) {
94             while(l < size) {
95                 push(l);
96                 l <<= 1;
97                 if(g(op(sm, d[l]))) sm = op(sm, d[
98                     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     {
108     assert(0 <= r && r <= n && g(e()));
109     if(r == 0) return 0;
110     r += size;
111     for(int i = log; i >= 1; i--) push((r -
112         1) >> i);
113     S sm = e();
114     do {
115         r--;
116         while(r > 1 && (r & 1)) r >>= 1;
117         if(!g(op(d[r], sm))) {
118             while(r < size) {
119                 push(r);
120                 r = r << 1 | 1;
121                 if(g(op(d[r], sm))) sm = op(d[r
122                     --], sm);
123             }
124             return r + 1 - size;
125         }
126     } while((r & -r) != r);
127     return 0;
128 }
```

2.3 LiChao

```
1 struct LiChao { // min
2     int n;
3     vector<pll> seg;
4     LiChao(int _n) : n(_n) {
5         seg.assign(4 * n + 5, pll(0, INF));
6     }
7     ll cal(pll line, ll x) { return line.F * x
8         + line.S; }
```

```

8 void insert(int l, int r, int id, pll line
9 ) {
10     if(l == r) {
11         if(cal(line, l) < cal(seg[id], l)) seg
12             [id] = line;
13         return;
14     }
15     int mid = (l + r) / 2;
16     if(line.F > seg[id].F) swap(line, seg[id]
17 );
18     if(cal(line, mid) <= cal(seg[id], mid))
19     {
20         seg[id] = line;
21         insert(l, mid, id * 2, seg[id]);
22     }
23     else insert(mid + 1, r, id * 2 + 1, line
24 );
25 }
26 ll query(int l, int r, int id, ll x) {
27     if(x < l || x > r) return INF;
28     if(l == r) return cal(seg[id], x);
29     int mid = (l + r) / 2;
30     ll val = 0;
31     if(x <= mid) val = query(l, mid, id * 2,
32 x);
33     else val = query(mid + 1, r, id * 2 + 1,
34 x);
35     return min(val, cal(seg[id], x));
36 }
37 };

```

```

26 L[R[c]] = L[c], R[L[c]] = R[c];
27 TRAV(i, D, c) TRAV(j, R, i) {
28     D[U[D[j]]] = U[j] = D[j];
29     siz[col[j]]--;
30 }
31 }
32 void recover(int c) {
33     int i, j;
34     TRAV(i, U, c) TRAV(j, L, i) {
35         U[D[j]] = D[U[j]] = j;
36         siz[col[j]]++;
37     }
38     L[R[c]] = R[L[c]] = c;
39 }
40 bool dance(int dep) {
41     if(!R[0]) return ans = dep, true;
42     int i, j, c = R[0];
43     TRAV(i, R, 0) if(siz[i] < siz[c]) c = i;
44     remove(c);
45     TRAV(i, D, c) {
46         stk[dep] = row[i];
47         TRAV(j, R, i) remove(col[j]);
48         if(dance(dep + 1)) return true;
49         TRAV(j, L, i) recover(col[j]);
50     }
51     recover(c);
52     return false;
53 }
54 vi solve() {
55     if(!dance(1)) return {};
56     return vi(stk.begin() + 1, stk.begin() +
57 ans);
58 };

```

2.4 DLX

```

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

```

2.5 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(j, 1, lg) {
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 };

```

2.6 static-range-lis

```

1 #define MEM(a, x, n) memset(a, x, sizeof(int
2     ) * n)
3 using I = int*;
4 struct static_range_lis {
5     int n, ps = 0;
6     I invp, res_monge, pool;
7     vector<vector<pii>> qry;
8     vi ans;
9     static_range_lis(vi a) : n(SZ(a)), qry(n +
10         1) {
11         // a must be permutation of [0, n)
12         pool = (I) malloc(sizeof(int) * n * 100)
13             ;
14         invp = A(n), res_monge = A(n);
15         REP(i, n) invp[a[i]] = i;
16     }
17     inline I A(int x) { return pool + (ps += x
18         ) - x; }
19     void add_query(int l, int r) { qry[l].pb({
20         r, SZ(ans)}), ans.pb(r - 1); }
21     void unit_monge_mult(I a, I b, I r, int n)
22     {
23         if(n == 2) {
24             if(!a[0] && !b[0]) r[0] = 0, r[1] = 1;
25             else r[0] = 1, r[1] = 0;
26             return;
27         }
28         if(n == 1) return r[0] = 0, void();
29         int lps = ps, d = n / 2;
30         I a1 = A(d), a2 = A(n - d), b1 = A(d),
31             b2 = A(n - d);
32         I mpa1 = A(d), mpa2 = A(n - d), mpb1 = A(
33             d), mpb2 = A(n - d);
34         int p[2] = {};
35         REP(i, n) {
36             if(a[i] < d) a1[p[0]] = a[i], mpa1[p
37                 [0]++] = i;
38             else a2[p[1]] = a[i] - d, mpa2[p[1]++]
39                 = i;
40         }
41         p[0] = p[1] = 0;
42         REP(i, n) {
43             if(b[i] < d) b1[p[0]] = b[i], mpb1[p
44                 [0]++] = i;
45             else b2[p[1]] = b[i] - d, mpb2[p[1]++]
46                 = i;
47         }
48         I c1 = A(d), c2 = A(n - d);
49         unit_monge_mult(a1, b1, c1, d);
50         unit_monge_mult(a2, b2, c2, n - d);
51         I cpx = A(n), cpy = A(n), cqx = A(n),
52             cpy = A(n);
53         REP(i, d) cpx[mpa1[i]] = mpb1[c1[i]],
54             cpy[mpa1[i]] = 0;
55         REP(i, n - d) cpx[mpa2[i]] = mpb2[c2[i]
56             ], cpy[mpa2[i]] = 1;
57         REP(i, n) r[i] = cpx[i];
58         REP(i, n) cqx[cpx[i]] = i, cpy[cpx[i]] =
59             cpy[i];
60         int hi = n, lo = n, his = 0, los = 0;
61         REP(i, n) {
62             if(cqy[i] ^ (cqx[i] >= hi)) his--;
63             while(hi > 0 && his < 0) {
64                 hi--;
65                 if(cpy[hi] ^ (cpx[hi] > i)) his++;
66             }

```

```

50 while(lo > 0 && los <= 0) {
51     lo--;
52     if(cpy[lo] ^ (cpx[lo] >= i)) los++;
53 }
54 if(los > 0 && hi == lo) r[lo] = i;
55 if(cqy[i] ^ (cqx[i] >= lo)) los--;
56 }
57 ps = lps;
58 }
59 void subunit_monge_mult(I a, I b, I c, int
60     n) {
61     int lps = ps;
62     I za = A(n), zb = A(n), res = A(n), vis
63         = A(n), mpa = A(n), mpb = A(n), rb =
64         A(n);
65     MEM(vis, 0, n), MEM(mpa, -1, n), MEM(mpb
66         , -1, n), MEM(rb, -1, n);
67     int ca = n;
68     IREP(i, n) if(a[i] != -1) vis[a[i]] = 1,
69         za[--ca] = a[i], mpa[ca] = i;
70     IREP(i, n) if(!vis[i]) za[--ca] = i;
71     MEM(vis, -1, n);
72     REP(i, n) if(b[i] != -1) vis[b[i]] = i;
73     ca = 0;
74     REP(i, n) if(vis[i] != -1) mpb[ca] = i,
75         rb[vis[i]] = ca++;
76     REP(i, n) if(rb[i] == -1) rb[i] = ca++;
77     REP(i, n) zb[rb[i]] = i;
78     unit_monge_mult(za, zb, res, n);
79     MEM(c, -1, n);
80     REP(i, n) if(mpa[i] != -1 && mpb[res[i]]
81         != -1) c[mpa[i]] = mpb[res[i]];
82     ps = lps;
83 }
84 void solve(I p, I ret, int n) {
85     if(n == 1) return ret[0] = -1, void();
86     int lps = ps, d = n / 2;
87     I pl = A(d), pr = A(n - d);
88     REP(i, d) pl[i] = p[i];
89     REP(i, n - d) pr[i] = p[i + d];
90     I vis = A(n); MEM(vis, -1, n);
91     REP(i, d) vis[pl[i]] = i;
92     I tl = A(d), tr = A(n - d), mpl = A(d),
93         mpr = A(n - d);
94     int ca = 0;
95     REP(i, n) if(vis[i] != -1) mpl[ca] = i,
96         tl[vis[i]] = ca++;
97     ca = 0; MEM(vis, -1, n);
98     REP(i, n - d) vis[pr[i]] = i;
99     REP(i, n) if(vis[i] != -1) mpr[ca] = i,
100         tr[vis[i]] = ca++;
101     I vl = A(d), vr = A(n - d);
102     solve(tl, vl, d), solve(tr, vr, n - d);
103     I sl = A(n), sr = A(n);
104     iota(sl, sl + n, 0); iota(sr, sr + n, 0)
105         ;
106     REP(i, d) sl[mpl[i]] = (vl[i] == -1 ? -1
107         : mpl[vl[i]]);
108     REP(i, n - d) sr[mpr[i]] = (vr[i] == -1
109         ? -1 : mpr[vr[i]]);
110     subunit_monge_mult(sl, sr, ret, n);
111     ps = lps;
112 }
113 vi solve() {
114     solve(invp, res_monge, n);
115     vi fenw(n + 1);

```

```

103 IREP(i, n) {
104     if(res_monge[i] != -1) {
105         for(int p = res_monge[i] + 1; p <= n
            ; p += p & -p) fenw[p]++;
106     }
107     for(auto& z : qry[i]){
108         auto [id, c] = z;
109         for(int p = id; p; p -= p & -p) ans[
            c] -= fenw[p];
110     }
111 }
112 free(pool);
113 return ans;
114 }
115 };

```

2.7 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.eb(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 };

```

2.8 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)
            , a[i]) - _a.begin();
8     sz = SZ(_a);
9 }
10 void add_query(int l, int r) { L.push_back
    (l), R.push_back(r); }
11 vector<ll> solve() {
12     const int q = SZ(L);
13     const int B = max(1.0, SZ(a) / sqrt(q));
14     vi ord(q);
15     iota(ALL(ord), 0);
16     sort(ALL(ord), [&](int i, int j) {
17         if(L[i] / B == L[j] / B) {
18             return L[i] / B & 1 ? R[i] > R[j] :
                R[i] < R[j];
19         }
20         return L[i] < L[j];
21     });
22     ans.resize(q);
23     fenwick<ll> fenw(sz + 1);
24     ll cnt = 0;
25     auto AL = [&](int i) {
26         cnt += fenw.sum(0, a[i] - 1);
27         fenw.add(a[i], +1);
28     };
29     auto AR = [&](int i) {
30         cnt += fenw.sum(a[i] + 1, sz);
31         fenw.add(a[i], +1);
32     };
33     auto DL = [&](int i) {
34         cnt -= fenw.sum(0, a[i] - 1);
35         fenw.add(a[i], -1);
36     };
37     auto DR = [&](int i) {
38         cnt -= fenw.sum(a[i] + 1, sz);
39         fenw.add(a[i], -1);
40     };
41     int l = 0, r = 0;
42     REP(i, q) {
43         int id = ord[i], ql = L[id], qr = R[id]
            ;
44         while(l > ql) AL(--l);
45         while(r < qr) AR(++r);
46         while(l < ql) DL(l++);
47         while(r > qr) DR(--r);
48         ans[id] = cnt;
49     }
50     return ans;
51 }
52 };

```

2.9 LCT

```

1 template<class S,
2         S (*e)(),
3         S (*op)(S, S),
4         S (*reversal)(S),
5         class F,
6         F (*id)(),
7         S (*mapping)(F, S),
8         F (*composition)(F, F)>
9 struct lazy_lct {
10 struct Node {

```

```

11     S val = e(), sum = e();
12     F lz = id();
13     bool rev = false;
14     int sz = 1;
15     Node *l = nullptr, *r = nullptr, *p =
        nullptr;
16     Node() {}
17     Node(const S& s) : val(s), sum(s) {}
18     bool is_root() const { return p ==
        nullptr || (p->l != this && p->r !=
            this); }
19 };
20 int n;
21 vector<Node> a;
22 lazy_lct() : n(0) {}
23 explicit lazy_lct(int _n) : lazy_lct(
        vector<S>(_n, e())) {}
24 explicit lazy_lct(const vector<S>& v) : n(
        SZ(v)) { REP(i, n) a.eb(v[i]); }
25 Node* access(int u) {
26     Node* v = &a[u];
27     Node* last = nullptr;
28     for(Node* p = v; p != nullptr; p = p->p)
        splay(p), p->r = last, pull(last =
            p);
29     splay(v);
30     return last;
31 }
32 void make_root(int u) { access(u), a[u].
    rev ^= 1, push(&a[u]); }
33 void link(int u, int v) { make_root(v), a[
    v].p = &a[u]; }
34 void cut(int u) {
35     access(u);
36     if(a[u].l != nullptr) a[u].l->p =
        nullptr, a[u].l = nullptr, pull(&a[u]
            );
37 }
38 void cut(int u, int v) { make_root(u), cut
    (v); }
39 bool is_connected(int u, int v) {
40     if(u == v) return true;
41     return access(u), access(v), a[u].p !=
        nullptr;
42 }
43 int get_lca(int u, int v) { return access(
    u), access(v) - &a[0]; }
44 void set(int u, const S& s) { access(u), a
    [u].val = s, pull(&a[u]); }
45 S get(int u) { return access(u), a[u].val;
    }
46 void apply(int u, int v, const F& f) {
    make_root(u), access(v), all_apply(&a[
        v], f), push(&a[v]); }
47 S prod(int u, int v) { return make_root(u)
    , access(v), a[v].sum; }
48 void rotate(Node* v) {
49     auto attach = [&](Node* p, bool side,
        Node* c) {
50         (side ? p->r : p->l) = c;
51         pull(p);
52         if(c != nullptr) c->p = p;
53     };
54     Node *p = v->p, *g = p->p;
55     bool rgt = (p->r == v);
56     bool rt = p->is_root();

```

```

57     attach(p, rgt, (rgt ? v->l : v->r));
58     attach(v, !rgt, p);
59     if(!rt) attach(g, (g->r == p), v);
60     else v->p = g;
61 }
62 void splay(Node* v) {
63     push(v);
64     while(!v->is_root()) {
65         auto p = v->p;
66         auto g = p->p;
67         if(!p->is_root()) push(g);
68         push(p), push(v);
69         if(!p->is_root()) rotate((g->r == p)
            ? (p->r == v) ? p : v);
70         rotate(v);
71     }
72 }
73 void all_apply(Node* v, F f) {
74     v->val = mapping(f, v->val), v->sum =
        mapping(f, v->sum);
75     v->lz = composition(f, v->lz);
76 }
77 void push(Node* v) {
78     if(v->lz != id()) {
79         if(v->l != nullptr) all_apply(v->l, v
            ->lz);
80         if(v->r != nullptr) all_apply(v->r, v
            ->lz);
81         v->lz = id();
82     }
83     if(v->rev) {
84         swap(v->l, v->r);
85         if(v->l != nullptr) v->l->rev ^= 1;
86         if(v->r != nullptr) v->r->rev ^= 1;
87         v->sum = reversal(v->sum);
88         v->rev = false;
89     }
90 }
91 void pull(Node* v) {
92     v->sz = 1;
93     v->sum = v->val;
94     if(v->l != nullptr) {
95         push(v->l);
96         v->sum = op(v->l->sum, v->sum);
97         v->sz += v->l->sz;
98     }
99     if(v->r != nullptr) {
100         push(v->r);
101         v->sum = op(v->sum, v->r->sum);
102         v->sz += v->r->sz;
103     }
104 }
105 };

```

2.10 segtree-beats

```

1 struct segtree_beats {
2     static constexpr ll INF = numeric_limits<
        ll>::max() / 2.1;
3     struct alignas(32) Node {
4         ll sum = 0, g1 = 0, l1 = 0;
5         ll g2 = -INF, gc = 1, l2 = INF, lc = 1,
            add = 0;

```

```

6  };
7  ll n, log;
8  vector<Node> v;
9  segtree_beats() {}
10 segtree_beats(int n) : segtree_beats(
    vector<ll>(_n)) {}
11 segtree_beats(const vector<ll>& vc) {
12     n = 1, log = 0;
13     while(n < SZ(vc)) n <= 1, log++;
14     v.resize(2 * n);
15     REP(i, SZ(vc)) v[i + n].sum = v[i + n].
        g1 = v[i + n].l1 = vc[i];
16     for(ll i = n - 1; i; --i) update(i);
17 }
18 void range_chmin(int l, int r, ll x) {
    inner_apply<1>(l, r, x); }
19 void range_chmax(int l, int r, ll x) {
    inner_apply<2>(l, r, x); }
20 void range_add(int l, int r, ll x) {
    inner_apply<3>(l, r, x); }
21 void range_update(int l, int r, ll x) {
    inner_apply<4>(l, r, x); }
22 ll range_min(int l, int r) { return
    inner_fold<1>(l, r); }
23 ll range_max(int l, int r) { return
    inner_fold<2>(l, r); }
24 ll range_sum(int l, int r) { return
    inner_fold<3>(l, r); }
25 void update(int k) {
26     Node& p = v[k];
27     Node& l = v[k * 2];
28     Node& r = v[k * 2 + 1];
29     p.sum = l.sum + r.sum;
30     if(l.g1 == r.g1) {
31         p.g1 = l.g1;
32         p.g2 = max(l.g2, r.g2);
33         p.gc = l.gc + r.gc;
34     } else {
35         bool f = l.g1 > r.g1;
36         p.g1 = f ? l.g1 : r.g1;
37         p.gc = f ? l.gc : r.gc;
38         p.g2 = max(f ? r.g1 : l.g1, f ? l.g2 :
            r.g2);
39     }
40     if(l.l1 == r.l1) {
41         p.l1 = l.l1;
42         p.l2 = min(l.l2, r.l2);
43         p.lc = l.lc + r.lc;
44     } else {
45         bool f = l.l1 < r.l1;
46         p.l1 = f ? l.l1 : r.l1;
47         p.lc = f ? l.lc : r.lc;
48         p.l2 = min(f ? r.l1 : l.l1, f ? l.l2 :
            r.l2);
49     }
50 }
51 void push_add(int k, ll x) {
52     Node& p = v[k];
53     p.sum += x << (log + __builtin_clz(k) -
        31);
54     p.g1 += x, p.l1 += x;
55     if(p.g2 != -INF) p.g2 += x;
56     if(p.l2 != INF) p.l2 += x;
57     p.add += x;
58 }
59 void push_min(int k, ll x) {
60     Node& p = v[k];
61     p.sum += (x - p.g1) * p.gc;
62     if(p.l1 == p.g1) p.l1 = x;
63     if(p.l2 == p.g1) p.l2 = x;
64     p.g1 = x;
65 }
66 void push_max(int k, ll x) {
67     Node& p = v[k];
68     p.sum += (x - p.l1) * p.lc;
69     if(p.g1 == p.l1) p.g1 = x;
70     if(p.g2 == p.l1) p.g2 = x;
71     p.l1 = x;
72 }
73 void push(int k) {
74     Node& p = v[k];
75     if(p.add != 0) {
76         push_add(k * 2, p.add);
77         push_add(k * 2 + 1, p.add);
78         p.add = 0;
79     }
80     if(p.g1 < v[k * 2].g1) push_min(k * 2, p
        .g1);
81     if(p.l1 > v[k * 2].l1) push_max(k * 2, p
        .l1);
82     if(p.g1 < v[k * 2 + 1].g1) push_min(k *
        2 + 1, p.g1);
83     if(p.l1 > v[k * 2 + 1].l1) push_max(k *
        2 + 1, p.l1);
84 }
85 void subtree_chmin(int k, ll x) {
86     if(v[k].g1 <= x) return;
87     if(v[k].g2 < x) {
88         push_min(k, x);
89         return;
90     }
91 }
92 push(k);
93 subtree_chmin(k * 2, x), subtree_chmin(k
    * 2 + 1, x);
94 update(k);
95 }
96 void subtree_chmax(int k, ll x) {
97     if(x <= v[k].l1) return;
98     if(x < v[k].l2) {
99         push_max(k, x);
100        return;
101    }
102 }
103 push(k);
104 subtree_chmax(k * 2, x), subtree_chmax(k
    * 2 + 1, x);
105 update(k);
106 }
107 template<int cmd>
108 inline void apply(int k, ll x) {
109     if constexpr(cmd == 1) subtree_chmin(k,
        x);
110     if constexpr(cmd == 2) subtree_chmax(k,
        x);
111     if constexpr(cmd == 3) push_add(k, x);
112     if constexpr(cmd == 4) subtree_chmin(k,
        x), subtree_chmax(k, x);
113 }
114 template<int cmd>
115 void inner_apply(int l, int r, ll x) {
116     if(l == r) return;
117     l += n, r += n;
118     for(int i = log; i >= 1; i--) {
119         if(((l >> i) << i) != 1) push(l >> i);
120         if(((r >> i) << i) != r) push((r - 1)
            >> i);
121     }
122     {
123         int l2 = l, r2 = r;
124         while (l < r) {
125             if(l & 1) _apply<cmd>(l++, x);
126             if(r & 1) _apply<cmd>(--r, x);
127             l >>= 1, r >>= 1;
128         }
129         l = l2, r = r2;
130     }
131     for(int i = 1; i <= log; i++) {
132         if(((l >> i) << i) != 1) update(l >> i
            );
133         if(((r >> i) << i) != r) update((r -
            1) >> i);
134     }
135 }
136 template<int cmd>
137 inline e() {
138     if constexpr(cmd == 1) return INF;
139     if constexpr(cmd == 2) return -INF;
140     return 0;
141 }
142 template<int cmd>
143 inline void op(ll& a, const Node& b) {
144     if constexpr(cmd == 1) a = min(a, b.l1);
145     if constexpr(cmd == 2) a = max(a, b.g1);
146     if constexpr(cmd == 3) a += b.sum;
147 }
148 template<int cmd>
149 ll inner_fold(int l, int r) {
150     if(l == r) return e<cmd>();
151     l += n, r += n;
152     for(int i = log; i >= 1; i--) {
153         if(((l >> i) << i) != 1) push(l >> i);
154         if(((r >> i) << i) != r) push((r - 1)
            >> i);
155     }
156     ll lx = e<cmd>(), rx = e<cmd>();
157     while (l < r) {
158         if(l & 1) op<cmd>(lx, v[l++]);
159         if(r & 1) op<cmd>(rx, v[--r]);
160         l >>= 1, r >>= 1;
161     }
162     if constexpr(cmd == 1) lx = min(lx, rx);
163     if constexpr(cmd == 2) lx = max(lx, rx);
164     if constexpr(cmd == 3) lx += rx;
165     return lx;
166 }
167 }
168 };
169 vector<int> vx, vy;
170 struct q { int piv, s, e, x; };
171 struct tree {
172     vector<int> seg, tag;
173     tree(int n) : seg(_n * 16), tag(_n * 16)
        {}
174     void add(int ql, int qr, int x, int v, int
        l, int r) {
175         if(qr <= 1 || r <= ql) return;
176         if(ql <= 1 && r <= qr) {
177             tag[v] += x;
178             if(tag[v] == 0) {
179                 if(l != r) seg[v] = seg[2 * v] + seg
                    [2 * v + 1];
180                 else seg[v] = 0;
181             } else seg[v] = vx[r] - vx[l];
182         } else {
183             int mid = (l + r) / 2;
184             add(ql, qr, x, 2 * v, l, mid);
185             add(ql, qr, x, 2 * v + 1, mid, r);
186             if(tag[v] == 0 && l != r) seg[v] = seg
                    [2 * v] + seg[2 * v + 1];
187         }
188     }
189     int q() { return seg[1]; }
190 };
191 int main() {
192     int n; cin >> n;
193     vector<int> x1(n), x2(n), y_(n), y2(n);
194     for (int i = 0; i < n; i++) {
195         cin >> x1[i] >> x2[i] >> y_[i] >> y2[i];
196         // L R D U
197         vx.pb(x1[i]), vx.pb(x2[i]);
198         vy.pb(y_[i]), vy.pb(y2[i]);
199     }
200     vx = sort_unique(vx);
201     vy = sort_unique(vy);
202     vector<q> a(2 * n);
203     REP(i, n) {
204         x1[i] = lower_bound(ALL(vx), x1[i]) - vx
            .begin();
205         x2[i] = lower_bound(ALL(vx), x2[i]) - vx
            .begin();
206         y_[i] = lower_bound(ALL(vy), y_[i]) - vy
            .begin();
207         y2[i] = lower_bound(ALL(vy), y2[i]) - vy
            .begin();
208         a[2 * i] = {y_[i], x1[i], x2[i], +1};
209         a[2 * i + 1] = {y2[i], x1[i], x2[i],
            -1};
210     }
211     sort(ALL(a), [](q a, q b) { return a.piv <
        b.piv; });
212     tree seg(n);
213     ll ans = 0;
214     REP(i, 2 * n) {
215         int j = i;
216         while (j < 2 * n && a[i].piv == a[j].piv)
            {
217             seg.add(a[j].s, a[j].e, a[j].x, 1, 0,
                vx.size());
218             j++;
219         }
220         if(a[i].piv + 1 != SZ(vy)) ans += 1LL *
            seg.q() * (vy[a[i].piv + 1] - vy[a[i]
                ].piv);
221     }
222 }

```

2.11 union-of-rectangles

```

1 // 2
2 // 1 10 1 10
3 // 0 2 0 2
4 // ans = 84

```

```

56     i = j - 1;
57 }
58 cout << ans << "\n";
59 }

```

2.12 CHT

```

1 struct line_t {
2     mutable ll k, m, p;
3     bool operator<(const line_t& o) const {
4         return k < o.k; }
5     bool operator<(ll x) const { return p < x; }
6 };
7 template<bool MAX>
8 struct CHT : multiset<line_t, less<>> {
9     const ll INF = 1e18L;
10    bool isect(iterator x, iterator y) {
11        if(y == end()) return x->p = INF, 0;
12        if(x->k == y->k) {
13            x->p = (x->m > y->m ? INF : -INF);
14        } else {
15            x->p = floor_div(y->m - x->m, x->k - y->k); // see Math
16        }
17        return x->p >= y->p;
18    }
19    void add_line(ll k, ll m) {
20        if(!MAX) k = -k, m = -m;
21        auto z = insert({k, m, 0}); y = z++, x = y;
22        while(isect(y, z)) z = erase(z);
23        if(x != begin() && isect(--x, y)) isect(x, y = erase(y));
24        while((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
25    }
26    ll get(ll x) {
27        assert(!empty());
28        auto l = *lower_bound(x);
29        return (l.k * x + l.m) * (MAX ? +1 : -1);
30    };
31 };

```

2.13 treap

```

1 struct Node {
2     bool rev = false;
3     int sz = 1, pri = rng();
4     Node *l = NULL, *r = NULL, *p = NULL;
5     // TODO
6 }
7 void pull(Node& v) {
8     v->sz = 1 + size(v->l) + size(v->r);
9     // TODO
10 }
11 void push(Node& v) {
12     if(v->rev) {
13         swap(v->l, v->r);

```

```

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

```

2.14 VEB

```

1 template<int B, typename ENABLE = void>
2 struct VEB {
3     constexpr static int K = B / 2, R = (B +
4         1) / 2, M = 1 << B, S = 1 << K, MASK =
5         (1 << R) - 1;
6     array<VEB<R>, S> child;
7     VEB<K> act = {};
8     int mn = M, mx = -1;
9     bool empty() { return mx < mn; }
10    bool contains(int i) { return find_next(i)
11        == i; }
12    int find_next(int i) { // >=
13        if(i <= mn) return mn;
14        if(i > mx) return M;
15        int j = i >> R, x = i & MASK;
16        int res = child[j].find_next(x);
17        if(res <= MASK) return (j << R) + res;

```

```

18        j = act.find_next(j + 1);
19        return j >= S ? mx : (j << R) + child[j]
20            .find_next(0);
21    }
22    int find_prev(int i) { // <=
23        if(i >= mx) return mx;
24        if(i < mn) return -1;
25        int j = i >> R, x = i & MASK;
26        int res = child[j].find_prev(x);
27        if(res >= 0) return (j << R) + res;
28        j = act.find_prev(j - 1);
29        return j < 0 ? mn : (j << R) + child[j].
30            find_prev(MASK);
31    }
32    void insert(int i) {
33        if(i <= mn) {
34            if(i == mn) return;
35            swap(mn, i);
36            if(i == M) mx = mn;
37            if(i >= mx) return;
38        } else if(i >= mx) {
39            if(i == mx) return;
40            swap(mx, i);
41            if(i <= mn) return;
42        }
43        int j = i >> R;
44        if(child[j].empty()) act.insert(j);
45        child[j].insert(i & MASK);
46    }
47    void erase(int i) {
48        if(i <= mn) {
49            if(i < mn) return;
50            i = mn = find_next(mn + 1);
51            if(i >= mx) {
52                if(i > mx) mx = -1;
53                return;
54            }
55        } else if(i >= mx) {
56            if(i > mx) return;
57            i = mx = find_prev(mx - 1);
58            if(i <= mn) return;
59        }
60        int j = i >> R;
61        child[j].erase(i & MASK);
62        if(child[j].empty()) act.erase(j);
63    }
64    void clear() {
65        mn = M, mx = -1, act.clear();
66        REP(i, S) child[i].clear();
67    }
68 };
69 template<int B>
70 struct VEB<B, enable_if_t<(B <= 6)>> {
71     constexpr static int M = 1 << B;
72     unsigned long long act = 0;
73     bool empty() { return !act; }
74     void clear() { act = 0; }
75     bool contains(int i) { return find_next(i)
76         == i; }
77     void insert(int i) { act |= 1ULL << i; }
78     void erase(int i) { act &= ~(1ULL << i); }
79     int find_next(int i) {
80         ull tmp = act >> i;
81         return (tmp ? i + __builtin_ctzll(tmp) :
82             M);

```

```

77 }
78 int find_prev(int i) {
79     ull tmp = act << (63 - i);
80     return (tmp ? i - __builtin_clzll(tmp) :
81         -1);
82 }
83 };

```

2.15 rect-add-rect-sum

```

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

```



```

44     b11.add(j, -query.val);
45 } else {
46     const auto& query = sum_qry[q];
47     int i = lower_bound(ALL(ys), query.
48         yl) - ys.begin();
49     int j = lower_bound(ALL(ys), query.
50         yr) - ys.begin();
51     T x = get<0>(o);
52     T yi = query.yl, yj = query.yr;
53     if(qtype & 1) swap(i, j), swap(yi,
54         yj);
55     ret[q] += b00.get(i - 1) + b01.get(i
56         - 1) * yi + b10.get(i - 1) * x
57         + b11.get(i - 1) * x * yi;
58     ret[q] -= b00.get(j - 1) + b01.get(j
59         - 1) * yj + b10.get(j - 1) * x
60         + b11.get(j - 1) * x * yj;
61 }
62 return ret;
63 };

```

2.16 CDQ

```

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

```

2.17 segtree

```

1 template<class S, S (*e)(), S (*op)(S, S)>
2 struct segtree {
3     int n, size, log;
4     vector<S> st;
5     void update(int v) { st[v] = op(st[v <<
6         1], st[v << 1 | 1]); }
7     segtree(int _n) : segtree(vector<S>(_n, e
8         ())) {}
9     segtree(const vector<S>& a) : n(sz(a)) {

```

```

10     log = __lg(2 * n - 1), size = 1 << log;
11     st.resize(size << 1, e());
12     REP(i, n) st[size + i] = a[i];
13     for(int i = size - 1; i; i--) update(i);
14 }
15 void set(int p, S val) {
16     st[p += size] = val;
17     for(int i = 1; i <= log; ++i) update(p
18         >> i);
19 }
20 S get(int p) const {
21     return st[p + size];
22 }
23 S prod(int l, int r) const {
24     assert(0 <= l && l <= r && r <= n);
25     S sm1 = e(), smr = e();
26     l += size, r += size;
27     while(l < r) {
28         if(l & 1) sm1 = op(sm1, st[l++]);
29         if(r & 1) smr = op(st[--r], smr);
30         l >>= 1;
31         r >>= 1;
32     }
33     return op(sm1, smr);
34 }
35 S all_prod() const { return st[1]; }
36 template<class F> int max_right(int l, F f)
37     const {
38     assert(0 <= l && l <= n && f(e()));
39     if(l == n) return n;
40     l += size;
41     S sm = e();
42     do {
43         while(~l & 1) l >>= 1;
44         if(!f(op(sm, st[l]))) {
45             while(l < size) {
46                 l <<= 1;
47                 if(f(op(sm, st[l]))) sm = op(sm,
48                     st[l++]);
49             }
50             return l - size;
51         }
52         sm = op(sm, st[l++]);
53     } while((l & -l) != 1);
54     return n;
55 }
56 template<class F> int min_left(int r, F f)
57     const {
58     assert(0 <= r && r <= n && f(e()));
59     if(r == 0) return 0;
60     r += size;
61     S sm = e();
62     do {
63         r--;
64         while(r > 1 && (r & 1)) r >>= 1;
65         if(!f(op(st[r], sm))) {
66             while(r < size) {
67                 r <<= 1 | 1;
68                 if(f(op(st[r], sm))) sm = op(st[r
69                     --], sm);
70             }
71             return r + 1 - size;
72         }
73     } while((r & -r) != 1);
74     sm = op(st[r], sm);
75     while((r & -r) != 1);
76     return 0;

```

3 Flow-Matching

3.1 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), fill(ALL(vr), 0);
26         ql = qr = 0, qu[qr++] = s, vr[s] = 1;
27         while(true) {
28             T d;
29             while(ql < qr) {
30                 for(int x = 0, y = qu[ql++]; x < n;
31                     ++x) {
32                     if(!vl[x] && slk[x] >= (d = hl[x]
33                         + hr[y] - w[x][y])) {
34                         pre[x] = y;
35                         if(d) slk[x] = d;
36                         else if(!check(x)) return;
37                     }
38                 }
39             }
40             d = INF;
41             REP(x, n) if(!vl[x] && d > slk[x]) d =
42                 slk[x];
43             REP(x, n) {
44                 if(vl[x]) hl[x] += d;
45                 else slk[x] -= d;
46                 if(vr[x]) hr[x] -= d;
47             }
48             REP(x, n) if(!vl[x] && !slk[x] && !
49                 check(x)) return;
50         }
51     }
52     T solve() {
53         fill(ALL(fl), -1);
54         fill(ALL(fr), -1);

```

```

45     fill(ALL(hr), 0);
46     REP(i, n) hl[i] = *max_element(ALL(w[i])
47         );
48     REP(i, n) bfs(i);
49     T ans = 0;
50     REP(i, n) ans += w[i][fl[i]]; // i 跟 fl
51         [i] 配對
52     return ans;
53 }
54 };

```

3.2 bipartite-matching

```

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

```

3.3 Dinic-LowerBound

```

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

```

3.4 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;

```

```

12     vector<Edge> edges;
13     vector<vi> g;
14     vector<T> d;
15     vector<bool> inq;
16     vi pedge;
17     MCMF(int _n) : n(_n), g(_n), d(_n), inq(_n)
18         {}, pedge(_n) {}
19     void add_edge(int u, int v, S cap, T cost)
20     {
21         g[u].pb(SZ(edges));
22         edges.pb({u, v, cap, cost});
23         g[v].pb(SZ(edges));
24         edges.pb({v, u, 0, -cost});
25     }
26     bool spfa(int s, int t) {
27         bool found = false;
28         fill(ALL(d), INF);
29         d[s] = 0;
30         inq[s] = true;
31         queue<int> q;
32         q.push(s);
33         while(!q.empty()) {
34             int u = q.front(); q.pop();
35             if(u == t) found = true;
36             inq[u] = false;
37             for(auto& id : g[u]) {
38                 const auto& e = edges[id];
39                 if(e.cap > 0 && d[u] + e.cost < d[e
40                     .to]) {
41                     d[e.to] = d[u] + e.cost;
42                     pedge[e.to] = id;
43                     if(!inq[e.to]) {
44                         q.push(e.to);
45                         inq[e.to] = true;
46                     }
47                 }
48             }
49         }
50         return found;
51     }
52     pair<S, T> flow(int s, int t, S f = INF) {
53         S cap = 0;
54         T cost = 0;
55         while(f > 0 && spfa(s, t)) {
56             S send = f;
57             int u = t;
58             while(u != s) {
59                 const Edge& e = edges[pedge[u]];
60                 send = min(send, e.cap);
61                 u = e.from;
62             }
63             u = t;
64             while(u != s) {
65                 Edge& e = edges[pedge[u]];
66                 e.cap -= send;
67                 Edge& b = edges[pedge[u] ^ 1];
68                 b.cap += send;
69                 u = e.from;
70             }
71             cap += send;
72             f -= send;
73             cost += send * d[t];
74         }
75         return {cap, cost};
76     }
77 };

```

3.5 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                ]){
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                    onstk[v] = 1;
29                    stk.push_back(v);
30                    if(SPFA(m)) return true;
31                    stk.pop_back();
32                    onstk[v] = 0;
33                }
34            }
35        }
36        onstk[u] = 0;
37        stk.pop_back();
38        return false;
39    }
40    int solve() {
41        for(int i = 0; i < n; i += 2) match[i] =
42            i + 1, match[i+1] = i;
43        while(true) {
44            int found = 0;
45            for(int i=0; i<n; i++) dis[i] = onstk[i
46                ] = 0;
47            for(int i=0; i<n; i++){
48                stk.clear();
49                if(!onstk[i] && SPFA(i)){
50                    found = 1;
51                    while(stk.size()>2){
52                        int u = stk.back(); stk.pop_back
53                            ();
54                        int v = stk.back(); stk.pop_back
55                            ();
56                        match[u] = v;
57                        match[v] = u;
58                    }
59                }
60            }
61            if(!found) break;
62        }

```

```

53     }
54     int ans = 0;
55     for(int i=0; i<n; i++) ans += edge[i][
56         match[i]];
57     return ans / 2;
58 }

```

3.6 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

4. For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 5. For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 1. For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
 2. Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 3. Find the minimum weight perfect matching on G' .
 - Project selection problem
 1. If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 3. The mincut is equivalent to the maximum profit of a subset of projects.
 - 0/1 quadratic programming

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

can be minimized by the mincut of the following graph:

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

3.7 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(w) {}
9     };
10    int n, n_x;
11    edge g[maxn * 2][maxn * 2];
12    int lab[maxn * 2];
13    int match[maxn * 2], slack[maxn * 2], st[
14        maxn * 2], pa[maxn * 2];
15    vector<int> flo[maxn * 2];
16    queue<int> q;
17    int e_delta(const edge &e) { return lab[e.
18        u] + lab[e.v] - g[e.u][e.v].w * 2; }
19    void update_slack(int u, int x) { if(!
20        slack[x] || e_delta(g[u][x]) < e_delta
21        (g[slack[x]][x])) slack[x] = u; }
22    void set_slack(int x) {

```

```

23    slack[x] = 0;
24    REP(u, n) if(g[u + 1][x].w > 0 && st[u +
25        1] != x && S[st[u + 1]] == 0)
26        update_slack(u + 1, x);
27    }
28    void q_push(int x) {
29        if(x <= n) q.push(x);
30        else REP(i, SZ(flo[x])) q_push(flo[x][i
31        ]);
32    }
33    void set_st(int x, int b) {
34        st[x] = b;
35        if(x > n) REP(i, SZ(flo[x])) set_st(flo[
36        x][i], b);
37    }
38    int get_pr(int b, int xr) {
39        int pr = find(ALL(flo[b]), xr) - flo[b].
40        begin();
41        if(pr % 2 == 1) {
42            reverse(1 + ALL(flo[b]));
43            return SZ(flo[b]) - pr;
44        }
45        return pr;
46    }
47    void set_match(int u, int v) {
48        match[u] = g[u][v].v;
49        if(u <= n) return;
50        edge e = g[u][v];
51        int xr = flo_from[u][e.u], pr = get_pr(u
52        , xr);
53        for(int i = 0; i < pr; ++i) set_match(
54            flo[u][i], flo[u][i ^ 1]);
55        set_match(xr, v);
56        rotate(flo[u].begin(), flo[u].begin() +
57            pr, flo[u].end());
58    }
59    void augment(int u, int v) {
60        while(true) {
61            int xnv = st[match[u]];
62            set_match(u, v);
63            if(!xnv) return;
64            set_match(xnv, st[pa[xnv]]);
65            u = st[pa[xnv]], v = xnv;
66        }
67    }
68    int get_lca(int u, int v) {
69        static int t = 0;
70        for(++t; u || v; swap(u, v)) {
71            if(u == 0) continue;
72            if(vis[u] == t) return u;
73            vis[u] = t;
74            if(u = st[match[u]]) u = st[pa[u]];
75        }
76        return 0;
77    }
78    void add_blossom(int u, int lca, int v) {
79        int b = n + 1;
80        while(b <= n_x && st[b]) ++b;
81        if(b > n_x) n_x++;
82        lab[b] = S[b] = 0;
83        match[b] = match[lca];
84        flo[b].clear(); flo[b].pb(lca);
85        for(int x = u, y; x != lca; x = st[pa[y
86        ]]) flo[b].pb(x), flo[b].pb(y = st[
87        match[x]]); q.push(y);
88        reverse(1 + ALL(flo[b]));

```

```

89    for(int x = v, y; x != lca; x = st[pa[y
90        ]]) flo[b].pb(x), flo[b].pb(y = st[
91        match[x]]); q.push(y);
92    set_st(b, b);
93    REP(x, n_x) g[b][x + 1].w = g[x + 1][b].
94    w = 0;
95    REP(x, n) flo_from[b][x + 1] = 0;
96    REP(i, SZ(flo[b])) {
97        int xs = flo[b][i];
98        REP(x, n_x) if(g[b][x + 1].w == 0 ||
99            e_delta(g[xs][x + 1]) < e_delta(g[
100                b][x + 1])) g[b][x + 1] = g[xs][x
101                + 1], g[x + 1][b] = g[x + 1][xs];
102        REP(x, n) if(flo_from[xs][x + 1])
103            flo_from[b][x + 1] = xs;
104    }
105    set_slack(b);
106    }
107    void expand_blossom(int b) {
108        REP(i, SZ(flo[b])) set_st(flo[b][i], flo
109        [b][i]);
110        int xr = flo_from[b][g[b][pa[b]].u], pr
111        = get_pr(b, xr);
112        for(int i = 0; i < pr; i += 2) {
113            int xs = flo[b][i], xns = flo[b][i +
114            1];
115            pa[xs] = g[xns][xs].u;
116            S[xs] = 1, S[xns] = 0;
117            slack[xs] = 0, set_slack(xns);
118            q_push(xns);
119        }
120        S[xr] = 1, pa[xr] = pa[b];
121        for(size_t i = pr + 1; i < SZ(flo[b]);
122            ++i) {
123            int xs = flo[b][i];
124            S[xs] = -1, set_slack(xs);
125        }
126        st[b] = 0;
127    }
128    bool on_found_edge(const edge &e) {
129        int u = st[e.u], v = st[e.v];
130        if(S[v] == -1) {
131            pa[v] = e.u, S[v] = 1;
132            int nu = st[match[v]];
133            slack[v] = slack[nu] = 0;
134            S[nu] = 0, q_push(nu);
135        }
136        else if(S[v] == 0) {
137            int lca = get_lca(u, v);
138            if(!lca) return augment(u, v), augment(
139                v, u), true;
140            else add_blossom(u, lca, v);
141        }
142        return false;
143    }
144    bool matching() {
145        memset(S + 1, -1, sizeof(int) * n_x);
146        memset(slack + 1, 0, sizeof(int) * n_x);
147        q = queue<int>();
148        REP(x, n_x) if(st[x + 1] == x + 1 && !
149            match[x + 1]) pa[x + 1] = 0, S[x +
150            1] = 0, q_push(x + 1);
151        if(q.empty()) return false;
152        while(true) {
153            while(!q.empty()) {
154                int u = q.front(); q.pop();
155                if(S[st[u]] == 1) continue;

```

```

156            for(int v = 1; v <= n; ++v)
157                if(g[u][v].w > 0 && st[u] != st[v]
158                ) {
159                    if(e_delta(g[u][v]) == 0) {
160                        if(on_found_edge(g[u][v]))
161                            return true;
162                        else update_slack(u, st[v]);
163                    }
164                }
165            int d = inf;
166            for(int b = n + 1; b <= n_x; ++b) if(
167                st[b] == b && S[b] == 1) d = min(d
168                , lab[b] / 2);
169            for(int x = 1; x <= n_x; ++x) {
170                if(st[x] == x && slack[x]) {
171                    if(S[x] == -1) d = min(d, e_delta(
172                        g[slack[x]][x]));
173                    else if(S[x] == 0) d = min(d,
174                        e_delta(g[slack[x]][x]) / 2);
175                }
176            }
177            REP(u, n) {
178                if(S[st[u + 1]] == 0) {
179                    if(lab[u + 1] <= d) return 0;
180                    lab[u + 1] -= d;
181                }
182                else if(S[st[u + 1]] == 1) lab[u +
183                1] += d;
184            }
185            q = queue<int>();
186            for(int x = 1; x <= n_x; ++x)
187                if(st[x] == x && slack[x] && st[
188                    slack[x]] != x && e_delta(g[
189                        slack[x]][x]) == 0)
190                    if(on_found_edge(g[slack[x]][x]))
191                        return true;
192            for(int b = n + 1; b <= n_x; ++b)
193                if(st[b] == b && S[b] == 1 && lab[b]
194                == 0) expand_blossom(b);
195        }
196        return false;
197    }
198    pair<ll, int> solve() {
199        memset(match + 1, 0, sizeof(int) * n);
200        n_x = n;
201        int n_matches = 0;
202        ll tot_weight = 0;
203        for(int u = 0; u <= n; ++u) st[u] = u,
204            flo[u].clear();
205        int w_max = 0;
206        for(int u = 1; u <= n; ++u)
207            for(int v = 1; v <= n; ++v) {
208                flo_from[u][v] = (u == v ? 0);
209                w_max = max(w_max, g[u][v].w);
210            }
211        for(int u = 1; u <= n; ++u) lab[u] =
212            w_max;
213        while(matching()) ++n_matches;
214        for(int u = 1; u <= n; ++u)
215            if(match[u] && match[u] < u)
216                tot_weight += g[u][match[u]].w;

```

```

180     return make_pair(tot_weight, n_matches);
181 }
182 void add_edge(int u, int v, int w) { g[u][
    v].w = g[v][u].w = w; }
183 void init(int _n) : n(_n) {
184     REP(u, n) REP(v, n) g[u + 1][v + 1] =
        edge(u + 1, v + 1, 0);
185 }
186 };

```

3.8 general-matching

```

1 struct GeneralMaxMatch {
2     int n;
3     vector<pii> es;
4     vi g, vis, mate; // i 與 mate[i] 配對 (
        mate[i] == -1 代表沒有匹配)
5     GeneralMaxMatch(int n) : n(n), g(n, -1),
        mate(n, -1) {}
6     bool dfs(int u) {
7         if(vis[u]) return false;
8         vis[u] = true;
9         for(int ei = g[u]; ei != -1; ei++) {
10             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; ei++) {
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.pb(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); iota(ALL(o), 0);
36         int ans = 0;
37         REP(it, 100) {
38             shuffle(ALL(o), rng);
39             vis.assign(n, false);
40             for(auto i : o) if(mate[i] == -1) ans
                += dfs(i);
41         }
42         return ans;
43     }
44 };

```

3.9 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)
            , cap(z) {}
8     };
9     constexpr T INF = 1E9;
10    int n;
11    vector<Edge> edges;
12    vector<vi> g;
13    vi cur, h; // h : Level graph
14    Dinic(int _n) : n(_n), g(_n) {}
15    void add_edge(int u, int v, T c) {
16        g[u].pb(SZ(edges));
17        edges.pb(u, v, c);
18        g[v].pb(SZ(edges));
19        edges.pb(v, u, 0);
20    }
21    bool bfs(int s, int t) {
22        h.assign(n, -1);
23        queue<int> q;
24        h[s] = 0;
25        q.push(s);
26        while(!q.empty()) {
27            int u = q.front(); q.pop();
28            for(int i : g[u]) {
29                const auto& e = edges[i];
30                int v = e.to;
31                if(e.cap > 0 && h[v] == -1) {
32                    h[v] = h[u] + 1;
33                    if(v == t) return true;
34                    q.push(v);
35                }
36            }
37        }
38        return false;
39    }
40    T dfs(int u, int t, T f) {
41        if(u == t) return f;
42        T r = f;
43        for(int& i = cur[u]; i < SZ(g[u]); ++i)
44        {
45            int j = g[u][i];
46            const auto& e = edges[j];
47            int v = e.to;
48            T c = e.cap;
49            if(c > 0 && h[v] == h[u] + 1) {
50                T a = dfs(v, t, min(r, c));
51                edges[j].cap -= a;
52                edges[j ^ 1].cap += a;
53                if((r -= a) == 0) return f;
54            }
55        }
56        return f - r;
57    }
58    T flow(int s, int t, T f = INF) {
59        T ans = 0;
60        while(f > 0 && bfs(s, t)) {
61            cur.assign(n, 0);
62            T cur = dfs(s, t, f);
63        }
64    }
65 };

```

```

62     ans += cur;
63     f -= cur;
64 }
65 return ans;
66 }
67 };

```

3.10 max-clique

```

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

```

```

48     dfs(remark);
49     return clique;
50 }
51 };

```

4 Geometry

4.1 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 }

```

4.2 half-plane

```

1 typedef pair<double, double> pdd;
2 pdd operator-(pdd a, pdd b) { return {a.F - b.F, a
    .S - b.S}; }
3 pdd operator+(pdd a, pdd b) { return {a.F + b.F, a
    .S + b.S}; }
4 pdd operator*(pdd a, double x) { return {a.F * x,
    a.S * x}; }
5 double dot(pdd a, pdd b) { return a.F * b.F + a.S
    * b.S; }
6 double cross(pdd a, pdd b) { return a.F * b.S - a.S
    * b.F; }
7 struct bpmj {
8     const double eps = 1e-8;
9     int n, m, id, l, r;
10    pdd pt[55], q[1100];
11    struct line {
12        pdd x, y;
13        double z;
14        line(pdd _x, pdd _y) : x(_x), y(_y) { z = atan2(
            y.S, y.F); }
15        line() {}
16    };
17 };

```

```

16 bool operator<(const line &a)const{
17     return z<a.z;}
18 }a[550],dq[1005];
19 pdd get_(line x,line y){
20     pdd v=x.x-y.x;
21     double d=cross(y.y,v)/cross(x.y,y.y);
22     return x.x+x.y*d;
23 }
24 void solve(){
25     dq[l=r=1]=a[1];
26     for(int i=2;i<=id;++i){
27         while(l<r&&cross(a[i].y,q[r-1]-a[i].x)
28             <=eps) --r;
29         while(l<r&&cross(a[i].y,q[l]-a[i].x)<=
30             eps) ++l;
31         dq[++r]=a[i];
32         if(fabs(cross(dq[r].y,dq[r-1].y))<=eps)
33             continue;
34         if(cross(dq[r].y,a[i].x-dq[r].x)>eps)
35             dq[r]=a[i];
36         if(l<r) q[r-1]=get_(dq[r-1],dq[r]);
37     }
38     while(l<r&&cross(dq[l].y,q[r-1]-dq[l].x)
39         <=eps) --r;
40     if(r-l<=1) return;
41     q[r]=get_(dq[l],dq[r]);
42 }
43 void cal(){
44     double ans=0;
45     q[r+1]=q[l];
46     for(int i=1;i<=r;++i) ans+=cross(q[i],q[
47         i+1]);
48     cout<<fixed<<setprecision(3)<<ans/2<<"\n
49     ";
50 }
51 void main_(){
52     cin>>n;
53     for(int x,y,i=0;i<n;++i){
54         cin>>m;
55         for(int i=0;i<m;++i) cin>>pt[i].F>>pt[
56             i].S;
57         pt[m]=pt[0];
58         for(int i=0;i<m;++i) a[++id]=line(pt[i
59             ],pt[i+1]-pt[i]);
60     }
61     sort(a+1,a+1+id);
62     solve();
63     cal();
64 }
65 }valderyyaya;

```

4.3 point

```

1 using P = pair<ll, ll>;
2 P operator+(P a, P b) { return P{a.X + b.X,
3     a.Y + b.Y}; }
4 P operator-(P a, P b) { return P{a.X - b.X,
5     a.Y - b.Y}; }
6 P operator*(P a, ll b) { return P{a.X * b, a
7     .Y * b}; }

```

```

5 P operator/(P a, ll b) { return P{a.X / b, a
6     .Y / b}; }
7 ll dot(P a, P b) { return a.X * b.X + a.Y *
8     b.Y; }
9 ll cross(P a, P b) { return a.X * b.Y - a.Y
10    * b.X; }
11 ll abs2(P a) { return dot(a, a); }
12 double abs(P a) { return sqrt(abs2(a)); }
13 int sign(ll x) { return x < 0 ? -1 : (x == 0
14    ? 0 : 1); }
15 int ori(P a, P b, P c) { return sign(cross(b
16    - a, c - a)); }
17 bool collinear(P a, P b, P c) { return sign(
18    cross(a - c, b - c)) == 0; }
19 bool btw(P a, P b, P c) {
20     if(!collinear(a, b, c)) return 0;
21     return sign(dot(a - c, b - c)) <= 0;
22 }
23 bool seg_intersect(P a, P b, P c, P d) {
24     int a123 = ori(a, b, c);
25     int a124 = ori(a, b, d);
26     int a341 = ori(c, d, a);
27     int a342 = ori(c, d, b);
28     if(a123 == 0 && a124 == 0) {
29         return btw(a, b, c) || btw(a, b, d) ||
30             btw(c, d, a) || btw(c, d, b);
31     }
32     return a123 * a124 <= 0 && a341 * a342 <=
33         0;
34 }
35 P intersect(P a, P b, P c, P d) {
36     int a123 = cross(b - a, c - a);
37     int a124 = cross(b - a, d - a);
38     return (d * a123 - c * a124) / (a123 -
39         a124);
40 }
41 struct line { P A, B; };
42 P vec(line L) { return L.B - L.A; }
43 P projection(P p, line L) { return L.A + vec
44     (L) / abs(vec(L)) * dot(p - L.A, vec(L))
45     / abs(vec(L)); }

```

4.4 定理

- 皮克定理

– 若一個多邊形的所有頂點都在整數點上，則該多邊形的面積 $S = a + \frac{b}{2} - 1$ ，其中 a 為內部格點數目， b 為邊上格點數目。

4.5 min-enclosing-circle

```

1 pdd excenter(pdd x, pdd y, pdd z) {
2     #define f(x, y) (x*x+y*y)
3     auto [x1, y1] = x;
4     auto [x2, y2] = y;
5     auto [x3, y3] = z;
6     double d1 = f(x2, y2) - f(x1, y1), d2 = f(
7         x3, y3) - f(x2, y2);

```

```

7 double fm = 2 * ((y3 - y2) * (x2 - x1) - (
8     y2 - y1) * (x3 - x2));
9 double ans_x = ((y3 - y2) * d1 - (y2 - y1)
10    * d2) / fm;
11 double ans_y = ((x2 - x1) * d2 - (x3 - x2)
12    * d1) / fm;
13 #undef f
14 return {ans_x, ans_y};
15 }
16 pdd min_enclosing_circle(vector<pdd> dots,
17    double& r) {
18     random_shuffle(ALL(dots));
19     pdd C = dots[0];
20     r = 0;
21     #define check(i, j) REP(i, j) if(abs(dots[
22         i] - C) > r)
23     check(i, SZ(dots)) {
24         C = dots[i], r = 0;
25         check(j, i) {
26             C = (dots[i] + dots[j]) / 2.0;
27             r = abs(dots[i] - C);
28             check(k, j) {
29                 C = excenter(dots[i], dots[j], dots[
30                     k]);
31                 r = abs(dots[i] - C);
32             }
33         }
34     }
35     #undef check
36     return C;
37 }

```

4.6 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(
5         dots))) {
6         for(int i = 1, t = SZ(ans); i < SZ(dots)
7             ; ans.pb(dots[i++])) {
8             while(SZ(ans) > t && ori(ans[SZ(ans) -
9                 2], ans.back(), dots[i]) < 0) {
10                 ans.ppb();
11             }
12         }
13     }
14     ans.ppb();
15     swap(ans, dots);
16 }

```

4.7 polar-angle-sort

```

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

```

```

5 if(sign(cross(a, b)) == 0) return abs2(a)
6     < abs2(b);
7 return sign(cross(a, b)) > 0;
8 }

```

4.8 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(l + 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    vector<P> p;
16    for(int i = l; i < r; ++i) if(SQ(a[i].x -
17        midx) < d) p.pb(a[i]);
18    REP(i, sz(p)) {
19        for(int j = i + 1; j < sz(p); ++j) {
20            d = min(d, SQ(p[i].x - p[j].x) + SQ(
21                p[i].y - p[j].y));
22            if(SQ(p[i].y - p[j].y) > d) break;
23        }
24    }
25    return d; // 距離平方
26 }

```

5 Graph

5.1 centroid-tree

```

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

```

```

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         ;
25     vis[cd] = true;
26     for(auto v : g[cd]) {
27         if(vis[v]) continue;
28         int child = f(f, v);
29         h[cd].pb(child);
30     }
31     return cd;
32 };
33 int root = build(build, 0);
34 return {root, h};
35 }

```

5.2 chromatic-number

```

1 // vi to(n);
2 // to[u] != 1 << v;
3 // to[v] != 1 << u;
4 int chromatic_number(vi g) {
5     constexpr int MOD = 998244353;
6     int n = SZ(g);
7     vector<int> I(1 << n); I[0] = 1;
8     FOR(s, 1, 1 << n) {
9         int v = __builtin_ctz(s), t = s ^ (1 <<
10             v);
11         I[s] = (I[t] + I[t & ~g[v]]) % MOD;
12     }
13     auto f = I;
14     FOR(k, 1, n + 1) {
15         int sum = 0;
16         REP(s, 1 << n) {
17             if((__builtin_popcount(s) ^ n) & 1)
18                 sum -= f[s];
19             else sum += f[s];
20             sum = ((sum % MOD) + MOD) % MOD;
21             f[s] = 1LL * f[s] * I[s] % MOD;
22         }
23         if(sum != 0) return k;
24     }
25     return 48763;
26 }

```

5.3 count-bridge-online

```

1 vector<int> par, dsu_2ecc, dsu_cc,
2     dsu_cc_size, last_visit;
3 int bridges, lca_iteration;
4 void init(int n) {
5     par.assign(n, -1);
6     dsu_2ecc.resize(n);
7     dsu_cc.resize(n);
8     dsu_cc_size.assign(n, 1);
9     lca_iteration = 0;

```

```

9     last_visit.assign(n, 0);
10     iota(ALL(dsu_cc), 0);
11     dsu_2ecc = dsu_cc;
12     bridges = 0;
13 }
14 int find_2ecc(int v) {
15     if(v == -1) return -1;
16     return dsu_2ecc[v] == v ? v : dsu_2ecc[v]
17         = find_2ecc(dsu_2ecc[v]);
18 }
19 int find_cc(int v) {
20     v = find_2ecc(v);
21     return dsu_cc[v] == v ? v : dsu_cc[v] =
22         find_cc(dsu_cc[v]);
23 }
24 void make_root(int v) {
25     v = find_2ecc(v);
26     int root = v, child = -1;
27     while(v != -1) {
28         int p = find_2ecc(par[v]);
29         par[v] = child;
30         dsu_cc[v] = root;
31         child = v;
32         v = p;
33     }
34     dsu_cc_size[root] = dsu_cc_size[child];
35 }
36 void merge_path(int a, int b) {
37     ++lca_iteration;
38     vector<int> path_a, path_b;
39     int lca = -1;
40     while(lca == -1) {
41         if(a != -1) {
42             a = find_2ecc(a);
43             path_a.push_back(a);
44             if(last_visit[a] ==
45                 lca_iteration){
46                 lca = a;
47                 break;
48             }
49             last_visit[a] = lca_iteration;
50             a = par[a];
51         }
52         if(b != -1) {
53             b = find_2ecc(b);
54             path_b.push_back(b);
55             if(last_visit[b] ==
56                 lca_iteration){
57                 lca = b;
58                 break;
59             }
60             last_visit[b] = lca_iteration;
61             b = par[b];
62         }
63     }
64     for(int v : path_a) {
65         dsu_2ecc[v] = lca;
66         if(v == lca) break;
67         --bridges;
68     }
69     for(int v : path_b) {
70         dsu_2ecc[v] = lca;
71         if(v == lca) break;
72         --bridges;
73     }

```

```

74 }
75 void add_edge(int a, int b) {
76     a = find_2ecc(a), b = find_2ecc(b);
77     if(a == b) return;
78     int ca = find_cc(a), cb = find_cc(b);
79     if(ca != cb) {
80         ++bridges;
81         if(dsu_cc_size[ca] > dsu_cc_size[cb]
82             ) swap(a, b), swap(ca, cb);
83         make_root(a);
84         par[a] = dsu_cc[a] = b;
85         dsu_cc_size[cb] += dsu_cc_size[a];
86     } else merge_path(a, b);
87 }

```

5.4 2-SAT

```

1 struct two_sat {
2     int n; SCC g;
3     vector<bool> ans;
4     two_sat(int _n) : n(_n), g(_n * 2) {}
5     void add_or(int u, bool x, int v, bool y)
6         {
7         g.add_edge(2 * u + !x, 2 * v + y);
8         g.add_edge(2 * v + !y, 2 * u + x);
9     }
10    bool solve() {
11        ans.resize(n);
12        auto id = g.solve();
13        REP(i, n) {
14            if(id[2 * i] == id[2 * i + 1]) return
15                false;
16            ans[i] = (id[2 * i] < id[2 * i + 1]);
17        }
18        return true;
19    }
20 }

```

5.5 lowlink

```

1 struct lowlink {
2     int n, cnt = 0, tecc_cnt = 0, tvcc_cnt =
3         0;
4     vector<vector<pii>> g;
5     vector<pii> edges;
6     vi roots, id, low, tecc_id, tvcc_id;
7     vector<bool> is_bridge, is_cut,
8         is_tree_edge;
9     lowlink(int _n) : n(_n), g(_n), is_cut(_n,
10         false), id(_n, -1), low(_n, -1) {}
11    void add_edge(int u, int v) {
12        g[u].eb(v, SZ(edges));
13        g[v].eb(u, SZ(edges));
14        edges.eb(u, v);
15        is_bridge.pb(false);
16        is_tree_edge.pb(false);
17        tvcc_id.pb(-1);
18    }
19    void dfs(int u, int peid = -1) {
20        static vi stk;

```

```

21        static int rid;
22        if(peid < 0) rid = cnt;
23        if(peid == -1) roots.pb(u);
24        id[u] = low[u] = cnt++;
25        for(auto [v, eid] : g[u]) {
26            if(eid == peid) continue;
27            if(id[v] < id[u]) stk.pb(eid);
28            if(id[v] >= 0) low[u] = min(low[u], id
29                [v]);
30        }
31        else {
32            is_tree_edge[eid] = true;
33            dfs(v, eid);
34            low[u] = min(low[u], low[v]);
35            if((id[u] == rid && id[v] != rid +
36                1) || (id[u] != rid && low[v] >=
37                    id[u])) {
38                is_cut[u] = true;
39            }
40            if(low[v] >= id[u]) {
41                while(true) {
42                    int e = stk.back();
43                    stk.pop_back();
44                    tvcc_id[e] = tvcc_cnt;
45                    if(e == eid) break;
46                }
47                tvcc_cnt++;
48            }
49        }
50    }
51    void build() {
52        REP(i, n) if(id[i] < 0) dfs(i);
53        REP(i, SZ(edges)) {
54            auto [u, v] = edges[i];
55            if(id[u] > id[v]) swap(u, v);
56            is_bridge[i] = (id[u] < low[v]);
57        }
58    }
59    vector<vi> two_ecc() { // 邊雙
60        tecc_cnt = 0;
61        tecc_id.assign(n, -1);
62        vi stk;
63        REP(i, n) {
64            if(tecc_id[i] != -1) continue;
65            tecc_id[i] = tecc_cnt;
66            stk.pb(i);
67            while(SZ(stk)) {
68                int u = stk.back(); stk.pop_back();
69                for(auto [v, eid] : g[u]) {
70                    if(tecc_id[v] >= 0 || is_bridge[
71                        eid]) continue;
72                    tecc_id[v] = tecc_cnt;
73                    stk.pb(v);
74                }
75                tecc_cnt++;
76            }
77        }
78        vector<vi> comp(tecc_cnt);
79        REP(i, n) comp[tecc_id[i]].pb(i);
80        return comp;
81    }
82    vector<vi> bcc_vertices() { // 點雙
83        vector<vi> comp(tvcc_cnt);
84        REP(i, SZ(edges)) {
85            comp[tvcc_id[i]].pb(edges[i].first);

```

```

79     comp[tvcc_id[i]].pb(edges[i].second);
80 }
81 for(auto& v : comp) {
82     sort(ALL(v));
83     v.erase(unique(ALL(v)), v.end());
84 }
85 REP(i, n) if(!SZ(g[i])) comp.pb({i});
86 return comp;
87 }
88 vector<vi> bcc_edges() {
89     vector<vi> ret(tvcc_cnt);
90     REP(i, SZ(edges)) ret[tvcc_id[i]].pb(i);
91     return ret;
92 }
93 };

```

5.6 manhattan-mst

```

1 template<class T> // [w, u, v]
2 vector<tuple<T, int, int>> manhattan_mst(
3     vector<T> xs, vector<T> ys) {
4     const int n = SZ(xs);
5     vi idx(n); iota(ALL(idx), 0);
6     vector<tuple<T, int, int>> ret;
7     REP(s, 2) {
8         REP(t, 2) {
9             auto cmp = [&](int i, int j) { return
10                 xs[i] + ys[i] < xs[j] + ys[j]; };
11             sort(ALL(idx), cmp);
12             map<T, int> sweep;
13             for(int i : idx) {
14                 for(auto it = sweep.lower_bound(-ys[
15                     i]); it != sweep.end(); it =
16                     sweep.erase(it)) {
17                     int j = it->second;
18                     if(xs[i] - xs[j] < ys[i] - ys[j])
19                         break;
20                     ret.pb(abs(xs[i] - xs[j]) + abs(ys[
21                         i] - ys[j]), i, j);
22                 }
23                 sweep[-ys[i]] = i;
24             }
25             swap(xs, ys);
26         }
27         for(auto& x : xs) x = -x;
28     }
29     sort(ALL(ret));
30     return ret;
31 }

```

5.7 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         function<void(int)> dfs1 = [&](int u) {
13             id[u] = 1;
14             for(auto v : g[u]) if(id[v] == 0) dfs1
15                 (v);
16             top.pb(u);
17         };
18         REP(v, n) if(id[v] == 0) dfs1(v);
19         fill(ALL(id), -1);
20         function<void(int, int)> dfs2 = [&](int
21             u, int x) {
22             id[u] = x;
23             for(auto v : h[u]) if(id[v] == -1)
24                 dfs2(v, x);
25         };
26         for(int i = n - 1, cnt = 0; i >= 0; --i)
27             {
28                 int u = top[i];
29                 if(id[u] == -1) dfs2(u, cnt++);
30             }
31         return id;
32     };
33 };

```

5.8 HLD

```

1 struct HLD {
2     int n;
3     vector<vi> g;
4     vi siz, par, depth, top, tour, fi, id;
5     sparse_table<pii, min> st;
6     HLD(int n) : n(n), g(n), siz(n), par(
7         n), depth(n), top(n), fi(n), id(n)
8     ) {
9         tour.reserve(n);
10     }
11     void add_edge(int u, int v) {
12         g[u].push_back(v);
13         g[v].push_back(u);
14     }
15     void build(int root = 0) {
16         par[root] = -1;
17         top[root] = root;
18         vector<pii> euler_tour;
19         euler_tour.reserve(2 * n - 1);
20         dfs_sz(root);
21         dfs_link(euler_tour, root);
22         st = sparse_table<pii, min>(euler_tour);
23     }
24     int get_lca(int u, int v) {
25         int L = fi[u], R = fi[v];
26         if(L > R) swap(L, R);
27         return st.prod(L, R).second;
28     }
29     bool is_anc(int u, int v) {
30         return id[u] <= id[v] && id[v] < id[u] +
31             siz[u];
32     }
33     bool on_path(int a, int b, int x) {
34         return is_ancestor(x, a) || is_ancestor
35             (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[
35         get_lca(u, v)];
36 }
37 int kth_anc(int u, int k) {
38     if(depth[u] < k) return -1;
39     int d = depth[u] - k;
40     while(depth[top[u]] > d) u = par[top[u]
41         ];
42     return tour[id[u] + d - depth[u]];
43 }
44 int kth_node_on_path(int a, int b, int k)
45 {
46     int z = get_lca(a, b);
47     int fi = depth[a] - depth[z];
48     int se = depth[b] - depth[z];
49     if(k < 0 || k > fi + se) return -1;
50     if(k < fi) return kth_anc(a, k);
51     return kth_anc(b, fi + se - k);
52 }
53 vector<pii> get_path(int u, int v, bool
54     include_lca = true) {
55     if(u == v && !include_lca) return {};
56     vector<pii> seg;
57     while(top[u] != top[v]) {
58         if(depth[top[u]] > depth[top[v]]) swap
59             (u, v);
60         seg.pb(id[top[v]], id[v]);
61         v = par[top[v]];
62     }
63     if(depth[u] > depth[v]) swap(u, v); // u
64     is_lca
65     if(u != v || include_lca) seg.pb(id[u] +
66         !include_lca, id[v]);
67     return seg;
68 }
69 void dfs_sz(int u) {
70     if(par[u] != -1) g[u].erase(find(ALL(g[u]
71         )), par[u]));
72     siz[u] = 1;
73     for(auto& v : g[u]) {
74         par[v] = u;
75         depth[v] = depth[u] + 1;
76         dfs_sz(v);
77         siz[u] += siz[v];
78         if(siz[v] > siz[g[u][0]]) swap(v, g[u]
79             [0]);
80     }
81 }
82 void dfs_link(vector<pii>& euler_tour, int
83     u) {
84     fi[u] = SZ(euler_tour);
85     id[u] = SZ(tour);
86     euler_tour.pb(depth[u], u);
87     tour.pb(u);
88     for(auto v : g[u]) {
89         top[v] = (v == g[u][0] ? top[u] : v);
90         dfs_link(euler_tour, v);
91         euler_tour.pb(depth[u], u);
92     }
93 }

```

5.9 planar

```

1 struct FringeOpposedSubset {
2     deque<int> left, right;
3     FringeOpposedSubset() = default;
4     FringeOpposedSubset(int h) : left{h},
5         right() {}
6 };
7 template<typename T>
8 void extend(T& a, T& b, bool rev = false) {
9     rev ? a.insert(a.begin(), b.rbegin(), b.
10         rend())
11         : a.insert(a.end(), b.begin(), b.end()
12             );
13 }
14 struct Fringe {
15     deque<FringeOpposedSubset> FOPs;
16     Fringe(int h) : FOPs{h} {}
17     bool operator<(const Fringe& o) const {
18         return std::tie(FOPs.back().left.back(),
19             FOPs.front().left.front()) <
20             std::tie(o.FOPs.back().left.back(),
21                 o.FOPs.front().left.front());
22     }
23     void merge(Fringe& o) {
24         o.merge_t_alike_edges();
25         merge_t_opposite_edges_into(o);
26         if (FOPs.front().right.empty())
27             o.align_duplicates(FOPs.back().left.
28                 front());
29     }
30     void merge_t_alike_edges() {
31         FringeOpposedSubset ans;
32         for (auto& FOP : FOPs) {
33             if (!FOP.right.empty()) throw
34                 runtime_error("Exception");
35             extend(ans.left, FOP.left);
36         }
37         FOPs = {ans};
38     }
39     void merge_t_opposite_edges_into(Fringe& o)
40     {
41         while (FOPs.front().right.empty() &&
42             FOPs.front().left.front() > o.
43             FOPs.front().left.back()) {
44             extend(o.FOPs.front().right, FOPs.
45                 front().left);
46             FOPs.pop_front();
47         }
48     }
49     void align_duplicates(int dfs_h) {
50         if (FOPs.front().left.back() == dfs_h) {
51             FOPs.front().left.pop_back();
52             swap_side();
53         }
54     }
55     void swap_side() {
56         if (FOPs.front().left.empty() ||
57             (!FOPs.front().right.empty() &&
58                 FOPs.front().left.back() > FOPs.
59                 front().right.back())) {

```



```

52 swap(FOPs.front().left, FOPs.front().
53       right);
54 }
55 void make_onion_structure(Fringe& o) {
56     auto low = &FOPs.front().left, high = &
57       FOPs.front().right;
58     if (FOPs.front().left.front() >= FOPs.
59       front().right.front())
60         swap(low, high);
61     if (o.FOPs.front().left.back() < low->
62       front())
63         throw runtime_error("Exception");
64     if (o.FOPs.front().left.back() < high->
65       front()) {
66         extend(*low, o.FOPs.front().left, true
67             );
68         extend(*high, o.FOPs.front().right,
69             true);
70         o.FOPs.front().left.clear();
71         o.FOPs.front().right.clear();
72     }
73 }
74 auto lr_condition(int deep) const {
75     bool L = !FOPs.front().left.empty() &&
76       FOPs.front().left.front() >= deep;
77     bool R = !FOPs.front().right.empty() &&
78       FOPs.front().right.front() >= deep;
79     return make_pair(L, R);
80 }
81 void prune(int deep) {
82     auto [left, right] = lr_condition(deep);
83     while (!FOPs.empty() && (left || right))
84     {
85         if (left) FOPs.front().left.pop_front
86           ();
87         if (right) FOPs.front().right.
88           pop_front();
89         if (FOPs.front().left.empty() && FOPs.
90           front().right.empty())
91             FOPs.pop_front();
92         else
93             swap_side();
94         if (!FOPs.empty()) tie(left, right) =
95           lr_condition(deep);
96     }
97 }
98 unique_ptr<Fringe> get_merged_fringe(deque<
99   unique_ptr<Fringe>>& upper) {
100     if (upper.empty()) return nullptr;
101     sort(upper.begin(), upper.end(), [](auto&
102       a, auto& b) { return *a < *b; });
103     for (auto it = next(upper.begin()); it !=
104       upper.end(); ++it)
105         upper.front()->merge(*it);
106     return move(upper.front());
107 }
108 void merge_fringes(vector<deque<unique_ptr<
109   Fringe>>>& fringes, int deep) {
110     auto mf = get_merged_fringe(fringes.back()
111       );
112     fringes.pop_back();
113     if (mf) {
114         mf->prune(deep);

```

```

98     if (mf->FOPs.size()) fringes.back().
99       push_back(move(mf));
100 }
101 struct Edge {
102     int from, to;
103     Edge(int from, int to) : from(from), to(to)
104     {}
105     bool operator==(const Edge& o) const {
106         return from == o.from && to == o.to;
107     }
108 };
109 struct Graph {
110     int n = 0;
111     vector<vector<int>> neighbor;
112     vector<Edge> edges;
113     void add_edge(int from, int to) {
114         if (from == to) return;
115         edges.emplace_back(from, to);
116         edges.emplace_back(to, from);
117     }
118     void build() {
119         sort(edges.begin(), edges.end(), [](
120           const auto& a, const auto& b) {
121             return a.from < b.from || (a.from == b
122               .from && a.to < b.to);
123         });
124         edges.erase(unique(edges.begin(), edges.
125           end()), edges.end());
126         n = 0;
127         for (auto& e : edges) n = max(n, max(e.
128           from, e.to) + 1);
129         neighbor.resize(n);
130         for (auto& e : edges) neighbor[e.from].
131           push_back(e.to);
132     }
133 };
134 Graph g;
135 vector<int> Deep;
136 vector<deque<unique_ptr<Fringe>>> fringes;
137 bool dfs(int x, int parent = -1) {
138     for (int y : g.neighbor[x]) {
139         if (y == parent) continue;
140         if (Deep[y] < 0) { // tree edge
141             fringes.push_back({});
142             Deep[y] = Deep[x] + 1;
143             if (!dfs(y, x)) return false;
144         } else if (Deep[x] > Deep[y]) { //
145             back edge
146             fringes.back().push_back(make_unique<
147               Fringe>(Deep[y]));
148         }
149     }
150     try {
151         if (fringes.size() > 1) merge_fringes(
152           fringes, Deep[parent]);
153     } catch (const exception& e) {
154         return false;
155     }
156     return true;
157 }
158 bool is_planar() {
159     Deep.assign(g.n, -1);
160     for (int i = 0; i < g.n; ++i) {

```

```

154     fringes.clear();
155     Deep[i] = 0;
156     if (!dfs(i)) return false;
157 }
158 return true;
159 }
160 int main() {
161     int n, m, u, v;
162     cin >> n >> m;
163     for (int i = 0; i < m; ++i) {
164         cin >> u >> v;
165         g.add_edge(u, v);
166     }
167     g.build();
168     cout << (is_planar() ? "YES" : "NO") <<
169     endl;
170     return 0;

```

5.10 BCC-tree

```

1 struct BlockCutTree {
2     int n;
3     vector<vi> g;
4     vi dfn, low, stk;
5     int cnt = 0, cur = 0;
6     vector<pii> edges;
7     BlockCutTree(int _n) : n(_n), g(_n), dfn(
8       _n), low(_n) {}
9     void ae(int u, int v) {
10         g[u].pb(v);
11         g[v].pb(u);
12     }
13     void dfs(int x) {
14         stk.pb(x);
15         dfn[x] = low[x] = cur++;
16         for (auto y : g[x]) {
17             if (dfn[y] == -1) {
18                 dfs(y);
19                 low[x] = min(low[x], low[y]);
20                 if (low[y] == dfn[x]) {
21                     int v;
22                     do {
23                         v = stk.back(), stk.pop_back();
24                         edges.eb(n + cnt, v);
25                         while (v != y);
26                     } while (v != y);
27                     edges.eb(x, n + cnt);
28                     cnt++;
29                 } else low[x] = min(low[x], dfn[y]);
30             }
31         }
32     }
33     pair<int, vector<pii>> work() {
34         REP(i, n) {
35             if (dfn[i] == -1) {
36                 stk.clear();
37                 dfs(i);
38             }
39         }
40         return {cnt, edges};

```

5.11 triangle-sum

```

1 // Three vertices a < b < c connected by
2 // three edges {a, b}, {a, c}, {b, c}. Find
3 // xa * xb * xc over all triangles.
4 int triangle_sum(vector<array<int, 2>> edges
5   , vi x) {
6     int n = SZ(x);
7     vi deg(n);
8     vector<vi> g(n);
9     for (auto& [u, v] : edges) {
10         if (u > v) swap(u, v);
11         deg[u]++; deg[v]++;
12     }
13     REP(i, n) g[i].reserve(deg[i]);
14     for (auto [u, v] : edges) {
15         if (deg[u] > deg[v]) swap(u, v);
16         g[u].pb(v);
17     }
18     vi val(n);
19     __int128 ans = 0;
20     REP(a, n) {
21         for (auto b : g[a]) val[b] = x[b];
22         for (auto b : g[a]) {
23             ll tmp = 0;
24             for (auto c : g[b]) tmp += val[c];
25             ans += __int128(tmp) * x[a] * x[b];
26         }
27         for (auto b : g[a]) val[b] = 0;
28     }
29     return ans % mod;

```

6 Math

6.1 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
3   , int b) {
4     ll v = floor_sum(n, m, a, b);
5     int l = -1, r = m - 1;
6     while (r - l > 1) {
7         int k = (l + r) / 2;
8         if (floor_sum(n, m, a, b + (m - 1 - k)) <
9           v + n) r = k;
10         else l = k;
11     }
12     return r;

```

6.2 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 }

```

6.3 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 0;
17     }
18     return 1;
19 }
20 bool is_prime(ll n) {
21     if(n <= 1) return 0;
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 }

```

6.4 Floor-Sum

```

1  // sum_{i=0}^{n-1} floor((ai + b) / c)
2  // in O(a + b + c + n)
3  ll floor_sum(ll n, ll a, ll b, ll c) {
4      assert(0 <= n && n < (1LL << 32));
5      assert(1 <= c && c < (1LL << 32));
6      ull ans = 0;
7      if(a < 0) {
8          ull a2 = (a % c + c) % c;
9          ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / c);
10         a = a2;
11     }
12     if(b < 0) {
13         ull b2 = (b % c + c) % c;
14         ans -= 1ULL * n * ((b2 - b) / c);
15         b = b2;
16     }
17     ull N = n, C = c, A = a, B = b;
18     while(true) {
19         if(A >= C) {
20             ans += N * (N - 1) / 2 * (A / C);
21             A %= C;
22         }
23         if(B >= C) {
24             ans += N * (B / C);
25             B %= C;
26         }
27         ull y_max = A * N + B;
28         if(y_max < C) break;
29         N = y_max / C, B = y_max % C;
30         swap(C, A);
31     }
32     return ans;
33 }

```

6.5 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(i, 2, n + 2) {
9              e = 1LL * e * f % m;
10             if(A.find(e) != A.end()) return n * i - A[e];
11         }
12     }
13     return -1;
14 }

```

6.6 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 }

```

6.7 數字

- Bernoulli numbers

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

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

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

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

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

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

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

- Pentagonal number theorem

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

- Catalan numbers

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

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

- Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

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

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

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

- 次方和

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

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

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

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

General form:

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

6.8 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
100000000000037 2305843009213693951
4611686018427387847 9223372036854775783
18446744073709551557 */

```

6.9 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(j, i, n) if(a[j][i] != 0) {
7             idx = j;
8             break;
9         }
10        if(idx == -1) return 0;
11        if(i != idx) {
12            ret *= T(-1);
13            swap(a[i], a[idx]);
14        }
15        ret *= a[i][i];
16        T inv = T(1) / a[i][i];
17        REP(j, n) a[i][j] *= inv;
18        FOR(j, i + 1, n) {
19            T x = a[j][i];
20            if(x == 0) continue;
21            FOR(k, i, n) {
22                a[j][k] -= a[i][k] * x;
23            }
24        }
25    }
26    return ret;
27 }

```

6.10 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 }

```

6.11 NTT

```

1 const ll mod = (119 << 23) + 1, root = 62;
2 // = 998244353
3 // For p < 2^30 there is also e.g. 5 << 25,
4 // 7 << 26, 479 << 21
5 // and 483 << 21 (same root). The last two
6 // are > 10^9.
7 typedef vector<ll> vl;
8 void ntt(vl &a) {
9     int n = SZ(a), L = 31 - __builtin_clz(n);
10    static vl rt(2, 1);
11    for(static int k = 2, s = 2; k < n; k *= 2, s++) {
12        rt.resize(n);

```

```

10    ll z[] = {1, mod_pow(root, mod >> s, mod
11    )};
12    FOR(i, k, 2 * k) rt[i] = rt[i / 2] * z[i
13    & 1] % mod;
14    }
15    vi rev(n);
16    REP(i, n) rev[i] = (rev[i / 2] | (i & 1)
17    << L) / 2;
18    REP(i, n) if (i < rev[i]) swap(a[i], a[rev
19    [i]]);
20    for(int k = 1; k < n; k *= 2)
21    for(int i = 0; i < n; i += 2 * k) REP(j,
22    k) {
23        ll z = rt[j + k] * a[i + j + k] % mod,
24        &ai = a[i + j];
25        a[i + j + k] = ai - z + (z > ai ? mod
26        : 0);
27        ai += (ai + z >= mod ? z - mod : z);
28    }
29    vl conv(const vl &a, const vl &b) {
30        if(a.empty() || b.empty()) return {};
31        int s = SZ(a) + SZ(b) - 1, B = 32 -
32        __builtin_clz(s), n = 1 << B;
33        ll inv = mod_pow(n, mod - 2, mod);
34        vl L(a), R(b), out(n);
35        L.resize(n), R.resize(n);
36        ntt(L), ntt(R);
37        REP(i, n) out[-i & (n - 1)] = inv * L[i] %
38        mod * R[i] % mod;
39        ntt(out);
40        return {out.begin(), out.begin() + s};
41    }

```

6.12 Poly

```

1 template<int mod>
2 struct Poly {
3     vector<ll> a;
4     Poly() {}
5     Poly(int n) : a(n) {}
6     Poly(const vector<ll> &a) : a(a) {}
7     Poly(const initializer_list<ll> &a) : a(a) {}
8     int size() const { return SZ(a); }
9     void resize(int n) { a.resize(n); }
10    void shrink() {
11        while(size() && a.back() == 0) a.pop_back();
12    }
13    ll at(int idx) const {
14        return idx >= 0 && idx < size() ? a[idx]
15        : 0;
16    }
17    ll& operator[](int idx) { return a[idx]; }
18    friend Poly operator+(const Poly& a, const
19    Poly& b) {
20        Poly c(max(SZ(a), SZ(b)));
21        REP(i, SZ(c)) c[i] = (a.at(i) + b.at(i))
22        % mod;
23        return c;
24    }
25    friend Poly operator-(const Poly& a, const
26    Poly& b) {

```

```

27    Poly c(max(SZ(a), SZ(b)));
28    REP(i, SZ(c)) c[i] = (a.at(i) - b.at(i)
29    + mod) % mod;
30    return c;
31    }
32    friend Poly operator*(Poly a, Poly b) {
33        return Poly(conv(a.a, b.a)); // see NTT.
34        cpp
35    }
36    friend Poly operator*(ll a, Poly b) {
37        REP(i, SZ(b)) (b[i] *= a) %= mod;
38        return b;
39    }
40    friend Poly operator*(Poly a, ll b) {
41        REP(i, SZ(a)) (a[i] *= b) %= mod;
42        return a;
43    }
44    Poly& operator+=(Poly b) { return (*this)
45    = (*this) + b; }
46    Poly& operator-=(Poly b) { return (*this)
47    = (*this) - b; }
48    Poly& operator*=(Poly b) { return (*this)
49    = (*this) * b; }
50    Poly& operator*=(ll b) { return (*this) =
51    (*this) * b; }
52    #define MSZ if(m == -1) m = size();
53    Poly mulxk(int k) const {
54        auto b = a;
55        b.insert(b.begin(), k, 0);
56        return Poly(b);
57    }
58    Poly modxk(int k) const {
59        k = min(k, size());
60        return Poly(vector<ll>(a.begin(), a.
61        begin() + k));
62    }
63    Poly divxk(int k) const {
64        if(size() <= k) return Poly();
65        return Poly(vector<ll>(a.begin() + k, a.
66        end()));
67    }
68    Poly deriv() const {
69        if(!SZ(a)) return Poly();
70        Poly c(size() - 1);
71        REP(i, size() - 1) c[i] = (i + 1LL) * a[
72        i + 1] % mod;
73        return c;
74    }
75    Poly integr() const {
76        Poly c(size() + 1);
77        REP(i, size()) c[i + 1] = a[i] * mod_pow
78        (i + 1, mod - 2, mod) % mod;
79        return c;
80    }
81    Poly inv(int m = -1) const { MSZ;
82        Poly x{mod_pow(a[0], mod - 2, mod)};
83        int k = 1;
84        while(k < m) {
85            k *= 2;
86            x = (x * (Poly{2} - modxk(k) * x)).
87            modxk(k);
88        }
89        return x.modxk(m);
90    }
91    Poly log(int m = -1) const { MSZ;

```

```

92    return (deriv() * inv(m)).integr().modxk
93    (m);
94    }
95    Poly exp(int m = -1) const { MSZ;
96        Poly x{1};
97        int k = 1;
98        while(k < m) {
99            k *= 2;
100           x = (x * (Poly{1} - x.log(k) + modxk(k)
101           )).modxk(k);
102        }
103        return x.modxk(m);
104    }
105    Poly pow(ll k, int m = -1) const { MSZ;
106        if(k == 0) {
107            Poly b(m); b[0] = 1;
108            return b;
109        }
110        int s = 0, sz = size();
111        while(s < sz && a[s] == 0) s++;
112        if(s == sz) return *this;
113        if(m > 0 && s >= (sz + k - 1) / k)
114        return Poly(m);
115        if(s * k >= m) return Poly(m);
116        return (((divxk(s) * mod_pow(a[s], mod
117        - 2, mod)).log(m) * (k % mod)).exp(m)
118        * mod_pow(a[s], k, mod)).mulxk(s *
119        k).modxk(m);
120    }
121    bool has_sqrt() const {
122        if(size() == 0) return true;
123        int x = 0;
124        while(x < size() && a[x] == 0) x++;
125        if(x == size()) return true;
126        if(x % 2 == 1) return false;
127        ll y = a[x];
128        return (y == 0 || mod_pow(y, (mod - 1) / 2,
129        mod) == 1);
130    }
131    Poly sqrt(int m = -1) const { MSZ;
132        if(size() == 0) return Poly();
133        int x = 0;
134        while(x < size() && a[x] == 0) x++;
135        if(x == size()) return Poly(size());
136        Poly f = divxk(x);
137        Poly g{mod_sqrt(f[0], mod)};
138        ll inv2 = mod_pow(2, mod - 2, mod);
139        for(int i = 1; i < m; i *= 2) {
140            g = (g + f.modxk(i * 2) * g.inv(i * 2)
141            ) * inv2;
142        }
143        return g.modxk(m).mulxk(x / 2);
144    }
145    Poly shift(ll c) const {
146        int n = size();
147        Poly b(*this);
148        ll f = 1;
149        REP(i, n) {
150            (b[i] *= f) %= mod;
151            (f *= i + 1) %= mod;
152        }
153        reverse(ALL(b.a));
154        Poly exp_cx(vector<ll>(n, 1));
155        FOR(i, 1, n) exp_cx[i] = exp_cx[i - 1] *
156        c % mod * mod_pow(i, mod - 2, mod) %
157        mod;

```

```

133 b = (b * exp_cx).modxk(n);
134 reverse(ALL(b.a));
135 (f *= mod_pow(n, mod-2, mod)) %= mod;
136 ll z = mod_pow(f, mod-2, mod);
137 IREP(i, n) {
138     (b[i] *= z) %= mod;
139     (z *= i) %= mod;
140 }
141 return b;
142 }
143 Poly mult(Poly b) const {
144     int n = SZ(b);
145     if(!n) return Poly();
146     reverse(ALL(b.a));
147     return ((*this) * b).divxk(n - 1);
148 }
149 vector<ll> eval(vector<ll> x) const {
150     if(size() == 0) return vector<ll>(SZ(x), 0);
151     const int n = max(SZ(x), size());
152     vector<Poly> q(4 * n);
153     vector<ll> ans(SZ(x));
154     x.resize(n);
155     function<void(int, int, int)> build =
156         [&](int p, int l, int r) {
157             if(r - l == 1) q[p] = Poly{1, mod - x[l]};
158             else {
159                 int m = (l + r) / 2;
160                 build(2 * p, l, m), build(2 * p + 1, m, r);
161                 q[p] = q[2 * p] * q[2 * p + 1];
162             }
163         };
164     build(1, 0, n);
165     function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly& num) {
166         if(r - l == 1) {
167             if(l < SZ(ans)) ans[l] = num.at(0);
168         } else {
169             int m = (l + r) / 2;
170             work(2 * p, l, m, num.mult(q[2 * p + 1]).modxk(m - 1));
171             work(2 * p + 1, m, r, num.mult(q[2 * p]).modxk(r - m));
172         }
173     };
174     work(1, 0, n, mult(q[1].inv(n)));
175     return ans;
176 }

```

6.13 Simplex

```

1 /*
2  * Description: Solves a general linear
3  * maximization problem: maximize $c^T x$
4  * subject to $Ax \le b$, $x \ge 0$.
5  * Returns -inf if there is no solution, inf
6  * if there are arbitrarily good
7  * solutions, or the maximum value of $c^T x$
8  * otherwise.

```

```

4  * The input vector is set to an optimal $x$
5  * (or in the unbounded case, an
6  * arbitrary solution fulfilling the
7  * constraints).
8  * Numerical stability is not guaranteed.
9  * For better performance, define
10 * variables such that $x = 0$ is viable.
11 * Usage:
12 * vvd A = {{1,-1}, {-1,1}, {-1,-2}};
13 * vd b = {1,1,-4}, c = {-1,-1}, x;
14 * T val = LPSolver(A, b, c).solve(x);
15 * Time: O(NM * #pivots), where a pivot may
16 * be e.g. an edge relaxation. O(2^n) in
17 * the general case.
18 * 將最小化改成最大化 -> 去除等式 -> 去除大
19 * 於等於 -> 去除自由變數 · 將 x1 用 x1-x3
20 * 取代
21 */
22 typedef double T; // Long double, Rational,
23 double + mod<P>...
24 typedef vector<T> vd;
25 typedef vector<vd> vvd;
26 struct LP {
27     const T eps = 1e-8, inf = 1/.0;
28     #define MP make_pair
29     #define ltj(X) if(s == -1 || MP(X[j],N[j])
30     < MP(X[s],N[s])) s=j
31     int m, n;
32     vi N, B;
33     vvd D;
34     LP(const vvd& A, const vd& b, const vd& c)
35     : m(SZ(b)), n(SZ(c)), N(n+1), B(m), D
36     (m+2, vd(n+2)) {
37         REP(i, m) REP(j, n) D[i][j] = A[i][j];
38         REP(i, m) { B[i] = n+i; D[i][n] = -1; D[
39             i][n+1] = b[i]; }
40         REP(j, n) { N[j] = j; D[m][j] = -c[j]; }
41         N[n] = -1; D[m+1][n] = 1;
42     }
43     void pivot(int r, int s) {
44         T *a = D[r].data(), inv = 1 / a[s];
45         REP(i, m + 2) if(i != r && abs(D[i][s])
46             > eps) {
47             T *b = D[i].data(), inv2 = b[s] * inv;
48             REP(j, n + 2) b[j] -= a[j] * inv2;
49             b[s] = a[s] * inv2;
50         }
51         REP(j, n + 2) if(j != s) D[r][j] *= inv;
52         REP(i, m + 2) if(i != r) D[i][s] *= -inv;
53         D[r][s] = inv;
54         swap(B[r], N[s]);
55     }
56     bool simplex(int phase) {
57         int x = m + phase - 1;
58         while(true) {
59             int s = -1;
60             REP(j, n + 1) if(N[j] != -phase) ltj(D
61                 [x]);
62             if(D[x][s] >= -eps) return true;
63             int r = -1;
64             REP(i, m) {
65                 if(D[i][s] <= eps) continue;

```

```

52         if(r == -1 || MP(D[i][n+1] / D[i][s
53             ], B[i]) < MP(D[r][n+1] / D[r][s
54             ], B[r])) r = i;
55     }
56     if(r == -1) return false;
57     pivot(r, s);
58 }
59 T solve(vd &x) {
60     int r = 0;
61     FOR(i, 1, m) if(D[i][n+1] < D[r][n+1]) r
62         = i;
63     if(D[r][n+1] < -eps) {
64         pivot(r, n);
65         if(!simplex(2) || D[m+1][n+1] < -eps)
66             return -inf;
67         REP(i, m) if(B[i] == -1) {
68             int s = 0;
69             FOR(j, 1, n + 1) ltj(D[i][j]);
70             pivot(i, s);
71         }
72     }
73     bool ok = simplex(1); x = vd(n);
74     REP(i, m) if(B[i] < n) x[B[i]] = D[i][n
75         +1];
76     return ok ? D[m][n+1] : inf;
77 }

```

6.14 Triangle

```

1 // Counts x, y >= 0 such that Ax + By <= C.
2 // Requires A, B > 0. Runs in Log time.
3 // Also representable as sum_{0 <= x <= C /
4 // A} floor((C - Ax) / B + 1).
5 ll count_triangle(ll A, ll B, ll C) {
6     if(C < 0) return 0;
7     if(A < B) swap(A, B);
8     ll m = C / A, k = A / B;
9     ll h = (C - m * A) / B + 1;
10    return m * (m + 1) / 2 * k + (m + 1) * h
11    + count_triangle(B, A - k * B, C -
12    B * (k * m + h));
13 }
14 // Counts 0 <= x < RA, 0 <= y < RB such that
15 // Ax + By <= C. Requires A, B > 0.
16 ll count_triangle_rectangle_intersection(ll
17 A, ll B, ll C, ll RA, ll RB) {
18     if(C < 0 || RA <= 0 || RB <= 0) return
19     0;
20     if(C >= A * (RA - 1) + B * (RB - 1))
21         return RA * RB;
22     return count_triangle(A, B, C) -
23     count_triangle(A, B, C - A * RA) -
24     count_triangle(A, B, C - B * RB);
25 }

```

6.15 Chinese-Remainder

```

1 // (rem, mod) {0, 0} for no solution
2 pair<ll, ll> crt(ll r0, ll m0, ll r1, ll m1)
3 {
4     r0 = (r0 % m0 + m0) % m0;
5     r1 = (r1 % m1 + m1) % m1;
6     if(m0 < m1) swap(r0, r1), swap(m0, m1);
7     if(m0 % m1 == 0) {
8         if(r0 % m1 != r1) return {0, 0};
9     }
10    ll g, im, qq;
11    g = ext_gcd(m0, m1, im, qq);
12    ll u1 = (m1 / g);
13    if((r1 - r0) % g) return {0, 0};
14    ll x = (r1 - r0) / g % u1 * im % u1;
15    r0 += x * m0;
16    m0 *= u1;
17    if(r0 < 0) r0 += m0;
18    return {r0, m0};
19 }

```

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

```

```

40 }
41 f(f, 0, 1);
42 sort(ALL(res));
43 return res;
44 }

```

6.17 Mod-Sqrt

```

1 // return -1 if sqrt DNE
2 ll mod_sqrt(ll a, ll mod) {
3     a %= mod;
4     if(mod == 2 || a < 2) return a;
5     if(mod_pow(a, (mod-1)/2, mod) != 1) return
6         -1;
7     ll b = 1;
8     while(mod_pow(b, (mod-1)/2, mod) == 1) b
9         ++;
10    int m = mod-1, e = __builtin_ctz(m);
11    m >>= e;
12    ll x = mod_pow(a, (m-1)/2, mod);
13    ll y = a * x % mod * x % mod;
14    x = x * a % mod;
15    ll z = mod_pow(b, m, mod);
16    while(y != 1) {
17        int j = 0;
18        ll t = y;
19        while(t != 1) t = t * t % mod, j++;
20        z = mod_pow(z, 1LL << (e - j - 1), mod);
21        x = x*z%mod, z = z*z%mod, y = y*z%mod;
22        e = j;
23    }
24    return min(x, mod-x); // neg is $mod-x$
25 }

```

6.18 Combination

```

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

```

6.19 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 %
5         a) % MOD) % MOD;

```

```

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

```

6.20 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
15         ? -1 : +1); });
16 }
17 template<class T>
18 void xor_transform(vector<T>& a, bool inv) {
19     fwht(a, [&](T& x, T& y) {
20         T z = x + y;
21         y = x - y;
22         x = z;
23     });
24     if(inv) {
25         T z = T(1) / T(SZ(a));
26         for(auto& x : a) x *= z;
27     }
28 }
29 template<class T>
30 vector<T> convolution(vector<T> a, vector<T>
31     b) {
32     assert(SZ(a) == SZ(b));
33     transform(a, false), transform(b, false);
34     REP(i, SZ(a)) a[i] *= b[i];
35     transform(a, true);
36     return a;
37 }
38 template<class T>
39 vector<T> subset_convolution(const vector<T>
40     & f, const vector<T> & g) {
41     assert(SZ(f) == SZ(g));
42     int n = SZ(f);
43     assert(ppc(n) == 1);
44     const int lg = __lg(n);
45     vector<vector<T>> fhat(lg + 1, vector<T>(n
46         )), ghat(fhat);

```

```

43 REP(i, n) fhat[ppc(i)][i] = f[i], ghat[ppc
44     (i)][i] = g[i];
45 REP(i, lg + 1) or_transform(fhat[i], false
46     ), or_transform(ghat[i], false);
47 vector<vector<T>> h(lg + 1, vector<T>(n));
48 REP(m, n) REP(i, lg + 1) REP(j, i + 1) h[i
49     ][m] += fhat[j][m] * ghat[i - j][m];
50 REP(i, lg + 1) or_transform(h[i], true);
51 vector<T> res(n);
52 REP(i, n) res[i] = h[ppc(i)][i];
53 return res;

```

6.21 Aliens

```

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

```

6.22 Berlekamp-Massey

```

1 // - [1, 2, 4, 8, 16] -> (1, [1, -2])
2 // - [1, 1, 2, 3, 5, 8] -> (2, [1, -1, -1])
3 // - [0, 0, 0, 0, 1] -> (5, [1, 0, 0, 0, 0,
4     998244352]) (mod 998244353)
5 // - [] -> (0, [1])
6 // - [0, 0, 0] -> (0, [1])
7 // - [-2] -> (1, [1, 2])
8 template<class T>
9 pair<int, vector<T>> BM(const vector<T> & S)
10 {
11     using poly = vector<T>;
12     int N = SZ(S);
13     poly C_rev{1}, B{1};
14     int L = 0, m = 1;
15     T b = 1;
16     auto adjust = [&](poly C, const poly &B, T
17         d, T b, int m) -> poly {
18         C.resize(max(SZ(C), SZ(B) + m));
19         T a = d / b;
20         REP(i, SZ(B)) C[i + m] -= a * B[i];
21         return C;
22     };
23     REP(n, N) {
24         T d = S[n];
25         REP(i, L) d += C_rev[i + 1] * S[n - 1 -
26             i];
27         if(d == 0) m++;
28         else if (2 * L <= n) {
29             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 }
31 return {L, C_rev};
32 }
33 // Calculate $x^N \bmod f(x)$
34 // Complexity: $O(K^2 \log N)$ ($K$: deg. of
35     $f$)
36 // (4, [1, -1, -1]) -> [2, 3]
37 // (x^4 = (x^2 + x + 2)(x^2 - x - 1) + 3x +
38     2)
39 template<class T>
40 vector<T> monomial_mod_polynomial(long long
41     N, const vector<T> &f_rev) {
42     assert(!f_rev.empty() && f_rev[0] == 1);
43     int K = SZ(f_rev) - 1;
44     if(!K) return {};
45     int D = 64 - __builtin_clzll(N);
46     vector<T> ret(K, 0);
47     ret[0] = 1;
48     auto self_conv = [&](vector<T> x) -> vector
49         <T> {
50         int d = SZ(x);
51         vector<T> ret(d * 2 - 1);
52         REP(i, d) {
53             ret[i * 2] += x[i] * x[i];
54             REP(j, i) ret[i + j] += x[i] * x[j] *
55                 2;
56         }
57         return ret;
58     };
59     for(int d = D; d--;) {
60         ret = self_conv(ret);
61         for(int i = 2 * K - 2; i >= K; i--) {
62             REP(j, K) ret[i - j - 1] -= ret[i] *
63                 f_rev[j + 1];
64         }
65         ret.resize(K);
66         if (N >> d & 1) {
67             vector<T> c(K);
68             c[0] = -ret[K - 1] * f_rev[K];
69             for(int i = 1; i < K; i++) c[i] = ret[
70                 i - 1] - ret[K - 1] * f_rev[K - i
71                 ];
72             ret = c;
73         }
74     }
75     return ret;
76 }
77 // Guess k-th element of the sequence,
78     assuming linear recurrence
79 template<class T>
80 T guess_kth_term(const vector<T> &a, long
81     k) {
82     assert(k >= 0);
83     if(k < 1LL * SZ(a)) return a[k];
84     auto f = BM<T>(a).second;
85     auto g = monomial_mod_polynomial<T>(k, f);
86     T ret = 0;
87     REP(i, SZ(g)) ret += g[i] * a[i];
88     return ret;
89 }

```


6.23 定理

- Cramer's rule

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

- 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 $\frac{1}{n} \sum_{i=0}^{n-1} m^{\gcd(i, n)}$. 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 $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

- Cayley's Formula

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

- Erdős–Gallai theorem

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

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

- Gale–Ryser theorem

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

- Fulkerson–Chen–Anstee theorem

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

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

- Möbius inversion formula

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

- Spherical cap

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

6.24 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 }
```

6.25 生成函數

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

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

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

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

- Special Generating Function

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

6.26 GCD-Convolution

```
1 // 2, 3, 5, 7, ...
2 vector<int> prime_enumerate(int N) {
3   vector<bool> sieve(N / 3 + 1, 1);
4   for(int p = 5, d = 4, i = 1, sqn = sqrt(N);
5     p <= sqn; p += d = 6 - d, i++) {
6     if(!sieve[i]) continue;
7     for(int q = p * p / 3, r = d * p / 3 + (
8       d * p % 3 == 2), s = 2 * p; q < SZ(sieve); q += r = s - r) sieve[q] = 0;
9   }
10  vector<int> ret{2, 3};
11  for(int p = 5, d = 4, i = 1; p <= N; p += d = 6 - d, i++) {
12    if(sieve[i]) {
13      ret.pb(p);
14    }
15  }
16  while(SZ(ret) && ret.back() > N) ret.pop_back();
17  return ret;
18 }
19 struct divisor_transform {
20   template<class T>
21   static void zeta_transform(vector<T>& a) {
22     int n = a.size() - 1;
23     for(auto p : prime_enumerate(n)) {
24       for(int i = 1; i * p <= n; i++) {
25         a[i * p] += a[i];
26       }
27     }
28   }
29   template<class T>
30   static void mobius_transform(vector<T>& a) {
31     int n = a.size() - 1;
32     for(auto p : prime_enumerate(n)) {
33       for(int i = n / p; i > 0; i--) {
34         a[i * p] -= a[i];
35       }
36     }
37   }
38 };
39 struct multiple_transform {
40   template<class T>
41   static void zeta_transform(vector<T>& a) {
42     int n = a.size() - 1;
43     for(auto p : prime_enumerate(n)) {
44       for(int i = n / p; i > 0; i--) {
45         a[i] += a[i * p];
46       }
47     }
48   }
49   template<class T>
50   static void mobius_transform(vector<T>& a) {
51     int n = a.size() - 1;
52     for(auto p : prime_enumerate(n)) {
53       for(int i = 1; i * p <= n; i++) {
54         a[i] -= a[i * p];
55       }
56     }
57   }
58 };
59 // lcm: multiple -> divisor
60 template<class T>
61 vector<T> gcd_convolution(const vector<T>& a, const vector<T>& b) {
62   assert(a.size() == b.size());
63   auto f = a, g = b;
64   multiple_transform::zeta_transform(f);
65   multiple_transform::zeta_transform(g);
66   REP(i, SZ(f)) f[i] *= g[i];
67   multiple_transform::mobius_transform(f);
68   return f;
69 }
```

6.27 歐幾里得類算法

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

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

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

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

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

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

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

6.28 Linear-Sieve

```

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

```

```

21         phi[i * j] = phi[i] * phi[j];
22     }
23 }
24 }
25 }

```

6.29 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 = 0 distinct elements: $B(n)$ = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, ...

7 Misc

7.1 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));

```

7.2 python

```

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

```

7.3 timer

```

1 clock_t T1 = clock();
2 double getCurrentTime() { return (double) (
3     clock() - T1) / CLOCKS_PER_SEC; }

```

7.4 next-combination

```

1 // Example: 1 -> 2, 4 -> 8, 12(1100) ->
2 // 17(1001)
3 ll next_combination(ll comb) {
4     ll x = comb & -comb, y = comb + x;
5     return ((comb & ~y) / x >> 1) | y;
6 }

```

7.5 rng

```

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

```

7.6 gc

```

1 inline char gc() {
2     static const size_t sz = 65536;
3     static char buf[sz];
4     static char *p = buf, *end = buf;
5     if(p == end) end = buf + fread(buf, 1, sz,
6         stdin), p = buf;
7     return *p++;
8 }

```

7.7 rotate90

```

1 vector<vector<T>> rotate90(const vector<
2     vector<T>>& a) {
3     int n = sz(a), m = sz(a[0]);
4     vector<vector<T>> b(m, vector<T>(n));
5     REP(i, n) REP(j, m) b[j][i] = a[i][m - 1
6         - j];
7     return b;
8 }

```

8 String

8.1 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     }
9     return s.substr(i, n - i);
10 }

```

```

7     if(s[i + k] <= s[j + k]) j += k + 1;
8     else i += k + 1;
9     j += (i == j);
10 }
11 int ans = i < n ? i : j;
12 return s.substr(ans, n);
13 }

```

8.2 AC

```

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

```

8.3 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); int j = 0;
5     FOR(i, 1, n) {

```

```

6   if(i <= j + z[j]) z[i] = min(z[i - j], j
7       + z[j] - i);
7   while(i + z[i] < n && a[i + z[i]] == a[z
8       [i]]) z[i]++;
8   if(i + z[i] > j + z[j]) j = i;
9   }
10  return z;
11 }

```

8.4 rolling-hash

```

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

```

8.5 hash61

```

1  const ll M30 = (1LL << 30) - 1;
2  const ll M31 = (1LL << 31) - 1;
3  const ll M61 = (1LL << 61) - 1;
4  ull modulo(ull x){
5      ull xu = x >> 61;
6      ull xd = x & M61;
7      ull res = xu + xd;
8      if(res >= M61) res -= M61;
9      return res;
10 }
11 ull mul(ull a, ull b){
12     ull au = a >> 31, ad = a & M31;
13     ull bu = b >> 31, bd = b & M31;
14     ull mid = au * bd + ad * bu;
15     ull midu = mid >> 30;
16     ull midd = mid & M30;
17     return modulo(au * bu * 2 + midu + (midd
18         << 31) + ad * bd);
19 }

```

8.6 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);

```

8.7 SAIS

```

1  // mississippi
2  // 10 7 4 1 0 9 8 6 3 5 2
3  vi SAIS(string a) {
4      int n = SZ(a), m = *max_element(ALL(a)) +
5          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     IREP(i, n - 1) s[i] = a[i] != a[i + 1] ? a
11         [i] < 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         IREP(i, SZ(ls)) 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         IREP(i, n) 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(i, 1, n) if(ok(i)) lms.pb(i);
35     ind(lms);
36     if(SZ(lms)) {
37         int p = -1, w = 0;
38         for(auto v : sa) if(v && ok(v)) {
39             if(p != -1 && same(p, v)) w--;
40             val[p = v] = w++;
41         }
42         auto b = lms;
43         for(auto& v : b) v = val[v];
44         b = SAIS(b);
45         for(auto& v : b) v = lms[v];
46         ind(b);
47     }
48     return sa;
49 }

```

8.8 KMP

```

1  // abacababa -> [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(i, 1, n) {
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 }

```

8.9 wildcard-pattern-matching

```

1  // 0 <= i <= n - m に対し、s[i, i + m) == t
2  // かどうか
3  // abc*b*a***a
4  // *b*a
5  // 10111011
6  template<class T, class U = modint998244353>
7  vector<bool> wildcard_matching(const vector<
8      T> &s, const vector<T> &t, T wildcard) {
9      const int n = s.size(), m = t.size();
10     vector<U> s1(n), s2(n), s3(n), t1(m), t2
11         (m), t3(m);
12     REP(i, n) {
13         s1[i] = s[i] == wildcard ? 0 : s[i]
14             == 0 ? wildcard : s[i];
15         s2[i] = s1[i] * s1[i], s3[i] = s2[i]
16             * s1[i];
17     }
18     REP(j, m) {
19         t1[j] = t[m - 1 - j] == wildcard ? 0
20             : t[m - 1 - j] == 0 ? wildcard
21             : t[m - 1 - j];
22         t2[j] = t1[j] * t1[j], t3[j] = t2[j]
23             * t1[j];
24     }
25     vector<U> u13 = convolution(s1, t3);
26     vector<U> u22 = convolution(s2, t2);
27     vector<U> u31 = convolution(s3, t1);
28     vector<bool> res(n - m + 1);
29     REP(i, n - m + 1) res[i] = u13[i + m -
30         1] - 2 * u22[i + m - 1] + u31[i + m
31         - 1] == 0;
32     return res;
33 }

```

8.10 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,
4      cnt, first_pos; };
5  Node SA[N]; int sz;

```

```

4  void sa_init() { SA[0].link = -1, SA[0].len
5      = 0, sz = 1; }
6  int sa_extend(int p, int c) {
7      int u = sz++;
8      SA[u].first_pos = SA[u].len = SA[p].len +
9          1;
10     SA[u].cnt = 1;
11     while(p != -1 && SA[p].go[c] == 0) {
12         SA[p].go[c] = u;
13         p = SA[p].link;
14     }
15     if(p == -1) {
16         SA[u].link = 0;
17         return u;
18     }
19     int q = SA[p].go[c];
20     if(SA[p].len + 1 == SA[q].len) {
21         SA[u].link = q;
22         return u;
23     }
24     int x = sz++;
25     SA[x].link = SA[q];
26     SA[x].cnt = 0;
27     SA[x].len = SA[p].len + 1;
28     SA[q].link = SA[u].link = x;
29     while(p != -1 && SA[p].go[c] == q) {
30         SA[p].go[c] = x;
31         p = SA[p].link;
32     }
33     return u;
34 }

```

8.11 manacher

```

1  // length: (z[i] - (i & 1)) / 2 * 2 + (i &
2      1)
3  vi manacher(string t) {
4      string s = "&";
5      for(char c : t) s.pb(c), s.pb('%');
6      int l = 0, r = 0;
7      vi z(SZ(s));
8      REP(i, SZ(s)) {
9          z[i] = r > i ? min(z[2 * l - i], r - i)
10              : 1;
11          while(s[i + z[i]] == s[i - z[i]]) z[i]
12              ++;
13          if(z[i] + i > r) r = z[i] + 1, l = i;
14      }
15     return z;
16 }

```

ACM ICPC Judge Test - NTHU LinkCutTreap

C++ Resource Test

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

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

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

```
62
63 void runtime_error_5() {
64     // maybe illegal instruction
65     int a = 7122, b = 0;
66     cout << (a / b) << endl;
67 }
68
69 void runtime_error_6() {
70     // floating point exception
71     volatile int a = 7122, b = 0;
72     cout << (a / b) << endl;
73 }
74
75 void runtime_error_7() {
76     // call to abort.
77     assert(false);
78 }
79
80 } // namespace system_test
81
82 #include <sys/resource.h>
83 void print_stack_limit() { // only work in
84     Linux
85     struct rlimit l;
86     getrlimit(RLIMIT_STACK, &l);
87     cout << "stack_size = " << l.rlim_cur << "
88         byte" << endl;
89 }
```