

Contents

1 Basic	2	6 Math	22
1.1 Default code	2	6.1 chineseRemainder	22
1.2 test	2	6.2 PiCount	22
1.3 Shell script	2	6.3 numbers	22
1.4 Pragma	2	6.4 Estimation	22
1.5 readchar	2	6.5 floor sum	22
1.6 vimrc	2	6.6 QuadraticResidue	22
1.7 Texas holdem	2	6.7 floor enumeration	23
1.8 black magic	2	6.8 ax+by=gcd	23
2 Graph	3	6.9 cantor expansion	23
2.1 SCC	3	6.10 Generating function	23
2.2 Minimum Arborescence	3	6.11 Fraction	23
2.3 Dominator Tree	3	6.12 Gaussian gcd	23
2.4 MinimumMeanCycle	3	6.13 Theorem	23
2.5 Minimum Clique Cover	4	6.14 Determinant	24
2.6 Maximum Clique Dyn	4	6.15 ModMin	24
2.7 Minimum Arborescence fast	4	6.16 Primes	24
2.8 BCC Vertex	4	6.17 Pollard Rho	24
2.9 NumberOfMaximalClique	5	6.18 Simultaneous Equations	24
2.10 2SAT	5	6.19 Big number	25
2.11 Virtual Tree	5	6.20 Euclidean	25
2.12 Bridge	5	6.21 Miller Rabin	26
2.13 MinimumSteinerTree	5	6.22 Berlekamp-Massey	26
2.14 Vizing	6	6.23 floor ceil	26
2.15 Maximum Clique	6	6.24 fac no p	26
3 Data Structure	6	6.25 DiscreteLog	26
3.1 2D Segment Tree	6	6.26 SimplexConstruction	26
3.2 Sparse table	7	6.27 Simplex Algorithm	26
3.3 Binary Index Tree	7	6.28 SchreierSims	27
3.4 Segment Tree	7	7 Polynomial	27
3.5 BIT kth	7	7.1 Polynomial Operation	27
3.6 Centroid Decomposition	7	7.2 Fast Walsh Transform	28
3.7 DSU	8	7.3 Number Theory Transform	28
3.8 Smart Pointer	8	7.4 Value Poly	29
3.9 IntervalContainer	8	7.5 NTT.2	29
3.10 KDTree useful	8	7.6 Newton	29
3.11 min heap	10	7.7 Fast Fourier Transform	29
3.12 LiChaoST	10	8 Geometry	29
3.13 Treap	10	8.1 PolyUnion	29
3.14 link cut tree	11	8.2 external bisector	30
3.15 Heavy light Decomposition	11	8.3 Convexhull3D	30
3.16 Leftist Tree	11	8.4 Triangulation Voronoi	30
3.17 KDTree	12	8.5 Default code int	30
3.18 Range Chmin Chmax Add Range Sum	12	8.6 Polar Angle Sort	31
3.19 discrete trick	13	8.7 Default code	31
4 Flow Matching	13	8.8 PointInConvex Slow	31
4.1 Maximum Simple Graph Matching	13	8.9 Intersection of polygon and circle	31
4.2 Kuhn Munkres	13	8.10 Tangent line of two circles	31
4.3 Model	14	8.11 CircleCover	31
4.4 MincostMaxflow dijkstra	14	8.12 Heart	32
4.5 isap	15	8.13 PointSegDist	32
4.6 Gomory Hu tree	15	8.14 Minkowski Sum	32
4.7 MincostMaxflow	15	8.15 TangentPointToHull	32
4.8 SW-mincut	15	8.16 Intersection of two circles	32
4.9 Maximum Weight Matching	16	8.17 PointInConvex	32
4.10 Minimum Weight Matching wrong	17	8.18 Intersection of line and circle	32
4.11 Bipartite Matching	17	8.19 Trapezoidalization	32
4.12 BoundedFlow	17	8.20 point in circle	33
4.13 Dinic	18	8.21 PolyCut	33
4.14 MinCostCirculation	18	8.22 minDistOfTwoConvex	33
5 String	18	8.23 DelaunayTriangulation	33
5.1 Smallest Rotation	18	8.24 rotatingSweepLine	34
5.2 Manacher	18	8.25 Intersection of line and convex	34
5.3 De Bruijn sequence	18	8.26 3Dpoint	34
5.4 SAM	19	8.27 HPIGeneralLine	35
5.5 Aho-Corasick Automatan	19	8.28 minMaxEnclosingRectangle	35
5.6 SAIS-old	19	8.29 Half plane intersection	35
5.7 Z-value	20	8.30 Vector in poly	35
5.8 exSAM	20	8.31 DelaunayTriangulation dq	35
5.9 SAIS	20	8.32 Minimum Enclosing Circle	36
5.10 SAIS-C++20	20	8.33 Convex hull	36
5.11 PalTree	21	9 Else	36
5.12 MainLorentz	21	9.1 ManhattanMST	36
5.13 KMP	21	9.2 Mos Algorithm With modification	36
5.14 Suffix Array	21	9.3 BitsetLCS	36
		9.4 BinarySearchOnFraction	36
		9.5 SubsetSum	37
		9.6 DynamicConvexTrick	37
		9.7 DynamicMST	37
		9.8 Matroid	37
		9.9 cyclicLCS	37
		9.10 HilbertCurve	38
		9.11 Mos Algorithm On Tree	38
		9.12 AdaptiveSimpson	38
		9.13 min plus convolution	39
		9.14 cyc tsearch	39
		9.15 All LCS	39
		9.16 NQueens	39
		9.17 Mos Algorithm	39
		9.18 simulated annealing	39
		9.19 DLX	39
		9.20 tree hash	40

9.21 DynamicConvexTrick bb . 40 11 Python 40
 10 JAVA 40
 10.1 Big number 40

1 Basic

1.1 Default code [f616de]

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int, int> pii;
5 typedef pair<ll, ll> pll;
6 #define X first
7 #define Y second
8 #define SZ(a) ((int)a.size())
9 #define ALL(v) v.begin(), v.end()
10 #define pb push_back
```

1.2 test [e6a98d]

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define int ll
5 #define pii pair<int, int>
6 #define X first
7 #define Y second
8 #define F first
9 #define S second
10 #define vi vector<int>
11 #define SZ(a) ((int)a.size())
12 #define ALL(v) v.begin(), v.end()
13 #define pb push_back
14 #define eb emplace_back
15 #define push emplace
16 #define lb(x, v) lower_bound(ALL(x), v)
17 #define ub(x, v) upper_bound(ALL(x), v)
18 #define re(x) reverse(ALL(x))
19 #define uni(x) x.resize(unique(ALL(x)) - x.begin())
20 #define inf 1000000000
21 #define INF 1000000000000000000
22 #define mod 1000000007
23 #define MOD 998244353
24 #define get_bit(x, y) ((x>>y)&1)
25 #define mkp make_pair
26 #define IO ios_base::sync_with_stdio(0); cin.tie(0);
27 void abc() {cerr << endl;}
28 template
29 <typename T, typename ...U> void abc(T a, U ...b) {
30   cerr << a << ' ', abc(b...);
31 }
32 #ifdef debug
33 #define
34   test(args...) abc("[ " + string(#args) + "]", args)
35 #else
36 #define test(args...) void(0)
37 #endif
38 template<class T> bool ckmin
39   (T& a, const T& b) { return b<a ? a=b, 1 : 0; }
40 template<class T> bool ckmax
41   (T& a, const T& b) { return a<b ? a=b, 1 : 0; }
42 inline void solve() {
43 }
44 signed main() {
45   IO;
46   solve();
47 }
```

1.3 Shell script [5dd9f6]

```
1 g++ -O2 -
2   std=c++17 -Dbbq -Wall -Wextra -Wshadow -o $1 $1.cpp
3 chmod +x compile.sh
```

1.4 Pragma [5feeb8]

```
1 #pragma GCC optimize("Ofast,no-stack-protector")
2 #pragma GCC optimize("no-math-errno,unroll-loops")
3 #pragma GCC target("sse,sse2,sse3,ssse3,sse4")
4 #pragma GCC target("popcnt,abm,mmx,avx,arch=skylake")
5 __builtin_ia32_ldmxcsr(__builtin_ia32_stmxcsr())|0x8040
```

1.5 readchar [dacef1]

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

1.6 vimrc [152334]

```
1 "This file should be placed at ~/.vimrc"
2 se nu ai hls et ru ic is sc cul
3 se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
4 syntax on
5 hi cursorline cterm=none ctermbg=89
6 set bg=dark
7 inoremap {<CR> {<CR>}<Esc>ko<tab>
8 "Select region and type :Hash to hash your selection."
9 ca Hash w !cpp -dD -P -fpreprocessed
10 \ | tr -d '[:space:]' \ | md5sum \ | cut -c-6
11 testing
```

1.7 Texas holdem [61f99b]

```
1 char suit[4]={'C','D','H','S'}, ranks[13]={'2',
2   '3','4','5','6','7','8','9','T','J','Q','K','A'};
3 int rk[256];
4 /*
5   for(int i=0;i<13;++i)
6     rk[ranks[i]]=i;
7   for(int i=0;i<4;++i)
8     rk[suit[i]]=i;
9 */
10 struct cards{
11   vector<pii> v;
12   int suit_count[4],hands;
13   void reset(){v.clear(),FILL(suit_count,0),hands=-1;}
14   void insert(char a,char b){//suit,rank
15     ++suit_count[rk[a]];
16     int flag=0;
17     for(auto &i:v)
18       if(i.Y==rk[b])
19         {
20           ++i.X,flag=1;
21           break;
22         }
23     if(!flag) v.pb(pii(1,rk[b]));
24 }
25 void insert(string s){insert(s[0],s[1]);}
26 void ready(){
27   int Straight=0,Flush
28     =(max_element(suit_count,suit_count+4)==5);
29   sort(ALL(v),[](ii a,ii b){return a>b;});
30   if(SZ(v)==5&&v[0].Y==v[1].Y+1&&v[1].Y
31     ==v[2].Y+1&&v[2].Y==v[3].Y+1&&v[3].Y==v[4].Y+1)
32     Straight=1;
33   else if(SZ(v)==5&&v[0].Y==12&&
34     v[1].Y==3&&v[2].Y==2&&v[3].Y==1&&v[4].Y==0)
35     v[0].Y=3,v[1].Y=2,v[2].Y=1,v[3].Y=0,v[4].Y=-1,Straight=1;
36   if(Straight&&Flush) hands=1;
37   else if(v[0].X==4) hands=2;
38   else if(v[0].X==3&&v[1].X==2) hands=3;
39   else if(Flush) hands=4;
40   else if(Straight) hands=5;
41   else if(v[0].X==3) hands=6;
42   else if(v[0].X==2&&v[1].X==2) hands=7;
43   else if(v[0].X==2) hands=8;
44   else hands=9;
45 }
46 bool operator>(const cards &a)const{
47   if(hands==a.hands) return v>a.v;
48   return hands<a.hands;
49 }
50 };
```

1.8 black magic [9612b4]

```
1 #include <ext/pb_ds/priority_queue.hpp>
2 #include <ext/pb_ds/assoc_container.hpp> // rb_tree
3 #include <ext/rope> // rope
4 using namespace __gnu_pbds;
5 using namespace __gnu_cxx; // rope
6 typedef __gnu_pbds::priority_queue<int> heap;
7 int main() {
```

```

8  heap h1, h2; // max heap
9  h1.push(1), h1.push(3), h2.push(2), h2.push(4);
10 h1.join(h2); // h1 = {1, 2, 3, 4}, h2 = {};
11 tree<ll, null_type, less<ll>, rb_tree_tag
    , tree_order_statistics_node_update> st;
12 tree<ll, ll, less<ll>, rb_tree_tag
    , tree_order_statistics_node_update> mp;
13 for (int x : {0, 3, 20, 50}) st.insert(x);
14 assert(st.
    order_of_key(3) == 1 && st.order_of_key(4) == 2);
15 assert(*st.find_by_order
    (2) == 20 && *st.lower_bound(4) == 20);
16 rope<char> *root[10]; // nsqrt(n)
17 root[0] = new rope<char>();
18 root[1] = new rope<char>(*root[0]);
19 // root[1]->insert(pos, 'a');
20 // root[1]->at(pos); 0-base
21 // root[1]->erase(pos, size);
22 }
23 // __int128_t, __float128_t
24 // for (int i = bs._Find_first
    (); i < bs.size(); i = bs._Find_next(i));

```

2 Graph

2.1 SCC [517e91]

```

1 struct SCC { // 0-base
2     int n, dft, nsc;
3     vector<int> low, dfn, bln, instack, stk;
4     vector<vector<int>> G;
5     void dfs(int u) {
6         low[u] = dfn[u] = ++dft;
7         instack[u] = 1, stk.pb(u);
8         for (int v : G[u])
9             if (!dfn[v])
10                dfs(v), low[u] = min(low[u], low[v]);
11            else if (instack[v] && dfn[v] < dfn[u])
12                low[u] = min(low[u], dfn[v]);
13        if (low[u] == dfn[u]) {
14            for (; stk.back() != u; stk.pop_back())
15                bln[stk
16                .back()] = nsc, instack[stk.back()] = 0;
17            instack[u] = 0, bln[u] = nsc++, stk.pop_back();
18        }
19        SCC(int _n): n(_n), dft(), nsc
20        (), low(n), dfn(n), bln(n), instack(n), G(n) {}
21        void add_edge(int u, int v) {
22            G[u].pb(v);
23        }
24        void solve() {
25            for (int i = 0; i < n; ++i)
26                if (!dfn[i]) dfs(i);
27    }; // scc_id(i): bln[i]

```

2.2 Minimum Arborescence [4c8d8d]

```

1 struct zhu_liu { // O(VE)
2     struct edge {
3         int u, v;
4         ll w;
5     };
6     vector<edge> E; // 0-base
7     int pe[N], id[N], vis[N];
8     ll in[N];
9     void init() { E.clear(); }
10    void add_edge(int u, int v, ll w) {
11        if (u != v) E.pb(edge{u, v, w});
12    }
13    ll build(int root, int n) {
14        ll ans = 0;
15        for (;;) {
16            fill_n(in, n, INF);
17            for (int i = 0; i < SZ(E); ++i)
18                if (E[i].u != E[i].v && E[i].w < in[E[i].v])
19                    pe[E[i].v] = i, in[E[i].v] = E[i].w;
20            for (int u = 0; u < n; ++u) // no solution
21                if (u != root && in[u] == INF) return -INF;
22            int cntnode = 0;
23            fill_n(id, n, -1), fill_n(vis, n, -1);
24            for (int u = 0; u < n; ++u) {
25                if (u != root) ans += in[u];
26                int v = u;
27                while (vis[v] != u && !~id[v] && v != root)
28                    vis[v] = u, v = E[pe[v]].u;

```

```

29        if (v != root && !~id[v]) {
30            for (int x = E[pe[v]].u; x != v;
31                x = E[pe[x]].u)
32                id[x] = cntnode;
33            id[v] = cntnode++;
34        }
35    }
36    if (!cntnode) break; // no cycle
37    for (int u = 0; u < n; ++u)
38        if (!~id[u]) id[u] = cntnode++;
39    for (int i = 0; i < SZ(E); ++i) {
40        int v = E[i].v;
41        E[i].u = id[E[i].u], E[i].v = id[E[i].v];
42        if (E[i].u != E[i].v) E[i].w -= in[v];
43    }
44    n = cntnode, root = id[root];
45    return ans;
46 }
47 };
48 };

```

2.3 Dominator Tree [915f9c]

```

1 struct dominator_tree { // 1-base
2     vector<int> G[N], rG[N];
3     int n, pa[N], dfn[N], id[N], Time;
4     int semi[N], idom[N], best[N];
5     vector<int> tree[N]; // dominator_tree
6     void init(int _n) {
7         n = _n;
8         for (int i = 1; i <= n; ++i)
9             G[i].clear(), rG[i].clear();
10    }
11    void add_edge(int u, int v) {
12        G[u].pb(v), rG[v].pb(u);
13    }
14    void dfs(int u) {
15        id[dfn[u] = ++Time] = u;
16        for (auto v : G[u])
17            if (!dfn[v]) dfs(v), pa[dfn[v]] = dfn[u];
18    }
19    int find(int y, int x) {
20        if (y <= x) return y;
21        int tmp = find(pa[y], x);
22        if (semi[best[y]] > semi[best[pa[y]]])
23            best[y] = best[pa[y]];
24        return pa[y] = tmp;
25    }
26    void tarjan(int root) {
27        Time = 0;
28        for (int i = 1; i <= n; ++i) {
29            dfn[i] = idom[i] = 0;
30            tree[i].clear();
31            best[i] = semi[i] = i;
32        }
33        dfs(root);
34        for (int i = Time; i > 1; --i) {
35            int u = id[i];
36            for (auto v : rG[u])
37                if (v = dfn[v]) {
38                    find(v, i);
39                    semi[i] = min(semi[i], semi[best[v]]);
40                }
41            tree[semi[i]].pb(i);
42            for (auto v : tree[pa[i]]) {
43                find(v, pa[i]);
44                idom[v] =
45                    semi[best[v]] == pa[i] ? pa[i] : best[v];
46            }
47            tree[pa[i]].clear();
48        }
49        for (int i = 2; i <= Time; ++i) {
50            if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
51            tree[id[idom[i]]].pb(id[i]);
52        }
53    }
54 };

```

2.4 MinimumMeanCycle [e8ed41]

```

1 ll road[N][N]; // input here
2 struct MinimumMeanCycle {
3     ll dp[N + 5][N], n;
4     pll solve() {
5         ll a = -1, b = -1, L = n + 1;
6         for (int i = 2; i <= L; ++i)
7             for (int k = 0; k < n; ++k)

```

```

8     for (int j = 0; j < n; ++j)
9         dp[i][j] =
10             min(dp[i - 1][k] + road[k][j], dp[i][j]);
11     for (int i = 0; i < n; ++i) {
12         if (dp[L][i] >= INF) continue;
13         ll ta = 0, tb = 1;
14         for (int j = 1; j < n; ++j)
15             if (dp[j][i] < INF &&
16                 ta * (L - j) < (dp[L][i] - dp[j][i]) * tb)
17                 ta = dp[L][i] - dp[j][i], tb = L - j;
18         if (ta == 0) continue;
19         if (a == -1 || a * tb > ta * b) a = ta, b = tb;
20     }
21     if (a != -1) {
22         ll g = __gcd(a, b);
23         return pll(a / g, b / g);
24     }
25     return pll(-1LL, -1LL);
26 }
27 void init(int _n) {
28     n = _n;
29     for (int i = 0; i < n; ++i)
30         for (int j = 0; j < n; ++j) dp[i + 2][j] = INF;
31 }
32 };

```

2.5 Minimum Clique Cover [745700]

```

1 struct Clique_Cover { // 0-base, O(n2^n)
2     int co[1 << N], n, E[N];
3     int dp[1 << N];
4     void init(int _n) {
5         n = _n, fill_n(dp, 1 << n, 0);
6         fill_n(E, n, 0), fill_n(co, 1 << n, 0);
7     }
8     void add_edge(int u, int v) {
9         E[u] |= 1 << v, E[v] |= 1 << u;
10    }
11    int solve() {
12        for (int i = 0; i < n; ++i)
13            co[1 << i] = E[i] | (1 << i);
14        co[0] = (1 << n) - 1;
15        dp[0] = (n & 1) * 2 - 1;
16        for (int i = 1; i < (1 << n); ++i) {
17            int t = i & -i;
18            dp[i] = -dp[i ^ t];
19            co[i] = co[i ^ t] & co[t];
20        }
21        for (int i = 0; i < (1 << n); ++i)
22            co[i] = (co[i] & i) == i;
23        fwt(co, 1 << n, 1);
24        for (int ans = 1; ans < n; ++ans) {
25            int sum = 0; // probabilistic
26            for (int i = 0; i < (1 << n); ++i)
27                sum += (dp[i] * co[i]);
28            if (sum) return ans;
29        }
30        return n;
31    }
32 };

```

2.6 Maximum Clique Dyn [09472e]

```

1 struct MaxClique { // fast when N <= 100
2     bitset<N> G[N], cs[N];
3     int ans, sol[N], q, cur[N], d[N], n;
4     void init(int _n) {
5         n = _n;
6         for (int i = 0; i < n; ++i) G[i].reset();
7     }
8     void add_edge(int u, int v) {
9         G[u][v] = G[v][u] = 1;
10    }
11    void pre_dfs(vector<int> &r, int l, bitset<N> mask) {
12        if (l < 4) {
13            for (int i : r) d[i] = (G[i] & mask).count();
14            sort(ALL(r), [&](int x, int y) { return d[x] > d[y]; });
15        }
16        vector<int> c(SZ(r));
17        int lft = max(ans - q + 1, 1), rgt = 1, tp = 0;
18        cs[1].reset(), cs[2].reset();
19        for (int p : r) {
20            int k = 1;
21            while ((cs[k] & G[p]).any()) ++k;
22            if (k > rgt) cs[++rgt + 1].reset();
23            cs[k][p] = 1;

```

```

24         if (k < lft) r[tp++] = p;
25     }
26     for (int k = lft; k <= rgt; ++k)
27         for (int p = cs[k]._Find_first(); p < N; p = cs[k]._Find_next(p))
28             r[tp] = p, c[tp] = k, ++tp;
29     dfs(r, c, l + 1, mask);
30 }
31 void dfs(vector<
32     int> &r, vector<int> &c, int l, bitset<N> mask) {
33     while (!r.empty()) {
34         int p = r.back();
35         r.pop_back(), mask[p] = 0;
36         if (q + c.back() <= ans) return;
37         cur[q++] = p;
38         vector<int> nr;
39         for (int i : r) if (G[p][i]) nr.pb(i);
40         if (!nr.empty()) pre_dfs(nr, l, mask & G[p]);
41         else if (q > ans) ans = q, copy_n(cur, q, sol);
42         c.pop_back(), --q;
43     }
44 }
45 int solve() {
46     vector<int> r(n);
47     ans = q = 0, iota(ALL(r), 0);
48     pre_dfs(r, 0, bitset<N>(string(n, '1')));
49     return ans;
50 };

```

2.7 Minimum Arborescence fast [48ee1b]

```

1 /* TODO
2 DSU: disjoint set
3 - DSU(n), .boss(x), .Union(x, y)
4 min_heap<
5     T, Info>: min heap for type {T, Info} with lazy tag
6     - .push({w, i}),
7       .top(), .join(heap), .pop(), .empty(), .add_lazy(v)
8 */
9 struct E { int s, t; ll w; }; // 0-base
10 vector<int> dmst(const vector<E> &e, int n, int root) {
11     vector<min_heap<ll, int>> h(n * 2);
12     for (int i = 0; i < SZ(e); ++i)
13         h[e[i].t].push({e[i].w, i});
14     DSU dsu(n * 2);
15     vector<int> v(n * 2, -1), pa(n * 2, -1), r(n * 2);
16     v[root] = n + 1;
17     int pc = n;
18     for (int i = 0; i < n; ++i) if (v[i] == -1) {
19         for (int p = i; v[p] == -1 || v[p] == i; p = dsu.boss(e[r[p]].s)) {
20             if (v[p] == i) {
21                 int q = p; p = pc++;
22                 do {
23                     h[q].add_lazy(-h[q].top().X);
24                     pa[q] = p, dsu.Union(p, q), h[p].join(h[q]);
25                 } while ((q = dsu.boss(e[r[q]].s)) != p);
26             }
27             v[p] = i;
28             while (!h[p].empty() && dsu.boss(e[h[p].top().Y].s) == p)
29                 h[p].pop();
30             if (h[p].empty()) return {}; // no solution
31             r[p] = h[p].top().Y;
32         }
33     }
34     vector<int> ans;
35     for (int i = pc
36         - 1; i >= 0; i--) if (i != root && v[i] != n) {
37         for (int f = e[r[i]].t; ~f && v[f] != n; f = pa[f])
38             v[f] = n;
39         ans.pb(r[i]);
40     }
41     return ans; // default minimize, returns edgeid array
42 } // O(Ef(E)), f(E) from min_heap

```

2.8 BCC Vertex [f56bab]

```

1 struct BCC { // 0-base
2     int n, dft, nbcc;
3     vector<int> low, dfn, bln, stk, is_ap, cir;
4     vector<vector<int>> G, bcc, nG;
5     void make_bcc(int u) {
6         bcc.emplace_back(1, u);
7         for (; stk.back() != u; stk.pop_back())
8             bln[stk.back()] = nbcc, bcc[nbcc].pb(stk.back());

```

```

9     stk.pop_back(), bln[u] = nbcc++;
10 }
11 void dfs(int u, int f) {
12     int child = 0;
13     low[u] = dfn[u] = ++dft, stk.pb(u);
14     for (int v : G[u])
15         if (!dfn[v]) {
16             dfs(v, u), ++child;
17             low[u] = min(low[u], low[v]);
18             if (dfn[u] <= low[v]) {
19                 is_ap[u] = 1, bln[u] = nbcc;
20                 make_bcc(v), bcc.back().pb(u);
21             }
22             } else if (dfn[v] < dfn[u] && v != f)
23                 low[u] = min(low[u], dfn[v]);
24     if (f == -1 && child < 2) is_ap[u] = 0;
25     if (f == -1 && child == 0) make_bcc(u);
26 }
27 BCC(int _n): n(_n), dft(),
28     nbcc(), low(n), dfn(n), bln(n), is_ap(n), G(n) {}
29 void add_edge(int u, int v) {
30     G[u].pb(v), G[v].pb(u);
31 }
32 void solve() {
33     for (int i = 0; i < n; ++i)
34         if (!dfn[i]) dfs(i, -1);
35 }
36 void block_cut_tree() {
37     cir.resize(nbcc);
38     for (int i = 0; i < n; ++i)
39         if (is_ap[i])
40             bln[i] = nbcc++;
41     cir.resize(nbcc, 1), nG.resize(nbcc);
42     for (int i = 0; i < nbcc && !cir[i]; ++i)
43         for (int j : bcc[i])
44             if (is_ap[j])
45                 nG[i].pb(bln[j]), nG[bln[j]].pb(i);
46 } // up to 2 * n - 2 nodes!! bln[i] for id

```

2.9 NumberofMaximalClique [66fef5]

```

1 struct BronKerbosch { // 1-base
2     int n, a[N], g[N][N];
3     int S, all[N][N], some[N][N], none[N][N];
4     void init(int _n) {
5         n = _n;
6         for (int i = 1; i <= n; ++i)
7             for (int j = 1; j <= n; ++j) g[i][j] = 0;
8     }
9     void add_edge(int u, int v) {
10         g[u][v] = g[v][u] = 1;
11     }
12     void dfs(int d, int an, int sn, int nn) {
13         if (S > 1000) return; // pruning
14         if (sn == 0 && nn == 0) ++S;
15         int u = some[d][0];
16         for (int i = 0; i < sn; ++i) {
17             int v = some[d][i];
18             if (g[u][v]) continue;
19             int tsn = 0, tnn = 0;
20             copy_n(all[d], an, all[d + 1]);
21             all[d + 1][an] = v;
22             for (int j = 0; j < sn; ++j)
23                 if (g[v][some[d][j]])
24                     some[d + 1][tsn++] = some[d][j];
25             for (int j = 0; j < nn; ++j)
26                 if (g[v][none[d][j]])
27                     none[d + 1][tnn++] = none[d][j];
28             dfs(d + 1, an + 1, tsn, tnn);
29             some[d][i] = 0, none[d][nn++] = v;
30         }
31     }
32     int solve() {
33         iota(some[0], some[0] + n, 1);
34         S = 0, dfs(0, 0, n, 0);
35         return S;
36     }
37 };

```

2.10 2SAT [d0abc7]

```

1 struct SAT { // 0-base
2     int n;
3     vector<bool> istring;
4     SCC scc;
5     SAT(int _n): n(_n), istring(n + n), scc(n + n) {}

```

```

6     int rv(int a) {
7         return a >= n ? a - n : a + n;
8     }
9     void add_clause(int a, int b) {
10         scc.add_edge(rv(a), b), scc.add_edge(rv(b), a);
11     }
12     bool solve() {
13         scc.solve();
14         for (int i = 0; i < n; ++i) {
15             if (scc.bln[i] == scc.bln[i + n]) return false;
16             istring[i] = scc.bln[i] < scc.bln[i + n];
17             istring[i + n] = !istring[i];
18         }
19         return true;
20     }
21 };

```

2.11 Virtual Tree [551777]

```

1 vector<int> vG[N];
2 int top, st[N];
3
4 void insert(int u) {
5     if (top == -1) return st[++top] = u, void();
6     int p = LCA(st[top], u);
7     if (p == st[top]) return st[++top] = u, void();
8     while (top >= 1 && dep[st[top - 1]] >= dep[p])
9         vG[st[top - 1]].pb(st[top]), --top;
10    if (st[top] != p)
11        vG[p].pb(st[top]), --top, st[++top] = p;
12    st[++top] = u;
13 }
14
15 void reset(int u) {
16     for (int i : vG[u]) reset(i);
17     vG[u].clear();
18 }
19
20 void solve(vector<int> &v) {
21     top = -1;
22     sort(ALL(v),
23         [&](int a, int b) { return dfn[a] < dfn[b]; });
24     for (int i : v) insert(i);
25     while (top > 0) vG[st[top - 1]].pb(st[top]), --top;
26     // do something
27     reset(v[0]);
28 }

```

2.12 Bridge [f72ae7]

```

1 struct ECC { // 0-base
2     int n, dft, ecnt, necc;
3     vector<int> low, dfn, bln, is_bridge, stk;
4     vector<vector<pii>> G;
5     void dfs(int u, int f) {
6         dfn[u] = low[u] = ++dft, stk.pb(u);
7         for (auto [v, e] : G[u])
8             if (!dfn[v])
9                 dfs(v, e), low[u] = min(low[u], low[v]);
10            else if (e != f)
11                low[u] = min(low[u], dfn[v]);
12            if (low[u] == dfn[u]) {
13                if (f != -1) is_bridge[f] = 1;
14                for (; stk.back() != u; stk.pop_back())
15                    bln[stk.back()] = necc;
16                bln[u] = necc++, stk.pop_back();
17            }
18        }
19    ECC(int _n): n(_n), dft(),
20        , ecnt(), necc(), low(n), dfn(n), bln(n), G(n) {}
21    void add_edge(int u, int v) {
22        G[u].pb(pii(v, ecnt)), G[v].pb(pii(u, ecnt++));
23    }
24    void solve() {
25        is_bridge.resize(ecnt);
26        for (int i = 0; i < n; ++i)
27            if (!dfn[i]) dfs(i, -1);
28    } // ecc_id(i): bln[i]

```

2.13 MinimumSteinerTree [e6662f]

```

1 struct SteinerTree { // 0-base
2     int n, dst[N][N], dp[1 << T][N], tdst[N];
3     int vcst[N]; // the cost of vertices
4     void init(int _n) {
5         n = _n;

```

```

6   for (int i = 0; i < n; ++i) {
7       fill_n(dst[i], n, INF);
8       dst[i][i] = vcst[i] = 0;
9   }
10 }
11 void chmin(int &x, int val) {
12     x = min(x, val);
13 }
14 void add_edge(int ui, int vi, int wi) {
15     chmin(dst[ui][vi], wi);
16 }
17 void shortest_path() {
18     for (int k = 0; k < n; ++k)
19         for (int i = 0; i < n; ++i)
20             for (int j = 0; j < n; ++j)
21                 chmin(dst[i][j], dst[i][k] + dst[k][j]);
22 }
23 int solve(const vector<int>& ter) {
24     shortest_path();
25     int t = SZ(ter), full = (1 << t) - 1;
26     for (int i = 0; i <= full; ++i)
27         fill_n(dp[i], n, INF);
28     copy_n(vcst, n, dp[0]);
29     for (int msk = 1; msk <= full; ++msk) {
30         if (!(msk & (msk - 1))) {
31             int who = __lg(msk);
32             for (int i = 0; i < n; ++i)
33                 dp[msk]
34                     [i] = vcst[ter[who]] + dst[ter[who]][i];
35         }
36         for (int i = 0; i < n; ++i)
37             for (int sub = (
38                 msk - 1) & msk; sub; sub = (sub - 1) & msk)
39                 chmin(dp[msk][i],
40                     dp[sub][i] + dp[msk ^ sub][i] - vcst[i]);
41         for (int i = 0; i < n; ++i) {
42             tdst[i] = INF;
43             for (int j = 0; j < n; ++j)
44                 chmin(tdst[i], dp[msk][j] + dst[j][i]);
45         }
46         copy_n(tdst, n, dp[msk]);
47     }
48     return *min_element(dp[full], dp[full] + n);
49 }
50 }; // O(V 3^T + V^2 2^T)

```

2.14 Vizing [a24f68]

```

1 namespace vizing { // returns
2     edge coloring in adjacent matrix G. 1 - based
3 const int N = 105;
4 int C[N][N], G[N][N], X[N], vst[N], n;
5 void init(int _n) { n = _n;
6     for (int i = 0; i <= n; ++i)
7         for (int j = 0; j <= n; ++j)
8             C[i][j] = G[i][j] = 0;
9 }
10 void solve(vector<pii> &E) {
11     auto update = [&](int u) {
12         for (X[u] = 1; C[u][X[u]]; ++X[u]);
13     };
14     auto color = [&](int u, int v, int c) {
15         int p = G[u][v];
16         G[u][v] = G[v][u] = c;
17         C[u][c] = v, C[v][c] = u;
18         C[u][p] = C[v][p] = 0;
19         if (p) X[u] = X[v] = p;
20         else update(u), update(v);
21         return p;
22     };
23     auto flip = [&](int u, int c1, int c2) {
24         int p = C[u][c1];
25         swap(C[u][c1], C[u][c2]);
26         if (p) G[u][p] = G[p][u] = c2;
27         if (!C[u][c1]) X[u] = c1;
28         if (!C[u][c2]) X[u] = c2;
29         return p;
30     };
31     fill_n(X + 1, n, 1);
32     for (int t = 0; t < SZ(E); ++t) {
33         int u = E[t].X, v0 = E[t].Y, v = v0, c0 = X[u], c = c0, d;
34         vector<pii> L;
35         fill_n(vst + 1, n, 0);
36         while (!G[u][v0]) {
37             L.emplace_back(v, d = X[v]);
38             if (!C[v][c]) for (int a = SZ(
39                 L) - 1; a >= 0; --a) c = color(u, L[a].X, c);

```

```

37     else if (!C[u][d]) for (int a = SZ(L)
38         ) - 1; a >= 0; --a) color(u, L[a].X, L[a].Y);
39     else if (vst[d]) break;
40     else vst[d] = 1, v = C[u][d];
41 }
42 if (!G[u][v0]) {
43     for (; v; v = flip(v, c, d), swap(c, d));
44     if (int a; C[u][c0]) {
45         for (
46             a = SZ(L) - 2; a >= 0 && L[a].Y != c; --a);
47         for (; a >= 0; --a) color(u, L[a].X, L[a].Y);
48     }
49     else --t;
50 }
51 } // namespace vizing

```

2.15 Maximum Clique [03ff71]

```

1 struct Maximum_Clique {
2     typedef bitset<MAXN> bst;
3     bst N[MAXN], empty;
4     int p[MAXN], n, ans;
5     void BronKerbosch2(bst R, bst P, bst X) {
6         if (P == empty && X == empty)
7             return ans = max(ans, (int)R.count()), void();
8         bst tmp = P | X;
9         int u;
10        if ((R | P | X).count() <= ans) return;
11        for (int uu = 0; uu < n; ++uu) {
12            u = p[uu];
13            if (tmp[u] == 1) break;
14        }
15        // if (double(clock())/CLOCKS_PER_SEC > .999)
16        // return;
17        bst now2 = P & ~N[u];
18        for (int vv = 0; vv < n; ++vv) {
19            int v = p[vv];
20            if (now2[v] == 1) {
21                R[v] = 1;
22                BronKerbosch2(R, P & N[v], X & N[v]);
23                R[v] = 0, P[v] = 0, X[v] = 1;
24            }
25        }
26    }
27    void init(int _n) {
28        n = _n;
29        for (int i = 0; i < n; ++i) N[i].reset();
30    }
31    void add_edge(int u, int v) {
32        N[u][v] = N[v][u] = 1;
33    }
34    int solve() { // remember srand
35        bst R, P, X;
36        ans = 0, P.flip();
37        for (int i = 0; i < n; ++i) p[i] = i;
38        random_shuffle(p, p + n), BronKerbosch2(R, P, X);
39        return ans;
40    }
41 };

```

3 Data Structure

3.1 2D Segment Tree [6985fc]

```

1 int num[501][501], N, M; // input here
2 struct seg_2D {
3     struct node {
4         int data;
5         node *lc, *rc;
6     } * root;
7     node *merge(node *a, node *b, int l, int r) {
8         node *p = new node;
9         p->data = max(a->data, b->data);
10        if (l == r) return p;
11        int m = l + r >> 1;
12        p->lc = merge(a->lc, b->lc, l, m);
13        p->rc = merge(a->rc, b->rc, m + 1, r);
14        return p;
15    }
16    node *build(int l, int r, int x) {
17        node *p = new node;
18        if (l == r) return p->data = num[x][l], p;
19        int m = l + r >> 1;
20        p->lc = build(l, m, x), p->rc = build(m + 1, r, x);
21        p->data = max(p->lc->data, p->rc->data);

```



```

22     return p;
23 }
24 int query(int L, int R, int l, int r, node *p) {
25     if (L <= l && R >= r) return p->data;
26     int m = l + r >> 1, re = 0;
27     if (L <= m) re = query(L, R, l, m, p->lc);
28     if (R > m)
29         re = max(re, query(L, R, m + 1, r, p->rc));
30     return re;
31 }
32 };
33 struct seg_1D {
34     struct node {
35         seg_2D data;
36         node *lc, *rc;
37     } * root;
38     node *s_build(int l, int r) {
39         node *p = new node;
40         if (l == r)
41             return p->data.root = p->data.build(1, M, l), p;
42         int m = l + r >> 1;
43         p->lc = s_build(l, m), p->rc = s_build(m + 1, r);
44         p->data.root = p->data.merge(
45             p->lc->data.root, p->rc->data.root, 1, M);
46         return p;
47     }
48     int s_query(int L, int R, int l, int r, node *p,
49         int yl, int yr) {
50         if (L <= l && R >= r)
51             return p->data.query(yl, yr, 1, M, p->data.root);
52         int m = l + r >> 1, re = 0;
53         if (L <= m)
54             re = s_query(L, R, l, m, p->lc, yl, yr);
55         if (R > m)
56             re = max(
57                 re, s_query(L, R, m + 1, r, p->rc, yl, yr));
58         return re;
59     }
60     void init() { root = s_build(1, N); }
61     int query(int xl, int xr, int yl, int yr) {
62         return s_query(xl, xr, 1, N, root, yl, yr);
63     }
64 };

```

3.2 Sparse table [cef484]

```

1 struct Sparse_table {
2     int st[__lg(MAXN) + 1][MAXN], n;
3     void init(int _n, int *data) {
4         n = _n;
5         for (int i = 0; i < n; ++i) st[0][i] = data[i];
6         for (int i = 1, t = 2; t < n; t <= 1, i++)
7             for (int j = 0; j + t <= n; j++)
8                 st[i][j]
9                     = max(st[i - 1][j], st[i - 1][j + t / 2]);
10    }
11    int query(int a, int b) {
12        int t = __lg(b - a + 1);
13        return max(st[t][a], st[t][b - (1 << t) + 1]);
14    }
15 };

```

3.3 Binary Index Tree [18be78]

```

1 struct Binary_Index_Tree {
2     int bit[MAXN + 1], lazy[MAXN + 1], n;
3     int lb(int x) { return x & -x; }
4     void init(int _n, int *data) {
5         n = _n;
6         for (int i = 1, t; i <= n; ++i) {
7             bit[i] = data[i], lazy[i] = 0, t = i - lb(i);
8             for (int j = i - 1; j > t; j -= lb(j))
9                 bit[i] += bit[j];
10        }
11    }
12    void suf_modify(int x, int v) {
13        for (int t = x; t; t -= lb(t)) lazy[t] += v;
14        for (int t = x + lb(x); t && t <= n; t += lb(t))
15            bit[t] += v * (x - t + lb(t));
16    }
17    void modify(int x, int v) {
18        for (; x; x -= lb(x)) bit[x] += v;
19    }
20    int query(int x) {
21        int re = 0;
22        for (int t = x; t; t -= lb(t))
23            re += lazy[t] * lb(t) + bit[t];

```

```

24     for (int t = x + lb(x); t && t <= n; t += lb(t))
25         re += lazy[t] * (x - t + lb(t));
26     return re;
27 }
28 };

```

3.4 Segment Tree [0f243e]

```

1 struct Segment_Tree {
2     struct node {
3         int data, lazy;
4         node *l, *r;
5         node() : data(0), lazy(0), l(0), r(0) {}
6         void up() {
7             if (l) data = max(l->data, r->data);
8         }
9         void down() {
10            if (l) {
11                l->data += lazy, l->lazy += lazy;
12                r->data += lazy, r->lazy += lazy;
13            }
14            lazy = 0;
15        }
16    } * root;
17    int l, r;
18    node *build(int l, int r, int *data) {
19        node *p = new node();
20        if (l == r) return p->data = data[l], p;
21        int m = (l + r) / 2;
22        p->l = build(l, m, data),
23        p->r = build(m + 1, r, data);
24        return p->up(), p;
25    }
26    void s_modify(
27        int L, int R, int l, int r, node *p, int x) {
28        if (r < L || l > R) return;
29        p->down();
30        if (L <= l && R >= r)
31            return p->data += x, p->lazy += x, void();
32        int m = (l + r) / 2;
33        s_modify(L, R, l, m, p->l, x);
34        s_modify(L, R, m + 1, r, p->r, x);
35        p->up();
36    }
37    int s_query(int L, int R, int l, int r, node *p) {
38        p->down();
39        if (L <= l && R >= r) return p->data;
40        int m = (l + r) / 2;
41        if (R <= m) return s_query(L, R, l, m, p->l);
42        if (L > m) return s_query(L, R, m + 1, r, p->r);
43        return max(s_query(L, R, l, m, p->l),
44            s_query(L, R, m + 1, r, p->r));
45    }
46    void init(int L, int R, int *data) {
47        l = L, r = R;
48        root = build(l, r, data);
49    }
50    void modify(int L, int R, int x) {
51        s_modify(L, R, l, r, root, x);
52    }
53    int query(int L, int R) {
54        return s_query(L, R, l, r, root);
55    }
56 };

```

3.5 BIT kth [7de9a0]

```

1 int bit[N + 1]; // N = 2 ^ k
2 int query_kth(int k) {
3     int res = 0;
4     for (int i = N >> 1; i >= 1; i >>= 1)
5         if (bit[res + i] < k)
6             k -= bit[res + i];
7     return res + 1;
8 }

```

3.6 Centroid Decomposition [6971c7]

```

1 struct Cent_Dec { // 1-base
2     vector<pll> G[N];
3     pll info[N]; // store info. of itself
4     pll upinfo[N]; // store info. of climbing up
5     int n, pa[N], layer[N], sz[N], done[N];
6     ll dis[__lg(N) + 1][N];
7     void init(int _n) {
8         n = _n, layer[0] = -1;
9         fill_n(pa + 1, n, 0), fill_n(done + 1, n, 0);

```

```

10   for (int i = 1; i <= n; ++i) G[i].clear();
11   }
12   void add_edge(int a, int b, int w) {
13       G[a].pb(pll(b, w)), G[b].pb(pll(a, w));
14   }
15   void get_cent(
16       int u, int f, int &mx, int &c, int num) {
17       int mxsz = 0;
18       sz[u] = 1;
19       for (pll e : G[u])
20           if (!done[e.X] && e.X != f) {
21               get_cent(e.X, u, mx, c, num);
22               sz[u] += sz[e.X], mxsz = max(mxsz, sz[e.X]);
23           }
24       if (mx > max(mxsz, num - sz[u]))
25           mx = max(mxsz, num - sz[u]), c = u;
26   }
27   void dfs(int u, int f, ll d, int org) {
28       // if required, add self info or climbing info
29       dis[layer[org]][u] = d;
30       for (pll e : G[u])
31           if (!done[e.X] && e.X != f)
32               dfs(e.X, u, d + e.Y, org);
33   }
34   int cut(int u, int f, int num) {
35       int mx = 1e9, c = 0, lc;
36       get_cent(u, f, mx, c, num);
37       done[c] = 1, pa[c] = f, layer[c] = layer[f] + 1;
38       for (pll e : G[c])
39           if (!done[e.X]) {
40               if (sz[e.X] > sz[c])
41                   lc = cut(e.X, c, num - sz[c]);
42               else lc = cut(e.X, c, sz[e.X]);
43               upinfo[lc] = pll(), dfs(e.X, c, e.Y, c);
44           }
45       return done[c] = 0, c;
46   }
47   void build() { cut(1, 0, n); }
48   void modify(int u) {
49       for (int a = u, ly = layer[a]; a;
50           a = pa[a], --ly) {
51           info[a].X += dis[ly][u], ++info[a].Y;
52           if (pa[a])
53               upinfo[a].X += dis[ly - 1][u], ++upinfo[a].Y;
54       }
55   }
56   ll query(int u) {
57       ll rt = 0;
58       for (int a = u, ly = layer[a]; a;
59           a = pa[a], --ly) {
60           rt += info[a].X + info[a].Y * dis[ly][u];
61           if (pa[a])
62               rt -=
63                   upinfo[a].X + upinfo[a].Y * dis[ly - 1][u];
64       }
65       return rt;
66   }
67   };

```

3.7 DSU [e8502d]

```

1   struct DSU {
2       vector<int> arr;
3       DSU(int n = 0): arr(n) {
4           iota(ALL(arr), 0);
5       }
6       int boss(int x) {
7           if (arr[x] == x) return x;
8           return arr[x] = boss(arr[x]);
9       }
10      bool Union(int x, int y) {
11          x = boss(x), y = boss(y);
12          if (x == y) return 0;
13          arr[y] = x;
14          return 1;
15      }
16  };

```

3.8 Smart Pointer [7f0fff]

```

1   #ifndef REFERENCE_POINTER
2   #define REFERENCE_POINTER
3   template <typename T> struct _RefCounter {
4       T data;
5       int ref;
6       _RefCounter(const T &d = 0) : data(d), ref(0) {}
7   };

```

```

8   template <typename T> struct reference_pointer {
9       _RefCounter<T> *p;
10      T *operator->() { return &p->data; }
11      T &operator*() { return p->data; }
12      operator _RefCounter<T> *() { return p; }
13      reference_pointer &operator=(
14          const reference_pointer &t) {
15          if (p && !--p->ref) delete p;
16          p = t.p;
17          p && ++p->ref;
18          return *this;
19      }
20      reference_pointer(_RefCounter<T> *t = 0) : p(t) {
21          p && ++p->ref;
22      }
23      reference_pointer(const reference_pointer &t)
24          : p(t.p) {
25          p && ++p->ref;
26      }
27      ~reference_pointer() {
28          if (p && !--p->ref) delete p;
29      }
30  };
31  template <typename T>
32  inline reference_pointer<T> new_reference(
33      const T &nd) {
34      return reference_pointer<T>(new _RefCounter<T>(nd));
35  }
36  #endif
37  // note:
38  reference_pointer<int> a;
39  a = new_reference(5);
40  a = new_reference<int>(5);
41  a = new_reference((int)5);
42  reference_pointer<int> b = a;
43
44  struct P {
45      int a, b;
46      P(int _a, int _b) : a(_a), b(_b) {}
47  } p(2, 3);
48  reference_pointer<P> a;
49  c = new_reference(P(1, 2));
50  c = new_reference<P>(P(1, 2));
51  c = new_reference(p);

```

3.9 IntervalContainer [dbcccd]

```

1   /* Add and
2      remove intervals from a set of disjoint intervals.
3      * Will merge the added interval with
4      any overlapping intervals in the set when adding.
5      * Intervals are [inclusive, exclusive). */
6   set<pii>::
7   iterator addInterval(set<pii>& is, int L, int R) {
8       if (L == R) return is.end();
9       auto it = is.lower_bound({L, R}), before = it;
10      while (it != is.end() && it->X <= R) {
11          R = max(R, it->Y);
12          before = it = is.erase(it);
13      }
14      if (it != is.begin() && (--it)->Y >= L) {
15          L = min(L, it->X);
16          R = max(R, it->Y);
17          is.erase(it);
18      }
19      return is.insert(before, pii(L, R));
20  }
21  void removeInterval(set<pii>& is, int L, int R) {
22      if (L == R) return;
23      auto it = addInterval(is, L, R);
24      auto r2 = it->Y;
25      if (it->X == L) is.erase(it);
26      else (int&)it->Y = L;
27      if (R != r2) is.emplace(R, r2);
28  }

```

3.10 KDTree useful [22a1d3]

```

1   template <typename T, size_t kd> // kd??????
2   class kd_tree {
3   public:
4       struct point {
5           T d[kd];
6           inline T dist(const point &x) const {
7               T ret = 0;
8               for (size_t i = 0; i < kd; ++i)
9                   ret += std::abs(d[i] - x.d[i]);

```



```

10     return ret;
11 }
12 inline bool operator==(const point &p) {
13     for (size_t i = 0; i < kd; ++i) {
14         if (d[i] != p.d[i]) return 0;
15     }
16     return 1;
17 }
18 inline bool operator<(const point &b) const {
19     return d[0] < b.d[0];
20 }
21 };
22
23 private:
24 struct node {
25     node *l, *r;
26     point pid;
27     int s;
28     node(const point &p) : l(0), r(0), pid(p), s(1) {}
29     inline void up() {
30         s = (l ? l->s : 0) + 1 + (r ? r->s : 0);
31     }
32 } * root;
33 const double alpha, loga;
34 const T INF; //????INF,?????
35 int maxn;
36 struct __cmp {
37     int sort_id;
38     inline bool operator()(
39         const node *x, const node *y) const {
40         return operator()(x->pid, y->pid);
41     }
42     inline bool operator()(
43         const point &x, const point &y) const {
44         if (x.d[sort_id] != y.d[sort_id])
45             return x.d[sort_id] < y.d[sort_id];
46         for (size_t i = 0; i < kd; ++i) {
47             if (x.d[i] != y.d[i]) return x.d[i] < y.d[i];
48         }
49         return 0;
50     }
51 } cmp;
52 void clear(node *o) {
53     if (!o) return;
54     clear(o->l);
55     clear(o->r);
56     delete o;
57 }
58 inline int size(node *o) { return o ? o->s : 0; }
59 std::vector<node *> A;
60 node *build(int k, int l, int r) {
61     if (l > r) return 0;
62     if (k == kd) k = 0;
63     int mid = (l + r) / 2;
64     cmp.sort_id = k;
65     std::nth_element(A.begin() + l, A.begin() + mid,
66         A.begin() + r + 1, cmp);
67     node *ret = A[mid];
68     ret->l = build(k + 1, l, mid - 1);
69     ret->r = build(k + 1, mid + 1, r);
70     ret->up();
71     return ret;
72 }
73 inline bool isbad(node *o) {
74     return size(o->l) > alpha * o->s ||
75         size(o->r) > alpha * o->s;
76 }
77 void flatten(node *u,
78     typename std::vector<node *>::iterator &it) {
79     if (!u) return;
80     flatten(u->l, it);
81     *it = u;
82     flatten(u->r, ++it);
83 }
84 inline void rebuild(node *u, int k) {
85     if ((int)A.size() < u->s) A.resize(u->s);
86     typename std::vector<node *>::iterator it =
87         A.begin();
88     flatten(u, it);
89     u = build(k, 0, u->s - 1);
90 }
91 bool insert(
92     node *u, int k, const point &x, int dep) {
93     if (!u) {
94         u = new node(x);
95         return dep <= 0;
96     }
97     ++u->s;
98     cmp.sort_id = k;
99     if (insert(cmp(x, u->pid) ? u->l : u->r,
100         (k + 1) % kd, x, dep - 1)) {
101         if (!isbad(u)) return 1;
102         rebuild(u, k);
103     }
104     return 0;
105 }
106 node *findmin(node *o, int k) {
107     if (!o) return 0;
108     if (cmp.sort_id == k)
109         return o->l ? findmin(o->l, (k + 1) % kd) : o;
110     node *l = findmin(o->l, (k + 1) % kd);
111     node *r = findmin(o->r, (k + 1) % kd);
112     if (l && !r) return cmp(l, o) ? l : o;
113     if (!l && r) return cmp(r, o) ? r : o;
114     if (!l && !r) return o;
115     if (cmp(l, r)) return cmp(l, o) ? l : o;
116     return cmp(r, o) ? r : o;
117 }
118 bool erase(node *u, int k, const point &x) {
119     if (!u) return 0;
120     if (u->pid == x) {
121         if (u->r)
122             ;
123         else if (u->l) {
124             u->r = u->l;
125             u->l = 0;
126         } else {
127             delete u;
128             u = 0;
129             return 1;
130         }
131         --u->s;
132         cmp.sort_id = k;
133         u->pid = findmin(u->r, (k + 1) % kd)->pid;
134         return erase(u->r, (k + 1) % kd, u->pid);
135     }
136     cmp.sort_id = k;
137     if (erase(cmp(x, u->pid) ? u->l : u->r,
138         (k + 1) % kd, x)) {
139         --u->s;
140         return 1;
141     } else return 0;
142 }
143 inline T heuristic(const T h[]) const {
144     T ret = 0;
145     for (size_t i = 0; i < kd; ++i) ret += h[i];
146     return ret;
147 }
148 int qM;
149 std::priority_queue<std::pair<T, point>> pQ;
150 void nearest(
151     node *u, int k, const point &x, T *h, T &mndist) {
152     if (u == 0 || heuristic(h) >= mndist) return;
153     T dist = u->pid.dist(x), old = h[k];
154     /*mndist=std::min(mndist,dist);*/
155     if (dist < mndist) {
156         pQ.push(std::make_pair(dist, u->pid));
157         if ((int)pQ.size() == qM + 1) {
158             mndist = pQ.top().first, pQ.pop();
159         }
160     }
161     if (x.d[k] < u->pid.d[k]) {
162         nearest(u->l, (k + 1) % kd, x, h, mndist);
163         h[k] = std::abs(x.d[k] - u->pid.d[k]);
164         nearest(u->r, (k + 1) % kd, x, h, mndist);
165     } else {
166         nearest(u->r, (k + 1) % kd, x, h, mndist);
167         h[k] = std::abs(x.d[k] - u->pid.d[k]);
168         nearest(u->l, (k + 1) % kd, x, h, mndist);
169     }
170     h[k] = old;
171 }
172 std::vector<point> in_range;
173 void range(
174     node *u, int k, const point &mi, const point &ma) {
175     if (!u) return;
176     bool is = 1;
177     for (int i = 0; i < kd; ++i)
178         if (u->pid.d[i] < mi.d[i] ||
179             ma.d[i] < u->pid.d[i]) {
180             is = 0;
181             break;

```

```

182     }
183     if (is) in_range.push_back(u->pid);
184     if (mi.d[k] <= u->pid.d[k])
185         range(u->l, (k + 1) % kd, mi, ma);
186     if (ma.d[k] >= u->pid.d[k])
187         range(u->r, (k + 1) % kd, mi, ma);
188 }
189
190 public:
191 kd_tree(const T &INF, double a = 0.75)
192     : root(0), alpha(a), loga(log2(1.0 / a)), INF(INF),
193       maxn(1) {}
194 inline void clear() {
195     clear(root), root = 0, maxn = 1;
196 }
197 inline void build(int n, const point *p) {
198     clear(root), A.resize(maxn = n);
199     for (int i = 0; i < n; ++i) A[i] = new node(p[i]);
200     root = build(0, 0, n - 1);
201 }
202 inline void insert(const point &x) {
203     insert(root, 0, x, std::lg(size(root)) / loga);
204     if (root->s > maxn) maxn = root->s;
205 }
206 inline bool erase(const point &p) {
207     bool d = erase(root, 0, p);
208     if (root && root->s < alpha * maxn) rebuild();
209     return d;
210 }
211 inline void rebuild() {
212     if (root) rebuild(root, 0);
213     maxn = root->s;
214 }
215 inline T nearest(const point &x, int k) {
216     qM = k;
217     T mndist = INF, h[kd] = {};
218     nearest(root, 0, x, h, mndist);
219     mndist = pQ.top().first;
220     pQ = std::priority_queue<std::pair<T, point>>();
221     return mndist; /*???x?k????*/
222 }
223 inline const std::vector<point> &range(
224     const point &mi, const point &ma) {
225     in_range.clear();
226     range(root, 0, mi, ma);
227     return in_range; /*???mi?ma???vector*/
228 }
229 inline int size() { return root ? root->s : 0; }
230 };

```

3.11 min heap [b3de3d]

```

1 template<class T, class Info>
2 struct min_heap {
3     priority_queue<pair<T, Info>, vector
4         <pair<T, Info>>, greater<pair<T, Info>>> pq;
5     T lazy = 0;
6     void push(pair<T, Info> v) {
7         pq.emplace(v.X - lazy, v.Y);
8     }
9     pair<T, Info> top() {
10         return make_pair(pq.top().X + lazy, pq.top().Y);
11     }
12     void join(min_heap &rgt) {
13         if (SZ(pq) < SZ(rgt.pq)) {
14             swap(pq, rgt.pq);
15             swap(lazy, rgt.lazy);
16         }
17         while (!rgt.pq.empty()) {
18             push(rgt.top());
19             rgt.pop();
20         }
21     }
22     void pop() {
23         pq.pop();
24     }
25     bool empty() {
26         return pq.empty();
27     }
28     void add_lazy(T v) {
29         lazy += v;
30     }
31 };

```

3.12 LiChaoST [2c55c3]

```
1 struct L {
```

```

2     ll m, k, id;
3     L() : id(-1) {}
4     L(ll a, ll b, ll c) : m(a), k(b), id(c) {}
5     ll at(ll x) { return m * x + k; }
6 };
7 class LiChao { // maintain max
8 private:
9     int n; vector<L> nodes;
10    void insert(int l, int r, int rt, L ln) {
11        int m = (l + r) >> 1;
12        if (nodes[rt].id == -1)
13            return nodes[rt] = ln, void();
14        bool atLeft = nodes[rt].at(l) < ln.at(l);
15        if (nodes[rt].at(m) < ln.at(m))
16            atLeft ^= 1, swap(nodes[rt], ln);
17        if (r - l == 1) return;
18        if (atLeft) insert(l, m, rt << 1, ln);
19        else insert(m, r, rt << 1 | 1, ln);
20    }
21    ll query(int l, int r, int rt, ll x) {
22        int m = (l + r) >> 1; ll ret = -INF;
23        if (nodes[rt].id != -1) ret = nodes[rt].at(x);
24        if (r - l == 1) return ret;
25        if (x
26            < m) return max(ret, query(l, m, rt << 1, x));
27        return max(ret, query(m, r, rt << 1 | 1, x));
28    }
29 public:
30     LiChao(int n_) : n(n_), nodes(n * 4) {}
31     void insert(L ln) { insert(0, n, 1, ln); }
32     ll query(ll x) { return query(0, n, 1, x); }
33 };

```

3.13 Treap [4a5ee3]

```

1 struct node {
2     int data, sz;
3     node *l, *r;
4     node(int k) : data(k), sz(1), l(0), r(0) {}
5     void up() {
6         sz = 1;
7         if (l) sz += l->sz;
8         if (r) sz += r->sz;
9     }
10    void down() {}
11 };
12 int sz(node *a) { return a ? a->sz : 0; }
13 node *merge(node *a, node *b) {
14     if (!a || !b) return a ? a : b;
15     if (rand() % (sz(a) + sz(b)) < sz(a))
16         return a->down(), a->r = merge(a->r, b), a->up(),
17             a;
18     return b->down(), b->l = merge(a, b->l), b->up(), b;
19 }
20 void split(node *o, node *&a, node *&b, int k) {
21     if (!o) return a = b = 0, void();
22     o->down();
23     if (o->data <= k)
24         a = o, split(o->r, a->r, b, k), a->up();
25     else b = o, split(o->l, a, b->l, k), b->up();
26 }
27 void split2(node *o, node *&a, node *&b, int k) {
28     if (sz(o) <= k) return a = o, b = 0, void();
29     o->down();
30     if (sz(o->l) + 1 <= k)
31         a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
32     else b = o, split2(o->l, a, b->l, k);
33     o->up();
34 }
35 node *kth(node *o, int k) {
36     if (k <= sz(o->l)) return kth(o->l, k);
37     if (k == sz(o->l) + 1) return o;
38     return kth(o->r, k - sz(o->l) - 1);
39 }
40 int Rank(node *o, int key) {
41     if (!o) return 0;
42     if (o->data < key)
43         return sz(o->l) + 1 + Rank(o->r, key);
44     else return Rank(o->l, key);
45 }
46 bool erase(node *o, int k) {
47     if (!o) return 0;
48     if (o->data == k) {
49         node *t = o;
50         o->down(), o = merge(o->l, o->r);
51         delete t;
52         return 1;

```

```

53 }
54 node *t = k < o->data ? o->l : o->r;
55 return erase(t, k) ? o->up(), 1 : 0;
56 }
57 void insert(node *o, int k) {
58     node *a, *b;
59     split(o, a, b, k),
60     o = merge(a, merge(new node(k), b));
61 }
62 void interval(node *o, int l, int r) {
63     node *a, *b, *c;
64     split2(o, a, b, l - 1), split2(b, b, c, r);
65     // operate
66     o = merge(a, merge(b, c));
67 }

```

3.14 link cut tree [703f02]

```

1 struct Splay { // xor-sum
2     static Splay nil;
3     Splay *ch[2], *f;
4     int val, sum, rev, size;
5     Splay (int
6         _val = 0) : val(_val), sum(_val), rev(0), size(1)
7     { f = ch[0] = ch[1] = &nil; }
8     bool isr()
9     { return f->ch[0] != this && f->ch[1] != this; }
10    int dir()
11    { return f->ch[0] == this ? 0 : 1; }
12    void setCh(Splay *c, int d) {
13        ch[d] = c;
14        if (c != &nil) c->f = this;
15        pull();
16    }
17    void give_tag(int r) {
18        if (r) swap(ch[0], ch[1]), rev ^= 1;
19    }
20    void push() {
21        if (ch[0] != &nil) ch[0]->give_tag(rev);
22        if (ch[1] != &nil) ch[1]->give_tag(rev);
23        rev = 0;
24    }
25    void pull() {
26        // take care of the nil!
27        size = ch[0]->size + ch[1]->size + 1;
28        sum = ch[0]->sum ^ ch[1]->sum ^ val;
29        if (ch[0] != &nil) ch[0]->f = this;
30        if (ch[1] != &nil) ch[1]->f = this;
31    }
32    Splay::nil;
33    Splay *nil = &Splay::nil;
34    void rotate(Splay *x) {
35        Splay *p = x->f;
36        int d = x->dir();
37        if (!p->isr()) p->f->setCh(x, p->dir());
38        else x->f = p->f;
39        p->setCh(x->ch[!d], d);
40        x->setCh(p, !d);
41        p->pull(), x->pull();
42    }
43    void splay(Splay *x) {
44        vector<Splay*> splayVec;
45        for (Splay *q = x;; q = q->f) {
46            splayVec.pb(q);
47            if (q->isr()) break;
48        }
49        reverse(ALL(splayVec));
50        for (auto it : splayVec) it->push();
51        while (!x->isr()) {
52            if (x->f->isr()) rotate(x);
53            else if (x->dir() == x->f->dir())
54                rotate(x->f), rotate(x);
55            else rotate(x), rotate(x);
56        }
57    }
58    Splay* access(Splay *x) {
59        Splay *q = nil;
60        for (; x != nil; x = x->f)
61            splay(x), x->setCh(q, 1), q = x;
62        return q;
63    }
64    void root_path(Splay *x) { access(x), splay(x); }
65    void chroot(Splay *x) {
66        root_path(x), x->give_tag(1);
67        x->push(), x->pull();
68    }
69    void split(Splay *x, Splay *y) {

```

```

69     chroot(x), root_path(y);
70 }
71 void link(Splay *x, Splay *y) {
72     root_path(x), chroot(y);
73     x->setCh(y, 1);
74 }
75 void cut(Splay *x, Splay *y) {
76     split(x, y);
77     if (y->size != 5) return;
78     y->push();
79     y->ch[0] = y->ch[0]->f = nil;
80 }
81 Splay* get_root(Splay *x) {
82     for (root_path(x); x->ch[0] != nil; x = x->ch[0])
83         x->push();
84     splay(x);
85     return x;
86 }
87 bool conn(Splay *x, Splay *y) {
88     return get_root(x) == get_root(y);
89 }
90 Splay* lca(Splay *x, Splay *y) {
91     access(x), root_path(y);
92     if (y->f == nil) return y;
93     return y->f;
94 }
95 void change(Splay *x, int val) {
96     splay(x), x->val = val, x->pull();
97 }
98 int query(Splay *x, Splay *y) {
99     split(x, y);
100    return y->sum;
101 }

```

3.15 Heavy light Decomposition [b91cf9]

```

1 struct Heavy_light_Decomposition { // 1-base
2     int n, ulink[N], deep[N], mxson[N], w[N], pa[N];
3     int t, pl[N], data[N], val[N]; // val: vertex data
4     vector<int> G[N];
5     void init(int _n) {
6         n = _n;
7         for (int i = 1; i <= n; ++i)
8             G[i].clear(), mxson[i] = 0;
9     }
10    void add_edge(int a, int b) {
11        G[a].pb(b), G[b].pb(a);
12    }
13    void dfs(int u, int f, int d) {
14        w[u] = 1, pa[u] = f, deep[u] = d++;
15        for (int &i : G[u])
16            if (i != f) {
17                dfs(i, u, d), w[u] += w[i];
18                if (w[mxson[u]] < w[i]) mxson[u] = i;
19            }
20    }
21    void cut(int u, int link) {
22        data[pl[u] = ++t] = val[u], ulink[u] = link;
23        if (!mxson[u]) return;
24        cut(mxson[u], link);
25        for (int i : G[u])
26            if (i != pa[u] && i != mxson[u])
27                cut(i, i);
28    }
29    void build() { dfs(1, 1, 1), cut(1, 1), /*build*/; }
30    int query(int a, int b) {
31        int ta = ulink[a], tb = ulink[b], res = 0;
32        while (ta != tb) {
33            if (deep[ta] > deep[tb]) swap(ta, tb), swap(a, b);
34            // query(pl[ta], pl[tb])
35            tb = ulink[b = pa[tb]];
36        }
37        if (pl[a] > pl[b]) swap(a, b);
38        // query(pl[a], pl[b])
39    }
40 };

```

3.16 Leftist Tree [2201dc]

```

1 struct node {
2     ll v, data, sz, sum;
3     node *l, *r;
4     node(ll k)
5         : v(0), data(k), sz(1), l(0), r(0), sum(k) {}
6 };
7 ll sz(node *p) { return p ? p->sz : 0; }

```

```

8 ll V(node *p) { return p ? p->v : -1; }
9 ll sum(node *p) { return p ? p->sum : 0; }
10 node *merge(node *a, node *b) {
11     if (!a || !b) return a ? a : b;
12     if (a->data < b->data) swap(a, b);
13     a->r = merge(a->r, b);
14     if (V(a->r) > V(a->l)) swap(a->r, a->l);
15     a->v = V(a->r) + 1, a->sz = sz(a->l) + sz(a->r) + 1;
16     a->sum = sum(a->l) + sum(a->r) + a->data;
17     return a;
18 }
19 void pop(node *&o) {
20     node *tmp = o;
21     o = merge(o->l, o->r);
22     delete tmp;
23 }

```

3.17 KDTree [85f231]

```

1 namespace kdt {
2     int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn],
3         yl[maxn], yr[maxn];
4     point p[maxn];
5     int build(int l, int r, int dep = 0) {
6         if (l == r) return -1;
7         function<bool(const point &, const point &)> f =
8             [dep](const point &a, const point &b) {
9                 if (dep & 1) return a.x < b.x;
10                else return a.y < b.y;
11            };
12         int m = (l + r) >> 1;
13         nth_element(p + l, p + m, p + r, f);
14         xl[m] = xr[m] = p[m].x;
15         yl[m] = yr[m] = p[m].y;
16         lc[m] = build(l, m, dep + 1);
17         if (~lc[m]) {
18             xl[m] = min(xl[m], xl[lc[m]]);
19             xr[m] = max(xr[m], xr[lc[m]]);
20             yl[m] = min(yl[m], yl[lc[m]]);
21             yr[m] = max(yr[m], yr[lc[m]]);
22         }
23         rc[m] = build(m + 1, r, dep + 1);
24         if (~rc[m]) {
25             xl[m] = min(xl[m], xl[rc[m]]);
26             xr[m] = max(xr[m], xr[rc[m]]);
27             yl[m] = min(yl[m], yl[rc[m]]);
28             yr[m] = max(yr[m], yr[rc[m]]);
29         }
30         return m;
31     }
32     bool bound(const point &q, int o, long long d) {
33         double ds = sqrt(d + 1.0);
34         if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
35             q.y < yl[o] - ds || q.y > yr[o] + ds)
36             return false;
37         return true;
38     }
39     long long dist(const point &a, const point &b) {
40         return (a.x - b.x) * 1ll * (a.x - b.x) +
41             (a.y - b.y) * 1ll * (a.y - b.y);
42     }
43     void dfs(
44         const point &q, long long &d, int o, int dep = 0) {
45         if (!bound(q, o, d)) return;
46         long long cd = dist(p[o], q);
47         if (cd != 0) d = min(d, cd);
48         if ((dep & 1) && q.x < p[o].x ||
49             !(dep & 1) && q.y < p[o].y) {
50             if (~lc[o]) dfs(q, d, lc[o], dep + 1);
51             if (~rc[o]) dfs(q, d, rc[o], dep + 1);
52         } else {
53             if (~rc[o]) dfs(q, d, rc[o], dep + 1);
54             if (~lc[o]) dfs(q, d, lc[o], dep + 1);
55         }
56     }
57     void init(const vector<point> &v) {
58         for (int i = 0; i < v.size(); ++i) p[i] = v[i];
59         root = build(0, v.size());
60     }
61     long long nearest(const point &q) {
62         long long res = 1e18;
63         dfs(q, res, root);
64         return res;
65     }
66 } // namespace kdt

```

3.18 RangeChmin Chmax Add Range Sum [cd19b2]

```

1 #include <algorithm>
2 #include <iostream>
3 using namespace std;
4 typedef long long ll;
5
6 const int MAXC = 200005;
7 const ll INF = 1e18;
8
9 struct node {
10     ll sum;
11     ll mx, mxcnt, smx;
12     ll mi, micnt, smi;
13     ll lazymax, lazymin, lazyadd;
14     node(ll k = 0)
15         : sum(k), mxcnt(1), smx(-INF), mi(k),
16           micnt(1), smi(INF), lazymax(-INF), lazymin(INF),
17           lazyadd(0) {}
18     node operator+(const node &a) const {
19         node rt;
20         rt.sum = sum + a.sum;
21         rt.mx = max(mx, a.mx);
22         rt.mi = min(mi, a.mi);
23         if (mx == a.mx) {
24             rt.mxcnt = mxcnt + a.mxcnt;
25             rt.smx = max(smx, a.smx);
26         } else if (mx > a.mx) {
27             rt.mxcnt = mxcnt;
28             rt.smx = max(smx, a.mx);
29         } else {
30             rt.mxcnt = a.mxcnt;
31             rt.smx = max(mx, a.smx);
32         }
33         if (mi == a.mi) {
34             rt.micnt = micnt + a.micnt;
35             rt.smi = min(smi, a.smi);
36         } else if (mi < a.mi) {
37             rt.micnt = micnt;
38             rt.smi = min(smi, a.mi);
39         } else {
40             rt.micnt = a.micnt;
41             rt.smi = min(mi, a.smi);
42         }
43         rt.lazymax = -INF;
44         rt.lazymin = INF;
45         rt.lazyadd = 0;
46         return rt;
47     }
48 } seg[MAXC << 2];
49
50 ll a[MAXC];
51
52 void give_tag_min(int rt, ll t) {
53     if (t >= seg[rt].mx) return;
54     seg[rt].lazymin = t;
55     seg[rt].lazymax = min(seg[rt].lazymax, t);
56     seg[rt].sum -= seg[rt].mxcnt * (seg[rt].mx - t);
57     if (seg[rt].mx == seg[rt].smi) seg[rt].smi = t;
58     if (seg[rt].mx == seg[rt].mi) seg[rt].mi = t;
59     seg[rt].mx = t;
60 }
61
62 void give_tag_max(int rt, ll t) {
63     if (t <= seg[rt].mi) return;
64     seg[rt].lazymax = t;
65     seg[rt].sum += seg[rt].micnt * (t - seg[rt].mi);
66     if (seg[rt].mi == seg[rt].smx) seg[rt].smx = t;
67     if (seg[rt].mi == seg[rt].mx) seg[rt].mx = t;
68     seg[rt].mi = t;
69 }
70
71 void give_tag_add(int l, int r, int rt, ll t) {
72     seg[rt].lazyadd += t;
73     if (seg[rt].lazymax != -INF) seg[rt].lazymax += t;
74     if (seg[rt].lazymin != INF) seg[rt].lazymin += t;
75     seg[rt].mx += t;
76     if (seg[rt].smx != -INF) seg[rt].smx += t;
77     seg[rt].mi += t;
78     if (seg[rt].smi != INF) seg[rt].smi += t;
79     seg[rt].sum += (ll)(r - l + 1) * t;
80 }
81
82 void tag_down(int l, int r, int rt) {
83     if (seg[rt].lazyadd != 0) {
84         int mid = (l + r) >> 1;

```

```

85     give_tag_add(l, mid, rt << 1, seg[rt].lazyadd);
86     give_tag_add(
87         mid + 1, r, rt << 1 | 1, seg[rt].lazyadd);
88     seg[rt].lazyadd = 0;
89 }
90 if (seg[rt].lazymin != INF) {
91     give_tag_min(rt << 1, seg[rt].lazymin);
92     give_tag_min(rt << 1 | 1, seg[rt].lazymin);
93     seg[rt].lazymin = INF;
94 }
95 if (seg[rt].lazymax != -INF) {
96     give_tag_max(rt << 1, seg[rt].lazymax);
97     give_tag_max(rt << 1 | 1, seg[rt].lazymax);
98     seg[rt].lazymax = -INF;
99 }
100 }
101
102 void build(int l, int r, int rt) {
103     if (l == r) return seg[rt] = node(a[l]), void();
104     int mid = (l + r) >> 1;
105     build(l, mid, rt << 1);
106     build(mid + 1, r, rt << 1 | 1);
107     seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
108 }
109
110 void modifymax(
111     int L, int R, int l, int r, int rt, ll t) {
112     if (L <= l && R >= r && t < seg[rt].smi)
113         return give_tag_max(rt, t);
114     if (l != r) tag_down(l, r, rt);
115     int mid = (l + r) >> 1;
116     if (L <= mid) modifymax(L, R, l, mid, rt << 1, t);
117     if (R > mid)
118         modifymax(L, R, mid + 1, r, rt << 1 | 1, t);
119     seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
120 }
121
122 void modifymin(
123     int L, int R, int l, int r, int rt, ll t) {
124     if (L <= l && R >= r && t > seg[rt].smx)
125         return give_tag_min(rt, t);
126     if (l != r) tag_down(l, r, rt);
127     int mid = (l + r) >> 1;
128     if (L <= mid) modifymin(L, R, l, mid, rt << 1, t);
129     if (R > mid)
130         modifymin(L, R, mid + 1, r, rt << 1 | 1, t);
131     seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
132 }
133
134 void modifyadd(
135     int L, int R, int l, int r, int rt, ll t) {
136     if (L <= l && R >= r)
137         return give_tag_add(l, r, rt, t);
138     if (l != r) tag_down(l, r, rt);
139     int mid = (l + r) >> 1;
140     if (L <= mid) modifyadd(L, R, l, mid, rt << 1, t);
141     if (R > mid)
142         modifyadd(L, R, mid + 1, r, rt << 1 | 1, t);
143     seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
144 }
145
146 ll query(int L, int R, int l, int r, int rt) {
147     if (L <= l && R >= r) return seg[rt].sum;
148     if (l != r) tag_down(l, r, rt);
149     int mid = (l + r) >> 1;
150     if (R <= mid) return query(L, R, l, mid, rt << 1);
151     if (L > mid)
152         return query(L, R, mid + 1, r, rt << 1 | 1);
153     return query(L, R, l, mid, rt << 1) +
154         query(L, R, mid + 1, r, rt << 1 | 1);
155 }
156
157 int main() {
158     ios::sync_with_stdio(0), cin.tie(0);
159     int n, m;
160     cin >> n >> m;
161     for (int i = 1; i <= n; ++i) cin >> a[i];
162     build(1, n, 1);
163     while (m--) {
164         int k, x, y;
165         ll t;
166         cin >> k >> x >> y, ++x;
167         if (k == 0) cin >> t, modifymin(x, y, 1, n, 1, t);
168         else if (k == 1)
169             cin >> t, modifymax(x, y, 1, n, 1, t);
170         else if (k == 2)

```

```

171         cin >> t, modifyadd(x, y, 1, n, 1, t);
172     } else cout << query(x, y, 1, n, 1) << "\n";
173 }
174 }

```

3.19 discrete trick [2062d6]

```

1 vector<int> val;
2 // build
3 sort(ALL
4     (val)), val.resize(unique(ALL(val)) - val.begin());
5 // index of x
6 upper_bound(ALL(val), x) - val.begin();
7 // max idx <= x
8 upper_bound(ALL(val), x) - val.begin();
9 // max idx < x
10 lower_bound(ALL(val), x) - val.begin();

```

4 Flow Matching

4.1 Maximum Simple Graph Matching [390d20]

```

1 struct Matching { // 0-base
2     queue<int> q; int n;
3     vector<int> fa, s, vis, pre, match;
4     vector<vector<int>>> G;
5     int Find(int u)
6     { return u == fa[u] ? u : fa[u] = Find(fa[u]); }
7     int LCA(int x, int y) {
8         static int tk = 0; tk++; x = Find(x); y = Find(y);
9         for (; swap(x, y)) if (x != n) {
10             if (vis[x] == tk) return x;
11             vis[x] = tk;
12             x = Find(pre[match[x]]);
13         }
14     }
15     void Blossom(int x, int y, int l) {
16         for (; Find(x) != l; x = pre[y]) {
17             pre[x] = y, y = match[x];
18             if (s[y] == 1) q.push(y), s[y] = 0;
19             for (int z: {x, y}) if (fa[z] == z) fa[z] = l;
20         }
21     }
22     bool Bfs(int r) {
23         iota(ALL(fa), 0); fill(ALL(s), -1);
24         q = queue<int>(); q.push(r); s[r] = 0;
25         for (; !q.empty(); q.pop()) {
26             for (int x = q.front(); int u : G[x])
27                 if (s[u] == -1) {
28                     if (pre[u] = x, s[u] = 1, match[u] == n) {
29                         for (int a = u, b = x, last;
30                             b != n; a = last, b = pre[a])
31                             last = match[b], match[b] = a, match[a] = b;
32                         return true;
33                     }
34                     q.push(match[u]); s[match[u]] = 0;
35                 } else if (!s[u] && Find(u) != Find(x)) {
36                     int l = LCA(u, x);
37                     Blossom(x, u, l); Blossom(u, x, l);
38                 }
39             }
40         return false;
41     }
42     Matching(int _n) : n(_n), fa(n + 1), s(n + 1), vis
43         (n + 1), pre(n + 1, n), match(n + 1, n), G(n) {}
44     void add_edge(int u, int v)
45     { G[u].pb(v), G[v].pb(u); }
46     int solve() {
47         int ans = 0;
48         for (int x = 0; x < n; ++x)
49             if (match[x] == n) ans += Bfs(x);
50         return ans;
51     } // match[x] == n means not matched
52 };

```

4.2 Kuhn Munkres [61bbd0]

```

1 struct KM { // 0-base, maximum matching
2     ll w[N][N], hl[N], hr[N], slk[N];
3     int fl[N], fr[N], pre[N], qu[N], ql, qr, n;
4     bool vl[N], vr[N];
5     void init(int _n) {
6         n = _n;
7         for (int i = 0; i < n; ++i)
8             fill_n(w[i], n, -INF);
9     }

```



```

10 void add_edge(int a, int b, ll wei) {
11     w[a][b] = wei;
12 }
13 bool Check(int x) {
14     if (vl[x] == 1, ~fl[x])
15         return vr[qu[qr++]] = fl[x] = 1;
16     while (~x) swap(x, fr[fl[x] = pre[x]]);
17     return 0;
18 }
19 void bfs(int s) {
20     fill_n(slk,
21         , n, INF), fill_n(vl, n, 0), fill_n(vr, n, 0);
22     ql = qr = 0, qu[qr++] = s, vr[s] = 1;
23     for (ll d;;) {
24         while (ql < qr)
25             for (int x = 0, y = qu[ql++]; x < n; ++x)
26                 if (!vl[x] && slk
27                     [x] >= (d = hl[x] + hr[y] - w[x][y])) {
28                     if (pre[x] = y, d) slk[x] = d;
29                     else if (!Check(x)) return;
30                 }
31         d = INF;
32         for (int x = 0; x < n; ++x)
33             if (!vl[x] && d > slk[x]) d = slk[x];
34         for (int x = 0; x < n; ++x) {
35             if (vl[x]) hl[x] += d;
36             else slk[x] -= d;
37             if (vr[x]) hr[x] -= d;
38         }
39     }
40     for (int x = 0; x < n; ++x)
41         if (!vl[x] && !slk[x] && !Check(x)) return;
42 }
43 ll solve() {
44     fill_n(fl,
45         , n, -1), fill_n(fr, n, -1), fill_n(hr, n, 0);
46     for (int i = 0; i < n; ++i)
47         hl[i] = *max_element(w[i], w[i] + n);
48     for (int i = 0; i < n; ++i) bfs(i);
49     ll res = 0;
50     for (int i = 0; i < n; ++i) res += w[i][fl[i]];
51     return res;
52 }

```

4.3 Model

- Maximum/Minimum flow with lower bound / Circulation problem
 - Construct super source S and sink T .
 - For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
 - For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
 - If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.
 - To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
 - To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
 - The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
 - Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 - DFS from unmatched vertices in X .
 - $x \in X$ is chosen iff x is unvisited.
 - $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
 - Construct super source S and sink T
 - For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$
 - For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
 - For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$
 - For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$
 - Flow from S to T , the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
 - Binary search on answer, suppose we're checking answer T
 - Construct a max flow model, let K be the sum of all weights
 - Connect source $s \rightarrow v, v \in G$ with capacity K
 - For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w

- For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 - T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 - For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
 - Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 - Find the minimum weight perfect matching on G' .
 - Project selection problem
 - If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 - Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 - The mincut is equivalent to the maximum profit of a subset of projects.
 - Dual of minimum cost maximum flow
 - Capacity c_{uv} , Flow f_{uv} , Cost w_{uv} , Required Flow difference for vertex b_u .
 - If all w_{uv} are integers, then optimal solution can happen when all p_u are integers.

$$\min \sum_{uv} w_{uv} f_{uv} \quad \min \sum_u b_u p_u + \sum_{uv} c_{uv} \max(0, p_v - p_u - w_{uv})$$

$$-f_{uv} \geq -c_{uv} \Leftrightarrow \sum_v f_{vu} - \sum_v f_{uv} = -b_u \quad p_u \geq 0$$

4.4 MincostMaxflow dijkstra [94c520]

```

1 struct MinCostMaxFlow { // 0-base
2     struct Edge {
3         ll from, to, cap, flow, cost, rev;
4     } *past[N];
5     vector<Edge> G[N];
6     int inq[N], n, s, t;
7     ll dis[N], up[N], pot[N];
8     bool BellmanFord() {
9         fill_n(dis, n, INF), fill_n(inq, n, 0);
10        queue<int> q;
11        auto relax = [&](int u, ll d, ll cap, Edge *e) {
12            if (cap > 0 && dis[u] > d) {
13                dis[u] = d, up[u] = cap, past[u] = e;
14                if (!inq[u]) inq[u] = 1, q.push(u);
15            }
16        };
17        relax(s, 0, INF, 0);
18        while (!q.empty()) {
19            int u = q.front();
20            q.pop(), inq[u] = 0;
21            for (auto &e : G[u]) {
22                ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
23                relax(e.to, d2, min(up[u], e.cap - e.flow), &e);
24            }
25        }
26        return dis[t] != INF;
27    }
28    bool Dijkstra() {
29        fill_n(dis, n, INF);
30        priority_queue<pll, vector<pll>, greater<pll>> pq;
31        auto relax = [&](int u, ll d, ll cap, Edge *e) {
32            if (cap > 0 && dis[u] > d) {
33                dis[u] = d, up[u] = cap, past[u] = e;
34                pq.push(pll(d, u));
35            }
36        };
37        relax(s, 0, INF, 0);
38        while (!pq.empty()) {
39            auto [d, u] = pq.top();
40            pq.pop();
41            if (dis[u] != d) continue;
42            for (auto &e : G[u]) {
43                ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
44                relax(e.to, d2, min(up[u], e.cap - e.flow), &e);
45            }
46        }
47        return dis[t] != INF;
48    }
49    void solve(int _s
50        , int _t, ll &flow, ll &cost, bool neg = true) {
51        s = _s, t = _t, flow = 0, cost = 0;
52        if (neg) BellmanFord(), copy_n(dis, n, pot);
53        for (; Dijkstra(); copy_n(dis, n, pot)) {
54            for (int
55                i = 0; i < n; ++i) dis[i] += pot[i] - pot[s];
56            flow += up[t], cost += up[t] * dis[t];
57        }
58    }
59 }

```



```

55     for (int i = t; past[i]; i = past[i]->from) {
56         auto &e = *past[i];
57         e.flow += up[t], G[e.to][e.rev].flow -= up[t];
58     }
59 }
60 }
61 void init(int _n) {
62     n = _n, fill_n(pot, n, 0);
63     for (int i = 0; i < n; ++i) G[i].clear();
64 }
65 void add_edge(ll a, ll b, ll cap, ll cost) {
66     G[a].pb(Edge{a, b, cap, 0, cost, SZ(G[b])});
67     G[b].pb(Edge{b, a, 0, 0, -cost, SZ(G[a]) - 1});
68 }
69 };

```

4.5 isap [a2dc77]

```

1 struct Maxflow {
2     static const int MAXV = 20010;
3     static const int INF = 1000000;
4     struct Edge {
5         int v, c, r;
6         Edge(int _v, int _c, int _r)
7             : v(_v), c(_c), r(_r) {}
8     };
9     int s, t;
10    vector<Edge> G[MAXV * 2];
11    int iter[MAXV * 2], d[MAXV * 2], gap[MAXV * 2], tot;
12    void init(int x) {
13        tot = x + 2;
14        s = x + 1, t = x + 2;
15        for (int i = 0; i <= tot; i++) {
16            G[i].clear();
17            iter[i] = d[i] = gap[i] = 0;
18        }
19    }
20    void addEdge(int u, int v, int c) {
21        G[u].push_back(Edge(v, c, SZ(G[v])));
22        G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
23    }
24    int dfs(int p, int flow) {
25        if (p == t) return flow;
26        for (int &i = iter[p]; i < SZ(G[p]); i++) {
27            Edge &e = G[p][i];
28            if (e.c > 0 && d[p] == d[e.v] + 1) {
29                int f = dfs(e.v, min(flow, e.c));
30                if (f) {
31                    e.c -= f;
32                    G[e.v][e.r].c += f;
33                    return f;
34                }
35            }
36        }
37        if (--gap[d[p]] == 0) d[s] = tot;
38        else {
39            d[p]++;
40            iter[p] = 0;
41            ++gap[d[p]];
42        }
43        return 0;
44    }
45    int solve() {
46        int res = 0;
47        gap[0] = tot;
48        for (res = 0; d[s] < tot; res += dfs(s, INF))
49            ;
50        return res;
51    }
52 } flow;

```

4.6 Gomory Hu tree [62c88c]

```

1 MaxFlow Dinic;
2 int g[MAXN];
3 void GomoryHu(int n) { // 0-base
4     fill_n(g, n, 0);
5     for (int i = 1; i < n; ++i) {
6         Dinic.reset();
7         add_edge(i, g[i], Dinic.maxflow(i, g[i]));
8         for (int j = i + 1; j <= n; ++j)
9             if (g[j] == g[i] && ~Dinic.dis[j])
10                 g[j] = i;
11     }
12 }

```

4.7 MincostMaxflow [0722e9]

```

1 struct MinCostMaxFlow { // 0-base
2     struct Edge {
3         ll from, to, cap, flow, cost, rev;
4     } *past[N];
5     vector<Edge> G[N];
6     int inq[N], n, s, t;
7     ll dis[N], up[N], pot[N];
8     bool BellmanFord() {
9         fill_n(dis, n, INF), fill_n(inq, n, 0);
10        queue<int> q;
11        auto relax = [&](int u, ll d, ll cap, Edge *e) {
12            if (cap > 0 && dis[u] > d) {
13                dis[u] = d, up[u] = cap, past[u] = e;
14                if (!inq[u]) inq[u] = 1, q.push(u);
15            }
16        };
17        relax(s, 0, INF, 0);
18        while (!q.empty()) {
19            int u = q.front();
20            q.pop(), inq[u] = 0;
21            for (auto &e : G[u]) {
22                ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
23                relax
24                    (e.to, d2, min(up[u], e.cap - e.flow), &e);
25            }
26            return dis[t] != INF;
27        }
28    }
29    void solve(int _s
30        , int _t, ll &flow, ll &cost, bool neg = true) {
31        s = _s, t = _t, flow = 0, cost = 0;
32        if (neg) BellmanFord(), copy_n(dis, n, pot);
33        for (; BellmanFord(); copy_n(dis, n, pot)) {
34            for (int
35                i = 0; i < n; ++i) dis[i] += pot[i] - pot[s];
36            flow += up[t], cost += up[t] * dis[t];
37            for (int i = t; past[i]; i = past[i]->from) {
38                auto &e = *past[i];
39                e.flow += up[t], G[e.to][e.rev].flow -= up[t];
40            }
41        }
42    }
43    void init(int _n) {
44        n = _n, fill_n(pot, n, 0);
45        for (int i = 0; i < n; ++i) G[i].clear();
46    }
47    void add_edge(ll a, ll b, ll cap, ll cost) {
48        G[a].pb(Edge{a, b, cap, 0, cost, SZ(G[b])});
49        G[b].pb(Edge{b, a, 0, 0, -cost, SZ(G[a]) - 1});
50    }
51 };

```

4.8 SW-mincut [8e90f0]

```

1 struct SW{ // global min cut, O(V^3)
2     #define REP for (int i = 0; i < n; ++i)
3     static const int MXN = 514, INF = 2147483647;
4     int vst[MXN], edge[MXN][MXN], wei[MXN];
5     void init(int n) {
6         REP fill_n(edge[i], n, 0);
7     }
8     void addEdge(int u, int v, int w){
9         edge[u][v] += w; edge[v][u] += w;
10    }
11    int search(int &s, int &t, int n){
12        fill_n(vst, n, 0), fill_n(wei, n, 0);
13        s = t = -1;
14        int mx, cur;
15        for (int j = 0; j < n; ++j) {
16            mx = -1, cur = 0;
17            REP if (wei[i] > mx) cur = i, mx = wei[i];
18            vst[cur] = 1, wei[cur] = -1;
19            s = t; t = cur;
20            REP if (!vst[i]) wei[i] += edge[cur][i];
21        }
22        return mx;
23    }
24    int solve(int n) {
25        int res = INF;
26        for (int x, y; n > 1; n--){
27            res = min(res, search(x, y, n));
28            REP edge[i][x] = (edge[x][i] += edge[y][i]);
29            REP {
30                edge[y][i] = edge[n - 1][i];
31                edge[i][y] = edge[i][n - 1];
32            }
33        }
34    }
35 };

```

```

32     } // edge[y][y] = 0;
33 }
34 return res;
35 }
36 } sw;

```

4.9 Maximum Weight Matching [a10467]

```

1 #define REP(i, l, r) for (int i=l; i<=(r); ++i)
2 struct WeightGraph { // 1-based
3     struct edge { int u, v, w; }; int n, nx;
4     vector<int> lab; vector<vector<edge>> g;
5     vector<int> slk, match, st, pa, S, vis;
6     vector<vector<int>> flo, flo_from; queue<int> q;
7     WeightGraph(int n_) : n(n_), nx(n * 2), lab(nx + 1),
8         g(nx + 1, vector<edge>(nx + 1)), slk(nx + 1),
9         flo(nx + 1), flo_from(nx + 1, vector(n + 1, 0)) {
10         match = st = pa = S = vis = slk;
11         REP(u, 1, n) REP(v, 1, n) g[u][v] = {u, v, 0};
12     }
13     int E(edge e)
14     { return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2; }
15     void update_slk(int u, int x, int &s)
16     { if (!s || E(g[u][x]) < E(g[s][x])) s = u; }
17     void set_slk(int x) {
18         slk[x] = 0;
19         REP(u, 1, n)
20             if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
21                 update_slk(u, x, slk[x]);
22     }
23     void q_push(int x) {
24         if (x <= n) q.push(x);
25         else for (int y : flo[x]) q.push(y);
26     }
27     void set_st(int x, int b) {
28         st[x] = b;
29         if (x > n) for (int y : flo[x]) set_st(y, b);
30     }
31     vector<int> split_flo(auto &f, int xr) {
32         auto it = find(ALL(f), xr);
33         if (auto pr = it - f.begin(); pr % 2 == 1)
34             reverse(1 + ALL(f), it = f.end() - pr);
35         auto res = vector(f.begin(), it);
36         return f.erase(f.begin(), it), res;
37     }
38     void set_match(int u, int v) {
39         match[u] = g[u][v].v;
40         if (u <= n) return;
41         int xr = flo_from[u][g[u][v].u];
42         auto &f = flo[u], z = split_flo(f, xr);
43         REP(i, 0, SZ(z) - 1) set_match(z[i], z[i ^ 1]);
44         set_match(xr, v); f.insert(f.end(), ALL(z));
45     }
46     void augment(int u, int v) {
47         for (;;) {
48             int xnv = st[match[u]]; set_match(u, v);
49             if (!xnv) return;
50             set_match(v = xnv, u = st[pa[xnv]]);
51         }
52     }
53     int lca(int u, int v) {
54         static int t = 0; ++t;
55         for (++t; u || v; swap(u, v)) if (u) {
56             if (vis[u] == t) return u;
57             vis[u] = t, u = st[match[u]];
58             if (u) u = st[pa[u]];
59         }
60         return 0;
61     }
62     void add_blossom(int u, int o, int v) {
63         int b = find(n + 1 + ALL(st), 0) - begin(st);
64         lab[b] = 0, S[b] = 0, match[b] = match[o];
65         vector<int> f = {o};
66         for (int t : {u, v}) {
67             reverse(1 + ALL(f));
68             for (int x = t, y; x != o; x = st[pa[y]])
69                 f.pb(x), f.pb(y = st[match[x]]), q_push(y);
70         }
71         flo[b] = f; set_st(b, b);
72         REP(x, 1, nx) g[b][x].w = g[x][b].w = 0;
73         fill(ALL(flo_from[b]), 0);
74         for (int xs : flo[b]) {
75             REP(x, 1, nx)
76                 if (g[b][x].w == 0 || E(g[xs][x]) < E(g[b][x]))
77                     g[b][x] = g[xs][x], g[x][b] = g[x][xs];
78             REP(x, 1, n)
79                 if (flo_from[xs][x]) flo_from[b][x] = xs;

```

```

80     }
81     set_slk(b);
82 }
83 void expand_blossom(int b) {
84     for (int x : flo[b]) set_st(x, x);
85     int xr = flo_from[b][g[b][pa[b]].u], xs = -1;
86     for (int x : split_flo(flo[b], xr)) {
87         if (xs == -1) { xs = x; continue; }
88         pa[xs] = g[x][xs].u, S[xs] = 1, S[x] = 0;
89         slk[xs] = 0, set_slk(x), q_push(x), xs = -1;
90     }
91     for (int x : flo[b])
92         if (x == xr) S[x] = 1, pa[x] = pa[b];
93         else S[x] = -1, set_slk(x);
94     st[b] = 0;
95 }
96 bool on_found_edge(const edge &e) {
97     if (int u = st[e.u], v = st[e.v]; S[v] == -1) {
98         int nu = st[match[v]]; pa[v] = e.u; S[v] = 1;
99         slk[v] = slk[nu] = S[nu] = 0; q_push(nu);
100     } else if (S[v] == 0) {
101         if (int o = lca(u, v)) add_blossom(u, o, v);
102         else return augment(u, v), augment(v, u), true;
103     }
104     return false;
105 }
106 bool matching() {
107     fill(ALL(S), -1), fill(ALL(slk), 0);
108     q = queue<int>();
109     REP(x, 1, nx) if (st[x] == x && !match[x])
110         pa[x] = S[x] = 0, q_push(x);
111     if (q.empty()) return false;
112     for (;;) {
113         while (SZ(q)) {
114             int u = q.front(); q.pop();
115             if (S[st[u]] == 1) continue;
116             REP(v, 1, n)
117                 if (g[u][v].w > 0 && st[u] != st[v]) {
118                     if (E(g[u][v]) != 0)
119                         update_slk(u, st[v], slk[st[v]]);
120                     else if
121                         (on_found_edge(g[u][v])) return true;
122                 }
123             int d = INF;
124             REP(b, n + 1, nx) if (st[b] == b && S[b] == 1)
125                 d = min(d, lab[b] / 2);
126             REP(x, 1, nx)
127                 if (int
128                     s = slk[x]; st[x] == x && s && S[x] <= 0)
129                     d = min(d, E(g[s][x]) / (S[x] + 2));
130             REP(u, 1, n)
131                 if (S[st[u]] == 1) lab[u] += d;
132                 else if (S[st[u]] == 0) {
133                     if (lab[u] <= d) return false;
134                     lab[u] -= d;
135                 }
136             REP(b, n + 1, nx) if (st[b] == b && S[b] >= 0)
137                 lab[b] += d * (2 - 4 * S[b]);
138             REP(x, 1, nx)
139                 if (int
140                     s = slk[x]; st[x] == x &&
141                     s && st[s] != x && E(g[s][x]) == 0)
142                     if (on_found_edge(g[s][x])) return true;
143             REP(b, n + 1, nx)
144                 if (st[b] == b && S[b] == 1 && lab[b] == 0)
145                     expand_blossom(b);
146         }
147         return false;
148     }
149     pair<ll, int> solve() {
150         fill(ALL(match), 0);
151         REP(u, 0, n) st[u] = u, flo[u].clear();
152         int w_max = 0;
153         REP(u, 1, n) REP(v, 1, n) {
154             flo_from[u][v] = (u == v ? u : 0);
155             w_max = max(w_max, g[u][v].w);
156         }
157         fill(ALL(lab), w_max);
158         int n_matches = 0; ll tot_weight = 0;
159         while (matching()) ++n_matches;
160         REP(u, 1, n) if (match[u] && match[u] < u)
161             tot_weight += g[u][match[u]].w;
162         return make_pair(tot_weight, n_matches);
163     }
164     void add_edge(int u, int v, int w)
165     { g[u][v].w = g[v][u].w = w; }

```

164

4.10 Minimum Weight Matching wrong [f27d66]

```

1 struct Graph { // 0-base (Perfect Match), n is even
2     int n, match[N], onstk[N], stk[N], tp;
3     ll edge[N][N], dis[N];
4     void init(int _n) {
5         n = _n, tp = 0;
6         for (int i = 0; i < n; ++i) fill_n(edge[i], n, 0);
7     }
8     void add_edge(int u, int v, ll w) {
9         edge[u][v] = edge[v][u] = w;
10    }
11    bool SPFA(int u) {
12        stk[tp++] = u, onstk[u] = 1;
13        for (int v = 0; v < n; ++v)
14            if (!onstk[v] && match[u] != v) {
15                int m = match[v];
16                if (dis[m] >
17                    dis[u] - edge[v][m] + edge[u][v]) {
18                    dis[m] = dis[u] - edge[v][m] + edge[u][v];
19                    onstk[v] = 1, stk[tp++] = v;
20                    if (onstk[m] || SPFA(m)) return 1;
21                    --tp, onstk[v] = 0;
22                }
23            }
24        onstk[u] = 0, --tp;
25        return 0;
26    }
27    ll solve() { // find a match
28        for (int i = 0; i < n; ++i) match[i] = i ^ 1;
29        while (1) {
30            int found = 0;
31            fill_n(dis, n, 0);
32            fill_n(onstk, n, 0);
33            for (int i = 0; i < n; ++i)
34                if (tp == 0, !onstk[i] && SPFA(i))
35                    for (found = 1; tp >= 2; ) {
36                        int u = stk[--tp];
37                        int v = stk[--tp];
38                        match[u] = v, match[v] = u;
39                    }
40            if (!found) break;
41        }
42        ll ret = 0;
43        for (int i = 0; i < n; ++i)
44            ret += edge[i][match[i]];
45        return ret >> 1;
46    }
47 };

```

4.11 Bipartite Matching [623c76]

```

1 struct Bipartite_Matching { // 0-base
2     int mp[N], mq[N], dis[N + 1], cur[N], l, r;
3     vector<int> G[N + 1];
4     bool dfs(int u) {
5         for (int &i = cur[u]; i < SZ(G[u]); ++i) {
6             int e = G[u][i];
7             if (mq[e] == l
8                 || (dis[mq[e]] == dis[u] + 1 && dfs(mq[e])))
9                 return mp[mq[e] = u] = e, 1;
10        }
11        return dis[u] = -1, 0;
12    }
13    bool bfs() {
14        queue<int> q;
15        fill_n(dis, l + 1, -1);
16        for (int i = 0; i < l; ++i)
17            if (!~mp[i])
18                q.push(i), dis[i] = 0;
19        while (!q.empty()) {
20            int u = q.front();
21            q.pop();
22            for (int e : G[u])
23                if (!~dis[mq[e]])
24                    q.push(mq[e]), dis[mq[e]] = dis[u] + 1;
25        }
26        return dis[l] != -1;
27    }
28    int matching() {
29        int res = 0;
30        fill_n(mp, l, -1), fill_n(mq, r, l);
31        while (bfs()) {
32            fill_n(cur, l, 0);
33            for (int i = 0; i < l; ++i)

```

```

33        res += (!~mp[i] && dfs(i));
34    }
35    return res; // (i, mp[i] != -1)
36 }
37 void add_edge(int s, int t) { G[s].pb(t); }
38 void init(int _l, int _r) {
39     l = _l, r = _r;
40     for (int i = 0; i <= l; ++i)
41         G[i].clear();
42 }
43 };

```

4.12 BoundedFlow [e8670b]

```

1 struct BoundedFlow { // 0-base
2     struct edge {
3         int to, cap, flow, rev;
4     };
5     vector<edge> G[N];
6     int n, s, t, dis[N], cur[N], cnt[N];
7     void init(int _n) {
8         n = _n;
9         for (int i = 0; i < n + 2; ++i)
10             G[i].clear(), cnt[i] = 0;
11    }
12    void add_edge(int u, int v, int lcap, int rcap) {
13        cnt[u] -= lcap, cnt[v] += lcap;
14        G[u].pb(edge{v, rcap, lcap, SZ(G[v])});
15        G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
16    }
17    void add_edge(int u, int v, int cap) {
18        G[u].pb(edge{v, cap, 0, SZ(G[v])});
19        G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
20    }
21    int dfs(int u, int cap) {
22        if (u == t || !cap) return cap;
23        for (int &i = cur[u]; i < SZ(G[u]); ++i) {
24            edge &e = G[u][i];
25            if (dis[e.to] == dis[u] + 1 && e.cap != e.flow) {
26                int df = dfs(e.to, min(e.cap - e.flow, cap));
27                if (df) {
28                    e.flow += df, G[e.to][e.rev].flow -= df;
29                    return df;
30                }
31            }
32        }
33        dis[u] = -1;
34        return 0;
35    }
36    bool bfs() {
37        fill_n(dis, n + 3, -1);
38        queue<int> q;
39        q.push(s), dis[s] = 0;
40        while (!q.empty()) {
41            int u = q.front();
42            q.pop();
43            for (edge &e : G[u])
44                if (!~dis[e.to] && e.flow != e.cap)
45                    q.push(e.to), dis[e.to] = dis[u] + 1;
46        }
47        return dis[t] != -1;
48    }
49    int maxflow(int _s, int _t) {
50        s = _s, t = _t;
51        int flow = 0, df;
52        while (bfs()) {
53            fill_n(cur, n + 3, 0);
54            while ((df = dfs(s, INF))) flow += df;
55        }
56        return flow;
57    }
58    bool solve() {
59        int sum = 0;
60        for (int i = 0; i < n; ++i)
61            if (cnt[i] > 0)
62                add_edge(n + 1, i, cnt[i]), sum += cnt[i];
63            else if (cnt[i] < 0) add_edge(i, n + 2, -cnt[i]);
64        if (sum != maxflow(n + 1, n + 2)) sum = -1;
65        for (int i = 0; i < n; ++i)
66            if (cnt[i] > 0)
67                G[n + 1].pop_back(), G[i].pop_back();
68            else if (cnt[i] < 0)
69                G[i].pop_back(), G[n + 2].pop_back();
70        return sum != -1;
71    }
72    int solve(int _s, int _t) {
73        add_edge(_t, _s, INF);

```

```

74     if (!solve()) return -1; // invalid flow
75     int x = G[_t].back().flow;
76     return G[_t].pop_back(), G[_s].pop_back(), x;
77 }
78 };

```

4.13 Dinic [ba0999]

```

1 struct MaxFlow { // 0-base
2     struct edge {
3         int to, cap, flow, rev;
4     };
5     vector<edge> G[MAXN];
6     int s, t, dis[MAXN], cur[MAXN], n;
7     int dfs(int u, int cap) {
8         if (u == t || !cap) return cap;
9         for (int &i = cur[u]; i < (int)G[u].size(); ++i) {
10             edge &e = G[u][i];
11             if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
12                 int df = dfs(e.to, min(e.cap - e.flow, cap));
13                 if (df) {
14                     e.flow += df;
15                     G[e.to][e.rev].flow -= df;
16                     return df;
17                 }
18             }
19         }
20         dis[u] = -1;
21         return 0;
22     }
23     bool bfs() {
24         fill_n(dis, n, -1);
25         queue<int> q;
26         q.push(s), dis[s] = 0;
27         while (!q.empty()) {
28             int tmp = q.front();
29             q.pop();
30             for (auto &u : G[tmp])
31                 if (!dis[u.to] && u.flow != u.cap) {
32                     q.push(u.to);
33                     dis[u.to] = dis[tmp] + 1;
34                 }
35         }
36         return dis[t] != -1;
37     }
38     int maxflow(int _s, int _t) {
39         s = _s, t = _t;
40         int flow = 0, df;
41         while (bfs()) {
42             fill_n(cur, n, 0);
43             while ((df = dfs(s, INF))) flow += df;
44         }
45         return flow;
46     }
47     void init(int _n) {
48         n = _n;
49         for (int i = 0; i < n; ++i) G[i].clear();
50     }
51     void reset() {
52         for (int i = 0; i < n; ++i)
53             for (auto &j : G[i]) j.flow = 0;
54     }
55     void add_edge(int u, int v, int cap) {
56         G[u].pb(edge{v, cap, 0, (int)G[v].size()});
57         G[v].pb(edge{u, 0, 0, (int)G[u].size() - 1});
58     }
59 };

```

4.14 MinCostCirculation [86e6a8]

```

1 struct MinCostCirculation { // 0-base
2     struct Edge {
3         ll from, to, cap, fcap, flow, cost, rev;
4     } *past[N];
5     vector<Edge> G[N];
6     ll dis[N], inq[N], n;
7     void BellmanFord(int s) {
8         fill_n(dis, n, INF), fill_n(inq, n, 0);
9         queue<int> q;
10         auto relax = [&](int u, ll d, Edge *e) {
11             if (dis[u] > d) {
12                 dis[u] = d, past[u] = e;
13                 if (!inq[u]) inq[u] = 1, q.push(u);
14             }
15         };
16         relax(s, 0, 0);
17         while (!q.empty()) {

```

```

18             int u = q.front();
19             q.pop(), inq[u] = 0;
20             for (auto &e : G[u])
21                 if (e.cap > e.flow)
22                     relax(e.to, dis[u] + e.cost, &e);
23         }
24     }
25     void try_edge(Edge &cur) {
26         if (cur.cap > cur.flow) return ++cur.cap, void();
27         BellmanFord(cur.to);
28         if (dis[cur.from] + cur.cost < 0) {
29             ++cur.flow, --G[cur.to][cur.rev].flow;
30             for (int
31                 i = cur.from; past[i]; i = past[i]->from) {
32                 auto &e = *past[i];
33                 ++e.flow, --G[e.to][e.rev].flow;
34             }
35             ++cur.cap;
36         }
37     }
38     void solve(int mxlg) {
39         for (int b = mxlg; b >= 0; --b) {
40             for (int i = 0; i < n; ++i)
41                 for (auto &e : G[i])
42                     e.cap *= 2, e.flow *= 2;
43             for (int i = 0; i < n; ++i)
44                 for (auto &e : G[i])
45                     if (e.fcap >> b & 1)
46                         try_edge(e);
47         }
48     }
49     void init(int _n) { n = _n;
50         for (int i = 0; i < n; ++i) G[i].clear();
51     }
52     void add_edge(ll a, ll b, ll cap, ll cost) {
53         G[a].pb(Edge{b, a, 0, 0, 0, cost, SZ(G[b]) + (a == b)});
54         G[b].pb(Edge{b, a, 0, 0, 0, -cost, SZ(G[a]) - 1});
55     }
56 } mcmf; // O(VE * ElogC)

```

5 String

5.1 Smallest Rotation [d69462]

```

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

```

5.2 Manacher [11ebce]

```

1 int z[MAXN]; // 0-base
2 /* center i: radius z[i * 2 + 1] / 2
3    center i, i + 1: radius z[i * 2 + 2] / 2
4    both aba, abba have radius 2 */
5 void Manacher(string tmp) {
6     string s = "%";
7     int l = 0, r = 0;
8     for (char c : tmp) s.pb(c), s.pb('%');
9     for (int i = 0; i < SZ(s); ++i) {
10        z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
11        while (i - z[i] >= 0 && i + z[i] < SZ(s)
12            && s[i + z[i]] == s[i - z[i]]) ++z[i];
13        if (z[i] + i > r) r = z[i] + i, l = i;
14    }
15 }

```

5.3 De Bruijn sequence [151f80]

```

1 constexpr int MAXC = 10, MAXN = 1e5 + 10;
2 struct DBSeq {
3     int C, N, K, L, buf[MAXC * MAXN]; // K <= C^N
4     void dfs(int *out, int t, int p, int &ptr) {
5         if (ptr >= L) return;
6         if (t > N) {
7             if (N % p) return;
8             for (int i = 1; i <= p && ptr < L; ++i)

```

```

9     out[ptr++] = buf[i];
10 } else {
11     buf[t] = buf[t - p], dfs(out, t + 1, p, ptr);
12     for (int j = buf[t - p] + 1; j < C; ++j)
13         buf[t] = j, dfs(out, t + 1, t, ptr);
14 }
15 }
16 void solve(int _c, int _n, int _k, int *out) {
17     int p = 0;
18     C = _c, N = _n, K = _k, L = N + K - 1;
19     dfs(out, 1, 1, p);
20     if (p < L) fill(out + p, out + L, 0);
21 }
22 } dbs;

```

5.4 SAM [4d0baa]

```

1 const int MAXM = 1000010;
2 struct SAM {
3     int tot, root, lst, mom[MAXM], mx[MAXM];
4     int nxt[MAXM][33], cnt[MAXM], in[MAXM];
5     int newNode() {
6         int res = ++tot;
7         fill(nxt[res], nxt[res] + 33, 0);
8         mom[res] = mx[res] = cnt[res] = in[res] = 0;
9         return res;
10    }
11    void init() {
12        tot = 0;
13        root = newNode();
14        mom[root] = 0, mx[root] = 0;
15        lst = root;
16    }
17    void push(int c) {
18        int p = lst;
19        int np = newNode();
20        mx[np] = mx[p] + 1;
21        for (; p && nxt[p][c] == 0; p = mom[p])
22            nxt[p][c] = np;
23        if (p == 0) mom[np] = root;
24        else {
25            int q = nxt[p][c];
26            if (mx[p] + 1 == mx[q]) mom[np] = q;
27            else {
28                int nq = newNode();
29                mx[nq] = mx[p] + 1;
30                for (int i = 0; i < 33; i++)
31                    nxt[nq][i] = nxt[q][i];
32                mom[nq] = mom[q];
33                mom[q] = nq;
34                mom[np] = nq;
35                for (; p && nxt[p][c] == q; p = mom[p])
36                    nxt[p][c] = nq;
37            }
38        }
39        lst = np, cnt[np] = 1;
40    }
41    void push(char *str) {
42        for (int i = 0; str[i]; i++)
43            push(str[i] - 'a' + 1);
44    }
45    void count() {
46        for (int i = 1; i <= tot; ++i)
47            ++in[mom[i]];
48        queue<int> q;
49        for (int i = 1; i <= tot; ++i)
50            if (!in[i]) q.push(i);
51        while (!q.empty()) {
52            int u = q.front();
53            q.pop();
54            cnt[mom[u]] += cnt[u];
55            if (!--in[mom[u]])
56                q.push(mom[u]);
57        }
58    }
59 } sam;

```

5.5 Aho-Corasick Automatan [8c56e8]

```

1 struct AC_Automatan {
2     int nx[len][sigma], fl[len], cnt[len], ord[len], top;
3     int rnx[len][sigma]; // node actually be reached
4     int newNode() {
5         fill_n(nx[top], sigma, -1);
6         return top++;
7     }
8     void init() {top = 1, newNode(); }

```

```

9     int input(string &s) {
10         int X = 1;
11         for (char c : s) {
12             if (!nx[X][c - 'A']) nx[X][c - 'A'] = newNode();
13             X = nx[X][c - 'A'];
14         }
15         return X; // return the end node of string
16     }
17     void make_fl() {
18         queue<int> q;
19         q.push(1), fl[1] = 0;
20         for (int t = 0; !q.empty(); ) {
21             int R = q.front();
22             q.pop(), ord[t++] = R;
23             for (int i = 0; i < sigma; ++i)
24                 if (~nx[R][i]) {
25                     int X = rnx[R][i] = nx[R][i], Z = fl[R];
26                     for (; Z && !~nx[Z][i]; ) Z = fl[Z];
27                     fl[X] = Z ? nx[Z][i] : 1, q.push(X);
28                 }
29             else rnx[R][i] = R > 1 ? rnx[fl[R]][i] : 1;
30         }
31     }
32     void solve() {
33         for (int i = top - 2; i > 0; --i)
34             cnt[fl[ord[i]]] += cnt[ord[i]];
35     }
36 } ac;

```

5.6 SAIS-old [ea9200]

```

1 class SAIS {
2 public:
3     int *SA, *H;
4     // zero based, string content MUST > 0
5     // result height H[i] is LCP(SA[i - 1], SA[i])
6     // string, length, |sigma|
7     void build(int *s, int n, int m = 128) {
8         copy_n(s, n, _s);
9         _h[0] = _s[n++] = 0;
10        sais(_s, _sa, _p, _q, _t, _c, n, m);
11        mkhei(n);
12        SA = _sa + 1;
13        H = _h + 1;
14    }
15 private:
16    bool _t[N * 2];
17    int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2],
18        r[N], _sa[N * 2], _h[N];
19    void mkhei(int n) {
20        for (int i = 0; i < n; i++) r[_sa[i]] = i;
21        for (int i = 0; i < n; i++)
22            if (r[i]) {
23                int ans = i > 0 ? max(_h[r[i - 1]] - 1, 0) : 0;
24                while (_s[i + ans] == _s[_sa[r[i] - 1] + ans])
25                    ans++;
26                _h[r[i]] = ans;
27            }
28    }
29    void sais(int *s, int *sa, int *p, int *q, bool *t,
30        int *c, int n, int z) {
31        bool uniq = t[n - 1] = 1, neq;
32        int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
33            lst = -1;
34
35        #define MAGIC(XD)
36        fill_n(sa, n, 0);
37        copy_n(c, z, x);
38        XD;
39        copy_n(c, z - 1, x + 1);
40        for (int i = 0; i < n; i++)
41            if (sa[i] && !t[sa[i] - 1])
42                sa[x[sa[i] - 1]++] = sa[i] - 1;
43        copy_n(c, z, x);
44        for (int i = n - 1; i >= 0; i--)
45            if (sa[i] && t[sa[i] - 1])
46                sa[--x[sa[i] - 1]] = sa[i] - 1;
47
48        fill_n(c, z, 0);
49        for (int i = 0; i < n; i++) uniq &= ++c[s[i]] < 2;
50        partial_sum(c, c + z, c);
51        if (uniq) {
52            for (int i = 0; i < n; i++) sa[--c[s[i]]] = i;
53            return;
54        }
55        for (int i = n - 2; i >= 0; i--)

```

```

57     t[i] = (s[i] == s[i + 1] ? t[i + 1]
58           : s[i] < s[i + 1]);
59     MAGIC(for (int i = 1; i <= n - 1;
60            i++) if (t[i] && !t[i - 1]))
61         sa[--x[s[i]]] = p[q[i] = nn++] = i;
62     for (int i = 0; i < n; i++)
63         if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
64             neq = (lst < 0) ||
65                   !equal(s + lst,
66                          s + lst + p[q[sa[i]] + 1] - sa[i],
67                          s + sa[i]);
68             ns[q[lst = sa[i]]] = nmxz += neq;
69         }
70     sais(ns, nsa, p + nn, q + n, t + n, c + z, nn,
71          nmxz + 1);
72     MAGIC(for (int i = nn - 1; i >= 0; i--)
73            sa[--x[s[p[nsa[i]]]]] = p[nsa[i]]);
74 }
75 } sa;

```

5.7 Z-value [2e5c4c]

```

1 int z[MAXn];
2 void make_z(const string &s) {
3     int l = 0, r = 0;
4     for (int i = 1; i < SZ(s); ++i) {
5         for (z[i] = max(0, min(r - i + 1, z[i - l]));
6             i + z[i] < SZ(s) && s[i + z[i]] == s[z[i]];
7             ++z[i])
8             ;
9         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
10    }
11 }

```

5.8 exSAM [0b980b]

```

1 struct exSAM {
2     int len[N * 2], link[N * 2]; // maxlength, suflink
3     int next[N * 2][CNUM], tot; // [0, tot), root = 0
4     int lenSorted[N * 2]; // topo. order
5     int cnt[N * 2]; // occurence
6     int newnode() {
7         fill_n(next[tot], CNUM, 0);
8         len[tot] = cnt[tot] = link[tot] = 0;
9         return tot++;
10    }
11    void init() { tot = 0, newnode(), link[0] = -1; }
12    int insertSAM(int last, int c) {
13        int cur = next[last][c];
14        len[cur] = len[last] + 1;
15        int p = link[last];
16        while (p != -1 && !next[p][c])
17            next[p][c] = cur, p = link[p];
18        if (p == -1) return link[cur] = 0, cur;
19        int q = next[p][c];
20        if (len
21            [p] + 1 == len[q]) return link[cur] = q, cur;
22        int clone = newnode();
23        for (int i = 0; i < CNUM; ++i)
24            next[
25                clone][i] = len[next[q][i]] ? next[q][i] : 0;
26        len[clone] = len[p] + 1;
27        while (p != -1 && next[p][c] == q)
28            next[p][c] = clone, p = link[p];
29        link[link[cur] = clone] = link[q];
30        link[q] = clone;
31        return cur;
32    }
33    void insert(const string &s) {
34        int cur = 0;
35        for (auto ch : s) {
36            int &nxt = next[cur][int(ch - 'a')];
37            if (!nxt) nxt = newnode();
38            cnt[cur = nxt] += 1;
39        }
40    }
41    void build() {
42        queue<int> q;
43        q.push(0);
44        while (!q.empty()) {
45            int cur = q.front();
46            q.pop();
47            for (int i = 0; i < CNUM; ++i)
48                if (next[cur][i])
49                    q.push(insertSAM(cur, i));
50        }
51        vector<int> lc(tot);

```

```

50     for (int i = 1; i < tot; ++i) ++lc[len[i]];
51     partial_sum(ALL(lc), lc.begin());
52     for (int i
53           = 1; i < tot; ++i) lenSorted[--lc[len[i]]] = i;
54 }
55 void solve() {
56     for (int i = tot - 2; i >= 0; --i)
57         cnt[link[lenSorted[i]]] += cnt[lenSorted[i]];
58 }

```

5.9 SAIS [afbad7]

```

1 namespace sfx {
2     bool _t[N * 2];
3     int SA[N * 2], H[N], RA[N];
4     int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2];
5     // zero based, string content MUST > 0
6     // SA[i]: SA[i]-th
7     // suffix is the i-th lexicographically smallest suffix.
8     // H[i]: longest
9     // common prefix of suffix SA[i] and suffix SA[i - 1].
10    void pre(int *sa, int *c, int n, int z)
11    { fill_n(sa, n, 0), copy_n(c, z, x); }
12    void induce
13    (int *sa, int *c, int *s, bool *t, int n, int z) {
14        copy_n(c, z - 1, x + 1);
15        for (int i = 0; i < n; ++i)
16            if (sa[i] && !t[sa[i] - 1])
17                sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
18        copy_n(c, z, x);
19        for (int i = n - 1; i >= 0; --i)
20            if (sa[i] && t[sa[i] - 1])
21                sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
22    }
23    void sais(int *s, int *sa
24              , int *p, int *q, bool *t, int *c, int n, int z) {
25        bool uniq = t[n - 1] = true;
26        int nn = 0,
27              nmxz = -1, *nsa = sa + n, *ns = s + n, last = -1;
28        fill_n(c, z, 0);
29        for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
30        partial_sum(c, c + z, c);
31        if (uniq) {
32            for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
33            return;
34        }
35        for (int i = n - 2; i >= 0; --i)
36            t[i] = (
37                s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
38        pre(sa, c, n, z);
39        for (int i = 1; i <= n - 1; ++i)
40            if (t[i] && !t[i - 1])
41                sa[--x[s[i]]] = p[q[i] = nn++] = i;
42        induce(sa, c, s, t, n, z);
43        for (int i = 0; i < n; ++i)
44            if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
45                bool neq = last < 0 || !equal
46                    (s + sa[i], s + p[q[sa[i]] + 1], s + last);
47                ns[q[last = sa[i]]] = nmxz += neq;
48            }
49        sais(ns,
50            nsa, p + nn, q + n, t + n, c + z, nn, nmxz + 1);
51        pre(sa, c, n, z);
52        for (int i = nn - 1; i >= 0; --i)
53            sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
54        induce(sa, c, s, t, n, z);
55    }
56    void mkhei(int n) {
57        for (int i = 0, j = 0; i < n; ++i) {
58            if (RA[i])
59                for (; _s[i + j] == _s[SA[RA[i] - 1] + j]; ++j);
60            H[RA[i]] = j, j = max(0, j - 1);
61        }
62    }
63    void build(int *s, int n) {
64        copy_n(s, n, _s), _s[n] = 0;
65        sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
66        copy_n(SA + 1, n, SA);
67        for (int i = 0; i < n; ++i) RA[SA[i]] = i;
68        mkhei(n);
69    }

```

5.10 SAIS-C++20 [2cd2ea]

```

1 auto sais(const auto &s) {
2     const int n = SZ(s), z = ranges::max(s) + 1;

```



```

3  if (n == 1) return vector{0};
4  vector<int> c(z); for (int x : s) ++c[x];
5  partial_sum(ALL(c), begin(c));
6  vector<int> sa(n); auto I = views::iota(0, n);
7  vector<bool> t(n, true);
8  for (int i = n - 2; i >= 0; --i)
9      t[i] = (
10         s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
11  auto is_lms = views::filter([&t](int x) {
12      return x && t[x] && !t[x - 1];
13  });
14  auto induce = [&] {
15      for (auto x = c; int y : sa)
16          if (y-- && !t[y]) sa[x[s[y] - 1]++] = y;
17      for (auto x = c; int y : sa | views::reverse)
18          if (y-- && t[y]) sa[--x[s[y]]] = y;
19  };
20  vector<int> lms, q(n); lms.reserve(n);
21  for (auto x = c; int i : I | is_lms)
22      q[i] = SZ(lms), lms.pb(sa[--x[s[i]]] = i);
23  induce(); vector<int> ns(SZ(lms));
24  for (int j = -1, nz = 0; int i : sa | is_lms) {
25      if (j >= 0) {
26          int len = min({n - i, n - j, lms[q[i] + 1] - i});
27          ns[q[i]] = nz += lexicographical_compare(
28              begin(s) + j, begin(s) + j + len,
29              begin(s) + i, begin(s) + i + len);
30      }
31      j = i;
32  }
33  fill(ALL(sa), 0); auto nsa = sais(ns);
34  for (auto x = c; int y : nsa | views::reverse)
35      y = lms[y], sa[--x[s[y]]] = y;
36  return induce(), sa;
37  // sa[i]: sa[i]-th suffix
38  // is the i-th lexicographically smallest suffix.
39  // hi[i]: LCP of suffix sa[i] and suffix sa[i - 1].
40  struct Suffix {
41      int n; vector<int> sa, hi, ra;
42      Suffix
43          (const auto &s, int _n) : n(_n), hi(n), ra(n) {
44          vector<int> s(n + 1); // s[n] = 0;
45          copy_n(_s, n, begin(s)); // _s shouldn't contain 0
46          sa = sais(s); sa.erase(sa.begin());
47          for (int i = 0; i < n; ++i) ra[sa[i]] = i;
48          for (int i = 0, h = 0; i < n; ++i) {
49              if (!ra[i]) { h = 0; continue; }
50              for (int j = sa[ra[i] - 1]; max
51                  (i, j) + h < n && s[i + h] == s[j + h];) ++h;
52              hi[ra[i]] = h ? h - 1 : 0;
53          }
54      }
55  };

```

5.11 PalTree [9bd3fb]

```

1  struct palindromic_tree {
2      struct node {
3          int next[26], fail, len;
4          int cnt, num; // cnt: appear times, num: number of
5                          // pal. suf.
6          node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
7              for (int i = 0; i < 26; ++i) next[i] = 0;
8          }
9      };
10     vector<node> St;
11     vector<char> s;
12     int last, n;
13     palindromic_tree() : St(2), last(1), n(0) {
14         St[0].fail = 1, St[1].len = -1, s.pb(-1);
15     }
16     inline void clear() {
17         St.clear(), s.clear(), last = 1, n = 0;
18         St.pb(0), St.pb(-1);
19         St[0].fail = 1, s.pb(-1);
20     }
21     inline int get_fail(int x) {
22         while (s[n - St[x].len - 1] != s[n])
23             x = St[x].fail;
24         return x;
25     }
26     inline void add(int c) {
27         s.push_back(c == 'a' ? 'a', ++n;
28         int cur = get_fail(last);
29         if (!St[cur].next[c]) {
30             int now = SZ(St);

```

```

31         St.pb(St[cur].len + 2);
32         St[now].fail =
33             St[get_fail(St[cur].fail)].next[c];
34         St[cur].next[c] = now;
35         St[now].num = St[St[now].fail].num + 1;
36     }
37     last = St[cur].next[c], ++St[last].cnt;
38 }
39 inline void count() { // counting cnt
40     auto i = St.rbegin();
41     for (; i != St.rend(); ++i) {
42         St[i->fail].cnt += i->cnt;
43     }
44 }
45 inline int size() { // The number of diff. pal.
46     return SZ(St) - 2;
47 }
48 };

```

5.12 MainLorentz [2981c4]

```

1  vector<pair<int, int>> rep[kN]; // 0-base [l, r]
2  void main_lorentz(const string &s, int sft = 0) {
3      const int n = s.size();
4      if (n == 1) return;
5      const int nu = n / 2, nv = n - nu;
6      const string u = s.substr(0, nu), v = s.substr(nu),
7          ru(u.rbegin(), u.rend()), rv(v.rbegin(), v.rend());
8      main_lorentz(u, sft), main_lorentz(v, sft + nu);
9      const auto z1 = Zalgo(ru), z2 = Zalgo(v + '#' + u),
10         z3 = Zalgo(ru + '#' + rv), z4 = Zalgo(v);
11      auto get_z = [](const vector<int> &z, int i) {
12          return
13              (0 <= i && i < (int)z.size()) ? z[i] : 0;
14      };
15      auto add_rep
16          = [&](bool left, int c, int l, int k1, int k2) {
17          const
18              int L = max(1, l - k2), R = min(l - left, k1);
19          if (L > R) return;
20          if (left)
21              rep[l].emplace_back(sft + c - R, sft + c - L);
22          else
23              rep[l].emplace_back
24                  (sft + c - R - l + 1, sft + c - L - l + 1);
25      };
26      for (int cntr = 0; cntr < n; cntr++) {
27          int l, k1, k2;
28          if (cntr < nu) {
29              l = nu - cntr;
30              k1 = get_z(z1, nu - cntr);
31              k2 = get_z(z2, nv + 1 + cntr);
32          } else {
33              l = cntr - nu + 1;
34              k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
35              k2 = get_z(z4, (cntr - nu) + 1);
36          }
37          if (k1 + k2 >= l)
38              add_rep(cntr < nu, cntr, l, k1, k2);
39      }
40  } // p | in [l, r] => s[p, p + i) = s[p + i, p + 2i)

```

5.13 KMP [32f229]

```

1  int F[MAXN];
2  vector<int> match(string A, string B) {
3      vector<int> ans;
4      F[0] = -1, F[1] = 0;
5      for (int i = 1, j = 0; i < SZ(B); F[++i] = ++j) {
6          if (B[i] == B[j]) F[i] = F[j]; // optimize
7          while (j != -1 && B[i] != B[j]) j = F[j];
8      }
9      for (int i = 0, j = 0; i < SZ(A); ++i) {
10         while (j != -1 && A[i] != B[j]) j = F[j];
11         if (++j == SZ(B)) ans.pb(i + 1 - j), j = F[j];
12     }
13     return ans;
14 }

```

5.14 Suffix Array [b981d5]

```

1  struct suffix_array {
2      int box[MAXN], tp[MAXN], m;
3      bool not_equ(int a, int b, int k, int n) {
4          return ra[a] != ra[b] || a + k >= n ||
5              b + k >= n || ra[a + k] != ra[b + k];
6      }
7      void radix(int *key, int *it, int *ot, int n) {

```

```

8   fill_n(box, m, 0);
9   for (int i = 0; i < n; ++i) ++box[key[i]];
10  partial_sum(box, box + m, box);
11  for (int i = n - 1; i >= 0; --i)
12    ot[--box[key[it[i]]]] = it[i];
13  }
14  void make_sa(const string &s, int n) {
15    int k = 1;
16    for (int i = 0; i < n; ++i) ra[i] = s[i];
17    do {
18      iota(tp, tp + k, n - k), iota(sa + k, sa + n, 0);
19      radix(ra + k, sa + k, tp + k, n - k);
20      radix(ra, tp, sa, n);
21      tp[sa[0]] = 0, m = 1;
22      for (int i = 1; i < n; ++i) {
23        m += not_equ(sa[i], sa[i - 1], k, n);
24        tp[sa[i]] = m - 1;
25      }
26      copy_n(tp, n, ra);
27      k *= 2;
28    } while (k < n && m != n);
29  }
30  void make_he(const string &s, int n) {
31    for (int j = 0, k = 0; j < n; ++j) {
32      if (ra[j])
33        for (; s[j + k] == s[sa[ra[j]] - 1] + k; ++k)
34          ;
35      he[ra[j]] = k, k = max(0, k - 1);
36    }
37  }
38  int sa[MAXN], ra[MAXN], he[MAXN];
39  void build(const string &s) {
40    int n = SZ(s);
41    fill_n
42      (sa, n, 0), fill_n(ra, n, 0), fill_n(he, n, 0);
43    fill_n(box, n, 0), fill_n(tp, n, 0), m = 256;
44    make_sa(s, n), make_he(s, n);
45  }
};

```

6 Math

6.1 chineseRemainder [0e2467]

```

1  ll solve(ll x1, ll m1, ll x2, ll m2) {
2    ll g = gcd(m1, m2);
3    if ((x2 - x1) % g) return -1; // no sol
4    m1 /= g; m2 /= g;
5    pll p = exgcd(m1, m2);
6    ll lcm = m1 * m2 * g;
7    ll res = p.first * (x2 - x1) * m1 + x1;
8    // be careful with overflow
9    return (res % lcm + lcm) % lcm;
10 }

```

6.2 PiCount [29fb4b]

```

1  ll PrimeCount(ll n) { // n ~ 10^13 => < 2s
2    if (n <= 1) return 0;
3    int v = sqrt(n), s = (v + 1) / 2, pc = 0;
4    vector<int> smalls(v + 1), skip(v + 1), roughs(s);
5    vector<ll> larges(s);
6    for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
7    for (int i = 0; i < s; ++i) {
8      roughs[i] = 2 * i + 1;
9      larges[i] = (n / (2 * i + 1) + 1) / 2;
10   }
11   for (int p = 3; p <= v; ++p) {
12     if (smalls[p] > smalls[p - 1]) {
13       int q = p * p;
14       ++pc;
15       if (1LL * q * q > n) break;
16       skip[p] = 1;
17       for (int i = q; i <= v; i += 2 * p) skip[i] = 1;
18       int ns = 0;
19       for (int k = 0; k < s; ++k) {
20         int i = roughs[k];
21         if (skip[i]) continue;
22         ll d = 1LL * i * p;
23         larges[ns] = larges[k] - (d <= v ? larges
24           [smalls[d] - pc] : smalls[n / d]) + pc;
25         roughs[ns++] = i;
26       }
27       s = ns;
28       for (int j = v / p; j >= p; --j) {
29         int c =
30           smalls[j] - pc, e = min(j * p + p, v + 1);

```

```

29         for (int i = j * p; i < e; ++i) smalls[i] -= c;
30       }
31     }
32   }
33   for (int k = 1; k < s; ++k) {
34     const ll m = n / roughs[k];
35     ll t = larges[k] - (pc + k - 1);
36     for (int l = 1; l < k; ++l) {
37       int p = roughs[l];
38       if (1LL * p * p > m) break;
39       t -= smalls[m / p] - (pc + l - 1);
40     }
41     larges[0] -= t;
42   }
43   return larges[0];
44 }

```

6.3 numbers

- 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. $\pi(j) > j$.

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

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

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

6.4 Estimation

n	2	3	4	5	6	7	8	9	20	30	40	50	100		
$p(n)$	2	3	5	7	11	15	22	30	627	5604	4e4	2e5	2e8		
n	100	1e3	1e6	1e9	1e12	1e15	1e18								
$d(i)$	12	32	240	1344	6720	26880	103680								
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\binom{2n}{n}$	2	6	20	70	252	924	3432	12870	48620	184756	7e5	2e6	1e7	4e7	1.5e8
n	2	3	4	5	6	7	8	9	10	11	12	13			
B_n	2	5	15	52	203	877	4140	21147	115975	7e5	4e6	3e7			

6.5 floor sum [76c575]

```

1  ll floor_sum(ll n, ll m, ll a, ll b) {
2    ll ans = 0;
3    if (a >= m)
4      ans += (n - 1) * n * (a / m) / 2, a %= m;
5    if (b >= m)
6      ans += n * (b / m), b %= m;
7    ll y_max
8      = (a * n + b) / m, x_max = (y_max * m - b);
9    if (y_max == 0) return ans;
10   ans += (n - (x_max + a - 1) / a) * y_max;
11   ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
12   return ans;
13 } // sum^{n-1}_0 floor((a * i + b) / m) in log(n + m + a + b)

```

6.6 QuadraticResidue [0b50c4]

```

1  int Jacobi(int a, int m) {
2    int s = 1;
3    for (; m > 1; ) {
4      a %= m;
5      if (a == 0) return 0;
6      const int r = __builtin_ctz(a);
7      if ((r & 1) && ((m + 2) & 4)) s = -s;
8      a >>= r;
9      if (a & m & 2) s = -s;
10     swap(a, m);
11   }

```

```

12     return s;
13 }
14
15 int QuadraticResidue(int a, int p) {
16     if (p == 2) return a & 1;
17     const int jc = Jacobi(a, p);
18     if (jc == 0) return 0;
19     if (jc == -1) return -1;
20     int b, d;
21     for (; ; ) {
22         b = rand() % p;
23         d = (1LL * b * b + p - a) % p;
24         if (Jacobi(d, p) == -1) break;
25     }
26     int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
27     for (int e = (1LL + p) >> 1; e; e >>= 1) {
28         if (e & 1) {
29             tmp = (1LL *
30                 g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p;
31             g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
32             g0 = tmp;
33         }
34         tmp = (1LL
35             * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
36         f1 = (2LL * f0 * f1) % p;
37         f0 = tmp;
38     }
39     return g0;
40 }

```

6.7 floor enumeration [fc55c8]

```

1 // enumerating x = floor(n / i), [l, r]
2 for (int l = 1, r; l <= n; l = r + 1) {
3     int x = n / l;
4     r = n / x;
5 }

```

6.8 ax+by=gcd [43bd81]

```

1 pll exgcd(ll a, ll b) {
2     if (b == 0) return pll(1, 0);
3     ll p = a / b;
4     pll q = exgcd(b, a % b);
5     return pll(q.Y, q.X - q.Y * p);
6 }
7 /* ax+by=res, let x be minimum non-negative
8 g, p = gcd(a, b), exgcd(a, b) * res / g
9 if p.X < 0: t = (abs(p.X) + b / g - 1) / (b / g)
10 else: t = -(p.X / (b / g))
11 p += (b / g, -a / g) * t */

```

6.9 cantor expansion [2d801a]

```

1 #define MAXN 11
2 int factorial[MAXN];
3 inline void init(){
4     factorial[0]=1;
5     for(int i=1;i<=MAXN;++i){
6         factorial[i]=factorial[i-1]*i;
7     }
8 }
9 inline int encode(const std::vector<int> &s){
10     int n=s.size(),res=0;
11     for(int i=0;i<n;++i){
12         int t=0;
13         for(int j=i+1;j<n;++j){
14             if(s[j]<s[i])++t;
15         }
16         res+=t*factorial[n-i-1];
17     }
18     return res;
19 }
20 inline std::vector<int> decode(int a,int n){
21     std::vector<int> res;
22     std::vector<bool> vis(n,0);
23     for(int i=n-1;i>=0;--i){
24         int t=a/factorial[i],j;
25         for(j=0;j<n;++j){
26             if(!vis[j]){
27                 if(t==0)break;
28                 --t;
29             }
30         }
31         res.push_back(j);
32         vis[j]=1;
33         a%=factorial[i];

```

```

34     }
35     return res;
36 }

```

6.10 Generating function

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

6.11 Fraction [666134]

```

1 struct fraction {
2     ll n, d;
3     fraction
4         (const ll &n=0, const ll &d=1): n(_n), d(_d) {
5         ll t = gcd(n, d);
6         n /= t, d /= t;
7         if (d < 0) n = -n, d = -d;
8     }
9     fraction operator-(const fraction &b) const
10     { return fraction(-n, d); }
11     fraction operator+(const fraction &b) const
12     { return fraction(n * b.d + b.n * d, d * b.d); }
13     fraction operator-(const fraction &b) const
14     { return fraction(n * b.d - b.n * d, d * b.d); }
15     fraction operator*(const fraction &b) const
16     { return fraction(n * b.n, d * b.d); }
17     fraction operator/(const fraction &b) const
18     { return fraction(n * b.d, d * b.n); }
19     void print() {
20         cout << n;
21         if (d != 1) cout << "/" << d;
22     }
23 };

```

6.12 Gaussian gcd [616465]

```

1 cpx gaussian_gcd(cpx a, cpx b) {
2     #define rnd
3     (a, b) ((a >= 0 ? a * 2 + b : a * 2 - b) / (b * 2))
4     ll c = a.real() * b.real() + a.imag() * b.imag();
5     ll d = a.imag() * b.real() - a.real() * b.imag();
6     ll r = b.real() * b.real() + b.imag() * b.imag();
7     if (c % r == 0 && d % r == 0) return b;
8     return gaussian_gcd
9         (b, a - cpx(rnd(c, r), rnd(d, r)) * b);
10 }

```

6.13 Theorem

- 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

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

- 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 $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$ 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

$$d_1 + \dots + d_n \text{ is even and } \sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k) \text{ holds for every } 1 \leq k \leq n.$$

Gale–Ryser theorem

A pair of sequences of nonnegative integers $a_1 \geq \dots \geq a_n$ and b_1, \dots, b_n

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

Fulkerson–Chen–Anstee theorem

A sequence $(a_1, b_1), \dots, (a_n, b_n)$ of nonnegative integer pairs

with $a_1 \geq \dots \geq a_n$ is digraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and

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

Pick's theorem

For simple polygon, when points are all integer, we have $A = \#\{\text{lattice points in the interior}\} + \frac{\#\{\text{lattice points on the boundary}\}}{2} - 1$.

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 $= 2\pi r h = \pi(a^2 + h^2) = 2\pi r^2(1 - \cos \theta)$.

Lagrange multiplier

- Optimize $f(x_1, \dots, x_n)$ when k constraints $g_i(x_1, \dots, x_n) = 0$.
- Lagrangian function $\mathcal{L}(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k) = f(x_1, \dots, x_n) - \sum_{i=1}^k \lambda_i g_i(x_1, \dots, x_n)$.
- The solution corresponding to the original constrained optimization is always a saddle point of the Lagrangian function.

Nearest points of two skew lines

- Line 1: $v_1 = p_1 + t_1 d_1$
- Line 2: $v_2 = p_2 + t_2 d_2$
- $n = d_1 \times d_2$
- $n_1 = d_1 \times n$
- $n_2 = d_2 \times n$
- $c_1 = p_1 + \frac{(p_2 - p_1) \cdot n_2}{d_1 \cdot n_2} d_1$
- $c_2 = p_2 + \frac{(p_1 - p_2) \cdot n_1}{d_2 \cdot n_1} d_2$

Derivatives/Integrals

Integration by parts: $\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$

$$\left| \begin{aligned} \frac{d}{dx} \sin^{-1} x &= \frac{1}{\sqrt{1-x^2}} & \frac{d}{dx} \cos^{-1} x &= -\frac{1}{\sqrt{1-x^2}} & \frac{d}{dx} \tan^{-1} x &= \frac{1}{1+x^2} \\ \frac{d}{dx} \tan x &= 1 + \tan^2 x & \int \tan x dx &= -\frac{\ln|\cos x|}{1} \\ \int e^{-x^2} dx &= \frac{\sqrt{\pi}}{2} \operatorname{erf}(x) & \int x e^{ax} dx &= \frac{e^{ax}}{a^2} (ax - 1) \\ \int \sqrt{a^2 + x^2} dx &= \frac{1}{2} (x\sqrt{a^2 + x^2} + a^2 \operatorname{asinh}(x/a)) \end{aligned} \right|$$

Spherical Coordinate

$$(x, y, z) = (r \sin \theta \cos \phi, r \sin \theta \sin \phi, r \cos \theta)$$

$$(r, \theta, \phi) = (\sqrt{x^2 + y^2 + z^2}, \arccos(z/\sqrt{x^2 + y^2 + z^2}), \operatorname{atan2}(y, x))$$

Rotation Matrix

$$M(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}, R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

6.14 Determinant [a4d696]

```

1 struct Matrix {
2     int n, m;
3     ll M[MAXN][MAXN];
4     int row_swap(int i, int j) {
5         if (i == j) return 0;
6         for (int k = 0; k < m; ++k)
7             swap(M[i][k], M[j][k]);
8         return 1;
9     }
10    ll det() { // return the number of swaps
11        int rt = 0;
12        for (int i = 0; i < n; ++i) {
13            int piv = i;
14            while (piv < n && !M[piv][i]) ++piv;
15            if (piv == n) continue;
16            rt += row_swap(i, piv);
17            for (int j = i + 1; j < n; ++j) {
18                while (M[j][i]) {
19                    int tmp = P - M[i][i] / M[j][i];
20                    for (int k = i; k < m; ++k)
21                        M[i][k] = (M[j][k] * tmp + M[i][k]) % P;
22                    rt += row_swap(i, j);
23                }
24            }
25        }
26        rt = (rt & 1) ? P - 1 : 1;
27        for (int i = 0; i < n; ++i)
28            rt = rt * M[i][i] % P;
29        return rt;
30    } // round(rt) if using double to cal. int. det
31 }
32 
```

6.15 ModMin [05065e]

```

1 // min{k | l <= ((ak) mod m) <= r}, no solution -> -1
2 ll mod_min(ll a, ll m, ll l, ll r) {
3     if (a == 0) return l ? -1 : 0;
4     if (ll k = (l + a - 1) / a; k * a <= r)
5         return k;
6     ll b = m / a, c = m % a;
7     if (ll y = mod_min(c, a, a - r % a, a - l % a))
8         return (l + y * c + a - 1) / a + y * b;
9     return -1;
10 }

```

6.16 Primes [2464ae]

```

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

```

6.17 Pollard Rho [a5802e]

```

1 map<ll, int> cnt;
2 void PollardRho(ll n) {
3     if (n == 1) return;
4     if (prime(n)) return ++cnt[n], void();
5     if (n % 2
6         == 0) return PollardRho(n / 2), ++cnt[2], void();
7     ll x = 2, y = 2, d = 1, p = 1;
8     #define f(x, n, p) ((mul(x, x, n) + p) % n)
9     while (true) {
10        if (d != n && d != 1) {
11            PollardRho(n / d);
12            PollardRho(d);
13            return;
14        }
15        if (d == n) ++p;
16        x = f(x, n, p), y = f(f(y, n, p), n, p);
17        d = gcd(abs(x - y), n);
18    }
19 }

```

6.18 Simultaneous Equations [b8b03f]

```

1 struct matrix { //m variables, n equations
2     int n, m;
3     fraction M[MAXN][MAXN + 1], sol[MAXN];
4     int solve() { // -1: inconsistent, >= 0: rank
5         for (int i = 0; i < n; ++i) {
6             int piv = 0;

```

```

7   while (piv < m && !M[i][piv].n) ++piv;
8   if (piv == m) continue;
9   for (int j = 0; j < n; ++j) {
10      if (i == j) continue;
11      fraction tmp = -M[j][piv] / M[i][piv];
12      for (int k = 0; k <=
13         m; ++k) M[j][k] = tmp * M[i][k] + M[j][k];
14   }
15   int rank = 0;
16   for (int i = 0; i < n; ++i) {
17      int piv = 0;
18      while (piv < m && !M[i][piv].n) ++piv;
19      if (piv == m && M[i][m].n) return -1;
20      else if (piv
21         < m) ++rank, sol[piv] = M[i][m] / M[i][piv];
22   }
23   return rank;
24 };

```

6.19 Big number [1c17ab]

```

1  template<typename T>
2  inline string to_string(const T& x){
3      stringstream ss;
4      return ss<<x,ss.str();
5  }
6  struct bigN:vector<ll>{
7      const static int base=1000000000,width=log10(base);
8      bool negative;
9      bigN(const_iterator
10         a,const_iterator b):vector<ll>(a,b){}
11      bigN(string s){
12         if(s.empty())return;
13         if(s[0]=='-')negative=1,s=s.substr(1);
14         else negative=0;
15         for(int i=int(s.size())-1;i>=0;i-=width){
16             ll t=0;
17             for(int j=max(0,i-width+1);j<=i;++j)
18                 t=t*10+s[j]-'0';
19             push_back(t);
20         }
21         trim();
22     }
23     template<typename T>
24     bigN(const T &x):bigN(to_string(x)){}
25     bigN():negative(0){}
26     void trim(){
27         while(size()&&!back())pop_back();
28         if(empty())negative=0;
29     }
30     void carry(int _base=base){
31         for(size_t i=0;i<size();++i){
32             if(at(i)>=0&&at(i)<_base)continue;
33             if(i+1u==size())push_back(0);
34             int r=at(i)%_base;
35             if(r<0)r+=_base;
36             at(i+1)+=(at(i)-r)/_base,at(i)=r;
37         }
38     }
39     int abscmp(const bigN &b)const{
40         if(size()>b.size())return 1;
41         if(size()<b.size())return -1;
42         for(int i=int(size())-1;i>=0;--i){
43             if(at(i)>b[i])return 1;
44             if(at(i)<b[i])return -1;
45         }
46         return 0;
47     }
48     int cmp(const bigN &b)const{
49         if(negative!=b.negative)return negative?-1:1;
50         return negative?-abscmp(b):abscmp(b);
51     }
52     bool operator<(const bigN&b)const{return cmp(b)<0;}
53     bool operator>(const bigN&b)const{return cmp(b)>0;}
54     bool operator<=(const bigN&b)const{return cmp(b)<=0;}
55     bool operator>=(const bigN&b)const{return cmp(b)>=0;}
56     bool operator==(const bigN&b)const{return !cmp(b);}
57     bool operator!=(const bigN&b)const{return cmp(b)!=0;}
58     bigN abs()const{
59         bigN res=*this;
60         return res.negative=0, res;
61     }
62     bigN operator-(const bigN &b)const{
63         bigN res=*this;
64         return res.negative=!negative,res.trim(),res;

```

```

65     }
66     bigN operator+(const bigN &b)const{
67         if(negative)return -(-(*this)+(-b));
68         if(b.negative)return *this-(-b);
69         bigN res=*this;
70         if(b.size()>size())res.resize(b.size());
71         for(size_t i=0;i<b.size();++i)res[i]+=b[i];
72         return res.trim(),res;
73     }
74     bigN operator-(const bigN &b)const{
75         if(negative)return -(-(*this)-(-b));
76         if(b.negative)return *this+(-b);
77         if(abscmp(b)<0)return -(b-(*this));
78         bigN res=*this;
79         if(b.size()>size())res.resize(b.size());
80         for(size_t i=0;i<b.size();++i)res[i]-=b[i];
81         return res.trim(),res;
82     }
83     bigN operator*(const bigN &b)const{
84         bigN res;
85         res.negative=negative!=b.negative;
86         res.resize(size()+b.size());
87         for(size_t i=0;i<size();++i)
88             for(size_t j=0;j<b.size();++j)
89                 if((res[i+j]+=at(i)*b[j])>=base){
90                     res[i+j+1]+=res[i+j]/base;
91                     res[i+j]%=base;
92                     // %^k%carry · | · ,!
93                 }
94         return res.trim(),res;
95     }
96     bigN operator/(const bigN &b)const{
97         int norm=base/(b.back()+1);
98         bigN x=abs()*norm;
99         bigN y=b.abs()*norm;
100         bigN q,r;
101         q.resize(x.size());
102         for(int i=int(x.size())-1;i>=0;--i){
103             r=r*base+x[i];
104             int s1=r.size()<=y.size()?0:r[y.size()];
105             int s2=r.size()<y.size()?0:r[y.size()-1];
106             int d=(ll(base)*s1+s2)/y.back();
107             r=r-y*d;
108             while(r.negative)r=r+y,--d;
109             q[i]=d;
110         }
111         q.negative=negative!=b.negative;
112         return q.trim(),q;
113     }
114     bigN operator%(const bigN &b)const{
115         return *this-(*this/b)*b;
116     }
117     friend istream& operator>>(istream &ss,bigN &b){
118         string s;
119         return ss>>s, b=s, ss;
120     }
121     friend
122     ostream& operator<<(ostream &ss,const bigN &b){
123         if(b.negative)ss<<"-";
124         ss<<(b.empty()?0:b.back());
125         for(int i=int(b.size())-2;i>=0;--i)
126             ss<<setw(width)<<setfill('0')<<b[i];
127         return ss;
128     }
129     template<typename T>
130     operator T(){
131         stringstream ss;
132         ss<<*this;
133         T res;
134         return ss>>res,res;
135     }
136 };

```

6.20 Euclidean

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

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

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

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

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

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

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

6.21 Miller Rabin [969881]

```

1 // n < 4,759,123,141      3 : 2, 7, 61
2 // n < 1,122,004,669,633 4 : 2, 13, 23, 1662803
3 // n < 3,474,749,660,383 6 : primes <= 13
4 // n < 2^64              7 :
5 // 2, 325, 9375, 28178, 450775, 9780504, 1795265022
6 bool Miller_Rabin(ll a, ll n) {
7     if ((a = a % n) == 0) return 1;
8     if (n % 2 == 0) return n == 2;
9     ll tmp = (n - 1) / ((n - 1) & (1 - n));
10    ll t = __lg(((n - 1) & (1 - n))), x = 1;
11    for (; tmp; tmp >>= 1, a = mul(a, a, n))
12        if (tmp & 1) x = mul(x, a, n);
13    if (x == 1 || x == n - 1) return 1;
14    while (--t)
15        if ((x = mul(x, x, n)) == n - 1) return 1;
16    return 0;
17 }

```

6.22 Berlekamp-Massey [cdb091]

```

1 template <typename T>
2 vector<T> BerlekampMassey(const vector<T> &output) {
3     vector<T> d(SZ(output) + 1), me, he;
4     for (int f = 0, i = 1; i <= SZ(output); ++i) {
5         for (int j = 0; j < SZ(me); ++j)
6             d[i] += output[i - j - 1] * me[j];
7         if ((d[i] - output[i - 1]) == 0) continue;
8         if (me.empty()) {
9             me.resize(f = i);
10            continue;
11        }
12        vector<T> o(i - f - 1);
13        T k = -d[i] / d[f]; o.pb(-k);
14        for (T x : he) o.pb(x * k);
15        o.resize(max(SZ(o), SZ(me)));
16        for (int j = 0; j < SZ(me); ++j) o[j] += me[j];
17        if (i - f + SZ(he) >= SZ(me)) he = me, f = i;
18        me = o;
19    }
20    return me;
21 }

```

6.23 floor ceil [f84849]

```

1 int floor(int a, int b)
2 { return a / b - (a % b && (a < 0) ^ (b < 0)); }
3 int ceil(int a, int b)
4 { return a / b + (a % b && (a < 0) ^ (b > 0)); }

```

6.24 fac no p [86ad89]

```

1 // O(p^k + log^2 n), pk = p^k
2 ll prod[MAXP];
3 ll fac_no_p(ll n, ll p, ll pk) {
4     prod[0] = 1;
5     for (int i = 1; i <= pk; ++i)
6         if (i % p) prod[i] = prod[i - 1] * i % pk;
7         else prod[i] = prod[i - 1];
8     ll rt = 1;
9     for (; n; n /= p) {
10        rt = rt * mpow(prod[pk], n / pk, pk) % pk;
11        rt = rt * prod[n % pk] % pk;
12    }
13    return rt;
14 } // (n! without factor p) % p^k

```

6.25 DiscreteLog [21f791]

```

1 int DiscreteLog(int s, int x, int y, int m) {
2     constexpr int kStep = 32000;
3     unordered_map<int, int> p;
4     int b = 1;
5     for (int i = 0; i < kStep; ++i) {
6         p[y] = i;
7         y = 1LL * y * x % m;
8         b = 1LL * b * x % m;
9     }
10    for (int i = 0; i < m + 10; i += kStep) {
11        s = 1LL * s * b % m;
12        if (p.find(s) != p.end()) return i + kStep - p[s];
13    }
14    return -1;
15 }
16 int DiscreteLog(int x, int y, int m) {
17     if (m == 1) return 0;
18     int s = 1;
19     for (int i = 0; i < 100; ++i) {
20         if (s == y) return i;
21         s = 1LL * s * x % m;
22     }
23     if (s == y) return 100;
24     int p = 100 + DiscreteLog(s, x, y, m);
25     if (fpow(x, p, m) != y) return -1;
26     return p;
27 }

```

6.26 SimplexConstruction

Primal	Dual
Maximize $c^T x$ s.t. $Ax \leq b, x \geq 0$	Minimize $b^T y$ s.t. $A^T y \geq c, y \geq 0$
Maximize $c^T x$ s.t. $Ax \leq b$	Minimize $b^T y$ s.t. $A^T y = c, y \geq 0$
Maximize $c^T x$ s.t. $Ax = b, x \geq 0$	Minimize $b^T y$ s.t. $A^T y \geq c$

\bar{x} and \bar{y} are optimal if and only if for all $i \in [1, n]$, either $\bar{x}_i = 0$ or $\sum_{j=1}^m A_{ji} \bar{y}_j = c_i$ holds and for all $i \in [1, m]$ either $\bar{y}_i = 0$ or $\sum_{j=1}^n A_{ij} \bar{x}_j = b_j$ holds.

- In case of minimization, let $c'_i = -c_i$
- $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j \rightarrow \sum_{1 \leq i \leq n} -A_{ji} x_i \leq -b_j$
- $\sum_{1 \leq i \leq n} A_{ji} x_i = b_j$
 - $\sum_{1 \leq i \leq n} A_{ji} x_i \leq b_j$
 - $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j$
- If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.27 Simplex Algorithm [ad99b3]

```

1 const int MAXN = 11000, MAXM = 405;
2 const double eps = 1E-10;
3 double a[MAXN][MAXM], b[MAXN], c[MAXN];
4 double d[MAXN][MAXM], x[MAXN];
5 int ix[MAXN + MAXM]; // !!! array all indexed from 0
6 // max{cx} subject to {Ax <= b, x >= 0}
7 // n: constraints, m: vars !!!
8 // x[] is the optimal solution vector
9 // usage :
10 // value = simplex(a, b, c, N, M);
11 double simplex(int n, int m) {
12     ++m;
13     fill_n(d[n], m + 1, 0);
14     fill_n(d[n + 1], m + 1, 0);
15     iota(ix, ix + n + m, 0);
16     int r = n, s = m - 1;
17     for (int i = 0; i < n; ++i) {
18         for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
19         d[i][m - 1] = 1;
20         d[i][m] = b[i];
21         if (d[r][m] > d[i][m]) r = i;
22     }
23     copy_n(c, m - 1, d[n]);
24     d[n + 1][m - 1] = -1;
25     for (double dd; ) {
26         if (r < n) {
27             swap(ix[s], ix[r + m]);
28             d[r][s] = 1.0 / d[r][s];
29             for (int j = 0; j <= m; ++j)
30                 if (j != s) d[r][j] *= -d[r][s];
31             for (int i = 0; i <= n + 1; ++i) if (i != r) {
32                 for (int j = 0; j <= m; ++j) if (j != s)
33                     d[i][j] += d[r][j] * d[i][s];
34                 d[i][s] *= d[r][s];
35             }
36         }
37         r = s = -1;
38         for (int j = 0; j < m; ++j)
39             if (s < 0 || ix[s] > ix[j]) {

```



```

40     if (d[n + 1][j] > eps ||
41         (d[n + 1][j] > -eps && d[n][j] > eps))
42         s = j;
43     }
44     if (s < 0) break;
45     for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
46         if (r < 0 ||
47             (dd = d[r][m]
48              / d[r][s] - d[i][m] / d[i][s]) < -eps ||
49             (dd < eps && ix[r + m] > ix[i + m]))
50             r = i;
51     }
52     if (r < 0) return -1; // not bounded
53 }
54 double ans = 0;
55 fill_n(x, m, 0);
56 for (int i = m; i <
57     n + m; ++i) { // the missing enumerated x[i] = 0
58     if (ix[i] < m - 1) {
59         ans += d[i - m][m] * c[ix[i]];
60         x[ix[i]] = d[i - m][m];
61     }
62 }
63 return ans;

```

6.28 SchreierSims [b17b78]

```

1 namespace schreier {
2 int n;
3 vector<vector<vector<int>>> bkets, binv;
4 vector<vector<int>> lk;
5 vector<int> operator
6     *(const vector<int> &a, const vector<int> &b) {
7     vector<int> res(SZ(a));
8     for (int i = 0; i < SZ(a); ++i) res[i] = b[a[i]];
9     return res;
10 }
11 vector<int> inv(const vector<int> &a) {
12     vector<int> res(SZ(a));
13     for (int i = 0; i < SZ(a); ++i) res[a[i]] = i;
14     return res;
15 }
16 int filter(const vector<int> &g, bool add = true) {
17     n = SZ(bkets);
18     vector<int> p = g;
19     for (int i = 0; i < n; ++i) {
20         assert(p[i] >= 0 && p[i] < SZ(lk[i]));
21         if (lk[i][p[i]] == -1) {
22             if (add) {
23                 bkts[i].pb(p);
24                 binv[i].pb(inv(p));
25                 lk[i][p[i]] = SZ(bkets[i]) - 1;
26             }
27             return i;
28         }
29         p = p * binv[i][lk[i][p[i]]];
30     }
31     return -1;
32 }
33 bool inside(const
34     vector<int> &g) { return filter(g, false) == -1; }
35 void solve(const vector<vector<int>> &gen, int _n) {
36     n = _n;
37     bkts.clear(), bkts.resize(n);
38     binv.clear(), binv.resize(n);
39     lk.clear(), lk.resize(n);
40     vector<int> iden(n);
41     iota(iden.begin(), iden.end(), 0);
42     for (int i = 0; i < n; ++i) {
43         lk[i].resize(n, -1);
44         bkts[i].pb(iden);
45         binv[i].pb(iden);
46         lk[i][i] = 0;
47     }
48     for (int i = 0; i < SZ(gen); ++i) filter(gen[i]);
49     queue<pair<pii, pii>> upd;
50     for (int i = 0; i < n; ++i)
51         for (int j = i; j < n; ++j)
52             for (int k = 0; k < SZ(bkets[i]); ++k)
53                 for (int l = 0; l < SZ(bkets[j]); ++l)
54                     upd.emplace(pii(i, k), pii(j, l));
55     while (!upd.empty()) {
56         auto a = upd.front().X;
57         auto b = upd.front().Y;
58         upd.pop();

```

```

57     int res = filter(bkets[a.X][a.Y] * bkts[b.X][b.Y]);
58     if (res == -1) continue;
59     pii pr = pii(res, SZ(bkets[res]) - 1);
60     for (int i = 0; i < n; ++i)
61         for (int j = 0; j < SZ(bkets[i]); ++j) {
62             if (i <= res) upd.emplace(pii(i, j), pr);
63             if (res <= i) upd.emplace(pr, pii(i, j));
64         }
65     }
66 }
67 ll size() {
68     ll res = 1;
69     for (int i = 0; i < n; ++i) res = res * SZ(bkets[i]);
70     return res;
71 }

```

7 Polynomial

7.1 Polynomial Operation [dcba1b]

```

1 #define
2     fi(s, n) for (int i = (int)(s); i < (int)(n); ++i)
3 template<int MAXN, ll P, ll RT> // MAXN = 2^k
4 struct Poly : vector<ll> { // coefficients in [0, P)
5     using vector<ll>::vector;
6     static NTT<MAXN, P, RT> ntt;
7     int n() const { return (int)size(); } // n() >= 1
8     Poly(const Poly &p, int m) : vector<ll>(m) {
9         copy_n(p.data(), min(p.n(), m), data());
10    }
11    Poly& irev() {
12        return reverse(data(), data() + n()), *this; }
13    Poly& isz(int m) { return resize(m), *this; }
14    Poly& iadd(const Poly &rhs) { // n() == rhs.n()
15        fi(0, n()) if
16            (((*this)[i] += rhs[i]) >= P) (*this)[i] -= P;
17        return *this;
18    }
19    Poly& imul(ll k) {
20        fi(0, n()) (*this)[i] = (*this)[i] * k % P;
21        return *this;
22    }
23    Poly Mul(const Poly &rhs) const {
24        int m = 1;
25        while (m < n() + rhs.n() - 1) m <= 1;
26        Poly X(*this, m), Y(rhs, m);
27        ntt(X.data(), m), ntt(Y.data(), m);
28        fi(0, m) X[i] = X[i] * Y[i] % P;
29        ntt(X.data(), m, true);
30        return X.isz(n() + rhs.n() - 1);
31    }
32    Poly Inv() const { // (*this)[0] != 0, 1e5/95ms
33        if (n() == 1) return {ntt.minv((*this)[0])};
34        int m = 1;
35        while (m < n() * 2) m <= 1;
36        Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(m);
37        Poly Y(*this, m);
38        ntt(Xi.data(), m), ntt(Y.data(), m);
39        fi(0, m) {
40            Xi[i] *= (2 - Xi[i] * Y[i]) % P;
41            if ((Xi[i] % P) < 0) Xi[i] += P;
42        }
43        ntt(Xi.data(), m, true);
44        return Xi.isz(n());
45    }
46    Poly Sqrt()
47        const { // Jacobi((*this)[0], P) = 1, 1e5/235ms
48        if (n()
49            == 1) return {QuadraticResidue((*this)[0], P)};
50        Poly
51            X = Poly(*this, (n() + 1) / 2).Sqrt().isz(n());
52        return
53            X.iadd(Mul(X.Inv()).isz(n())).imul(P / 2 + 1);
54    }
55    pair<Poly, Poly> DivMod
56        (const Poly &rhs) const { // (rhs.)back() != 0
57        if (n() < rhs.n()) return {0, *this};
58        const int m = n() - rhs.n() + 1;
59        Poly X(rhs); X.irev().isz(m);
60        Poly Y(*this); Y.irev().isz(m);
61        Poly Q = Y.Mul(X.Inv()).isz(m).irev();
62        X = rhs.Mul(Q), Y = *this;
63        fi(0, n()) if ((Y[i] -= X[i]) < 0) Y[i] += P;
64        return {Q, Y.isz(max(1, rhs.n() - 1))};
65    }
66    Poly Dx() const {
67        Poly ret(n() - 1);

```

```

60     fi(0,
        ret.n()) ret[i] = (i + 1) * (*this)[i + 1] % P;
61     return ret.isz(max(1, ret.n()));
62 }
63 Poly Sx() const {
64     Poly ret(n() + 1);
65     fi(0, n())
66         ret[i + 1] = ntt.minv(i + 1) * (*this)[i] % P;
67     return ret;
68 }
69 Poly _tmul(int nn, const Poly &rhs) const {
70     Poly Y = Mul(rhs).isz(n() + nn - 1);
71     return Poly(Y.data() + n() - 1, Y.data() + Y.n());
72 }
73 vector<ll> _eval(const
74     vector<ll> &x, const vector<Poly> &up) const {
75     const int m = (int)x.size();
76     if (!m) return {};
77     vector<Poly> down(m * 2);
78     // down[1] = DivMod(up[1]).second;
79     // fi(2, m *
80         2) down[i] = down[i / 2].DivMod(up[i]).second;
81     down[1] = Poly(up[1])
82         .irev().isz(n()).Inv().irev()._tmul(m, *this);
83     fi(2, m * 2) down[i]
84         = up[i ^ 1]._tmul(up[i].n() - 1, down[i / 2]);
85     vector<ll> y(m);
86     fi(0, m) y[i] = down[m + i][0];
87     return y;
88 }
89 static vector<Poly> _tree1(const vector<ll> &x) {
90     const int m = (int)x.size();
91     vector<Poly> up(m * 2);
92     fi(0, m) up[m + i] = {(x[i] ? P - x[i] : 0), 1};
93     for (int i = m - 1; i
94         > 0; --i) up[i] = up[i * 2].Mul(up[i * 2 + 1]);
95     return up;
96 }
97 vector
98     <ll> Eval(const vector<ll> &x) const { // 1e5, 1s
99     auto up = _tree1(x); return _eval(x, up);
100 }
101 static Poly Interpolate(const vector
102     <ll> &x, const vector<ll> &y) { // 1e5, 1.4s
103     const int m = (int)x.size();
104     vector<Poly> up = _tree1(x), down(m * 2);
105     vector<ll> z = up[1].Dx()._eval(x, up);
106     fi(0, m) z[i] = y[i] * ntt.minv(z[i]) % P;
107     fi(0, m) down[m + i] = {z[i]};
108     for (int i = m -
109         1; i > 0; --i) down[i] = down[i * 2].Mul(up[i
110             * 2 + 1]).iadd(down[i * 2].Mul(up[i * 2]));
111     return down[1];
112 }
113 Poly Ln() const { // (*this)[0] == 1, 1e5/170ms
114     return Dx().Mul(Inv()).Sx().isz(n());
115 }
116 Poly Exp() const { // (*this)[0] == 0, 1e5/360ms
117     if (n() == 1) return {1};
118     Poly X = Poly(*this, (n() + 1) / 2).Exp().isz(n());
119     Poly Y = X.Ln(); Y[0] = P - 1;
120     fi(0, n())
121         if ((Y[i] = (*this)[i] - Y[i]) < 0) Y[i] += P;
122     return X.Mul(Y).isz(n());
123 }
124 // M := P(P - 1). If k >= M, k := k % M + M.
125 Poly Pow(ll k) const {
126     int nz = 0;
127     while (nz < n() && !(*this)[nz]) ++nz;
128     if (nz * min(k, (ll)n()) >= n()) return Poly(n());
129     if (!k) return Poly(Poly{1}, n());
130     Poly X(data() + nz, data() + nz + n() - nz * k);
131     const ll c = ntt.mpow(X[0], k % (P - 1));
132     return X.Ln().imul
133         (k % P).Exp().imul(c).irev().isz(n()).irev();
134 }
135 static ll
136     LinearRecursion(const vector<ll> &a, const vector
137     <ll> &coef, ll n) { // a_n = \sum c_j a_{n-j}
138     const int k = (int)a.size();
139     assert((int)coef.size() == k + 1);
140     Poly C(k + 1), W(Poly{1}, k), M = {0, 1};
141     fi(1, k + 1) C[k - i] = coef[i] ? P - coef[i] : 0;
142     C[k] = 1;
143     while (n) {
144         if (n % 2) W = W.Mul(M).DivMod(C).second;
145         n /= 2, M = M.Mul(M).DivMod(C).second;
146     }
147     ll ret = 0;
148     fi(0, k) ret = (ret + W[i] * a[i]) % P;
149     return ret;
150 }
151 #undef fi
152 using Poly_t = Poly<131072 * 2, 998244353, 3>;
153 template<> decltype(Poly_t::ntt) Poly_t::ntt = {};

```

7.2 Fast Walsh Transform [820c20]

```

1 /* x: a[j], y: a[j + (L >> 1)]
2 or: (y += x * op), and: (x += y * op)
3 xor: (x, y = (x + y) * op, (x - y) * op)
4 invop: or, and, xor = -1, -1, 1/2 */
5 void fwt(int *a, int n, int op) { //or
6     for (int L = 2; L <= n; L <= 1)
7         for (int i = 0; i < n; i += L)
8             for (int j = i; j < i + (L >> 1); ++j)
9                 a[j + (L >> 1)] += a[j] * op;
10 }
11 const int N = 21;
12 int f[
13     N][1 << N], g[N][1 << N], h[N][1 << N], ct[1 << N];
14 void
15     subset_convolution(int *a, int *b, int *c, int L) {
16     // c_k = \sum_{i+j=k, i&j=0} a_i * b_j
17     int n = 1 << L;
18     for (int i = 1; i < n; ++i)
19         ct[i] = ct[i & (i - 1)] + 1;
20     for (int i = 0; i < n; ++i)
21         f[ct[i]][i] = a[i], g[ct[i]][i] = b[i];
22     for (int i = 0; i <= L; ++i)
23         fwt(f[i], n, 1), fwt(g[i], n, 1);
24     for (int i = 0; i <= L; ++i)
25         for (int j = 0; j <= i; ++j)
26             for (int x = 0; x < n; ++x)
27                 h[i][x] += f[j][x] * g[i - j][x];
28     for (int i = 0; i <= L; ++i)
29         fwt(h[i], n, -1);
30     for (int i = 0; i < n; ++i)
31         c[i] = h[ct[i]][i];
32 }

```

7.3 Number Theory Transform [9a0ea6]

```

1 // (2^16)+1, 65537, 3
2 // 7*17*(2^23)+1, 998244353, 3
3 // 1255*(2^20)+1, 1315962881, 3
4 // 51*(2^25)+1, 1711276033, 29
5 template<int MAXN, ll P, ll RT> //MAXN must be 2^k
6 struct NTT {
7     ll w[MAXN];
8     ll mpow(ll a, ll n);
9     ll minv(ll a) { return mpow(a, P - 2); }
10    NTT() {
11        ll dw = mpow(RT, (P - 1) / MAXN);
12        w[0] = 1;
13        for (int
14            i = 1; i < MAXN; ++i) w[i] = w[i - 1] * dw % P;
15    }
16    void bitrev(ll *a, int n) {
17        int i = 0;
18        for (int j = 1; j < n - 1; ++j) {
19            for (int k = n >> 1; (i ^ k) < k; k >>= 1);
20            if (j < i) swap(a[i], a[j]);
21        }
22    }
23    void operator()(
24        ll *a, int n, bool inv = false) { // 0 <= a[i] < P
25        bitrev(a, n);
26        for (int L = 2; L <= n; L <= 1) {
27            int dx = MAXN / L, dl = L >> 1;
28            for (int i = 0; i < n; i += L) {
29                for (int
30                    j = i, x = 0; j < i + dl; ++j, x += dx) {
31                    ll tmp = a[j + dl] * w[x] % P;
32                    if ((a[j]
33                        + dl] = a[j] - tmp) < 0) a[j + dl] += P;
34                    if ((a[j] += tmp) >= P) a[j] -= P;
35                }
36            }
37        }
38        if (inv) {
39            reverse(a + 1, a + n);

```

```

36     ll invn = minv(n);
37     for (int
        i = 0; i < n; ++i) a[i] = a[i] * invn % P;
38 }
39 }
40 };

```

7.4 Value Poly [6438ba]

```

1 struct Poly {
2     mint base; // f(x) = poly[x - base]
3     vector<mint> poly;
4     Poly(mint b = 0, mint x = 0): base(b), poly(1, x) {}
5     mint get_val(const mint &x) {
6         if (x >= base && x < base + SZ(poly))
7             return poly[x - base];
8         mint rt = 0;
9         vector<mint> lmul(SZ(poly), 1), rmul(SZ(poly), 1);
10        for (int i = 1; i < SZ(poly); ++i)
11            lmul[i] = lmul[i - 1] * (x - (base + i - 1));
12        for (int i = SZ(poly) - 2; i >= 0; --i)
13            rmul[i] = rmul[i + 1] * (x - (base + i + 1));
14        for (int i = 0; i < SZ(poly); ++i)
15            rt += poly[i] * ifac[i] * inegfac
16                [SZ(poly) - 1 - i] * lmul[i] * rmul[i];
17        return rt;
18    }
19    void raise() { // g(x) = sigma{base:x} f(x)
20        if (SZ(poly) == 1 && poly[0] == 0)
21            return;
22        mint nw = get_val(base + SZ(poly));
23        poly.pb(nw);
24        for (int i = 1; i < SZ(poly); ++i)
25            poly[i] += poly[i - 1];
26    }
27 };

```

7.5 NTT.2 [6997db]

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4 constexpr int MAXN = 1 << 20;
5 template<int MOD, int RT>
6 struct Zp {
7     #define OP(op) static int op(int x, int y)
8     OP(add) { return (x += y) >= MOD ? x - MOD : x; }
9     OP(sub) { return (x -= y) < 0 ? x + MOD : x; }
10    OP(mul) { return int(ll(x) * y % MOD); }
11    static int mpow(int a, int n) {
12        int r = 1;
13        while (n) {
14            if (n % 2) r = mul(r, a);
15            n /= 2, a = mul(a, a);
16        }
17        return r;
18    }
19    static int minv(int a) { return mpow(a, MOD - 2); }
20    struct NTT; struct Poly;
21    static NTT ntt;
22 };
23 template<int MOD, int RT>
24 struct Zp<MOD, RT>::NTT {
25     int w[MAXN];
26     NTT() {
27         int s =
28             MAXN / 2, dw = mpow(RT, (MOD - 1) / MAXN);
29         for (; s; s >>= 1, dw = mul(dw, dw)) {
30             w[s] = 1;
31             for (int j = 1; j < s; ++j)
32                 w[s + j] = mul(w[s + j - 1], dw);
33         }
34     }
35     void apply
36         (int *a, int n, bool inv = 0) { // 0 <= a_i < P
37         for (int i = 0, j = 1; j < n - 1; ++j) {
38             for (
39                 int k = n >> 1; (i ^= k) < k; k >>= 1);
40             if (j < i) swap(a[i], a[j]);
41         }
42         for (int s = 1; s < n; s <= 1) {
43             for (int i = 0; i < n; i += s * 2) {
44                 for (int j = 0; j < s; ++j) {
45                     int tmp
46                         = mul(a[i + s + j], w[s + j]);
47                     a[i + s + j] = sub(a[i + j], tmp);
48                     a[i + j] = add(a[i + j], tmp);
49                 }
50             }
51         }
52     }
53 };

```

```

45     }
46 }
47 }
48 if (!inv) return;
49 int iv = minv(n); reverse(a + 1, a + n);
50 for (int
    i = 0; i < n; ++i) a[i] = mul(a[i], iv);
51 }
52 };
53 template<int MOD, int RT>
54 typename Zp<MOD, RT>::NTT Zp<MOD, RT>::ntt;
55 using ctx1 = Zp<998244353, 3>;
56 int a[MAXN];
57 int main() {
58     ios::sync_with_stdio(false);
59     cin.tie(nullptr);
60     for (int i = 0; i < 10; ++i) {
61         a[i] = rand() % 100;
62         cout << a[i] << " \n"[i == 9];
63     }
64     ctx1::ntt.apply(a, MAXN);
65     for (int i = 0; i < 10; ++i) {
66         cout << a[i] << " \n"[i == 9];
67     }
68     ctx1::ntt.apply(a, MAXN, 1);
69     for (int i = 0; i < 10; ++i) {
70         cout << a[i] << " \n"[i == 9];
71     }
72     return 0;
73 }

```

7.6 Newton

Given $F(x)$ where

$$F(x) = \sum_{i=0}^{\infty} \alpha_i (x - \beta)^i$$

for β being some constant. Polynomial P such that $F(P) = 0$ can be found iteratively. Denote by Q_k the polynomial such that $F(Q_k) = 0 \pmod{x^{2^k}}$, then

$$Q_{k+1} = Q_k - \frac{F(Q_k)}{F'(Q_k)} \pmod{x^{2^{k+1}}}$$

7.7 Fast Fourier Transform [e5f7dc]

```

1 template<int MAXN>
2 struct FFT {
3     using val_t = complex<double>;
4     const double PI = acos(-1);
5     val_t w[MAXN];
6     FFT() {
7         for (int i = 0; i < MAXN; ++i) {
8             double arg = 2 * PI * i / MAXN;
9             w[i] = val_t(cos(arg), sin(arg));
10        }
11    }
12    void bitrev(val_t *a, int n); // see NTT
13    void trans
14        (val_t *a, int n, bool inv = false); // see NTT;
15    // remember to replace LL with val_t
16 };

```

8 Geometry

8.1 PolyUnion [bf776d]

```

1 double rat(pll a, pll b) {
2     return sign
3         (b.X ? (double)a.X / b.X : (double)a.Y / b.Y);
4 } // all poly. should be ccw
5 double polyUnion(vector<vector<pll>> &poly) {
6     double res = 0;
7     for (auto &p : poly)
8         for (int a = 0; a < SZ(p); ++a) {
9             pll A = p[a], B = p[(a + 1) % SZ(p)];
10            vector
11                <pair<double, int>> segs = {{0, 0}, {1, 0}};
12            for (auto &q : poly) {
13                if (&p == &q) continue;
14                for (int b = 0; b < SZ(q); ++b) {
15                    pll C = q[b], D = q[(b + 1) % SZ(q)];
16                    int sc = ori(A, B, C), sd = ori(A, B, D);
17                    if (sc != sd && min(sc, sd) < 0) {
18                        double sa = cross(D
19                            - C, A - C), sb = cross(D - C, B - C);
20                    }
21                }
22            }
23        }
24 }

```

```

17     segs.emplace_back
18         (sa / (sa - sb), sign(sc - sd));
19     }
20     if (!sc && !sd &&
21         &q < &p && sign(dot(B - A, D - C)) > 0) {
22         segs.emplace_back(rat(C - A, B - A), 1);
23         segs.emplace_back(rat(D - A, B - A), -1);
24     }
25     sort(ALL(segs));
26     for (auto &s : segs) s.X = clamp(s.X, 0.0, 1.0);
27     double sum = 0;
28     int cnt = segs[0].second;
29     for (int j = 1; j < SZ(segs); ++j) {
30         if (!cnt) sum += segs[j].X - segs[j - 1].X;
31         cnt += segs[j].Y;
32     }
33     res += cross(A, B) * sum;
34 }
35 return res / 2;
36 }

```

8.2 external_bisector [f088cc]

```

1 pdd external_bisector(pdd p1, pdd p2, pdd p3) { //213
2     pdd l1 = p2 - p1, l2 = p3 - p1;
3     L2 = L2 * abs(l1) / abs(l2);
4     return l1 + l2;
5 }

```

8.3 Convexhull3D [fc330d]

```

1 struct convex_hull_3D {
2     struct Face {
3         int a, b, c;
4         Face(int ta, int tb, int tc): a(ta), b(tb), c(tc) {}
5     }; // return the faces with pt indexes
6     vector<Face> res;
7     vector<Point> P;
8     convex_hull_3D(const vector<Point> &_P): res(), P(_P) {
9         // all points coplanar case will WA, O(n^2)
10        int n = SZ(P);
11        if (n <= 2) return; // be careful about edge case
12        // ensure first 4 points are not coplanar
13        swap(P[1], *find_if(ALL(P), [&](auto p) { return
14            sign(abs2(P[0] - p)) != 0; }));
15        swap(P[2], *find_if(ALL(P), [&](auto p) { return
16            sign(abs2(cross3(p, P[0], P[1]))) != 0; }));
17        swap(P[3], *find_if(ALL(P), [&](auto p) { return
18            sign(volume(P[0], P[1], P[2], p)) != 0; }));
19        vector<vector<int>> flag(n, vector<int>(n));
20        res.emplace_back(0, 1, 2); res.emplace_back(2, 1, 0);
21        for (int i = 3; i < n; ++i) {
22            vector<Face> next;
23            for (auto f : res) {
24                int d = sign(volume(P[f.a], P[f.b], P[f.c], P[i]));
25                if (d <= 0) next.pb(f);
26                int ff = (d > 0) - (d < 0);
27                flag[f.a][f.b] = flag[f.b][f.c] = flag[f.c][f.a] = ff;
28            }
29            for (auto f : res) {
30                auto F = [&](int x, int y) {
31                    if (flag[x][y] > 0 && flag[y][x] <= 0)
32                        next.emplace_back(x, y, i);
33                };
34                F(f.a, f.b); F(f.b, f.c); F(f.c, f.a);
35            }
36            res = next;
37        }
38        bool same(Face s, Face t) {
39            if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.a])) != 0) return 0;
40            if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.b])) != 0) return 0;
41            if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.c])) != 0) return 0;
42            return 1;
43        }
44        int polygon_face_num() {
45            int ans = 0;
46            for (int i = 0; i < SZ(res); ++i)
47                ans += none_of(res.begin(), res.begin()
48                    + i, [&](Face g) { return same(res[i], g); });
49        }
50    };

```

```

46     return ans;
47 }
48 double get_volume() {
49     double ans = 0;
50     for (auto f : res)
51         ans +=
52             volume(Point(0, 0, 0), P[f.a], P[f.b], P[f.c]);
53     return fabs(ans / 6);
54 }
55 double get_dis(Point p, Face f) {
56     Point p1 = P[f.a], p2 = P[f.b], p3 = P[f.c];
57     double a = (p2.y - p1.y) * (p3.z - p1.z) - (p2.z - p1.z) * (p3.y - p1.y);
58     double b = (p2.z - p1.z) * (p3.x - p1.x) - (p2.x - p1.x) * (p3.z - p1.z);
59     double c = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
60     double d = 0 - (a * p1.x + b * p1.y + c * p1.z);
61     return fabs(a * p.x + b * p.y + c * p.z + d) / sqrt(a * a + b * b + c * c);
62 }
63 // n^2 delaunay: facets with negative z normal of
64 // convexhull of (x, y, x^2 + y^2), use a pseudo-point
65 // (0, 0, inf) to avoid degenerate case

```

8.4 Triangulation Voronoi [433667]

```

1 // all coord. is even
2 // , you may want to call halfPlaneInter after then
3 vector<vector<Line>> vec;
4 void build_voronoi_line(int n, pll *arr) {
5     tool.init(n, arr); // Delaunay
6     vec.clear(), vec.resize(n);
7     for (int i = 0; i < n; ++i)
8         for (auto e : tool.head[i]) {
9             int u = tool.oidx[i], v = tool.oidx[e.id];
10            pll m = (arr[v]
11                + arr[u]) / 2LL, d = perp(arr[v] - arr[u]);
12            vec[u].pb(Line(m, m + d));
13        }
14 }

```

8.5 Default code int [111a95]

```

1 typedef pair<double, double> pdd;
2 typedef pair<pll, pll> Line;
3 pll operator+(pll a, pll b) {
4     return pll(a.X + b.X, a.Y + b.Y);
5 }
6 pll operator-(pll a, pll b) {
7     return pll(a.X - b.X, a.Y - b.Y);
8 }
9 pll operator*(pll a, pll b) {
10    return pll(a.X * b.X, a.Y * b.Y);
11 }
12 pll operator/(pll a, pll b) {
13    return pll(a.X / b.X, a.Y / b.Y);
14 }
15 pdd operator/(pll a, double b) {
16    return pdd(a.X / b, a.Y / b);
17 }
18 ll dot(pll a, pll b) {
19    return a.X * b.X + a.Y * b.Y;
20 }
21 ll cross(pll a, pll b) {
22    return a.X * b.Y - a.Y * b.X;
23 }
24 ll abs2(pll a) {
25    return dot(a, a);
26 }
27 int sign(ll a) {
28    return a == 0 ? 0 : a > 0 ? 1 : -1;
29 }
30 int ori(pll a, pll b, pll c) {
31    return sign(cross(b - a, c - a));
32 }
33 bool collinearity(pll p1, pll p2, pll p3) {
34    return sign(cross(p1 - p3, p2 - p3)) == 0;
35 }
36 bool btw(pll p1, pll p2, pll p3) {
37    if (!collinearity(p1, p2, p3)) return 0;
38    return sign(dot(p1 - p3, p2 - p3)) <= 0;
39 }
40 bool seg_intersect(pll p1, pll p2, pll p3, pll p4) {
41    int a123 = ori(p1, p2, p3);
42    int a124 = ori(p1, p2, p4);
43    int a341 = ori(p3, p4, p1);
44    int a342 = ori(p3, p4, p2);
45    if (a123 == 0 && a124 == 0)
46        return btw(p1, p2, p3) || btw(p1, p2, p4) ||
47            btw(p3, p4, p1) || btw(p3, p4, p2);
48    return a123 * a124 <= 0 && a341 * a342 <= 0;
49 }
50 pdd intersect(pll p1, pll p2, pll p3, pll p4) {
51    ll a123 = cross(p2 - p1, p3 - p1);
52    ll a124 = cross(p2 - p1, p4 - p1);
53    return (p4 * a123
54        - p3 * a124) / double(a123 - a124); // C^3 / C^2
55 }

```

```

43 }
44 pll perp(pll p1)
45 { return pll(-p1.Y, p1.X); }

```

8.6 Polar Angle Sort [2804b5]

```

1 int cmp(pll a, pll b, bool same = true) {
2     #define is_neg(k) (
3         sign(k.Y) < 0 || (sign(k.Y) == 0 && sign(k.X) < 0))
4     int A = is_neg(a), B = is_neg(b);
5     if (A != B)
6         return A < B;
7     if (sign(cross(a, b)) == 0)
8         return same ? abs2(a) < abs2(b) : -1;
9     return sign(cross(a, b)) > 0;
10 }

```

8.7 Default code [3efc61]

```

1 typedef pair<double, double> pdd;
2 typedef pair<pdd, pdd> Line;
3 struct Cir{ pdd O; double R; };
4 const double eps = 1e-8;
5 pdd operator+(pdd a, pdd b)
6 { return pdd(a.X + b.X, a.Y + b.Y); }
7 pdd operator-(pdd a, pdd b)
8 { return pdd(a.X - b.X, a.Y - b.Y); }
9 pdd operator*(pdd a, double b)
10 { return pdd(a.X * b, a.Y * b); }
11 pdd operator/(pdd a, double b)
12 { return pdd(a.X / b, a.Y / b); }
13 double dot(pdd a, pdd b)
14 { return a.X * b.X + a.Y * b.Y; }
15 double cross(pdd a, pdd b)
16 { return a.X * b.Y - a.Y * b.X; }
17 double abs2(pdd a)
18 { return dot(a, a); }
19 double abs(pdd a)
20 { return sqrt(dot(a, a)); }
21 int sign(double a)
22 { return fabs(a) < eps ? 0 : a > 0 ? 1 : -1; }
23 int ori(pdd a, pdd b, pdd c)
24 { return sign(cross(b - a, c - a)); }
25 bool collinearity(pdd p1, pdd p2, pdd p3)
26 { return sign(cross(p1 - p3, p2 - p3)) == 0; }
27 bool btw(pdd p1, pdd p2, pdd p3) {
28     if (!collinearity(p1, p2, p3)) return 0;
29     return sign(dot(p1 - p3, p2 - p3)) <= 0;
30 }
31 bool seg_intersect(pdd p1, pdd p2, pdd p3, pdd p4) {
32     int a123 = ori(p1, p2, p3);
33     int a124 = ori(p1, p2, p4);
34     int a341 = ori(p3, p4, p1);
35     int a342 = ori(p3, p4, p2);
36     if (a123 == 0 && a124 == 0)
37         return btw(p1, p2, p3) || btw(p1, p2, p4) ||
38             btw(p3, p4, p1) || btw(p3, p4, p2);
39     return a123 * a124 <= 0 && a341 * a342 <= 0;
40 }
41 pdd intersect(pdd p1, pdd p2, pdd p3, pdd p4) {
42     double a123 = cross(p2 - p1, p3 - p1);
43     double a124 = cross(p2 - p1, p4 - p1);
44     return p4
45         * a123 - p3 * a124) / (a123 - a124); // C^3 / C^2
46 }
47 pdd perp(pdd p1)
48 { return pdd(-p1.Y, p1.X); }
49 pdd projection(pdd p1, pdd p2, pdd p3)
50 { return p1 + (
51     p2 - p1) * dot(p3 - p1, p2 - p1) / abs2(p2 - p1); }
52 pdd reflection(pdd p1, pdd p2, pdd p3)
53 { return p3 + perp(p2 - p1
54     ) * cross(p3 - p1, p2 - p1) / abs2(p2 - p1) * 2; }
55 pdd linearTransformation
56 (pdd p0, pdd p1, pdd q0, pdd q1, pdd r) {
57     pdd dp = p1 - p0
58     , dq = q1 - q0, num(cross(dp, dq), dot(dp, dq));
59     return q0 + pdd(
60         cross(r - p0, num), dot(r - p0, num)) / abs2(dp);
61 } // from line p0--p1 to q0--q1, apply to r

```

8.8 PointInConvex Slow [dd78ba]

```

1 bool PointInConvex(const vector<pll> &C, pdd p) {
2     if (SZ(C) == 0) return false;
3     if (SZ(C) == 1) return abs(C[0] - p) < eps;
4     if (SZ(C) == 2) return btw(C[0], C[1], p);

```

```

5     for (int i = 0; i < SZ(C); ++i) {
6         const int j = i + 1 == SZ(C) ? 0 : i + 1;
7         if (cross(C[j] - C[i], p - C[i]) < -eps)
8             return false;
9     }
10    return true;
11 }

```

8.9 Intersection of polygon and circle [cbe8f5]

```

1 // Divides into multiple triangle, and sum up
2 const double PI=acos(-1);
3 double _area(pdd pa, pdd pb, double r){
4     if(abs(pa)<abs(pb)) swap(pa, pb);
5     if(abs(pb)<eps) return 0;
6     double S, h, theta;
7     double a=abs(pb),b=abs(pa),c=abs(pb-pa);
8     double cosB = dot(pb,pb-pa) / a / c, B = acos(cosB);
9     double cosC = dot(pa,pb) / a / b, C = acos(cosC);
10    if(a > r){
11        S = (C/2)*r*r;
12        h = a*b*sin(C)/c;
13        if (h < r && B
14            < PI/2) S -= (acos(h/r)*r*r - h*sqrt(r*r-h*h));
15    }
16    else if(b > r){
17        theta = PI - B - asin(sin(B)/r*a);
18        S = .5*a*r*sin(theta) + (C-theta)/2*r*r;
19    }
20    else S = .5*sin(C)*a*b;
21    return S;
22 }
23 double area_poly_circle(const
24     vector<pdd> poly,const pdd &O,const double r){
25     double S=0;
26     for(int i=0;i<SZ(poly);++i)
27         S+=_area(poly[i]-O,poly[(i+1)%SZ(poly)
28             ]-O,r)*ori(O,poly[i],poly[(i+1)%SZ(poly)]);
29     return fabs(S);
30 }

```

8.10 Tangent line of two circles [5ad86c]

```

1 vector<Line>
2     > go( const Cir& c1 , const Cir& c2 , int sign1 ){
3     // sign1 = 1 for outer tang, -1 for inter tang
4     vector<Line> ret;
5     double d_sq = abs2(c1.O - c2.O);
6     if (sign(d_sq) == 0) return ret;
7     double d = sqrt(d_sq);
8     pdd v = (c2.O - c1.O) / d;
9     double c = (c1.R - sign1 * c2.R) / d;
10    if (c * c > 1) return ret;
11    double h = sqrt(max(0.0, 1.0 - c * c));
12    for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
13        pdd n = pdd(v.X * c - sign2 * h * v.Y,
14            v.Y * c + sign2 * h * v.X);
15        pdd p1 = c1.O + n * c1.R;
16        pdd p2 = c2.O + n * (c2.R * sign1);
17        if (sign(p1.X - p2.X) == 0 and
18            sign(p1.Y - p2.Y) == 0)
19            p2 = p1 + perp(c2.O - c1.O);
20        ret.pb(Line(p1, p2));
21    }
22    return ret;
23 }

```

8.11 CircleCover [1d09aa]

```

1 const int N = 1021;
2 struct CircleCover {
3     int C;
4     Cir c[N];
5     bool g[N][N], overlap[N][N];
6     // Area[i] : area covered by at least i circles
7     double Area[ N ];
8     void init(int _C){ C = _C; }
9     struct Teve {
10         pdd p; double ang; int add;
11         Teve() {}
12         Teve(pdd _a
13             , double _b, int _c):p(_a), ang(_b), add(_c){}
14         bool operator<(const Teve &a)const {
15             return ang < a.ang; }
16     }eve[N * 2];
17     // strict: x = 0, otherwise x = -1
18     bool disjunct(Cir &a, Cir &b, int x)

```



```

18 {return sign(abs(a.0 - b.0) - a.R - b.R) > x;}
19 bool contain(Cir &a, Cir &b, int x)
20 {return sign(a.R - b.R - abs(a.0 - b.0)) > x;}
21 bool contain(int i, int j) {
22     /* c[j] is non-strictly in c[i]. */
23     return (sign
24         (c[i].R - c[j].R) > 0 || (sign(c[i].R - c[j].
25         R) == 0 && i < j)) && contain(c[i], c[j], -1);
26 }
27 void solve(){
28     fill_n(Area, C + 2, 0);
29     for(int i = 0; i < C; ++i)
30         for(int j = 0; j < C; ++j)
31             overlap[i][j] = contain(i, j);
32     for(int i = 0; i < C; ++i)
33         for(int j = 0; j < C; ++j)
34             g[i][j] = !(overlap[i][j] || overlap[j][i] ||
35             disjunct(c[i], c[j], -1));
36     for(int i = 0; i < C; ++i){
37         int E = 0, cnt = 1;
38         for(int j = 0; j < C; ++j)
39             if(j != i && overlap[j][i])
40                 ++cnt;
41         for(int j = 0; j < C; ++j)
42             if(i != j && g[i][j]) {
43                 pdd aa, bb;
44                 CCinter(c[i], c[j], aa, bb);
45                 double A =
46                     atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
47                 double B =
48                     atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
49                 eve[E++] = Teve
50                     (bb, B, 1), eve[E++] = Teve(aa, A, -1);
51                 if(B > A) ++cnt;
52             }
53         if(E == 0) Area[cnt] += pi * c[i].R * c[i].R;
54         else{
55             sort(eve, eve + E);
56             eve[E] = eve[0];
57             for(int j = 0; j < E; ++j){
58                 cnt += eve[j].add;
59                 Area[cnt]
60                     += cross(eve[j].p, eve[j + 1].p) * .5;
61                 double theta = eve[j + 1].ang - eve[j].ang;
62                 if (theta < 0) theta += 2. * pi;
63                 Area[cnt] += (theta
64                     - sin(theta)) * c[i].R * c[i].R * .5;
65             }
66         }
67     }
68 }
69 };

```

8.12 Heart [4698ba]

```

1 pdd circenter
2 (pdd p0, pdd p1, pdd p2) { // radius = abs(center)
3     p1 = p1 - p0, p2 = p2 - p0;
4     double x1 = p1.X, y1 = p1.Y, x2 = p2.X, y2 = p2.Y;
5     double m = 2. * (x1 * y2 - y1 * x2);
6     center.X = (x1 * x1
7         * y2 - x2 * x2 * y1 + y1 * y2 * (y1 - y2)) / m;
8     center.Y = (x1 * x2
9         * (x2 - x1) - y1 * y1 * x2 + x1 * y2 * y2) / m;
10    return center + p0;
11 }
12 pdd incenter
13 (pdd p1, pdd p2, pdd p3) { // radius = area / s * 2
14     double a =
15         abs(p2 - p3), b = abs(p1 - p3), c = abs(p1 - p2);
16     double s = a + b + c;
17     return (a * p1 + b * p2 + c * p3) / s;
18 }
19 pdd masscenter(pdd p1, pdd p2, pdd p3)
20 { return (p1 + p2 + p3) / 3; }
21 pdd orthcenter(pdd p1, pdd p2, pdd p3)
22 { return masscenter
23     (p1, p2, p3) * 3 - circenter(p1, p2, p3) * 2; }

```

8.13 PointSegDist [See686]

```

1 double PointSegDist(pdd q0, pdd q1, pdd p) {
2     if (sign(abs(q0 - q1)) == 0) return abs(q0 - p);
3     if (sign(dot(q1 - q0,
4         p - q0)) >= 0 && sign(dot(q0 - q1, p - q1)) >= 0)
5         return fabs(cross(q1 - q0, p - q0) / abs(q0 - q1));
6     return min(abs(p - q0), abs(p - q1));

```

8.14 Minkowski Sum [95f4a0]

```

1 vector<pll> Minkowski
2 (vector<pll> A, vector<pll> B) { // |A|,|B|>=3
3     hull(A), hull(B);
4     vector<pll> C(1, A[0] + B[0]), s1, s2;
5     for (int i = 0; i < SZ(A); ++i)
6         s1.pb(A[(i + 1) % SZ(A)] - A[i]);
7     for (int i = 0; i < SZ(B); ++i)
8         s2.pb(B[(i + 1) % SZ(B)] - B[i]);
9     for (int i = 0, j = 0; i < SZ(A) || j < SZ(B);)
10         if (j >= SZ
11             (B) || (i < SZ(A) && cross(s1[i], s2[j]) >= 0))
12             C.pb(B[j % SZ(B)] + A[i++]);
13         else
14             C.pb(A[i % SZ(A)] + B[j++]);
15     return hull(C), C;

```

8.15 TangentPointToHull [5668cc]

```

1 /* The point should be strictly out of hull
2    return arbitrary point on the tangent line */
3 pii get_tangent(vector<pll> &C, pll p) {
4     auto gao = [&](int s) {
5         return cyc_tsearch(SZ(C), [&](int x, int y)
6             { return ori(p, C[x], C[y]) == s; });
7     };
8     return pii(gao(1), gao(-1));
9 } // return (a, b), ori(p, C[a], C[b]) >= 0

```

8.16 Intersection of two circles [b062ba]

```

1 bool CCinter(Cir &a, Cir &b, pdd &p1, pdd &p2) {
2     pdd o1 = a.O, o2 = b.O;
3     double r1 =
4         a.R, r2 = b.R, d2 = abs2(o1 - o2), d = sqrt(d2);
5     if(d < max
6         (r1, r2) - min(r1, r2) || d > r1 + r2) return 0;
7     pdd u = (o1 + o2) * 0.5
8         + (o1 - o2) * ((r2 * r2 - r1 * r1) / (2 * d2));
9     double A = sqrt((r1 + r2 + d) *
10         (r1 - r2 + d) * (r1 + r2 - d) * (-r1 + r2 + d));
11     pdd v
12         = pdd(o1.Y - o2.Y, -o1.X + o2.X) * A / (2 * d2);
13     p1 = u + v, p2 = u - v;
14     return 1;

```

8.17 PointInConvex [9136f4]

```

1 bool PointInConvex
2 (const vector<pll> &C, pll p, bool strict = true) {
3     int a = 1, b = SZ(C) - 1, r = !strict;
4     if (SZ(C) == 0) return false;
5     if (SZ(C) < 3) return r && btw(C[0], C.back(), p);
6     if (ori(C[0], C[a], C[b]) > 0) swap(a, b);
7     if (ori
8         (C[0], C[a], p) >= r || ori(C[0], C[b], p) <= -r)
9         return false;
10    while (abs(a - b) > 1) {
11        int c = (a + b) / 2;
12        (ori(C[0], C[c], p) > 0 ? b : a) = c;
13    }
14    return ori(C[a], C[b], p) < r;

```

8.18 Intersection of line and circle [894afd]

```

1 vector<pdd> circleLine(pdd c, double r, pdd a, pdd b) {
2     pdd p
3         = a + (b - a) * dot(c - a, b - a) / abs2(b - a);
4     double s = cross
5         (b - a, c - a), h2 = r * r - s * s / abs2(b - a);
6     if (h2 < 0) return {};
7     if (h2 == 0) return {p};
8     pdd h = (b - a) / abs(b - a) * sqrt(h2);
9     return {p - h, p + h};

```

8.19 Trapezoidalization [4e01c8]

```

1 template<class T>
2 struct SweepLine {
3     struct cmp {
4         cmp(const SweepLine &swp): swp(swp) {}
5         bool operator()(int a, int b) const {
6             if (abs(swp.get_y(a) - swp.get_y(b)) <= swp.eps)

```



```

7     return swp.slope_cmp(a, b);
8     return swp.get_y(a) + swp.eps < swp.get_y(b);
9 }
10 const SweepLine &swp;
11 } _cmp;
12 T curTime, eps, curQ;
13 vector<Line> base;
14 multiset<int, cmp> sweep;
15 multiset<pair<T, int>> event;
16 vector<typename multiset<int, cmp>::iterator> its;
17 vector
18     <typename multiset<pair<T, int>>::iterator> eits;
19 bool slope_cmp(int a, int b) const {
20     assert(a != -1);
21     if (b == -1) return 0;
22     return sign(cross(base
23         [a].Y - base[a].X, base[b].Y - base[b].X)) < 0;
24 }
25 T get_y(int idx) const {
26     if (idx == -1) return curQ;
27     Line l = base[idx];
28     if (l.X.X == l.Y.X) return l.Y.Y;
29     return ((curTime - l.X.X) * l.Y.Y
30         + (l.Y.X - curTime) * l.X.Y) / (l.Y.X - l.X.X);
31 }
32 void insert(int idx) {
33     its[idx] = sweep.insert(idx);
34     if (its[idx] != sweep.begin())
35         update_event(*prev(its[idx]));
36     update_event(idx);
37     event.emplace(base[idx].Y.X, idx + 2 * SZ(base));
38 }
39 void erase(int idx) {
40     assert(eits[idx] == event.end());
41     auto p = sweep.erase(its[idx]);
42     its[idx] = sweep.end();
43     if (p != sweep.begin())
44         update_event(*prev(p));
45 }
46 void update_event(int idx) {
47     if (eits[idx] != event.end())
48         event.erase(eits[idx]);
49     eits[idx] = event.end();
50     auto nxt = next(its[idx]);
51     if (nxt ==
52         sweep.end() || !slope_cmp(idx, *nxt)) return;
53     auto t = intersect(base[idx].
54         X, base[idx].Y, base[*nxt].X, base[*nxt].Y).X;
55     if (t + eps < curTime || t
56         >= min(base[idx].Y.X, base[*nxt].Y.X)) return;
57     eits[idx] = event.emplace(t, idx + SZ(base));
58 }
59 void swp(int idx) {
60     assert(eits[idx] != event.end());
61     eits[idx] = event.end();
62     int nxt = *next(its[idx]);
63     swap((int&)*its[idx], (int&)*its[nxt]);
64     swap(its[idx], its[nxt]);
65     if (its[nxt] != sweep.begin())
66         update_event(*prev(its[nxt]));
67     update_event(idx);
68 }
69 // only expected to call the functions below
70 SweepLine(T t, T e, vector
71     <Line> vec): _cmp(*this), curTime(t), eps(e)
72     , curQ(), base(vec), sweep(_cmp), event(), its(SZ
73     (vec), sweep.end()), eits(SZ(vec), event.end()) {
74     for (int i = 0; i < SZ(base); ++i) {
75         auto &[p, q] = base[i];
76         if (p > q) swap(p, q);
77         if (p.X <= curTime && curTime <= q.X)
78             insert(i);
79         else if (curTime < p.X)
80             event.emplace(p.X, i);
81     }
82 }
83 void setTime(T t, bool ers = false) {
84     assert(t >= curTime);
85     while (!event.empty() && event.begin()->X <= t) {
86         auto [et, idx] = *event.begin();
87         int s = idx / SZ(base);
88         idx %= SZ(base);
89         if (abs(et - t) <= eps && s == 2 && !ers) break;
90         curTime = et;
91         event.erase(event.begin());
92         if (s == 2) erase(idx);
93     }
94 }

```

```

84     else if (s == 1) swp(idx);
85     else insert(idx);
86 }
87 curTime = t;
88 }
89 T nextEvent() {
90     if (event.empty()) return INF;
91     return event.begin()->X;
92 }
93 int lower_bound(T y) {
94     curQ = y;
95     auto p = sweep.lower_bound(-1);
96     if (p == sweep.end()) return -1;
97     return *p;
98 }
99 };

```

8.20 point in circle [9ae6d9]

```

1 // return q'
2 s relation with circumcircle of tri(p[0],p[1],p[2])
3 bool in_cc(const array<pll, 3> &p, pll q) {
4     __int128 det = 0;
5     for (int i = 0; i < 3; ++i)
6         det += __int128(abs2(p[i]) - abs2(q)) *
7             cross(p[(i + 1) % 3] - q, p[(i + 2) % 3] - q);
8     return det > 0; // in: >0, on: =0, out: <0
9 }

```

8.21 PolyCut [417264]

```

1 vector<pdd> cut(vector<pdd> poly, pdd s, pdd e) {
2     vector<pdd> res;
3     for (int i = 0; i < SZ(poly); ++i) {
4         pdd cur
5             = poly[i], prv = i ? poly[i - 1] : poly.back();
6         bool side = ori(s, e, cur) < 0;
7         if (side != (ori(s, e, prv) < 0))
8             res.pb(intersect(s, e, cur, prv));
9         if (side)
10             res.pb(cur);
11     }
12     return res;
13 }

```

8.22 minDistOfTwoConvex [d62c1f]

```

1 double ConvexHullDist(vector<pdd> A, vector<pdd> B) {
2     for (auto &p : B) p = {-p.X, -p.Y};
3     auto C = Minkowski(A, B); // assert SZ(C) > 0
4     if (PointInConvex(C, pdd(0, 0))) return 0;
5     double
6         ans = PointSegDist(C.back(), C[0], pdd(0, 0));
7     for (int i = 0; i + 1 < SZ(C); ++i) {
8         ans = min(ans
9             , PointSegDist(C[i], C[i + 1], pdd(0, 0)));
10    }
11    return ans;
12 }

```

8.23 DelaunayTriangulation [1d8107]

```

1 /* Delaunay Triangulation:
2 Given a sets of points on 2D plane, find a
3 triangulation such that no points will strictly
4 inside circumcircle of any triangle.
5 find : return a triangle contain given point
6 add_point : add a point into triangulation
7 A Triangle is in triangulation iff. its has_chd is 0.
8 Region of triangle u: iterate each u.edge[i].tri,
9 each points are u.p[(i+1)%3], u.p[(i+2)%3]
10 Voronoi diagram: for each triangle in triangulation,
11 the bisector of all its edges will split the region.
12 nearest point will belong to the triangle containing it
13 */
14 const
15     ll inf = MAXC * MAXC * 100; // lower_bound unknown
16 struct Tri;
17 struct Edge {
18     Tri* tri; int side;
19     Edge(): tri(0), side(0){}
20     Edge(Tri* _tri, int _side): tri(_tri), side(_side){}
21 };
22 struct Tri {
23     pll p[3];
24     Edge edge[3];
25     Tri* chd[3];
26     Tri() {}

```

```

26 Tri(const pll& p0, const pll& p1, const pll& p2) {
27     p[0] = p0; p[1] = p1; p[2] = p2;
28     chd[0] = chd[1] = chd[2] = 0;
29 }
30 bool has_chd() const { return chd[0] != 0; }
31 int num_chd() const {
32     return !!chd[0] + !!chd[1] + !!chd[2];
33 }
34 bool contains(pll const& q) const {
35     for (int i = 0; i < 3; ++i)
36         if (ori(p[i], p[(i + 1) % 3], q) < 0)
37             return 0;
38     return 1;
39 }
40 } pool[N * 10], *tris;
41 void edge(Edge a, Edge b) {
42     if(a.tri) a.tri->edge[a.side] = b;
43     if(b.tri) b.tri->edge[b.side] = a;
44 }
45 struct Trig { // Triangulation
46     Trig() {
47         the_root
48             = // Tri should at least contain all points
49             new(tris++) Tri(pll(-inf, -inf),
50                 pll(inf + inf, -inf), pll(-inf, inf + inf));
51     }
52     Tri* find(pll p) { return find(the_root, p); }
53     void add_point(const
54         pll &p) { add_point(find(the_root, p), p); }
55     Tri* the_root;
56     static Tri* find(Tri* root, const pll &p) {
57         while (1) {
58             if (!root->has_chd())
59                 return root;
60             for (int i = 0; i < 3 && root->chd[i]; ++i)
61                 if (root->chd[i]->contains(p)) {
62                     root = root->chd[i];
63                     break;
64                 }
65         }
66         assert(0); // "point not found"
67     }
68     void add_point(Tri* root, pll const& p) {
69         Tri* t[3];
70         /* split it into three triangles */
71         for (int i = 0; i < 3; ++i)
72             t[i] = new(tris
73                 ++ Tri(root->p[i], root->p[(i + 1) % 3], p);
74         for (int i = 0; i < 3; ++i)
75             edge(Edge(t[i], 0), Edge(t[(i + 1) % 3], 1));
76         for (int i = 0; i < 3; ++i)
77             edge(Edge(t[i], 2), root->edge[(i + 2) % 3]);
78         for (int i = 0; i < 3; ++i)
79             root->chd[i] = t[i];
80         for (int i = 0; i < 3; ++i)
81             flip(t[i], 2);
82     }
83     void flip(Tri* tri, int pi) {
84         Tri* trj = tri->edge[pi].tri;
85         int pj = tri->edge[pi].side;
86         if (!trj) return;
87         if (!in_cc(tri->p
88             [0], tri->p[1], tri->p[2], trj->p[pj])) return;
89         /* flip edge between tri, trj */
90         Tri* trk = new(tris++) Tri
91             (tri->p[(pi + 1) % 3], trj->p[pj], tri->p[pi]);
92         Tri* trl = new(tris++) Tri
93             (trj->p[(pj + 1) % 3], tri->p[pi], trj->p[pj]);
94         edge(Edge(trk, 0), Edge(trl, 0));
95         edge(Edge(trk, 1), tri->edge[(pi + 2) % 3]);
96         edge(Edge(trk, 2), trj->edge[(pj + 1) % 3]);
97         edge(Edge(trl, 1), trj->edge[(pj + 2) % 3]);
98         edge(Edge(trl, 2), tri->edge[(pi + 1) % 3]);
99         tri->chd
100             [0] = trk; tri->chd[1] = trl; tri->chd[2] = 0;
101         trj->chd
102             [0] = trk; trj->chd[1] = trl; trj->chd[2] = 0;
103         flip(trk, 1); flip(trk, 2);
104         flip(trl, 1); flip(trl, 2);
105     }
106 };
107 vector<Tri*> triang; // vector of all triangle
108 set<Tri*> vst;
109 void go(Tri* now) { // store all tri into triang
110     if (vst.find(now) != vst.end())
111         return;

```

```

103     vst.insert(now);
104     if (!now->has_chd())
105         return triang.pb(now);
106     for (int i = 0; i < now->num_chd(); ++i)
107         go(now->chd[i]);
108 }
109 void build(int n, pll* ps) { // build triangulation
110     tris = pool; triang.clear(); vst.clear();
111     random_shuffle(ps, ps + n);
112     Trig tri; // the triangulation structure
113     for (int i = 0; i < n; ++i)
114         tri.add_point(ps[i]);
115     go(tri.the_root);
116 }

```

8.24 rotatingSweepLine [374fec]

```

1 void rotatingSweepLine(vector<pii> &ps) {
2     int n = SZ(ps), m = 0;
3     vector<int> id(n), pos(n);
4     vector<pii> line(n * (n - 1));
5     for (int i = 0; i < n; ++i)
6         for (int j = 0; j < n; ++j)
7             if (i != j) line[m++] = pii(i, j);
8     sort(ALL(line), [&](pii a, pii b) {
9         return cmp(ps[a.Y] - ps[a.X], ps[b.Y] - ps[b.X]);
10    }); // cmp(): polar angle compare
11    iota(ALL(id), 0);
12    sort(ALL(id), [&](int a, int b) {
13        if (ps[a].Y != ps[b].Y) return ps[a].Y < ps[b].Y;
14        return ps[a] < ps[b];
15    }); // initial order, since (1, 0) is the smallest
16    for (int i = 0; i < n; ++i) pos[id[i]] = i;
17    for (int i = 0; i < m; ++i) {
18        auto l = line[i];
19        // do something
20        tie(pos[l.X], pos[l.Y], id[pos[l.X]], id[pos[l.Y]
21            ]) = make_tuple(pos[l.Y], pos[l.X], l.Y, l.X);
22    }
23 }

```

8.25 Intersection of line and convex [e14a5c]

```

1 int TangentDir(vector<pll> &C, pll dir) {
2     return cyc_tsearch(SZ(C), [&](int a, int b) {
3         return cross(dir, C[a]) > cross(dir, C[b]);
4     });
5 }
6 #define cmpl(i) sign(cross(C[i] - a, b - a))
7 pii lineHull(pll a, pll b, vector<pll> &C) {
8     int A = TangentDir(C, a - b);
9     int B = TangentDir(C, b - a);
10    int n = SZ(C);
11    if (cmpl(A) < 0 || cmpl(B) > 0)
12        return pii(-1, -1); // no collision
13    auto gao = [&](int l, int r) {
14        for (int t = l; (l + 1) % n != r; ) {
15            int m = ((l + r + (l < r ? 0 : n)) / 2) % n;
16            (cmpl(m) == cmpl(t) ? l : r) = m;
17        }
18        return (l + !cmpl(r)) % n;
19    };
20    pii res = pii(gao(B, A), gao(A, B)); // (i, j)
21    if (res.X == res.Y) // touching the corner i
22        return pii(res.X, -1);
23    if (!
24        cmpl(res.X) && !cmpl(res.Y)) // along side i, i+1
25        switch ((res.X - res.Y + n + 1) % n) {
26            case 0: return pii(res.X, res.X);
27            case 2: return pii(res.Y, res.Y);
28        }
29    /* crossing sides (i, i+1) and (j, j+1)
30       crossing corner i is treated as side (i, i+1)
31       returned
32       in the same order as the line hits the convex */
33    return res;
34 } // convex cut: (r, l]

```

8.26 3Dpoint [374a83]

```

1 struct Point {
2     double x, y, z;
3     Point(double _x = 0, double
4         _y = 0, double _z = 0): x(_x), y(_y), z(_z){}
5     Point(pdd p) { x = p.X, y = p.Y, z = abs2(p); }
6 };
7 Point operator-(Point p1, Point p2)

```

```

7 { return
    Point(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z); }
8 Point operator+(Point p1, Point p2)
9 { return
    Point(p1.x + p2.x, p1.y + p2.y, p1.z + p2.z); }
10 Point operator*(Point p1, double v)
11 { return Point(p1.x * v, p1.y * v, p1.z * v); }
12 Point operator/(Point p1, double v)
13 { return Point(p1.x / v, p1.y / v, p1.z / v); }
14 Point cross(Point p1, Point p2)
15 { return Point(p1.y * p2.z - p1.z * p2.y, p1.z
    * p2.x - p1.x * p2.z, p1.x * p2.y - p1.y * p2.x); }
16 double dot(Point p1, Point p2)
17 { return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z; }
18 double abs(Point a)
19 { return sqrt(dot(a, a)); }
20 Point cross3(Point a, Point b, Point c)
21 { return cross(b - a, c - a); }
22 double area(Point a, Point b, Point c)
23 { return abs(cross3(a, b, c)); }
24 double volume(Point a, Point b, Point c, Point d)
25 { return dot(cross3(a, b, c), d - a); }
26 //Azimuthal
    angle (longitude) to x-axis in interval [-pi, pi]
27 double phi(Point p) { return atan2(p.y, p.x); }
28 //Zenith
    angle (latitude) to the z-axis in interval [0, pi]
29 double theta(Point p)
    { return atan2(sqrt(p.x * p.x + p.y * p.y), p.z); }
30 Point masscenter(Point a, Point b, Point c, Point d)
31 { return (a + b + c + d) / 4; }
32 pdd proj(Point a, Point b, Point c, Point u) {
33 // proj. u to the plane of a, b, and c
    Point e1 = b - a;
    Point e2 = c - a;
    e1 = e1 / abs(e1);
    e2 = e2 - e1 * dot(e2, e1);
    e2 = e2 / abs(e2);
    Point p = u - a;
    return pdd(dot(p, e1), dot(p, e2));
41 }
42 Point
    rotate_around(Point p, double angle, Point axis) {
43 double s = sin(angle), c = cos(angle);
44 Point u = axis / abs(axis);
45 return u
    * dot(u, p) * (1 - c) + p * c + cross(u, p) * s;
46 }

```

8.27 HPIGeneralLine [e36115]

```

1 using i128 = __int128;
2 struct LN {
3     ll a, b, c; // ax + by + c <= 0
4     pll dir() const { return pll(a, b); }
5     LN(ll ta, ll tb, ll tc) : a(ta), b(tb), c(tc) {}
6     LN(pll S,
        pll T): a((T-S).Y), b(-(T-S).X), c(cross(T,S)) {}
7 };
8 pdd intersect(LN A, LN B) {
9     double c = cross(A.dir(), B.dir());
10    i128 a = i128(A.c) * B.a - i128(B.c) * A.a;
11    i128 b = i128(A.c) * B.b - i128(B.c) * A.b;
12    return pdd(-b / c, a / c);
13 }
14 bool cov(LN l, LN A, LN B) {
15    i128 c = cross(A.dir(), B.dir());
16    i128 a = i128(A.c) * B.a - i128(B.c) * A.a;
17    i128 b = i128(A.c) * B.b - i128(B.c) * A.b;
18    return
        sign(a * l.b - b * l.a + c * l.c) * sign(c) >= 0;
19 }
20 bool operator<(LN a, LN b) {
21     if (int c =
        cmp(a.dir(), b.dir(), false); c != -1) return c;
22     return i128(abs(b.a) + abs
        (b.b)) * a.c > i128(abs(a.a) + abs(a.b)) * b.c;
23 }

```

8.28 minMaxEnclosingRectangle [d47db9]

```

1 const double INF = 1e18, qi = acos(-1) / 2 * 3;
2 pdd solve(vector<pll> &dots) {
3     #define diff(u, v) (dots[u] - dots[v])
4     #define vec(v) (dots[v] - dots[i])
5     hull(dots);
6     double Max = 0, Min = INF, deg;

```

```

7     int n = SZ(dots);
8     dots.pb(dots[0]);
9     for (int i = 0, u = 1, r = 1, l = 1; i < n; ++i) {
10        pll nw = vec(i + 1);
11        while (cross(nw, vec(u + 1)) > cross(nw, vec(u)))
12            u = (u + 1) % n;
13        while (dot(nw, vec(r + 1)) > dot(nw, vec(r)))
14            r = (r + 1) % n;
15        if (!i) l = (r + 1) % n;
16        while (dot(nw, vec(l + 1)) < dot(nw, vec(l)))
17            l = (l + 1) % n;
18        Min = min(Min, (double)(dot(nw, vec(r)) - dot
            (nw, vec(l))) * cross(nw, vec(u)) / abs2(nw));
19        deg = acos(dot(diff(r
            , l), vec(u)) / abs(diff(r, l)) / abs(vec(u)));
20        deg = (qi - deg) / 2;
21        Max = max(Max, abs(diff
            (r, l)) * abs(vec(u)) * sin(deg) * sin(deg));
22    }
23    return pdd(Min, Max);
24 }

```

8.29 Half plane intersection [c3e180]

```

1 pll area_pair(Line a, Line b)
2 { return pll(cross(a.Y
    - a.X, b.X - a.X), cross(a.Y - a.X, b.Y - a.X)); }
3 bool isin(Line l0, Line l1, Line l2) {
4     // Check inter(l1, l2) strictly in l0
5     auto [a02X, a02Y] = area_pair(l0, l2);
6     auto [a12X, a12Y] = area_pair(l1, l2);
7     if (a12X - a12Y < 0) a12X *= -1, a12Y *= -1;
8     return (__int128
        ) a02Y * a12X - (__int128) a02X * a12Y > 0;
9 }
10 /* Having solution, check size > 2 */
11 /* --^-- Line.X --^-- Line.Y --^-- */
12 vector<Line> halfPlaneInter(vector<Line> arr) {
13     sort(ALL(arr), [&](Line a, Line b) -> int {
14         if (cmp(a.Y - a.X, b.Y - b.X, 0) != -1)
15             return cmp(a.Y - a.X, b.Y - b.X, 0);
16         return ori(a.X, a.Y, b.Y) < 0;
17     });
18     deque<Line> dq(1, arr[0]);
19     auto pop_back = [&](int t, Line p) {
20         while (SZ(dq)
            ) >= t && !isin(p, dq[SZ(dq) - 2], dq.back())
            dq.pop_back();
21     };
22 };
23 auto pop_front = [&](int t, Line p) {
24     while (SZ(dq) >= t && !isin(p, dq[0], dq[1]))
25         dq.pop_front();
26 };
27 for (auto p : arr)
28     if (cmp(
        dq.back().Y - dq.back().X, p.Y - p.X, 0) != -1)
29         pop_back(2, p), pop_front(2, p), dq.pb(p);
30     pop_back(3, dq[0]), pop_front(3, dq.back());
31     return vector<Line>(ALL(dq));
32 }

```

8.30 Vector in poly [4c9a2f]

```

1 // ori(a
    , b, c) >= 0, valid: "strict" angle from a-b to a-c
2 bool btwangle(pll a, pll b, pll c, pll p, int strict) {
3     return
        ori(a, b, p) >= strict && ori(a, p, c) >= strict;
4 }
5 // whether vector
    {cur, p} in counter-clockwise order prv, cur, nxt
6 bool inside
    (pll prv, pll cur, pll nxt, pll p, int strict) {
7     if (ori(cur, nxt, prv) >= 0)
8         return btwangle(cur, nxt, prv, p, strict);
9     return !btwangle(cur, prv, nxt, p, !strict);
10 }

```

8.31 DelaunayTriangulation dq [e6fa02]

```

1 /* Delaunay Triangulation:
2 Given a sets of points on 2D plane, find a
3 triangulation such that no points will strictly
4 inside circumcircle of any triangle. */
5 struct Edge {
6     int id; // oidx[id]
7     list<Edge>::iterator twin;

```

```

8   Edge(int _id = 0):id(_id) {}
9   };
10  struct Delaunay { // 0-base
11      int n, oidx[N];
12      list<Edge> head[N]; // result udir. graph
13      pll p[N];
14      void init(int _n, pll _p[]) {
15          n = _n, iota(oidx, oidx + n, 0);
16          for (int i = 0; i < n; ++i) head[i].clear();
17          sort(oidx, oidx + n, [&](int a, int b)
18              { return _p[a] < _p[b]; });
19          for (int i = 0; i < n; ++i) p[i] = _p[oidx[i]];
20          divide(0, n - 1);
21      }
22      void addEdge(int u, int v) {
23          head[u].push_front(Edge(v));
24          head[v].push_front(Edge(u));
25          head[u].begin()->twin = head[v].begin();
26          head[v].begin()->twin = head[u].begin();
27      }
28      void divide(int l, int r) {
29          if (l == r) return;
30          if (l + 1 == r) return addEdge(l, l + 1);
31          int mid = (l + r) >> 1, nw[2] = {l, r};
32          divide(l, mid), divide(mid + 1, r);
33          auto gao = [&](int t) {
34              pll pt[2] = {p[nw[0]], p[nw[1]]};
35              for (auto it : head[nw[t]]) {
36                  int v = ori(pt[1], pt[0], p[it.id]);
37                  if (v > 0 || (v == 0 && abs2(pt
38                      [t ^ 1] - p[it.id]) < abs2(pt[1] - pt[0])))
39                      return nw[t] = it.id, true;
40              }
41              return false;
42          };
43          while (gao(0) || gao(1));
44          addEdge(nw[0], nw[1]); // add tangent
45          while (true) {
46              pll pt[2] = {p[nw[0]], p[nw[1]]};
47              int ch = -1, sd = 0;
48              for (int t = 0; t < 2; ++t)
49                  for (auto it : head[nw[t]])
50                      if (ori(pt[0], pt[1],
51                          p[it.id]) > 0 && (ch == -1 || in_cc
52                              ({pt[0], pt[1], p[ch]}, p[it.id])))
53                          ch = it.id, sd = t;
54              if (ch == -1) break; // upper common tangent
55              for (auto it = head
56                  [nw[sd]].begin(); it != head[nw[sd]].end(); )
57                  if (seg_strict_intersect
58                      (pt[sd], p[it->id], pt[sd ^ 1], p[ch]))
59                      head[it->id].erase
60                          (it->twin), head[nw[sd]].erase(it++);
61              else ++it;
62              nw[sd] = ch, addEdge(nw[0], nw[1]);
63          }
64      } tool;

```

8.32 Minimum Enclosing Circle [5f3cdb]

```

1  pdd Minimum_Enclosing_Circle
2      (vector<pdd> dots, double &r) {
3      pdd cent;
4      random_shuffle(ALL(dots));
5      cent = dots[0], r = 0;
6      for (int i = 1; i < SZ(dots); ++i)
7          if (abs(dots[i] - cent) > r) {
8              cent = dots[i], r = 0;
9              for (int j = 0; j < i; ++j)
10                  if (abs(dots[j] - cent) > r) {
11                      cent = (dots[i] + dots[j]) / 2;
12                      r = abs(dots[i] - cent);
13                      for (int k = 0; k < j; ++k)
14                          if (abs(dots[k] - cent) > r)
15                              cent = excenter
16                                  (dots[i], dots[j], dots[k], r);
17                  }
18          }
19      return cent;
20  }

```

8.33 Convex hull [2a3008]

```

1  void hull(vector<pll> &dots) { // n=1 => ans = {}
2      sort(dots.begin(), dots.end());
3      vector<pll> ans(1, dots[0]);

```

```

4      for (int ct = 0; ct < 2; ++ct, reverse(ALL(dots)))
5          for (int i = 1,
6              t = SZ(ans); i < SZ(dots); ans.pb(dots[i++]))
7              while (SZ(ans) > t && ori
8                  (ans[SZ(ans) - 2], ans.back(), dots[i]) <= 0)
9                  ans.pop_back(), ans.swap(dots);

```

9 Else

9.1 ManhattanMST [90cf5a]

```

1  void solve(Point *a, int n) {
2      sort(a, a + n, [](const Point &p, const Point &q) {
3          return p.x + p.y < q.x + q.y;
4      });
5      set<Point> st; // greater<Point::x>
6      for (int i = 0; i < n; ++i) {
7          for (auto it = st.lower_bound(
8              a[i]); it != st.end(); it = st.erase(it)) {
9              if (it ->
10                  x - it -> y < a[i].x - a[i].y) break;
11              es.push_back
12                  ({it -> u, a[i].u, dist(*it, a[i])});
13          }
14          st.insert(a[i]);
15      }
16  }
17  void MST(Point *a, int n) {
18      for (int t = 0; t < 2; ++t) {
19          solve(a, n);
20          for (int
21              i = 0; i < n; ++i) swap(a[i].x, a[i].y);
22          solve(a, n);
23          for (int
24              i = 0; i < n; ++i) a[i].x = -a[i].x;
25      }
26  }

```

9.2 Mos Algorithm With modification [021725]

```

1  /*
2  Mo's Algorithm With modification
3  Block: N^{2/3}, Complexity: N^{5/3}
4  */
5  struct Query {
6      int L, R, LBid, RBid, T;
7      Query(int l, int r, int t):
8          L(l), R(r), LBid(l / blk), RBid(r / blk), T(t) {}
9      bool operator<(const Query &q) const {
10          if (LBid != q.LBid) return LBid < q.LBid;
11          if (RBid != q.RBid) return RBid < q.RBid;
12          return T < q.T;
13      }
14  };
15  void solve(vector<Query> query) {
16      sort(ALL(query));
17      int L=0, R=0, T=-1;
18      for (auto q : query) {
19          while (T < q.T) addTime(L, R, ++T); // TODO
20          while (T > q.T) subTime(L, R, T--); // TODO
21          while (R < q.R) add(arr[++R]); // TODO
22          while (L > q.L) add(arr[--L]); // TODO
23          while (R > q.R) sub(arr[R--]); // TODO
24          while (L < q.L) sub(arr[L--]); // TODO
25          // answer query
26      }
27  }

```

9.3 BitsetLCS [027ab4]

```

1  cin >> n >> m;
2  for (int i = 1; x; i <= n; ++i)
3      cin >> x, p[x].set(i);
4  for (int i = 1; x; i <= m; ++i) {
5      cin >> x, (g = f) |= p[x];
6      f.shiftLeftByOne(), f.set(0);
7      ((f = g - f) ^= g) &= g;
8  }
9  cout << f.count() << '\n';

```

9.4 BinarySearchOnFraction [dec1bd]

```

1  struct Q {
2      ll p, q;
3      Q go(Q b, ll d) { return {p + b.p*d, q + b.q*d}; }
4  };
5  bool pred(Q);

```

```

6 // returns smallest p/q in [lo, hi] such that
7 // pred(p/q) is true, and 0 <= p,q <= N
8 Q frac_bs(ll N) {
9     Q lo{0, 1}, hi{1, 0};
10    if (pred(lo)) return lo;
11    assert(pred(hi));
12    bool dir = 1, L = 1, H = 1;
13    for (; L || H; dir = !dir) {
14        ll len = 0, step = 1;
15        for (int t = 0; t < 2 && (t ? step/=2 : step*=2);)
16            if (Q mid = hi.go(lo, len + step);
17                mid.p > N || mid.q > N || dir ^ pred(mid))
18                t++;
19        else len += step;
20        swap(lo, hi = hi.go(lo, len));
21        (dir ? L : H) = !len;
22    }
23    return dir ? hi : lo;
24 }

```

9.5 SubsetSum [8fa070]

```

1 template<size_t S> // sum(a) < S
2 bitset<S> SubsetSum(const int *a, int n) {
3     vector<int> c(S);
4     bitset<S> dp; dp[0] = 1;
5     for (int i = 0; i < n; ++i) ++c[a[i]];
6     for (size_t i = 1; i < S; ++i) {
7         while (c[i] > 2) c[i] -= 2, ++c[i * 2];
8         while (c[i]--) dp |= dp << i;
9     }
10    return dp;
11 }

```

9.6 DynamicConvexTrick [477879]

```

1 // only works for integer coordinates!! maintain max
2 struct Line {
3     mutable ll a, b, p;
4     bool operator
5         <(const Line &rhs) const { return a < rhs.a; }
6     bool operator<(ll x) const { return p < x; }
7 };
8 struct DynamicHull : multiset<Line, less<>> {
9     static const ll kInf = 1e18;
10    ll Div(ll a,
11           ll b) { return a / b - ((a ^ b) < 0 && a % b); }
12    bool isect(iterator x, iterator y) {
13        if (y == end()) { x->p = kInf; return 0; }
14        if (x
15            ->a == y->a) x->p = x->b > y->b ? kInf : -kInf;
16        else x->p = Div(y->b - x->b, x->a - y->a);
17        return x->p >= y->p;
18    }
19    void addline(ll a, ll b) {
20        auto z = insert({a, b, 0}); y = z++, x = y;
21        while (isect(y, z)) z = erase(z);
22        if (x != begin
23            ()) && isect(--x, y)) isect(x, y = erase(y));
24        while ((y = x) != begin
25            ()) && (--x)->p >= y->p) isect(x, erase(y));
26    }
27    ll query(ll x) {
28        auto l = *lower_bound(x);
29        return l.a * x + l.b;
30    }
31 };

```

9.7 DynamicMST [9ac74a]

```

1 int cnt[maxn], cost[maxn], st[maxn], ed[maxn];
2 pair<int, int> qr[maxn];
3 // qr[i].first = id of edge to
4 // be changed, qr[i].second = weight after operation
5 // cnt[i] = number of operation on edge i
6 // call solve(0, q - 1, v,
7 // 0), where v contains edges i such that cnt[i] == 0
8 void contract(int l, int
9               r, vector<int> v, vector<int> &x, vector<int> &y) {
10    sort(v.begin(), v.end(), [&](int i, int j) {
11        if (cost[i] == cost[j]) return i < j;
12        return cost[i] < cost[j];
13    });
14    djs.save();
15    for (int i = l; i <= r;
16        ++i) djs.merge(st[qr[i].first], ed[qr[i].first]);

```

```

14 for (int i = 0; i < (int)v.size(); ++i) {
15     if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
16         x.push_back(v[i]);
17         djs.merge(st[v[i]], ed[v[i]]);
18     }
19 }
20 djs.undo();
21 djs.save();
22 for (int i = 0; i < (
23     int)x.size(); ++i) djs.merge(st[x[i]], ed[x[i]]);
24 for (int i = 0; i < (int)v.size(); ++i) {
25     if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
26         y.push_back(v[i]);
27         djs.merge(st[v[i]], ed[v[i]]);
28     }
29 }
30 djs.undo();
31 }
32 void solve(int l, int r, vector<int> v, long long c) {
33     if (l == r) {
34         cost[qr[l].first] = qr[l].second;
35         if (st[qr[l].first] == ed[qr[l].first]) {
36             printf("%lld\n", c);
37             return;
38         }
39         int minv = qr[l].second;
40         for (int i = 0; i < (int
41             )v.size(); ++i) minv = min(minv, cost[v[i]]);
42         printf("%lld\n", c + minv);
43         return;
44     }
45     int m = (l + r) >> 1;
46     vector<int> lv = v, rv = v;
47     vector<int> x, y;
48     for (int i = m + 1; i <= r; ++i) {
49         cnt[qr[i].first]--;
50         if (cnt
51             [qr[i].first] == 0) lv.push_back(qr[i].first);
52     }
53     contract(l, m, lv, x, y);
54     long long lc = c, rc = c;
55     djs.save();
56     for (int i = 0; i < (int)x.size(); ++i) {
57         lc += cost[x[i]];
58         djs.merge(st[x[i]], ed[x[i]]);
59     }
60     solve(l, m, y, lc);
61     djs.undo();
62     x.clear(), y.clear();
63     for (int i = m + 1; i <= r; ++i) cnt[qr[i].first]++;
64     for (int i = l; i <= m; ++i) {
65         cnt[qr[i].first]--;
66         if (cnt
67             [qr[i].first] == 0) rv.push_back(qr[i].first);
68     }
69     contract(m + 1, r, rv, x, y);
70     djs.save();
71     for (int i = 0; i < (int)x.size(); ++i) {
72         rc += cost[x[i]];
73         djs.merge(st[x[i]], ed[x[i]]);
74     }
75     solve(m + 1, r, y, rc);
76     djs.undo();
77     for (int i = l; i <= m; ++i) cnt[qr[i].first]++;
78 }

```

9.8 Matroid

Start from $S = \emptyset$. In each iteration, let

- $Y_1 = \{x \notin S \mid S \cup \{x\} \in I_1\}$
- $Y_2 = \{x \notin S \mid S \cup \{x\} \in I_2\}$

If there exists $x \in Y_1 \cap Y_2$, insert x into S . Otherwise for each $x \in S, y \notin S$, create edges

- $x \rightarrow y$ if $S - \{x\} \cup \{y\} \in I_1$.
- $y \rightarrow x$ if $S - \{x\} \cup \{y\} \in I_2$.

Find a *shortest* path (with BFS) starting from a vertex in Y_1 and ending at a vertex in Y_2 which doesn't pass through any other vertices in Y_2 , and alternate the path. The size of S will be incremented by 1 in each iteration. For the weighted case, assign weight $w(x)$ to vertex x if $x \in S$ and $-w(x)$ if $x \notin S$. Find the path with the minimum number of edges among all minimum length paths and alternate it.

9.9 cyclicLCS [9b01d1]

```

1 #define L 0
2 #define LU 1
3 #define U 2

```



```

4 const int mov[3][2] = {0, -1, -1, -1, -1, 0};
5 int al, bl;
6 char a[MAXL * 2], b[MAXL * 2]; // 0-indexed
7 int dp[MAXL * 2][MAXL];
8 char pred[MAXL * 2][MAXL];
9 inline int lcs_length(int r) {
10     int i = r + al, j = bl, l = 0;
11     while (i > r) {
12         char dir = pred[i][j];
13         if (dir == LU) l++;
14         i += mov[dir][0];
15         j += mov[dir][1];
16     }
17     return l;
18 }
19 inline void reroot(int r) { // r = new base row
20     int i = r, j = 1;
21     while (j <= bl && pred[i][j] != LU) j++;
22     if (j > bl) return;
23     pred[i][j] = L;
24     while (i < 2 * al && j <= bl) {
25         if (pred[i + 1][j] == U) {
26             i++;
27             pred[i][j] = L;
28         } else if (j < bl && pred[i + 1][j + 1] == LU) {
29             i++;
30             j++;
31             pred[i][j] = L;
32         } else {
33             j++;
34         }
35     }
36 }
37 int cyclic_lcs() {
38     // a, b, al, bl should be properly filled
39     // note: a WILL be altered in process
40     // -- concatenated after itself
41     char tmp[MAXL];
42     if (al > bl) {
43         swap(al, bl);
44         strcpy(tmp, a);
45         strcpy(a, b);
46         strcpy(b, tmp);
47     }
48     strcpy(tmp, a);
49     strcat(a, tmp);
50     // basic lcs
51     for (int i = 0; i <= 2 * al; i++) {
52         dp[i][0] = 0;
53         pred[i][0] = U;
54     }
55     for (int j = 0; j <= bl; j++) {
56         dp[0][j] = 0;
57         pred[0][j] = L;
58     }
59     for (int i = 1; i <= 2 * al; i++) {
60         for (int j = 1; j <= bl; j++) {
61             if (a[i - 1] == b[j - 1])
62                 dp[i][j] = dp[i - 1][j - 1] + 1;
63             else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
64             if (dp[i][j - 1] == dp[i][j]) pred[i][j] = L;
65             else if (a[i - 1] == b[j - 1]) pred[i][j] = LU;
66             else pred[i][j] = U;
67         }
68     }
69     // do cyclic lcs
70     int clcs = 0;
71     for (int i = 0; i < al; i++) {
72         clcs = max(clcs, lcs_length(i));
73         reroot(i + 1);
74     }
75     // recover a
76     a[al] = '\0';
77     return clcs;
78 }

```

9.10 HilbertCurve [bc6dec]

```

1 ll hilbert(int n, int x, int y) {
2     ll res = 0;
3     for (int s = n / 2; s; s >>= 1) {
4         int rx = (x & s) > 0;
5         int ry = (y & s) > 0;
6         res += s * 1ll * s * ((3 * rx) ^ ry);
7         if (ry == 0) {
8             if (rx == 1) x = s - 1 - x, y = s - 1 - y;
9             swap(x, y);

```

```

10     }
11     }
12     return res;
13 } // n = 2^k

```

9.11 Mos Algorithm On Tree [90ac22]

```

1 /*
2 Mo's Algorithm On Tree
3 Preprocess:
4 1) LCA
5 2) dfs with in[u] = dft++, out[u] = dft++
6 3) ord[in[u]] = ord[out[u]] = u
7 4) bitset<MAXN> inset
8 */
9 struct Query {
10     int L, R, LBid, lca;
11     Query(int u, int v) {
12         int c = LCA(u, v);
13         if (c == u || c == v)
14             q.lca = -1, q.L = out[c ^ u ^ v], q.R = out[c];
15         else if (out[u] < in[v])
16             q.lca = c, q.L = out[u], q.R = in[v];
17         else
18             q.lca = c, q.L = out[v], q.R = in[u];
19         q.Lid = q.L / blk;
20     }
21     bool operator<(const Query &q) const {
22         if (LBid != q.LBid) return LBid < q.LBid;
23         return R < q.R;
24     }
25 };
26 void flip(int x) {
27     if (inset[x]) sub(arr[x]); // TODO
28     else add(arr[x]); // TODO
29     inset[x] = ~inset[x];
30 }
31 void solve(vector<Query> query) {
32     sort(ALL(query));
33     int L = 0, R = 0;
34     for (auto q : query) {
35         while (R < q.R) flip(ord[++R]);
36         while (L > q.L) flip(ord[--L]);
37         while (R > q.R) flip(ord[R--]);
38         while (L < q.L) flip(ord[L++]);
39         if (~q.lca) add(arr[q.lca]);
40         // answer query
41         if (~q.lca) sub(arr[q.lca]);
42     }
43 }

```

9.12 AdaptiveSimpson [c048eb]

```

1 template<typename Func, typename d = double>
2 struct Simpson {
3     using pdd = pair<d, d>;
4     Func f;
5     pdd mix(pdd l, pdd r, optional<d> fm = {}) {
6         d h = (r.X - l.X) / 2, v = fm.value_or(f(l.X + h));
7         return {v, h / 3 * (l.Y + 4 * v + r.Y)};
8     }
9     d eval(pdd l, pdd r, d fm, d eps) {
10         pdd m((l.X + r.X) / 2, fm);
11         d s = mix(l, r, fm).second;
12         auto [flm, sl] = mix(l, m);
13         auto [fmr, sr] = mix(m, r);
14         d delta = sl + sr - s;
15         if (abs(delta)
16             ) <= 15 * eps) return sl + sr + delta / 15;
17         return eval(l, m, flm, eps / 2) +
18             eval(m, r, fmr, eps / 2);
19     }
20     d eval(d l, d r, d eps) {
21         return eval(
22             ({l, f(l)}, {r, f(r)}), f((l + r) / 2), eps);
23     }
24     d eval2(d l, d r, d eps, int k = 997) {
25         d h = (r - l) / k, s = 0;
26         for (int i = 0; i < k; ++i, l += h)
27             s += eval(l, l + h, eps / k);
28         return s;
29     };
30 template<typename Func>
31 Simpson<Func> make_simpson(Func f) { return {f}; }

```


9.13 min plus convolution [b08fbf]

```

1 // a is convex a[i+1]-a[i] <= a[i+2]-a[i+1]
2 vector<int> min_plus_convolution
  (vector<int> &a, vector<int> &b) {
3   int n = SZ(a), m = SZ(b);
4   vector<int> c(n + m - 1, INF);
5   auto dc = [&](auto Y, int l, int r, int jl, int jr) {
6     if (l > r) return;
7     int mid = (l + r) / 2, from = -1, &best = c[mid];
8     for (int j = jl; j <= jr; ++j)
9       if (int i = mid - j; i >= 0 && i < n)
10        if (best > a[i] + b[j])
11         best = a[i] + b[j], from = j;
12     Y(Y, l, mid - 1, jl, from), Y(Y, mid + 1, r, from, jr);
13   };
14   return dc(dc, 0, n - 1 + m - 1, 0, m - 1), c;
15 }

```

9.14 cyc tsearch [3dac64]

```

1 /* bool pred(int a, int b);
2  f(0) ~ f(n - 1) is a cyclic-shift U-function
3  return idx s.t. pred(x, idx) is false forall x*/
4 int cyc_tsearch(int n, auto pred) {
5   if (n == 1) return 0;
6   int l = 0, r = n; bool rv = pred(1, 0);
7   while (r - l > 1) {
8     int m = (l + r) / 2;
9     if (pred(0, m) ? rv : pred(m, (m + 1) % n)) r = m;
10    else l = m;
11   }
12   return pred(l, r % n) ? l : r % n;
13 }

```

9.15 All LCS [5548b0]

```

1 void all_lcs(string s, string t) { // 0-base
2   vector<int> h(SZ(t));
3   iota(ALL(h), 0);
4   for (int a = 0; a < SZ(s); ++a) {
5     int v = -1;
6     for (int c = 0; c < SZ(t); ++c)
7       if (s[a] == t[c] || h[c] < v)
8         swap(h[c], v);
9     // LCS(s[0, a], t[b, c]) =
10    // c - b + 1 - sum([h[i] >= b] | i <= c)
11    // h[i] might become -1 !!
12  }
13 }

```

9.16 NQueens [68bc5d]

```

1 void solve
  (vector<int> &ret, int n) { // no sol when n=2,3
2   if (n % 6 == 2) {
3     for (int i = 2; i <= n; i += 2) ret.pb(i);
4     ret.pb(3); ret.pb(1);
5     for (int i = 7; i <= n; i += 2) ret.pb(i);
6     ret.pb(5);
7   } else if (n % 6 == 3) {
8     for (int i = 4; i <= n; i += 2) ret.pb(i);
9     ret.pb(2);
10    for (int i = 5; i <= n; i += 2) ret.pb(i);
11    ret.pb(1); ret.pb(3);
12  } else {
13    for (int i = 2; i <= n; i += 2) ret.pb(i);
14    for (int i = 1; i <= n; i += 2) ret.pb(i);
15  }
16 }

```

9.17 Mos Algorithm

- Mo's Algorithm With Addition Only
 - Sort queries same as the normal Mo's algorithm.
 - For each query $[l, r]$:
 - If $l/blk = r/blk$, brute-force.
 - If $l/blk \neq r/blk$, initialize $curL := (l/blk + 1) \cdot blk$, $curR := curL - 1$.
 - If $r > curR$, increase $curR$.
 - decrease $curL$ to fit l , and then undo after answering
- Mo's Algorithm With Offline Second Time
 - Require: Changing answer \equiv adding $f([l, r], r+1)$.
 - Require: $f([l, r], r+1) = f([l, r], r+1) - f([l, l], r+1)$.
 - Part1: Answer all $f([l, r], r+1)$ first.
 - Part2: Store $curR \rightarrow R$ for $curL$ (reduce the space to $O(N)$), and then answer them by the second offline algorithm.
 - Note: You must do the above symmetrically for the left boundaries.

9.18 simulated annealing [d19317]

```

1 double factor = 100000;
2 const int base = 1e9; // remember to run ~ 10 times
3 for (int it = 1; it <= 100000; ++it) {
4   // ans:
5   answer, nw: current value, rnd(): mt19937 rnd()
6   if (exp(-(nw - ans) / factor) >= (double)(rnd() % base) / base)
7     ans = nw;
8   factor *= 0.99995;
9 }

```

9.19 DLX [5d57fa]

```

1 #define TRAV(i, link, start)
2   for (int i = link[start]; i != start; i = link[i])
3 template<
4   bool E> // E: Exact, NN: num of 1s, RR: num of rows
5 struct DLX {
6   int lt[NN], rg[NN], up[NN], dn[NN]
7   , rw[NN], cl[NN], bt[NN], s[NN], head, sz, ans;
8   int rows, columns;
9   bool vis[NN];
10  bitset<RR> sol, cur; // not sure
11  void remove(int c) {
12    if (E) lt[rg[c]] = lt[c], rg[lt[c]] = rg[c];
13    TRAV(i, dn, c) {
14      if (E) {
15        TRAV(j, rg, i)
16        up[dn[j]] = up[j], dn[up[j]] = dn[j], --s[cl[j]];
17      } else {
18        lt[rg[i]] = lt[i], rg[lt[i]] = rg[i];
19      }
20    }
21  }
22  void restore(int c) {
23    TRAV(i, up, c) {
24      if (E) {
25        TRAV(j, lt, i)
26        ++s[cl[j]], up[dn[j]] = j, dn[up[j]] = j;
27      } else {
28        lt[rg[i]] = rg[lt[i]] = i;
29      }
30    }
31    if (E) lt[rg[c]] = c, rg[lt[c]] = c;
32  }
33  void init(int c) {
34    rows = 0, columns = c;
35    for (int i = 0; i < c; ++i) {
36      up[i] = dn[i] = bt[i] = i;
37      lt[i] = i == 0 ? c : i - 1;
38      rg[i] = i == c - 1 ? c : i + 1;
39      s[i] = 0;
40    }
41    rg[c] = 0, lt[c] = c - 1;
42    up[c] = dn[c] = -1;
43    head = c, sz = c + 1;
44  }
45  void insert(const vector<int> &col) {
46    if (col.empty()) return;
47    int f = sz;
48    for (int i = 0; i < (int)col.size(); ++i) {
49      int c = col[i], v = sz++;
50      dn[bt[c]] = v;
51      up[v] = bt[c], bt[c] = v;
52      rg[v] = (i + 1 == (int)col.size() ? f : v + 1);
53      rw[v] = rows, cl[v] = c;
54      ++s[c];
55      if (i > 0) lt[v] = v - 1;
56    }
57    ++rows, lt[f] = sz - 1;
58  }
59  int h() {
60    int ret = 0;
61    fill_n(vis, sz, false);
62    TRAV(x, rg, head) {
63      if (vis[x]) continue;
64      vis[x] = true, ++ret;
65      TRAV(i, dn, x) TRAV(j, rg, i) vis[cl[j]] = true;
66    }
67    return ret;
68  }
69  void dfs(int dep) {
70    if (dep + (E ? 0 : h()) >= ans) return;
71    if (rg[head] == head) return sol = cur, ans = dep, void();

```

```

69     if (dn[rg[head]] == rg[head]) return;
70     int w = rg[head];
71     TRAV(x, rg, head) if (s[x] < s[w]) w = x;
72     if (E) remove(w);
73     TRAV(i, dn, w) {
74         if (!E) remove(i);
75         TRAV(j, rg, i) remove(E ? cl[j] : j);
76         cur.set(rw[i]), dfs(dep + 1), cur.reset(rw[i]);
77         TRAV(j, lt, i) restore(E ? cl[j] : j);
78         if (!E) restore(i);
79     }
80     if (E) restore(w);
81 }
82 int solve() {
83     for (int i = 0; i < columns; ++i)
84         dn[bt[i]] = i, up[i] = bt[i];
85     ans = 1e9, sol.reset(), dfs(0);
86     return ans;
87 }
88 };

```

9.20 tree hash [95e839]

```

1 ull seed;
2 ull shift(ull x) {
3     x ^= x << 13;
4     x ^= x >> 7;
5     x ^= x << 17;
6     return x;
7 }
8 ull dfs(int u, int f) {
9     ull sum = seed;
10    for (int i : G[u])
11        if (i != f)
12            sum += shift(dfs(i, u));
13    return sum;
14 }

```

9.21 DynamicConvexTrick bb [85e4f7]

```

1 // only works for integer coordinates!!
2
3 bool Flag; // 0: insert Line, 1: lower_bound x
4 template<class val = ll,
5         class compare = less<val>> // sort lines with comp
6 struct DynamicConvexTrick{
7     static const ll minx = 0, maxx = ll(1e9) + 5;
8     static compare comp;
9     struct Line{
10         val a, b, l, r; // line ax + b in [l, r]
11         Line(val _a, val _b, val _l = minx,
12              val _r = maxx):a(_a), b(_b), l(_l), r(_r){}
13         val operator () (val x) const {
14             return a * x + b;
15         }
16     };
17     struct cmp{
18         bool operator () (const Line a, const Line b){
19             if(Flag == 0) return comp(a.a, b.a);
20             return a.r < b.l;
21         }
22     };
23     inline val idiv(val a, val b){
24         return a / b - (a % b && a < 0 ^ b < 0);
25     }
26     set<Line, cmp> st;
27     void ins(val a, val b){
28         Flag = 0;
29         Line L(a, b);
30         auto it = st.lower_bound(L);
31         if(it != st.begin() && it != st.end())
32             if(!comp((*prev(it))(it->l - 1), L(
33                 it->l - 1)) && !comp((*it)(it->l), L(it->l)))
34                 return;
35         while(it != st.end()){
36             if(it->a == L.a && !comp(it->b, L.b)) return;
37             if(comp(
38                 ((*it)(it->r), L(it->r))) it = st.erase(it);
39             else{
40                 Line M = *it;
41                 st.erase(it);
42                 L.r = max(idiv(L.b - M.b, M.a - L.a), minx);
43                 M.l = L.r + 1;
44                 it = st.insert(M).X;
45                 break;
46             }
47         }
48     }
49 };

```

```

44 while(it != st.begin()){
45     auto pit = prev(it);
46     if(comp((*pit)(pit->l), L(pit->l))) st.erase(pit);
47     else{
48         Line M = *pit;
49         st.erase(pit);
50         M.r =
51             min(idiv(L.b - M.b, M.a - L.a), maxx - 1);
52         L.l = M.r + 1;
53         st.insert(M);
54         break;
55     }
56 }
57 st.insert(L);
58 }
59 val operator () (val x){
60     Flag = 1;
61     auto it = st.lower_bound({0, 0, x, x});
62     return (*it)(x);
63 }
64 };
65 DynamicConvexTrick<> DCT;

```

10 JAVA

10.1 Big number [05dd09]

```

1 import java.util.Scanner;
2 import java.math.BigInteger;
3
4 public class JAVA{
5     public static void main(String[] args){
6         Scanner cin = new Scanner(System.in);
7         String a, b, c;
8         while(cin.hasNext()){
9             a = cin.next();
10            b = cin.next();
11            c = cin.next();
12            BigInteger ia = new BigInteger(a);
13            BigInteger ic = new BigInteger(c);
14            if(b.charAt(0) == '+')
15                System.out.printf("%s\n", ia.add(ic));
16            if(b.charAt(0) == '-')
17                System.out.printf("%s\n", ia.subtract(ic));
18            if(b.charAt(0) == '*')
19                System.out.printf("%s\n", ia.multiply(ic));
20            if(b.charAt(0) == '/')
21                System.out.printf("%s\n", ia.divide(ic));
22        }
23    }
24 }

```

11 Python

11.1 misc

```

1 from decimal import *
2 setcontext(Context(prec
3     =MAX_PREC, Emax=MAX_EMAX, rounding=ROUND_FLOOR))
4 print(Decimal(input()) * Decimal(input()))
5 from fractions import Fraction
6 Fraction
7     ('3.14159').limit_denominator(10).numerator # 22

```