Contents

1 Basic 1
- 1.1 vimrc 1
- 1.2 readchar 1
- 1.3 BigIntIO 1
- 1.4 Black Magic 1
- 1.5 Pragma Optimization 1
- 1.6 Bitset 1

2 Graph 2
- 2.1 BCC Vertex* 2
- 2.2 Bridge* 2
- 2.3 SCC* 2
- 2.4 2SAT* 2
- 2.5 MinimumMeanCycle* 2
- 2.6 Virtual Tree* 3
- 2.7 Maximum Clique Dyn* 3
- 2.8 Minimum Steiner Tree* 3
- 2.9 Dominator Tree* . . 3
- 2.10 Minimum Clique Cover* 4
- 2.11 NumberofMaximalClique* 4

3 Data Structure 4
- 3.1 Discrete Trick 4
- 3.2 BIT kth* 4
- 3.3 Interval Container* . 4
- 3.4 Leftist Tree 4
- 3.5 Heavy light Decomposition* 5
- 3.6 Centroid Decomposition* 5
- 3.7 LiChaoST* 5
- 3.8 Link cut tree* 5
- 3.9 KDTree 6
- 3.10 Treap 6

4 Flow/Matching 7
- 4.1 Bipartite Matching* 7
- 4.2 Kuhn Munkres* . . . 7
- 4.3 MincostMaxflow* . . 7
- 4.4 Maximum Simple Graph Matching* . . 8
- 4.5 Maximum Weight Matching* 8
- 4.6 SW-mincut 9
- 4.7 BoundedFlow*(Dinic*) 9
- 4.8 Gomory Hu tree* . . 10
- 4.9 Minimum Cost Circulation* 10
- 4.10 Flow Models 10

5 String 11
- 5.1 KMP 11
- 5.2 Z-value* 11
- 5.3 Manacher* 11
- 5.4 SAIS* 11
- 5.5 Aho-Corasick Automatan* 11
- 5.6 Smallest Rotation . . 11
- 5.7 De Bruijn sequence* 12
- 5.8 Extended SAM* . . . 12
- 5.9 PalTree* 12
- 5.10 Main Lorentz 12

6 Math 13
- 6.1 ax+by=gcd(only exgcd *) 13
- 6.2 Floor and Ceil 13
- 6.3 Floor Enumeration . 13
- 6.4 Mod Min 13
- 6.5 Linear Mod Inverse . 13
- 6.6 Linear Filter Mu . . . 13
- 6.7 Gaussian integer gcd 13
- 6.8 GaussElimination . . 13
- 6.9 Miller Rabin* 13
- 6.10 Simultaneous Equations 13
- 6.11 Pollard Rho* 14
- 6.12 Simplex Algorithm . 14
  - 6.12.1 Construction . 14
- 6.13 chineseRemainder . . 14
- 6.14 Factorial without prime factor* 14
- 6.15 Discrete Log* 14
- 6.16 Berlekamp Massey . 14
- 6.17 Primes 15
- 6.18 Theorem 15
- 6.19 Estimation 15
- 6.20 Euclidean Algorithms 15
- 6.21 General Purpose Numbers 16
- 6.22 Tips for Generating Functions 16

7 Polynomial 16
- 7.1 Fast Fourier Transform 16
- 7.2 Number Theory Transform* 16
- 7.3 Fast Walsh Transform* 16
- 7.4 Polynomial Operation 17
- 7.5 Value Polynomial . . 18
- 7.6 Newton's Method . . 18

8 Geometry 18
- 8.1 Basic 18
- 8.2 KD Tree 18
- 8.3 Sector Area 19
- 8.4 Half Plane Intersection 19
- 8.5 Rotating Sweep Line 19
- 8.6 Triangle Center . . . 19
- 8.7 Polygon Center . . . 19
- 8.8 Maximum Triangle . 19
- 8.9 Point in Polygon . . . 19
- 8.10 Circle 20
- 8.11 Tangent of Circles and Points to Circle . 20
- 8.12 Area of Union of Circles 20
- 8.13 Minimun Distance of 2 Polygons 21
- 8.14 2D Convex Hull . . . 21
- 8.15 3D Convex Hull . . . 22
- 8.16 Minimum Enclosing Circle 22
- 8.17 Closest Pair 22

9 Else 23
- 9.1 Cyclic Ternary Search* 23
- 9.2 Mo's Algorithm(With modification) 23
- 9.3 Mo's Algorithm On Tree 23
- 9.4 Additional Mo's Algorithm Trick 23
- 9.5 Hilbert Curve 23
- 9.6 DynamicConvexTrick* 23
- 9.7 All LCS* 24
- 9.8 AdaptiveSimpson* . 24
- 9.9 Simulated Annealing 24
- 9.10 Tree Hash* 24
- 9.11 Binary Search On Fraction 24
- 9.12 Bitset LCS 24

10 Python 24
- 10.1 Misc 24

1 Basic

1.1 vimrc

```
"This file should be placed at ~/.vimrc"
se nu ai hls et ru ic is sc cul
se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
syntax on
hi cursorline cterm=none ctermbg=89
set bg=dark
inoremap {<CR> {<CR>}<Esc>ko<tab>
"Select
    region and then type :Hash to hash your selection."
"Useful for verifying that there aren't mistypes."
ca Hash w !cpp -dD -P -fpreprocessed
    \| tr -d '[:space:]' \| md5sum \| cut -c-6
```

1.2 readchar [0754b0]

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

1.3 BigIntIO [ea947e]

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
bool cmp(__int128 x, __int128 y) { return x > y; }
```

1.4 Black Magic [d566f1]

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
#include <ext/rope>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
#include <ext/pb_ds/assoc_container.hpp>
typedef
     tree<int, null_type, std::less<int>, rb_tree_tag
    , tree_order_statistics_node_update> tree_set;
typedef cc_hash_table<int, int> umap;
typedef priority_queue<int> heap;

int main() {
  // rb tree
  tree_set s;
  s.insert(71); s.insert(22);
  assert(*s.find_by_order
      (0) == 22); assert(*s.find_by_order(1) == 71);
  assert(s.order_of_key
      (22) == 0); assert(s.order_of_key(71) == 1);
  s.erase(22);
  assert(*s.find_by_order
      (0) == 71); assert(s.order_of_key(71) == 0);
  // mergable heap
  heap a, b; a.join(b);
  // persistant
  rope<char> r[2];
  r[1] = r[0];
  std::string st = "abc";
  r[1].insert(0, st.c_str());
  r[1].erase(1, 1);
  std::cout << r[1].substr(0, 2) << std::endl;
  return 0;
}
```

1.5 Pragma Optimization [7b330a]

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

1.6 Bitset [282252]

```
#include<bits/stdc++.h>
using namespace std;

int main () {
    bitset<4> bit;
```

```
    bit.all(); // all bit is true, ret tru;
    bit.any();  // any bit is true, ret true
    bit.none(); // all bit is false, ret true
    bit.count();
    bit.to_string('0', '1');//with parmeter
    bit.reset(); // set all to true
    bit.set(); // set all to false
    std::bitset<8> b3{0}, b4{42};
    std::hash<std::bitset<8>> hash_fn8;
    hash_fn8(b3); hash_fn8(b4);
}
```

2 Graph

2.1 BCC Vertex* [ed8308]

```
struct BCC { // 0-base
  int n, dft, nbcc;
  vector<int> low, dfn, bln, stk, is_ap, cir;
  vector<vector<int>> G, bcc, nG;
  void make_bcc(int u) {
    bcc.emplace_back(1, u);
    for (; stk.back() != u; stk.pop_back())
      bln[stk.back()] = nbcc, bcc[nbcc].pb(stk.back());
    stk.pop_back(), bln[u] = nbcc++;
  }
  void dfs(int u, int f) {
    int child = 0;
    low[u] = dfn[u] = ++dft, stk.pb(u);
    for (int v : G[u])
      if (!dfn[v]) {
        dfs(v, u), ++child;
        low[u] = min(low[u], low[v]);
        if (dfn[u] <= low[v]) {
          is_ap[u] = 1, bln[u] = nbcc;
          make_bcc(v), bcc.back().pb(u);
        }
      } else if (dfn[v] < dfn[u] && v != f)
        low[u] = min(low[u], dfn[v]);
    if (f == -1 && child < 2) is_ap[u] = 0;
    if (f == -1 && child == 0) make_bcc(u);
  }
  BCC(int _n): n(_n), dft(),
      nbcc(), low(n), dfn(n), bln(n), is_ap(n), G(n) {}
  void add_edge(int u, int v) {
    G[u].pb(v), G[v].pb(u);
  }
  void solve() {
    for (int i = 0; i < n; ++i)
      if (!dfn[i]) dfs(i, -1);
  }
  void block_cut_tree() {
    cir.resize(nbcc);
    for (int i = 0; i < n; ++i)
      if (is_ap[i])
        bln[i] = nbcc++;
    cir.resize(nbcc, 1), nG.resize(nbcc);
    for (int i = 0; i < nbcc && !cir[i]; ++i)
      for (int j : bcc[i])
        if (is_ap[j])
          nG[i].pb(bln[j]), nG[bln[j]].pb(i);
  } // up to 2 * n - 2 nodes!! bln[i] for id
};
```

2.2 Bridge* [0ccada]

```
struct ECC { // 0-base
  int n, dft, ecnt, necc;
  vector<int> low, dfn, bln, is_bridge, stk;
  vector<vector<pii>> G;
  void dfs(int u, int f) {
    dfn[u] = low[u] = ++dft, stk.pb(u);
    for (auto [v, e] : G[u])
      if (!dfn[v])
        dfs(v, e), low[u] = min(low[u], low[v]);
      else if (e != f)
        low[u] = min(low[u], dfn[v]);
    if (low[u] == dfn[u]) {
      if (f != -1) is_bridge[f] = 1;
      for (; stk.back() != u; stk.pop_back())
        bln[stk.back()] = necc;
      bln[u] = necc++, stk.pop_back();
    }
  }
  ECC(int _n): n(_n), dft()
      , ecnt(), necc(), low(n), dfn(n), bln(n), G(n) {}
  void add_edge(int u, int v) {
    G[u].pb(pii(v, ecnt)), G[v].pb(pii(u, ecnt++));
  }
  void solve() {
    is_bridge.resize(ecnt);
    for (int i = 0; i < n; ++i)
      if (!dfn[i]) dfs(i, -1);
  }
}; // ecc_id(i): bln[i]
```

2.3 SCC* [22afe1]

```
struct SCC { // 0-base
  int n, dft, nscc;
  vector<int> low, dfn, bln, instack, stk;
  vector<vector<int>> G;
  void dfs(int u) {
    low[u] = dfn[u] = ++dft;
    instack[u] = 1, stk.pb(u);
    for (int v : G[u])
      if (!dfn[v])
        dfs(v), low[u] = min(low[u], low[v]);
      else if (instack[v] && dfn[v] < dfn[u])
        low[u] = min(low[u], dfn[v]);
    if (low[u] == dfn[u]) {
      for (; stk.back() != u; stk.pop_back())
        bln[stk
          .back()] = nscc, instack[stk.back()] = 0;
      instack[u] = 0, bln[u] = nscc++, stk.pop_back();
    }
  }
  SCC(int _n): n(_n), dft(), nscc
      (), low(n), dfn(n), bln(n), instack(n), G(n) {}
  void add_edge(int u, int v) {
    G[u].pb(v);
  }
  void solve() {
    for (int i = 0; i < n; ++i)
      if (!dfn[i]) dfs(i);
  }
}; // scc_id(i): bln[i]
```

2.4 2SAT* [e839e5]

```
struct SAT { // 0-base
  int n;
  vector<bool> istrue;
  SCC scc;
  SAT(int _n): n(_n), istrue(n + n), scc(n + n) {}
  int rv(int a) {
    return a >= n ? a - n : a + n;
  }
  void add_clause(int a, int b) {
    scc.add_edge(rv(a), b), scc.add_edge(rv(b), a);
  }
  bool solve() {
    scc.solve();
    for (int i = 0; i < n; ++i) {
      if (scc.bln[i] == scc.bln[i + n]) return false;
      istrue[i] = scc.bln[i] < scc.bln[i + n];
      istrue[i + n] = !istrue[i];
    }
    return true;
  }
};
```

2.5 MinimumMeanCycle* [4be648]

```
ll road[N][N]; // input here
struct MinimumMeanCycle {
  ll dp[N + 5][N], n;
  pll solve() {
    ll a = -1, b = -1, L = n + 1;
    for (int i = 2; i <= L; ++i)
      for (int k = 0; k < n; ++k)
        for (int j = 0; j < n; ++j)
          dp[i][j] =
            min(dp[i - 1][k] + road[k][j], dp[i][j]);
    for (int i = 0; i < n; ++i) {
      if (dp[L][i] >= INF) continue;
      ll ta = 0, tb = 1;
      for (int j = 1; j < n; ++j)
        if (dp[j][i] < INF &&
          ta * (L - j) < (dp[L][i] - dp[j][i]) * tb)
          ta = dp[L][i] - dp[j][i], tb = L - j;
      if (ta == 0) continue;
      if (a == -1 || a * tb > ta * b) a = ta, b = tb;
    }
```

```
    if (a != -1) {
      ll g = __gcd(a, b);
      return pll(a / g, b / g);
    }
    return pll(-1LL, -1LL);
  }
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j) dp[i + 2][j] = INF;
  }
};
```

2.6 Virtual Tree* [80f7cb]

```
vector<int> vG[N];
int top, st[N];

void insert(int u) {
  if (top == -1) return st[++top] = u, void();
  int p = LCA(st[top], u);
  if (p == st[top]) return st[++top] = u, void();
  while (top >= 1 && dep[st[top - 1]] >= dep[p])
    vG[st[top - 1]].pb(st[top]), --top;
  if (st[top] != p)
    vG[p].pb(st[top]), --top, st[++top] = p;
  st[++top] = u;
}

void reset(int u) {
  for (int i : vG[u]) reset(i);
  vG[u].clear();
}

void solve(vector<int> &v) {
  top = -1;
  sort(ALL(v),
    [&](int a, int b) { return dfn[a] < dfn[b]; });
  for (int i : v) insert(i);
  while (top > 0) vG[st[top - 1]].pb(st[top]), --top;
  // do something
  reset(v[0]);
}
```

2.7 Maximum Clique Dyn* [4a6b3d]

```
struct MaxClique { // fast when N <= 100
  bitset<N> G[N], cs[N];
  int ans, sol[N], q, cur[N], d[N], n;
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) G[i].reset();
  }
  void add_edge(int u, int v) {
    G[u][v] = G[v][u] = 1;
  }
  void pre_dfs(vector<int> &r, int l, bitset<N> mask) {
    if (l < 4) {
      for (int i : r) d[i] = (G[i] & mask).count();
      sort(ALL(r)
        , [&](int x, int y) { return d[x] > d[y]; });
    }
    vector<int> c(SZ(r));
    int lft = max(ans - q + 1, 1), rgt = 1, tp = 0;
    cs[1].reset(), cs[2].reset();
    for (int p : r) {
      int k = 1;
      while ((cs[k] & G[p]).any()) ++k;
      if (k > rgt) cs[++rgt + 1].reset();
      cs[k][p] = 1;
      if (k < lft) r[tp++] = p;
    }
    for (int k = lft; k <= rgt; ++k)
      for (int p = cs[k]._Find_first
        (); p < N; p = cs[k]._Find_next(p))
        r[tp] = p, c[tp] = k, ++tp;
    dfs(r, c, l + 1, mask);
  }
  void dfs(vector<
      int> &r, vector<int> &c, int l, bitset<N> mask) {
    while (!r.empty()) {
      int p = r.back();
      r.pop_back(), mask[p] = 0;
      if (q + c.back() <= ans) return;
      cur[q++] = p;
      vector<int> nr;
      for (int i : r) if (G[p][i]) nr.pb(i);
      if (!nr.empty()) pre_dfs(nr, l, mask & G[p]);
      else if (q > ans) ans = q, copy_n(cur, q, sol);
      c.pop_back(), --q;
    }
  }
  int solve() {
    vector<int> r(n);
    ans = q = 0, iota(ALL(r), 0);
    pre_dfs(r, 0, bitset<N>(string(n, '1')));
    return ans;
  }
};
```

2.8 Minimum Steiner Tree* [cbf811]

```
struct SteinerTree { // 0-base
  int n, dst[N][N], dp[1 << T][N], tdst[N];
  int vcst[N]; // the cost of vertexs
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) {
      fill_n(dst[i], n, INF);
      dst[i][i] = vcst[i] = 0;
    }
  }
  void chmin(int &x, int val) {
    x = min(x, val);
  }
  void add_edge(int ui, int vi, int wi) {
    chmin(dst[ui][vi], wi);
  }
  void shortest_path() {
    for (int k = 0; k < n; ++k)
      for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
          chmin(dst[i][j], dst[i][k] + dst[k][j]);
  }
  int solve(const vector<int>& ter) {
    shortest_path();
    int t = SZ(ter), full = (1 << t) - 1;
    for (int i = 0; i <= full; ++i)
      fill_n(dp[i], n, INF);
    copy_n(vcst, n, dp[0]);
    for (int msk = 1; msk <= full; ++msk) {
      if (!(msk & (msk - 1))) {
        int who = __lg(msk);
        for (int i = 0; i < n; ++i)
          dp[msk
              ][i] = vcst[ter[who]] + dst[ter[who]][i];
      }
      for (int i = 0; i < n; ++i)
        for (int sub = (
            msk - 1) & msk; sub; sub = (sub - 1) & msk)
          chmin(dp[msk][i],
              dp[sub][i] + dp[msk ^ sub][i] - vcst[i]);
      for (int i = 0; i < n; ++i) {
        tdst[i] = INF;
        for (int j = 0; j < n; ++j)
          chmin(tdst[i], dp[msk][j] + dst[j][i]);
      }
      copy_n(tdst, n, dp[msk]);
    }
    return *min_element(dp[full], dp[full] + n);
  }
}; // O(V 3^T + V^2 2^T)
```

2.9 Dominator Tree* [e95beb]

```
struct dominator_tree { // 1-base
  vector<int> G[N], rG[N];
  int n, pa[N], dfn[N], id[N], Time;
  int semi[N], idom[N], best[N];
  vector<int> tree[N]; // dominator_tree
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; ++i)
      G[i].clear(), rG[i].clear();
  }
  void add_edge(int u, int v) {
    G[u].pb(v), rG[v].pb(u);
  }
  void dfs(int u) {
    id[dfn[u] = ++Time] = u;
    for (auto v : G[u])
      if (!dfn[v]) dfs(v), pa[dfn[v]] = dfn[u];
  }
  int find(int y, int x) {
```

```
    if (y <= x) return y;
    int tmp = find(pa[y], x);
    if (semi[best[y]] > semi[best[pa[y]]])
      best[y] = best[pa[y]];
    return pa[y] = tmp;
  }
  void tarjan(int root) {
    Time = 0;
    for (int i = 1; i <= n; ++i) {
      dfn[i] = idom[i] = 0;
      tree[i].clear();
      best[i] = semi[i] = i;
    }
    dfs(root);
    for (int i = Time; i > 1; --i) {
      int u = id[i];
      for (auto v : rG[u])
        if (v = dfn[v]) {
          find(v, i);
          semi[i] = min(semi[i], semi[best[v]]);
        }
      tree[semi[i]].pb(i);
      for (auto v : tree[pa[i]]) {
        find(v, pa[i]);
        idom[v] =
          semi[best[v]] == pa[i] ? pa[i] : best[v];
      }
      tree[pa[i]].clear();
    }
    for (int i = 2; i <= Time; ++i) {
      if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
      tree[id[idom[i]]].pb(id[i]);
    }
  }
};
```

2.10 Minimum Clique Cover* [5951ca]

```
struct Clique_Cover { // 0-base, O(n2^n)
  int co[1 << N], n, E[N];
  int dp[1 << N];
  void init(int _n) {
    n = _n, fill_n(dp, 1 << n, 0);
    fill_n(E, n, 0), fill_n(co, 1 << n, 0);
  }
  void add_edge(int u, int v) {
    E[u] |= 1 << v, E[v] |= 1 << u;
  }
  int solve() {
    for (int i = 0; i < n; ++i)
      co[1 << i] = E[i] | (1 << i);
    co[0] = (1 << n) - 1;
    dp[0] = (n & 1) * 2 - 1;
    for (int i = 1; i < (1 << n); ++i) {
      int t = i & -i;
      dp[i] = -dp[i ^ t];
      co[i] = co[i ^ t] & co[t];
    }
    for (int i = 0; i < (1 << n); ++i)
      co[i] = (co[i] & i) == i;
    fwt(co, 1 << n, 1);
    for (int ans = 1; ans < n; ++ans) {
      int sum = 0; // probabilistic
      for (int i = 0; i < (1 << n); ++i)
        sum += (dp[i] *= co[i]);
      if (sum) return ans;
    }
    return n;
  }
};
```

2.11 NumberofMaximalClique* [c163d7]

```
struct BronKerbosch { // 1-base
  int n, a[N], g[N][N];
  int S, all[N][N], some[N][N], none[N][N];
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j) g[i][j] = 0;
  }
  void add_edge(int u, int v) {
    g[u][v] = g[v][u] = 1;
  }
  void dfs(int d, int an, int sn, int nn) {
    if (S > 1000) return; // pruning
    if (sn == 0 && nn == 0) ++S;
    int u = some[d][0];
    for (int i = 0; i < sn; ++i) {
      int v = some[d][i];
      if (g[u][v]) continue;
      int tsn = 0, tnn = 0;
      copy_n(all[d], an, all[d + 1]);
      all[d + 1][an] = v;
      for (int j = 0; j < sn; ++j)
        if (g[v][some[d][j]])
          some[d + 1][tsn++] = some[d][j];
      for (int j = 0; j < nn; ++j)
        if (g[v][none[d][j]])
          none[d + 1][tnn++] = none[d][j];
      dfs(d + 1, an + 1, tsn, tnn);
      some[d][i] = 0, none[d][nn++] = v;
    }
  }
  int solve() {
    iota(some[0], some[0] + n, 1);
    S = 0, dfs(0, 0, n, 0);
    return S;
  }
};
```

3 Data Structure

3.1 Discrete Trick

```
vector<int> val;
// build
sort(ALL(val)), val.resize(unique(ALL(val)) - val.begin());
// index of x
upper_bound(ALL(val), x) - val.begin();
// max idx <= x
upper_bound(ALL(val), x) - val.begin();
// max idx < x
lower_bound(ALL(val), x) - val.begin();
```

3.2 BIT kth* [7d1b5f]

```
int bit[N + 1]; // N = 2 ^ k
int query_kth(int k) {
    int res = 0;
    for (int i = N >> 1; i >= 1; i >>= 1)
        if (bit[res + i] < k)
            k -= bit[res += i];
    return res + 1;
}
```

3.3 Interval Container* [78516e]

```
/* Add and remove intervals from a set of disjoint intervals.
 * Will merge the added interval with any overlapping intervals in the set when adding.
 * Intervals are [inclusive, exclusive). */
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while (it != is.end() && it->X <= R) {
    R = max(R, it->Y);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->Y >= L) {
    L = min(L, it->X);
    R = max(R, it->Y);
    is.erase(it);
  }
  return is.insert(before, pii(L, R));
}
void removeInterval(set<pii>& is, int L, int R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->Y;
  if (it->X == L) is.erase(it);
  else (int&)it->Y = L;
  if (R != r2) is.emplace(R, r2);
}
```

3.4 Leftist Tree [bbd228]

```
struct node {
  ll v, data, sz, sum;
  node *l, *r;
  node(ll k)
    : v(0), data(k), sz(1), l(0), r(0), sum(k) {}
```

```
};
ll sz(node *p) { return p ? p->sz : 0; }
ll V(node *p) { return p ? p->v : -1; }
ll sum(node *p) { return p ? p->sum : 0; }
node *merge(node *a, node *b) {
  if (!a || !b) return a ? a : b;
  if (a->data < b->data) swap(a, b);
  a->r = merge(a->r, b);
  if (V(a->r) > V(a->l)) swap(a->r, a->l);
  a->v = V(a->r) + 1, a->sz = sz(a->l) + sz(a->r) + 1;
  a->sum = sum(a->l) + sum(a->r) + a->data;
  return a;
}
void pop(node *&o) {
  node *tmp = o;
  o = merge(o->l, o->r);
  delete tmp;
}
```

3.5 Heavy light Decomposition* [babe8a]

```
struct Heavy_light_Decomposition { // 1-base
  int n, ulink[N], deep[N], mxson[N], w[N], pa[N];
  int t, pl[N], data[N], val[N]; // val: vertex data
  vector<int> G[N];
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; ++i)
      G[i].clear(), mxson[i] = 0;
  }
  void add_edge(int a, int b) {
    G[a].pb(b), G[b].pb(a);
  }
  void dfs(int u, int f, int d) {
    w[u] = 1, pa[u] = f, deep[u] = d++;
    for (int &i : G[u])
      if (i != f) {
        dfs(i, u, d), w[u] += w[i];
        if (w[mxson[u]] < w[i]) mxson[u] = i;
      }
  }
  void cut(int u, int link) {
    data[pl[u] = ++t] = val[u], ulink[u] = link;
    if (!mxson[u]) return;
    cut(mxson[u], link);
    for (int i : G[u])
      if (i != pa[u] && i != mxson[u])
        cut(i, i);
  }
  void build() { dfs(1, 1, 1), cut(1, 1), /*build*/; }
  int query(int a, int b) {
    int ta = ulink[a], tb = ulink[b], res = 0;
    while (ta != tb) {
      if (deep
            [ta] > deep[tb]) swap(ta, tb), swap(a, b);
      // query(pl[tb], pl[b])
      tb = ulink[b = pa[tb]];
    }
    if (pl[a] > pl[b]) swap(a, b);
    // query(pl[a], pl[b])
  }
};
```

3.6 Centroid Decomposition* [4eccaf]

```
struct Cent_Dec { // 1-base
  vector<pll> G[N];
  pll info[N]; // store info. of itself
  pll upinfo[N]; // store info. of climbing up
  int n, pa[N], layer[N], sz[N], done[N];
  ll dis[__lg(N) + 1][N];
  void init(int _n) {
    n = _n, layer[0] = -1;
    fill_n(pa + 1, n, 0), fill_n(done + 1, n, 0);
    for (int i = 1; i <= n; ++i) G[i].clear();
  }
  void add_edge(int a, int b, int w) {
    G[a].pb(pll(b, w)), G[b].pb(pll(a, w));
  }
  void get_cent(
    int u, int f, int &mx, int &c, int num) {
    int mxsz = 0;
    sz[u] = 1;
    for (pll e : G[u])
      if (!done[e.X] && e.X != f) {
        get_cent(e.X, u, mx, c, num);
        sz[u] += sz[e.X], mxsz = max(mxsz, sz[e.X]);
      }
    if (mx > max(mxsz, num - sz[u]))
      mx = max(mxsz, num - sz[u]), c = u;
  }
  void dfs(int u, int f, ll d, int org) {
    // if required, add self info or climbing info
    dis[layer[org]][u] = d;
    for (pll e : G[u])
      if (!done[e.X] && e.X != f)
        dfs(e.X, u, d + e.Y, org);
  }
  int cut(int u, int f, int num) {
    int mx = 1e9, c = 0, lc;
    get_cent(u, f, mx, c, num);
    done[c] = 1, pa[c] = f, layer[c] = layer[f] + 1;
    for (pll e : G[c])
      if (!done[e.X]) {
        if (sz[e.X] > sz[c])
          lc = cut(e.X, c, num - sz[c]);
        else lc = cut(e.X, c, sz[e.X]);
        upinfo[lc] = pll(), dfs(e.X, c, e.Y, c);
      }
    return done[c] = 0, c;
  }
  void build() { cut(1, 0, n); }
  void modify(int u) {
    for (int a = u, ly = layer[a]; a;
         a = pa[a], --ly) {
      info[a].X += dis[ly][u], ++info[a].Y;
      if (pa[a])
        upinfo[a].X += dis[ly - 1][u], ++upinfo[a].Y;
    }
  }
  ll query(int u) {
    ll rt = 0;
    for (int a = u, ly = layer[a]; a;
         a = pa[a], --ly) {
      rt += info[a].X + info[a].Y * dis[ly][u];
      if (pa[a])
        rt -=
          upinfo[a].X + upinfo[a].Y * dis[ly - 1][u];
    }
    return rt;
  }
};
```

3.7 LiChaoST* [4a61ec]

```
struct L {
  ll m, k, id;
  L() : id(-1) {}
  L(ll a, ll b, ll c) : m(a), k(b), id(c) {}
  ll at(ll x) { return m * x + k; }
};
class LiChao { // maintain max
private:
  int n; vector<L> nodes;
  void insert(int l, int r, int rt, L ln) {
    int m = (l + r) >> 1;
    if (nodes[rt].id == -1)
      return nodes[rt] = ln, void();
    bool atLeft = nodes[rt].at(l) < ln.at(l);
    if (nodes[rt].at(m) < ln.at(m))
      atLeft ^= 1, swap(nodes[rt], ln);
    if (r - l == 1) return;
    if (atLeft) insert(l, m, rt << 1, ln);
    else insert(m, r, rt << 1 | 1, ln);
  }
  ll query(int l, int r, int rt, ll x) {
    int m = (l + r) >> 1; ll ret = -INF;
    if (nodes[rt].id != -1) ret = nodes[rt].at(x);
    if (r - l == 1) return ret;
    if (x
        < m) return max(ret, query(l, m, rt << 1, x));
    return max(ret, query(m, r, rt << 1 | 1, x));
  }
public:
  LiChao(int n_) : n(n_), nodes(n * 4) {}
  void insert(L ln) { insert(0, n, 1, ln); }
  ll query(ll x) { return query(0, n, 1, x); }
};
```

3.8 Link cut tree* [5f036a]

```
struct Splay { // xor-sum
  static Splay nil;
  Splay *ch[2], *f;
```

```
  int val, sum, rev, size;
  Splay (int
      _val = 0) : val(_val), sum(_val), rev(0), size(1)
  { f = ch[0] = ch[1] = &nil; }
  bool isr()
  { return f->ch[0] != this && f->ch[1] != this; }
  int dir()
  { return f->ch[0] == this ? 0 : 1; }
  void setCh(Splay *c, int d) {
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
  }
  void give_tag(int r) {
    if (r) swap(ch[0], ch[1]), rev ^= 1;
  }
  void push() {
    if (ch[0] != &nil) ch[0]->give_tag(rev);
    if (ch[1] != &nil) ch[1]->give_tag(rev);
    rev = 0;
  }
  void pull() {
    // take care of the nil!
    size = ch[0]->size + ch[1]->size + 1;
    sum = ch[0]->sum ^ ch[1]->sum ^ val;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
  }
} Splay::nil;
Splay *nil = &Splay::nil;
void rotate(Splay *x) {
  Splay *p = x->f;
  int d = x->dir();
  if (!p->isr()) p->f->setCh(x, p->dir());
  else x->f = p->f;
  p->setCh(x->ch[!d], d);
  x->setCh(p, !d);
  p->pull(), x->pull();
}
void splay(Splay *x) {
  vector<Splay*> splayVec;
  for (Splay *q = x;; q = q->f) {
    splayVec.pb(q);
    if (q->isr()) break;
  }
  reverse(ALL(splayVec));
  for (auto it : splayVec) it->push();
  while (!x->isr()) {
    if (x->f->isr()) rotate(x);
    else if (x->dir() == x->f->dir())
      rotate(x->f), rotate(x);
    else rotate(x), rotate(x);
  }
}
Splay* access(Splay *x) {
  Splay *q = nil;
  for (; x != nil; x = x->f)
    splay(x), x->setCh(q, 1), q = x;
  return q;
}
void root_path(Splay *x) { access(x), splay(x); }
void chroot(Splay *x){
  root_path(x), x->give_tag(1);
  x->push(), x->pull();
}
void split(Splay *x, Splay *y) {
  chroot(x), root_path(y);
}
void link(Splay *x, Splay *y) {
  root_path(x), chroot(y);
  x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
  split(x, y);
  if (y->size != 5) return;
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}
Splay* get_root(Splay *x) {
  for (root_path(x); x->ch[0] != nil; x = x->ch[0])
    x->push();
  splay(x);
  return x;
}
bool conn(Splay *x, Splay *y) {
  return get_root(x) == get_root(y);
}
Splay* lca(Splay *x, Splay *y) {
  access(x), root_path(y);
  if (y->f == nil) return y;
  return y->f;
}
void change(Splay *x, int val) {
  splay(x), x->val = val, x->pull();
}
int query(Splay *x, Splay *y) {
  split(x, y);
  return y->sum;
}
```

3.9 KDTree [74016d]

```
namespace kdt {
int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn],
  yl[maxn], yr[maxn];
point p[maxn];
int build(int l, int r, int dep = 0) {
  if (l == r) return -1;
  function<bool(const point &, const point &)> f =
    [dep](const point &a, const point &b) {
      if (dep & 1) return a.x < b.x;
      else return a.y < b.y;
    };
  int m = (l + r) >> 1;
  nth_element(p + l, p + m, p + r, f);
  xl[m] = xr[m] = p[m].x;
  yl[m] = yr[m] = p[m].y;
  lc[m] = build(l, m, dep + 1);
  if (~lc[m]) {
    xl[m] = min(xl[m], xl[lc[m]]);
    xr[m] = max(xr[m], xr[lc[m]]);
    yl[m] = min(yl[m], yl[lc[m]]);
    yr[m] = max(yr[m], yr[lc[m]]);
  }
  rc[m] = build(m + 1, r, dep + 1);
  if (~rc[m]) {
    xl[m] = min(xl[m], xl[rc[m]]);
    xr[m] = max(xr[m], xr[rc[m]]);
    yl[m] = min(yl[m], yl[rc[m]]);
    yr[m] = max(yr[m], yr[rc[m]]);
  }
  return m;
}
bool bound(const point &q, int o, long long d) {
  double ds = sqrt(d + 1.0);
  if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
    q.y < yl[o] - ds || q.y > yr[o] + ds)
    return false;
  return true;
}
long long dist(const point &a, const point &b) {
  return (a.x - b.x) * 1ll * (a.x - b.x) +
    (a.y - b.y) * 1ll * (a.y - b.y);
}
void dfs(
  const point &q, long long &d, int o, int dep = 0) {
  if (!bound(q, o, d)) return;
  long long cd = dist(p[o], q);
  if (cd != 0) d = min(d, cd);
  if ((dep & 1) && q.x < p[o].x ||
    !(dep & 1) && q.y < p[o].y) {
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
  } else {
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
  }
}
void init(const vector<point> &v) {
  for (int i = 0; i < v.size(); ++i) p[i] = v[i];
  root = build(0, v.size());
}
long long nearest(const point &q) {
  long long res = 1e18;
  dfs(q, res, root);
  return res;
}
} // namespace kdt
```

3.10 Treap [73589f]

```
struct node {
  int data, sz;
```

```
  node *l, *r;
  node(int k) : data(k), sz(1), l(0), r(0) {}
  void up() {
    sz = 1;
    if (l) sz += l->sz;
    if (r) sz += r->sz;
  }
  void down() {}
};
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
  if (!a || !b) return a ? a : b;
  if (rand() % (sz(a) + sz(b)) < sz(a))
    return a->down(), a->r = merge(a->r, b), a->up(),
                     a;
  return b->down(), b->l = merge(a, b->l), b->up(), b;
}
void split(node *o, node *&a, node *&b, int k) {
  if (!o) return a = b = 0, void();
  o->down();
  if (o->data <= k)
    a = o, split(o->r, a->r, b, k), a->up();
  else b = o, split(o->l, a, b->l, k), b->up();
}
void split2(node *o, node *&a, node *&b, int k) {
  if (sz(o) <= k) return a = o, b = 0, void();
  o->down();
  if (sz(o->l) + 1 <= k)
    a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
  else b = o, split2(o->l, a, b->l, k);
  o->up();
}
node *kth(node *o, int k) {
  if (k <= sz(o->l)) return kth(o->l, k);
  if (k == sz(o->l) + 1) return o;
  return kth(o->r, k - sz(o->l) - 1);
}
int Rank(node *o, int key) {
  if (!o) return 0;
  if (o->data < key)
    return sz(o->l) + 1 + Rank(o->r, key);
  else return Rank(o->l, key);
}
bool erase(node *&o, int k) {
  if (!o) return 0;
  if (o->data == k) {
    node *t = o;
    o->down(), o = merge(o->l, o->r);
    delete t;
    return 1;
  }
  node *&t = k < o->data ? o->l : o->r;
  return erase(t, k) ? o->up(), 1 : 0;
}
void insert(node *&o, int k) {
  node *a, *b;
  split(o, a, b, k),
    o = merge(a, merge(new node(k), b));
}
void interval(node *&o, int l, int r) {
  node *a, *b, *c;
  split2(o, a, b, l - 1), split2(b, b, c, r);
  // operate
  o = merge(a, merge(b, c));
}
```

4 Flow/Matching

4.1 Bipartite Matching* [f07280]

```
struct Bipartite_Matching { // 0-base
  int mp[N], mq[N], dis[N + 1], cur[N], l, r;
  vector<int> G[N + 1];
  bool dfs(int u) {
    for (int &i = cur[u]; i < SZ(G[u]); ++i) {
      int e = G[u][i];
      if (mq[e] == l
          || (dis[mq[e]] == dis[u] + 1 && dfs(mq[e])))
        return mp[mq[e] = u] = e, 1;
    }
    return dis[u] = -1, 0;
  }
  bool bfs() {
    queue<int> q;
    fill_n(dis, l + 1, -1);
    for (int i = 0; i < l; ++i)
      if (!~mp[i])
        q.push(i), dis[i] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int e : G[u])
        if (!~dis[mq[e]])
          q.push(mq[e]), dis[mq[e]] = dis[u] + 1;
    }
    return dis[l] != -1;
  }
  int matching() {
    int res = 0;
    fill_n(mp, l, -1), fill_n(mq, r, l);
    while (bfs()) {
      fill_n(cur, l, 0);
      for (int i = 0; i < l; ++i)
        res += (!~mp[i] && dfs(i));
    }
    return res; // (i, mp[i] != -1)
  }
  void add_edge(int s, int t) { G[s].pb(t); }
  void init(int _l, int _r) {
    l = _l, r = _r;
    for (int i = 0; i <= l; ++i)
      G[i].clear();
  }
};
```

4.2 Kuhn Munkres* [edf909]

```
struct KM { // 0-base, maximum matching
  ll w[N][N], hl[N], hr[N], slk[N];
  int fl[N], fr[N], pre[N], qu[N], ql, qr, n;
  bool vl[N], vr[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i)
      fill_n(w[i], n, -INF);
  }
  void add_edge(int a, int b, ll wei) {
    w[a][b] = wei;
  }
  bool Check(int x) {
    if (vl[x] = 1, ~fl[x])
      return vr[qu[qr++] = fl[x]] = 1;
    while (~x) swap(x, fr[fl[x] = pre[x]]);
    return 0;
  }
  void bfs(int s) {
    fill_n(slk
         , n, INF), fill_n(vl, n, 0), fill_n(vr, n, 0);
    ql = qr = 0, qu[qr++] = s, vr[s] = 1;
    for (ll d;;) {
      while (ql < qr)
        for (int x = 0, y = qu[ql++]; x < n; ++x)
          if (!vl[x] && slk
              [x] >= (d = hl[x] + hr[y] - w[x][y])) {
            if (pre[x] = y, d) slk[x] = d;
            else if (!Check(x)) return;
          }
      d = INF;
      for (int x = 0; x < n; ++x)
        if (!vl[x] && d > slk[x]) d = slk[x];
      for (int x = 0; x < n; ++x) {
        if (vl[x]) hl[x] += d;
        else slk[x] -= d;
        if (vr[x]) hr[x] -= d;
      }
      for (int x = 0; x < n; ++x)
        if (!vl[x] && !slk[x] && !Check(x)) return;
    }
  }
  ll solve() {
    fill_n(fl
         , n, -1), fill_n(fr, n, -1), fill_n(hr, n, 0);
    for (int i = 0; i < n; ++i)
      hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) bfs(i);
    ll res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
  }
};
```

4.3 MincostMaxflow* [47d9d2]

```
struct MinCostMaxFlow { // 0-base
```

```
  struct Edge {
    ll from, to, cap, flow, cost, rev;
  } *past[N];
  vector<Edge> G[N];
  int inq[N], n, s, t;
  ll dis[N], up[N], pot[N];
  bool BellmanFord() {
    fill_n(dis, n, INF), fill_n(inq, n, 0);
    queue<int> q;
    auto relax = [&](int u, ll d, ll cap, Edge *e) {
      if (cap > 0 && dis[u] > d) {
        dis[u] = d, up[u] = cap, past[u] = e;
        if (!inq[u]) inq[u] = 1, q.push(u);
      }
    };
    relax(s, 0, INF, 0);
    while (!q.empty()) {
      int u = q.front();
      q.pop(), inq[u] = 0;
      for (auto &e : G[u]) {
        ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
        relax
          (e.to, d2, min(up[u], e.cap - e.flow), &e);
      }
    }
    return dis[t] != INF;
  }
  void solve(int _s
    , int _t, ll &flow, ll &cost, bool neg = true) {
    s = _s, t = _t, flow = 0, cost = 0;
    if (neg) BellmanFord(), copy_n(dis, n, pot);
    for (; BellmanFord(); copy_n(dis, n, pot)) {
      for (int
          i = 0; i < n; ++i) dis[i] += pot[i] - pot[s];
      flow += up[t], cost += up[t] * dis[t];
      for (int i = t; past[i]; i = past[i]->from) {
        auto &e = *past[i];
        e.flow += up[t], G[e.to][e.rev].flow -= up[t];
      }
    }
  }
  void init(int _n) {
    n = _n, fill_n(pot, n, 0);
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void add_edge(ll a, ll b, ll cap, ll cost) {
    G[a].pb(Edge{a, b, cap, 0, cost, SZ(G[b])});
    G[b].pb(Edge{b, a, 0, 0, -cost, SZ(G[a]) - 1});
  }
};
```

4.4 Maximum Simple Graph Matching* [233755]

```
struct Matching { // 0-base
  queue<int> q; int n;
  vector<int> fa, s, vis, pre, match;
  vector<vector<int>> G;
  int Find(int u)
  { return u == fa[u] ? u : fa[u] = Find(fa[u]); }
  int LCA(int x, int y) {
    static int tk = 0; tk++; x = Find(x); y = Find(y);
    for (;; swap(x, y)) if (x != n) {
      if (vis[x] == tk) return x;
      vis[x] = tk;
      x = Find(pre[match[x]]);
    }
  }
  void Blossom(int x, int y, int l) {
    for (; Find(x) != l; x = pre[y]) {
      pre[x] = y, y = match[x];
      if (s[y] == 1) q.push(y), s[y] = 0;
      for (int z: {x, y}) if (fa[z] == z) fa[z] = l;
    }
  }
  bool Bfs(int r) {
    iota(ALL(fa), 0); fill(ALL(s), -1);
    q = queue<int>(); q.push(r); s[r] = 0;
    for (; !q.empty(); q.pop()) {
      for (int x = q.front(); int u : G[x])
        if (s[u] == -1) {
          if (pre[u] = x, s[u] = 1, match[u] == n) {
            for (int a = u, b = x, last;
                b != n; a = last, b = pre[a])
              last =
                  match[b], match[b] = a, match[a] = b;
            return true;
          }
          q.push(match[u]); s[match[u]] = 0;
        } else if (!s[u] && Find(u) != Find(x)) {
          int l = LCA(u, x);
          Blossom(x, u, l); Blossom(u, x, l);
        }
    }
    return false;
  }
  Matching(int _n) : n(_n), fa(n + 1), s(n + 1), vis
      (n + 1), pre(n + 1, n), match(n + 1, n), G(n) {}
  void add_edge(int u, int v)
  { G[u].pb(v), G[v].pb(u); }
  int solve() {
    int ans = 0;
    for (int x = 0; x < n; ++x)
      if (match[x] == n) ans += Bfs(x);
    return ans;
  } // match[x] == n means not matched
};
```

4.5 Maximum Weight Matching* [c80005]

```
#define REP(i, l, r) for (int i=(l); i<=(r); ++i)
struct WeightGraph { // 1-based
  struct edge { int u, v, w; }; int n, nx;
  vector<int> lab; vector<vector<edge>> g;
  vector<int> slk, match, st, pa, S, vis;
  vector<vector<int>> flo, flo_from; queue<int> q;
  WeightGraph(int n_) : n(n_), nx(n * 2), lab(nx + 1),
    g(nx + 1, vector<edge>(nx + 1)), slk(nx + 1),
    flo(nx + 1), flo_from(nx + 1, vector(n + 1, 0)) {
    match = st = pa = S = vis = slk;
    REP(u, 1, n) REP(v, 1, n) g[u][v] = {u, v, 0};
  }
  int E(edge e)
  { return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2; }
  void update_slk(int u, int x, int &s)
  { if (!s || E(g[u][x]) < E(g[s][x])) s = u; }
  void set_slk(int x) {
    slk[x] = 0;
    REP(u, 1, n)
      if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
        update_slk(u, x, slk[x]);
  }
  void q_push(int x) {
    if (x <= n) q.push(x);
    else for (int y : flo[x]) q_push(y);
  }
  void set_st(int x, int b) {
    st[x] = b;
    if (x > n) for (int y : flo[x]) set_st(y, b);
  }
  vector<int> split_flo(auto &f, int xr) {
    auto it = find(ALL(f), xr);
    if (auto pr = it - f.begin(); pr % 2 == 1)
      reverse(1 + ALL(f)), it = f.end() - pr;
    auto res = vector(f.begin(), it);
    return f.erase(f.begin(), it), res;
  }
  void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    int xr = flo_from[u][g[u][v].u];
    auto &f = flo[u], z = split_flo(f, xr);
    REP(i, 0, SZ(z) - 1) set_match(z[i], z[i ^ 1]);
    set_match(xr, v); f.insert(f.end(), ALL(z));
  }
  void augment(int u, int v) {
    for (;;) {
      int xnv = st[match[u]]; set_match(u, v);
      if (!xnv) return;
      set_match(v = xnv, u = st[pa[xnv]]);
    }
  }
  int lca(int u, int v) {
    static int t = 0; ++t;
    for (++t; u || v; swap(u, v)) if (u) {
      if (vis[u] == t) return u;
      vis[u] = t, u = st[match[u]];
      if (u) u = st[pa[u]];
    }
    return 0;
  }
  void add_blossom(int u, int o, int v) {
    int b = find(n + 1 + ALL(st), 0) - begin(st);
    lab[b] = 0, S[b] = 0, match[b] = match[o];
```

```
    vector<int> f = {o};
    for (int t : {u, v}) {
      reverse(1 + ALL(f));
      for (int x = t, y; x != o; x = st[pa[y]])
        f.pb(x), f.pb(y = st[match[x]]), q_push(y);
    }
    flo[b] = f; set_st(b, b);
    REP(x, 1, nx) g[b][x].w = g[x][b].w = 0;
    fill(ALL(flo_from[b]), 0);
    for (int xs : flo[b]) {
      REP(x, 1, nx)
        if (g[b][x].w == 0 || E(g[xs][x]) < E(g[b][x]))
          g[b][x] = g[xs][x], g[x][b] = g[x][xs];
      REP(x, 1, n)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slk(b);
  }
  void expand_blossom(int b) {
    for (int x : flo[b]) set_st(x, x);
    int xr = flo_from[b][g[b][pa[b]].u], xs = -1;
    for (int x : split_flo(flo[b], xr)) {
      if (xs == -1) { xs = x; continue; }
      pa[xs] = g[x][xs].u, S[xs] = 1, S[x] = 0;
      slk[xs] = 0, set_slk(x), q_push(x), xs = -1;
    }
    for (int x : flo[b])
      if (x == xr) S[x] = 1, pa[x] = pa[b];
      else S[x] = -1, set_slk(x);
    st[b] = 0;
  }
  bool on_found_edge(const edge &e) {
    if (int u = st[e.u], v = st[e.v]; S[v] == -1) {
      int nu = st[match[v]]; pa[v] = e.u; S[v] = 1;
      slk[v] = slk[nu] = S[nu] = 0; q_push(nu);
    } else if (S[v] == 0) {
      if (int o = lca(u, v)) add_blossom(u, o, v);
      else return augment(u, v), augment(v, u), true;
    }
    return false;
  }
  bool matching() {
    fill(ALL(S), -1), fill(ALL(slk), 0);
    q = queue<int>();
    REP(x, 1, nx) if (st[x] == x && !match[x])
      pa[x] = S[x] = 0, q_push(x);
    if (q.empty()) return false;
    for (;;) {
      while (SZ(q)) {
        int u = q.front(); q.pop();
        if (S[st[u]] == 1) continue;
        REP(v, 1, n)
          if (g[u][v].w > 0 && st[u] != st[v]) {
            if (E(g[u][v]) != 0)
              update_slk(u, st[v], slk[st[v]]);
            else if
                (on_found_edge(g[u][v])) return true;
          }
      }
      int d = INF;
      REP(b, n + 1, nx) if (st[b] == b && S[b] == 1)
        d = min(d, lab[b] / 2);
      REP(x, 1, nx)
        if (int
              s = slk[x]; st[x] == x && s && S[x] <= 0)
          d = min(d, E(g[s][x]) / (S[x] + 2));
      REP(u, 1, n)
        if (S[st[u]] == 1) lab[u] += d;
        else if (S[st[u]] == 0) {
          if (lab[u] <= d) return false;
          lab[u] -= d;
        }
      REP(b, n + 1, nx) if (st[b] == b && S[b] >= 0)
        lab[b] += d * (2 - 4 * S[b]);
      REP(x, 1, nx)
        if (int s = slk[x]; st[x] == x &&
            s && st[s] != x && E(g[s][x]) == 0)
          if (on_found_edge(g[s][x])) return true;
      REP(b, n + 1, nx)
        if (st[b] == b && S[b] == 1 && lab[b] == 0)
          expand_blossom(b);
    }
    return false;
  }
  pair<ll, int> solve() {
    fill(ALL(match), 0);
    REP(u, 0, n) st[u] = u, flo[u].clear();
    int w_max = 0;
    REP(u, 1, n) REP(v, 1, n) {
      flo_from[u][v] = (u == v ? u : 0);
      w_max = max(w_max, g[u][v].w);
    }
    fill(ALL(lab), w_max);
    int n_matches = 0; ll tot_weight = 0;
    while (matching()) ++n_matches;
    REP(u, 1, n) if (match[u] && match[u] < u)
      tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
  }
  void add_edge(int u, int v, int w)
  { g[u][v].w = g[v][u].w = w; }
};
```

4.6 SW-mincut [90bfe6]

```
struct SW{ // global min cut, O(V^3)
  #define REP for (int i = 0; i < n; ++i)
  static const int MXN = 514, INF = 2147483647;
  int vst[MXN], edge[MXN][MXN], wei[MXN];
  void init(int n) {
    REP fill_n(edge[i], n, 0);
  }
  void addEdge(int u, int v, int w){
    edge[u][v] += w; edge[v][u] += w;
  }
  int search(int &s, int &t, int n){
    fill_n(vst, n, 0), fill_n(wei, n, 0);
    s = t = -1;
    int mx, cur;
    for (int j = 0; j < n; ++j) {
      mx = -1, cur = 0;
      REP if (wei[i] > mx) cur = i, mx = wei[i];
      vst[cur] = 1, wei[cur] = -1;
      s = t; t = cur;
      REP if (!vst[i]) wei[i] += edge[cur][i];
    }
    return mx;
  }
  int solve(int n) {
    int res = INF;
    for (int x, y; n > 1; n--){
      res = min(res, search(x, y, n));
      REP edge[i][x] = (edge[x][i] += edge[y][i]);
      REP {
        edge[y][i] = edge[n - 1][i];
        edge[i][y] = edge[i][n - 1];
      } // edge[y][y] = 0;
    }
    return res;
  }
} sw;
```

4.7 BoundedFlow*(Dinic*) [4ae8ab]

```
struct BoundedFlow { // 0-base
  struct edge {
    int to, cap, flow, rev;
  };
  vector<edge> G[N];
  int n, s, t, dis[N], cur[N], cnt[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n + 2; ++i)
      G[i].clear(), cnt[i] = 0;
  }
  void add_edge(int u, int v, int lcap, int rcap) {
    cnt[u] -= lcap, cnt[v] += lcap;
    G[u].pb(edge{v, rcap, lcap, SZ(G[v])});
    G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
  }
  void add_edge(int u, int v, int cap) {
    G[u].pb(edge{v, cap, 0, SZ(G[v])});
    G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
  }
  int dfs(int u, int cap) {
    if (u == t || !cap) return cap;
    for (int &i = cur[u]; i < SZ(G[u]); ++i) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u] + 1 && e.cap != e.flow) {
        int df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
          e.flow += df, G[e.to][e.rev].flow -= df;
          return df;
```

```
        }
      }
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    fill_n(dis, n + 3, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (edge &e : G[u])
        if (!~dis[e.to] && e.flow != e.cap)
          q.push(e.to), dis[e.to] = dis[u] + 1;
    }
    return dis[t] != -1;
  }
  int maxflow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, df;
    while (bfs()) {
      fill_n(cur, n + 3, 0);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
  bool solve() {
    int sum = 0;
    for (int i = 0; i < n; ++i)
      if (cnt[i] > 0)
        add_edge(n + 1, i, cnt[i]), sum += cnt[i];
      else if (cnt[i] < 0) add_edge(i, n + 2, -cnt[i]);
    if (sum != maxflow(n + 1, n + 2)) sum = -1;
    for (int i = 0; i < n; ++i)
      if (cnt[i] > 0)
        G[n + 1].pop_back(), G[i].pop_back();
      else if (cnt[i] < 0)
        G[i].pop_back(), G[n + 2].pop_back();
    return sum != -1;
  }
  int solve(int _s, int _t) {
    add_edge(_t, _s, INF);
    if (!solve()) return -1; // invalid flow
    int x = G[_t].back().flow;
    return G[_t].pop_back(), G[_s].pop_back(), x;
  }
};
```

4.8 Gomory Hu tree* [5f2460]

```
MaxFlow Dinic;
int g[MAXN];
void GomoryHu(int n) { // 0-base
  fill_n(g, n, 0);
  for (int i = 1; i < n; ++i) {
    Dinic.reset();
    add_edge(i, g[i], Dinic.maxflow(i, g[i]));
    for (int j = i + 1; j <= n; ++j)
      if (g[j] == g[i] && ~Dinic.dis[j])
        g[j] = i;
  }
}
```

4.9 Minimum Cost Circulation* [cb40c6]

```
struct MinCostCirculation { // 0-base
  struct Edge {
    ll from, to, cap, fcap, flow, cost, rev;
  } *past[N];
  vector<Edge> G[N];
  ll dis[N], inq[N], n;
  void BellmanFord(int s) {
    fill_n(dis, n, INF), fill_n(inq, n, 0);
    queue<int> q;
    auto relax = [&](int u, ll d, Edge *e) {
      if (dis[u] > d) {
        dis[u] = d, past[u] = e;
        if (!inq[u]) inq[u] = 1, q.push(u);
      }
    };
    relax(s, 0, 0);
    while (!q.empty()) {
      int u = q.front();
      q.pop(), inq[u] = 0;
      for (auto &e : G[u])
        if (e.cap > e.flow)
          relax(e.to, dis[u] + e.cost, &e);
    }
  }
  void try_edge(Edge &cur) {
    if (cur.cap > cur.flow) return ++cur.cap, void();
    BellmanFord(cur.to);
    if (dis[cur.from] + cur.cost < 0) {
      ++cur.flow, --G[cur.to][cur.rev].flow;
      for (int
           i = cur.from; past[i]; i = past[i]->from) {
        auto &e = *past[i];
        ++e.flow, --G[e.to][e.rev].flow;
      }
    }
    ++cur.cap;
  }
  void solve(int mxlg) {
    for (int b = mxlg; b >= 0; --b) {
      for (int i = 0; i < n; ++i)
        for (auto &e : G[i])
          e.cap *= 2, e.flow *= 2;
      for (int i = 0; i < n; ++i)
        for (auto &e : G[i])
          if (e.fcap >> b & 1)
            try_edge(e);
    }
  }
  void init(int _n) { n = _n;
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void add_edge(ll a, ll b, ll cap, ll cost) {
    G[a].pb(Edge
        {a, b, 0, cap, 0, cost, SZ(G[b]) + (a == b)});
    G[b].pb(Edge{b, a, 0, 0, 0, -cost, SZ(G[a]) - 1});
  }
} mcmf; // O(VE * ElogC)
```

4.10 Flow Models

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x,y,l,u), connect $x \to y$ with capacity $u-l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ 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 $t \to s$ 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.
  5. 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)
  1. Redirect every edge: $y \to x$ if $(x,y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x,y,c), connect $x \to y$ with $(cost,cap) = (c,1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost,cap) = (-c,1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost,cap) = (0,d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost,cap) = (0,-d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u,v,w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e\in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u,v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.

- Project selection problem
  1. If $p_v > 0$, create edge (s,v) with capacity p_v; otherwise, create edge (v,t) with capacity $-p_v$.
  2. Create edge (u,v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.
- Dual of minimum cost maximum flow
  1. Capacity c_{uv}, Flow f_{uv}, Cost w_{uv}, Required Flow difference for vertex b_u.
  2. If all w_{uv} are integers, then optimal solution can happen when all p_u are integers.

$$\begin{aligned} \min\sum_{uv} w_{uv} f_{uv} & \\ -f_{uv} \geq -c_{uv} & \Leftrightarrow \min\sum_{u} b_u p_u + \sum_{uv} c_{uv} \max(0, p_v - p_u - w_{uv}) \\ \sum_{v} f_{vu} - \sum_{v} f_{uv} = -b_u & \qquad\qquad p_u \geq 0 \end{aligned}$$

5 String

5.1 KMP [9e1cd1]

```
int F[MAXN];
vector<int> match(string A, string B) {
  vector<int> ans;
  F[0] = -1, F[1] = 0;
  for (int i = 1, j = 0; i < SZ(B); F[++i] = ++j) {
    if (B[i] == B[j]) F[i] = F[j]; // optimize
    while (j != -1 && B[i] != B[j]) j = F[j];
  }
  for (int i = 0, j = 0; i < SZ(A); ++i) {
    while (j != -1 && A[i] != B[j]) j = F[j];
    if (++j == SZ(B)) ans.pb(i + 1 - j), j = F[j];
  }
  return ans;
}
```

5.2 Z-value* [e2dc6f]

```
int z[MAXn];
void make_z(const string &s) {
  int l = 0, r = 0;
  for (int i = 1; i < SZ(s); ++i) {
    for (z[i] = max(0, min(r - i + 1, z[i - l]));
         i + z[i] < SZ(s) && s[i + z[i]] == s[z[i]];
         ++z[i])
      ;
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
}
```

5.3 Manacher* [bfe74e]

```
int z[MAXN]; // 0-base
/* center i: radius z[i * 2 + 1] / 2
   center i, i + 1: radius z[i * 2 + 2] / 2
   both aba, abba have radius 2 */
void Manacher(string tmp) {
  string s = "%";
  int l = 0, r = 0;
  for (char c : tmp) s.pb(c), s.pb('%');
  for (int i = 0; i < SZ(s); ++i) {
    z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
    while (i - z[i] >= 0 && i + z[i] < SZ(s)
           && s[i + z[i]] == s[i - z[i]]) ++z[i];
    if (z[i] + i > r) r = z[i] + i, l = i;
  }
}
```

5.4 SAIS* [e9a275]

```
auto sais(const auto &s) {
  const int n = SZ(s), z = ranges::max(s) + 1;
  if (n == 1) return vector{0};
  vector<int> c(z); for (int x : s) ++c[x];
  partial_sum(ALL(c), begin(c));
  vector<int> sa(n); auto I = views::iota(0, n);
  vector<bool> t(n, true);
  for (int i = n - 2; i >= 0; --i)
    t[i] = (
      s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
  auto is_lms = views::filter([&t](int x) {
    return x && t[x] && !t[x - 1];
  });
  auto induce = [&] {
    for (auto x = c; int y : sa)
      if (y--) if (!t[y]) sa[x[s[y] - 1]++] = y;
    for (auto x = c; int y : sa | views::reverse)
      if (y--) if (t[y]) sa[--x[s[y]]] = y;
  };
  vector<int> lms, q(n); lms.reserve(n);
  for (auto x = c; int i : I | is_lms)
    q[i] = SZ(lms), lms.pb(sa[--x[s[i]]] = i);
  induce(); vector<int> ns(SZ(lms));
  for (int j = -1, nz = 0; int i : sa | is_lms) {
    if (j >= 0) {
      int len = min({n - i, n - j, lms[q[i] + 1] - i});
      ns[q[i]] = nz += lexicographical_compare(
          begin(s) + j, begin(s) + j + len,
          begin(s) + i, begin(s) + i + len);
    }
    j = i;
  }
  fill(ALL(sa), 0); auto nsa = sais(ns);
  for (auto x = c; int y : nsa | views::reverse)
    y = lms[y], sa[--x[s[y]]] = y;
  return induce(), sa;
}
// sa[i]: sa[i]-th suffix
     is the i-th lexicographically smallest suffix.
// hi[i]: LCP of suffix sa[i] and suffix sa[i - 1].
struct Suffix {
  int n; vector<int> sa, hi, ra;
  Suffix
      (const auto &_s, int _n) : n(_n), hi(n), ra(n) {
    vector<int> s(n + 1); // s[n] = 0;
    copy_n(_s, n, begin(s)); // _s shouldn't contain 0
    sa = sais(s); sa.erase(sa.begin());
    for (int i = 0; i < n; ++i) ra[sa[i]] = i;
    for (int i = 0, h = 0; i < n; ++i) {
      if (!ra[i]) { h = 0; continue; }
      for (int j = sa[ra[i] - 1]; max
          (i, j) + h < n && s[i + h] == s[j + h];) ++h;
      hi[ra[i]] = h ? h-- : 0;
    }
  }
};
```

5.5 Aho-Corasick Automatan* [91c6c0]

```
struct AC_Automatan {
  int nx[len][sigma], fl[len], cnt[len], ord[len], top;
  int rnx[len][sigma]; // node actually be reached
  int newnode() {
    fill_n(nx[top], sigma, -1);
    return top++;
  }
  void init() { top = 1, newnode(); }
  int input(string &s) {
    int X = 1;
    for (char c : s) {
      if (!~nx[X][c - 'A']) nx[X][c - 'A'] = newnode();
      X = nx[X][c - 'A'];
    }
    return X; // return the end node of string
  }
  void make_fl() {
    queue<int> q;
    q.push(1), fl[1] = 0;
    for (int t = 0; !q.empty(); ) {
      int R = q.front();
      q.pop(), ord[t++] = R;
      for (int i = 0; i < sigma; ++i)
        if (~nx[R][i]) {
          int X = rnx[R][i] = nx[R][i], Z = fl[R];
          for (; Z && !~nx[Z][i]; ) Z = fl[Z];
          fl[X] = Z ? nx[Z][i] : 1, q.push(X);
        }
        else rnx[R][i] = R > 1 ? rnx[fl[R]][i] : 1;
    }
  }
  void solve() {
    for (int i = top - 2; i > 0; --i)
      cnt[fl[ord[i]]] += cnt[ord[i]];
  }
} ac;
```

5.6 Smallest Rotation [e74dc0]

```
string mcp(string s) {
  int n = SZ(s), i = 0, j = 1;
  s += s;
  while (i < n && j < n) {
    int k = 0;
    while (k < n && s[i + k] == s[j + k]) ++k;
```

```
    if (s[i + k] <= s[j + k]) j += k + 1;
    else i += k + 1;
    if (i == j) ++j;
  }
  int ans = i < n ? i : j;
  return s.substr(ans, n);
}
```

5.7 De Bruijn sequence* [f601c2]

```
constexpr int MAXC = 10, MAXN = 1e5 + 10;
struct DBSeq {
  int C, N, K, L, buf[MAXC * MAXN]; // K <= C^N
  void dfs(int *out, int t, int p, int &ptr) {
    if (ptr >= L) return;
    if (t > N) {
      if (N % p) return;
      for (int i = 1; i <= p && ptr < L; ++i)
        out[ptr++] = buf[i];
    } else {
      buf[t] = buf[t - p], dfs(out, t + 1, p, ptr);
      for (int j = buf[t - p] + 1; j < C; ++j)
        buf[t] = j, dfs(out, t + 1, t, ptr);
    }
  }
  void solve(int _c, int _n, int _k, int *out) {
    int p = 0;
    C = _c, N = _n, K = _k, L = N + K - 1;
    dfs(out, 1, 1, p);
    if (p < L) fill(out + p, out + L, 0);
  }
} dbs;
```

5.8 Extended SAM* [58fa19]

```
struct exSAM {
  int len[N * 2], link[N * 2]; // maxlength, suflink
  int next[N * 2][CNUM], tot; // [0, tot), root = 0
  int lenSorted[N * 2]; // topo. order
  int cnt[N * 2]; // occurence
  int newnode() {
    fill_n(next[tot], CNUM, 0);
    len[tot] = cnt[tot] = link[tot] = 0;
    return tot++;
  }
  void init() { tot = 0, newnode(), link[0] = -1; }
  int insertSAM(int last, int c) {
    int cur = next[last][c];
    len[cur] = len[last] + 1;
    int p = link[last];
    while (p != -1 && !next[p][c])
      next[p][c] = cur, p = link[p];
    if (p == -1) return link[cur] = 0, cur;
    int q = next[p][c];
    if (len
        [p] + 1 == len[q]) return link[cur] = q, cur;
    int clone = newnode();
    for (int i = 0; i < CNUM; ++i)
      next[
          clone][i] = len[next[q][i]] ? next[q][i] : 0;
    len[clone] = len[p] + 1;
    while (p != -1 && next[p][c] == q)
      next[p][c] = clone, p = link[p];
    link[link[cur] = clone] = link[q];
    link[q] = clone;
    return cur;
  }
  void insert(const string &s) {
    int cur = 0;
    for (auto ch : s) {
      int &nxt = next[cur][int(ch - 'a')];
      if (!nxt) nxt = newnode();
      cnt[cur = nxt] += 1;
    }
  }
  void build() {
    queue<int> q;
    q.push(0);
    while (!q.empty()) {
      int cur = q.front();
      q.pop();
      for (int i = 0; i < CNUM; ++i)
        if (next[cur][i])
          q.push(insertSAM(cur, i));
    }
    vector<int> lc(tot);
    for (int i = 1; i < tot; ++i) ++lc[len[i]];
    partial_sum(ALL(lc), lc.begin());
    for (int i
         = 1; i < tot; ++i) lenSorted[--lc[len[i]]] = i;
  }
  void solve() {
    for (int i = tot - 2; i >= 0; --i)
      cnt[link[lenSorted[i]]] += cnt[lenSorted[i]];
  }
};
```

5.9 PalTree* [675736]

```
struct palindromic_tree {
  struct node {
    int next[26], fail, len;
    int cnt, num; // cnt: appear times, num: number of
                  // pal. suf.
    node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
      for (int i = 0; i < 26; ++i) next[i] = 0;
    }
  };
  vector<node> St;
  vector<char> s;
  int last, n;
  palindromic_tree() : St(2), last(1), n(0) {
    St[0].fail = 1, St[1].len = -1, s.pb(-1);
  }
  inline void clear() {
    St.clear(), s.clear(), last = 1, n = 0;
    St.pb(0), St.pb(-1);
    St[0].fail = 1, s.pb(-1);
  }
  inline int get_fail(int x) {
    while (s[n - St[x].len - 1] != s[n])
      x = St[x].fail;
    return x;
  }
  inline void add(int c) {
    s.push_back(c -= 'a'), ++n;
    int cur = get_fail(last);
    if (!St[cur].next[c]) {
      int now = SZ(St);
      St.pb(St[cur].len + 2);
      St[now].fail =
        St[get_fail(St[cur].fail)].next[c];
      St[cur].next[c] = now;
      St[now].num = St[St[now].fail].num + 1;
    }
    last = St[cur].next[c], ++St[last].cnt;
  }
  inline void count() { // counting cnt
    auto i = St.rbegin();
    for (; i != St.rend(); ++i) {
      St[i->fail].cnt += i->cnt;
    }
  }
  inline int size() { // The number of diff. pal.
    return SZ(St) - 2;
  }
};
```

5.10 Main Lorentz [eaf279]

```
vector<pair<int, int>> rep[kN]; // 0-base [l, r]
void main_lorentz(const string &s, int sft = 0) {
  const int n = s.size();
  if (n == 1) return;
  const int nu = n / 2, nv = n - nu;
  const string u = s.substr(0, nu), v = s.substr(nu),
        ru(u.rbegin
             (), u.rend()), rv(v.rbegin(), v.rend());
  main_lorentz(u, sft), main_lorentz(v, sft + nu);
  const auto z1 = Zalgo(ru), z2 = Zalgo(v + '#' + u),
             z3 = Zalgo(ru + '#' + rv), z4 = Zalgo(v);
  auto get_z = [](const vector<int> &z, int i) {
    return
         (0 <= i and i < (int)z.size()) ? z[i] : 0; };
  auto add_rep
      = [&](bool left, int c, int l, int k1, int k2) {
    const
         int L = max(1, l - k2), R = min(l - left, k1);
    if (L > R) return;
    if (left)
         rep[l].emplace_back(sft + c - R, sft + c - L);
    else rep[l].emplace_back
        (sft + c - R - l + 1, sft + c - L - l + 1);
  };
```

```
  for (int cntr = 0; cntr < n; cntr++) {
    int l, k1, k2;
    if (cntr < nu) {
      l = nu - cntr;
      k1 = get_z(z1, nu - cntr);
      k2 = get_z(z2, nv + 1 + cntr);
    } else {
      l = cntr - nu + 1;
      k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
      k2 = get_z(z4, (cntr - nu) + 1);
    }
    if (k1 + k2 >= l)
      add_rep(cntr < nu, cntr, l, k1, k2);
  }
} // p \in [l, r] => s[p, p + i) = s[p + i, p + 2i)
```

6 Math

6.1 ax+by=gcd(only exgcd *) [5fef50]

```
pll exgcd(ll a, ll b) {
  if (b == 0) return pll(1, 0);
  ll p = a / b;
  pll q = exgcd(b, a % b);
  return pll(q.Y, q.X - q.Y * p);
}
/* ax+by=res, let x be minimum non-negative
g, p = gcd(a, b), exgcd(a, b) * res / g
if p.X < 0: t = (abs(p.X) + b / g - 1) / (b / g)
else: t = -(p.X / (b / g))
p += (b / g, -a / g) * t */
```

6.2 Floor and Ceil [1ffa73]

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

6.3 Floor Enumeration [67ad61]

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

6.4 Mod Min [038fef]

```
// min{k | l <= ((ak) mod m) <= r}, no solution -> -1
ll mod_min(ll a, ll m, ll l, ll r) {
  if (a == 0) return l ? -1 : 0;
  if (ll k = (l + a - 1) / a; k * a <= r)
    return k;
  ll b = m / a, c = m % a;
  if (ll y = mod_min(c, a, a - r % a, a - l % a))
    return (l + y * c + a - 1) / a + y * b;
  return -1;
}
```

6.5 Linear Mod Inverse [aa1426]

```
inv[1] = 1;
for( int i = 2; i
     <= N; ++i ) inv[i] = ((mod-mod/i)*inv[mod%i])%mod;
```

6.6 Linear Filter Mu [663a36]

```
void getMu() {
  mu[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (!flg[i]) p[++tot] = i, mu[i] = -1;
    for (int j = 1; j <= tot && i * p[j] <= n; ++j) {
      flg[i * p[j]] = 1;
      if (i % p[j] == 0) {
        mu[i * p[j]] = 0;
        break;
      }
      mu[i * p[j]] = -mu[i];
    }
  }
}
```

6.7 Gaussian integer gcd [4fcbff]

```
cpx gaussian_gcd(cpx a, cpx b) {
#define rnd
     (a, b) ((a >= 0 ? a * 2 + b : a * 2 - b) / (b * 2))
  ll c = a.real() * b.real() + a.imag() * b.imag();
  ll d = a.imag() * b.real() - a.real() * b.imag();
  ll r = b.real() * b.real() + b.imag() * b.imag();
  if (c % r == 0 && d % r == 0) return b;
  return gaussian_gcd
       (b, a - cpx(rnd(c, r), rnd(d, r)) * b);
}
```

6.8 GaussElimination [c016c9]

```
void GAS(V<V<double>>&vc) {
    int len = vc.size();
    for( int i = 0; i < len; ++i ) {
        int idx = find_if(vc.begin()+i,vc.end(),[&](
            auto&v) {return v[i] != 0;} ) - vc.begin();
        if( idx == len ) continue;
        if( i != idx ) swap( vc[idx], vc[i] );
        double pivot = vc[i][i];
        for_each( vc[i].begin(), vc
            [i].end(), [&]( auto &a ) { a/=pivot; } );
        for( int j = 0; j < len; ++j ) {
            if( i == j ) continue;
            if( vc[j][i] != 0 ) {
                double mul = vc[j][i]/vc[i][i];
                transform( vc[j].begin(), vc[j].end
                    (), vc[i].begin(), vc[j].begin(),
                        [&](auto &a, auto &b ) {
                        return a-b*mul;
                        });
            }
        }
    }
};
```

6.9 Miller Rabin* [14b81a]

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

6.10 Simultaneous Equations [21b2e1]

```
struct matrix { //m variables, n equations
  int n, m;
  fraction M[MAXN][MAXN + 1], sol[MAXN];
  int solve() { //-1: inconsistent, >= 0: rank
    for (int i = 0; i < n; ++i) {
      int piv = 0;
      while (piv < m && !M[i][piv].n) ++piv;
      if (piv == m) continue;
      for (int j = 0; j < n; ++j) {
        if (i == j) continue;
        fraction tmp = -M[j][piv] / M[i][piv];
        for (int k = 0; k <=
             m; ++k) M[j][k] = tmp * M[i][k] + M[j][k];
      }
    }
    int rank = 0;
    for (int i = 0; i < n; ++i) {
      int piv = 0;
      while (piv < m && !M[i][piv].n) ++piv;
      if (piv == m && M[i][m].n) return -1;
      else if (piv
           < m) ++rank, sol[piv] = M[i][m] / M[i][piv];
    }
    return rank;
  }
};
```

6.11 Pollard Rho* [fff0fc]

```cpp
map<ll, int> cnt;
void PollardRho(ll n) {
  if (n == 1) return;
  if (prime(n)) return ++cnt[n], void();
  if (n % 2
      == 0) return PollardRho(n / 2), ++cnt[2], void();
  ll x = 2, y = 2, d = 1, p = 1;
  #define f(x, n, p) ((mul(x, x, n) + p) % n)
  while (true) {
    if (d != n && d != 1) {
      PollardRho(n / d);
      PollardRho(d);
      return;
    }
    if (d == n) ++p;
    x = f(x, n, p), y = f(f(y, n, p), n, p);
    d = gcd(abs(x - y), n);
  }
}
```

6.12 Simplex Algorithm [40618e]

```cpp
const int MAXN = 11000, MAXM = 405;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM];
double d[MAXN][MAXM], x[MAXM];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// value = simplex(a, b, c, N, M);
double simplex(int n, int m){
  ++m;
  fill_n(d[n], m + 1, 0);
  fill_n(d[n + 1], m + 1, 0);
  iota(ix, ix + n + m, 0);
  int r = n, s = m - 1;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
    d[i][m - 1] = 1;
    d[i][m] = b[i];
    if (d[r][m] > d[i][m]) r = i;
  }
  copy_n(c, m - 1, d[n]);
  d[n + 1][m - 1] = -1;
  for (double dd;; ) {
    if (r < n) {
      swap(ix[s], ix[r + m]);
      d[r][s] = 1.0 / d[r][s];
      for (int j = 0; j <= m; ++j)
        if (j != s) d[r][j] *= -d[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
        for (int j = 0; j <= m; ++j) if (j != s)
          d[i][j] += d[r][j] * d[i][s];
        d[i][s] *= d[r][s];
      }
    }
    r = s = -1;
    for (int j = 0; j < m; ++j)
      if (s < 0 || ix[s] > ix[j]) {
        if (d[n + 1][j] > eps ||
            (d[n + 1][j] > -eps && d[n][j] > eps))
          s = j;
      }
    if (s < 0) break;
    for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
      if (r < 0 ||
          (dd = d[r][m]
             / d[r][s] - d[i][m] / d[i][s]) < -eps ||
          (dd < eps && ix[r + m] > ix[i + m]))
        r = i;
    }
    if (r < 0) return -1; // not bounded
  }
  if (d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  fill_n(x, m, 0);
  for (int i = m; i <
       n + m; ++i) { // the missing enumerated x[i] = 0
    if (ix[i] < m - 1){
      ans += d[i - m][m] * c[ix[i]];
      x[ix[i]] = d[i-m][m];
    }
  }
  return ans;
}
```

6.12.1 Construction

Primal	**Dual**
Maximize $c^\intercal x$ s.t. $Ax \le b, x \ge 0$	Minimize $b^\intercal y$ s.t. $A^\intercal y \ge c, y \ge 0$
Maximize $c^\intercal x$ s.t. $Ax \le b$	Minimize $b^\intercal y$ s.t. $A^\intercal y = c, y \ge 0$
Maximize $c^\intercal x$ s.t. $Ax = b, x \ge 0$	Minimize $b^\intercal y$ s.t. $A^\intercal y \ge c$

$\bar{x}$ 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.

1. In case of minimization, let $c'_i = -c_i$
2. $\sum_{1\le i\le n} A_{ji}x_i \ge b_j \to \sum_{1\le i\le n} -A_{ji}x_i \le -b_j$
3. $\sum_{1\le i\le n} A_{ji}x_i = b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \le b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \ge b_j$
4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.13 chineseRemainder [fe9f25]

```cpp
ll solve(ll x1, ll m1, ll x2, ll m2) {
  ll g = gcd(m1, m2);
  if ((x2 - x1) % g) return -1; // no sol
  m1 /= g; m2 /= g;
  pll p = exgcd(m1, m2);
  ll lcm = m1 * m2 * g;
  ll res = p.first * (x2 - x1) * m1 + x1;
  // be careful with overflow
  return (res % lcm + lcm) % lcm;
}
```

6.14 Factorial without prime factor* [dcffcb]

```cpp
// O(p^k + log^2 n), pk = p^k
ll prod[MAXP];
ll fac_no_p(ll n, ll p, ll pk) {
  prod[0] = 1;
  for (int i = 1; i <= pk; ++i)
    if (i % p) prod[i] = prod[i - 1] * i % pk;
    else prod[i] = prod[i - 1];
  ll rt = 1;
  for (; n; n /= p) {
    rt = rt * mpow(prod[pk], n / pk, pk) % pk;
    rt = rt * prod[n % pk] % pk;
  }
  return rt;
} // (n! without factor p) % p^k
```

6.15 Discrete Log* [ba4ac0]

```cpp
int DiscreteLog(int s, int x, int y, int m) {
  constexpr int kStep = 32000;
  unordered_map<int, int> p;
  int b = 1;
  for (int i = 0; i < kStep; ++i) {
    p[y] = i;
    y = 1LL * y * x % m;
    b = 1LL * b * x % m;
  }
  for (int i = 0; i < m + 10; i += kStep) {
    s = 1LL * s * b % m;
    if (p.find(s) != p.end()) return i + kStep - p[s];
  }
  return -1;
}
int DiscreteLog(int x, int y, int m) {
  if (m == 1) return 0;
  int s = 1;
  for (int i = 0; i < 100; ++i) {
    if (s == y) return i;
    s = 1LL * s * x % m;
  }
  if (s == y) return 100;
  int p = 100 + DiscreteLog(s, x, y, m);
  if (fpow(x, p, m) != y) return -1;
  return p;
}
```

6.16 Berlekamp Massey [9380b8]

```cpp
template <typename T>
vector<T> BerlekampMassey(const vector<T> &output) {
  vector<T> d(SZ(output) + 1), me, he;
  for (int f = 0, i = 1; i <= SZ(output); ++i) {
    for (int j = 0; j < SZ(me); ++j)
      d[i] += output[i - j - 2] * me[j];
    if ((d[i] -= output[i - 1]) == 0) continue;
    if (me.empty()) {
```

```
        me.resize(f = i);
        continue;
      }
      vector<T> o(i - f - 1);
      T k = -d[i] / d[f]; o.pb(-k);
      for (T x : he) o.pb(x * k);
      o.resize(max(SZ(o), SZ(me)));
      for (int j = 0; j < SZ(me); ++j) o[j] += me[j];
      if (i - f + SZ(he) >= SZ(me)) he = me, f = i;
      me = o;
    }
    return me;
}
```

6.17 Primes

```
/* 12721 13331 14341 75577 123457 222557
   556679 999983 1097774749 1076767633 100102021
  999997771 1001010013 1000512343 987654361 999991231
   999888733 98789101 987777733 999991921 1010101333
   1010102101 1000000000039 1000000000000037
   2305843009213693951 4611686018427387847
   9223372036854775783 18446744073709551557 */
```

6.18 Theorem

- Cramer's rule

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

- 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 $d_{ij}=-d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.
- Cayley's Formula
  - Given a degree sequence $d_1,d_2,\ldots,d_n$ for each *labeled* vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees.
  - Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1,2,\ldots,k$ belong to different components. Then $T_{n,k}=kn^{n-k-1}$.
- Erdős–Gallai theorem
  A sequence of nonnegative integers $d_1\ge\cdots\ge d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1+\cdots+d_n$ is even and $\sum_{i-1}^{k}d_i\le k(k-1)+\sum_{i=k+1}^{n}\min(d_i,k)$ holds for every $1\le k\le n$.
- Gale–Ryser theorem
  A pair of sequences of nonnegative integers $a_1\ge\cdots\ge a_n$ and $b_1,\ldots,b_n$ is bigraphic if and only if $\sum_{i=1}^{n}a_i=\sum_{i=1}^{n}b_i$ and $\sum_{i=1}^{k}a_i\le\sum_{i=1}^{n}\min(b_i,k)$ holds for every $1\le k\le n$.
- Fulkerson–Chen–Anstee theorem
  A sequence $(a_1,b_1),\ldots,(a_n,b_n)$ of nonnegative integer pairs with $a_1\ge\cdots\ge a_n$ is digraphic if and only if $\sum_{i=1}^{n}a_i=\sum_{i=1}^{n}b_i$ and $\sum_{i=1}^{k}a_i\le\sum_{i=1}^{k}\min(b_i,k-1)+\sum_{i=k+1}^{n}\min(b_i,k)$ holds for every $1\le k\le n$.
- 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
  - $f(n)=\sum_{d|n}g(d)\Leftrightarrow g(n)=\sum_{d|n}\mu(d)f(\frac{n}{d})$
  - $f(n)=\sum_{n|d}g(d)\Leftrightarrow g(n)=\sum_{n|d}\mu(\frac{d}{n})f(d)$
- Spherical cap
  - A portion of a sphere cut off by a plane.
  - r: sphere radius, a: radius of the base of the cap, h: height of the cap, θ: $\arcsin(a/r)$.
  - Volume $=\pi h^2(3r-h)/3=\pi h(3a^2+h^2)/6=\pi r^3(2+\cos\theta)(1-\cos\theta)^2/3$.
  - Area $=2\pi rh=\pi(a^2+h^2)=2\pi r^2(1-\cos\theta)$.
- Lagrange multiplier
  - Optimize $f(x_1,\ldots,x_n)$ when k constraints $g_i(x_1,\ldots,x_n)=0$.
  - Lagrangian function $\mathcal{L}(x_1,\ldots,x_n,\lambda_1,\ldots,\lambda_k)=f(x_1,\ldots,x_n)-\sum_{i=1}^{k}\lambda_i g_i(x_1,\ldots,x_n)$.
  - 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: $\boldsymbol{v}_1=\boldsymbol{p}_1+t_1\boldsymbol{d}_1$
  - Line 2: $\boldsymbol{v}_2=\boldsymbol{p}_2+t_2\boldsymbol{d}_2$
  - $\boldsymbol{n}=\boldsymbol{d}_1\times\boldsymbol{d}_2$
  - $\boldsymbol{n}_1=\boldsymbol{d}_1\times\boldsymbol{n}$
  - $\boldsymbol{n}_2=\boldsymbol{d}_2\times\boldsymbol{n}$
  - $\boldsymbol{c}_1=\boldsymbol{p}_1+\frac{(\boldsymbol{p}_2-\boldsymbol{p}_1)\cdot\boldsymbol{n}_2}{\boldsymbol{d}_1\cdot\boldsymbol{n}_2}\boldsymbol{d}_1$
  - $\boldsymbol{c}_2=\boldsymbol{p}_2+\frac{(\boldsymbol{p}_1-\boldsymbol{p}_2)\cdot\boldsymbol{n}_1}{\boldsymbol{d}_2\cdot\boldsymbol{n}_1}\boldsymbol{d}_2$
- Derivatives/Integrals
  Integration by parts: $\int_a^b f(x)g(x)dx=[F(x)g(x)]_a^b-\int_a^b F(x)g'(x)dx$

$$\left|\begin{array}{c|c|c} \dfrac{d}{dx}\sin^{-1}x=\dfrac{1}{\sqrt{1-x^2}} & \dfrac{d}{dx}\cos^{-1}x=-\dfrac{1}{\sqrt{1-x^2}} & \dfrac{d}{dx}\tan^{-1}x=\dfrac{1}{1+x^2} \\ \dfrac{d}{dx}\tan x=1+\tan^2 x & \displaystyle\int\tan ax=-\dfrac{\ln|\cos ax|}{a} & \\ \displaystyle\int e^{-x^2}=\dfrac{\sqrt{\pi}}{2}\mathrm{erf}(x) & \displaystyle\int xe^{ax}dx=\dfrac{e^{ax}}{a^2}(ax-1) & \end{array}\right|$$

$$\int\sqrt{a^2+x^2}=\frac{1}{2}\left(x\sqrt{a^2+x^2}+a^2\operatorname{asinh}(x/a)\right)$$

- Spherical Coordinate

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

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

- Rotation Matrix

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

6.19 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

6.20 Euclidean Algorithms

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

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

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

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

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

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

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

6.21 General Purpose Numbers

- Bernoulli numbers
  $B_0-1, B_1^{\pm}=\pm\frac{1}{2}, B_2=\frac{1}{6}, B_3=0$
  $\sum_{j=0}^{m}\binom{m+1}{j}B_j=0$, EGF is $B(x)=\frac{x}{e^x-1}=\sum_{n=0}^{\infty}B_n\frac{x^n}{n!}$.
  $S_m(n)=\sum_{k=1}^{n}k^m=\frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k^+n^{m+1-k}$
- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.
  $S(n,k)=S(n-1,k-1)+kS(n-1,k), S(n,1)=S(n,n)=1$
  $S(n,k)=\frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n$
  $x^n=\sum_{i=0}^{n}S(n,i)(x)_i$
- Pentagonal number theorem
  $\prod_{n=1}^{\infty}(1-x^n)=1+\sum_{k=1}^{\infty}(-1)^k\left(x^{k(3k+1)/2}+x^{k(3k-1)/2}\right)$
- Catalan numbers
  $C_n^{(k)}=\frac{1}{(k-1)n+1}\binom{kn}{n}$
  $C^{(k)}(x)=1+x[C^{(k)}(x)]^k$
- Eulerian numbers
  Number of permutations $\pi\in S_n$ in which exactly k elements are greater than the previous element. k j:s.t. $\pi(j)>\pi(j+1), k+1$ j:s.t. $\pi(j)\geq j$, k j:s.t. $\pi(j)>j$.
  $E(n,k)=(n-k)E(n-1,k-1)+(k+1)E(n-1,k)$
  $E(n,0)=E(n,n-1)=1$
  $E(n,k)=\sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$

6.22 Tips for Generating Functions

- Ordinary Generating Function $A(x)=\sum_{i\geq 0}a_ix^i$
  - $A(rx)\Rightarrow r^na_n$
  - $A(x)+B(x)\Rightarrow a_n+b_n$
  - $A(x)B(x)\Rightarrow\sum_{i=0}^{n}a_ib_{n-i}$
  - $A(x)^k\Rightarrow\sum_{i_1+i_2+\cdots+i_k=n}a_{i_1}a_{i_2}\ldots a_{i_k}$
  - $xA(x)'\Rightarrow na_n$
  - $\frac{A(x)}{1-x}\Rightarrow\sum_{i=0}^{n}a_i$
- Exponential Generating Function $A(x)=\sum_{i\geq 0}\frac{a_i}{i!}x_i$
  - $A(x)+B(x)\Rightarrow a_n+b_n$
  - $A^{(k)}(x)\Rightarrow a_{n+k}$
  - $A(x)B(x)\Rightarrow\sum_{i=0}^{n}\binom{n}{i}a_ib_{n-i}$
  - $A(x)^k\Rightarrow\sum_{i_1+i_2+\cdots+i_k=n}\binom{n}{i_1,i_2,\ldots,i_k}a_{i_1}a_{i_2}\ldots a_{i_k}$
  - $xA(x)\Rightarrow na_n$
- Special Generating Function
  - $(1+x)^n=\sum_{i\geq 0}\binom{n}{i}x^i$
  - $\frac{1}{(1-x)^n}=\sum_{i\geq 0}\binom{i}{n-1}x^i$

7 Polynomial

7.1 Fast Fourier Transform [9fec80]

```
const int maxn = 131072;
using cplx = complex<double>;
const cplx I = cplx(0, 1);
const double pi = acos(-1);
cplx omega[maxn + 1];

void prefft() {
    for (int i = 0; i <= maxn
        ; ++i) omega[i] = exp(i * 2 * pi / maxn * I);
}

void bin(vector<cplx> &a, int n) {
    int lg;
    for (lg = 0; (1 << lg) < n; ++lg); --lg;
    vector<cplx> tmp(n);
    for (int i = 0; i < n; ++i) {
        int to = 0;
        for (int j = 0; (1 << j) <
            n; ++j) to |= (((i >> j) & 1) << (lg - j));
        tmp[to] = a[i];
    }
    for (int i = 0; i < n; ++i) a[i] = tmp[i];
}

void fft(vector<cplx> &a, int n) {
    bin(a, n);
    for (int step = 2; step <= n; step <<= 1) {
        int to = step >> 1;
        for (int i = 0; i < n; i += step) {
            for (int k = 0; k < to; ++k) {
                cplx x = a[i
                    + to + k] * omega[maxn / step * k];
                a[i + to + k] = a[i + k] - x;
                a[i + k] += x;
            }
        }
    }
}

void ifft(vector<cplx> &a, int n) {
    fft(a, n);
    reverse(a.begin() + 1, a.end());
    for (int i = 0; i < n; ++i) a[i] /= n;
}

vector<int> multiply(const vector<
    int> &a, const vector<int> &b, bool trim = false) {
    int d = 1;
    while
        (d < max(a.size(), b.size())) d <<= 1; d <<= 1;
    vector<cplx> pa(d), pb(d);
    for (int i
        = 0; i < a.size(); ++i) pa[i] = cplx(a[i], 0);
    for (int i
        = 0; i < b.size(); ++i) pb[i] = cplx(b[i], 0);
    fft(pa, d); fft(pb, d);
    for (int i = 0; i < d; ++i) pa[i] *= pb[i];
    ifft(pa, d);
    vector<int> r(d);
    for (int
        i = 0; i < d; ++i) r[i] = round(pa[i].real());
    if (trim)
        while (r.size() && r.back() == 0) r.pop_back();
    return r;
}
```

Prime	Root	Prime	Root
7681	17	167772161	3
12289	11	104857601	3
40961	3	985661441	3
65537	3	998244353	3
786433	10	1107296257	10
5767169	3	2013265921	31
7340033	3	2810183681	11
23068673	3	2885681153	3
469762049	3	605028353	3

7.2 Number Theory Transform* [e1eb36]

```
vector<int> omega;
void Init() {
  omega.resize(kN + 1);
  long long x = fpow(kRoot, (Mod - 1) / kN);
  omega[0] = 1;
  for (int i = 1; i <= kN; ++i) {
    omega[i] = 1LL * omega[i - 1] * x % kMod;
  }
}
void Transform(vector<int> &v, int n) {
  BitReverse(v, n);
  for (int s = 2; s <= n; s <<= 1) {
    int z = s >> 1;
    for (int i = 0; i < n; i += s) {
      for (int k = 0; k < z; ++k) {
        int x = 1LL
            * v[i + k + z] * omega[kN / s * k] % kMod;
        v[i + k + z] = (v[i + k] + kMod - x) % kMod;
        (v[i + k] += x) %= kMod;
      }
    }
  }
}
void InverseTransform(vector<int> &v, int n) {
  Transform(v, n);
  for (int i = 1; i < n / 2; ++i) swap(v[i], v[n - i]);
  const int kInv = fpow(n, kMod - 2);
  for (int i
      = 0; i < n; ++i) v[i] = 1LL * v[i] * inv % kMod;
}
```

7.3 Fast Walsh Transform* [36c9f5]

```
/* x: a[j], y: a[j + (L >> 1)]
or: (y += x * op), and: (x += y * op)
xor: (x, y = (x + y) * op, (x - y) * op)
invop: or, and, xor = -1, -1, 1/2 */
void fwt(int *a, int n, int op) { //or
  for (int L = 2; L <= n; L <<= 1)
    for (int i = 0; i < n; i += L)
      for (int j = i; j < i + (L >> 1); ++j)
        a[j + (L >> 1)] += a[j] * op;
```

```
}
const int N = 21;
int f[
  N][1 << N], g[N][1 << N], h[N][1 << N], ct[1 << N];
void
    subset_convolution(int *a, int *b, int *c, int L) {
  // c_k = \sum_{i | j = k, i & j = 0} a_i * b_j
  int n = 1 << L;
  for (int i = 1; i < n; ++i)
    ct[i] = ct[i & (i - 1)] + 1;
  for (int i = 0; i < n; ++i)
    f[ct[i]][i] = a[i], g[ct[i]][i] = b[i];
  for (int i = 0; i <= L; ++i)
    fwt(f[i], n, 1), fwt(g[i], n, 1);
  for (int i = 0; i <= L; ++i)
    for (int j = 0; j <= i; ++j)
      for (int x = 0; x < n; ++x)
        h[i][x] += f[j][x] * g[i - j][x];
  for (int i = 0; i <= L; ++i)
    fwt(h[i], n, -1);
  for (int i = 0; i < n; ++i)
    c[i] = h[ct[i]][i];
}
```

7.4 Polynomial Operation [37b8c7]

```
#define
     fi(s, n) for (int i = (int)(s); i < (int)(n); ++i)
template<int MAXN, ll P, ll RT> // MAXN = 2^k
struct Poly : vector<ll> { // coefficients in [0, P)
  using vector<ll>::vector;
  static NTT<MAXN, P, RT> ntt;
  int n() const { return (int)size(); } // n() >= 1
  Poly(const Poly &p, int m) : vector<ll>(m) {
    copy_n(p.data(), min(p.n(), m), data());
  }
  Poly& irev()
       { return reverse(data(), data() + n()), *this; }
  Poly& isz(int m) { return resize(m), *this; }
  Poly& iadd(const Poly &rhs) { // n() == rhs.n()
    fi(0, n()) if
         (((*this)[i] += rhs[i]) >= P) (*this)[i] -= P;
    return *this;
  }
  Poly& imul(ll k) {
    fi(0, n()) (*this)[i] = (*this)[i] * k % P;
    return *this;
  }
  Poly Mul(const Poly &rhs) const {
    int m = 1;
    while (m < n() + rhs.n() - 1) m <<= 1;
    Poly X(*this, m), Y(rhs, m);
    ntt(X.data(), m), ntt(Y.data(), m);
    fi(0, m) X[i] = X[i] * Y[i] % P;
    ntt(X.data(), m, true);
    return X.isz(n() + rhs.n() - 1);
  }
  Poly Inv() const { // (*this)[0] != 0, 1e5/95ms
    if (n() == 1) return {ntt.minv((*this)[0])};
    int m = 1;
    while (m < n() * 2) m <<= 1;
    Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(m);
    Poly Y(*this, m);
    ntt(Xi.data(), m), ntt(Y.data(), m);
    fi(0, m) {
      Xi[i] *= (2 - Xi[i] * Y[i]) % P;
      if ((Xi[i] %= P) < 0) Xi[i] += P;
    }
    ntt(Xi.data(), m, true);
    return Xi.isz(n());
  }
  Poly Sqrt()
       const { // Jacobi((*this)[0], P) = 1, 1e5/235ms
    if (n()
         == 1) return {QuadraticResidue((*this)[0], P)};
    Poly
         X = Poly(*this, (n() + 1) / 2).Sqrt().isz(n());
    return
          X.iadd(Mul(X.Inv()).isz(n())).imul(P / 2 + 1);
  }
  pair<Poly, Poly> DivMod
      (const Poly &rhs) const { // (rhs.)back() != 0
    if (n() < rhs.n()) return {{0}, *this};
    const int m = n() - rhs.n() + 1;
    Poly X(rhs); X.irev().isz(m);
    Poly Y(*this); Y.irev().isz(m);
    Poly Q = Y.Mul(X.Inv()).isz(m).irev();
    X = rhs.Mul(Q), Y = *this;
    fi(0, n()) if ((Y[i] -= X[i]) < 0) Y[i] += P;
    return {Q, Y.isz(max(1, rhs.n() - 1))};
  }
  Poly Dx() const {
    Poly ret(n() - 1);
    fi(0,
         ret.n()) ret[i] = (i + 1) * (*this)[i + 1] % P;
    return ret.isz(max(1, ret.n()));
  }
  Poly Sx() const {
    Poly ret(n() + 1);
    fi(0, n())
         ret[i + 1] = ntt.minv(i + 1) * (*this)[i] % P;
    return ret;
  }
  Poly _tmul(int nn, const Poly &rhs) const {
    Poly Y = Mul(rhs).isz(n() + nn - 1);
    return Poly(Y.data() + n() - 1, Y.data() + Y.n());
  }
  vector<ll> _eval(const
       vector<ll> &x, const vector<Poly> &up) const {
    const int m = (int)x.size();
    if (!m) return {};
    vector<Poly> down(m * 2);
    // down[1] = DivMod(up[1]).second;
    // fi(2, m *
         2) down[i] = down[i / 2].DivMod(up[i]).second;
    down[1] = Poly(up[1])
         .irev().isz(n()).Inv().irev()._tmul(m, *this);
    fi(2, m * 2) down[i]
         = up[i ^ 1]._tmul(up[i].n() - 1, down[i / 2]);
    vector<ll> y(m);
    fi(0, m) y[i] = down[m + i][0];
    return y;
  }
  static vector<Poly> _tree1(const vector<ll> &x) {
    const int m = (int)x.size();
    vector<Poly> up(m * 2);
    fi(0, m) up[m + i] = {(x[i] ? P - x[i] : 0), 1};
    for (int i = m - 1; i
         > 0; --i) up[i] = up[i * 2].Mul(up[i * 2 + 1]);
    return up;
  }
  vector
      <ll> Eval(const vector<ll> &x) const { // 1e5, 1s
    auto up = _tree1(x); return _eval(x, up);
  }
  static Poly Interpolate(const vector
      <ll> &x, const vector<ll> &y) { // 1e5, 1.4s
    const int m = (int)x.size();
    vector<Poly> up = _tree1(x), down(m * 2);
    vector<ll> z = up[1].Dx()._eval(x, up);
    fi(0, m) z[i] = y[i] * ntt.minv(z[i]) % P;
    fi(0, m) down[m + i] = {z[i]};
    for (int i = m -
          1; i > 0; --i) down[i] = down[i * 2].Mul(up[i
         * 2 + 1]).iadd(down[i * 2 + 1].Mul(up[i * 2]));
    return down[1];
  }
  Poly Ln() const { // (*this)[0] == 1, 1e5/170ms
    return Dx().Mul(Inv()).Sx().isz(n());
  }
  Poly Exp() const { // (*this)[0] == 0, 1e5/360ms
    if (n() == 1) return {1};
    Poly X = Poly(*this, (n() + 1) / 2).Exp().isz(n());
    Poly Y = X.Ln(); Y[0] = P - 1;
    fi(0, n())
         if ((Y[i] = (*this)[i] - Y[i]) < 0) Y[i] += P;
    return X.Mul(Y).isz(n());
  }
  // M := P(P - 1). If k >= M, k := k % M + M.
  Poly Pow(ll k) const {
    int nz = 0;
    while (nz < n() && !(*this)[nz]) ++nz;
    if (nz * min(k, (ll)n()) >= n()) return Poly(n());
    if (!k) return Poly(Poly {1}, n());
    Poly X(data() + nz, data() + nz + n() - nz * k);
    const ll c = ntt.mpow(X[0], k % (P - 1));
    return X.Ln().imul
         (k % P).Exp().imul(c).irev().isz(n()).irev();
  }
  static ll
      LinearRecursion(const vector<ll> &a, const vector
      <ll> &coef, ll n) { // a_n = \sum c_j a_(n-j)
    const int k = (int)a.size();
```

```cpp
    assert((int)coef.size() == k + 1);
    Poly C(k + 1), W(Poly {1}, k), M = {0, 1};
    fi(1, k + 1) C[k - i] = coef[i] ? P - coef[i] : 0;
    C[k] = 1;
    while (n) {
      if (n % 2) W = W.Mul(M).DivMod(C).second;
      n /= 2, M = M.Mul(M).DivMod(C).second;
    }
    ll ret = 0;
    fi(0, k) ret = (ret + W[i] * a[i]) % P;
    return ret;
  }
};
#undef fi
using Poly_t = Poly<131072 * 2, 998244353, 3>;
template<> decltype(Poly_t::ntt) Poly_t::ntt = {};
```

7.5 Value Polynomial [fad6e7]

```cpp
struct Poly {
  mint base; // f(x) = poly[x - base]
  vector<mint> poly;
  Poly(mint b = 0, mint x = 0): base(b), poly(1, x) {}
  mint get_val(const mint &x) {
    if (x >= base && x < base + SZ(poly))
      return poly[x - base];
    mint rt = 0;
    vector<mint> lmul(SZ(poly), 1), rmul(SZ(poly), 1);
    for (int i = 1; i < SZ(poly); ++i)
      lmul[i] = lmul[i - 1] * (x - (base + i - 1));
    for (int i = SZ(poly) - 2; i >= 0; --i)
      rmul[i] = rmul[i + 1] * (x - (base + i + 1));
    for (int i = 0; i < SZ(poly); ++i)
      rt += poly[i] * ifac[i] * inegfac
          [SZ(poly) - 1 - i] * lmul[i] * rmul[i];
    return rt;
  }
  void raise() { // g(x) = sigma{base:x} f(x)
    if (SZ(poly) == 1 && poly[0] == 0)
      return;
    mint nw = get_val(base + SZ(poly));
    poly.pb(nw);
    for (int i = 1; i < SZ(poly); ++i)
      poly[i] += poly[i - 1];
  }
};
```

7.6 Newton's Method

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

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

8 Geometry

8.1 Basic [b068f0]

```cpp
bool same
    (double a, double b) { return abs(a - b) < eps; }

struct P {
  double x, y;
  P() : x(0), y(0) {}
  P(double x, double y) : x(x), y(y) {}
  P operator + (P b) { return P(x + b.x, y + b.y); }
  P operator - (P b) { return P(x - b.x, y - b.y); }
  P operator * (double b) { return P(x * b, y * b); }
  P operator / (double b) { return P(x / b, y / b); }
  double operator * (P b) { return x * b.x + y * b.y; }
  double operator ^ (P b) { return x * b.y - y * b.x; }
  double abs() { return hypot(x, y); }
  P unit() { return *this / abs(); }
  P rot(double o) {
    double c = cos(o), s = sin(o);
    return P(c * x - s * y, s * x + c * y);
  }
  double angle() { return atan2(y, x); }
};

struct L {
  // ax + by + c = 0
  double a, b, c, o;
  P pa, pb;
  L() : a(0), b(0), c(0), o(0), pa(), pb() {}
  L(P pa, P pb) : a(pa.y - pb.y), b(pb.x - pa.x
      ), c(pa ^ pb), o(atan2(-a, b)), pa(pa), pb(pb) {}
  P project(P p) { return pa + (pb - pa).unit
      () * ((pb - pa) * (p - pa) / (pb - pa).abs()); }
  P reflect(P p) { return p + (project(p) - p) * 2; }
  double get_ratio(P p) { return (p - pa) * (
      pb - pa) / ((pb - pa).abs() * (pb - pa).abs()); }
};

bool SegmentIntersect(P p1, P p2, P p3, P p4) {
  if (max(p1.x, p2.x) < min(p3.x, p4.x) ||
      max(p3.x, p4.x) < min(p1.x, p2.x)) return false;
  if (max(p1.y, p2.y) < min(p3.y, p4.y) ||
      max(p3.y, p4.y) < min(p1.y, p2.y)) return false;
  return sign((p3 - p1) ^
      (p4 - p1)) * sign((p3 - p2) ^ (p4 - p2)) <= 0 &&
    sign((p1 - p3) ^
      (p2 - p3)) * sign((p1 - p4) ^ (p2 - p4)) <= 0;
}

bool parallel
    (L x, L y) { return same(x.a * y.b, x.b * y.a); }

P Intersect
    (L x, L y) { return P(-x.b * y.c + x.c * y.b, x
    .a * y.c - x.c * y.a) / (-x.a * y.b + x.b * y.a); }
```

8.2 KD Tree [36d550]

```cpp
namespace kdt {
int root, lc[maxn],
    rc[maxn], xl[maxn], xr[maxn], yl[maxn], yr[maxn];
point p[maxn];
int build(int l, int r, int dep = 0) {
  if (l == r) return -1;
  function<bool(const point &, const point
      &)> f = [dep](const point &a, const point &b) {
    if (dep & 1) return a.x < b.x;
    else return a.y < b.y;
  };
  int m = (l + r) >> 1;
  nth_element(p + l, p + m, p + r, f);
  xl[m] = xr[m] = p[m].x;
  yl[m] = yr[m] = p[m].y;
  lc[m] = build(l, m, dep + 1);
  if (~lc[m]) {
    xl[m] = min(xl[m], xl[lc[m]]);
    xr[m] = max(xr[m], xr[lc[m]]);
    yl[m] = min(yl[m], yl[lc[m]]);
    yr[m] = max(yr[m], yr[lc[m]]);
  }
  rc[m] = build(m + 1, r, dep + 1);
  if (~rc[m]) {
    xl[m] = min(xl[m], xl[rc[m]]);
    xr[m] = max(xr[m], xr[rc[m]]);
    yl[m] = min(yl[m], yl[rc[m]]);
    yr[m] = max(yr[m], yr[rc[m]]);
  }
  return m;
}
bool bound(const point &q, int o, long long d) {
  double ds = sqrt(d + 1.0);
  if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
      q.y <
          yl[o] - ds || q.y > yr[o] + ds) return false;
  return true;
}
long long dist(const point &a, const point &b) {
  return (a.x - b.x) * 1ll * (a.x - b.x) +
    (a.y - b.y) * 1ll * (a.y - b.y);
}
void dfs(
    const point &q, long long &d, int o, int dep = 0) {
  if (!bound(q, o, d)) return;
  long long cd = dist(p[o], q);
  if (cd != 0) d = min(d, cd);
  if ((dep & 1)
      && q.x < p[o].x || !(dep & 1) && q.y < p[o].y) {
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
  } else {
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
  }
```

```
}
void init(const vector<point> &v) {
  for (int i = 0; i < v.size(); ++i) p[i] = v[i];
  root = build(0, v.size());
}
long long nearest(const point &q) {
  long long res = 1e18;
  dfs(q, res, root);
  return res;
}}
```

8.3 Sector Area [ec8913]

```
// calc area of sector which include a, b
double SectorArea(P a, P b, double r) {
  double o = atan2(a.y, a.x) - atan2(b.y, b.x);
  while (o <= 0) o += 2 * pi;
  while (o >= 2 * pi) o -= 2 * pi;
  o = min(o, 2 * pi - o);
  return r * r * o / 2;
}
```

8.4 Half Plane Intersection [0954c1]

```
bool jizz(L l1,L l2,L l3){
  P p=Intersect(l2,l3);
  return ((l1.pb-l1.pa)^(p-l1.pa))<-eps;
}

bool cmp(const L &a,const L &b){
  return same(
      a.o,b.o)?(((b.pb-b.pa)^(a.pb-b.pa))>eps):a.o<b.o;
}

// availble area for L l is (l.pb-l.pa)^(p-l.pa)>0
vector<P> HPI(vector<L> &ls){
  sort(ls.begin(),ls.end(),cmp);
  vector<L> pls(1,ls[0]);
  for(int i=0;i<(int)ls.size();++i)if(!
      same(ls[i].o,pls.back().o))pls.push_back(ls[i]);
  deque<int> dq; dq.push_back(0); dq.push_back(1);
#define meow(a,b,c
    ) while(dq.size()>1u && jizz(pls[a],pls[b],pls[c]))
  for(int i=2;i<(int)pls.size();++i){
    meow(i,dq.back(),dq[dq.size()-2])dq.pop_back();
    meow(i,dq[0],dq[1])dq.pop_front();
    dq.push_back(i);
  }
  meow(dq
      .front(),dq.back(),dq[dq.size()-2])dq.pop_back();
  meow(dq.back(),dq[0],dq[1])dq.pop_front();
  if(dq.size()<3u)return vector
      <P>(); // no solution or solution is not a convex
  vector<P> rt;
  for(int i=0;i<(int)dq.size();++i)rt.push_back
      (Intersect(pls[dq[i]],pls[dq[(i+1)%dq.size()]]));
  return rt;
}
```

8.5 Rotating Sweep Line [b9fa8d]

```
void rotatingSweepLine(vector<pair<int,int>> &ps){
  int n=int(ps.size());
  vector<int> id(n),pos(n);
  vector<pair<int,int>> line(n*(n-1)/2);
  int m=-1;
  for(int i=0;i<n;++i)for
      (int j=i+1;j<n;++j)line[++m]=make_pair(i,j); ++m;
  sort(line.begin(),line.end(),[&](const
       pair<int,int> &a,const pair<int,int> &b)->bool{
      if(ps
          [a.first].first==ps[a.second].first)return 0;
      if(ps
          [b.first].first==ps[b.second].first)return 1;
      return (double
          )(ps[a.first].second-ps[a.second].second)/(ps
          [a.first].first-ps[a.second].first) < (double
          )(ps[b.first].second-ps[b.second].second
          )/(ps[b.first].first-ps[b.second].first);
      });
  for(int i=0;i<n;++i)id[i]=i;
  sort(id.begin(),id.end(),[&](const
       int &a,const int &b){ return ps[a]<ps[b]; });
  for(int i=0;i<n;++i)pos[id[i]]=i;

  for(int i=0;i<m;++i){
    auto l=line[i];
    // meow
    tie(pos[l.first],pos[l.second],
        id[pos[l.first]],id[pos[l.second]])=make_tuple
        (pos[l.second],pos[l.first],l.second,l.first);
  }
}
```

8.6 Triangle Center [33473a]

```
Point TriangleCircumCenter(Point a, Point b, Point c) {
  Point res;
  double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
  double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
  double ax = (a.x + b.x) / 2;
  double ay = (a.y + b.y) / 2;
  double bx = (c.x + b.x) / 2;
  double by = (c.y + b.y) / 2;
  double r1 = (sin(a2) * (ax - bx) + cos(a2) * (by
      - ay)) / (sin(a1) * cos(a2) - sin(a2) * cos(a1));
  return Point(ax + r1 * cos(a1), ay + r1 * sin(a1));
}

Point TriangleMassCenter(Point a, Point b, Point c) {
  return (a + b + c) / 3.0;
}

Point TriangleOrthoCenter(Point a, Point b, Point c) {
  return TriangleMassCenter(a, b
      , c) * 3.0 - TriangleCircumCenter(a, b, c) * 2.0;
}

Point TriangleInnerCenter(Point a, Point b, Point c) {
  Point res;
  double la = len(b - c);
  double lb = len(a - c);
  double lc = len(a - b);
  res.x = (
      la * a.x + lb * b.x + lc * c.x) / (la + lb + lc);
  res.y = (
      la * a.y + lb * b.y + lc * c.y) / (la + lb + lc);
  return res;
}
```

8.7 Polygon Center [728c3a]

```
Point BaryCenter(vector<Point> &p, int n) {
  Point res(0, 0);
  double s = 0.0, t;
  for (int i = 1; i < p.size() - 1; i++) {
    t = Cross(p[i] - p[0], p[i + 1] - p[0]) / 2;
    s += t;
    res.x += (p[0].x + p[i].x + p[i + 1].x) * t;
    res.y += (p[0].y + p[i].y + p[i + 1].y) * t;
  }
  res.x /= (3 * s);
  res.y /= (3 * s);
  return res;
}
```

8.8 Maximum Triangle [55b8cb]

```
double ConvexHullMaxTriangleArea
    (Point p[], int res[], int chnum) {
  double area = 0, tmp;
  res[chnum] = res[0];
  for (int i = 0, j = 1, k = 2; i < chnum; i++) {
    while (fabs(Cross(p[
        res[j]] - p[res[i]], p[res[(k + 1) % chnum]] -
        p[res[i]])) > fabs(Cross(p[res[j]] - p[res[i]],
         p[res[k]] - p[res[i]]))) k = (k + 1) % chnum;
    tmp = fabs(Cross(
        p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]));
    if (tmp > area) area = tmp;
    while (fabs(Cross(p[
        res[(j + 1) % chnum]] - p[res[i]], p[res[k]] -
        p[res[i]])) > fabs(Cross(p[res[j]] - p[res[i]],
         p[res[k]] - p[res[i]]))) j = (j + 1) % chnum;
    tmp = fabs(Cross(
        p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]));
    if (tmp > area) area = tmp;
  }
  return area / 2;
}
```

8.9 Point in Polygon [88cf80]

```
int pip(vector<P> ps, P p) {
  int c = 0;
```

```
  for (int i = 0; i < ps.size(); ++i) {
    int a = i, b = (i + 1) % ps.size();
    L l(ps[a], ps[b]);
    P q = l.project(p);
    if ((p - q).abs() < eps && l.inside(q)) return 1;
    if (same(ps[
        a].y, ps[b].y) && same(ps[a].y, p.y)) continue;
    if (ps[a].y > ps[b].y) swap(a, b);
    if (ps[a].y <= p.y && p.y <
         ps[b].y && p.x <= ps[a].x + (ps[b].x - ps[a].x
        ) / (ps[b].y - ps[a].y) * (p.y - ps[a].y)) ++c;
  }
  return (c & 1) * 2;
}
```

8.10 Circle [b6844a]

```
struct C {
  P c;
  double r;
  C(P c = P(0, 0), double r = 0) : c(c), r(r) {}
};

vector<P> Intersect(C a, C b) {
  if (a.r > b.r) swap(a, b);
  double d = (a.c - b.c).abs();
  vector<P> p;
  if (same(a.r + b.r,
        d)) p.push_back(a.c + (b.c - a.c).unit() * a.r);
  else if (a.r + b.r > d && d + a.r >= b.r) {
    double o = acos
        ((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d));
    P i = (b.c - a.c).unit();
    p.push_back(a.c + i.rot(o) * a.r);
    p.push_back(a.c + i.rot(-o) * a.r);
  }
  return p;
}
double IntersectArea(C a, C b) {
  if (a.r > b.r) swap(a, b);
  double d = (a.c - b.c).abs();
  if (d >= a.r + b.r - eps) return 0;
  if (d + a.r <= b.r + eps) return sq(a.r) * acos(-1);
  double p = acos
      ((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d));
  double q = acos
      ((sq(b.r) + sq(d) - sq(a.r)) / (2 * b.r * d));
  return p * sq(a.r) + q * sq(b.r) - a.r * d * sin(p);
}
// remove second
     level if to get points for line (defalut: segment)
vector<P> CircleCrossLine(P a, P b, P o, double r) {
  double x = b.x - a.x, y = b.y - a.y, A = sq(x) + sq(y
      ), B = 2 * x * (a.x - o.x) + 2 * y * (a.y - o.y);
  double C = sq(a.x - o.x
      ) + sq(a.y - o.y) - sq(r), d = B * B - 4 * A * C;
  vector<P> t;
  if (d >= -eps) {
    d = max(0., d);
    double i = (-B - sqrt(d)) / (2 * A);
    double j = (-B + sqrt(d)) / (2 * A);
    if (i - 1.0 <= eps && i >=
        -eps) t.emplace_back(a.x + i * x, a.y + i * y);
    if (j - 1.0 <= eps && j >=
        -eps) t.emplace_back(a.x + j * x, a.y + j * y);
  }
  return t;
}
// calc area
     intersect by circle with radius r and triangle OAB
double AreaOfCircleTriangle(P a, P b, double r) {
  bool ina = a.abs() < r, inb = b.abs() < r;
  auto p = CircleCrossLine(a, b, P(0, 0), r);
  if (ina) {
    if (inb) return abs(a ^ b) / 2;
    return SectorArea(b, p[0], r) + abs(a ^ p[0]) / 2;
  }
  if (inb) return
      SectorArea(p[0], a, r) + abs(p[0] ^ b) / 2;
  if (p.size() == 2u) return SectorArea(a, p[0], r)
      + SectorArea(p[1], b, r) + abs(p[0] ^ p[1]) / 2;
  else return SectorArea(a, b, r);
}
// for any triangle
double AreaOfCircleTriangle(vector<P> ps, double r) {
  double ans = 0;
  for (int i = 0; i < 3; ++i) {
    int j = (i + 1) % 3;
    double o = atan2
        (ps[i].y, ps[i].x) - atan2(ps[j].y, ps[j].x);
    if (o >= pi) o = o - 2 * pi;
    if (o <= -pi) o = o + 2 * pi;
    ans += AreaOfCircleTriangle
        (ps[i], ps[j], r) * (o >= 0 ? 1 : -1);
  }
  return abs(ans);
}
```

8.11 Tangent of Circles and Points to Circle [477789]

```
vector<L> tangent(C a, C b) {
#define Pij \
  P i = (b.c - a.c).unit() * a.r, j = P(i.y, -i.x);\
  z.emplace_back(a.c + i, a.c + i + j);
#define deo(I,J) \
  double d = (a
      .c - b.c).abs(), e = a.r I b.r, o = acos(e / d);\
  P i =
      (b.c - a.c).unit(), j = i.rot(o), k = i.rot(-o);\
  z.emplace_back(a.c + j * a.r, b.c J j * b.r);\
  z.emplace_back(a.c + k * a.r, b.c J k * b.r);
  if (a.r < b.r) swap(a, b);
  vector<L> z;
  if ((a.c - b.c).abs() + b.r < a.r) return z;
  else if (same((a.c - b.c).abs() + b.r, a.r)) { Pij; }
  else {
    deo(-,+);
    if (same(d, a.r + b.r)) { Pij; }
    else if (d > a.r + b.r) { deo(+,-); }
  }
  return z;
}

vector<L> tangent(C c, P p) {
  vector<L> z;
  double d = (p - c.c).abs();
  if (same(d, c.r)) {
    P i = (p - c.c).rot(pi / 2);
    z.emplace_back(p, p + i);
  } else if (d > c.r) {
    double o = acos(c.r / d);
    P i = (p - c.c).unit
        (), j = i.rot(o) * c.r, k = i.rot(-o) * c.r;
    z.emplace_back(c.c + j, p);
    z.emplace_back(c.c + k, p);
  }
  return z;
}
```

8.12 Area of Union of Circles [0590f1]

```
vector<pair<double, double>> CoverSegment(C &a, C &b) {
  double d = (a.c - b.c).abs();
  vector<pair<double, double>> res;
  if (same(a.r + b.r, d)) ;
  else if (d <= abs(a.r - b.r) + eps) {
    if (a.r < b.r) res.emplace_back(0, 2 * pi);
  } else if (d < abs(a.r + b.r) - eps) {
    double o = acos((sq(a.r) + sq(d) - sq(b
      .r)) / (2 * a.r * d)), z = (b.c - a.c).angle();
    if (z < 0) z += 2 * pi;
    double l = z - o, r = z + o;
    if (l < 0) l += 2 * pi;
    if (r > 2 * pi) r -= 2 * pi;
    if (l > r) res.emplace_back
        (l, 2 * pi), res.emplace_back(0, r);
    else res.emplace_back(l, r);
  }
  return res;
}
double CircleUnionArea
    (vector<C> c) { // circle should be identical
  int n = c.size();
  double a = 0, w;
  for (int i = 0; w = 0, i < n; ++i) {
    vector<pair<double, double>> s = {{2 * pi, 9}}, z;
    for (int j = 0; j < n; ++j) if (i != j) {
      z = CoverSegment(c[i], c[j]);
      for (auto &e : z) s.push_back(e);
    }
    sort(s.begin(), s.end());
    auto F = [&] (double t) { return c[i].r * (c[i].r *
        t + c[i].c.x * sin(t) - c[i].c.y * cos(t)); };
```

```
    for (auto &e : s) {
      if (e.first > w) a += F(e.first) - F(w);
      w = max(w, e.second);
    }
  }
  return a * 0.5;
}
```

8.13 Minimun Distance of 2 Polygons [e9c988]

```
// p, q is convex
double TwoConvexHullMinDist
    (Point P[], Point Q[], int n, int m) {
  int YMinP = 0, YMaxQ = 0;
  double tmp, ans = 999999999;
  for (i =
        0; i < n; ++i) if(P[i].y < P[YMinP].y) YMinP = i;
  for (i =
        0; i < m; ++i) if(Q[i].y > Q[YMaxQ].y) YMaxQ = i;
  P[n] = P[0], Q[m] = Q[0];
  for (int i = 0; i < n; ++i) {
    while (tmp = Cross(
        Q[YMaxQ + 1] - P[YMinP + 1], P[YMinP] - P[YMinP
         + 1]) > Cross(Q[YMaxQ] - P[YMinP + 1], P[YMinP
         ] - P[YMinP + 1])) YMaxQ = (YMaxQ + 1) % m;
    if (tmp < 0) ans = min(ans, PointToSegDist
        (P[YMinP], P[YMinP + 1], Q[YMaxQ]));
    else ans = min(ans, TwoSegMinDist(P[
        YMinP], P[YMinP + 1], Q[YMaxQ], Q[YMaxQ + 1]));
    YMinP = (YMinP + 1) % n;
  }
  return ans;
}
```

8.14 2D Convex Hull [d97646]

```
bool operator<(const P &a, const P &b) {
  return same(a.x, b.x) ? a.y < b.y : a.x < b.x;
}
bool operator>(const P &a, const P &b) {
  return same(a.x, b.x) ? a.y > b.y : a.x > b.x;
}

#define crx(a, b, c) ((b - a) ^ (c - a))

vector<P> convex(vector<P> ps) {
  vector<P> p;
  sort(ps.begin(), ps.end(), [&] (P a, P b) { return
       same(a.x, b.x) ? a.y < b.y : a.x < b.x; });
  for (int i = 0; i < ps.size(); ++i) {
    while (p.size() >= 2 && crx(p[p.size() -
        2], ps[i], p[p.size() - 1]) >= 0) p.pop_back();
    p.push_back(ps[i]);
  }
  int t = p.size();
  for (int i = (int)ps.size() - 2; i >= 0; --i) {
    while (p.size() > t && crx(p[p.size() -
        2], ps[i], p[p.size() - 1]) >= 0) p.pop_back();
    p.push_back(ps[i]);
  }
  p.pop_back();
  return p;
}

int sgn(double
     x) { return same(x, 0) ? 0 : x > 0 ? 1 : -1; }

P isLL(P p1, P p2, P q1, P q2) {
  double a = crx(q1, q2, p1), b = -crx(q1, q2, p2);
  return (p1 * b + p2 * a) / (a + b);
}

struct CH {
  int n;
  vector<P> p, u, d;
  CH() {}
  CH(vector<P> ps) : p(ps) {
    n = ps.size();
    rotate(p.begin
        (), min_element(p.begin(), p.end()), p.end());
    auto t = max_element(p.begin(), p.end());
    d = vector<P>(p.begin(), next(t));
    u = vector<P>(t, p.end()); u.push_back(p[0]);
  }
  int find(vector<P> &v, P d) {
    int l = 0, r = v.size();
    while (l + 5 < r) {
      int L = (l * 2 + r) / 3, R = (l + r * 2) / 3;
      if (v[L] * d > v[R] * d) r = R;
      else l = L;
    }
    int x = l;
    for (int i = l +
         1; i < r; ++i) if (v[i] * d > v[x] * d) x = i;
    return x;
  }
  int findFarest(P v) {
    if (v.y > 0 || v.y == 0 && v.x > 0) return
         ((int)d.size() - 1 + find(u, v)) % p.size();
    return find(d, v);
  }
  P get(int l, int r, P a, P b) {
    int s = sgn(crx(a, b, p[l % n]));
    while (l + 1 < r) {
      int m = (l + r) >> 1;
      if (sgn(crx(a, b, p[m % n])) == s) l = m;
      else r = m;
    }
    return isLL(a, b, p[l % n], p[(l + 1) % n]);
  }
  vector<P> getLineIntersect(P a, P b) {
    int X = findFarest((b - a).rot(pi / 2));
    int Y = findFarest((a - b).rot(pi / 2));
    if (X > Y) swap(X, Y);
    if (sgn
        (crx(a, b, p[X])) * sgn(crx(a, b, p[Y])) < 0)
         return {get(X, Y, a, b), get(Y, X + n, a, b)};
    return {}; // tangent case falls here
  }
  void update_tangent(P q, int i, int &a, int &b) {
    if (sgn(crx(q, p[a], p[i])) > 0) a = i;
    if (sgn(crx(q, p[b], p[i])) < 0) b = i;
  }
  void bs(int l, int r, P q, int &a, int &b) {
    if (l == r) return;
    update_tangent(q, l % n, a, b);
    int s = sgn(crx(q, p[l % n], p[(l + 1) % n]));
    while (l + 1 < r) {
      int m = (l + r) >> 1;
      if (sgn(crx
          (q, p[m % n], p[(m + 1) % n])) == s) l = m;
      else r = m;
    }
    update_tangent(q, r % n, a, b);
  }
  int x = l;
  for (int i = l
       + 1; i < r; ++i) if (v[i] * d > v[x] * d) x = i;
  return x;
}
int findFarest(P v) {
  if (v.y > 0 || v.y == 0 && v.x > 0) return
       ((int)d.size() - 1 + find(u, v)) % p.size();
  return find(d, v);
}
P get(int l, int r, P a, P b) {
  int s = sgn(crx(a, b, p[l % n]));
  while (l + 1 < r) {
    int m = (l + r) >> 1;
    if (sgn(crx(a, b, p[m % n])) == s) l = m;
    else r = m;
  }
  return isLL(a, b, p[l % n], p[(l + 1) % n]);
}
vector<P> getIS(P a, P b) {
  int X = findFarest((b - a).spin(pi / 2));
  int Y = findFarest((a - b).spin(pi / 2));
  if (X > Y) swap(X, Y);
  if (sgn(crx(a, b, p[X])) * sgn(crx(a, b, p[Y])) <
       0) return {get(X, Y, a, b), get(Y, X + n, a, b)};
  return {};
}
void update_tangent(P q, int i, int &a, int &b) {
  if (sgn(crx(q, p[a], p[i])) > 0) a = i;
  if (sgn(crx(q, p[b], p[i])) < 0) b = i;
}
void bs(int l, int r, P q, int &a, int &b) {
  if (l == r) return;
  update_tangent(q, l % n, a, b);
  int s = sgn(crx(q, p[l % n], p[(l + 1) % n]));
  while (l + 1 < r) {
    int m = (l + r) >> 1;
```

```
    if (sgn
          (crx(q, p[m % n], p[(m + 1) % n])) == s) l = m;
    else r = m;
  }
  update_tangent(q, r % n, a, b);
}
bool contain(P p) {
  if (p.x < d[0].x || p.x > d.back().x) return 0;
  auto it
      = lower_bound(d.begin(), d.end(), P(p.x, -1e12));
  if (it->x == p.x) {
    if (it->y > p.y) return 0;
  } else if (crx(*prev(it), *it, p) < -eps) return 0;
  it = lower_bound
      (u.begin(), u.end(), P(p.x, 1e12), greater<P>());
  if (it->x == p.x) {
    if (it->y < p.y) return 0;
  } else if (crx(*prev(it), *it, p) < -eps) return 0;
  return 1;
}
bool get_tangent(P p, int &a, int &b) { // b -> a
  if (contain(p)) return 0;
  a = b = 0;
  int i
      = lower_bound(d.begin(), d.end(), p) - d.begin();
  bs(0, i, p, a, b);
  bs(i, d.size(), p, a, b);
  i = lower_bound(
      u.begin(), u.end(), p, greater<P>()) - u.begin();
  bs((int
      )d.size() - 1, (int)d.size() - 1 + i, p, a, b);
  bs((int)d.size()
       - 1 + i, (int)d.size() - 1 + u.size(), p, a, b);
  return 1;
}
};
```

8.15 3D Convex Hull [c1ae8f]

```
double
     absvol(const P a,const P b,const P c,const P d) {
  return abs(((b-a)^(c-a))*(d-a))/6;
}
struct convex3D {
  static const int maxn=1010;
  struct T{
    int a,b,c;
    bool res;
    T(){}
    T(int a,int
         b,int c,bool res=1):a(a),b(b),c(c),res(res){}
  };
  int n,m;
  P p[maxn];
  T f[maxn*8];
  int id[maxn][maxn];
  bool on(T &t,P &q){
    return ((
        p[t.c]-p[t.b])^(p[t.a]-p[t.b]))*(q-p[t.a])>eps;
  }
  void meow(int q,int a,int b){
    int g=id[a][b];
    if(f[g].res){
      if(on(f[g],p[q]))dfs(q,g);
      else{
        id[q][b]=id[a][q]=id[b][a]=m;
        f[m++]=T(b,a,q,1);
      }
    }
  }
  void dfs(int p,int i){
    f[i].res=0;
    meow(p,f[i].b,f[i].a);
    meow(p,f[i].c,f[i].b);
    meow(p,f[i].a,f[i].c);
  }
  void operator()(){
    if(n<4)return;
    if([&](){
        for(int i=1;i<n;++i)if(abs
            (p[0]-p[i])>eps)return swap(p[1],p[i]),0;
        return 1;
        }() || [&](){
        for(int i=2;i<n;++i)if(abs((p[0]-p[i])
            ^(p[1]-p[i]))>eps)return swap(p[2],p[i]),0;
        return 1;
        }() || [&](){
        for(int i
            =3;i<n;++i)if(abs(((p[1]-p[0])^(p[2]-p[0]))
            *(p[i]-p[0]))>eps)return swap(p[3],p[i]),0;
        return 1;
        }())return;
    for(int i=0;i<4;++i){
      T t((i+1)%4,(i+2)%4,(i+3)%4,1);
      if(on(t,p[i]))swap(t.b,t.c);
      id[t.a][t.b]=id[t.b][t.c]=id[t.c][t.a]=m;
      f[m++]=t;
    }
    for(int i=4;i<n;++i)for
        (int j=0;j<m;++j)if(f[j].res && on(f[j],p[i])){
      dfs(i,j);
      break;
    }
    int mm=m; m=0;
    for(int i=0;i<mm;++i)if(f[i].res)f[m++]=f[i];
  }
  bool same(int i,int j){
    return !(absvol(p[f[i].a],p[f[i
        ].b],p[f[i].c],p[f[j].a])>eps || absvol(p[f[i].
        a],p[f[i].b],p[f[i].c],p[f[j].b])>eps || absvol
        (p[f[i].a],p[f[i].b],p[f[i].c],p[f[j].c])>eps);
  }
  int faces(){
    int r=0;
    for(int i=0;i<m;++i){
      int iden=1;
      for(int j=0;j<i;++j)if(same(i,j))iden=0;
      r+=iden;
    }
    return r;
  }
} tb;
```

8.16 Minimum Enclosing Circle [7e5b31]

```
pt center(const pt &a, const pt &b, const pt &c) {
  pt p0 = b - a, p1 = c - a;
  double c1 = norm2(p0) * 0.5, c2 = norm2(p1) * 0.5;
  double d = p0 ^ p1;
  double x = a.x + (c1 * p1.y - c2 * p0.y) / d;
  double y = a.y + (c2 * p0.x - c1 * p1.x) / d;
  return pt(x, y);
}

circle min_enclosing(vector<pt> &p) {
  random_shuffle(p.begin(), p.end());
  double r = 0.0;
  pt cent;
  for (int i = 0; i < p.size(); ++i) {
    if (norm2(cent - p[i]) <= r) continue;
    cent = p[i];
    r = 0.0;
    for (int j = 0; j < i; ++j) {
      if (norm2(cent - p[j]) <= r) continue;
      cent = (p[i] + p[j]) / 2;
      r = norm2(p[j] - cent);
      for (int k = 0; k < j; ++k) {
        if (norm2(cent - p[k]) <= r) continue;
        cent = center(p[i], p[j], p[k]);
        r = norm2(p[k] - cent);
      }
    }
  }
  return circle(cent, sqrt(r));
}
```

8.17 Closest Pair [7f292a]

```
double closest_pair(int l, int r) {
  // p should be sorted
       increasingly according to the x-coordinates.
  if (l == r) return 1e9;
  if (r - l == 1) return dist(p[l], p[r]);
  int m = (l + r) >> 1;
  double d =
       min(closest_pair(l, m), closest_pair(m + 1, r));
  vector<int> vec;
  for (int i = m; i >= l &&
       fabs(p[m].x - p[i].x) < d; --i) vec.push_back(i);
  for (int i = m + 1; i <= r &&
       fabs(p[m].x - p[i].x) < d; ++i) vec.push_back(i);
  sort(vec.begin(), vec.end()
       , [&](int a, int b) { return p[a].y < p[b].y; });
  for (int i = 0; i < vec.size(); ++i) {
```

```
    for (int j = i + 1; j < vec.size()
        && fabs(p[vec[j]].y - p[vec[i]].y) < d; ++j) {
      d = min(d, dist(p[vec[i]], p[vec[j]]));
    }
  }
  return d;
}
```

9 Else

9.1 Cyclic Ternary Search* [28a883]

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

9.2 Mo's Algorithm(With modification) [5dec12]

```
/*
Mo's Algorithm With modification
Block: N^{2/3}, Complexity: N^{5/3}
*/
struct Query {
  int L, R, LBid, RBid, T;
  Query(int l, int r, int t):
    L(l), R(r), LBid(l / blk), RBid(r / blk), T(t) {}
  bool operator<(const Query &q) const {
    if (LBid != q.LBid) return LBid < q.LBid;
    if (RBid != q.RBid) return RBid < q.RBid;
    return T < b.T;
  }
};
void solve(vector<Query> query) {
  sort(ALL(query));
  int L=0, R=0, T=-1;
  for (auto q : query) {
    while (T < q.T) addTime(L, R, ++T); // TODO
    while (T > q.T) subTime(L, R, T--); // TODO
    while (R < q.R) add(arr[++R]); // TODO
    while (L > q.L) add(arr[--L]); // TODO
    while (R > q.R) sub(arr[R--]); // TODO
    while (L < q.L) sub(arr[L++]); // TODO
    // answer query
  }
}
```

9.3 Mo's Algorithm On Tree [4a7f74]

```
/*
Mo's Algorithm On Tree
Preprocess:
1) LCA
2) dfs with in[u] = dft++, out[u] = dft++
3) ord[in[u]] = ord[out[u]] = u
4) bitset<MAXN> inset
*/
struct Query {
  int L, R, LBid, lca;
  Query(int u, int v) {
    int c = LCA(u, v);
    if (c == u || c == v)
      q.lca = -1, q.L = out[c ^ u ^ v], q.R = out[c];
    else if (out[u] < in[v])
      q.lca = c, q.L = out[u], q.R = in[v];
    else
      q.lca = c, q.L = out[v], q.R = in[u];
    q.Lid = q.L / blk;
  }
  bool operator<(const Query &q) const {
    if (LBid != q.LBid) return LBid < q.LBid;
    return R < q.R;
  }
};
void flip(int x) {
    if (inset[x]) sub(arr[x]); // TODO
    else add(arr[x]); // TODO
    inset[x] = ~inset[x];
}
void solve(vector<Query> query) {
  sort(ALL(query));
  int L = 0, R = 0;
  for (auto q : query) {
    while (R < q.R) flip(ord[++R]);
    while (L > q.L) flip(ord[--L]);
    while (R > q.R) flip(ord[R--]);
    while (L < q.L) flip(ord[L++]);
    if (~q.lca) add(arr[q.lca]);
    // answer query
    if (~q.lca) sub(arr[q.lca]);
  }
}
```

9.4 Additional Mo's Algorithm Trick

- Mo's Algorithm With Addition Only
  - Sort querys same as the normal Mo's algorithm.
  - For each query $[l,r]$:
  - If $l/blk = r/blk$, brute-force.
  - If $l/blk \neq curL/blk$, 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: $f([l,r],r+1) = f([1,r],r+1) - f([1,l),r+1)$.
  - Part1: Answer all $f([1,r],r+1)$ first.
  - Part2: Store $curR \to R$ 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.

9.5 Hilbert Curve [ed5979]

```
ll hilbert(int n, int x, int y) {
  ll res = 0;
  for (int s = n / 2; s; s >>= 1) {
    int rx = (x & s) > 0;
    int ry = (y & s) > 0;
    res += s * 1ll * s * ((3 * rx) ^ ry);
    if (ry == 0) {
      if (rx == 1) x = s - 1 - x, y = s - 1 - y;
      swap(x, y);
    }
  }
  return res;
} // n = 2^k
```

9.6 DynamicConvexTrick* [6a6f6d]

```
// only works for integer coordinates!! maintain max
struct Line {
  mutable ll a, b, p;
  bool operator
      <(const Line &rhs) const { return a < rhs.a; }
  bool operator<(ll x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
  static const ll kInf = 1e18;
  ll Div(ll a,
      ll b) { return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = kInf; return 0; }
    if (x
        ->a == y->a) x->p = x->b > y->b ? kInf : -kInf;
    else x->p = Div(y->b - x->b, x->a - y->a);
    return x->p >= y->p;
  }
  void addline(ll a, ll b) {
    auto z = insert({a, b, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin
        () && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin
        () && (--x)->p >= y->p) isect(x, erase(y));
  }
  ll query(ll x) {
    auto l = *lower_bound(x);
    return l.a * x + l.b;
  }
};
```

9.7 All LCS* [ae68f0]

```
void all_lcs(string s, string t) { // 0-base
  vector<int> h(SZ(t));
  iota(ALL(h), 0);
  for (int a = 0; a < SZ(s); ++a) {
    int v = -1;
    for (int c = 0; c < SZ(t); ++c)
      if (s[a] == t[c] || h[c] < v)
        swap(h[c], v);
    // LCS(s[0, a], t[b, c]) =
    // c - b + 1 - sum([h[i] >= b] | i <= c)
    // h[i] might become -1 !!
  }
}
```

9.8 AdaptiveSimpson* [dc2085]

```
template<typename Func, typename d = double>
struct Simpson {
  using pdd = pair<d, d>;
  Func f;
  pdd mix(pdd l, pdd r, optional<d> fm = {}) {
    d h = (r.X - l.X) / 2, v = fm.value_or(f(l.X + h));
    return {v, h / 3 * (l.Y + 4 * v + r.Y)};
  }
  d eval(pdd l, pdd r, d fm, d eps) {
    pdd m((l.X + r.X) / 2, fm);
    d s = mix(l, r, fm).second;
    auto [flm, sl] = mix(l, m);
    auto [fmr, sr] = mix(m, r);
    d delta = sl + sr - s;
    if (abs(delta
        ) <= 15 * eps) return sl + sr + delta / 15;
    return eval(l, m, flm, eps / 2) +
      eval(m, r, fmr, eps / 2);
  }
  d eval(d l, d r, d eps) {
    return eval
        ({l, f(l)}, {r, f(r)}, f((l + r) / 2), eps);
  }
  d eval2(d l, d r, d eps, int k = 997) {
    d h = (r - l) / k, s = 0;
    for (int i = 0; i < k; ++i, l += h)
      s += eval(l, l + h, eps / k);
    return s;
  }
};
template<typename Func>
Simpson<Func> make_simpson(Func f) { return {f}; }
```

9.9 Simulated Annealing [b14262]

```
double factor = 100000;
const int base = 1e9; // remember to run ~ 10 times
for (int it = 1; it <= 1000000; ++it) {
    // ans:
        answer, nw: current value, rnd(): mt19937 rnd()
    if (exp(-(nw - ans
        ) / factor) >= (double)(rnd() % base) / base)
        ans = nw;
    factor *= 0.99995;
}
```

9.10 Tree Hash* [e57357]

```
ull seed;
ull shift(ull x) {
  x ^= x << 13;
  x ^= x >> 7;
  x ^= x << 17;
  return x;
}
ull dfs(int u, int f) {
  ull sum = seed;
  for (int i : G[u])
    if (i != f)
      sum += shift(dfs(i, u));
  return sum;
}
```

9.11 Binary Search On Fraction [951597]

```
struct Q {
  ll p, q;
  Q go(Q b, ll d) { return {p + b.p*d, q + b.q*d}; }
};
bool pred(Q);
// returns smallest p/q in [lo, hi] such that
// pred(p/q) is true, and 0 <= p,q <= N
Q frac_bs(ll N) {
  Q lo{0, 1}, hi{1, 0};
  if (pred(lo)) return lo;
  assert(pred(hi));
  bool dir = 1, L = 1, H = 1;
  for (; L || H; dir = !dir) {
    ll len = 0, step = 1;
    for (int t = 0; t < 2 && (t ? step/=2 : step*=2);)
      if (Q mid = hi.go(lo, len + step);
          mid.p > N || mid.q > N || dir ^ pred(mid))
        t++;
      else len += step;
    swap(lo, hi = hi.go(lo, len));
    (dir ? L : H) = !!len;
  }
  return dir ? hi : lo;
}
```

9.12 Bitset LCS [a82d86]

```
cin >> n >> m;
for (int i = 1, x; i <= n; ++i)
  cin >> x, p[x].set(i);
for (int i = 1, x; i <= m; i++) {
  cin >> x, (g = f) |= p[x];
  f.shiftLeftByOne(), f.set(0);
  ((f = g - f) ^= g) &= g;
}
cout << f.count() << '\n';
```

10 Python

10.1 Misc

```
from decimal import *
setcontext(Context(prec
    =MAX_PREC, Emax=MAX_EMAX, rounding=ROUND_FLOOR))
print(Decimal(input()) * Decimal(input()))
from fractions import Fraction
Fraction
    ('3.14159').limit_denominator(10).numerator # 22

map(int,input().split())
# N*M
arr2d = [ [ list
    (map(int,input().split())) ] for i in range(N) ]
" ".join(str(i) for i in a)
#a^b%M
pow(a,b,M)
```