Contents

1 Basic

1.1 vimrc

```
se is nu ru et tgc sc hls cin cino+=j1 sw=2 sts=2 bs=2
    mouse=a "encoding=utf-8 ls=2
syn on | colo desert | filetype indent on
inoremap {<CR> {<CR>}<ESC>O
map <F8> <ESC>:w<CR>:!g++ "%" -o "%<" -g -std=gnu++20 -
    DCKISEKI -Wall -Wextra -Wshadow -Wfatal-errors -
    Wconversion -fsanitize=address,undefined,float-
    divide-by-zero,float-cast-overflow && echo success<
    CR>
map <F9> <ESC>:w<CR>:!g++ "%" -o "%<" -O2 -g -std=gnu
    ++20 && echo success<CR>
map <F10> <ESC>:!./"%<"<CR>
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space
    :]' \| md5sum \| cut -c-6
let c_no_curly_error=1
" setxkbmap -option caps:ctrl_modifier
```

1.2 Debug Macro [851d50]

```
#define all(x) begin(x), end(x)
#ifdef CKISEKI
#include <experimental/iterator>
#define safe cerr<<__PRETTY_FUNCTION__<<" line "<<
    __LINE__<<" safe\n"
#define debug(a...) debug_(#a, a)
#define orange(a...) orange_(#a, a)
void debug_(const char *s, auto ...a) {
 cerr << "\e[1;32m(" << s << ") = (";
 int f = 0;
 (..., (cerr << (f++ ? ", " : "") << a));
 cerr << ")\e[0m\n";
}
void orange_(const char *s, auto L, auto R) {
 cerr << "\e[1;33m[ " << s << " ] = [ ";
 using namespace experimental;
 copy(L, R, make_ostream_joiner(cerr, ", "));
 cerr << " ]\e[0m\n";
}
#else
#define safe ((void)0)
#define debug(...) safe
#define orange(...) safe
#endif
```

1.3 Increase Stack [b6856c]

```
const int size = 256 << 20;
register long rsp asm("rsp");
char *p = (char*)malloc(size)+size, *bak = (char*)rsp;
__asm__("movq %0, %%rsp\n"::"r"(p));
// main
__asm__("movq %0, %%rsp\n"::"r"(bak));
```

1.4 Pragma Optimization [6006f6]

```
#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)
```

1.5 IO Optimization [c9494b]

```
static inline int gc() {
 constexpr int B = 1<<20; static char buf[B], *p, *q;
 if (p == q) q = (p = buf) + fread(buf, 1, B, stdin);
 return q == buf ? EOF : *p++;
}
```

1.6 SVG Writer [57436c]

```
class SVG {
 void p(string_view s) { o << s; }
 void p(string_view s, auto v, auto... vs) {
  auto i = s.find('$');
  o << s.substr(0, i) << v, p(s.substr(i + 1), vs...);
 }
 ofstream o; string c = "red";
public:
 SVG(auto f,auto x1,auto y1,auto x2,auto y2) : o(f) {
  p("<svg xmlns='http://www.w3.org/2000/svg' "
   "viewBox='$ $ $ $'>\n"
   "<style>*{stroke-width:0.5%;}</style>\n",
   x1, -y2, x2 - x1, y2 - y1); }
```

```
 ~SVG() { p("</svg>\n"); }
 SVG &color(string nc) { return c = nc, *this; }
 void line(auto x1, auto y1, auto x2, auto y2) {
  p("<line x1='$' y1='$' x2='$' y2='$' stroke='$'/>\n",
   x1, -y1, x2, -y2, c); }
 void circle(auto x, auto y, auto r) {
  p("<circle cx='$' cy='$' r='$' stroke='$' "
   "fill='none'/>\n", x, -y, r, c); }
 void text(auto x, auto y, string s, int w = 12) {
  p("<text x='$' y='$' font-size='$px'>$</text>\n",
   x, -y, w, s); }
};
```

2 Data Structure

2.1 Dark Magic [095f25]

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
// heap tags: paring/binary/binomial/rc_binomial/thin
template<typename T>
using pbds_heap=__gnu_pbds::prioity_queue<T,less<T>, \
                        pairing_heap_tag>;
// pbds_heap::point_iterator
// x = pq.push(10); pq.modify(x, 87); a.join(b);
// tree tags: rb_tree_tag/ov_tree_tag/splay_tree_tag
template<typename T>
using ordered_set = tree<T, null_type, less<T>,
   rb_tree_tag, tree_order_statistics_node_update>;
// find_by_order, order_of_key
// hash tables: cc_hash_table/gp_hash_table
```

2.2 Link-Cut Tree [5e4f69]

```
template <typename Val, typename SVal> class LCT {
 struct node {
  int pa, ch[2];
  bool rev;
  Val v, prod, rprod;
  SVal sv, sub, vir;
  node() : pa{0}, ch{0, 0}, rev{false}, v{}, prod{},
    rprod{}, sv{}, sub{}, vir{} {};
 };
#define cur o[u]
#define lc cur.ch[0]
#define rc cur.ch[1]
 vector<node> o;
 bool is_root(int u) const {
  return o[cur.pa].ch[0] != u && o[cur.pa].ch[1] != u;
    }
 bool is_rch(int u) const {
  return o[cur.pa].ch[1] == u && !is_root(u); }
 void down(int u) {
  if (not cur.rev) return;
  if (lc) set_rev(lc);
  if (rc) set_rev(rc);
  cur.rev = false;
 }
 void up(int u) {
  cur.prod = o[lc].prod * cur.v * o[rc].prod;
  cur.rprod = o[rc].rprod * cur.v * o[lc].rprod;
  cur.sub = cur.vir + o[lc].sub + o[rc].sub + cur.sv;
 }
 void set_rev(int u) {
  swap(lc, rc), swap(cur.prod, cur.rprod);
  cur.rev ^= 1;
 }
 void rotate(int u) {
  int f = cur.pa, g = o[f].pa, l = is_rch(u);
  if (cur.ch[l ^ 1]) o[cur.ch[l ^ 1]].pa = f;
  if (not is_root(f)) o[g].ch[is_rch(f)] = u;
  o[f].ch[l] = cur.ch[l ^ 1];
  cur.ch[l ^ 1] = f;
  cur.pa = g, o[f].pa = u;
  up(f);
 }
 void splay(int u) {
  vector<int> stk = {u};
  while (not is_root(stk.back()))
   stk.push_back(o[stk.back()].pa);
  while (not stk.empty()) {
   down(stk.back());
   stk.pop_back();
  }
  for (int f = cur.pa; not is_root(u); f = cur.pa) {
   if (!is_root(f))
    rotate(is_rch(u) == is_rch(f) ? f : u);
   rotate(u);
  }
  up(u);
 }
 void access(int x) {
  for (int u = x, last = 0; u; u = cur.pa) {
   splay(u);
   cur.vir = cur.vir + o[rc].sub - o[last].sub;
   rc = last; up(last = u);
  }
  splay(x);
 }
 int find_root(int u) {
  int la = 0;
  for (access(u); u; u = lc) down(la = u);
  return la;
 }
 void split(int x, int y) { chroot(x); access(y); }
 void chroot(int u) { access(u); set_rev(u); }
public:
 LCT(int n = 0) : o(n + 1) {}
 int add(const Val &v = {}) {
  o.push_back(v);
  return int(o.size()) - 2;
 }
 void set_val(int u, const Val &v) {
  splay(++u); cur.v = v; up(u);
 }
 void set_sval(int u, const SVal &v) {
  splay(++u); cur.sv = v; up(u);
 }
 Val query(int x, int y) {
  split(++x, ++y); return o[y].prod;
 }
 SVal subtree(int p, int u) {
  chroot(++p); access(++u);
  return cur.vir + cur.sv;
 }
 bool connected(int u, int v) {
  return find_root(++u) == find_root(++v); }
 void link(int x, int y) {
  chroot(++x); access(++y);
  o[y].vir = o[y].vir + o[x].sub;
  up(o[x].pa = y);
 }
 void cut(int x, int y) {
  split(++x, ++y);
  o[y].ch[0] = o[x].pa = 0; up(y);
 }
#undef cur
#undef lc
#undef rc
};
```

2.3 LiChao Segment Tree [b9c827]

```
struct L {
 int m, k, id;
 L() : id(-1) {}
 L(int a, int b, int c) : m(a), k(b), id(c) {}
 int at(int x) { return m * x + k; }
};
class LiChao {
private:
 int n; vector<L> nodes;
 static int lc(int x) { return 2 * x + 1; }
 static int rc(int x) { return 2 * x + 2; }
 void insert(int l, int r, int id, L ln) {
  int m = (l + r) >> 1;
  if (nodes[id].id == -1)
   return nodes[id] = ln, void();
  bool atLeft = nodes[id].at(l) < ln.at(l);
  if (nodes[id].at(m) < ln.at(m))
   atLeft ^= 1, swap(nodes[id], ln);
  if (r - l == 1) return;
  if (atLeft) insert(l, m, lc(id), ln);
  else insert(m, r, rc(id), ln);
 }
 int query(int l, int r, int id, int x) {
  int m = (l + r) >> 1, ret = 0;
```

```
  if (nodes[id].id != -1) ret = nodes[id].at(x);
  if (r - l == 1) return ret;
  if (x < m) return max(ret, query(l, m, lc(id), x));
  return max(ret, query(m, r, rc(id), x));
 }
public:
 LiChao(int n_) : n(n_), nodes(n * 4) {}
 void insert(L ln) { insert(0, n, 0, ln); }
 int query(int x) { return query(0, n, 0, x); }
};
```

2.4 Treap [ae576c]

```
__gnu_cxx::sfmt19937 rnd(7122); // <ext/random>
namespace Treap {
struct node {
 int size, pri; node *lc, *rc, *pa;
 node() : size(1), pri(rnd()), lc(0), rc(0), pa(0) {}
 void pull() {
  size = 1; pa = 0;
  if (lc) { size += lc->size; lc->pa = this; }
  if (rc) { size += rc->size; rc->pa = this; }
 }
};
int SZ(node *x) { return x ? x->size : 0; }
node *merge(node *L, node *R) {
 if (not L or not R) return L ? L : R;
 if (L->pri > R->pri)
  return L->rc = merge(L->rc, R), L->pull(), L;
 else
  return R->lc = merge(L, R->lc), R->pull(), R;
}
void splitBySize(node *o, int k, node *&L, node *&R) {
 if (not o) L = R = 0;
 else if (int s = SZ(o->lc) + 1; s <= k)
  L=o, splitBySize(o->rc, k-s, L->rc, R), L->pull();
 else
  R=o, splitBySize(o->lc, k, L, R->lc), R->pull();
} // SZ(L) == k
int getRank(node *o) { // 1-base
 int r = SZ(o->lc) + 1;
 for (; o->pa; o = o->pa)
  if (o->pa->rc == o) r += SZ(o->pa->lc) + 1;
 return r;
}
} // namespace Treap
```

2.5 Linear Basis [138d5d]

```
template <int BITS, typename S = int> struct Basis {
 static constexpr S MIN = numeric_limits<S>::min();
 array<pair<llu, S>, BITS> b;
 Basis() { b.fill({0, MIN}); }
 void add(llu x, S p) {
  for (int i = BITS-1; i>=0; i--) if (x >> i & 1) {
   if (b[i].first == 0) return b[i]={x, p}, void();
   if (b[i].second < p)
    swap(b[i].first, x), swap(b[i].second, p);
   x ^= b[i].first;
  }
 }
 optional<llu> query_kth(llu v, llu k) {
  vector<pair<llu, int>> o;
  for (int i = 0; i < BITS; i++)
   if (b[i].first) o.emplace_back(b[i].first, i);
  if (k >= (1ULL << o.size())) return {};
  for (int i = int(o.size()) - 1; i >= 0; i--)
   if ((k >> i & 1) ^ (v >> o[i].second & 1))
    v ^= o[i].first;
  return v;
 }
 Basis filter(S l) {
  Basis res = *this;
  for (int i = 0; i < BITS; i++)
   if (res.b[i].second < l) res.b[i] = {0, MIN};
  return res;
 }
};
```

2.6 Binary Search On Segtree [6c61c0]

```
// find_first = l -> minimal x s.t. check( [l, x) )
// find_last = r -> maximal x s.t. check( [x, r) )
int find_first(int l, auto &&check) {
 if (l >= n) return n + 1;
 l += sz; push(l); Monoid sum; // identity
 do {
  while ((l & 1) == 0) l >>= 1;
  if (auto s = sum + nd[l]; check(s)) {
   while (l < sz) {
    prop(l); l = (l << 1);
    if (auto nxt = sum + nd[l]; not check(nxt))
     sum = nxt, l++;
   }
   return l + 1 - sz;
  } else sum = s, l++;
 } while (lowbit(l) != l);
 return n + 1;
}
int find_last(int r, auto &&check) {
 if (r <= 0) return -1;
 r += sz; push(r - 1); Monoid sum; // identity
 do {
  r--;
  while (r > 1 and (r & 1)) r >>= 1;
  if (auto s = nd[r] + sum; check(s)) {
   while (r < sz) {
    prop(r); r = (r << 1) | 1;
    if (auto nxt = nd[r] + sum; not check(nxt))
     sum = nxt, r--;
   }
   return r - sz;
  } else sum = s;
 } while (lowbit(r) != r);
 return -1;
}
```

3 Graph

3.1 2-SAT (SCC) [09167a]

```
class TwoSat { // test @ CSES Giant Pizza
private:
 int n; vector<vector<int>> G, rG, sccs;
 vector<int> ord, idx, vis, res;
 void dfs(int u) {
  vis[u] = true;
  for (int v : G[u]) if (!vis[v]) dfs(v);
  ord.push_back(u);
 }
 void rdfs(int u) {
  vis[u] = false; idx[u] = sccs.size() - 1;
  sccs.back().push_back(u);
  for (int v : rG[u]) if (vis[v]) rdfs(v);
 }
public:
 TwoSat(int n_) : n(n_), G(n), rG(n), idx(n), vis(n),
    res(n) {}
 void add_edge(int u, int v) {
  G[u].push_back(v); rG[v].push_back(u);
 }
 void orr(int x, int y) {
  if ((x ^ y) == 1) return;
  add_edge(x ^ 1, y); add_edge(y ^ 1, x);
 }
 bool solve() {
  for (int i = 0; i < n; ++i) if (not vis[i]) dfs(i);
  for (int u : ord | views::reverse)
   if (vis[u]) sccs.emplace_back(), rdfs(u);
  for (int i = 0; i < n; i += 2)
   if (idx[i] == idx[i + 1]) return false;
  vector<bool> c(sccs.size());
  for (size_t i = 0; i < sccs.size(); ++i)
   for (int z : sccs[i])
    res[z] = c[i], c[idx[z ^ 1]] = !c[i];
  return true;
 }
 bool get(int x) { return res[x]; }
 int get_id(int x) { return idx[x]; }
 int count() { return sccs.size(); }
};
```

3.2 BCC [6ac6db]

```
class BCC {
 int n, ecnt, bcnt;
 vector<vector<pair<int, int>>> g;
 vector<int> dfn, low, bcc, stk;
 vector<bool> ap, bridge;
 void dfs(int u, int f) {
```

```
  dfn[u] = low[u] = dfn[f] + 1;
  int ch = 0;
  for (auto [v, t] : g[u]) if (bcc[t] == -1) {
   bcc[t] = 0; stk.push_back(t);
   if (dfn[v]) {
    low[u] = min(low[u], dfn[v]);
    continue;
   }
   ++ch, dfs(v, u);
   low[u] = min(low[u], low[v]);
   if (low[v] > dfn[u]) bridge[t] = true;
   if (low[v] < dfn[u]) continue;
   ap[u] = true;
   while (not stk.empty()) {
    int o = stk.back(); stk.pop_back();
    bcc[o] = bcnt;
    if (o == t) break;
   }
   bcnt += 1;
  }
  ap[u] = ap[u] and (ch != 1 or u != f);
 }
public:
 BCC(int n_) : n(n_), ecnt(0), bcnt(0), g(n), dfn(n),
    low(n), stk(), ap(n) {}
 void add_edge(int u, int v) {
  g[u].emplace_back(v, ecnt);
  g[v].emplace_back(u, ecnt++);
 }
 void solve() {
  bridge.assign(ecnt, false); bcc.assign(ecnt, -1);
  for (int i = 0; i < n; ++i) if (!dfn[i]) dfs(i, i);
 }
 int bcc_id(int x) const { return bcc[x]; }
 bool is_ap(int x) const { return ap[x]; }
 bool is_bridge(int x) const { return bridge[x]; }
};
```

3.3 Round Square Tree [cf6d74]

```
struct RST { // be careful about isolate point
 int n; vector<vector<int>> T;
 RST(auto &G) : n(int(G.size())), T(n) {
  vector<int> stk, vis(n), low(n);
  auto dfs = [&](auto self, int u, int d) -> void {
   low[u] = vis[u] = d; stk.push_back(u);
   for (int v : G[u]) if (!vis[v]) {
    self(self, v, d + 1);
    if (low[v] == vis[u]) {
     int cnt = int(T.size()); T.emplace_back();
     for (int x = -1; x != v; stk.pop_back())
      T[cnt].push_back(x = stk.back());
     T[u].push_back(cnt); // T is rooted
    } else low[u] = min(low[u], low[v]);
   } else low[u] = min(low[u], vis[v]);
  };
  for (int u = 0; u < n; u++)
   if (!vis[u]) dfs(dfs, u, 1);
 } // T may be forest; after dfs, stk are the roots
}; // test @ 2020 Shanghai K
```

3.4 Edge TCC [5a2668]

```
vector<vector<int>> ETCC(auto &adj) {
 const int n = static_cast<int>(adj.size());
 vector<int> up(n), low(n), in, out, nx, id;
 in = out = nx = id = vector<int>(n, -1);
 int dfc = 0, cnt = 0; Dsu dsu(n);
 auto merge = [&](int u, int v) {
  dsu.join(u, v); up[u] += up[v]; };
 auto dfs = [&](auto self, int u, int p) -> void {
  in[u] = low[u] = dfc++;
  for (int v : adj[u]) if (v != u) {
   if (v == p) { p = -1; continue; }
   if (in[v] == -1) {
    self(self, v, u);
    if (nx[v] == -1 && up[v] <= 1) {
     up[u] += up[v]; low[u] = min(low[u], low[v]);
     continue;
    }
    if (up[v] == 0) v = nx[v];
    if (low[u] > low[v])
     low[u] = low[v], swap(nx[u], v);
    for (; v != -1; v = nx[v]) merge(u, v);
   } else if (in[v] < in[u]) {
    low[u] = min(low[u], in[v]); up[u]++;
   } else {
    for (int &x = nx[u]; x != -1 &&
      in[x] <= in[v] && in[v] < out[x]; x = nx[x])
     merge(u, x);
    up[u]--;
   }
  }
  out[u] = dfc;
 };
 for (int i = 0; i < n; i++)
  if (in[i] == -1) dfs(dfs, i, -1);
 for (int i = 0; i < n; i++)
  if (dsu.anc(i) == i) id[i] = cnt++;
 vector<vector<int>> comps(cnt);
 for (int i = 0; i < n; i++)
  comps[id[dsu.anc(i)]].push_back(i);
 return comps;
} // test @ yosupo judge
```

3.5 DMST [f4317e]

```
using lld = int64_t;
struct E { int s, t; lld w; }; // 0-base
struct PQ {
 struct P {
  lld v; int i;
  bool operator>(const P &b) const { return v > b.v; }
 };
 min_heap<P> pq; lld tag;
 void push(P p) { p.v -= tag; pq.emplace(p); }
 P top() { P p = pq.top(); p.v += tag; return p; }
 void join(PQ &b) {
  if (pq.size() < b.pq.size())
   swap(pq, b.pq), swap(tag, b.tag);
  while (!b.pq.empty()) push(b.top()), b.pq.pop();
 }
};
vector<int> dmst(const vector<E> &e, int n, int root) {
 vector<PQ> h(n * 2);
 for (int i = 0; i < int(e.size()); ++i)
  h[e[i].t].push({e[i].w, i});
 vector<int> a(n * 2); iota(all(a), 0);
 vector<int> v(n * 2, -1), pa(n * 2, -1), r(n * 2);
 auto o = [&](auto Y, int x) -> int {
  return x==a[x] ? x : a[x] = Y(Y, a[x]); };
 auto S = [&](int i) { return o(o, e[i].s); };
 int pc = v[root] = n;
 for (int i = 0; i < n; ++i) if (v[i] == -1)
  for (int p = i; v[p]<0 || v[p]==i; p = S(r[p])) {
   if (v[p] == i)
    for (int q = pc++; p != q; p = S(r[p])) {
     h[p].tag -= h[p].top().v; h[q].join(h[p]);
     pa[p] = a[p] = q;
    }
   while (S(h[p].top().i) == p) h[p].pq.pop();
   v[p] = i; r[p] = h[p].top().i;
  }
 vector<int> ans;
 for (int i = pc - 1; i >= 0; i--) if (v[i] != n) {
  for (int f = e[r[i]].t; f != -1 && v[f] != n; f = pa[
    f])
   v[f] = n;
  ans.push_back(r[i]);
 }
 return ans; // default minimize, returns edgeid array
}
```

3.6 Dominator Tree [ea5b7c]

```
struct Dominator {
 vector<vector<int>> g, r, rdom; int tk;
 vector<int> dfn, rev, fa, sdom, dom, val, rp;
 Dominator(int n) : g(n), r(n), rdom(n), tk(0) {
  dfn = rev = fa = sdom = dom =
   val = rp = vector<int>(n, -1); }
 void add_edge(int x, int y) { g[x].push_back(y); }
 void dfs(int x) {
  rev[dfn[x] = tk] = x;
  fa[tk] = sdom[tk] = val[tk] = tk; tk++;
  for (int u : g[x]) {
   if (dfn[u] == -1) dfs(u), rp[dfn[u]] = dfn[x];
   r[dfn[u]].push_back(dfn[x]);
```

```
  }
 }
 void merge(int x, int y) { fa[x] = y; }
 int find(int x, int c = 0) {
  if (fa[x] == x) return c ? -1 : x;
  if (int p = find(fa[x], 1); p != -1) {
   if (sdom[val[x]] > sdom[val[fa[x]]])
    val[x] = val[fa[x]];
   fa[x] = p;
   return c ? p : val[x];
  } else return c ? fa[x] : val[x];
 }
 vector<int> build(int s, int n) {
  // return the father of each node in dominator tree
  dfs(s); // p[i] = -2 if i is unreachable from s
  for (int i = tk - 1; i >= 0; --i) {
   for (int u : r[i])
    sdom[i] = min(sdom[i], sdom[find(u)]);
   if (i) rdom[sdom[i]].push_back(i);
   for (int u : rdom[i]) {
    int p = find(u);
    dom[u] = (sdom[p] == i ? i : p);
   }
   if (i) merge(i, rp[i]);
  }
  vector<int> p(n, -2); p[s] = -1;
  for (int i = 1; i < tk; ++i)
   if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
  for (int i = 1; i < tk; ++i)
   p[rev[i]] = rev[dom[i]];
  return p;
 } // test @ yosupo judge
};
```

3.7 Edge Coloring [029763]

```
// max(d_u) + 1 edge coloring, time: O(NM)
int C[kN][kN], G[kN][kN]; // 1-based, G: ans
void clear(int N) {
 for (int i = 0; i <= N; i++)
  for (int j = 0; j <= N; j++)
    C[i][j] = G[i][j] = 0;
}
void solve(vector<pair<int, int>> &E, int N) {
 int X[kN] = {}, a;
 auto update = [&](int u) {
  for (X[u] = 1; C[u][X[u]]; X[u]++);
 };
 auto color = [&](int u, int v, int c) {
  int p = G[u][v];
  G[u][v] = G[v][u] = c;
  C[u][c] = v, C[v][c] = u;
  C[u][p] = C[v][p] = 0;
  if (p) X[u] = X[v] = p;
  else update(u), update(v);
  return p;
 };
 auto flip = [&](int u, int c1, int c2) {
  int p = C[u][c1];
  swap(C[u][c1], C[u][c2]);
  if (p) G[u][p] = G[p][u] = c2;
  if (!C[u][c1]) X[u] = c1;
  if (!C[u][c2]) X[u] = c2;
  return p;
 };
 for (int i = 1; i <= N; i++) X[i] = 1;
 for (int t = 0; t < E.size(); t++) {
  auto [u, v] = E[t];
  int v0 = v, c = X[u], c0 = c, d;
  vector<pair<int, int>> L; int vst[kN] = {};
  while (!G[u][v0]) {
   L.emplace_back(v, d = X[v]);
   if (!C[v][c]) for(a=L.size()-1;a>=0;a--)
     c = color(u, L[a].first, c);
   else if(!C[u][d])for(a=L.size()-1;a>=0;a--)
     color(u, L[a].first, L[a].second);
   else if (vst[d]) break;
   else vst[d] = 1, v = C[u][d];
  }
  if (!G[u][v0]) {
   for (; v; v = flip(v, c, d), swap(c, d));
   if (C[u][c0]) { a = int(L.size()) - 1;
    while (--a >= 0 && L[a].second != c);
    for(;a>=0;a--)color(u,L[a].first,L[a].second);
   } else t--;
  }
 }
}
```

3.8 Centroid Decomposition [63b2fb]

```
struct Centroid {
 using G = vector<vector<pair<int, int>>>;
 vector<vector<int64_t>> Dist;
 vector<int> Pa, Dep;
 vector<int64_t> Sub, Sub2;
 vector<int> Cnt, Cnt2;
 vector<int> vis, sz, mx, tmp;
 void DfsSz(const G &g, int x) {
  vis[x] = true, sz[x] = 1, mx[x] = 0;
  for (auto [u, w] : g[x]) if (not vis[u]) {
   DfsSz(g, u); sz[x] += sz[u];
   mx[x] = max(mx[x], sz[u]);
  }
  tmp.push_back(x);
 }
 void DfsDist(const G &g, int x, int64_t D = 0) {
  Dist[x].push_back(D); vis[x] = true;
  for (auto [u, w] : g[x])
   if (not vis[u]) DfsDist(g, u, D + w);
 }
 void DfsCen(const G &g, int x, int D = 0, int p = -1)
   {
  tmp.clear(); DfsSz(g, x);
  int M = tmp.size(), C = -1;
  for (int u : tmp) {
   if (max(M - sz[u], mx[u]) * 2 <= M) C = u;
   vis[u] = false;
  }
  DfsDist(g, C);
  for (int u : tmp) vis[u] = false;
  Pa[C] = p, vis[C] = true, Dep[C] = D;
  for (auto [u, w] : g[C])
   if (not vis[u]) DfsCen(g, u, D + 1, C);
 }
 Centroid(int N, G g)
   : Sub(N), Sub2(N), Cnt(N), Cnt2(N), Dist(N), Pa(N),
    Dep(N), vis(N), sz(N), mx(N) { DfsCen(g, 0); }
 void Mark(int v) {
  int x = v, z = -1;
  for (int i = Dep[v]; i >= 0; --i) {
   Sub[x] += Dist[v][i], Cnt[x]++;
   if (z != -1)
    Sub2[z] += Dist[v][i], Cnt2[z]++;
   x = Pa[z = x];
  }
 }
 int64_t Query(int v) {
  int64_t res = 0;
  int x = v, z = -1;
  for (int i = Dep[v]; i >= 0; --i) {
   res += Sub[x] + 1LL * Cnt[x] * Dist[v][i];
   if (z != -1)
    res -= Sub2[z] + 1LL * Cnt2[z] * Dist[v][i];
   x = Pa[z = x];
  }
  return res;
 }
};
```

3.9 Lowbit Decomposition [760ac1]

```
class LBD {
 int timer, chains;
 vector<vector<int>> G;
 vector<int> tl, tr, chain, head, dep, pa;
 // chains : number of chain
 // tl, tr[u] : subtree interval in the seq. of u
 // head[i] : head of the chain i
 // chian[u] : chain id of the chain u is on
 void predfs(int u, int f) {
  dep[u] = dep[pa[u] = f] + 1;
  for (int v : G[u]) if (v != f) {
   predfs(v, u);
   if (lowbit(chain[u]) < lowbit(chain[v]))
    chain[u] = chain[v];
  }
```

```
  if (chain[u] == 0) chain[u] = ++chains;
 }
 void dfschain(int u, int f) {
  tl[u] = timer++;
  if (head[chain[u]] == -1)
   head[chain[u]] = u;
  for (int v : G[u])
   if (v != f and chain[v] == chain[u])
    dfschain(v, u);
  for (int v : G[u])
   if (v != f and chain[v] != chain[u])
    dfschain(v, u);
  tr[u] = timer;
 }
public:
 LBD(int n) : timer(0), chains(0), G(n), tl(n), tr(n),
   chain(n), head(n + 1, -1), dep(n), pa(n) {}
 void add_edge(int u, int v) {
  G[u].push_back(v); G[v].push_back(u);
 }
 void decompose() { predfs(0, 0); dfschain(0, 0); }
 PII get_subtree(int u) { return {tl[u], tr[u]}; }
 vector<PII> get_path(int u, int v) {
  vector<PII> res;
  while (chain[u] != chain[v]) {
   if (dep[head[chain[u]]] < dep[head[chain[v]]])
    swap(u, v);
   int s = head[chain[u]];
   res.emplace_back(tl[s], tl[u] + 1);
   u = pa[s];
  }
  if (dep[u] < dep[v]) swap(u, v);
  res.emplace_back(tl[v], tl[u] + 1);
  return res;
 }
};
```

3.10 Virtual Tree [ad5cf5]

```
vector<pair<int, int>> build(vector<int> vs, int r) {
 vector<pair<int, int>> res;
 sort(vs.begin(), vs.end(), [](int i, int j) {
  return dfn[i] < dfn[j]; });
 vector<int> s = {r};
 for (int v : vs) if (v != r) {
  if (int o = lca(v, s.back()); o != s.back()) {
   while (s.size() >= 2) {
    if (dfn[s[s.size() - 2]] < dfn[o]) break;
    res.emplace_back(s[s.size() - 2], s.back());
    s.pop_back();
   }
   if (s.back() != o) {
    res.emplace_back(o, s.back());
    s.back() = o;
   }
  }
  s.push_back(v);
 }
 for (size_t i = 1; i < s.size(); ++i)
  res.emplace_back(s[i - 1], s[i]);
 return res; // (x, y): x->y
}
```

3.11 Tree Hashing [d6a9f9]

```
vector<int> g[maxn]; llu h[maxn];
llu F(llu z) { // xorshift64star from iwiwi
 z ^= z >> 12; z ^= z << 25; z ^= z >> 27;
 return z * 2685821657736338717LL;
}
llu hsah(int u, int f) {
 llu r = 127; // bigger?
 for (int v : g[u]) if (v != f) r += hsah(v, u);
 return h[u] = F(r);
} // test @ UOJ 763 & yosupo library checker
```

3.12 Mo's Algorithm on Tree

```
dfs u:
 push u
 iterate subtree
 push u
Let P = LCA(u, v) with St(u)<=St(v)
if (P == u) query[St(u), St(v)]
else query[Ed(u), St(v)], query[St(P), St(P)]
```

3.13 Count Cycles [c7e8f2]

```
// ord = sort by deg decreasing, rk[ord[i]] = i
// D[i] = edge point from rk small to rk big
for (int x : ord) { // c3
 for (int y : D[x]) vis[y] = 1;
 for (int y : D[x]) for (int z : D[y]) c3 += vis[z];
 for (int y : D[x]) vis[y] = 0;
}
for (int x : ord) { // c4
 for (int y : D[x]) for (int z : adj[y])
  if (rk[z] > rk[x]) c4 += vis[z]++;
 for (int y : D[x]) for (int z : adj[y])
  if (rk[z] > rk[x]) --vis[z];
} // both are O(M*sqrt(M)), test @ 2022 CCPC guangzhou
```

3.14 MaximalClique [293730]

```
// contain a self loop u to u, than u won't in clique
template <size_t maxn> class MaxClique {
private:
 using bits = bitset<maxn>;
 bits popped, G[maxn], ans;
 size_t deg[maxn], deo[maxn], n;
 void sort_by_degree() {
  popped.reset();
  for (size_t i = 0; i < n; ++i)
   deg[i] = G[i].count();
  for (size_t i = 0; i < n; ++i) {
   size_t mi = maxn, id = 0;
   for (size_t j = 0; j < n; ++j)
    if (not popped[j] and deg[j] < mi)
     mi = deg[id = j];
   popped[deo[i] = id] = 1;
   for (size_t u = G[i]._Find_first(); u < n;
     u = G[i]._Find_next(u))
    --deg[u];
  }
 }
 void BK(bits R, bits P, bits X) {
  if (R.count() + P.count() <= ans.count()) return;
  if (not P.count() and not X.count()) {
   if (R.count() > ans.count()) ans = R;
   return;
  }
  /* greedily chosse max degree as pivot
  bits cur = P | X; size_t pivot = 0, sz = 0;
  for ( size_t u = cur._Find_first() ;
   u < n ; u = cur._Find_next( u ) )
    if ( deg[ u ] > sz ) sz = deg[ pivot = u ];
  cur = P & ( ~G[ pivot ] );
  */ // or simply choose first
  bits cur = P & (~G[(P | X)._Find_first()]);
  for (size_t u = cur._Find_first(); u < n;
    u = cur._Find_next(u)) {
   if (R[u]) continue;
   R[u] = 1;
   BK(R, P & G[u], X & G[u]);
   R[u] = P[u] = 0, X[u] = 1;
  }
 }
public:
 void init(size_t n_) {
  n = n_;
  for (size_t i = 0; i < n; ++i) G[i].reset();
  ans.reset();
 }
 void add_edges(int u, bits S) { G[u] = S; }
 void add_edge(int u, int v) { G[u][v] = G[v][u] = 1; }
 int solve() {
  sort_by_degree(); // or simply iota( deo... )
  for (size_t i = 0; i < n; ++i)
   deg[i] = G[i].count();
  bits pob, nob = 0; pob.set();
  for (size_t i = n; i < maxn; ++i) pob[i] = 0;
  for (size_t i = 0; i < n; ++i) {
   size_t v = deo[i];
   bits tmp;
   tmp[v] = 1;
   BK(tmp, pob & G[v], nob & G[v]);
   pob[v] = 0, nob[v] = 1;
  }
  return static_cast<int>(ans.count());
 }
```

```
};
```

3.15 MaximumClique [aee5d8]

```
constexpr size_t kN = 150; using bits = bitset<kN>;
struct MaxClique {
 bits G[kN], cs[kN];
 int ans, sol[kN], q, cur[kN], d[kN], 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> &v, int i, bits mask) {
  if (i < 4) {
   for (int x : v) d[x] = (int)(G[x] & mask).count();
   sort(all(v), [&](int x, int y) {
    return d[x] > d[y]; });
  }
  vector<int> c(v.size());
  cs[1].reset(), cs[2].reset();
  int l = max(ans - q + 1, 1), r = 2, tp = 0, k;
  for (int p : v) {
   for (k = 1; (cs[k] & G[p]).any(); ++k);
   if (k >= r) cs[++r].reset();
   cs[k][p] = 1;
   if (k < l) v[tp++] = p;
  }
  for (k = l; k < r; ++k)
   for (auto p = cs[k]._Find_first();
     p < kN; p = cs[k]._Find_next(p))
    v[tp] = (int)p, c[tp] = k, ++tp;
  dfs(v, c, i + 1, mask);
 }
 void dfs(vector<int> &v, vector<int> &c,
   int i, bits mask) {
  while (!v.empty()) {
   int p = v.back(); v.pop_back(); mask[p] = 0;
   if (q + c.back() <= ans) return;
   cur[q++] = p;
   vector<int> nr;
   for (int x : v) if (G[p][x]) nr.push_back(x);
   if (!nr.empty()) pre_dfs(nr, i, mask & G[p]);
   else if (q > ans) ans = q, copy_n(cur, q, sol);
   c.pop_back(); --q;
  }
 }
 int solve() {
  vector<int> v(n); iota(all(v), 0);
  ans = q = 0; pre_dfs(v, 0, bits(string(n, '1')));
  return ans; // sol[0 ~ ans-1]
 }
} cliq; // test @ yosupo judge
```

3.16 Minimum Mean Cycle [e23bc0]

```
// WARNING: TYPE matters
struct Edge { int s, t; llf c; };
llf solve(vector<Edge> &e, int n) {
 // O(VE), returns inf if no cycle, mmc otherwise
 vector<VI> prv(n + 1, VI(n)), prve = prv;
 vector<vector<llf>> d(n + 1, vector<llf>(n, inf));
 d[0] = vector<llf>(n, 0);
 for (int i = 0; i < n; i++) {
  for (int j = 0; j < (int)e.size(); j++) {
   auto [s, t, c] = e[j];
   if (d[i][s] < inf && d[i + 1][t] > d[i][s] + c) {
    d[i + 1][t] = d[i][s] + c;
    prv[i + 1][t] = s; prve[i + 1][t] = j;
   }
  }
 }
 llf mmc = inf; int st = -1;
 for (int i = 0; i < n; i++) {
  llf avg = -inf;
  for (int k = 0; k < n; k++) {
   if (d[n][i] < inf - eps)
    avg = max(avg, (d[n][i] - d[k][i]) / (n - k));
   else avg = inf;
  }
  if (avg < mmc) tie(mmc, st) = tie(avg, i);
 }
 if (st == -1) return inf;
 vector<int> vst(n), eid, cycle, rho;
 for (int i = n; !vst[st]; st = prv[i--][st]) {
  vst[st]++; eid.emplace_back(prve[i][st]);
  rho.emplace_back(st);
 }
 while (vst[st] != 2) {
  int v = rho.back(); rho.pop_back();
  cycle.emplace_back(v); vst[v]++;
 }
 reverse(all(eid)); eid.resize(cycle.size());
 return mmc;
}
```

4 Flow & Matching

4.1 HopcroftKarp [930040]

```
struct HK {
 vector<int> l, r, a, p; int ans;
 HK(int n, int m, auto &g) : l(n,-1), r(m,-1), ans(0) {
  for (bool match = true; match;) {
   match = false; a.assign(n, -1); p.assign(n, -1);
   queue<int> q;
   for (int i = 0; i < n; i++)
    if (l[i] == -1) q.push(a[i] = p[i] = i);
   // bitset<maxn> nvis, t; nvis.set();
   while (!q.empty()) {
    int z, x = q.front(); q.pop();
    if (l[a[x]] != -1) continue;
    for (int y : g[x]) { // or iterate t = g[x]&nvis
     // nvis.reset(y);
     if (r[y] == -1) {
      for (z = y; z != -1; )
       r[z] = x, swap(l[x], z), x = p[x];
      match = true; ++ans; break;
     } else if (p[r[y]] == -1)
      q.push(z = r[y]), p[z] = x, a[z] = a[x];
    }
   }
  }
 }
};
```

4.2 Dijkstra Cost Flow [06d5f2]

```
template <typename F, typename C> class MCMF {
 static constexpr F INF_F = numeric_limits<F>::max();
 static constexpr C INF_C = numeric_limits<C>::max();
 struct E {
  int to, r; F f; C c;
  E(int a, int b, F x, C y)
   : to(a), r(b), f(x), c(y) {}
 };
 vector<vector<E>> g; vector<pair<int, int>> f;
 vector<F> up; vector<C> d, h;
 optional<pair<F, C>> step(int S, int T) {
  priority_queue<pair<C, int>> q;
  q.emplace(d[S] = 0, S), up[S] = INF_F;
  while (not q.empty()) {
   auto [l, u] = q.top(); q.pop();
   if (up[u] == 0 or l != -d[u]) continue;
   for (int i = 0; i < int(g[u].size()); ++i) {
    auto e = g[u][i]; int v = e.to;
    auto nd = d[u] + e.c + h[u] - h[v];
    if (e.f <= 0 or d[v] <= nd) continue;
    f[v] = {u, i}; up[v] = min(up[u], e.f);
    q.emplace(-(d[v] = nd), v);
   }
  }
  if (d[T] == INF_C) return nullopt;
  for (size_t i = 0; i < d.size(); i++) h[i]+=d[i];
  for (int i = T; i != S; i = f[i].first) {
   auto &eg = g[f[i].first][f[i].second];
   eg.f -= up[T]; g[eg.to][eg.r].f += up[T];
  }
  return pair{up[T], h[T]};
 }
public:
 MCMF(int n) : g(n), f(n), up(n), d(n, INF_C) {}
 void add_edge(int s, int t, F c, C w) {
  g[s].emplace_back(t, int(g[t].size()), c, w);
  g[t].emplace_back(s, int(g[s].size()) - 1, 0, -w);
 }
 pair<F, C> solve(int a, int b) {
  h.assign(g.size(), 0);
  F c = 0; C w = 0;
```

```
  while (auto r = step(a, b)) {
   c += r->first, w += r->first * r->second;
   fill(d.begin(), d.end(), INF_C);
  }
  return {c, w};
 }
};
```

4.3 Dinic [32c53e]

```
template <typename Cap = int64_t> class Dinic {
private:
 struct E { int to, rev; Cap cap; }; int n, st, ed;
 vector<vector<E>> G; vector<size_t> lv, idx;
 bool BFS(int k) {
  lv.assign(n, 0); idx.assign(n, 0);
  queue<int> bfs; bfs.push(st); lv[st] = 1;
  while (not bfs.empty() and not lv[ed]) {
   int u = bfs.front(); bfs.pop();
   for (auto e: G[u]) if (e.cap >> k and !lv[e.to])
    bfs.push(e.to), lv[e.to] = lv[u] + 1;
  }
  return lv[ed];
 }
 Cap DFS(int u, Cap f = numeric_limits<Cap>::max()) {
  if (u == ed) return f;
  Cap ret = 0;
  for (auto &i = idx[u]; i < G[u].size(); ++i) {
   auto &[to, rev, cap] = G[u][i];
   if (cap <= 0 or lv[to] != lv[u] + 1) continue;
   Cap nf = DFS(to, min(f, cap));
   ret += nf; cap -= nf; f -= nf;
   G[to][rev].cap += nf;
   if (f == 0) return ret;
  }
  if (ret == 0) lv[u] = 0;
  return ret;
 }
public:
 void init(int n_) { G.assign(n = n_, vector<E>()); }
 void add_edge(int u, int v, Cap c) {
  G[u].push_back({v, int(G[v].size()), c});
  G[v].push_back({u, int(G[u].size())-1, 0});
 }
 Cap max_flow(int st_, int ed_) {
  st = st_, ed = ed_; Cap ret = 0;
  for (int i = 63; i >= 0; --i)
   while (BFS(i)) ret += DFS(st);
  return ret;
 }
}; // test @ luogu P3376
```

4.4 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. Also, f is a mincost valid flow.
     - 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'.
- Submodular functions minimization
  - For a function $f : 2^V \to \mathbb{R}$, f is a submodular function iff
    * $\forall S, T \subseteq V, f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$, or
    * $\forall X \subseteq Y \subseteq V, x \notin Y, f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$.
  - To minimize $\sum_i \theta_i(x_i) + \sum_{i<j} \phi_{ij}(x_i, x_j) + \sum_{i<j<k} \psi_{ijk}(x_i, x_j, x_k)$
  - If $\theta_i(1) \geq \theta_i(0)$, add edge ($S$, i, $\theta_i(1) - \theta_i(0)$) and $\theta_i(0)$ to answer; otherwise, (i, T, $\theta_i(0) - \theta_i(1)$) and $\theta_i(1)$.
  - Add edges (i, j, $\phi_{ij}(0,1) + \phi_{ij}(1,0) - \phi_{ij}(0,0) - \phi_{ij}(1,1)$).
  - Denote x_{ijk} as helper nodes. Let $P = \psi_{ijk}(0,0,0) + \psi_{ijk}(0,1,1) + \psi_{ijk}(1,0,1) + \psi_{ijk}(1,1,0) - \psi_{ijk}(0,0,1) - \psi_{ijk}(0,1,0) - \psi_{ijk}(1,0,0) - \psi_{ijk}(1,1,1)$. Add $-P$ to answer. If $P \geq 0$, add edges (i, x_{ijk}, P), (j,x_{ijk},P), (k,x_{ijk},P), (x_{ijk}, T, P); otherwise (x_{ijk},i,$-P$), (x_{ijk},j,$-P$), (x_{ijk},k,$-P$), (S, x_{ijk}, $-P$).
  - The minimum cut of this graph will be the the minimum value of the function above.

4.5 General Graph Matching [5f2293]

```
struct Matching {
 queue<int> q; int ans, n;
 vector<int> fa, s, v, pre, match;
 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 (v[x] == tk) return x;
   v[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(auto &&g, int r) {
  iota(all(fa), 0); ranges::fill(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(auto &&g) : ans(0), n(int(g.size())),
 fa(n+1), s(n+1), v(n+1), pre(n+1, n), match(n+1, n) {
  for (int x = 0; x < n; ++x)
   if (match[x] == n) ans += Bfs(g, x);
 } // match[x] == n means not matched
}; // test @ yosupo judge
```

4.6 Global Min-Cut [1f0306]

```
const int maxn = 500 + 5;
int w[maxn][maxn], g[maxn];
bool v[maxn], del[maxn];
void add_edge(int x, int y, int c) {
 w[x][y] += c; w[y][x] += c;
}
pair<int, int> phase(int n) {
 memset(v, false, sizeof(v));
 memset(g, 0, sizeof(g));
 int s = -1, t = -1;
 while (true) {
  int c = -1;
  for (int i = 0; i < n; ++i) {
   if (del[i] || v[i]) continue;
   if (c == -1 || g[i] > g[c]) c = i;
  }
  if (c == -1) break;
  v[s = t, t = c] = true;
  for (int i = 0; i < n; ++i) {
   if (del[i] || v[i]) continue;
   g[i] += w[c][i];
  }
 }
 return make_pair(s, t);
}
int mincut(int n) {
 int cut = 1e9;
 memset(del, false, sizeof(del));
 for (int i = 0; i < n - 1; ++i) {
  int s, t; tie(s, t) = phase(n);
  del[t] = true; cut = min(cut, g[t]);
  for (int j = 0; j < n; ++j) {
   w[s][j] += w[t][j]; w[j][s] += w[j][t];
  }
 }
 return cut;
}
```

4.7 GomoryHu Tree [7473bb]

```
vector<tuple<int, int, int>> GomoryHu(int n){
 vector<tuple<int, int, int>> rt;
 vector<int> g(n);
 for (int i = 1; i < n; ++i) {
  int t = g[i];
  flow.reset(); // clear flows on all edge
  rt.emplace_back(i, t, flow.max_flow(i, t));
  flow.walk(i); // bfs points that connected to i (use
    edges with .cap > 0)
  for (int j = i + 1; j < n; ++j)
   if (g[j]==t&&flow.connect(j)) // check if i can
    reach j
    g[j] = i;
 }
 return rt;
}
```

4.8 Kuhn Munkres [2c09ed]

```
struct KM { // maximize, test @ UOJ 80
 int n, l, r; lld ans; // fl and fr are the match
 vector<lld> hl, hr; vector<int> fl, fr, pre, q;
 void bfs(const auto &w, int s) {
  vector<int> vl(n), vr(n); vector<lld> slk(n, INF);
  l = r = 0; vr[q[r++] = s] = true;
  const auto check = [&](int x) -> bool {
   if (vl[x] || slk[x] > 0) return true;
   vl[x] = true; slk[x] = INF;
   if (fl[x] != -1) return vr[q[r++] = fl[x]] = true;
   while (x != -1) swap(x, fr[fl[x] = pre[x]]);
   return false;
  };
  while (true) {
   while (l < r)
    for (int x = 0, y = q[l++]; x < n; ++x) if (!vl[x])
     if (chmin(slk[x], hl[x] + hr[y] - w[x][y]))
      if (pre[x] = y, !check(x)) return;
   lld d = ranges::min(slk);
   for (int x = 0; x < n; ++x)
    vl[x] ? hl[x] += d : slk[x] -= d;
   for (int x = 0; x < n; ++x) if (vr[x]) hr[x] -= d;
   for (int x = 0; x < n; ++x) if (!check(x)) return;
  }
 }
 KM(int n_, const auto &w) : n(n_), ans(0),
  hl(n), hr(n), fl(n, -1), fr(fl), pre(n), q(n) {
  for (int i = 0; i < n; ++i) hl[i]=ranges::max(w[i]);
  for (int i = 0; i < n; ++i) bfs(w, i);
  for (int i = 0; i < n; ++i) ans += w[i][fl[i]];
 }
};
```

4.9 Minimum Cost Circulation [0f0e85]

```
int vis[N], visc, fa[N], fae[N], head[N], mlc = 1;
struct ep {
 int to, next;
 ll flow, cost;
} e[M << 1];
void adde(int u, int v, ll fl, int cs) {
 e[++mlc] = {v, head[u], fl, cs};
 head[u] = mlc;
 e[++mlc] = {u, head[v], 0, -cs};
 head[v] = mlc;
}
void dfs(int u) {
 vis[u] = 1;
 for (int i = head[u], v; i; i = e[i].next)
  if (!vis[v = e[i].to] and e[i].flow)
   fa[v] = u, fae[v] = i, dfs(v);
}
ll phi(int x) {
 static ll pi[N];
 if (x == -1) return 0;
 if (vis[x] == visc) return pi[x];
 return vis[x] = visc, pi[x] = phi(fa[x]) - e[fae[x]].
    cost;
}
void pushflow(int x, ll &cost) {
 int v = e[x ^ 1].to, u = e[x].to;
 ++visc;
 while (v != -1) vis[v] = visc, v = fa[v];
 while (u != -1 && vis[u] != visc)
  vis[u] = visc, u = fa[u];
 vector<int> cyc;
 int e2 = 0, pa = 2;
 ll f = e[x].flow;
 for (int i = e[x ^ 1].to; i != u; i = fa[i]) {
  cyc.push_back(fae[i]);
  if (e[fae[i]].flow < f)
   f = e[fae[e2 = i] ^ (pa = 0)].flow;
 }
 for (int i = e[x].to; i != u; i = fa[i]) {
  cyc.push_back(fae[i] ^ 1);
  if (e[fae[i] ^ 1].flow < f)
   f = e[fae[e2 = i] ^ (pa = 1)].flow;
 }
 cyc.push_back(x);
 for (int cyc_i : cyc) {
  e[cyc_i].flow -= f, e[cyc_i ^ 1].flow += f;
  cost += 1ll * f * e[cyc_i].cost;
 }
 if (pa == 2) return;
 int le = x ^ pa, l = e[le].to, o = e[le ^ 1].to;
 while (l != e2) {
  vis[o] = 0;
  swap(le ^= 1, fae[o]), swap(l, fa[o]), swap(l, o);
 }
}
ll simplex() { // 1-based
 ll cost = 0;
 memset(fa, -1, sizeof(fa)), dfs(1);
 vis[1] = visc = 2, fa[1] = -1;
 for (int i = 2, pre = -1; i != pre; i = (i == mlc ? 2
    : i + 1))
  if (e[i].flow and e[i].cost < phi(e[i ^ 1].to) - phi(
    e[i].to))
   pushflow(pre = i, cost);
 return cost;
}
```

4.10 Minimum Cost Max Flow [6d1b01]

```
template <typename F, typename C> class MCMF {
 static constexpr F INF_F = numeric_limits<F>::max();
 static constexpr C INF_C = numeric_limits<C>::max();
 struct E {
```

```
  int to, r;
  F f; C c;
  E() {}
  E(int a, int b, F x, C y)
   : to(a), r(b), f(x), c(y) {}
 };
 vector<vector<E>> g;
 vector<pair<int, int>> f;
 vector<bool> inq;
 vector<F> up; vector<C> d;
 optional<pair<F, C>> step(int S, int T) {
  queue<int> q;
  for (q.push(S), d[S] = 0, up[S] = INF_F;
    not q.empty(); q.pop()) {
   int u = q.front(); inq[u] = false;
   if (up[u] == 0) continue;
   for (int i = 0; i < int(g[u].size()); ++i) {
    auto e = g[u][i]; int v = e.to;
    if (e.f <= 0 or d[v] <= d[u] + e.c)
     continue;
    d[v] = d[u] + e.c; f[v] = {u, i};
    up[v] = min(up[u], e.f);
    if (not inq[v]) q.push(v);
    inq[v] = true;
   }
  }
  if (d[T] == INF_C) return nullopt;
  for (int i = T; i != S; i = f[i].first) {
   auto &eg = g[f[i].first][f[i].second];
   eg.f -= up[T];
   g[eg.to][eg.r].f += up[T];
  }
  return pair{up[T], d[T]};
 }
public:
 MCMF(int n) : g(n),f(n),inq(n),up(n),d(n,INF_C) {}
 void add_edge(int s, int t, F c, C w) {
  g[s].emplace_back(t, int(g[t].size()), c, w);
  g[t].emplace_back(s, int(g[s].size()) - 1, 0, -w);
 }
 pair<F, C> solve(int a, int b) {
  F c = 0; C w = 0;
  while (auto r = step(a, b)) {
   c += r->first, w += r->first * r->second;
   fill(inq.begin(), inq.end(), false);
   fill(d.begin(), d.end(), INF_C);
  }
  return {c, w};
 }
};
```

4.11 Weighted Matching [94ca35]

```
#define pb emplace_back
#define rep(i, l, r) for (int i=(l); i<=(r); ++i)
struct WeightGraph { // 1-based
 static const int inf = INT_MAX;
 struct edge { int u, v, w; }; int n, nx;
 vector<int> lab; vector<vector<edge>> g;
 vector<int> slack, 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)),slack(nx + 1),
  flo(nx + 1), flo_from(nx + 1, vector(n + 1, 0)) {
  match = st = pa = S = vis = slack;
  rep(u, 1, n) rep(v, 1, n) g[u][v] = {u, v, 0};
 }
 int ED(edge e) {
  return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2; }
 void update_slack(int u, int x, int &s) {
  if (!s || ED(g[u][x]) < ED(g[s][x])) s = u; }
 void set_slack(int x) {
  slack[x] = 0;
  for (int u = 1; u <= n; ++u)
   if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
    update_slack(u, x, slack[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, int(z.size())-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(xnv, st[pa[xnv]]);
   u = st[pa[xnv]], v = 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 = int(find(n + 1 + all(st), 0) - begin(st));
  lab[b] = 0, S[b] = 0; match[b] = match[o];
  vector<int> f = {o};
  for (int x = u, y; x != o; x = st[pa[y]])
   f.pb(x), f.pb(y = st[match[x]]), q_push(y);
  reverse(1 + all(f));
  for (int x = v, 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);
  for (int x = 1; x <= nx; ++x)
   g[b][x].w = g[x][b].w = 0;
  for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
  for (int xs : flo[b]) {
   for (int x = 1; x <= nx; ++x)
    if (g[b][x].w == 0 || ED(g[xs][x]) < ED(g[b][x]))
     g[b][x] = g[xs][x], g[x][b] = g[x][xs];
   for (int x = 1; x <= n; ++x)
    if (flo_from[xs][x]) flo_from[b][x] = xs;
  }
  set_slack(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;
   slack[xs] = 0; set_slack(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_slack(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;
   slack[v] = slack[nu] = 0; 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() {
  ranges::fill(S, -1); ranges::fill(slack, 0);
  q = queue<int>();
```

```
  for (int x = 1; x <= nx; ++x)
   if (st[x] == x && !match[x])
    pa[x] = 0, S[x] = 0, q_push(x);
  if (q.empty()) return false;
  for (;;) {
   while (q.size()) {
    int u = q.front(); q.pop();
    if (S[st[u]] == 1) continue;
    for (int v = 1; v <= n; ++v)
     if (g[u][v].w > 0 && st[u] != st[v]) {
      if (ED(g[u][v]) != 0)
       update_slack(u, st[v], slack[st[v]]);
      else if (on_found_edge(g[u][v])) return true;
     }
   }
   int d = inf;
   for (int b = n + 1; b <= nx; ++b)
    if (st[b] == b && S[b] == 1)
     d = min(d, lab[b] / 2);
   for (int x = 1; x <= nx; ++x)
    if (int s = slack[x]; st[x] == x && s && S[x] <= 0)
     d = min(d, ED(g[s][x]) / (S[x] + 2));
   for (int u = 1; u <= n; ++u)
    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]);
   for (int x = 1; x <= nx; ++x)
    if (int s = slack[x]; st[x] == x &&
      s && st[s] != x && ED(g[s][x]) == 0)
     if (on_found_edge(g[s][x])) return true;
   for (int b = n + 1; b <= nx; ++b)
    if (st[b] == b && S[b] == 1 && lab[b] == 0)
     expand_blossom(b);
  }
  return false;
 }
 pair<lld, int> solve() {
  ranges::fill(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);
  }
  for (int u = 1; u <= n; ++u) lab[u] = w_max;
  int n_matches = 0; lld 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 set_edge(int u, int v, int w) {
  g[u][v].w = g[v][u].w = w; }
};
```

5 Math

5.1 Common Bounds

$p(0)=1,\ p(n)=\sum_{k\in\mathbb{Z}\setminus\{0\}}(-1)^{k+1}p(n-k(3k-1)/2)\approx 0.145/n\cdot\exp(2.56\sqrt{n})$

n	100	1e3	1e6	1e9	1e12	1e15	1e18
$\max_{i\le n}(d(i))$	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

5.2 Stirling Number

First Kind

$S_1(n,k)$ counts the number of permutations of n elements with k disjoint cycles.

$$S_1(n,k)=(n-1)\cdot S_1(n-1,k)+S_1(n-1,k-1)$$

$$x(x+1)\dots(x+n-1)=\sum_{k=0}^{n}S_1(n,k)x^k$$

$$g(x)=x(x+1)\dots(x+n-1)=\sum_{k=0}^{n}a_kx^k$$

$$\Rightarrow g(x+n)=\sum_{k=0}^{n}\frac{b_k}{(n-k)!}x^{n-k},$$

$$b_k=\sum_{i=0}^{k}((n-i)!a_{n-i})\cdot(\frac{n^{k-i}}{(k-i)!})$$

Second Kind

$S_2(n,k)$ counts the number of ways to partition a set of n elements into k nonempty sets.

$$S_2(n,k)=S_2(n-1,k-1)+k\cdot S_2(n-1,k)$$

$$S_2(n,k)=\sum_{i=0}^{k}\binom{k}{i}i^n(-1)^{k-i}=\sum_{i=0}^{k}\frac{(-1)^i}{i!}\cdot\frac{(k-i)^n}{(k-i)!}$$

5.3 ax+by=gcd [d0cbdd]

```
// ax+ny = 1, ax+ny == ax == 1 (mod n)
void exgcd(lld x, lld y, lld &g, lld &a, lld &b) {
 if (y == 0) g = x, a = 1, b = 0;
 else exgcd(y, x % y, g, b, a), b -= (x / y) * a;
}
```

5.4 Chinese Remainder [d69e74]

```
// please ensure r_i\in[0,m_i)
bool crt(lld &m1, lld &r1, lld m2, lld r2) {
  if (m2 > m1) swap(m1, m2), swap(r1, r2);
  lld g, a, b; exgcd(m1, m2, g, a, b);
  if ((r2 - r1) % g != 0) return false;
  m2 /= g; lld D = (r2 - r1) / g % m2 * a % m2;
  r1 += (D < 0 ? D + m2 : D) * m1; m1 *= m2;
  assert (r1 >= 0 && r1 < m1);
  return true;
}
```

5.5 DiscreteLog [86e463]

```
template<typename Int>
Int BSGS(Int x, Int y, Int M) {
 // x^? \equiv y (mod M)
 Int t = 1, c = 0, g = 1;
 for (Int M_ = M; M_ > 0; M_ >>= 1) g = g * x % M;
 for (g = gcd(g, M); t % g != 0; ++c) {
  if (t == y) return c;
  t = t * x % M;
 }
 if (y % g != 0) return -1;
 t /= g, y /= g, M /= g;
 Int h = 0, gs = 1;
 for (; h * h < M; ++h) gs = gs * x % M;
 unordered_map<Int, Int> bs;
 for (Int s = 0; s < h; bs[y] = ++s) y = y * x % M;
 for (Int s = 0; s < M; s += h) {
  t = t * gs % M;
  if (bs.count(t)) return c + s + h - bs[t];
 }
 return -1;
}
```

5.6 Quadratic Residue [1eabad]

```
int get_root(int n, int P) { // ensure 0 <= n < p
 if (P == 2 or n == 0) return n;
 auto check = [&](int x) {
  return modpow(x, (P - 1) / 2, P); };
 if (check(n) != 1) return -1;
 mt19937 rnd(7122); lld z = 1, w;
 while (check(w = (z * z - n + P) % P) != P - 1)
  z = rnd() % P;
 const auto M = [P, w](auto &u, auto &v) {
  auto [a, b] = u; auto [c, d] = v;
  return make_pair((a * c + b * d % P * w) % P,
    (a * d + b * c) % P);
 };
 pair<lld, lld> r(1, 0), e(z, 1);
 for (int w = (P + 1) / 2; w; w >>= 1, e = M(e, e))
  if (w & 1) r = M(r, e);
 return r.first; // sqrt(n) mod P where P is prime
}
```

5.7 Extended Euler

$$a^b\equiv\begin{cases}a^{(b \bmod \varphi(m))+\varphi(m)} & \text{if } (a,m)\neq 1\wedge b\geq\varphi(m)\\ a^{b \bmod \varphi(m)} & \text{otherwise}\end{cases}\pmod m$$

5.8 Extended FloorSum

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

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

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

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

5.9 Extended Euclidean [e09892]

```
template <typename T>
auto euclid(lld a, lld b, lld c, lld n, T U, T R) {
 b %= c;
 if (a >= c)
  return euclid(a % c, b, c, n, U, mpow(U, a / c) * R);
 lld m = (i128(a) * n + b) / c;
 if (!m) return mpow(R, n);
 return mpow(R, (c - b - 1) / a) * U *
        euclid(c, c - b - 1, a, m - 1, R, U) *
        mpow(R, n - (i128(c) * m - b - 1) / a);
}
// time complexity is O(log max(a, c))
// 給定二維座標系上的一次函數 $y = (ax + b) / c$
// 維護一個矩陣 $A = I$，考慮 $x \in [0, n)$
// 每次向右穿過網格的垂直線時，乘上一個矩陣 $R$
// 每次向上穿過網格的水平線時，乘上一個矩陣 $U$
// 若剛好經過一個整點，那麼先乘 $U$ 再乘 $R$
```

5.10 FloorSum [fb5917]

```
// @param n `n < 2^32`
// @param m `1 <= m < 2^32`
// @return sum_{i=0}^{n-1} floor((ai + b)/m) mod 2^64
llu floor_sum_unsigned(llu n, llu m, llu a, llu b) {
 llu ans = 0;
 while (true) {
  if (a >= m) ans += n*(n-1)/2 * (a/m), a %= m;
  if (b >= m) ans += n * (b/m), b %= m;
  if (llu y_max = a * n + b; y_max >= m) {
   n = (llu)(y_max / m), b = (llu)(y_max % m);
   swap(m, a);
  } else break;
 }
 return ans;
}
lld floor_sum(lld n, lld m, lld a, lld b) {
 llu ans = 0;
 if (a < 0) {
  llu a2 = (a % m + m), d = (a2 - a) / m;
  ans -= 1ULL * n * (n - 1) / 2 * d; a = a2;
 }
 if (b < 0) {
  llu b2 = (b % m + m), d = (b2 - b) / m;
  ans -= 1ULL * n * d; b = b2;
 }
 return ans + floor_sum_unsigned(n, m, a, b);
}
```

5.11 ModMin [253e4d]

```
// min{k | l <= ((ak) mod m) <= r}
optional<llu> mod_min(u32 a, u32 m, u32 l, u32 r) {
  if (a == 0) return l ? nullopt : 0;
  if (auto k = llu(l + a - 1) / a; k * a <= r)
   return k;
  auto b = m / a, c = m % a;
  if (auto y = mod_min(c, a, a - r % a, a - l % a))
   return (l + *y * c + a - 1) / a + *y * b;
 return nullopt;
}
```

5.12 FWT [f82550]

```
/* or convolution:
 * x = (x0, x0+x1), inv = (x0, x1-x0) w/o final div
 * and convolution:
 * x = (x0+x1, x1), inv = (x0-x1, x1) w/o final div */
void fwt(int x[], int N, bool inv = false) {
 for (int d = 1; d < N; d <<= 1)
  for (int s = 0; s < N; s += d * 2)
   for (int i = s; i < s + d; i++) {
    int j = i + d, ta = x[i], tb = x[j];
     x[i] = add(ta, tb);
     x[j] = sub(ta, tb);
    }
 if (inv) {
  const int invn = modinv(N);
  for (int i = 0; i < N; i++)
   x[i] = mul(x[i], invn);
 }
}
```

5.13 Packed FFT [321552]

```
int round2k(size_t n) {
 int sz = 1; while (sz < int(n)) sz *= 2; return sz; }
VL convolution(const VI &a, const VI &b) {
 const int sz = round2k(a.size() + b.size() - 1);
 // Should be able to handle N <= 10^5, C <= 10^4
 vector<P> v(sz);
 for (size_t i = 0; i < a.size(); i++) v[i].RE(a[i]);
 for (size_t i = 0; i < b.size(); i++) v[i].IM(b[i]);
 fft(v.data(), sz, /*inv=*/false);
 auto rev = v; reverse(1 + all(rev));
 for (int i = 0; i < sz; i++) {
  P A = (v[i] + conj(rev[i])) / P(2, 0);
  P B = (v[i] - conj(rev[i])) / P(0, 2);
  v[i] = A * B;
 }
 VL c(sz); fft(v.data(), sz, /*inv=*/true);
 for (int i = 0; i < sz; ++i) c[i] = roundl(RE(v[i]));
 return c;
}
VI convolution_mod(const VI &a, const VI &b) {
 const int sz = round2k(a.size() + b.size() - 1);
 vector<P> fa(sz), fb(sz);
 for (size_t i = 0; i < a.size(); ++i)
  fa[i] = P(a[i] & ((1 << 15) - 1), a[i] >> 15);
 for (size_t i = 0; i < b.size(); ++i)
  fb[i] = P(b[i] & ((1 << 15) - 1), b[i] >> 15);
 fft(fa.data(), sz); fft(fb.data(), sz);
 auto rfa = fa; reverse(1 + all(rfa));
 for (int i = 0; i < sz; ++i) fa[i] *= fb[i];
 for (int i = 0; i < sz; ++i) fb[i] *= conj(rfa[i]);
 fft(fa.data(), sz, true); fft(fb.data(), sz, true);
 vector<int> res(sz);
 for (int i = 0; i < sz; ++i) {
  lld A = (lld)roundl(RE((fa[i] + fb[i]) / P(2, 0)));
  lld C = (lld)roundl(IM((fa[i] - fb[i]) / P(0, 2)));
  lld B = (lld)roundl(IM(fa[i])); B %= p; C %= p;
  res[i] = (A + (B << 15) + (C << 30)) % p;
 }
 return res;
} // test @ yosupo judge with long double
```

5.14 CRT for arbitrary mod [e4dde7]

```
const int mod = 1000000007;
const int M1 = 985661441; // G = 3 for M1, M2, M3
const int M2 = 998244353;
const int M3 = 1004535809;
int superBigCRT(lld A, lld B, lld C) {
  static_assert (M1 < M2 && M2 < M3);
  constexpr lld r12 = modpow(M1, M2-2, M2);
  constexpr lld r13 = modpow(M1, M3-2, M3);
  constexpr lld r23 = modpow(M2, M3-2, M3);
  constexpr lld M1M2 = 1LL * M1 * M2 % mod;
  B = (B - A + M2) * r12 % M2;
  C = (C - A + M3) * r13 % M3;
  C = (C - B + M3) * r23 % M3;
  return (A + B * M1 + C * M1M2) % mod;
}
```

5.15 NTT / FFT [41c1f2]

```
template <int mod, int G, int maxn> struct NTT {
 static_assert (maxn == (maxn & -maxn));
 int roots[maxn];
 NTT () {
  int r = modpow(G, (mod - 1) / maxn);
  for (int i = maxn >> 1; i; i >>= 1) {
   roots[i] = 1;
   for (int j = 1; j < i; j++)
    roots[i + j] = mul(roots[i + j - 1], r);
   r = mul(r, r);
   // for (int j = 0; j < i; j++) // FFT (tested)
   //  roots[i+j] = polar<llf>(1, PI * j / i);
```

```cpp
  }
 }
 // n must be 2^k, and 0 <= F[i] < mod
 template <typename T>
 void operator()(int F[], T n, bool inv = false) {
  for (T i = 0, j = 0; i < n; i++) {
   if (i < j) swap(F[i], F[j]);
   for (T k = n>>1; (j^=k) < k; k>>=1);
  }
  for (T s = 1; s < n; s *= 2) {
   for (T i = 0; i < n; i += s * 2) {
    for (T j = 0; j < s; j++) {
     int a = F[i+j], b = mul(F[i+j+s], roots[s+j]);
     F[i+j] = add(a, b);  // a + b
     F[i+j+s] = sub(a, b); // a - b
    }
   }
  }
  if (inv) {
   int iv = modinv(int(n));
   for (T i = 0; i < n; i++) F[i] = mul(F[i], iv);
   reverse(F + 1, F + n);
  }
 }
};
```

5.16 Formal Power Series [b0d137]

```cpp
#define fi(l, r) for (size_t i = (l); i < (r); ++i)
using S = vector<int>;
auto Mul(auto a, auto b, size_t sz) {
 a.resize(sz), b.resize(sz);
 ntt(a.data(), sz); ntt(b.data(), sz);
 fi(0, sz) a[i] = mul(a[i], b[i]);
 return ntt(a.data(), sz, true), a;
}
S Newton(const S &v, int init, auto &&iter) {
 S Q = { init };
 for (int sz = 2; Q.size() < v.size(); sz *= 2) {
  S A{begin(v), begin(v) + min(sz, int(v.size()))};
  A.resize(sz * 2), Q.resize(sz * 2);
  iter(Q, A, sz * 2); Q.resize(sz);
 }
 return Q.resize(v.size()), Q;
}
S Inv(const S &v) { // v[0] != 0
 return Newton(v, modinv(v[0]),
  [](S &X, S &A, int sz) {
   ntt(X.data(), sz), ntt(A.data(), sz);
   for (int i = 0; i < sz; i++)
    X[i] = mul(X[i], sub(2, mul(X[i], A[i])));
   ntt(X.data(), sz, true); });
}
S Dx(S A) {
 fi(1, A.size()) A[i - 1] = mul(i, A[i]);
 return A.empty() ? A : (A.pop_back(), A);
}
S Sx(S A) {
 A.insert(A.begin(), 0);
 fi(1, A.size()) A[i] = mul(modinv(int(i)), A[i]);
 return A;
}
S Ln(const S &A) { // coef[0] == 1; res[0] == 0
 auto B = Sx(Mul(Dx(A), Inv(A), bit_ceil(A.size()*2)));
 return B.resize(A.size()), B;
}
S Exp(const S &v) { // coef[0] == 0; res[0] == 1
 return Newton(v, 1,
  [](S &X, S &A, int sz) {
   auto Y = X; Y.resize(sz / 2); Y = Ln(Y);
   fi(0, Y.size()) Y[i] = sub(A[i], Y[i]);
   Y[0] = add(Y[0], 1); X = Mul(X, Y, sz); });
}
S Pow(S a, lld M) { // period mod*(mod-1)
 assert(!a.empty() && a[0] != 0);
 const auto imul = [&a](int s) {
  for (int &x: a) x = mul(x, s); }; int c = a[0];
 imul(modinv(c)); a = Ln(a); imul(int(M % mod));
 a = Exp(a); imul(modpow(c, int(M % (mod - 1))));
 return a; // mod x^N where N=a.size()
}
S Sqrt(const S &v) { // need: QuadraticResidue
 assert(!v.empty() && v[0] != 0);
 const int r = get_root(v[0]); assert(r != -1);
 return Newton(v, r,
  [](S &X, S &A, int sz) {
   auto Y = X; Y.resize(sz / 2);
   auto B = Mul(A, Inv(Y), sz);
   for (int i = 0, inv2 = mod / 2 + 1; i < sz; i++)
    X[i] = mul(inv2, add(X[i], B[i])); });
}
S Mul(auto &&a, auto &&b) {
 const auto n = a.size() + b.size() - 1;
 auto R = Mul(a, b, bit_ceil(n));
 return R.resize(n), R;
}
S MulT(S a, S b, size_t k) {
 assert(b.size()); reverse(all(b)); auto R = Mul(a, b);
 R = vector(R.begin() + b.size() - 1, R.end());
 return R.resize(k), R;
}
S Eval(const S &f, const S &x) {
 if (f.empty()) return vector(x.size(), 0);
 const int n = int(max(x.size(), f.size()));
 auto q = vector(n * 2, S(2, 1)); S ans(n);
 fi(0, x.size()) q[i + n][1] = sub(0, x[i]);
 for (int i = n - 1; i > 0; i--)
  q[i] = Mul(q[i << 1], q[i << 1 | 1]);
 q[1] = MulT(f, Inv(q[1]), n);
 for (int i = 1; i < n; i++) {
  auto L = q[i << 1], R = q[i << 1 | 1];
  q[i << 1 | 0] = MulT(q[i], R, L.size());
  q[i << 1 | 1] = MulT(q[i], L, R.size());
 }
 for (int i = 0; i < n; i++) ans[i] = q[i + n][0];
 return ans.resize(x.size()), ans;
}
pair<S, S> DivMod(const S &A, const S &B) {
 assert(!B.empty() && B.back() != 0);
 if (A.size() < B.size()) return {{}, A};
 const auto sz = A.size() - B.size() + 1;
 S X = B; reverse(all(X)); X.resize(sz);
 S Y = A; reverse(all(Y)); Y.resize(sz);
 S Q = Mul(Inv(X), Y);
 Q.resize(sz); reverse(all(Q)); X = Mul(Q, B); Y = A;
 fi(0, Y.size()) Y[i] = sub(Y[i], X[i]);
 while (Y.size() && Y.back() == 0) Y.pop_back();
 while (Q.size() && Q.back() == 0) Q.pop_back();
 return {Q, Y};
} // empty means zero polynomial
int LinearRecursionKth(S a, S c, int64_t k) {
 const auto d = a.size(); assert(c.size() == d + 1);
 const auto sz = bit_ceil(2 * d + 1), o = sz / 2;
 S q = c; for (int &x: q) x = sub(0, x); q[0]=1;
 S p = Mul(a, q); p.resize(sz); q.resize(sz);
 for (int r; r = (k & 1), k; k >>= 1) {
  fill(d + all(p), 0); fill(d + 1 + all(q), 0);
  ntt(p.data(), sz); ntt(q.data(), sz);
  for (size_t i = 0; i < sz; i++)
   p[i] = mul(p[i], q[(i + o) & (sz - 1)]);
  for (size_t i = 0, j = o; j < sz; i++, j++)
   q[i] = q[j] = mul(q[i], q[j]);
  ntt(p.data(), sz, true); ntt(q.data(), sz, true);
  for (size_t i = 0; i < d; i++) p[i] = p[i << 1 | r];
  for (size_t i = 0; i <= d; i++) q[i] = q[i << 1];
 } // Bostan-Mori
 return mul(p[0], modinv(q[0]));
} // a_n = \sum c_j a_(n-j), c_0 is not used
```

5.17 Partition Number [9bb845]

```cpp
ans[0] = tmp[0] = 1;
for (int i = 1; i * i <= n; i++) {
 for (int rep = 0; rep < 2; rep++)
  for (int j = i; j <= n - i * i; j++)
   modadd(tmp[j], tmp[j-i]);
 for (int j = i * i; j <= n; j++)
  modadd(ans[j], tmp[j - i * i]);
}
```

5.18 Pi Count [715863]

```cpp
struct S { int rough; lld large; int id; };
lld PrimeCount(lld n) { // n ~ 10^13 => < 1s
 if (n <= 1) return 0;
 const int v = static_cast<int>(sqrtl(n)); int pc = 0;
 vector<int> smalls(v + 1), skip(v + 1); vector<S> z;
```

```
 for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
 for (int i : views::iota(0, (v + 1) / 2))
  z.emplace_back(2*i+1, (n / (2*i+1) + 1) / 2, i);
 for (int p = 3; p <= v; ++p)
  if (smalls[p] > smalls[p - 1]) {
  const int q = p * p; ++pc;
  if (1LL * q * q > n) break;
  skip[p] = 1;
  for (int i = q; i <= v; i += 2 * p) skip[i] = 1;
  int ns = 0;
  for (auto e : z) if (!skip[e.rough]) {
   lld d = 1LL * e.rough * p;
   e.large += pc - (d <= v ? z[smalls[d] - pc].large :
    smalls[n / d]);
   e.id = ns; z[ns++] = e;
  }
  z.resize(ns);
  for (int j = v / p; j >= p; --j) {
   int c = smalls[j] - pc, e = min(j * p + p, v + 1);
   for (int i = j * p; i < e; ++i) smalls[i] -= c;
  }
 }
 lld ans = z[0].large; z.erase(z.begin());
 for (auto &[rough, large, k] : z) {
  const lld m = n / rough; --k;
  ans -= large - (pc + k);
  for (auto [p, _, l] : z)
   if (l >= k || p * p > m) break;
   else ans += smalls[m / p] - (pc + l);
 }
 return ans;
} // test @ yosupo library checker w/ n=1e11, 68ms
```

5.19 Miller Rabin [fbd812]

```
bool isprime(llu x) {
 auto witn = [&](llu a, int t) {
  for (llu a2; t--; a = a2) {
   a2 = mmul(a, a, x);
   if (a2 == 1 && a != 1 && a != x - 1) return true;
  }
  return a != 1;
 };
 if (x <= 2 || ~x & 1) return x == 2;
 int t = countr_zero(x-1); llu odd = (x-1) >> t;
 for (llu m:
  {2, 325, 9375, 28178, 450775, 9780504, 1795265022})
  if (m % x != 0 && witn(mpow(m % x, odd, x), t))
   return false;
 return true;
} // test @ luogu 143 & yosupo judge, ~1700ms for Q=1e5
 // if use montgomery, ~250ms for Q=1e5
```

5.20 Pollard Rho [57ad88]

```
// does not work when n is prime or n == 1
// return any non-trivial factor
llu pollard_rho(llu n) {
 static mt19937_64 rnd(120821011);
 if (!(n & 1)) return 2;
 llu y = 2, z = y, c = rnd() % n, p = 1, i = 0, t;
 auto f = [&](llu x) {
  return madd(mmul(x, x, n), c, n); };
 do {
  p = mmul(msub(z = f(f(z)), y = f(y), n), p, n);
  if (++i &= 63) if (i == (i & -i)) t = gcd(p, n);
 } while (t == 1);
 return t == n ? pollard_rho(n) : t;
} // test @ yosupo judge, ~270ms for Q=100
 // if use montgomery, ~70ms for Q=100
```

5.21 Berlekamp Massey [a94d00]

```
template <typename T>
vector<T> BerlekampMassey(const vector<T> &output) {
 vector<T> d(output.size() + 1), me, he;
 for (size_t f = 0, i = 1; i <= output.size(); ++i) {
  for (size_t j = 0; j < me.size(); ++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.push_back(-k);
  for (T x : he) o.push_back(x * k);
  if (o.size() < me.size()) o.resize(me.size());
  for (size_t j = 0; j < me.size(); ++j) o[j] += me[j];
  if (i-f+he.size() >= me.size()) he = me, f = i;
  me = o;
 }
 return me;
}
```

5.22 Gauss Elimination [9dea40]

```
void gauss(vector<vector<llf>> A, vector<llf> b) {
 const int n = A.size(), m = A[0].size();
 for (int i = 0; i < m; ++i) {
  int p = -1;
  for (int j = i; j < n; ++j)
   if (abs(A[j][i]) > eps)
    if (p == -1 || abs(A[j][i]) > abs(A[p][i]))
     p = j;
  if (p == -1) continue;
  swap(A[p], A[i]);
  swap(b[p], b[i]);
  for (int j = 0; j < n; ++j) if (j != i) {
   llf z = A[j][i] / A[i][i];
   for (int k = 0; k < m; ++k)
    A[j][k] -= z * A[i][k];
   b[j] -= z * b[i];
  }
 }
}
```

5.23 Charateristic Polynomial [ff2159]

```
#define rep(x, y, z) for (int x=y; x<z; x++)
using VI = vector<int>; using VVI = vector<VI>;
void Hessenberg(VVI &H, int N) {
 for (int i = 0; i < N - 2; ++i) {
  for (int j = i + 1; j < N; ++j) if (H[j][i]) {
   rep(k, i, N) swap(H[i+1][k], H[j][k]);
   rep(k, 0, N) swap(H[k][i+1], H[k][j]);
   break;
  }
  if (!H[i + 1][i]) continue;
  for (int j = i + 2; j < N; ++j) {
   int co = mul(modinv(H[i + 1][i]), H[j][i]);
   rep(k, i, N) subeq(H[j][k], mul(H[i+1][k], co));
   rep(k, 0, N) addeq(H[k][i+1], mul(H[k][j], co));
  }
 }
}
VI CharacteristicPoly(VVI &A) {
 int N = (int)A.size(); Hessenberg(A, N);
 VVI P(N + 1, VI(N + 1)); P[0][0] = 1;
 for (int i = 1; i <= N; ++i) {
  rep(j, 0, i+1) P[i][j] = j ? P[i-1][j-1] : 0;
  for (int j = i - 1, val = 1; j >= 0; --j) {
   int co = mul(val, A[j][i - 1]);
   rep(k, 0, j+1) subeq(P[i][k], mul(P[j][k], co));
   if (j) val = mul(val, A[j][j - 1]);
  }
 }
 if (N & 1) for (int &x: P[N]) x = sub(0, x);
 return P[N]; // test: 2021 PTZ Korea K
}
```

5.24 Simplex [c9c93b]

```
namespace simplex {
// maximize c^Tx under Ax <= B and x >= 0
// return VD(n, -inf) if the solution doesn't exist
// return VD(n, +inf) if the solution is unbounded
using VD = vector<llf>;
using VVD = vector<vector<llf>>;
const llf eps = 1e-9, inf = 1e+9;
int n, m; VVD d; vector<int> p, q;
void pivot(int r, int s) {
 llf inv = 1.0 / d[r][s];
 for (int i = 0; i < m + 2; ++i)
  for (int j = 0; j < n + 2; ++j)
   if (i != r && j != s)
    d[i][j] -= d[r][j] * d[i][s] * inv;
 for(int i=0;i<m+2;++i) if (i != r) d[i][s] *= -inv;
 for(int j=0;j<n+2;++j) if (j != s) d[r][j] *= +inv;
 d[r][s] = inv; swap(p[r], q[s]);
```

```
}
bool phase(int z) {
 int x = m + z;
 while (true) {
  int s = -1;
  for (int i = 0; i <= n; ++i) {
   if (!z && q[i] == -1) continue;
   if (s == -1 || d[x][i] < d[x][s]) s = i;
  }
  if (s == -1 || d[x][s] > -eps) return true;
  int r = -1;
  for (int i = 0; i < m; ++i) {
   if (d[i][s] < eps) continue;
   if (r == -1 ||
    d[i][n+1]/d[i][s] < d[r][n+1]/d[r][s]) r = i;
  }
  if (r == -1) return false;
  pivot(r, s);
 }
}
VD solve(const VVD &a, const VD &b, const VD &c) {
 m = (int)b.size(), n = (int)c.size();
 d = VVD(m + 2, VD(n + 2));
 for (int i = 0; i < m; ++i)
  for (int j = 0; j < n; ++j) d[i][j] = a[i][j];
 p.resize(m), q.resize(n + 1);
 for (int i = 0; i < m; ++i)
  p[i] = n + i, d[i][n] = -1, d[i][n + 1] = b[i];
 for (int i = 0; i < n; ++i) q[i] = i,d[m][i] = -c[i];
 q[n] = -1, d[m + 1][n] = 1;
 int r = 0;
 for (int i = 1; i < m; ++i)
  if (d[i][n + 1] < d[r][n + 1]) r = i;
 if (d[r][n + 1] < -eps) {
  pivot(r, n);
  if (!phase(1) || d[m + 1][n + 1] < -eps)
   return VD(n, -inf);
  for (int i = 0; i < m; ++i) if (p[i] == -1) {
   int s = min_element(d[i].begin(), d[i].end() - 1)
       - d[i].begin();
   pivot(i, s);
  }
 }
 if (!phase(0)) return VD(n, inf);
 VD x(n);
 for (int i = 0; i < m; ++i)
  if (p[i] < n) x[p[i]] = d[i][n + 1];
 return x;
}}
```

5.25 Simplex Construction

Standard form: maximize $\sum_{1\le i\le n} c_i x_i$ such that for all $1 \le j \le m$, $\sum_{1\le i\le n} A_{ji}x_i \le b_j$.and $x_i \ge 0$ for all $1 \le i \le n$.

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 $x_i - x'_i$

5.26 Adaptive Simpson [09669e]

```
llf simp(llf l, llf r) {
 llf m = (l + r) / 2;
 return (f(l) + f(r) + 4.0 * f(m)) * (r - l) / 6.0;
}
llf F(llf L, llf R, llf v, llf eps) {
 llf M = (L + R) / 2, vl = simp(L, M), vr = simp(M, R);
 if (abs(vl + vr - v) <= 15 * eps)
  return vl + vr + (vl + vr - v) / 15.0;
 return F(L, M, vl, eps / 2.0) +
     F(M, R, vr, eps / 2.0);
} // call F(l, r, simp(l, r), 1e-6)
```

6 Geometry

6.1 Basic Geometry [f50abd]

```
#define IM imag
#define RE real
using lld = int64_t;
using llf = long double;
using PT = std::complex<lld>;
using PF = std::complex<llf>;
using P = PT;
llf abs(P p) { return sqrtl(norm(p)); }
PF toPF(PT p) { return PF{RE(p), IM(p)}; }
int sgn(lld x) { return (x > 0) - (x < 0); }
lld dot(P a, P b) { return RE(conj(a) * b); }
lld cross(P a, P b) { return IM(conj(a) * b); }
int ori(P a, P b, P c) {
 return sgn(cross(b - a, c - a));
}
int quad(P p) {
 return (IM(p) == 0) // use sgn for PF
  ? (RE(p) < 0 ? 3 : 1) : (IM(p) < 0 ? 0 : 2);
}
int argCmp(P a, P b) {
 // returns 0/+-1, starts from theta = -PI
 int qa = quad(a), qb = quad(b);
 if (qa != qb) return sgn(qa - qb);
 return sgn(cross(b, a));
}
P rot90(P p) { return P{-IM(p), RE(p)}; }
template <typename V> llf area(const V & pt) {
 lld ret = 0;
 for (int i = 1; i + 1 < (int)pt.size(); i++)
  ret += cross(pt[i] - pt[0], pt[i+1] - pt[0]);
 return ret / 2.0;
}
template <typename V> PF center(const V & pt) {
 P ret = 0; lld A = 0;
 for (int i = 1; i + 1 < (int)pt.size(); i++) {
  lld cur = cross(pt[i] - pt[0], pt[i+1] - pt[0]);
  ret += (pt[i] + pt[i + 1] + pt[0]) * cur; A += cur;
 }
 return toPF(ret) / llf(A * 3);
}
PF project(PF p, PF q) { // p onto q
 return dot(p, q) * q / dot(q, q); // dot<llf>
}
```

6.2 2D Convex Hull [ecba37]

```
// from NaCl, counterclockwise, be careful of n<=2
vector<P> convex_hull(vector<P> v) {
 sort(all(v)); // by X then Y
 if (v[0] == v.back()) return {v[0]};
 int t = 0, s = 1; vector<P> h(v.size() + 1);
 for (int _ = 2; _--; s = t--, reverse(all(v)))
  for (P p : v) {
   while (t>s && ori(p, h[t-1], h[t-2]) >= 0) t--;
   h[t++] = p;
  }
 return h.resize(t), h;
}
```

6.3 2D Farthest Pair [8b5844]

```
// p is CCW convex hull w/o colinear points
int n = (int)p.size(), pos = 1; lld ans = 0;
for (int i = 0; i < n; i++) {
 P e = p[(i + 1) % n] - p[i];
 while (cross(e, p[(pos + 1) % n] - p[i]) >
     cross(e, p[pos] - p[i]))
  pos = (pos + 1) % n;
 for (int j: {i, (i + 1) % n})
  ans = max(ans, norm(p[pos] - p[j]));
} // tested @ AOJ CGL_4_B
```

6.4 MinMax Enclosing Rect [e4470c]

```
// from 8BQube, plz ensure p is strict convex hull
const llf INF = 1e18, qi = acos(-1) / 2 * 3;
pair<llf, llf> solve(const vector<P> &p) {
 llf mx = 0, mn = INF; int n = (int)p.size();
 for (int i = 0, u = 1, r = 1, l = 1; i < n; ++i) {
#define Z(v) (p[(v) % n] - p[i])
  P e = Z(i + 1);
  while (cross(e, Z(u + 1)) > cross(e, Z(u))) ++u;
  while (dot(e, Z(r + 1)) > dot(e, Z(r))) ++r;
  if (!i) l = r + 1;
  while (dot(e, Z(l + 1)) < dot(e, Z(l))) ++l;
  P D = p[r % n] - p[l % n];
  llf H = cross(e, Z(u)) / llf(norm(e));
  mn = min(mn, dot(e, D) * H);
  llf B = sqrt(norm(D)) * sqrt(norm(Z(u)));
  llf deg = (qi - acos(dot(D, Z(u)) / B)) / 2;
  mx = max(mx, B * sin(deg) * sin(deg));
```

```
 }
 return {mn, mx};
} // test @ UVA 819
```

6.5 Minkowski Sum [602806]

```
// A, B are strict convex hull rotate to min by (X, Y)
vector<P> Minkowski(vector<P> A, vector<P> B) {
 const int N = (int)A.size(), M = (int)B.size();
 vector<P> sa(N), sb(M), C(N + M + 1);
 for (int i = 0; i < N; i++) sa[i] = A[(i+1)%N]-A[i];
 for (int i = 0; i < M; i++) sb[i] = B[(i+1)%M]-B[i];
 C[0] = A[0] + B[0];
 for (int i = 0, j = 0; i < N || j < M; ) {
  P e = (j>=M || (i<N && cross(sa[i], sb[j])>=0))
   ? sa[i++] : sb[j++];
  C[i + j] = e;
 }
 partial_sum(all(C), C.begin()); C.pop_back();
 return convex_hull(C); // just to remove colinear
}
```

6.6 Segment Intersection [60d016]

```
struct Seg { // closed segment
 P st, dir; // represent st + t*dir for 0<=t<=1
 Seg(P s, P e) : st(s), dir(e - s) {}
 static bool valid(lld p, lld q) {
  // is there t s.t. 0 <= t <= 1 && qt == p ?
  if (q < 0) q = -q, p = -p;
  return 0 <= p && p <= q;
 }
 vector<P> ends() const { return { st, st + dir }; }
};
template <typename T> bool isInter(T A, P p) {
 if (A.dir == P(0)) return p == A.st; // BE CAREFUL
 return cross(p - A.st, A.dir) == 0 &&
  T::valid(dot(p - A.st, A.dir), norm(A.dir));
}
template <typename U, typename V>
bool isInter(U A, V B) {
 if (cross(A.dir, B.dir) == 0) { // BE CAREFUL
  bool res = false;
  for (P p: A.ends()) res |= isInter(B, p);
  for (P p: B.ends()) res |= isInter(A, p);
  return res;
 }
 P D = B.st - A.st; lld C = cross(A.dir, B.dir);
 return U::valid(cross(D, B.dir), C) &&
  V::valid(cross(D, A.dir), C);
}
```

6.7 Half Plane Intersection [31e216]

```
struct Line {
 P st, ed, dir;
 Line (P s, P e) : st(s), ed(e), dir(e - s) {}
}; using LN = const Line &;
PF intersect(LN A, LN B) {
 llf t = cross(B.st - A.st, B.dir) /
  llf(cross(A.dir, B.dir));
 return toPF(A.st) + toPF(A.dir) * t; // C^3 / C^2
}
bool cov(LN l, LN A, LN B) {
 i128 u = cross(B.st-A.st, B.dir);
 i128 v = cross(A.dir, B.dir);
 // ori(l.st, l.ed, A.st + A.dir*(u/v)) <= 0?
 i128 x = RE(A.dir) * u + RE(A.st - l.st) * v;
 i128 y = IM(A.dir) * u + IM(A.st - l.st) * v;
 return sgn(x*IM(l.dir) - y*RE(l.dir)) * sgn(v) >= 0;
} // x, y are C^3, also sgn<i128> is needed
bool operator<(LN a, LN b) {
 if (int c = argCmp(a.dir, b.dir)) return c == -1;
 return ori(a.st, a.ed, b.st) < 0;
}
// cross(pt-line.st, line.dir)<=0 <-> pt in half plane
// the half plane is the LHS when going from st to ed
llf HPI(vector<Line> &q) {
 sort(q.begin(), q.end());
 int n = (int)q.size(), l = 0, r = -1;
 for (int i = 0; i < n; i++) {
  if (i && !argCmp(q[i].dir, q[i-1].dir)) continue;
  while (l < r && cov(q[i], q[r-1], q[r])) --r;
  while (l < r && cov(q[i], q[l], q[l+1])) ++l;
  q[++r] = q[i];
 }
 while (l < r && cov(q[l], q[r-1], q[r])) --r;
 while (l < r && cov(q[r], q[l], q[l+1])) ++l;
 n = r - l + 1; // q[l .. r] are the lines
 if (n <= 1 || !argCmp(q[l].dir, q[r].dir)) return 0;
 vector<PF> pt(n);
 for (int i = 0; i < n; i++)
  pt[i] = intersect(q[i+l], q[(i+1)%n+l]);
 return area(pt);
} // test @ 2020 Nordic NCPC : BigBrother
```

6.8 SegmentDist (Sausage) [9d8603]

```
// be careful of abs<complex<int>> (replace _abs below)
llf PointSegDist(P A, Seg B) {
 if (B.dir == P(0)) return _abs(A - B.st);
 if (sgn(dot(A - B.st, B.dir)) *
   sgn(dot(A - B.ed, B.dir)) <= 0)
  return abs(cross(A - B.st, B.dir)) / _abs(B.dir);
 return min(_abs(A - B.st), _abs(A - B.ed));
}
llf SegSegDist(const Seg &s1, const Seg &s2) {
 if (isInter(s1, s2)) return 0;
 return min({
   PointSegDist(s1.st, s2),
   PointSegDist(s1.ed, s2),
   PointSegDist(s2.st, s1),
   PointSegDist(s2.ed, s1) });
} // test @ QOJ2444 / PTZ19 Summer.D3
```

6.9 Rotating Sweep Line [8aff27]

```
struct Event {
 P d; int u, v;
 bool operator<(const Event &b) const {
  return sgn(cross(d, b.d)) > 0; }
};
P makePositive(P z) { return cmpxy(z, 0) ? -z : z; }
void rotatingSweepLine(const vector<P> &p) {
 const int n = int(p.size());
 vector<Event> e; e.reserve(n * (n - 1) / 2);
 for (int i = 0; i < n; i++)
  for (int j = i + 1; j < n; j++)
   e.emplace_back(makePositive(p[i] - p[j]), i, j);
 sort(all(e));
 vector<int> ord(n), pos(n);
 iota(all(ord), 0);
 sort(all(ord), [&p](int i, int j) {
  return cmpxy(p[i], p[j]); });
 for (int i = 0; i < n; i++) pos[ord[i]] = i;
 const auto makeReverse = [](auto &v) {
  sort(all(v)); v.erase(unique(all(v)), v.end());
  vector<pair<int,int>> segs;
  for (size_t i = 0, j = 0; i < v.size(); i = j) {
   for (; j < v.size() && v[j] - v[i] <= j - i; j++);
   segs.emplace_back(v[i], v[j - 1] + 1 + 1);
  }
  return segs;
 };
 for (size_t i = 0, j = 0; i < e.size(); i = j) {
  /* do here */
  vector<size_t> tmp;
  for (; j < e.size() && !(e[i] < e[j]); j++)
   tmp.push_back(min(pos[e[j].u], pos[e[j].v]));
  for (auto [l, r] : makeReverse(tmp)) {
   reverse(ord.begin() + l, ord.begin() + r);
   for (int t = l; t < r; t++) pos[ord[t]] = t;
  }
 }
}
```

6.10 Polygon Cut [fdd064]

```
using P = PF;
vector<P> cut(const vector<P>& poly, P s, P e) {
 vector<P> res;
 for (size_t i = 0; i < poly.size(); i++) {
  P cur = poly[i], prv = i ? poly[i-1] : poly.back();
  bool side = ori(s, e, cur) < 0;
  if (side != (ori(s, e, prv) < 0))
   res.push_back(intersect({s, e}, {cur, prv}));
  if (side)
   res.push_back(cur);
 }
 return res;
}
```

6.11 Point In Simple Polygon [037c52]

```
bool PIP(const vector<P> &p, P z, bool strict = true) {
 int cnt = 0, n = (int)p.size();
 for (int i = 0; i < n; i++) {
  P A = p[i], B = p[(i + 1) % n];
  if (isInter(Seg(A, B), z)) return !strict;
  auto zy = IM(z), Ay = IM(A), By = IM(B);
  cnt ^= ((zy<Ay) - (zy<By)) * ori(z, A, B) > 0;
 }
 return cnt;
}
```

6.12 Point In Hull (Fast) [060ba1]

```
bool PIH(const vector<P> &h, P z, bool strict = true) {
 int n = (int)h.size(), a = 1, b = n - 1, r = !strict;
 if (n < 3) return r && isInter(Seg(h[0], h[n-1]), z);
 if (ori(h[0],h[a],h[b]) > 0) swap(a, b);
 if (ori(h[0],h[a],z) >= r || ori(h[0],h[b],z) <= -r)
  return false;
 while (abs(a - b) > 1) {
  int c = (a + b) / 2;
  (ori(h[0], h[c], z) > 0 ? b : a) = c;
 }
 return ori(h[a], h[b], z) < r;
}
```

6.13 Point In Polygon (Fast) [71725b]

```
vector<int> PIPfast(vector<P> p, vector<P> q) {
 const int N = int(p.size()), Q = int(q.size());
 vector<pair<P, int>> evt;
 vector<Seg> edge;
 for (int i = 0; i < N; i++) {
  int a = i, b = (i + 1) % N;
  P A = p[a], B = p[b];
  assert (A < B || B < A); // std::operator<
  if (B < A) swap(A, B);
  evt.emplace_back(A, i);
  evt.emplace_back(B, ~i);
  edge.emplace_back(A, B);
 }
 for (int i = 0; i < Q; i++)
  evt.emplace_back(q[i], i + N);
 sort(all(evt));
 auto vtx = p; sort(all(vtx));
 auto eval = [](const Seg &a, lld x) -> llf {
  if (RE(a.dir) == 0) {
   assert (x == RE(a.st));
   return IM(a.st) + llf(IM(a.dir)) / 2;
  }
  llf t = (x - RE(a.st)) / llf(RE(a.dir));
  return IM(a.st) + IM(a.dir) * t;
 };
 lld cur_x = 0;
 auto cmp = [&](const Seg &a, const Seg &b) -> bool {
  if (int s = sgn(eval(a, cur_x) - eval(b, cur_x)))
   return s == -1;
  int s = sgn(cross(b.dir, a.dir));
  if (cur_x != RE(a.st) && cur_x != RE(b.st)) s *= -1;
  return s == -1;
 };
 namespace pbds = __gnu_pbds;
 using Tree = pbds::tree<Seg, int, decltype(cmp),
     pbds::rb_tree_tag,
     pbds::tree_order_statistics_node_update>;
 Tree st(cmp);
 vector<int> ans(Q);
 for (auto [ep, i] : evt) {
  cur_x = RE(ep);
  if (i < 0) { // remove
   st.erase(edge[~i]);
  } else if (i < N) { // insert
   auto [it, succ] = st.insert({edge[i], i});
   assert (succ);
  } else {
   int qid = i - N;
   if (binary_search(all(vtx), ep)) { // on vertex
    ans[qid] = 1;
    continue;
   }
   Seg H(ep, ep); // ??
   auto it = st.lower_bound(H);
   if (it != st.end() && isInter(it->first, ep)) {
    ans[qid] = 1; // on edge
    continue;
   }
   if (it != st.begin() && isInter(prev(it)->first, ep)
    ) {
    ans[qid] = 1; // on edge
    continue;
   }
   auto rk = st.order_of_key(H);
   if (rk % 2 == 0) ans[qid] = 0; // outside
   else ans[qid] = 2; // inside
  }
 }
 return ans;
} // test @ AOJ CGL_3_C
```

6.14 Tangent of Points To Hull [6d7cd7]

```
pair<int, int> get_tangent(const vector<P> &v, P p) {
 const auto gao = [&, N = int(v.size())](int s) {
  const auto lt = [&](int x, int y) {
   return ori(p, v[x % N], v[y % N]) == s; };
  int l = 0, r = N; bool up = lt(0, 1);
  while (r - l > 1) {
   int m = (l + r) / 2;
   if (lt(m, 0) ? up : !lt(m, m+1)) r = m;
   else l = m;
  }
  return (lt(l, r) ? r : l) % N;
 }; // test @ codeforces.com/gym/101201/problem/E
 return {gao(-1), gao(1)}; // (a,b):ori(p,v[a],v[b])<0
} // plz ensure that point strictly out of hull
```

6.15 Circle Class & Intersection [d5df51]

```
llf FMOD(llf x) {
 if (x < -PI) x += PI * 2;
 if (x > PI) x -= PI * 2;
 return x;
}
struct Cir { PF o; llf r; };
// be careful when tangent
vector<llf> intersectAngle(Cir a, Cir b) {
 PF dir = b.o - a.o; llf d2 = norm(dir);
 if (norm(a.r - b.r) >= d2) { // norm(x) := |x|^2
  if (a.r < b.r) return {-PI, PI}; // a in b
  else return {}; // b in a
 } else if (norm(a.r + b.r) <= d2) return {};
 llf dis = abs(dir), theta = arg(dir);
 llf phi = acos((a.r * a.r + d2 - b.r * b.r) /
   (2 * a.r * dis)); // is acos_safe needed ?
 llf L = FMOD(theta - phi), R = FMOD(theta + phi);
 return { L, R };
}
vector<PF> intersectPoint(Cir a, Cir b) {
 llf d = abs(a.o - b.o);
 if (d > b.r+a.r || d < abs(b.r-a.r)) return {};
 llf dt = (b.r*b.r - a.r*a.r)/d, d1 = (d+dt)/2;
 PF dir = (a.o - b.o) / d;
 PF u = dir * d1 + b.o;
 PF v = rot90(dir) * sqrt(max(0.0L, b.r*b.r-d1*d1));
 return {u + v, u - v};
} // test @ AOJ CGL probs
```

6.16 Circle Common Tangent [d97f1c]

```
// be careful of tangent / exact same circle
// sign1 = 1 for outer tang, -1 for inter tang
vector<Line> common_tan(const Cir &a, const Cir &b, int
     sign1) {
 if (norm(a.o - b.o) < eps) return {};
 llf d = abs(a.o - b.o), c = (a.r - sign1 * b.r) / d;
 PF v = (b.o - a.o) / d;
 if (c * c > 1) return {};
 if (abs(c * c - 1) < eps) {
  PF p = a.o + c * v * a.r;
  return {Line(p, p + rot90(b.o - a.o))};
 }
 vector<Line> ret; llf h = sqrt(max(0.0L, 1-c*c));
 for (int sign2 : {1, -1}) {
  PF n = c * v + sign2 * h * rot90(v);
  PF p1 = a.o + n * a.r;
  PF p2 = b.o + n * (b.r * sign1);
  ret.emplace_back(p1, p2);
 }
```

```
 return ret;
}
```

6.17 Line-Circle Intersection [10786a]

```
vector<PF> LineCircleInter(PF p1, PF p2, PF o, llf r) {
 PF ft = p1 + project(o-p1, p2-p1), vec = p2-p1;
 llf dis = abs(o - ft);
 if (abs(dis - r) < eps) return {ft};
 if (dis > r) return {};
 vec = vec * sqrt(r * r - dis * dis) / abs(vec);
 return {ft + vec, ft - vec}; // sqrt_safe?
}
```

6.18 Poly-Circle Intersection [8e5133]

```
// Divides into multiple triangle, and sum up
// from 8BQube, test by HDU2892 & AOJ CGL_7_H
llf _area(PF pa, PF pb, llf r) {
 if (abs(pa) < abs(pb)) swap(pa, pb);
 if (abs(pb) < eps) return 0;
 llf S, h, theta;
 llf a = abs(pb), b = abs(pa), c = abs(pb - pa);
 llf cB = dot(pb, pb-pa) / a / c, B = acos_safe(cB);
 llf cC = dot(pa, pb) / a / b, C = acos_safe(cC);
 if (a > r) {
  S = (C / 2) * r * r; h = a * b * sin(C) / c;
  if (h < r && B < PI / 2)
   S -= (acos_safe(h/r)*r*r - h*sqrt_safe(r*r-h*h));
 } else if (b > r) {
  theta = PI - B - asin_safe(sin(B) / r * a);
  S = 0.5 * a*r*sin(theta) + (C-theta)/2 * r * r;
 } else
  S = 0.5 * sin(C) * a * b;
 return S;
}
llf area_poly_circle(const vector<PF> &v, PF O, llf r)
    {
 llf S = 0;
 for (size_t i = 0, N = v.size(); i < N; ++i)
  S += _area(v[i] - O, v[(i + 1) % N] - O, r) *
     ori(O, v[i], v[(i + 1) % N]);
 return abs(S);
}
```

6.19 Minimum Covering Circle [92bb15]

```
Cir getCircum(P a, P b, P c){ // P = complex<llf>
 P z1 = a - b, z2 = a - c; llf D = cross(z1, z2) * 2;
 auto c1 = dot(a + b, z1), c2 = dot(a + c, z2);
 P o = rot90(c2 * z1 - c1 * z2) / D;
 return { o, abs(o - a) };
}
Cir minCircleCover(vector<P> p) {
 assert (!p.empty());
 ranges::shuffle(p, mt19937(114514));
 Cir c = { 0, 0 };
 for(size_t i = 0; i < p.size(); i++) {
  if (abs(p[i] - c.o) <= c.r) continue;
  c = { p[i], 0 };
  for (size_t j = 0; j < i; j++) {
   if (abs(p[j] - c.o) <= c.r) continue;
   c.o = (p[i] + p[j]) / llf(2);
   c.r = abs(p[i] - c.o);
   for (size_t k = 0; k < j; k++) {
    if (abs(p[k] - c.o) <= c.r) continue;
    c = getCircum(p[i], p[j], p[k]);
   }
  }
 }
 return c;
} // test @ TIOJ 1093 & luogu P1742
```

6.20 Circle Union [073c1c]

```
#define eb emplace_back
struct Teve { // test@SPOJ N=1000, 0.3~0.5s
 PF p; llf a; int add; // point, ang, add
 Teve(PF x, llf y, int z) : p(x), a(y), add(z) {}
 bool operator<(Teve &b) const { return a < b.a; }
};
// strict: x = 0, otherwise x = -1
bool disjunct(Cir &a, Cir &b, int x)
{ return sgn(abs(a.o - b.o) - a.r - b.r) > x; }
bool contain(Cir &a, Cir &b, int x)
{ return sgn(a.r - b.r - abs(a.o - b.o)) > x; }
vector<llf> CircleUnion(vector<Cir> &c) {
 // area[i] : area covered by at least i circles
 int N = (int)c.size(); vector<llf> area(N + 1);
 vector<vector<int>> overlap(N, vector<int>(N));
 auto g = overlap; // use simple 2darray to speedup
 for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j) {
   /* c[j] is non-strictly in c[i]. */
   overlap[i][j] = i != j &&
     (sgn(c[i].r - c[j].r) > 0 ||
      (sgn(c[i].r - c[j].r) == 0 && i < j)) &&
     contain(c[i], c[j], -1);
  }
 for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
   g[i][j] = i != j && !(overlap[i][j] ||
     overlap[j][i] || disjunct(c[i], c[j], -1));
 for (int i = 0; i < N; ++i) {
  vector<Teve> eve; int cnt = 1;
  for (int j = 0; j < N; ++j) cnt += overlap[j][i];
  // if (cnt > 1) continue; (if only need area[1])
  for (int j = 0; j < N; ++j) if (g[i][j]) {
   auto IP = intersectPoint(c[i], c[j]);
   PF aa = IP[1], bb = IP[0];
   llf A = arg(aa - c[i].o), B = arg(bb - c[i].o);
   eve.eb(bb, B, 1); eve.eb(aa, A, -1);
   if (B > A) ++cnt;
  }
  if (eve.empty()) area[cnt] += PI*c[i].r*c[i].r;
  else {
   sort(eve.begin(), eve.end());
   eve.eb(eve[0]); eve.back().a += PI * 2;
   for (size_t j = 0; j + 1 < eve.size(); j++) {
    cnt += eve[j].add;
    area[cnt] += cross(eve[j].p, eve[j+1].p) *.5;
    llf t = eve[j + 1].a - eve[j].a;
    area[cnt] += (t-sin(t)) * c[i].r * c[i].r *.5;
   }
  }
 }
 return area;
}
```

6.21 Polygon Union [2bff43]

```
llf rat(P a, P b) { return sgn(RE(b)) ? llf(RE(a))/RE(b
    ) : llf(IM(a))/IM(b); }
llf polyUnion(vector<vector<P>>& poly) {
 llf ret = 0; // area of poly[i] must be non-negative
 rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
  P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
  vector<pair<llf, int>> segs{{0, 0}, {1, 0}};
  rep(j,0,sz(poly)) if (i != j) {
   rep(u,0,sz(poly[j])) {
    P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])
    ];
    if (int sc = ori(A, B, C), sd = ori(A, B, D); sc !=
     sd) {
     llf sa = cross(D-C, A-C), sb = cross(D-C, B-C);
     if (min(sc, sd) < 0)
      segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
    } else if (!sc && !sd && j<i && sgn(dot(B-A,D-C))
    >0){
     segs.emplace_back(rat(C - A, B - A), 1);
     segs.emplace_back(rat(D - A, B - A), -1);
    }
   }
  }
  sort(segs.begin(), segs.end());
  for (auto &s : segs) s.first = clamp<llf>(s.first, 0,
     1);
  llf sum = 0;
  int cnt = segs[0].second;
  rep(j,1,sz(segs)) {
   if (!cnt) sum += segs[j].first - segs[j - 1].first;
   cnt += segs[j].second;
  }
  ret += cross(A,B) * sum;
 }
 return ret / 2;
}
```

6.22 3D Point [46b73b]

```
struct P3 {
 lld x, y, z;
 P3 operator^(const P3 &b) const {
  return {y*b.z-b.y*z, z*b.x-b.z*x, x*b.y-b.x*y};
 }
 //Azimuthal angle (longitude) to x-axis. \in [-pi, pi]
 llf phi() const { return atan2(y, x); }
 //Zenith angle (latitude) to the z-axis. \in [0, pi]
 llf theta() const { return atan2(sqrt(x*x+y*y),z); }
};
P3 ver(P3 a, P3 b, P3 c) { return (b - a) ^ (c - a); }
lld volume(P3 a, P3 b, P3 c, P3 d) {
 return dot(ver(a, b, c), d - a);
}
P3 rotate_around(P3 p, llf angle, P3 axis) {
 llf s = sin(angle), c = cos(angle);
 P3 u = normalize(axis);
 return u*dot(u, p)*(1-c) + p * c + cross(u, p)*s;
}
```

6.23 3D Convex Hull [01652a]

```
struct Face {
 int a, b, c;
 Face(int ta, int tb, int tc) : a(ta), b(tb), c(tc) {}
};
auto preprocess(const vector<P3> &pt) {
 auto G = pt.begin();
 auto a = find_if(all(pt), [&](P3 z) {
  return z != *G; }) - G;
 auto b = find_if(all(pt), [&](P3 z) {
  return ver(*G, pt[a], z) != P3(0, 0, 0); }) - G;
 auto c = find_if(all(pt), [&](P3 z) {
  return volume(*G, pt[a], pt[b], z) != 0; }) - G;
 vector<size_t> id;
 for (size_t i = 0; i < pt.size(); i++)
  if (i != a && i != b && i != c) id.push_back(i);
 return tuple{a, b, c, id};
}
// return the faces with pt indexes
// all points coplanar case will WA
vector<Face> convex_hull_3D(const vector<P3> &pt) {
 const int n = int(pt.size());
 if (n <= 3) return {}; // be careful about edge case
 vector<Face> now;
 vector<vector<int>> z(n, vector<int>(n));
 auto [a, b, c, ord] = preprocess(pt);
 now.emplace_back(a, b, c); now.emplace_back(c, b, a);
 for (auto i : ord) {
  vector<Face> next;
  for (const auto &f : now) {
   lld v = volume(pt[f.a], pt[f.b], pt[f.c], pt[i]);
   if (v <= 0) next.push_back(f);
   z[f.a][f.b] = z[f.b][f.c] = z[f.c][f.a] = sgn(v);
  }
  const auto F = [&](int x, int y) {
   if (z[x][y] > 0 && z[y][x] <= 0)
    next.emplace_back(x, y, i);
  };
  for (const auto &f : now)
   F(f.a, f.b), F(f.b, f.c), F(f.c, f.a);
  now = next;
 }
 return now;
}
// n^2 delaunay: facets with negative z normal of
// convexhull of (x, y, x^2 + y^2), use a pseudo-point
// (0, 0, inf) to avoid degenerate case
// test @ SPOJ CH3D
// llf area = 0, vol = 0; // surface area / volume
// for (auto [a, b, c]: faces)
//  area += abs(ver(p[a], p[b], p[c]))/2.0,
//  vol += volume(P3(0, 0, 0), p[a], p[b], p[c])/6.0;
```

6.24 3D Projection [68f350]

```
using P3F = valarray<llf>;
P3F toP3F(P3 p) { return {p.x, p.y, p.z}; }
llf dot(P3F a, P3F b) {
 return a[0]*b[0]+a[1]*b[1]+a[2]*b[2];
}
P3F housev(P3 A, P3 B, int s) {
 const llf a = abs(A), b = abs(B);
 return toP3F(A) / a + s * toP3F(B) / b;
}
P project(P3 p, P3 q) {
 P3 o(0, 0, 1);
 P3F u = housev(q, o, q.z > 0 ? 1 : -1);
 auto pf = toP3F(p);
 auto np = pf - 2 * u * dot(u, pf) / dot(u, u);
 return P(np[0], np[1]);
} // project p onto the plane q^Tx = 0
```

6.25 3D Skew Line Nearest Point

- $L_1 : \boldsymbol{v}_1 = \boldsymbol{p}_1 + t_1\boldsymbol{d}_1, L_2 : \boldsymbol{v}_2 = \boldsymbol{p}_2 + t_2\boldsymbol{d}_2$
- $\boldsymbol{n} = \boldsymbol{d}_1 \times \boldsymbol{d}_2$
- $\boldsymbol{n}_1 = \boldsymbol{d}_1 \times \boldsymbol{n}, \boldsymbol{n}_2 = \boldsymbol{d}_2 \times \boldsymbol{n}$
- $\boldsymbol{c}_1 = \boldsymbol{p}_1 + \frac{(\boldsymbol{p}_2-\boldsymbol{p}_1)\cdot\boldsymbol{n}_2}{\boldsymbol{d}_1\cdot\boldsymbol{n}_2}\boldsymbol{d}_1, \boldsymbol{c}_2 = \boldsymbol{p}_2 + \frac{(\boldsymbol{p}_1-\boldsymbol{p}_2)\cdot\boldsymbol{n}_1}{\boldsymbol{d}_2\cdot\boldsymbol{n}_1}\boldsymbol{d}_2$

6.26 Delaunay [3a4ff1]

```
/* please ensure input points are unique */
/* A triangulation such that no points will strictly
inside circumcircle of any triangle. C should be big
enough s.t. the initial triangle contains all points */
#define L(i) ((i)==0 ? 2 : (i)-1)
#define R(i) ((i)==2 ? 0 : (i)+1)
#define F3 for (int i = 0; i < 3; i++)
bool is_inf(P z) { return RE(z) <= -C || RE(z) >= C; }
bool in_cc(const array<P,3> &p, P q) {
 i128 inf_det = 0, det = 0, inf_N, N;
 F3 {
  if (is_inf(p[i]) && is_inf(q)) continue;
  else if (is_inf(p[i])) inf_N = 1, N = -norm(q);
  else if (is_inf(q)) inf_N = -1, N = norm(p[i]);
  else inf_N = 0, N = norm(p[i]) - norm(q);
  lld D = cross(p[R(i)] - q, p[L(i)] - q);
  inf_det += inf_N * D; det += N * D;
 }
 return inf_det != 0 ? inf_det > 0 : det > 0;
}
P v[maxn];
struct Tri;
struct E {
 Tri *t; int side;
 E(Tri *t_=0, int side_=0) : t(t_), side(side_) {}
};
struct Tri {
 array<int,3> p; array<Tri*,3> ch; array<E,3> e;
 Tri(int a=0, int b=0, int c=0) : p{a, b, c}, ch{} {}
 bool has_chd() const { return ch[0] != nullptr; }
 bool contains(int q) const {
  F3 if (ori(v[p[i]], v[p[R(i)]], v[q]) < 0)
   return false;
  return true;
 }
 bool check(int q) const {
  return in_cc({v[p[0]], v[p[1]], v[p[2]]}, v[q]); }
} pool[maxn * 10], *it, *root;
void link(const E &a, const E &b) {
 if (a.t) a.t->e[a.side] = b;
 if (b.t) b.t->e[b.side] = a;
}
void flip(Tri *A, int a) {
 auto [B, b] = A->e[a]; /* flip edge between A,B */
 if (!B || !A->check(B->p[b])) return;
 Tri *X = new (it++) Tri(A->p[R(a)], B->p[b], A->p[a]);
 Tri *Y = new (it++) Tri(B->p[R(b)], A->p[a], B->p[b]);
 link(E(X, 0), E(Y, 0));
 link(E(X, 1), A->e[L(a)]); link(E(X, 2), B->e[R(b)]);
 link(E(Y, 1), B->e[L(b)]); link(E(Y, 2), A->e[R(a)]);
 A->ch = B->ch = {X, Y, nullptr};
 flip(X, 1); flip(X, 2); flip(Y, 1); flip(Y, 2);
}
void add_point(int p) {
 Tri *r = root;
 while (r->has_chd()) for (Tri *c: r->ch)
  if (c && c->contains(p)) { r = c; break; }
 array<Tri*, 3> t; /* split into 3 triangles */
 F3 t[i] = new (it++) Tri(r->p[i], r->p[R(i)], p);
 F3 link(E(t[i], 0), E(t[R(i)], 1));
 F3 link(E(t[i], 2), r->e[L(i)]);
 r->ch = t;
 F3 flip(t[i], 2);
}
auto build(const vector<P> &p) {
 it = pool; int n = (int)p.size();
```

```
 vector<int> ord(n); iota(all(ord), 0);
 shuffle(all(ord), mt19937(114514));
 root = new (it++) Tri(n, n + 1, n + 2);
 copy_n(p.data(), n, v); v[n++] = P(-C, -C);
 v[n++] = P(C * 2, -C); v[n++] = P(-C, C * 2);
 for (int i : ord) add_point(i);
 vector<array<int, 3>> res;
 for (Tri *now = pool; now != it; now++)
  if (!now->has_chd()) res.push_back(now->p);
 return res;
}
```

6.27 Build Voronoi [94f000]

```
void build_voronoi_cells(auto &&p, auto &&res) {
 vector<vector<int>> adj(p.size());
 for (auto f: res) F3 {
  int a = f[i], b = f[R(i)];
  if (a >= p.size() || b >= p.size()) continue;
  adj[a].emplace_back(b);
 }
 // use `adj` and `p` and HPI to build cells
 for (size_t i = 0; i < p.size(); i++) {
  vector<Line> ls = frame; // the frame
  for (int j : adj[i]) {
   P m = p[i] + p[j], d = rot90(p[j] - p[i]);
   assert (norm(d) != 0);
   ls.emplace_back(m, m + d); // doubled coordinate
  } // HPI(ls)
 }
}
```

6.28 kd Tree (Nearest Point) [dbade8]

```
struct KDTree {
 struct Node {
  int x, y, x1, y1, x2, y2, id, f; Node *L, *R;
 } tree[maxn], *root;
 lld dis2(int x1, int y1, int x2, int y2) {
  lld dx = x1 - x2, dy = y1 - y2;
  return dx * dx + dy * dy;
 }
 static bool cmpx(Node& a, Node& b){return a.x<b.x;}
 static bool cmpy(Node& a, Node& b){return a.y<b.y;}
 void init(vector<pair<int,int>> &ip) {
  const int n = ip.size();
  for (int i = 0; i < n; i++) {
   tree[i].id = i;
   tree[i].x = ip[i].first;
   tree[i].y = ip[i].second;
  }
  root = build(0, n-1, 0);
 }
 Node* build(int L, int R, int d) {
  if (L>R) return nullptr; int M = (L+R)/2;
  nth_element(tree+L,tree+M,tree+R+1,d%2?cmpy:cmpx);
  Node &o = tree[M]; o.f = d % 2;
  o.x1 = o.x2 = o.x; o.y1 = o.y1 = o.y;
  o.L = build(L, M-1, d+1); o.R = build(M+1, R, d+1);
  for (Node *s: {o.L, o.R}) if (s) {
   o.x1 = min(o.x1, s->x1); o.x2 = max(o.x2, s->x2);
   o.y1 = min(o.y1, s->y1); o.y2 = max(o.y2, s->y2);
  }
  return tree+M;
 }
 bool touch(int x, int y, lld d2, Node *r){
  lld d = sqrt(d2)+1;
  return x >= r->x1 - d && x <= r->x2 + d &&
         y >= r->y1 - d && y <= r->y2 + d;
 }
 using P = pair<lld, int>;
 void dfs(int x, int y, P &mn, Node *r) {
  if (!r || !touch(x, y, mn.first, r)) return;
  mn = min(mn, P(dis2(r->x, r->y, x, y), r->id));
  if (r->f == 1 ? y < r->y : x < r->x)
   dfs(x, y, mn, r->L), dfs(x, y, mn, r->R);
  else
   dfs(x, y, mn, r->R), dfs(x, y, mn, r->L);
 }
 int query(int x, int y) {
  P mn(INF, -1); dfs(x, y, mn, root);
  return mn.second;
 }
} tree;
```

6.29 kd Closest Pair (3D ver.) [84d9eb]

```
llf solve(vector<P> v) {
 shuffle(v.begin(), v.end(), mt19937());
 unordered_map<lld, unordered_map<lld,
  unordered_map<lld, int>>> m;
 llf d = dis(v[0], v[1]);
 auto Idx = [&d] (llf x) -> lld {
  return round(x * 2 / d) + 0.1; };
 auto rebuild_m = [&m, &v, &Idx](int k) {
  m.clear();
  for (int i = 0; i < k; ++i)
   m[Idx(v[i].x)][Idx(v[i].y)]
    [Idx(v[i].z)] = i;
 }; rebuild_m(2);
 for (size_t i = 2; i < v.size(); ++i) {
  const lld kx = Idx(v[i].x), ky = Idx(v[i].y),
     kz = Idx(v[i].z); bool found = false;
  for (int dx = -2; dx <= 2; ++dx) {
   const lld nx = dx + kx;
   if (m.find(nx) == m.end()) continue;
   auto& mm = m[nx];
   for (int dy = -2; dy <= 2; ++dy) {
    const lld ny = dy + ky;
    if (mm.find(ny) == mm.end()) continue;
    auto& mmm = mm[ny];
    for (int dz = -2; dz <= 2; ++dz) {
     const lld nz = dz + kz;
     if (mmm.find(nz) == mmm.end()) continue;
     const int p = mmm[nz];
     if (dis(v[p], v[i]) < d) {
      d = dis(v[p], v[i]);
      found = true;
     }
    }
   }
  }
  if (found) rebuild_m(i + 1);
  else m[kx][ky][kz] = i;
 }
 return d;
}
```

6.30 Simulated Annealing [4e0fe5]

```
llf anneal() {
 mt19937 rnd_engine(seed);
 uniform_real_distribution<llf> rnd(0, 1);
 const llf dT = 0.001;
 // Argument p
 llf S_cur = calc(p), S_best = S_cur;
 for (llf T = 2000; T > EPS; T -= dT) {
  // Modify p to p_prime
  const llf S_prime = calc(p_prime);
  const llf delta_c = S_prime - S_cur;
  llf prob = min((llf)1, exp(-delta_c / T));
  if (rnd(rnd_engine) <= prob)
   S_cur = S_prime, p = p_prime;
  if (S_prime < S_best) // find min
   S_best = S_prime, p_best = p_prime;
 }
 return S_best;
}
```

6.31 Triangle Centers [adb146]

```
O = ... // see min circle cover
G = (A + B + C) / 3;
H = G * 3 - O * 2; // orthogonal center
llf a = abs(B - C), b = abs(A - C), c = abs(A - B);
I = (a * A + b * B + c * C) / (a + b + c);
// FermatPoint: minimizes sum of distance
// if max. angle >= 120 deg then vertex
// otherwise, make eq. triangle AB'C, CA'B, BC'A
// line AA', BB', CC' intersects at P
```

7 Stringology

7.1 Hash [ce7fad]

```
template <int P = 127, int Q = 1051762951>
class Hash {
 vector<int> h, p;
public:
 Hash(const auto &s) : h(s.size()+1), p(s.size()+1) {
  for (size_t i = 0; i < s.size(); ++i)
   h[i + 1] = add(mul(h[i], P), s[i]);
```

```
  generate(all(p), [x = 1, y = 1, this]() mutable {
   return y = x, x = mul(x, P), y; });
 }
 int query(int l, int r) const { // 1-base (l, r]
  return sub(h[r], mul(h[l], p[r - l]));
 }
};
```

7.2 Suffix Array [216598]

```
auto sais(const auto &s) {
 const int n = (int)s.size(), z = ranges::max(s) + 1;
 vector<int> c(z); for (int x : s) ++c[x];
 partial_sum(all(c), begin(c));
 vector<int> sa(n); auto I = ranges::iota_view(0, n);
 if (ranges::max(c) <= 1) {
  for (int i : I) sa[--c[s[i]]] = i;
  return sa;
 }
 vector<bool> t(n); t[n - 1] = 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 = ranges::views::filter([&t](int x) {
  return x && t[x] && !t[x - 1]; });
 const 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] = int(lms.size());
  lms.push_back(sa[--x[s[i]]] = i);
 }
 induce(); vector<int> ns(lms.size());
 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;
 }
 ranges::fill(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 lexigraphically smallest suffix.
// hi[i]: LCP of suffix sa[i] and suffix sa[i - 1].
struct Suffix {
 int n; vector<int> sa, hi, rev;
 Suffix(const string &s) : n(int(s.size())),
  hi(n), rev(n) {
  vector<int> _s(n + 1); _s[n] = 0;
  copy(all(s), begin(_s)); // s shouldn't contain 0
  sa = sais(_s); sa.erase(sa.begin());
  for (int i = 0; i < n; i++) rev[sa[i]] = i;
  for (int i = 0, h = 0; i < n; ++i) {
   if (!rev[i]) { h = 0; continue; }
   for (int j = sa[rev[i] - 1]; i + h < n
     && s[i + h] == s[j + h]; ++h);
   hi[rev[i]] = h ? h-- : 0;
  }
 }
};
```

7.3 Ex SAM [58374b]

```
struct exSAM {
 int len[maxn * 2], link[maxn * 2]; // maxlen, suflink
 int next[maxn * 2][maxc], tot; // [0, tot), root = 0
 int ord[maxn * 2]; // topo. order (sort by length)
 int cnt[maxn * 2]; // occurence
 int newnode() {
  fill_n(next[tot], maxc, 0);
  return len[tot] = cnt[tot] = link[tot] = 0, 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 < maxc; ++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 (char 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 < maxc; ++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) ord[--lc[len[i]]] = i;
 }
 void solve() {
  for (int i = tot - 2; i >= 0; --i)
   cnt[link[ord[i]]] += cnt[ord[i]];
 }
};
```

7.4 KMP [281185]

```
vector<int> kmp(const auto &s) {
 vector<int> f(s.size());
 for (int i = 1, k = 0; i < (int)s.size(); ++i) {
  while (k > 0 && s[i] != s[k]) k = f[k - 1];
  f[i] = (k += (s[i] == s[k]));
 }
 return f;
}
vector<int> search(const auto &s, const auto &t) {
 // return 0-indexed occurrence of t in s
 vector<int> f = kmp(t), r;
 for (int i = 0, k = 0; i < (int)s.size(); ++i) {
  while(k > 0 && s[i] != t[k]) k = f[k - 1];
  k += (s[i] == t[k]);
  if (k == (int)t.size()) {
   r.push_back(i-t.size()+1);
   k = f[k - 1];
  }
 }
 return r;
}
```

7.5 Z value [6a7fd0]

```
vector<int> Zalgo(const string &s) {
 vector<int> z(s.size(), s.size());
 for (int i = 1, l = 0, r = 0; i < z[0]; ++i) {
  int j = clamp(r - i, 0, z[i - l]);
  for (; i + j < z[0] and s[i + j] == s[j]; ++j);
  if (i + (z[i] = j) > r) r = i + z[l = i];
 }
 return z;
}
```

7.6 Manacher [c938a9]

```
vector<int> manacher(const string &S) {
 const int n = (int)S.size(), m = n * 2 + 1;
 vector<int> z(m);
 string t = "."; for (char c: S) t += c, t += '.';
```

```
 for (int i = 1, l = 0, r = 0; i < m; ++i) {
  z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
  while (i - z[i] >= 0 && i + z[i] < m) {
   if (t[i - z[i]] == t[i + z[i]]) ++z[i];
   else break;
  }
  if (i + z[i] > r) r = i + z[i], l = i;
 }
 return z; // the palindrome lengths are z[i] - 1
}
/* for (int i = 1; i + 1 < m; ++i) {
  int l = (i - z[i] + 2) / 2, r = (i + z[i]) / 2;
  if (l != r) // [l, r) is maximal palindrome
} */
```

7.7 Lyndon Factorization [d22cc9]

```
// partition s = w[0] + w[1] + ... + w[k-1],
// w[0] >= w[1] >= ... >= w[k-1]
// each w[i] strictly smaller than all its suffix
void duval(const auto &s, auto &&report) {
 for (int n = (int)s.size(), i = 0, j, k; i < n; ) {
  for (j = i + 1, k = i; j < n && s[k] <= s[j]; j++)
   k = (s[k] < s[j] ? i : k + 1);
  // if (i < n / 2 && j >= n / 2) {
  // for min cyclic shift, call duval(s + s)
  // then here s.substr(i, n / 2) is min cyclic shift
  // }
  for (; i <= k; i += j - k)
   report(i, j - k); // s.substr(l, len)
 }
} // tested @ luogu 6114, 1368 & UVA 719
```

7.8 Main Lorentz [615b8f]

```
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);
 }
}
```

7.9 BWT [5a9b3a]

```
vector<int> v[SIGMA];
void BWT(char *ori, char *res) {
 // make ori -> ori + ori
 // then build suffix array
}
void iBWT(char *ori, char *res) {
 for (int i = 0; i < SIGMA; i++) v[i].clear();
 const int len = strlen(ori);
 for (int i = 0; i < len; i++)
  v[ori[i] - 'a'].push_back(i);
 vector<int> a;
 for (int i = 0, ptr = 0; i < SIGMA; i++)
  for (int j : v[i]) {
   a.push_back(j);
   ori[ptr++] = 'a' + i;
  }
 for (int i = 0, ptr = 0; i < len; i++) {
  res[i] = ori[a[ptr]];
  ptr = a[ptr];
 }
 res[len] = 0;
}
```

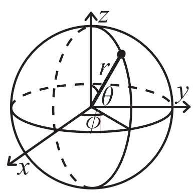
7.10 Palindromic Tree [0673ee]

```
struct PalindromicTree {
 struct node {
  int nxt[26], f, len; // num = depth of fail link
  int cnt, num;    // = #pal_suffix of this node
  node(int l = 0) : nxt{}, f(0), len(l), cnt(0), num(0)
     {}
 };
 vector<node> st; vector<char> s; int last, n;
 void init() {
  st.clear(); s.clear();
  last = 1; n = 0;
  st.push_back(0); st.push_back(-1);
  st[0].f = 1; s.push_back(-1);
 }
 int getFail(int x) {
  while (s[n - st[x].len - 1] != s[n]) x = st[x].f;
  return x;
 }
 void add(int c) {
  s.push_back(c -= 'a'); ++n;
  int cur = getFail(last);
  if (!st[cur].nxt[c]) {
   int now = st.size();
   st.push_back(st[cur].len + 2);
   st[now].f = st[getFail(st[cur].f)].nxt[c];
   st[cur].nxt[c] = now;
   st[now].num = st[st[now].f].num + 1;
  }
  last = st[cur].nxt[c]; ++st[last].cnt;
 }
 void dpcnt() { // cnt = #occurence in whole str
  for (int i = st.size() - 1; i >= 0; i--)
   st[st[i].f].cnt += st[i].cnt;
 }
 int size() { return st.size() - 2; }
} pt;
/* usage
string s; cin >> s; pt.init();
for (int i = 0; i < SZ(s); i++) {
 int prvsz = pt.size(); pt.add(s[i]);
 if (prvsz != pt.size()) {
  int r = i, l = r - pt.st[pt.last].len + 1;
  // pal @ [l,r]: s.substr(l, r-l+1)
 }
} */
```

8 Misc

8.1 Theorems

Spherical Coordinate



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

Sherman-Morrison formula

$\left(A+uv^\intercal\right)^{-1} = A^{-1} - \frac{A^{-1}uv^\intercal A^{-1}}{1+v^\intercal A^{-1}u}$

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 $\det(\tilde{L}_{rr})$.

Random Walk on Graph

Let P be the transition matrix of a strongly connected directed graph, $\sum_j P_{i,j} = 1$. Let $F_{i,j}$ be the expected time to reach j from i. Let g_i be the expected time from i to i, $G = \text{diag}(g)$ and J be a matrix all of 1, i.e. $J_{i,j} = 1$. Then, $F = J - G + PF$

First solve G: let $\pi P = \pi$ be a stationary distribution. Then $\pi_i g_i = 1$. The rank of $I - P$ is $n - 1$, so we first solve a special solution X such that $(I - P)X = J - G$ and adjust X to F by $F_{i,j} = X_{i,j} - X_{j,j}$.

Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniform randomly) 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're $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ 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 non-negative integers $d_1 \geq d_2 \geq \ldots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \ldots + d_n$ is even and

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

holds for all $1 \leq k \leq n$.

Havel–Hakimi algorithm

find the vertex who has greatest degree unused, connect it with other greatest vertex.

Gale–Ryser theorem

A pair of sequences of nonnegative integers $a_1 \geq \cdots \geq a_n$ 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 $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{n} \min(b_i, k)$ holds for every $1 \leq k \leq n$.

Fulkerson–Chen–Anstee theorem

A sequence $(a_1, b_1), \ldots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \cdots \geq a_n$ is digraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k)$ holds for every $1 \leq k \leq n$.

Euler's planar graph formula

$V - E + F = C + 1$. $E \leq 3V - 6$ (when $V \geq 3$)

Pick's theorem

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

Matroid Intersection

Given matroids $M_1 = (G, I_1), M_2 = (G, I_2)$, find maximum $S \in I_1 \cap I_2$. For each iteration, build the directed graph and find a shortest path from s to t.

- $s \to x : S \sqcup \{x\} \in I_1$
- $x \to t : S \sqcup \{x\} \in I_2$
- $y \to x : S \setminus \{y\} \sqcup \{x\} \in I_1$ (y is in the unique circuit of $S \sqcup \{x\}$)
- $x \to y : S \setminus \{y\} \sqcup \{x\} \in I_2$ (y is in the unique circuit of $S \sqcup \{x\}$)

Alternate the path, and $|S|$ will increase by 1. Let $R = \min(\text{rank}(I_1), \text{rank}(I_2)), N = |G|$. In each iteration, $|E| = O(RN)$. For weighted case, assign weight $-w(x)$ and $w(x)$ to $x \in S$ and $x \notin S$, resp. Use Bellman-Ford to find the weighted shortest path. The maximum iteration of Bellman-Ford is $2R + 1$.

Dual of LP

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

Parallel Axis Theorem

The second moment of area $I_z = \int\int x^2 + y^2 \mathrm{d}A$ $I_{z'} = I_z + Ad^2$ where d is the distance between two parallel axis z, z'.

8.2 Weight Matroid Intersection [d00ee8]

```
struct Matroid {
 Matroid(bitset<N>); // init from an independent set
 bool can_add(int); // check if break independence
 Matroid remove(int); // removing from the set
};
auto matroid_intersection(const vector<int> &w) {
 const int n = (int)w.size(); bitset<N> S;
 for (int sz = 1; sz <= n; sz++) {
  Matroid M1(S), M2(S); vector<vector<pii>> e(n + 2);
  for (int j = 0; j < n; j++) if (!S[j]) {
   if (M1.can_add(j)) e[n].eb(j, -w[j]);
   if (M2.can_add(j)) e[j].eb(n + 1, 0);
  }
  for (int i = 0; i < n; i++) if (S[i]) {
   Matroid T1 = M1.remove(i), T2 = M2.remove(i);
   for (int j = 0; j < n; j++) if (!S[j]) {
    if (T1.can_add(j)) e[i].eb(j, -w[j]);
    if (T2.can_add(j)) e[j].eb(i, w[i]);
   }
  } // maybe implicit build graph for more speed
  vector<pii> d(n + 2, {INF, 0}); d[n] = {0, 0};
  vector<int> prv(n + 2, -1);
  // change to SPFA for more speed, if necessary
  for (int upd = 1; upd--; )
   for (int u = 0; u < n + 2; u++)
    for (auto [v, c] : e[u]) {
     pii x(d[u].first + c, d[u].second + 1);
     if (x < d[v]) d[v] = x, prv[v] = u, upd = 1;
    }
  if (d[n + 1].first >= INF) break;
  for (int x = prv[n+1]; x!=n; x = prv[x]) S.flip(x);
  // S is the max-weighted independent set w/ size sz
 }
 return S;
} // from Nacl
```

8.3 Stable Marriage

1: Initialize $m \in M$ and $w \in W$ to *free*
2: **while** $\exists$ *free* man m who has a woman w to propose to **do**
3: $w \leftarrow$ first woman on m's list to whom m has not yet proposed
4: **if** $\exists$ some pair (m', w) **then**
5: **if** w prefers m to m' **then**
6: $m' \leftarrow$ *free*
7: $(m, w) \leftarrow$ *engaged*
8: **end if**
9: **else**
10: $(m, w) \leftarrow$ *engaged*
11: **end if**
12: **end while**

8.4 Bitset LCS [330ab1]

```
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';
```

8.5 Prefix Substring LCS [7d8faf]

```
void all_lcs(string S, string T) { // 0-base
 vector<size_t> h(T.size()); iota(all(h), 1);
 for (size_t a = 0; a < S.size(); ++a) {
  for (size_t c = 0, v = 0; c < T.size(); ++c)
   if (S[a] == T[c] || h[c] < v) swap(h[c], v);
  // here, LCS(s[0, a], t[b, c]) =
  // c - b + 1 - sum([h[i] > b] | i <= c)
 }
} // test @ yosupo judge
```

8.6 Convex 1D/1D DP [e5ab4b]

```
struct S { int i, l, r; };
auto solve(int n, int k, auto &w) {
 vector<int64_t> dp(n + 1);
 auto f = [&](int l, int r) -> int64_t {
  if (r - l > k) return -INF;
  return dp[l] + w(l + 1, r);
 };
 dp[0] = 0;
 deque<S> dq; dq.emplace_back(0, 1, n);
 for (int i = 1; i <= n; ++i) {
  dp[i] = f(dq.front().i, i);
  while (!dq.empty() && dq.front().r <= i)
   dq.pop_front();
  dq.front().l = i + 1;
  while (!dq.empty() &&
    f(i, dq.back().l) >= f(dq.back().i, dq.back().l))
   dq.pop_back();
  int p = i + 1;
  if (!dq.empty()) {
```

```
   auto [j, l, r] = dq.back();
   for (int s = 1 << 20; s; s >>= 1)
    if (l + s <= n && f(i, l + s) < f(j, l + s))
     l += s;
   dq.back().r = l; p = l + 1;
  }
  if (p <= n) dq.emplace_back(i, p, n);
 }
 return dp;
} // test @ tioj 烏龜疊疊樂
```

8.7 ConvexHull Optimization [b4318e]

```
struct L {
 mutable lld a, b, p;
 bool operator<(const L &r) const {
  return a < r.a; /* here */ }
 bool operator<(lld x) const { return p < x; }
};
lld Div(lld a, lld b) {
 return a / b - ((a ^ b) < 0 && a % b); }
struct DynamicHull : multiset<L, less<>> {
 static const lld kInf = 1e18;
 bool Isect(iterator x, iterator y) {
  if (y == end()) { x->p = kInf; return false; }
  if (x->a == y->a)
   x->p = x->b > y->b ? kInf : -kInf; /* here */
  else x->p = Div(y->b - x->b, x->a - y->a);
  return x->p >= y->p;
 }
 void Insert(lld a, lld 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));
 }
 lld Query(lld x) {  // default chmax
  auto l = *lower_bound(x); // to chmin:
  return l.a * x + l.b;    // modify the 2 "<>"
 }
};
```

8.8 Min Plus Convolution [464dcd]

```
// a is convex a[i+1]-a[i] <= a[i+2]-a[i+1]
vector<int> min_plus_convolution(auto &a, auto &b) {
 const int n = (int)a.size(), m = (int)b.size();
 vector<int> c(n + m - 1, numeric_limits<int>::max());
 auto dc = [&](auto Y, int l, int r, int jl, int jr) {
  if (l > r) return;
  int mid = (l + r) / 2, from = -1, &best = c[mid];
  for (int j = jl; j <= jr; j++)
   if (int i = mid - j; i >= 0 && i < n)
    if (best > a[i]+b[j]) best = a[i]+b[j], from = j;
  Y(Y, l, mid-1, jl, from); Y(Y, mid+1, r, from, jr);
 };
 return dc(dc, 0, n-1+m-1, 0, m-1), c;
}
```

8.9 De-Bruijn [aa7700]

```
vector<int> de_bruijn(int k, int n) {
 // return cyclic string of len k^n s.t. every string
 // of len n using k char appears as a substring.
 vector<int> aux(n + 1), res;
 auto db = [&](auto self, int t, int p) -> void {
  if (t <= n)
   for (int i = aux[t - p]; i < k; ++i, p = t)
    aux[t] = i, self(self, t + 1, p);
  else if (n % p == 0) for (int i = 1; i <= p; ++i)
   res.push_back(aux[i]);
 };
 return db(db, 1, 1), res;
}
```

8.10 Josephus Problem [7f9ceb]

```
lld f(lld n, lld m, lld k) { // n people kill m for
    each turn
 lld s = (m - 1) % (n - k); // O(k)
 for (lld i = n - k + 1; i <= n; i++) s = (s + m) % i;
 return s;
}
lld kth(lld n, lld m, i128 k) { // died at kth
 if (m == 1) return k;      // O(m log(n))
 for (k = k*m+m-1; k >= n; k = k-n + (k-n)/(m-1));
 return k;
} // k and result are 0-based, test @ CF 101955
```

8.11 N Queens Problem [31f83e]

```
void solve(VI &ret, int n) { // no sol when n=2,3
 if (n % 6 == 2) {
  for (int i = 2; i <= n; i += 2) ret.push_back(i);
  ret.push_back(3); ret.push_back(1);
  for (int i = 7; i <= n; i += 2) ret.push_back(i);
  ret.push_back(5);
 } else if (n % 6 == 3) {
  for (int i = 4; i <= n; i += 2) ret.push_back(i);
  ret.push_back(2);
  for (int i = 5; i <= n; i += 2) ret.push_back(i);
  ret.push_back(1); ret.push_back(3);
 } else {
  for (int i = 2; i <= n; i += 2) ret.push_back(i);
  for (int i = 1; i <= n; i += 2) ret.push_back(i);
 }
}
```

8.12 Tree Knapsack [f42766]

```
vector<int> G[N]; int dp[N][K]; pair<int,int> obj[N];
void dfs(int u, int mx) {
 for (int s : G[u]) {
  auto [w, v] = obj[s];
  if (mx < w) continue;
  for (int i = 0; i <= mx - w; i++)
   dp[s][i] = dp[u][i];
  dfs(s, mx - w);
  for (int i = w; i <= mx; i++)
   dp[u][i] = max(dp[u][i], dp[s][i - w] + v);
 }
}
```

8.13 Manhattan MST [1008bc]

```
vector<array<int, 3>> manhattanMST(vector<P> ps) {
 vector<int> id(ps.size()); iota(all(id), 0);
 vector<array<int, 3>> edges;
 for (int k = 0; k < 4; k++) {
  sort(all(id), [&](int i, int j) {
   return (ps[i] - ps[j]).x < (ps[j] - ps[i]).y; });
  map<int, int> sweep;
  for (int i : id) {
   for (auto it = sweep.lower_bound(-ps[i].y);
      it != sweep.end(); sweep.erase(it++)) {
    if (P d = ps[i] - ps[it->second]; d.y > d.x) break;
    else edges.push_back({d.y + d.x, i, it->second});
   }
   sweep[-ps[i].y] = i;
  }
  for (P &p : ps)
   if (k & 1) p.x = -p.x;
   else swap(p.x, p.y);
 }
 return edges; // [{w, i, j}, ...]
} // test @ yosupo judge
```

8.14 Binary Search On Fraction [765c5a]

```
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;
}
```

8.15 Barrett Reduction [d44617]

```
struct FastMod {
 using Big = __uint128_t; llu b, m;
 FastMod(llu b) : b(b), m(-1ULL / b) {}
 llu reduce(llu a) { // a % b
  llu r = a - (llu)((Big(m) * a) >> 64) * b;
  return r >= b ? r - b : r;
 }
};
```

8.16 Montgomery [47d32c]

```
struct Mont { // Montgomery multiplication
 constexpr static int W = 64;
 llu mod, R1Mod, R2Mod, NPrime;
 void set_mod(llu _mod) {
  mod = _mod; assert(mod & 1);
  llu xinv = 1;
  for (int i = 1; i < W; i++) // Hensel lifting
   if ((xinv * mod) >> i & 1) xinv |= 1ULL << i;
  assert(xinv * mod == 1);
  const u128 R = (u128(1) << W) % mod;
  R1Mod = static_cast<llu>(R);
  R2Mod = static_cast<llu>(R * R % mod);
  NPrime = -xinv;
 }
 llu redc(llu a, llu b) const {
  auto T = static_cast<u128>(a) * b;
  u128 m = static_cast<llu>(T) * NPrime;
  T += m * mod; T >>= W;
  return static_cast<llu>(T >= mod ? T - mod : T);
 }
 llu from(llu x) const {
  assert(x < mod); return redc(x, R2Mod);
 }
 llu get(llu a) const { return redc(a, 1); }
 llu one() const { return R1Mod; }
} mont;
// a * b % mod == get(redc(from(a), from(b)))
```