Contents

1 Basic

1.1 Pragma / IO

```cpp
#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)
#include<unistd.h>
char OB[65536]; int OP;
inline char RC() {
  static char buf[65536], *p = buf, *q = buf;
  return p == q && (q = (p
      = buf) + read(0, buf, 65536)) == buf ? -1 : *p++;
}
inline int R() {
  static char c;
  while((c = RC()) < '0'); int a = c ^ '0';
  while((c = RC()) >= '0') a *= 10, a += c ^ '0';
  return a;
}
inline void W(int n) {
  static char buf[12], p;
  if (n == 0) OB[OP++]='0'; p = 0;
  while (n) buf[p++] = '0' + (n % 10), n /= 10;
  for (--p; p >= 0; --p) OB[OP++] = buf[p];
  if (OP > 65520) write(1, OB, OP), OP = 0;
}
```

1.2 Debug Macro

```cpp
void db() { cerr << endl; }
template <typename T, typename ...U>
void db(T i, U ...j) { cerr << i << ' ', db(j...); }
#ifdef ABS
#define bug(x...) db("[" + string(#x) + "]", x)
#define safe cerr << __PRETTY_FUNCTION__
    << " line " << __LINE__ << " safe" << endl
#else
#define bug(x...) void(0)
#define safe void(0)
#endif
```

2 Data Structure

2.1 Heavy-Light Decomposition

```cpp
int n, q, dfn = 0;
int val[maxn], sz[maxn], head[maxn], dep[maxn
    ], st[maxn * 4], par[maxn], loc[maxn], id[maxn];
vector<int> adj[maxn];

void dfs(int pos, int prev){
  sz[pos] = 1;
  if(prev != -1) adj[pos].erase
      (find(adj[pos].begin(), adj[pos].end(), prev));
  for(auto &x : adj[pos]){
    par[x] = pos, dep[x] = dep[pos] + 1;
    dfs(x, pos);
    sz[pos] += sz[x];
    if(sz[x] > sz[adj[pos][0]]) swap(x, adj[pos][0]);
  }
}
void decompose(int pos, int h){
  id[dfn++] = pos;
  head[pos] = h, loc[pos] = dfn - 1;
  // upd(loc[pos], val[pos]);
  for(auto x : adj[pos]){
    if(x == adj[pos][0]) decompose(x, h);
    else decompose(x, x);
  }
}
void build(){
  dfs(0, -1);
  decompose(0, 0);
  //build_segtree();
}
int solve(int a, int b){
  int ret = 0;
  while(head[a] != head[b]){
    if(dep[head[a]] > dep[head[b]]) swap(a, b);
    ret = max(ret, qry(loc[head[b]], loc[b]));
    b = par[head[b]];
  }
  if(dep[a] > dep[b]) swap(a, b);
  return max(ret, qry(loc[a], loc[b]));
}
```

2.2 Centroid Decomposition

```cpp
vector<pll> adj[maxn];
ll dist[20][maxn]; // distance to kth-layer-parent
int sz[maxn], del[maxn], par[maxn], cdep[maxn];
ll cnt[maxn], sum[maxn], re[maxn]; // re: subtree->par
int n, q;

void dfssz(int pos, int prev){
    sz[pos] = 1;
    for(auto [x, w] : adj[pos]){
        if(del[x] || x == prev) continue;
        dfssz(x, pos);
        sz[pos] += sz[x];
    }
}
int get_centroid(int pos, int prev, int siz){
    for(auto [x, w] : adj[pos]){
        if(!del[x] && x != prev && sz[x] >
            siz / 2) return get_centroid(x, pos, siz);
    }
    return pos;
}
void get_dist(int pos, int prev, int layer){
    for(auto [x, w] : adj[pos]){
        if(del[x] || x == prev) continue;
        dist[layer][x] = dist[layer][pos] + w;
        get_dist(x, pos, layer);
    }
}
void cd(int pos, int layer = 1, int p = 0){
    dfssz(pos, -1);
    int cen = get_centroid(pos, -1, sz[pos]);
    del[cen] = 1;
    dist[layer][cen] = 0;
    cdep[cen] = layer;
    par[cen] = p;
    get_dist(cen, -1, layer);
    for(auto [x, w] : adj[cen]){
        if(!del[x]){
            cd(x, layer + 1, cen);
        }
    }
}
void upd(int p){
    for(int x = p, d = cdep[x]; d; x = par[x], d--){
        sum[x] += dist[d][p];
        re[x] += dist[d - 1][p];
        cnt[x] ++;
    }
}
ll qry(int p){
    ll pre = 0, ans = 0;
    for(int x = p, d = cdep[x]; d; x = par[x], d--){
        ans += sum
            [x] - re[x] + (cnt[x] - pre) * dist[d][p];
        pre = cnt[x];
    }
    return ans;
}
```

2.3 Link Cut Tree

```
struct LCT{
  int ch[maxn
      ][2], par[maxn], rev[maxn], xr[maxn], val[maxn];
  int get(int x){ return ch[par[x]][1] == x;}
  int isroot(int x){
      return ch[par[x]][0] != x && ch[par[x]][1] != x;}
  void push(int x){
    if(rev[x]){
      if(rs) swap(ch[rs][0], ch[rs][1]), rev[rs] ^= 1;
      if(ls) swap(ch[ls][0], ch[ls][1]), rev[ls] ^= 1;
      rev[x] = 0;
    }
  }
  void pull(int x){
    xr[x] = xr[ls] ^ xr[rs] ^ val[x];
  }
  void rotate(int x){
    int y = par[x], z = par[y], k = get(x);
    if(!isroot(y)) ch[z][ch[z][1] == y] = x;
    ch[y][k] = ch[x][!k], par[ch[x][!k]] = y;
    ch[x][!k] = y, par[y] = x;
    par[x] = z;
    pull(y), pull(x);
  }
  void update(int x){
    if(!isroot(x)) update(par[x]);
    push(x);
  }
  void splay(int x){
    update(x);
    for(int
        p = par[x]; !isroot(x); rotate(x), p = par[x]){
      if(!isroot(p)) rotate(get(p) == get(x) ? p : x);
    }
  }
  void access(int x){
    for(int p = 0; x != 0; p = x, x = par[x]){
      splay(x);
      ch[x][1] = p;
      pull(x);
    }
  }
  void make_root(int x){
    access(x);
    splay(x);
    swap(ls, rs);
    rev[x] ^= 1;
  }
  void link(int x, int y){
    make_root(x);
    splay(x);
    if(find_root(y) == x) return;
    par[x] = y;
  }
  void cut(int x, int y){
    make_root(x);
    access(y);
    splay(x);
    if(par[y] != x || ch[y][0]) return;
    ch[x][1] = par[y] = 0;
  }
  int find_root(int x){
    access(x);
    splay(x);
    push(x);
    while(ls) x = ls, push(x);
    splay(x);
    return x;
  }
  void split(int x, int y){
    make_root(x);
    access(y);
    splay(y);
  }
  void upd(int x, int y){
    access(x);
    splay(x);
    val[x] = y;
    pull(x);
  }
} st;
```

2.4 LiChaoST

```
struct line{
  ll m, k;
  line(){}
  line(ll _m, ll _k) : m(_m), k(_k){}
  ll val(ll x){ return m * x + k; }
};

struct node{
  line ans;
  node *l, *r;
  int siz;
  node(){}
  node(line l) : ans(l), l(nullptr), r(nullptr){ }
};
node sgt[maxn];

int root[maxn], cnt = 0;

struct segtree{
  node *rt;
  int n, siz;
  segtree() : n(maxc * 2), siz(0), rt(nullptr){}
  void insert(node* &k, int l, int r, line cur){
    if(!k){
      k = new node(cur);
      siz++;
      return;
    }
    if(l == r){
      if(k->ans.val(l) > cur.val(l)) k->ans = cur;
      return;
    }
    int m = (l + r) / 2;
    if(k->ans.val(m) > cur.val(m)) swap(k->ans, cur);
    if(cur.m > k->ans.m) insert(k->l, l, m, cur);
    else insert(k->r, m + 1, r, cur);
  }
  void insert
      (ll m, ll k) { insert(rt, 0, n, line(m, k)); }
  void insert(line l) { insert(rt, 0, n, l);}
  ll qry(node *k, int l, int r, int pos){
    if(!k) return INF;
    if(l == r) return k->ans.val(pos);
    int m = (l + r) / 2;
    return min(k->ans.val(pos), pos <= m ? qry
        (k->l, l, m, pos) : qry(k->r, m + 1, r, pos));
  }
  ll qry(int pos) { return qry(rt, 0, n, pos); }
};
```

2.5 Leftist Heap

```
struct LeftistTree{
  int cnt, rt[maxn
      ], lc[maxn * 20], rc[maxn * 20], d[maxn * 20];
  int v[maxn * 20];
  LeftistTree(){}
  int newnode(pll nd){
    cnt++;
    v[cnt] = nd;
    return cnt;
  }
  int merge(int x, int y){
    if(!x || !y) return x + y;
    if(v[x] > v[y]) swap(x, y);
    int p = ++cnt;
    lc[p] = lc[x], v[p] = v[x];
    rc[p] = merge(rc[x], y);
    if(d[lc[p]] < d[rc[p]]) swap(lc[p], rc[p]);
    d[p] = d[rc[p]] + 1;
    return p;
  }
} st;
```

2.6 Treap

```
struct node{
  int val, pri, c = 1;
  node *l, *r;
  node(int _val) :
       val(_val), pri(rand()), l(nullptr), r(nullptr){}
  void recalc();
} *rt;
int cnt(node *t){ return t ? t->c : 0;}
void node::recalc(){
  c = cnt(l) + cnt(r) + 1;
}
pair<node*, node*> split(node *t, int val){
  if(!t) return {nullptr, nullptr};
  if(cnt(t->l) < val){
    auto p = split(t->r, val - cnt(t->l) - 1);
    t->r = p.first;
    t->recalc();
    return {t, p.second};
  }
  else{
    auto p = split(t->l, val);
    t->l = p.second;
    t->recalc();
    return {p.first, t};
  }
}
node* merge(node *a, node *b){
  if(!a || !b) return a ? a : b;
  if(a->pri > b->pri){
    a->r = merge(a->r, b);
    a->recalc();
    return a;
  }
  else{
    b->l = merge(a, b->l);
    b->recalc();
    return b;
  }
}
node *insert(node *t, int k){
  auto [a, b] = split(t, k);
  return merge(merge(a, new node(k)), b);
}
node* remove(node *t, int k){
  auto [a, b] = split(t, k - 1);
  auto [b, c] = split(b, k);
  return merge(a, c);
}
```

2.7 Chtholly Tree

```
struct ChthollyTree {
  struct interval {
    int l, r;
    ll v;
    interval (
        int _l, int _r, ll _v) : l(_l), r(_r), v(_v) {}
  };
  struct cmp {
    bool operator ()
         (const interval &a, const interval& b) const {
      return a.l < b.l;
    }
  };
  set <interval, cmp> s;
  vector <interval> split(int l, int r) {
    // split
         into [0, l), [l, r), [r, n) and return [l, r)
    vector <interval> del, ans, re;
    auto it = s.lower_bound(interval(l, -1, 0));
    if (it
        != s.begin() && (it == s.end() || l < it->l)) {
      --it;
      del.pb(*it);
      if (r < it->r) {
        re.pb(interval(it->l, l, it->v));
        ans.pb(interval(l, r, it->v));
        re.pb(interval(r, it->r, it->v));
      } else {
        re.pb(interval(it->l, l, it->v));
        ans.pb(interval(l, it->r, it->v));
      }
      ++it;
    }
    for (; it != s.end() && it->r <= r; ++it) {
      ans.pb(*it);
      del.pb(*it);
    }
    if (it != s.end() && it->l < r) {
      del.pb(*it);
      ans.pb(interval(it->l, r, it->v));
      re.pb(interval(r, it->r, it->v));
    }
    for (interval &i : del)
      s.erase(i);
    for (interval &i : re)
      s.insert(i);
    return ans;
  }
  void merge(vector <interval> a) {
    for (interval &i : a)
      s.insert(i);
  }
};
```

2.8 Persistent Segment Tree

```
struct Seg {
  // Persistent Segment
       Tree, single point modify, range query sum
  // 0-indexed, [l, r)
  static Seg mem[M], *pt;
  int l, r, m, val;
  Seg* ch[2];
  Seg () = default;
  Seg (int _l
      , int _r) : l(_l), r(_r), m(l + r >> 1), val(0) {
    if (r - l > 1) {
      ch[0] = new (pt++) Seg(l, m);
      ch[1] = new (pt++) Seg(m, r);
    }
  }
  void pull() {val = ch[0]->val + ch[1]->val;}
  Seg* modify(int p, int v) {
    Seg *now = new (pt++) Seg(*this);
    if (r - l == 1) {
      now->val = v;
    } else {
      now->ch[p >= m] = ch[p >= m]->modify(p, v);
      now->pull();
    }
    return now;
  }
  int query(int a, int b) {
    if (a <= l && r <= b) return val;
    int ans = 0;
    if (a < m) ans += ch[0]->query(a, b);
    if (m < b) ans += ch[1]->query(a, b);
    return ans;
  }
} Seg::mem[M], *Seg::pt = mem;
// Init Tree
Seg *root = new (Seg::pt++) Seg(0, n);
```

2.9 Range Chmin Chmax Add Range Sum

```
#include <algorithm>
#include <iostream>
using namespace std;
typedef long long ll;

const int MAXC = 200005;
const ll INF = 1e18;
```

```
struct node {
  ll sum;
  ll mx, mxcnt, smx;
  ll mi, micnt, smi;
  ll lazymax, lazymin, lazyadd;
  node(ll k = 0)
    : sum(k), mx(k), mxcnt(1), smx(-INF), mi(k),
      micnt(1), smi(INF), lazymax(-INF), lazymin(INF),
      lazyadd(0) {}
  node operator+(const node &a) const {
    node rt;
    rt.sum = sum + a.sum;
    rt.mx = max(mx, a.mx);
    rt.mi = min(mi, a.mi);
    if (mx == a.mx) {
      rt.mxcnt = mxcnt + a.mxcnt;
      rt.smx = max(smx, a.smx);
    } else if (mx > a.mx) {
      rt.mxcnt = mxcnt;
      rt.smx = max(smx, a.mx);
    } else {
      rt.mxcnt = a.mxcnt;
      rt.smx = max(mx, a.smx);
    }
    if (mi == a.mi) {
      rt.micnt = micnt + a.micnt;
      rt.smi = min(smi, a.smi);
    } else if (mi < a.mi) {
      rt.micnt = micnt;
      rt.smi = min(smi, a.mi);
    } else {
      rt.micnt = a.micnt;
      rt.smi = min(mi, a.smi);
    }
    rt.lazymax = -INF;
    rt.lazymin = INF;
    rt.lazyadd = 0;
    return rt;
  }
} seg[MAXC << 2];

ll a[MAXC];

void give_tag_min(int rt, ll t) {
  if (t >= seg[rt].mx) return;
  seg[rt].lazymin = t;
  seg[rt].lazymax = min(seg[rt].lazymax, t);
  seg[rt].sum -= seg[rt].mxcnt * (seg[rt].mx - t);
  if (seg[rt].mx == seg[rt].smi) seg[rt].smi = t;
  if (seg[rt].mx == seg[rt].mi) seg[rt].mi = t;
  seg[rt].mx = t;
}

void give_tag_max(int rt, ll t) {
  if (t <= seg[rt].mi) return;
  seg[rt].lazymax = t;
  seg[rt].sum += seg[rt].micnt * (t - seg[rt].mi);
  if (seg[rt].mi == seg[rt].smx) seg[rt].smx = t;
  if (seg[rt].mi == seg[rt].mx) seg[rt].mx = t;
  seg[rt].mi = t;
}

void give_tag_add(int l, int r, int rt, ll t) {
  seg[rt].lazyadd += t;
  if (seg[rt].lazymax != -INF) seg[rt].lazymax += t;
  if (seg[rt].lazymin != INF) seg[rt].lazymin += t;
  seg[rt].mx += t;
  if (seg[rt].smx != -INF) seg[rt].smx += t;
  seg[rt].mi += t;
  if (seg[rt].smi != INF) seg[rt].smi += t;
  seg[rt].sum += (ll)(r - l + 1) * t;
}

void tag_down(int l, int r, int rt) {
  if (seg[rt].lazyadd != 0) {
    int mid = (l + r) >> 1;
    give_tag_add(l, mid, rt << 1, seg[rt].lazyadd);
    give_tag_add(
      mid + 1, r, rt << 1 | 1, seg[rt].lazyadd);
    seg[rt].lazyadd = 0;
  }
  if (seg[rt].lazymin != INF) {
    give_tag_min(rt << 1, seg[rt].lazymin);
    give_tag_min(rt << 1 | 1, seg[rt].lazymin);
    seg[rt].lazymin = INF;
  }
  if (seg[rt].lazymax != -INF) {
    give_tag_max(rt << 1, seg[rt].lazymax);
    give_tag_max(rt << 1 | 1, seg[rt].lazymax);
    seg[rt].lazymax = -INF;
  }
}

void build(int l, int r, int rt) {
  if (l == r) return seg[rt] = node(a[l]), void();
  int mid = (l + r) >> 1;
  build(l, mid, rt << 1);
  build(mid + 1, r, rt << 1 | 1);
  seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}

void modifymax(
  int L, int R, int l, int r, int rt, ll t) {
  if (L <= l && R >= r && t < seg[rt].smi)
    return give_tag_max(rt, t);
  if (l != r) tag_down(l, r, rt);
  int mid = (l + r) >> 1;
  if (L <= mid) modifymax(L, R, l, mid, rt << 1, t);
  if (R > mid)
    modifymax(L, R, mid + 1, r, rt << 1 | 1, t);
  seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}

void modifymin(
  int L, int R, int l, int r, int rt, ll t) {
  if (L <= l && R >= r && t > seg[rt].smx)
    return give_tag_min(rt, t);
  if (l != r) tag_down(l, r, rt);
  int mid = (l + r) >> 1;
  if (L <= mid) modifymin(L, R, l, mid, rt << 1, t);
  if (R > mid)
    modifymin(L, R, mid + 1, r, rt << 1 | 1, t);
  seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}

void modifyadd(
  int L, int R, int l, int r, int rt, ll t) {
  if (L <= l && R >= r)
    return give_tag_add(l, r, rt, t);
  if (l != r) tag_down(l, r, rt);
  int mid = (l + r) >> 1;
  if (L <= mid) modifyadd(L, R, l, mid, rt << 1, t);
  if (R > mid)
    modifyadd(L, R, mid + 1, r, rt << 1 | 1, t);
  seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}

ll query(int L, int R, int l, int r, int rt) {
  if (L <= l && R >= r) return seg[rt].sum;
  if (l != r) tag_down(l, r, rt);
  int mid = (l + r) >> 1;
  if (R <= mid) return query(L, R, l, mid, rt << 1);
  if (L > mid)
    return query(L, R, mid + 1, r, rt << 1 | 1);
  return query(L, R, l, mid, rt << 1) +
    query(L, R, mid + 1, r, rt << 1 | 1);
}

int main() {
  ios::sync_with_stdio(0), cin.tie(0);
  int n, m;
  cin >> n >> m;
  for (int i = 1; i <= n; ++i) cin >> a[i];
  build(1, n, 1);
  while (m--) {
    int k, x, y;
    ll t;
    cin >> k >> x >> y, ++x;
    if (k == 0) cin >> t, modifymin(x, y, 1, n, 1, t);
    else if (k == 1)
      cin >> t, modifymax(x, y, 1, n, 1, t);
    else if (k == 2)
      cin >> t, modifyadd(x, y, 1, n, 1, t);
    else cout << query(x, y, 1, n, 1) << "\n";
  }
}
```

2.10 Range Set

```
struct RangeSet { // [l, r)
  set <pii> S;
  void cut(int x) {
    auto it = S.lower_bound({x + 1, -1});
    if (it == S.begin()) return;
    auto [l, r] = *prev(it);
    if (l >= x || x >= r) return;
    S.erase(prev(it));
    S.insert({l, x});
    S.insert({x, r});
  }
  vector <pii> split(int l, int r) {
    // remove and return ranges in [l, r)
    cut(l), cut(r);
    vector <pii> res;
    while (true) {
      auto it = S.lower_bound({l, -1});
      if (it == S.end() || r <= it->first) break;
      res.pb(*it), S.erase(it);
    }
    return res;
  }
  void insert(int l, int r) {
    // add a range [l, r), [l, r) not in S
    auto it = S.lower_bound({l, r});
    if (it != S.begin() && prev(it)->second == l)
      l = prev(it)->first, S.erase(prev(it));
    if (it != S.end() && r == it->first)
      r = it->second, S.erase(it);
    S.insert({l, r});
  }
  bool count(int x) {
    auto it = S.lower_bound({x + 1, -1});
    return it != S.begin() && prev(it)->first <= x
           && x < prev(it)->second;
  }
};
```

2.11 vEB Tree

```
using u64=uint64_t;
constexpr int lsb(u64 x)
{ return x?__builtin_ctzll(x):1<<30; }
constexpr int msb(u64 x)
{ return x?63-__builtin_clzll(x):-1; }
template<int N, class T=void>
struct veb{
  static const int M=N>>1;
  veb<M> ch[1<<N-M];
  veb<N-M> aux;
  int mn,mx;
  veb():mn(1<<30),mx(-1){}
  constexpr int mask(int x){return x&((1<<M)-1);}
  bool empty(){return mx==-1;}
  int min(){return mn;}
  int max(){return mx;}
  bool have(int x){
    return x==mn?true:ch[x>>M].have(mask(x));
  }
  void insert_in(int x){
    if(empty()) return mn=mx=x,void();
    if(x<mn) swap(x,mn);
    if(x>mx) mx=x;
    if(ch[x>>M].empty()) aux.insert_in(x>>M);
    ch[x>>M].insert_in(mask(x));
  }
  void erase_in(int x){
    if(mn==mx) return mn=1<<30,mx=-1,void();
    if(x==mn) mn=x=(aux.min()<<M)^ch[aux.min()].min();
    ch[x>>M].erase_in(mask(x));
    if(ch[x>>M].empty()) aux.erase_in(x>>M);
    if(x==mx){
      if(aux.empty()) mx=mn;
      else mx=(aux.max()<<M)^ch[aux.max()].max();
    }
  }
  void insert(int x){
    if(!have(x)) insert_in(x);
  }
  void erase(int x){
    if(have(x)) erase_in(x);
  }
  int next(int x){// >=x
    if(x>mx) return 1<<30;
    if(x<=mn) return mn;
    if(mask(x)<=ch[x>>M].max())
      return ((x>>M)<<M)^ch[x>>M].next(mask(x));
    int y=aux.next((x>>M)+1);
    return (y<<M)^ch[y].min();
  }
  int prev(int x){// <x
    if(x<=mn) return -1;
    if(x>mx) return mx;
    if(x<=(aux.min()<<M)+ch[aux.min()].min())
      return mn;
    if(mask(x)>ch[x>>M].min())
      return ((x>>M)<<M)^ch[x>>M].prev(mask(x));
    int y=aux.prev(x>>M);
    return (y<<M)^ch[y].max();
  }
};
template<int N>
struct veb<N,typename enable_if<N<=6>::type>{
  u64 a;
  veb():a(0){}
  void insert_in(int x){a|=1ull<<x;}
  void insert(int x){a|=1ull<<x;}
  void erase_in(int x){a&=~(1ull<<x);}
  void erase(int x){a&=~(1ull<<x);}
  bool have(int x){return a>>x&1;}
  bool empty(){return a==0;}
  int min(){return lsb(a);}
  int max(){return msb(a);}
  int next(int x){return lsb(a&~((1ull<<x)-1));}
  int prev(int x){return msb(a&((1ull<<x)-1));}
};
```

2.12 KD Tree

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

2.13 pbds

```
#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
```

3 Graph

3.1 SCC

```
struct SCC {
  int n, nscc, _id;
  vector<vector<int>> g;
  vector<int> dep, low, scc_id, stk;
  void dfs(int v) {
    dep[v] = low[v] = _id++, stk.pb(v);
    for (int u : g[v]) if (scc_id[u] == -1) {
      if (low[u] == -1) dfs(u);
      low[v] = min(low[v], low[u]);
    }
    if (low[v] == dep[v]) {
      int id = nscc++, x;
      do {
        x = stk.back(), stk.pop_back(), scc_id[x] = id;
      } while (x != v);
    }
  }
  void build() {
    for (int i = 0; i < n; ++i) if (low[i] == -1)
      dfs(i);
  }
  void add_edge(int u, int v) { g[u].pb(v); }
  SCC (int _n) : n(_n), nscc(0), _id(0), g(n), dep(n),
    low(n, -1), scc_id(n, -1), stk() {}
};
```

3.2 BCC Vertex

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

3.3 Directed MST

```
using D = int;
struct edge { int u, v; D w; };
// 0-based, return index of edges
vector<int> dmst(vector<edge> &e, int n, int root) {
  using T = pair <D, int>;
  using PQ = pair
      <priority_queue <T, vector <T>, greater <T>>, D>;
  auto push = [](PQ &pq, T v) {
    pq.first.emplace(v.first - pq.second, v.second);
  };
  auto top = [](const PQ &pq) -> T {
    auto r = pq.first.top();
    return {r.first + pq.second, r.second};
  };
  auto join = [&push, &top](PQ &a, PQ &b) {
    if (a.first.size() < b.first.size()) swap(a, b);
    while (!b.first.empty())
      push(a, top(b)), b.first.pop();
  };
  vector<PQ> h(n * 2);
  for (int i = 0; i < e.size(); ++i)
    push(h[e[i].v], {e[i].w, i});
  vector<int>
       a(n * 2), v(n * 2, -1), pa(n * 2, -1), r(n * 2);
  iota(all(a), 0);
  auto o = [&](int x) { int y;
    for (y = x; a[y] != y; y = a[y]);
    for (int ox = x; x != y; ox = x)
      x = a[x], a[ox] = y;
    return y;
  };
  v[root] = n + 1;
  int pc = n;
  for (int i = 0; i < n; ++i) if (v[i] == -1) {
    for (int p =
        i; v[p] == -1 || v[p] == i; p = o(e[r[p]].u)) {
      if (v[p] == i) {
        int q = p; p = pc++;
        do {
          h[q].second = -h[q].first.top().first;
          join(h[pa[q] = a[q] = p], h[q]);
        } while ((q = o(e[r[q]].u)) != p);
      }
      v[p] = i;
      while (!h[p].first
          .empty() && o(e[top(h[p]).second].u) == p)
        h[p].first.pop();
      r[p] = top(h[p]).second;
    }
  }
  vector<int> ans;
  for (int i = pc - 1; i >= 0; i--)
    if (i != root && v[i] != n) {
      for (int f = e[r[i]].
          v; f != -1 && v[f] != n; f = pa[f]) v[f] = n;
      ans.pb(r[i]);
    }
  return ans;
}
```

3.4 Negative Cycle

```
vector<pll> adj[maxn];
template <typename T>
struct NegativeCycle {
  vector <T> dis;
  vector <int> rt;
  int n; T INF;
  vector <int> cycle;
  NegativeCycle () = default;
  NegativeCycle
      (int _n) : n(_n), INF(numeric_limits<T>::max()) {
    dis.assign(n, 0), rt.assign(n, -1);
    int relax = -1;
    for (int t = 0; t < n; ++t) {
      relax = -1;
      for (int i = 0; i < n; ++i) {
        for (auto
            [j, w] : adj[i]) if (dis[j] > dis[i] + w) {
          dis[j] = dis[i] + w, rt[j] = i;
          relax = j;
        }
      }
    }
    if (relax != -1) {
      int s = relax;
      for (int i = 0; i < n; ++i) s = rt[s];
      vector <bool> vis(n, false);
      while (!vis[s]) {
        cycle.push_back(s), vis[s] = true;
        s = rt[s];
      }
      reverse(cycle.begin(), cycle.end());
    }
  }
};
```

3.5 Dominator Tree

```
int in[maxn], id[maxn], par[maxn], dfn = 0;
int mn[maxn], idom[maxn], sdom[maxn], ans[maxn];
int fa[maxn]; // dsu
int n, m;

struct edge{
  int to, id;
  edge(){}
  edge(int _to, int _id) : to(_to), id(_id){}
};
vector<edge> adj[3][maxn];

void dfs(int pos){
  in[pos] = ++dfn;
  id[dfn] = pos;
  for(auto [x, id] : adj[0][pos]){
    if(in[x]) continue;
    dfs(x);
    par[x] = pos;
  }
}

int find(int x){
  if(fa[x] == x) return x;
  int tmp = fa[x];
  fa[x] = find(fa[x]);
  if(in[sdom[mn[tmp]]] < in[sdom[mn[x]]]){
    mn[x] = mn[tmp];
  }
  return fa[x];
}

void tar(int st){
  dfs(st);
  for(int
      i = 0; i < n; i++) mn[i] = sdom[i] = fa[i] = i;
  for(int i = dfn; i >= 2; i--){
    int pos = id[i], res = INF; // res : in(x) of sdom
    for(auto [x, id] : adj[1][pos]){
      if(!in[x]) continue;
      find(x);
      if(in[pos] > in[x]) res = min(res, in[x]);
      else res = min(res, in[sdom[mn[x]]]);
    }
    sdom[pos] = id[res];
    fa[pos] = par[pos];
    adj[2][sdom[pos]].eb(pos, 0);
    pos = par[pos];
    for(auto [x, id] : adj[2][pos]){
      find(x);
      if(sdom[mn[x]] == pos){
        idom[x] = pos;
      }
      else{
        idom[x] = mn[x];
      }
    }
    adj[2][pos].clear();
  }
  for(int i = 2; i <= dfn; i++){
    int x = id[i];
    if(idom[x] != sdom[x]) idom[x] = idom[idom[x]];
  }
}
```

3.6 Maximum Clique

```
struct MaximumClique{
    typedef bitset<maxn> bst;
    bst adj[maxn], empt;
    int p[maxn], n, ans;
    void init(int _n){
        n = _n;
        for(int i = 0; i < n; i++) adj[i].reset();
    }
    void BronKerbosch(bst R, bst P, bst X){
        if(P == empt && X == empt){
            ans = max(ans, (int)R.count());
            return;
        }
        bst tmp = P | X;
        if((R | P | X).count() <= ans) return;
        int u;
        for(int i = 0; i < n; i++){
            if(tmp[u = p[i]]) break;
        }
        bst lim = P & ~adj[u];
        for(int i = 0; i < n; i++){
            int v = p[i];
            if(lim[v]){
                R[v] = 1;
                BronKerbosch
                    (R, P & adj[v], X & adj[v]);
                R[v] = 0, P[v] = 0, X[v] = 1;
            }
        }
    }
    void add_edge(int a, int b){
        adj[a][b] = adj[b][a] = 1;
    }
    int solve(){
        bst R, P, X;
        ans = 0, P.flip();
        iota(p, p + n, 0);
        random_shuffle
            (p, p + n), BronKerbosch(R, P, X);
        return ans;
    }
};
```

3.7 Virtual Tree

```
// need lca, in, out
vector<pll> virtual_tree(vector<int> &v) {
  auto cmp = [&](int x, int y) {return in[x] < in[y];};
  sort(all(v), cmp);
  int sz = (int)v.size();
  for (int i = 0; i + 1 < sz; ++i)
    v.pb(lca(v[i], v[i + 1]));
  sort(all(v), cmp);
  v.resize(unique(all(v)) - v.begin());
  vector <int> stk(1, v[0]);
  vector <pll> res;
  for (int i = 1; i < (int)v.size(); ++i) {
    int x = v[i];
    while (out[stk.back()] < out[x]) stk.pop_back();
    res.emplace_back(stk.back(), x), stk.pb(x);
  }
  return res;
}
```

3.8 Minimum Steiner Tree

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

4 Flow / Matching

4.1 Dinic

```
struct Dinic{
  struct edge{
    ll to, cap;
    edge(){}
    edge(int _to, ll _cap) : to(_to), cap(_cap){}
  };
  vector<edge> e;
  vector<vector<int>> adj;
  vector<int> iter, level;
  int n, s, t;
  void init(int _n, int _s, int _t){
    n = _n, s = _s, t = _t;
    adj = vector<vector<int>>(n);
    iter = vector<int>(n);
    level = vector<int>(n);
    e.clear();
  }
  void add_edge(int from, int to, ll cap){
    adj[from].pb(e.size()), adj[to].pb(e.size() + 1);
    e.pb(edge(to, cap)), e.pb(edge(from, 0));
  }
  void bfs(){
    fill(level.begin(), level.end(), -1);
    level[s] = 0;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
      int cur = q.front(); q.pop();
      for(auto id : adj[cur]){
        auto [to, cap] = e[id];
        if(level[to] == -1 && cap){
          level[to] = level[cur] + 1;
          q.push(to);
        }
      }
    }
  }
  ll dfs(int pos, ll flow){
    if(pos == t) return flow;
    for(int &i = iter[pos]; i < adj[pos].size(); i++){
      auto [to, cap] = e[adj[pos][i]];
      if(level[to] == level[pos] + 1 && cap){
        ll tmp = dfs(to, min(flow, cap));
        if(tmp){
          e[adj[pos][i]].cap -= tmp;
          e[adj[pos][i] ^ 1].cap += tmp;
          return tmp;
        }
      }
    }
    return 0;
  }
} flow;
```

4.2 Min Cost Max Flow

```
struct MCMF{
    using T = ll;
    struct edge{
        int to;
        T cap, cost;
        edge(){}
        edge(int _to, T _cap, T
             _cost) : to(_to), cap(_cap), cost(_cost){}
    };
    vector<edge> e;
    vector<vector<int>> adj;
    vector<int> iter, inq;
    vector<T> dist;
    int n, s, t;
    void init(int _n, int _s, int _t){
        n = _n, s = _s, t = _t;
        adj = vector<vector<int>>(n);
        iter = vector<int>(n);
        dist = vector<T>(n);
        inq = vector<int>(n);
        e.clear();
    }
    void add_edge(int from, int to, T cap, T cost = 0){
        adj[from
            ].pb(e.size()), adj[to].pb(e.size() + 1);
        e.pb(edge(to
            , cap, cost)), e.pb(edge(from, 0, -cost));
    }
    bool spfa(){
        fill(dist.begin(), dist.end(), INF);
        queue<int> q;
        q.push(s);
        dist[s] = 0, inq[s] = 1;
        while(!q.empty()){
            int pos = q.front(); q.pop();
            inq[pos] = 0;
            for(auto id : adj[pos]){
                auto [to, cap, cost] = e[id];
                if(cap && dist[to] > dist[pos] + cost){
                    dist[to] = dist[pos] + cost;
                    if(!inq
                        [to]) q.push(to), inq[to] = 1;
                }
            }
        }
        return dist[t] != INF;
    }
    T dfs(int pos, T flow){
        if(pos == t) return flow;
        inq[pos] = 1;
        for(int
             &i = iter[pos]; i < adj[pos].size(); i++){
            auto [to, cap, cost] = e[adj[pos][i]];
            if(!inq[to] &&
                 dist[to] == dist[pos] + cost && cap){
                T tmp = dfs(to, min(flow, cap));
                if(tmp){
                    inq[pos] = 0;
                    e[adj[pos][i]].cap -= tmp;
                    e[adj[pos][i] ^ 1].cap += tmp;
                    return tmp;
                }
            }
        }
        inq[pos] = 0;
        return 0;
    }
    pair<T, T> mcmf(){
        T flow = 0, cost = 0;
        while(true){
            if(!spfa()) break;
            fill(iter.begin(), iter.end(), 0);
            T tmp;
            while((tmp = dfs(s, INF)) > 0){
                flow += tmp, cost += tmp * dist[t];
            }
        }
        return {flow, cost};
    }
} flow;
```

4.3 Gomory Hu

```
void Gomory_Hu_Tree(vector<int> st){
  if(st.size() <= 1) return;
  int s = st[0], t = st[1];
  flow.init(n, s, t);
  for(auto [a, b, w] : e) flow.add_edge(a, b, w);
  int cost = flow.flow();
  flow.bfs();
  adj[s].eb(t, cost), adj[t].eb(s, cost);
  vector<int> a, b;
  for(auto x : st){
    if(flow.level[x] == -1) a.pb(x);
    else b.pb(x);
  }
  Gomory_Hu_Tree(a);
  Gomory_Hu_Tree(b);
}
```

4.4 SW Min Cut

```
int edge[maxn][maxn], par[maxn], siz[maxn];
int dist[maxn], vis[maxn], done[maxn];
int n, m;
int root(int x)
{ return x == par[x] ? x : par[x] = root(par[x]); }
int contract(int &s, int &t){
  memset(dist, 0, sizeof(dist));
  memset(vis, 0, sizeof(vis));
  int mincut = INF, id, maxc;
  for(int i = 0; i < n; i++){
    id = maxc = -1;
    for(int j = 0; j < n; j++){
      if(!done[j] && !vis[j] && dist[j] > maxc){
        id = j;
        maxc = dist[j];
      }
    }
    if(id == -1) return mincut;
    s = t, t = id;
    mincut = maxc;
    vis[id] = true;
    for(int j = 0; j < n; j++){
      if(!done[j] && !vis[j]) dist[j] += edge[id][j];
    }
  }
  return mincut;
}
int Stoer_Wagner(){
  int mincut = INF, s, t, tmp;
  for(int i = 1; i < n; i++){
    tmp = contract(s, t);
    done[t] = true;
    mincut = min(mincut, tmp);
    if(!mincut) return 0;
    for(int j = 0; j < n; j++){
      if(!done
          [j]) edge[s][j] = (edge[j][s] += edge[j][t]);
    }
  }
  return mincut;
}
```

4.5 Hopcroft Karp

```
int mx[maxn], my[maxn], dx[maxn], dy[maxn], vis[maxn];
vector<int> adj[maxn];
int l, r, m;

int dfs(int pos){
    for(auto x : adj[pos]){
        if(!vis[x] && dy[x] == dx[pos] + 1){
            vis[x] = 1;
            if(my[x] != -1 && dy[x] == lim) continue;
            if(my[x] == -1 || dfs(my[x])){
                my[x] = pos, mx[pos] = x;
                return true;
            }
        }
    }
    return false;
}

int bfs(){
    fill(dx, dx + l, -1);
    fill(dy, dy + r, -1);
    queue<int> q;
    for(int i = 0; i < l; i++){
        if(mx[i] == -1) dx[i] = 0, q.push(i);
    }
    lim = INF;
    while(!q.empty()){
        int pos = q.front(); q.pop();
        if(dx[pos] > lim) break;
        for(auto x : adj[pos]){
            if(dy[x] == -1){
                dy[x] = dx[pos] + 1;
                if(my[x] == -1) lim = dy[x];
                else dx
                    [my[x]] = dy[x] + 1, q.push(my[x]);
            }
        }
    }
    return lim != INF;
}

void Hopcroft_Karp(){
    int res = 0;
    for(int i = 0; i < l; i++) mx[i] = -1;
    for(int i = 0; i < r; i++) my[i] = -1;
    while(bfs()){
        fill(vis, vis + l + r, 0);
        for(int i = 0; i < l; i++){
            if(mx[i] == -1 && dfs(i)) res++;
        }
    }
}
```

4.6 Kuhn Munkres

```
struct Hungarian{
    using T = ll;
    vector<T> lx, ly, slack;
    vector<int> vx, vy, match;
    vector<vector<T>> w;
    queue<int> q;
    int n;
    void init(int _n){
        n = _n;
        lx.resize(n), ly.resize(n), slack.resize(n);
        vx.resize
            (n), vy.resize(n), match.resize(n, -1);
        w.resize(n, vector<T>(n));
    }
    void inp(int x, int y, int val){
        w[x][y] = val;
        lx[x] = max(lx[x], val);
    }
    int dfs(int x){
        if(vx[x]) return false;
        vx[x] = 1;
        for(int i = 0; i < n; i++){
            if(lx[x] + ly[i] == w[x][i] && !vy[i]){
                vy[i] = true;
                if(match[i] == -1 || dfs(match[i])){
                    match[i] = x;
                    return true;
                }
            }

        }
        return false;
    }
    int pdfs(int x){
        fill(vx.begin(), vx.end(), 0);
        fill(vy.begin(), vy.end(), 0);
        return dfs(x);
    }
    void upd(int x){
        for(int i = 0; i < n; i++){
            if(!slack[i]) continue;
            slack[i] =
                min(slack[i], lx[x] + ly[i] - w[x][i]);
            if(!slack[i] && !vy[i]) q.push(i);
        }
    }
    void relabel(){
        T mn = numeric_limits<T>::max() / 3;
        for(int i = 0; i < n; i++){
            if(!vy[i]) mn = min(mn, slack[i]);
        }
        for(int i = 0; i < n; i++){
            if(vx[i]) lx[i] -= mn;
            if(vy[i]) ly[i] += mn;
            else{
                slack[i] -= mn;
                if(!slack[i]) q.push(i);
            }
        }
    }
    auto solve(){
        for(int i = 0; i < n; i++){
            if(pdfs(i)) continue;
            while(!q.empty()) q.pop();
            fill(slack.begin(), slack.end(), INF);
            for(int
                 j = 0; j < n; j++) if(vx[j]) upd(j);
            int ok = 0;
            while(!ok){
                relabel();
                while(!q.empty()){
                    int j = q.front(); q.pop();
                    if(match[j] == -1){
                        pdfs(i);
                        ok = 1;
                        break;
                    }
                    vy[j] = vx
                        [match[j]] = 1, upd(match[j]);
                }
            }
        }
        T ans = 0;
        for(int i = 0; i < n; i++){
            ans += w[match[i]][i];
        }
        for(int i = 0; i < n; i++) lx[match[i]] = i;
        return make_pair(ans, lx);
    }
} h;
```

4.7 General Graph Matching

```
struct Matching { // 0-based
  int n, tk;
  vector <vector <int>> g;
  vector <int> fa, pre, match, s, t;
  queue <int> q;
  int Find(int u) {
    return u == fa[u] ? u : fa[u] = Find(fa[u]);
  }
  int lca(int x, int y) {
    tk++;
    x = Find(x), y = Find(y);
    for (; ; swap(x, y)) {
      if (x != n) {
        if (t[x] == tk) return x;
        t[x] = tk;
        x = Find(pre[match[x]]);
      }
    }
  }
  void blossom(int x, int y, int l) {
    while (Find(x) != l) {
      pre[x] = y, y = match[x];
      if (s[y] == 1) q.push(y), s[y] = 0;
      if (fa[x] == x) fa[x] = l;
      if (fa[y] == y) fa[y] = l;
      x = pre[y];
    }
  }
  bool bfs(int r) {
    iota(all(fa), 0), fill(all(s), -1);
    while (!q.empty()) q.pop();
    q.push(r);
    s[r] = 0;
    while (!q.empty()) {
      int x = q.front(); q.pop();
      for (int u : g[x]) {
        if (s[u] == -1) {
          pre[u] = x, s[u] = 1;
          if (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;
  }
  int solve() {
    int res = 0;
    for (int x = 0; x < n; ++x) {
      if (match[x] == n) res += bfs(x);
    }
    return res;
  }
  void add_edge(int u, int v) {
    g[u].push_back(v), g[v].push_back(u);
  }
  Matching (int _n) : n(_n), tk(0), g(n), fa(n + 1),
    pre(n + 1, n), match(n + 1, n), s(n + 1), t(n) {}
};
```

4.8 Weighted General Graph Matching

```
struct WeightGraph { // 1-based
  static const int inf = INT_MAX;
  static const int maxn = 514;
  struct edge {
    int u, v, w;
    edge(){}
    edge(int u, int v, int w): u(u), v(v), w(w) {}
  };
  int n, n_x;
  edge g[maxn * 2][maxn * 2];
  int lab[maxn * 2];
  int match[maxn *
       2], slack[maxn * 2], st[maxn * 2], pa[maxn * 2];
  int flo_from
      [maxn * 2][maxn + 1], S[maxn * 2], vis[maxn * 2];
  vector<int> flo[maxn * 2];
  queue<int> q;
  int e_delta(const edge &e) {
      return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2; }
  void update_slack
      (int u, int x) { if (!slack[x] || e_delta(g[
      u][x]) < e_delta(g[slack[x]][x])) slack[x] = 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);
  }
  void q_push(int x) {
    if (x <= n) q.push(x);
    else for (size_t i
        = 0; i < flo[x].size(); i++) q_push(flo[x][i]);
  }
  void set_st(int x, int b) {
    st[x] = b;
    if (x > n) for (size_t i = 0;
         i < flo[x].size(); ++i) set_st(flo[x][i], b);
  }
  int get_pr(int b, int xr) {
    int pr = find(flo[
        b].begin(), flo[b].end(), xr) - flo[b].begin();
    if (pr % 2 == 1) {
      reverse(flo[b].begin() + 1, flo[b].end());
      return (int)flo[b].size() - pr;
    }
    return pr;
  }
  void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    edge e = g[u][v];
    int xr = flo_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i
        < pr; ++i) set_match(flo[u][i], flo[u][i ^ 1]);
    set_match(xr, v);
    rotate(flo[
        u].begin(), flo[u].begin() + pr, flo[u].end());
  }
  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 get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
      if (u == 0) continue;
      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 lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo
          [b].push_back(y = st[match[x]]), q_push(y);
    reverse(flo[b].begin() + 1, flo[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo
          [b].push_back(y = st[match[x]]), q_push(y);
    set_st(b, b);
    for (int x
         = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
    for (size_t i = 0; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      for (int x = 1; x <= n_x; ++x)
        if (g[b][x].w ==
            0 || e_delta(g[xs][x]) < e_delta(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 (size_t i = 0; i < flo[b].size(); ++i)
      set_st(flo[b][i], flo[b][i]);
    int xr =
        flo_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
      int xs = flo[b][i], xns = flo[b][i + 1];
      pa[xs] = g[xns][xs].u;
      S[xs] = 1, S[xns] = 0;
      slack[xs] = 0, set_slack(xns);
      q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
  }
  bool on_found_edge(const edge &e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
      pa[v] = e.u, S[v] = 1;
      int nu = st[match[v]];
      slack[v] = slack[nu] = 0;
      S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
      int lca = get_lca(u, v);
      if (!
          lca) return augment(u,v), augment(v,u), true;
      else add_blossom(u, lca, v);
    }
    return false;
  }
  bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++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 (e_delta(g[u][v]) == 0) {
              if (on_found_edge(g[u][v])) return true;
            } else update_slack(u, st[v]);
          }
      }
      int d = inf;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b]
            == b && S[b] == 1) d = min(d, lab[b] / 2);
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x]) {
          if (S[x] ==
              -1) d = min(d, e_delta(g[slack[x]][x]));
          else if (S[x] == 0)
              d = min(d, e_delta(g[slack[x]][x]) / 2);
        }
      for (int u = 1; u <= n; ++u) {
        if (S[st[u]] == 0) {
          if (lab[u] <= d) return 0;
          lab[u] -= d;
        } else if (S[st[u]] == 1) lab[u] += d;
      }
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b) {
          if (S[st[b]] == 0) lab[b] += d * 2;
          else if (S[st[b]] == 1) lab[b] -= d * 2;
        }
      q = queue<int>();
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x] && st[slack
            [x]] != x && e_delta(g[slack[x]][x]) == 0)
          if (on_found_edge
              (g[slack[x]][x])) return true;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b && S
            [b] == 1 && lab[b] == 0) expand_blossom(b);
    }
    return false;
  }
  pair<long long, int> solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int
         u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v) {
        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;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
      if (match[u] && match[u] < u)
        tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
  }
  void add_edge(int ui, int
       vi, int wi) { g[ui][vi].w = g[vi][ui].w = wi; }
  void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v)
        g[u][v] = edge(u, v, 0);
  }
};
```

4.9 Flow Models

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x,y,l,u), connect $x \to y$ with capacity $u-l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f+f' \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X,Y)
  1. Redirect every edge: $y \to x$ if $(x,y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x,y,c), connect $x \to y$ with $(cost,cap) = (c,1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost,cap) = (-c,1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost,cap) = (0,d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost,cap) = (0,-d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C+K$
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v, v \in G$ with capacity K
  4. For each edge (u,v,w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u,v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.
- Project selection problem
  1. If $p_v > 0$, create edge (s,v) with capacity p_v; otherwise, create edge (v,t) with capacity $-p_v$.
  2. Create edge (u,v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.
- Dual of minimum cost maximum flow
  1. Capacity c_{uv}, Flow f_{uv}, Cost w_{uv}, Required Flow difference for vertex b_u.
  2. If all w_{uv} are integers, then optimal solution can happen when all p_u are integers.

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

5 String

5.1 Z-Value

```
vector<int> z(string s){
    vector<int> z(s.size());
    int x = 0, y = 0;
    for(int i = 1; i < s.size(); i++){
        z[i] = max(0LL, min(z[i - x], y - i));
        while(i +
                z[i] < s.size() && s[i + z[i]] == s[z[i]]){
            x = i, y = i + z[i], z[i]++;
        }
    }
    return z;
}
```

5.2 KMP

```
vector<int> KMP(string s){
    vector<int> f(s.size());
    for(int i = 1; i < s.size(); i++){
        f[i] = f[i] - 1;
        while(f
            [i] && s[i] != s[f[i]]) f[i] = f[f[i] - 1];
        if(s[f[i]] == s[i]) f[i]++;
    }
    return f;
}
```

5.3 Manacher

```
vector<int> manacher(string s){
    int n = 2 * s.size() + 1;
    string ss(n, '#');
    for(int
         i = 0; i < n / 2; i++) ss[i * 2 + 1] = s[i];
    swap(s, ss);
    vector<int> f(n);
    int m = 0, len = 0;
    for(int i = 0; i < n; i++){
        f[i]
            = max(0LL, min(f[m + m - i], m + len - i));
        while(i + f[i] < n && i
            - f[i] >= 0 && s[i + f[i]] == s[i - f[i]]){
            m = i, len = f[i], f[i]++;
        }
    }
    return f;
}
```

5.4 Suffix Array

```
struct SuffixArray{
    int ch[2][maxn], sa[maxn], cnt[maxn], n;
    string s;
    void init(string _s){
        s = _s, n = s.size();
        Get_SA();
        Get_LCP();
    }
    void Get_SA(){
        int *x = ch[0], *y = ch[1], m = 256;
        for(int i = 0; i < m; i++) cnt[i] = 0;
        for(int i = 0; i < n; i++) cnt[x[i] = s[i]]++;
        for(int
             i = 1; i < m; i++) cnt[i] += cnt[i - 1];
        for(int i = 0; i < n; i++) sa[--cnt[x[i]]] = i;
        for(int k = 1;; k <<= 1){
            for(int i = 0; i < m; i++) cnt[i] = 0;
            for(int i = 0; i < n; i++) cnt[x[i]]++;
            for(int i
                 = 1; i < m; i++) cnt[i] += cnt[i - 1];
            int p = 0;
            for(int i = n - k; i < n; i++) y[p++] = i;
            for(int i = 0; i < n;
                i++) if(sa[i] >= k) y[p++] = sa[i] - k;
            for(int i = n - 1;
                i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];
            y[sa[0]] = p = 0;
            for(int i = 1; i < n; i++){
                int a = sa[i], b = sa[i - 1];
                if(a + k < n && b + k < n && x[a
                    ] == x[b] && x[a + k] == x[b + k]);
                else p++;
                y[a] = p;
            }
            if(p == n - 1) break;
            swap(x, y);
            m = p + 1;
        }
    }
    int rnk[maxn], lcp[maxn];
    void Get_LCP(){
        for(int i = 0; i < n; i++) rnk[sa[i]] = i;
        int val = 0;
        for(int i = 0; i < n; i++){
            if(val) val--;
            if(!rnk[i]){
                lcp[0] = val = 0;
                continue;
            }
            int b = sa[rnk[i] - 1];
            while(b + val < n && i + val
                < n && s[b + val] == s[i + val]) val++;
            lcp[rnk[i]] = val;
        }
    }
} sa;
```

5.5 SAIS

```
int sa[N << 1], rk[N], lcp[N];
// string ASCII value need > 0
namespace sfx {
bool _t[N << 1];
int _s[N << 1], _c[N << 1], x[N], _p[N], _q[N << 1];
void pre(int *sa, int *c, int n, int z) {
  fill_n(sa, n, 0), copy_n(c, z, x);
}
void induce
    (int *sa, int *c, int *s, bool *t, int n, int z) {
  copy_n(c, z - 1, x + 1);
  for (int i = 0; i < n; ++i)
    if (sa[i] && !t[sa[i] - 1])
      sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
  copy_n(c, z, x);
  for (int i = n - 1; i >= 0; --i)
    if (sa[i] && t[sa[i] - 1])
      sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
}
void sais(int *s, int *sa
    , int *p, int *q, bool *t, int *c, int n, int z) {
  bool uniq = t[n - 1] = true;
  int nn = 0,
      nmxz = -1, *nsa = sa + n, *ns = s + n, last = -1;
  fill_n(c, z, 0);
  for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
  partial_sum(c, c + z, c);
  if (uniq) {
    for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
    return;
  }
  for (int i = n - 2; i >= 0; --i)
    if (s[i] == s[i + 1]) t[i] = t[i + 1];
    else t[i] = s[i] < s[i + 1];
  pre(sa, c, n, z);
  for (int i = 1; i <= n - 1; ++i)
    if (t[i] && !t[i - 1])
      sa[--x[s[i]]] = p[q[i] = nn++] = i;
  induce(sa, c, s, t, n, z);
  for (int i = 0; i < n; ++i)
    if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
      bool neq = last < 0 || !equal
          (s + sa[i], s + p[q[sa[i]] + 1], s + last);
      ns[q[last = sa[i]]] = nmxz += neq;
    }
  sais(ns,
       nsa, p + nn, q + n, t + n, c + z, nn, nmxz + 1);
  pre(sa, c, n, z);
  for (int i = nn - 1; i >= 0; --i)
    sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
  induce(sa, c, s, t, n, z);
}
void buildSA(string s) {
  int n = s.length();
  for (int i = 0; i < n; ++i) _s[i] = s[i];
  _s[n] = 0;
  sais(_s, sa, _p, _q, _t, _c, n + 1, 256);
  for (int i = 1; i <= n; ++i) sa[i - 1] = sa[i];
} // buildLCP()...
}
```

5.6 Suffix Automaton

```
struct SuffixAutomaton{
  int len[maxn], link[maxn]; // maxn >= 2 * n - 1
    map<char, int> nxt[maxn];
    int cnt[maxn], distinct[maxn];
  bool is_clone[maxn];
    int first_pos[maxn];
    vector<int> inv_link[maxn]; //suffix references
  int sz = 1, last = 0;
  void init(string s){
    link[0] = -1;
    for(auto x : s) sa_extend(x);
  }
  void sa_extend(char c){
    int cur = sz++;
        cnt[cur] = 1;
    len[cur] = len[last] + 1;
    first_pos[cur] = len[cur] - 1;
  int p = last;
  while(p != -1 && !nxt[p].count(c)){
    nxt[p][c] = cur;
    p = link[p];
  }
  if(p == -1) link[cur] = 0;
  else{
    int q = nxt[p][c];
    if(len[q] == len[p] + 1) link[cur] = q;
    else{
      int clone = sz++;
      is_clone[clone] = true;
      first_pos[clone] = q;
      len[clone] = len[p] + 1;
      nxt[clone] = nxt[q];
      link[clone] = link[q];
      while(p != -1 && nxt[p][c] == q) {
        nxt[p][c] = clone;
        p = link[p];
      }
      link[cur] = link[q] = clone;
    }
  }
  last = cur;
}
ll getDistinct(int pos){ // number
     of distinct substr. starting at pos(inc. empty)
  if(distinct[pos]) return distinct[pos];
  distinct[pos] = 1;
  for(auto [c, next]
      : nxt[pos]) distinct[pos] += getDistinct(next);
  return cnt[pos];
}
ll numDistinct(){
  return getDistinct
      (0) - 1; // excluding an empty string
}
ll numDistinct2(){
  ll tot = 0;
  for(int i
      = 1; i < sz; i++) tot += len[i] - len[link[i]];
  return tot;
}
  void compute_cnt(){ // endpos set size
  vector<vector<int>> v(sz);
  for(int i = 1; i < sz; i++) v[len[i]].pb(i);
  for(int
       i = sz - 1; i > 0; i--) for(auto x : v[i]) {
    cnt[link[x]] += cnt[x];
  }
  }
string distinct_kth(ll k){
      // substring
           kth (not distinct) -> compute_cnt()
  numDistinct();
  string s;
  ll cur = 0, tally = 0;
  while(tally < k){
    for(auto [c, next] : nxt[cur]){
      if(tally + distinct[next] >= k){
        tally += 1;
        s += c;
        cur = next;
        break;
      }
      tally += distinct[next];
    }
  }
  return s;
}
//inverse links
void genLink(){
    for(int i = 1; i < sz; i++){
          inv_link[link[i]].pb(i);
    }
}
void get_all_occur(vector<int>& oc, int v){
    if(!is_clone[v]) oc.pb(first_pos[v]);
    for(auto u : inv_link[v]) get_all_occur(oc, u);
}
vector<int> all_occ(string s){ // get all occ of s
    int cur = 0;
    for(auto x : s){
        if(!nxt[cur].count(x)) return {};
        cur = nxt[cur][x];
    }
```

```
      vector<int> oc;
      get_all_occur(oc, cur);
      for(auto &x : oc
          ) x += 1 - s.length(); // starting positions
      sort(oc.begin(), oc.end());
      return oc;
    }
    int lcs(string t){
      int v = 0, l = 0, ans = 0;
      for(auto x : t){
        while(v && !nxt[v].count(x)){
          v = link[v];
          l = len[v];
        }
        if(nxt[v].count(x)){
          v = nxt[v][x];
          l++;
        }
        ans = max(ans, l);
      }
      return ans;
    }
};
```

5.7 Palindrome Tree

```
struct EERTREE{
  int sz, tot, last;
  int cnt[maxn], ch[maxn][26],
       len[maxn], fail[maxn], dif[maxn], slink[maxn];
  int g[maxn], dp[maxn];
  char s[maxn];
  int node(int l){
    sz++;
    memset(ch[sz], 0, sizeof(ch[sz]));
    len[sz] = l;
    fail[sz] = cnt[sz] = 0;
    return sz;
  }
  void init(){
    sz = -1;
    last = 0;
    s[tot = 0] = '$';
    node(0);
    node(-1);
    fail[0] = 1;
  }
  int getfail(int x){
    while(s[tot - len[x] - 1] != s[tot]) x = fail[x];
    return x;
  }
  void insert(char c){
    s[++tot] = c;
    int now = getfail(last);
    if(!ch[now][c - 'a']){
      int x = node(len[now] + 2);
      fail[x] = ch[getfail(fail[now])][c - 'a'];
      ch[now][c - 'a'] = x;
      dif[x] = len[x] - len[fail[x]];
      if(dif[x] == dif[fail[x]]){
        slink[x] = slink[fail[x]];
      }
      else slink[x] = fail[x];
    }
    last = ch[now][c - 'a'];
    cnt[last]++;
  }
  int process
      (string s){ // minimum palindrome partitioning
    for(int i = 0; i < s.size(); i++){
      insert(s[i]);
      dp[i] = INF;
      for(int x = last; x > 1; x = slink[x]){
        if(i - len[slink[x]] - dif[x] >=
             0) g[x] = dp[i - len[slink[x]] - dif[x]];
        if(dif[x] ==
            dif[fail[x]]) g[x] = min(g[x], g[fail[x]]);
        dp[i] = min(dp[i], g[x] + 1);
      }
    }
    return dp[s.size() - 1];
  }
} pam;
```

5.8 AC Automaton

```
namespace AC{
  int ch[maxn][26],
       fail[maxn], idx[maxn], last[maxn], pt[maxn];
  int val[maxn], cnt[maxn], tot = 0;
    // val[i] = # of times node
        (i) is visited, cnt[i] = # of occ. of str(i)
  void init(){
    memset(ch,
         0, sizeof(ch)), memset(fail, 0, sizeof(fail));
    memset(idx,
        0, sizeof(idx)), memset(last, 0, sizeof(last));
    memset(val
        , 0, sizeof(val)), memset(cnt, 0, sizeof(cnt));
    tot = 0;
  }
  void insert(string &s, int id){ // id is 1-based
    int cur = 0;
    for(int i = 0; i < s.size(); i++){
      if(!ch[cur
          ][s[i] - 'a']) ch[cur][s[i] - 'a'] = ++tot;
      cur = ch[cur][s[i] - 'a'];
    }
    if(idx[cur] == 0) idx[cur] = id;
    else pt[id] = idx[cur];
  }
  void build(){
    queue<int> q;
    for(int i = 0; i < 26; i++){
      if(ch[0][i]) q.push(ch[0][i]);
    }
    while(!q.empty()){
      int u = q.front(); q.pop();
      for(int i = 0; i < 26; i++){
        if(ch[u][i]) {
          fail[ch[u][i]] = ch[fail[u]][i];
          q.push(ch[u][i]);
        }
        else ch[u][i] = ch[fail[u]][i];
        last[ch[u][i]] = idx[fail[ch[u][i]]]
             ? fail[ch[u][i]] : last[fail[ch[u][i]]];
      }
    }
  }
  int qry(string &s){
    int u = 0, ret = 0;
    for(int i = 0; i < s.size(); i++){
      u = ch[u][s[i] - 'a'];
      for(int j = u; j; j = last[j]) val[j] ++;
    }
    for(int i = 0; i <= tot; i++){
      if(idx[i])
          ret = max(ret, val[i]), cnt[idx[i]] = val[i];
    }
    return ret;
  }
};
```

5.9 Lyndon Factorization

```
vector<string> duval(string s){
  int n = s.length(), i = 0;
    vector<string> fac;
  while(i < n){
    int j = i + 1, k = i; // i <= k < j
    while(j < n && s[k] <= s[j]){
      if(s[k] < s[j]) k = i;
      else k++;
      j++;
    }
    while(i <= k){
      fac.pb(s.substr(i, j - k));
      i += j - k;
    }
  }
  return fac;
}
```

6 Math

6.1 Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 fpow(u64 a, u64 b, u64 n){
  u64 ret = 1;
  while(b > 0){
    if(b & 1) ret = (u128)ret * a % n;
    a = (u128)a * a % n;
    b >>= 1;
  }
  return ret;
}
bool check_composite(u64 n, u64 a, u64 d, int s){
  u64 x = fpow(a, d, n);
  if(x == 1 || x == n - 1) return false;
  for(int r = 1; r < s; r++){
    x = (u128)x * x % n;
    if(x == n - 1) return false;
  }
  return true;
}
bool MillerRabin(u64 n){
  if(n < 2) return false;
  int s = 0;
  u64 d = n - 1;
  while(!(d & 1)){
    d >>= 1;
    s++;
  }
  for(auto a : {2, 3, 5, 7, 11, 13, 17,
       19, 23, 29, 31, 37}){ // sufficient for n < 2^64
    if(n == a) return true;
    if(check_composite(n, a, d, s)) return false;
  }
  return true;
}
```

6.2 Pollard Rho

```
ll f(ll t, ll c, ll n){
  return (t * t + c) % n;
}

ll Pollard_Rho(ll x){
  ll t = 0;
  ll c = rand() % (x - 1) + 1;
  ll s = t;
  ll val = 1;
  for(int goal = 1;; goal <<= 1, s = t, val = 1){
    for(int step = 1; step <= goal; step++){
      t = f(t, c, x);
      val = val * abs(t - s) % x;
      if(!val) return x;
      if(step % 127 == 0){
        ll d = __gcd(val, x);
        if(d > 1) return d;
      }
    }
    ll d = __gcd(val, x);
    if(d > 1) return d;
  }
}
```

6.3 EXT GCD

```
ll extgcd(ll a, ll b, ll &x, ll &y){
  if(b == 0){
    x = 1, y = 0;
    return a;
  }
  int res = extgcd(b, a % b, y, x);
  y -= (a / b) * x;
  return res;
}
```

6.4 Chinese Remainder Theorem

```
ll CRT(vector<ll> p, vector<ll> a){
  ll n = p.size(), prod = 1, ret = 0;
  for(int i = 0; i < n; i++) prod *= p[i];
  for(int i = 0; i < n; i++){
    ll m = (prod / p[i]);
    ll x, y;
    extgcd(m, p[i], x, y);
    ret = ((ret + a[i] * m * x) % prod + prod) % prod;
  }
  return ret;
}
```

6.5 Powerful Number Sieve

```
void linearsieve(){
  phi[1] = 1;
  for(int i = 2; i < maxn; i++){
    if(!lp[i]) pr.pb(i), lp[i] = i, phi[i] = i - 1;
    for(auto x : pr){
      if(i * x >= maxn) break;
      lp[i * x] = x;
      if(lp[i] == x){
        phi[i * x] = phi[i] * x;
        break;
      }
      phi[i * x] = phi[i] * (x - 1);
    }
  }
  for(int i = 1; i < maxn
      ; i++) sum[i] = (sum[i - 1] + i * phi[i]) % N;
}

int s2(int n){
  static const int inv6 = inv(6);
  n %= N;
  return n * (n + 1) % N * (2 * n + 1) % N * inv6 % N;
}

int G(int n){
  static const int inv2 = inv(2);
  if(n < maxn) return sum[n];
  if(mp_G.count(n)) return mp_G[n];
  int ans = s2(n);
  for(int i = 2, j; i <= n; i = j + 1){
    j = n / (n / i);
    (ans -= (i + j) % N * (j -
        i + 1) % N * inv2 % N * G(n / i) % N - N) %= N;
  }
  return mp_G[n] = ans;
}

void dfs(int d, int hd, int p){ // dfs 出所有 PN
  (ans += hd * G(n / d)) %= N;
  for(int i = p; i < pr.size(); i++){
    if(d > n / pr[i] / pr[i]) break;
    int c = 2;
    for(int x
         = d * pr[i] * pr[i]; x <= n; x *= pr[i], c++){
      if(!vis[i][c]){
        int f = fpow(pr[i], c);
        f = f * (f - 1) % N;
        int g = pr[i] * (pr[i] - 1) % N;
        int t = pr[i] * pr[i] % N;
        for(int j = 1; j <= c; j++){
          (f -= g * h[i][c - j] % N - N) %= N;
          (g *= t) %= N;
        }
        h[i][c] = f;
        vis[i][c] = true;
      }
      if(h[i][c]) dfs(x, hd * h[i][c] % N, i + 1);
    }
  }
}

linearsieve();
for(int i = 0; i < pr.size(); i++) h[i][0] = 1;
dfs(1, 1, 0);
```

6.6 Min25 Sieve

```
template <typename U, typename V> struct min25 {
  lld n; int sq;
  vector<U> Ss, Sl, Spre; vector<V> Rs, Rl;
  Sieve sv; vector<lld> quo;
  U &S(lld d) { return d < sq ? Ss[d] : Sl[n / d]; }
  V &R(lld d) { return d < sq ? Rs[d] : Rl[n / d]; }
  min25(lld n_) : n(n_), sq((int)sqrt(n) + 1),
    Ss(sq), Sl(sq), Spre(sq), Rs(sq), Rl(sq), sv(sq) {
    for (lld i = 1, Q; i <= n; i = n / Q + 1)
      quo.push_back(Q = n / i);
  }
  U F_prime(auto &&f, auto &&F) {
    for (lld p : sv.primes) Spre[p] = f(p);
    for (
        int i = 1; i < sq; i++) Spre[i] += Spre[i - 1];
    for (lld i : quo) S(i) = F(i) - F(1);
    for (lld p : sv.primes)
      for (lld i : quo) {
        if (p * p > i) break;
        S(i) -= f(p) * (S(i / p) - Spre[p - 1]);
      }
    return S(n);
  } // F_prime: \sum _ {p is prime, p <= n} f(p)
  V F_comp(auto &&g) {
    for (lld i : quo) R(i) = V(S(i));
    for (lld p : sv.primes | views::reverse)
      for (lld i : quo) {
        if (p * p > i) break;
        lld prod = p;
        for (
            int c = 1; prod * p <= i; ++c, prod *= p) {
          R(i) += g(p, c) * (R(i / prod) - V(Spre[p]));
          R(i) += g(p, c + 1);
        }
      }
    return R(n);
  } // F_comp: \sum _ {2 <= i <= n} g(i)
}; // O(n^{3/4} / log n)
/* U, V 都是環, 記 h: U -> V 代表 U 轉型成 V 的函數。
要求 h(x + y) = h(x) + h(y); f: lld -> U 是完全積性;
g 是積性函數且 h(f(p)) = g(p) 對於質數 p。
呼叫 F_comp 前需要先呼叫 F_prime 得到 S(i)。
S(i), R(i) 是 F_prime 和 F_comp 在 n/k 點的值。
F(i) = \sum _ {j <= i} f(j) 和 f(i) 需要快速求值。
g(p, c) := g(pow(p, c)) 需要快速求值。
例如若 g(p) 是度數 d 的多項式則可以構造 f(p) 是維護
pow(p, c) 的 (d+1)-tuple */
```

6.7 Floor Sum

```
//f(n, a, b, c) = sum_{0<=i<=n}{(ai + b)/c},
//g(n, a, b, c) = sum_{0<=i<=n}{i(ai + b)/c},
//h(n, a, b, c) = sum_{0<=i<=n}{((ai + b)/c)^2},
const int N = 998244353;
const int i2 = (N + 1) / 2, i6 = 166374059;
struct info{
  ll f, g, h;
  info(){f = g = h = 0;}
};
info calc(ll n, ll a, ll b, ll c){
  ll ac = a / c, bc = b / c,
      m = (a * n + b) / c, n1 = n + 1, n21 = n * 2 + 1;
  info d;
  if(a == 0){
    d.f = bc * n1 % N;
    d.g = bc * n % N * n1 % N * i2 % N;
    d.h = bc * bc % N * n1 % N;
    return d;
  }
  if(a >= c || b >= c){
    d.f = n * n1 % N * i2 % N * ac % N + bc * n1 % N;
    d.g = ac * n % N * n1 % N * n21
         % N * i6 % N + bc * n % N * n1 % N * i2 % N;
    d.h = ac * ac
        % N * n % N * n1 % N * n21 % N * i6 % N + bc *
        bc % N * n1 % N + ac * bc % N * n % N * n1 % N;

    info e = calc(n, a % c, b % c, c);

    d.h +=
         e.h + 2 * bc * e.f % N + 2 * ac % N * e.g % N;
    d.g += e.g, d.f += e.f;
    d.f %= N, d.g %= N, d.h %= N;
    return d;
  }
  info e = calc(m - 1, c, c - b - 1, a);
  d.f = (n * m % N - e.f + N) % N;
  d.g = m * n % N *
      n1 % N - e.h - e.f; d.g = (d.g * i2 % N + N) % N;
  d.h = n * m % N * (m + 1) % N -
      2 * e.g - 2 * e.f - d.f; d.h = (d.h % N + N) % N;
  return d;
}
```

6.8 Euclidean

$m=\lfloor\frac{an+b}{c}\rfloor$

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

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

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

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

6.9 Big Number

```
template<typename T>
inline string to_string(const T& x){
  stringstream ss;
  return ss<<x,ss.str();
}
struct bigN:vector<ll>{
  const static int base=1000000000,width=log10(base);
  bool negative;
  bigN(const_iterator
       a,const_iterator b):vector<ll>(a,b){}
  bigN(string s){
    if(s.empty())return;
    if(s[0]=='-')negative=1,s=s.substr(1);
    else negative=0;
    for(int i=int(s.size())-1;i>=0;i-=width){
      ll t=0;
      for(int j=max(0,i-width+1);j<=i;++j)
        t=t*10+s[j]-'0';
      push_back(t);
    }
    trim();
  }
  template<typename T>
    bigN(const T &x):bigN(to_string(x)){}
  bigN():negative(0){}
  void trim(){
    while(size()&&!back())pop_back();
    if(empty())negative=0;
  }
  void carry(int _base=base){
    for(size_t i=0;i<size();++i){
      if(at(i)>=0&&at(i)<_base)continue;
      if(i+1u==size())push_back(0);
      int r=at(i)%_base;
      if(r<0)r+=_base;
      at(i+1)+=(at(i)-r)/_base,at(i)=r;
    }
  }
  int abscmp(const bigN &b)const{
    if(size()>b.size())return 1;
    if(size()<b.size())return -1;
    for(int i=int(size())-1;i>=0;--i){
      if(at(i)>b[i])return 1;
      if(at(i)<b[i])return -1;
    }
    return 0;
  }
  int cmp(const bigN &b)const{
    if(negative!=b.negative)return negative?-1:1;
    return negative?-abscmp(b):abscmp(b);
  }
  bool operator<(const bigN&b)const{return cmp(b)<0;}
  bool operator>(const bigN&b)const{return cmp(b)>0;}
  bool operator<=(const bigN&b)const{return cmp(b)<=0;}
  bool operator>=(const bigN&b)const{return cmp(b)>=0;}
  bool operator==(const bigN&b)const{return !cmp(b);}
  bool operator!=(const bigN&b)const{return cmp(b)!=0;}
  bigN abs()const{
    bigN res=*this;
    return res.negative=0, res;
  }
  bigN operator-()const{
    bigN res=*this;
    return res.negative=!negative,res.trim(),res;
  }
  bigN operator+(const bigN &b)const{
    if(negative)return -(-(*this)+(-b));
    if(b.negative)return *this-(-b);
    bigN res=*this;
    if(b.size()>size())res.resize(b.size());
    for(size_t i=0;i<b.size();++i)res[i]+=b[i];
  return res.carry(),res.trim(),res;
 }
 bigN operator-(const bigN &b)const{
  if(negative)return -(-(*this)-(-b));
  if(b.negative)return *this+(-b);
  if(abscmp(b)<0)return -(b-(*this));
  bigN res=*this;
  if(b.size()>size())res.resize(b.size());
  for(size_t i=0;i<b.size();++i)res[i]-=b[i];
  return res.carry(),res.trim(),res;
 }
 bigN operator*(const bigN &b)const{
  bigN res;
  res.negative=negative!=b.negative;
  res.resize(size()+b.size());
  for(size_t i=0;i<size();++i)
   for(size_t j=0;j<b.size();++j)
    if((res[i+j]+=at(i)*b[j])>=base){
     res[i+j+1]+=res[i+j]/base;
     res[i+j]%=base;
    }//¼ªk¥Îcarry·|·¸¦ì
  return res.trim(),res;
 }
 bigN operator/(const bigN &b)const{
  int norm=base/(b.back()+1);
  bigN x=abs()*norm;
  bigN y=b.abs()*norm;
  bigN q,r;
  q.resize(x.size());
  for(int i=int(x.size())-1;i>=0;--i){
   r=r*base+x[i];
   int s1=r.size()<=y.size()?0:r[y.size()];
   int s2=r.size()<y.size()?0:r[y.size()-1];
   int d=(ll(base)*s1+s2)/y.back();
   r=r-y*d;
   while(r.negative)r=r+y,--d;
   q[i]=d;
  }
  q.negative=negative!=b.negative;
  return q.trim(),q;
 }
 bigN operator%(const bigN &b)const{
  return *this-(*this/b)*b;
 }
 friend istream& operator>>(istream &ss,bigN &b){
  string s;
  return ss>>s, b=s, ss;
 }
 friend
    ostream& operator<<(ostream &ss,const bigN &b){
  if(b.negative)ss<<'-';
  ss<<(b.empty()?0:b.back());
  for(int i=int(b.size())-2;i>=0;--i)
   ss<<setw(width)<<setfill('0')<<b[i];
  return ss;
 }
 template<typename T>
  operator T(){
   stringstream ss;
   ss<<*this;
   T res;
   return ss>>res,res;
  }
};
```

6.10 Determinant

```
struct Matrix {
 int n, m;
 ll M[MAXN][MAXN];
 int row_swap(int i, int j) {
  if (i == j) return 0;
  for (int k = 0; k < m; ++k)
   swap(M[i][k], M[j][k]);
  return 1;
 }
 ll det() { // return the number of swaps
  int rt = 0;
  for (int i = 0; i < n; ++i) {
   int piv = i;
   while (piv < n && !M[piv][i]) ++piv;
   if (piv == n) continue;
   rt += row_swap(i, piv);
   for (int j = i + 1; j < n; ++j) {
    while (M[j][i]) {
     int tmp = P - M[i][i] / M[j][i];
     for (int k = i; k < m; ++k)
      M[i][k] = (M[j][k] * tmp + M[i][k]) % P;
     rt += row_swap(i, j);
    }
   }
  }
  rt = (rt & 1) ? P - 1 : 1;
  for (int i = 0; i < n; ++i)
   rt = rt * M[i][i] % P;
  return rt;
  // round(rt) if using double to cal. int. det
 }
};
```

6.11 Discrete Log

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

6.12 Berlekamp Massey

```
// need add, sub, mul
vector <int> BerlekampMassey(vector <int> a) {
 // find min |c|
    such that a_n = sum c_j * a_{n - j - 1}, 0-based
 // O(N^2), if |c| = k, |a| >= 2k sure correct
 auto f = [&](vector<int> v, ll c) {
  for (int &x : v) x = mul(x, c);
  return v;
 };
 vector <int> c, best;
 int pos = 0, n = (int)a.size();
 for (int i = 0; i < n; ++i) {
  int error = a[i];
  for (int j = 0; j < (int)c.size(); ++j)
   error = sub(error, mul(c[j], a[i - 1 - j]));
  if (error == 0) continue;
  int inv = Pow(error, mod - 2);
  if (c.empty()) {
   c.resize(i + 1), pos = i, best.pb(inv);
  } else {
   vector <int> fix = f(best, error);
   fix.insert(fix.begin(), i - pos - 1, 0);
   if (fix.size() >= c.size()) {
    best = f(c, sub(0, inv));
    best.insert(best.begin(), inv);
    pos = i, c.resize(fix.size());
   }
   for (int j = 0; j < (int)fix.size(); ++j)
    c[j] = add(c[j], fix[j]);
  }
 }
 return c;
}
```

6.13 Gussian Elimination

```
using VI = vector<int>; // be careful if A.empty()
using VVI = vector<VI>; // ensure that 0 <= x < mod
pair<VI, VVI> gauss(VVI A, VI b) { // solve Ax=b
 const int N = (int)A.size(), M = (int)A[0].size();
 vector<int> depv, free(M, true); int rk = 0;
 for (int i = 0; i < M; i++) {
    int p = -1;
    for (int j = rk; j < N; j++)
      if (p == -1 || abs(A[j][i]) > abs(A[p][i]))
        p = j;
    if (p == -1 || A[p][i] == 0) continue;
    swap(A[p], A[rk]); swap(b[p], b[rk]);
    const int inv = modinv(A[rk][i]);
    for (int &x : A[rk]) x = mul(x, inv);
    b[rk] = mul(b[rk], inv);
    for (int j = 0; j < N; j++) if (j != rk) {
      int z = A[j][i];
      for (int k = 0; k < M; k++)
        A[j][k] = sub(A[j][k], mul(z, A[rk][k]));
      b[j] = sub(b[j], mul(z, b[rk]));
    }
    depv.push_back(i); free[i] = false; ++rk;
  }
  for (int i = rk; i < N; i++)
    if (b[i] != 0) return {{}, {}}; // not consistent
  VI x(M); VVI h;
  for (int i = 0; i < rk; i++) x[depv[i]] = b[i];
  for (int i = 0; i < M; i++) if (free[i]) {
    h.emplace_back(M); h.back()[i] = 1;
    for (int j = 0; j < rk; j++)
      h.back()[depv[j]] = sub(0, A[j][i]);
  }
  return {x, h}; // solution = x + span(h[i])
}
```

6.14 Golden Search

```
llf gss(llf a, llf b, auto &&f) {
  llf r = (sqrt(5)-1)/2, eps = 1e-7;
  llf x1 = b - r*(b-a), x2 = a + r*(b-a);
  llf f1 = f(x1), f2 = f(x2);
  while (b-a > eps)
    if (f1 < f2) { //change to > to find maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - r*(b-a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
}
```

6.15 Pi Count

```
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
```

6.16 Quadratic Residue

```
int get_root(int n, int P) { // ensure 0 <= n < p
  if (P == 2 or n == 0) return n;
  auto check = [&](lld x) {
    return modpow(int(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 q = (P + 1) / 2; q; q >>= 1, e = M(e, e))
    if (q & 1) r = M(r, e);
  return
       int(r.first); // sqrt(n) mod P where P is prime
}
```

6.17 Simplex

```
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;
}} // use double instead of long double if possible
```

6.18 Simplex Construction

Standard form: maximize $\sum_{1\le i\le n} c_i x_i$ such that $\sum_{1\le i\le n} A_{ji}x_i \le b_j$ for all $1\le j\le m$ 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. $\sum_{1\le i\le n} A_{ji}x_i = b_j \to$ add $\le$ and $\ge$.
4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.19 Theorem

- Kirchhoff's Theorem
  Denote L be a $n\times n$ matrix as the Laplacian matrix of graph G, where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i,j) in G.
  - The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
  - The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.
- Tutte's Matrix
  Let D be a $n\times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i<j$ and $(i,j)\in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.
- Cayley's Formula
  - Given a degree sequence $d_1, d_2, \ldots, d_n$ for each *labeled* vertices, there are
    $$\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$$
    spanning trees.
  - Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1,2,\ldots,k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.
- Erdős–Gallai Theorem
  A sequence of non-negative integers $d_1 \ge d_2 \ge \ldots \ge d_n$ 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
  $$\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$
  holds for all $1\le k\le n$.
- Burnside's Lemma
  Let X be a set and G be a group that acts on X. For $g\in G$, denote by X^g the elements fixed by g:
  $$X^g = \{x\in X \mid gx \in X\}$$
  Then
  $$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|.$$
- Gale–Ryser theorem
  A pair of sequences of nonnegative integers $a_1 \ge \cdots \ge a_n$ and $b_1, \ldots, b_n$ is bigraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \le \sum_{i=1}^{n}\min(b_i, k)$ holds for every $1\le k\le n$. Sequences a and b called bigraphic if there is a labeled simple bipartite graph such that a and b is the degree sequence of this bipartite graph.
- Fulkerson–Chen–Anstee theorem
  A sequence $(a_1,\ b_1)$, $\ldots$, $(a_n,\ b_n)$ of nonnegative integer pairs with $a_1 \ge \cdots \ge a_n$ is digraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \le \sum_{i=1}^{k}\min(b_i, k-1) + \sum_{i=k+1}^{n}\min(b_i, k)$ holds for every $1\le k\le n$. Sequences a and b called digraphic if there is a labeled simple directed graph such that each vertex v_i has indegree a_i and outdegree b_i.
- Pick's theorem
  For simple polygon, when points are all integer, we have $A = \#\{\text{lattice points in the interior}\} + \frac{\#\{\text{lattice points on the boundary}\}}{2} - 1$
- Möbius inversion formula
  - $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n}\mu(d) f(\frac{n}{d})$
  - $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d}\mu(\frac{d}{n}) f(d)$

7 Polynomial

7.1 NTT

```
const int N = 998244353, g = 3;

void NTT(vector
    <ll> &a, bool invert = 0){ // interative version
  int n = a.size();
  int lg_n = __lg(n);
  for(int i = 1, j = 0; i < n; i++){
    int bit = n >> 1;
    for(; j & bit; bit >>= 1) j ^= bit;
    j ^= bit;
    if(i < j) swap(a[i], a[j]);
  }
  for(int len = 2; len <= n; len <<= 1){
    ll wn = fpow(g, (N - 1) / len);
    if(invert) wn = inv(wn);
    for(int i = 0; i < n; i += len){
      ll w = 1;
      for(int j = 0; j < len / 2; j++){
        ll u
            = a[i + j], v = a[i + j + len / 2] * w % N;
        a[i + j] = (u + v) % N;
        a[i + j + len / 2] = (u - v + N) % N;
        (w *= wn) %= N;
      }
    }
  }
  ll n_1 = inv(n);
  if(invert) for(auto &x : a) (x *= n_1) %= N;
}
```

7.2 FFT

```
using cd = complex<double>;
const double PI = acos(-1);

void FFT(vector
    <cd> &a, bool invert = 0){ // interative version
  int n = a.size();
  int lg_n = __lg(n);
  for(int i =
      1, j = 0; i < n; i++){ //bit-reversal permutation
    int bit = n >> 1;
    for(; j & bit; bit >>= 1) j ^= bit;
    j ^= bit;
    if(i < j) swap(a[i], a[j]);
  }
  for(int len = 2; len <= n; len <<= 1){
    double ang = 2 * PI / len * (invert? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for(int i = 0; i < n; i += len){
      cd w(1);
      for(int j = 0; j < len / 2; j++){
        cd u = a[i + j], v = a[i + j + len / 2] * w;
        a[i + j] = u + v;
        a[i + j + len / 2] = u - v;
        w *= wlen;
      }
    }
  }
  if(invert) for(auto &x : a) x /= n;
}
```

7.3 Primes

Prime	Root	Prime	Root
7681	17	167772161	3
12289	11	104857601	3
40961	3	985661441	3
65537	3	998244353	3
786433	10	1107296257	10
5767169	3	2013265921	31
7340033	3	2810183681	11
23068673	3	2885681153	3
469762049	3	605028353	3
2061584302081	7	1945555039024054273	5
2748779069441	3	9223372036737335297	3

7.4 Fast Walsh Transform

```
void fwt(vector<int> &a, bool inv){
  int n = 1;
  while(n < a.size()) n *= 2;
  a.resize(n);
  for(int len = 1; 2 * len <= n; len <<= 1){
    for(int i = 0; i < n; i += 2 * len){
      for(int j = 0; j < len; j++){
        int &u =
          a[i + j], &v = a[i + j + len]; tie(u, v) =
        // inv ? pll(u - v, v) : pll(u + v, v); // and
        // inv ? pll(u, v - u) : pll(u, u + v); // or
        pll(u + v, u - v); // xor
      }
    }
  }
  if(inv) for(auto &x : a) x /= n; // xor only
}
```

7.5 Fast Liear Recursion

```
int FastLinearRecursion
    (vector <int> a, vector <int> c, ll k) {
  // a_n = sigma c_j * a_{n - j - 1}, 0-based
  // O(NlogNlogK), |a| = |c|
  int n = a.size();
  if (k < n) return a[k];
  vector <int> base(n + 1, 1);
  for (int i = 0; i < n; ++i)
    base[i] = sub(0, c[n - i - 1]);
  vector <int> poly(n);
  (n == 1 ? poly[0] = c[n - 1] : poly[1] = 1);
  auto calc = [&](vector <int> p1, vector <int> p2) {
    // O(n^2) bruteforce or O(nlogn) NTT
    return Divide(Mul(p1, p2), base).second;
  };
  vector <int> res(n, 0); res[0] = 1;
  for (; k; k >>= 1, poly = calc(poly, poly)) {
    if (k & 1) res = calc(res, poly);
  }
  int ans = 0;
  for (int i = 0; i < n; ++i)
    ans = add(ans, mul(res[i], a[i]));
  return ans;
}
```

7.6 Operations

```
int get_root(int n, int P) { // ensure 0 <= n < p
  if (P == 2 or n == 0) return n;
  auto check = [&](lld x) {
    return modpow(int(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 q = (P + 1) / 2; q; q >>= 1, e = M(e, e))
    if (q & 1) r = M(r, e);
  return
       int(r.first); // sqrt(n) mod P where P is prime
}
```

8 Geometry

8.1 Basic

```
struct pt{
    double x, y;
    pt(){}
    pt(double _x, double _y) : x(_x), y(_y){}
};
pt operator + (pt a, pt b)
{ return pt(a.x + b.x, a.y + b.y); }
pt operator - (pt a, pt b)
{ return pt(a.x - b.x, a.y - b.y); }
pt operator * (pt a, double p)
{ return pt(a.x * p, a.y * p); }
pt operator / (pt a, double p)
{ return pt(a.x / p, a.y / p); }
bool operator < (const pt &a, const pt &b)
{ return a.x < b.x || (a.x == b.x && a.y < b.y); }
bool operator == (const pt &a, const pt &b)
{ return a.x == b.x && a.y == b.y; }
double dot(pt a, pt b)
{ return a.x * b.x + a.y * b.y; }
double cross(pt a, pt b)
{ return a.x * b.y - a.y * b.x; }
double len(pt a)
{ return sqrt(dot(a, a)); }
double angle(pt a, pt b)
{ return acos(dot(a, b) / len(a) / len(b)); }
double area2(pt a, pt b, pt c)
{ return cross(b - a, c - a); }

const double eps = 1e-9;
int dcmp(double x){
  if(fabs(x) < eps) return 0;
  return x < 0? -1 : 1;
}

inline int ori(pt a, pt b, pt c){
  double area = cross(b - a, c - a);
  if(area > -eps && area < eps) return 0;
  return area > 0 ? 1 : -1;
}

inline int btw(pt a, pt b, pt c){ // [a, c, b]
  if(fabs(cross(b - a, c - a)) > eps) return false;
  if(dot(b - a, c - a)
       > -eps && len(c - a) <= len(b - a)) return true;
  return false;
}

bool intersect(pt a, pt b, pt c, pt d){
  if(a == c || a == d || b == c || b == d) return true;
  int a123 = ori(a, b, c), a124 = ori(a,
       b, d), a341 = ori(c, d, a), a342 = ori(c, d, b);
  if(a123 == 0 && a124 == 0){
    if(btw(a, b, c) || btw(a, b, d
        ) || btw(c, d, a) || btw(c, d, b)) return true;
    else return false;
  }
  else if(a123
       * a124 <= 0 && a341 * a342 <= 0) return true;
  return false;
}

istream &operator>>(istream &s, pt &a){
  s >> a.x >> a.y;
  return s;
}
```

8.2 Convex Hull

```
vector<pt> ConvexHull(vector<pt> a) {
  int n = a.size();
  sort(a.begin(), a.end());
  vector <Pt> ans = {a[0]};
  for (int t : {0, 1}) {
    int m = ans.size();
    for (int i = 1; i < n; ++i) {
      while (ans.size() > m && ori(ans[ans.size() - 2],
        ans.back(), pt[i]) <= 0) ans.pop_back();
      ans.pb(pt[i]);
    }
    reverse(all(pt));
  }
  if (ans.size() > 1) ans.pop_back();
  return ans;
}
```

8.3 Minkowski Sum

```
void reorder(vector<pt> &a){
    int pos = 0;
    for(int j = 1; j < a.size(); j++){
        if(a[j].x < a[pos].x || (a[j].x
            == a[pos].x && a[j].y < a[pos].y)) pos = j;
    }
    rotate(a.begin(), a.begin() + pos, a.end());
}

vector<pt> minkowski(vector<pt> a, vector<pt> b){
    // for(int i = 0;
         i < b.size(); i++) b[i] = {-b[i].x, -b[i].y};
        最短距離: 把 Q 鏡像, 找凸包到 (0, 0) 的最短距離
    reorder(a), reorder(b);
    a.pb(a[0]), a.pb(a[1]);
    b.pb(b[0]), b.pb(b[1]);
    vector<pt> res;
    int i = 0, j = 0;
    while(i < a.size() - 2 || j < b.size() - 2){
        res.pb(a[i] + b[j]);
        int c
            = cross(a[i + 1] - a[i], b[j + 1] - b[j]);
        if(c >= 0 && i < a.size() - 2) i++;
        if(c <= 0 && j < b.size() - 2) j++;
    }
    return res;
}
```

8.4 Intersection of Circle and Line

```
vector<Pt> CircleLineInter(Cir c, Line l) {
  Pt p = l.a + (l.b - l.a)
       * ((c.o - l.a) * (l.b - l.a)) / abs2(l.b - l.a);
  double s = (l.b - l.a) ^ (c.o
      - l.a), h2 = c.r * c.r - s * s / abs2(l.b - l.a);
  if (sign(h2) == -1) return {};
  if (sign(h2) == 0) return {p};
  Pt h = (l.b - l.a) / abs(l.b - l.a) * sqrt(h2);
  return {p - h, p + h};
}
```

8.5 Intersection of Circles

```
vector<Pt> CirclesInter(Cir c1, Cir c2) {
  double d2 = abs2(c1.o - c2.o), d = sqrt(d2);
  if (d < max(c1.r, c2.r)
       - min(c1.r, c2.r) || d > c1.r + c2.r) return {};
  Pt u = (c1.o + c2.o) / 2 + (c1.o -
      c2.o) * ((c2.r * c2.r - c1.r * c1.r) / (2 * d2));
  double A = sqrt((c1.r + c2.r + d) * (c1.r - c2.
      r + d) * (c1.r + c2.r - d) * (-c1.r + c2.r + d));
  Pt v = Pt(c1
      .o.y - c2.o.y, -c1.o.x + c2.o.x) * A / (2 * d2);
  if (sign(v.x) == 0 && sign(v.y) == 0) return {u};
  return {u + v, u - v};
}
```

8.6 Point in Convex

```
bool PointInConvex
    (const vector<Pt> &C, Pt p, bool strict = true) {
  // only works when no three points are collinear
  int a = 1, b = int(C.size()) - 1, r = !strict;
  if (C.size() == 0) return false;
  if (C.size() < 3) return r && btw(C[0], C.back(), p);
  if (ori(C[0], C[a], C[b]) > 0) swap(a, b);
  if (ori(C[0], C[a], p
      ) >= r || ori(C[0], C[b], p) <= -r) return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (ori(C[0], C[c], p) > 0 ? b : a) = c;
  }
  return ori(C[a], C[b], p) < r;
}
```

8.7 Minimum Enclosing Circle

```
pt circle(pt a, pt b, pt c){
  pt m1 = (a + b) / 2, m2 = (a + c) / 2,
       d1 = (b - a).rot().norm(), d2 = (c - a).norm();
  double tar = dot(m2, d2) - dot(m1, d2);
  double k = tar / dot(d1, d2);
  return m1 + d1 * k;
}

pair<pt, double> min_enclosing(vector<pt> &a) {
    random_shuffle(a.begin(), a.end());
  pt c = {0, 0};
  double r2 = 0;
  for(int i = 0; i < n; i++){
    if((a[i] - c).len2() <= r2) continue;
    c = a[i], r2 = 0;
    for(int j = 0; j < i; j++){
      if((a[j] - c).len2() <= r2) continue;
      c = (a[i] + a[j]) / 2, r2 = (a[i] - c).len2();
      for(int k = 0; k < j; k++){
        if((a[k] - c).len2() <= r2) continue;
        c = circle
            (a[i], a[j], a[k]), r2 = (a[k] - c).len2();
      }
    }
  }
  return make_pair(c, sqrt(r2));
}
```

8.8 Rotating Caliper

```
void RotatingCaliper(vector<pt> &pts) {
    int n = pts.size();
    for(int i = 0, j = 2; i < n; ++i) {
        int ni = (i + 1) % n;
        while (true) {
            int nj = (j + 1) % n;
            if (area(pts[j], pts[i], pts
                [ni]) < area(pts[nj], pts[i], pts[ni]))
                j = nj;
            else break;
        }
        // do something
    }
}
```

8.9 Rotating Sweep Line

```
struct Event {
  Pt d; int u, v;
  bool operator < (const Event &b) const {
    return sign(d ^ b.d) > 0; }
};
Pt ref(Pt o) {return pos(o) == 1 ? Pt(-o.x, -o.y) : o;}
void RotatingSweepLine(vector <Pt> &pt) {
  int n = pt.size();
  vector <int> ord(n), pos(n);
  vector <Event> e;
  for (int i = 0; i < n; ++i)
    for (int j = i + 1; j < n; ++j) if (i ^ j)
      e.pb({ref(pt[i] - pt[j]), i, j});
  sort(all(e));
  iota(all(ord), 0);
  sort(all(ord), [&](int i, int j) {
    return (sign(pt[i].y - pt[j].y) == 0 ?
        pt[i].x < pt[j].x : pt[i].y < pt[j].y); });
  for (int i = 0; i < n; ++i) pos[ord[i]] = i;
  const auto makeReverse = [](auto &v) {
    sort(all(v)); v.resize(unique(all(v)) - v.begin());
    vector <pii> segs;
    for (int 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 (int i = 0, j = 0; i < e.size(); i = j) {
    vector<int> tmp;
    for (; j < e.size() && !(e[i] < e[j]); j++)
      tmp.pb(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;
      // update value here
    }
  }
}
```

8.10 Delaunay Triangulation

```
/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle.
find : return a triangle contain given point
add_point : add a point into triangulation
A Triangle is in triangulation iff. its has_chd is 0.
Region of triangle u: iterate each u.edge[i].tri,
each points are u.p[(i+1)%3], u.p[(i+2)%3]
Voronoi diagram: for each triangle in triangulation,
the bisector of all its edges will split the region.
nearest point will belong to the triangle containing it
 */
const
     ll inf = MAXC * MAXC * 100; // lower_bound unknown
struct Tri;
struct Edge {
  Tri* tri; int side;
  Edge(): tri(0), side(0){}
  Edge(Tri* _tri, int _side): tri(_tri), side(_side){}
};
struct Tri {
  pll p[3];
  Edge edge[3];
  Tri* chd[3];
  Tri() {}
  Tri(const pll& p0, const pll& p1, const pll& p2) {
    p[0] = p0; p[1] = p1; p[2] = p2;
    chd[0] = chd[1] = chd[2] = 0;
  }
  bool has_chd() const { return chd[0] != 0; }
  int num_chd() const {
    return !!chd[0] + !!chd[1] + !!chd[2];
  }
  bool contains(pll const& q) const {
    for (int i = 0; i < 3; ++i)
      if (ori(p[i], p[(i + 1) % 3], q) < 0)
        return 0;
    return 1;
  }
} pool[N * 10], *tris;
void edge(Edge a, Edge b) {
  if(a.tri) a.tri->edge[a.side] = b;
  if(b.tri) b.tri->edge[b.side] = a;
}
struct Trig { // Triangulation
  Trig() {
    the_root
        = // Tri should at least contain all points
      new(tris++) Tri(pll(-inf, -inf),
          pll(inf + inf, -inf), pll(-inf, inf + inf));
  }
  Tri* find(pll p) { return find(the_root, p); }
  void add_point(const
      pll &p) { add_point(find(the_root, p), p); }
  Tri* the_root;
  static Tri* find(Tri* root, const pll &p) {
    while (1) {
      if (!root->has_chd())
        return root;
      for (int i = 0; i < 3 && root->chd[i]; ++i)
        if (root->chd[i]->contains(p)) {
          root = root->chd[i];
          break;
        }
    }
    assert(0); // "point not found"
  }
  void add_point(Tri* root, pll const& p) {
    Tri* t[3];
    /* split it into three triangles */
    for (int i = 0; i < 3; ++i)
      t[i] = new(tris
        ++) Tri(root->p[i], root->p[(i + 1) % 3], p);
    for (int i = 0; i < 3; ++i)
      edge(Edge(t[i], 0), Edge(t[(i + 1) % 3], 1));
    for (int i = 0; i < 3; ++i)
      edge(Edge(t[i], 2), root->edge[(i + 2) % 3]);
    for (int i = 0; i < 3; ++i)
      root->chd[i] = t[i];
    for (int i = 0; i < 3; ++i)
      flip(t[i], 2);
  }
  void flip(Tri* tri, int pi) {
    Tri* trj = tri->edge[pi].tri;
    int pj = tri->edge[pi].side;
    if (!trj) return;
    if (!in_cc(tri->p
        [0], tri->p[1], tri->p[2], trj->p[pj])) return;
    /* flip edge between tri,trj */
    Tri* trk = new(tris++) Tri
        (tri->p[(pi + 1) % 3], trj->p[pj], tri->p[pi]);
    Tri* trl = new(tris++) Tri
        (trj->p[(pj + 1) % 3], tri->p[pi], trj->p[pj]);
    edge(Edge(trk, 0), Edge(trl, 0));
    edge(Edge(trk, 1), tri->edge[(pi + 2) % 3]);
    edge(Edge(trk, 2), trj->edge[(pj + 1) % 3]);
    edge(Edge(trl, 1), trj->edge[(pj + 2) % 3]);
    edge(Edge(trl, 2), tri->edge[(pi + 1) % 3]);
    tri->chd
        [0] = trk; tri->chd[1] = trl; tri->chd[2] = 0;
    trj->chd
        [0] = trk; trj->chd[1] = trl; trj->chd[2] = 0;
    flip(trk, 1); flip(trk, 2);
    flip(trl, 1); flip(trl, 2);
  }
};
vector<Tri*> triang; // vector of all triangle
set<Tri*> vst;
void go(Tri* now) { // store all tri into triang
  if (vst.find(now) != vst.end())
    return;
  vst.insert(now);
  if (!now->has_chd())
    return triang.pb(now);
  for (int i = 0; i < now->num_chd(); ++i)
    go(now->chd[i]);
}
void build(int n, pll* ps) { // build triangulation
  tris = pool; triang.clear(); vst.clear();
  random_shuffle(ps, ps + n);
  Trig tri; // the triangulation structure
  for (int i = 0; i < n; ++i)
    tri.add_point(ps[i]);
  go(tri.the_root);
}
```

8.11 3D Point

```
struct Pt {
  double x, y, z;
  Pt(double _x = 0, double
        _y = 0, double _z = 0): x(_x), y(_y), z(_z){}
  Pt operator + (const Pt &o) const
  { return Pt(x + o.x, y + o.y, z + o.z); }
  Pt operator - (const Pt &o) const
  { return Pt(x - o.x, y - o.y, z - o.z); }
  Pt operator * (const double &k) const
  { return Pt(x * k, y * k, z * k); }
  Pt operator / (const double &k) const
  { return Pt(x / k, y / k, z / k); }
  double operator * (const Pt &o) const
  { return x * o.x + y * o.y + z * o.z; }
  Pt operator ^ (const Pt &o) const
  { return {Pt(y * o.z - z
        * o.y, z * o.x - x * o.z, x * o.y - y * o.x)}; }
};
double abs2(Pt o) { return o * o; }
double abs(Pt o) { return sqrt(abs2(o)); }
Pt cross3(Pt a, Pt b, Pt c)
{ return (b - a) ^ (c - a); }
double area(Pt a, Pt b, Pt c)
{ return abs(cross3(a, b, c)); }
double volume(Pt a, Pt b, Pt c, Pt d)
{ return cross3(a, b, c) * (d - a); }
bool coplaner(Pt a, Pt b, Pt c, Pt d)
{ return sign(volume(a, b, c, d)) == 0; }
Pt proj(Pt o, Pt a, Pt b, Pt c) // o proj to plane abc
{ Pt n = cross3(a, b, c);
  return o - n * ((o - a) * (n / abs2(n)));}
Pt LinePlaneInter(Pt u, Pt v, Pt a, Pt b, Pt c) {
  // intersection of line uv and plane abc
  Pt n = cross3(a, b, c);
  double s = n * (u - v);
  if (sign(s) == 0) return {-1, -1, -1}; // not found
  return v + (u - v) * ((n * (a - v)) / s);
}
```

8.12 3D Convex Hull

```
struct CH3D {
  struct face{int a, b, c; bool ok;} F[8 * N];
  double dblcmp(Pt &p,face &f)
  {return
        cross3(P[f.a], P[f.b], P[f.c]) * (p - P[f.a]);}
  int g[N][N], num, n;
  Pt P[N];
  void deal(int p,int a,int b) {
    int f = g[a][b];
    face add;
    if (F[f].ok) {
      if (dblcmp(P[p],F[f]) > eps) dfs(p,f);
      else
        add.a =
             b, add.b = a, add.c = p, add.ok = 1, g[p][
            b] = g[a][p] = g[b][a] = num, F[num++]=add;
    }
  }
  void dfs(int p, int now) {
    F[now].ok = 0;
    deal(p, F[now].b, F[now].a), deal(p, F[
        now].c, F[now].b), deal(p, F[now].a, F[now].c);
  }
  bool same(int s,int t){
    Pt &a = P[F[s].a];
    Pt &b = P[F[s].b];
    Pt &c = P[F[s].c];
    return fabs(volume(a, b, c, P[F[t].a
        ])) < eps && fabs(volume(a, b, c, P[F[t].b])) <
         eps && fabs(volume(a, b, c, P[F[t].c])) < eps;
  }
  void init(int _n){n = _n, num = 0;}
  void solve() {
    face add;
    num = 0;
    if(n < 4) return;
    if([&](){
        for (int i = 1; i < n; ++i)
        if (abs(P[0] - P[i]) > eps)
        return swap(P[1], P[i]), 0;
        return 1;
        }() || [&](){
        for (int i = 2; i < n; ++i)
        if (abs(cross3(P[i], P[0], P[1])) > eps)
        return swap(P[2], P[i]), 0;
        return 1;
        }() || [&](){
        for (int i = 3; i < n; ++i)
        if (fabs(((P[0] - P[1])
            ^ (P[1] - P[2])) * (P[0] - P[i])) > eps)
        return swap(P[3], P[i]), 0;
        return 1;
        }())return;
    for (int i = 0; i < 4; ++i) {
      add.a = (i + 1) % 4, add.b = (i
          + 2) % 4, add.c = (i + 3) % 4, add.ok = true;
      if (dblcmp(P[i],add) > 0) swap(add.b, add.c);
      g[add.a][add.
          b] = g[add.b][add.c] = g[add.c][add.a] = num;
      F[num++] = add;
    }
    for (int i = 4; i < n; ++i)
      for (int j = 0; j < num; ++j)
        if (F[j].ok && dblcmp(P[i],F[j]) > eps) {
          dfs(i, j);
          break;
        }
    for (int tmp = num, i = (num = 0); i < tmp; ++i)
      if (F[i].ok) F[num++] = F[i];
  }
  double get_area() {
    double res = 0.0;
    if (n == 3)
      return abs(cross3(P[0], P[1], P[2])) / 2.0;
    for (int i = 0; i < num; ++i)
      res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
    return res / 2.0;
  }
  double get_volume() {
    double res = 0.0;
    for (int i = 0; i < num; ++i)
      res += volume(Pt
          (0, 0, 0), P[F[i].a], P[F[i].b], P[F[i].c]);
    return fabs(res / 6.0);
  }
  int triangle() {return num;}
  int polygon() {
    int res = 0;
    for (int i = 0,
         flag = 1; i < num; ++i, res += flag, flag = 1)
      for (int j = 0; j < i && flag; ++j)
        flag &= !same(i,j);
    return res;
  }
  Pt getcent(){
    Pt ans(0, 0, 0), temp = P[F[0].a];
    double v = 0.0, t2;
    for (int i = 0; i < num; ++i)
      if (F[i].ok == true) {
        Pt p1 =
             P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c];
        t2 = volume(temp, p1, p2, p3) / 6.0;
        if (t2>0)
          ans.x += (p1.x + p2.
              x + p3.x + temp.x) * t2, ans.y += (p1.y +
               p2.y + p3.y + temp.y) * t2, ans.z += (p1
              .z + p2.z + p3.z + temp.z) * t2, v += t2;
      }
    ans.x
        /= (4 * v), ans.y /= (4 * v), ans.z /= (4 * v);
    return ans;
  }
  double pointmindis(Pt p) {
    double rt = 99999999;
    for(int i = 0; i < num; ++i)
      if(F[i].ok == true) {
        Pt p1 =
             P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c];
        double a = (p2.y - p1.y) * (p3
            .z - p1.z) - (p2.z - p1.z) * (p3.y - p1.y);
        double b = (p2.z - p1.z) * (p3
            .x - p1.x) - (p2.x - p1.x) * (p3.z - p1.z);
        double c = (p2.x - p1.x) * (p3
            .y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
        double
             d = 0 - (a * p1.x + b * p1.y + c * p1.z);
        double temp = fabs(a * p.x + b * p.y +
            c * p.z + d) / sqrt(a * a + b * b + c * c);
        rt = min(rt, temp);
      }
    return rt;
  }
};
```

9 Misc

9.1 Binary Search on Fraction

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

9.2 Random

```
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t a) const {
    static const uint64_t FIXED_RANDOM = chrono::
        steady_clock::now().time_since_epoch().count();
    return splitmix64(i + FIXED_RANDOM);
  }
};
unordered_map <int, int, custom_hash> m1;
random_device rd; mt19937 rng(rd())
```

9.3 Bit Hack

```
ll next_perm(ll v) { ll t = v | (v - 1);
  return (t + 1) |
    (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1)); }
```

9.4 Dynamic MST

```
int cnt[maxn], cost[maxn], st[maxn], ed[maxn];
pair<int, int> qr[maxn];
// qr[i].first = id of edge to
     be changed, qr[i].second = weight after operation
// cnt[i] = number of operation on edge i
// call solve(0, q - 1, v,
     0), where v contains edges i such that cnt[i] == 0

void contract(int l, int
    r, vector<int> v, vector<int> &x, vector<int> &y) {
  sort(v.begin(), v.end(), [&](int i, int j) {
      if (cost[i] == cost[j]) return i < j;
      return cost[i] < cost[j];
      });
  djs.save();
  for (int i = l; i <= r;
      ++i) djs.merge(st[qr[i].first], ed[qr[i].first]);
  for (int i = 0; i < (int)v.size(); ++i) {
    if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
      x.push_back(v[i]);
      djs.merge(st[v[i]], ed[v[i]]);
    }
  }
  djs.undo();
  djs.save();
  for (int i = 0; i < (
      int)x.size(); ++i) djs.merge(st[x[i]], ed[x[i]]);
  for (int i = 0; i < (int)v.size(); ++i) {
    if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
      y.push_back(v[i]);
      djs.merge(st[v[i]], ed[v[i]]);
    }
  }
  djs.undo();
}

void solve(int l, int r, vector<int> v, long long c) {
  if (l == r) {
    cost[qr[l].first] = qr[l].second;
    if (st[qr[l].first] == ed[qr[l].first]) {
      printf("%lld\n", c);
      return;
    }
    int minv = qr[l].second;
    for (int i = 0; i < (int
        )v.size(); ++i) minv = min(minv, cost[v[i]]);
    printf("%lld\n", c + minv);
    return;
  }
  int m = (l + r) >> 1;
  vector<int> lv = v, rv = v;
  vector<int> x, y;
  for (int i = m + 1; i <= r; ++i) {
    cnt[qr[i].first]--;
    if (cnt
        [qr[i].first] == 0) lv.push_back(qr[i].first);
  }
  contract(l, m, lv, x, y);
  long long lc = c, rc = c;
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) {
    lc += cost[x[i]];
    djs.merge(st[x[i]], ed[x[i]]);
  }
  solve(l, m, y, lc);
  djs.undo();
  x.clear(), y.clear();
  for (int i = m + 1; i <= r; ++i) cnt[qr[i].first]++;
  for (int i = l; i <= m; ++i) {
    cnt[qr[i].first]--;
    if (cnt
        [qr[i].first] == 0) rv.push_back(qr[i].first);
  }
  contract(m + 1, r, rv, x, y);
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) {
    rc += cost[x[i]];
    djs.merge(st[x[i]], ed[x[i]]);
  }
  solve(m + 1, r, y, rc);
  djs.undo();
  for (int i = l; i <= m; ++i) cnt[qr[i].first]++;
}
```

9.5 Manhattan MST

```
void solve(Point *a, int n) {
    sort(a, a + n, [](const Point &p, const Point &q) {
      return p.x + p.y < q.x + q.y;
    });
    set<Point> st; // greater<Point::x>
    for (int i = 0; i < n; ++i) {
      for (auto it = st.lower_bound(
          a[i]); it != st.end(); it = st.erase(it)) {
        if (it ->
            x - it -> y < a[i].x - a[i].y) break;
        es.push_back
          ({it -> u, a[i].u, dist(*it, a[i])});
      }
      st.insert(a[i]);
    }
}
void MST(Point *a, int n) {
    for (int t = 0; t < 2; ++t) {
        solve(a, n);
        for (int
            i = 0; i < n; ++i) swap(a[i].x, a[i].y);
        solve(a, n);
        for (int i = 0; i < n; ++i) a[i].x = -a[i].x;
    }
}
```

9.6 DP Optimization Conditions

9.6.1 Totally Monotone (Concave/Convex)

$\forall i<i',j<j', B[i][j] \le B[i'][j] \implies B[i][j'] \le B[i'][j']$
$\forall i<i',j<j', B[i][j] \ge B[i'][j] \implies B[i][j'] \ge B[i'][j']$

9.6.2 Monge Condition (Concave/Convex)

$\forall i<i',j<j', B[i][j]+B[i'][j'] \ge B[i][j']+B[i'][j]$
$\forall i<i',j<j', B[i][j]+B[i'][j'] \le B[i][j']+B[i'][j]$

9.6.3 Optimal Split Point

If

$$B[i][j]+B[i+1][j+1] \ge B[i][j+1]+B[i+1][j]$$

then

$$H_{i,j-1} \le H_{i,j} \le H_{i+1,j}$$

9.7 Mo's Algo With Modification

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

9.8 Mo's Algo On Tree

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

9.9 Mo's Algorithm

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

9.10 Hilbert Curve

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

9.11 SMAWK

```
bool select(int r, int u, int v){
    // if f(r, v) is better than f(r, u), return true
    return f(r, u) < f(r, v);
}
// For all 2x2 submatrix: (x < y => y is better than x)
// If M[1][0] < M[1][1], M[0][0] < M[0][1]
// If M[1][0] == M[1][1], M[0][0] <= M[0][1]
// M[i][ans[i]] is the best value in the i-th row
vector<int> solve(vector<int> &r, vector<int> &c){
  if(r.size() == 1){
    vector<int> opt(1, 0);
    for(int i = 1; i < c.size(); i++){
      if(select(r[0], c[opt[0]], c[i])){
        opt[0] = i;
      }
    }
    return opt;
  }
  //reduce
  vector<int> st, rev;
  for(int i = 0; i < c.size(); i++){
    while(!st.empty()
        && select(r[st.size() - 1], st.back(), c[i])){
      st.pop_back();
      rev.pop_back();
    }
    if(st.size() < r.size()){
      st.pb(c[i]);
      rev.pb(i);
    }
  }
  //interpolate
  vector<int> half;
  for(int i = 0; i < r.size(); i += 2){
    half.pb(r[i]);
  }
  vector<int> ans(r.size());
  auto interp = solve(half, st);
  for(int i = 0;
      i < interp.size(); i++) ans[i * 2] = interp[i];
  for(int i = 1; i < ans.size(); i += 2){
    int s = ans[i - 1], e = (i
        + 1 < ans.size() ? ans[i + 1] : st.size() - 1);
    ans[i] = s;
    for(int j = s + 1; j <= e; j++){
      if(select(r[i], st[ans[i]], st[j])) ans[i] = j;
    }
  }
  for(int
        i = 0; i < ans.size(); i++) ans[i] = rev[ans[i]];
  return ans;
}

vector<int> smawk(int n, int m){
  vector<int> r(n), c(m);
  iota(r.begin(), r.end(), 0);
  iota(c.begin(), c.end(), 0);
  return solve(r, c);
}
```

9.12 Simulate Annealing

```
double anneal() {
  mt19937 rnd_engine(time(0));
  uniform_real_distribution<double> rng(0, 1);
  const double dT = 0.001;
  // Argument p
  double S_cur = calc(p), S_best = S_cur;
  for (double T = 2000; T > eps; T -= dT) {
    // Modify p to p_prime
    const double S_prime = calc(p_prime);
    const double delta_c = S_prime - S_cur;
    double prob = min((double)1, exp(-delta_c / T));
    if (rng(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;
}
```

9.13 Python

```
from [decimal, fractions, math, random] import *
arr = list(map(int, input().split())) # input
setcontext(Context
    (prec=10, Emax=MAX_EMAX, rounding=ROUND_FLOOR))
Decimal('1.1') / Decimal('0.2')
Fraction(3, 7)
Fraction(Decimal('1.14'))
Fraction('1.2').limit_denominator(4).numerator
Fraction(cos(pi / 3)).limit_denominator()
S = set(), S.add((a, b)), S.remove((a, b)) # set
if not (a, b) in S:
D = dict(), D[(a, b)] = 1, del D[(a, b)] # dict
for (a, b) in D.items():
arr = [randint(1, C) for i in range(N)]
choice([8, 6, 4, 1]) # random pick one
```

9.14 Matroid

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

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

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

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

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