Central South University

Trio of Tomorrow Winds

Boyang Zhao, Guochen Xu, Xuanjing Li

2022 年 12 月 2 日

General (1)

1.1 赛前必读

1. 提交前，再次检查数据范围。不但要注意最大范围，更要注意小数据、极端数据（例如 $n = 1$）
2. 提交前，再次检查输入输出格式，是否有 spj，是否需要取模，模数是多少。

1.2 Tricks

1.2.1 调试

在第一行加一下 `#define _GLIBCXX_DEBUG` 让 stl 检查越界之类的问题。

```
void dbg() { std::cerr << "\n"; }
template <class T, class... Ts>
void dbg(T &&a, Ts &&...b) {
  std::cerr << a << " ", dbg(b...);
}
```

1.2.2 结论

1. 不存在长度为偶数的简单回路，则该图是仙人掌图（一条边最多属于一个环）
2. 随机点集的凸包点数是 $O(\log n)$。
3. **判断边集是否是割集**：随便选一个生成树，然后给每条非树边在 2^{64} 范围内随机一个权值，然后每条树边的权值等于跨越他的非树边权值的 xor，对于每次询问，不连通 iff 删掉的边权的集合线性相关。

1.2.3 字符串

1. 自动机类的，有时候会和虚树结合，需要注意

1.2.4 根号算法

根号分治，序列分块，值域分块，莫队，线段树上只 pushup 小点，块状链表，对操作序列分块，KD 树

1.2.5 历史最大值

信息为列向量 $(max, hmax)^T$，标记维护矩阵 $\begin{bmatrix} a & -\infty \\ b & c \end{bmatrix}$。

```
struct Tag {
  LL a = 0, b = NINF, c = 0;
  void apply(const Tag &r) {
    b = std::max(b + r.c, a + r.b);
    a += r.a, c += r.c;
  }
};
```

1.2.6 线段树经典 $O(\log)$ update

楼房重建 设 $\text{Query}(o, x)$ 表示节点 o 之前的最大值为 x 时的答案，维护 $val_o = \text{Query}(rs, \max_{ls})$。然后 Query 的复杂度是 $O(\log n)$。

1.2.7 一些离线算法

一个比较重要的思想是把时间轴看作一维，例如线段树分治，对时间分块（操作序列分块）等。

整体二分 例如 K 大数查询：一些可重集，区间加一个元素 c，求区间并的第 k 大。

乍一看需要在整体二分中用二维数据结构来维护，但是你每次递归到两边前，先把 $k -= c_i$ 就可以了。

也就是说，你求解 $[l, r]$ 之前，需要考虑 $(-\infty, l)$ 的贡献，但是并不一定要事先在数据结构上加减，可以直接把这个贡献存下来。

1.2.8 Slope Trick

分段函数，每段都是斜率是整数的一次函数，并且是凸的，可以维护最右边直线 L，再用数据结构维护分段点的可重集 S（每过一个分段点，斜率变化 1）。两个函数相加，可以表示为 $(L_1 + L_2, S_1 \cup S_2)$。

凸函数的 $(min, +)$ 卷积，就是闵可夫斯基和。

1.3 .bashrc

```
alias c='g++ -Wall -Wextra -O2 -std=c++17'
```

1.4 .emacs

```
(electric-pair-mode t)
(global-linum-mode t)
(setq c-default-style "linux")
(setq c-basic-offset 2)
(defalias 'yes-or-no-p 'y-or-n-p)
(defconst cs
  '("cc-mode" (c-offsets-alist . ((innamespace . 0)
                                   (inlambda . 0)))))
(c-add-style "cs" cs)
(setq c-default-style "cs")
```

1.5 卡常

```
#pragma GCC
↪  optimize("-Ofast","-funroll-all-loops","-ffast-math")
#pragma GCC optimize("-fno-math-errno")
#pragma GCC optimize("-funsafe-math-optimizations")
#pragma GCC optimize("-freciprocal-math")
#pragma GCC optimize("-fno-trapping-math")
#pragma GCC optimize("-ffinite-math-only")
#pragma GCC optimize("-fno-stack-protector")
#pragma GCC target ("avx2","sse4.2","fma")

// Fast IO
inline char gc() {
  const int L = 1 << 16;
  static char s[L], *p, *q;
  static auto i = std::cin.rdbuf();
  return p == q ? p = s, q = p + i->sgetn(p, L), p == q ? EOF
  ↪  : *p++ : *p++;
}
int read() {
  int a, c;
  while ((a = gc()) < 40);
  if (a == '-') return -read();
  while ((c = gc()) >= 48) a = a * 10 + c - 480;
  return a - 48;
}

using A = __attribute((vector_size(16))) int;
// 16 是字节数，这个代表 4 个 int，里面这个数要是 2 的幂

struct Barrett {
  uint64_t m, im;
  Barrett() = default;
  Barrett(int n) : m(n), im(-1ULL / m) {}
  int rem(uint64_t n) {
    uint64_t x = __uint128_t(n) * im >> 64, r = n - x * m;
    return m <= r ? r - m : r;
  }
};
```

1.6 Random

```
std::mt19937_64 rng((std::random_device())());
LL rnd(LL l, LL r) {
  if (l > r) std::swap(l, r);
  return std::uniform_int_distribution<LL>(l, r)(rng);
}
```

1.7 Hash

模 $2^{64} - 1$ 哈希，比单哈（自然溢出或者 int 取模）慢（1.5-2 倍），比双哈快。

```
struct H {
```

```
  typedef uint64_t ULL;
  ULL x; H(ULL x = 0) : x(x) {}
#define OP(O, A, B) H operator O(H o) { \
  ULL r = x; asm(A "addq %%rdx, %0\n adcq $0,%0"\
  : "+a"(r) : B); return r;}
  OP(+, , "d"(o.x)) OP(*, "mul %1\n", "r"(o.x) : "rdx")
  H operator-(H o) { return *this + ~o.x; }
  ULL get() const { return x + !~x; }
  bool operator==(H o) const { return get() == o.get(); }
  bool operator<(H o) const { return get() < o.get(); }
};
```

如果不给用 asm，可以写模 $2^{61}-1$ 哈希，好写，快于双哈

```
u64 modfma(const u64 &a, const u64 &b, const u64 &c) {
  u128 ret = u128(a) * b + c;
  ret = (ret & md) + (ret >> 61);
  return ret >= md ? ret - md : ret;
}
```

```
uint64_t xorshift(uint64_t x) {
  x ^= x << 13;
  x ^= x >> 7;
  x ^= x << 17;
  return x;
}
uint64_t splitmix64(uint64_t x) {
  x += 0x9e3779b97f4a7c15;
  x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
  x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
  return x ^ (x >> 31);
}
```

1.8 PBDS

1.8.1 Priority Queque

```
/*
template<
    typename Value_Type,
    typename Cmp_Fn = std::less<Value_Type>,
    typename Tag = pairing_heap_tag,
    typename Allocator = std::allocator<char> >
class priority_queue;
*/

#include <ext/pb_ds/priority_queue.hpp>
using __gnu_pbds::priority_queue;

// q.erase(it);
// q.modify(it, val);
// q.join(ohter); other will be cleared
```

1.8.2 Tree

find_by_order(k)：第 k 小，从 0 开始。

order_of_key(x)：查 $(-\infty, x)$ 的个数。

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template <class T, class U = null_type, class Comp =
↪  std::less<T>>
using Tree =
  tree<T, U, Comp, rb_tree_tag,
  ↪  tree_order_statistics_node_update>;
```

1.8.3 HashTable

```
#include <ext/pb_ds/assoc_container.hpp>
const int RANDOM =
std::chrono::high_resolution_clock::
now().time_since_epoch().count();
struct CHash {
  const uint64_t C = LL(4e18 * acos(0)) | 71;
  LL operator()(LL x) const { return __builtin_bswap64((x ^
  ↪  RANDOM) * C); }
};
using HashTable = __gnu_pbds::gp_hash_table<LL, int, CHash>;
```

1.9 模拟退火

```
LL ans = 1e18;
while ((double)clock() / CLOCKS_PER_SEC < 0.98) {
  for (int i = 0; i < m; i++) {
    c[i] = rng() & 1;
  }
  LL now = cal();
  ans = std::min(ans, now);
  double t = 1e9, coef = 0.99;
  while (t > 0.1) {
    int i = rng() % m;
    c[i] ^= 1;
    LL x = cal();
    if (exp((now - x) / t) > randDouble()) {
      now = x;
      ans = std::min(ans, x);
    } else {
      c[i] ^= 1;
    }
    t *= coef;
  }
}
```

1.10 开栈

```
int _size = 256 << 20;  // 256MB
char *p = (char *)malloc(_size) + _size;
__asm__("movl %0,%%esp\n" : : "r"(p));

// linux
#include <sys/resource.h>
void increase_stack_size() {
  const rlim_t ks = 64 * 1024 * 1024;
  struct rlimit rl;
  int res = getrlimit(RLIMIT_STACK, &rl);
  if (res == 0) {
    if (rl.rlim_cur < ks) {
      rl.rlim_cur = ks;
      res = setrlimit(RLIMIT_STACK, &rl);
    }
  }
}
```

Data Structures (2)

2.1 Line Container

```
struct Line {
  mutable LL k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(LL x) const { return p < x; }
};

struct LineContainer : std::multiset<Line, std::less<>> {
  // (for doubles, use INF = 1/.0, div(a,b) = a/b)
  static const LL INF = LLONG_MAX;
  LL div(LL a, LL b) {  // floored division
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = INF, 0;
    if (x->k == y->k)
      x->p = x->m > y->m ? INF : -INF;
    else
      x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
```

```
  void add(LL k, LL m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
  }
  LL query(LL x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

2.2 Lichao

```
struct Line {
  int k;
  LL b;
  LL f(int x) {
    return 1LL * k * x + b;
  }
} t[N << 2];
#define ls o << 1
#define rs o << 1 | 1
void ins(int o, int l, int r, Line x) {
  int m = l + r >> 1;
  bool lv = x.f(l) > t[o].f(l), mv = x.f(m) > t[o].f(m), rv = x.f(r) > t[o].f(r);
  if (mv) std::swap(x, t[o]);
  if (lv == rv || l == r) return;
  lv != mv ? ins(ls, l, m, x) : ins(rs, m + 1, r, x);
}
LL ask(int o, int l, int r, int x) {
  int m = l + r >> 1;
  if (x == m) return t[o].f(x);
  return std::max(t[o].f(x), x < m ? ask(ls, l, m, x) : ask(rs, m + 1, r, x));
}
```

2.3 Scapegoat Tree

```
#define ls ch[0]
#define rs ch[1]
struct Node {
  int ch[2], val, siz;
} t[N];
void pushup(int o) {
  t[o].siz = t[t[o].ls].siz + t[t[o].rs].siz + 1;
}
bool bad(int o) {
  return t[o].siz * 0.73 < std::max(t[t[o].ls].siz, t[t[o].rs].siz);
}
int a[N], top, cnt;
void dfs(int x) {
  if (!x) return;
  dfs(t[x].ls), a[++top] = x, dfs(t[x].rs);
}
int build(int l, int r) {
  if (l > r) return 0;
  int m = l + r >> 1, o = a[m];
  t[o].ls = build(l, m - 1), t[o].rs = build(m + 1, r);
  pushup(o);
  return o;
}
void insert(int &root, int x) {
  int p = ++cnt, o;
  t[p] = {{0, 0}, x, 1};
  if (!root) {
    root = p; return;
  }
  o = root, top = 0;
  for (;;) {
    a[++top] = o, t[o].siz++;
    int &v = t[o].ch[x > t[o].val];
    if (!v) {
      v = p; break;
    }
    o = v;
  }
  for (int i = 1; i <= top; i++) {
    if (bad(a[i])) {
      int &o = i > 1 ? t[a[i - 1]].ch[a[i] == t[a[i - 1]].rs] : root;
      top = 0, dfs(o), o = build(1, top);
      break;
    }
  }
}
```

2.4 LCT

```
struct LCT {
  int ch[N][2], fa[N];
  bool rev[N];
  bool nrt(int o) { return fa[o] && (ch[fa[o]][0] == o || ch[fa[o]][1] == o); }
  int dir(int o) { return ch[fa[o]][1] == o; }
  void sch(int o, int d, int x) {
    ch[o][d] = x;
    if (x) fa[x] = o;
  }
  void rotate(int o) {
    int k = dir(o), p = fa[o];
    fa[o] = fa[p];
    if (nrt(p)) ch[fa[o]][dir(p)] = o;
    sch(p, k, ch[o][!k]), sch(o, !k, p);
    pushup(p);
    pushup(o);
  }
  void push(int o) {
    if (nrt(o)) push(fa[o]);
    pushdown(o);
  }
  void splay(int o) {
    for (push(o); nrt(o); rotate(o)) {
      if (nrt(fa[o])) {
        rotate(dir(fa[o]) == dir(o) ? fa[o] : o);
      }
    }
    pushup(o);
  }
  void access(int o) {
    for (int x = 0, i = o; i; i = fa[x = i]) {
      splay(i), ch[i][1] = x, pushup(i);
    }
    splay(o);
  }
  void evert(int o) { access(o), reverse(o), pushdown(o); }
  void expose(int x, int y) { evert(x), access(y); }
  bool link(int x, int y) {
    if (x == y) return false;
    expose(x, y);
    if (fa[x]) return false;
    fa[x] = y;
    return true;
  }
  bool cut(int x, int y) {
    expose(x, y);
    if (fa[x] == y && !ch[x][0] && !ch[x][1] && !ch[y][1]) {
      fa[x] = ch[y][0] = 0;
      pushup(y);
      return true;
    }
    return false;
  }
  void pushup(int o) {}
```

```cpp
  void reverse(int o) {
    rev[o] ^= 1;
    std::swap(ch[o][0], ch[o][1]);
  }
  void pushdown(int o) {
    if (rev[o]) {
      if (ch[o][0]) reverse(ch[o][0]);
      if (ch[o][1]) reverse(ch[o][1]);
      rev[o] = false;
    }
  }
} lct;
```

2.5 Top Tree

对于基于点的 Toptree，和 LCT 差不多用就行了。

对于带边信息的 Toptree，按照下面的 `link` 函数来连边。`link` 时建立的储存边信息的 base cluster **永远不会**被 `pushup` 到。而修改边 (u, v) 的信息时，执行 `u->evert()`, `v->access()` 此时 `u == v->ls`，`e == u->rs`。

对于左右儿子是空的情况，需要格外小心！

根据实战经验，开内存池对于 Toptree 几乎没有优化效果，所以放心大胆直接 `new`。

```cpp
#define ls ch[0]
#define rs ch[1]
#define ms ch[2]
#define R(x) static_cast<RakeTree *>(x)
#define C(x) static_cast<CompressTree *>(x)
#define G(x, y, z) (x ? x->y : z)

struct SplayTree {
  using Node = SplayTree;
  Node *ch[3] = {}, *fa = 0;

  bool nrt() const { return fa && (fa->ls == this || fa->rs ==
↪  this); }
  int dir() const { return fa->rs == this; }
  void sch(int d, Node *x) {
    ch[d] = x;
    if (x) x->fa = this;
  }
  void rotate() {
    Node *p = fa;
    int k = dir();
    fa = p->fa;
    if (fa) fa->ch[fa->ms == p ? 2 : p->dir()] = this;
    p->sch(k, ch[!k]), sch(!k, p);
    p->pushup();
    pushup();
  }

  void pull() {
    if (nrt()) fa->pull();
    pushdown();
  }

  void splay() {
    for (pull(); nrt(); rotate()) {
      if (fa->nrt()) {
        (dir() == fa->dir() ? fa : this)->rotate();
      }
    }
    pushup();
  }

  virtual ~SplayTree() {}

  virtual void pushup() {}
  virtual void pushdown() {}
};


struct RakeTree : SplayTree {
  void pushup();
  void pushdown();
};
struct CompressTree : SplayTree {
  void access() {
    if (splay(), rs) {
      auto r = new RakeTree;
      r->sch(0, ms), r->sch(2, rs), r->pushup();
      rs = 0, sch(2, r), pushup();
    }
    for (; fa; rotate()) {
      fa->splay();
      auto m = fa, p = m->fa;
      p->splay(), m->pushdown();
      if (p->rs) {
        m->sch(2, p->rs), m->pushup();
      } else {
        if (!m->ls) {
          p->sch(2, m->rs);
        } else if (!m->rs) {
          p->sch(2, m->ls);
        } else {
          auto x = m->ls;
          x->fa = 0;
          while (x->pushdown(), x->rs) x = x->rs;
          x->splay(), p->sch(2, x);
          x->sch(1, m->rs), x->pushup();
        }
        // delete m;
      }
      p->sch(1, this), pushdown();
    }
    pushup();
  }
  void evert() { access(), reverse(), pushdown(); }

  bool rev = false;

  void reverse() {
    rev ^= 1, std::swap(ls, rs);
    // [WARNING] remember to reverse the information
  }

  void pushdown() {
    if (rev) {
      if (ls) C(ls)->reverse();
      if (rs) C(rs)->reverse();
      rev = false;
    }
  }

  void pushup() {}
};

CompressTree *link(CompressTree *x, CompressTree *y, int len)
↪  {
  x->access(), y->evert();
  auto e = new CompressTree; // e is the edge
  // record edge information here
  x->sch(1, y), y->sch(0, e);
  y->pushup(), x->pushup();
  return x;
}
```

2.6 Tree Block

```cpp
int siz[N], bot[N], bel[N], tot;
bool key[N];

int B; // sqrt(n) * 1.5

struct Block {
  int top, bot;
} b[N];
```

```cpp
void build(int x) { // 0->1, build(0)
  static int q[N], r;
  auto add = [&](int d) {
    int o = tot++;
    b[o] = {x, d};
    for (int i = 1; i <= r; i++) {
      int u = q[i];
      bel[u] = o;
      if (key[u]) continue;
      for (int v : g[u]) q[++r] = v;
    }
  };
  int s = 0, c = 0;
  for (int y : g[x]) {
    build(y);
    s += siz[y];
    if (bot[y]) {
      c++;
      bot[x] = bot[y];
    }
  }
  if (1 + s >= B || c > 1 || !x) {
    std::sort(g[x].begin(), g[x].end(),
              [](int i, int j) { return siz[i] < siz[j]; });
    siz[x] = 1;
    bot[x] = x;
    key[x] = true;
    s = c = r = 0;
    for (int y : g[x]) {
      if (c && bot[y] || siz[y] + s >= B) {
        add(c);
        s = c = r = 0;
      }
      c |= bot[y];
      s += siz[y];
      q[++r] = y;
    }
    add(c);
  } else {
    siz[x] = 1 + s;
  }
}
```

2.7 Cartesian Tree

笛卡尔树，最小值为根，p 是 i 的父亲。

```cpp
std::vector<int> p(n), s(n);
for (int i = 0, t = 0; i < n; i++) {
  int x = -1;
  while (t && a[s[t - 1]] > a[i]) x = s[--t];
  p[i] = t ? s[t - 1] : i;
  s[t++] = i;
  if (x != -1) p[x] = i;
}
```

2.8 WBLT

```cpp
void rotate(int o, int k) {
  int x = t[o].ch[k];
  pushdown(x);
  t[o].ch[k] = t[x].ch[k];
  t[x].ch[k] = t[x].ch[!k];
  t[x].ch[!k] = t[o].ch[!k];
  t[o].ch[!k] = x;
  pushup(x);
}
void maintain(int o) {
  if (t[o].siz <= A + 1) return;
  pushdown(o);
  if (t[t[o].ls].siz * A < t[o].siz) {
    rotate(o, 1);
    // maintain(t[o].ls);
  }
  if (t[t[o].rs].siz * A < t[o].siz) {
    rotate(o, 0);
    // maintain(t[o].rs);
  }
}
```

用于维护区间的乱搞写法，把上面 maintain 的注释去掉。

```cpp
void split(int o, int k) {
  if (t[t[o].ls].siz == k) return;
  pushdown(o);
  if (t[t[o].ls].siz > k) {
    split(t[o].ls, k);
    rotate(o, 0);
  } else {
    split(t[o].rs, k - t[t[o].ls].siz);
    rotate(o, 1);
  }
}
void split(int o, int k, int &x, int &y) {
  if (k <= 0) {
    x = 0, y = o;
  } else if (k >= t[o].siz) {
    x = o, y = 0;
  } else {
    split(o, k);
    x = t[o].ls, y = t[o].rs;
  }
}
int merge(int o, int x, int y) {
  if (x && y) {
    t[o].ls = x, t[o].rs = y;
    pushup(o);
    maintain(o);
    return o;
  }
  return x ? (maintain(x), x) : (maintain(y), y);
}
/*
split(r1 = root, r, b, c);
split(r2 = b, l - 1, a, b);
b = merge(r2, a, b);
root = merge(r1, b, c);
*/
```

Strings (3)

3.1 SAM

后缀树 反串建 SAM 后的 parent tree 相当于原串后缀树，后缀树可以接受原串的所有子串。对于字典序 K 小子串问题，可以用后缀树 + 二分来做。

DAG SAM 的自动机部分是 DAG，也可以接受原串的所有子串。上面的问题使用该 DAG 求的话需要进行 DAG 链剖分才能 $O(\log)$ 询问。另外，自动机 +parent 树边合起来起来也是 DAG，有些题可以在上面操作。

染色 暴力向上跳类似 access 区间染色，树剖之后可以转换成珂朵莉树。广义 SAM 暴力跳（不经过重复点）的复杂度是 $O(n\sqrt{n})$，这个有时候很好用，但是还是要多想一下区间染色等等，每个串经过的点相当于虚树，也可以考虑一下。

自底向上合并 可以在 parent 树上自底向上合并（启发式或者线段树），还要注意到 len 是自底向上递减的，可能存在一些性质。后缀数组 h 从大到小合并也类似，这个可以用并查集维护。

```cpp
constexpr int N(2e6 + 5);
std::array<int, 26> ch[N];
int len[N], fa[N], cnt;
// fa[0] = -1, cnt = 0, ch[0].fill(0)
```

```cpp
int ins(int last, int x) {
  if (ch[last][x] && len[last] + 1 == len[ch[last][x]]) return
  ↪ ch[last][x];
  int p = last, np = ++cnt, q;
  len[np] = len[p] + 1;
  ch[np].fill(0);
  for (; ~p && !ch[p][x]; p = fa[p]) ch[p][x] = np;
  if (p == -1) {
    fa[np] = 0;
  } else if (len[q = ch[p][x]] == len[p] + 1) {
    fa[np] = q;
  } else {
    int nq = p == last ? np : ++cnt;
    len[nq] = len[p] + 1;
    ch[nq].fill(0);
    ch[nq] = ch[q];
    for (; ~p && ch[p][x] == q; p = fa[p]) ch[p][x] = nq;
    fa[np] = nq;
    fa[nq] = fa[q];
    fa[q] = nq;
  }
  return np;
}
```

3.1.1 DAG 链剖分

预处理 **注意 pos：需要在 ins 的时候把 clone 节点的 pos 更新！** dfs 序从 0 开始（0 的 dfs 序是 0），同一条重链上的 pos 是连续的，ht 是到重链底端的距离，从零开始。

```cpp
int in[N], id[N], dfn, pos[N], ht[N];
void dag_hld() {
  std::vector<LL> f(cnt + 1, 1), g(cnt + 1, 1);
  std::vector<int> p(cnt + 1);
  std::iota(p.begin(), p.end(), 0);
  std::sort(p.begin(), p.end(), [&](int i, int j) { return
  ↪ len[i] < len[j]; });
  for (int i : p) {
    for (int j = 0; j < 26; j++) {
      if (ch[i][j]) f[ch[i][j]] += f[i];
    }
  }
  for (int i = cnt; i >= 0; i--) {
    int x = p[i];
    for (int j = 0; j < 26; j++) {
      if (ch[x][j]) g[x] += g[ch[x][j]];
    }
  }
  std::function<void(int)> dfs = [&](int x) {
    in[x] = dfn++;
    id[in[x]] = x;
    for (int i = 0; i < 26; i++) {
      int y = ch[x][i];
      if (y && 2 * f[x] > f[y] && 2 * g[y] > g[x]) {
        dfs(y);
        pos[x] = pos[y] - 1;
        ht[x] = ht[y] + 1;
      }
    }
  };
  for (int i : p) if (!i || !in[i]) dfs(i);
}
```

定位 定位 $S[l,r)$。

```cpp
int o = 0, match = 0;
while (l < r) {
  o = ch[o][s[l] - 'a'];
  if (!o) break;
  int k = std::min({ht[o] + 1, r - l, sa.lcp(l, pos[o])});
  // interval [in[o], in[o] + k)
  l += k;
  match += k;
  o = id[in[o] + k - 1];
}
```

3.2 Suffix Array

3.2.1 倍增

h 表示排名为 i 和排名 $i-1$ 的后缀的 lcp 长度。

```cpp
struct SuffixArray {
  std::vector<int> p, r, h;
  template <class T>
  SuffixArray(const T &s) : p(s.size()), r(s.size()),
  ↪ h(s.size() - 1) {
    int n = s.size(), m = 1;
    std::iota(p.begin(), p.end(), 0);
    std::sort(p.begin(), p.end(), [&](int i, int j) { return
    ↪ s[i] < s[j]; });
    r[p[0]] = 0;
    for (int i = 1; i < n; i++) {
      r[p[i]] = s[p[i]] != s[p[i - 1]] ? m++ : m - 1;
    }
    std::vector<int> x(n), c(n);
    for (int k = 1; m < n; k <<= 1) {
      int t = 0;
      for (int i = 0; i < k; i++) x[t++] = n - k + i;
      for (int i : p) if (i >= k) x[t++] = i - k;
      for (int i = 0; i < m; i++) c[i] = 0;
      for (int i = 0; i < n; i++) c[r[i]]++;
      for (int i = 1; i < m; i++) c[i] += c[i - 1];
      for (int i = n - 1; i >= 0; i--) p[--c[r[x[i]]]] = x[i];
      r.swap(x), r[p[0]] = 0, m = 1;
      for (int i = 1; i < n; i++) {
        r[p[i]] = x[p[i - 1]] < x[p[i]] || p[i - 1] + k == n
        ↪ || x[p[i - 1] + k] < x[p[i] + k] ? m++ : m - 1;
      }
    }
    for (int i = 0, k = 0; i < n; i++) {
      if (k) k--;
      if (!r[i]) continue;
      for (int j = p[r[i] - 1]; i + k < n && j + k < n && s[i
      ↪ + k] == s[j + k]; k++) ;
      h[r[i] - 1] = k;
    }
  }
};
```

3.2.2 SAIS

```cpp
std::vector<int> suffixSort(const std::vector<int> &s, int k)
↪ {
#define IM(x) (t[x] && x && !t[x - 1])
  int n = s.size();

  std::vector<bool> t(n);
  std::vector<int> b(k), l(k), r(k), p(n, -1), p1;

  t.back() = true;
  for (int i = n - 2; i >= 0; i--) {
    t[i] = s[i] < s[i + 1] || t[i + 1] && s[i] == s[i + 1];
    if (IM(i + 1)) p1.push_back(i + 1);
  }
  std::reverse(p1.begin(), p1.end());

  int n1 = p1.size();
  for (int i = 0; i < n; i++) b[s[i]]++;
  for (int i = 0, sum = 0; i < k; i++)
    sum += b[i], r[i] = sum, l[i] = sum - b[i];

  std::vector<int> s1(n1), a(n1);

  b = r;
  for (int i = n1 - 1; i >= 0; i--) p[--b[s[p1[i]]]] = p1[i];
  b = l;
  for (int i = 0; i < n; i++)
    if (int j = p[i] - 1; j >= 0 && !t[j]) p[b[s[j]]++] = j;
  b = r;
```

```cpp
    for (int i = n - 1; i >= 0; i--)
      if (int j = p[i] - 1; j >= 0 && t[j]) p[--b[s[j]]] = j;
    for (int i = 0, j = 0; i < n; i++)
      if (IM(p[i])) a[j++] = p[i];

    int ch = 0;
    for (int i = 0, pr = -1; i < n1; i++) {
      int pos = a[i], j = 0;
      for (;;) {
        if (pr == -1 || s[pos + j] != s[pr + j] || t[pos + j] != t[pr + j]) {
          pr = pos, ch++;
          break;
        } else if (j && (IM(pos + j) || IM(pr + j)))
          break;
        j++;
      }
      p[pos] = ch - 1;
    }

    for (int i = 0; i < n1; i++) s1[i] = p[p1[i]];
    if (ch != n1) {
      a = suffixSort(s1, ch);
    } else {
      for (int i = 0; i < n1; i++) a[s1[i]] = i;
    }

    b = r;
    std::fill_n(p.begin(), n, -1);
    for (int i = n1 - 1; i >= 0; i--) p[--b[s[p1[a[i]]]]] = p1[a[i]];
    for (int i = 0; i < n; i++)
      if (int j = p[i] - 1; j >= 0 && !t[j]) p[l[s[j]]++] = j;
    for (int i = n - 1; i >= 0; i--)
      if (int j = p[i] - 1; j >= 0 && t[j]) p[--r[s[j]]] = j;
    return p;
  #undef IM
  }

  struct SuffixArray {
    std::vector<int> p, r, h;
    SuffixArray(const std::string &s)
      : p(suffixSort({s.data(), s.data() + s.size() + 1}, 128)),
        r(s.size()), h(s.size() - 1) {
      p.erase(p.begin());
      int n = s.size();
      for (int i = 0; i < n; i++) r[p[i]] = i;
      for (int i = 0, k = 0; i < n; i++) {
        if (k) k--;
        if (!r[i]) continue;
        for (int j = p[r[i] - 1]; i + k < n && j + k < n && s[i + k] == s[j + k]; k++) ;
        h[r[i] - 1] = k;
      }
    }
  };
```

3.3 Palindrome Tree

```cpp
template <int N, int A>
struct PAM {
  std::array<int, A> ch[N];
  int fa[N], len[N], cnt;
  char s[N * 2];
  int l, r, pl, pr;

  // std::array<int, A> fch[N];
  // int diff[N], slink[N];
  // std::vector<std::array<int, 3>> rec;

  void init(int n) {
    fa[0] = fa[1] = cnt = 1, len[1] = -1;
    ch[0].fill(0), ch[1].fill(0);
    l = n + 1, r = n, s[l] = s[r] = -1;
    pl = pr = 0;

    // fch[0].fill(1), fch[1].fill(1);

    // rec.clear();
  }
  template <int D>
  int ins(char *now, int p) {
    auto jmp = [&](int x) {
      // while (now[D * ~len[x]] != *now)
      //   x = now[D * ~len[fa[x]]] == *now ? fa[x] : slink[x];

      // return now[D * ~len[x]] != *now ? fch[x][*now] : x;

      for (; now[D * ~len[x]] != *now; x = fa[x]) ;
      return x;
    };
    int x = *now;
    if (!ch[p = jmp(p)][x]) {
      // rec.back()[2] = p;
      int q = ++cnt;
      ch[q].fill(0);

      len[q] = len[p] + 2;
      fa[q] = ch[jmp(fa[p])][x];
      ch[p][x] = q;

      // fch[q] = fch[fa[q]];
      // fch[q][now[D * -len[fa[q]]]] = fa[q];

      // diff[q] = len[q] - len[fa[q]];
      // slink[q] = diff[q] == diff[fa[q]] ? slink[fa[q]] : fa[q];
    }
    return ch[p][x];
  }
  void push_back(char x) {
    // rec.push_back({pl << 1 | 1, pr, -1});
    s[++r] = x, s[r + 1] = -1;
    pr = ins<1>(s + r, pr);
    if (len[pr] == r - l + 1) pl = pr;
  }
  void push_front(char x) {
    // rec.push_back({pl << 1, pr, -1});
    s[--l] = x, s[l - 1] = -1;
    pl = ins<-1>(s + l, pl);
    if (len[pl] == r - l + 1) pr = pl;
  }
  // void undo() {  // require fch or slink jump
  //   auto [u, v, p] = rec.back();
  //   rec.pop_back();
  //   pl = u >> 1, pr = v;
  //   int x;
  //   if (u & 1) {
  //     x = s[r], s[r--] = -1;
  //   } else {
  //     x = s[l], s[l++] = -1;
  //   }
  //   if (p != -1) {
  //     ch[p][x] = 0, cnt--;
  //   }
  // }
};
```

节点：原串的**一个**本质不同的回文子串；长度 len[o]：回文子串长度；转移边 ch[o][x]：两边同时加字符 x；失配边 fa[o]：最长回文后缀，到根的路径是所有回文后缀。有两个根，奇根 1 和偶根 0，fa[0] = 1, len[1] = -1。

本质不同回文子串数：除根以外的点数；**回文子串出现次数**：失配树上对应节点子树大小 siz[o]；**奇偶性**：回文树边两端点 len 奇偶性相同，但失配树边不是。

不超过一半的最长回文后缀对应节点 f[o]：

```cpp
if (len[q] <= 2) {
  f[q] = fa[q];
} else {
  for (int &i = f[q] = f[p]; now[~len[i]] != *now || (len[i] +
  ↪  2 << 1) > len[q]; i = fa[i]) ;
  f[q] = ch[f[q]][x];
}
```

回文串的回文后 (前) 缀与其 Border 一一对应。某个串的回文后 (前) 缀长度可以拆分为 $O(\log n)$ 个等差数列。

设 v 为回文串 u 的最长回文严格前 (后) 缀，若 $2|v| \geq |u|$，那么 v 只会在 u 中恰好匹配两次，分别作为前缀和后缀。

用 slink[o] 来表示上一个等差数列的开头，一直跳 slink 最多会跳 log 次，可以将插入复杂度变为单次最坏 $O(\log n)$，在回溯操作时保证了复杂度最坏 $O(n \log n)$。

应用：区间本质不同字串数

```cpp
for (int i = 1; i <= n; i++) {
  int x = pos[i];
  root[i] = root[i - 1];
  for (; x > 1; x = slink[x]) {
    int j = ask(1, 1, cnt, in[x], out[x]);
    if (j) {
      ins(root[i], 1, n, j - len[x] + 1, -1);
    }
    ins(root[i], 1, n, i - len[slink[x]] - diff[x] + 1, 1);
  }
  update(1, 1, cnt, in[pos[i]], i);
}
```

相等问题转化为回文问题 (1) 把一个串 S 分割成偶数段 $s_1, s_2, \ldots, s_k$，且 $s_i = s_{k-i+1}$，求方案数，设 $S' = S_1 S_n S_2 S_{n-1} \ldots$ 问题转化为求 S' 偶回文划分的方案数；(2) 给定两个串 S 和 T，翻转 S 中最少的区间使得两串相等，设 $S' = S_1 T_1 S_2 T_2 \ldots$ 转化为 S' 的最少偶回文划分。

3.4 Manacher

```cpp
std::vector<int> d1(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 1 : std::min(d1[l + r - i], r - i);
  while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
    k++;
  }
  d1[i] = k--;
  if (i + k > r) {
    l = i - k;
    r = i + k;
  }
}
std::vector<int> d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 0 : std::min(d2[l + r - i + 1], r - i +
  ↪  1);
  while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i +
  ↪  k]) {
    k++;
  }
  d2[i] = k--;
  if (i + k > r) {
    l = i - k - 1;
    r = i + k;
  }
}
```

3.5 Z-function

对于个长度为 n 的字符串 s。定义函数 $z[i]$ 表示 s 和 $s[i, n-1]$（即以 $s[i]$ 开头的后缀）的最长公共前缀（LCP）的长度。z 被称为 s 的 Z 函数。特别地，$z[0] = 0$。

```cpp
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
      z[i] = max(0, r - i + 1);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

3.6 Min Rotation

```cpp
int minRotation(std::string s) {
  int a = 0, n = s.size();
  s += s;
  for (int b = 0; b < n; b++) {
    for (int k = 0; k < n; k++) {
      if (a + k == b || s[a + k] < s[b + k]) {
        b += std::max(0, k - 1);
        break;
      }
      if (s[a + k] > s[b + k]) {
        a = b;
        break;
      }
    }
  }
  return a; // (a, a + n)
}
```

3.7 Lyndon Word

一个字符串 s 是一个 Lyndon Word，当且仅当 s 是其所有后缀中最小的字符串。

Lyndon 分解：把字符串 s 分成若干部分 $s = s_1 s_2 s_3 \cdots s_m$，使得每个 s_i 都是 Lyndon Word，且 $\forall 1 \leq i < m : s_i \geq s_{i+1}$，这种划分存在且唯一。

```cpp
// Decom s to [0, i_1), [i_1, i_2), ..., [i_k, n)
for (int i = 0; i < n;) {
  int j = i, k = i + 1;
  while (k < n && s[j] <= s[k])
    j = s[j] != s[k++] ? i : j + 1;
  while (i <= j) res.push_back(i += k - j);
}
```

Math (4)

4.1 Number Theory

4.1.1 Fast Eratosthenes

```cpp
std::vector<int> sieve(int n) { // 1e9->1.1s
  if (n < 2) return {};
  constexpr int S = 31623; // S*S >= n
  std::array<bool, S + 1> np{};
  std::vector<int> pr = {2};
  pr.reserve(int(n / log(n) * 1.1));

  std::vector<PII> cp;
  for (int i = 3; i <= S; i += 2)
    if (!np[i]) {
      cp.push_back({i, i * i / 2});
      for (int j = i * i; j <= S; j += 2 * i) np[j] = 1;
    }
  for (int l = 1, r = n + 1 >> 1; l <= r; l += S) {
    std::array<bool, S> b{};
    for (auto &[p, k] : cp)
      for (int i = k; i < S + l; k = i += p) b[i - l] = 1;
    for (int i = 0; i < S && i < r - l; i++)
```

```
    if (!b[i]) pr.push_back((l + i) * 2 + 1);
  }
  return pr;
}
```

4.1.2 欧拉定理

$$a^b \equiv \begin{cases} a^b, & b < \varphi(m), \\ a^{(b \bmod \varphi(m)) + \varphi(m)}, & b \geq \varphi(m). \end{cases} \pmod m$$

4.1.3 GCD

```
LL exgcd(LL a, LL b, LL &x, LL &y) {
  if (!b) return x = 1, y = 0, a;
  LL g = exgcd(b, a % b, y, x);
  y -= a / b * x;
  return g;
}
LL inv(LL a, LL p) {
  LL x, y;
  exgcd(a, p, x, y);
  return (x % p + p) % p;
}
```

u128 卡常版本

```
u128 ctz(u128 x) {
  if (u64(x) == 0) return __builtin_ctzll(u64(x >> 64)) + 64;
  return __builtin_ctzll(u64(x));
}

u128 gcd(u128 a, u128 b) {
  if (b == 0) return a;
  int shift = ctz(a | b);
  b >>= ctz(b);
  while (a) {
    a >>= ctz(a);
    if (a < b) std::swap(a, b);
    a -= b;
  }
  return b << shift;
}
```

$O(M)$ 预处理，$O(1)$ 查询的快速 GCD（其实挺慢，不一定比得过上面那个卡常版本）

```
constexpr int N(5005), B(1000), M(B * B); // M 为值域
int a[N], b[N], n, ans;
std::bitset<M + 1> np;
int pr[M + 5], pn;
std::array<int, 3> vs[M + 5];
int gs[B + 5][B + 5];
void init() {
  vs[1].fill(1), np[1] = 1;
  for (int i = 2; i <= M; i++) {
    if (!np[i]) pr[++pn] = i, vs[i] = {1, 1, i};
    for (int j = 1; pr[j] * i <= M; j++) {
      np[i * pr[j]] = 1;
      auto &v = vs[i * pr[j]];
      v = {vs[i][0] * pr[j], vs[i][1], vs[i][2]};
      if (v[1] < v[0]) std::swap(v[1], v[0]);
      if (v[2] < v[1]) std::swap(v[2], v[1]);
      if (i % pr[j] == 0) break;
    }
  }
  for (int i = 1; i <= B; i++) {
    gs[0][i] = gs[i][0] = i;
    for (int j = 1; j <= B; j++) gs[i][j] = gs[j % i][i];
  }
}
int gcd(int a, int b) {
  int g = 1;
  for (int i = 0, x; i < 3; i++) {
    x = vs[a][i] > B
        ? b % vs[a][i] ? 1 : vs[a][i]
        : gs[vs[a][i]][b % vs[a][i]];
    b /= x, g *= x;
  }
  return g;
}
```

4.1.4 CRT

解同余方程 $x \equiv a \pmod m, x \equiv b \pmod n$。如果 $|a| < m, |b| < n$ 则 $0 \leq x < \mathrm{lcm}(m, n)$. Assumes $mn < 2^{62}$.

```
LL crt(LL a, LL m, LL b, LL n) {
  if (n > m) std::swap(a, b), std::swap(m, n);
  LL x, y, g = exgcd(m, n, x, y);
  assert((a - b) % g == 0);  // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m * n / g : x;
}
```

4.1.5 Kummer 定理

设 m, n 为正整数，p 为素数，则 $\binom{n+m}{n}$（质因数分解后）p 的幂次等于 $m+n$ 在 p 进制下的进位次数。特别地，$\binom{n+m}{n} \equiv 1 \pmod 2$ 等价于 $n \ \& \ m = 0$。

4.1.6 Lucas

```
struct BinP {
  int p, q, m;  // p^q <= 2e7
  std::vector<int> fac, ifac;
  int delta;

  using ULL = uint64_t;
  using u128 = __uint128_t;
  ULL im, ip;

  BinP(int _p, int _q) : p(_p), q(_q), m(1) {
    assert(_q > 0);
    while (_q--) m *= p;
    im = ULL(-1) / m + 1;
    ip = ULL(-1) / p + 1;
    fac.resize(m);
    ifac.resize(m);
    fac[0] = fac[1] = 1;
    for (int i = 2; i < m; i++) {
      fac[i] = i % p ? mod(1LL * fac[i - 1] * i) : fac[i - 1];
    }
    ifac[m - 1] = fpow(fac[m - 1], m / p * (p - 1) - 1);
    for (int i = m - 1; i; i--) {
      ifac[i - 1] = i % p ? mod(1LL * ifac[i] * i) : ifac[i];
    }
    delta = p == 2 && q >= 3 ? 1 : m - 1;
  }

  LL mod(ULL n) {
    LL r = n - ULL((u128(n) * im) >> 64) * m;
    return r < 0 ? r + m : r;
  }
  int fpow(int a, LL k) {
    int r = 1;
    for (; k; k >>= 1, a = mod(1LL * a * a)) {
      if (k & 1) r = mod(1LL * r * a);
    }
    return r;
  }
  LL div_p(ULL n) {
    ULL x = (u128(n) * ip) >> 64;
    LL r = n - x * p;
    return r < 0 ? x - 1 : x;
  }
  std::pair<LL, int> quo_p(ULL n) {
    ULL x = (u128(n) * ip) >> 64;
    LL r = n - x * p;
    if (r < 0) r += m, x--;
    return {LL(x), r};
```

```cpp
  }

  LL lucas(LL n, LL m) {
    int res = 1;
    while (n) {
      int n0, m0;
      std::tie(n, n0) = quo_p(n);
      std::tie(m, m0) = quo_p(m);
      if (n0 < m0) return 0;
      res = mod(1LL * res * fac[n0]);
      res = mod(1LL * res * ifac[m0]);
      res = mod(1LL * res * ifac[n0 - m0]);
    }
    return res;
  }
  LL bin(LL n, LL m) {
    if (n < m || n < 0 || m < 0) return 0;
    if (q == 1) return lucas(n, m);
    LL r = n - m;
    int e0 = 0, eq = 0, i = 0;
    int res = 1;
    while (n) {
      res = mod(1LL * res * fac[mod(n)]);
      res = mod(1LL * res * ifac[mod(m)]);
      res = mod(1LL * res * ifac[mod(r)]);
      n = div_p(n), m = div_p(m), r = div_p(r);
      int eps = n - m - r;
      e0 += eps;
      if (e0 >= q) return 0;
      if (++i >= q) eq += eps;
    }
    res = mod(1LL * res * fpow(delta, eq));
    res = mod(1LL * res * fpow(p, e0));
    return res;
  }
};

struct BinA {
  int mod;  // <= 1e7
  std::vector<int> m;
  std::vector<LL> ims;
  std::vector<BinP> cs;

  BinA(LL x) : mod(x) {
    assert(1 <= x);
    for (int i = 2; i * i <= x; i++) {
      if (x % i == 0) {
        int j = 0, k = 1;
        while (x % i == 0) x /= i, j++, k *= i;
        m.push_back(k);
        cs.emplace_back(i, j);
      }
    }
    if (x != 1) {
      m.push_back(x);
      cs.emplace_back(x, 1);
    }
    LL m0 = 1;
    for (int i = 0; i < m.size(); i++) {
      LL m1 = m[i];
      if (m0 < m1) std::swap(m0, m1);
      ims.push_back(inv(m0, m1));
      m0 *= m1;
    }
  }
  LL bin(LL n, LL k) {
    if (mod == 1) return 0;
    LL r0 = 0, m0 = 1;
    for (int i = 0; i < m.size(); i++) {
      LL r1 = cs[i].bin(n, k), m1 = m[i], im = ims[i];
      if (m0 < m1) std::swap(r0, r1), std::swap(m0, m1);
      LL x = (r1 - r0) * im % m1;
      r0 += x * m0;
      m0 *= m1;
      if (r0 < 0) r0 += m0;
    }
    return r0;
  }
};
```

4.1.7 $O(1)$ 逆元

```cpp
// online
template <int P>
struct Inv {
  static constexpr int B = 1024, M = B * B, N = M + 1;
  int l[N], r[N], y[N], inv[M * 2];
  Inv() {
    inv[1] = 1;
    for (int i = 2; i < M * 2; i++) {
      inv[i] = LL(P - P / i) * inv[P % i] % P;
    }
    y[0] = 1;
    for (int i = 1; i < B; i++) {
      for (int j = i; j < B; j++) {
        int k = i * M / j;
        if (!y[k]) y[k] = j;
      }
    }
    l[0] = 1;
    for (int i = 1; i <= M; i++) l[i] = y[i] ?: l[i - 1];
    r[M] = 1;
    for (int i = M - 1; ~i; i--) r[i] = y[i] ?: r[i + 1];
    for (int i = 0; i <= M; i++) y[i] = std::min(l[i], r[i]);
  }
  int operator()(int x) {
    if (x < M * 2) return inv[x];
    int k = 1LL * x * M / P, j = 1LL * y[k] * x % P;
    return LL(j < M * 2 ? inv[j] : P - inv[P - j]) * y[k] % P;
  }
};

// offline
std::vector<int> inv(const std::vector<int> &a) {
  int n = a.size();
  std::vector<int> b(n + 1);
  b[n] = 1;
  for (int i = n; i; i--) b[i - 1] = 1LL * b[i] * a[i - 1] %
  ↪  P;
  for (int i = 0, o = fpow(b[0]); i < n; i++) {
    b[i] = 1LL * o * b[i + 1] % P;
    o = 1LL * o * a[i] % P;
  }
  b.pop_back();
  return b;
}
```

4.2 Matrix

4.2.1 Determinant

```cpp
int det(std::vector<std::vector<int>> &a) {
  int n = a.size();
  int ans = 1;
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      while (a[j][i]) {
        auto t = a[i][i] / a[j][i]; // div
        if (t) for (int k = i; k < n; k++) a[i][k] -= a[j][k]
        ↪  * t;
        std::swap(a[i], a[j]), ans *= -1;
      }
    }
    ans *= a[i][i];
    if (!ans) return 0;
  }
  return ans;
}
```

4.2.2 Inverse

```cpp
using M = std::vector<std::vector<int>>;
M inv(M a) {
#define REP(i, a, b) for (int i(a); i < (b); i++)
  int n = a.size();
  std::vector<int> p(n, -1), q(n, -1);
  REP(k, 0, n) {
    REP(i, k, n) if (p[k] == -1) REP(j, k, n) if (a[i][j]) {
      p[k] = i, q[k] = j; break;
    }
    if (p[k] == -1) return {};
    std::swap(a[k], a[p[k]]);
    REP(i, 0, n) std::swap(a[i][k], a[i][q[k]]);
    if (!a[k][k]) return {};
    a[k][k] = fpow(a[k][k]);
    REP(i, 0, n) if (i != k) a[k][i] = 1LL * a[k][i] * a[k][k] % P;
    REP(i, 0, n) if (i != k) REP(j, 0, n) if (j != k)
      a[i][j] = (a[i][j] + 1LL * (P - a[i][k]) * a[k][j]) % P;
    REP(i, 0, n)
      if (i != k) a[i][k] = 1LL * (P - a[i][k]) * a[k][k] % P;
  }
  for (int k = n - 1; k >= 0; k--) {
    std::swap(a[k], a[q[k]]);
    REP(i, 0, n) std::swap(a[i][k], a[i][p[k]]);
  }
  return a;
}
```

4.2.3 Char Poly

```cpp
std::vector<Z> charPoly(std::vector<std::vector<Z>> a) {
  int n = a.size();
  for (int j = 0; j < n - 2; j++) {
    for (int i = j + 1; i < n; i++) {
      if (!a[i][j]) continue;
      std::swap(a[i], a[j + 1]);
      for (int k = 0; k < n; k++)
        std::swap(a[k][i], a[k][j + 1]);
      break;
    }
    if (a[j + 1][j]) {
      auto inv = a[j + 1][j].inv();
      for (int i = j + 2; i < n; i++) {
        auto c = a[i][j] * inv;
        for (int k = 0; k < n; k++) a[i][k] -= a[j + 1][k] * c;
        for (int k = 0; k < n; k++) a[k][j + 1] += a[k][i] * c;
      }
    }
  }
  std::vector<std::vector<Z>> h(n + 1);
  h[0] = {1};
  for (int i = 0; i < n; i++) {
    Z prod = 1;
    h[i + 1].resize(h[i].size() + 1);
    for (int j = 0; j < h[i].size(); j++) {
      h[i + 1][j + 1] = h[i][j];
      h[i + 1][j] -= h[i][j] * a[i][i];
    }
    for (int j = 0; j < i; j++) {
      prod *= a[i - j][i - j - 1];
      auto c = a[i - j - 1][i] * prod;
      for (int k = 0; k < h[i - j - 1].size(); k++)
        h[i + 1][k] -= h[i - j - 1][k] * c;
    }
  }
  return h[n];
}
```

4.2.4 Minor

求余子式 $M_{i,j}$（将第 i 行第 j 列划掉后的行列式），$O(n^3)$

```cpp
M cal(M &a) {
  int n = a.size();
  assert(n && a[0].size() == n);
  M b(n, V(n * 2));
  for (int i = 0; i < n; i++) {
    std::copy(a[i].begin(), a[i].end(), b[i].begin());
    b[i][i + n] = 1;
  }

  int cnt = 0, sgn = 1;
  for (int i = 0, t = 0; i < n && t < n; i++, t++) {
    for (int j = i + 1; j < n; j++) {
      while (b[j][t]) {
        int v = b[i][t] / b[j][t];
        if (v) for (int k = t; k < n * 2; k++) b[i][k] = (b[i][k] + LL(P - v) * b[j][k]) % P;
        b[i].swap(b[j]), sgn *= -1;
      }
    }
    if (!b[i][t]) {
      i--;
      if (++cnt >= 2) return M(n, V(n));
    }
  }
  if (!cnt) {
    inc(sgn, P);
    for (int i = n - 1; i >= 0; i--) {
      sgn = 1LL * sgn * b[i][i] % P;
      for (int j = i + 1; j < n; j++) {
        int v = P - b[i][j];
        for (int k = n; k < n * 2; k++)
          b[i][k] = (b[i][k] + 1LL * v * b[j][k]) % P;
      }
      int v = fpow(b[i][i]);
      for (int k = n; k < n * 2; k++)
        b[i][k] = 1LL * b[i][k] * v % P;
    }
    M r(n, V(n));
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        r[i][j] = 1LL * sgn * b[i][j + n] % P;
      }
    }
    return r;
  }

  auto gauss = [&](M &a, V &p) {
    int r = -1;
    for (int i = 0, t = 0; i < n && t < n; i++, t++) {
      for (int j = i + 1; j < n; j++) {
        while (a[j][t]) {
          int v = a[i][t] / a[j][t];
          if (v) for (int k = t; k < n; k++) a[i][k] = (a[i][k] + LL(P - v) * a[j][k]) % P;
          a[i].swap(a[j]);
        }
      }
      if (!a[i][t]) i--, r = t;
    }
    p[r] = 1;
    for (int i = n - 2, k; i >= 0; i--) {
      k = i < r ? i : i + 1;
      p[k] = 0;
      for (int j = k + 1; j < n; j++)
        p[k] = (p[k] + 1LL * p[j] * a[i][j]) % P;
      p[k] = LL(P - p[k]) * fpow(a[i][k]) % P;
    }
    return r;
  };

  M c(n, V(n));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
```

```
    c[i][j] = a[j][i];

  V p(n), q(n);
  int u = gauss(c, p), v = gauss(b, q);

  a.erase(a.begin() + u);
  for (auto &i : a) i.erase(i.begin() + v);

  int res = det(a);
  if (u + v & 1) res = P - res;

  M r(n, V(n));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      r[i][j] = 1LL * p[j] * q[i] % P * res % P;
  return r;
}
```

4.3 Burnside

4.3.1 Burnside

设 A 和 B 为有限集合，X 为一些从 A 到 B 的映射组成的集合。G 是 A 上的置换群，且 X 中的映射在 G 中的置换作用下封闭。X/G 表示 G 作用在 X 上产生的所有等价类的集合（若 X 中的两个映射经过 G 中的置换作用后相等，则它们在同一等价类中），则

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|$$

其中 $|S|$ 表示集合 S 中元素的个数，且

$$X^g = \{x|x\in X, g(x) = x\}$$

4.3.2 Pólya

在与 Burnside 引理相同的前置条件下，若 X 为 ** 所有 ** 从 A 到 B 的映射，内容修改为

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|B|^{c(g)}$$

其中 $c(g)$ 表示置换 g 能拆分成的不相交的循环置换的数量。

4.4 BM

对于 $n \ge k$，$\sum_{i=0}^{k} q_i a_{n-i} = 0$，其中 $q_0 \ne 0$。
数列 a_n 的生成函数为 $A(x)$。
上式左边看作卷积，右边看作生成函数 $P(x)$ 的 $n \ge k$ 时的系数。

$$Q(x)A(x) = P(x)$$

$$A(x) = \frac{P(x)}{Q(x)}$$

$P(x)$ 为 $k-1$ 次多项式，$Q(x)$ 为 k 次多项式。

```
// usage: q = bm(a),p=(q*a).pre(b.size()-1), divAt(p,q,n);
using Poly = std::vector<int>;
Poly bm(Poly a) {
  Poly p = {1}, q = {1};
  int l = 0;
  for (int r = 1; r <= a.size(); r++) {
    int d = 0;
    for (int j = 0; j <= l; j++)
      d = (d + 1LL * a[r - 1 - j] * p[j]) % P;
    q.insert(q.begin(), 0);
    if (!d) continue;
    Poly t = p;
    if (q.size() > t.size()) t.resize(q.size());
    for (int i = 0; i < q.size(); i++)
      t[i] = (t[i] - 1LL * d * q[i]) % P;
    if (2 * l <= r - 1) {
      q = p;
      int od = fpow(d);
      for (int &x : q) x = 1LL * x * od % P;
      l = r - l;
    }
    p.swap(t);
  }
  assert((int)p.size() == l + 1);
  for (auto &x : p) if (x < 0) x += P;
  return p;
}
int divAt(Poly p, Poly q, LL n) {
  for (; n; n >>= 1) {
    Poly r(q);
    for (int i = 1; i < r.size(); i += 2) r[i] = P - r[i];
    Poly u = p * r, v = q * r;
    p.clear();
    for (int i = n & 1; i < u.size(); i += 2)
      p.push_back(u[i]);
    for (int i = 0; i < q.size(); i++) q[i] = v[i * 2];
  }
  return 1LL * p[0] * fpow(q[0]) % P;
}
```

4.5 LGV Lemma

设 $G(V,E)$ 是一个 DAG，起点集 $A = \{a_1,\dots,a_n\} \subseteq V$，终点集 $B = \{b_1,\dots,b_n\} \subseteq V$ 。
记 ω_e 表示有向边 e 的边权，ω_e 属于某个**交换环**。
对于有向路径 P，定义 $\omega(P)$ 表示路径上边权的乘积，即

$$\omega(P) = \prod_{e\in P}\omega_e$$

对于任意两点 a,b，记 $e(a,b)$ 表示从 a 到 b 的所有路径的权值之和，即

$$e(a,b) = \sum_{P:a\to b}\omega(P)$$

特别地，若 $\omega_e = 1$ ，$e(a,b)$ 表示从 a 到 b 的路径数。
记矩阵

$$M = \begin{pmatrix} e(a_1,b_1) & e(a_1,b_2) & \cdots & e(a_1,b_n) \\ e(a_2,b_1) & e(a_2,b_2) & \cdots & e(a_2,b_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(a_n,b_1) & e(a_n,b_2) & \cdots & e(a_n,b_n) \end{pmatrix}$$

定义从 A 到 B 的 n 元不相交路径组表示 G 中的 n 条有向路径 $(P_1,\dots,P_n)$ 满足：

1. 存在 $\{1,2,\dots,n\}$ 的排列 σ 使得对于 $i=1,2,\dots,n$, 路径 P_i 是从 a_i 到 b_{σ_i} 的有向路径。
2. 对于任意 $i \ne j$，路径 P_i, P_j 没有公共点。

记 $\sigma(P)$ 表示 1 中的排列 σ 。
Lindström–Gessel–Viennot lemma

$$\det(M) = \sum_{(P_1,P_2,\dots,P_n):A\to B}\operatorname{sgn}(\sigma(P))\prod_{i=1}^{n}\omega(P_i)$$

其中

$$\operatorname{sgn}(\sigma(P)) = (-1)^{\pi(\sigma(P))}$$

特别地，若 $\sigma(P)$ 只可能是 $\{1,2,\dots,n\}$ ，令 $\omega_e = 1$，此时 $\det(M)$ 等于 A 到 B 的不相交路径数。

4.6 Min25

约定

- $\mathbf{P}$ 表示全体质数集合。
- p_i 表示第 i 个质数，特别地，$p_0 = 1$。
- lpf_i 表示 i 的最小质因子。

计算数论函数 $f(n)$ 的前缀和，要求 $f(p_i)$ 可以快速计算（如低阶多项式等），且对于合数 n，$f(n)$ 可以写成形如 $f(n) = A(\mathrm{lpf}_n^e)B\left(\frac{n}{\mathrm{lpf}_n^e}\right)$ 的式子，其中 $A(p^e)$ 可以快速求值或预处理。
当 $f(n)$ 是积性函数时，$A(p^e) = f(p^e), B(n) = f(n)$。
$f(n)$ 可以不是积性函数，而且 Part 2 部分的 cal 函数不需要记忆化，可以通过增加参数来增加功能。典型的例子是 $f(\sum p^c) = [\sum c \bmod 2 = 0\ \&\ 2\max\{c\} \le \sum c]$，通过增加参数 sum, max 表示已经考虑过的质数（前 $i-1$ 个质数）的指数和与最大值。

```cpp
constexpr int N(2e5 + 5);
int np[N], pr[N], pn;
Z sum[N][2], pw[N][45];
void sieve(int n) {
  pn = 0;
  for (int i = 2; i <= n; i++) {
    if (!np[i]) {
      np[i] = i;
      pr[++pn] = i;
      sum[pn][0] = sum[pn - 1][0] + i;
      sum[pn][1] = sum[pn - 1][1] + 1LL * i * i;
      pw[pn][0] = 1;
      for (int j = 1; j < 40; j++) {
        pw[pn][j] = pw[pn][j - 1] * i;
      }
    }
    for (int j = 1; j <= pn && pr[j] * i <= n; j++) {
      np[i * pr[j]] = pr[j];
      if (i % pr[j] == 0) break;
    }
  }
}

LL n, val[N];
int m, vn, id1[N], id2[N];
int &gid(LL x) { return x <= m ? id1[x] : id2[n / x]; }
Z g1[N], g2[N];
Z f1(int n) { // 2 + ... + n
  return 1LL * n * (n + 1) / 2 - 1;
}
Z f2(int n) { // 2^2 + ... + n^2
  return Z(n) * (n + 1) * (2 * n + 1) / 6 - 1;
}
Z f(int i, int k) { // f(p_i^k)
  return pw[i][k] * (pw[i][k] - 1);
}
Z cal(LL x, LL i) {
  if (x <= 1 || pr[i] > x) return 0;
  int k = gid(x);
  Z ans = (g2[k] - g1[k]) - (sum[i - 1][1] - sum[i - 1][0]);
  for (int j = i; j <= pn && 1LL * pr[j] * pr[j] <= x; j++) {
    for (LL k = 1, s = pr[j]; s * pr[j] <= x; k++, s *= pr[j])
    ↪ {
      ans += f(j, k + 1) + f(j, k) * cal(x / s, j + 1);
    }
  }
  return ans;
}
void solve() {
  std::cin >> n;
  m = sqrt(n);
  sieve(m);
  vn = 0;
  for (LL l = 1, r; l <= n; l = r + 1) {
    LL x = n / l;
    r = n / x;
    val[++vn] = x;
    gid(x) = vn;
    g1[vn] = f1(x % P);
    g2[vn] = f2(x % P);
  }
  for (int i = 1; i <= pn; i++) {
    LL t = 1LL * pr[i] * pr[i];
    for (int j = 1; val[j] >= t; j++) {
      LL x = val[j] / pr[i];
      int k = gid(x);
      g1[j] -= (g1[k] - sum[i - 1][0]) * pr[i];
      g2[j] -= (g2[k] - sum[i - 1][1]) * pr[i] * pr[i];
    }
  }
  std::cout << cal(n, 1) + 1 << "\n";
}
```

如果只算素数个数，用下面的代码，还有卡常版本

```cpp
constexpr int N(1e6 + 5);
LL f1[N], f2[N], x0, w1, w2, dd;
double inv[N];
LL cal(LL n) {
  int lim = sqrt(n);
  for (int i = 1; i <= lim; i++)
    f1[i] = i - 1, f2[i] = n / i - 1, inv[i] = 1.0 / i;
  for (int p = 2; p <= lim; p++)
    if (f1[p] != f1[p - 1]) {
      x0 = f1[p - 1], w1 = lim / p, w2 = std::min(0LL + lim,
      ↪  (n / p / p)),
      dd = n / p;
      for (int i = 1; i <= lim / p; i++) f2[i] += x0 - f2[i *
      ↪  p];
      for (int i = lim / p + 1; i <= w2; i++)
        f2[i] += x0 - f1[(int)(inv[i] * dd + 1e-7)];
      for (int i = lim; i >= 1LL * p * p; i--)
        f1[i] += x0 - f1[(int)(inv[p] * i + 1e-7)];
    }
  return f2[1];
}

LL cal(LL n) {
  if (n <= 1) return 0;
  if (n == 2) return 1;
  int v = sqrtl(n), s = v + 1 >> 1, pc = 0;
  std::vector<int> a(s), b(s);
  std::vector<LL> c(s);
  std::vector<bool> vis(v + 1);
  for (int i = 1; i < s; ++i) a[i] = i;
  for (int i = 0; i < s; ++i) b[i] = 2 * i + 1;
  for (int i = 0; i < s; ++i) c[i] = (n / (2 * i + 1) - 1) /
  ↪  2;
  for (int p = 3; p <= v; p += 2)
    if (!vis[p]) {
      int q = p * p;
      if (LL(q) * q > n) break;
      vis[p] = true;
      for (int i = q; i <= v; i += 2 * p) vis[i] = true;
      int ns = 0;
      for (int k = 0; k < s; ++k) {
        int i = b[k];
        if (vis[i]) continue;
        LL d = LL(i) * p;
        c[ns] = c[k] - (d <= v ? c[a[d >> 1] - pc] : a[n / d -
        ↪  1 >> 1]) + pc;
        b[ns++] = i;
      }
      s = ns;
      for (int i = v - 1 >> 1, j = ((v / p) - 1) | 1; j >= p;
      ↪  j -= 2) {
        int c = a[j >> 1] - pc;
        for (int e = (j * p) >> 1; i >= e; --i) a[i] -= c;
      }
      ++pc;
    }
  c[0] += LL(s + 2 * (pc - 1)) * (s - 1) / 2;
  for (int k = 1; k < s; ++k) c[0] -= c[k];
  for (int l = 1; l < s; ++l) {
    int q = b[l];
    LL m = n / q;
    int e = a[m / q - 1 >> 1] - pc;
    if (e < l + 1) break;
    LL t = 0;
    for (int k = l + 1; k <= e; ++k) t += a[m / b[k] - 1 >>
    ↪  1];
    c[0] += t - LL(e - l) * (pc + l - 1);
  }
  return c[0] + 1;
}
```

4.7 Powerful Number

```cpp
void dfs(LL x, Z o, int k) {
  ans += o * calc(n / x);
```

```
  if(k <= plen && x > n / p[k] / p[k]) return;
  for(int s = k; s <= plen; s++) {
    if(s > 1 && x > n / p[s] / p[s]) break;
    LL y = (LL)p[s] * p[s];
    Z sum(p[s]);
    for(int j = 2; x * y <= n; j++, y *= p[s]) {
      sum *= p[s];
      Z u(y); u *= inv[j];
      u -= sum, sum += u;
      dfs(x * y, o * u, s + 1);
    }
  }
}
```

4.8 Mod Log

```
int modLog(int a, int b, int p) { // Assume: 0 ≤ a,b < p
  if (b == 1 || p == 1) return 0;
  int d = std::gcd(a, p);
  if (d != 1) {
    if (b % d) return -1;
    p /= d;
    int ans = modLog(a, 1LL * b / d * inv(a / d, p) % p, p);
    return ~ans ? ans + 1 : ans;
  }
  HashTable::Map mp;
  int t = sqrt(p) + 1, pw = 1;
  for (int i = 0; i < t; i++, pw = 1LL * pw * a % p) {
    mp[1LL * pw * b % p] = i;
  }
  for (int i = 1, a = pw; i <= t; i++, pw = 1LL * pw * a % p)
  ↪  {
    auto it = mp.find(pw);
    if (it != mp.end()) return i * t - it->second;
  }
  return -1;
}
```

4.9 Mod Sqrt

```
// Jacobi symbol (a/m)
//   m > 0, m: odd
int jacobi(LL a, LL m) {
  int s = 1;
  if (a < 0) a = a % m + m;
  while (m > 1)  {
    if (!(a %= m)) return 0;
    int r = __builtin_ctzll(a);
    if (r & 1 && m + 2 & 4) s = -s;
    if ((a >>= r) & m & 2) s = -s;
    std::swap(a, m);
  }
  return s;
}

// sqrt(a) (mod p)
//   p: prime, p < 2^31, -p^2 <= a <= P^2
//   (b + sqrt(b^2 - a))^((p+1)/2) in F_p(sqrt(b^2 - a))
std::vector<LL> modSqrt(LL a, LL p) {
  static std::mt19937 rnd(1024);
  if (p == 2) return {a & 1};
  int j = jacobi(a, p);
  if (j == 0) return {0};
  if (j == -1) return {};
  LL b, d;
  for (;;) {
    b = rnd() % p, d = (b * b - a) % p;
    if (d < 0) d += p;
    if (jacobi(d, p) == -1) break;
  }
  LL x = b, y = 1, z = 1, w = 0, t;
  for (LL k = p + 1 >> 1; k; k >>= 1) {
    if (k & 1) {
      t = (z * x + d * (w * y % p)) % p;
      w = (z * y + w * x) % p, z = t;
    }
    t = (x * x + d * (y * y % p)) % p;
    y = (2 * x * y) % p, x = t;
  }
  if (z < p - z) return {z, p - z};
  return {p - z, z};
}
```

4.10 Mod Mul

The following algorithm computes $a \times b \bmod c$ for $0 \le a, b \le c < 7.268 \cdot 10^{18}$: [1]

```
ULL modMul(ULL a, ULL b, ULL c) {
  LL r = a * b - c * ULL(1.L / c * a * b);
  return r + c * (r < 0) - c * (r >= (LL)c);
}
```

[1] this number equals $r \cdot 2^{64}$, where $r = (\sqrt{177} - 7)/16 \approx 0.394$ is the positive solution to the equation $8x^2 + 7x = 4$.

4.11 Pollard Rho

```
LL fpow(LL x, LL k, LL p) {
  LL r = 1;
  for (; k; k >>= 1, x = modMul(x, x, p)) {
    if (k & 1) r = modMul(r, x, p);
  }
  return r;
}
bool isPrime(LL n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  LL s = __builtin_ctzll(n - 1), d = n >> s;

  // int: {2, 7, 61}
  for (LL a : {2, 325, 9375, 28178, 450775, 9780504,
  ↪  1795265022}) {
    LL p = fpow(a % n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--) {
      p = modMul(p, p, n);
    }
    if (p != n - 1 && i != s) return false;
  }
  return true;
}
LL rho(LL n) {
  auto f = [&](LL x) { return modMul(x, x, n) + 1; };
  LL p = 2, q;
  for (LL i = 1, x = 0, y = 0, t = 30; t++ % 40 || std::gcd(p,
  ↪  n) == 1; x = f(x), y = f(f(y))) {
    if (x == y) x = ++i, y = f(x);
    q = modMul(p, x < y ? y - x : x - y, n);
    if (q) p = q;
  }
  return std::gcd(p, n);
}
std::vector<LL> factor(LL n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  LL x = rho(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), r.begin(), r.end());
  return l;
}
```

4.12 万欧

$$F(P, R, Q, N, X, Y) = \prod_{i=0}^{N} \left(Y^{f(i)-f(i-1)} X \right)$$

其中 $f(i) = \lfloor \frac{Pi+R}{Q} \rfloor, f(-1) = 0$。$X, Y \in$ 半群 $(S, \times)$。
S 即半群 $(S, \times)$ 的结构体，这里假定它是幺半群（否则可以加一个幺元进去），则 S() 代表幺元。
要注意：是从 0 开始的。

```
LL div(LL p, LL i, LL r, LL q) {
  return (p * i + r) / q; // int128 may be in need
}
S cal(LL p, LL r, LL q, LL n, const S &x, const S &y) {
  if (n < 0) return S();
  if (r >= q) {
    return fpow(y, r / q) * cal(p, r % q, q, n, x, y);
  }
  if (p >= q) {
    return x * cal(p % q, p % q + r, q, n - 1, fpow(y, p / q)
    ↪ * x, y);
  }
  if (p == 0) {
    return fpow(x, n + 1);
  }
  LL m = div(p, n, r, q);
  return cal(q, q - r + p - 1, p, m - 1, y, x) * fpow(x, n + 1
  ↪ - (!m ? 0 : div(q, m, p - r - 1, p)));
}
```

4.13 高斯整数

高斯质数，对于任意一个数字 c 能否表达成 $c^2 = a^2 + b^2$ 的形式实际上可以写成高斯整数的形式 $c^2 = (a+bi)(a-bi)$，那么我们可以对每一个质数单独分解。
根据高斯质数的规律，质数分为三种：$2, 4k+1, 4k+3$ 型。本质上只有 $4k+1$ 型会对答案有翻倍的贡献，设 k_j 为 $4k+1$ 型的质数相应的指数，则在一个二维平面上的合法解即为：$4\prod_j (k_j+1)$。
如何分解 $4k+1$ 型的质数 p，可以考虑找到 $k^2 \equiv -1 \pmod p$ 的任意一个 k，则 $k^2+1 = (k+i)(k-i)$，所以 $p|(k+i)(k-i)$，我们直接取 $\gcd(p, k+i)$ 即可得到分解中任意一解，再用 p 除即可得到另一解。
考虑找到 k，我们可以随机一个 $t \neq 0 \pmod p$，再令 $k = t^{\frac{p-1}{4}} \mod p$，据二次剩余理论，$k^2 \equiv \pm 1 \pmod p$，且取值为 -1 的概率为 $\frac{1}{2}$，随便找几个数试一下即可。

```
using i128 = __int128_t;
i128 absl(const i128 &u) { return u < 0 ? -u : u; }
i128 div(const i128 &a, const i128 &n) {
  i128 now = (a % n + n) % n;
  return (a - now) / n;
}
struct G {
  i128 r, i;
  G() {}
  G(i128 _r, i128 _i) : r(_r), i(_i) {}
  friend G operator+(const G &a, const G &b) {
    return {a.r + b.r, a.i + b.i};
  }
  friend G operator-(const G &a, const G &b) {
    return {a.r - b.r, a.i - b.i};
  }
  friend G operator*(const G &a, const G &b) {
    return {a.r * b.r - a.i * b.i, a.i * b.r + b.i * a.r};
  }
  friend G operator*(const G &a, const i128 &b) {
    return {a.r * b, a.i * b};
  }
  friend G operator/(const G &a, const i128 &b) {
    return {a.r / b, a.i / b};
  }
  friend bool operator==(const G &a, const G &b) {
    return a.r == b.r && a.i == b.i;
  }
  friend G operator/(const G &a, const G &b) {
    i128 len = b.r * b.r + b.i * b.i;
    G c = a * G(b.r, -b.i);
    return {div(2 * c.r + len, len * 2),
            div(2 * c.i + len, len * 2)};
  }
  friend G operator%(const G &a, const G &b) {
    G u = a / b;
    return a - (u * b);
  }
};
G norm(const G &u) {
  if (u.r == 0) return G(u.i < 0 ? -u.i : u.i, 0);
  switch ((u.r < 0) << 1 | (u.i < 0)) {
    case 0: return G(u.r, u.i); break;
    case 1: return G(-u.i, u.r); break;
    case 2: return G(u.i, -u.r); break;
    case 3: return G(-u.r, -u.i);break;
  }
  return G();
}
G gcd(G a, G b) {
  if (b.r == 0 && b.i == 0) return a;
  return gcd(b, a % b);
}
G Gpow(G a, i128 k) {
  G ans(1, 0);
  for (; k; k >>= 1, a = a * a) {
    if (k & 1) ans = ans * a;
  }
  return ans;
}
G GaussPrime(LL p) {
  if (p % 4 == 1) {
    LL t = 1;
    while (pow_mod(t, p >> 1, p) != p - 1) t++;
    return gcd(G(p, 0), G(pow_mod(t, p >> 2, p), 1));
  } else if (p == 2)
    return G{1, 1};
  else
    return G{p, 0};
}
vector<G> solveComposite(LL n) {  // 求高斯整数
  // vector<PII> {(p_i, c_i)} 质因数分解
  auto prm = Factorization::work(n);
  vector<G> v{{1, 0}};
  LL coef = 1;
  for (auto u : prm) {
    LL p = u.first, k = u.second << 1;
    if (p % 4 != 1) {
      for (int i = 0; i < (k >> 1); i++) coef *= p;
    } else {
      G a = GaussPrime(p);
      G b = G(p, 0) / a;
      G q = Gpow(a, k);
      auto tmp = v;
      v.resize((k + 1) * tmp.size());
      int l = 0;
      for (int i = 0; i <= k; i++) {
        for (const auto &e : tmp) v[l++] = e * q;
        q = q / a * b;
      }
    }
  }
  for (auto &u : v) u = norm(u) * coef;
  return v;
}
```

4.14 Simplex

```
using D = double;
using LD = long double;
using F = __float128;

constexpr int N(50);
constexpr F EPS(1e-16);

int sgn(D x) { return x < -EPS ? -1 : (x > EPS ? 1 : 0); }

struct Simplex {
```

```cpp
F a[N][N], ans[N];
int id[N];
int work(int n, int m) {
  auto pivot = [&](int l, int e) {
    std::swap(id[n + l], id[e]);
    F d = -a[l][e];
    a[l][e] = -1;
    for (int i = 0; i <= n; i++) a[l][i] /= d;
    for (int i = 0; i <= m; i++) {
      if (i != l && sgn(a[i][e])) {
        d = a[i][e], a[i][e] = 0;
        for (int j = 0; j <= n; j++) {
          a[i][j] += d * a[l][j];
        }
      }
    }
  };

  a[0][0] = 0;
  for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
      a[i][j] = -a[i][j];
    }
  }
  for (int i = 1; i <= n + m; i++) id[i] = i;
  for (;;) {
    int l = 1, e = 0;
    for (int i = 1; i <= m; i++) {
      if (a[i][0] < a[l][0]) l = i;
    }
    if (sgn(a[l][0]) >= 0) break;
    for (int j = 1; j <= n; j++) {
      if (sgn(a[l][j]) > 0 && (!e || id[j] > id[e])) {
        e = j;
      }
    }
    if (!e) return 0;
    pivot(l, e);
  }

  for (;;) {
    int l = 0, e = 1;
    F min = 1e9;
    for (int j = 1; j <= n; j++) {
      if (a[0][j] > a[0][e]) e = j;
    }
    if (sgn(a[0][e]) <= 0) {
      for (int i = 1; i <= n; i++) ans[i] = 0;
      for (int i = 1; i <= m; i++) ans[id[i + n]] = a[i][0];
      return 2;
    }
    for (int i = 1; i <= m; i++) {
      if (sgn(a[i][e]) < 0) {
        F t = -a[i][0] / a[i][e];
        if (sgn(t - min) < 0 ||
            sgn(t - min) == 0 && (!l || id[i] > id[l])) {
          min = t, l = i;
        }
      }
    }
    if (!l) return 1;
    pivot(l, e);
  }
}
} s;

void solve() {
  int n, m, t;
  std::cin >> n >> m >> t;

  // ANS = max{∑ c_i x_i}
  for (int i = 1; i <= n; i++) {
    D x;
    std::cin >> x;
    s.a[0][i] = x;
  }

  // limits : ∑ a_{i,j} x_j ≤ b_i
  for (int i = 1; i <= m; i++) {
    D x;
    for (int j = 1; j <= n; j++) {
      std::cin >> x;
      s.a[i][j] = x;
    }
    std::cin >> x;
    s.a[i][0] = x;
  }

  int r = s.work(n, m);
  if (!r) {
    std::cout << "Infeasible\n";
  } else if (r == 1) {
    std::cout << "Unbounded\n";
  } else {
    std::cout << (D)s.a[0][0] << "\n";
    if (t) {
      for (int i = 1; i <= n; i++) {
        std::cout << (D)s.ans[i] << " \n"[i == n];
      }
    }
  }
}
```

4.15 Matrix-Tree 定理

4.15.1 无向图生成树个数

```cpp
a[x][x]++, a[y][y]++, a[x][y]--, a[y][x]--;
// 随便划掉第 i 行第 i 列，求行列式
```

A 的 $n-1$ 个非零特征值，则 $t(G)=\frac{1}{n}\lambda_1\lambda_2\cdots\lambda_{n-1}$

4.15.2 有向图外向树形图个数

```cpp
a[i][j]--, a[j][j]++;
// 随便划掉第 r 行第 r 列，求行列式，得到以 r 为根的答案
```

Geometry (5)

5.1 Geo2D

5.1.1 Tricks

有一堆互不相交的凸多边形，建立它们之间的树关系

拆成上下凸壳，在每个多边形最左边的点画一条竖直线，用 set 维护这条线与多边形的交点，找到当前最左点的前驱，如果它是下凸壳，则为它的父亲，否则是它的兄弟。用扫描线，在 set 中加入和删除多边形的边，比较函数中求边与当前扫描线的交点纵坐标，由于交点相对位置不变，因此可以这样维护。

5.1.2 Pt

```cpp
using D = double;
using LD = long double;
using T = D;

constexpr T EPS(1e-10);
const T PI = acosl(-1.L);
int sgn(T x) { return (x > EPS) - (x < -EPS); }
int cmp(T x, T y) { return sgn(x - y); }

struct Pt {
  T x, y;
  void read() {
    int a, b;
    cin >> a >> b, x = a, y = b;
```

```cpp
  }
  void print() { cout << x << " " << y << "\n"; }

  explicit Pt(T x = 0, T y = 0) : x(x), y(y) {}
  Pt operator+(const Pt &r) const { return Pt(x + r.x, y +
  ↪  r.y); }
  Pt operator-(const Pt &r) const { return Pt(x - r.x, y -
  ↪  r.y); }
  Pt operator*(T r) const { return Pt(x * r, y * r); }
  Pt operator/(T r) const { return Pt(x / r, y / r); }
  bool operator<(const Pt &r) const { return x < r.x || x ==
  ↪  r.x && y < r.y; }
  bool operator==(const Pt &r) const { return !cmp(x, r.x) &&
  ↪  !cmp(y, r.y); }

  T operator^(const Pt &r) const { return x * r.x + y * r.y; }
  ↪  // dot
  T operator*(const Pt &r) const { return x * r.y - y * r.x; }
  ↪  // cross
  T cross(Pt a, Pt b) const { return (a - *this) * (b -
  ↪  *this); }

  T len2() const { return x * x + y * y; }
  T len() const { return sqrt(len2()); }

  Pt rotate(T r) const {
    return Pt(x * cos(r) - y * sin(r), x * sin(r) + y *
    ↪  cos(r));
  }

  Pt unit() const { return *this / len(); }
  Pt perp() const { return Pt(-y, x); }
  Pt norm() const { return perp().unit(); }

  Pt proj(Pt a, Pt b) const {
    Pt d = b - a;
    return a + d * ((d ^ (*this - a)) / d.len2());
  }
  Pt refl(Pt a, Pt b) const { return proj(a, b) * 2 - *this; }

  bool onSeg(Pt a, Pt b) const {
    return !sgn(cross(a, b)) && sgn((*this - a) ^ (*this - b))
    ↪  <= 0;
  }
  int side(Pt a, Pt b) const { return sgn(a.cross(b, *this));
  ↪  }
  int pos() const { return y > 0 || y == 0 && x > 0; }
};
bool argcmp(Pt a, Pt b) {
  return a.pos() != b.pos() ? a.pos() : a * b > EPS;
}
```

5.1.3 旋转平移流

对点 r 执行将向量 $\mathbf{P_0P_1}$ 变换到 $\mathbf{Q_0Q_1}$ 的线性变换。

```cpp
Pt linearTransformation(const Pt &p0, const Pt &p1,
                        const Pt &q0, const Pt &q1,
                        const Pt &r) {
  Pt dp = p1 - p0, dq = q1 - q0, num(dp * dq, dp ^ dq);
  return q0 + Pt((r - p0) * num, (r - p0) ^ num) / dp.len2();
}
```

5.1.4 直线与线段

```cpp
int checkLL(Pt a, Pt b, Pt c, Pt d) { // 0, 1, -1(INF)
  return sgn((b - a) * (d - c)) ? 1 : a.side(c, d) ? 0 : -1;
}
Pt getLL(Pt a, Pt b, Pt c, Pt d) {
  T p = c.cross(b, d), q = c.cross(d, a);
  return (a * p + b * q) / (p + q);
}
vector<Pt> getSS(Pt a, Pt b, Pt c, Pt d) { // 0, 1, 2(INF,
↪  return the segment endpoint)
  T oa = c.cross(d, a), ob = c.cross(d, b), oc = a.cross(b,
  ↪  c), od = a.cross(b, d);
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0) {
    return {(a * ob - b * oa) / (ob - oa)};
  }
  vector<Pt> r;
  if (a.onSeg(c, d)) r.push_back(a);
  if (b.onSeg(c, d)) r.push_back(b);
  if (c.onSeg(a, b)) r.push_back(c);
  if (d.onSeg(a, b)) r.push_back(d);
  return r;
}
T distPL(Pt p, Pt a, Pt b) { return fabs(p.cross(a, b)) / (b -
↪  a).len(); }
T distPS(Pt p, Pt a, Pt b) {
  if (a == b) return (a - p).len();
  T d = (b - a).len2(), t = min(d, max(0.0, (b - a) ^ (p -
  ↪  a)));
  return ((p - a) * d - (b - a) * t).len() / d;
}
T distSS(Pt a, Pt b, Pt c, Pt d) {
  return getSS(a, b, c, d).empty() ? min({
      distPS(a, c, d), distPS(b, c, d), distPS(c, a, b),
      ↪  distPS(d, a, b)
    }) : 0.0;
}
```

5.1.5 Polygon

```cpp
using Poly = std::vector<Pt>;
// polygon area: O(n)
T area(const Poly &p) {
  if (p.empty()) return 0;
  T a = p.back() * p[0];
  for (int i = 1; i < p.size(); i++)
    a += p[i - 1] * p[i];
  return a / 2;
}

// point inside polygon: O(n)
int inPoly(Pt a, const Poly &p, int strict = 1) {
  int t = 0, n = p.size();
  for (int i = 0; i < n; ++i) {
    Pt q = p[(i + 1) % n];
    if (a.onSeg(p[i], q)) return !strict;
    t ^= ((a.y < p[i].y) - (a.y < q.y)) * a.cross(p[i], q) >
    ↪  0;
  }
  return t;
}

// axes of polygon: O(n)
vector<pair<Pt, Pt>> polygonAxes(Poly a) {
  int n = a.size(), m = 3 * n;
  a.push_back(a[0]), a.push_back(a[1]);
  vector<pair<T, T>> s(n * 2);
  for (int i = 1, j = 0; i <= n; ++i, j += 2)
    s[j] = {(a[i] - a[i - 1]).len2(), 0},
    s[j + 1] = {a[i].cross(a[i - 1], a[i + 1]),
                (a[i + 1] - a[i]) ^ (a[i - 1] - a[i])};
  s.insert(s.end(), s.begin(), s.begin() + n);
  vector<int> f(m), res;
  for (int r = 0, p = 0, i = 0; i < m; i++) {
    f[i] = r > i ? min(r - i, f[p * 2 - i]) : 1;
    while (i >= f[i] && i + f[i] < m && s[i - f[i]] == s[i +
    ↪  f[i]]) ++f[i];
    if (i + f[i] > r) r = i + f[i], p = i;
    if (f[i] > n) res.push_back(i);
  }
  auto get = [&](int i) {
    int x = (i + 1) / 2;
    return i & 1 ? a[x] : (a[x] + a[x + 1]) / 2;
  };
  vector<pair<Pt, Pt>> axe;
  for (auto i : res) axe.emplace_back(get(i), get(i - n));
  return axe;
}
```

5.1.6 凸包

```cpp
// point inside convex hull: O(log n)
int inHull(Pt p, const Poly &h, int strict = 1) {
  if (h.empty()) return 0;
  int a = 1, b = h.size() - 1, c, r = !strict;
  if (b < 2) return r && p.onSeg(h[0], h[b]);
  if (h[b].side(h[0], h[a]) > 0) swap(a, b);
  if (p.side(h[0], h[a]) >= r || p.side(h[0], h[b]) <= -r) return 0;
  while (abs(a - b) > 1) c = (a + b) >> 1, (p.side(h[0], h[c]) > 0 ? b : a) = c;
  return p.side(h[a], h[b]) < r;
}

// convex hull: O(n log n)
Poly convexHull(Poly p) {
  if (p.size() <= 1) return p;
  sort(p.begin(), p.end());
  Poly h(p.size() + 1);
  int s = 0, t = 0, i = 2;
  for (; i--; s = --t, reverse(p.begin(), p.end()))
    for (auto a : p) {
      while (t >= s + 2 && h[t - 2].cross(h[t - 1], a) <= 0) t--;
      h[t++] = a;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}

// mincowsky sum: O(n + m)
Poly mincowskySum(Poly a, Poly b) {
  int n = a.size(), m = b.size();
  Poly s(n + m);
  Pt a0 = a[0], b0 = b[0];
  for (int i = 0; i < n; i++)
    a[i] = i + 1 < n ? a[i + 1] - a[i] : a0 - a[i];
  for (int i = 0; i < m; i++)
    b[i] = i + 1 < m ? b[i + 1] - b[i] : b0 - b[i];
  s[0] = a0 + b0;
  for (int i = 0, j = 0, k = 1; k < n + m; k++) {
    if (j >= m || i < n && a[i] * b[j] > 0) {
      s[k] = a[i++];
    } else {
      s[k] = b[j++];
    }
    s[k] = s[k] + s[k - 1];
  }
  return s;
}

// 凸多边形关于某一方向的极点
// 复杂度 O(log n)
// 参考资料：https://codeforces.com/blog/entry/48868
template <class F>
int extreme(const Poly &p, const F &dir) {
  auto check = [&](int i) {
    return dir(p[i]) * (p[(i + 1) % p.size()] - p[i]) >= 0;
  };
  auto dir0 = dir(p[0]);
  auto c0 = check(0);
  if (!c0 && check(p.size() - 1)) return 0;
  auto cmp = [&](const Pt &v) {
    int i = &v - p.data();
    if (!i) return 1;
    auto c = check(i);
    auto t = dir0 * (v - p[0]);
    if (i == 1 && c == c0 && !t) return 1;
    return c ^ (c == c0 && t <= 0);
  };
  return std::partition_point(p.begin(), p.end(), cmp) - p.begin();
}

// 过凸多边形外一点求凸多边形的切线，返回切点下标，必须保证点在多边形外
// 复杂度 O(log n)
PII getTgHP(const Poly &p, const Pt &a) {
  int i = extreme(p, [&](const Pt &v) { return v - a; });
  int j = extreme(p, [&](const Pt &v) { return a - v; });
  return {i, j};
}

// 求平行于给定直线的凸多边形的切线，返回切点下标
// 复杂度 O(log n)
PII getTgHL(const Poly &p, const Pt &v) {
  int i = extreme(p, [&](...) { return v; });
  int j = extreme(p, [&](...) { return -v; });
  return {i, j};
}
```

5.1.7 半平面交

一般都是 double，以下代码未经大量测试，抄的 jiangly。

```cpp
struct Line {
  Pt p, v;
  bool operator<(const Line &r) const {
    int k = v.pos() - r.v.pos();
    return k < 0 || k == 0 && v * r.v > 0;
  }
};

Pt getLL(Line a, Line b) {
  return getLL(a.p, a.p + a.v, b.p, b.p + b.v);
}
int side(Pt p, Line l) {
  return sgn(l.v * (p - l.p));
}
std::vector<Pt> hp(std::vector<Line> a) {
  if (a.empty()) return {};
  std::sort(a.begin(), a.end());
  int n = a.size(), l, r;
  std::vector<Pt> p(n);
  std::vector<Line> q(n);
  q[l = r = 0] = a[0];
  for (int i = 1; i < n; i++) {
    while (l < r && side(p[r - 1], a[i]) <= 0) r--;
    while (l < r && side(p[l], a[i]) <= 0) l++;
    if (!sgn(a[i].v * q[r].v)) {
      if (sgn(a[i].v ^ q[r].v) > 0) {
        if (side(a[i].p, q[r]) > 0) {
          assert(l == r);
          q[r] = a[i];
        }
        continue;
      }
      return {};
    }
    q[++r] = a[i];
    if (l < r) p[r - 1] = getLL(q[r], q[r - 1]);
  }
  while (l < r && side(p[r - 1], q[l]) <= 0) r--;
  if (r - l <= 1) return {};
  p[r] = getLL(q[r], q[l]);
  return {p.begin() + l, p.begin() + r + 1};
}
```

5.1.8 Circles

```cpp
double getRadius(Pt a, Pt b, Pt c) {
  Pt u = c - a, v = b - a;
  double k = u * v * 2;
  return u.len() * v.len() * (b - c).len() / abs(k); // a = 2r sinA
}
Pt getCenter(Pt a, Pt b, Pt c) {
  Pt u = c - a, v = b - a;
  double k = u * v * 2;
  return a + (u * v.len2() - v * u.len2()).perp() / k;
}
```

```cpp
struct Circle : Pt {
  T r;
  Circle() {}
  Circle(T x, T y, T r) : Pt(x, y), r(r) {}
  Circle(Pt o, T r) : Pt(o), r(r) {}
  Circle(Pt a, Pt b, Pt c) : Pt(getCenter(a, b, c)),
  ↪  r(getRadius(a, b, c)) {}
  Circle(vector<Pt> p) {
    int n = p.size();
    assert(n);
    shuffle(p.begin(), p.end(), mt19937(233));
    Pt o = p[0];
    r = 0;
    for (int i = 0; i < n; i++) {
      if (cmp((o - p[i]).len(), r) <= 0) continue;
      o = p[i], r = 0;
      for (int j = 0; j < i; j++) {
        if (cmp((o - p[j]).len(), r) <= 0) continue;
        o = (p[i] + p[j]) / 2;
        r = (o - p[i]).len();
        for (int k = 0; k < j; k++) {
          if (cmp((o - p[k]).len(), r) <= 0) continue;
          o = getCenter(p[i], p[j], p[k]);
          r = (o - p[i]).len();
        }
      }
    }
    x = o.x, y = o.y;
  }

  // 圆的反演。平面任意一点 P，反演点 Q 满足 Q 在射线 OP 上且
  ↪  |OP| · |OQ| = r^2。
  // 设圆上任意一点 A，则 △AOP ∼ △QOA。
  Pt invP(Pt p) {  // d1d2 = r^2
    Pt v = p - *this;
    return *this + v * (r * r / v.len2());
  }
  Circle invC(Circle c) {  // o is not on C
    Pt v = c - *this;
    T z = r * r / (v.len2() - c.r * c.r);
    return {*this + v * z, z * c.r};
  }
  pair<Pt, Pt> invCL(Circle c) {  // o must be on C,
  ↪  unverified
    Pt v = c - *this, p = *this + v * (r * r / (2 * c.r *
    ↪  c.r));
    return {p, p + v.perp()};
  }
  Circle invL(Pt a, Pt b) {  // o is not on L
    Pt p = proj(a, b), v = p - *this;
    T r2 = r * r / 2;
    return {*this + v * (r2 / v.len2()), r2 / v.len()};
  }
};

vector<Pt> getCL(Circle c, Pt a, Pt b) {
  Pt k = c.proj(a, b);
  T d = distPL(c, a, b);
  if (cmp(c.r, d) < 0) return {};
  Pt x = (b - a).unit() * sqrt(fmax(0, c.r * c.r - d * d));
  return {k - x, k + x};
}

// get tangent numbers
// checkInter: checkTgCC % 4 != 0
// corner case: a == b
int checkTgCC(Circle a, Circle b) {
  T d = (a - b).len();
  int u = cmp(d, a.r + b.r);
  return u >= 0 ? u + 3 : cmp(d, abs(a.r - b.r)) + 1;
}
vector<Pt> getCC(Circle a, Circle b) {
  if (!(checkTgCC(a, b) & 3)) return {};
  Pt v = b - a;
  T d2 = v.len2(), k = (a.r * a.r - b.r * b.r + d2) / (2 *
  ↪  d2), x = a.r * a.r / d2 - k * k;
  Pt o = a + v * k, z = v.perp() * sqrt(fmax(0, x));
  return {o - z, o + z};
}
vector<Pt> getTgCP(Circle c, Pt p) {
  Pt v = p - c;
  T r2 = c.r * c.r, d2 = v.len2();
  if (cmp(r2, d2) > 0) return {};
  T x = r2 / d2, y = x * (1 - x);
  Pt o = c + v * x, z = v.perp() * sqrt(fmax(0, y));
  return {o - z, o + z};
}
vector<pair<Pt, Pt>> getTgCC(Circle a, Circle b, int k = 1) {
↪  // k=1/-1: out/in, from kactl
  Pt v = b - a;
  T dr = a.r - b.r * k, d2 = v.len2(), h2 = d2 - dr * dr;
  if (sgn(d2) == 0 || sgn(h2) < 0) return {};
  vector<pair<Pt, Pt>> res;
  for (int t : {-1, 1}) {
    Pt z = (v * dr + v.perp() * sqrt(fmax(0, h2)) * t) / d2;
    res.push_back({a + z * a.r, b + z * (b.r * k)});
  }
  if (sgn(h2) == 0) res.pop_back();
  return res;
}
T getAreaCT(Pt p, Pt q, T r) { // Area OP -> OQ in circle r,
↪  from kactl
  auto arg = [](Pt p, Pt q) { return atan2(p * q, p ^ q); };
  T r2 = r * r / 2;
  Pt d = q - p;
  T a = (d ^ p) / d.len2(), b = (p.len2() - r * r) / d.len2(),
  ↪  det = a * a - b;
  if (sgn(det) <= 0) return arg(p, q) * r2;
  T s = fmax(0, -a - sqrt(fmax(0, det))), t = fmin(1, -a +
  ↪  sqrt(fmax(0, det)));
  if (t < 0 || 1 <= s) return arg(p, q) * r2;
  Pt u = p + d * s, v = p + d * t;
  return arg(p, u) * r2 + u * v / 2 + arg(v, q) * r2;
}
T getAreaCP(Circle c, Poly p) {
  if (p.size() <= 1) return 0;
  T ans = getAreaCT(p.back() - c, p[0] - c, c.r);
  for (int i = 0; i + 1 < p.size(); i++) {
    ans += getAreaCT(p[i] - c, p[i + 1] - c, c.r);
  }
  return ans;
}
T getAreaCC(Circle a, Circle b) {
  Pt v = b - a;
  LD d2 = (LD)v.x * v.x + (LD)v.y * v.y, d = sqrt(d2), r1 =
  ↪  a.r * a.r, r2 = b.r * b.r;
  if (cmp(d, a.r + b.r) >= 0) return 0;
  if (cmp(b.r, d + a.r) >= 0) return PI * r1;
  if (cmp(a.r, d + b.r) >= 0) return PI * r2;
  LD a1 = 2 * acos((d2 + r1 - r2) / (2 * d * a.r));
  LD a2 = 2 * acos((d2 + r2 - r1) / (2 * d * b.r));
  return ((a1 - sin(a1)) * r1 + (a2 - sin(a2)) * r2) / 2;
}
```

5.1.9 合并上凸壳

```cpp
// 用区间 [x, y) 表示 a 的点，用 [x, x] 表示 b 的点，合并凸包。
// 每次考虑加入点 bi，找 a 上的两个切线。
// 注意，这个是上凸壳
struct Interval {
  int l, r;
  int len() const { return r - l; }
};
std::vector<Interval> merge(const Poly &a, const Poly &b) {
  std::vector<Interval> s(b.size() * 2 + 5);
  int n = a.size(), m = b.size(), t = 0;
  for (int i = 0, j = 0; i < n || j < m; j++) {
    if (j == m) {
```

```
      s[t++] = {i, n};
      break;
    }

    int pos = std::lower_bound(a.begin() + i, a.begin() + n,
    ↪  b[j]) - a.begin();
    if (i < pos) {
      s[t++] = {i, pos};
      i = pos;
    }

    auto getLast = [&](Interval p) {
      return p.l == p.r ? b[p.r] : a[p.r - 1];
    };

    Pt p = b[j];
    while (t > 1 || t == 1 && s[0].r - s[0].l > 1) {
      if (t > 1 && sgn(getLast(s[t - 2]).cross((s[t - 1].len()
      ↪  ? a : b)[s[t - 1].l] ,p)) >= 0) {
        t--;
        continue;
      }
      if (s[t - 1].len()) {
        int l = s[t - 1].l, r = s[t - 1].r;
        while (r - l > 1) {
          int m = l + r >> 1;
          (sgn(a[m - 1].cross(a[m], p)) >= 0 ? r : l) = m;
        }
        s[t - 1].r = r;
      }
      break;
    }

    if (t && i < n && sgn(getLast(s[t - 1]).cross(b[j], a[i]))
    ↪  >= 0) continue;

    s[t++] = {j, j};

    int l = i, r = n;
    while (r - l > 1) {
      int m = l + r >> 1;
      (sgn(b[j].cross(a[m - 1], a[m])) >= 0 ? l : r) = m;
    }
    i = l;
  }
  s.resize(t);
  return s;
}
```

5.1.10 动态凸包

wxw 写的动态凸包

只能加，就是求切线。

```
struct UpperHull {
  Pt val[N], l[N], r[N];
  ll sum[N];
  int lc[N], rc[N], key[N];
  int root, tot;

  int newNode(Pt x) {
    ++tot;
    lc[tot] = rc[tot] = 0;
    key[tot] = rng();
    l[tot] = r[tot] = val[tot] = x;
    sum[tot] = 0;
    return tot;
  }

  void update(int u) {
    sum[u] = sum[lc[u]] + sum[rc[u]];
    if (lc[u]) {
      sum[u] += cross(val[u], r[lc[u]]);
      l[u] = l[lc[u]];
    } else
      l[u] = val[u];
    if (rc[u]) {
      sum[u] += cross(l[rc[u]], val[u]);
      r[u] = r[rc[u]];
    } else
      r[u] = val[u];
  }

  int merge(int u, int v) {
    if (!u || !v) return u | v;
    if (key[u] > key[v]) {
      rc[u] = merge(rc[u], v);
      update(u);
      return u;
    } else {
      lc[v] = merge(u, lc[v]);
      update(v);
      return v;
    }
  }

  // < x
  void split(int u, Pt x, int &l, int &r) {
    if (!u) {
      l = r = 0;
      return;
    }
    if (val[u] < x) {
      l = u;
      split(rc[u], x, rc[l], r);
      update(l);
    } else {
      r = u;
      split(lc[u], x, l, lc[r]);
      update(r);
    }
  }

  Pt res;
  void findl(int u, Pt x, Pt y) {
    if (!u) return;
    if (lc[u]) {
      if (cross(val[u] - r[lc[u]], x - r[lc[u]]) >= 0) {
        res = val[u];
        findl(lc[u], x, y);
      } else {
        findl(rc[u], x, val[u]);
      }
    } else {
      if (y.x != INF && cross(val[u] - y, x - y) >= 0) {
        res = val[u];
        return;
      }
      findl(rc[u], x, val[u]);
    }
  }

  void findr(int u, Pt x, Pt y) {
    if (!u) return;
    if (rc[u]) {
      if (cross(val[u] - x, l[rc[u]] - x) >= 0) {
        res = val[u];
        findr(rc[u], x, y);
      } else {
        findr(lc[u], x, val[u]);
      }
    } else {
      if (y.x != INF && cross(val[u] - x, y - x) >= 0) {
        res = val[u];
        return;
      }
      findr(lc[u], x, val[u]);
    }
```

```
}

int st[100], top;

void insert(Pt x, int op = 0) {
  int a, b;
  split(root, x, a, b);
  if (b && l[b] == x) {
    root = merge(a, b);
    return;
  }
  if (a && b && cross(x - r[a], l[b] - r[a]) >= 0) {
    root = merge(a, b);
    return;
  }
  int now = newNode(x);
  res = Pt(INF);
  findl(a, x, Pt(INF));
  if (res.x != INF) {
    int c, d;
    split(a, res, c, d);
    if (op) st[++top] = d;
    a = c;
  }
  res = Pt(INF);
  findr(b, x, Pt(INF));
  if (res.x != INF) {
    int c, d;
    split(b, res + Pt(0, 1), c, d);
    if (op) st[++top] = c;
    b = d;
  }
  root = merge(merge(a, now), b);
  if (op) st[++top] = -now;
}

void roolback() {
  while (top) {
    if (st[top] < 0) {
      int a, b, c;
      split(root, val[-st[top]], a, b);
      split(b, val[-st[top]] + Pt(0, 1), b, c);
      root = merge(a, c);
    } else {
      int a, b;
      split(root, val[st[top]], a, b);
      root = merge(merge(a, st[top]), b);
    }
    --top;
  }
}

ll getarea() {
  if (!root) return 0;
  return abs(sum[root] + cross(l[root], r[root]));
}

void ini() { root = tot = top = 0; }
} H1, H2;
```

带删除

能加能删，$O(n\log^2 n)$，可以求两个凸包的桥（$O(\log n)$ 合并两不相交凸包）

没办法实时维护面积。

```
struct Pt {
  LL x, y;
  bool operator<(const Pt &r) const {
    return x < r.x || x == r.x && y < r.y;
  }
  // ...
};

#define ls o << 1
#define rs o << 1 | 1

struct UpperHull {
  struct Node {
    int l, r, bl, br;
  };

  int n, s;
  std::vector<Pt> ps;
  std::vector<Node> t;
  std::vector<int> in, id;

  UpperHull(const std::vector<Pt> &a)
    : n(a.size()), s(1 << std::__lg(n * 2 - 1)),
      ps(a), t(s * 2, {-1, -1, -1, -1}), in(n), id(n) {
    std::vector<std::pair<Pt, int>> pi(n);
    for (int i = 0; i < n; i++) {
      pi[i] = {ps[i], i};
    }
    std::sort(pi.begin(), pi.end());
    for (int i = 0; i < n; i++) {
      ps[i] = pi[i].first;
      in[pi[i].second] = i;
      id[i] = pi[i].second;
    }
    for (int i = 0; i < n; i++) {
      t[i + s] = {i, i + 1, i, i};
    }
    for (int i = s - 1; i; i--) {
      pushup(i);
    }
  }

  bool ok(int o) { return t[o].r != -1; }
  bool wk(int &o) {
    if (!ok(ls)) return o = rs, true;
    if (!ok(rs)) return o = ls, true;
    return false;
  }
  void pushup(int o) {
    if (!ok(ls) && !ok(rs)) {
      t[o].r = -1;
    } else if (!ok(ls)) {
      t[o] = t[rs];
    } else if (!ok(rs)) {
      t[o] = t[ls];
    } else {
      int p = ls, q = rs;
      LL x = ps[t[q].l].x;
      while (p < s || q < s) {
        while (p < s && wk(p)) ;
        while (q < s && wk(q)) ;
        if (p >= s && q >= s) break;

        int a = t[p].bl, b = t[p].br, c = t[q].bl, d =
        ↪  t[q].br;
        if (a != b && ps[a].cross(ps[b], ps[c]) > 0) {
          p <<= 1;
        } else if (c != d && ps[b].cross(ps[c], ps[d]) > 0) {
          q = q << 1 | 1;
        } else if (a == b) {
          q <<= 1;
        } else if (c == d) {
          p = p << 1 | 1;
        } else {
          LLL c1 = ps[a].cross(ps[b], ps[c]), c2 =
          ↪  ps[b].cross(ps[a], ps[d]);
          if (c1 + c2 == 0 || c1 * ps[d].x + c2 * ps[c].x <
          ↪  (c1 + c2) * x) {
            p = p << 1 | 1;
          } else {
            q <<= 1;
```

```
          }
        }
      }
      t[o] = {t[ls].l, t[rs].r, t[p].l, t[q].l};
    }
  }
  void up(int i) {
    while (i > 1) pushup(i >>= 1);
  }
  void add(int i) {
    i = in[i];
    t[i + s] = {i, i + 1, i, i};
    up(i + s);
  }
  void del(int i) {
    i = in[i];
    t[i + s] = {-1, -1, -1, -1};
    up(i + s);
  }

  std::vector<int> res;
  void dfs(int o, int l, int r) {
    if (!ok(o) || l >= r) return;
    while (o < s && wk(o)) ;
    if (o >= s) {
      res.push_back({t[o].l});
      return;
    }
    if (r <= t[o].bl + 1) return dfs(ls, l, r);
    if (t[o].br <= l) return dfs(rs, l, r);
    dfs(ls, l, t[o].bl + 1);
    dfs(rs, t[o].br, r);
  }
  std::vector<int> get() {
    res.clear();
    dfs(1, 0, n);
    for (int &i : res) i = id[i];
    return res;
  }
};
```

5.1.11 平面最近点对

```
double closestPair(vector<Pt> p) {
  if (p.size() <= 1) return 0;
  sort(p.begin(), p.end(), [&](Pt i, Pt j) {
    return i.y < j.y;
  });
  set<Pt> s;
  double ans = 1e18;
  int j = 0;
  for (Pt x : p) {
    Pt d {ans, 0};
    while (cmp(p[j].y, x.y - ans) < 0) s.erase(p[j++]);
    auto l = s.lower_bound(x - d), r = s.upper_bound(x + d);
    for (; l != r; l++) ans = min(ans, (*l - x).len());
    s.insert(x);
  }
  return ans;
}
```

Graphs (6)

6.1 Trees

6.1.1 树上路径交并

两个树上路径求交有个固定算法：路径 A 的两个端点和路径 B 的两个端点两两求 LCA，然后取四个中深度最大的那两个点，它们之间就是交路径。当这两个点相同，并且深度小于两个路径之一时，则说明不存在交。

【ZJOI2019 语言】当若干路径的交不为空时，将路径端点按 DFS 序排序，相邻、首尾点相连，此时他们的并上的每条边会恰好被经过 2 次。

6.1.2 直径可并性

当边权非负时，树上点集 $A \cup B$ 直径的两个端点一定是 A, B 各自直径的端点之二。

6.1.3 动态直径 - 欧拉序

$D(a,b) = d(a) + d(b) - 2d(LCA(a,b))$

LCA 可以用欧拉序转换为 RMQ 问题，然后问题就变成维护 $\max_{x \le z \le y} a_x + a_y - 2a_z$，可以用线段树维护。

修改比较简单时，可以使用简化欧拉序，减小常数。即 `euler[in[x] - 1] = fa[x]`，此时相当于维护两个序列 `a[in[x]] = d[x], b[in[x] - 1] = d[fa[x]]`，$\max_{x \le z < y} a_x + a_y - 2b_z$，同样用线段树维护。

6.1.4 虚树

对于两次排序的简易写法，a_i 在虚树上的父亲为 $\mathrm{lca}(a_i, a_{i-1})$，有时候比较方便。下面为排序加栈的写法

```
std::sort(a.begin(), a.end(), [&](int i, int j){ return in[i]
↪  < in[j]; });
int t = 0;
auto pop = [&] { g[s[t - 1]].push_back(s[t]), t--; };
for (int x : a) {
  if (t && out[x] > out[s[t]]) {
    int p = lca(s[t], x);
    while (t > 1 && in[s[t - 1]] >= in[p]) pop();
    if (s[t] != p) g[p].push_back(s[t]), s[t] = p;
  }
  s[++t] = x;
}
while (t > 1) pop();
```

6.2 Bipartite Graph

6.2.1 最大匹配

最小点覆盖 = 最大匹配，去掉最小点覆盖剩下的点是最大独立集。

偏序集最长反链：拆点二分图的最大独立集 I，左右部均属于 I 的点。

```
struct Bipartite {
  std::vector<int> d, g, l, r;
  int match;
  Bipartite(int n, int m, const std::vector<PII> &e)
      : d(n + 1), g(e.size()), l(n, -1), r(m, -1), match(0) {
    std::vector<int> a, p, q(n);
    for (auto &[x, y] : e) d[x]++;
    for (int i = 1; i <= n; i++) d[i] += d[i - 1];
    for (auto &[x, y] : e) g[--d[x]] = y;

    for (;;) {
      a.assign(n, -1), p.assign(n, -1);
      int t = 0;
      for (int i = 0; i < n; i++)
        if (l[i] == -1) q[t++] = a[i] = p[i] = i;
      bool found = false;
      for (int i = 0; i < t; i++) {
        int x = q[i];
        if (~l[a[x]]) continue;
        for (int j = d[x]; j < d[x + 1]; j++) {
          int y = g[j];
          if (r[y] == -1) {
            while (~y) r[y] = x, std::swap(l[x], y), x = p[x];
            found = true, match++;
            break;
          }
          if (p[r[y]] == -1)
            q[t++] = y = r[y], p[y] = x, a[y] = a[x];
        }
      }
      if (!found) break;
    }
  }

  std::vector<int> minCover() {
```

```cpp
    int n = l.size(), m = r.size();
    std::vector<bool> vl(n, true), vr(m);
    for (int i : r) if (~i) vl[i] = false;
    std::vector<int> q, res;
    for (int i = 0; i < n; i++) if (vl[i]) q.push_back(i);
    while (!q.empty()) {
      int x = q.back(); q.pop_back();
      vl[x] = true;
      for (int i = d[x]; i < d[x + 1]; i++) {
        int y = g[i];
        if (!vr[y] && ~r[y]) vr[y] = true, q.push_back(r[y]);
      }
    }
    for (int i = 0; i < n; i++) if (!vl[i]) res.push_back(i);
    for (int i = 0; i < m; i++) if (vr[i]) res.push_back(i + n);

    assert(res.size() == match);
    return res;
  }
};
```

6.2.2 最大权匹配

costs 表示匹配数为 0, 1, 2, . . . 时的最大权值之和，l 表示 x->y，r 表示 y->x

```cpp
template <class T>
struct MaxAssignment {
  static const T INF = std::numeric_limits<T>::max();
  std::vector<T> costs;
  std::vector<int> l, r;

  MaxAssignment(int n, int m, std::vector<std::vector<T>> a)
    : costs(n + 1), l(n, -1), r(m, -1) {
    assert(0 <= n && n <= m);
    assert(int(a.size()) == n);
    for (int i = 0; i < n; i++) {
      assert(int(a[i].size()) == m);
      for (auto x : a[i]) assert(x >= 0);
    }

    std::vector<T> u(n, INF), v(m);
    std::vector<int> f(m);

    for (int cur = 0; cur < n; cur++) {
      std::vector<int> vx(n), vy(m), p(n, -1), q;
      std::vector<T> d(m, INF);

      auto update = [&](int x) {
        for (int y = 0; y < m; y++)
          if (smin(d[y], u[x] + v[y] - a[x][y])) f[y] = x;
      };

      for (int x = 0; x < n; x++)
        if (l[x] == -1) q.push_back(x), vx[x] = 1, update(x);

      int ex, ey, ql = 0;
      for (;;) {
        while (ql < q.size()) {
          int x = q[ql++];
          for (int y = 0; y < m; y++) {
            if (a[x][y] == u[x] + v[y] && !vy[y]) {
              if (r[y] == -1) {
                ex = x, ey = y; goto found;
              }
              q.push_back(r[y]), p[r[y]] = x;
              vy[y] = vx[r[y]] = 1, update(r[y]);
            }
          }
        }

        T w = INF;
        for (int y = 0; y < m; y++)
          if (!vy[y]) smin(w, d[y]);
        for (int x = 0; x < n; x++)
          if (vx[x]) u[x] -= w;
        for (int y = 0; y < m; y++)
          vy[y] ? v[y] += w : d[y] -= w;
        for (int y = 0; y < m; y++) {
          if (!vy[y] && d[y] == 0) {
            if (r[y] == -1) {
              ex = f[y], ey = y; goto found;
            }
            q.push_back(r[y]), p[r[y]] = f[y];
            vy[y] = vx[r[y]] = 1, update(r[y]);
          }
        }
      }

      found:
      costs[cur + 1] = costs[cur];
      for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
        costs[cur + 1] += a[x][y];
        if (l[x] != -1) costs[cur + 1] -= a[x][l[x]];
        ty = l[x], l[x] = y, r[y] = x;
      }
    }
  }
};
```

6.3 Max Flow

6.3.1 HLPP

```cpp
template <class T>
struct MaxFlow {
  std::vector<std::tuple<int, int, T, T>> es;

  void add(int x, int y, T z, T r = 0) {
    es.push_back({x, y, z, r});
  }

  struct Edge {
    int v, r; T w;
  };

  T flow(int n, int s, int t) {
    std::vector<Edge> e(es.size() * 2);
    std::vector<int> a(n + 1), h(n), c(2 * n);
    std::vector<std::vector<int>> hs(2 * n);
    std::vector<T> ec(n);
    // bool leftOfMinCut(int x) { return h[x] >= n; }
    for (auto &[x, y, z, r] : es) a[x]++, a[y]++;
    for (int i = 1; i <= n; i++) a[i] += a[i - 1];
    for (auto &[x, y, z, r] : es) {
      e[--a[x]] = {y, --a[y], z}, e[a[y]] = {x, a[x], r};
    }

    auto up = [&](int i, T f) {
      int v = e[i].v, j = e[i].r;
      if (!ec[v] && f) hs[h[v]].push_back(v);
      e[i].w -= f, e[j].w += f;
      ec[v] += f, ec[e[j].v] -= f;
    };

    auto cur = a;
    h[s] = n, ec[t] = 1, c[0] = n - 1;
    for (int i = a[s]; i < a[s + 1]; i++) up(i, e[i].w);
    int k = 0;
    for (;;) {
      while (hs[k].empty()) if (!k--) return -ec[s];
      int x = hs[k].back();
      hs[k].pop_back();
      while (ec[x] > 0) {
        int &o = cur[x];
        if (o >= a[x + 1]) {
          h[x] = 1e9;
          for (int i = a[x]; i < a[x + 1]; i++)
```

```
          if (e[i].w && smin(h[x], h[e[i].v] + 1))
            o = i;
        if (c[h[x]]++, !--c[k] && k < n)
          for (int i = 0; i < n; i++)
            if (k < h[i] && h[i] < n)
              c[h[i]]--, h[i] = n + 1;
        k = h[x];
      } else if (e[o].w && h[x] == h[e[o].v] + 1) {
        up(o, std::min(ec[x], e[o].w));
      } else {
        o++;
      }
    }
  }
}
};
```

6.3.2 Dinic

```
template <class T>
struct Dinic {
  struct Edge {
    int v, next;
    T w;
  } e[M];
  int s, t, head[N], cur[N], tot, q[N], d[N], l, r;
  Dinic() { memset(head, -1, sizeof head); }
  void add(int x, int y, T z) {
    e[tot] = {y, head[x], z}, head[x] = tot++;
    e[tot] = {x, head[y], (T)0}, head[y] = tot++;
  }
  T dfs(int x, T flow) {
    if (x == t || !flow) return flow;
    T used = 0;
    for (int &i = cur[x]; ~i; i = e[i].next) {
      int y = e[i].v;
      if (!e[i].w || d[y] != d[x] - 1) continue;
      T f = dfs(y, std::min(flow - used, e[i].w));
      used += f, e[i].w -= f, e[i ^ 1].w += f;
      if (flow == used) return flow;
    }
    if (!used) d[x] = -1;
    return used;
  }
  T flow(int s_, int t_, T lim =
↪   std::numeric_limits<T>().max()) {
    s = s_, t = t_;
    T ans = 0;
    while (ans < lim) {
      memset(d, -1, sizeof d);
      q[l = r = 0] = t;
      d[t] = 0;
      while (l <= r) {
        int x = q[l++];
        for (int i = head[x]; ~i; i = e[i].next) {
          int y = e[i].v;
          if (e[i ^ 1].w && d[y] == -1) {
            d[y] = d[x] + 1;
            q[++r] = y;
          }
        }
      }
      if (d[s] == -1) break;
      memcpy(cur, head, sizeof head);
      ans += dfs(s, lim - ans);
    }
    return ans;
  }
};
```

使用 dinic 通过加强版的方法：

1. Scaling, 按 Cap 从大到小加边，limit 倍增。
2. 两轮，第一轮不加反向边，第二轮加反向边

```
for(int tp:{0,1})for(int p=1e9,i=0;;p/=20){
  for(;i<m&&d[i].w>=p;++i)
    if(tp)v[d[i].v]+=i*2+1;
    else ins(d[i].u,d[i].v,d[i].w);
  ans+=dinic();if(!p)break;
}
```

6.4 上下界网络流

6.4.1 无源汇有上下界可行流

```
n=read(),m=read();tot=1;s=0,t=n+1;
int sum=0;REP(i,1,m){
  int x=read(),y=read(),b=read(),c=read();
  add(x,y,c-b);upp[i]=c;fl[y]+=b;fl[x]-=b;
}
REP(i,1,n)if(fl[i])if(fl[i]>0)add(s,i,fl[i]);
else add(i,t,-fl[i]),sum-=fl[i];
int ans=0;while(bfs()){
  REP(i,0,t)cur[i]=head[i];
  ans+=dfs(s,inf);
}
if(ans==sum){
  puts("YES");
  REP(i,1,m)printf("%d\n",upp[i]-e[i<<1].w);
}else puts("NO");
```

6.4.2 有源汇有上下界最大流

```
int S,T;
n=read(),m=read(),S=read(),T=read();tot=1;
int sum=0;REP(i,1,m){
  int x=read(),y=read(),b=read(),c=read();
  add(x,y,c-b);fl[y]+=b;fl[x]-=b;
}
add(T,S,inf);
REP(i,1,n)if(fl[i])if(fl[i]>0)add(0,i,fl[i]);
else add(i,n+1,-fl[i]),sum-=fl[i];
s=0,t=n+1;
if(dinic()==sum){
  s=S,t=T,printf("%d\n",dinic());
}else puts("please go home to sleep");
```

6.4.3 有源汇有上下界最小流

```
int S,T;
n=read(),m=read(),S=read(),T=read();tot=1;
int sum=0;REP(i,1,m){
  int x=read(),y=read(),b=read(),c=read();
  add(x,y,c-b);fl[y]+=b;fl[x]-=b;
}
REP(i,1,n)if(fl[i])if(fl[i]>0)add(0,i,fl[i]);
else add(i,n+1,-fl[i]),sum-=fl[i];
s=0,t=n+1;sum-=dinic();add(T,S,inf);sum-=dinic();
if(!sum)printf("%d\n",e[tot].w);
else puts("please go home to sleep");
```

6.5 MCMF

费用关于流量是凸函数：斜率递增的直线。

如果边权很小，可以跑最大流进行增广

```
template <class T, class C>
struct MCMF {
  struct Edge {
    int v;
    T w; C c;
  };
  std::vector<Edge> e;
  std::vector<std::vector<int>> g;
  MCMF(int n) : e(), g(n) {}

  int add(int x, int y, T z, C c) {
    int m = e.size();
    e.push_back({y, z, c});
```

```
  e.push_back({x, 0, -c});
  g[x].push_back(m), g[y].push_back(m ^ 1);
  return m;
}
std::pair<T, C> flow(int s, int t, T lim =
↪  std::numeric_limits<T>::max()) {
  int n = g.size();

  assert(0 <= s && s < n);
  assert(0 <= t && t < n);
  assert(s != t);

  std::vector<C> dual(n), d;
  std::vector<int> p(n), vis;

  auto dual_ref = [&]() -> bool {
    std::priority_queue<std::pair<C, int>> q;
    d.assign(n, std::numeric_limits<C>::max());
    vis.assign(n, 0);
    d[s] = 0, q.push({0, s});
    while (!q.empty()) {
      int x = q.top().second;
      q.pop();
      if (vis[x]) continue;
      vis[x] = 1;
      if (x == t) break;
      for (int i : g[x]) {
        int y = e[i].v;
        if (vis[y] || !e[i].w) continue;
        C c = e[i].c - dual[y] + dual[x];
        if (d[y] - d[x] > c) {
          d[y] = d[x] + c;
          q.push({-d[y], y});
          p[y] = i;
        }
      }
    }
    if (!vis[t]) return false;
    for (int i = 0; i < n; i++) {
      if (vis[i]) dual[i] -= d[t] - d[i];
    }
    return true;
  };

  T flow = 0;
  C cost = 0;
  while (flow < lim && dual_ref()) {
    T f = lim - flow;
    for (int x = t; x != s; x = e[p[x] ^ 1].v) {
      smin(f, e[p[x]].w);
    }
    for (int x = t; x != s; x = e[p[x] ^ 1].v) {
      e[p[x]].w -= f, e[p[x] ^ 1].w += f;
    }
    flow += f, cost -= f * dual[s];
  }
  return {flow, cost};
 }
};
```

6.6 BCC

6.6.1 e-BCC

考虑了重边的问题

```
void gen(int x) {
  bcc++;
  do {
    bel[s[t]] = bcc;
  } while (s[t--] != x);
}
void dfs(int x, int p) {
  in[x] = low[x] = ++dfn, s[++t] = x;
  for (int i = head[x]; ~i; i = e[i].next) {
    int y = e[i].v;
    if (!in[y]) {
      dfs(y, i);
      smin(low[x], low[y]);
      if (low[y] > in[x]) gen(y);
    } else if (i ^ p ^ 1) {
      smin(low[x], in[y]);
    }
  }
  if (p == -1) gen(x);
}
```

6.6.2 v-BCC

（广义）圆方树。h 表示圆方树，要注意是有根的，还要注意一个圆点可能有多个方儿子。

```
std::vector<int> g[N], h[N * 2];
int in[N], low[N], dfn, bcc;
void dfs(int x) {
  low[x] = in[x] = ++dfn, h[0].push_back(x);
  for (int y : g[x]) {
    if (!in[y]) {
      int k = h[0].size();
      dfs(y);
      smin(low[x], low[y]);
      if (low[y] >= in[x]) {
        h[x].push_back(++bcc + n);
        h[bcc + n].assign(h[0].begin() + k, h[0].end());
        h[0].resize(k);
      }
    } else {
      smin(low[x], in[y]);
    }
  }
}
```

需要得到点双中的边，此时需要考虑重边的问题。

```
if (in_edge == e) continue;
if (in[y]) {
  smin(low[x], in[y]);
  if (in[y] < in[x]) st.push_back(e);
} else {
  int si = st.size();
  dfs(y, e);
  smin(low[x], low[y]);
  if (low[y] == in[x]) {
    st.push_back(e);
    // f(vi(st.begin() + si, st.end()));
    st.resize(si);
  } else if (low[y] < in[x]) {
    st.push_back(e);
  } else {
    // e is bridge
  }
}
```

6.7 Two Sat

```
struct TwoSat {
  int n, cur;
  std::vector<std::vector<int>> g;
  std::vector<int> val, up, in, s;

  int addVar() { return g.resize(g.size() + 2), n++; }

  TwoSat(int n = 0)
      : n(n), cur(0), g(n * 2), val(n, -1), up(n * 2), in(n *
      ↪  2), s() {}

  void add(int x, int y) { // x || y
    x = std::max(2 * x, -1 - 2 * x);
    y = std::max(2 * y, -1 - 2 * y);
    g[x ^ 1].push_back(y);
    g[y ^ 1].push_back(x);
  }
```

```cpp
  void atMostOne(const std::vector<int> &a) {
    if (a.size() <= 1) return;
    int x = ~a[0];
    for (int i = 2, v; i < a.size(); i++, x = ~v)
      add(x, ~a[i]), add(x, v = addVar()), add(~a[i], v);
    add(x, ~a[1]);
  }
  void set(int x) { add(x, x); }

  int dfs(int i) {
    int low = up[i] = ++cur, x;
    s.push_back(i);
    for (int j : g[i])
      if (!in[j]) smin(low, up[j] ?: dfs(j));
    cur++;
    if (low == up[i]) {
      do {
        x = s.back(), s.pop_back();
        in[x] = cur;
        if (val[x >> 1] == -1) val[x >> 1] = x & 1 ^ 1;
      } while (x != i);
    }
    return up[i] = low;
  }

  bool solve() {
    for (int i = 0; i < n * 2; i++)
      if (!in[i]) dfs(i);
    for (int i = 0; i < n; i++)
      if (in[i << 1] == in[i << 1 | 1]) return false;
    return true;
  }
};
```

6.8 Dominator Tree

- `tarjan(s)` 求以 s 为起点的支配树。
- `fa` 支配树上父亲，满足 $fa[x] < x, fa[1] = 0$。
- `dom` 支配树上儿子。

注意上面的东西都是以 dfs 序来标号的，所以要求原图中 x 的支配点，应该写 `id[fa[in[x]]]`。

时间复杂度 $O(n \log n)$，其中 log 是并查集。

```cpp
std::vector<int> g[N], h[N], dom[N];
int in[N], id[N], dfn, par[N], f[N], val[N], sem[N], fa[N];

void add(int x, int y) {
  g[x].push_back(y);
  h[y].push_back(x);
}
void dfs(int x) {
  in[x] = ++dfn, id[in[x]] = x;
  for (int y : g[x]) {
    if (in[y]) continue;
    dfs(y);
    par[in[y]] = in[x];
  }
}
int find(int x) {
  if (x == f[x]) return x;
  int y = find(f[x]);
  if (sem[val[f[x]]] < sem[val[x]]) val[x] = val[f[x]];
  return f[x] = y;
}
void tarjan(int s) {
  dfs(s);
  for (int i = 1; i <= dfn; i++) {
    f[i] = sem[i] = val[i] = i;
  }
  for (int y = dfn; y > 1; y--) {
    int x = par[y];
    for (int v : h[id[y]]) {
      int z = in[v];
      if (!z) continue;
      find(z);
      smin(sem[y], sem[val[z]]);
    }
    dom[sem[y]].push_back(y);
    f[y] = x;
    for (int v : dom[x]) {
      find(v);
      fa[v] = sem[val[v]] < x ? val[v] : x;
    }
    dom[x].clear();
  }
  for (int i = 2; i <= dfn; i++) {
    if (fa[i] != sem[i]) fa[i] = fa[fa[i]];
    dom[fa[i]].push_back(i);
  }
  fa[1] = 0;
}
```

6.9 EulerWalk

欧拉路径判定（是否存在）：

有向图欧拉路径：图中恰好存在 1 个点出度比入度多 1（这个点即为起点 S），1 个点入度比出度多 1（这个点即为终点 T），其余节点出度 = 入度。

有向图欧拉回路：所有点的入度 = 出度（起点 S 和终点 T 可以为任意点）。

无向图欧拉路径：图中恰好存在 2 个点的度数是奇数，其余节点的度数为偶数，这两个度数为奇数的点即为欧拉路径的起点 S 和终点 T。

无向图欧拉回路：所有点的度数都是偶数（起点 S 和终点 T 可以为任意点）。

字典序最小：将 g 从小到大排序即可。

```cpp
std::vector<PII> g[N]; // (to, edge_id)
int n, m;
std::vector<int> eulerWalk(int src = 0) {
  std::vector<int> d(n), cur(n), v(m), r, s = {src};
  d[src]++;  // to allow Euler paths , not just cycles
  while (!s.empty()) {
    int x = s.back(), &i = cur[x];
    if (i == g[x].size()) {
      r.push_back(x);
      s.pop_back();
      continue;
    }
    auto [y, e] = g[x][i++];
    if (!v[e]) {
      d[x]--, d[y]++;
      v[e] = 1;
      s.push_back(y);
    }
  }
  if (r.size() != m + 1) return {};
  for (int x : d) if (x < 0) return {};
  return {r.rbegin(), r.rend()};
}
```

6.10 General Matching

```cpp
struct Graph {
  int n;
  std::vector<std::vector<int>> g;
  Graph(int n) : n(n), g(n) {}
  void add(int x, int y) { g[x].push_back(y),
  ↪  g[y].push_back(x); }
  std::vector<int> solve() {
    std::vector<int> ans(n, -1), vis(n), fa(n), f(n), dep(n),
    ↪  q;
    auto find = [&](int x) {
      while (f[x] != x) x = f[x] = f[f[x]];
      return x;
    };
    auto lca = [&](int x, int y) {
      x = find(x), y = find(y);
      while (x != y) {
        if (dep[x] < dep[y]) std::swap(x, y);
```

```cpp
        x = find(fa[ans[x]]);
      }
      return x;
    };
    auto blossom = [&](int x, int y, int p) {
      while (find(x) != p) {
        fa[x] = y, y = ans[x];
        if (!vis[y]) vis[y] = 1, q.push_back(y);
        f[x] = f[y] = p, x = fa[y];
      }
    };
    auto augment = [&](int x) {
      std::iota(f.begin(), f.end(), 0);
      std::fill(vis.begin(), vis.end(), -1);
      q = {x}, vis[x] = 1, dep[x] = 0;
      for (int i = 0; i < q.size(); i++) {
        int x = q[i];
        for (int y : g[x]) {
          if (vis[y] == -1) {
            vis[y] = 0, fa[y] = x, dep[y] = dep[x] + 1;
            if (ans[y] == -1) {
              for (int t; x != -1; y = t, x = y == -1 ? -1 :
              ↪  fa[y])
                t = ans[x], ans[y] = x, ans[x] = y;
              return;
            }
            vis[ans[y]] = 1, dep[ans[y]] = dep[x] + 2;
            q.push_back(ans[y]);
          } else if (vis[y] == 1 && find(y) != find(x)) {
            int p = lca(x, y);
            blossom(x, y, p), blossom(y, x, p);
          }
        }
      }
    };
    for (int i = 0; i < n; i++) {
      if (ans[i] != -1) continue;
      for (int j : g[i]) {
        if (ans[j] == -1) {
          ans[i] = j, ans[j] = i;
          break;
        }
      }
    }
    for (int u = 0; u < n; ++u)
      if (ans[u] == -1) augment(u);
    return ans;
  }
};
```

6.11 最小割树

图上面两点间的最小割等于树上两点间路径边权的最小值

```cpp
Graph<int> g(n);

while (m--) {
  int x, y, z;
  std::cin >> x >> y >> z;
  g.add(x, y, z, true); // double_edged
}

std::vector<int> f(n);
f[0] = -1;
for (int i = 1; i < n; i++) {
  // 退流
  for (int j = 0; j < g.e.size(); j += 2) {
    g.e[j].w = g.e[j ^ 1].w = g.e[j].w + g.e[j ^ 1].w >> 1;
  }

  int k = g.flow(i, f[i]);
  /* edge (i, f[i], k) */

  for (int j = i + 1; j < n; j++) {
    // 注意：判断 j 和 i 联通，板子里 bfs 是从 t 向 s 跑的，所以判 -1
    if (g.d[j] == -1 && f[i] == f[j]) {
      f[j] = i;
    }
  }
}
```

6.12 K Shortest Path

```cpp
using T = LL;
struct Edge {
  int x, y; T z;
};
struct Heap {
  struct Node {
    int ls, rs, h, v;
    T w;
  } t[N * 40];
  int cnt;
  int newNode(int v, T w) {
    t[++cnt] = {0, 0, 1, v, w};
    return cnt;
  }
  int merge(int x, int y) {
    if (!x) return y;
    if (!y) return x;
    if (t[x].w > t[y].w) std::swap(x, y);
    t[++cnt] = t[x], x = cnt;
    t[x].rs = merge(t[x].rs, y);
    if (t[t[x].ls].h < t[t[x].rs].h) std::swap(t[x].ls,
    ↪  t[x].rs);
    t[x].h = t[t[x].rs].h + 1;
    return x;
  }
} h;

std::vector<T> kShortestPath(int n, int k, int s, int t, const
↪  std::vector<Edge> &e) {
  int m = e.size();
  std::vector<int> deg(n + 1), g(m);
  for (auto &[x, y, z] : e) deg[y]++;
  for (int i = 1; i <= n; i++) deg[i] += deg[i - 1];
  for (int i = 0; i < m; i++) g[--deg[e[i].y]] = i;

  std::vector<T> d(n, -1);
  std::vector<int> fa(n, -1), p;

  using Q = std::pair<T, int>;
  std::priority_queue<Q, std::vector<Q>, std::greater<Q>> q;

  p.reserve(n);
  d[t] = 0, q.push({0, t});

  std::vector<bool> vis(n);
  while (!q.empty()) {
    int x = q.top().second;
    q.pop();
    if (vis[x]) continue;
    vis[x] = true;
    p.push_back(x);
    for (int i = deg[x]; i < deg[x + 1]; i++) {
      auto &[y, _, z] = e[g[i]];
      if (d[y] == -1 || d[y] > d[x] + z) {
        d[y] = d[x] + z, fa[y] = g[i];
        q.push({d[y], y});
      }
    }
  }

  if (d[s] == -1) std::vector<T>(k, -1);
  std::vector<int> heap(n);
  h.cnt = 0;
  for (int i = 0; i < m; i++) {
    auto &[x, y, z] = e[i];
    if (d[x] != -1 && d[y] != -1 && fa[x] != i) {
```

```
      heap[x] = h.merge(heap[x], h.newNode(y, d[y] + z -
      ↪  d[x]));
    }
  }

  for (int x : p) {
    if (x != t) heap[x] = h.merge(heap[x], heap[e[fa[x]].y]);
  }
  if (heap[s]) q.push({d[s] + h.t[heap[s]].w, heap[s]});
  std::vector<T> res = {d[s]};

  for (int i = 1; i < k && !q.empty(); i++) {
    auto [w, o] = q.top();
    q.pop();

    res.push_back(w);

    int j = h.t[o].v;
    if (heap[j]) q.push({w + h.t[heap[j]].w, heap[j]});
    for (auto s : {h.t[o].ls, h.t[o].rs}) {
      if (s) q.push({w + h.t[s].w - h.t[o].w, s});
    }
  }
  res.resize(k, -1);
  return res;
}
```

6.13 DMST

```
template <class W>
struct DMST {
  struct Node {
    int ls, rs, x, y;
    W w, a;
  };
  std::vector<int> h;
  std::vector<Node> t;

  void apply(int o, W x) { if (o) t[o].w -= x, t[o].a += x; }
  void pushdown(int o) {
    apply(t[o].ls, t[o].a), apply(t[o].rs, t[o].a);
    t[o].a = 0;
  }
  int merge(int x, int y) {
    if (!x || !y) return x | y;
    if (t[y].w < t[x].w) std::swap(x, y);
    pushdown(x);
    t[x].rs = merge(t[x].rs, y);
    std::swap(t[x].ls, t[x].rs);
    return x;
  }
  void pop(int x) {
    int &o = h[x];
    pushdown(o), o = merge(t[o].ls, t[o].rs);
  }

  DMST(int n, int m = 0) : h(n), t(1) { t.reserve(m + 1); }

  void add(int x, int y, W z) {
    t.push_back({0, 0, x, y, z, W(0)});
    h[y] = merge(h[y], t.size() - 1);
  }

  template <class T = W>
  std::pair<T, std::vector<int>> solve(int r) {
    T ans = 0;
    int n = h.size();
    std::vector<int> p(n);
    std::vector<PII> c;
    Dsu a(n);
    RDsu b(n);
    for (int i = 0, x, y, z, k; i < n; i++) {
      if (i == r) continue;
      for (x = i;;) {
        if (!h[x]) return {0, {}};
        p[x] = h[x];
        ans += t[p[x]].w;
        apply(p[x], t[p[x]].w);
        if (a.merge(x, b.find(t[p[x]].x))) break;
        y = b.find(t[p[x]].x), k = b.time();
        while (b.merge(x, y)) {
          h[z = b.find(x)] = merge(h[x], h[y]);
          x = z, y = b.find(t[p[y]].x);
        }
        c.emplace_back(p[x], k);
        while (h[x] && b.same(t[h[x]].x, x)) pop(x);
      }
    }

    for (auto it = c.rbegin(); it != c.rend(); it++) {
      int u = b.find(t[it->first].y);
      b.undo(it->second);
      int v = b.find(t[p[u]].y);
      p[v] = std::exchange(p[u], it->first);
    }

    for (int i = 0; i < n; i++) p[i] = i == r ? -1 :
    ↪  t[p[i]].x;
    return {ans, p};
  }
};
```

Combinatorial (7)

7.1 Poly

7.1.1 DFT

```
constexpr u32 P(998244353), G(3), L(1 << 19);
int fpow(int x, int k = P - 2) {
  int r = 1;
  for (; k; k >>= 1, x = 1LL * x * x % P)
    if (k & 1) r = 1LL * r * x % P;
  return r;
}
void inc(int &x, int y) { if ((x += y) >= P) x -= P; }
int madd(int x) { return x >= P ? x - P : x; }
int msub(int x) { return x < 0 ? x + P : x; }
int cmul(u32 x, u64 y, u64 iy) { return madd(x * y - (iy * x
↪  >> 32) * P); }
u32 w[L][2];
int fac[L], ifac[L], inv[L], _ = [] {
  w[L / 2][0] = 1;
  for (int i = L / 2 + 1, x = fpow(G, (P - 1) / L); i < L;
  ↪  i++) {
    w[i][0] = cmul(x, w[i - 1][0], w[i - 1][1]);
    w[i][1] = (u64(w[i][0]) << 32) / P;
  }
  for (int i = L / 2 - 1; i >= 0; i--)
    w[i][0] = w[i << 1][0], w[i][1] = w[i << 1][1];

  fac[0] = 1;
  for (int i = 1; i < L; i++) fac[i] = 1LL * fac[i - 1] * i %
  ↪  P;
  ifac[L - 1] = fpow(fac[L - 1]);
  for (int i = L - 1; i; i--) {
    ifac[i - 1] = 1LL * ifac[i] * i % P;
    inv[i] = 1LL * ifac[i] * fac[i - 1] % P;
  }
  return 0;
}();

void dft(int *a, int n) {
  for (int k = n >> 1; k; k >>= 1) {
    for (int i = 0; i < n; i += k << 1) {
      for (int j = 0; j < k; j++) {
```

```
        int &x = a[i + j], y = a[i + j + k];
        a[i + j + k] = cmul(x - y + P, w[k + j][0], w[k + j][1]);
        inc(x, y);
      }
    }
  }
}
void idft(int *a, int n) {
  for (int k = 1; k < n; k <<= 1) {
    for (int i = 0; i < n; i += k << 1) {
      for (int j = 0; j < k; j++) {
        int &x = a[i + j], y = cmul(a[i + j + k], w[k + j][0], w[k + j][1]);
        a[i + j + k] = msub(x - y);
        inc(x, y);
      }
    }
  }
  u64 x = P - (P - 1) / n, ix = (x << 32) / P;
  for (int i = 0; i < n; i++) a[i] = cmul(a[i], x, ix);
  std::reverse(a + 1, a + n);
}

void dftD(int *a, int n) {
  std::copy_n(a, n, a + n);
  idft(a + n, n);
  for (int i = 0; i < n; i++) a[n + i] = cmul(a[n + i], w[n + i][0], w[n + i][1]);
  dft(a + n, n);
}

int norm(int n) { return 1 << std::__lg(n * 2 - 1); }
```

7.1.2 Struct Poly

```
struct Poly : std::vector<int> {
#define T (*this)
  using vector::vector;
  int size() const { return vector::size(); }
  void append(const Poly &r) { insert(end(), r.begin(), r.end()); }
  Poly pre(int k) const { return k < size() ? Poly(begin(), begin() + k) : T; }
  Poly &resize(int k) { return vector::resize(k), T; }
  Poly rev() const { return Poly(rbegin(), rend()); }
  friend void dft(Poly &a) { dft(a.data(), a.size()); }
  friend void idft(Poly &a) { idft(a.data(), a.size()); }
  friend Poly conv(const Poly &a, const Poly &b, int n) {
    Poly p(a), q;
    dft(p.resize(n));
    idft(p ^= &a == &b ? p : (dft((q = b).resize(n)), q));
    return p;
  }
  friend Poly operator*(const Poly &a, const Poly &b) {
    if (a.empty() || b.empty()) return {};
    int len = a.size() + b.size() - 1;
    if (a.size() < 16 || b.size() < 16) {
      Poly c(len);
      for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < b.size(); j++)
          c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
      return c;
    }
    return conv(a, b, norm(len)).pre(len);
  }
  Poly mulT(Poly b) { return T * b.rev() >> b.size() - 1; }

  Poly deriv() const {
    if (empty()) return {};
    Poly r(size() - 1);
    for (int i = 1; i < size(); i++) r[i - 1] = 1LL * i * T[i] % P;
    return r;
  }
  Poly integ() const {
    if (empty()) return {};
    Poly r(size() + 1);
    for (int i = 0; i < size(); i++) r[i + 1] = 1LL * ::inv[i + 1] * T[i] % P;
    return r;
  }
  Poly inv(int m) const {
    Poly x = {fpow(T[0])};
    for (int k = 1; k < m; k *= 2)
      x.append(-((conv(pre(k * 2), x, k * 2) >> k) * x).pre(k));
    return x.resize(m);
  }
  Poly log(int m) const { return (deriv() * inv(m)).integ().resize(m); }
  Poly exp(int m) const {
    Poly x = {1};
    for (int k = 1; k < m; k *= 2)
      x.append((x * (pre(k * 2) - x.log(k * 2) >> k)).pre(k));
    return x.resize(m);
  }
  Poly sqrt(int m) const {
    Poly x = {1}, y = {1};
    for (int k = 1; k < m; k *= 2) {
      x.append(((pre(k * 2) - x * x >> k) * y).pre(k) * (P + 1 >> 1));
      if (k * 2 < m)
        y.append(-((conv(x.pre(k * 2), y, k * 2) >> k) * y).pre(k));
    }
    return x.resize(m);
  }
  Poly powexp(LL k, int m) const {
    if (!k) return Poly{1}.resize(m);
    int z = 0;
    while (z < size() && !T[z]) z++;
    if (z == size() || z >= (m + k - 1) / k) return Poly(m);
    int r = m - z * k;
    return (((T >> z).log(r) * (k % P)).exp(r) * fpow(T[z], k % (P - 1)) << z * k).resize(m);
  }
  Poly operator-() const {
    Poly r(T);
    for (auto &x : r) if (x) x = P - x;
    return r;
  }
  Poly &operator+=(const Poly &r) {
    if (r.size() > size()) resize(r.size());
    for (int i = 0; i < r.size(); i++) inc(T[i], r[i]);
    return T;
  }
  Poly &operator-=(const Poly &r) {
    if (r.size() > size()) resize(r.size());
    for (int i = 0; i < r.size(); i++) inc(T[i], P - r[i]);
    return T;
  }
  Poly &operator^=(const Poly &r) {
    assert(r.size() == size());
    for (int i = 0; i < size(); i++) T[i] = 1LL * T[i] * r[i] % P;
    return T;
  }
  Poly &operator*=(int r) {
    for (int &x : T) x = 1LL * x * r % P;
    return T;
  }
  Poly operator+(const Poly &r) const { return Poly(T) += r; }
  Poly operator-(const Poly &r) const { return Poly(T) -= r; }
  Poly operator^(const Poly &r) const { return Poly(T) ^= r; }
  Poly operator*(int r) const { return Poly(T) *= r; }
  Poly &operator<<=(int k) { return insert(begin(), k, 0), T; }
  Poly operator<<(int r) const { return Poly(T) <<= r; }
```

```
  Poly operator>>(int r) const { return r >= size() ? Poly() :
  ↪  Poly(begin() + r, end()); }
  Poly &operator>>=(int r) { return T = T >> r; }
#undef T
};
```

7.1.3 Qoly - RelaxedConvolution

```
using Fn = std::function<int(int)>;
struct Q {
  Q() = default;
  Q(Fn &&fn) : f(std::forward<Fn>(fn)) {}

  Fn f;
  Poly c;
  virtual void await(int n) {
    while (c.size() < n) c.push_back(f(c.size()));
  }
  int at(int k) { return c[k]; }

  std::vector<Poly> d;  // [0, 2^k) dft
  Poly &get(int k) {
    if (k >= d.size()) d.resize(k + 1);
    auto &p = d[k];
    if (p.empty()) dft(p = take(0, 1 << k));
    return p;
  }
  Poly take(int l, int r) {
    await(r);
    return {c.begin() + l, c.begin() + r};
  }
};

struct FixQ : Q {
  FixQ(const Poly &p) { c = p; }
  void await(int n) override {
    if (c.size() < n) c.resize(n);
  }
};

struct RC : Q {
  Q *a, *b;
  int m;
  RC(Q *a, Q *b) : a(a), b(b), m(0) {}
  void await(int n) override {
    constexpr int M(32);
    a->await(n), b->await(n);
    for (; m < n; m++) {
      if (!m) {
        c.push_back((u64)a->at(0) * b->at(0) % P);
        continue;
      }
      int k = m & -m, l = m - k, r = m + k, t = std::__lg(k);
      if (c.size() < r) c.resize(r);
      c[m] = (c[m] + (u64)a->at(0) * b->at(m) + (u64)a->at(m)
      ↪  * b->at(0)) % P;
      if (m == k) {
        Poly p = a->get(t);
        idft(p ^= b->get(t));
        for (int i = m; i < r; i++)
          inc(c[i], p[i - m]), inc(c[i], P - c[i - m]);
      } else if (k < M) {
        for (int i = m; i < r; i++)
          for (int j = l; j < m; j++)
            c[i] = ((u64)a->at(j) * b->at(i - j) +
                    (u64)b->at(j) * a->at(i - j) + c[i]) % P;
      } else {
        k *= 2, t++;
        Poly x(b->take(l, m)), z(a->take(l, m));
        Poly &y(a->get(t)), &w(b->get(t));
        dft(x.resize(k)), dft(z.resize(k));
        for (int i = 0; i < k; i++)
          x[i] = ((u64)x[i] * y[i] + (u64)z[i] * w[i]) % P;
        idft(x);
        for (int i = m; i < r; i++) inc(c[i], x[i - l]);
      }
    }
  }
};

struct Qoly {
  Q *q;
  Qoly(Q *q = new Q) : q(q) {}
  Qoly(const Poly &p) : q(new FixQ(p)) {}
  Qoly(Fn &&f) : q(new Q{std::forward<Fn>(f)}) {}
  void set(Qoly &&r) { q->f = std::move(r.q->f); }
  int at(int k) const { return q->await(k + 1), q->at(k); }
  int operator[](int k) const { return at(k); }

  friend Qoly operator*(Qoly a, Qoly b) { return {new RC{a.q,
  ↪  b.q}}; }
  Qoly operator*(int x) {
    return {[=, *this](int i) { return 1LL * at(i) * x % P;
    ↪  }};
  }
  Qoly operator+(Qoly b) {
    return {[=, *this](int i) { return madd(at(i) + b[i]); }};
  }
  Qoly operator-(Qoly b) {
    return {[=, *this](int i) { return msub(at(i) - b[i]); }};
  }
  Qoly replace(const Poly &p) {
    return {[=, *this](int i) { return i < p.size() ? p[i] :
    ↪  at(i); }};
  }
  Qoly deriv() {
    return {[=, *this](int i) { return LL(i + 1) * at(i + 1) %
    ↪  P; }};
  }
  Qoly integ(int c = 0) {
    return {[=, *this](int i) { return i ? 1LL * at(i - 1) *
    ↪  ::inv[i] % P : c; }};
  }
  Qoly operator>>(int k) {
    return {[=, *this](int i) { return at(i + k); }};
  }
  Qoly operator<<(int k) {
    return {[=, *this](int i) { return i >= k ? at(i - k) : 0;
    ↪  }};
  }
  Qoly log(int c = 0) { return (deriv() * inv()).integ(c); }
  Qoly exp() {
    Qoly g;
    return g.set((g * deriv()).integ(1)), g;
  }
  Qoly inv() {
    // (f0+f1)g=1, g=(1-f1g)/f0
    Qoly g, h = g * operator>>(1);
    g.q->f = [=, *this, k = 0](int i) mutable {
      return i ? h[i - 1] * LL(P - k) % P : k = fpow(at(0));
    };
    return g;
  }
  Qoly sqrt() {
    // g0^2+g1^2+2g0g1=f, g1 = (f-f0-g1^2)/(2g0)
    Qoly g, g1 = g >> 1, h = g1 * g1;
    g.q->f = [=, *this, k = 0](int i) mutable {
      if (!i) {
        auto v = modSqrt(at(0), P);
        assert(!v.empty());
        k = fpow(2 * v[0]);
        return v[0];
      }
      return LL(at(i) + (i >= 2 ? P - h[i - 2] : 0)) * k % P;
    };
    return g;
  }

  Qoly harmo(bool p, bool inv) {  // inv is not verified
```

```
    // \sum_{k>=1} [s ? (-1)^{k-1} : 1] F(x^k)/k
    return {[=, *this, t = Poly()](int i) mutable {
      if (!i) return 0;
      if (!(i & i - 1)) {
        t.resize(i * 2);
        for (int j = 1; j < i; j++) {
          u64 x = inv ? P - t[j] : at(j);
          for (int k = (i + j - 1) / j; j * k < i * 2; k++)
            t[j * k] = (t[j * k] + (p && ~k & 1 ? P - x : x) *
            ↪  ::inv[k]) % P;
        }
      }
      return inc(t[i], at(i)), t[i];
    }};
  }

  Qoly mset() { return harmo(0, 0).exp(); }
  Qoly pset() { return harmo(1, 0).exp(); }
  Qoly imset() { return log().harmo(0, 1); }
  Qoly ipset() { return log().harmo(1, 1); }
};
```

7.1.4 Poly EI

```
struct SegTree {
  int n, s;
  std::vector<Poly> p;
  SegTree(Poly a) : n(a.size()), s(norm(n)), p(s * 2) {
    for (int i = 0; i < s; i++)
      p[i + s] = Poly({1, i < n && a[i] ? P - a[i] : 0});
    for (int i = s - 1; i; i--) {
      int l = i * 2, r = l + 1, k = p[l].size() - 1 << 1;
      dft(p[l].resize(k)), dft(p[r].resize(k));
      idft(p[i] = p[l] ^ p[r]);
      p[i].push_back((p[i][0] - 1 + P) % P);
      p[i][0] = 1;
    }
  }
  Poly eval(Poly f) {
    int m = f.size();
    if (m == 1) return Poly(n, f[0]);
    Poly q = f.rev() * p[1].inv(m);
    q.resize(m);
    if (m > s) {
      q >>= m - s;
    } else {
      q <<= s - m;
    }
    for (int k = s, o = 1; k > 1; k >>= 1) {
      for (int i = 0; i < s; i += k, o++) {
        if (i >= n) continue;
        int *a = &q[i], *l = p[o * 2].data(), *r = p[o * 2 +
        ↪  1].data();
        dft(a, k);
        Poly x(k), y(k);
        for (int j = 0; j < k; j++) {
          x[j] = 1LL * a[j] * r[j] % P;
          y[j] = 1LL * a[j] * l[j] % P;
        }
        idft(x), idft(y);
        std::copy(x.begin() + k / 2, x.end(), a);
        std::copy(y.begin() + k / 2, y.end(), a + k / 2);
      }
    }
    return q.pre(n);
  }
  Poly interpolate(Poly b) {
    assert(b.size() == n);
    Poly q = eval(Poly(p[1]).resize(n + 1).rev().deriv());
    for (int i = 0; i < n; i++) q[i] = 1LL * fpow(q[i]) * b[i]
    ↪  % P;
    q.resize(s);
    for (int k = 1, h = s >> 1; k < s; k <<= 1, h >>= 1)
      for (int i = 0, o = h; i < s; i += k << 1, o++) {
        if (i >= n) continue;
        int *a = &q[i];
        Poly x(k * 2), y(k * 2);
        for (int j = 0; j < k; j++) x[j] = a[j], y[j] = q[i +
        ↪  j + k];
        dft(x), dft(y);
        for (int j = 0; j < k * 2; j++)
          a[j] = (1LL * x[j] * p[o * 2 + 1][j] + 1LL * y[j] *
          ↪  p[o * 2][j]) % P;
        idft(a, k * 2);
      }
    return q.pre(n).rev();
  }
};
```

7.1.5 DivAt

```
int divAt(Poly f, Poly g, int64_t k) { // 5e5, 1e18, 2s
  int len = std::max(f.size(), g.size()), n = norm(len);

  std::vector<int> rw(n);
  for (int i = n / 2, x = fpow(G, P - 1 - P / (n * 4)); i; i
  ↪  >>= 1)
    rw[i] = x = 1LL * x * x % P;
  rw[0] = 1;
  for (int i = 3; i < n; i++)
    if (i & i - 1) rw[i] = 1LL * rw[i & i - 1] * rw[i & -i] %
    ↪  P;
  for (int &i : rw) i = (i & 1 ? i + P : i) >> 1;

  dft(f.resize(n * 2)), dft(g.resize(n * 2));
  for (;;) {
    for (int i = 0; i < n * 2; i++) f[i] = 1LL * f[i] * g[i ^
    ↪  1] % P;
    if (k & 1) {
      for (int i = 0; i < n; i++)
        f[i] = LL(f[i * 2] - f[i * 2 + 1] + P) * rw[i] % P;
    } else {
      for (int i = 0; i < n; i++) {
        f[i] = f[i * 2] + f[i * 2 + 1];
        f[i] = (f[i] & 1 ? f[i] >= P ? f[i] - P : f[i] + P :
        ↪  f[i]) >> 1;
      }
    }
    for (int i = 0; i < n; i++) g[i] = 1LL * g[i * 2] * g[i *
    ↪  2 + 1] % P;
    if ((k >>= 1) < n) break;
    dftD(f.data(), n), dftD(g.data(), n);
  }
  idft(f.resize(n)), idft(g.resize(n)), g = g.inv(n);
  int ans = 0;
  for (int i = 0; i <= k; i++) ans = (ans + 1LL * f[i] * g[k -
  ↪  i]) % P;
  return ans;
}

Poly invRange(Poly q, int64_t l, int64_t r) {
  assert(l <= r);
  assert(r - l + 1 <= 5e5);
  int len = std::max<int>(q.size(), r - l + 1), m = 1 <<
  ↪  std::__lg(len * 2 - 1);
  std::function<Poly(Poly&, int64_t)> cal = [&](Poly &a,
  ↪  int64_t n) -> Poly {
    if (n == 0) {
      Poly res(len);
      int c = 0;
      for (int i = 0; i < m; i++) inc(c, a[i]);
      res.back() = 1LL * m * fpow(c) % P;
      return res;
    }
    dftD(a.data(), m);
    Poly b(m * 2);
    for (int i = 0; i < m; i++) b[i] = 1LL * a[i * 2] * a[i *
    ↪  2 + 1] % P;
    auto c = cal(b, n >> 1);
```

```
    std::fill(b.begin(), b.end(), 0);
    for (int i = 0, o = n & 1 ^ 1; i < len; i++) b[i * 2 ^ o]
    ↪ = c[i];
    dft(b);
    for (int i = 0; i < m * 2; i++) b[i] = 1LL * b[i] * a[i ^
    ↪ 1] % P;
    idft(b);
    return Poly(b.begin() + len, b.begin() + len * 2);
  };
  q.resize(m * 2);
  dft(q.data(), m);
  q = cal(q, r);
  return Poly(q.end() - (r - l + 1), q.end());
}
int divAt2(Poly f, Poly g, int64_t n) {
  g = invRange(g, n - ((int)f.size() - 1), n);
  g = f * g;
  return g[f.size() - 1];
}
```

7.1.6 Shift

```
// input : Poly A(x), output : A(x + b)
Poly shift(Poly a, int b) {
  if (!b) return a;
  int n = a.size();
  for (int i = 0; i < n; i++) a[i] = 1LL * a[i] * fac[i] % P;
  std::reverse(a.begin(), a.end());
  Poly p(n);
  for (int i = 0; i < n; i++) p[i] = 1LL * fpow(b, i) *
  ↪ ifac[i] % P;
  a = a * p;
  a.resize(n);
  std::reverse(a.begin(), a.end());
  for (int i = 0; i < n; i++) a[i] = 1LL * a[i] * ifac[i] % P;
  return a;
}

// input : f(0), f(1), ..., f(n - 1) (Point Value)
// output : f(t), f(t + 1), ..., f(t + m - 1)
// (if m is default, m = n)
Poly PtShift(const Poly &f, LL t, int m = -1) {
  if (m == -1) m = f.size();
  int k = f.size() - 1;
  t %= P;
  if (t <= k) {
    Poly r(m);
    int p = 0;
    for (int i = t; i <= k && p < m; i++) r[p++] = f[i];
    if (k + 1 < t + m) {
      auto q = PtShift(f, k + 1, m - p);
      for (int i = k + 1; i < t + m; i++) {
        r[p++] = q[i - k - 1];
      }
    }
    return r;
  }
  if (t + m > P) {
    auto p = PtShift(f, t, P - t);
    auto q = PtShift(f, 0, m - p.size());
    p.insert(p.end(), q.begin(), q.end());
    return p;
  }

  Poly d(k + 1), h(m + k), r(m);
  for (int i = 0; i <= k; i++) {
    d[i] = 1LL * ifac[i] * ifac[k - i] % P * f[i] % P;
    if (k - i & 1 && d[i]) d[i] = P - d[i];
  }
  for (int i = 0; i < m + k; i++) h[i] = fpow(t - k + i);
  d = d * h;

  int cur = t;
  for (int i = 1; i <= k; i++) cur = cur * (t - i) % P;
  for (int i = 0; i < m; i++) {
    r[i] = 1LL * cur * d[k + i] % P;
    cur = (t + i + 1) * h[i] % P * cur % P;
  }
  return r;
}
```

7.1.7 Division

```
Poly operator/(Poly a, Poly b) {
  int n = a.size(), m = b.size();
  if (n < m) return {0};
  int k = norm(n - m + 1);
  return (a.rev().resize(k) * b.rev().inv(k)).pre(n - m +
  ↪ 1).rev();
}
std::pair<Poly, Poly> div(Poly a, Poly b) {
  int m = b.size();
  Poly c = a / b;
  return {c, a.pre(m - 1) - (b * c).pre(m - 1)};
}
Poly operator%(Poly a, Poly b) {
  return div(a, b).second;
}
```

7.1.8 Primitive Root

```
int primitiveRoot(int m) {
  std::vector<int> pr;
  int g = 1, s = m - 1;  // ! phi(m)
  for (int i = 2; i * i <= s; i++) {
    if (s % i == 0) {
      pr.push_back(i);
      while (s % i == 0) s /= i;
    }
  }
  if (s > 1) pr.push_back(s);
  for (;;) {
    for (int p : pr)
      if (fpow(g, (m - 1) / p, m) == 1) goto I;
    return g;
  I:
    g++;
  }
}
```

7.1.9 Fast Factorial

```
// O(√P log P)
int factorial(int n) {
  if (n <= 1) return 1;
  LL v = 1;
  while(v * v < n) v <<= 1;
  int iv = fpow(v % P);
  Poly g = {1, mod(v + 1)};
  for (int d = 1; d != v; d <<= 1) {
    Poly a = PtShift(g, 1LL * d * iv % P);
    Poly b = PtShift(g, (d * v + v) * iv % P);
    Poly c = PtShift(g, (d * v + d + v) * iv % P);
    for (int i = 0; i <= d; i++) {
      g[i] = 1LL * g[i] * a[i] % P;
      b[i] = 1LL * b[i] * c[i] % P;
    }
    g.insert(g.end(), b.begin(), b.end() - 1);
  }
  int r = 1;
  LL i = 0;
  while (i + v <= n) r = 1LL * r * g[i / v] % P, i += v;
  while (i < n) r = 1LL * r * ++i % P;
  return r;
}
```

7.1.10 Compound and Inv

```
// useless algorithm
/*
```

```cpp
 * Description: F(G(x))
 * Time: O(n sqrt n log n + n ^ 2)
 */
Poly compound(Poly f, Poly g) {
  int n = f.size(), k = norm(2 * n - 1), m = sqrt(n) + 1;
  std::vector<Poly> p(m + 1);
  Poly q(k), h(k), t;
  q[0] = 1, p[0] = q;
  auto fd = [&](auto &f) { dft(f.resize(k)); };
  auto bd = [&](auto &f) { idft(f), f.resize(n); };
  fd(g);
  for (int i = 1; i <= m; ++i) {
    fd(p[i] = p[i - 1]), bd(p[i] ^= g);
  }
  fd(g = p[m]);
  for (int i = 0; i < m; i++, t.clear(), bd(q)) {
    for (int j = 0; j < m && j < n - i * m; j++)
      t += p[j] * f[i * m + j];
    fd(t), fd(q);
    for (int j = 0; j < k; ++j) {
      h[j] = (h[j] + 1LL * t[j] * q[j]) % P;
      q[j] = 1LL * q[j] * g[j] % P;
    }
  }
  return bd(h), h;
}

/*
 * Description: get F(x) for F(G(x)) = x
 * Time: O(n sqrt n log n + n ^ 2)
 */
Poly compoundInv(Poly g) {
  int n = g.size(), k = norm(2 * n - 1), m = sqrt(n) + 1;
  auto fd = [&](auto &f) { dft(f.resize(k)); };
  auto bd = [&](auto &f) { idft(f), f.resize(n); };
  std::vector<Poly> p(m + 1);
  Poly q(k), f(n);
  q[0] = 1, p[0] = q, fd(g = (g >> 1).inv(n));
  for (int i = 1; i <= m; ++i) fd(p[i] = p[i - 1]), bd(p[i] ^=
↪ g);
  fd(g = p[m]);
  for (int i = 0, t = 1; i < m; i++) {
    for (int j = 1; j <= m && t < n; j++, t++) {
      for (int r = 0; r < t; ++r)
        f[t] = (f[t] + 1LL * p[j][r] * q[t - 1 - r]) % P;
      f[t] = 1LL * f[t] * inv[t] % P;
    }
    if (i + 1 < m) fd(q), bd(q ^= g);
  }
  return f;
}
```

7.2 Largrange

7.2.1 Formula

$$f(x)=\sum_{i=1}^{n} y_i \prod_{i\neq j}\frac{x-x_j}{x_i-x_j}$$

7.2.2 Interpolate Iota

```cpp
// from yousupo
// calculate F(x) where F(i) = f_i, i = 0...n-1
int interpolateIota(const Poly &f, int x) {
  int n = f.size();
  std::vector<int> r(n + 1);
  r[n] = 1;
  for (int i = n - 1; i >= 0; --i) {
    r[i] = 1LL * (x - i) * r[i + 1] % P;
  }
  int ans = 0, l = 1;
  for (int i = 0; i < n; ++i) {
    ans = (ans + (n - 1 - i & 1 ? -1LL : 1LL) * ifac[i] *
    ↪ ifac[n - 1 - i] % P * f[i] % P * l % P * r[i + 1]) %
    ↪ P;
    l = 1LL * (x - i) * l % P;
  }
  return (ans + P) % P;
}
```

7.3 Min-max 容斥

$$\mathrm{kmax}_{i\in S}\, x_i=\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}\min_{j\in T} x_j$$

7.4 FWT

7.4.1 2-FWT

Xor

$$H_{i,p \operatorname{xor} q}=H_{i,p}H_{i,q}$$

容易构造出 $\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}$ 或 $\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$。

其中 $\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$ 有特殊含义 $H_{i,j}=(-1)^{\operatorname{popcnt}(i \operatorname{and} j)}$，使用该矩阵，$H^{-1}=\begin{bmatrix}\frac12 & \frac12\\ \frac12 & -\frac12\end{bmatrix}$。

注意到 $H^{-1}=\frac12 H$，那么 $H_{2^k}^{-1}=\frac{1}{2^k}H_{2^k}$，所以对于异或卷积，逆变换相当于正变换后每一项除以 n，与傅里叶变换类似。

7.4.2 子集卷积

$$c_n=\sum_{p,q}[p \operatorname{or} q=n][p \operatorname{and} q=0]a_pb_q$$

将限制条件转化为 $[p \operatorname{or} q=n][\operatorname{popcnt}(p)+\operatorname{popcnt}(q)=\operatorname{popcnt}(n)]$。

利用占位多项式

$$A_n(x)=a_nx^{\operatorname{popcnt}(n)}$$
$$B_n(x)=b_nx^{\operatorname{popcnt}(n)}$$
$$C_n(x)=\sum_{p \operatorname{or} q=n}A_p(x)B_q(x)$$
$$c_n=[x^{\operatorname{popcnt}(n)}]C_n(x)$$

也可以交换两个维度，定义两个向量的或卷积为

$$(\mathbf{A} \operatorname{or} \mathbf{B})_n=\sum_{p \operatorname{or} q=n}a_pb_q$$

则

$$\mathbf{A}_n=([\operatorname{popcnt}(0)=n]a_0,[\operatorname{popcnt}(1)=n]a_1,\dots)$$
$$\mathbf{B}_n=([\operatorname{popcnt}(0)=n]b_0,[\operatorname{popcnt}(1)=n]b_1,\dots)$$
$$\mathbf{C}_n=\mathbf{A}_n \operatorname{or} \mathbf{B}_n$$

7.4.3 k-FWT

Xor

设 $i=i_{k-1}\dots i_1i_0, j=j_{k-1}\dots j_1j_0$ 其中 $i_l,j_l\in[0,m)$，定义 m 进制下 xor 运算为

$$(i \operatorname{xor} j)_l=(i_l+j_l) \bmod m$$

发现这个就是循环卷积

对于 ω_m 不存在的情形，需要扩域：

分圆多项式

$$\Phi_n(x)=\prod_{i=0}^{n-1}(x-\omega_n^i)^{[i\perp n]}$$

易证

$$\prod_{d|n}\Phi_d(x)=x^n-1$$

莫比乌斯反演可得

$$\Phi_n(x)=\prod_{d|n}(x^d-1)^{\mu(n/d)}$$

定理：

1. 分圆多项式在 Q 上不可约。

2. 在模 $\Phi_n(x)$ 意义下，x 的阶恰好为 k。

所以可以用 x 代替 ω_m，运算在模 $\Phi_m(x)$ 意义下进行。
如果 $\Phi_m(x)$ 比较简单，可以直接模 $\Phi_m(x)$，否则，由于 $\Phi_m(x) \mid (x^m - 1)$
所以可以先在模 $x^m - 1$ 意义下运算，即做长度为 m 的循环卷积，最后再模 $\Phi_m(x)$。
暴力取模的代码

```
for (int i = n; i >= m; i--) if (g[i]) {
  int t = 1LL * g[i] * fpow(f[m]) % P;
  for (int j = m; j >= 0; j--)
    g[i - j] = (g[i - j] - 1LL * f[j] * t) % P;
}
```

7.5 Various FFT

7.5.1 General

$$P(x) = A(x) + iB(x)$$
$$P^2(x) = A^2(x) - B^2(x) + i2A(x)B(x)$$
$$A(x)B(x) = \frac{1}{2}\operatorname{Imag} P^2(x)$$
$$P(x) = A(x) + iB(x)$$
$$\overline{P}(x) = A(x) - iB(x)$$
$$A(x) = \frac{P(x) + \overline{P}(x)}{2}$$
$$B(x) = \frac{P(x) - \overline{P}(x)}{2i}$$
$$\overline{P}(\omega^k) = \overline{P(\omega^{-k})}$$

7.5.2 FFT Complex

```
namespace FFT {
const long double PI = acosl(-1.L);
struct C {
  double x, y;
  C(double x = 0, double y = 0) : x(x), y(y) {}
  C operator!() const { return C(x, -y); }
};
inline C operator*(C a, C b) { return C(a.x * b.x - a.y * b.y,
    a.y * b.x + b.y * a.x); }
inline C operator+(C a, C b) { return C(a.x + b.x, a.y + b.y);
  }
inline C operator-(C a, C b) { return C(a.x - b.x, a.y - b.y);
  }
constexpr int L(1 << 20);

/*
int r[L << 1], _ = [] {
  for (int i = 0; i < L / 2; i++) w[i + L / 2] = C(cosl(2.L *
    PI * i / L), sinl(2.L * PI * i / L));
  for (int i = L / 2 - 1; i; i--) w[i] = w[i << 1];
  for (int i = 1; i <= L; i <<= 1) {
    for (int j = 0; j < i; j++) {
      r[i + j] = r[i + j >> 1] | (i >> 1) * (j & 1);
    }
  }
  return 0;
}();
*/

C w[L];
void dft(C *a, int n) {
  for (static int k = (w[1].x = 1, 2); k < n; k <<= 1) {
    for (int i = 0; i < k; i++) {
      w[i + k] = i & 1 ? C(cosl(PI * i / k), sinl(PI * i / k))
        : w[i + k >> 1];
    }
  }
  for (int k = n >> 1; k; k >>= 1) {
    for (int i = 0; i < n; i += k << 1) {
      for (int j = 0; j < k; j++) {
        C &x = a[i + j], y = a[i + j + k];
        a[i + j + k] = (x - y) * w[k + j], x = x + y;
      }
    }
  }
}
void idft(C *a, int n) {
  for (int k = 1; k < n; k <<= 1) {
    for (int i = 0; i < n; i += k << 1) {
      for (int j = 0; j < k; j++) {
        C &x = a[i + j], y = a[i + j + k] * w[k + j];
        a[i + j + k] = x - y, x = x + y;
      }
    }
  }
  for (int i = 0; i < n; i++) a[i].x /= n, a[i].y /= n;
  std::reverse(a + 1, a + n);
}
}
```

7.5.3 MTT

```
Poly conv(const Poly &a, const Poly &b, int n) {  // wrong
  when n = 1
  using namespace FFT;
  assert((n & n - 1) == 0);
  std::vector<C> f(n), g(n), p(n), q(n);
  for (int i = 0; i < a.size(); i++) {
    f[i] = C(a[i] >> 15, a[i] & 0x7fff);
  }
  for (int i = 0; i < b.size(); i++) {
    g[i] = C(b[i] >> 15, b[i] & 0x7fff);
  }
  dft(f.data(), n), dft(g.data(), n);
  C h1(0.5, 0), h2(0, -0.5);
  for (int k = 1, i = 0, j; k < n; k <<= 1) {
    for (; i < k * 2; i++) {
      auto z = !f[j = i ^ k - 1];
      p[i] = h1 * g[i] * (f[i] + z);
      q[i] = h2 * g[i] * (f[i] - z);
    }
  }
  idft(p.data(), n), idft(q.data(), n);

  Poly r(n);
  for (int i = 0; i < n; i++) {
    LL x, y, z, w;
    x = p[i].x + 0.5;
    y = p[i].y + 0.5;
    z = q[i].x + 0.5;
    w = q[i].y + 0.5;
    r[i] = ((x % P << 30) + ((y + z) % P << 15) + w) % P;
  }
  return r;
}
```

7.5.4 三模数

```
constexpr int P1(998244353), P2(1004535809), P3(469762049);

Tp<P1, 3> tp1;
Tp<P2, 3> tp2;
Tp<P3, 3> tp3;

constexpr LL P12(1LL * P1 * P2);
constexpr int Inv1(tp2.fpow(P1)), Inv2(tp3.fpow(P12 % P3));

int get(int a, int b, int c) {
  LL x = LL(b - a + P2) * Inv1 % P2 * P1 + a;
  return (LL(c - x % P3 + P3) * Inv2 % P3 * (P12 % P) % P + x)
    % P;
}
```

7.6 Chirp Z-Transform

给定一个 n 项多项式 $P(x)$ 以及 c, m，请计算 $P(c^0), P(c^1), \ldots, P(c^{m-1})$。
所有答案都对 998244353 取模。

关键公式：

$$ij = \binom{i+j}{2} - \binom{i}{2} - \binom{j}{2}$$

```
Poly f(n + m - 1, 1), ipc(std::max(n, m), 1), g(n);
for (int i = 2, x = 1; i < n + m - 1; i++) {
  x = 1LL * x * c % P;
  f[i] = 1LL * f[i - 1] * x % P;
}
for (int i = 2, u = fpow(c), x(1); i < ipc.size(); i++) {
  x = 1LL * x * u % P;
  ipc[i] = 1LL * ipc[i - 1] * x % P;
}
for (int i = 0; i < n; i++) {
  std::cin >> g[i];
}
g ^= ipc.pre(n);
std::reverse(g.begin(), g.end());
f = conv(f, g, norm(n + m - 1)) >> (n - 1);
f ^= ipc.pre(m);
for (int i = 0; i < m; i++) {
  std::cout << f[i] << " \n"[i + 1 == m];
}
```

7.7 Factorial Poly

```
namespace FP {
Poly toFp(Poly a) { // Factorial Polynomial
  int n = a.size();
  Poly p(n);
  std::iota(p.begin(), p.end(), 0);
  Poly b = PolyEI::SegTree(p).multiEval(a);
  for (int i = 0; i < n; i++) p[i] = i & 1 ? P - ifac[i] :
  ↪ ifac[i];
  for (int i = 0; i < n; i++) b[i] = 1LL * b[i] * ifac[i] % P;
  p = p * b;
  p.resize(n);
  return p;
}
Poly toOp(Poly a) {
  int m = a.size();
  Poly q = a * Poly(ifac, ifac + m);
  q.resize(m);
  for (int i = 0; i < m; i++) {
    q[i] = 1LL * q[i] * (m - i & 1 ? ifac[m - i - 1] : P -
    ↪ ifac[m - i - 1]) % P;
  }
  Poly p(m);
  std::iota(p.begin(), p.end(), 0);
  PolyEI::SegTree t(p);
  int n = norm(m);
  q.resize(n);
  for (int k = 1, h = n >> 1; k < n; k <<= 1, h >>= 1)
    for (int i = 0, o = h; i < n; i += k << 1, o++) {
      if (i >= m) continue;
      int *a = &q[i], *b = &q[i + k];
      int const *l = t.p[o << 1].data(), *r = t.p[o << 1 |
      ↪ 1].data();
      Poly foo(k << 1), bar(k << 1);
      for (int j = 0; j < k; j++) foo[j] = a[j];
      for (int j = 0; j < k; j++) bar[j] = b[j];
      dft(foo), dft(bar);
      for (int j = 0; j < k << 1; j++)
        foo[j] = (1LL * foo[j] * r[j] + 1LL * bar[j] * l[j]) %
        ↪ P;
      idft(foo);
      std::copy(foo.begin(), foo.end(), a);
    }
  q.resize(m);
  std::reverse(q.begin(), q.end());
  return q;
}
Poly fpMul(Poly a, Poly b) {
  int n = a.size() + b.size() - 1;
  Poly p(ifac, ifac + n);
  a = a * p, a.resize(n);
  b = b * p, b.resize(n);
  for (int i = 0; i < n; i++) a[i] = 1LL * a[i] * b[i] % P *
  ↪ fac[i] % P;
  for (int i = 1; i < n; i += 2) p[i] = P - p[i];
  a = a * p;
  a.resize(n);
  return a;
}
}  // namespace FP
```

7.8 Stirling

```
namespace Stirling {
Poly getS1(int n) {
  if (n == 0) return {1};
  Poly p = getS1(n / 2);
  p = p * shift(p, p.size() - 1);
  if (n & 1) p = (p << 1) + p * (n - 1);
  return p;
}
Poly getS2(int n) {
  Poly a(n + 1), b(n + 1);
  for (int i = 0; i <= n; i++) {
    a[i] = i & 1 ? P - ifac[i] : ifac[i];
    b[i] = 1LL * fpow(i, n) * ifac[i] % P;
  }
  a = a * b;
  a.resize(n + 1);
  return a;
}
Poly getS1(int n, int k) { // S1(i, k), i = 0...n
  Poly p(n + 1);
  for (int i = 1; i <= n; i++) {
    p[i] = inv[i];
  }
  p = power(p, k);
  for (int i = 0; i <= n; i++) {
    p[i] = 1LL * p[i] * fac[i] % P * ifac[k] % P;
  }
  return p;
}
Poly getS2(int n, int k) { // S2(i, k), i = 0...n
  if (n < k) return Poly(n + 1);
  std::function<Poly(int)> cal = [&](int n) -> Poly {
    if (n == 0) return {1};
    Poly p = cal(n / 2);
    p = p * shift(p, (P + 1 - (int)p.size()) % P);
    if (n & 1) p = (p << 1) - p * n;
    return p;
  };
  Poly p = cal(k);
  std::reverse(p.begin(), p.end());
  p.resize(n - k + 1);
  return inverse(p) << k;
}
}  // namespace Stirling
```

7.9 Ferrers 棋盘

对于 Ferrers 棋盘 $S = F(a_1, a_2, \cdots, a_n)$ $(a_1 \le a_2 \le \cdots \le a_n)$,

$$\sum_{k=0}^{n} r_k(S) x^{\underline{n-k}} = \prod_{k=1}^{n} (x + a_k - k + 1)$$

$r_k(S)$ 是指放 k 个车相互攻击不到的方案数。

Various (8)

8.1 最长公共子序列

```cpp
constexpr int N(70005), B(63);
int tot;
struct Bitset {
  uint64_t a[N / B + 1];
  void set(int p) { a[p / B] |= 1ULL << p % B; }
  int count() {
    int s = 0;
    for (int i = 0; i < tot; i++)
      s += __builtin_popcountll(a[i]);
    return s;
  }
  void run(const Bitset &o) {
    uint64_t c = 1;
    for (int i = 0; i < tot; ++i) {
      auto x = a[i], y = x | o.a[i];
      x += x + c + (~y & (1ULL << B) - 1);
      a[i] = x & y, c = x >> B;
    }
  }
} dp, f[N];
void solve() {
  int n, m;
  std::cin >> n >> m;
  for (int i = 0, x; i < n; ++i) {
    std::cin >> x;
    f[x].set(i);
  }
  tot = (n - 1) / 63 + 1;
  for (int i = 0, x; i < m; ++i) {
    std::cin >> x, dp.run(f[x]);
  }
  std::cout << dp.count() << "\n";
}
```

8.2 模意义真分数还原

$q \equiv \dfrac{x}{a} \pmod p$，$|a| \le m$

```cpp
PII approx(int p, int q, int m) {
  int x = q, y = p, a = 1, b = 0;
  while (x > m) {
    std::swap(x, y);
    std::swap(a, b);
    a -= x / y * b, x %= y;
  }
  return {x, a};
}
```

8.3 杂

向上取整整除分块 $[i, \lfloor \dfrac{n-1}{\lceil \frac{n}{i} \rceil - 1} \rfloor]$

n 个点 k 个连通块的生成树方案 $n^{k-2}\prod_{i=1}^{k} siz_i$

杜教筛 $g(1)S(n) = \sum_{i=1}^{n}(f*g)(i) - \sum_{j=2}^{n} g(j)S(\lfloor \frac{n}{j} \rfloor)$

(x,y) 曼哈顿距离 $\to (x+y, x-y)$ 切比雪夫距离 (x,y) 切比雪夫距离 $\to (\dfrac{x+y}{2}, \dfrac{x-y}{2})$ 曼哈顿距离

错排数 $= \lceil 0.5 + \frac{n!}{e} \rceil$

$\ln(1-x^V) = -\sum_{i\ge 1} \dfrac{x^{Vi}}{i}$

$x^{\bar{n}} = \sum_i S_1(n,i)x^i$

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

m_i 为不同的质数。设 $M = \prod_{i=1}^{n} m_i$，$t_i \times \dfrac{M}{m_i} \equiv 1 \pmod{m_i}$，则 $x \equiv \sum_{i=1}^{n} a_i t_i \dfrac{M}{m_i}$。

$V - E + F = 2$，$S = n + \frac{s}{2} - 1$。(n 为内部，s 为边上)

用途：对于相邻的不相等的值，在中间画一条线（最外也画），连通块个数 $= 1 + E - V +$ 内部框个数.

注意全都是不含矩形边界上的。

π^{-1} 最小时 π 最小，π 最大等价于 π^{-1} 最大?

五边形数 GF：$\frac{x(2x+1)}{(1-x)^3}$

五边形数：$\frac{3n^2-n}{2}$，广义含非正，逆为分拆数 GF（注意系数正负和 n 取值奇偶性相同）

贝尔数（划分集合方案数）EGF：$\exp(e^x-1)$，$B_n = \sum_{i=0}^{n} S_2(n,i)$，伯努利数 EGF：$\dfrac{x}{e^x-1}$

$S_1(i,m)$ EGF：$\dfrac{(\sum_{i\ge 0}\frac{x^i}{i})^m}{m!}$，$S_2(i,m)$ EGF：$\dfrac{(e^x-1)^m}{m!}$

多项式牛顿迭代：如果已知 $G(F(x)) \equiv 0 \pmod{x^{2n}}$，$G(F_*(x)) \equiv 0 \pmod{x^n}$，则有 $F(x) \equiv F_*(x) - \frac{G(F_*(x))}{G'(F_*(x))} \pmod{x^{2n}}$。求导时孤立的多项式视为常数。

$\int_0^1 t^a(1-t)^b \mathrm{d}t = \dfrac{a!b!}{(a+b+1)!}$，$\sum_{i=0}^{n-1} i^k = \dfrac{n^{k+1}}{k+1}$

Appendix (9)

9.1 NTT Primes

p	r	k	g	p	r	k	g	p	r	k	g
3	1	1	2	23068673	11	21	3	6597069766657	3	41	5
5	1	2	2	104857601	25	22	3	39582418599937	9	42	5
17	1	4	3	167772161	5	25	3	79164837199873	9	43	5
97	3	5	5	469762049	7	26	3	263882790666241	15	44	7
193	3	6	5	1004535809	479	21	3	1231453023109120	35	45	3
257	1	8	3	2013265921	15	27	31	1337006139375620	19	46	3
7681	15	9	17	2281701377	17	27	3	3799912185593860	27	47	5
12289	3	12	11	3221225473	3	30	5	4222124650659840	15	48	19
40961	5	13	3	75161927681	35	31	3	7881299347898370	7	50	6
65537	1	16	3	77309411329	9	33	7	31525197391593500	7	52	3
786433	3	18	10	206158430209	3	36	22	180143985094820000	5	55	6
5767169	11	19	3	2061584302081	15	37	7	1945555039024050000	27	56	5
7340033	7	20	3	2748779069441	5	39	3	4179340454199820000	29	57	3

9.2 D and Omega

n	n 前第一个质数	n 后第一个质数	$\max\{\omega(n)\}$	$\max\{d(n)\}$
10^1	10^1-3	10^1+1	2	4
10^2	10^2-3	10^2+1	3	12
10^3	10^3-3	10^3+13	4	32
10^4	10^4-27	10^4+7	5	64
10^5	10^5-9	10^5+3	6	128
10^6	10^6-17	10^6+3	7	240
10^7	10^7-9	10^7+19	8	448
10^8	10^8-11	10^8+7	8	768
10^9	10^9-63	10^9+7	9	1344
10^{10}	$10^{10}-33$	$10^{10}+19$	10	2304
10^{11}	$10^{11}-23$	$10^{11}+3$	10	4032
10^{12}	$10^{12}-11$	$10^{12}+39$	11	6720
10^{13}	$10^{13}-29$	$10^{13}+37$	12	10752
10^{14}	$10^{14}-27$	$10^{14}+31$	12	17280
10^{15}	$10^{15}-11$	$10^{15}+37$	13	26880
10^{16}	$10^{16}-63$	$10^{16}+61$	13	41472
10^{17}	$10^{17}-3$	$10^{17}+3$	14	64512
10^{18}	$10^{18}-11$	$10^{18}+3$	15	103680
10^{19}	$10^{19}-39$	$10^{19}+51$	16	161280