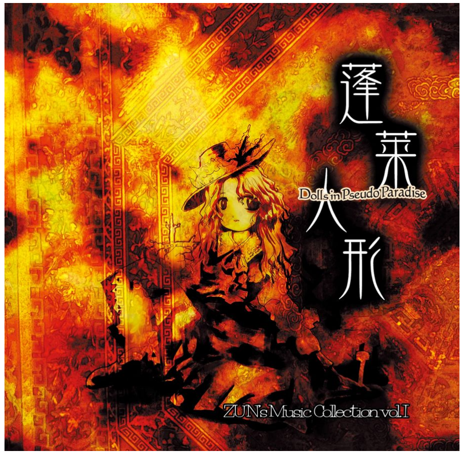
Our village of honest men originally consisted of only eight people.
We all picked up and moved to a mountain in the east. Two years of honest and boring daily life passed us by.
One day, one of us found a little hole by a peach tree.
Yes, after that we wandered into this paradise.
And right away, I quit being human.

— *Dolls in Pseudo Paradise*

蓬莱人形算法模板库

Reference Document for *Dolls in Pseudo Paradise*



Coach		Contestant		
赵文博	邱思宇	丁文涛	张彭凯	陈煜航
Wenbo Zhao	Siyu Qiu	Wentao Ding	Pengkai Zhang	Yuhang Chen

Harbin Institute of Technology

2024-2025

目录

1 动态规划

2 数据结构

2.1 平衡树

2.1.1 无旋 Treap

```cpp
#include "../../header.cpp"
mt19937_64 MT(114514);
namespace Treap{
  const int SIZ = 1e6 + 1e5 + 3;
  int F[SIZ], C[SIZ], S[SIZ], W[SIZ], X[SIZ
    ][2], sz;
  u64 H[SIZ];
  int  newnode(int w){
    W[++ sz] = w, C[sz] = S[sz] = 1, H[sz] =
      MT();
    return sz;
  }
  void pushup(int x){
    S[x] = C[x] + S[X[x][0]] + S[X[x][1]];
```

```cpp
  }
  pair<int, int> split(int u, int x){
    if(u == 0)
      return make_pair(0, 0);
    if(W[u] > x){
      auto [a, b] = split(X[u][0], x);
      X[u][0] = b, pushup(u);
      return make_pair(a, u);
    } else {
      auto [a, b] = split(X[u][1], x);
      X[u][1] = a, pushup(u);
      return make_pair(u, b);
    }
  }
  int merge(int a, int b){
    if(a == 0 || b == 0)
      return a | b;
    if(H[a] < H[b]){
      X[a][1] = merge(X[a][1], b), pushup(a);
      return a;
    } else {
      X[b][0] = merge(a, X[b][0]), pushup(b);
      return b;
    }
  }
  void insert(int &root, int w){
    auto [p, q] = split(root, w);
    auto [a, b] = split(p, w - 1);
    if(b != 0){
      ++ S[b], ++ C[b];
    } else b = newnode(w);
    p = merge(a, b), root = merge(p, q);
  }
  void erase(int &root, int w){
    auto [p, q] = split(root, w);
    auto [a, b] = split(p, w - 1);
    -- C[b], -- S[b];
    p  = C[b] == 0 ? a : merge(a, b);
    root = merge(p, q);
  }
  int  find_rank(int &root, int w){
    int x = root, o = x, a = 0;
    for(;x;){
      if(w <  W[x])
        o = x, x = X[x][0];
      else {
        a += S[X[x][0]];
        if(w == W[x]){ o = x; break; }
        a += C[x];
        o = x, x = X[x][1];
      }
    }
    return a + 1;
  }
  int  find_kth(int &root, int w){
    int x = root, o = x, a = 0;
    for(;x;){
      if(w <= S[X[x][0]])
        o = x, x = X[x][0];
      else {
        w -= S[X[x][0]];
        if(w <= C[x]){ o = x; break; }
        w -= C[x];
        o = x, x = X[x][1];
      }
    }
    return W[x];
  }
  int  find_pre(int &root, int w){
return find_kth(root, find_rank(root, w) - 1);
  }
  int  find_suc(int &root, int w){
return find_kth(root, find_rank(root, w + 1));
  }
}
```

2.2 珂朵莉树

```cpp
#include "../header.cpp"
namespace ODT {
  // <pos_type, value_type>
  map <int, long long> M;
  // 分裂为 [1, p) [p, +inf)，返回后者迭代器
  auto split(int p) {
    auto it = prev(M.upper_bound(p));
    return M.insert(
      it,
      make_pair(p, it -> second)
    );
  }
  // 区间赋值
  void assign(int l, int r, int v) {
    auto it = split(l);
    split(r + 1);
    while (it -> first != r + 1) {
      it = M.erase(it);
    }
    M[l] = v;
  }
  // // 执行操作
  // void perform(int l, int r) {
  //   auto it = split(l);
  //   split(r + 1);
  //   while (it -> first != r + 1) {
  //     // Do something ...
  //     it = next(it);
  //   }
  // }
};
```

2.3 可并堆

```cpp
#include "../header.cpp"
namespace LeftHeap{
  const int SIZ = 1e5 + 3;
  int W[SIZ],D[SIZ], L[SIZ], R[SIZ], F[SIZ],s;
  bool E[SIZ];
  int merge(int u, int v){
    if(u == 0 || v == 0)
      return u | v;
    if(W[u] > W[v] || (W[u] == W[v] && u > v))
      swap(u, v);
    int &lc = L[u], &rc = R[u];
    rc = merge(rc, v);
    if(D[lc] < D[rc]) swap(lc, rc);
    D[u] = min(D[lc], D[rc]) + 1;
    if(lc != 0) F[lc] = u;
    if(rc != 0) F[rc] = u;
    return u;
  }
  void pop(int &root){
    int root0 = merge(L[root], R[root]);
    F[root0] = F[root] = root0;
    E[root] = true, root = root0;
  }
  int top(int &root){ return W[root]; }
  int getfa(int u){
    return u == F[u] ? u : F[u] = getfa(F[u]);
  }
  int newnode(int w){
    ++ s, W[s] = w, F[s] = s, D[s] = 1;
    return s;
  }
}
```

2.4 KD 树

```cpp
#include "../header.cpp"
const int LG = 18;  // 二进制分组合并
struct pt { // x: 坐标; v: 权值; l, r: 儿子
  int x[2], v, sum, l, r, L[2], R[2];
} t[MAXN], l, h;    // l, h: 查询的左上/右下角
int rt[LG], b[MAXN], cnt;
```

```cpp
void update(int p) {
  t[p].sum = t[t[p].l].sum + t[t[p].r].sum + t[p].v;
  for (int k: {0, 1}) {
    t[p].L[k] = t[p].R[k] = t[p].x[k];
    if(t[p].l)
t[p].L[k] = min(t[p].L[k], t[t[p].l].L[k]),
t[p].R[k] = max(t[p].R[k], t[t[p].l].R[k]);
    if(t[p].r)
t[p].L[k] = min(t[p].L[k], t[t[p].r].L[k]),
t[p].R[k] = max(t[p].R[k], t[t[p].r].R[k]);
  }
}
int build(int l, int r, int dep = 0){ // 重构
  int p{(l + r) >> 1};
  nth_element(b + l, b + p, b + r + 1, [dep](int x, int y) { return t[x].x[dep] < t[y].x[dep]; });
  int x{b[p]};
  if(l < p) t[x].l = build(l, p - 1, dep ^ 1);
  if(p < r) t[x].r = build(p + 1, r, dep ^ 1);
  update(x);
  return x;
}
void append(int &p) {   // 收集 p 子树等待重构
  if(!p) return;
  b[++cnt] = p;
  append(t[p].l), append(t[p].r), p = 0;
}
int query(int p) {       // 查询 p 子树矩阵内和
  if (!p) return 0;
  bool flag = false;
  for (int k : {0, 1}) flag |= (!(l.x[k] <= t[p].L[k] && t[p].R[k] <= h.x[k]));
  if (!flag) return t[p].sum;
  for (int k : {0, 1})
    if (t[p].R[k] < l.x[k] || h.x[k] < t[p].L[k]) return 0;
  int ans = 0;
  flag = false;
  for (int k : {0, 1}) flag |= (!(l.x[k] <= t[p].x[k] && t[p].x[k] <= h.x[k]));
  if (!flag) ans = t[p].v;
  return ans += query(t[p].l) + query(t[p].r);
}
int main() {
  int n; cin >> n; n = 0; // n: 当前 KDT 大小
  while (true) {
    int op; cin >> op;
    if (op == 1) {
      int x, y, A; cin >> x >> y >> A;
      t[++ n] = {{x, y}, A};
      b[cnt = 1] = n;
      for (int sz = 0;; ++sz) // 二进制合并
        if (!rt[sz]) {
          rt[sz] = build(1, cnt); break;
        } else append(rt[sz]);
    } else if (op == 2) {
      cin >> l.x[0] >> l.x[1] >> h.x[0] >> h.x[1];
      int ans = 0;
      for(int i = 0; i < LG; ++i) ans += query(rt[i]);
      cout << ans << "\n";
    } else break;
  }
  return 0;
}
```

2.5 Link Cut 树

```cpp
#include "../header.cpp"
namespace LinkCutTree{
  const int SIZ = 1e5 + 3;
  int F[SIZ], C[SIZ], S[SIZ], W[SIZ], A[SIZ], X[SIZ][2], size;
  bool T[SIZ];
  bool is_root(int x){ return X[F[x]][0] != x && X[F[x]][1] != x;}
  bool is_rson(int x){ return X[F[x]][1] == x;}
  int  new_node(int w){ // 创建节点, 返回编号
    ++ size;
    W[size] = w, C[size] = S[size] = 1;
    A[size] = w, F[size] = 0;
    X[size][0] = X[size][1] = 0;
    return size;
  }
  void push_up(int x){
    S[x] = C[x] + S[X[x][0]] + S[X[x][1]];
    A[x] = W[x] ^ A[X[x][0]] ^ A[X[x][1]];
  }
  void push_down(int x){
    if(!T[x]) return;
    int lc = X[x][0], rc = X[x][1];
    if(lc)T[lc] ^= 1, swap(X[lc][0],X[lc][1]);
    if(rc)T[rc] ^= 1, swap(X[rc][0],X[rc][1]);
    T[x] = false;
  }
  void update(int x){
    if(!is_root(x))update(F[x]); push_down(x);
  }
  void rotate(int x){
    int y = F[x], z = F[y];
    bool f = is_rson(x);
    bool g = is_rson(y);
    if(is_root(y)){
      F[x] = z, F[y] = x;
      X[y][ f] = X[x][!f], F[X[x][!f]] = y;
      X[x][!f] = y;
    } else {
      F[x] = z, F[y] = x;
      X[z][ g] = x;
      X[y][ f] = X[x][!f], F[X[x][!f]] = y;
      X[x][!f] = y;
    }
    push_up(y), push_up(x);
  }
  void splay(int x){  // 旋到树根
    update(x);
    for(int f = F[x];f = F[x], !is_root(x);rotate(x))
      if(!is_root(f)) rotate(is_rson(x) == is_rson(f) ? f : x);
  }
  int  access(int x){ // 将根到 x 变成实链
    int p;
    for(p = 0;x;p = x, x = F[x]){
      splay(x), X[x][1] = p, push_up(x);
    }
    return p;
  }
  void make_root(int x){  // 使 x 为所在树的根
    x = access(x);
    T[x] ^= 1, swap(X[x][0], X[x][1]);
  }
  int find_root(int x){   // 查找 x 所在树的根
    access(x), splay(x), push_down(x);
    while(X[x][0]) x = X[x][0], push_down(x);
    splay(x);
    return x;
  }
  void link(int x, int y){  // 连边
    make_root(x), splay(x), F[x] = y;
  }
  void cut(int x, int p){   // 删边
    make_root(x), access(p), splay(p), X[p][0] = F[x] = 0;
  }
  void modify(int x, int w){// 修改点权
    splay(x), W[x] = w, push_up(x);
  }
}
```

2.6 线段树

2.6.1 李超树

```cpp
#include "../../header.cpp"
struct Line{ int id; double k, b; Line() = default;};
namespace LCSeg{
  const int SIZ = 2e5 + 3;
  struct Line T[SIZ];
  #define lc(t) (t << 1)
  #define rc(t) (t << 1 | 1)
  bool cmp(int p, Line x, Line y){
    double w1 = x.k * p + x.b;
    double w2 = y.k * p + y.b;
    double d = w1 - w2;
    if(fabs(d) < 1e-8) return x.id > y.id;
    return d < 0;
  }
  void merge(int t, int a, int b, Line x, Line y){
    int c = a + b >> 1;
    if(cmp(c, x, y)) swap(x, y);
    if(cmp(a, y, x)){
      T[t] = x; if(a != b) merge(rc(t), c + 1, b, T[rc(t)], y);
    } else {
      T[t] = x; if(a != b) merge(lc(t), a, c, T[lc(t)], y);
    }
  }
  // 插入线段 (l, f(l)) -- (r, f(r))
  void modify(int t, int a, int b, int l, int r, Line x){
    if(l <= a && b <= r) merge(t, a, b, T[t], x);
    else {
      int c = a + b >> 1;
      if(l <= c) modify(lc(t), a, c, l, r, x);
      if(r >  c) modify(rc(t), c + 1, b, l, r, x);
    }
  }
  // 查询 x = p 位置最高的线段（有多条取编号最小）
  void query(int t, int a, int b, int p, Line &x){
    if(cmp(p, x, T[t])) x = T[t];
    if(a != b){
      int c = a + b >> 1;
      if(p <= c) query(lc(t), a,     c, p, x);
      if(p >  c) query(rc(t), c + 1, b, p, x);
    }
  }
}
```

2.6.2 线段树 3

例题 给出一个长度为 n 的数列 A，同时定义一个辅助数组 B，B 开始与 A 完全相同。接下来进行了 m 次操作，操作有五种类型，按以下格式给出：

- `1 l r k`：对于所有的 $i \in [l, r]$，将 A_i 加上 k（k 可以为负数）。
- `2 l r v`：对于所有的 $i \in [l, r]$，将 A_i 变成 $\min(A_i, v)$。
- `3 l r`：求 $\sum_{i=l}^{r} A_i$。
- `4 l r`：对于所有的 $i \in [l, r]$，求 A_i 的最大值。
- `5 l r`：对于所有的 $i \in [l, r]$，求 B_i 的最大值。

在每一次操作后，我们都进行一次更新，让 $B_i \leftarrow \max(B_i, A_i)$。

```cpp
#include "../../header.cpp"
int A[MAXN];
struct Node{
  i64 sum; int len, max1, max2, max_cnt, his_mx;
  Node():
    sum(0), max1(-INF), max2(-INF), max_cnt(0), his_mx(-INF), len(0) {}
  Node(int w):
    sum(w), max1(   w), max2(-INF), max_cnt(1), his_mx(   w), len(1) {}
  bool update(int w1, int w2, int h1, int h2){
    his_mx = max({his_mx, max1 + h1});
    max1 += w1, max2 += w2;
    sum += 1ll * w1 * max_cnt + 1ll * w2 * (len - max_cnt);
    return max1 > max2;
  }
};
struct Tag{
  int max_add, max_his_add, umx_add, umx_his_add; bool have;
  void update(int w1, int w2, int h1, int h2){
    max_his_add = max(max_his_add, max_add + h1);
    umx_his_add = max(umx_his_add, umx_add + h2);
    max_add += w1, umx_add += w2, have = true;
  }
  void clear(){
    max_add = max_his_add = umx_add = umx_his_add = have = 0;
  }
};
struct Node operator +(Node a, Node b){
  Node t;
  t.max1 = max(a.max1, b.max1);
  if(t.max1 != a.max1){
    if(a.max1 > t.max2) t.max2 = a.max1;
  } else{
    if(a.max2 > t.max2) t.max2 = a.max2;
    t.max_cnt += a.max_cnt;
  }
  if(t.max1 != b.max1){
    if(b.max1 > t.max2) t.max2 = b.max1;
  } else{
    if(b.max2 > t.max2) t.max2 = b.max2;
    t.max_cnt += b.max_cnt;
  }
  t.sum = a.sum + b.sum, t.len = a.len + b.len;
  t.his_mx = max(a.his_mx, b.his_mx);
  return t;
}
namespace Seg{
  const int SIZ = 2e6 + 3;
  struct Node W[SIZ]; struct Tag T[SIZ];
  #define lc(t) (t << 1)
  #define rc(t) (t << 1 | 1)
  void push_up(int t, int a, int b){
    W[t] = W[lc(t)] + W[rc(t)];
  }
  void push_down(int t, int a, int b){
    if(a == b) T[t].clear();
    if(T[t].have){
      int c = a + b >> 1, x = lc(t), y = rc(t);
      int w = max(W[x].max1, W[y].max1);
      int w1 = T[t].max_add, w2 = T[t].umx_add, w3 = T[t].max_his_add, w4 = T[t].umx_his_add;
      if(w == W[x].max1)
        W[x].update(w1, w2, w3, w4),
        T[x].update(w1, w2, w3, w4);
      else
        W[x].update(w2, w2, w4, w4),
        T[x].update(w2, w2, w4, w4);
      if(w == W[y].max1)
```

```cpp
      W[y].update(w1, w2, w3, w4),
      T[y].update(w1, w2, w3, w4);
    else
      W[y].update(w2, w2, w4, w4),
      T[y].update(w2, w2, w4, w4);
    T[t].clear();
  }
}
void build(int t, int a, int b){
  if(a == b){W[t] = Node(A[a]), T[t].clear();} else {
    int c = a + b >> 1; T[t].clear();
    build(lc(t), a,   c);
    build(rc(t), c + 1, b);
    push_up(t, a, b);
  }
}
void modiadd(int t, int a, int b, int l, int r, int w){
  if(l <= a && b <= r){
    T[t].update(w, w, w, w);
    W[t].update(w, w, w, w);
  } else {
    int c = a + b >> 1; push_down(t, a, b);
    if(l <= c) modiadd(lc(t), a,   c, l, r, w);
    if(r >  c) modiadd(rc(t), c + 1, b, l, r, w);
    push_up(t, a, b);
  }
}
void modimin(int t, int a, int b, int l, int r, int w){
  if(l <= a && b <= r){
    if(w >= W[t].max1) return; else
    if(w >  W[t].max2){
      int k = w - W[t].max1;
      T[t].update(k, 0, k, 0);
      W[t].update(k, 0, k, 0);
    } else {
      int c = a + b >> 1;
      push_down(t, a, b);
      modimin(lc(t), a,   c, l, r, w);
      modimin(rc(t), c + 1, b, l, r, w);
      push_up(t, a, b);
    }
  } else {
    int c = a + b >> 1; push_down(t, a, b);
    if(l <= c) modimin(lc(t), a,   c, l, r, w);
    if(r >  c) modimin(rc(t), c + 1, b, l, r, w);
    push_up(t, a, b);
  }
}
Node query(int t, int a, int b, int l, int r){
  if(l <= a && b <= r) return W[t];
  int c = a + b >> 1; Node ret; push_down(t, a, b);
  if(l <= c) ret = ret + query(lc(t), a,   c, l, r);
  if(r >  c) ret = ret + query(rc(t), c + 1, b, l, r);
  return ret;
}
}
int qread();
int main(){
  int n = qread(), m = qread();
  for(int i = 1;i <= n;++ i)
    A[i] = qread();
  Seg :: build(1, 1, n);
  for(int i = 1;i <= m;++ i){
    int op = qread();
    if(op == 1){
      int l = qread(), r = qread(), w = qread();
      Seg :: modiadd(1, 1, n, l, r, w);
    } else if(op == 2){
      int l = qread(), r = qread(), w = qread();
      Seg :: modimin(1, 1, n, l, r, w);
    } else if(op == 3){
      int l = qread(), r = qread();
      auto p = Seg :: query(1, 1, n, l, r);
      printf("%lld\n", p.sum);
    } else if(op == 4){
      int l = qread(), r = qread();
      auto p = Seg :: query(1, 1, n, l, r);
      printf("%d\n", p.max1);
    } else if(op == 5){
      int l = qread(), r = qread();
      auto p = Seg :: query(1, 1, n, l, r);
      printf("%d\n", p.his_mx);
    }
  }
  return 0;
}
```

2.7 根号数据结构

3 树论

3.1 点分树

3.1.1 例题

给定 n 个点组成的树，点有点权 v_i。m 个操作，分为两种：

- `0 x k` 查询距离 x 不超过 k 的所有点的点权之和；
- `0 x y` 将点 x 的点权修改为 y。

```cpp
#include "../header.cpp"
vector<int> E[MAXN];
namespace LCA{
  const int MAXH = 18 + 3;
  int D[MAXN], F[MAXN];
  int P[MAXN], Q[MAXN], o, h = 18;
  void dfs(int u, int f){
    ++ o;
    P[u] = o, Q[o] = u;
    F[u] = f, D[u] = D[f] + 1;
    for(auto &v : E[u]) if(v != f)
      dfs(v, u);
  }
  int ST[MAXN][MAXH];
  int cmp(int a, int b){
    return D[a] < D[b] ? a : b;
  }
  int T[MAXN], n;
  void init(int _n);  // 初始化 ST 表
  int lca(int a, int b){
    if(a == b) return a;
    int l = P[a], r = P[b];
    if(l > r) swap(l, r);
    ++ l;
    int d = T[r - l + 1];
    return F[cmp(ST[l][d], ST[r - (1 << d) + 1][d])];
  }
  int dis(int a, int b);
}
namespace BIT{
  void add(int D[], int n, int p, int w){
    ++ p;
    while(p <= n) D[p] += w, p += p & -p;
  }
  int pre(int D[], int n, int p){
    if(p < 0) return 0;
```

```cpp
    p = min(n, p + 1);
    int r = 0;
    while(p >  0) r += D[p], p -= p & -p;
    return r;
  }
}
namespace PTree{
  vector<int> EE[MAXN];
  bool V[MAXN];
  int S[MAXN], L[MAXN], *D1[MAXN], *D2[MAXN];
  using LCA :: dis, BIT :: add, BIT :: pre;
  void dfs1(int s, int &g, int u, int f){
    S[u] = 1;
    int maxsize = 0;
    for(auto &v : E[u]) if(v != f && !V[v]){
      dfs1(s, g, v, u);
      maxsize = max(maxsize, S[v]);
      S[u] += S[v];
    }
    maxsize = max(maxsize, s - S[u]);
    if(maxsize <= s / 2) g = u;
  }
  int n;
  void build(int s, int &g, int u, int f){
    dfs1(s, g, u, f);
    V[g] = true, L[g] = s;
    for(auto &u : E[g]) if(!V[u]){
      int h = 0;
      if(S[u] < S[g]) build(S[u], h, u, 0);
      else            build(s - S[g], h, u, 0);
      EE[g].push_back(h);
      EE[h].push_back(g);
    }
  }
  int F[MAXN];
  void dfs2(int u, int f){
    F[u] = f;
    for(auto &v : EE[u]) if(v != f){
      dfs2(v, u);
    }
  }
  void build(int _n){   // 建树（需初始化 LCA）
    n = _n;
    int s = n, g = 0;
    dfs1(s, g, 1, 0);
    V[g] = true, L[g] = s;
    for(auto &u : E[g]){
      int h = 0;
      if(S[u] < S[g]) build(S[u], h, u, 0);
      else build(s - S[g], h, u, 0);
      EE[g].push_back(h);
      EE[h].push_back(g);
    }
    dfs2(g, 0);
    for(int i = 1;i <= n;++ i){
      L[i] += 2;
      D1[i] = new int[L[i] + 3];
      D2[i] = new int[L[i] + 3];
      for(int j = 0;j < L[i] + 3;++ j)
        D1[i][j] = D2[i][j] = 0;
    }
  }
  void modify(int x, int w){  // 修改点权
    int u = x;
    while(1){
      add(D1[x], L[x], dis(u, x), w);
      int y = F[x];
      if(y != 0){
        int e = LCA :: dis(x, y);
        add(D2[x], L[x], dis(u, y), w);
        x = y;
      } else break;
    }
  }
  int query(int x, int d){
    int ans = 0, u = x;
    while(1){
      ans += pre(D1[x], L[x], d - dis(u, x));
      int y = F[x];
      if(y != 0){
        int e = dis(x, y);
        ans-= pre(D2[x], L[x], d - dis(u, y));
        x = y;
      } else break;
    }
    return ans;
  }
}
```

3.2 树哈希

3.2.1 用法

有根树求出每个子树的哈希值，儿子间顺序可交换。

```cpp
#include "../header.cpp"
u64 xor_shift(u64 x);
u64 H[MAXN];
vector <int> E[MAXN];
void dfs(int u, int f){
  H[u] = 1;
  for(auto &v: E[u]) if(v != f)
    dfs(v, u), H[u] += H[v];
  H[u] = xor_shift(H[u]); // !important
}
```

3.3 Prufer 序列

```cpp
#include "../header.cpp"
int D[MAXN], F[MAXN], P[MAXN];
vector<int> tree2prufer(int n){
  vector <int> P(n);
  for(int i = 1, j = 1;i <= n - 2;++ i, ++ j){
    while(D[j]) ++ j;
    P[i] = F[j];
    while(i<= n - 2 && !--D[P[i]] && P[i] < j)
      P[i + 1] = F[P[i]], i ++;
  }
  return P;
}
vector<int> prufer2tree(int n){
  vector <int> F(n);
  for(int i = 1, j = 1;i <= n - 1;++ i, ++ j){
    while(D[j]) ++ j;
    F[j] = P[i];
    while(i<= n - 1 && !--D[P[i]] && P[i] < j)
      F[P[i]] = P[i + 1], i ++;
  }
  return F;
}
```

3.4 虚树

```cpp
#include "../header.cpp"
vector<pair<int, int> > E[MAXN];
namespace LCA{
  const int SIZ = 5e5 + 3;
  int D[SIZ], H[SIZ], F[SIZ], P[SIZ], Q[SIZ],
    o;
  void dfs(int u, int f){
    P[u] = ++ o, Q[o] = u, F[u] = f, D[u] = D[
      f] + 1;
    for(auto &[v, w] : E[u]) if(v != f){
      H[v] = H[u] + w, dfs(v, u);
    }
  }
  const int MAXH = 18 + 3;
  int h = 18;
  int ST[SIZ][MAXH];
  int cmp(int a, int b){
    return D[a] < D[b] ? a : b;
  }
  int T[SIZ], n;
```

```cpp
  void init(int _n, int root);
  int lca(int a, int b);
  int dis(int a, int b);
}
bool cmp(int a, int b){
  return LCA :: P[a] < LCA :: P[b];
}
bool I[MAXN];
vector <int> E1[MAXN], V1;
void solve(vector <int> &V){
  using LCA :: lca; using LCA :: D;
  stack <int> S;
  sort(V.begin(), V.end(), cmp);
  S.push(1);
  int v, l;
  for(auto &u : V) I[u] = true;
  for(auto &u : V) if(u != 1){
    int f = lca(u, S.top());
    l = -1;
    while(D[v = S.top()] > D[f]){
      if(l != -1)
        E1[v].push_back(l);
      V1.push_back(l = v), S.pop();
    }
    if(l != -1)
      E1[f].push_back(l);
    if(f != S.top()) S.push(f);
    S.push(u);
  }
  l = -1;
  while(!S.empty()){
    v = S.top();
    if(l != -1) E1[v].push_back(l);
    V1.push_back(l = v), S.pop();
  }
  // dfs(1, 0); // SOLVE HERE !!!
  for(auto &u : V1)
    E1[u].clear(), I[u] = false;
  V1.clear();
}
```

4 图论

4.1 三元环计数

无向图：考虑将所有点按度数从小往大排序，然后将每条边定向，由排在前面的指向排在后面的，得到一个有向图。然后考虑枚举一个点，再枚举一个点，暴力数，具体见代码。结论是，这样定向后，每个点的出度是 $O(\sqrt{m})$ 的。复杂度 $O(m\sqrt{m})$。有向图：不难发现，上述方法枚举了三个点，计算有向图三元环也就只需要处理下方向的事，这个由于算法够暴力，随便改改就能做了。

```cpp
#include "../header.cpp"
// 无向图
ll solve1(){
    ll n, m; cin >> n >> m;
    vector<pair<ll, ll>> Edges(m);
    vector<vector<ll>> G(n + 2);
    vector<ll> deg(n + 2);
    for (auto &[i, j] : Edges)
        cin >> i >> j, ++deg[i], ++deg[j];
    for (auto [i, j] : Edges) {
        if (deg[i] > deg[j] || (deg[i] == deg[j] && i > j)) swap(i, j);
        G[i].emplace_back(j);
    }
    vector<ll> val(n + 2);
    ll ans = 0;
    for (ll i = 1; i <= n; ++i) {
        for (auto j : G[i]) ++val[j];
        for (auto j : G[i])
      for (auto k : G[j]) ans += val[k];
        for (auto j : G[i]) val[j] = 0;
    }
    return ans;
}
// 有向图
ll solve2(){
    ll n, m; cin >> n >> m;
    vector<pair<ll, ll>> Edges(m);
    vector<vector<pair<ll, ll>>> G(n + 2);
    vector<ll> deg(n + 2);
    for (auto &[i, j] : Edges)
    cin >> i >> j, ++deg[i], ++deg[j];
    for (auto [i, j] : Edges) {
        ll flg = 0;
        if (deg[i] > deg[j] || (deg[i] == deg[j] && i > j)) swap(i, j), flg = 1;
        G[i].emplace_back(j, flg);
    }
    vector<ll> in(n + 2), out(n + 2);
    ll ans = 0;
    for (ll i = 1; i <= n; ++i) {
        for(auto [j, w] : G[i])
      w ? (++in[j]) : (++out[j]);
        for(auto [j, w1] : G[i])
      for (auto [k, w2] : G[j]) {
        if (w1 == w2)ans += w1 ? in[k] : out[k];
        }
        for(auto [j, w] : G[i])in[j] = out[j] = 0;
    }
    return ans;
}
```

4.2 四元环计数

- 无向图：类似，由于定向后出度结论过于强大，可以暴力。讨论了三种情况。
- 有向图：缺少题目，但应当类似三元环计数有向形式记录定向边和原边的正反关系。因为此法最强的结论是定向后出度 $O(\sqrt{m})$，实际上方法很暴力，应当不难数有向形式的。

```cpp
#include "../header.cpp"
ll solve(){
  ll n, m; cin >> n >> m;
  vector<pair<ll, ll>> Edges(m);
  vector<vector<ll>> G(n + 2), iG(n + 2);
  vector<ll> deg(n + 2);
  for (auto &[i, j] : Edges)
    cin >> i >> j, ++deg[i], ++deg[j];
  for (auto [i, j] : Edges) {
    if (deg[i] > deg[j] || (deg[i] == deg[j] && i > j)) swap(i, j);
    G[i].emplace_back(j), iG[j].emplace_back(i);
  }
  ll ans = 0;
  vector<ll> v1(n + 2), v2(n + 2);
  for (ll i = 1; i <= n; ++i) {
    for (auto j : G[i])
      for (auto k : G[j]) ++v1[k];
    for (auto j : iG[i])
      for (auto k : G[j])
        ans += v1[k], ++v2[k];
    for (auto j : G[i])
      for (auto k : G[j])
        ans += v1[k] * (v1[k] - 1) / 2, v1[k] = 0;
    for (auto j : iG[i]) for (auto k : G[j]) {
      if (deg[k] > deg[i] || (deg[k] == deg[i] && k > i)) ans += v2[k] * (v2[k] - 1) / 2;
      v2[k] = 0;
    }
  }
```

```
  return ans;
}
```

4.3 支配树

```
// From Alex_Wei
#include "../header.cpp"
int n, m, dn, F[MAXN], ind[MAXN], dfn[MAXN];
int sdom[MAXN], idom[MAXN], sz[MAXN];
vector<int> E[MAXN], rE[MAXN], buc[MAXN];
void dfs(int u, int f) {
  ind[dfn[u] = ++dn] = u, F[u] = f;
  for(auto &v: E[u]) if(!dfn[v]) dfs(v, u);
}
struct dsu {
  // M 维护 sdom 最小的点的编号
  int F[MAXN], M[MAXN];
  int find(int x) {
    if(F[x] == x) return F[x];
    int f = F[x];
    F[x] = find(f);
    if(sdom[M[f]] < sdom[M[x]]) M[x] = M[f];
    return F[x];
  }
  int get(int x) { return find(x), M[x]; }
} tr;
int main() {
  cin >> n >> m;
  for(int i = 1; i <= m; i++) {
    int u, v; cin >> u >> v;
    E[u].push_back(v), rE[v].push_back(u);
  }
  dfs(1, 0), sdom[0] = n + 1;
  for(int i = 1; i <= n; i++) tr.F[i] = i;
  for(int i = n; i; -- i){
    int u = ind[i];
    for(auto &v: buc[i]) idom[v] = tr.get(v);
    if(i == 1) break;
    sdom[u] = i;
    for(auto &v: rE[u]) {
      sdom[u] = min(sdom[u], dfn[v] <= i ? dfn
        [v] : sdom[tr.get(v)]);
    }
    tr.M[u] = u, tr.F[u] = F[u];
    buc[sdom[u]].push_back(u);
  }
  for(int i = 2; i <= n; i++) {
    int u = ind[i];
    if(sdom[idom[u]] != sdom[u])
      idom[u] = idom[idom[u]];
    else idom[u] = sdom[u];
  }
  for(int i = n;i;i --){
    sz[i] += 1;
    if(i > 1) sz[ind[idom[i]]] += sz[i];
  }
  for(int i = 1;i <= n;++ i){
    cout << sz[i] << " \n"[i == n];
  }
  return 0;
}
```

4.4 二分图最大匹配

```
#include "../header.cpp"
vector <int> G[MAXN];
bool V[MAXN];
int ML[MAXN], MR[MAXN];
bool kuhn(int u){
    V[u] = true;
    for(auto &v: G[u]) if(MR[v] == 0){
        ML[u] = v, MR[v] = u;
        return true;
    }
    for(auto &v: G[u]) if(!V[MR[v]] && kuhn(MR
      [v])){
        ML[u] = v, MR[v] = u;
        return true;
    }
    return false;
}
void solve(int L, int R){
    for(int i = 1;i <= L;++ i){
        shuffle(G[i].begin(), G[i].end(), MT);
    } // 需要打乱避免构造
    while(1){
        bool ok = false;
        memset(V, false, sizeof(V));
        for(int i = 1;i <= L;++ i)
            ok |= !ML[i] && kuhn(i);
        if(!ok) break;
    }
}
```

4.5 一般图最大匹配

```
#include "../header.cpp"
int n;
vector <int> E[MAXN];
queue  <int> Q;
int vis[MAXN], F[MAXN], col[MAXN], pre[MAXN],
  mat[MAXN], tmp;
int getfa(int x){
  return x == F[x] ? x : F[x] = getfa(F[x]);
}
int lca(int x, int y){
  for(++ tmp;;x = pre[mat[x]], swap(x, y))
    if(vis[x = getfa(x)] == tmp) return x;
      else vis[x] = x ? tmp : 0;
}
void flower(int x, int y, int z){
  while(getfa(x) != z){
    pre[x] = y, y = mat[x], F[x] = F[y] = z;
    x = pre[y];
    if(col[y] == 2)
      Q.push(y), col[y] = 1;
  }
}
bool aug(int u){
  for(int i = 1;i <= n;++ i)
    col[i] = pre[i] = 0, F[i] = i;
  Q = queue<int>({ u }), col[u] = 1;
  while(!Q.empty()){
    auto x = Q.front(); Q.pop();
    for(auto &v: E[x]){
      int y = v, z;
      if(col[y] == 2) continue;
      if(col[y] == 1) {
        z = lca(x, y);
        flower(x, y, z), flower(y, x, z);
      } else
      if(!mat[y]){
        for(pre[y] = x; y;){
          mat[y] = x = pre[y], swap(y,mat[x]);
        }
        return true;
      } else {
        pre[y] = x, col[y] = 2;
        Q.push(mat[y]), col[mat[y]] = 1;
      }
    }
  }
  return false;
}
```

4.6 2-SAT

4.6.1 例题

n 个变量 m 个条件，形如若 $x_i = a$ 则 $y_j = b$，找到任意一组可行解或者报告无解。

```
#include "tarjan-scc.cpp"
const int MAXN = 1e6 + 3;
```

```cpp
int X[MAXN][2], o;
int main(){
  ios :: sync_with_stdio(false);
  int n, m;
  cin >> n >> m;
  for(int i = 1;i <= n;++ i)
    X[i][0] = ++ o, X[i][1] = ++ o;
  for(int i = 1;i <= m;++ i){
    int a, x, b, y;
    cin >> a >> x >> b >> y;
    SCC :: add(X[a][!x], X[b][y]);
    SCC :: add(X[b][!y], X[a][x]);
  }
  for(int i = 1;i <= o;++ i)
    if(!SCC :: F[i])
      SCC :: dfs(i);
  bool ok = true;
  for(int i = 1;i <= n;++ i){
    if(SCC :: C[X[i][0]] == SCC :: C[X[i][1]])
      ok = false;
  }
  if(ok){
    cout << "POSSIBLE" << endl;
    for(int i = 1;i <= n;++ i){
      int a = SCC :: C[X[i][0]];
      int b = SCC :: C[X[i][1]];
      cout << (a >= b) << " ";
    }
    cout << endl;
  } else {
    cout << "IMPOSSIBLE" << endl;
  }
  return 0;
}
```

4.7 割点

```cpp
#include "../header.cpp"
vector<int> V[MAXN];
int n, m, o, D[MAXN], L[MAXN];
bool F[MAXN], C[MAXN];
// 对每个连通块调用 dfs(i, i)
void dfs(int u, int g){
  L[u] = D[u] = ++ o, F[u] = true; int s = 0;
  for(auto &v : V[u]){
    if(!F[v]){
      dfs(v, g), ++ s;
      L[u] = min(L[u], L[v]);
      if(u != g && L[v] >= D[u]) C[u] = true;
    } else {
      L[u] = min(L[u], D[v]);
    }
  }
  // C[u] 为真表示该点是割点
  if(u == g && s > 1) C[u] = true;
}
```

4.8 边双连通分量

```cpp
#include "../header.cpp"
vector <vector<int>> A;
vector <pair<int, int>> V[MAXN];
stack  <int> S;
int D[MAXN], L[MAXN], o; bool I[MAXN];
void dfs(int u, int l){
  D[u] = L[u] = ++ o; I[u] = true, S.push(u);
  int s = 0;
  for(auto &[v, g] : V[u]) if(g != l) {
    if(D[v]){
      if(I[v])    L[u] = min(L[u], D[v]);
    } else {
      dfs(v, g), L[u] = min(L[u], L[v]), ++ s;
    }
  }
  if(D[u] == L[u]){
    vector <int> T;
    while(S.top() != u){
      int v = S.top(); S.pop();
      T.push_back(v), I[v] = false;
    }
    T.push_back(u), S.pop(), I[u] = false;
    A.push_back(T);
  }
}
```

4.9 点双连通分量

```cpp
#include "../header.cpp"
vector <vector<int>> A;
vector <int> V[MAXN];
stack  <int> S;
int D[MAXN], L[MAXN], o; bool I[MAXN];
void dfs(int u, int f){
  D[u] = L[u] = ++ o; I[u] = true, S.push(u);
  int s = 0;
  for(auto &v : V[u]) if(v != f){
    if(D[v]){
      if(I[v])   L[u] = min(L[u], D[v]);
    } else {
      dfs(v, u), L[u] = min(L[u], L[v]), ++ s;
      if(L[v] >= D[u]){
        vector <int> T;
        while(S.top() != v){
          int t = S.top(); S.pop();
          T.push_back(t), I[t] = false;
        }
        T.push_back(v), S.pop(), I[v] = false;
        T.push_back(u);
        A.push_back(T);
      }
    }
  }
  if(f == 0 && s == 0)
    A.push_back({u}); // 孤立点特判
}
```

4.10 强连通分量

```cpp
#include "../header.cpp"
namespace SCC {
  vector <int> V[MAXN];
  stack  <int> S;
  int D[MAXN], L[MAXN], C[MAXN], o, s;
  bool F[MAXN], I[MAXN];
  void add(int u, int v){ V[u].push_back(v); }
  void dfs(int u){
    L[u] = D[u] = ++ o, S.push(u), I[u] = F[u] = true;
    for(auto &v : V[u]){
      if(F[v]){
        if(I[v]) L[u] = min(L[u], D[v]);
      } else {
        dfs(v),  L[u] = min(L[u], L[v]);
      }
    }
    if(L[u] == D[u]){
      int c = ++ s;
      while(S.top() != u){
        int v = S.top(); S.pop();
        I[v] = false;
        C[v] = c;
      }
      S.pop(), I[u] = false, C[u] = c;
    }
  }
}
```

5 网络流

5.1 费用流

```cpp
#include "../header.cpp"
namespace MCMF{
```

```cpp
  int H[MAXN], V[MAXM], N[MAXM], W[MAXM], F[MAXM], o = 1, n;
  void add(int u, int v, int f, int c){
    V[++ o] = v, N[o] = H[u], H[u] = o, F[o] = f, W[o] =  c;
    V[++ o] = u, N[o] = H[v], H[v] = o, F[o] = 0, W[o] = -c;
    n = max({n, u, v});
  }
  void clear(){
    for(int i = 1;i <= n;++ i) H[i] = 0;
    n = 0, o = 1;
  }
  bool I[MAXN]; i64 D[MAXN];
  bool spfa(int s, int t){
    queue <int> Q;
    Q.push(s), I[s] = true;
    for(int i = 1;i <= n;++ i)
      D[i] = INFL;
    D[s] = 0;
    while(!Q.empty()){
      int u = Q.front();Q.pop(), I[u] = false;
      for(int i = H[u];i;i = N[i]){
        int &v = V[i], &f = F[i], &w = W[i];
        if(f && D[u] + w < D[v]){
          D[v] = D[u] + w;
          if(!I[v]) Q.push(v), I[v] = true;
        }
      }
    }
    return D[t] != INFL;
  }
  int C[MAXN]; bool T[MAXN];
  pair<i64, i64> dfs(int s, int t, int u, i64 maxf){
    if(u == t)
      return make_pair(maxf, 0);
    i64 totf = 0, totc = 0;
    T[u] = true;
    for(int &i = C[u];i;i = N[i]){
      int &v = V[i], &f = F[i], &w = W[i];
      if(f && D[v] == D[u] + w && !T[v]){
        auto [f, c] = dfs(s, t, v, min(1ll * F[i], maxf));
        F[i  ] -= f, F[i ^ 1] += f;
        totf += f, maxf -= f;
        totc += 1ll * f * W[i] + c;
        if(maxf == 0){
          T[u] = false;
          return make_pair(totf, totc);
        }
      }
    }
    T[u] = false;
    return make_pair(totf, totc);
  }
  pair<i64, i64> mcmf(int s, int t){
    i64 ans1 = 0, ans2 = 0;
    while(spfa(s, t)){
      memcpy(C, H, sizeof(int) * (n + 3));
      auto [f, c] = dfs(s, t, s, INFL);
      ans1 += f, ans2 += c;
    }
    return make_pair(ans1, ans2);
  }
}
```

5.2 最小割树

5.2.1 用法

给定无向图求出最小割树，点 u 和 v 作为起点终点的最小割为树上 u 到 v 路径上边权的最小值。

```cpp
#include "../header.cpp"
namespace Dinic{
  const i64 INF = 1e18;
  const int SIZ = 1e5 + 3;
  int n, m;
  int H[SIZ], V[SIZ], N[SIZ], F[SIZ], t = 1;
  int add(int u, int v, int f){
    V[++ t] = v, N[t] = H[u], F[t] = f, H[u] = t;
    V[++ t] = u, N[t] = H[v], F[t] = 0, H[v] = t;
    n = max(n, u);
    n = max(n, v);
    return t - 1;
  }
  void clear(){
    for(int i = 1;i <= n;++ i) H[i] = 0;
    n = m = 0, t = 1;
  }
  int D[SIZ];
  bool bfs(int s, int t){
    queue <int> Q;
    for(int i = 1;i <= n;++ i) D[i] = 0;
    Q.push(s), D[s] = 1;
    while(!Q.empty()){
      int u = Q.front(); Q.pop();
      for(int i = H[u];i;i = N[i]){
        const int &v = V[i], &f = F[i];
        if(f != 0 && !D[v])
          D[v] = D[u] + 1, Q.push(v);
      }
    }
    return D[t] != 0;
  }
  int C[SIZ];
  i64 dfs(int s, int t, int u, i64 maxf){
    if(u == t)
      return maxf;
    i64 totf = 0;
    for(int &i = C[u];i;i = N[i]){
      const int &v = V[i];
      const int &f = F[i];
      if(D[v] == D[u] + 1){
        i64 ff = dfs(s, t, v, min(maxf, 1ll * f));
        totf += ff, maxf -= ff;
        F[i] -= ff, F[i ^ 1] += ff;
        if(maxf == 0) return totf;
      }
    }
    return totf;
  }
  i64 dinic(int s, int t){
    i64 ans = 0;
    while(bfs(s, t)){
      memcpy(C, H, sizeof(int) * (n + 3));
      ans += dfs(s, t, s, INF);
    }
    return ans;
  }
}
namespace GHTree{
  const int INF  = 1e9;
  int n, m, U[MAXM], V[MAXM], W[MAXM], A[MAXM], B[MAXM];
  void add(int u, int v, int w){
    ++ m;
    U[m] = u, V[m] = v, W[m] = w;
    A[m] = Dinic :: add(u, v, w);
    B[m] = Dinic :: add(v, u, w);
    n = max({n, u, v});
  }
  vector <pair<int, int> > E[MAXN];
  void build(vector <int> N){
    int s = N.front(), t = N.back();
    if(s == t) return;
    for(int i = 1;i <= m;++ i){
      int a = A[i]; Dinic :: F[a] = W[i], Dinic :: F[a ^ 1] = 0;
      int b = B[i]; Dinic :: F[b] = W[i],
```

```cpp
        Dinic :: F[b ^ 1] = 0;
    }
    int w = Dinic :: dinic(s, t);
    E[s].push_back(make_pair(t, w));
    E[t].push_back(make_pair(s, w));
    vector <int> P, Q;
    for(auto &u : N){
      if(Dinic :: D[u] != 0)
        P.push_back(u);
      else
        Q.push_back(u);
    }
    build(P), build(Q);
  }
  int D[MAXN];
  int cut(int s, int t){
    queue <int> Q; Q.push(s);
    for(int i = 1;i <= n;++ i)
      D[i] = -1;
    D[s] = INF;
    while(!Q.empty()){
      int u = Q.front(); Q.pop();
      for(auto &[v, w] : E[u]){
        if(D[v] == -1){
          D[v] = min(D[u], w);
          Q.push(v);
        }
      }
    }
    return D[t];
  }
}
```

5.3 最大流

```cpp
#include "../header.cpp"
namespace Dinic{
  const i64 INF = 1e18;
  int n, H[MAXN], V[MAXM], N[MAXM], F[MAXM], t = 1;
  void add(int u, int v, int f){
    V[++ t] = v, N[t] = H[u], F[t] = f, H[u] = t;
    V[++ t] = u, N[t] = H[v], F[t] = 0, H[v] = t;
    n = max({n, u, v});
  }
  void clear(){
    for(int i = 1;i <= n;++ i)
      H[i] = 0;
    n = 0, t = 1;
  }
  i64 D[MAXN];
  bool bfs(int s, int t){
    queue <int> Q;
    for(int i = 1;i <= n;++ i) D[i] = 0;
    Q.push(s), D[s] = 1;
    while(!Q.empty()){
      int u = Q.front(); Q.pop();
      for(int i = H[u];i;i = N[i]){
        const int &v = V[i], &f = F[i];
        if(f != 0 && !D[v])
          D[v] = D[u] + 1, Q.push(v);
      }
    }
    return D[t] != 0;
  }
  int C[MAXN];
  i64 dfs(int s, int t, int u, i64 maxf){
    if(u == t) return maxf;
    i64 totf = 0;
    for(int &i = C[u];i;i = N[i]){
      const int &v = V[i], &f = F[i];
      if(f && D[v] == D[u] + 1){
        i64 f = dfs(s, t, v, min(1ll * f, maxf));
        F[i] -= f, F[i ^ 1] += f, totf += f, maxf -= f;
        if(maxf == 0)
          return totf;
      }
    }
    return totf;
  }
  i64 dinic(int s, int t){
    i64 ans = 0;
    while(bfs(s, t)){
      memcpy(C, H, sizeof(int) * (n + 3));
      ans += dfs(s, t, s, INFL);
    }
    return ans;
  }
}
```

5.4 上下界费用流

5.4.1 用法

- `add(u, v, l, r, c)`：连一条容量在 $[l, r]$ 的从 u 到 v 的费用为 c 的边；
- `solve()`：计算无源汇最小费用可行流；
- `solve(s, t)`：计算有源汇最小费用最大流。

```cpp
#define add add0
#include "flow-cost.cpp"
#undef add
namespace MCMF{
  i64 cost0; int G[MAXN];
  void add(int u, int v, int l, int r, int c){
    G[v] += l, G[u] -= l;
    cost0 += 1ll * l * c;
    add0(u, v, r - l, c);
  }
  i64 solve(){
    int s = ++ n, t = ++ n;
    i64 sum = 0;
    for(int i = 1;i <= n - 2;++ i){
      if(G[i] < 0)
        add0(i, t, -G[i], 0);
      else
        add0(s, i,  G[i], 0), sum += G[i];
    }
    auto res = mcmf(s, t);
    if(res.first != sum)
      return -1;
    return res.second + cost0;
  }
  i64 solve(int s0, int t0){
    add0(t0, s0, INF, 0);
    int s = ++ n;
    int t = ++ n;
    i64 sum = 0;
    for(int i = 1;i <= n - 2;++ i){
      if(G[i] < 0)
        add0(i, t, -G[i], 0);
      else
        add0(s, i,  G[i], 0), sum += G[i];
    }
    auto res = mcmf(s, t);
    if(res.first != sum)
      return -1;
    return res.second + cost0;
  }
}
```

5.5 上下界最大流

5.5.1 用法

- `add(u, v, l, r, c)`：连一条容量在 $[l, r]$ 的从 u 到 v 的边；
- `solve()`：检查是否存在无源汇可行流；

- `solve(s, t)`：计算有源汇最大流。

```cpp
#define add add0
#include "flow-max.cpp"
#undef add
namespace Dinic{
  int G[MAXN];
  void add(int u, int v, int l, int r){
    G[v] += l, G[u] -= l;
    add0(u, v, r - l);
  }
  void clear(){
    for(int i = 1;i <= t;++ i){
      N[i] = F[i] = V[i] = 0;
    }
    for(int i = 1;i <= n;++ i){
      H[i] = G[i] = C[i] = 0;
    }
    t = 1, n = 0;
  }
  bool solve(){ // 为真表示有解
    int s = ++ n, t = ++ n;
    i64 sum = 0;
    for(int i = 1;i <= n - 2;++ i){
      if(G[i] < 0)
        add0(i, t, -G[i]);
      else
        add0(s, i,  G[i]), sum += G[i];
    }
    return sum != dinic(s, t);
  }
  i64 solve(int s0, int t0){
    add0(t0, s0, INF);
    int s = ++ n, t = ++ n;
    i64 sum = 0;
    for(int i = 1;i <= n - 2;++ i){
      if(G[i] < 0)
        add0(i, t, -G[i]);
      else
        add0(s, i,  G[i]), sum += G[i];
    }
    return dinic(s, t) == sum ? dinic(s0, t0) : -1;
  }
}
```

6 数学

6.1 线性代数

6.1.1 行列式

```cpp
#include "../../header.cpp"
struct Mat{
  int n, m, W[MAXN][MAXN];
  Mat(int _n = 0, int _m = 0){
    n = _n, m = _m;
    for(int i = 1;i <= n;++ i)
      for(int j = 1;j <= m;++ j)
        W[i][j] = 0;
  }
};
int mat_det(Mat a){
  int ans = 1;
  const int &n = a.n;
  for(int i = 1;i <= n;++ i){
    int f = -1;
    for(int j = i;j <= n;++ j) if(a.W[j][i]){
      f = j; break;
    }
    if(f == -1) return 0;
    if(f != i){
      for(int j = 1;j <= n;++ j)
        swap(a.W[i][j], a.W[f][j]);
      ans = MOD - ans;
    }
    for(int j = i + 1;j <= n;++ j) if(a.W[j][i
      ]){
      while(a.W[j][i]){
        int u = a.W[i][i], v = a.W[j][i];
        if(u > v){
          for(int k = 1;k <= n;++ k)
            swap(a.W[i][k], a.W[j][k]);
          ans = MOD - ans, swap(u, v);
        }
        int rate = v / u;
        for(int k = 1;k <= n;++ k){
          a.W[j][k] = (a.W[j][k] - 1ll * rate
            * a.W[i][k] % MOD + MOD) % MOD;
        }
      }
    }
  }
  for(int i = 1;i <= n;++ i)
    ans = 1ll * ans * a.W[i][i] % MOD;
  return ans;
}
```

6.2 大步小步

6.2.1 用法

给定 a, p 求出 x 使得 $a^x = y \pmod p$，其中 p 为质数。

```cpp
#include "../header.cpp"
namespace BSGS {
  unordered_map <int, int> M;
  int solve(int a, int y, int p){
    M.clear();
    int B = sqrt(p);
    int w1 = y, u1 = power(a, p - 2, p);
    int w2 = 1, u2 = power(a, B, p);
    for(int i = 0;i < B;++ i){
      M[w1] = i;
      w1 = 1ll * w1 * u1 % p;
    }
    for(int i = 0;i < p / B;++ i){
      if(M.count(w2)){
        return i * B + M[w2];
      }
      w2 = 1ll * w2 * u2 % p;
    }
    return -1;
  } // a ^ x = y (mod p)
}
```

6.3 树图计数

6.3.1 LGV 定理叙述

设 G 是一张有向无环图，边带权，每个点的度数有限。给定起点集合 $A = \{a_1, a_2, \cdots, a_n\}$，终点集合 $B = \{b_1, b_2, \cdots, b_n\}$。

- 一段路径 $p : v_0 \to^{w_1} v_1 \to^{w_2} v_2 \to \cdots \to^{w_k} v_k$ 的权值：$\omega(p) = \prod w_i$。
- 一对顶点 (a, b) 的权值：$e(a, b) = \sum_{p:a\to b} \omega(p)$。

设矩阵 M 如下：

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

从 A 到 B 得到一个不相交的路径组 $p=(p_1,p_2,\cdots,p_n)$，其中从 a_i 到达 b_{π_i}，π 是一个排列。定义 $\sigma(\pi)$ 是 π 逆序对的数量。

给出 LGV 的叙述如下：

$$\det(M)=\sum_{p:A\to B}(-1)^{\sigma(\pi)}\prod_{i=1}^{n}\omega(p_i)$$

可以将边权视作边的重数，那么 $e(a,b)$ 就可以视为从 a 到 b 的不同路径方案数。

6.3.2 矩阵树定理

对于无向图，

- 定义度数矩阵 $D_{i,j}=[i=j]\deg(i)$；
- 定义邻接矩阵 $E_{i,j}=E_{j,i}$ 是从 i 到 j 的边数个数；
- 定义拉普拉斯矩阵 $L=D-E$。

对于无向图的矩阵树定理叙述如下：

$$t(G)=\det(L_i)=\frac{1}{n}\lambda_1\lambda_2\cdots\lambda_{n-1}$$

其中 L_i 是将 L 删去第 i 行和第 i 列得到的子式。

对于有向图，类似于无向图定义入度矩阵、出度矩阵、邻接矩阵 $D^{\text{in}},D^{\text{out}},E$，同时定义拉普拉斯矩阵 $L^{\text{in}}=D^{\text{in}}-E,L^{\text{out}}-E$。

$$t^{\text{leaf}}(G,k)=\det(L_k^{\text{in}})$$
$$t^{\text{root}}(G,k)=\det(L_k^{\text{out}})$$

其中 $t^{\text{leaf}}(G,k)$ 表示以 k 为根的叶向树，$t^{\text{root}}(G,k)$ 表示以 k 为根的根向树。

6.3.3 BEST 定理

对于一个有向欧拉图 G，记点 i 的出度为 out_i，同时 G 的根向生成树个数为 T。T 可以任意选取根。则 G 的本质不同的欧拉回路个数为：

$$T\prod_i(\text{out}_i-1)!$$

6.3.4 图连通方案数

n 个点的图，k 个连通块，第 i 个大小为 s_i，添加 $k-1$ 条边使得它们连通的方案数：

$$Ans=n^{k-2}\cdot\prod_{i=1}^{k}s_i$$

6.4 中国剩余定理

6.4.1 定理

对于线性方程：

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

如果 a_i 两两互质，可以得到 x 的解 $x\equiv L \pmod M$，其中 $M=\prod m_i$，而 L 由下式给出：

$$L=\left(\sum a_i m_i\times\left((M/m_i)^{-1} \bmod m_i\right)\right) \bmod M$$

```cpp
#include "../header.cpp"
i64 A[MAXN], B[MAXN], M = 1;
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y);
int main(){
  int n; cin >> n;
  for(int i = 1;i <= n;++ i){
    cin >> B[i] >> A[i], M *= B[i];
  }
  i64 L = 0;
  for(int i = 1;i <= n;++ i){
    i64 m = M / B[i], b, k;
    exgcd(m, B[i], b, k);
    L = (L + (__int128)A[i] * m * b) % M;
  }
  L = (L % M + M) % M;
  cout << L << endl;
  return 0;
}
```

6.5 狄利克雷前缀和

$$s(i)=\sum_{d|i}f_d$$

```cpp
#include "../header.cpp"
unsigned A[MAXN];
int p, P[MAXN]; bool V[MAXN];
void solve(int n){
  for(int i = 2;i <= n;++ i){
    if(!V[i]){
      P[++ p] = i;
      for(int j = 1;j <= n / i;++ j){
        A[j * i] += A[j];
      } // 前缀和
    }
    for(int j = 1;j <= p&&P[j] <= n / i;++ j){
      V[i * P[j]] = true;
      if(i % P[j] == 0) break;
    }
  }
}
```

6.6 万能欧几里得

6.6.1 类欧几里得（万能欧几里得）

一种神奇递归，对 $y=\left\lfloor\dfrac{Ax+B}{C}\right\rfloor$ 向右和向上走的每步进行压缩，做到 $O(\log V)$ 复杂度。其中 $A\ge C$ 就是直接压缩，向右之后必有至少 $\lfloor A/C\rfloor$ 步向上。$A<C$ 实际上切换 x,y 轴后，相当于压缩了一个上取整折线，而上取整下取整可以互化，便又可以递归。

代码中从 $(0,0)$ 走到 $(n,\lfloor(An+B)/C\rfloor)$，假设了 $A,B,C\ge 0,C\ne 0$（类欧基本都作此假设），U,R 矩阵是从右往左乘的，对列向量进行优化，和实际操作顺序恰好相反。快速幂的 log 据说可以被递归过程均摊掉，实际上并不会导致变成两个 log。

```cpp
Matrix solve(ll n, ll A, ll B, ll C, Matrix R, Matrix U) { // (0, 0) 走到 (n, (An+B)/C)
  if (A >= C) return solve(n, A % C, B, C, U.qpow(A / C) * R, U);
  ll l = B / C, r = (A * n + B) / C;
  if (l == r) return R.qpow(n) * U.qpow(l);
    // l = r -> l = r or A = 0 or n = 0.
  ll p = (C * r - B - 1) / A + 1;
  return R.qpow(n - p) * U * solve(r - l - 1, C, C - B % C + A - 1, A, U, R) * U.qpow(l);
}
```

6.7 扩展欧几里得

6.7.1 内容

给定 a, b，求出 $ax + by = \gcd(a, b)$ 的一组 x, y。

```cpp
int exgcd(int a, int b, int &x, int &y){
  if(a == 0){
    x = 0, y = 1; return b;
  } else {
    int x0 = 0, y0 = 0;
    int d = exgcd(b % a, a, x0, y0);
    x = y0 - (b / a) * x0, y = x0;
    return d;
  }
}
```

6.8 快速离散对数

6.8.1 用法

给定原根 g 以及模数 M，T 次询问 x 的离散对数。

复杂度 $\mathcal{O}(\mathrm{M}^{2/3} + T\log \mathrm{M})$。

```cpp
#include "../header.cpp"
namespace BSGS {
  unordered_map <int, int> M;
  int B, U, P, g;
  void init(int g, int P0, int B0);
  int solve(int y);
}
const int MAXN = 1e5 + 3;
int H[MAXN], P[MAXN], H0, p, h, g, M;
bool V[MAXN];
int solve(int x){
  if(x <= h) return H[x];
  int v = M / x, r = M % x;
  if(r < x - r) return ((H0 + solve(r)) % (M - 1) - H[v] + M - 1) % (M - 1);
  else          return (solve(x - r) - H[v + 1] + M - 1) % (M - 1);
}
int main(){
  ios :: sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> g >> M;
  h = sqrt(M) + 1;
  BSGS :: init(g, M, sqrt(1ll * M * sqrt(M) / log10(M)));
  H0 = BSGS :: solve(M - 1);
  H[1] = 0;
  for(int i = 2;i <= h;++ i){
    if(!V[i]){
      P[++ p] = i;
      H[i] = BSGS :: solve(i);
    }
    for(int j = 1;j <= p&&P[j] <= h / i;++ j){
      int &p = P[j];
      H[i * p] = (H[i] + H[p]) % (M - 1);
      V[i * p] = true;
      if(i % p == 0) break;
    }
  }
  int T; cin >> T;
  while(T --){
    int x; cin >> x;
    cout << solve(x) << "\n";
  }
  return 0;
}
```

6.9 原根

```cpp
#include "../header.cpp"
int getphi(int x);                 // 求解 phi
vector <int> getprime(int x);     // 求解质因数
bool test(int g, int m, int mm,vector<int>&P){
  for(auto &p: P)
    if(power(g, mm / p, m) == 1)
      return false;
  return true;
}
int get_genshin(int m){
  int mm = getphi(m);
  vector <int> P = getprime(mm);
  for(int i = 1;;++ i)
    if(test(i, m, mm, P)) return i;
}
```

6.10 拉格朗日插值

6.10.1 定理

给定 n 个横坐标不同的点 (x_i, y_i)，可以唯一确定一个 $n-1$ 阶多项式如下：

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

6.11 min-max 容斥

6.11.1 定理

$$\max_{i\in S}\{x_i\} = \sum_{T\subseteq S}(-1)^{|T|-1}\min_{j\in T}\{x_j\}$$

$$\min_{i\in S}\{x_i\} = \sum_{T\subseteq S}(-1)^{|T|-1}\max_{j\in T}\{x_j\}$$

期望意义下上式依然成立。

另外设 $\max^k$ 表示第 k 大的元素，可以推广为如下式子：

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

此外在数论上可以得到：

$$\operatorname{lcm}_{i\in S}\{x_i\} = \prod_{T\subseteq S}\left(\gcd_{j\in T}\{x_j\}\right)^{(-1)^{|T|-1}}$$

6.11.2 应用

对于计算 “n 个属性都出现的期望时间” 问题，设第 i 个属性第一次出现的时间是 t_i，所求即为 $\max(t_i)$，使用 min-max 容斥转为计算 $\min(t_i)$。

比如 n 个独立物品，每次抽中物品 i 的概率是 p_i，问期望抽多少次抽中所有物品。那么就可以计算 $\min_S$ 表示第一次抽中物品集合 S 内物品的时间，可以得到：

$$\max_U = \sum_{S\subseteq U}(-1)^{|S|-1}\min_S = \sum_{S\subseteq U}(-1)^{|S|-1}\cdot\frac{1}{\sum_{x\in S}p_x}$$

6.12 Barrett 取模

6.12.1 用法

调用 init 计算出 S 和 X，得到计算 $\lfloor x/P \rfloor = (x \times X)/2^{60+S}$。从而计算 $x \bmod P = x - P \times \lfloor x/P \rfloor$。

```cpp
#include "../header.cpp"
i64 S = 0, X = 0;
void init(int MOD){
  while((1 << (S + 1)) < MOD) S ++;
  X = ((i80)1 << 60 + S) / MOD + !!(((i80)1 << 60 + S) % MOD);
  cerr << S << " " << X << endl;
```

```cpp
}
int power(i64 x, int y, int MOD){
  i64 r = 1;
  while(y){
    if(y & 1){
      r = r * x;
      r = r - MOD * ((i80)r * X >> 60 + S);
    }
    x = x * x;
    x = x - MOD * ((i80)x * X >> 60 + S);
    y >>= 1;
  }
  return r;
}
```

6.13 Pollard's Rho

6.13.1 用法

- 调用 test(n) 判断 n 是否是质数;
- 调用 rho(n) 计算 n 分解质因数后的结果，不保证结果有序。

```cpp
#include "../header.cpp"
i64 step(i64 a, i64 c, i64 m){
  return ((i80)a * a + c) % m;
}
i64 multi(i64 a, i64 b, i64 m){
  return (i80) a * b % m;
}
i64 power(i64 a, i64 b, i64 m){
  i64 r = 1;
  while(b){
    if(b & 1) r = multi(r, a, m);
    b >>= 1,  a = multi(a, a, m);
  }
  return r;
}
mt19937_64 MT;
bool test(i64 n){
  if(n < 3 || n % 2 == 0) return n == 2;
  i64 u = n - 1, t = 0;
  while(u % 2 == 0) u /= 2, t += 1;
  int test_time = 20;
  for(int i = 1; i <= test_time;++ i){
    i64 a = MT() % (n - 2) + 2;
    i64 v = power(a, u, n);
    if(v == 1) continue;
    int s;
    for(s = 0;s < t;++ s){
      if(v == n - 1) break;
      v = multi(v, v, n);
    }
    if(s == t) return false;
  }
  return true;
}
basic_string<i64> rho(i64 n){
  if(n == 1)  return { };
  if(test(n)) return {n};
  i64 a  = MT() % (n - 1) + 1;
  i64 x1 = MT() % (n - 1), x2 = x1;
  for(int i = 1;;i <<= 1){
    i64 tot = 1;
    for(int j = 1;j <= i;++ j){
      x2 = step(x2, a, n);
      tot = multi(tot, llabs(x1 - x2), n);
      if(j % 127 == 0){
        i64 d = __gcd(tot, n);
        if(d > 1)
          return rho(d) + rho(n / d);
      }
    }
    i64 d = __gcd(tot, n);
    if(d > 1)
      return rho(d) + rho(n / d);
    x1 = x2;
  }
}
```

6.14 polya 定理

6.14.1 Burnside 引理

记所有染色方案的集合为 X, 其中单个染色方案为 x。一种对称操作 $g \in X$ 作用于染色方案 $x \in X$ 上可以得到另外一种染色 x'。

将所有对称操作作为集合 G, 那么 $Gx = \{gx \mid g \in G\}$ 是与 x 本质相同的染色方案的集合，形式化地称为 x 的轨道。统计本质不同染色方案数，就是统计不同轨道个数。

Burnside 引理说明如下:

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

其中 X^g 表示在 $g \in G$ 的作用下，不动点的集合。不动点被定义为 $x = gx$ 的 x。

6.14.2 Polya 定理

对于通常的染色问题，X 可以看作一个长度为 n 的序列，每个元素是 1 到 m 的整数。可以将 n 看作面数、m 看作颜色数。Polya 定理叙述如下:

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

其中 $c(g)$ 表示对一个序列做轮换操作 g 可以分解成多少个置换环。

然而，增加了限制（比如要求某种颜色必须要多少个），就无法直接应用 Polya 定理，需要利用 Burnside 引理进行具体问题具体分析。

6.14.3 应用

给定 n 个点 n 条边的环，现在有 n 种颜色，给每个顶点染色，询问有多少种本质不同的染色方案。

显然 X 是全体元素在 1 到 n 之间长度为 n 的序列，G 是所有可能的单次旋转方案，共有 n 种，第 i 种方案会把 1 置换到 i。于是:

$$\begin{aligned}
\text{ans} &= \frac{1}{|G|}\sum_{i=1}^{n} m^{c(g_i)}\\
&= \frac{1}{n}\sum_{i=1}^{n} n^{\gcd(i,n)}\\
&= \frac{1}{n}\sum_{d|n}^{n} n^d \sum_{i=1}^{n}[\gcd(i,n)=d]\\
&= \frac{1}{n}\sum_{d|n}^{n} n^d \varphi(n/d)
\end{aligned}$$

```cpp
#include "../header.cpp"
vector <tuple<int, int> > P;
void solve(int step, int n, int d, int f, int
  &ans){
  if(step == P.size()){
    ans = (ans + 1ll * power(n, n / d) * f) %
      MOD;
  } else {
    auto [w, c] = P[step];
    int dd = 1, ff = 1;
    for(int i = 0;i <= c;++ i){
      solve(step + 1, n, d * dd, f * ff, ans);
```

```
        ff = ff * (w - (i == 0)), dd = dd * w;
      }
    }
}
int main(){
  int T; cin >> T;
  while(T --){
    int n, t; cin >> n; t = n;
    for(int i = 2;i <= n / i;++ i) if(n % i == 0){
      int w = i, c = 0;
      while(t % i == 0) t /= i, c ++;
      P.push_back({ w, c });
    }
    if(t != 1) P.push_back({ t, 1 });
    int ans = 0;
    solve(0, n, 1, 1, ans);
    ans = 1ll * ans * power(n, MOD - 2) % MOD;
    cout << ans << endl;
    P.clear();
  }
  return 0;
}
```

6.15 min25 筛

设有一个积性函数 $f(n)$，满足 $f(p^k)$ 可以快速求，考虑搞一个在质数位置和 $f(n)$ 相等的 $g(n)$，满足它有完全积性，并且单点和前缀和都可以快速求，然后通过第一部分筛出 g 在质数位置的前缀和，从而相当于得到 f 在质数位置的前缀和，然后利用它，做第二部分，求出 f 的前缀和。

1. $G_k(n) = \sum_{i=1}^{n}[\mathrm{mindiv}(i) > p_k \text{ or isprime}(i)]g(i)$（$p_0 = 1$），则有 $G_k(n) = G_{k-1}(n) - g(p_k) \times (G_{k-1}(n/p_k) - G_{k-1}(p_{k-1}))$，复杂度 $O(n^{3/4}/\log n)$。
2. $F_k(n) = \sum_{i=1}^{n}[\mathrm{mindiv}(i) \geq p_k]f(i) = \sum[h \geq k, p_h^2 \leq n]\sum[c \geq 1, p_h^{c+1} \leq n](f(p_h^c)F_{h+1}(n/p_h^c) + f(p_h^{c+1})) + F_{\mathrm{prime}}(n) - F_{\mathrm{prime}}(p_{k-1})$，在 $n \leq 10^{13}$ 可以证明复杂度 $O(n^{3/4}/\log n)$。

常见细节问题：

- 由于 n 通常是 10^{10} 到 10^{11} 的数，导致 n 会爆 int，n^2 会爆 long long，而且往往会用自然数幂和，更容易爆，所以要小心。
- 记 $s = \lfloor\sqrt{n}\rfloor$，由于 F 递归时会去找 F_{h+1}，会访问到 s 以内最大的质数往后的一个质数，而已经证明对于所有 $n \in \mathbb{N}^+$，$[n+1, 2n]$ 中有至少一个质数，所以只需要筛到 $2s$ 即可。
- 注意补回 $f(1)$。

```
// 预处理，1 所在的块也算进去了
namespace init {
    ll init_n, sqrt_n;
    vector<ll> np, p, id1, id2, val;
    ll cnt;
    void main(ll n) {
        init_n = n, sqrt_n = sqrt(n);
        ll M = sqrt_n * 2; // 筛出一个 > floor(sqrt(n)) 的质数，避免后续讨论边界
        np.resize(M + 1), p.resize(M + 1);
        for (ll i = 2; i <= M; ++i) {
            if (!np[i]) p[++p[0]] = i;
            for (ll j = 1; j <= p[0]; ++j) {
                if (i * p[j] > M) break;
                np[i * p[j]] = 1;
                if (i % p[j] == 0) break;
            }
        }
        p[0] = 1;
        id1.resize(sqrt_n + 1), id2.resize(sqrt_n + 1);
        val.resize(1);
        for (ll l = 1, r, v; l <= n; l = r + 1) {
            v = n / l, r = n / v;
            if (v <= sqrt_n) id1[v] = ++cnt;
            else id2[init_n / v] = ++cnt;
            val.emplace_back(v);
        }
    }
    ll id(ll n) {
        if (n <= sqrt_n) return id1[n];
        else return id2[init_n / n];
    }
}
using namespace init;
// 计算 G_k，两个参数分别是 g 从 2 开始的前缀和和 g
auto calcG = [&] (auto&& sum, auto&& g) -> vector<ll> {
    vector<ll> G(cnt + 1);
    for (int i = 1; i <= cnt; ++i) G[i] = sum(val[i]);
    ll pre = 0;
    for (int i = 1; p[i] * p[i] <= n; ++i) {
        for (int j = 1; j <= cnt; ++j) {
            if (p[i] * p[i] > val[j]) break;
            ll tmp = id(val[j] / p[i]);
            G[j] = (G[j] - g(p[i]) * (G[tmp] - pre)) % MD;
        }
        pre = (pre + g(p[i])) % MD;
    }
    for (int i = 1; i <= cnt; ++i) G[i] = (G[i] % MD + MD) % MD;
    return G;
};
// 计算 F_k，直接搜，不用记忆化。`fp` 是 F_prime，`pc` 是 p^c，其中 `f(p[h] ^ c)` 要替换掉。
function<ll(ll, int)> calcF = [&] (ll m, int k) {
    if (p[k] > m) return 0;
    ll ans = (fp[id(m)] - fp[id(p[k - 1])]) % MD;
    for (int h = k; p[h] * p[h] <= m; ++h) {
        ll pc = p[h], c = 1;
        while (pc * p[h] <= m) {
            ans = (ans + calcF(m / pc, h + 1) * f(p[h] ^ c)) % MD;
            ++c, pc = pc * p[h], ans = (ans + f(p[h] ^ c)) % MD;
        }
    }
    return ans;
};
```

6.16 杜教筛

6.16.1 用法

对于积性函数 f，找到易求前缀和的积性函数 g, h 使得 $h = f * g$，根据递推式计算 $S(n) = \sum_{i=1}^{n} f(i)$:

$$S(n) = H(n) - \sum_{d=1}^{n} g(d) \times S(\left\lfloor\frac{n}{d}\right\rfloor)$$

6.16.2 例题

- 对于 $f = \varphi$，寻找 $g = 1, h = \mathrm{id}$；
- 对于 $f = \mu$，寻找 $g = 1, h = \varepsilon$。

6.17 二次剩余

6.17.1 用法

多次询问, 每次询问给定奇素数 p 以及 y, 在 $\mathcal{O}(\log p)$ 复杂度计算 x 使得 $x^2 \equiv 0 \pmod p$ 或者无解。

```cpp
#include "../header.cpp"
// 检查 x 在模 p 意义下是否有二次剩余
bool check(int x, int p){
  return power(x, (p - 1) / 2, p) == 1;
}
struct Node { int real, imag; };
Node mul(const Node a, const Node b, int p, int v){
  int nreal = (1ll * a.real * b.real
    + 1ll * a.imag * b.imag % p * v) % p;
  int nimag = (1ll * a.real * b.imag
    + 1ll * a.imag * b.real) % p;
  return { (nreal), nimag };
}
Node power(Node a, int b, int p, int v){
  Node r = { 1, 0 };
  while(b){
    if(b & 1) r = mul(r, a, p, v);
    b >>= 1,  a = mul(a, a, p, v);
  }
  return r;
}
mt19937 MT;
// 无解 x1 = x2 = -1, 唯一解 x1 = x2
void solve(int n, int p, int &x1, int &x2){
  if(n == 0){ x1 = x2 =  0; return; }
  if(!check(n, p)){ x1 = x2 = -1; return; }
  i64 a, t;
  do {
    a = MT() % p;
  }while(check(t = (a * a - n + p) % p, p));
  Node u = { a, 1 };
  x1 = power(u, (p + 1) / 2, p, t).real;
  x2 = (p - x1) % p;
  if(x1 > x2) swap(x1, x2);
}
```

6.18 单位根反演

6.18.1 定理

给出单位根反演如下:

$$[d \mid n] = \frac{1}{d}\sum_{i=0}^{d-1}\omega_d^{ni}$$

7 多项式

7.1 最小多项式

```cpp
#include "../header.cpp"
vector<int> bm(const vector<int> &a) {
  vector<int> v, ls;
  int k = -1, delta = 0;
  for (int i = 0; i < a.size();i ++) {
    int tmp = 0;
    for (int j = 0;j < v.size();j ++)
      tmp = (tmp + (ll)a[i - j - 1] * v[j]) % MD;
    if (a[i] == tmp) continue;
    if (k < 0) { k = i; delta = (a[i] - tmp + MD) % MD; v = vector<int>(i + 1);
      continue; }
    vector<int> u = v;
    int val = (ll)(a[i] - tmp + MD) * qpow(delta, MD - 2) % MD;
    v.resize(max(v.size(), ls.size()+ i - k));
    (v[i - k - 1] += val) %= MD;
    for (int j = 0; j < (int)ls.size(); j++){
      v[i - k + j] = (v[i - k + j] - (ll)val * ls[j]) % MD;
      if(v[i - k + j] < 0) v[i - k + j] += MD;
    }
    if (u.size() + k < ls.size() + i){
      ls = u; k = i, delta = a[i] - tmp;
      if (delta < 0) delta += MD;
    }
  }
  for (auto &x : v) x = (MD - x) % MD;
  v.insert(v.begin(), 1);
  return v;
} // ∀i, ∑_{j=0}^{m} a_{i-j} v_j = 0
```

7.2 NTT 全家桶

```cpp
#include "../header.cpp"
using Poly = vector<ll>;
#define lg(x) ((x) == 0 ? -1 : __lg(x))
#define Size(x) int(x.size())
namespace NTT_ns {
  const long long G = 3, invG = inv(G);
  vector<int> rev;
  void NTT(ll* F, int len, int sgn) {
    rev.resize(len);
    for (int i = 1; i < len; ++i) {
      rev[i] = (rev[i >> 1] >> 1) | ((i & 1) * (len >> 1));
      if (i < rev[i]) swap(F[i], F[rev[i]]);
    }
    for (int tmp = 1; tmp < len; tmp <<= 1) {
      ll w1 = qpow(sgn ? G : invG,
        (MD - 1) / (tmp << 1));
      for (int i = 0; i < len; i += tmp << 1){
        for(ll j = 0, w = 1; j < tmp; ++j,
          w = w * w1 % MD) {
          ll x = F[i + j];
          ll y = F[i + j + tmp] * w % MD;
          F[i + j] = (x + y) % MD;
          F[i + j + tmp] = (x - y + MD) % MD;
        }
      }
    }
    if (sgn == 0) {
      ll inv_len = inv(len);
      for (int i = 0; i < len; ++i)
        F[i] = F[i] * inv_len % MD;
    }
  }
}
using NTT_ns::NTT;
Poly operator * (Poly F, Poly G) {
  int siz = Size(F) + Size(G) - 1;
  int len = 1 << (lg(siz - 1) + 1);
  if (siz <= 300) {
    Poly H(siz);
    for (int i = Size(F) - 1; ~i; --i)
      for (int j = Size(G) - 1; ~j; --j)
        H[i + j] =(H[i + j] + F[i] * G[j])%MD;
    return H;
  }
  F.resize(len), G.resize(len);
  NTT(F.data(), len, 1),NTT(G.data(), len, 1);
  for (int i = 0; i < len; ++i)
    F[i] = F[i] * G[i] % MD;
  NTT(F.data(), len, 0), F.resize(siz);
  return F;
}
Poly operator + (Poly F, Poly G) {
  int siz = max(Size(F), Size(G));
  F.resize(siz), G.resize(siz);
  for (int i = 0; i < siz; ++i)
    F[i] = (F[i] + G[i]) % MD;
  return F;
}
Poly operator - (Poly F, Poly G) {
  int siz = max(Size(F), Size(G));
  F.resize(siz), G.resize(siz);
  for (int i = 0; i < siz; ++i)
```

```cpp
    F[i] = (F[i] - G[i] + MD) % MD;
  return F;
}
Poly lsh(Poly F, int k) {
  F.resize(Size(F) + k);
  for (int i = Size(F) - 1; i >= k; --i)
    F[i] = F[i - k];
  for (int i = 0; i < k; ++i) F[i] = 0;
  return F;
}
Poly rsh(Poly F, int k) {
  int siz = Size(F) - k;
  for (int i = 0; i < siz;++i)F[i] = F[i + k];
  return F.resize(siz), F;
}
Poly cut(Poly F, int len) {
  return F.resize(len), F;
}
Poly der(Poly F) {
  int siz = Size(F) - 1;
  for (int i = 0; i < siz; ++i)
    F[i] = F[i + 1] * (i + 1) % MD;
  return F.pop_back(), F;
}
Poly inte(Poly F) {
  F.emplace_back(0);
  for (int i = Size(F) - 1; ~i; --i)
    F[i] = F[i - 1] * inv(i) % MD;
  return F[0] = 0, F;
}
Poly inv(Poly F) {
  int siz = Size(F); Poly G{inv(F[0])};
  for (int i = 2; (i >> 1) < siz; i <<= 1) {
    G= G + G - G * G * cut(F, i), G.resize(i);
  }
  return G.resize(siz), G;
}
Poly ln(Poly F) {
  return cut(inte(cut(der(F) * inv(F), Size(F)
    )), Size(F));
}
Poly exp(Poly F) {
  int siz = Size(F); Poly G{1};
  for (int i = 2; (i >> 1) < siz; i <<= 1) {
    G = G * (Poly{1} - ln(cut(G, i)) + cut(F,
      i)), G.resize(i);
  }
  return G.resize(siz), G;
}
```

7.3 定系数短多项式卷积

有 n 个多项式 $F_i = \sum_{j=1}^{k} w_j x^{a_{i,j}}$，$(w_i)$ 序列固定，并且 k 很小，$a_i \in [0, 2^m)$，现在要把他们位运算卷积起来，求最终的序列。

异或版本，复杂度 $O(2^k((n+m)k + 2^m(m+\log V)))$，按通常大小关系可以认为是 $O(nk2^k + m2^{m+k})$，最后那个 $\log V$ 是快速幂，可以 $O(n2^k)$ 预处理去掉：

```cpp
#include "poly-fwt.cpp"
#define vv vector
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  ll n, k, m;
  cin >> n >> m, k = 3;
  vv<ll> w(k);
  for(auto &i : w) cin >> i;
  vv<vv<ll>> a(n, vv<ll>(k));
  for(auto &i : a) for(auto &j : i) cin >> j;
  ll uk = 1 << k, V = 1 << m;
  vv<vv<ll>> c(V, vv<ll>(uk));
  vv<ll> val(uk);
  for (ll i = 0; i < uk; ++i) {
    for (ll p = 0; p < k; ++p)
      if ((i >> p) & 1)
        val[i] = (val[i] - w[p] + MD) % MD;
      else val[i] = (val[i] + w[p]) % MD;
    vv<ll> f(V);
    for (ll j = 0; j < n; ++j) {
      ll z = 0;
      for (ll p = 0; p < k; ++p)
        if ((i >> p) & 1) z ^= a[j][p];
      ++f[z];
    }
    FWT(f.data(), V, 1, 1, 1, MD - 1);
    for (ll j = 0; j < V; ++j) c[j][i] = f[j];
  }
  for (ll i = 0; i < V; ++i) FWT(c[i].data(),
    uk, inv2, inv2, inv2, MD - inv2);
  vv<ll> Ans(V, 1);
  for (ll i = 0; i < V; ++i)
    for (ll j = 0; j < uk; ++j)
      Ans[i] = Ans[i] * qpow(val[j], c[i][j])
        % MD;
  FWT(Ans.data(), V, inv2, inv2, inv2, MD -
    inv2);
  for (ll i = 0; i < V; ++i)
    cout << Ans[i] << " \n"[i == V - 1];
  return 0;
}
```

7.4 FWT 全家桶

$$(FA)_i = \sum_j \prod_{k=n}^{0} c(B(i,k), B(j,k)) A_j, c_{*,i} c_{*,j} = c_{*,i\oplus j}$$

```cpp
#include "../header.cpp"
void FWT(ll *F, ll len, ll c00, ll c01, ll c10
  , ll c11) {
  for (ll t = 1; t < len; t <<= 1) {
    for (ll i = 0; i < len; i += t * 2) {
      for (ll j = 0; j < t; ++j) {
        ll x = F[i + j];
        ll y = F[i + j + t];
        F[i + j] = (x * c00 + y * c01) % MD;
        F[i + j + t] = (x * c10 + y * c11) % MD;
      }
    }
  }
}
```

7.5 任意模数 NTT

```cpp
#include "../header.cpp"
using cp = complex<ld>;
vector<int> rev;
vector<cp> om;   // exp(sgn * pi * k / len)
void FFT(cp* F, int len, int sgn) {
  rev.resize(len);
  for (int i = 1; i < len; ++i) {
    rev[i] = (rev[i >> 1] >> 1)
      | ((i & 1) * (len >> 1));
    if (i < rev[i]) swap(F[i], F[rev[i]]);
  }
  const ld pi = std::acos(ld(-1));
  om.resize(len);
  for (int i = 0; i < len; ++i) {
    om[i] = polar(ld(1), sgn * i * pi / len);
  }
  for (int tmp = 1; tmp < len; tmp <<= 1){
    for (int i = 0; i < len; i += tmp << 1){
      int K = len / tmp, pos = 0;
      for (int j = 0; j < tmp; ++j, pos += K){
        cp x = F[i + j];
        cp y = F[i + j + tmp] * om[pos];
        F[i + j] = x + y;
```

```cpp
        F[i + j + tmp] = x - y;
      }
    }
  }
  if (sgn == -1) {
    cp inv_len(ld(1) / len);
    for (int i = 0; i < len; ++i)
      F[i] = F[i] * inv_len;
  }
}
ll MD, M; // 输入 MD 后，需要设置 M 为 sqrt(MD)
using Poly = vector<ll>;
Poly polyMul(Poly F, Poly G, int tmp = 0) { // tmp 用于循环卷积技巧，卡常
  for (auto &k : F) k %= MD;
  for (auto &k : G) k %= MD;
  int n = (int)F.size() - 1, m = (int)G.size() - 1;
  if (tmp == 0) tmp = n + m + 1;
  int len = 1;
  while (len < tmp) len <<= 1;
  vector<cp> P(len), tP(len), Q(len);
  for (int i = 0; i <= n; ++i)
    P[i] = cp(F[i] / M, F[i] % M),
    tP[i] = cp(F[i] / M, -(F[i] % M));
  for (int i = 0; i <= m; ++i)
    Q[i] = cp(G[i] / M, G[i] % M);
  for(auto &X: {&P, &Q, &tP})
    FFT(X -> data(), len, 1);
  for (int i = 0; i < len; ++i)
    P[i] *= Q[i], tP[i] *= Q[i];
  FFT(P.data(), len, -1);
  FFT(tP.data(), len, -1);
  vector<ll> H(n + m + 1);
  for (int i = 0; i < tmp; ++i) {
    H[i] = ll((P[i].real() + tP[i].real()) / 2 + 0.5) % MD * M % MD * M % MD
      + ll(P[i].imag() + 0.5) % MD * M % MD
      + ll((tP[i].real() - P[i].real()) / 2 + 0.5) % MD;
    H[i] = (H[i] + MD) % MD;
  }
  return H;
}
```

8 字符串

8.1 AC 自动机

```cpp
#include "../header.cpp"
namespace ACAM{
  int C[MAXN][MAXM], F[MAXN], o;
  void insert(char *S){
    int p = 0, len = 0;
    for(int i = 0;S[i];++ i){
      int e = S[i] - 'a';
      if(C[p][e]) p = C[p][e];
        else      p = C[p][e] = ++ o;
      ++ len;
    }
  }
  void build(){
    queue <int> Q; Q.push(0);
    while(!Q.empty()){
      int u = Q.front(); Q.pop();
      for(int i = 0;i < 26;++ i){
        int p = F[u], v = C[u][i];
        if(v == 0) continue;
        while(!C[p][i] && p != 0) p = F[p];
        if(C[p][i] && C[p][i] != v)
          F[v] = C[p][i];
        Q.push(v);
      }
    }
  }
}
```

8.2 扩展 KMP

8.2.1 定义

$$z_i = |\mathrm{lcp}(b, \mathrm{suffix}(b, i))|$$

```cpp
#include "../header.cpp"
int Z[MAXN];
void exkmp(char A[]){
  int l = 0, r = 0; Z[1] = 0;
  for(int i = 2;A[i];++ i){
    Z[i] = i <= r ? min(r - i + 1, Z[i - l + 1]) : 0;
    while(A[Z[i] + 1] == A[i + Z[i]]) ++ Z[i];
    if(i + Z[i] - 1 > r)
      r = i + Z[i] - 1, l = i;
  }
}
```

8.3 最小表示法

```cpp
#include "../header.cpp"
int min_pos(const vector<int> &a) {
  int n = a.size(), i = 0, j = 1, k = 0;
  while (i < n && j < n && k < n) {
    int u = a[(i + k) % n],v = a[(j + k) % n];
    int t = u > v ? 1 : (u < v ? -1 : 0);
    if (t == 0) k ++; else {
      if (t > 0) i += k + 1; else j += k + 1;
      if (i == j) j++;
      k = 0;
    }
  }
  return min(i, j);
}
```

8.4 回文自动机

```cpp
#include "../header.cpp"
namespace PAM{
  const int SIZ = 5e5 + 3;
  int n, s, F[SIZ], L[SIZ], D[SIZ];
  int M[SIZ][MAXM];
  char S[SIZ];
  void init(){
    S[0] = '#', n = 1;
    F[s = 0] = -1, L[0] = -1, D[0] = 0;
    F[s = 1] =  0, L[1] =  0, D[1] = 0;
  }
  void extend(int &last, char c){
    S[++ n] = c;
    int e = c - 'a', a = last;
    while(c != S[n - 1 - L[a]]) a = F[a];
    if(M[a][e]){
      last = M[a][e];
    } else {
      int cur = M[a][e] = ++ s;
      L[cur] = L[a] + 2;
      if(a == 0){
        F[cur] = 1;
      } else {
        int b = F[a];
        while(c != S[n - 1 - L[b]])
          b = F[b];
        F[cur] = M[b][e];
      }
      D[cur] = D[F[cur]] + 1, last = cur;
    }
  }
}
```

8.5 后缀数组（倍增）

```cpp
#include "../header.cpp"
```

```cpp
int n, m, A[MAXN], B[MAXN], C[MAXN], R[MAXN], P[MAXN], Q[MAXN];
char S[MAXN];
int main(){
  scanf("%s", S), n = strlen(S), m = 256;
  for(int i = 0;i < n;++ i) R[i] = S[i];
  for (int k = 1;k <= n;k <<= 1){
    for(int i = 0;i < n;++ i){
      Q[i] = ((i + k > n - 1) ? 0 : R[i + k]);
      P[i] = R[i];
      m = max(m, R[i]);
    }
#define fun(a, b, c) \
memset(C, 0, sizeof(int) * (m + 1));          \
for(int i = 0;i <  n;++ i) C[a] += 1;         \
for(int i = 1;i <= m;++ i) C[i] += C[i - 1]; \
for(int i = n - 1;i >= 0;-- i) c[-- C[a]] = b;
    fun(Q[  i ],   i , B)
    fun(P[B[i]], B[i], A)
#undef fun
    int p = 1; R[A[0]] = 1;
    for(int i = 1;i <= n - 1;++ i){
      bool f1 = P[A[i]] == P[A[i - 1]];
      bool f2 = Q[A[i]] == Q[A[i - 1]];
      R[A[i]] = f1 && f2 ? R[A[i - 1]] : ++ p;
    }
    if (m == n) break;
  }
  for(int i = 0;i < n;++ i)
    printf("%u ", A[i] + 1);
  return 0;
}
```

8.6 广义后缀自动机（离线）

```cpp
#include "../header.cpp"
namespace SAM{
  const int SIZ = 2e6 + 3;
  int M[SIZ][MAXM], L[SIZ], F[SIZ], S[SIZ], s = 0, h = 25;
  void init(){ F[0] = -1, s = 0; }
  void extend(int &last, char c){
    int e = c - 'a';
    int cur = ++ s;
    L[cur] = L[last] + 1;
    int p = last;
    while(p != -1 && !M[p][e])
      M[p][e] = cur, p = F[p];
    if(p == -1){
      F[cur] = 0;
    } else {
      int q = M[p][e];
      if(L[p] + 1 == L[q]){
        F[cur] = q;
      } else {
        int clone = ++ s;
        L[clone] = L[p] + 1, F[clone] = F[q];
        for(int i = 0;i <= h;++ i)
          M[clone][i] = M[q][i];
        while(p != -1 && M[p][e] == q)
          M[p][e] = clone, p = F[p];
        F[cur] = F[q] = clone;
      }
    }
    last = cur;
  }
}
namespace Trie{
  int M[MAXN][MAXM], O[MAXN], s, h = 25;
  void insert(char *S);
  void build_sam(){
    queue <int> Q; Q.push(0);
    while(!Q.empty()){
      int u = Q.front(); Q.pop();
      for(int i = 0;i <= h;++ i){
        char c = i + 'a';
        if(M[u][i]){
          int v = M[u][i];
          O[v] = O[u];
          SAM :: extend(O[v], c);
          Q.push(v);
        }
      }
    }
  }
}
```

8.7 广义后缀自动机（在线）

```cpp
#include "../header.cpp"
namespace SAM{
  int M[MAXN][MAXM], L[MAXN], F[MAXN], S[MAXN], s = 0, h = 25;
  void init(){
    F[0] = -1, s = 0;
  }
  // 每次插入新字符串前将 last 清零
  void extend(int &last, char c){
    int e = c - 'a';
    if(M[last][e]){
      int p = last;
      int q = M[last][e];
      if(L[q] == L[last] + 1){
        last = q;
      } else {
        int clone = ++ s;
        L[clone] = L[p] + 1, F[clone] = F[q];
        for(int i = 0;i <= h;++ i)
          M[clone][i] = M[q][i];
        while(p != -1 && M[p][e] == q)
          M[p][e] = clone, p = F[p];
        F[q] = clone;
        last = clone;
      }
    } else {
      int cur = ++ s;
      L[cur] = L[last] + 1;
      int p = last;
      while(p != -1 && !M[p][e])
        M[p][e] = cur, p = F[p];
      if(p == -1){
        F[cur] = 0;
      } else {
        int q = M[p][e];
        if(L[p] + 1 == L[q]){
          F[cur] = q;
        } else {
          int clone = ++ s;
          L[clone] = L[p] + 1,F[clone] = F[q];
          for(int i = 0;i <= h;++ i)
            M[clone][i] = M[q][i];
          while(p != -1 && M[p][e] == q)
            M[p][e] = clone, p = F[p];
          F[cur] = F[q] = clone;
        }
      }
      last = cur;
    }
  }
}
```

8.8 后缀自动机

```cpp
#include "../header.cpp"
namespace SAM{
  int M[MAXN][MAXM], L[MAXN], F[MAXN], S[MAXN], last = 0, s = 0, h = 25;
  void init(){
    F[0] = -1, last = s = 0;
  }
  void extend(char c){
    int cur = ++ s, e = c - 'a';
    L[cur] = L[last] + 1, S[cur] = 1;
```

```
    int p = last;
    while(p != -1 && !M[p][e])
      M[p][e] = cur, p = F[p];
    if(p == -1){
      F[cur] = 0;
    } else {
      int q = M[p][e];
      if(L[p] + 1 == L[q]){
        F[cur] = q;
      } else {
        int clone = ++ s;
        L[clone] = L[p] + 1, F[clone] = F[q];
        S[clone] = 0;
        for(int i = 0;i <= h;++ i)
          M[clone][i] = M[q][i];
        while(p != -1 && M[p][e] == q)
          M[p][e] = clone, p = F[p];
        F[cur] = F[q] = clone;
      }
    }
    last = cur;
  }
}
```

9 计算几何

9.1 二维圆形

```
#include "2d.cpp"
struct circ: pp { db r; };  // 圆形
circ circ_i(pp a, pp b, pp c) {
  db A = dis(a,b), B = dis(a,c), C = dis(a,b);
  return {
    (a * A + b * B + c * C) / (A + B + C),
    abs((b - a) * (c - a)) / (A + B + C)
  };
}  // 三点确定内心
circ circ_2pp(pp a, pp b){
  return {(a + b) / 2, dis(a, b) / 2};
}  // 两点确定直径
circ circ_3pp(pp a, pp b, pp c) {
  pp bc = c - b, ca = a - c, ab = b - a;
  pp o = (b + c - r90(bc) * (ca % ab) / (ca * ab)) / 2;
  return {o, (a - o).abs()};
}  // 三点确定外心
circ minimal(vector <pp> V){  // 最小圆覆盖
  shuffle(V.begin(), V.end(), MT);
  circ C(V[0], 0);
  for(int i = 0;i < V.size();++ i) {
    if (dis((pp)C, V[i]) < C.r) continue;
    C = circ_2pp(V[i], V[0]);
    for(int j = 0;j < i;++ j) {
      if (dis((pp)C, V[j]) < C.r) continue;
      C = circ_2pp(V[i], V[j]);
      for(int k = 0;k < j;++ k) {
        if (dis((pp)C, V[k]) < C.r) continue;
        C = circ_3pp(V[i], V[j], V[k]);
      }
    }
  }
  return C;
}
```

9.2 Delaunay 三角剖分

```
// From Let it Rot. 在整数坐标同样可用
#include "2d.cpp"
using Q = struct Quad *;
const pp arb(LLONG_MAX, LLONG_MAX);
struct Quad {
  Q rot, o; pp p = arb; bool mark;
  pp& F() { return r() -> p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;
ll cross(pp a, pp b, pp c) {
  return (b - a) * (c - a);
}
// p 是否在 a, b, c 外接圆中
bool incirc(pp p, pp a, pp b, pp c) {
  i80 p2 = p.norm(), A = a.norm() - p2, B = b.norm() - p2, C = c.norm() - p2;
  a = a - p, b = b - p, c = c - p;
  return (a * b) * C + (b * c) * A + (c * a) * B > 0;
}
Q link(pp orig, pp dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r -> o; r -> r() -> r() = r;
  for(int i = 0;i < 4;++ i)
    r = r -> rot, r -> p = arb,
    r -> o = i & 1 ? r : r -> r();
  r -> p = orig, r -> F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a -> o -> rot -> o, b -> o -> rot ->o);
  swap(a -> o, b -> o);
}
Q conn(Q a, Q b) {
  Q q = link(a -> F(), b -> p);
  splice(q, a -> next());
  splice(q -> r(), b);
  return q;
}
pair<Q, Q> rec(const vector<pp> &s){
  int N = size(s);
  if(N <= 3) {
    Q a = link(s[0], s[1]), b = link(s[1], s.back());
    if(N == 2) return {a, a -> r()};
    splice(a -> r(), b);
    ll side = cross(s[0], s[1], s[2]);
    Q c = side ? conn(b, a) : 0;
    return { side < 0 ? c -> r() : a, side < 0 ? c : b -> r() };
  }
#define H(e) e -> F(), e -> p
#define valid(e) (cross(e->F(), H(base)) > 0)
  int half = N / 2;
  auto [ra, A]=rec({s.begin(), s.end()-half});
  auto [B, rb]=rec({s.end() - half, s.end()});
  while((cross(B -> p, H(A)) < 0 && (A = A -> next())) || (cross(A -> p, H(B)) > 0 && (B = B -> r() -> o)));
  Q base = conn(B -> r(), A);
  if(A -> p == ra -> p) ra = base -> r();
  if(B -> p == rb -> p) rb = base;
#define DEL(e, init, dir) \
  Q e = init -> dir; \
  if(valid(e)) \
    for(;incirc(e -> dir -> F(), H(base), e -> F());) { \
      Q t = e -> dir; \
      splice(e, e -> prev()); \
      splice(e -> r(), e -> r() -> prev()); \
      e -> o = H, H = e, e = t; \
    }
  for(;;) {
    DEL(LC, base -> r(), o);
    DEL(RC, base, prev());
    if(!valid(LC) && !valid(RC)) break;
    if(!valid(LC) || (valid(RC) && incirc(H(RC), H(LC))))
      base = conn(RC, base -> r());
    else
      base = conn(base -> r(), LC -> r());
  }
  return {ra, rb};
}
```

```cpp
// 返回若干逆时针三角形
vector<pp> deluanay(vector<pp> a){
  if((int)size(a) < 2) return {};
  sort(a.begin(), a.end()); // unique
  Q e = rec(a).first;
  vector<Q> q = {e};
  while(cross(e -> o -> F(), e -> F(), e -> p)
    < 0) e = e -> o;
  #define ADD { Q c = e; do { c -> mark = 1; a
    .push_back(c -> p); q.push_back(c -> r()),
    c = c -> next(); } while(c != e); }
  ADD; a.clear();
  for(int qi = 0;qi < size(q);)
    if(!(e = q[qi++]) -> mark) ADD;
  return a;
}
```

9.3 动态凸包（不支持删除）

```cpp
#include "2d.cpp"
struct Hull {
  set<pp> S;
  using Iter = set<pp>::iterator;
  long long ans = 0;
  long long area(pp a, pp b, pp c) {
    return llabs((b - a) * (c - a));
  }
  Iter remove_dir(Iter it, bool dir, pp x){
    Iter nxt;
    while (dir ? (it != S.begin() && (nxt =
      prev(it), 1)) : ((nxt = next(it)) != S.
      end())) {
      if (((*it - *nxt) * (x - *it) < 0) ==
        dir) break;
      ans += area(*nxt, *it, x), S.erase(it),
        it = nxt;
    }
    return it;
  }
  void insert(pp x){
    if (S.empty()){ S.insert(x); return; }
    auto r = S.lower_bound(x);
    if (r == S.end()) {
      remove_dir(prev(r), 1, x);
      S.insert(x);
    } else if (r == S.begin()) {
      remove_dir(r, 0, x);
      S.insert(x);
    } else {
      auto l = prev(r);
      if (((*r - *l) * (x - *l)) < 0)
        return; // 在凸包外侧
      ans += area(*l, *r, x);
      l = remove_dir(l, 1, x);
      r = remove_dir(r, 0, x);
      S.insert(x);
    }
  }
};
```

9.4 半平面交

```cpp
// From Let it Rot
#include "2d-lines.cpp"
vector<pp> HPI(vector<line> vs) {
  auto cmp = [](line a, line b) {
    return paraS(a, b) ? dis(a) < dis(b)
      : ::cmp(pp(a), pp(b));
  };
  sort(vs.begin(), vs.end(), cmp);
  int ah = 0, at = 0, n = size(vs);
  vector<line> deq(n + 1);
  vector<pp> ans(n);
  deq[0] = vs[0];
  for(int i = 1;i <= n;++i) {
    line o = i < n ? vs[i] : deq[ah];
    if(paraS(vs[i - 1], o)) continue;
    // maybe <=
    while(ah < at && check(deq[at - 1], deq[at
      ], o) < 0)
      -- at;
    if(i != n)
    while(ah < at && check(deq[ah], deq[ah +
      1], o) < 0)
      ++ ah;
    if(!is_parallel(o, deq[at])) {
      ans[at] = o & deq[at], deq[++at] = o;
    }
  }
  if(at - ah <= 2) return {};
  return {ans.begin() + ah, ans.begin() + at};
}
```

9.5 二维线段

```cpp
// From Let it Rot
#include "2d.cpp"
struct line : pp {
  db z; // a * x + b * y + c (= or >) 0
  line() = default;
  line(db a, db b, db c): pp{a, b}, z(c){}
  // 有向平面 a -> b 左侧区域
  line(pp a, pp b): pp(r90(b - a)), z(a * b){}
  db operator()(pp a) const { // ax + by + c
    return a % pp(*this) + z;
  }
  line vertical() const {
    return {y, -x, 0};
  } // 过 O 的垂直线
  line parallel(pp o) {
    return {x, y, z - this -> operator()(o)};
  } // 过 O 的平行线
};
pp operator & (line x, line y) { // 求交
  return pp{
    pp{x.z, x.y} * pp{y.z, y.y},
    pp{x.x, x.z} * pp{y.x, y.z}
  } / -(pp(x) * pp(y));
} // 注意此处精度误差较大，res.y 需要较高精度
pp project(pp x, line l){   // 投影
  return x - pp(l) * (l(x) / l.norm());
}
pp reflact(pp x, line l){   // 对称
  return x - pp(l) * (l(x) / l.norm()) * 2;
}
db dis(line l, pp x = {0, 0}){  // 有向点距离
  return l(x) / l.abs();
}
bool is_parallel(line x, line y){ // 判断平行
  return equal(pp(x) * pp(y), 0);
}
bool is_vertical(line x, line y){ // 判断垂直
  return equal(pp(x) % pp(y), 0);
}
bool online(pp x, line l) {     // 判断点在线
  return equal(l(x), 0);
}
int ccw(pp a, pp b, pp c) {
  int s = sign((b - a) * (c - a));
  if(s == 0) {
    if(sign((b - a) % (c - a)) == -1)
      return 2;
    if((c - a).norm() > (b - a).norm() + EPS)
      return -2;
  }
  return s;
}
db det(line a, line b, line c) {
  pp A = a, B = b, C = c;
  return c.z * (A * B) + a.z * (B * C) + b.z *
    (C * A);
}
db check(line a, line b, line c) { // sgn same
```

```cpp
  as c(a & b), 0 if error
  return sign(det(a, b, c)) * sign(pp(a) * pp(b));
}
bool paraS(line a, line b) { // 射线同向
  return is_parallel(a, b) && pp(a)%pp(b) > 0;
}
```

9.6 Voronoi 图

```cpp
// From Let it Rot
#include "2d-lines.cpp"
vector<line> cut(const vector<line> & o, line l) {
  vector<line> res;
  int n = size(o);
  for(int i = 0;i < n;++i) {
    line a = o[i], b = o[(i + 1) % n], c = o[(i + 2) % n];
    int va = check(a, b, l), vb = check(b, c, l);
    if(va > 0 || vb > 0 || (va == 0 && vb == 0))
      res.push_back(b);
    if(va >= 0 && vb < 0)
      res.push_back(l);
  }
  return res.size() <= 2 ? vector<line>{}:res;
} // 切凸包
line bisector(pp a, pp b) {return line(a.x - b.x, a.y - b.y, (b.norm() - a.norm()) / 2); }
vector<vector<line>> voronoi(vector<pp> p) {
  int n = p.size(); auto b = p;
  shuffle(b.begin(), b.end(), MT);
  const db V = 1e5; // 边框大小，重要
  vector<vector<line>> a(n, {
    { V, 0, V * V}, {0,  V, V * V},
    {-V, 0, V * V}, {0, -V, V * V},
  });
  for(int i = 0;i < n;++i) {
    for(pp x : b) if((x - p[i]).abs() > EPS){
      a[i] = cut(a[i], bisector(p[i], x));
    }
  }
  return a;
}
```

9.7 二维基础

```cpp
#include "../header.cpp"
const db EPS = 1e-9, PI  = acos(-1);
bool equal(db a, db b){ return fabs(a - b) < EPS; }
int sign(db a){ if(equal(a, 0)) return 0;
  return a > 0 ? 1 : -1;}
db sqr(db x) { return x * x; }
struct v2{   // 二维向量
  db x, y;
  v2(db _x = 0, db _y = 0) : x(_x), y(_y){}
  db norm() const {return (x * x + y * y);}
  db abs() const {return sqrt(x * x + y * y);}
  db arg() const {return atan2(y, x); }
};
bool operator ==(v2 a, v2 b){
  return a.x == b.x && a.y == b.y;
}
bool operator < (v2 a, v2 b){
    return a.x == b.x ? a.y < b.y : a.x < b.x;
}
v2 r90(v2 x) { return {-x.y, x.x}; }
v2 operator +(v2 a, v2 b){
  return {a.x + b.x, a.y + b.y}; }
v2 operator -(v2 a, v2 b){
  return {a.x - b.x, a.y - b.y}; }
v2 operator *(v2 a, db w){
  return {a.x * w, a.y * w}; }
v2 operator /(v2 a, db w){
  return {a.x / w, a.y / w}; }
db operator *(v2 a, v2 b){  // 叉乘，b > a 为负
  return a.x * b.y - a.y * b.x; }
db operator %(v2 a, v2 b){  // 点乘
  return a.x * b.x + a.y * b.y; }
bool equal(v2 a, v2 b){
  return equal(a.x, b.x) && equal(a.y, b.y);
}
using pp = v2;
pp rotate(pp a, db t){
  db c = cos(t), s = sin(t);
  return pp(a.x * c - a.y * s, a.y * c + a.x * s);
}
db dis(pp a, pp b){
  return (a - b).abs();
}
int half(pp x){ // 为 1 则 arg >= \pi
  return x.y < 0 || (x.y == 0 && x.x <= 0);
}
// int half(pp x){return x.y < -EPS || (std::fabs(x.y) < EPS && x.x < EPS);}
bool cmp(pp a, pp b) {  // arg a < arg b
  return half(a) == half(b) ? a * b > 0 : half(b);
}
```

10 其他

10.1 笛卡尔树

```cpp
#include "../header.cpp"
// Li: 左儿子; Ri: 右儿子
int n, L[MAXN], R[MAXN], A[MAXN];
void build(){
  stack <int> S; A[n + 1] = -1e9;
  for(int i = 1;i <= n + 1;++ i){
    int v = 0;
    while(!S.empty() && A[S.top()] > A[i]){
      auto u = S.top();
      R[u] = v, v  = u, S.pop();
    }
    L[i] = v, S.push(i);
  }
}
```

10.2 CDQ 分治

10.2.1 例题

给定三元组序列 (a_i, b_i, c_i)，求解 $f(i) = \sum_j [a_j \le a_i \wedge b_j \le b_i \wedge c_j \le c_i]$。

```cpp
#include "../header.cpp"
struct Node{int id, a, b, c;}A[MAXN], B[MAXN];
bool cmp(Node a, Node b){
  return a.a == b.a ? (a.b == b.b ? (a.c == b.c ? a.c < b.c : a.id < b.id) : a.b < b.b) : a.a < b.a;
}
int K[MAXN], H[MAXN], n, m, D[MAXM];
namespace BIT{
  void modify(int x, int w);  // 加到 m
  void query(int x, int &r);
}
using BIT :: modify, BIT :: query;
void cdq(int l, int r){
  if(l != r){
    int t = l + r >> 1;
    cdq(l, t), cdq(t + 1, r);
    int p = l, q = t + 1, u = l;
    while(p <= t && q <= r){
      if(A[p].b <= A[q].b)
modify(A[p].c, 1), B[u ++] = A[p ++];
      else
```

```cpp
query(A[q].c, K[A[q].id]), B[u ++] = A[q ++];
    }
    while(p <= t) modify(A[p].c, 1), B[u ++] = A[p ++];
    while(q <= r) query(A[q].c, K[A[q].id]), B[u ++] = A[q ++];
    up(l, t, i) modify(A[i].c, -1);
    up(l, r, i) A[i] = B[i];
  }
}
int main(){
  n = qread(), m = qread();
  up(1, n, i) A[i].id = i, A[i].a = qread(), A[i].b = qread(), A[i].c = qread();
  sort(A + 1, A + 1 + n, cmp), cdq(1, n);
  sort(A + 1, A + 1 + n, cmp);
  dn(n, 1, i){
    if(A[i].a == A[i + 1].a && A[i].b == A[i + 1].b && A[i].c == A[i + 1].c)
      K[A[i].id] = K[A[i + 1].id];
    H[K[A[i].id]] ++;
  }
  up(0, n - 1, i) printf("%d\n", H[i]);
  return 0;
}
```

10.3 日期公式

```cpp
// 从公元 1 年 1 月 1 日经过了多少天
int getday(int y, int m, int d) {
  if(m < 3) -- y, m += 12;
  return (365 * y + y / 4 - y / 100 + y / 400
    + (153 * (m - 3) + 2) / 5 + d - 307);
}
// 公元 1 年 1 月 1 日往后数 n 天是哪一天
void date(int n, int &y, int &m, int &d) {
  n += 429 + ((4 * n + 1227) / 146097 + 1) * 3 / 4;
  y = (4 * n - 489) / 1461, n -= y * 1461 / 4;
  m = (5 * n - 1) / 153,  d = n - m * 153 / 5;
  if (--m > 12) m -= 12, ++y;
}
```

10.4 pbds 库

```cpp
#include "../header.cpp"
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>     // 树
#include <ext/pb_ds/priority_queue.hpp> // 堆
using namespace __gnu_pbds;
// insert, erase, order_of_key, find_by_order,
// [lower|upper]_bound, join, split, size
__gnu_pbds::tree<int, null_type, less<int>,
    rb_tree_tag,
    tree_order_statistics_node_update> T;
// push, pop, top, size, empty, modify, erase,
// join 其中 modify 修改寄存器, join 合并堆
// 还可以用 rc_binomial_heap_tag
__gnu_pbds::priority_queue<int, less<int>,
    pairing_heap_tag> Q1, Q2;
```

10.5 Python 技巧

```python
import random
random.normalvariate(0.5, 0.1) # 正态分布
l = [str(i) for i in range(9)] # 列表
sorted(l), min(l), max(l), len(l)
random.shuffle(l)
l.sort(key=lambda x:x ^ 1,reverse=True)
from functools import cmp_to_key
l.sort(key=cmp_to_key(lambda x,y:(y^1)-(x^1)))
from itertools import *
for i in product('ABCD', repeat=2):
    pass # AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD
for i in permutations('ABCD', repeat=2):
    pass # AB AC AD BA BC BD CA CB CD DA DB DC
for i in combinations('ABCD', repeat=2):
    pass # AB AC AD BC BD CD
for i in combinations_with_replacement('ABCD', repeat=2):
    pass # AA AB AC AD BB BC BD CC CD DD
from fractions import Fraction
a.numerator, a.denominator, str(a)
a = Fraction(0.233).limit_denominator(1000)
from decimal import Decimal, getcontext, FloatOperation
getcontext().prec = 100
getcontext().rounding = getattr(decimal, 'ROUND_HALF_EVEN')
# default; other: FLOOR, CELILING, DOWN, ...
getcontext().traps[FloatOperation] = True
Decimal((0, (1, 4, 1, 4), -3)) # 1.414
a = Decimal(1<<31) / Decimal(100000)
print(f"{a:.9f}") # 21474.83648
print(a.sqrt(), a.ln(), a.log10(), a.exp(), a ** 2)
a = 1-2j
print(a.real, a.imag, a.conjugate())
import sys, atexit, io
_INPUT_LINES = sys.stdin.read().splitlines()
input = iter(_INPUT_LINES).__next__
_OUTPUT_BUFFER = io.StringIO()
sys.stdout = _OUTPUT_BUFFER
@atexit.register
def write():
    sys.__stdout__.write(_OUTPUT_BUFFER.getvalue())
```

10.6 快速测试

```bash
export CXXFLAGS='-g -Wall -fsanitize=address,undefined -Dzwb -std=gnu++20'
mk() { g++ -O2 -Dzwb -std=gnu++20 1.cpp − o1; }
ulimit -s 1048576
ulimit -v 1048576
```

10.7 自适应辛普森

```cpp
#include "../header.cpp"
db simpson(db (*f)(db), db l, db r){
  return (r - l) / 6 *
    (f(l) + 4 * f((l + r) / 2) + f(r));
}
db adapt_simpson(db (*f)(db), db l, db r, db EPS, int step){
  db mid = (l + r) / 2;
  db w0 = simpson(f, l, r), w1 = simpson(f, l, mid), w2 = simpson(f, mid, r);
  return fabs(w0 - w1 - w2) < EPS && step < 0?
    w1 + w2 :
    adapt_simpson(f, l, mid, EPS, step - 1) +
    adapt_simpson(f, mid, r, EPS, step - 1);
}
```

10.8 对拍脚本

```bash
while true; do
  ./gen > 1.in; ./naive < 1.in > std.out; ./a < 1.in > 1.out
  if diff 1.out std.out; then echo ac; else echo wa; break fi
done
```

10.9 伪随机生成

```cpp
#include "../header.cpp"
u32 xorshift32(u32 &x){ return
  x ^= x << 13, x ^= x >> 17, x ^= x << 5; }
u64 xorshift64(u64 &x){ return
  x ^= x << 13, x ^= x >> 7, x ^= x << 17; }
```

$n \le$	10	100	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}	10^{11}	10^{12}	10^{13}	10^{14}	10^{15}	10^{16}	10^{17}	10^{18}
$\max\omega(n)$	2	3	4	5	6	7	8	8	9	10	10	11	12	12	13	13	14	15
$\max d(n)$	4	12	32	64	128	240	448	768	1344	2304	4032	6720	10752	17280	26880	41472	64512	103680
$\pi(n)$	4	25	168	1229	9592	78498	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10			$n/\ln(n)$			
$n =$	2	3	4	5	6	7	8	9	10	11	12	15	20	25	30	40	50	114
$\log_{10} n$	0.301	0.477	0.602	0.698	0.778	0.845	0.903	0.954	1	1.041	1.079	1.176	1.301	1.398	1.477			
$C(n, n/2)$	2	3	6	10	20	35	70	126	252	462	924	6435	184756	5200300	155117520			
$\mathrm{LCM}(1\dots n)$	2	6	12	60	60	420	840	2520	2520	27720	27720	360360	232792560	26771144400	1.444e14			
P_n	2	3	5	7	11	15	22	30	42	56	77	176	627	1958	5604	37338	204226	10^9

- 质数：$918733, 940649, 923641, 992402371, 903936311, 977499077, 908370375016605079, 973150773996892879, 990191901472122599, 10^{18}+3$；
- NTT 模数：$167772161, 1004535809, 2924438830427668481 = 174310137655 \times 2^{24} + 1$。

二项式系数

$$\sum_{k=0}^{n}\binom{k}{m}=\binom{n+1}{m+1},\quad \sum_{k=0}^{r}\binom{r-k}{m}\binom{s+k}{n}=\binom{r+s+1}{m+n+1}$$

斐波那契数

$$\sum_{k=1}^{n} f_k^2 = f_n f_{n+1},\quad \sum_{k=1}^{n} k f_k = n f_{n+2} - f_{n+3} + 2,\quad \sum_{k=0}^{n} f_k f_{n-k} = \frac{1}{5}(n-1) f_n + \frac{2}{5} n f_{n-1}$$

$$f_{n+k} = f_n f_{k+1} + f_{n-1} f_k,\quad (-1)^k f_{n-k} = f_n f_{k-1} - f_{n-1} f_k$$

```python
def fib(n): # F(n), F(n + 1)
  if not n: return (0, 1)
  a, b = fib(n >> 1); c = a * (2 * b - a); d = a * a + b * b
  return (d, c + d) if n & 1 else (c, d)
```

斯特林数

第一类 n 个元素集合分作 k 个非空轮换方案数。

$$\begin{bmatrix}n+1\\k\end{bmatrix} = n\begin{bmatrix}n\\k\end{bmatrix} + \begin{bmatrix}n\\k-1\end{bmatrix},\quad x^{\underline{n}} = \sum_k \begin{bmatrix}n\\k\end{bmatrix}(-1)^{n-k}x^k,\quad x^{\overline{n}} = \sum_k \begin{bmatrix}n\\k\end{bmatrix} x^k$$

第二类 把 n 个元素集合分作 k 个非空子集方案数。

$$\begin{Bmatrix}n+1\\k\end{Bmatrix} = k\begin{Bmatrix}n\\k\end{Bmatrix} + \begin{Bmatrix}n\\k-1\end{Bmatrix},\quad x^n = \sum_k \begin{Bmatrix}n\\k\end{Bmatrix} x^{\underline{k}} = \sum_k \begin{Bmatrix}n\\k\end{Bmatrix}(-1)^{n-k} x^{\overline{k}}$$

$$m!\begin{Bmatrix}n\\m\end{Bmatrix} = \sum_k \binom{m}{k} k^n (-1)^{m-k}$$

求法 对于列，使用下述 EGF；对于行，第一类使用 $x^{\overline{n}}$，第二类使用二项式反演。

$$\sum_{n=0}^{\infty}\begin{bmatrix}n\\k\end{bmatrix}\frac{x^n}{n!} = \frac{x^k}{k!}\left(\frac{\ln(1-x)}{x}\right)^k,\quad \sum_{n=0}^{\infty}\begin{Bmatrix}n\\k\end{Bmatrix}\frac{x^n}{n!} = \frac{x^k}{k!}\left(\frac{e^x-1}{x}\right)^k$$

欧拉数

恰好有 m 次上升（$p_i > p_{i-1}$）的长为 n 的排列个数记为 $\left\langle {n \atop m-1} \right\rangle$。

$$\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle,\quad \left\langle {n \atop m} \right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$$

贝尔数（1, 1, 2, 5, 15, 52, 203, 877, 4140）

n 个元素集合划分的方案数。

$$B_n = \sum_{k=1}^{n}\begin{Bmatrix}n\\k\end{Bmatrix},\quad B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k,\quad B_{p^m+n} \equiv mB_n + B_{n+1} \pmod p,\quad B(x) = e^{e^x-1}$$

拆分数

将整数 n 拆分成若干个正整数之和的方案数为 $p(n)$。

```python
def penta(k):
  return k*(3*k-1)//2
def compute_partition(goal):
  p = [1]
  for n in range(1,goal+1):
    p.append(0)
    for k in range(1,n+1):
      c = (-1)**(k+1)
      for t in [penta(k), penta(-k)]:
        if (n-t) >= 0: p[n] = p[n] + c*p[n-t]
  return p
```