Team Reference Document

Kilo_5723

2024 年 3 月 23 日

目录

1 数据结构

1.1 笛卡尔树

```

struct node {
  array<node *, 2> ch;
  int val, idx;
  node(int val, int idx) : val(val), idx(idx) { ch[0] = ch[1] = NULL; }
};
node *build(vector<int> &a) {
  vector<node *> stk;
  for (int i = 0; i < a.size(); i++) {
    node *last = NULL;
    while (stk.size() && stk.back()->val > a[i]) {
      last = stk.back();
      stk.pop_back();
    }
    node *u = new node(a[i], i);
    if (stk.size()) stk.back()->ch[1] = u;
    if (last) u->ch[0] = last;
    stk.push_back(u);
  }
  return stk[0];
}
```

1.2 Link-Cut Tree

```
struct lct {
  vector<array<int, 2>> ch;
  vector<int> fa;
  vector<int> rev;
  vector<int> siz;
  lct(int n)
      : ch(n + 1, {0, 0}), fa(n + 1, 0), rev(n + 1, false), siz(n + 1, 1) {
    siz[0] = 0;  // initialize nil
  }
  void update(int u) { siz[u] = siz[ch[u][0]] + siz[ch[u][1]] + 1; }
  void reverse(int u) {
    if (u) rev[u] ^= 1, swap(ch[0], ch[1]);
  }
  void pushdown(int u) {
    if (rev[u]) reverse(ch[u][0]), reverse(ch[u][1]);
    rev[u] = false;
  }
  int chid(int u) {
    if (ch[fa[u]][0] == u) return 0;
    if (ch[fa[u]][1] == u) return 1;
    return -1;
  }
  bool isroot(int u) { return chid(u) == -1; }
  void rotate(int u) {
    int v = fa[u], w = fa[v], k = chid(u), x = ch[u][!k];
    if (!isroot(v)) ch[w][chid(v)] = u;
    ch[u][!k] = v, ch[v][k] = x;
    if (x) fa[x] = v;
    fa[v] = u, fa[u] = w;
    update(v), update(u);
  }
  void clearup(int u) {
    if (!isroot(u)) clearup(fa[u]);
    pushdown(u);
  }
  void splay(int u) {
    clearup(u);
    while (!isroot(u)) {
      int f = fa[u];
      if (!isroot(f)) rotate(chid(u) == chid(f) ? f : u);
      rotate(u);
    }
  }
  int access(int u) {
    int v;
    for (v = 0; u; v = u, u = fa[u]) splay(u), ch[u][1] = v, update(u);
    return v;
  }
  void makeroot(int u) {
    u = access(u);
    reverse(u);
  }
  void link(int u, int v) {
    makeroot(u);
    splay(u);
    fa[u] = v;
  }
  void cut(int u,int v){
    makeroot(u);
    access(v);
```

```
    splay(v);
    ch[v][0]=fa[u]=0;
    update(v);
  }
};
```

1.3 李超线段树

```
#include <optional>

#include "bits/stdc++.h"
using namespace std;
const char el = '\n';
typedef long long ll;

const ll inf = 1e18;
struct node {
  node *ls, *rs;
  int pl, pm, pr;
  int xl, xm, xr;
  ll k, b;
  int tag;
  node(int l, int r) {
    pl = l, pr = r, pm = l + (r - l >> 1);
    xl = xm = xr = 0;
    k = 0, b = 0;
    tag = 0;
    if (l == r) {
      ls = rs = NULL;
      return;
    }
    ls = new node(l, pm);
    rs = new node(pm + 1, r);
  }
  void addtag(int t) {
    tag += t;
    b -= k * t;
    xl += t, xm += t, xr += t;
  }
  void pushtag() {
    if (ls) ls->addtag(tag);
    if (rs) rs->addtag(tag);
    tag = 0;
  }
  void addfunc(int l, int r, ll _k, ll _b) {
    if (r < pl || l > pr) return;
    pushtag();
    if (l <= pl && r >= pr) {
      if (k * xm + b < _k * xm + _b) {
        swap(k, _k);
        swap(b, _b);
      }
      if (k * xl + b < _k * xl + _b && ls) ls->addfunc(l, r, _k, _b);
      if (k * xr + b < _k * xr + _b && rs) rs->addfunc(l, r, _k, _b);
      return;
    }
    if (ls) ls->addfunc(l, r, _k, _b);
    if (rs) rs->addfunc(l, r, _k, _b);
  }
  void addx(int p, int v) {
    if (pr < p) return;
    if (pl >= p) {
      addtag(v);
      return;
    }
    if (pl >= p) xl += v;
    if (pm >= p) xm += v;
    if (pr >= p) xr += v;
    if (ls) ls->addfunc(pl, pr, k, b);
    if (rs) rs->addfunc(pl, pr, k, b);
    pushtag();
    if (ls) ls->addx(p, v);
    if (rs) rs->addx(p, v);
    k = 0, b = 0;
  }
  optional<pair<int, ll>> query(int p) {
    if (p < pl || p > pr) return nullopt;
    if (pl == pr) return pair<int, ll>{xm, k * xm + b};
    pushtag();
    auto res = ls->query(p);
    if (!res) res = rs->query(p);
    auto [x, y] = *res;
    return pair<int, ll>{x, max(y, k * x + b)};
  }
};
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(15);
```

```
  int tt;
  cin >> tt;
  while (tt--) {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> a(m + 1);
    vector<int> c(m + 1);
    while (n--) {
      int l, r;
      cin >> l >> r;
      a[l].push_back(r);
      c[r]++;
    }
    vector<ll> p(m + 1);
    for (int i = 1; i <= m; i++) cin >> p[i];
    node *rt = new node(0, m + 1);
    for (int i = 1; i <= m; i++) {
      auto [x, y] = *rt->query(i);
      rt->addx(i, -c[i - 1]);
      sort(a[i].begin(), a[i].end());
      for (auto j : a[i]) rt->addx(j + 1, 1);
      rt->addfunc(0, m + 1, p[i], y);
    }
    cout << rt->query(m + 1)->second << el;
  }
  return 0;
}
// https://codeforces.com/contest/1830/problem/F
```

1.4 可持久化线段树

```
// ver 2.0
struct node {
  int l, r;
  node *ls, *rs;
  int tag;
  node() : l(0), r(0), ls(nullptr), rs(nullptr), tag(0) {}
  static void* operator new(size_t count) {
    static node *begin = nullptr, *end = nullptr;
    if (begin == end) begin = (node*)malloc(count * 1000), end = begin + 1000;
    return begin++;
  }
};
node* build(int l, int r) {
  node* u = new node();
  u->l = l, u->r = r;
  if (l == r) return u;
  int m = l + (r - l) / 2;
  u->ls = build(l, m);
  u->rs = build(m + 1, r);
  return u;
}
node* addtag(node* u, int val) {
  u = new node(*u);
  u->tag += val;
  return u;
}
node* pushdown(node* u) {
  u = new node(*u);
  if (u->tag) {
    u->ls = addtag(u->ls, u->tag);
    u->rs = addtag(u->rs, u->tag);
    u->tag = 0;
  }
  return u;
}
node* add(node* u, int l, int r, int val) {
  if (u->l > r || u->r < l) return u;
  if (l <= u->l && u->r <= r) return addtag(u, val);
  u = pushdown(u);
  u->ls = add(u->ls, l, r, val);
  u->rs = add(u->rs, l, r, val);
  return u;
}
node* merge(node* u, node* v, int p) {
  if (u->r <= p) return u;
  if (v->l > p) return v;
  u = pushdown(u), v = pushdown(v);
  node* w = new node(*u);
  w->ls = merge(u->ls, v->ls, p);
  w->rs = merge(u->rs, v->rs, p);
  return w;
}
int val(node* u, int p) {
  if (u->r < p || u->l > p) return 0;
  if (u->l == u->r) return u->tag;
  return u->tag + val(u->ls, p) + val(u->rs, p);
}
```

1.5 区间处理

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct range {
  int l, r;
  bool valid() const { return r >= l; }
  bool cover(range rg) const { return l <= rg.l && r >= rg.r; }
  bool cross(range rg) const { return l <= rg.r && r >= rg.l; }
};
bool operator<(range a, range b) {
  return a.l < b.l || a.l == b.l && a.r > b.r;
}
struct rngseq {
  static const int inf = 1e9 + 7;
  set<range> seq;
  rngseq() : seq({{-inf, -inf}, {inf, inf}}) {}
  void add(range rg) {
    if (!rg.valid()) return;
    seq.insert(rg);
    rt->qadd(rg.l, rg.r, 1);
  }
  void rmv(range rg) {
    if (!rg.valid()) return;
    seq.erase(rg);
    rt->qadd(rg.l, rg.r, -1);
  }
  void insert(range rg) {
    auto it = seq.upper_bound(rg);
    auto tmp = *prev(it);
    if (tmp.cover(rg)) return;
    if (tmp.cross(rg)) {
      rg.l = tmp.l;
      rmv(tmp);
    }
    while (rg.cross(*it)) {
      tmp = *it++;
      rg.r = max(rg.r, tmp.r);
      rmv(tmp);
    }
    add(rg);
  }
  void erase(range rg) {
    auto it = seq.upper_bound(rg);
    auto tmp = *prev(it);
    if (tmp.cover(rg)) {
      rmv(tmp);
      add({tmp.l, rg.l - 1}), add({rg.r + 1, tmp.r});
      return;
    }
    if (tmp.cross(rg)) {
      rmv(tmp);
      add({tmp.l, rg.l - 1});
    }
    while (rg.cross(*it)) {
      tmp = *it++;
      rmv(tmp);
      add({rg.r + 1, tmp.r});
    }
  }
};
```

1.6 圆方树

```cpp
#include <algorithm>
#include <cstdio>
#include <vector>

const int MN = 100005;

int N, M, cnt;
std::vector<int> G[MN], T[MN * 2];

int dfn[MN], low[MN], dfc;
int stk[MN], tp;

void Tarjan(int u) {
  printf("  Enter : #%d\n", u);
  low[u] = dfn[u] = ++dfc;                // low 初始化为当前节点 dfn
  stk[++tp] = u;                          // 加入栈中
  for (int v : G[u]) {                    // 遍历 u 的相邻节点
    if (!dfn[v]) {                        // 如果未访问过
      Tarjan(v);                          // 递归
      low[u] = std::min(low[u], low[v]);  // 未访问的和 low 取 min
      if (low[v] == dfn[u]) {  // 标志着找到一个以 u 为根的点双连通分量
        ++cnt;                 // 增加方点个数
        printf("  Found a New BCC #%d.\n", cnt - N);
        // 将点双中除了 u 的点退栈，并在圆方树中连边
```

```
        for (int x = 0; x != v; --tp) {
          x = stk[tp];
          T[cnt].push_back(x);
          T[x].push_back(cnt);
          printf("    BCC #%d has vertex #%d\n", cnt - N, x);
        }
        // 注意 u 自身也要连边（但不退栈）
        T[cnt].push_back(u);
        T[u].push_back(cnt);
        printf("    BCC #%d has vertex #%d\n", cnt - N, u);
      }
    } else
      low[u] = std::min(low[u], dfn[v]);  // 已访问的和 dfn 取 min
  }
  printf("  Exit : #%d : low = %d\n", u, low[u]);
  printf("  Stack:\n    ");
  for (int i = 1; i <= tp; ++i) printf("%d, ", stk[i]);
  puts("");
}

int main() {
  scanf("%d%d", &N, &M);
  cnt = N;  // 点双 / 方点标号从 N 开始
  for (int i = 1; i <= M; ++i) {
    int u, v;
    scanf("%d%d", &u, &v);
    G[u].push_back(v);  // 加双向边
    G[v].push_back(u);
  }
  // 处理非连通图
  for (int u = 1; u <= N; ++u)
    if (!dfn[u]) Tarjan(u), --tp;
  // 注意到退出 Tarjan 时栈中还有一个元素即根，将其退栈
  return 0;
}
```

1.7 线段树

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

const ll inf = 1e18;
struct node {
  node *ls, *rs;
  int l, r;
  ll val, sum;
  pair<ll, int> mxm, mnm;
  static node *newnd() {
    static const int buff = 1000;
    static node *ptr = new node[buff], *cur = ptr;
    if (cur == ptr + buff) ptr = new node[buff], cur = ptr;
    return cur++;
  }
  void update() {
    sum = ls->sum + rs->sum;
    mxm = max(ls->mxm, rs->mxm);
    mnm = min(ls->mnm, rs->mnm);
  }
  void build(int _l, int _r, vector<ll> &a) {
    l = _l, r = _r;
    if (l == r) {
      val = a[l];
      mxm = mnm = {val, l};
      sum = abs(val);
      return;
    }
    int m = (l + r) / 2;
    ls = new node(), ls->build(l, m, a);
    rs = new node(), rs->build(m + 1, r, a);
    update();
  }
  void modify(int p, int a) {
    if (p < l || p > r) return;
    if (l == r) {
      val = val + a;
      mxm = mnm = {val, l};
      sum = abs(val);
      return;
    }
    ls->modify(p, a), rs->modify(p, a);
    update();
  }
  int ppos(int _l, int _r) {
    if (_r < l || _l > r) return -1;
    if (_l <= l && _r >= r) {
      if (mxm.first < 0) return -1;
      if (l == r) return l;
    }
```

```
    auto res = ls->ppos(_l, _r);
    if (~res)
      return res;
    else
      return rs->ppos(_l, _r);
  }
  pair<ll, int> qmxm(int _l, int _r) {
    if (_r < l || _l > r) return {-inf, -1};
    if (_l <= l && _r >= r) return mxm;
    return max(ls->qmxm(_l, _r), rs->qmxm(_l, _r));
  }
  pair<ll, int> qmnm(int _l, int _r) {
    if (_r < l || _l > r) return {inf, -1};
    if (_l <= l && _r >= r) return mnm;
    return min(ls->qmnm(_l, _r), rs->qmnm(_l, _r));
  }
  ll qsum(int _l, int _r) {
    if (_r < l || _l > r) return 0;
    if (_l <= l && _r >= r) return sum;
    return ls->qsum(_l, _r) + rs->qsum(_l, _r);
  }
};
auto newnd = node::newnd;

const ll inf = 1e18;
int ls(int u) { return u << 1; }
int rs(int u) { return u << 1 | 1; }
struct segtree {
  vector<ll> mnm;
  int l, r;
  void build(int l_, int r_) {
    l = l_, r = r_;
    int k = 1;
    while (k < r - l + 1) k <<= 1;
    mnm.resize(k << 1);
  }
  void update(int u) { mnm[u] = min(mnm[ls(u)], mnm[rs(u)]); }
  void init(int mode, int l = -1, int r = -1, int u = 1) {
    if (!~l) l = this->l, r = this->r;
    if (l == r) {
      mnm[u] = (~mode ? mode ? inf : l : -l);
      return;
    }
    int m = l + (r - l) / 2;
    init(mode, l, m, ls(u));
    init(mode, m + 1, r, rs(u));
    update(u);
  }
  void modify(int p, ll a, int l = -1, int r = -1, int u = 1) {
    if (!~l) l = this->l, r = this->r;
    if (p < l || p > r) return;
    if (l == r) {
      mnm[u] = a;
      return;
    }
    int m = l + (r - l) / 2;
    modify(p, a, l, m, ls(u));
    modify(p, a, m + 1, r, rs(u));
    update(u);
  }
  ll qmnm(int l_, int r_, int l = -1, int r = -1, int u = 1) {
    if (!~l) l = this->l, r = this->r;
    if (r_ < l || l_ > r) return inf;
    if (l_ <= l && r_ >= r) return mnm[u];
    int m = l + (r - l) / 2;
    return min(qmnm(l_, r_, l, m, ls(u)), qmnm(l_, r_, m + 1, r, rs(u)));
  }
};
```

1.8 Treap

```
struct node {
  array<node *, 2> ch;
  cplx val, sum;
  int rank;
  node(cplx val) : val(val) {
    ch[0] = ch[1] = NULL;
    sum = val;
    rank = rand();
  }
  void update() {
    sum = val;
    if (ch[0]) sum = ch[0]->sum + sum;
    if (ch[1]) sum = sum + ch[1]->sum;
  }
};
bool operator<(cplx a, cplx b) {
  auto d = det(a, b);
  if (d) return d < 0;
```

```
  if (a.x != b.x) return a.x < b.x;
  if (a.y != b.y) return a.y < b.y;
  if (a.z != b.z) return a.z < b.z;
  return false;
}
void rotate(node *&u, int c) {
  auto v = u->ch[c];
  u->ch[c] = v->ch[!c];
  v->ch[!c] = u;
  u->update();
  v->update();
  u = v;
}
node *insert(node *rt, cplx val) {
  if (!rt) return new node(val);
  auto c = rt->val < val;
  rt->ch[c] = insert(rt->ch[c], val);
  if (rt->ch[c]->rank < rt->rank) rotate(rt, c);
  rt->update();
  return rt;
}
void erase(node *&rt, cplx val) {
  if (val < rt->val) {
    erase(rt->ch[0], val);
    rt->update();
    return;
  }
  if (rt->val < val) {
    erase(rt->ch[1], val);
    rt->update();
    return;
  }
  auto tmp = rt;
  if (rt->ch[0]) {
    if (rt->ch[1]) {
      auto c = rt->ch[0]->rank > rt->ch[1]->rank;
      rotate(rt, c);
      erase(rt->ch[!c], val);
      rt->update();
    } else {
      rt = rt->ch[0];
      delete tmp;
    }
  } else {
    if (rt->ch[1]) {
      rt = rt->ch[1];
      delete tmp;
    } else {
      rt = NULL;
      delete tmp;
    }
  }
}


struct node {
  array<node *, 2> ch;
  ui p, n, avg;
  ui sl, sr, tag, sum;
  ui rank;
  node(int l, int r) {
    ch[0] = ch[1] = NULL;
    sl = p = l, sr = n = r;
    avg = tag = sum = 0;
    rank = rand();
  }
  void addtag(ui t) {
    tag += t, avg += t;
    sum += (sr - sl) * t;
  }
  void pushdown() {
    if (ch[0]) ch[0]->addtag(tag);
    if (ch[1]) ch[1]->addtag(tag);
    tag = 0;
  }
  void update() {
    sum = (n - p) * avg;
    sl = ch[0] ? ch[0]->sl : p;
    sr = ch[1] ? ch[1]->sr : n;
    if (ch[0]) sum += ch[0]->sum;
    if (ch[1]) sum += ch[1]->sum;
  }
  void add(ui l, ui r, ui v) {
    if (l >= sr || r <= sl) return;
    if (l <= sl && r >= sr) {
      addtag(v);
      return;
    }
    pushdown();
    if (l <= p && r >= n) avg += v;
```

```
    if (ch[0]) ch[0]->add(l, r, v);
    if (ch[1]) ch[1]->add(l, r, v);
    update();
  }
  ui qsum(ui l, ui r) {
    if (l >= sr || r <= sl) return 0;
    if (l <= sl && r >= sr) return sum;
    pushdown();
    ui res = 0;
    if (l <= p && r >= n) res += (n - p) * avg;
    if (ch[0]) res += ch[0]->qsum(l, r);
    if (ch[1]) res += ch[1]->qsum(l, r);
    return res;
  }
};
node *merge(node *u, node *v) {
  if (!u) return v;
  if (!v) return u;
  if (u->rank < v->rank) {
    u->pushdown();
    u->ch[1] = merge(u->ch[1], v);
    u->update();
    return u;
  } else {
    v->pushdown();
    v->ch[0] = merge(u, v->ch[0]);
    v->update();
    return v;
  }
}
pair<node *, node *> split(node *u, ui p) {
  if (!u) return {NULL, NULL};
  u->pushdown();
  if (u->p <= p) {
    auto [l, r] = split(u->ch[1], p);
    u->ch[1] = l;
    u->update();
    return {u, r};
  } else {
    auto [l, r] = split(u->ch[0], p);
    u->ch[0] = r;
    u->update();
    return {l, u};
  }
}
node *insert(node *u, ui p) {
  auto [l, r] = split(u, p);
  auto *v = l;
  while (v->ch[1]) v = v->ch[1];
  if (v->p == p) return merge(l, r);
  auto w = new node(p, v->n);
  v->n = p;
  w->avg = v->avg;
  v->update(), w->update();
  w = merge(l, w), w = merge(w, r);
  return w;
}
```

1.9 Trie

```

struct node {
  array<node *, alpha> ch;
  array<node *, sizf> fa;
  int dep;
  ll val;
  node() {
    for (auto &v : ch) v = NULL;
    for (auto &v : fa) v = NULL;
    dep = 0;
    val = 0;
  }
};
node *rt;
struct cmp {
  bool operator()(node *a, node *b) {
    if (a == b) return false;
    if (!a) return true;
    if (!b) return false;
    if (a->dep != b->dep) return a->dep > b->dep;
    for (int i = sizf - 1; i >= 0; i--)
      if (a->fa[i] != b->fa[i]) a = a->fa[i], b = b->fa[i];
    node *u = a->fa[0];
    for (int i = 0; i < alpha; i++) {
      if (a == u->ch[i]) return false;
      if (b == u->ch[i]) return true;
    }
  }
  bool operator()(pair<node *, int> a, pair<node *, int> b) {
```

```
    return cmp()(a.first, b.first);
  }
};
node *add(node *u, int n) {
  if (n / 10) u = add(u, n / 10);
  n %= 10;
  if (u->ch[n]) return u->ch[n];
  u->ch[n] = new node();
  auto v = u->ch[n];
  v->fa[0] = u;
  for (int i = 1; i < sizf; i++) v->fa[i] = v->fa[i - 1]->fa[i - 1];
  v->dep = u->dep + 1;
  v->val = (u->val * 10 + n) % mod;
  return v;
}
```

1.10 虚树

```
inline bool cmp(const int x, const int y) { return id[x] < id[y]; }

void build() {
  sort(h + 1, h + k + 1, cmp);
  sta[top = 1] = 1, g.sz = 0, g.head[1] = -1;
  // 1 号节点入栈，清空 1 号节点对应的邻接表，设置邻接表边数为 1
  for (int i = 1, l; i <= k; ++i)
    if (h[i] != 1) {
      // 如果 1 号节点是关键节点就不要重复添加
      l = lca(h[i], sta[top]);
      // 计算当前节点与栈顶节点的 LCA
      if (l != sta[top]) {
        // 如果 LCA 和栈顶元素不同，则说明当前节点不再当前栈所存的链上
        while (id[l] < id[sta[top - 1]])
          // 当次大节点的 Dfs 序大于 LCA 的 Dfs 序
          g.push(sta[top - 1], sta[top]), top--;
        // 把与当前节点所在的链不重合的链连接掉并且弹出
        if (id[l] > id[sta[top - 1]])
          // 如果 LCA 不等于次大节点（这里的大于其实和不等于没有区别）
          g.head[l] = -1, g.push(l, sta[top]), sta[top] = l;
        // 说明 LCA 是第一次入栈，清空其邻接表，连边后弹出栈顶元素，并将 LCA
        // 入栈
        else
          g.push(l, sta[top--]);
        // 说明 LCA 就是次大节点，直接弹出栈顶元素
      }
      g.head[h[i]] = -1, sta[++top] = h[i];
      // 当前节点必然是第一次入栈，清空邻接表并入栈
    }
  for (int i = 1; i < top; ++i)
    g.push(sta[i], sta[i + 1]);  // 剩余的最后一条链连接一下
  return;
}
```

2 博弈

2.1 国际象棋

```
#include <bits/stdc++.h>
using namespace std;
const char el = '\n';
typedef long long ll;
typedef unsigned long long ull;
array<ull, 64> rook, bishop, knight, queen, arch, chan, maha;
void write(ull t, ull p) {
  for (int i = 0; i < 8; i++, cout << el)
    for (int j = 0; j < 8; j++) {
      int x = i << 3 | j;
      cout << (p >> x & 1 ? "o " : t >> x & 1 ? "  " : "x ");
    }
  cout << el;
}
void init() {
  for (int i = 0; i < 8; i++)
    for (int j = 0; j < 8; j++) {
      auto &t = rook[i << 3 | j];
      t = 0;
      for (int x = 7; x >= 0; x--)
        for (int y = 7; y >= 0; y--) t = t << 1 | (x == i || y == j);
    }
  for (int i = 0; i < 8; i++)
    for (int j = 0; j < 8; j++) {
      auto &t = bishop[i << 3 | j];
      t = 0;
      for (int x = 7; x >= 0; x--)
        for (int y = 7; y >= 0; y--)
          t = t << 1 | (i + j == x + y || i - j == x - y);
    }
  for (int i = 0; i < 8; i++)
```

```
    for (int j = 0; j < 8; j++) {
      auto &t = knight[i << 3 | j];
      t = 0;
      for (int x = 7; x >= 0; x--)
        for (int y = 7; y >= 0; y--)
          t = t << 1 | ((x != i && y != j && abs(x - i) + abs(y - j) == 3) ||
                        x == i && y == j);
    }
  for (int i = 0; i < 64; i++) queen[i] = rook[i] | bishop[i];
  for (int i = 0; i < 64; i++) arch[i] = bishop[i] | knight[i];
  for (int i = 0; i < 64; i++) chan[i] = rook[i] | knight[i];
  for (int i = 0; i < 64; i++) maha[i] = queen[i] | knight[i];
}
array<ull, 64> piece(char ch) {
  switch (ch) {
    case 'R':
      return rook;
    case 'B':
      return bishop;
    case 'Q':
      return queen;
    case 'A':
      return arch;
    case 'C':
      return chan;
    case 'M':
      return maha;
  }
}
int cnt = 100;
bool win(string s, int p, ull safe, ull used) {
  if (p == s.size()) return false;
  bool res = false;
  const auto &pce = piece(s[p]);
  for (int i = 0; i < 64 && !res; i++)
    if ((safe >> i & 1) && !(used & pce[i]))
      res = res || !win(s, p + 1, safe & ~pce[i], used | (1ull << i));
  return res;
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(15);
  init();
  string s;
  cin >> s;
  cout << (win(s, 0, -1ull, 0) ? "Alice" : "Bob") << el;
  return 0;
}
// https://contest.ucup.ac/contest/1399/problem/7635
```

2.2 Nimber

```
#include <bits/stdc++.h>
using namespace std;
const char el = '\n';
typedef long long ll;
typedef unsigned long long ull;
array<ull, 64> rook, bishop, knight, queen, arch, chan, maha;
void write(ull t, ull p) {
  for (int i = 0; i < 8; i++, cout << el)
    for (int j = 0; j < 8; j++) {
      int x = i << 3 | j;
      cout << (p >> x & 1 ? "o " : t >> x & 1 ? "  " : "x ");
    }
  cout << el;
}
void init() {
  for (int i = 0; i < 8; i++)
    for (int j = 0; j < 8; j++) {
      auto &t = rook[i << 3 | j];
      t = 0;
      for (int x = 7; x >= 0; x--)
        for (int y = 7; y >= 0; y--) t = t << 1 | (x == i || y == j);
    }
  for (int i = 0; i < 8; i++)
    for (int j = 0; j < 8; j++) {
      auto &t = bishop[i << 3 | j];
      t = 0;
      for (int x = 7; x >= 0; x--)
        for (int y = 7; y >= 0; y--)
          t = t << 1 | (i + j == x + y || i - j == x - y);
    }
  for (int i = 0; i < 8; i++)
    for (int j = 0; j < 8; j++) {
      auto &t = knight[i << 3 | j];
      t = 0;
      for (int x = 7; x >= 0; x--)
        for (int y = 7; y >= 0; y--)
```

```
          t = t << 1 | ((x != i && y != j && abs(x - i) + abs(y - j) == 3) ||
                        x == i && y == j);
    }
  for (int i = 0; i < 64; i++) queen[i] = rook[i] | bishop[i];
  for (int i = 0; i < 64; i++) arch[i] = bishop[i] | knight[i];
  for (int i = 0; i < 64; i++) chan[i] = rook[i] | knight[i];
  for (int i = 0; i < 64; i++) maha[i] = queen[i] | knight[i];
}
array<ull, 64> piece(char ch) {
  switch (ch) {
    case 'R':
      return rook;
    case 'B':
      return bishop;
    case 'Q':
      return queen;
    case 'A':
      return arch;
    case 'C':
      return chan;
    case 'M':
      return maha;
  }
}
int cnt = 100;
bool win(string s, int p, ull safe, ull used) {
  if (p == s.size()) return false;
  bool res = false;
  const auto &pce = piece(s[p]);
  for (int i = 0; i < 64 && !res; i++)
    if ((safe >> i & 1) && !(used & pce[i]))
      res = res || !win(s, p + 1, safe & ~pce[i], used | (1ull << i));
  return res;
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(15);
  init();
  string s;
  cin >> s;
  cout << (win(s, 0, -1ull, 0) ? "Alice" : "Bob") << el;
  return 0;
}
// https://contest.ucup.ac/contest/1399/problem/7635
```

2.3 一般博弈

```

unordered_map<ll, int> result;
struct status {
  array<int, 10> a;
  status() { fill(a.begin(), a.end(), 0); }
  status flip() {
    status b;
    for (int i = 0; i < 5; i++) b.a[i] = a[i + 5];
    for (int i = 0; i < 5; i++) b.a[i + 5] = a[i];
    return b;
  }
  ll num() {
    ll res = 0;
    for (int i = 0; i < 10; i++) res = (res << 5) + a[i];
    return res;
  }
  vector<status> from() {
    vector<status> res;
    for (int i = 0; i < 5; i++)
      if (a[i])
        for (int j = 0; j < 5; j++)
          if (a[j + 5])
            if (i && j - i) {
              auto b = *this;
              b.a[j + 5]--;
              b.a[(j + 5 - i) % 5 + 5]++;
              res.push_back(b.flip());
            }
    // cout << "from " << res.size() << el;
    return res;
  }
  vector<status> to() {
    vector<status> res;
    for (int i = 0; i < 5; i++)
      if (a[i])
        for (int j = 0; j < 5; j++)
          if (a[j + 5])
            if (i && j) {
              auto b = *this;
              b.a[i]--;
```

```
              b.a[(i + j) % 5]++;
              res.push_back(b.flip());
            }
    // cout << "to: " << res.size() << el;
    return res;
  }
  bool win() {
    if (result.count(num())) return result[num()] == 1;
    for (auto v : to())
      if (result.count(v.num()) && result[v.num()] == -1) return true;
    return false;
  }
  bool lose() {
    if (result.count(num())) return result[num()] == -1;
    for (auto v : to())
      if (!result.count(v.num()) || result[v.num()] != 1) return false;
    return true;
  }
};
bool operator<(const status &a, const status &b) { return a.a < b.a; }
status read() {
  status a;
  for (int i = 0; i < 8; i++) {
    int t;
    cin >> t;
    a.a[t]++;
  }
  for (int i = 0; i < 8; i++) {
    int t;
    cin >> t;
    a.a[t + 5]++;
  }
  return a;
}

set<status> losestat(int t) {
  if (!t) {
    status tmp;
    tmp.a[5] = 8;
    return {tmp};
  }
  set<status> res;
  auto tmp = losestat(t - 1);
  for (auto v : tmp) {
    for (int i = 0; i < 5; i++) {
      auto t = v;
      t.a[i]++;
      res.insert(t);
    }
  }
  return res;
}
set<status> winstat(int t) {
  if (!t) {
    status tmp;
    tmp.a[0] = 8;
    return {tmp};
  }
  set<status> res;
  auto tmp = winstat(t - 1);
  for (auto v : tmp) {
    for (int i = 0; i < 5; i++) {
      auto t = v;
      t.a[i + 5]++;
      res.insert(t);
    }
  }
  return res;
}
void preprocess() {
  auto wst = winstat(8), lst = losestat(8);
  vector<status> win, lose;
  for (auto v : wst) {
    win.push_back(v);
    result[v.num()] = 1;
  }
  for (auto v : lst) {
    lose.push_back(v);
    result[v.num()] = -1;
  }
  while (win.size() || lose.size()) {
    if (win.size()) {
      for (auto wn : win)
        for (auto ls : wn.from()) {
          if (!result.count(ls.num()) && ls.lose()) {
            lose.push_back(ls);
            result[ls.num()] = -1;
          }
        }
      win.clear();
```

```
    } else {
      for (auto ls : lose)
        for (auto wn : ls.from()) {
          if (!result.count(wn.num()) && wn.win()) {
            win.push_back(wn);
            result[wn.num()] = 1;
          }
        }
      lose.clear();
    }
  }
}
```

3 图论

3.1 点分树

```
#include "bits/stdc++.h"
using namespace std;
const char el = '\n';
typedef long long ll;
const ll inf = 1e18;
struct graph {
  vector<vector<pair<int, ll>>> e;
  vector<ll> r;
  graph(int n) : e(n + 1), r(n + 1) {}
  void adde(int u, int v, ll w) {
    e[u].push_back({v, w});
    e[v].push_back({u, w});
  }
  vector<int> siz;
  int calcsiz(int u, int f) {
    siz[u] = 1;
    for (auto [v, w] : e[u])
      if (v != f && ~siz[v]) siz[u] += calcsiz(v, u);
    return siz[u];
  }
  vector<vector<pair<int, ll>>> fa;
  vector<vector<pair<ll, int>>> ch;
  void init(int u, int f, int rt, ll d) {
    fa[u].push_back({rt, d});
    ch[rt].push_back({r[u] - d, u});
    for (auto [v, w] : e[u])
      if (v != f && ~siz[v]) init(v, u, rt, d + w);
  }
  int split(int u) {
    int n = calcsiz(u, u) / 2;
    while (true) {
      bool flg = true;
      for (auto [v, w] : e[u])
        if (siz[v] < siz[u] && siz[v] > n) {
          u = v;
          flg = false;
          break;
        }
      if (flg) break;
    }
    siz[u] = -1;
    init(u, u, u, 0);
    sort(ch[u].begin(), ch[u].end());
    for (auto [v, w] : e[u])
      if (~siz[v]) v = split(v);
    return u;
  }
  int rt;
  void ctrdecomp() {
    int n = e.size();
    siz.assign(n, 0);
    fa.assign(n, {});
    ch.assign(n, {});
    rt = split(1);
  }
  vector<int> solve() {
    vector<int> dis(e.size(), -1);
    dis[1] = 0;
    queue<int> que;
    que.push(1);
    while (que.size()) {
      int u = que.front();
      que.pop();
      for (auto [f, d] : fa[u]) {
        while (ch[f].size()) {
          auto [w, v] = ch[f].back();
          if (d > w) break;
          ch[f].pop_back();
          if (~dis[v]) continue;
          que.push(v);
          dis[v] = dis[u] + 1;
```

```
        }
      }
    }
    return dis;
  }
};
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(15);
  int tt;
  cin >> tt;
  while (tt--) {
    int n;
    cin >> n;
    graph g(n);
    for (int i = 2; i <= n; i++) cin >> g.r[i];
    for (int i = 1; i < n; i++) {
      ll u, v, w;
      cin >> u >> v >> w;
      g.adde(u, v, w);
    }
    g.ctrdecomp();
    auto res = g.solve();
    for (int i = 2; i <= n; i++) cout << res[i] << ' ';
    cout << el;
  }
  return 0;
}
```

3.2 树链剖分

```
#include <bits/stdc++.h>
using namespace std;
struct graph {
  vector<vector<int>> e;
  vector<int> fa, dep, siz, hvs, top, dfn, rnk;
  vector<bit> seg;
  graph(int n) : seg(maxd, n) {
    e.resize(n + 1);
    fa.resize(n + 1);
    dep.resize(n + 1);
    siz.resize(n + 1);
    hvs.resize(n + 1);
    top.resize(n + 1);
    dfn.resize(n + 1);
    rnk.resize(n + 1);
  }
  void adde(int u, int v) {
    e[u].push_back(v);
    e[v].push_back(u);
  }
  void subts(int u, int f, int d) {
    fa[u] = f, dep[u] = d;
    siz[u] = 1;
    hvs[u] = -1;
    for (auto v : e[u])
      if (v != f) {
        subts(v, u, d + 1);
        siz[u] += siz[v];
        if (!~hvs[u] || siz[v] > siz[hvs[u]]) hvs[u] = v;
      }
  }
  int tot;
  void ordfs(int u, int t) {
    top[u] = t;
    tot++, rnk[tot] = u, dfn[u] = tot;
    if (~hvs[u]) ordfs(hvs[u], t);
    for (auto v : e[u])
      if (v != fa[u] && v != hvs[u]) ordfs(v, v);
  }
  void hvydecomp(int rt) {
    subts(rt, -1, 0);
    tot = 0;
    ordfs(rt, rt);
  }
  void modify(int u, int v, int k, int d) {
    while (top[u] != top[v]) {
      if (dep[top[u]] < dep[top[v]]) swap(u, v);
      seg[d].add(dfn[top[u]], dfn[u], k);
      u = fa[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v);
    seg[d].add(dfn[v], dfn[u], k);
    u = v;
    while (d--) {
      seg[d].add(dfn[u], dfn[u], k);
      u = fa[u];
      seg[d].add(dfn[u], dfn[u], k);
```

```
    }
  }
  int query(int u) {
    int ans = 0;
    for (int i = 0; i < maxd; i++) {
      ans += seg[i].query(dfn[u]);
      u = fa[u];
    }
    return ans;
  }
};
```

3.3 费用流

```
#pragma once

#include <algorithm>
#include <optional>
#include <queue>
#include <vector>

namespace costflow {

typedef long long flow_t;
typedef long long cost_t;

const flow_t inf_flow = 1e18;
const cost_t inf_cost = 1e18;

struct CostEdge {
  int from, to;
  cost_t cost;
  flow_t cap, low = 0, flow = 0;
};

int num_node(const std::vector<CostEdge> &edges) {
  int n = 0;
  for (const auto &e : edges) n = std::max({n, e.from, e.to});
  return n;
}
std::pair<flow_t, cost_t> get_flow(const std::vector<CostEdge> &edges, int s) {
  flow_t flow = 0;
  cost_t cost = 0;
  for (const auto &e : edges) {
    if (e.from == s) flow += e.flow;
    cost += e.flow * e.cost;
  }
  return {flow, cost};
}

struct CostFlow {
  struct Edge {
    int from, to;
    cost_t cost;
    flow_t cap;
  };
  int n;
  std::vector<std::vector<int>> eid;
  std::vector<Edge> edge;
  void build(const std::vector<CostEdge> &edges) {
    n = num_node(edges);
    eid.assign(n + 1, {});
    edge.clear();
    int num_edges = 0;
    for (const auto &e : edges) {
      eid[e.from].push_back(num_edges++);
      edge.push_back({e.from, e.to, e.cost, e.cap - e.flow});
      eid[e.to].push_back(num_edges++);
      edge.push_back({e.to, e.from, -e.cost, e.flow});
    }
  }
  std::vector<cost_t> dis;
  std::vector<int> pre;
  bool spfa(int s, int t) {
    if (s > n || t > n) return false;
    dis.assign(n + 1, inf_cost);
    pre.assign(n + 1, 0);
    std::vector<bool> inque(n + 1);
    std::queue<int> que;
    dis[s] = 0;
    que.push(s);
    inque[s] = true;
    while (que.size()) {
      int u = que.front();
      // cerr << 'u' << ' ' << u << endl;
      que.pop();
      inque[u] = false;
      for (auto i : eid[u]) {
        const auto &e = edge[i];
```

```
        if (e.cap && dis[e.to] > dis[u] + e.cost) {
          dis[e.to] = dis[u] + e.cost;
          pre[e.to] = i;
          if (!inque[e.to]) {
            que.push(e.to);
            inque[e.to] = true;
          }
        }
      }
    }
    return dis[t] < inf_cost;
  }
  std::pair<flow_t, cost_t> maxflow(int s, int t) {
    flow_t flow = 0;
    cost_t cost = 0;
    while (spfa(s, t)) {
      flow_t detf = inf_flow;
      cost_t detc = 0;
      for (int u = t, i = pre[u]; u != s; u = edge[i].from, i = pre[u]) {
        detf = std::min(detf, edge[i].cap);
        detc += edge[i].cost;
      }
      for (int u = t, i = pre[u]; u != s; u = edge[i].from, i = pre[u]) {
        edge[i].cap -= detf;
        edge[i ^ 1].cap += detf;
      }
      flow += detf;
      cost += detf * detc;
    }
    return {flow, cost};
  }
  std::vector<CostEdge> to_edge() {
    std::vector<CostEdge> edges;
    for (int i = 0; i < edge.size(); i += 2)
      edges.push_back({
          .from = edge[i].from,
          .to = edge[i].to,
          .cost = edge[i].cost,
          .cap = edge[i].cap + edge[i ^ 1].cap,
          .flow = edge[i ^ 1].cap,
      });
    return edges;
  }
};

struct Processor {
  std::vector<bool> neg;
  std::vector<flow_t> low;
  std::vector<flow_t> excess;
  void init(std::vector<CostEdge> &edges) {
    int n = num_node(edges);
    neg.clear();
    neg.reserve(edges.size());
    low.clear();
    low.reserve(edges.size());
    excess.assign(n + 1, 0);
  }
  void rmv_low(std::vector<CostEdge> &edges) {
    for (auto &e : edges) {
      low.push_back(e.low);
      if (e.flow >= e.low) {
        e.flow -= e.low;
      } else {
        excess[e.from] -= e.low - e.flow;
        excess[e.to] += e.low - e.flow;
        e.flow = 0;
      }
      e.cap -= e.low;
      e.low = 0;
    }
  }
  void add_low(std::vector<CostEdge> &edges) {
    reverse(low.begin(), low.end());
    for (auto &e : edges) {
      e.low = low.back();
      e.flow += e.low;
      e.cap += e.low;
      low.pop_back();
    }
  }
  void rmv_neg(std::vector<CostEdge> &edges) {
    for (auto &e : edges) {
      neg.push_back(e.cost < 0);
      if (e.cost < 0) {
        excess[e.from] -= e.cap - e.flow;
        excess[e.to] += e.cap - e.flow;
        e.flow = e.cap;
      }
      if (e.cost > 0) {
        excess[e.from] += e.flow;
```

```
        excess[e.to] -= e.flow;
        e.flow = 0;
      }
    }
  }
};

bool excess_flow(std::vector<CostEdge> &edges,
                 const std::vector<flow_t> &excess) {
  int n = num_node(edges), m = edges.size();
  for (int i = 1; i <= n; i++) {
    if (excess[i] > 0)
      edges.push_back({.from = n + 1, .to = i, .cost = 0, .cap = excess[i]});
    if (excess[i] < 0)
      edges.push_back({.from = i, .to = n + 2, .cost = 0, .cap = -excess[i]});
  }
  CostFlow g;
  g.build(edges);
  g.maxflow(n + 1, n + 2);
  edges = g.to_edge();
  for (int i = m; i < edges.size(); i++)
    if (edges[i].flow != edges[i].cap) return false;
  edges.resize(m);
  return true;
}

std::optional<std::pair<flow_t, cost_t>> feasible_flow(
    std::vector<CostEdge> &edges, int s = 0, int t = 0) {
  if (s && t) edges.push_back({.from = t, .to = s, .cost = 0, .cap = inf_flow});
  Processor p;
  p.init(edges);
  p.rmv_low(edges);
  p.rmv_neg(edges);
  if (!excess_flow(edges, p.excess)) return std::nullopt;
  if (s && t) edges.pop_back();
  p.add_low(edges);
  return get_flow(edges, s);
}

std::optional<std::pair<flow_t, cost_t>> maximum_flow(
    std::vector<CostEdge> &edges, int s, int t) {
  edges.push_back({.from = t, .to = s, .cost = 0, .cap = inf_flow});
  Processor p;
  p.init(edges);
  p.rmv_low(edges);
  p.rmv_neg(edges);
  if (!excess_flow(edges, p.excess)) return std::nullopt;
  edges.pop_back();
  CostFlow g;
  g.build(edges);
  g.maxflow(s, t);
  edges = g.to_edge();
  p.add_low(edges);
  return get_flow(edges, s);
}

std::optional<std::pair<flow_t, cost_t>> minimum_flow(
    std::vector<CostEdge> &edges, int s, int t) {
  edges.push_back({.from = t, .to = s, .cost = 0, .cap = inf_flow});
  Processor p;
  p.init(edges);
  p.rmv_low(edges);
  p.rmv_neg(edges);
  if (!excess_flow(edges, p.excess)) return std::nullopt;
  edges.pop_back();
  CostFlow g;
  for (auto &e : edges) e.cost = -e.cost;
  Processor q;
  q.rmv_neg(edges);
  excess_flow(edges, q.excess);
  g.build(edges);
  g.maxflow(t, s);
  edges = g.to_edge();
  for (auto &e : edges) e.cost = -e.cost;
  p.add_low(edges);
  return get_flow(edges, s);
}

}  // namespace costflow
```

3.4 点分树

```
#include "bits/stdc++.h"
using namespace std;
const char el = '\n';
typedef long long ll;
const ll inf = 1e18;
struct graph {
  vector<vector<pair<int, ll>>> e;
```

```
  vector<ll> r;
  graph(int n) : e(n + 1), r(n + 1) {}
  void adde(int u, int v, ll w) {
    e[u].push_back({v, w});
    e[v].push_back({u, w});
  }
  vector<int> siz;
  int calcsiz(int u, int f) {
    siz[u] = 1;
    for (auto [v, w] : e[u])
      if (v != f && ~siz[v]) siz[u] += calcsiz(v, u);
    return siz[u];
  }
  vector<vector<pair<int, ll>>> fa;
  vector<vector<pair<ll, int>>> ch;
  void init(int u, int f, int rt, ll d) {
    fa[u].push_back({rt, d});
    ch[rt].push_back({r[u] - d, u});
    for (auto [v, w] : e[u])
      if (v != f && ~siz[v]) init(v, u, rt, d + w);
  }
  int split(int u) {
    int n = calcsiz(u, u) / 2;
    while (true) {
      bool flg = true;
      for (auto [v, w] : e[u])
        if (siz[v] < siz[u] && siz[v] > n) {
          u = v;
          flg = false;
          break;
        }
      if (flg) break;
    }
    siz[u] = -1;
    init(u, u, u, 0);
    sort(ch[u].begin(), ch[u].end());
    for (auto [v, w] : e[u])
      if (~siz[v]) v = split(v);
    return u;
  }
  int rt;
  void ctrdecomp() {
    int n = e.size();
    siz.assign(n, 0);
    fa.assign(n, {});
    ch.assign(n, {});
    rt = split(1);
  }
  vector<int> solve() {
    vector<int> dis(e.size(), -1);
    dis[1] = 0;
    queue<int> que;
    que.push(1);
    while (que.size()) {
      int u = que.front();
      que.pop();
      for (auto [f, d] : fa[u]) {
        while (ch[f].size()) {
          auto [w, v] = ch[f].back();
          if (d > w) break;
          ch[f].pop_back();
          if (~dis[v]) continue;
          que.push(v);
          dis[v] = dis[u] + 1;
        }
      }
    }
    return dis;
  }
};
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(15);
  int tt;
  cin >> tt;
  while (tt--) {
    int n;
    cin >> n;
    graph g(n);
    for (int i = 2; i <= n; i++) cin >> g.r[i];
    for (int i = 1; i < n; i++) {
      ll u, v, w;
      cin >> u >> v >> w;
      g.adde(u, v, w);
    }
    g.ctrdecomp();
    auto res = g.solve();
    for (int i = 2; i <= n; i++) cout << res[i] << ' ';
    cout << el;
```

```
  }
  return 0;
}
```

3.5 强连通分量分解

```
#include <bits/stdc++.h>
using namespace std;
struct graph {
  vector<vector<int>> e, r;
  graph(int n) {
    e.resize(n + 1);
    r.resize(n + 1);
  }
  void adde(int u, int v) {
    e[u].push_back(v);
    r[v].push_back(u);
  }
  vector<bool> vis;
  vector<int> ord;
  vector<int> col;
  void travel(int u, int f) {
    vis[u] = true;
    for (auto v : e[u])
      if (!vis[v]) travel(v, u);
    ord.push_back(u);
  }
  void color(int u, int f, int c) {
    col[u] = c;
    for (auto v : r[u])
      if (!col[v]) color(v, u, c);
  }
  vector<int> decomp() {
    vis.assign(e.size(), false);
    ord.clear();
    for (int i = 1; i < e.size(); i++)
      if (!vis[i]) travel(i, i);
    reverse(ord.begin(), ord.end());
    col.assign(e.size(), 0);
    for (auto v : ord)
      if (!col[v]) color(v, v, v);
    return col;
  }
};
```

3.6 拓扑排序

```
#include <bits/stdc++.h>
using namespace std;
struct graph {
  vector<vector<int>> e;
  graph(int n) : e(n + 1) {}
  void adde(int u, int v) { e[u].push_back(v); }
  optional<vector<int>> toposort() {
    int n = e.size() - 1;
    vector<int> d(n + 1);
    queue<int> que;
    for (int u = 1; u <= n; u++)
      for (auto v : e[u]) d[v]++;
    for (int i = 1; i <= n; i++)
      if (!d[i]) que.push(i);
    vector<int> res;
    while (!que.empty()) {
      auto u = que.front();
      res.push_back(u);
      que.pop();
      for (auto v : e[u]) {
        if (!--d[v]) que.push(v);
      }
    }
    if (res.size() != n) return nullopt;
    return res;
  }
};
```

3.7 二分图匹配

```
#pragma once

#include <vector>

#include "maxflow.h"

namespace match {

std::vector<std::pair<int, int>> max_match(
```

```
    const std::vector<std::pair<int, int>> &edges) {
  int n = 1, m = 1;
  for (auto [l, r] : edges) {
    n = std::max(n, l);
    m = std::max(m, r);
  }
  std::vector<maxflow::FlowEdge> fedge;
  int s = n + m + 1, t = n + m + 2;
  for (int i = 1; i <= n; i++) fedge.push_back({.from = s, .to = i, .cap = 1});
  for (int i = 1; i <= m; i++)
    fedge.push_back({.from = i + n, .to = t, .cap = 1});
  for (auto [l, r] : edges) fedge.push_back({.from = l, .to = r + n, .cap = 1});
  maxflow::maximum_flow(fedge, s, t);
  std::vector<std::pair<int, int>> res;
  for (auto e : fedge)
    if (e.from != s && e.to != t && e.flow == 1)
      res.push_back({e.from, e.to - n});
  return res;
}

void dfs(int u, std::vector<bool> &vl, std::vector<bool> &vr,
         std::vector<std::vector<int>> &ltr, std::vector<int> &rtl) {
  if (vl[u]) return;
  vl[u] = true;
  for (auto v : ltr[u]) {
    vr[v] = true;
    dfs(rtl[v], vl, vr, ltr, rtl);
  }
}

std::pair<std::vector<int>, std::vector<int>> min_cover(
    const std::vector<std::pair<int, int>> &edges) {
  int n = 1, m = 1;
  for (auto [l, r] : edges) {
    n = std::max(n, l);
    m = std::max(m, r);
  }
  auto match = max_match(edges);
  std::vector<std::vector<int>> ltr(n + 1);
  std::vector<int> rtl(m + 1);
  std::vector<bool> vis(n + 1);
  for (auto [l, r] : match) {
    rtl[r] = l;
    vis[l] = true;
  }
  for (auto [l, r] : edges) ltr[l].push_back(r);
  std::vector<bool> vl(n + 1), vr(m + 1);
  for (int i = 1; i <= n; i++)
    if (!vis[i]) dfs(i, vl, vr, ltr, rtl);
  std::pair<std::vector<int>, std::vector<int>> res;
  for (int i = 1; i <= n; i++)
    if (!vl[i]) res.first.push_back(i);
  for (int i = 1; i <= m; i++)
    if (vr[i]) res.second.push_back(i);
  return res;
}

}  // namespace match
```

3.8 最大流

```
#pragma once

#include <algorithm>
#include <optional>
#include <queue>
#include <vector>

namespace maxflow {

typedef long long flow_t;
const flow_t inf_flow = 1e18;
const int inf_dep = 1e9;

struct FlowEdge {
  int from, to;
  flow_t cap, low = 0, flow = 0;
};

int num_node(const std::vector<FlowEdge> &edges) {
  int n = 0;
  for (const auto &e : edges) n = std::max({n, e.from, e.to});
  return n;
}

flow_t get_flow(const std::vector<FlowEdge> &edges, int s) {
  flow_t flow = 0;
  for (const auto &e : edges) {
    if (e.from == s) flow += e.flow;
```

```
  }
  return flow;
}

struct MaxFlow {
  struct Edge {
    int from, to;
    flow_t cap;
  };
  int n;
  std::vector<std::vector<int>> eid;
  std::vector<Edge> edge;
  void build(const std::vector<FlowEdge> &edges) {
    n = num_node(edges);
    eid.assign(n + 1, {});
    edge.clear();
    int num_edges = 0;
    for (const auto &e : edges) {
      eid[e.from].push_back(num_edges++);
      edge.push_back({e.from, e.to, e.cap - e.flow});
      eid[e.to].push_back(num_edges++);
      edge.push_back({e.to, e.from, e.flow});
    }
  }

  std::vector<int> dis;
  std::vector<int> cur;
  bool bfs(int s, int t) {
    if (s > n || t > n) return false;
    dis.assign(n + 1, inf_dep);
    cur.assign(n + 1, 0);
    std::queue<int> que;
    dis[s] = 0;
    que.push(s);
    while (que.size()) {
      int u = que.front();
      que.pop();
      for (auto i : eid[u]) {
        const auto &e = edge[i];
        if (e.cap && dis[e.to] > dis[u] + 1) {
          dis[e.to] = dis[u] + 1;
          que.push(e.to);
        }
      }
    }
    return dis[t] < inf_dep;
  }

  flow_t dfs(int s, int t, flow_t flim) {
    if (s == t) return flim;
    flow_t flow = 0;
    for (int &i = cur[s]; i < eid[s].size() && flow < flim; i++) {
      auto &e = edge[eid[s][i]];
      if (dis[e.to] == dis[s] + 1 && e.cap) {
        auto detf = dfs(e.to, t, std::min(flim - flow, e.cap));
        flow += detf;
        e.cap -= detf;
        edge[eid[s][i] ^ 1].cap += detf;
      }
      if (flow == flim) break;
    }
    return flow;
  }
  flow_t maxflow(int s, int t) {
    flow_t flow = 0;
    while (bfs(s, t)) {
      flow += dfs(s, t, inf_flow);
    }
    return flow;
  }

  std::vector<FlowEdge> to_edge() {
    std::vector<FlowEdge> edges;
    for (int i = 0; i < edge.size(); i += 2)
      edges.push_back({
          .from = edge[i].from,
          .to = edge[i].to,
          .cap = edge[i].cap + edge[i ^ 1].cap,
          .low = 0,
          .flow = edge[i ^ 1].cap,
      });
    return edges;
  }
};

struct Processor {
  std::vector<bool> neg;
  std::vector<flow_t> low;
  std::vector<flow_t> excess;
  void init(std::vector<FlowEdge> &edges) {
```

```
    int n = num_node(edges);
    neg.clear();
    neg.reserve(edges.size());
    low.clear();
    low.reserve(edges.size());
    excess.assign(n + 1, 0);
  }
  void rmv_low(std::vector<FlowEdge> &edges) {
    for (auto &e : edges) {
      low.push_back(e.low);
      if (e.flow >= e.low) {
        e.flow -= e.low;
      } else {
        excess[e.from] -= e.low - e.flow;
        excess[e.to] += e.low - e.flow;
        e.flow = 0;
      }
      e.cap -= e.low;
      e.low = 0;
    }
  }
  void add_low(std::vector<FlowEdge> &edges) {
    reverse(low.begin(), low.end());
    for (auto &e : edges) {
      e.low = low.back();
      e.flow += e.low;
      e.cap += e.low;
      low.pop_back();
    }
  }
};

bool excess_flow(std::vector<FlowEdge> &edges,
                 const std::vector<flow_t> &excess) {
  int n = num_node(edges), m = edges.size();
  for (int i = 1; i <= n; i++) {
    if (excess[i] > 0)
      edges.push_back({.from = n + 1, .to = i, .cap = excess[i]});
    if (excess[i] < 0)
      edges.push_back({.from = i, .to = n + 2, .cap = -excess[i]});
  }
  MaxFlow g;
  g.build(edges);
  g.maxflow(n + 1, n + 2);
  edges = g.to_edge();
  for (int i = m; i < edges.size(); i++)
    if (edges[i].flow != edges[i].cap) return false;
  edges.resize(m);
  return true;
}

std::optional<flow_t> feasible_flow(std::vector<FlowEdge> &edges, int s = 0,
                                    int t = 0) {
  if (s && t) edges.push_back({.from = t, .to = s, .cap = inf_flow});
  Processor p;
  p.init(edges);
  p.rmv_low(edges);
  if (!excess_flow(edges, p.excess)) return std::nullopt;
  if (s && t) edges.pop_back();
  p.add_low(edges);
  return get_flow(edges, s);
}

std::optional<flow_t> maximum_flow(std::vector<FlowEdge> &edges, int s, int t) {
  edges.push_back({.from = t, .to = s, .cap = inf_flow});
  Processor p;
  p.init(edges);
  p.rmv_low(edges);
  if (!excess_flow(edges, p.excess)) return std::nullopt;
  edges.pop_back();
  MaxFlow g;
  g.build(edges);
  g.maxflow(s, t);
  edges = g.to_edge();
  p.add_low(edges);
  return get_flow(edges, s);
}

std::optional<flow_t> minimum_flow(std::vector<FlowEdge> &edges, int s, int t) {
  edges.push_back({.from = t, .to = s, .cap = inf_flow});
  Processor p;
  p.init(edges);
  p.rmv_low(edges);
  if (!excess_flow(edges, p.excess)) return std::nullopt;
  edges.pop_back();
  MaxFlow g;
  Processor q;
  excess_flow(edges, q.excess);
  g.build(edges);
  g.maxflow(t, s);
```

```
  edges = g.to_edge();
  p.add_low(edges);
  return get_flow(edges, s);
}

}  // namespace maxflow
```

3.9 换根 DP

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct graph {
  vector<vector<int>> e;
  ll k, c;
  graph(int n) { e.resize(n + 1); }
  void adde(int u, int v) {
    e[u].push_back(v);
    e[v].push_back(u);
  }
  vector<int> dis, dep;
  ll ans;
  void setup(int u, int f) {
    dep[u] = 0;
    for (auto v : e[u])
      if (v != f) {
        dis[v] = dis[u] + 1;
        setup(v, u);
        dep[u] = max(dep[u], dep[v] + 1);
      }
  }
  void reroot(int u, int f) {
    int mxm1 = -1, mxm2 = -1;
    for (auto v : e[u]) {
      mxm2 = max(mxm2, dep[v]);
      if (mxm2 > mxm1) swap(mxm1, mxm2);
    }
    dep[u] = mxm1 + 1;
    ans = max(ans, dep[u] * k - dis[u] * c);
    for (auto v : e[u])
      if (v != f) {
        dep[u] = (dep[v] == mxm1 ? mxm2 : mxm1) + 1;
        reroot(v, u);
      }
  }
  ll solve(int s) {
    dis.resize(e.size());
    dep.resize(e.size());
    dep[s] = 0;
    setup(s, s);
    ans = 0;
    reroot(s, s);
    return ans;
  }
};
```

3.10 树哈希

```
/**
 * Author: Boboge adapted from peehs_moorhsum
 * Date: 23-02-06
 * Description: return the hash value of every subtree.
 * Time: O(N).
 * Status: tested on https://uoj.ac/problem/763
 */
std::mt19937_64 rnd(std::chrono::steady_clock::now().time_since_epoch().count());

struct treeHash {
    using ull = unsigned long long;
    ull bas = rnd();

    ull H(ull x) {
        return x * x * x * 114514 + 19260817;
    }

    ull F(ull x) {
        return H(x & ((1ll << 32) - 1)) + H(x >> 32);
    }

    std::vector<ull> h;

    treeHash(std::vector<std::vector<int>> &adj, int rt = 0) : h(adj.size()) {
        auto dfs = [&](auto dfs, int u, int fa) -> void {
            h[u] = bas;
            for (int v: adj[u]) {
                if (v == fa) continue;
                dfs(dfs, v, u);
                h[u] += F(h[v]);
```

```
            }
        };
        dfs(dfs, rt, -1);
    }
};
```

4 数学

4.1 数论分块

```
for (ll l = 1, r; l <= n; l = r + 1) {
  r = n / (n / l);
  ...  // for l<=i<=r, n/i = n/l
}
/*
https://qoj.ac/contest/1103/problem/5503
*/
```

4.2 FWT

```

const int k = 17, n = 1 << k;
const int mod = 1e9 + 7, inv2 = mod / 2 + 1;
struct modint {
  int n;
  modint(int n = 0) : n(n) {}
};
modint operator+(modint a, modint b) {
  return (a.n += b.n) >= mod ? a.n - mod : a.n;
}
modint operator-(modint a, modint b) {
  return (a.n -= b.n) < 0 ? a.n + mod : a.n;
}
modint operator*(modint a, modint b) { return 1ll * a.n * b.n % mod; }
vector<modint> fib(n);
vector<modint> fwt(vector<modint> a, void (*opr)(modint &, modint &)) {
  // cout << 1 << el;
  int n = a.size();
  for (int i = 0; 1 << i < n; i++)
    for (int j = 0; j < n; j++)
      if (j >> i & 1) opr(a[j - (1 << i)], a[j]);
  // cout << 2 << el;
  return a;
}
void fwtand(modint &a, modint &b) { a = a + b; }
void revand(modint &a, modint &b) { a = a - b; }
void fwtor(modint &a, modint &b) { b = b + a; }
void revor(modint &a, modint &b) { b = b - a; }
void fwtxor(modint &a, modint &b) {
  modint x = a + b, y = a - b;
  a = x, b = y;
}
void revxor(modint &a, modint &b) {
  modint x = a + b, y = a - b;
  a = x * inv2, b = y * inv2;
}
vector<modint> add(vector<modint> a, vector<modint> b) {
  for (int i = 0; i < a.size(); i++) a[i] = a[i] + b[i];
  return a;
}
vector<modint> mul(vector<modint> a, vector<modint> b) {
  for (int i = 0; i < a.size(); i++) a[i] = a[i] * b[i];
  return a;
}
vector<int> popc(n);
vector<modint> filter(vector<modint> a, int k) {
  for (int i = 0; i < n; i++)
    if (popc[i] != k) a[i] = 0;
  return a;
}
int main() {
  init();
  fib[0] = 0, fib[1] = 1;
  for (int i = 2; i < n; i++) fib[i] = fib[i - 1] + fib[i - 2];
  popc[0] = 0;
  for (int i = 1; i < n; i++) popc[i] = popc[i >> 1] + (i & 1);
  int m;
  cin >> m;
  vector<int> s(m);
  for (auto &v : s) cin >> v;
  vector<modint> a(n);
  for (auto v : s) a[v] = a[v] + 1;
  auto mid = mul(a, fib),
       rhs = mul(fwt(mul(fwt(a, fwtxor), fwt(a, fwtxor)), revxor), fib);
  vector b(k + 1, vector<modint>());
  vector c(k + 1, vector<modint>(n));
  vector<modint> lhs(n);
```

```
  for (int i = 0; i <= k; i++) b[i] = fwt(filter(a, i), fwtor);
  for (int i = 0; i <= k; i++)
    for (int j = 0; i + j <= k; j++) c[i + j] = add(c[i + j], mul(b[i], b[j]));
  for (int i = 0; i <= k; i++)
    lhs = add(lhs, filter(mul(fwt(c[i], revor), fib), i));
  auto res = fwt(mul(mul(fwt(lhs, fwtand), fwt(mid, fwtand)), fwt(rhs, fwtand)),
                 revand);
  modint ans;
  for (int i = 0; i < k; i++) ans = ans + res[1 << i];
  cout << ans.n << el;
  return 0;
}
```

4.3 高斯消元

```

const ll mod = 1e9 + 7, pri = 5;
const array<int, pri> inv = {0, 1, 3, 2, 4};
ll qpow(ll a, ll b, ll m = mod) {
  ll res = 1;
  while (b) {
    if (b & 1) res = res * a % m;
    a = a * a % m, b >>= 1;
  }
  return res;
}
vector<int> read() {
  string s;
  cin >> s;
  vector<int> a(s.size());
  for (int i = 0; i < s.size(); i++) a[i] = s[i] - 'a';
  return a;
}
void flip(vector<vector<int>> &a) {
  int n = a.size(), m = a[0].size();
  vector b(m, vector<int>(n));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++) b[j][i] = a[i][j];
  swap(a, b);
}
int gaussian(vector<vector<int>> &a) {
  int n = a.size(), m = a[0].size();
  int r = 0;
  for (int j = 0; j < m; j++) {
    int p = -1;
    for (int i = r; i < n; i++)
      if (a[i][j]) p = i;
    if (p == -1) continue;
    swap(a[r], a[p]);
    int t = inv[a[r][j]];
    for (int k = 0; k < m; k++) a[r][k] = a[r][k] * t % pri;
    for (int i = 0; i < n; i++)
      if (i != r) {
        int t = a[i][j];
        for (int k = 0; k < m; k++)
          a[i][k] = ((a[i][k] - a[r][k] * t) % pri + pri) % pri;
      }
    r++;
  }
  a.resize(r);
  return r;
}
bool solve(vector<vector<int>> &a, vector<int> &b) {
  int n = a.size(), m = a[0].size();
  for (int i = 0; i < n; i++) {
    int p = 0;
    while (!a[i][p]) p++;
    int t = b[p];
    for (int j = 0; j < m; j++) b[j] = ((b[j] - a[i][j] * t) % pri + pri) % pri;
  }
  for (auto v : b)
    if (v % pri) return false;
  return true;
}
```

4.4 Min25 筛

```
const int det = 100;
const ll mod = 1e9 + 7;
template <typename T>
struct min25 {
  ll n, m;
  vector<ll> p;
  vector<T> s;
  void sieve(ll n) {
    p = {1}, s = {T()};
    vector<int> minf(n);
    for (ll i = 2; i < n; i++) {
```

```
      if (!minf[i]) {
        minf[i] = p.size();
        p.push_back(i);
        s.push_back(s.back() + T::init(i));
      }
      for (ll j = 1; j <= minf[i] && i * p[j] < n; j++) minf[i * p[j]] = j;
    }
  }
  vector<ll> lis;
  vector<int> le, ge;
  vector<T> g;
  void init(ll n) {
    le.resize(m + det), ge.resize(m + det);
    for (ll i = 1, j; i <= n; i = n / j + 1) {
      j = n / i;
      int k = lis.size();
      lis.push_back(j);
      (j <= m ? le[j] : ge[n / j]) = k;
      g.push_back(T::plug(j));
    }
  }
  int id(ll v) { return v <= m ? le[v] : ge[n / v]; }
  vector<ll> f;
  void calcp() {
    for (int k = 1; k < p.size(); k++) {
      ll pk = p[k], sp = pk * pk;
      for (int i = 0; i < lis.size() && lis[i] >= sp; i++) {
        int j = id(lis[i] / pk);
        g[i] = g[i] - (g[j] - s[k - 1]) * T::item(pk);
      }
    }
    for (auto &v : g) f.push_back(v.val());
  }
  min25(ll n) : n(n) {
    m = sqrt(n);
    sieve(m + det);
    init(n);
    calcp();
  }
  ll query(ll n, int k = 1) {
    if (n < p[k] || n <= 1) return 0;
    const int i = id(n);
    ll ans = f[i] - s[k - 1].val();
    for (int i = k; i < p.size() && 1ll * p[i] * p[i] <= n; i++) {
      ll mul = p[i];
      for (int c = 1; mul * p[i] <= n; c++, mul *= p[i])
        ans += T::f(p[i], c) * query(n / mul, i + 1) % mod + T::f(p[i], c + 1);
    }
    return ans % mod;
  }
};
struct ez {
  ll a, b;
  ez(ll a = 0, ll b = 0) : a(a % mod), b(b % mod) {}
  ez operator+(ez r) { return {(a + r.a) % mod, (b + r.b) % mod}; }
  ez operator-(ez r) { return {(a - r.a) % mod, (b - r.b) % mod}; }
  ez operator*(ez r) { return {a * r.a % mod, b * r.b % mod}; }
  ll val() { return (a + b) % mod; }
  static ll f(ll p, ll c) { return p ^ c; }
  static ez init(ll p) { return {-1, p}; }
  static ez plug(ll p) {
    p %= mod;
    return {-p + 1, (p + 2) * (p - 1) % mod * (mod + 1 >> 1) % mod};
  }
  static ez item(ll p) { return {1, p}; }
};
```

4.5 Pollard—ρ

```
#include <bits/stdc++.h>
using namespace std;
using i64 = long long;
using i128 = __int128;
i64 fpow(i64 a, i64 t, i64 mod){
        i64 r = 1;
        for (; t; t >>= 1, a = (i128)a * a % mod) {
                if (t & 1) {
                        r = (i128)r * a % mod;
                }
        }
        return r;
}
i64 gcd(i64 a, i64 b){
        #define ctz __builtin_ctzll
        int shift = ctz(a | b);
        b >>= ctz(b);
        while (a) {
                a >>= ctz(a);
                if (a < b) swap(a, b);
```

```
                a -= b;
        }
        return b << shift;
}
bool check_prime(i64 n){
        static const int jp[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
        if (n == 1) return false;
        for (int p : jp) if (n % p == 0) return n == p;
        i64 r = n - 1, x, y;
        int e = 0;
        while (~r & 1) r >>= 1, ++e;
        for (int p : jp) {
                x = fpow(p, r, n);
                for (int t = 0; t < e && x > 1; ++t) {
                        y = (i128)x * x % n;
                        if (y == 1 && x != n - 1) return false;
                        x = y;
                }
                if (x != 1) return false;
        }
        return true;
}
i64 find(i64 n){
        static const int step = 1 << 7;
        i64 x, y = rand() % n;
        int c = rand() % n;
        auto f = [=](i64 x){ return ((i128)x * x + c) % n; } ;
        for (int l = 1; ; l <<= 1) {
                x = y;
                for (int i = 0; i < l; ++i) y = f(y);
                for (int k = 0; k < l; k += step) {
                        int e = std::min(step, l - k);
                        i64 g = 1, z = y;
                        for (int i = 0; i < e; ++i) g = (i128)g * ((y = f(y)) + n - x) % n;
                        g = gcd(g, n);
                        if (g == 1) continue;
                        if (g == n) for (g = 1, y = z; g == 1; ) y = f(y), g = gcd(y + n - x, n);
                        return g;
                }
        } //
}
void rho(i64 n, map<i64,int> &factor){
        while (~n & 1) n >>= 1, ++factor[2];
        if (n == 1) return ;
        if (check_prime(n)) {
                ++factor[n];
                return ;
        }
        i64 d;
        for (d = find(n); d == n; d = find(d));
        rho(d, factor), rho(n / d, factor);
}
int T;
i64 n;
int main(){
        for (cin >> T; T; --T) {
                map<i64, int> f;
                cin >> n;
                rho(n, f);
                if (f.size() > 1 || (--f.end())->second > 1) {
                        cout << (--f.end())->first << '\n';
                } else {
                        cout << "Prime\n";
                }
        }
}
```

4.6 多项式

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod = 998244353;
typedef double ld;
typedef complex<ld> cplx;
typedef vector<ll> poly;

ll qpow(ll a, ll b) {
  ll res = 1;
  while (b) {
    if (b & 1) res = res * a % mod;
    a = a * a % mod;
    b >>= 1;
  }
  return res;
}
ll inv(ll n) { return qpow(n, mod - 2); }

const auto pi = acosl(-1);
```

```
const int len = 15, mask = (1 << len) - 1;

struct unitroot {
  static vector<cplx> w;
  static vector<cplx> get_root(int n) {
    n = 1 << 32 - __builtin_clz(n);
    if (n > w.size()) {
      w.resize(n);
      for (int i = 0; i < n; i++)
        w[i] = cplx(cos(2 * i * pi / n), sin(2 * i * pi / n));
    }
    int m = w.size() / n;
    vector<cplx> res(n);
    for (int i = 0, j = 0; i < n; i++, j += m) res[i] = w[j];
    return res;
  }
};
vector<cplx> unitroot::w;

void fft(vector<cplx> &p, const vector<cplx> &w) {
  int n = w.size();
  for (int i = 1, j = 0; i < n - 1; ++i) {
    int s = n;
    do {
      s >>= 1;
      j ^= s;
    } while (~j & s);
    if (i < j) swap(p[i], p[j]);
  }
  for (int d = 0; (1 << d) < n; ++d) {
    int m = 1 << d, m2 = m * 2, rm = n >> (d + 1);
    for (int i = 0; i < n; i += m2)
      for (int j = 0; j < m; ++j) {
        auto &p1 = p[i + j + m], &p2 = p[i + j];
        auto t = w[rm * j] * p1;
        p1 = p2 - t;
        p2 = p2 + t;
      }
  }
}
poly operator+(const poly &a, const poly &b) {
  poly c(max(a.size(), b.size()));
  for (int i = 0; i < a.size(); i++) c[i] += a[i];
  for (int i = 0; i < b.size(); i++) c[i] += b[i];
  for (auto &v : c) v %= mod;
  return c;
}
poly operator-(poly b) {
  for (auto &v : b) v = v ? mod - v : 0;
  return b;
}
poly operator-(const poly &a, const poly &b) { return a + -b; }
poly operator*(const poly &a, const poly &b) {
  vector<cplx> w = unitroot::get_root(a.size() + b.size() - 1);
  int n = w.size();
  vector<cplx> A(n), B(n), C(n), D(n);
  for (int i = 0; i < a.size(); ++i) A[i] = cplx(a[i] >> len, a[i] & mask);
  for (int i = 0; i < b.size(); ++i) B[i] = cplx(b[i] >> len, b[i] & mask);
  fft(A, w), fft(B, w);
  for (int i = 0; i < n; ++i) {
    int j = (n - i) % n;
    cplx da = (A[i] - conj(A[j])) * cplx(0, -0.5),
         db = (A[i] + conj(A[j])) * cplx(0.5, 0),
         dc = (B[i] - conj(B[j])) * cplx(0, -0.5),
         dd = (B[i] + conj(B[j])) * cplx(0.5, 0);
    C[j] = da * dd + da * dc * cplx(0, 1);
    D[j] = db * dd + db * dc * cplx(0, 1);
  }
  fft(C, w), fft(D, w);
  poly res(a.size() + b.size() - 1);
  for (int i = 0; i < res.size(); ++i) {
    ll da = (ll)(C[i].imag() / n + 0.5) % mod,
       db = (ll)(C[i].real() / n + 0.5) % mod,
       dc = (ll)(D[i].imag() / n + 0.5) % mod,
       dd = (ll)(D[i].real() / n + 0.5) % mod;
    res[i] = ((dd << (len * 2)) + ((db + dc) << len) + da) % mod;
  }
  return res;
}
poly inv(poly a) {
  int n = a.size();
  if (a.size() == 1) return {inv(a[0])};
  poly b = inv({a.begin(), a.end() - n / 2});
  auto c = a * b;
  c.resize(n);
  a = b * (poly{2} - c);
  a.resize(n);
  return a;
}
poly operator/(poly a, poly b) {
```

```
  int n = a.size() + b.size() - 1;
  b.resize(n);
  a = a * inv(b);
  a.resize(n);
  return a;
}
```

4.7 GCD

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

unsigned long long gcd(unsigned long long a, unsigned long long b) {
  int shift = __builtin_ctzll(a | b);
  b >>= __builtin_ctzll(b);
  while (a) {
    a >>= __builtin_ctzll(a);
    if (a < b) swap(a, b);
    a -= b;
  }
  return b << shift;
}
```

5 字符串

5.1 哈希

```
#include "bits/stdc++.h"
using namespace std;
const char el = '\n';
typedef long long ll;
const ll mod1 = 1e9 + 7, mod2 = 1e9 + 9;
struct graph {
  vector<vector<pair<int, int>>> e;
  graph(int n) : e(n + 1) {}
  void adde(int u, int v, int w) { e[u].push_back({v, w}); }
  bool check() {
    vector<int> deg(e.size());
    for (auto &ed : e)
      for (auto &[to, wt] : ed) deg[to]++;
    for (int i = 1; i < e.size(); i++)
      if (deg[i] != e[i].size()) return false;
    return true;
  }
  void euler(int u, vector<int> &res) {
    while (e[u].size()) {
      auto [v, w] = e[u].back();
      e[u].pop_back();
      euler(v, res);
      res.push_back(w);
    }
  }
};
struct hsh {
  static const int k = 3;
  array<ll, k> a;
};
const hsh zero{0, 0, 0}, one{1, 1, 1}, mul{31, 57, 71},
    mod{998244353, 1e9 + 7, 1e9 + 9};

const int mul1 = 31, mul2 = 157;
pair<ll, ll> pmul(int n) {
  static vector<pair<ll, ll>> res = {{1, 1}};
  while (res.size() <= n) {
    auto [p, q] = res.back();
    res.push_back({p * mul1 % mod1, q * mul2 % mod2});
  }
  return res[n];
}
struct hshstr {
  vector<pair<ll, ll>> hsh;
  hshstr(string s = "") {
    hsh.push_back({0, 0});
    for (auto c : s) {
      auto [p, q] = hsh.back();
      hsh.push_back({(p * mul1 + c) % mod1, (q * mul2 + c) % mod2});
    }
  }
  pair<ll, ll> hash(int l, int r) {
    auto [p1, q1] = hsh[l];
    auto [p2, q2] = hsh[r];
    auto [mp, mq] = pmul(r - l);
    return {(p1 * mp - p2 + mod1) % mod1, (q1 * mq - q2 + mod2) % mod2};
  }
};
optional<pair<vector<int>, vector<int>>> solve(vector<hshstr> &a,
                                               vector<hshstr> &b, int n,
```

```
                                                int m) {
  map<pair<ll, ll>, int> id1, id2;
  int k = 0;
  for (auto &s : a) {
    pair<ll, ll> l = s.hash(0, m), r = s.hash(m, n);
    if (!id1.count(l)) id1[l] = ++k;
    if (!id2.count(r)) id2[r] = ++k;
  }
  for (auto &s : b) {
    pair<ll, ll> l = s.hash(0, n - m), r = s.hash(n - m, n);
    if (!id2.count(l)) return nullopt;
    if (!id1.count(r)) return nullopt;
  }
  graph g(k);
  int t = 0;
  for (auto &s : a) {
    pair<ll, ll> l = s.hash(0, m), r = s.hash(m, n);
    g.adde(id1[l], id2[r], ++t);
  }
  t = 0;
  for (auto &s : b) {
    pair<ll, ll> l = s.hash(0, n - m), r = s.hash(n - m, n);
    g.adde(id2[l], id1[r], ++t);
  }
  vector<int> res;
  if (!g.check()) return nullopt;
  g.euler(1, res);
  if (res.size() < 2 * a.size()) return nullopt;
  reverse(res.begin(), res.end());
  vector<int> p, q;
  for (int i = 0; i < res.size(); i += 2) p.push_back(res[i]);
  for (int i = 1; i < res.size(); i += 2) q.push_back(res[i]);
  return pair<vector<int>, vector<int>>{p, q};
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(15);
  int tt;
  cin >> tt;
  while (tt--) {
    int n, m;
    cin >> n >> m;
    vector<hshstr> a(n), b(n);
    for (auto &v : a) {
      string s;
      cin >> s;
      v = hshstr(s);
    }
    for (auto &v : b) {
      string s;
      cin >> s;
      v = hshstr(s);
    }
    bool flg = false;
    for (int i = 0; i < m; i++)
      if (auto res = solve(a, b, m, i)) {
        flg = true;
        for (auto v : res->first) cout << v << ' ';
        cout << el;
        for (auto v : res->second) cout << v << ' ';
        cout << el;
        break;
      }
    if (!flg) cout << -1 << el;
  }
  return 0;
}
```

5.2 KMP

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

vector<int> kmp(string s) {
  vector<int> res = {-1};
  for (auto c : s) {
    int cur = res.back();
    while (~cur && c != s[cur]) cur = res[cur];
    res.push_back(cur + 1);
  }
  return res;
}
```

5.3 后缀自动机

```
#include <bits/stdc++.h>
```

```
using namespace std;
typedef long long ll;

const int alpha = 26;
struct sam {
  struct node {
    array<node *, alpha> next;
    node *link;
    int len;
    int cnt;
    node() {
      fill(next.begin(), next.end(), (node *)NULL);
      link = NULL;
      len = 0;
    }
  };
  node *nd, *end;
  node *nnode() { return new (end++) node(); }
  node *root, *last;
  sam(int n) {
    end = nd = (node *)new char[sizeof(node) * (n << 1)];
    last = root = nnode();
  }
  void add(int c) {
    node *cur = nnode();
    cur->len = last->len + 1;
    cur->cnt = 1;
    node *p = last;
    last = cur;
    while (p && !p->next[c]) {
      p->next[c] = cur;
      p = p->link;
    }
    if (!p) {
      cur->link = root;
      return;
    }
    node *q = p->next[c];
    if (q->len == p->len + 1) {
      cur->link = q;
      return;
    }
    node *clone = nnode();
    clone->next = q->next;
    clone->link = q->link;
    clone->len = p->len + 1;
    while (p && p->next[c] == q) {
      p->next[c] = clone;
      p = p->link;
    }
    cur->link = q->link = clone;
  }
  ll solve() {
    vector<int> deg(end - nd);
    for (auto u = nd + 1; u < end; u++) deg[u->link - nd]++;
    queue<node *> que;
    for (auto u = nd; u < end; u++)
      if (!deg[u - nd]) que.push(u);
    ll res = 0;
    while (que.size()) {
      auto u = que.front();
      que.pop();
      if (auto v = u->link) {
        v->cnt += u->cnt;
        if (!--deg[v - nd]) que.push(v);
      }
      if (u->cnt > 1) res = max(res, 1ll * u->cnt * u->len);
    }
    return res;
  }
};
```

5.4 字典树

```
#include <bits/stdc++.h>
using namespace std;
const char el = '\n';
typedef long long ll;
const int alpha = 26;
struct node {
  array<node *, alpha> ch;
  int cnt;
  node() : cnt(0) { ch.fill(NULL); }
};
void add(node *u, const string &s) {
  for (auto c : s) {
    c -= 'a';
    if (!u->ch[c]) u->ch[c] = new node();
    u = u->ch[c];
```

```
    u->cnt++;
  }
}
pair<node *, ll> run(node *u, const string &s) {
  ll res = 0;
  for (auto c : s) {
    c -= 'a';
    if (!u->ch[c]) return {NULL, res};
    u = u->ch[c];
    res += u->cnt;
  }
  return {u, res};
}
```

6 计算几何

6.1 三维计算几何

```
#include "bits/stdc++.h"
using namespace std;
const char el = '\n';
typedef long long ll;
typedef long double ld;
// typedef __int128 double;
struct cplx {
  ld x, y, z;
  ld abs() { return sqrt(1.0 * x * x + 1.0 * y * y + 1.0 * z * z); }
};
cplx operator+(cplx a, cplx b) { return {a.x + b.x, a.y + b.y, a.z + b.z}; }
cplx operator-(cplx a, cplx b) { return {a.x - b.x, a.y - b.y, a.z - b.z}; }
cplx operator*(cplx a, ld b) { return {a.x * b, a.y * b, a.z * b}; }
cplx operator*(ld b, cplx a) { return {a.x * b, a.y * b, a.z * b}; }
cplx det(cplx a, cplx b) {
  return {a.y * b.z - b.y * a.z, a.z * b.x - b.z * a.x, a.x * b.y - b.x * a.y};
}
ld dot(cplx a, cplx b) { return a.x * b.x + a.y * b.y + a.z * b.z; }
const ld inf = 1e30, eps = 1e-8;
ld dist(array<cplx, 3> a, cplx b) {
  cplx h = det(a[0] - a[1], a[0] - a[2]);
  ld dt = dot(b, h);
  if (dt < eps) return inf;
  ld cp = dot(a[0], h);
  cplx o = cp / dt * b;
  ld sa = det(a[0] - o, a[1] - o).abs() + det(a[1] - o, a[2] - o).abs() +
          det(a[2] - o, a[0] - o).abs(),
     sb = h.abs();
  if (sa / sb > 1 + eps) return inf;
  return o.abs();
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(3);
  int n, m;
  cin >> n >> m;
  vector a(n, array<cplx, 3>());
  for (auto &t : a) {
    for (auto &[x, y, z] : t) cin >> x >> y >> z;
    if (dot(det(t[0] - t[1], t[0] - t[2]), t[0]) < 0) swap(t[1], t[2]);
  }
  while (m--) {
    cplx b;
    cin >> b.x >> b.y >> b.z;
    int p = -1;
    ld val = inf / 2;
    for (int i = 0; i < n; i++) {
      auto tmp = dist(a[i], b);
      if (tmp < val) {
        p = i, val = tmp;
      }
    }
    cout << p + 1 << el;
  }
  return 0;
}
```

6.2 圆凸包

```
#include <bits/stdc++.h>
using namespace std;
const char el = '\n';
typedef long long ll;
typedef double ld;
const ld pi = acosl(-1);
const ld eps = 1e-9;
struct func {
  ll x, y, z;
```

```
  ld val(ld t) { return x * cos(t) + y * sin(t) + z; }
  pair<ld, ld> zero() {
    ld l = sqrt(x * x + y * y);
    if (l < abs(z) - eps) return {0, 2 * pi};
    ld a = atan2(y, x) + 2 * pi, b = acos(-z / l);
    ld p = a - b, q = a + b;
    while (p > 2 * pi) p -= 2 * pi;
    while (q > 2 * pi) q -= 2 * pi;
    if (p > q) swap(p, q);
    return {p, q};
  }
};
bool operator==(func a, func b) {
  return a.x == b.x && a.y == b.y && a.z == b.z;
}
func operator+(func a, func b) { return {a.x + b.x, a.y + b.y, a.z + b.z}; }
func operator-(func a, func b) { return {a.x - b.x, a.y - b.y, a.z - b.z}; }
struct pcef {
  vector<ld> a;
  vector<func> f;
  ld val(ld x) {
    int p = upper_bound(a.begin(), a.end(), x - eps) - a.begin();
    p = max(min(p - 1, (int)f.size() - 1), 0);
    return f[p].val(x);
  }
  ld fnc(ld x) {
    return max(val(x) + val(x + pi), val(x + pi / 2) + val(x + pi * 3 / 2));
  }
  ld mxm(ld l, ld r) {
    ld res = min(fnc(l), fnc(r));
    while (r - l > eps) {
      ld m1 = (l + r) / 2, m2 = (m1 + r) / 2;
      ld v1 = fnc(m1), v2 = fnc(m2);
      res = min({res, v1, v2});
      if (v1 < v2)
        r = m2;
      else
        l = m1;
    }
    return res;
  }
};
pcef operator+(pcef a, pcef b) {
  pcef c = {{0}, {}};
  int p = 0, q = 0;
  ld xl = 0;
  while (p < a.f.size() && q < b.f.size()) {
    ld xr = min(a.a[p + 1], b.a[q + 1]);
    auto [x1, x2] = (a.f[p] - b.f[q]).zero();
    vector<ld> ax;
    if (x1 > xl && x1 < xr) ax.push_back(x1);
    if (x2 > xl && x2 < xr) ax.push_back(x2);
    ax.push_back(xr);
    for (auto xx : ax) {
      ld x = (xl + xx) / 2;
      auto f = a.f[p].val(x) > b.f[q].val(x) ? a.f[p] : b.f[q];
      if (c.f.size() && c.f.back() == f)
        c.a.back() = xx;
      else
        c.a.push_back(xx), c.f.push_back(f);
      xl = xx;
    }
    if (a.a[p + 1] < xr + eps) p++;
    if (b.a[q + 1] < xr + eps) q++;
  }
  return c;
};
pcef comb(const vector<func> &a, int l, int r) {
  if (r - l == 1) return pcef({{0, 2 * pi}, {a[l]}});
  int m = (l + r) / 2;
  return comb(a, l, m) + comb(a, m, r);
}
```

6.3 凸包

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
struct cplx {
  ll x, y;
  ll abs2() { return x * x + y * y; }
  ld abs() { return sqrt(abs2()); }
};
cplx operator+(const cplx &a, const cplx &b) { return {a.x + b.x, a.y + b.y}; }
cplx operator-(const cplx &a, const cplx &b) { return {a.x - b.x, a.y - b.y}; }
bool operator<(const cplx &a, const cplx &b) {
  return a.x != b.x ? a.x < b.x : a.y < b.y;
}
```

```
bool operator!=(const cplx &a, const cplx &b) { return a < b || b < a; }
bool operator==(const cplx &a, const cplx &b) { return !(a != b); }
ll dot(const cplx &a, const cplx &b) { return a.x * b.x + a.y * b.y; }
ll det(const cplx &a, const cplx &b) { return a.x * b.y - a.y * b.x; }
void diagsort(vector<cplx> &a, cplx o) {
  sort(a.begin(), a.end(), [&](const cplx &p, const cplx &q) {
    if ((o < p) != (o < q)) return o < p;
    auto d = det(p - o, q - o);
    if (d) return d > 0;
    return (p - o).abs2() > (q - o).abs2();
  });
  a.resize(unique(a.begin(), a.end(),
                  [&](const cplx &p, const cplx &q) {
                    return !det(p - o, q - o) && dot(p - o, q - o) > 0;
                  }) -
           a.begin());
}
vector<cplx> convex(vector<cplx> a) {
  sort(a.begin(), a.end());
  auto o = a[0];
  a.erase(a.begin());
  diagsort(a, o);
  vector<cplx> res = {o};
  for (auto v : a) {
    while (res.size() >= 2) {
      auto p = res[res.size() - 2], q = res[res.size() - 1];
      if (det(q - p, v - q) > 0) break;
      res.pop_back();
    }
    res.push_back(v);
  }
  return res;
}
double diameter(const vector<cplx> &a) {
  double res = (a.back() - a[0]).abs();
  for (int i = 1; i < a.size(); i++) res += (a[i] - a[i - 1]).abs();
  return res;
}

cplx find(vector<cplx> &conv, cplx a) {
  int l = 0, r = conv.size() - 1;
  while (r - l) {
    int m = (l + r) / 2;
    if (det(conv[m + 1] - conv[m], a) < 0)
      r = m;
    else
      l = m + 1;
  }
  return conv[l];
}
ll solve(vector<ll> &a) {
  vector<ll> s = {0};
  for (auto v : a) s.push_back(s.back() + v);
  vector<cplx> conv;
  for (int i = 0; i < s.size(); i++) {
    cplx r = {i, s[i]};
    while (conv.size() >= 2) {
      cplx p = conv[conv.size() - 2], q = conv.back();
      if (det(q - p, r - q) > 0) break;
      conv.pop_back();
    }
    conv.push_back(r);
  }
  ll res = 0;
  for (int i = 0; i < a.size(); i++) {
    auto p = find(conv, {1, a[i]});
    res = max(res, s[i] - p.y + (p.x - i) * a[i]);
  }
  return res;
}
```

6.4 扫描线

```
#include <bits/stdc++.h>
using namespace std;
const char el = '\n';
typedef long long ll;
typedef __int128 li;
const int inf = 3e6;
struct frac {
  li a, b;
  frac(li _a = 0, li _b = 1) : a(_a), b(_b) {
    if (b < 0) a = -a, b = -b;
  }
};
// note that without gcd plus operation also enlarges the fractor.
frac operator+(frac a, frac b) { return {a.a * b.b + b.a * a.b, a.b * b.b}; }
frac operator-(frac a, frac b) { return {a.a * b.b - b.a * a.b, a.b * b.b}; }
frac operator*(frac a, frac b) { return {a.a * b.a, a.b * b.b}; }
```

```
frac operator/(frac a, frac b) { return {a.a * b.b, a.b * b.a}; }
bool operator<(frac a, frac b) { return a.a * b.b < b.a * a.b; }
bool operator>(frac a, frac b) { return b < a; }
bool operator>=(frac a, frac b) { return !(a < b); }
bool operator<=(frac a, frac b) { return !(b < a); }
bool operator!=(frac a, frac b) { return a < b || b < a; }
bool operator==(frac a, frac b) { return !(a != b); }
frac abs(frac a) { return {a.a < 0 ? -a.a : a.a, a.b}; }
struct cplx {
  frac x, y;
};
cplx operator+(const cplx &a, const cplx &b) { return {a.x + b.x, a.y + b.y}; }
cplx operator-(const cplx &a, const cplx &b) { return {a.x - b.x, a.y - b.y}; }
bool operator<(const cplx &a, const cplx &b) {
  return a.x == b.x ? a.y < b.y : a.x < b.x;
}
bool operator!=(const cplx &a, const cplx &b) { return a < b || b < a; }
bool operator==(const cplx &a, const cplx &b) { return !(a != b); }
struct sgmt {
  cplx a, b;
  sgmt(cplx _a, cplx _b) : a(_a), b(_b) {
    if (b < a) swap(a, b);
  }
  frac cut(frac x) const {
    if (x == a.x) return a.y;
    if (x == b.x) return b.y;
    auto t1 = (b.x - x) * a.y, t2 = (x - a.x) * b.y;
    t1.a += t2.a;
    auto t3 = b.x - a.x;
    auto res = t1 / t3;
    return res;
  }
};
bool operator<(const sgmt &a, const sgmt &b) {
  if (a.a != b.a) return a.a < b.a;
  return a.b < b.b;
}
map<sgmt, vector<cplx>> solve(vector<sgmt> line, vector<cplx> vtx) {
  sort(line.begin(), line.end(), [&](sgmt a, sgmt b) { return a.a < b.a; });
  auto cmp1 = [&](sgmt a, sgmt b) {
    if (a.b != b.b) return a.b < b.b;
    return a.a < b.a;
  };
  set<sgmt, decltype(cmp1)> curl(cmp1);
  frac cx;
  auto cmp2 = [&](sgmt a, sgmt b) {
    frac ay = a.cut(cx), by = b.cut(cx);
    if (ay == by) return a.cut(cx + 1) < b.cut(cx + 1);
    return ay < by;
  };
  set<sgmt, decltype(cmp2)> scan(cmp2);
  auto findc = [&](cplx v) {
    auto [x, y] = v;
    return *scan.lower_bound({{x, y}, {x + 1, y - inf}});
  };
  auto cmp3 = [&](sgmt a, sgmt b) { return a.a.y < b.a.y; };
  set<sgmt, decltype(cmp3)> hori(cmp3);
  auto findh = [&](cplx v) -> optional<sgmt> {
    auto [x, y] = v;
    auto it = hori.upper_bound({{x, y}, {x, y}});
    if (it == hori.begin()) return nullopt;
    it = prev(it);
    if (it->b.y < y) return nullopt;
    return *it;
  };
  reverse(line.begin(), line.end());
  reverse(vtx.begin(), vtx.end());
  map<sgmt, vector<cplx>> res;
  while (vtx.size()) {
    hori.clear();
    frac x = vtx.back().x;
    vector<cplx> curv;
    while (vtx.size() && vtx.back().x <= x)
      curv.push_back(vtx.back()), vtx.pop_back();
    while (curl.size() && curl.begin()->b.x <= x)
      scan.erase(*curl.begin()), curl.erase(curl.begin());
    cx = x;
    while (line.size() && line.back().a.x <= x) {
      auto l = line.back();
      line.pop_back();
      if (l.b.x < x) continue;
      if (l.a.x == l.b.x)
        hori.insert(l);
      else
        scan.insert(l), curl.insert(l);
    }
    for (auto v : curv)
      if (auto t = findh(v))
        res[*t].push_back(v);
      else
```

```
      res[findc(v)].push_back(v);
  }
  return res;
}
const int k = 20;
struct graph {
  vector<vector<int>> e;
  graph(int n) : e(n + 1) {}
  void adde(int u, int v) {
    e[u].push_back(v);
    e[v].push_back(u);
  }
  vector<array<int, k>> f;
  vector<int> d;
  void makef(int u, int w) {
    f[u][0] = w;
    for (int i = 1; i < k; i++) f[u][i] = f[f[u][i - 1]][i - 1];
    d[u] = d[w] + 1;
    for (auto &v : e[u])
      if (v == w) {
        v = e[u].back(), e[u].pop_back();
        break;
      }
    for (auto v : e[u]) makef(v, u);
  }
  int lca(int u, int v) {
    if (d[u] < d[v]) swap(u, v);
    for (int i = 0; i < k; i++)
      if (d[u] - d[v] >> i & 1) u = f[u][i];
    if (u == v) return u;
    for (int i = k - 1; i >= 0; i--)
      if (f[u][i] != f[v][i]) u = f[u][i], v = f[v][i];
    return f[u][0];
  }
  vector<int> s, c;
  void addup(int u) {
    for (auto v : e[u]) s[v] += s[u], addup(v), c[u] += c[v];
  }
  vector<int> solve(vector<pair<int, int>> q) {
    int n = e.size();
    f.resize(n);
    d.assign(n, 0);
    makef(1, 0);
    s.assign(n, 0);
    c.assign(n, 0);
    for (auto [u, v] : q) {
      auto w = lca(u, v);
      s[w]++;
      c[u]++, c[v]++, c[w]--, c[f[w][0]]--;
    }
    addup(1);
    vector<int> res;
    for (auto [u, v] : q) {
      auto w = lca(u, v);
      res.push_back(s[u] + s[v] - 2 * s[w] + c[w] - 1);
    }
    return res;
  }
};
li read() {
  ll n;
  cin >> n;
  return n;
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(3);
  int n, m;
  cin >> n >> m;
  vector<cplx> a(n);
  for (auto &[x, y] : a) x.a = read(), y.a = read();
  a.insert(a.begin(), {0, 0});
  vector<pair<int, int>> edge(n - 1);
  for (auto &[u, v] : edge) cin >> u >> v;
  vector<sgmt> line;
  for (auto [u, v] : edge) line.push_back(sgmt(a[u], a[v]));
  vector<pair<cplx, cplx>> qry(m);
  for (auto &[a, b] : qry)
    a.x.a = read(), a.x.b = read(), a.y.a = read(), a.y.b = read(),
    b.x.a = read(), b.x.b = read(), b.y.a = read(), b.y.b = read();
  map<cplx, int> id;
  for (int i = 1; i <= n; i++) id[a[i]] = i;
  vector<cplx> vtx;
  for (auto [a, b] : qry)
    for (auto v : {a, b})
      if (!id.count(v)) vtx.push_back(v);
  sort(vtx.begin(), vtx.end());
  vtx.resize(unique(vtx.begin(), vtx.end()) - vtx.begin());
  for (int i = 0; i < vtx.size(); i++) id[vtx[i]] = n + 1 + i;
```

```
  auto vol = solve(line, vtx);
  graph g(id.size());
  for (const auto &[p, q] : line) {
    auto arr = vol[{p, q}];
    arr.insert(arr.begin(), p), arr.push_back(q);
    for (int i = 1; i < arr.size(); i++) g.adde(id[arr[i - 1]], id[arr[i]]);
  }
  vector<pair<int, int>> q;
  for (auto &[a, b] : qry) q.push_back({id[a], id[b]});
  vector<int> res = g.solve(q);
  for (auto v : res) cout << v << ' ';
  cout << el;
  return 0;
}
```

7 杂项

7.1 随机数生成

```

namespace myrand {
random_device r;
default_random_engine e(r());
std::uniform_int_distribution<int> g(0, 1e9);
}  // namespace myrand
```

7.2 STL 容器 +Lambda

```

  auto cmp = [&](int u, int v) {
    if (g.e[u].size() != g.e[v].size()) return g.e[u].size() < g.e[v].size();
    return u < v;
  };
  set<int, decltype(cmp)> st(cmp);
```

7.3 子集枚举

```

for (int p = 0; p < n; p++)
  for (int q = p; q; q = q - 1 & p) {
    int k = ~p & (n - 1);
    c[k + q] = max(c[k + q], a[k] + b[q]);
  }
```

7.4 二项式反演

$$f(n)=\sum_{i=0}^{n}\binom{n}{i}g(i)\Longleftrightarrow g(n)=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f(i)$$

$$f(n)=\sum_{i=m}^{n}\binom{n}{i}g(i)\Longleftrightarrow g(n)=\sum_{i=m}^{n}(-1)^{n-i}\binom{n}{i}f(i)$$

$f(n)$: n 选 i 个; $g(n)$: 恰好 n 个

$$f(n)=\sum_{i=n}^{m}\binom{i}{n}g(i)\Longleftrightarrow g(n)=\sum_{i=n}^{m}(-1)^{i-n}\binom{i}{n}f(i)$$

$f(n)$: 钦定 n 个; $g(n)$: 恰好 n 个

7.5 莫比乌斯反演

$$f(n)=\sum_{d|n}g(d)\Longleftrightarrow g(n)=\sum_{d|n}\mu(d)f(n/d)$$

$$f(n)=\sum_{n|d}g(d)\Longleftrightarrow g(n)=\sum_{n|d}\mu(d/n)f(d)$$

7.6 容斥原理

$$\left|\bigcup_{i=1}^{n}S_i\right|=\sum_{m=1}^{n}(-1)^{m-1}\sum_{a_i<a_{i+1}}\left|\bigcap_{i=1}^{m}S_{a_i}\right|$$

$$\left|\bigcap_{i=1}^{n}S_i\right|=|U|-\left|\bigcup_{i=1}^{n}\overline{S_i}\right|$$

$$f(S)=\sum_{S\subseteq T}g(T)\Longleftrightarrow g(S)=\sum_{S\subseteq T}(-1)^{|T|-|S|}f(T)$$

7.7 Min-Max 容斥

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

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