Sungkyunkwan University tomathtomathto

Contents

1 Data Structure

1.1 Dynamic Segment Tree

```
struct node {
  ll s, e, lz, val, li, ri;
  node(ll s, ll e, ll lz = 0)
  :s(s), e(e), lz(lz), val(0), li(0), ri(0) {}
  void updt(node &a, node &b) {
    val = a.val + b.val;
  }
};
struct seg {
  vector<node> v;
  void make(ll n) { v.emplace_back(1, n); }
  void mkch(ll vi) {
    ll s = v[vi].s;
    ll e = v[vi].e;
    if (!v[vi].li) {
      v[vi].li = v.size();
      v.emplace_back(s, (s + e) / 2, v[vi].val / (e - s + 1));
    }
    if (!v[vi].ri) {
      v[vi].ri = v.size();
      v.emplace_back((s + e) / 2 + 1, e, v[vi].val / (e - s + 1));
    }
  }
  void lazy(ll vi) {
    if (v[vi].s != v[vi].e) {
      mkch(vi);
      v[v[vi].li].lz += v[vi].lz;
      v[v[vi].ri].lz += v[vi].lz;
    }
    v[vi].val += (v[vi].e - v[vi].s + 1) * v[vi].lz;
    v[vi].lz = 0;
  }
  void updt(ll l, ll r, ll val, ll vi = 0) {
    ll s = v[vi].s;
    ll e = v[vi].e;
    if (r < s || e < l) {
      lazy(vi);
      return;
    }
    if (l <= s && e <= r) {
      v[vi].lz += val;
      lazy(vi);
      return;
    }
    lazy(vi);
    updt(l, r, val, v[vi].li);
    updt(l, r, val, v[vi].ri);
    v[vi].updt(v[v[vi].li], v[v[vi].ri]);
  }
  ll query(ll l, ll r, ll vi = 0) {
    ll s = v[vi].s;
    ll e = v[vi].e;
    if (r < s || e < l) return 0;
    lazy(vi);
    if (l <= s && e <= r) return v[vi].val;
    ll lval = query(l, r, v[vi].li);
    ll rval = query(l, r, v[vi].ri);
    return lval + rval;
  }
};

ll n;
seg tree;

int main() {
  ll q, m, k, a, b, c, d;
  cin >> n >> m >> k;
  tree.make(n);
  for (int i = 1; i < n; i++) {
    cin >> a;
    tree.updt(i, i, a);
  }
  q = m + k;
  while (q--) {
    cin >> a;
    if (a == 1) {
      cin >> b >> c >> d;
      tree.updt(b, c, d);
    }
    else {
      cin >> b >> c;
      cout << tree.query(b, c) << '\n';
    }
  }
  return 0;
}
```

1.2 2D Segment Tree

```
struct node1 {
  int li, ri, val;
  node1(int li = -1, int ri = -1, int val = 0) :li(li), ri(ri), val(val) {}
};

struct seg1 {
  int n, idx, val, l, r;
  vector<node1> v;
  seg1(int n) :n(n), v(1) {}
  void update(int idx, int val) {
    this->idx = idx;
    this->val = val;
    updt(0, 1, n);
  }
  void updt(int vi, int s, int e) {
    v[vi].val = max(v[vi].val, val);
    if (s == e) return;
    int m = s + e >> 1;
    if (idx <= m) {
      if (v[vi].li == -1) {
        v[vi].li = v.size();
        v.emplace_back();
      }
```

```
      updt(v[vi].li, s, m);
    }
    else {
      if (v[vi].ri == -1) {
        v[vi].ri = v.size();
        v.emplace_back();
      }
      updt(v[vi].ri, m + 1, e);
    }
  }
  int query(int l, int r) {
    this->l = l;
    this->r = r;
    return qry(0, 1, n);
  }
  int qry(int vi, int s, int e) {
    if (vi == -1 || r < s || e < l) return 0;
    if (l <= s && e <= r) return v[vi].val;
    int m = s + e >> 1;
    int lv = qry(v[vi].li, s, m);
    int rv = qry(v[vi].ri, m + 1, e);
    return max(lv, rv);
  }
};

struct node2 {
  int li, ri;
  seg1 seg;
  node2(int n, int li = -1, int ri = -1) :li(li), ri(ri), seg(n) {}
};

struct seg2 {
  int nx, ny, x, y, val, xs, xe, ys, ye;
  vector<node2> v;
  seg2(int nx, int ny) :nx(nx), ny(ny), v(1, ny) {}
  void update(int x, int y, int val) {
    this->x = x;
    this->y = y;
    this->val = val;
    updt(0, 1, nx);
  }
  void updt(int vi, int s, int e) {
    v[vi].seg.update(y, val);
    if (s == e) return;
    int m = s + e >> 1;
    if (x <= m) {
      if (v[vi].li == -1) {
        v[vi].li = v.size();
        v.emplace_back(ny);
      }
      updt(v[vi].li, s, m);
    }
    else {
      if (v[vi].ri == -1) {
        v[vi].ri = v.size();
        v.emplace_back(ny);
      }
      updt(v[vi].ri, m + 1, e);
    }
  }
  int query(int xs, int xe, int ys, int ye) {
    this->xs = xs;
    this->xe = xe;
    this->ys = ys;
    this->ye = ye;
    return qry(0, 1, nx);
  }
  int qry(int vi, int s, int e) {
    if (vi == -1 || xe < s || e < xs) return 0;
    if (xs <= s && e <= xe) return v[vi].seg.query(ys, ye);
    int m = s + e >> 1;
    int lv = qry(v[vi].li, s, m);
    int rv = qry(v[vi].ri, m + 1, e);
    return max(lv, rv);
  }
};

int main() {
  int m, n;
  cin >> m >> n;
  vector<vector<int>> a(n, vector<int>(m));
  for (int j = 0; j < m; ++j) {
    vector<int> v(n);
    for (int i = 0; i < n; ++i) {
      cin >> a[i][j];
      v[i] = a[i][j];
    }
    sort(v.begin(), v.end());
    for (int i = 0; i < n; ++i) {
      a[i][j] = lower_bound(v.begin(), v.end(), a[i][j]) - v.begin() + 1;
    }
  }
  sort(a.begin(), a.end());
  int ans = 0;
  if (m == 2) {
    seg1 seg(n);
    for (int i = 0; i < n; ++i) {
      int tmp = seg.query(1, a[i][1]) + 1;
      ans = max(ans, tmp);
      seg.update(a[i][1], tmp);
    }
  }
  else {
    seg2 seg(n, n);
    for (int i = 0; i < n; ++i) {
      int tmp = seg.query(1, a[i][1], 1, a[i][2]) + 1;
      ans = max(ans, tmp);
      seg.update(a[i][1], a[i][2], tmp);
    }
  }
  cout << ans;
}
```

1.3 K-th Fenwick

```
// tree[] = fenwick
int find_kth(int k) {
  int sum = 0, pos = 0;
  for (int i = __lg(node_size); i >= 0; i--) {
    int pos_nxt = pos | (1 << i);
    if (pos_nxt < node_size && sum + tree[pos_nxt] < k) {
      pos = pos_nxt;
      sum += tree[pos];
    }
  }
  return pos + 1;
}
```

1.4 Persistent Segment Tree

```
template<typename T> struct PST
{
  #define L 0
  #define R 1'000'000
  //pst의 범위
  struct vertex { int l, r; T val; };
  vector<vertex> pst;
  int make_tree(int nx) // head == nx인 pst를 복사하여 new head를 반환
  {
    if (nx == -1) pst.push_back({ -1, -1, T() });//
    else pst.push_back(pst[nx]);
    return (int)(pst.size()) - 1;
  }
  void update(int nx, T *arr, int ns = L, int ne = R) // head==nx인 트리의 ns부터 ne까지 한번에 갱신 O(ne-ns)
  {
    if (ns == ne)
    {
      pst[nx].val = arr[ns];
      return;
    }
    int mid = (ns + ne) >> 1; // 나누기2가 아닌 쉬프트1로 하는걸
    update(pst[nx].l = make_tree(pst[nx].l), arr, ns, mid);
    update(pst[nx].r = make_tree(pst[nx].r), arr, mid + 1, ne);
    pst[nx].val = pst[pst[nx].l].val + pst[pst[nx].r].val;
  }
  void update(int nx, int idx, T df, int ns = L, int ne = R) // head == nx인 트리의 idx를 df로 갱신
  {
    if (ns == ne)
    {
      pst[nx].val = df;//
      return;
    }
    int mid = (ns + ne) >> 1;
    if (idx <= mid) update(pst[nx].l = make_tree(pst[nx].l), idx, df, ns, mid);
    else update(pst[nx].r = make_tree(pst[nx].r), idx, df, mid + 1, ne);
    pst[nx].val = T();//
    if (pst[nx].l != -1) pst[nx].val = pst[nx].val + pst[pst[nx].l].val;//
    if (pst[nx].r != -1) pst[nx].val = pst[nx].val + pst[pst[nx].r].val;//
  }
  T get_sum(int nx, int qs, int qe, int ns = L, int ne = R) // 응
  {
    if (nx == -1) return T();//
    if (ne < qs || qe < ns) return T(); //
    if (qs <= ns && ne <= qe) return pst[nx].val;
    int mid = (ns + ne) >> 1;
    return get_sum(pst[nx].l, qs, qe, ns, mid) + get_sum(pst[nx].r, qs, qe, mid + 1,
    ne);//
  }
};
```

1.5 Persistent Segment Tree

```
struct PST {
  // K = the number of trees
  // size = the number of nodes
  int st, en, K, size;
  int A[20 * MAXN];
  int lchd[20 * MAXN];
  int rchd[20 * MAXN];
  int root[MAXN + 5];

  void init(int st, int en) {
    this->st = st, this->en = en;
    K = 0, size = 1;
    init(0, st, en);
  }

  void init(int nidx, int st, int en) {
    A[nidx] = 0;
    if (st == en) {
      return;
    }
    int mid = (st + en) >> 1;

    init(lchd[nidx] = size++, st, mid);
    init(rchd[nidx] = size++, mid + 1, en);
  }

  // p번째 트리 복사하여 새로운 트리 생성
  void add(int idx, int val, int p) {
    ++K;
    A[root[K] = size] = A[root[p]];
    lchd[size] = lchd[root[p]];
    rchd[size] = rchd[root[p]];
    update(size++, st, en, idx, val);
  }

  void update(int nidx, int st, int en, int idx, int val) {
    if (st == en) {
      A[nidx] += val;

      // A[nidx].val = val;

      return;
    }

    int mid = (st + en) >> 1;
```

```
      if (idx <= mid) {
        int lidx = lchd[nidx];
        lchd[nidx] = size;

        A[size] = A[lidx];
        lchd[size] = lchd[lidx];
        rchd[size] = rchd[lidx];

        update(size++, st, mid, idx, val);
      }
      else {
        int ridx = rchd[nidx];
        rchd[nidx] = size;

        A[size] = A[ridx];
        lchd[size] = lchd[ridx];
        rchd[size] = rchd[ridx];

        update(size++, mid + 1, en, idx, val);
      }

      A[nidx] = A[lchd[nidx]] + A[rchd[nidx]];
    }

    // k-th tree query
    int solve(int k, int le, int ri) {
      return solve(root[k], st, en, le, ri);
    }

    int solve(int nidx, int st, int en, int le, int ri) {
      if (st > ri || en < le || le > ri) {
        return 0;
      }
      if (le <= st && en <= ri) {
        return A[nidx];
      }
      int mid = (st + en) >> 1;
      return solve(lchd[nidx], st, mid, le, ri) + solve(rchd[nidx], mid + 1, en, le, ri);
    }

    // remove k trees most recently added
    void remove(int k) {
      size = root[K - k + 1];
      K -= k;
    }
  } pst;
```

2 Dynamic Programming

2.1 Convex Hull Optimization

```
struct line {
  ll slope, constant;
  line() : slope(0), constant(0) {}
  line(ll a, ll b) : slope(a), constant(b) {}
};
// x should be monotonic increasing
struct CHT {
  int sz;
  deque<line> L;
  CHT() : sz(0) {}
  double get_point(line L1, line L2) {
    return (double)(L2.constant - L1.constant) / (L1.slope - L2.slope);
  }
  bool increase(line L1, line L2, line L3) {
    double p1 = get_point(L1, L2);
    double p2 = get_point(L2, L3);
    return p1 <= p2;
  }
  void update(line L3) {
    while (sz > 1) {
      line L2 = L.back();
      L.pop_back();
      sz--;
      line L1 = L.back();
      if (increase(L1, L2, L3)) {
        L.push_back(L2);
        sz++;
        break;
      }
    }
    L.push_back(L3);
    sz++;
  }
  ll query(ll x) {
    while (sz > 1 && get_point(L[0], L[1]) < (double)x) {
      L.pop_front();
      sz--;
    }
    line L1 = L.front();
    return L1.slope * x + L1.constant;
  }
};
```

2.2 Alien Trick

```
// 즉 이분탐색을 돌렸을 때 최종적으로 기울기 a를 얻지 못 할 수 있지만, 기울기 a를 반드시 한 번
거치게 된다. 그 때의 접선 mid*x + mx에 대하여 x=K 일 때의 값 mid*K + mx가 답이 되고, 이것은
다른 어떤 접선에서의 x=K 일 때의 값보다 작거나 같으므로 answer = inf 로 놓고 answer =
min(answer, mid*K + mx) 를 반복하면 결국 답을 구할 수 있게 되는 것이다.
// 반대로 볼록함수 + 최솟값 구하기이면 연산할 때마다 mid를 더한다고 생각하고 x=K 일 때 접선의
값이 항상 크거나 같으므로 answer = -inf로 놓고 answer = max(answer, -mid*K + mn) 를 반복하면
된다.
signed main()
{
  int N; int K; cin >> N >> K;
  vector<int> A(N);
  for (auto &it : A) cin >> it;
  vector<int> suf(N + 1);
  for (int i = N - 1; i >= 0; --i)
  suf[i] = suf[i + 1] + A[i];
  int lf = 0, rg = 1e16;
  int ans = inf;
  while (lf <= rg)
  {
    int mid = (lf + rg) / 2;
```

```
    vector<int> dp(N + 1);
    vector<int> cnt(N + 1);
    vector<pair<int, int>> memo(N + 1);
    pair<int, int> mx = make_pair(-mid, 0);
    for (int i = N - 1; i >= 0; --i)
    {
      dp[i] = suf[i] + mx.first;
      cnt[i] = mx.second + 1;
      memo[i] = max(memo[i + 1], make_pair(dp[i], cnt[i]));
      mx = max(mx, make_pair(memo[i].first - mid - suf[i], memo[i].second));
    }
    ans = min(ans, memo[0].first + mid * K);
    if (cnt[0] <= K) rg = mid - 1;
    else lf = mid + 1;
  }
  cout << ans << '\n';
  return 0;
}
```

2.3 LiChao Tree

```
typedef pair<ll, ll> Line;

struct LiChaoTree {
  ll f(Line l, ll x) {
    return l.first * x + l.second;
  }
  struct Node {
    int lnode, rnode;
    ll xl, xr;
    Line l;
  };
  vector<Node> nodes;
  void init(ll xmin, ll xmax) {
    nodes.push_back({ -1,-1,xmin,xmax,{0,-1e18} });
  }
  void insert(int n, Line newline) {
    ll xl = nodes[n].xl, xr = nodes[n].xr;
    ll xm = (xl + xr) >> 1;
    Line llow = nodes[n].l, lhigh = newline;
    if (f(llow, xl) >= f(lhigh, xl)) swap(llow, lhigh);
    if (f(llow, xr) <= f(lhigh, xr)) {
      nodes[n].l = lhigh;
      return;
    }
    else if (f(llow, xm) <= f(lhigh, xm)) {
      nodes[n].l = lhigh;
      if (nodes[n].rnode == -1) {
        nodes[n].rnode = nodes.size();
        nodes.push_back({ -1,-1,xm + 1,xr,{0,-1e18} });
      }
      insert(nodes[n].rnode, llow);
    }
    else {
      nodes[n].l = llow;
      if (nodes[n].lnode == -1) {
        nodes[n].lnode = nodes.size();
        nodes.push_back({ -1,-1,xl,xm,{0,-1e18} });
      }
      insert(nodes[n].lnode, lhigh);
    }
  }
  ll get(int n, ll xq) {
    if (n == -1) return -1e18;
    ll xl = nodes[n].xl, xr = nodes[n].xr;
    ll xm = (xl + xr) >> 1;
    if (xq <= xm) return max(f(nodes[n].l, xq), get(nodes[n].lnode, xq));
    else return max(f(nodes[n].l, xq), get(nodes[n].rnode, xq));
  }
};
```

2.4 Knuth Optimization

```
int tc, n;
int m[5005];
int dp[5005][5005], pos[5005][5005];
int sum[5005];
const int INF = 1e9;

int main()
{
  for (scanf("%d", &tc); tc > 0; tc--) {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
      scanf("%d", &m[i]);
      pos[i][i] = i;
      sum[i] = sum[i - 1] + m[i];
    }
    for (int len = 2; len <= n; len++) {
      for (int i = 1; i <= n - len + 1; i++) {
        dp[i][i + len - 1] = INF;
        int s = pos[i][i + len - 2], f = pos[i + 1][i + len - 1];
        for (int j = s; j <= f; j++) {
          if (j < n && dp[i][i + len - 1] > dp[i][j] + dp[j + 1][i + len - 1]) {
            pos[i][i + len - 1] = j;
            dp[i][i + len - 1] = dp[i][j] + dp[j + 1][i + len - 1];
          }
        }
        dp[i][i + len - 1] += sum[i + len - 1] - sum[i - 1];
      }
    }
    printf("%d\n", dp[1][n]);
  }
  return 0;
}
```

2.5 Profile DP

```
#define all(x) (x).begin(), (x).end()
#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
constexpr int inf = 1e9 + 7;
int n, m, grid[9][9];
map<int, map<int, map<vector<int>, int>>> dp;
void norm(vector<int> &a)
{
  vector<int> comp;
  for (auto &it : a) if (it != -1) comp.push_back(it);
```

```
    sort(all(comp));
    comp.resize(unique(all(comp)) - comp.begin());
    for (auto &it : a) if (it != -1) it = (int)(lower_bound(all(comp), it) -
    comp.begin());
  }
  int f(int r, int c, vector<int> p)
  {
    if (c == m) return f(r + 1, 0, p);
    if (dp[r][c].count(p)) return dp[r][c][p];
    bool stopable = true;
    int before = -1;
    for (int i = 0; i < m; ++i)
    {
      if (p[i] != -1)
      {
        if (before == -1) before = p[i];
        else if (before != p[i])
        {
          stopable = false;
          break;
        }
      }
    }
    int &ret = dp[r][c][p] = inf;
    if (stopable) ret = 0;

    if (r == n) return ret;

    int canz = (p[0] == -1);
    for (int i = 1; i < m && !canz; ++i) if (p[0] == p[i]) canz = true;
    if (canz)
    {
      vector<int> np(m);
      for (int i = 0; i + 1 < m; ++i) np[i] = p[i + 1];
      np.back() = -1;
      norm(np);
      ret = min(ret, f(r, c + 1, np));
    }
    {
      int first = -1, second = -1;
      if (c - 1 >= 0 && p.back() != -1) first = p.back();
      if (r - 1 >= 0 && p.front() != -1) second = p.front();
      vector<int> np(m);
      for (int i = 0; i + 1 < m; ++i)
      np[i] = first != -1 && first == p[i + 1] || second != -1 && second == p[i + 1] ? inf
      : p[i + 1];
      np.back() = inf;
      norm(np);
      ret = min(ret, f(r, c + 1, np) + grid[r][c]);
    }
    return ret;
  }
  signed main(void)
  {
    cin >> n >> m;
    forn(i, n) forn(j, m) cin >> grid[i][j];
    vector<int> tmp(m, -1);
    cout << f(0, 0, tmp) << '\n';
    return 0;
  }
```

3 Graph

3.1 2-SAT

```
struct SAT {
  int n, sccn, dfsn;
  vector<int> scc, dfn, low;
  vector<vector<int>> adj;
  vector<bool> tf;
  stack<int> s;
  SAT(int n) :n(n), adj(2 * n + 5), scc(2 * n + 5), dfn(2 * n + 5), low(2 * n + 5), tf(n
  + 5) {}
  void add(int a, int b) {
    adj[a ^ 1].push_back(b);
    adj[b ^ 1].push_back(a);
  }
  void dfs(int idx) {
    low[idx] = dfn[idx] = ++dfsn;
    s.push(idx);
    for (int nxt : adj[idx]) {
      if (!dfn[nxt]) dfs(nxt);
      if (!scc[nxt]) low[idx] = min(low[idx], low[nxt]);
    }
    if (low[idx] == dfn[idx]) {
      ++sccn;
      while (1) {
        int tmp = s.top();
        s.pop();
        scc[tmp] = sccn;
        if (tmp == idx) break;
      }
    }
  }
  bool solve() {
    for (int i = 0; i < 2 * n; ++i) {
      if (!dfn[i]) dfs(i);
    }
    for (int i = 0; i < n; ++i) {
      if (scc[i << 1] == scc[i << 1 | 1]) return 0;
    }
    for (int i = 0; i < n; ++i) {
      tf[i] = (scc[i << 1 | 1] > scc[i << 1]);
    }
    return 1;
  }
};

int main() {
  int n, m;
  cin >> n >> m;
  SAT sat(n);
  for (int i = 0; i < m; ++i) {
    int a, b;
```

```
    cin >> a >> b;
    if (a > 0) a = 2 * a - 2;
    else a = 2 * abs(a) - 1;
    if (b > 0) b = 2 * b - 2;
    else b = 2 * abs(b) - 1;
    sat.add(a, b);
  }
  if (sat.solve()) {
    cout << "1\n";
    for (int i = 0; i < n; ++i) {
      cout << sat.tf[i] << ' ';
    }
  }
  else cout << 0;
}
```

3.2 BCC

```
vector<int> dfsn(n, -1);
vector<vector<pair<int, int>>> bcc;
function<int(int, int)> dfs = [&](int now, int before)->int
{
  static int dcnt = 0;
  static stack<pair<int, int>> stk;
  int result = dfsn[now] = dcnt++;
  for (int nxt : g[now])
  {
    if (nxt == before) continue;
    if (dfsn[now] > dfsn[nxt]) stk.push({ now, nxt });
    if (dfsn[nxt] != -1) result = min(result, dfsn[nxt]);
    else
    {
      int tmp = dfs(nxt, now);
      result = min(result, tmp);
      if (tmp >= dfsn[now])
      {
        vector<pair<int, int>> nbcc;
        while (!stk.empty() && stk.top() != make_pair(now, nxt))
        {
          nbcc.push_back(stk.top()); stk.pop();
        }
        nbcc.push_back(stk.top());
        stk.pop();
        bcc.push_back(nbcc);
      }
    }
  }
  return result;
};
for (int i = 0; i < n; ++i)
  if (dfsn[i] == -1) dfs(i, i);
```

3.3 Dominator Tree

```
struct dominator_tree
{
  //original graph
  int nn;
  vector<vector<int>> g;

  //dominator tree
  vector<vector<int>> tree;
  vector<int> L, R;

  //auxiliary data
  vector<vector<int>> rg;
  vector<vector<int>> bucket;
  vector<int> idom, sdom, pre, prv;
  vector<int> ancestor, label, preorder;

  dominator_tree(int nn) :nn(nn) { g.resize(nn); }
  void add_edge(int u, int v) { g[u].push_back(v); }
  void build(int s)
  {
    tree.resize(nn);
    L.resize(nn), R.resize(nn);

    rg.resize(nn);
    bucket.resize(nn);
    idom.resize(nn);
    sdom.resize(nn, -1);
    pre.resize(nn, -1);
    prv.resize(nn);

    ancestor.resize(nn, -1);
    label.resize(nn);

    dfs(s);
    if (preorder.size() == 1) return;
    for (int i = preorder.size() - 1; i >= 1; --i)
    {
      int w = preorder[i];
      for (int v : rg[w])
      {
        int u = eval(v);
        if (pre[sdom[u]] < pre[sdom[w]]) sdom[w] = sdom[u];
      }
      bucket[sdom[w]].push_back(w);
      link(prv[w], w);
      for (int v : bucket[prv[w]])
      {
        int u = eval(v);
        idom[v] = (u == v) ? sdom[v] : u;
      }
      bucket[prv[w]].clear();
    }
    for (int i = 1; i < preorder.size(); ++i)
    {
      int w = preorder[i];
      if (idom[w] != sdom[w]) idom[w] = idom[idom[w]];
      tree[idom[w]].push_back(w);
    }
    idom[s] = sdom[s] = -1;
    dfs2(s);
  }
```

```cpp
    void dfs(int v)
    {
      static int t = 0;
      pre[v] = ++t;
      sdom[v] = label[v] = v;
      preorder.push_back(v);
      for (int nxt : g[v])
      {
        if (sdom[nxt] == -1)
        {
          prv[nxt] = v;
          dfs(nxt);
        }
        rg[nxt].push_back(v);
      }
    }

    int eval(int v)
    {
      if (ancestor[v] == -1) return v;
      if (ancestor[ancestor[v]] == -1) return label[v];
      int u = eval(ancestor[v]);
      if (pre[sdom[u]] < pre[sdom[label[v]]]) label[v] = u;
      ancestor[v] = ancestor[u];
      return label[v];
    }
    inline void link(int u, int v) { ancestor[v] = u; }
    void dfs2(int v)
    {
      static int t = 0;
      L[v] = t++;
      for (int nxt : tree[v])
      dfs2(nxt);
      R[v] = t++;
    }
    inline bool dominates(int u, int v)
    {
      if (pre[v] == -1) return 1;
      return L[u] <= L[v] && R[v] <= R[u];
    }
  };
```

3.4 Bipartite Matching

```cpp
struct bimat {
  ll n, m;
  vector<ll> mata, matb, lv, work;
  vector<vector<ll>> adj;
  bimat(ll n, ll m) :n(n), m(m), mata(vector<ll>(n + 5)), matb(vector<ll>(m + 5)),
  adj(vector<vector<ll>>(n + 5)) {}
  void bfs() {
    lv = vector<ll>(n + 5);
    queue<ll> q;
    for (int i = 1; i < n; i++) {
      if (!mata[i]) {
        lv[i] = 1;
        q.push(i);
      }
    }
    while (!q.empty()) {
      ll top = q.front();
      q.pop();
      for (ll nxt : adj[top]) {
        nxt = matb[nxt];
        if (nxt && !lv[nxt]) {
          lv[nxt] = lv[top] + 1;
          q.push(nxt);
        }
      }
    }
  }
  bool dfs(ll idx) {
    ll sz = adj[idx].size();
    for (ll &i = work[idx]; i < sz; ++i) {
      ll v = adj[idx][i];
      ll nxt = matb[v];
      if (!nxt || (lv[nxt] == lv[idx] + 1 && dfs(nxt))) {
        matb[v] = idx;
        mata[idx] = v;
        return true;
      }
    }
    return false;
  }
  ll operator()() {
    ll ret = 0;
    while (1) {
      bfs();
      work = vector<ll>(n + 5);
      ll cnt = 0;
      for (int i = 1; i < n; i++) {
        if (!mata[i] && dfs(i)) ++cnt;
      }
      if (cnt) ret += cnt;
      else break;
    }
    return ret;
  }
  // void altpath(int x)
  // {
    //     if (altX[x])
    //         return;
    //     altX[x] = 1;
    //     for (auto &y : g[x])
    //     {
      //         if (xy[x] != y && yx[y] != -1)
      //         {
        //             altY[y] = 1;
        //             altpath(yx[y]);
        //         }
      //     }
    // }
  // void altpathing()
  // {
```

```
    //     forn(x, N) if (xy[x] == -1)
    //         altpath(x);
    // }
  // bool is_coverX(int x) { return !altX[x]; }
  // bool is_coverY(int y) { return altY[y]; }
};
```

3.5 Maxflow

```
struct Dinic {
  struct Edge {
    ll par, nxt, cap, flow;
    Edge(ll par, ll nxt, ll cap) :par(par), nxt(nxt), cap(cap), flow(0) {}
  };
  ll n, src, sink;
  vector<ll> st, cur, lv;
  vector<Edge> edge;
  Dinic(ll n) :n(n), src(n + 1), sink(n + 2), st(n + 5, -1) {}
  void addEdge(ll a, ll b, ll cap) {
    edge.emplace_back(st[a], b, cap);
    st[a] = edge.size() - 1;
    edge.emplace_back(st[b], a, 0);
    st[b] = edge.size() - 1;
  }
  bool bfs() {
    lv = vector<ll>(n + 5);
    lv[src] = 1;
    queue<ll> q;
    q.push(src);
    while (!q.empty()) {
      ll idx = q.front();
      q.pop();
      for (ll i = st[idx]; i != -1; i = edge[i].par) {
        Edge &e = edge[i];
        if (!lv[e.nxt] && e.flow < e.cap) {
          lv[e.nxt] = lv[idx] + 1;
          q.push(e.nxt);
        }
      }
    }
    return lv[sink];
  }
  ll dfs(ll idx, ll flow) {
    if (idx == sink) return flow;
    for (ll &i = cur[idx]; i != -1; i = edge[i].par) {
      Edge &e = edge[i];
      if (e.flow < e.cap && lv[e.nxt] == lv[idx] + 1) {
        ll tmp = dfs(e.nxt, min(flow, e.cap - e.flow));
        if (tmp) {
          e.flow += tmp;
          edge[i ^ 1].flow -= tmp;
          return tmp;
        }
      }
    }
    return 0;
  }
  ll solve() {
    ll ret = 0;
    while (bfs()) {
      cur = st;
      ll tmp;
      while (tmp = dfs(src, INF)) ret += tmp;
    }
    return ret;
  }
};
```

3.6 Mincost Maxflow

```
struct MCMF {
  struct Edge {
    int nxt, cap, cost, ridx;
  };

  int src, snk;
  vector<int> dist, work;
  vector<bool> vis;
  vector<vector<Edge>> adj;

  MCMF(int n, int src, int snk) : src(src), snk(snk), dist(n + 5), work(n + 5), vis(n +
  5), adj(n + 5) {}

  void addedge(int st, int en, int cap, int cost) {
    adj[st].push_back({ en, cap, cost, sz(adj[en]) });
    adj[en].push_back({ st, 0, -cost, sz(adj[st]) - 1 });
  }

  bool spfa() {
    fill(all(dist), inf);
    fill(all(vis), false);
    queue<int> que;
    dist[src] = 0; que.push(src);
    while (sz(que)) {
      int cur = que.front(); que.pop();
      vis[cur] = false;
      for (auto &[nxt, cap, cost, _] : adj[cur]) {
        if (cap > 0 && dist[nxt] > dist[cur] + cost) {
          dist[nxt] = dist[cur] + cost;
          if (vis[nxt] == false) {
            que.push(nxt);
            vis[nxt] = true;
          }
        }
      }
    }
    return dist[snk] != inf;
  }

  int dfs(int cur, int flow) {
    if (cur == snk) {
      return flow;
    }
    vis[cur] = true;
    for (int &i = work[cur]; i < sz(adj[cur]); i++) {
      auto &[nxt, cap, cost, ridx] = adj[cur][i];
```

```
      if (vis[nxt] || cap == 0 || dist[nxt] != dist[cur] + cost) {
        continue;
      }
      int &rcap = adj[nxt][ridx].cap;
      int ret = dfs(nxt, min(flow, cap));
      if (ret) {
        cap -= ret;
        rcap += ret;
        return ret;
      }
    }
    return 0;
  }

  pi solve() {
    pi ret = pi(0, 0);
    while (spfa()) {
      fill(all(work), 0);
      for (int flow = dfs(src, inf); flow; flow = dfs(src, inf)) {
        ret.first += flow;
        ret.second += flow * dist[snk];
      }
    }
    return ret;
  }
};
```

3.7 General Matching

```
constexpr int MOD = 998244353;
int power(int a, int n)
{
  int ret = 1;
  for (; n; a = 1LL * a * a % MOD, n >>= 1)
  if (n % 2) ret = 1LL * ret * a % MOD;
  return ret;
}
int N, M;
signed main()
{
  mt19937 rand(time(0));
  cin >> N >> M;
  vector<vector<int>> A(N, vector<int>(N));
  for (int i = 0; i < M; ++i)
  {
    int u, v; cin >> u >> v; --u, --v;
    int r = (int)rand() % (MOD - 1) + 1;
    A[u][v] = A[v][u] = r;
  }
  for (int i = 0; i < N; ++i)
  {
    for (int j = i; j < N; ++j)
    {
      if (A[j][i])
      {
        swap(A[i], A[j]);
        break;
      }
    }
    if (!A[i][i]) continue;
    int k = power(A[i][i], MOD - 2);
    for (int j = i; j < N; ++j) A[i][j] = (1LL * A[i][j] * k) % MOD;
    for (int j = i + 1; j < N; ++j)
    {
      if (A[j][i])
      {
        int x = A[j][i];
        for (int k = i; k < N; ++k) A[j][k] = (A[j][k] - 1LL * A[i][k] * x % MOD + MOD)
        % MOD;
      }
    }
  }
  int rk = 0;
  for (int i = 0; i < N; ++i) if (A[i][i]) ++rk;
  cout << rk / 2 << '\n';
  return 0;
}
```

3.8 Fast General Matching

```
#define entire(X) X.begin(),X.end()

struct Blossom {
  int n, t;
  vector<vector<int>> adj;
  vector<int> orig, par, vis, match, aux;
  queue<int> Q;
  Blossom(int n) : n{ n }, t{ 0 }, adj(n + 1), orig(n + 1), par(n + 1),
  vis(n + 1), match(n + 1), aux(n + 1) {}
  void connect(int a, int b) {
    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  void augment(int u, int v) {
    int pv = v, nv;
    do {
      pv = par[v], nv = match[pv];
      match[v] = pv, match[pv] = v;
      v = nv;
    } while (u != pv);
  }
  int lca(int v, int w) {
    ++t;
    while (1) {
      if (v) {
        if (aux[v] == t) return v;
        aux[v] = t, v = orig[par[match[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while (orig[v] != a) {
      par[v] = w, w = match[v];
      if (vis[w] == 1) Q.push(w), vis[w] = 0;
```

```
      orig[v] = orig[w] = a;
      v = par[w];
    }
  }
  bool bfs(int u) {
    fill(entire(vis), -1), iota(entire(orig), 0);
    Q = queue<int>(); Q.push(u), vis[u] = 0;
    while (!Q.empty()) {
      int v = Q.front(); Q.pop();
      for (int x : adj[v]) {
        if (vis[x] == -1) {
          par[x] = v, vis[x] = 1;
          if (!match[x]) return augment(u, x), true;
          Q.push(match[x]); vis[match[x]] = 0;
        }
        else if (vis[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a), blossom(v, x, a);
        }
      }
    }
    return false;
  }

  int solve() {
    int ans = 0;
    for (int x = 1; x <= n; x++) if (!match[x]) {
      for (int y : adj[x]) if (!match[y]) {
        match[x] = y, match[y] = x;
        ++ans; break;
      }
    }
    for (int i = 1; i <= n; i++) if (!match[i] && bfs(i)) ++ans;
    return ans;
  }
};

int main() {
  int n, m; cin >> n >> m;
  Blossom B(n);
  for (int i = 0; i < m; i++) {
    int a, b; cin >> a >> b;
    B.connect(a, b);
  }
  cout << B.solve();
}
```

3.9 Heavy-Light Decomposition

```
vector<int> adj[MAXN + 5];
int par[MAXN + 5], sub[MAXN + 5];
int in[MAXN + 5], inv[MAXN + 5], top[MAXN + 5], dep[MAXN + 5];
int num = 0;

// i를 루트로 하는 서브트리
// in[i] ~ in[i] + sub[i] - 1

// i를 포함하는 체인의 꼭대기 정점부터 i까지
// in[top[i]] ~ in[i]

void getsub(int cur) {
  sub[cur] = 1;

  if (sz(adj[cur]) >= 2 && adj[cur][0] == par[cur]) {
    swap(adj[cur][0], adj[cur][1]);
  }

  for (int i = 0; i < sz(adj[cur]); i++) {
    if (adj[cur][i] == par[cur]) {
      continue;
    }
    par[adj[cur][i]] = cur;
    getsub(adj[cur][i]);
    sub[cur] += sub[adj[cur][i]];
    if (sub[adj[cur][i]] > sub[adj[cur][0]]) {
      swap(adj[cur][i], adj[cur][0]);
    }
  }
}

void hld(int cur) {
  in[cur] = ++num;
  inv[num] = cur;

  for (auto nxt : adj[cur]) {
    if (nxt == par[cur]) {
      continue;
    }
    top[nxt] = nxt == adj[cur][0] ? top[cur] : nxt;
    dep[nxt] = dep[cur] + (nxt != adj[cur][0]);
    hld(nxt);
  }
}

int getlca(int a, int b) {
  while (top[a] != top[b]) {
    if (dep[a] > dep[b]) {
      swap(a, b);
    }
    b = par[top[b]];
  }
  return in[a] < in[b] ? a : b;
}
```

3.10 Centroid Decomposition

```
// 센트로이드 트리에서 z=lca(x,y) 라 할 때, 원본 트리에서 x에서 y까지의 경로는 z를 지나간다
// 노드 v에서 다른 노드까지의 경로는 센트로이드 트리에서 v의 조상(v 포함)들을 기준으로 분리되므로
모든 경로를 O(lgN)개로 표현할 수 있다
struct CentroidDecomposition {
  vector<vector<int>> tree;
  vector<vector<int>> CentTree; // result of Centroid Tree
  vector<bool> vis;
  vector<int> sub; // size of subtree
  int tot;
  CentroidDecomposition(int N) {
```

```
      tree.resize(N + 5);
      CentTree.resize(N + 5);
      vis.resize(N + 5);
      sub.resize(N + 5);
    }
    void get_sz(int cur, int prev) {
      sub[cur] = 1;
      for (int i = 0; i < tree[cur].size(); i++) {
        int nxt = tree[cur][i];
        if (!vis[nxt] && nxt != prev) {
          get_sz(nxt, cur);
          sub[cur] += sub[nxt];
        }
      }
    }
    int get_cen(int cur, int prev) {
      for (int i = 0; i < tree[cur].size(); i++) {
        int nxt = tree[cur][i];
        if (!vis[nxt] && nxt != prev && sub[nxt] > tot / 2)
        return get_cen(nxt, cur);
      }
      return cur;
    }
    int decomp(int cur, int prev) {
      get_sz(cur, prev);
      tot = sub[cur];
      int cen = get_cen(cur, prev);
      vis[cen] = true;
      for (int i = 0; i < tree[cen].size(); i++) {
        int nxt = tree[cen][i];
        int nxtcen;
        if (!vis[nxt] && nxt != prev) {
          nxtcen = decomp(nxt, cen);
          // par[nxtcen] = cen;
          CentTree[cen].push_back(nxtcen);
          // CentTree[nxtcen].push_back(cen);
        }
      }
      return cen;
    }
  };
```

4 Strings

4.1 Aho-Corasick Algorithm

```
// MAXN = 패턴 개수 * 패턴 크기
// MAXC = 문자 개수

struct Trie {
  int chd[MAXN + 5][MAXC];
  int fail[MAXN + 5]; // 실패링크 = (나 자신이 아닌) 가장 긴 접미사
  int output[MAXN + 5]; // 출력링크 = 접미사 중에 가장 긴 패턴의 끝
  int inv[MAXN + 5]; // node가 몇번째 패턴의 끝? (없다면 -1)
  int match[MAXN + 5]; // node의 접미사 중 패턴의 개수(즉, node를 밟는 순간 증가하는 패턴
  매칭의 수)
  int size = 1; // node 개수
  int n; // 패턴 개수

  int c2i(char c) {
    // return c - 'a';
    // return c - 'A';
  }

  // 패턴 삽입, bfs로 실패링크와 출력링크 만들기
  void init(vector<string> &pat) {
    n = sz(pat);
    for (int i = 0; i < n; i++) {
      int cur = 0;
      for (auto j : pat[i]) {
        int nxt = c2i(j);
        if (chd[cur][nxt] == 0) {
          inv[size] = -1;
          match[size] = 0;
          chd[cur][nxt] = size++;
        }
        cur = chd[cur][nxt];
      }
      inv[cur] = i;
      match[cur] = 1;
    }

    queue<int> que;
    for (int i = 0; i < MAXC; i++) {
      // 루트가 가리키는 노드의 실패링크는 루트이다
      if (chd[0][i]) {
        fail[chd[0][i]] = 0;
        que.push(chd[0][i]);
      }
    }
    while (sz(que)) {
      int cur = que.front(); que.pop();

      // 실패 링크의 매칭 수를 더한다
      match[cur] += match[fail[cur]];

      // 출력링크는 내가 패턴의 끝이라면 나 자신이 되고, 그렇지 않으면 실패링크의 출력링크이다
      if (inv[cur] != -1) {
        output[cur] = cur;
      }
      else {
        output[cur] = output[fail[cur]];
      }

      for (int i = 0; i < MAXC; i++) {
        // chd[cur][i] = 나 -> i
        if (chd[cur][i]) {
          // 나의 실패링크 prv
          int prv = fail[cur];
          // 루트 또는 i를 가리키는 노드가 있을 때까지 실패링크를 타고 이동
          while (prv && chd[prv][i] == 0) {
            prv = fail[prv];
          }
          // 그러한 prv를 찾았고 (prv -> i)는 (나 -> i)의 실패링크가 된다
```

```
        fail[chd[cur][i]] = chd[prv][i];
        que.push(chd[cur][i]);
      }
    }
  }
}

void solve(string s) {
  vector<int> cnt(n); // cnt[i] = i번째 패턴의 등장 횟수

  int cur = 0;
  // 문자열 s에서 패턴 검색
  for (auto i : s) {
    int nxt = c2i(i);

    while (cur && chd[cur][nxt] == 0) {
      cur = fail[cur];
    }
    cur = chd[cur][nxt];
    if (output[cur]) {
      ++cnt[inv[output[cur]]];
    }
  }

  // 패턴 등장 횟수 계산
  vector<int> order;
  queue<int> que;
  que.push(0);
  while (sz(que)) {
    cur = que.front(); que.pop();
    order.push_back(cur);
    for (int i = 0; i < MAXC; i++) {
      if (chd[cur][i]) {
        que.push(chd[cur][i]);
      }
    }
  }

  reverse(all(order));

  for (auto i : order) {
    if (inv[i] != -1 && output[fail[i]]) {
      cnt[inv[output[fail[i]]]] += cnt[inv[i]];
    }
  }
}

void clear() {
  memset(chd, 0, sizeof(int) * size * MAXC);
  size = 1;
}
} trie;
```

4.2 Suffix Array

```
#define forn(i, n) for(int i = 0; i < n; ++i)
int conv(char c) { return c - 'a'; }
vector<int> get_sa(const char *s, int n) {
  vector<int> sa(n);
  int m = 26; // 문자의 갯수
  vector<int> cnt(max(n, m)), x(n), y(n);
  forn(i, n) cnt[x[i] = conv(s[i])]++;
  forn(i, m - 1) cnt[i + 1] += cnt[i];
  for (int i = n - 1; i >= 0; --i) sa[--cnt[x[i]]] = i;
  for (int len = 1, p = 0; p + 1 < n; len <<= 1, m = p + 1) {
    p = 0;
    for (int i = n - len; i < n; ++i) y[p++] = i;
    forn(i, n) if (sa[i] >= len) y[p++] = sa[i] - len;

    forn(i, m) cnt[i] = 0;
    forn(i, n) cnt[x[i]]++;
    forn(i, m - 1) cnt[i + 1] += cnt[i];
    for (int i = n - 1; i >= 0; --i) sa[--cnt[x[y[i]]]] = y[i];
    y = x;
    p = 0;
    x[sa[0]] = 0;
    forn(i, n - 1)
    x[sa[i + 1]] = sa[i] + len < n &&sa[i + 1] + len < n &&y[sa[i]] == y[sa[i + 1]] &&
    y[sa[i] + len] == y[sa[i + 1] + len] ? p : ++p;
  }
  return sa;
}
vector<int> get_lcp(const char *s, int n, vector<int> &sa) {
  vector<int> lcp(n), rank(n);
  forn(i, n) rank[sa[i]] = i;
  int k = 0, j;
  for (int i = 0; i < n; lcp[rank[i++]] = k) {
    if (rank[i] - 1 >= 0)
    for (k ? k-- : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; ++k);
  }
  return lcp;
}
```

4.3 Manacher Algorithm

```
int ma[MAXN + 5]; // ma[i] = s[i-k .. i+k]가 palindrome인 최대 k

void init(string s) {
  for (int i = 1, r = 0, p = 0; i < n; i++) {
    if (i <= r) {
      ma[i] = min(ma[2 * p - i], r - i);
    }
    while (i - ma[i] - 1 >= 0 && i + ma[i] + 1 < n && s[i - ma[i] - 1] == s[i + ma[i] +
    1]) {
      ++ma[i];
    }
    if (r < i + ma[i]) {
      r = i + ma[i], p = i;
    }
  }
}
```

4.4 Z Algorithm

```
int z[MAXN + 5]; // z[i] = s[i..]의 prefix와 s의 prefix가 일치하는 최대 길이

void init(string s) {
```

```
    for (int i = 1, r = 0, l = 0; i < n; i++) {
      if (i <= r) {
        z[i] = min(r - i + 1, z[i - l]);
      }
      while (i + z[i] < n && s[i + z[i]] == S[z[i]]) {
        ++z[i];
      }
      if (r < i + z[i] - 1) {
        r = i + z[i] - 1, l = i;
      }
    }
  }
```

4.5 Rabin–Karp Algorithm

```
constexpr int HMOD = 1e9 + 7;
mt19937_64 rng(time(0));
uniform_int_distribution<int> distr(HMOD >> 3, HMOD >> 1);
using H = array<int, 2>;
const H base = { distr(rng), distr(rng) };
H operator+(H l, H r) {
  for (int i = 0; i < 2; ++i) if ((l[i] += r[i]) >= HMOD) l[i] -= HMOD;
  return l;
}
H operator-(H l, H r) {
  for (int i = 0; i < 2; ++i) if ((l[i] -= r[i]) < 0) l[i] += HMOD;
  return l;
}
H operator*(H l, H r) {
  for (int i = 0; i < 2; ++i) l[i] = (long long)l[i] * r[i] % HMOD;
  return l;
}
struct substring_hash {
  static vector<H> pows;
  vector<H> cum = { {0, 0} };
  substring_hash(string x) {
    for (auto c : x) cum.push_back(cum.back() * base + H({ c, c }));
  }
  void extend(int len) {
    while (len >= (int)pows.size()) pows.push_back(pows.back() * base);
  }
  H hash(int l, int r) {
    int len = r - l + 1;
    extend(len);
    return cum[r + 1] - cum[l] * pows[len];
  }
};
vector<H> substring_hash::pows = { {1, 1} };
```

4.6 KMP

```
int nh, ns;
char h[1000005], s[1000005]; // h=total string, s=target string
int fail[1000005]; // size=length of s
vector<int> res;

void KMP()
{
  nh = strlen(h);
  ns = strlen(s);
  for (int i = 1, j = 0; i < ns; i++) {
    fail[i] = 0;
    while (j > 0 && s[i] != s[j]) j = fail[j - 1];
    if (s[i] == s[j]) fail[i] = ++j;
  }
  for (int i = 0, j = 0; i < nh; i++) {
    while (j > 0 && h[i] != s[j]) j = fail[j - 1];
    if (h[i] == s[j]) {
      if (j == ns - 1) {
        res.push_back(i - ns + 1);
        j = fail[j];
      }
      else j++;
    }
  }
}
```

5 Geometry

5.1 Line Intersection

```
#define x first
#define y second
using point = pair<ll, ll>;
using line = pair<point, point>;

template<typename T>
istream &operator>>(istream &in, T &a) { return in >> a.x >> a.y; }
point operator-(point a, point b) { return point(a.x - b.x, a.y - b.y); }
ll outpro(point a, point b) { return a.x * b.y - b.x * a.y; }
ll ccw(point a, point b, point c) {
  ll dir = outpro(b - a, c - a);
  if (!dir) return 0;
  if (dir > 0) return 1;
  else return -1;
}
bool cross(line a, line b) {
  ll c = ccw(a.x, a.y, b.x) * ccw(a.x, a.y, b.y);
  ll d = ccw(b.x, b.y, a.x) * ccw(b.x, b.y, a.y);
  if (!c && !d) {
    if (a.x > a.y) swap(a.x, a.y);
    if (b.x > b.y) swap(b.x, b.y);
    return !(a.y < b.x || b.y < a.x);
  }
  return c <= 0 && d <= 0;
}

int main() {
  line l1, l2;
  cin >> l1 >> l2;
  cout << cross(l1, l2);
  return 0;
}
```

5.2 Convex Hull + Rotating Calipers

```
using point = pair<int, int>;
#define x first
#define y second
```

```
point operator-(point a, point b) { return make_pair(a.x - b.x, a.y - b.y); }
int ccw(point a, point b) {
  long long x = 1LL * a.x * b.y - 1LL * a.y * b.x;
  return x < 0 ? -1 : x > 0;
}
int ccw(point p, point a, point b) { return ccw(a - p, b - p); }
long long dist2(point a, point b) { return 1LL * (a.x - b.x) * (a.x - b.x) + 1LL * (a.y
- b.y) * (a.y - b.y); }
signed main(void)
{
  int T = 1; cin >> T;
  while (T--)
  {
    int n; cin >> n;
    point ps[n];
    for (int i = 0; i < n; ++i) cin >> ps[i].x >> ps[i].y;
    swap(ps[0], *min_element(ps, ps + n));
    sort(ps + 1, ps + n, [&](point a, point b)
    {
      long long x = ccw(ps[0], a, b);
      return x ? x > 0 : a < b;
    });
    vector<point> hull;
    for (int i = 0; i < n; hull.push_back(ps[i++]))
    while (hull.size() >= 2 && ccw(hull[hull.size() - 2], hull[hull.size() - 1], ps[i])
    <= 0)
    hull.pop_back();
    point a = ps[0], b = ps[1];
    for (int i = 0, p = 0; i < (int)hull.size(); ++i)
    {
      while (p + 1 < (int)hull.size() && ccw(hull[i + 1] - hull[i], hull[p + 1] -
      hull[p]) > 0)
      {
        if (dist2(a, b) < dist2(hull[i], hull[p])) tie(a, b) = tie(hull[i], hull[p]);
        ++p;
      }
      if (dist2(a, b) < dist2(hull[i], hull[p])) tie(a, b) = tie(hull[i], hull[p]);
    }
    cout << a.x << ' ' << a.y << ' ' << b.x << ' ' << b.y << '\n';
  }
  return 0;
}
```

5.3 Monotone Chain Convex Hull

```
#define sz(x) int((x).size())
#define all(x) (x).begin(),(x).end()
using point = pair<int, int>;
vector<point> A, uhull, dhull;

sort(all(A));

for (auto i : A) {
  while (sz(uhull) >= 2 && ccw(uhull[sz(uhull) - 2], uhull.back(), i) >= 0) {
    uhull.pop_back();
  }
  uhull.push_back(i);
  while (sz(dhull) >= 2 && ccw(dhull[sz(dhull) - 2], dhull.back(), i) <= 0) {
    dhull.pop_back();
  }
  dhull.push_back(i);
}

dhull.insert(dhull.end(), next(uhull.rbegin()), prev(uhull.rend()));
```

5.4 Sort by angle

```
// O = 원점
// 4사분면(y축 미포함) - 1사분면 - 2사분면 - 3사분면
bool cmp(pi l, pi r) {
  if (O < l != O < r) {
    return l > r;
  }

  if (ccw(O, l, r) != 0) {
    return ccw(O, l, r) == 1;
  }

  return dist2(O, l) < dist2(O, r);
}

// 원점과 (x,y) 를 잇는 선분과 +x축이 이루는 각(-PI ~ PI)
atan2(x, y)

// 회전변환
(x, y) = (x * cos - y * sin, x * sin + y * cos)
```

5.5 Half-Plane Intersection

```
typedef double db;

const db EPS = 1e-9;
inline int sign(db a) { return a < -EPS ? -1 : a > EPS; }
inline int cmp(db a, db b) { return sign(a - b); }
struct P {
  db x, y;
  P() {}
  P(db _x, db _y) : x(_x), y(_y) {}
  P operator+(P p) { return { x + p.x, y + p.y }; }
  P operator-(P p) { return { x - p.x, y - p.y }; }
  P operator*(db d) { return { x * d, y * d }; }
  P operator/(db d) { return { x / d, y / d }; }

  db dot(P p) { return x * p.x + y * p.y; }
  db det(P p) { return x * p.y - y * p.x; }
  int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); }
};
struct L { //ps[0] -> ps[1]
  P ps[2];
  P &operator[](int i) { return ps[i]; }
  P dir() { return ps[1] - ps[0]; }
  L(P a, P b) {
    ps[0] = a;
    ps[1] = b;
  }
  bool include(P p) { return sign((ps[1] - ps[0]).det(p - ps[0])) > 0; }
```

```cpp
};
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
P isLL(P p1, P p2, P q1, P q2) {
  db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
  return (p1 * a2 + p2 * a1) / (a1 + a2);
}
P isLL(L l1, L l2) { return isLL(l1[0], l1[1], l2[0], l2[1]); }
bool parallel(L l0, L l1) { return sign(l0.dir().det(l1.dir())) == 0; }
bool sameDir(L l0, L l1) { return parallel(l0, l1) && sign(l0.dir().dot(l1.dir())) == 1;
}
bool cmp(P a, P b) {
  if (a.quad() != b.quad()) {
    return a.quad() < b.quad();
  }
  else {
    return sign(a.det(b)) > 0;
  }
}
bool operator < (L l0, L l1) {
  if (sameDir(l0, l1)) {
    return l1.include(l0[0]);
  }
  else {
    return cmp(l0.dir(), l1.dir());
  }
}
bool check(L u, L v, L w) {
  return w.include(isLL(u, v));
}
vector<P> halfPlaneIS(vector<L> &l) {
  sort(l.begin(), l.end());
  deque<L> q;
  for (int i = 0; i < (int)l.size(); ++i) {
    if (i && sameDir(l[i], l[i - 1])) continue;
    while (q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i])) q.pop_back();
    while (q.size() > 1 && !check(q[1], q[0], l[i])) q.pop_front();
    q.push_back(l[i]);
  }
  while (q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0])) q.pop_back();
  while (q.size() > 2 && !check(q[1], q[0], q[q.size() - 1])) q.pop_front();
  vector<P> ret;
  if (q.size() >= 3) for (int i = 0; i < (int)q.size(); ++i) ret.push_back(isLL(q[i],
  q[(i + 1) % q.size()]));
  return ret;
}
```

5.6 Bulldozer Algorithm

```cpp
using point = pair<int, int>;
#define x first
#define y second
point operator-(point a, point b) { return point(a.x - b.x, a.y - b.y); }
long long det(point a, point b) { return 1LL * a.x * b.y - 1LL * a.y * b.x; }
signed main()
{
  int N; cin >> N;
  vector<pair<point, long long>> P(N);
  for (auto &it : P) cin >> it.first.x >> it.first.y >> it.second;
  sort(P.begin(), P.end(), [](auto &a, auto &b)
  {
    if (a.first.y == b.first.y) return a.first.x < b.first.x;
    return a.first.y < b.first.y;
  });
  vector<pair<int, int>> swp;
  for (int i = 0; i < N; ++i)
  for (int j = i + 1; j < N; ++j)
  swp.push_back({ i, j });
  auto comp2 = [&](pair<int, int> &a, pair<int, int> &b) -> bool
  {
    long long d = det(P[a.second].first - P[a.first].first, P[b.second].first -
    P[b.first].first);
    if (d) return d > 0;
    else
    {
      if (P[a.first].first != P[b.first].first) return P[a.first].first <
      P[b.first].first;
      return P[a.second].first < P[b.second].first;
    }
  };
  sort(swp.begin(), swp.end(), comp2);
  vector<pair<point, long long>> srt = P;
  vector<int> pos(N); iota(pos.begin(), pos.end(), 0);
  auto comp3 = [&](pair<int, int> &a, pair<int, int> &b) -> long long
  {
    return det(P[a.second].first - P[a.first].first, P[b.second].first -
    P[b.first].first);
  };
  for (int i = 0; i < (int)swp.size(); )
  {
    int j = i;
    while (j < (int)swp.size() && comp3(swp[i], swp[j]) == 0)
    {
      int x = swp[j].first, y = swp[j].second;
      int &px = pos[x], &py = pos[y];
      swap(srt[px], srt[py]);
      swap(px, py);
      ++j;
    }
    i = j;
  }
  return 0;
}
```

5.7 Polygon Cut

```cpp
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
    res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
    res.push_back(cur);
```

```
  }
  return res;
}
```

5.8 Point in Polygon

```
#define sz(x) int((x).size())
using point = pair<int, int>;

bool pip(vector<point> &A, point p) {
  int n = sz(A);
  int cnt = 0;

  for (int i = 0; i < n; i++) {
    // 다각형 변 위에 있는 경우
    if (ccw(p, A[i], A[i == n - 1 ? 0 : i + 1]) == 0) {
      if (min(A[i], A[i == n - 1 ? 0 : i + 1]) <= p && p <= max(A[i], A[i == n - 1 ? 0 :
      i + 1])) {
        return 1;
      }
    }
    else {
      point pp = point(int(1e9) + 5, p.y);
      // 다각형이 꼭짓점을 지날 때 위쪽 변만 카운트
      if (intersect(A[i], A[i == n - 1 ? 0 : i + 1], p, pp) && (ccw(A[i], p, pp) == 1 ||
      ccw(A[i == n - 1 ? 0 : i + 1], p, pp) == 1)) {
        ++cnt;
      }
    }
  }

  return cnt & 1;
}


// 볼록 다각형(반시계 정렬)
bool pip(vector<point> &A, point p) {
  int n = sz(A);

  if (ccw(A[0], A[n - 1], p) == 1 || ccw(A[0], A[1], p) == -1) {
    return 0;
  }

  int here = -1;

  for (int le = 1, ri = n - 1; le <= ri;) {
    int mid = (le + ri) / 2;
    if (ccw(A[0], A[mid], p) >= 0) {
      here = mid;
      le = mid + 1;
    }
    else {
      ri = mid - 1;
    }
  }

  assert(~here);

  if (ccw(A[0], A[here], p) == 0) {
    return min(A[0], A[here]) <= p && p <= max(A[0], A[here]);
  }

  return ccw(A[here], A[here == n - 1 ? 0 : here + 1], p) >= 0;
}
```

5.9 Closest Pair of Points

```
#define x first
#define y second
using point = pair<int, int>;

signed main() {
  int n; cin >> n;

  vector<point> A(n);

  for (auto &i : A) cin >> i.x >> i.y;

  sort(all(A));

  auto cmp = [](point l, point r) {
    return l.y != r.y ? l.y < r.y : l.x < r.x;
  };

  set<pi, decltype(cmp)> se(cmp);

  int mn = dist2(A[0], A[1]);
  se.insert(A[0]); se.insert(A[1]);

  for (int i = 2, j = 0; i < n; i++) {
    while (j < i) {
      int x = A[i].x - A[j].x;
      if (x * x < mn) {
        break;
      }
      se.erase(A[j++]);
    }
    int d = sqrt(mn) + 1;
    auto lo = se.lower_bound({ minf, A[i].y - d });
    auto hi = se.upper_bound({ inf, A[i].y + d });
    while (lo != hi) {
      mn = min(mn, dist2(A[i], *lo));
      ++lo;
    }
    se.insert(A[i]);
  }

  cout << mn << '\n';

  return 0;
}
```

6 Math

6.1 FFT

```
typedef complex<double> base;
void FFT(vector<base> &a, bool invert)
```

```
{
  int N = a.size();
  for (int i = 1, j = 0; i < N; i++) {
    int bit = N >> 1;
    for (; j >= bit; bit >>= 1) j -= bit;
    j += bit;
    if (i < j) swap(a[i], a[j]);
  }
  double ang = invert ? -M_PI : M_PI;
  for (int len = 2; len <= N; len <<= 1, ang /= 2) {
    base wlen(cos(ang), sin(ang));
    for (int i = 0; i < N; i += len) {
      base w(1);
      for (int j = 0; j < len >> 1; j++) {
        base u = a[i | j], v = a[i | j | len >> 1] * w;
        a[i | j] = u + v;
        a[i | j | len >> 1] = u - v;
        w *= wlen;
      }
    }
  }
  if (invert)
  for (base &x : a) x /= N;
}
//when result of A(x)*B(x) is needed, resize a and b by na+nb-1 before using function
//size of both a and b is equal to length of equation A(x) and B(x)
void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res)
{
  vector<base> fa(a.begin(), a.end());
  vector<base> fb(b.begin(), b.end());
  int N = 1, na = a.size(), nb = b.size();
  while (N < na + nb) N <<= 1;
  fa.resize(N), fb.resize(N);
  FFT(fa, false), FFT(fb, false);
  for (int i = 0; i < N; i++) fa[i] *= fb[i];
  FFT(fa, true);
  res.resize(N);
  for (int i = 0; i < N; i++) res[i] = (int)(fa[i].real() + (fa[i].real() > 0 ? 0.5 :
  -0.5));
}
```

6.2 NTT

```
/*
Numeric FFT
P == MOD == A*2^B + 1
R  A   B  P
5  3   30 3221225473
3  17  27 2281701377
31 15  27 2013265921
3  7   26 469762049
3  119 23 998244353
*/
const int A = 119, B = 23, MOD = A << B | 1, R = 3;//MOD = 998244353
int power(int a, int n) {
  int res = 1;
  for (; n; n >>= 1, a = ((long long)a * a) % MOD)
  if (n & 1) res = ((long long)res * a) % MOD;
  return res;
}
inline int inv(int a) { return power(a, MOD - 2); }
void NTT(vector<int> &a, bool invert)
{
  int N = a.size();
  for (int i = 1, j = 0; i < N; i++) {
    int bit = N >> 1;
    for (; j >= bit; bit >>= 1) j -= bit;
    j += bit;
    if (i < j) swap(a[i], a[j]);
  }
  int ang = invert ? inv(R) : R;
  for (int len = 2; len <= N; len <<= 1) {
    int wlen = power(ang, (A << B) / len);
    for (int i = 0; i < N; i += len) {
      int w = 1;
      for (int j = 0; j < len >> 1; j++) {
        int u = a[i | j], v = (long long)a[i | j | len >> 1] * w % MOD;
        a[i | j] = (u + v) % MOD;
        a[i | j | len >> 1] = (u - v + MOD) % MOD;
        w = (long long)w * wlen % MOD;
      }
    }
  }
  if (invert)
  {
    int j = inv(N);
    for (int &x : a) x = (long long)x * j % MOD;
  }
}
vector<int> operator*(vector<int> a, vector<int> b)
{
  int N = 1, na = a.size(), nb = b.size();
  while (N < na + nb - 1) N <<= 1;
  a.resize(N); b.resize(N);
  NTT(a, false), NTT(b, false);
  for (int i = 0; i < N; ++i) a[i] = (long long)a[i] * b[i] % MOD;
  NTT(a, true);
  return a;
}
```

6.3 FWHT

```
template<typename T>
void fwht(vector<T> &a, bool invert) {
  int bit = 31 - __builtin_clz((int)a.size());
  for (int i = 0; i < bit; ++i)
  {
    for (int j = 0; j < (1 << bit); ++j)
    {
      if ((j >> i) & 1) continue;
      T u = a[j], v = a[j | (1 << i)];
      a[j] = u + v;
      a[j | (1 << i)] = u - v;
    }
  }
  if (invert) for (int i = 0; i < (int)a.size(); ++i)
```

```
    a[i] /= (T)a.size();
  }
  template<typename T>
  vector<T> xor_mult(vector<T> a, vector<T> b)
  {
    int ma = max((int)a.size(), (int)b.size());
    int N = 1;
    while (N < ma) N <<= 1;
    a.resize(N); b.resize(N);
    fwht(a, false); fwht(b, false);
    for (int i = 0; i < N; ++i) a[i] *= b[i];
    fwht(a, true);
    return a;
  }
```

6.4 Berlekamp-Massey Algorithm

```
  const int mod = 1e9 + 7;
  using lint = long long;
  lint ipow(lint x, lint p) {
    lint ret = 1, piv = x;
    while (p) {
      if (p & 1) ret = ret * piv % mod;
      piv = piv * piv % mod;
      p >>= 1;
    }
    return ret;
  }
  vector<int> berlekamp_massey(vector<int> x) {
    vector<int> ls, cur;
    int lf, ld;
    for (int i = 0; i < x.size(); i++) {
      lint t = 0;
      for (int j = 0; j < cur.size(); j++) {
        t = (t + 1ll * x[i - j - 1] * cur[j]) % mod;
      }
      if ((t - x[i]) % mod == 0) continue;
      if (cur.empty()) {
        cur.resize(i + 1);
        lf = i;
        ld = (t - x[i]) % mod;
        continue;
      }
      lint k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
      vector<int> c(i - lf - 1);
      c.push_back(k);
      for (auto &j : ls) c.push_back(-j * k % mod);
      if (c.size() < cur.size()) c.resize(cur.size());
      for (int j = 0; j < cur.size(); j++) {
        c[j] = (c[j] + cur[j]) % mod;
      }
      if (i - lf + (int)ls.size() >= (int)cur.size()) {
        tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
      }
      cur = c;
    }
    for (auto &i : cur) i = (i % mod + mod) % mod;
    return cur;
  }
  int get_nth(vector<int> rec, vector<int> dp, lint n) {
    int m = rec.size();
    vector<int> s(m), t(m);
    s[0] = 1;
    if (m != 1) t[1] = 1;
    else t[0] = rec[0];
    auto mul = [&rec](vector<int> v, vector<int> w) {
      int m = v.size();
      vector<int> t(2 * m);
      for (int j = 0; j < m; j++) {
        for (int k = 0; k < m; k++) {
          t[j + k] += 1ll * v[j] * w[k] % mod;
          if (t[j + k] >= mod) t[j + k] -= mod;
        }
      }
      for (int j = 2 * m - 1; j >= m; j--) {
        for (int k = 1; k <= m; k++) {
          t[j - k] += 1ll * t[j] * rec[k - 1] % mod;
          if (t[j - k] >= mod) t[j - k] -= mod;
        }
      }
      t.resize(m);
      return t;
    };
    while (n) {
      if (n & 1) s = mul(s, t);
      t = mul(t, t);
      n >>= 1;
    }
    lint ret = 0;
    for (int i = 0; i < m; i++) ret += 1ll * s[i] * dp[i] % mod;
    return ret % mod;
  }
  int guess_nth_term(vector<int> x, lint n) {
    if (n < x.size()) return x[n];
    vector<int> v = berlekamp_massey(x);
    if (v.empty()) return 0;
    return get_nth(v, x, n);
  }
  struct elem { int x, y, v; }; // A_(x, y) <- v, 0-based. no duplicate please..
  vector<int> get_min_poly(int n, vector<elem> M) {
    // smallest poly P such that A^i = sum_{j < i} {A^j \times P_j}
    vector<int> rnd1, rnd2;
    mt19937 rng(0x14004);
    auto randint = [&rng](int lb, int ub) {
      return uniform_int_distribution<int>(lb, ub)(rng);
    };
    for (int i = 0; i < n; i++) {
      rnd1.push_back(randint(1, mod - 1));
      rnd2.push_back(randint(1, mod - 1));
    }
    vector<int> gobs;
    for (int i = 0; i < 2 * n + 2; i++) {
      int tmp = 0;
      for (int j = 0; j < n; j++) {
```

```
      tmp += 1ll * rnd2[j] * rnd1[j] % mod;
      if (tmp >= mod) tmp -= mod;
    }
    gobs.push_back(tmp);
    vector<int> nxt(n);
    for (auto &i : M) {
      nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
      if (nxt[i.x] >= mod) nxt[i.x] -= mod;
    }
    rnd1 = nxt;
  }
  auto sol = berlekamp_massey(gobs);
  reverse(sol.begin(), sol.end());
  return sol;
}
lint det(int n, vector<elem> M) {
  vector<int> rnd;
  mt19937 rng(0x14004);
  auto randint = [&rng](int lb, int ub) {
    return uniform_int_distribution<int>(lb, ub)(rng);
  };
  for (int i = 0; i < n; i++) rnd.push_back(randint(1, mod - 1));
  for (auto &i : M) {
    i.v = 1ll * i.v * rnd[i.y] % mod;
  }
  auto sol = get_min_poly(n, M)[0];
  if (n % 2 == 0) sol = mod - sol;
  for (auto &i : rnd) sol = 1ll * sol * ipow(i, mod - 2) % mod;
  return sol;
}

int main() {
  ll n;
  cin >> n;
  cout << guess_nth_term({ 0,1,1,2,3,5,8 }, n);
  return 0;
}
```

6.5 Miller–Rabin Primality Test + Pollard Rho Factorization

```
ll mult(ll p, ll q, ll m) {
  p %= m;
  q %= m;
  ll r = 0;
  ll w = p;
  while (q) {
    if (q % 2)
    r = (r + w) % m;
    w = (w + w) % m;
    q >>= 1;
  }
  return r;
}
ll fpow(ll x, ll y, ll p) {
  ll result = 1;
  x = x % p;
  while (y > 0) {
    if (y & 1)
    result = mult(result, x, p);
    y = y >> 1;
    x = mult(x, x, p);
  }
  return result;
}

ll gcd(ll a, ll b) {
  if (a < b)
  swap(a, b);
  while (b != 0) {
    ll r = a % b;
    a = b;
    b = r;
  }
  return a;
}
bool millerRabin(ll n, ll a) {
  ll r = 0;
  ll d = n - 1;
  while (d % 2 == 0) {
    r += 1;
    d /= 2;
  }
  ll x = fpow(a, d, n);
  if (x == 1 || x == n - 1)
  return true;
  for (int i = 0; i < r - 1; i++) {
    x = fpow(x, 2, n);
    if (x == n - 1)
    return true;
  }
  return false;
}
bool isPrime(ll n) {
  ll aL[] = { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 };
  if (n == 1) return false;
  if (n <= 3) return true;
  if (!(n % 2)) return false;
  for (auto a : aL) {
    if (n == a)
    return true;
    if (!millerRabin(n, a))
    return false;
  }
  return true;
}
ll pollardRho(ll n) {
  if (isPrime(n))
  return n;
  if (n == 1)
  return 1;
  if (!(n % 2))
  return 2;
  ll x = rand() % n + 2;
  ll y = x;
```

```
    ll c = rand() % n + 1;
    ll d = 1;
    while (d == 1) {
      x = (mult(x, x, n) + c + n) % n;
      y = (mult(y, y, n) + c + n) % n;
      y = (mult(y, y, n) + c + n) % n;
      d = gcd(abs(x - y), n);
      if (d == n)
      return pollardRho(n);
    }
    if (isPrime(d))
    return d;
    else
    return pollardRho(d);
  }
  int main() {
    vector<ll> v;
    ll n;
    cin >> n;
    int cnt = 0;
    while (n > 1) {
      ll div = pollardRho(n);
      v.push_back(div);
      n /= div;
      cnt++;
    }
    sort(v.begin(), v.end());
    v.push_back(2);
    ll mul = 1;
    for (int i = 0, j = 0; i < cnt; i++, j++) {
      if (v[i] != v[i + 1]) {
        mul *= fpow(v[i], j + 1, 1e18) - fpow(v[i], j, 1e18);
        j = -1;
      }
    }
    cout << mul;
  }
```

6.6 Extended Euclidean Algorithm + Chinese Remainder Theorem

```
  // gcd(a,b) 를 반환하고,
  // 방정식 ax + by = gcd(a,b) 의 해 x,y 를 구한다
  int eea(int a, int b, int &x, int &y) {
    if (b == 0) {
      x = 1, y = 0;
      return a;
    }
    int q = a / b;
    int ret = eea(b, a - b * q, x, y);
    y = x - (x = y) * q;
    return ret;
  }

  // ax == b (mod n) (0 <= b < n)
  // cx == d (mod m) (0 <= d < m)
  int crt(int a, int b, int n, int c, int d, int m) {
    int g = gcd(a, gcd(b, n));
    a /= g, b /= g, n /= g;
    int inv, _;
    if (eea(a, n, inv, _) != 1) {
      return -1;
    }
    b *= inv;

    g = gcd(c, gcd(d, m));
    c /= g, d /= g, m /= g;

    if (eea(c, m, inv, _) != 1) {
      return -1;
    }
    d *= inv;

    g = eea(n, m, inv, _);
    if ((d - b) % g) {
      return -1;
    }
    n /= g;
    m *= n;
    // now, m is new modular

    int ret = (__int128(n) * inv % m * (d - b) + b) % m;
    if (ret < 0) {
      ret += m;
    }
    return ret;
  }
```

6.7 Finding Modular Inverse

```
  int fact[MAXN + 5];
  int inv[MAXN + 5];
  int fact_inv[MAXN + 5];

  void init() {
    fact[0] = fact[1] = inv[0] = inv[1] = fact_inv[0] = fact_inv[1] = 1;
    for (int i = 2; i <= MAXN; i++) {
      fact[i] = ll(fact[i - 1]) * i % mod;
    }
    for (int i = 2; i <= MAXN; i++) {
      inv[i] = inv[mod % i] * (mod - mod / i) % mod;
    }
    for (int i = 2; i <= MAXN; i++) {
      fact_inv[i] = ll(fact_inv[i - 1]) * inv[i] % mod;
    }
  }
```

6.8 Mobius Function

```
  // 뫼비우스 함수 = 제곱수를 인수로 가지면 0, 소인수의 개수가 홀수면 -1, 짝수면 1
  // N 의 모든 약수의 뫼비우스 함수 값의 합은 N 이 1일때는 1, 아니면 0

  int mbs[MAXN + 5];
  void init() {
    mbs[1] = 1;
    for (int i = 1; i <= MAXN; i++) {
      for (int j = i * 2; j <= MAXN; j += i) {
```

```
        mbs[j] -= mbs[i];
      }
    }
  }
```

6.9 Euler Phi Function

```
// 오일러 파이 함수 = N 이하의 자연수 중에서 N 과 서로소인 것의 개수

// N 의 오일러 파이 함수 값 구하기 O(sqrtN)
int phi(int n) {
  int ret = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      ret -= ret / i;
      while (n % i == 0) {
        n /= i;
      }
    }
  }
  if (n > 1) {
    ret -= ret / n;
  }

  return ret;
}

// 1 ~ N 의 오일러 파이 함수 값 구하기 O(NlgN)
int phi[MAXN + 5];
void init() {
  iota(A + 1, A + MAXN + 1, 1);
  for (int i = 1; i <= MAXN; i++) {
    for (int j = i * 2; j <= MAXN; j += i) {
      phi[j] -= phi[i];
    }
  }
}
```

6.10 Gauss-Jordan Elimination

```
// tutte matrix A가 있을 때, rank(A) / 2가 높은 확률로 최대 매칭이 되므로 get<1>(Guass(A)) /
2를 출력하면 된다. 시간 복잡도는 당연히 O(N^3)
// 투테행렬은 인접행렬에서 1 값을 [1,P-1] 사이의 랜덤값으로 바꾼 것이다
template<typename T> // return {rref, rank, det, inv}
tuple<vector<vector<T>>, T, T, vector<vector<T>>> Gauss(vector<vector<T>> a, bool square
= true) {
  int n = a.size(), m = a[0].size(), rank = 0;
  vector<vector<T>> out(n, vector<T>(m, 0)); T det = T(1);
  for (int i = 0; i < n; i++) if (square) out[i][i] = T(1);
  for (int i = 0; i < m; i++) {
    if (rank == n) break;
    if (IsZero(a[rank][i])) {
      T mx = T(0); int idx = -1; // fucking precision error
      for (int j = rank + 1; j < n; j++) if (mx < abs(a[j][i])) mx = abs(a[j][i]), idx =
      j;
      if (idx == -1 || IsZero(a[idx][i])) { det = 0; continue; }
      for (int k = 0; k < m; k++) {
        a[rank][k] = Add(a[rank][k], a[idx][k]);
        if (square) out[rank][k] = Add(out[rank][k], out[idx][k]);
      }
    }
    det = Mul(det, a[rank][i]);
    T coeff = Div(T(1), a[rank][i]);
    for (int j = 0; j < m; j++) a[rank][j] = Mul(a[rank][j], coeff);
    for (int j = 0; j < m; j++) if (square) out[rank][j] = Mul(out[rank][j], coeff);
    for (int j = 0; j < n; j++) {
      if (rank == j) continue;
      T t = a[j][i]; // Warning: [j][k], [rank][k]
      for (int k = 0; k < m; k++) a[j][k] = Sub(a[j][k], Mul(a[rank][k], t));
      for (int k = 0; k < m; k++) if (square) out[j][k] = Sub(out[j][k],
      Mul(out[rank][k], t));
    }
    rank++;
  }
  return { a, rank, det, out };
}
```

7 Miscellaneous

7.1 Policy Based Data Structure

```
#include<bits/extc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<typename T, typename U = less<T>>
using ordered_set = tree<T, null_type, U, rb_tree_tag,
tree_order_statistics_node_update>;
// order_of_key(k) = The number of items in a set that are strictly smaller than k
// find_by_order(k) = It returns an iterator to the k-th(0-based) element

struct splitmix64_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }

  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
    std::chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

template <typename K, typename V, typename Hash = splitmix64_hash>
using hash_map = __gnu_pbds::gp_hash_table<K, V, Hash>;

template <typename K, typename Hash = splitmix64_hash>
using hash_set = hash_map<K, __gnu_pbds::null_type, Hash>;
```

7.2 GCC pragma

```
#pragma GCC optimize("O3,Ofast,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,avx,avx2")
```

7.3 Custom hash

```
struct myhash {
  size_t operator()(const H p) const {
    return hash<ll>()((ll)(p[0]) << 32 | p[1]);
  };
};
using myset = gp_hash_table<H, null_type, myhash>;
```

7.4 FastIO

```
#include<bits/stdc++.h>
#include<sys/stat.h>
#include<sys/mman.h>
#include<unistd.h>
using namespace std;

const int BUF_SZ = 1 << 20;
char writebuf[BUF_SZ];
char *INPUT, *OUTPUT = writebuf;

inline void get(int &n) {
  n = 0;

  bool flag = 0;
  if (*INPUT == '-') {
    flag = 1;
    ++INPUT;
  }

  for (char i = *INPUT++; i & 16; n = (n << 1) + (n << 3) + (i & 15), i = *INPUT++);

  if (flag) {
    n = -n;
  }
}

inline void put(int n) {
  int sz = OUTPUT - writebuf;
  if (BUF_SZ - sz < 25) {
    write(1, writebuf, sz);
    OUTPUT = writebuf;
  }

  if (n == 0) {
    *OUTPUT++ = '0';
    return;
  }

  if (n < 0) {
    *OUTPUT++ = '-';
    n = -n;
  }

  char *tmp = OUTPUT;

  while (n) {
    *OUTPUT++ = (n % 10) | 48;
    n /= 10;
  }

  reverse(tmp, OUTPUT);
}

int main() {
  struct stat stt; fstat(0, &stt);
  INPUT = (char *)mmap(0, stt.st_size, PROT_READ, MAP_SHARED, 0, 0);

  int TC; get(TC);
  int a, b;
  while (TC--) {
    get(a); get(b);
    put(a + b); *OUTPUT++ = '\n';
  }

  write(1, writebuf, OUTPUT - writebuf);

  return 0;
}
```

7.5 Euler tour

i에서 j까지의 경로 $(in[i] < in[j])$
LCA가 i이면 $in[i]..in[j]$에 한 번 등장
그렇지 않으면 $end[i]..in[j]$에 한 번 등장, LCA는 따로 처리

7.6 Bipartite Graph

Minimum Vertex Cover = 모든 간선과 인접한 최소 정점 집합의 크기 = 최대 이분 매칭

Maximum Independent Set = 어떤 두 정점도 인접하지 않은 최대 정점 집합 = minimum vertex cover 의 여집합

Minimum Path Cover = 각 경로의 정점이 겹치지 않으면서 모든 정점을 커버하는 최소 경로 집합의 크기 = 전체 정점 수 - 각 노드를 in, out으로 분할한 최대 이분매칭($indegree \leq 1, outdegree \leq 1$ 임을 이용)(매칭 수는 간선 개수와 같고 경로 개수는 정점 수 - 간선 수와 같다)

Anti-Chain(반사슬)이란 방향 그래프 정점 부분집합으로 반사슬 내의 어떤 두 정점도 위상이 없음을 만족해야 합니다.
예를 들어 $\{a, b, c, d\}$가 있고 $a \to b, c \to b, b \to d$의 간선으로 이루어진 그래프에서 a와 c는 서로 위상이 없기 때문에 $\{a, c\}$는 반사슬입니다.
$\{b, c\}$는 $c \to b$가 존재하므로 반사슬이 아닙니다. 마찬가지로 $\{c, d\}$도 $c \to b \to d$의 path가 정의되므로 반사슬이 아닙니다.
이러한 부분집합 중 가장 큰 것을 Maximum Anti-Chain이라고 합니다.
딜워스 정리(Dilworth's Theorem)를 보면 방향 그래프에서 Maximum Anti-Chain의 크기는 Minimum Path Cover와 같다고 증명되어 있습니다.

anti chain = vertex cover 의 여집합 = 최대독립집합
anti chain을 구하기 전에 플로이드-워셜을 돌려야 한다

7.7 Circulation, LR Flow

정점에 수요, 공급 개념이 추가된다. 공급은 -(수요) 라고 생각하면 된다.
수요라는 것은 그만큼의 유량을 필요로 한다는 것이므로 정점이 그만큼의 유량을 흡수해서 가져가게 된다.
즉, 정점 v의 수요를 demand(v) 라고 하면 정점으로 들어오는 유량의 합 - 정점에서 나가는 유량의 합 = demand(v) 이어야 한다.

서큘레이션 찾기
일단 양수 수요의 절대값 합과 음수 수요의 절댓값 합이 같아야 한다. 그 합을 D라 하자.

최대 유량 문제로 해결할 수 있다. 소스와 싱크를 만들고 소스에서 수요가 양수인 정점으로 간선을 잇고, 수요가 음수인 정점에서 싱크로 간선을 잇는다.
이제 최대 유량이 D이면 서큘레이션이 존재하는 것이다.
LR Flow = 간선 (u, v)에 하한 L, 상한 R이 존재하는 경우
간선 (u, v)에 용량 R-L을 부여하고, u의 수요에 +L, v의 수요에 -L을 한다. 즉 간선에 이미 L이 흐르고 있다고 모델링을 해주는 것이다.

소스와 싱크가 있는 그래프에서 LR flow 구하기
새로운 소스와 싱크를 만든다.
기존 싱크에서 기존 소스로 용량이 무한대인 간선을 잇는다.
위 내용처럼 모델링을 해주고, 새로운 소스에서 새로운 싱크로의 최대 유량이 하한 L의 합과 같은지 보면 된다.
최대 유량을 구하고 싶다면 이렇게 구성한 유량 그래프에서 원래 소스에서 원래 싱크로 가는 유량을 찾을 수 있을 때까지 계속 찾으면 된다.

7.8 2-SAT

0. A
(A or A)와 동치

1. $A = B$
(A or $\sim B$) and ($\sim A$ or B)와 동치

2. $A \neq B$
(A or B) and ($\sim A$ or $\sim B$)와 동치

3. $A => B$
($\sim A$ or B)와 동치

4. $\sim$(A and B)
($\sim A$ or $\sim B$)와 동치

5. (A and B) or (C and D)
(A or C) and (A or D) and (B or C) and (B or D)와 동치

6. A, B, C 중 2개 이상
(A or B) and (B or C) and (C or A)와 동치

7. A, B, C 중 1개 이하
($\sim A$ or $\sim B$) and ($\sim B$ or $\sim C$) and ($\sim C$ or $\sim A$)와 동치

8. A_1, A_2, ..., A_k 중 1개 이하
k개의 변수를 새로 만듭니다. 이를 B_1, B_2, ..., B_k라고 합시다. 이때 B_i는 (A_1 or A_2 or ... or A_i)와 동치이도록 하고 싶습니다. 이제 다음 조건들을 넣으면 됩니다.
1. $B_i \Rightarrow B_{i+1}$
2. $A_i \Rightarrow B_i$
3. $B_i \Rightarrow \sim A_{i+1}$
(여기서 $P \Rightarrow Q$는 "P이면 Q이다"라는 명제입니다! P에서 Q로 가는 간선이 아닙니다.)
1번과 2번은 B_i와 (A_1 or A_2 or ... or A_i)를 동치로 만드는 데 필요하고, 3번은 $i < j$에 대해 ($\sim A_i$ or $\sim A_j$)임을 보장시키는 데 필요합니다.
이전 글에서 보았듯이 각각의 $P \Rightarrow Q$는 ($\sim P$ or Q)로 바꿀 수 있습니다. 만들어진 절의 개수는 $O(k)$입니다.

9. A_1, A_2, ..., A_k 중 k-1개 이상
8번과 똑같이 하되 참거짓을 뒤집어 주기만 하면 됩니다.

7.9 Math

벨 수 - 크기 i인 집합을 분할하는 방법의 개수
집합 $\{1, 2, ..., i\}$ 을 분할한다고 할 때, 1을 원소로 갖는 집합의 크기가 k가 되도록 분할하는 경우의 수를 모두 더함
$dp[i] = \sum_{k=1}^{i} \binom{i-1}{k-1} \cdot dp[i-k]$

제2 스털링 수 - 크기 i인 집합을 k개의 공집합이 아닌 부분집합으로 분할하는 방법의 개수
집합 $\{1, 2, ..., i\}$ 을 분할한다고 할 때, i가 혼자만 있는 그룹으로 분할되거나, $\{1, 2, ..., i-1\}$ 에서 분할된 k개의 집합 중에 선택해서 들어가는 경우
$dp[i][k] = dp[i-1][k-1] + dp[i-1][k] \cdot k$

제1 스털링 수 - 크기 i인 집합을 k개의 공집합이 아닌 사이클(원순열)로 분할하는 방법의 개수
집합 $\{1, 2, ..., i\}$ 을 분할한다고 할 때, i가 혼자만 있는 그룹으로 분할되거나, $\{1, 2, ..., i-1\}$ 에서 분할된 k개의 사이클 중에 선택해서 들어가는 경우→이때 어떤 사이클로 들어갈 때, 정확히 그 사이클의 크기만큼 선택지가 존재함→총 선택지는 $i-1$
$dp[i][k] = dp[i-1][k-1] + dp[i-1][k] \cdot (i-1)$

하키스틱 패턴
$\binom{r}{r} + \binom{r+1}{r} + \cdots \binom{n}{r} = \binom{n+1}{r+1}$

픽의 정리
모든 꼭짓점이 격자점 위에 존재하는 단순 다각형의 넓이를 A, 격자 다각형 내부에 있는 격자점의 수를 i, 변 위에 있는 격자점의 수를 b라고 하면 다음이 성립한다.
$A = i + \frac{b}{2} - 1$

키르히호프 정리
무향 그래프의 (정점의 차수 대각 행렬) - (인접 행렬) L을 만든다. 행과 열을 하나씩 제거하고 이것을 L^*라 하자. 그래프의 spanning tree의 개수는 $det(L^*)$ 이다.

평면 그래프
꼭짓점의 수를 v, 변의 수를 e, 면의 수를 f라고 하면, 평면 그래프의 경우 다음의 식이 성립한다.
$v - e + f = 2$

카탈란 수
$\frac{1}{n+1} \cdot \binom{2n}{n}$
C_n은 -1과 1 값으로 만들어진 수열 $(a_1, a_2, \cdots, a_{2n})$ 에서 $a_1 + a_2 + \cdots + a_{2n} = 0$ 이고, 부분합 $a_1 + a_2 + \cdots + a_i = 0$ 이 모두 0 이상이 되도록 하는 방법의 수이다.
C_n은 -1과 1 값으로 만들어진 수열 $(a_1, a_2, \cdots, a_{2n+2})$ 에서 $a_1 + a_2 + \cdots + a_{2n+2} = 0$ 이고, 부분합 $a_1 + a_2 + \cdots + a_i = 0$ 이 모두 0 보다 크도록 하는 방법의 수이다.
C_n은 $n+1$개의 항에 괄호를 씌우는 모든 경우의 수이다. 혹은 $n+1$개의 항에 이항 연산을 적용하는 순서의 모든 가지수로도 볼 수 있다. 예를 들어 $n = 3$일 때, 4개의 항에 대하여 다섯 개의 괄호 표현식이 존재한다.
$((ab)c)d \quad (a(bc))d \quad (ab)(cd) \quad a((bc)d) \quad a(b(cd))$
이항 연산의 적용 순서는 이진 트리로도 나타낼 수 있다. 따라서 C_n은 $n+1$개의 리프 노드를 갖는 이진 순서 트리의 개수임을 알 수 있다.
C_n은 동형이 아닌 모든 full binary tree 가운데 자식을 가진 노드가 n개인 트리의 개수이다.(full binary tree는 한 개의 자식만 가진 노드가 없고, 모든 노드가 두 개의 자식을 가졌거나 혹은 리프 노드인 트리를 뜻한다.)
C_n은 $n+2$각형을 n개의 삼각형으로 나누는 방법의 수이다.
$(0, 0)$에서 (n, n)까지 격자점을 따라 오른쪽 또는 위쪽으로 한 칸씩 이동하는 경로 중, 대각선 $y = x$의 좌상단으로 넘어가지 않는 경로의 개수
좌괄호 n개와 우괄호 n개를 나열하여 올바른 괄호열을 만드는 경우의 수
크기 $2 * n$짜리 표의 각 칸에 1부터 $2n$까지의 수를 집어넣는데, 아래로 가거나 오른쪽으로 갈수록 수가 커지는 방법의 수

카탈란 convolution
? 길이가 L, 앞쪽 $('-')$가 F, ?를 제외한 나머지에서 $'('-')'$를 A라고 하면 $C(L, (L-A)/2) - C(L, (L-A)/2 + F + 1)$