UNAM

Contents

1 Data structures

1.1 Fenwick Tree

```
ll fwsum(ll k){
    ll s = 0;
    while(k >= 1){
        s += tr[k];
        k -= k & (-k);
    }
    return s;
}

void fwadd(ll k, ll x, int n){
    while(k <= n){
        tr[k] += x;
        k += k & (-k);
    }
}
```

1.2 Segment Tree

```
int x[400000] = {1000000001};
int n;
void update(int a, int b)
{
  a += n;
  x[a] = b;
  for (a /= 2; a >= 1; a /= 2)
  {
    x[a] = min(x[a * 2], x[a * 2 + 1]);
  }
}
int find(int a, int b)
{
  a += n;
  b += n;
  int s = 1000000000;
  while (a <= b)
  {
    if (a % 2 == 1)
      s = min(s, x[a++]);
    if (b % 2 == 0)
      s = min(s, x[b--]);
    a /= 2;
    b /= 2;
  }
  return s;
}
```

1.3 Segment tree(Lazy)

```
template <typename num_t>
struct segtree {
  int n, depth;
  vector<num_t> tree, lazy;

  void init(int s, long long* arr) {
    n = s;
    tree = vector<num_t>(4 * s, 0);
    lazy = vector<num_t>(4 * s, 0);
    init(0, 0, n - 1, arr);
  }

  num_t init(int i, int l, int r, long long* arr) {
    if (l == r) return tree[i] = arr[l];

    int mid = (l + r) / 2;
    num_t a = init(2 * i + 1, l, mid, arr),
          b = init(2 * i + 2, mid + 1, r, arr);
```

```
    return tree[i] = a.op(b);
  }

  void update(int l, int r, num_t v) {
  if (l > r) return;
    update(0, 0, n - 1, l, r, v);
  }

  num_t update(int i, int tl, int tr, int ql, int qr, num_t v) {
    eval_lazy(i, tl, tr);

  if (tr < ql || qr < tl) return tree[i];
    if (ql <= tl && tr <= qr) {
      lazy[i] = lazy[i].val + v.val;
      eval_lazy(i, tl, tr);
      return tree[i];
    }

    int mid = (tl + tr) / 2;
    num_t a = update(2 * i + 1, tl, mid, ql, qr, v),
          b = update(2 * i + 2, mid + 1, tr, ql, qr, v);
    return tree[i] = a.op(b);
  }

  num_t query(int l, int r) {
  if (l > r) return num_t::null_v;
    return query(0, 0, n-1, l, r);
  }

  // int get_first(int v, int tl, int tr, int l, int r, int x) {
  //   eval_lazy(0, tl, tr);
  //   if(tl > r || tr < l) return -1;
  //   if(tree[v].val < x) return -1;

  //   if (tl== tr) return tl;

  //   int tm = tl + (tr-tl)/2;
  //   int left = get_first(2*v+1, tl, tm, l, r, x);
  //   if(left != -1) return left;
  //   return get_first(2*v+2, tm+1, tr, l ,r, x);
  // }

  num_t query(int i, int tl, int tr, int ql, int qr) {
    eval_lazy(i, tl, tr);

    if (ql <= tl && tr <= qr) return tree[i];
    if (tr < ql || qr < tl) return num_t::null_v;

    int mid = (tl + tr) / 2;
    num_t a = query(2 * i + 1, tl, mid, ql, qr),
          b = query(2 * i + 2, mid + 1, tr, ql, qr);
    return a.op(b);
  }

  void eval_lazy(int i, int l, int r) {
    tree[i] = tree[i].lazy_op(lazy[i], (r - l + 1));
    if (l != r) {
      lazy[i * 2 + 1] = lazy[i].val + lazy[i * 2 + 1].val;
      lazy[i * 2 + 2] = lazy[i].val + lazy[i * 2 + 2].val;
    }

    lazy[i] = num_t();
  }
};
```

1.4 Segment tree types

```
struct max_t {
  long long val;
  static const long long null_v = -9223372036854775807LL;

  max_t(): val(0) {}
  max_t(long long v): val(v) {}

  max_t op(max_t& other) {
    return max_t(max(val, other.val));
  }

  max_t lazy_op(max_t& v, int size) {
    return max_t(val + v.val);
  }
};


struct min_t {
  long long val;
```

```
  static const long long null_v = 922337203685477580 7LL;

  min_t(): val(0) {}
  min_t(long long v): val(v) {}

  min_t op(min_t& other) {
    return min_t(min(val, other.val));
  }

  min_t lazy_op(min_t& v, int size) {
    return min_t(val + v.val);
  }
};


struct sum_t {
  long long val;
  static const long long null_v = 0;

  sum_t(): val(0) {}
  sum_t(long long v): val(v) {}

  sum_t op(sum_t& other) {
    return sum_t(val + other.val);
  }

  sum_t lazy_op(sum_t& v, int size) {
    return sum_t(val + v.val * size);
  }
};
```

1.5 DSU

```
class DisjointSets {
  private:
  vector<int> parents;
  vector<int> sizes;

  public:
  DisjointSets(int size) : parents(size), sizes(size, 1) {
    for (int i = 0; i < size; i++) { parents[i] = i; }
  }

  /** @return the "representative" node in x's component */
  int find(int x) {
    return parents[x] == x ? x : (parents[x] = find(parents[x]));
  }

  /** @return whether the merge changed connectivity */
  bool unite(int x, int y) {
    int x_root = find(x);
    int y_root = find(y);
    if (x_root == y_root) { return false; }

    if (sizes[x_root] < sizes[y_root]) { swap(x_root, y_root); }
    sizes[x_root] += sizes[y_root];
    parents[y_root] = x_root;
    return true;
  }

  /** @return whether x and y are in the same connected component */
  bool connected(int x, int y) { return find(x) == find(y); }
};
//recibe valores de un grafo cero indexado, retorna valores uno indexado
```

1.6 DSU with deletion

```
struct dsu_save {
    int v, rnkv, u, rnku;

    dsu_save() {}

    dsu_save(int _v, int _rnkv, int _u, int _rnku)
        : v(_v), rnkv(_rnkv), u(_u), rnku(_rnku) {}
};

struct dsu_with_rollbacks {
    vector<int> p, rnk;
    int comps;
    stack<dsu_save> op;

    dsu_with_rollbacks() {}

    dsu_with_rollbacks(int n) {
        p.resize(n);
        rnk.resize(n);
```

```
        for (int i = 0; i < n; i++) {
            p[i] = i;
            rnk[i] = 0;
        }
        comps = n;
    }

    int find_set(int v) {
        return (v == p[v]) ? v : find_set(p[v]);
    }

    bool unite(int v, int u) {
        v = find_set(v);
        u = find_set(u);
        if (v == u)
            return false;
        comps--;
        if (rnk[v] > rnk[u])
            swap(v, u);
        op.push(dsu_save(v, rnk[v], u, rnk[u]));
        p[v] = u;
        if (rnk[u] == rnk[v])
            rnk[u]++;
        return true;
    }

    void rollback() {
        if (op.empty())
            return;
        dsu_save x = op.top();
        op.pop();
        comps++;
        p[x.v] = x.v;
        rnk[x.v] = x.rnkv;
        p[x.u] = x.u;
        rnk[x.u] = x.rnku;
    }
};

struct query {
    int v, u;
    bool united;
    query(int _v, int _u) : v(_v), u(_u) {
    }
};

struct QueryTree {
    vector<vector<query>> t;
    dsu_with_rollbacks dsu;
    int T;

    QueryTree() {}

    QueryTree(int _T, int n) : T(_T) {
        dsu = dsu_with_rollbacks(n);
        t.resize(4 * T + 4);
    }

    void add_to_tree(int v, int l, int r, int ul, int ur, query& q) {
        if (ul > ur)
            return;
        if (l == ul && r == ur) {
            t[v].push_back(q);
            return;
        }
        int mid = (l + r) / 2;
        add_to_tree(2 * v, l, mid, ul, min(ur, mid), q);
        add_to_tree(2 * v + 1, mid + 1, r, max(ul, mid + 1), ur, q);
    }
     // nodes to unite in query, interval of time when nodes are
         connected
    void add_query(query q, int l, int r) {
        add_to_tree(1, 0, T - 1, l, r, q);
    }

    void dfs(int v, int l, int r, vector<int>& ans) {
        for (query& q : t[v]) {
            q.united = dsu.unite(q.v, q.u);
        }
        if (l == r)
            ans[l] = dsu.comps;
        else {
            int mid = (l + r) / 2;
            dfs(2 * v, l, mid, ans);
            dfs(2 * v + 1, mid + 1, r, ans);
        }
```

```
        for (query q : t[v]) {
            if (q.united)
                dsu.rollback();
        }
    }

    vector<int> solve() {
        vector<int> ans(T);
        dfs(1, 0, T - 1, ans);
        return ans;
    }
}; // when using map of pairs be careful with order as m[{a, b}] != m[{b
    , a}]
```

1.7 BIT

```
#define MAXN 10000
int bit[MAXN];
void update(int x, int val){
  for(; x < MAXN; x+=x&-x)
    bit[x] += val;
}
int get(int x){
  int ans = 0;
  for(; x; x-=x&-x)
    ans += bit[x];
  return ans;
}
```

1.8 BIT2D

```
#define MAXN 1000
int bit[MAXN][MAXN];

void update(int x, int y, int val){
  for(; x < MAXN; x+=x&-x)
    for(int j = y; j < MAXN; j+=j&-j)
      bit[x][j] += val;
}

int get(int x, int y){
  int ans = 0;
  for(; x; x-=x&-x)
    for(int j = y; j; j-=j&-j)
      ans += bit[x][j];
  return ans;
}

int get(int x1, int y1, int x2, int y2){
  return get(x2, y2) - get(x1-1, y2) - get(x2, y1-1) + get(x1-1, y1-1);
}
```

1.9 Oset

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using oset = __gnu_pbds::tree<
    T, __gnu_pbds::null_type, less<T>, __gnu_pbds::rb_tree_tag,
    __gnu_pbds::tree_order_statistics_node_update
>;
// order_of_key() primero mayor o igual;
// find_by_order() apuntador al elemento k;
// oset<pair<int,int>> os;
// os.insert({1,2});
// os.insert({2,3});
// os.insert({5,6});
// ll k=os.order_of_key({2,0});
// cout<<k<<endl; // 1
// pair<int,int> p=*os.find_by_order(k);
// cout<<p.f<<" "<<p.s<<endl; // 2 3
// os.erase(p);
// p=*os.find_by_order(k);
// cout<<p.f<<" "<<p.s<<endl; // 5 6


// check if upperbound or lowerbound does what you want
// because they give better time.
```

1.10 Multioset

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using oset = __gnu_pbds::tree<
    T, __gnu_pbds::null_type, less<T>, __gnu_pbds::rb_tree_tag,
    __gnu_pbds::tree_order_statistics_node_update
>;
```

```

//en main

oset<pair<int,int>> name;
    map<int,int> cuenta;
    function<void(int)> meter = [&] (int val) {
        name.insert({val,++cuenta[val]});
    };
    auto quitar = [&] (int val) {
        name.erase({val,cuenta[val]--});
    };

meter(x);
quitar(y);
multioset.order_of_key({y+1,-1})-multioset.order_of_key({x,0})
```

1.11 Treap

```
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
  if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
    auto pa = split(n->l, k);
    n->l = pa.second;
    n->recalc();
    return {pa.first, n};
  } else {
    auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
    n->r = pa.first;
    n->recalc();
    return {n, pa.second};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    l->recalc();
    return l;
  } else {
    r->l = merge(l, r->l);
    r->recalc();
    return r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto pa = split(t, pos);
  return merge(merge(pa.first, n), pa.second);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
}

// Usage
// create treap
// Node* name=nullptr;
// insert element
// name=ins(name, new Node(val), pos);
// Node* x = new Node(val);
// name = ins(name, x, pos);
// merge two treaps (name before x)
// name=merge(name, x);
// split treap (this will split treap in two treaps,
// first with elements [0, pos) and second with elements [pos, n))
// pa will be pair of two treaps
```

```
// auto pa = split(name, pos);
// move range [l, r) to index k
// move(name, l, r, k);
// iterate over treap
// each(name, [&](int val) {
//     cout << val << ' ';
// });
```

1.12 Treap with inversion

```
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  bool rev = 0;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
  void push();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }
void Node::push() {
  if (rev) {
    rev = 0;
    swap(l, r);
    if (l) l->rev ^= 1;
    if (r) r->rev ^= 1;
  }
}

template<class F> void each(Node* n, F f) {
  if (n) { n->push(); each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  n->push();
  if (cnt(n->l) >= k) {
    auto pa = split(n->l, k);
    n->l = pa.second;
    n->recalc();
    return {pa.first, n};
  } else {
    auto pa = split(n->r, k - cnt(n->l) - 1);
    n->r = pa.first;
    n->recalc();
    return {n, pa.second};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  l->push();
  r->push();
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    l->recalc();
    return l;
  } else {
    r->l = merge(l, r->l);
    r->recalc();
    return r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto pa = split(t, pos);
  return merge(merge(pa.first, n), pa.second);
}

// Example application: reverse the range [l, r]
void reverse(Node*& t, int l, int r) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l);
  tie(b,c) = split(b, r - l + 1);
  b->rev ^= 1;
  t = merge(merge(a, b), c);
}

void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l);
  tie(b,c) = split(b, r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
```

```
}
```

1.13 HLD

```
vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
void dfs(int u, int p = 0) {
  par[u][0] = p;
  dep[u] = dep[p] + 1;
  sz[u] = 1;
  for (int i = 1; i <= LG; i++) par[u][i] = par[par[u][i - 1]][i - 1];
  if (p) g[u].erase(find(g[u].begin(), g[u].end(), p));
  for (auto &v : g[u]) if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if(sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
    }
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = LG; k >= 0; k--) if (dep[par[u][k]] >= dep[v]) u = par[u
      ][k];
  if (u == v) return u;
  for (int k = LG; k >= 0; k--) if (par[u][k] != par[v][k]) u = par[u][k
      ], v = par[v][k];
  return par[u][0];
}
int kth(int u, int k) {
  assert(k >= 0);
  for (int i = 0; i <= LG; i++) if (k & (1 << i)) u = par[u][i];
  return u;
}
int T, head[N], st[N], en[N];
void dfs_hld(int u) {
  st[u] = ++T;
  for (auto v : g[u]) {
    head[v] = (v == g[u][0] ? head[u] : v);
    dfs_hld(v);
  }
  en[u] = T;
}
int n;
//verify that the operation is the correct one.
int query_up(int u, int v) {
  int ans = -inf;
  while(head[u] != head[v]) {
    ans = max(ans, t.query(1, 1, n, st[head[u]], st[u]));
    u = par[head[u]][0];
  }
  ans = max(ans, t.query(1, 1, n, st[v], st[u]));
  return ans;
}
int query(int u, int v) {
  int l = lca(u, v);
  int ans = query_up(u, l);
  if (v != l) ans = max(ans, query_up(v, kth(v, dep[v] - dep[l] - 1)));
  return ans;
}
```

2 Graphs

2.1 BFS

```
queue<int> q;
q.push(x);
vis[x]=1;
dis[x]=0;
while(!q.empty()){
  int s=q.front();
  q.pop();
  for(auto u:adj[s]){
    if(!vis[u]){
      vis[u]=1;
      dis[u]=dis[s]+1;
      q.push(u);
    }
  }
}
```

2.2 DFS

```
vector<vector<int>> adj;
vector<bool> vis;
void dfs(int a){
  vis[a]=1;
  for(auto u:adj[a]){
```

```
    if(!vis[u]){
      dfs(u);
    }
  }
}

//inside solve()

adj.assign(n,vector<int>());
vis.assign(n,0);
```

2.3 Bellman Ford

```
vector <tuple <ll, ll, ll>> edges(m);
for(int i = 0; i < m; i++){
  ll a, b, c; cin >> a >> b >> c;
  edges[i] ={a, b, c};
}

ll distance[n + 1];
for(int i = 0; i <= n; i++) distance[i] = INF;
distance[1] = 0;
for(int i = 1; i <= n - 1; i++){
  bool change = false;
  for(auto e: edges){
    ll a, b, w;
    tie(a, b, w) = e;
    ll temp = distance[b];
    distance[b] = min(distance[b], distance[a] + w);
    if(temp != distance[b]) change = true;
  }
  if(!change) break;
}
```

2.4 Dijkstra

```
vector<vector<pair<ll,ll>>> g;  // u->[(v,cost)]
vector<ll> dist;
int n; // be careful with redeclaration of n, works 0 indexed
void dijkstra(int x){
  dist.assign(n,-1);
  priority_queue<pair<ll,int> > q;
  dist[x]=0;
  q.push({0,x});
  while(!q.empty()){
    x=q.top().second;
    ll c=-q.top().first;
    q.pop();
    if(dist[x]!=c)continue;
    for(int i=0;i<g[x].size();i++){
      int y=g[x][i].first;
      ll c=g[x][i].second;
      if(dist[y]<0||dist[x]+c<dist[y])
        dist[y]=dist[x]+c,q.push({-dist[y],y});
    }
  }
}

void solve(){
  g.assign(n,vector<pair<ll,ll>>());
}
```

2.5 Floyd Warshal

```
ll distances[n + 1][n + 1];
for(int i = 1; i <= n; i++){
  for(int j = 1; j <= n; j++){
    if(i == j) distances[i][j] = 0;
    else if(adj[i][j]) distances[i][j] = adj[i][j];
    else distances[i][j] = INF;
  }
}

for(int k = 1; k <= n; k++){
  for(int i = 1; i <= n; i++){
    for(int j = 1; j <= n; j++){
      distances[i][j] = min(distances[i][j], distances[i][k] + distances
          [k][j]);
    }
  }
}
```

2.6 Strongly Connected Components

```
vector<vector<int>> adj,adjr;
vector<bool> vis;
vector<int> order,comp;
void dfs(int a){
```

```
  vis[a]=1;
  for(auto u:adj[a]){
    if(!vis[u]){
      dfs(u);
    }
  }
  order.pb(a);
}
void dfsr(int a,int k){
  vis[a]=1;
  comp[a]=k;
  for(auto u:adjr[a]){
    if(!vis[u]){
      dfsr(u,k);
    }
  }
}

void solve() {
  int n,m;cin>>n>>m;
  adj.assing(n,vector<int>());
  adjr.assing(n,vector<int>());
  comp.resize(n);
  for(int i=0;i<m;i++){
    int a,b;cin>>a>>b;a--;b--;
    adj[a].pb(b);
    adjr[b].pb(a);
  }
  vis.assign(n,0);
  for(int i=0;i<n;i++){
    if(!vis[i])dfs(i);
  }
  vis.assign(n,0);
  int c=0;
  for(int i=n-1;i>=0;i--){
    if(!vis[order[i]]){
      dfsr(order[i],c);
      c++;
    }
  }

}
```

2.7 Condensation Graph

```
//after scc
vector<vector<int>> adj_scc;

void cndstn(int c){
  adj_scc.assign(c,vector<int>());
  for(ll i=0;i<n;i++){
    for(auto u:adj[i]){
      if(comp[u]!=comp[i]){
        adj_scc[comp[i]].pb(comp[u]);
      }
    }
  }
}
```

2.8 Bridges Articulation

```
int n;
vector<vector<int>> adj;

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            //bridge
            //if (low[to] > tin[v]){}
            //articulation
            //if (low[to] >= tin[v] && p!=-1){}
        }
    }

}
```

```
26
27 void find_bridges() {
28     timer = 0;
29     visited.assign(n, false);
30     tin.assign(n, -1);
31     low.assign(n, -1);
32     for (int i = 0; i < n; ++i) {
33         if (!visited[i])
34             dfs(i);
35     }
36 }
```

2.9 LCA

```
1  class Tree
2  {
3  public:
4    const int root = 0;
5
6    const vector<vector<int>> &adj;
7    const int log2dist;
8    vector<int> par;
9    vector<vector<int>> pow2ends;
10   vector<int> depth;
11   /** works with 0 indexed graph ** /
12   /** check if graph is connected, if not, just iterate trough all nodes
         **/
13   /** use DFS to calculate the depths and parents of each node */
14   void process(int at, int prev)
15   {
16     depth[at] = depth[prev] + 1;
17     for (int n : adj[at])
18     {
19       if (n != prev)
20       {
21         process(n, at);
22         par[n] = at;
23       }
24     }
25   }
26
27 public:
28   Tree(const vector<vector<int>> &adj)
29       : adj(adj), log2dist(std::ceil(std::log2(adj.size()))), par(adj.
            size()),
30         pow2ends(par.size(), vector<int>(log2dist + 1)), depth(adj.size
              ())
31   {
32     par[root] = depth[root] = -1;
33     process(root, root);
34
35     for (int n = 0; n < par.size(); n++)
36     {
37       pow2ends[n][0] = par[n];
38     }
39     for (int p = 1; p <= log2dist; p++)
40     {
41       for (int n = 0; n < par.size(); n++)
42       {
43         int halfway = pow2ends[n][p - 1];
44         if (halfway == -1)
45         {
46           pow2ends[n][p] = -1;
47         }
48         else
49         {
50           pow2ends[n][p] = pow2ends[halfway][p - 1];
51         }
52       }
53     }
54   }
55
56   /** @return the kth parent of node n */
57   int kth_parent(int n, int k)
58   {
59     if (k > par.size())
60     {
61       return -1;
62     }
63     int at = n;
64     for (int pow = 0; pow <= log2dist; pow++)
65     {
66       if ((k & (1 << pow)) != 0)
67       {
68         at = pow2ends[at][pow];
69         if (at == -1)
```

```
        {
          break;
        }
      }
    }
    return at;
  }

  /** @return the LCA of nodes n1 and n2 */
  int lca(int n1, int n2)
  {
    if (depth[n1] < depth[n2])
    {
      return lca(n2, n1);
    }
    // lift n1 up to the same height as n2
    n1 = kth_parent(n1, depth[n1] - depth[n2]);
    if (n1 == n2)
    {
      return n2; // in this case, n2 is a direct ancestor of n1
    }

    // move the nodes up as long as they don't meet
    for (int i = log2dist; i >= 0; i--)
    {
      if (pow2ends[n1][i] != pow2ends[n2][i])
      {
        n1 = pow2ends[n1][i];
        n2 = pow2ends[n2][i];
      }
    }
    // at this point, the lca will be the parent of either node
    return pow2ends[n1][0];
  }
};
```

2.10 2sat

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

const int N = 3e5 + 9;

/*
zero Indexed
we have vars variables
F=(x_0 XXX y_0) and (x_1 XXX y_1) and ... (x_{vars-1} XXX y_{vars-1})
here {x_i,y_i} are variables
and XXX belongs to {OR,XOR}
is there any assignment of variables such that F=true
*/
struct twosat {
  int n;  // total size combining +, -. must be even.
  vector< vector<int> > g, gt;
  vector<bool> vis, res;
  vector<int> comp;
  stack<int> ts;
  twosat(int vars = 0) {
    n = vars << 1;
    g.resize(n);
    gt.resize(n);
  }

  //zero indexed, be careful
  //if you want to force variable a to be true in OR or XOR combination
  //add addOR (a,1,a,1);
  //if you want to force variable a to be false in OR or XOR combination
  //add addOR (a,0,a,0);

  //(x_a or (not x_b))-> af=1,bf=0
  void addOR(int a, bool af, int b, bool bf) {
    a += a + (af ^ 1);
    b += b + (bf ^ 1);
    g[a ^ 1].push_back(b);  // !a => b
    g[b ^ 1].push_back(a);  // !b => a
    gt[b].push_back(a ^ 1);
    gt[a].push_back(b ^ 1);
  }
  //(!x_a xor !x_b)-> af=0, bf=0
  void addXOR(int a, bool af, int b, bool bf) {
    addOR(a, af, b, bf);
    addOR(a, !af, b, !bf);
  }
  void _add(int a,bool af,int b,bool bf) {
    a += a + (af ^ 1);
    b += b + (bf ^ 1);
```

```
    g[a].push_back(b);
    gt[b].push_back(a);
  }
  //add this type of condition->
  //add(a,af,b,bf) means if a is af then b must need to be bf
  void add(int a,bool af,int b,bool bf) {
    _add(a, af, b, bf);
    _add(b, !bf, a, !af);
  }
  void dfs1(int u) {
    vis[u] = true;
    for(int v : g[u]) if(!vis[v]) dfs1(v);
    ts.push(u);
  }
  void dfs2(int u, int c) {
    comp[u] = c;
    for(int v : gt[u]) if(comp[v] == -1) dfs2(v, c);
  }
  bool ok() {
    vis.resize(n, false);
    for(int i = 0; i < n; ++i) if(!vis[i]) dfs1(i);
    int scc = 0;
    comp.resize(n, -1);
    while(!ts.empty()) {
      int u = ts.top();
      ts.pop();
      if(comp[u] == -1) dfs2(u, scc++);
    }
    res.resize(n / 2);
    for(int i = 0; i < n; i += 2) {
      if(comp[i] == comp[i + 1]) return false;
      res[i / 2] = (comp[i] > comp[i + 1]);
    }
    return true;
  }
};

int main() {
  int n, m; cin >> n >> m;
  twosat ts(n);
  for(int i = 0; i < m; i++){
    int u, v, k; cin >> u >> v >> k;
    --u; --v;
    if(k) ts.add(u, 0, v, 0), ts.add(u, 1, v, 1), ts.add(v, 0, u, 0), ts
        .add(v, 1, u, 1);
    else ts.add(u, 0, v, 1), ts.add(u, 1, v, 0), ts.add(v, 0, u, 1), ts.
        add(v, 1, u, 0);
  }
  int k = ts.ok();
  if(!k) cout<<"Impossible\n";
  else {
    vector<int> v;
    for(int i = 0; i < n; i++) if(ts.res[i]) v.push_back(i);
    cout << (int)v.size() << '\n';
    for(auto x: v) cout << x + 1 << ' ';
    cout << '\n';
  }
  return 0;
}
```

2.11 Bipartite check

```
int n;
vector<vector<int>> adj;

vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
for (int st = 0; st < n; ++st) {
  if (side[st] == -1) {
    q.push(st);
    side[st] = 0;
    while (!q.empty()) {
      int v = q.front();
      q.pop();
      for (int u : adj[v]) {
        if (side[u] == -1) {
          side[u] = side[v] ^ 1;
          q.push(u);
        } else {
          is_bipartite &= side[u] != side[v];
        }
      }
    }
  }
}
```

```
cout << (is_bipartite ? "YES" : "NO") << endl;
```

2.12 LCA with RMQ

```
class Tree //if lca needed just copy lca and kth_parent functions from lca.cpp or change return from max_edge_cost function
{
private:
  const int root = 0;

  const vector<vector<int>> &adj;
  const vector <vector<int>> &weight;
  const int log2dist;
  vector<int> par;
  vector<int> edgepar;
  vector<vector<int>> pow2ends;
  vector<vector<int>> maxedge;
  vector<int> depth;

  /** use DFS to calculate the depths and parents of each node */
  void process(int at, int prev)
  {
    int cont = 0;
    depth[at] = depth[prev] + 1;
    for (int n : adj[at])
    {
      if (n != prev)
      {
        process(n, at);
        par[n] = at;
        edgepar[n] = weight[at][cont];
      }
      cont++;
    }
  }

public:
  Tree(const vector<vector<int>> &adj, const vector <vector<int>> &weight)
      : adj(adj), weight(weight), log2dist(std::ceil(std::log2(adj.size()))), par(adj.size()), edgepar(adj.size()),
        pow2ends(par.size(), vector<int>(log2dist + 1)), maxedge(par.size(), vector <int>(log2dist + 1)), depth(adj.size())
  {
    par[root] = depth[root] = -1;
    edgepar[root] = 0;
    process(root, root);

    for (int n = 0; n < par.size(); n++)
    {
      pow2ends[n][0] = par[n];
      maxedge[n][0] = edgepar[n];
    }
    for (int p = 1; p <= log2dist; p++)
    {
      for (int n = 0; n < par.size(); n++)
      {
        int halfway = pow2ends[n][p - 1];
        if (halfway == -1)
        {
          pow2ends[n][p] = -1;
          maxedge[n][p] = -1;
        }
        else
        {
          pow2ends[n][p] = pow2ends[halfway][p - 1];
          maxedge[n][p] = max(maxedge[n][p - 1], maxedge[halfway][p - 1]);
        }
      }
    }
  }

  /** @return the kth parent of node n with max edge cost */
  pair <int, int> kth_parent_with_max_edge(int n, int k)
  {
    if (k > par.size())
    {
      pair <int, int> temp;
      temp.first = -1;
      temp.second = 0;
      return temp;
    }
    int at = n, maxcost = 0;
    for (int pot = 0; pot <= log2dist; pot++)
```

```
    {
      if ((k & (1 << pot)) != 0)
      {
        maxcost = max(maxedge[at][pot], maxcost);
        at = pow2ends[at][pot];
        if (at == -1)
        {
          break;
        }
      }
    }
    pair <int, int> ans;
    ans.first = at;
    ans.second = maxcost;
    return ans;
  }


  int max_edge_cost(int n1, int n2)
  {
    if (depth[n1] < depth[n2])
    {
      return max_edge_cost(n2, n1);
    }
    int maxcost;
    // lift n1 up to the same height as n2 and find max edge of lifting
    pair <int, int> temp = kth_parent_with_max_edge(n1, depth[n1] -
        depth[n2]);
    n1 = temp.first;
    maxcost = temp.second;
    if (n1 == n2)
    {
      return maxcost; // in this case, n2 is a direct ancestor of n1,
          return maxcost
    }

    // move the nodes up as long as they don't meet
    for (int i = log2dist; i >= 0; i--)
    {
      if (pow2ends[n1][i] != pow2ends[n2][i])
      {
        maxcost = max(maxcost, maxedge[n1][i]);
        n1 = pow2ends[n1][i];
        maxcost = max(maxcost, maxedge[n2][i]);
        n2 = pow2ends[n2][i];
      }
    }
    maxcost = max(maxcost, maxedge[n1][0]);
    maxcost = max(maxcost, maxedge[n2][0]);
    // at this point, the lca will be the parent of either node
    return maxcost;
  }
};
```

2.13 Find Centroid

```
const int maxn = 200010;

int n;
vector<int> adj[maxn];
int subtree_size[maxn];

int get_subtree_size(int node, int parent = -1) {
  int &res = subtree_size[node];
  res = 1;
  for (int i : adj[node]) {
    if (i == parent) { continue; }
    res += get_subtree_size(i, node);
  }
  return res;
}

int get_centroid(int node, int parent = -1) {
  for (int i : adj[node]) {
    if (i == parent) { continue; }

    if (subtree_size[i] * 2 > n) { return get_centroid(i, node); }
  }
  return node;
}

// Usage:
// get_subtree_size(0) to calculate subtree sizes
// get_centroid(0) to get the centroid of the tree
```

2.14 Centroid Decomposition

```cpp
vector<vector<int>> adj;
vector<bool> is_removed;
vector<int> subtree_size;
// probably you want to add a parent array here


/** DFS to calculate the size of the subtree rooted at `node` */
int get_subtree_size(int node, int parent = -1) {
  subtree_size[node] = 1;
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    subtree_size[node] += get_subtree_size(child, node);
  }
  return subtree_size[node];
}

/**
 * Returns a centroid (a tree may have two centroids) of the subtree
 * containing node `node` after node removals
 * @param node current node
 * @param tree_size size of current subtree after node removals
 * @param parent parent of u
 * @return first centroid found
 */
int get_centroid(int node, int tree_size, int parent = -1) {
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    if (subtree_size[child] * 2 > tree_size) {
      return get_centroid(child, tree_size, node);
    }
  }
  return node;
}

/** Build up the centroid decomposition recursively */
void build_centroid_decomp(int node = 0, int parent = -1) {
  int centroid = get_centroid(node, get_subtree_size(node));
  // add parent array here

  // do something

  is_removed[centroid] = true;

  for (int child : adj[centroid]) {
    if (is_removed[child]) { continue; }
    build_centroid_decomp(child, centroid);
  }
}
```

2.15 Koenig

```python

# ADD Maximum_Bipartite_Matching in python

def augment(u, bigraph, visit, timestamp, match):
    """ find augmenting path starting from u, by recursive DFS """
    for v in bigraph[u]:
        if visit[v] < timestamp:
            visit[v] = timestamp
            if match[v] is None or augment(match[v], bigraph,
                                           visit, timestamp, match):
                match[v] = u       # found an augmenting path
                return True
    return False


def max_bipartite_matching(bigraph):
    """Bipartie maximum matching

    :param bigraph: adjacency list, index = vertex in U,
                    value = neighbor list in V
    :comment: U and V can have different cardinalities
    :returns: matching list, match[v] == u iff (u, v) in matching
    :complexity: `O(|V|*|E|)`
    """
    nU = len(bigraph)   # nU = cardinality of U, nV = card. of V
    nV = max(max(adjlist, default=-1) for adjlist in bigraph) + 1
    match = [None] * nV
    visit = [-1] * nV   # timestamp of last visit
    for u in range(nU):
        augment(u, bigraph, visit, u, match)
    return match



def alternate(u, bigraph, visitU, visitV, matchV):
```

```
    """extend␣alternating␣tree␣from␣free␣vertex␣u.
␣␣␣␣␣␣visitU,␣visitV␣marks␣all␣vertices␣covered␣by␣the␣tree.
␣␣␣␣"""
    visitU[u] = True
    for v in bigraph[u]:
        if not visitV[v]:
            visitV[v] = True
            assert matchV[v] is not None   # otherwise match not maximum
            alternate(matchV[v], bigraph, visitU, visitV, matchV)


#
    ############################################################################

# Here starts Koenig algorithm
def koenig(bigraph):
    """Bipartie␣minimum␣vertex␣cover␣by␣Koenig's␣theorem

␣␣␣␣:param␣bigraph:␣adjacency␣list,␣index␣=␣vertex␣in␣U,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣value␣=␣neighbor␣list␣in␣V
␣␣␣␣:assumption:␣U␣=␣V␣=␣{0,␣1,␣2,␣...,␣n␣-␣1}␣for␣n␣=␣len(bigraph)
␣␣␣␣:returns:␣boolean␣table␣for␣U,␣boolean␣table␣for␣V
␣␣␣␣:comment:␣selected␣vertices␣form␣a␣minimum␣vertex␣cover,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣i.e.␣every␣edge␣is␣adjacent␣to␣at␣least␣one␣selected␣
    vertex
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣and␣number␣of␣selected␣vertices␣is␣minimum
␣␣␣␣:complexity:␣`O(|V|*|E|)`
␣␣␣␣"""
    V = range(len(bigraph))
    matchV = max_bipartite_matching(bigraph)
    matchU = [None for u in V]
    for v in V:                         # -- build the mapping from U to V
        if matchV[v] is not None:
            matchU[matchV[v]] = v
    visitU = [False for u in V]         # -- build max alternating forest
    visitV = [False for v in V]
    for u in V:
        if matchU[u] is None:           # -- starting with free vertices in
             U
            alternate(u, bigraph, visitU, visitV, matchV)
    inverse = [not b for b in visitU]
    return (inverse, visitV)
```

2.16 Max Bipartite Matching

```

int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    //... reading the graph ...

    mt.assign(k, -1);
    for (int v = 0; v < n; ++v) {
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d␣%d\n", mt[i] + 1, i + 1);
}
```

```

def augment(u, bigraph, visit, timestamp, match):
    """␣find␣augmenting␣path␣starting␣from␣u,␣by␣recursive␣DFS␣"""
    for v in bigraph[u]:
        if visit[v] < timestamp:
            visit[v] = timestamp
            if match[v] is None or augment(match[v], bigraph,
                                           visit, timestamp, match):
```

```
                match[v] = u       # found an augmenting path
                return True
    return False


def max_bipartite_matching(bigraph):
    """Bipartie maximum matching

    :param bigraph: adjacency list, index = vertex in U,
                    value = neighbor list in V
    :comment: U and V can have different cardinalities
    :returns: matching list, match[v] == u iff (u, v) in matching
    :complexity: `O(|V|*|E|)`
    """
    nU = len(bigraph)   # nU = cardinality of U, nV = card. of V
    nV = max(max(adjlist, default=-1) for adjlist in bigraph) + 1
    match = [None] * nV
    visit = [-1] * nV   # timestamp of last visit
    for u in range(nU):
        augment(u, bigraph, visit, u, match)
    return match



def alternate(u, bigraph, visitU, visitV, matchV):
    """extend alternating tree from free vertex u.
       visitU, visitV marks all vertices covered by the tree.
    """
    visitU[u] = True
    for v in bigraph[u]:
        if not visitV[v]:
            visitV[v] = True
            assert matchV[v] is not None   # otherwise match not maximum
            alternate(matchV[v], bigraph, visitU, visitV, matchV)
```

3 Math

3.1 Binary Exponentiation

```
long long binexp(long long a, long long b)
{
  long long res = 1;
  while (b > 0)
  {
    if (b & 1)
      res = res * a;
    a = a * a;
    b >>= 1;
  }
  return res;
}
```

3.2 Binom small numbers without mod

```
double bin[501][501];
void build(){
  bin[0][0] = 1;
  for(int k = 0; k < 501; k++){
    for(int n = k; n < 501; n++){
      if(k == 0){
        bin[n][k] = 1;
        continue;
      }
      if(n == k){
        bin[n][k] = 1;
        continue;
      }
      bin[n][k] = bin[n - 1][k - 1] + bin[n - 1][k];
    }
  }
}
```

3.3 Factorial

```
ll vals[1000001];
void fi(){
  vals[0]=1;
  vals[1]=1;
  for(int i=2;i<1000001;i++){
    vals[i]=i*vals[i-1];
    vals[i]%=MOD;
  }
}
```

3.4 Binomial Coefficient

```
ll binom(ll a,ll b){
```

```
  if(b>a)return 0;
  ll ans=vals[a];
  ans*=inverse(vals[b]);
  ans%=MOD;
  ans*=inverse(vals[a-b]);
  ans%=MOD;
  return ans;
}
```

3.5 Fibonacci (Logn)

```
void fib(ll n, ll&x, ll&y){
  if(n==0){
    x = 0;
    y = 1;
    return ;
  }

  if(n&1){
    fib(n-1, y, x);
    y=(y+x)%MOD;
  }else{
    ll a, b;
    fib(n>>1, a, b);
    y = (a*a+b*b)%MOD;
    x = (a*b + a*(b-a+MOD))%MOD;
  }
}

// Usage
// ll x, y;
// fib(10, x, y);
// cout << x << " " << y << endl;
// This will output 55 89
```

3.6 GCD

```
ll gcd(ll a,ll b){
    if(a==0)return b;
    return gcd(b%a,a);
}
```

3.7 Sieve O(nlog(log(n)))

```
const int kMaxV = 1e6;

int sieve[kMaxV + 1];

//stores some prime (not necessarly the minimum one)
void primes()
{
    for (int i = 4; i <= kMaxV; i += 2)
        sieve[i] = 2;
    for (int i = 3; i <= kMaxV / i; i += 2)
    {
        if (sieve[i])
            continue;
        for (int j = i * i; j <= kMaxV; j += i)
            sieve[j] = i;
    }
}

vector<int> PrimeFactors(int x)
{
    if (x == 1)
        return {};

    unordered_set<int> primes;
    while (sieve[x])
    {
        primes.insert(sieve[x]);
        x /= sieve[x];
    }
    primes.insert(x);
    return {primes.begin(), primes.end()};
}
```

3.8 Inverse Modulo

```
ll inverse(ll a,ll b=MOD,ll n=1,ll m=0){
  if(a==1){
    return n;
  }
  if(a<b){
    long long x=b/a;
    m+=(x*n);
    m=m%MOD;
```

```
    b=b%a;
    return inverse(a,b,n,m);
  }
  else if(b==1){
    return(MOD-m);
  }
  else{
    long long x=a/b;
    n+=(x*m);
    n=n%MOD;
    a=a%b;
    return inverse(a,b,n,m);
  }
}

// other inverse
ll inverse(ll a){
  return binexp(a,MOD-2);
}
```

3.9 Divisors

```
long long numberOfDivisors(long long num)
{
  long long total = 1;
  for (int i = 2; (long long)i * i <= num; i++)
  {
    if (num % i == 0)
    {
      int e = 0;
      do
      {
        e++;
        num /= i;
      } while (num % i == 0);
      total *= e + 1;
    }
  }
  if (num > 1)
  {
    total *= 2;
  }
  return total;
}

long long SumOfDivisors(long long num)
{
  long long total = 1;

  for (int i = 2; (long long)i * i <= num; i++)
  {
    if (num % i == 0)
    {
      int e = 0;
      do
      {
        e++;
        num /= i;
      } while (num % i == 0);

      long long sum = 0, pow = 1;
      do
      {
        sum += pow;
        pow *= i;
      } while (e-- > 0);
      total *= sum;
    }
  }
  if (num > 1)
  {
    total *= (1 + num);
  }
  return total;
}
```

3.10 NTT

```
// number theory transform

const int MOD = 998244353, ROOT = 3;
// const int MOD = 7340033, ROOT = 5;
// const int MOD = 167772161, ROOT = 3;
// const int MOD = 469762049, ROOT = 3;

int power(int base, int exp) {
```

```
    int res = 1;
    while (exp) {
        if (exp % 2) res = 1LL * res * base % MOD;
        base = 1LL * base * base % MOD;
        exp /= 2;
    }
    return res;
}

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]);
  }
  for (int len = 2; len <= n; len <<= 1) {
    int wlen = power(ROOT, (MOD - 1) / len);
    if (invert) wlen = power(wlen, MOD - 2);
    for (int i = 0; i < n; i += len) {
      int w = 1;
      for (int j = 0; j < len / 2; j++) {
        int u = a[i + j], v = 1LL * a[i + j + len / 2] * w % MOD;
        a[i + j] = u + v < MOD ? u + v : u + v - MOD;
        a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + MOD;
        w = 1LL * w * wlen % MOD;
      }
    }
  }
  if (invert) {
    int n_inv = power(n, MOD - 2);
    for (int& x : a) x = 1LL * x * n_inv % MOD;
  }
}

vector<int> multiply(vector<int>& a, vector<int>& b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  a.resize(n), b.resize(n);
  ntt(a, false), ntt(b, false);
  for (int i = 0; i < n; i++) a[i] = 1LL * a[i] * b[i] % MOD;
  ntt(a, true);
  return a;
}
// usage
// vector<int> a = {1, 2, 3}, b = {4, 5, 6};
// vector<int> c = multiply(a, b);
// for (int x : c) cout << x << " ";
```

3.11 FFT

```
const int N = 3e5 + 9;

const double PI = acos(-1);
struct base {
  double a, b;
  base(double a = 0, double b = 0) : a(a), b(b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
  int n = p.size(), i = 0;
  for(int j = 1; j < n - 1; ++j) {
    for(int k = n >> 1; k > (i ^= k); k >>= 1);
    if(j < i) swap(p[i], p[j]);
  }
  for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {
    double ang = 2 * PI / m;
    base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
    for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w * wn) {
        base t = w * p[j + l];
        p[j + l] = p[j] - t;
        p[j] = p[j] + t;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;
}
vector<long long> multiply(vector<int> &a, vector<int> &b) {
  int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
```

```
  while(sz < t) sz <<= 1;
  vector<base> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; ++i) {
    x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);
    y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);
  }
  fft(x), fft(y);
  for(int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
  fft(z, 1);
  vector<long long> ret(sz);
  for(int i = 0; i < sz; ++i) ret[i] = (long long) round(z[i].a);
  while((int)ret.size() > 1 && ret.back() == 0) ret.pop_back();
  return ret;
}
// usage
// vector<int> a = {1, 2, 3}, b = {4, 5, 6};
// vector<long long> c = multiply(a, b);
```

3.12 Matrix Exponentiation

```
struct Mat {
  int n, m;
  vector<vector<int>> a;
  Mat() { }
  Mat(int _n, int _m) {n = _n; m = _m; a.assign(n, vector<int>(m, 0)); }
  Mat(vector< vector<int> > v) { n = v.size(); m = n ? v[0].size() : 0;
      a = v; }
  inline void make_unit() {
    assert(n == m);
    for (int i = 0; i < n; i++)  {
      for (int j = 0; j < n; j++) a[i][j] = i == j;
    }
  }
  inline Mat operator + (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
      }
    }
    return ans;
  }
  inline Mat operator - (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
      }
    }
    return ans;
  }
  inline Mat operator * (const Mat &b) {
    assert(m == b.n);
    Mat ans = Mat(n, b.m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < b.m; j++) {
        for(int k = 0; k < m; k++) {
          ans.a[i][j] = (ans.a[i][j] + 1LL * a[i][k] * b.a[k][j] % mod)
              % mod;
        }
      }
    }
    return ans;
  }
  inline Mat pow(long long k) {
    assert(n == m);
    Mat ans(n, n), t = a; ans.make_unit();
    while (k) {
      if (k & 1) ans = ans * t;
      t = t * t;
      k >>= 1;
    }
    return ans;
  }
  inline Mat& operator += (const Mat& b) { return *this = (*this) + b; }
  inline Mat& operator -= (const Mat& b) { return *this = (*this) - b; }
  inline Mat& operator *= (const Mat& b) { return *this = (*this) * b; }
  inline bool operator == (const Mat& b) { return a == b.a; }
  inline bool operator != (const Mat& b) { return a != b.a; }
};

// Usage
// Mat a(n, n);
// Mat b(n, n);
```

```
// Mat c = a * b;
// Mat d = a + b;
// Mat e = a - b;
// Mat f = a.pow(k);
// a.a[i][j] = x;
```

3.13 Mobius

```
int mob[N];
void mobius() {
  mob[1] = 1;
  for (int i = 2; i < N; i++){
    mob[i]--;
    for (int j = i + i; j < N; j += i) {
      mob[j] -= mob[i];
    }
  }
}
```

3.14 Phi

```
void phi_1_to_n(int n) {
  vector<int> phi(n + 1);
  phi[0] = 0;
  phi[1] = 1;
  for (int i = 2; i <= n; i++)
    phi[i] = i - 1;

  for (int i = 2; i <= n; i++)
    for (int j = 2 * i; j <= n; j += i)
      phi[j] -= phi[i];
}
```

3.15 Prefix sum Phi

```
vector<ll> sieve(kMaxV + 1,0);
vector<ll> phi(kMaxV + 1,0);

void primes()
{
  phi[1]=1;
  vector<ll> pr;
  for(int i=2;i<kMaxV;i++){
    if(sieve[i]==0){
      sieve[i]=i;
      pr.pb(i);
      phi[i]=i-1;
    }
    for(auto p:pr){
      if(p>sieve[i]||i*p>=kMaxV)break;
      sieve[i*p]=p;
      phi[i*p]=(p==sieve[i]?p:p-1)*phi[i];
    }
  }
  for(int i=1;i<kMaxV;i++){
    phi[i]+=phi[i-1];
    phi[i]%=MOD;
  }
}

map<ll,ll> m;
ll PHI(ll a){
  if(a<kMaxV)return phi[a];
  if(m.count(a))return m[a];
  // if(a<3)return 1;
  m[a]=((((a%MOD)*((a+1)%MOD))%MOD)*inverse(2));
  m[a]%=MOD;
  long long i=2;
  while(i<=a){
      long long j=a/i;
      j=a/j;
      m[a]+=MOD;
      m[a]-=((j-i+1)*PHI(a/i))%MOD;
      m[a]%=MOD;
      i=j+1;
  }
  m[a]%=MOD;
  return m[a];
}
```

3.16 Identities

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$
$$C_n = \frac{1}{n+1}\binom{2n}{n}$$
$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$
$$\sigma(n) = O(\log(\log(n))) \text{ (number of divisors of } n)$$
$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$
$$\sum_{i=1}^{n} F_i = F_{n+2} - 1$$
$$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_iF_j$$

(Möbius Inv. Formula) $\mu(p^k) = [k=0] - [k=1]$ Let $g(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} g(d)\mu\left(\frac{n}{d}\right)$. (Dirichlet Convolution) Let f, g be arithmetic functions, then $(f*g)(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right)$. If f, g are multiplicative, then so is $f*g$.

$$n = \sum_{d|n} \phi(d)$$

Lucas' Theorem: $\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod p$ where $m = \sum_{i=0}^{k} m_i p^i$ and $n = \sum_{i=0}^{k} n_i p^i$.

3.17 Burnside's Lemma

Dado un grupo G de permutaciones y un conjunto X de n elementos, el número de órbitas de X bajo la acción de G es igual al promedio del número de puntos fijos de las permutaciones en G.
Formalmente, el número de órbitas es $\frac{1}{|G|}\sum_{g\in G} f(g)$ donde $f(g)$ es el número de puntos fijos de g.
Ejemplo: Dado un collar con n cuentas y 2 colores, el número de collares diferentes que se pueden formar es $\frac{1}{n}\sum_{i=0}^{n} f(i)$ donde $f(i)$ es el número de collares que quedan fijos bajo una rotación de i posiciones.
Para contar el número de collares que quedan fijos bajo una rotación de i posiciones, se puede usar la fórmula $f(i) = 2^{\gcd(i,n)}$.
Para un collar de n cuentas y k colores, el número de collares diferentes que se pueden formar es $\frac{1}{n}\sum_{i=0}^{n} k^{\gcd(i,n)}$
Ejemplo: Dado un cubo con 6 caras y k colores, el número de cubos diferentes que se pueden formar es $\frac{1}{24}\sum_{i=0}^{24} f(i)$ donde $f(i)$ es el número de cubos que quedan fijos bajo una rotación de i posiciones. Esta formula es igual a $\frac{1}{24}(n^6 + 3n^4 + 12n^3 + 8n^2)$

3.18 Recursion

Sea $f(n) = \sum_{i=1}^{k} a_i f(n-i)$ entonces podemos considerar la matriz:

$$\begin{bmatrix} f(n) \\ f(n-1) \\ \vdots \\ f(n-k+1) \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & \cdots & a_{k-1} & a_k \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} f(n-1) \\ f(n-2) \\ \vdots \\ f(n-k) \end{bmatrix}$$

De aqui podemos calcular $f(n)$ con exponenciación de matrices.

$$\begin{bmatrix} f(n) \\ f(n-1) \\ \vdots \\ f(n-k+1) \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & \cdots & a_{k-1} & a_k \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}^{n-k} \begin{bmatrix} f(k) \\ f(k-1) \\ \vdots \\ f(1) \end{bmatrix}$$

3.19 Theorems

Koeing's Theorem: La cardinalidad del emparejamiento maximo de una grafica bipartita es igual al minimum vertex cover.
Hall's Theorem: Una grafica bipartita G tiene un emparejamiento que cubre todos los nodos de G si y solo si para todo subconjunto S de nodos de G, el número de vecinos de S es mayor o igual a $|S|$.

4 Game Theory

4.1 Sprague-Grundy theorem

`https://codeforces.com/blog/entry/66040` Dado un juego con pilas p_1, p_2, ..., p_n sea $g(p)$ el nimber de la pila p, entonces el nimber del juego es $g(p_1) \oplus g(p_2) \oplus ... \oplus g(p_n)$. Para calcular el nimber de una pila, se puede usar la fórmula $g(r) = mex(\{g(r_1), g(r_2), ..., g(r_k)\})$ donde $r_1, r_2, ..., r_k$ son los posibles estados a los que se puede llegar desde r y $g(r) = 0$ si r es un estado perdedor.

5 Strings

5.1 Manacher

```
1  // Number of palindromes centered at each position
2
3  vector<int> manacher_odd(string s)
4  {
5    int n = s.size();
6    s = "$" + s + "^";
7    vector<int> p(n + 2);
8    int l = 1, r = 1;
9    for (int i = 1; i <= n; i++)
10   {
11     p[i] = max(0, min(r - i, p[l + (r - i)]));
12     while (s[i - p[i]] == s[i + p[i]])
```

```
    {
      p[i]++;
    }
    if (i + p[i] > r)
    {
      l = i - p[i], r = i + p[i];
    }
  }
  return vector<int>(begin(p) + 1, end(p) - 1);
}
vector<int> manacher(string s)
{
  string t;
  for (auto c : s)
  {
    t += string("#") + c;
  }
  auto res = manacher_odd(t + "#");
  return vector<int>(begin(res) + 1, end(res) - 1);
}

// usage
// vector<int> p = manacher("abacaba");
// this will return {2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2}
// vector<int> p = manacher("abaaba");
// this will return {2, 1, 4, 1, 2, 7, 2, 1, 4, 1, 2}
```

5.2 Suffix Array O(nlog(n))

```
using i64 = long long;

struct SuffixArray
{
  int n;
  std::vector<int> sa, rk, lc;
  SuffixArray(const std::string &s)
  {
    n = s.length();
    sa.resize(n);
    lc.resize(n - 1);
    rk.resize(n);
    std::iota(sa.begin(), sa.end(), 0);
    std::sort(sa.begin(), sa.end(), [&](int a, int b)
              { return s[a] < s[b]; });
    rk[sa[0]] = 0;
    for (int i = 1; i < n; ++i)
      rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
    int k = 1;
    std::vector<int> tmp, cnt(n);
    tmp.reserve(n);
    while (rk[sa[n - 1]] < n - 1)
    {
      tmp.clear();
      for (int i = 0; i < k; ++i)
        tmp.push_back(n - k + i);
      for (auto i : sa)
        if (i >= k)
          tmp.push_back(i - k);
      std::fill(cnt.begin(), cnt.end(), 0);
      for (int i = 0; i < n; ++i)
        ++cnt[rk[i]];
      for (int i = 1; i < n; ++i)
        cnt[i] += cnt[i - 1];
      for (int i = n - 1; i >= 0; --i)
        sa[--cnt[rk[tmp[i]]]] = tmp[i];
      std::swap(rk, tmp);
      rk[sa[0]] = 0;
      for (int i = 1; i < n; ++i)
        rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i
            - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
      k *= 2;
    }
    for (int i = 0, j = 0; i < n; ++i)
    {
      if (rk[i] == 0)
      {
        j = 0;
      }
      else
      {
        for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j]
            == s[sa[rk[i] - 1] + j];)
          ++j;
        lc[rk[i] - 1] = j;
      }
    }
```

```
  }
};
```

5.3 Z Function O(n)

```
// Mayor x tal que el prefijo de s de tamano x es igual al prefijo
//del sufijo que empieza en la posicion i y tiene tamano x

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

// usage
// vector<int> z = z_function("abacaba");
// this will return {0, 0, 1, 0, 3, 0, 1}
// vector<int> z = z_function("aaaaa");
// this will return {0, 4, 3, 2, 1}
// vector<int> z = z_function("aaabaab");
// this will return {0, 2, 1, 0, 2, 1, 0}
```

5.4 KMP

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

const int N = 3e5 + 9;

// returns the longest proper prefix array of pattern p
// where lps[i]=longest proper prefix which is also suffix of p[0...i]
vector<int> build_lps(string p) {
  int sz = p.size();
  vector<int> lps;
  lps.assign(sz + 1, 0);
  int j = 0;
  lps[0] = 0;
  for(int i = 1; i < sz; i++) {
    while(j >= 0 && p[i] != p[j]) {
      if(j >= 1) j = lps[j - 1];
      else j = -1;
    }
    j++;
    lps[i] = j;
  }
  return lps;
}
vector<int>ans;
// returns matches in vector ans in 0-indexed
void kmp(vector<int> lps, string s, string p) {
  int psz = p.size(), sz = s.size();
  int j = 0;
  for(int i = 0; i < sz; i++) {
    while(j >= 0 && p[j] != s[i])
      if(j >= 1) j = lps[j - 1];
      else j = -1;
    j++;
    if(j == psz) {
      j = lps[j - 1];
      // pattern found in string s at position i-psz+1
      ans.push_back(i - psz + 1);
    }
    // after each loop we have j=longest common suffix of s[0..i] which
        is also prefix of p
  }
}

int main() {
  int i, j, k, n, m, t;
  cin >> t;
  while(t--) {
    string s, p;
    cin >> s >> p;
    vector<int>lps = build_lps(p);
```

```
    kmp(lps, s, p);
    if(ans.empty()) cout << "Not␣Found\n";
    else {
      cout << ans.size() << endl;
      for(auto x : ans) cout << x << '␣';
      cout << endl;
    }
    ans.clear();
    cout << endl;
  }
  return 0;
}
```

6 DP Optimization

6.1 Convex Hull Trick

```
const ll is_query = -(1LL << 62);
struct line
{
  ll m, b;
  mutable function<const line *()> succ;
  bool operator<(const line &rhs) const
  {
    if (rhs.b != is_query)
      return m < rhs.m;
    const line *s = succ();
    if (!s)
      return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};

struct dynamic_hull : public multiset<line>
{ // will maintain upper hull for maximum
  const ll inf = LLONG_MAX;
  bool bad(iterator y)
  {
    auto z = next(y);
    if (y == begin())
    {
      if (z == end())
        return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end())
      return y->m == x->m && y->b <= x->b;

    /* compare two lines by slope, make sure denominator is not 0 */
    ll v1 = (x->b - y->b);
    if (y->m == x->m)
      v1 = x->b > y->b ? inf : -inf;
    else
      v1 /= (y->m - x->m);
    ll v2 = (y->b - z->b);
    if (z->m == y->m)
      v2 = y->b > z->b ? inf : -inf;
    else
      v2 /= (z->m - y->m);
    return v1 >= v2;
  }
  void insert_line(ll m, ll b)
  {
    auto y = insert({m, b});
    y->succ = [=]
    { return next(y) == end() ? 0 : &*next(y); };
    if (bad(y))
    {
      erase(y);
      return;
    }
    while (next(y) != end() && bad(next(y)))
      erase(next(y));
    while (y != begin() && bad(prev(y)))
      erase(prev(y));
  }
  ll eval(ll x)
  {
    auto l = *lower_bound((line){x, is_query});
    return l.m * x + l.b;
  }
}; // gives max, for min insert_line(*-1, *-1) and eval()*-1
```

6.2 Divide and Conquer DP

```cpp
int m, n;
vector<long long> dp_before, dp_cur;

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

long long solve() {
    dp_before.assign(n,0);
    dp_cur.assign(n,0);

    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

7 Other

7.1 Binary Search

```cpp
int l = 1, r = n + 1;

  while(r - l > 1){
    int mid = l + (r - l) / 2;
    bool check = true;

    //proceso

    if(check) l = mid;
    else r = mid;
  }
```

7.2 BinSearch with doubles

```cpp
for(int i = 0; i < 200 && r - l > 1e-9; i++){
    double mid = l + (r - l) / (double)2;
    if(check(mid, obj, scores)) l = mid;
    else r = mid;
  }
```

7.3 Ternary Search

```cpp
// ternary search no funciona si existe
// a[i]==a[i+1];
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x) in [l,
        r]
}
```

7.4 LIS DP

```cpp
vector<int> v;
v.pb(a[0]);
for(int i=1;i<n;i++){
  if(a[i]>v[v.size()-1]){
    v.pb(a[i]);
  }
  else{
    int b=lower_bound(v.begin(),v.end(),a[i])-v.begin();
    v[b]=a[i];
  }
}
```

7.5 Random numbers

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
```

7.6 XOR basis

```cpp
vector<ll> basis;
void add(ll x)
{
  for (int i = 0; i < basis.size(); i++)
  {
    x = min(x, x ^ basis[i]);
  }
  if (x != 0)
  {
    basis.pb(x);
  }
}
```

7.7 Bitsets

```cpp
// count the number of set bits in an integer
#pragma GCC target("popcnt")
(int) __builtin_popcount(x);
(int) __builtin_popcountll(x);
__builtin_clz(x); // count leading zeros

// declare bitset
bitset<64> b;

// count set bits in bitser
b.count();
```

7.8 All permutations

```cpp
sort(v.begin(),v.end());
while(next_permutation(v.begin(),v.end())){
  for(auto u:v){
    cout<<u<<"␣";
  }
  cout<<endl;
}

string s="asdfassd";
sort(s.begin(),s.end());
while(next_permutation(s.begin(),s.end())){
  cout<<s<<endl;
}
```

7.9 Print doubles

```cpp
cout<<fixed<<setprecision(10);
```

8 Flows

8.1 Maximum Flow O(V*E*E)

```cpp
long long max_flow(vector<vector<int>> adj, vector<vector<long long>>
    capacity,
                   int source, int sink)
{
  int n = adj.size();
  vector<int> parent(n, -1);
  // Find a way from the source to sink on a path with non-negative
      capacities
  auto reachable = [&]() -> bool
  {
    queue<int> q;
    q.push(source);
    while (!q.empty())
    {
      int node = q.front();
      q.pop();
```

```
      for (int son : adj[node])
      {
        long long w = capacity[node][son];
        if (w <= 0 || parent[son] != -1)
          continue;
        parent[son] = node;
        q.push(son);
      }
    }
    return parent[sink] != -1;
  };

  long long flow = 0;
  // While there is a way from source to sink with non-negative
      capacities
  while (reachable())
  {
    int node = sink;
    // The minimum capacity on the path from source to sink
    long long curr_flow = LLONG_MAX;
    while (node != source)
    {
      curr_flow = min(curr_flow, capacity[parent[node]][node]);
      node = parent[node];
    }
    node = sink;
    while (node != source)
    {
      // Subtract the capacity from capacity edges
      capacity[parent[node]][node] -= curr_flow;
      // Add the current flow to flow backedges
      capacity[node][parent[node]] += curr_flow;
      node = parent[node];
    }
    flow += curr_flow;
    fill(parent.begin(), parent.end(), -1);
  }

  return flow;
}
```

```
//vector<vector<long long>> capacity(n, vector<long long>(n));
//vector<vector<int>> adj(n);
//adj[a].push_back(b);
//adj[b].push_back(a);
//capacity[a][b] += c;
```

8.2 Maximum Flow O(V*V*E)

```
const int inf = 1000000000;

int n;
vector<vector<int>> capacity, flow;
vector<int> height, excess, seen;
queue<int> excess_vertices;

void push(int u, int v) {
  int d = min(excess[u], capacity[u][v] - flow[u][v]);
  flow[u][v] += d;
  flow[v][u] -= d;
  excess[u] -= d;
  excess[v] += d;
  if (d && excess[v] == d)
    excess_vertices.push(v);
}

void relabel(int u) {
  int d = inf;
  for (int i = 0; i < n; i++) {
    if (capacity[u][i] - flow[u][i] > 0)
      d = min(d, height[i]);
  }
  if (d < inf)
    height[u] = d + 1;
}

void discharge(int u) {
  while (excess[u] > 0) {
    if (seen[u] < n) {
      int v = seen[u];
      if (capacity[u][v] - flow[u][v] > 0 && height[u] > height[v])
        push(u, v);
      else
        seen[u]++;
```

```
    } else {
      relabel(u);
      seen[u] = 0;
    }
  }
}

int max_flow(int s, int t) {
  height.assign(n, 0);
  height[s] = n;
  flow.assign(n, vector<int>(n, 0));
  excess.assign(n, 0);
  excess[s] = inf;
  for (int i = 0; i < n; i++) {
    if (i != s)
      push(s, i);
  }
  seen.assign(n, 0);

  while (!excess_vertices.empty()) {
    int u = excess_vertices.front();
    excess_vertices.pop();
    if (u != s && u != t)
      discharge(u);
  }

  int max_flow = 0;
  for (int i = 0; i < n; i++)
    max_flow += flow[i][t];
  return max_flow;
}
```

8.3 Maximum matching $O(E\sqrt{V})$

```
// maximum matching in bipartite graph
vector<int> match, dist;
vector<vector<int>> g;
int n, m, k;
bool bfs()
{
  queue<int> q;
  // The alternating path starts with unmatched nodes
  for (int node = 1; node <= n; node++)
  {
    if (!match[node])
    {
      q.push(node);
      dist[node] = 0;
    }
    else
    {
      dist[node] = INF;
    }
  }

  dist[0] = INF;

  while (!q.empty())
  {
    int node = q.front();
    q.pop();
    if (dist[node] >= dist[0])
    {
      continue;
    }
    for (int son : g[node])
    {
      // If the match of son is matched
      if (dist[match[son]] == INF)
      {
        dist[match[son]] = dist[node] + 1;
        q.push(match[son]);
      }
    }
  }
  // Returns true if an alternating path has been found
  return dist[0] != INF;
}

// Returns true if an augmenting path has been found starting from
    vertex node
bool dfs(int node)
{
  if (node == 0)
  {
    return true;
```

```
  }
  for (int son : g[node])
  {
    if (dist[match[son]] == dist[node] + 1 && dfs(match[son]))
    {
      match[node] = son;
      match[son] = node;
      return true;
    }
  }
  dist[node] = INF;
  return false;
}

int hopcroft_karp()
{
  int cnt = 0;
  // While there is an alternating path
  while (bfs())
  {
    for (int node = 1; node <= n; node++)
    {
      // If node is unmatched but we can match it using an augmenting
          path
      if (!match[node] && dfs(node))
      {
        cnt++;
      }
    }
  }
  return cnt;
}
// usage
// n numero de puntos en la izquierda
// m numero de puntos en la derecha
// las aristas se guardan en g
// los puntos estan 1 indexados
// el punto 1 de m es el punto n+1 de g
// hopcroft_karp() devuelve el tamano del maximo matching
// match contiene el match de cada punto
// si match de i es 0, entonces i no esta matcheado
//
// https://judge.yosupo.jp/submission/247277
```

8.4 Minimum Cost Maximum Flow

```
// dado un acomodo de flujos con costos
// devuelve el costo minimo para un flujo especificado

struct Edge
{
  int from, to, capacity, cost;
  Edge(int _from, int _to, int _capacity, int _cost)
  {
    from = _from;
    to = _to;
    capacity = _capacity;
    cost = _cost;
  }
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int> &d, vector<int> &p)
{
  d.assign(n, INF);
  d[v0] = 0;
  vector<bool> inq(n, false);
  queue<int> q;
  q.push(v0);
  p.assign(n, -1);

  while (!q.empty())
  {
    int u = q.front();
    q.pop();
    inq[u] = false;
    for (int v : adj[u])
    {
      if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v])
      {
        d[v] = d[u] + cost[u][v];
        p[v] = u;
        if (!inq[v])
        {
```

```
          inq[v] = true;
          q.push(v);
        }
      }
    }
  }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t)
{
  adj.assign(N, vector<int>());
  cost.assign(N, vector<int>(N, 0));
  capacity.assign(N, vector<int>(N, 0));
  for (Edge e : edges)
  {
    adj[e.from].push_back(e.to);
    adj[e.to].push_back(e.from);
    cost[e.from][e.to] = e.cost;
    cost[e.to][e.from] = -e.cost;
    capacity[e.from][e.to] = e.capacity;
  }

  int flow = 0;
  int cost = 0;
  vector<int> d, p;
  while (flow < K)
  {
    shortest_paths(N, s, d, p);
    if (d[t] == INF)
      break;

    // find max flow on that path
    int f = K - flow;
    int cur = t;
    while (cur != s)
    {
      f = min(f, capacity[p[cur]][cur]);
      cur = p[cur];
    }

    // apply flow
    flow += f;
    cost += f * d[t];
    cur = t;
    while (cur != s)
    {
      capacity[p[cur]][cur] -= f;
      capacity[cur][p[cur]] += f;
      cur = p[cur];
    }
  }

  if (flow < K)
    return -1;
  else
    return cost;
}
```

8.5 Dinic

```
// Si en el grafo todos los vertices distintos
// de s y t cumplen que solo tienen una arista
// de entrada o una de salida la y dicha arista
// tiene capacidad 1 entonces la complejidad es
// O(E sqrt(v))

// si todas las aristas tinen capacidad 1
// el algoritmo tiene complejidad O(E sqrt(E))

struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
```

```
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].
                flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].
                flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

9 Geometry

9.1 Point struct

```
struct Point{
  ll x,y;

  Point () : x(), y() {}
  Point (ll _x, ll _y) : x(_x), y(_y){}

  Point operator + (const Point &a) const {
    return Point(x+a.x,y+a.y);
  }

  Point operator - (const Point &a) const {
    return Point(x-a.x, y-a.y);
  }

  // dot product
```

```
  // positivo si el angulo entre los vectores es agudo
  // 0 si son perpendiculares
  // negativo si el angulo es obtuso
  ll operator % (const Point &a) const {
    return x*a.x+y*a.y;
  }

  // cross product
  // positivo si el segundo esta en sentido antihorario
  // 0 si el angulo es 180
  // negativo si el segundo esta en sentido horario
  ll operator * (const Point &a) const {
    return x*a.y - y * a.x;
  }
};
```

9.2 Sort points

```
// This comparator sorts the points clockwise
// starting from the first quarter

bool getQ(Point a){
  if(a.y!=0){
    if(a.y>0)return 0;
    return 1;
  }
  if(a.x>0)return 0;
  return 1;
}
bool comp(Point a, Point b){
  if(getQ(a)!=getQ(b))return getQ(a)<getQ(b);
  return a*b>0;
}
```

9.3 Shoelace

Sean los puntos $p_1, p_2, ..., p_n$ en sentido horario, el área del polígono es

$$\frac{1}{2}\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i) = \frac{1}{2}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i - x_{i+1})$$

9.4 Ray casting

Dadp un punto y un poligono que no se intersecta, el punto esta dentro del poligono si el número de intersecciones de un rayo horizontal que pasa por el punto con los lados del poligono es impar.

9.5 Pick's Theorem

El área de un polígono con vértices en puntos con coordenadas enteras es $A = i + \frac{b}{2} - 1$ donde i es el número de puntos internos al polígono y b es el número de puntos en el borde del polígono.

10 Compile

10.1 Template

```
#include <bits/stdc++.h>
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
using namespace std;
#define pb push_back
#define ll long long
#define s second
#define f first
#define MOD 1000000007
#define INF 100000000000000

void solve(){

}

int main() {
  ios_base::sync_with_stdio(false);  cin.tie(0);  cout.tie(0);
  int t;cin>>t;for(int T=0;T<t;T++)
  solve();
}
```

10.2 Compile

```
g++-13 nombre.cpp -o nombre (compilar)
./nombre (ejecutar)
```