templateGeometry:

```
int dcmp(double x) { return abs(x) < 1e-7 ?
0 : (x<0 ? -1 : 1);}

class point{
    public:
    double x, y;
    point(double x=0, double
y=0):x(x),y(y){}
    point(const point& dt){ this->x=dt.x;
this->y=dt.y; }
    point operator - (const point& dt){
return point(x-dt.x,y-dt.y); }
    bool operator == (const point& dt)
const{ return (x==dt.x && y==dt.y); }//
point equality
    point operator + (const point& dt){
return point(x+dt.x,y+dt.y); }
    double cross(point a, point b) { return
(a-*this)*(b-*this); }
    friend istream& operator>>(istream& os,
point& dt);// point input
    friend ostream& operator<<(ostream& os,
const point& dt);// point output
};
double dot(point a, point b) { return a.x *
b.x + a.y * b.y; }
double cross(point a, point b) { return a.x
* b.y - a.y * b.x; }
class triangle{
    public:
    point a, b, c;
    triangle(point a=origin, point b=origin,
point c=origin):a(a),b(b),c(c){}
    bool pointInTriangle(point p) {
        return dcmp(cross(b - a, p - a)) >=
0
            && dcmp(cross(c - b, p - b)) >=
0
            && dcmp(cross(a - c, p - c)) >=
0;
    }
    double inside(point p){
        return abs(abs(area())

-abs(triangle(a,b,p).area())

-abs(triangle(b,c,p).area())

-abs(triangle(c,a,p).area()))<=0.0001;
    }
    double area(){ return
((a.x*b.y+b.x*c.y+c.x*a.y)-(a.y*b.x+b.y*c.x+
c.y*a.x))/2; }
    bool bonac(){
        if(area()!=0) return false;
        return ((a.x-b.x)*(b.x-c.x)>=0 &&
(a.y-b.y)*(b.y-c.y)>=0);
    }
    //clockwise = negative
};
point start;
bool comppoint(point a, point b){
    if(triangle(start,a,b).area()==0) return
((start|a)<(start|b));
    return (triangle(start,a,b).area()>0);
}
class polygon{
    private:
    public:
    vector<point> p;
    polygon(ll n){ p.resize(n); }
    polygon(vector<point> p =
vector<point>(0)):p(p){}
    polygon(stack<point> s){
        p.resize(s.size());
        for(ll i=s.size()-1; i>=0; i--){
            p[i]=s.top(); s.pop();
        }
    }
    polygon(const polygon& P){ this->p=P.p;
}
    ll size(){ return p.size();}
    friend istream& operator>>(istream& os,
polygon& P);// polygon input
    friend ostream& operator<<(ostream& os,
const polygon& P);
    double area(){
        double up=0, down=0;
        ll n = p.size();
        for(ll i=0; i<n; i++){
up+=p[i].x*p[(i+1)%n].y;
down+=p[i].y*p[(i+1)%n].x; }
        return (up-down)/2;
    }
    polygon hull(bool pure = true){
        ll n = size();
        point q[n+1];
        for(ll i=0; i<n; i++) q[i] = p[i];
        point far(INT_MAX,INT_MAX);
        ll leftdown=0;
        for(ll i=0; i<n; i++){
            if(q[i].x<far.x ||
(q[i].x==far.x && q[i].y<=far.y)){
                far = q[i];
                leftdown = i;
            }
        }
        swap(q[leftdown],q[0]);
        q[n]=q[0];
        start = q[0];
```

```
        sort(q+1,q+n,comppoint);
        if(!pure){
            ll g = n;

while(triangle(q[n],q[n-1],q[g-1]).area()==0
) g--;
            for(ll i=g; i<=(g+n-1)/2; i++)
swap(q[i],q[n-1-i+g]);
        }
        stack<point> ans;
        ans.push(q[0]);
        ans.push(q[1]);
        for(ll i=2; i<=n; i++){
            while(true){
                if(ans.size()==1) break;
                point edge1 = ans.top();
ans.pop();
                point edge2 = ans.top();
                double tm =
triangle(edge2,edge1,q[i]).area();
                if((pure && tm>0)||(!pure &&
tm>=0)){
                    ans.push(edge1);
                    break;
                }
            }
            if(i<n) ans.push(q[i]);
        }
        return polygon(ans);
    }
    ll pointInConvexPolygon(point a) {  //
1=outside, 0=on edge, -1=strictly inside
        ll n = p.size();
        assert(n >= 3);

        int lo = 1, hi = n - 1;
        while(hi - lo > 1) {
            int mid = (lo + hi) / 2;
            if(dcmp(cross(p[mid] - p[0], a -
p[0])) > 0) lo = mid;
            else    hi = mid;
        }

        bool in = triangle(p[0], p[lo],
p[hi]).pointInTriangle(a);
        if(!in) return 1;

        if(dcmp(cross(p[lo] - p[lo - 1], a -
p[lo - 1])) == 0) return 0;
        if(dcmp(cross(p[hi] - p[lo], a -
p[lo])) == 0) return 0;
        if(dcmp(cross(p[hi] - p[(hi + 1) %
n], a - p[(hi + 1) % n])) == 0) return 0;
        return -1;
    }
    ll inside(point a){ // 2: boundary, 1:
inside, 0: outside
        ll n = size();
        if(a==p[0]) return 2;
        if(!(triangle(p[0],p[1],a).area()>=0
&& triangle(p[0],p[n-1],a).area()<=0))
return 0;
        ll l = 1, r = n-1;
        while(l<r){
            ll m1 = (l+r)/2;
            ll m2 = m1+1;
            double f1 =
triangle(p[0],p[m1],a).area();
            double f2 =
triangle(p[0],p[m2],a).area();
            if(f1>=0 && f2<0){
                l = m1;
                r = m1;
            }
            else if(f1<0) r = m1;
            else l = m2;
        }
        if(l<n-1) cout<< a<< " "<< l<< " "<<
p[l]<< p[l+1]<< p[0];
        else cout<< a<< " "<< l<< " "<<
p[l]<< p[0];
        if(l<n-1 &&
!triangle(p[0],p[l],p[l+1]).inside(a))
return 0;
        if(l==n-1){
            return
(triangle(p[0],a,p[n-1]).bonac())? 2:0;
        }
        if(l==1 &&
triangle(p[0],a,p[1]).bonac()) return 2;
        return
(triangle(p[l],a,p[l+1]).bonac())? 2:1;
    }
};
void reorder_polygon(vector<point> & P){
    size_t pos = 0;
    for(size_t i = 1; i < P.size(); i++){
        if(P[i].y < P[pos].y || (P[i].y ==
P[pos].y && P[i].x < P[pos].x))
            pos = i;
    }
    rotate(P.begin(), P.begin() + pos,
P.end());
}
struct polarComp {
    point O, dir;
    polarComp(point O = point(0, 0), point
dir = point(1, 0))
        : O(O), dir(dir) {}
    bool half(point p) {
        return dcmp(cross(dir, p)) < 0 ||
(dcmp(cross(dir, p)) == 0 && dcmp(dot(dir,
p)) > 0);
    }
```

```
    bool operator()(point p, point q) {
        return make_tuple(half(p-O), 0) <
make_tuple(half(q-O), cross(p-O, q-O));
    }
};
polygon minkowski_sum(polygon A, polygon B){
    ll n = A.size(), m = B.size();
    rotate(A.p.begin(),
min_element(A.p.begin(), A.p.end()),
A.p.end());
    rotate(B.p.begin(),
min_element(B.p.begin(), B.p.end()),
B.p.end());

    A.p.push_back(A.p[0]);
B.p.push_back(B.p[0]);
    for(ll i = 0; i < n; i++) A.p[i] =
A.p[i+1] - A.p[i];
    for(ll i = 0; i < m; i++) B.p[i] =
B.p[i+1] - B.p[i];

    polygon C(n+m+1);
    C.p[0] = A.p.back() + B.p.back();
    merge(A.p.begin(), A.p.end()-1,
B.p.begin(), B.p.end()-1, C.p.begin()+1,
polarComp(point(0, 0), point(0, -1)));
    for(ll i = 1; i < C.p.size(); i++)
C.p[i] = C.p[i] + C.p[i-1];
    C.p.pop_back();
    return C;
}
```

Miller Robin Primality Test:

```
using u64 = uint64_t;
using u128 = __uint128_t;

bool check_composite(u64 n, u64 a, u64 d,
int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n, int iter=5) { //
returns true if n is probably prime, else
returns false.
    if (n < 4)
        return n == 2 || n == 3;

    int s = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (check_composite(n, a, d, s))
            return false;
    }
    return true;
}
```

Convex Hull Trick Linear:

```
/**
Linear Convex Hull Trick
Requirement:
    Minimum:
        M increasing, x decreasing,
useless(s-1, s-2, s-3)
        M decreasing, x increasing,
useless(s-3, s-2, s-1)
    Maximum:
        M increasing, x increasing,
useless(s-3, s-2, s-1)
        M decreasing, x decreasing,
useless(s-1, s-2, s-3)

If queries are in arbitrary order, use
query2 O(logn) per query.

Source: Rezwan, Anachor (query2)
**/

struct CHT {
    vector<LL> M;
    vector<LL> C;
    int ptr = 0;

    ///Use double comp if M,C is LL range
    bool useless(int l1, int l2, int l3) {
        return (C[l3]-C[l1])*(M[l1]-M[l2])
<= (C[l2]-C[l1])*(M[l1]-M[l3]);
    }

    LL f(int id, LL x) {
        return M[id]*x+C[id];
    }

    void add(LL m, LL c) {
        M.push_back(m);
        C.push_back(c);
        int s = M.size();
```

```
        while (s >= 3 && useless(s-3, s-2,
s-1)) {
            M.erase(M.end()-2);
            C.erase(C.end()-2);
            s--;
        }
    }

    LL query(LL x) {
        if (ptr >= M.size()) ptr =
M.size()-1;
        while (ptr < M.size()-1 && f(ptr, x)
> f(ptr+1, x)) ptr++; /// change > to < for
maximum
        return f(ptr, x);
    }

    LL query2(LL x) {
        int lo=0, hi=M.size()-1;
        while(lo<hi) {
            int mid = (lo+hi)/2;
            if (f(mid, x) > f(mid+1, x))
lo = mid+1; /// change > to < for maximum
            else
hi = mid;
        }
        return f(lo, x);
    }
};
```

Convex Hull Trick Dynamic:

```
///convex hull for maximizing
///in case of minimization, just
insert(-m,-c) ///and negate the result for
query

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

struct CHT: multiset<Line, less<>> {
      // (for doubles, use inf = 1/.0,
div(a,b) = a/b)
      static const ll inf = LLONG_MAX;
      ll div(ll a, ll b) { // floored
division
            return a / b - ((a ^ b) < 0 &&
a % b); }
      bool isect(iterator x, iterator y) {
            if (y == end()) return x->p =
inf, 0;
            if (x->k == y->k) x->p = x->m >
y->m ? inf : -inf;
            else x->p = div(y->m - x->m,
x->k - y->k);
            return x->p >= y->p;
      }
      void add(ll k, ll m) {
            auto z = insert({k, m, 0}), y =
z++, x = y;
            while (isect(y, z)) z =
erase(z);
            if (x != begin() && isect(--x,
y)) isect(x, y = erase(y));
            while ((y = x) != begin() &&
(--x)->p >= y->p)
                  isect(x, erase(y));
      }
      ll query(ll x) {
            assert(!empty());
            auto l = *lower_bound(x);
            return l.k * x + l.m;
      }
} ch;
```

2SAT

```
int n;
vector<vector<int>> adj, adj_t;
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;

void dfs1(int v) {
    used[v] = true;
    for (int u : adj[v]) {
        if (!used[u])
            dfs1(u);
    }
    order.push_back(v);
}
void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : adj_t[v]) {
        if (comp[u] == -1)
            dfs2(u, cl);
    }
}
bool solve_2SAT() {
    order.clear();
    used.assign(n, false);
    for (int i = 0; i < n; ++i) {
        if (!used[i])
            dfs1(i);
    }

    comp.assign(n, -1);
    for (int i = 0, j = 0; i < n; ++i) {
        int v = order[n - i - 1];
        if (comp[v] == -1)
```

```
            dfs2(v, j++);
    }

    assignment.assign(n / 2, false);
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        assignment[i / 2] = comp[i] > comp[i
+ 1];
    }
    return true;
}
void add_disjunction(int a, bool na, int b,
bool nb) {
    // na and nb signify whether a and b are
to be negated
    a = 2*a ^ na;
    b = 2*b ^ nb;
    int neg_a = a ^ 1;
    int neg_b = b ^ 1;
    adj[neg_a].push_back(b);
    adj[neg_b].push_back(a);
    adj_t[b].push_back(neg_a);
    adj_t[a].push_back(neg_b);
}
```

Articulation Points:

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list
of graph

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

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    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]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

Bridges:

```
void IS_BRIDGE(int v,int to); // some
function to process the found bridge
int n; // number of nodes
vector<vector<int>> adj; // adjacency list
of graph

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

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

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

Maxflow Dinic:

```
O(v2e), unit cap: O(e*rt(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)
                    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])
                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;
    }
};
```

Shortest cycle

```
int shortest_cycle(int n){
    int ans = INT_MAX;
    for (int i = 0; i < n; i++) {
        vector<int> dist(n, (int)(1e9));
        vector<int> par(n, -1);
        dist[i] = 0;
        queue<int> q;
        q.push(i);
        while (!q.empty()) {
            int x = q.front();
            q.pop();
            for (int child : gr[x]) {
                if(dist[child] ==
(int)(1e9)) {
dist[child] = 1 + dist[x];
par[child] = x;
q.push(child);
                }
else if (par[x] != child and par[child] !=
x)
                    ans = min(ans, dist[x] +
dist[child] + 1);
            }
        }
    }
    if (ans == INT_MAX)
        return -1;

    else
```

```
        return ans;
}
```

Hopcroft

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

struct HopcroftKarp {
  static const int inf = 1e9;
  int n;
  vector<int> l, r, d;
  vector<vector<int>> g;
  HopcroftKarp(int _n, int _m) {
    n = _n;
    int p = _n + _m + 1;
    g.resize(p);
    l.resize(p, 0);
    r.resize(p, 0);
    d.resize(p, 0);
  }
  void add_edge(int u, int v) {
    g[u].push_back(v + n); //right id is
increased by n, so is l[u]
  }
  bool bfs() {
    queue<int> q;
    for (int u = 1; u <= n; u++) {
      if (!l[u]) d[u] = 0, q.push(u);
      else d[u] = inf;
    }
    d[0] = inf;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto v : g[u]) {
        if (d[r[v]] == inf) {
          d[r[v]] = d[u] + 1;
          q.push(r[v]);
        }
      }
    }
    return d[0] != inf;
  }
  bool dfs(int u) {
    if (!u) return true;
    for (auto v : g[u]) {
      if(d[r[v]] == d[u] + 1 && dfs(r[v])) {
        l[u] = v;
        r[v] = u;
        return true;
      }
    }
    d[u] = inf;
    return false;
  }
  int maximum_matching() {
    int ans = 0;
    while (bfs()) {
      for(int u = 1; u <= n; u++) if (!l[u]
&& dfs(u)) ans++;
    }
    return ans;
  }
};
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m, q;
  cin >> n >> m >> q;
  HopcroftKarp M(n, m);
  while (q--) {
    int u, v;
    cin >> u >> v;
    M.add_edge(u, v);
  }
  cout << M.maximum_matching() << '\n';
  return 0;
}
```

Eulerian Tour:

```
int n, m; // Runs in O(E)
vector<vector<pair<int, int>>> g;
vector<int> path;
vector<bool> seen;
void dfs(int node) {
    while (!g[node].empty()) {
        auto [son, idx] = g[node].back();
        g[node].pop_back();
        if (seen[idx]) { continue; }
        seen[idx] = true;
        dfs(son);
    }
    path.push_back(node);
}
int main() {
    cin >> n >> m;
    vector<int> degree(n, 0);
    g.resize(n);
    degree.resize(n);
    seen.resize(m);
    for (int i = 0; i < m; i++) {
        int x, y;
        cin >> x >> y;
        x--, y--;
        g[x].emplace_back({y, i});
        g[y].emplace_back({x, i});
        degree[x]++;
        degree[y]++;
    }
    for (int node = 0; node < n; node++) {
        if (degree[node] % 2) {
            cout << "IMPOSSIBLE" << endl;
            return 0;
        }
    }
```

```
    dfs(0);
    if (path.size() != m + 1) {
        cout << "IMPOSSIBLE";
    } else {
        for (int node : path) { cout << node
+ 1 << ' '; }
    }
    cout << endl;
}
```

Fast Fourier Transform:

```
struct CD {
    double x, y;
    CD(double x=0, double y=0) :x(x), y(y)
{}
    CD operator+(const CD& o) { return
{x+o.x, y+o.y};}
    CD operator-(const CD& o) { return
{x-o.x, y-o.y};}
    CD operator*(const CD& o) { return
{x*o.x-y*o.y, x*o.y+o.x*y};}
    void operator /= (double d) { x/=d;
y/=d;}
    double real() {return x;}
    double imag() {return y;}
};
CD conj(const CD &c) {return CD(c.x, -c.y);}

typedef long long LL;
const double PI = acos(-1.0L);

namespace FFT {
    int N;
    vector<int> perm;
    vector<CD> wp[2];

    void precalculate(int n) {
        assert((n & (n-1)) == 0);
        N = n;
        perm = vector<int> (N, 0);
        for (int k=1; k<N; k<<=1) {
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }
        }
        wp[0] = wp[1] = vector<CD>(N);
        for (int i=0; i<N; i++) {
            wp[0][i] = CD( cos(2*PI*i/N),
sin(2*PI*i/N) );
            wp[1][i] = CD( cos(2*PI*i/N),
-sin(2*PI*i/N) );
        }
    }

    void fft(vector<CD> &v, bool invert =
false) {
        if (v.size() != perm.size())
precalculate(v.size());
        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);

        for (int len = 2; len <= N; len *=
2) {
            for (int i=0, d = N/len; i<N;
i+=len) {
                for (int j=0, idx=0;
j<len/2; j++, idx += d) {
                    CD x = v[i+j];
                    CD y =
wp[invert][idx]*v[i+j+len/2];
                    v[i+j] = x+y;
                    v[i+j+len/2] = x-y;
                }
            }
        }
        if (invert) {
            for (int i=0; i<N; i++) v[i]/=N;
        }
    }

     void pairfft(vector<CD> &a, vector<CD>
&b, bool invert = false) {
        int N = a.size();
        vector<CD> p(N);
        for (int i=0; i<N; i++) p[i] = a[i]
+ b[i] * CD(0, 1);
        fft(p, invert);
        p.push_back(p[0]);

        for (int i=0; i<N; i++) {
            if (invert) {
                a[i] = CD(p[i].real(), 0);
                b[i] = CD(p[i].imag(), 0);
            }
            else {
                a[i] =
(p[i]+conj(p[N-i]))*CD(0.5, 0);
                b[i] =
(p[i]-conj(p[N-i]))*CD(0, -0.5);
            }
        }
    }

    vector<LL> multiply(const vector<LL> &a,
const vector<LL> &b) {
        int n = 1;
        while (n < a.size()+ b.size())
n<<=1;
        vector<CD> fa(a.begin(), a.end()),
fb(b.begin(), b.end());
        fa.resize(n); fb.resize(n);
//        fft(fa); fft(fb);
```

```
        pairfft(fa, fb);
        for (int i=0; i<n; i++) fa[i] =
fa[i] * fb[i];
        fft(fa, true);
        vector<LL> ans(n);
        for (int i=0; i<n; i++)     ans[i] =
round(fa[i].real());
        return ans;
    }

    const int M = 1e9+7, B = sqrt(M)+1;
    vector<LL> anyMod(const vector<LL> &a,
const vector<LL> &b) {
        int n = 1;
        while (n < a.size()+ b.size())
n<<=1;
        vector<CD> al(n), ar(n), bl(n),
br(n);

        for (int i=0; i<a.size(); i++)
al[i] = a[i]%M/B, ar[i] = a[i]%M%B;
        for (int i=0; i<b.size(); i++)
bl[i] = b[i]%M/B, br[i] = b[i]%M%B;

        pairfft(al, ar); pairfft(bl, br);
//        fft(al); fft(ar); fft(bl);
fft(br);

        for (int i=0; i<n; i++) {
            CD ll = (al[i] * bl[i]), lr =
(al[i] * br[i]);
            CD rl = (ar[i] * bl[i]), rr =
(ar[i] * br[i]);
            al[i] = ll; ar[i] = lr;
            bl[i] = rl; br[i] = rr;
        }
        pairfft(al, ar, true); pairfft(bl,
br, true);
//        fft(al, true); fft(ar, true);
fft(bl, true); fft(br, true);
        vector<LL> ans(n);
        for (int i=0; i<n; i++) {
            LL right = round(br[i].real()),
left = round(al[i].real());;
            LL mid =
round(round(bl[i].real()) +
round(ar[i].real()));
            ans[i] = ((left%M)*B*B +
(mid%M)*B + right)%M;
        }
        return ans;
    }
}
//usage: vector<LL> a(n), b(m);
// vector<LL> ans = FFT::anyMod(a, b);
```

NTT (Number Theoretic Transform):

```
namespace NTT {
    vector<int> perm, wp[2];
    const int mod = 998244353, G = 3;
///G is the primitive root of M
    int root, inv, N, invN;

    int power(int a, int p) {
        int ans = 1;
        while (p) {
            if (p & 1) ans =
(1LL*ans*a)%mod;
            a = (1LL*a*a)%mod;
            p >>= 1;
        }
        return ans;
    }

    void precalculate(int n) {
        assert( (n&(n-1)) == 0 &&
(mod-1)%n==0);
        N = n;
        invN = power(N, mod-2);
        perm = wp[0] = wp[1] =
vector<int>(N);

        perm[0] = 0;
        for (int k=1; k<N; k<<=1)
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }

        root = power(G, (mod-1)/N);
        inv = power(root, mod-2);
        wp[0][0]=wp[1][0]=1;

        for (int i=1; i<N; i++) {
            wp[0][i] =
(wp[0][i-1]*1LL*root)%mod;
            wp[1][i] =
(wp[1][i-1]*1LL*inv)%mod;
        }
    }

    void fft(vector<int> &v, bool invert =
false) {
        if (v.size() != perm.size())
precalculate(v.size());
        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);

        for (int len = 2; len <= N; len *=
2) {
            for (int i=0, d = N/len; i<N;
i+=len) {
                for (int j=0, idx=0;
j<len/2; j++, idx += d) {
```

```
                    int x = v[i+j];
                    int y =
(wp[invert][idx]*1LL*v[i+j+len/2])%mod;
                    v[i+j] = (x+y>=mod ?
x+y-mod : x+y);
                    v[i+j+len/2] = (x-y>=0 ?
x-y : x-y+mod);
                }
            }
        }
        if (invert) {
            for (int &x : v) x =
(x*1LL*invN)%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);

        fft(a);
        fft(b);
        for (int i=0; i<n; i++) a[i] = (a[i]
* 1LL * b[i])%mod;
        fft(a, true);
        return a;
    }
};
```

Z function trivial:

```
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);gauss
    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;
}
```

KMP:

```
vll prefix_function(string s) {
    ll n = s.size();
    vll p(n,0);
    for(ll i = 1; i < n; i++) {
        ll j = p[i-1];
        while(j > 0 && s[i] != s[j]) j =
p[j-1];
        if(s[i] == s[j]) j++;
        p[i] = j;
    }
    return p;
}

ll kmp(string s, string t) {
    ll n = s.size(), m = t.size(), ans = 0,
j = 0;
    vll p = prefix_function(t);
    for (int i = 0; i < n; i++) {
        while(j > 0 && s[i] != t[j]) j =
p[j-1];
        if(s[i] == t[j]) j++;
        if(j==m){ j = p[j-1]; ans++; }
    }
    return ans;
}
```

Prefix function automaton:

If the current prefix function value is i (i.e., you've matched the first i characters of string s), and you now append the character 'a' + c, then:

aut[i][c] gives the new value of the prefix function for the updated string.

```
void compute_automaton(string s,
vector<vector<int>>& aut) {
    s += '#';
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(26));
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c != s[i])
                aut[i][c] = aut[pi[i-1]][c];
            else
                aut[i][c] = i + ('a' + c ==
s[i]);
        }
    }
}
```

Trie Usaco:

```
const int WMAX = 1e5 + 10;   //sum of length
of all strings
int trie_s[WMAX][26];
int node_count;
bool stop[WMAX];
void insert(string word) {
       int node = 0;
       for (char c : word) {
```

```
            if (trie_s[node][c - 'a'] == 0)
{ trie_s[node][c - 'a'] = ++node_count; }
            node = trie_s[node][c - 'a'];
        }
        stop[node] = true;
}
```

Suffix Array:

```
vvll c;
// string S;
// vll SA;
vll sort_cyclic_shifts(const string &s){
    ll n = s.size();
    const ll alphabet = 256;
    vll p(n), cnt(alphabet, 0);
    c.clear();
    c.emplace_back();
    c[0].resize(n);
    for (ll i=0; i<n; i++)
cnt[s[i]]++;
    for (ll i=1; i<alphabet; i++)  cnt[i] +=
cnt[i-1];
    for (ll i=0; i<n; i++)
p[--cnt[s[i]]] = i;
    c[0][p[0]] = 0;
    ll classes = 1;
    for (ll i=1; i<n; i++) {
        if (s[p[i]] != s[p[i-1]])
classes++;
        c[0][p[i]] = classes - 1;
    }
    vll pn(n), cn(n);
    cnt.resize(n);

    for (ll h=0; (1<<h) < n; h++) {
        for (ll i=0; i<n; i++) {
            pn[i] = p[i] - (1<<h);
            if (pn[i] < 0)  pn[i] += n;
        }
        fill(cnt.begin(), cnt.end(), 0);

        /// radix sort
        for (ll i = 0; i < n; i++)
cnt[c[h][pn[i]]]++;
        for (ll i = 1; i < classes; i++)
cnt[i] += cnt[i-1];
        for (ll i = n-1; i >= 0; i--)
p[--cnt[c[h][pn[i]]]] = pn[i];

        cn[p[0]] = 0;
        classes = 1;

        for (ll i=1; i<n; i++) {
            pll cur = {c[h][p[i]],
c[h][(p[i] + (1<<h))%n]};
            pll prev = {c[h][p[i-1]],
c[h][(p[i-1] + (1<<h))%n]};
            if (cur != prev)    ++classes;
            cn[p[i]] = classes - 1;
        }
        c.push_back(cn);
    }
    return p;
}

vll suffix_array_construction(string s){
    s += "!";
    vll sorted_shifts =
sort_cyclic_shifts(s);

sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}

/// LCP between the ith and jth (i != j)
suffix of the STRING
ll suffixLCP(ll i, ll j){
    assert(i != j);
    ll log_n = c.size()-1;

    ll ans = 0;
    for (ll k = log_n; k >= 0; k--) {
        // cout<< "= "<< k<< " "<< i<< " "<<
j<< " "<< c[k][i]<< " "<< c[k][j]<< endl;
        if (c[k][i] == c[k][j]) {
            ans += (ll)(1 << k);
            i += 1 << k;
            j += 1 << k;
        }
    }
    return ans;
}

vll lcp_construction(const string &s, const
vll &sa){
    ll n = s.size();
    vll rank(n, 0);
    vll lcp(n-1, 0);
     for (ll i=0; i<n; i++) rank[sa[i]] = i;
     for (ll i=0, k=0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        ll j = sa[rank[i] + 1];
        while (i + k < n && j + k < n &&
s[i+k] == s[j+k])  k++;
        lcp[rank[i]] = k;
        if (k)  k--;
    }
    return lcp;
}
```

MO + SQRT Decomposition:

```
void remove(idx);  // TODO: remove value at
idx from data structure
void add(idx);     // TODO: add value at idx
from data structure
int get_answer();  // TODO: extract the
current answer of the data structure

int block_size;
struct Query {
    int l, r, idx;
    bool operator<(Query other) const{
        return make_pair(l / block_size, r)
<
               make_pair(other.l /
block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query>
queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    // TODO: initialize data structure
    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always
reflect the range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);}
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);}
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;}
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;}
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

Mint (for matrix):

```
#pragma GCC optimize("03")
#pragma GCC optimize("unroll-loops")
struct mint {
  ll x; // typedef long long ll;
  mint(ll x=0):x((x%mod+mod)%mod){}
  mint operator-() const { return mint(-x);}
  mint& operator+=(const mint a) {
    if ((x += a.x) >= mod) x -= mod;
    return *this;
  }
  mint& operator-=(const mint a) {
    if ((x += mod-a.x) >= mod) x -= mod;
    return *this;
  }
  mint& operator*=(const mint a) { (x *=
a.x) %= mod; return *this;}
  mint operator*(const mint a) const {
return mint(*this) *= a;}
  mint pow(ll t) const {
    if (!t) return 1;
    mint a = pow(t>>1);
    a *= a;
    if (t&1) a *= *this;
    return a;
  }
};
ostream& operator<<(ostream& os, const mint&
a) { return os << a.x;}
```

Matrix:

```
struct Matrix {
  int h, w;
  vector<vector<T>> d;
  Matrix() {}
  Matrix(int h, int w, T val=0): h(h), w(w),
d(h, vector<T>(w,val)) {}
  Matrix& unit() {
    assert(h == w);
    F0R(i,h) d[i][i] = 1;
    return *this;
  }
  const vector<T>& operator[](int i) const {
return d[i];}
  vector<T>& operator[](int i) { return
d[i];}
  Matrix operator*(const Matrix& a) const {
    assert(w == a.h);
    Matrix r(h, a.w);
    F0R(i,h)F0R(k,w)F0R(j,a.w) {
      r[i][j] += d[i][k]*a[k][j];
    }
    return r;
  }
  Matrix pow(long long t) const {
    assert(h == w);
    if (!t) return Matrix(h,h).unit();
    if (t == 1) return *this;
    Matrix r = pow(t>>1);
    r = r*r;
    if (t&1) r = r*(*this);
    return r;
  }};
```

Hungarian:
Given an n*n matrix A, the task is to find a permutation p of length n such that the value sum(A[i][p[i]) is minimized.
The resulting complexity is O(n^3).

```
vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<int> minv (m+1, INF);
    vector<bool> used (m+1, false);
    do {
        used[j0] = true;
        int i0 = p[j0],  delta = INF,  j1;
        for (int j=1; j<=m; ++j)
            if (!used[j]) {
                int cur = A[i0][j]-u[i0]-v[j];
                if (cur < minv[j])
                    minv[j] = cur, way[j] = j0;
                if (minv[j] < delta)
                    delta = minv[j],  j1 = j;
            }
        for (int j=0; j<=m; ++j)
            if (used[j])
                u[p[j]] += delta,v[j] -= delta;
            else
                minv[j] -= delta;
        j0 = j1;
    } while (p[j0] != 0);
    do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    } while (j0);
}
vector<int> ans (n+1);
for (int j=1; j<=m; ++j)
    ans[p[j]] = j;
```

Centroid Decomposition:

```
 vector<bool> used(n);
vector<int> sz(n);
auto cd = [&] (auto cd, int v) -> void {
    auto getCentroid = [&] () {
        auto dfs = [&](auto f, int v, int p=-1) -> int {
            sz[v] = 1;
            for (int u : adj[v]) {
            if (u == p || used[u]) continue;
                sz[v] += f(f,u,v);
            }
            return sz[v];
        };
        int tot = dfs(dfs,v), c = -1;
        auto dfs2 = [&](auto f, int v, int p=-1) -> void {
            bool ok = (tot-sz[v])*2 <= tot;
            for (int u : adj[v]) {
                if (u == p || used[u]) continue;
                f(f,u,v);
                if (sz[u]*2 > tot) ok = false;
            }
            if (ok) c = v;        };
        dfs2(dfs2,v);
        return c;    };

    int c = getCentroid();
    used[c] = true;
    // process centroid
    // modify
    for (int u : adj[c]) {
        if (used[u]) continue;
        vector<pair<int,int>> ps;
        auto dfs = [&](auto f, int v, int p=-1, int dep=1) -> void {
            // modify
            for (int u : adj[v]) {
                if (u == p || used[u]) continue;
                f(f,u,v,dep+1);
            }
            // modify
        };
        dfs(dfs,u);
    }
    for (int u : adj[c]) {
        if (used[u]) continue;
        cd(cd,u);
    }
};
```

HLD:

```
const int N = 2e5 + 5, D = 19, S = (1 << D);
int n, q, v[N];
vector<int> adj[N];
int sz[N], p[N], dep[N];
int st[S], id[N], tp[N];
void update(int idx, int val) {
       st[idx += n] = val;
       for (idx /= 2; idx; idx /= 2) st[idx] = max(st[2 * idx], st[2 * idx + 1]);
}
int query(int lo, int hi) {
       int ra = 0, rb = 0;
       for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi /= 2) {
```

```
            if (lo & 1) ra = max(ra,
st[lo++]);
            if (hi & 1) rb = max(rb,
st[--hi]);
      }
      return max(ra, rb);
}
int dfs_sz(int cur, int par) {
      sz[cur] = 1;
      p[cur] = par;
      for (int chi : adj[cur]) {
            if (chi == par) continue;
            dep[chi] = dep[cur] + 1;
            p[chi] = cur;
            sz[cur] += dfs_sz(chi, cur);
      }
      return sz[cur];
}
int ct = 1;
void dfs_hld(int cur, int par, int top) {
      id[cur] = ct++, tp[cur] = top;
      update(id[cur], v[cur]);
      int h_chi = -1, h_sz = -1;
      for (int chi : adj[cur]) {
            if (chi == par) continue;
            if (sz[chi] > h_sz) {
                  h_sz = sz[chi];
                  h_chi = chi;
            }
      }
      if (h_chi == -1) return;
      dfs_hld(h_chi, cur, top);
      for (int chi : adj[cur]) {
            if (chi == par || chi == h_chi)
continue;
            dfs_hld(chi, cur, chi);
      }
}
```

```
int path(int x, int y) {
      int ret = 0;
      while (tp[x] != tp[y]) {
            if (dep[tp[x]] < dep[tp[y]])
swap(x, y);
            ret = max(ret, query(id[tp[x]],
id[x]));
            x = p[tp[x]];
      }
      if (dep[x] > dep[y]) swap(x, y);
      ret = max(ret, query(id[x], id[y]));
      return ret;
}
dfs_sz(1, 1);
dfs_hld(1, 1, 1);
update(id[s], v[s]);
res = path(a, b);
```

Auxiliary Tree:

```
vector<vector<int>> to2(n);
rep(ci,n) { // ci = color id
    vector<int>& vs = cvs[ci];
// cvs = nodes of that col
    if (vs.size() == 0) continue;
    sort(vs.begin(), vs.end(),
         [&](int a, int b) { return in[a] <
in[b];});
    int m = vs.size();
    rep(i,m-1) {
      vs.push_back(lca(vs[i],vs[i+1]));
    }
    sort(vs.begin(), vs.end(),
         [&](int a, int b) { return in[a] <
in[b];});
    vs.erase(unique(vs.begin(), vs.end()),
vs.end());
    {
      vector<int> st;
      for (int v : vs) {
        while (st.size()) {
          int p = st.back();
          if (in[p] < in[v] && in[v] <
out[p]) break;
          st.pop_back();
        }
        if (st.size())
to2[st.back()].push_back(v);
        st.push_back(v);
      }
    }
 // process aux tree
    for (int v : vs) to2[v] = vector<int>();
  }
```

Euler Tour Tree:

```
vector<int> et, first_in(n), ein(n-1),
eout(n-1);
auto dfs = [&] (auto dfs, int u, int p=-1)
-> void {
    first_in[u] = et.size();
    et.push_back(u);
    trav(v, adj[u]) if(v.to != p) {
        ein[v.id] =  et.size()-1;
        dfs(dfs, v.to, u);
        eout[v.id] = et.size()-1;
        et.push_back(u);
    }
};
```

Mobius:

```
void Mobius(){
    mu[1] = 1;
    for (ll i = 1; i < MAX; i++) if (mu[i])
            for (ll j = i + i; j < MAX; j +=
i)
```

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

Phi:

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

Chinese Remainder Theorem

```
ll egcd(ll a, ll b, ll &x, ll &y){
    if(b==0){
        x = 1; y = 0;
        return a;
    }
    ll x1, y1;
    ll d = egcd(b,a%b,x1,y1);
    x = y1; y = x1 - y1*(a/b);
    return d;
}

pll crt(pll a, pll b){
    ll s, t;
    ll g = egcd(a.fi,b.fi,s,t);
    if(a.se%g!=b.se%g) return mp(-1,-1);
    ll m = a.fi*b.fi;
    ll ss = ((s%b.fi*b.se%b.fi)%b.fi)*a.fi;
    ll tt = ((t%a.fi*a.se%a.fi)%a.fi)*b.fi;
    ll ans = ((ss+tt)%m + m)%m;
    return mp(m/g,(ans/g)%(m/g));
}
```

Catalan Number

```
//Bracket Sequence
#define mod 1000000007
int factorial[2000001];
void init(){
    int n = 2000000;
    factorial[0] = 1;
    for(int i = 1 ; i <= n ; i++){
        factorial[i] = (factorial[i - 1] *
i) % mod;
    }
}
int gcdExtended(int a, int b, int *x, int
*y) {
         if (a == 0) {
             *x = 0, *y = 1;
             return b;
         }
         int x1, y1;
         int gcd = gcdExtended(b%a, a, &x1,
&y1);
         *x = y1 - (b/a) * x1;
         *y = x1;
         return gcd;
     }
 int modInverse(int a, int m) {
         int x, y;
         int g = gcdExtended(a, m, &x, &y);
        int res = (x%m + m) % m;
         return res;
     }
signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n;
    cin >> n;
    init();
    if(n % 2 != 0){
        cout << "0\n";
    }
    else{
        n /= 2;
        int temp = factorial[n];
        temp = (temp * temp) % mod;
        temp = (temp * (n + 1)) % mod;
        temp = modInverse(temp, mod);
        int ans = (factorial[2 * n]*temp)%
mod;
        cout << ans << "\n";
    }
    return 0;
}

//Bracket sequence with prefix
const int MOD=1000000007;
long long int inverse(long long int i){
    if(i==1) return 1;
    return (MOD -
((MOD/i)*inverse(MOD%i))%MOD+MOD)%MOD;
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    ll n;
    cin>>n;
    if(n%2==1){
        cout<<0;
        return 0;
    }
    n/=2;
    string s;
    cin>>s;
    ll k=0,o=0;
    for(int i=0;i<s.size();i++){
        if(s[i]=='(') {
            k++;
```

```
            o++;
        }
        else k--;
        if(k<0){
            cout<<0;
            return 0;
        }
    }
    n-=o;
    if(k<0 || n<0 || 2*n+k<n){
        cout<<0;
        return 0;
    }
    ll fact[2*n+k+1];
    fact[0]=1;
    for(int i=1;i<=2*n+k;i++){
        fact[i]=(fact[i-1]*i)%MOD;
    }
    ll ans=
(fact[2*n+k]*inverse(fact[n]))%MOD;
    ans*=inverse(fact[n+k]);
    ans%=MOD;
    ans*=((k+1)*inverse(n+k+1))%MOD;
    ans%=MOD;
    cout<<ans;
}
```

Eulerian Tour:

```
int n, m; // Runs in O(E)
vector<vector<pair<int, int>>> g;
vector<int> path;
vector<bool> seen;
void dfs(int node) {
    while (!g[node].empty()) {
        auto [son, idx] = g[node].back();
        g[node].pop_back();
        if (seen[idx]) { continue; }
        seen[idx] = true;
        dfs(son);
    }
    path.push_back(node);
}
int main() {
    cin >> n >> m;
    vector<int> degree(n, 0);
    g.resize(n);
    degree.resize(n);
    seen.resize(m);
    for (int i = 0; i < m; i++) {
        int x, y;
        cin >> x >> y;
        x--, y--;
        g[x].emplace_back({y, i});
        g[y].emplace_back({x, i});
        degree[x]++;
        degree[y]++;
    }
    for (int node = 0; node < n; node++) {
        if (degree[node] % 2) {
            cout << "IMPOSSIBLE" << endl;
            return 0;
        }
    }
    dfs(0);
    if (path.size() != m + 1) {
        cout << "IMPOSSIBLE";
    } else {
        for (int node : path) { cout << node
+ 1 << ' '; }
    }
    cout << endl;
}
```

Sparse Table:

```
class SparseTable {
    public:
    int ** st;
    int K;
    int n;
    SparseTable(vector<int>a) {
        n = a.size();
        K = __lg(n) + 1;
        st = new int*[K];
        for(int i = 0; i < K; i++) st[i] =
new int[n];
        for(int i = 0; i < n; i++) {
            st[0][i] = a[i];
        }
        for (int i = 1; i <= K; i++) {
            for (int j = 0; j + (1 << i) <=
n; j++) {
                st[i][j] = min(st[i - 1][j],
st[i - 1][j + (1 << (i - 1))]);
            }
        }
    }
    ~SparseTable() {
        for(int i = 0; i < K; i++) delete[]
st[i];
        delete[] st;
    }
    int get(int L, int R) {
        int i = __lg(R - L + 1);
        int minimum = min(st[i][L], st[i][R
- (1 << i) + 1]);
    }
};
```

Hash:

```
class hashTable {
    public:
    int n, limit;
    string s;
    vector<vector<ll>>pref, suff;
```

```
    vector<ll> primes;
    vector<ll> m;
    vector<vector<ll>> p_pow, pinv_pow;
    /*
    use this:
    vector<ll> p = {773,709};
    vector<ll> mods = {281559881,398805713};
    */
    hashTable(int _n, string _s, int
_limit=2, vector<ll> _primes={773,709},
vector<ll> _mods = {281559881,398805713}) {
        this->s = _s; //1 indexed
        this->n = _n;
        this->limit = _limit;
        this->primes = _primes;
        this->m = _mods;
        this->p_pow.resize(limit);
        this->pinv_pow.resize(limit);
        for(int i = 0; i < limit; i++) {
            p_pow[i].resize(n+2),
pinv_pow[i].resize(n+2);
            p_pow[i][0] = pinv_pow[i][0] =
1;
            for(int j = 1;j <= n; j++) {
                p_pow[i][j] = (p_pow[i][j-1]
* primes[i]) % m[i];
            }
            pinv_pow[i][1] = inv(primes[i],
m[i]);
            for(int j = 2; j <= n; j++) {
                pinv_pow[i][j] =
(pinv_pow[i][j-1] * pinv_pow[i][1]) % m[i];
            }
        }
        pref.resize(this->limit);
        suff.resize(this->limit);
        for(int i = 0; i < limit; i++) {
            pref[i].resize(n+2);
            suff[i].resize(n+2);
        }
        precompute();
    }
    void precompute() {
        for(int ith_hash = 0; ith_hash <
limit; ith_hash++) {
            ll hash_value = 0;
            for(int i = 1; i <= n; i++) {
                hash_value = (hash_value +
(s[i] - 'a' + 1) * p_pow[ith_hash][i-1]) %
m[ith_hash];
                pref[ith_hash][i] =
hash_value;
            }
            pref[ith_hash][0] = 0;
        }
        for(int ith_hash = 0; ith_hash <
limit; ith_hash++) {
            ll hash_value = 0;
            for(int i = n; i >= 1; i--) {
hash_value = (hash_value + (s[i] - 'a' + 1)
* p_pow[ith_hash][n-i]) % m[ith_hash];
                suff[ith_hash][i] =
hash_value;
            }
            suff[ith_hash][n+1] = 0;
        }
    }
    ll get_pref(int l, int r, int ith_hash)
{
        ll here = sub(pref[ith_hash][r],
pref[ith_hash][l-1], m[ith_hash]);
        here = mult(here,
pinv_pow[ith_hash][l-1], m[ith_hash]);
        return here;
    }
    ll get_suff(int l, int r, int ith_hash)
{
        ll here = sub(suff[ith_hash][l],
suff[ith_hash][r+1], m[ith_hash]);
        here = mult(here,
pinv_pow[ith_hash][n-r], m[ith_hash]);
        return here;
    }
    long long binpow(long long a, long long
b, long long m) {
        a %= m;
        long long res = 1;
        while (b > 0) {
            if (b & 1)
                res = res * a % m;
            a = a * a % m;
            b >>= 1;
        }
        return res;
    }
    ll inv(ll x, ll m) {
        return binpow(x, m-2, m);
    }
    ll sub(ll x, ll y, ll m) {
        x %= m, y %= m;
        x-= y;
        if(x < 0) x += m;
        return x;
    }
    ll mult(ll x, ll y, ll m)  {
        x %= m, y %= m;
        return (x * y) % m;
    }
    bool is_palindrome(int l, int r) {
        if(get_pref(l, r, 0) == get_suff(l,
r, 0) && get_pref(l, r, 1) == get_suff(l, r,
1)) return true;
        return false;
```

```
    }
};
```

Persistent Segment Tree:

```
//q(1) -> a[k][i] = x
//q(2) -> a[k][l] + ... + a[k][r]
//a(3) -> append(a[k]), n += 1
struct Node {
	ll val;
	Node *l, *r;
	Node(ll x) : val(x), l(nullptr),
r(nullptr) {}
	Node(Node *ll, Node *rr) {
		l = ll, r = rr;
		val = 0;
		if (l) val += l->val;
		if (r) val += r->val;
	}
	Node(Node *cp) : val(cp->val),
l(cp->l), r(cp->r) {}
};

int n, cnt = 1;
ll a[200001];
Node *roots[200001];

Node *build(int l = 1, int r = n) {
	if (l == r) return new Node(a[l]);
	int mid = (l + r) / 2;
	return new Node(build(l, mid),
build(mid + 1, r));
}

Node *update(Node *node, int val, int pos,
int l = 1, int r = n) {
	if (l == r) return new Node(val);
	int mid = (l + r) / 2;
	if (pos > mid) return new
Node(node->l, update(node->r, val, pos, mid
+ 1, r));
	else return new Node(update(node->l,
val, pos, l, mid), node->r);
}

ll query(Node *node, int a, int b, int l =
1, int r = n) {
	if (l > b || r < a) return 0;
	if (l >= a && r <= b) return
node->val;
	int mid = (l + r) / 2;
	return query(node->l, a, b, l, mid) +
query(node->r, a, b, mid + 1, r);
}

int main() {
	ios_base::sync_with_stdio(0);
	cin.tie(0);
	int q;
	cin >> n >> q;
	for (int i = 1; i <= n; i++) cin >>
a[i];
	roots[cnt++] = build();

	while (q--) {
		int t;
		cin >> t;
		if (t == 1) {
			int k, i, x;
			cin >> k >> i >> x;
			roots[k] =
update(roots[k], x, i);
		} else if (t == 2) {
			int k, l, r;
			cin >> k >> l >> r;
			cout << query(roots[k],
l, r) << '\n';
		} else {
			int k;
			cin >> k;
			roots[cnt++] = new
Node(roots[k]);
		}
	}
	return 0;
}
```

BIT:

```
//point update range sum
struct FenwickTreeOneBasedIndexing {
    vector<int> bit;  // binary indexed tree
    int n;
    FenwickTreeOneBasedIndexing(int n) {
        this->n = n + 1;
        bit.assign(n + 1, 0);
    }
    FenwickTreeOneBasedIndexing(vector<int>
a)
        :
FenwickTreeOneBasedIndexing(a.size()) {
        for (size_t i = 0; i < a.size();
i++)
            add(i, a[i]);
    }
    int sum(int idx) {
        int ret = 0;
        for (++idx; idx > 0; idx -= idx &
-idx)
            ret += bit[idx];
        return ret;
    }
    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
```

```
    }
    void add(int idx, int delta) {
        for (++idx; idx < n; idx += idx &
-idx)
            bit[idx] += delta;
    }
};
//range update point query
void range_add(int l, int r, int val) {
    add(l, val);
    add(r + 1, -val);
}

int point_query(int idx) {
    int ret = 0;
    for (++idx; idx > 0; idx -= idx & -idx)
        ret += bit[idx];
    return ret;
}
```

//2D BIT:

```
//vector<vector<int>>bit of n*n
//sum(x,y)->
//for(x++;x>0;x-=x&-x) {
//  for(y++;y>0;y-=y&-y)res+=bit[x][y]
//}
```

Divide and Conquer DP:

```
int m, n;
vector<long long> dp_before, dp_cur;
long long C(int i, int j);   //we can do
MO's algo like addition and removal here, in
amortized O(1) time.

// 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];
}
```

SOS DP:

```
for (int s=m; s; s=(s-1)&m)
```

Indexed Set:

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template <typename T> using indexed_set =
tree<T,null_type,less<T>,rb_tree_tag,
tree_order_statistics_node_update> ; //
order_of_key(x) = # of elements smaller than
the element x, find_by_order(x) = x-th
element of the set
```

Linear Diophantine Equation

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c,
int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
void shift_solution(int & x, int & y, int a,
int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
```

```
}

int find_all_solutions(int a, int b, int c,
int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y,
g))
        return 0;
    a /= g;
    b /= g;
    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;
    shift_solution(x, y, a, b, (minx - x) /
b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;
    shift_solution(x, y, a, b, (maxx - x) /
b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;
    shift_solution(x, y, a, b, -(miny - y) /
a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy) return 0;
    int lx2 = x;
    shift_solution(x, y, a, b, -(maxy - y) /
a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2) swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);
    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

Treap

```
// given a string s, perform m operations of
the form:
// 1. reverse the substring s[l..r]
(1-indexed)
// 2. move the substring s[l..r] to the end
of the string

struct Node {
    Node *l = 0, *r = 0;
    char val;  // if it was a string
    int y, c = 1;

    bool rev = false;
    int sum;
    int add = 0;
       bool has_add = 0;

    Node(char val) : val(val), y(rand()) {}
    void recalc();
};
int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() {
    c = 1 + cnt(l) + cnt(r);
}
void push(Node* n) {   // push the reverse
flag down
    if (!n) return;
    if (n->rev) {
        swap(n->l, n->r); // reverse
children
        if (n->l) n->l->rev ^= true;
        if (n->r) n->r->rev ^= true;
        n->rev = false;
    }
    if (n->has_add) {
        n->val += n->add;
        n->sum += n->add * cnt(n);
        n->has_add = false;
        n->add = 0;

        if(n->l) n->l->has_add = true,
n->l->add += n->add;
        if(n->r) n->r->has_add = true,
n->r->add += n->add;

    }
}
pair<Node*, Node*> split(Node* n, int k) {
// Split the tree into two parts: the first
part contains the first k nodes
    if (!n) return {};
    push(n);
    if (cnt(n->l) >= k) {
        auto [L, R] = split(n->l, k);
        n->l = R;
        n->recalc();
        return {L, n};
    } else {
        auto [L, R] = split(n->r, k -
cnt(n->l) - 1);
        n->r = L;
        n->recalc();
        return {n, R};
    }
}

Node* merge(Node* l, Node* r) {
    if (!l) return r;
    if (!r) return l;
    push(l);
```

```
    push(r);
    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) {
//insert n at position pos, 0-indexed
    auto [l, r] = split(t, pos);
    return merge(merge(l, n), r);
}
Node* erase(Node* t, int pos) {  //erase at
pos, 0 indexed
       Node *a, *b, *c;
       tie(a, b) = split(t, pos);
       tie(b, c) = split(b, 1);
       delete b; // free memory if needed
       return merge(a, c);
}
void move(Node*& t, int l, int r, int k) {
// Example application: move the range [l,
r) to index 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));
}
// get range sum of [l, r)
int range_sum(Node*& t, int l, int r) {
       Node *a, *b, *c;
       tie(a, b) = split(t, l);
       tie(b, c) = split(b, r - l);
       int res = b->sum;
       t = merge(a, merge(b, c));
       return res;
}
// range add [l, r) by x
// if(b) b->add += x, b->has_add = true;
// merge back to t

// range reverse [l, r)
// if(b) b->rev ^= true;
// merge back to t

void each(Node* n, string& out) {
    if (!n) return;
    push(n);
    each(n->l, out);
    out += n->val;
    each(n->r, out);
}

int main() {
    int n, m;
    cin >> n >> m;
    string s;
    cin >> s;

    Node* treap = nullptr;
    for (char ch : s)
        treap = merge(treap, new Node(ch));

    while (m--) {
        int a, b;
        cin >> a >> b;
        Node *left, *mid, *right;
        tie(left, mid) = split(treap, a -
1);
        tie(mid, right) = split(mid, b - a +
1);
        treap = merge(left, right);
        mid->rev ^= true;
        treap = merge(treap, mid); // move
to end
    }

    string res;
    each(treap, res);  // print the array
    cout << res << '\n';
}
```

Gaussian Algorithm

```
const int N = 3e5 + 9;
const double eps = 1e-9;
int Gauss(vector<vector<double>> a,
vector<double> &ans) {
  int n = (int)a.size(), m =
(int)a[0].size() - 1;
  vector<int> pos(m, -1);
  double det = 1; int rank = 0;
  for(int col = 0, row = 0; col < m && row <
n; ++col) {
    int mx = row;
    for(int i = row; i < n; i++)
if(fabs(a[i][col]) > fabs(a[mx][col])) mx =
i;
    if(fabs(a[mx][col]) < eps) {det = 0;
continue;}
    for(int i = col; i <= m; i++)
swap(a[row][i], a[mx][i]);
    if (row != mx) det = -det;
    det *= a[row][col];
    pos[col] = row;
    for(int i = 0; i < n; i++) {
      if(i != row && fabs(a[i][col]) > eps)
{
```

```
        double c = a[i][col] / a[row][col];
        for(int j = col; j <= m; j++)
a[i][j] -= a[row][j] * c;
      }
    }
    ++row; ++rank;
  }
  ans.assign(m, 0);
  for(int i = 0; i < m; i++) {
    if(pos[i] != -1) ans[i] = a[pos[i]][m] /
a[pos[i]][i];
  }
  for(int i = 0; i < n; i++) {
    double sum = 0;
    for(int j = 0; j < m; j++) sum += ans[j]
* a[i][j];
    if(fabs(sum - a[i][m]) > eps) return -1;
//no solution
  }
  for(int i = 0; i < m; i++) if(pos[i] ==
-1) return 2; //infinite solutions
  return 1; //unique solution
}
int main() {
  int n, m; cin >> n >> m;
  vector< vector<double> > v(n);
  for(int i = 0; i < n; i++) {
    for(int j = 0; j <= m; j++) {
      double x; cin >> x; v[i].push_back(x);
    }
  }
  vector<double> ans;
  int k = Gauss(v, ans);
  if(k) for(int i = 0; i < n; i++) cout <<
fixed << setprecision(5) << ans[i] << ' ';
  else cout << "no solution\n";
  return 0;
}
```

```
int gauss (vector < bitset<N> > a, int n,
int m, bitset<N> & ans) {
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n;
++col) {
        for (int i=row; i<n; ++i)
            if (a[i][col]) {
                swap (a[i], a[row]);
                break;
            }
        if (! a[row][col])
            continue;
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row && a[i][col])
                a[i] ^= a[row];
        ++row;
    }
        // The rest of implementation is the
same as above
}
```

Lazy Segment tree recursive:

```
template <class S,
          S (*op)(S, S),
          S (*e)(),
          class F,
          S (*mapping)(F, S),
          F (*composition)(F, F),
          F (*id)()>
struct lazy_segtree {
//internally 1 indexing
    public:
    lazy_segtree() : lazy_segtree(0) {}
    explicit lazy_segtree(int n) :
lazy_segtree(std::vector<S>(n, e())) {}
    explicit lazy_segtree(const
std::vector<S>& v) : _n(int(v.size())) {
        //v is 1 indexed
        size = _n - 1;
        d = std::vector<S>(4 * _n + 5, e());
        lz = std::vector<F>(4 * _n + 5,
id());
        arr = v;
        build(1, 1, size);
    }
    void build(int tv, int tl, int tr) {
        if(tl == tr) d[tv] = arr[tl];
        else {
            int tm = (tl + tr) >> 1;
            build(tv*2, tl, tm);
            build(tv*2+1, tm+1, tr);
            d[tv] = op(d[tv*2], d[tv*2 +
1]);
        }
    }
    void push(int tv, int tl, int tr) {
        if(lz[tv]==id()) return;
        d[tv] = mapping(lz[tv], d[tv]);
        if(tl < tr) {
            lz[tv*2] = composition(lz[tv*2],
lz[tv]);
            lz[tv*2+1] =
composition(lz[tv*2+1], lz[tv]);
        }
        lz[tv] = id();
    }
    void update(int l, int r, F f, int tv,
int tl, int tr, int init) {
         //printf("now at %d %d\n", tl, tr);
        push(tv, tl, tr);
        if(r < l) return;
        if(l == tl && r == tr) {
            lz[tv] = f;
```

```
            push(tv, tl, tr);
        }
        else {
            int tm = (tl + tr) / 2;
            update(l, min(tm, r), f, tv*2,
tl, tm, init);
            update(max(tm+1, l), r, f,
tv*2+1, tm+1, tr, init);
            d[tv] = op(d[tv*2], d[tv*2+1]);
        }
    }
    S get(int l, int r, int tv, int tl, int
tr) {
        push(tv, tl, tr);
        if(r < l) return e();
        if(l == tl && r == tr) {
            return d[tv];
        }
        else {
            int tm = (tl + tr) / 2;
            return op(get(l, min(tm, r),
tv*2, tl, tm),
            get(max(tm+1, l), r, tv*2+1,
tm+1, tr));
        }
    }
    void apply(int l, int r, F f) {
        update(l, r-1, f, 1, 1, size, l);
    }
    S prod(int l, int r) {
        return get(l, r-1, 1, 1, size);
    }
  private:
    int _n, size, log;
    std::vector<S> d;
    std::vector<S> arr;
    std::vector<F> lz;
};
```

Aho Corasick:

```
struct AC {
struct state {
  int to[ALPHA],depth,sLink,
  int par,parLet,cnt,nxt[ALPHA];
}states[N];
vector<int> suff_tree[N]; int tot_nodes;
void init() {
  for(int i = 0; i < N; i++)
suff_tree[i].clear();
  tot_nodes = 1; clr(states);
//careful,memset TLE
}
int add_string(string &str) {
  int cur = 1;
  for(int i = 0; i < str.size(); i++) {
    int c = str[i]-'a';
    if(!states[cur].to[c]) {
      states[cur].to[c] = ++tot_nodes;
      states[tot_nodes].par = cur;

states[tot_nodes].depth=states[cur].depth+1;
      states[tot_nodes].parLet = c;
    }
    cur = states[cur].to[c];
  }
  return cur;
}
void push_links() {
  queue <int> qq;
  qq.push(1);
  while (!qq.empty()) {
    int node = qq.front();
    qq.pop();
    if (states[node].depth <= 1)
      states[node].sLink = 1;
    else {
      int cur =
states[states[node].par].sLink;
      int parLet = states[node].parLet;
      while (cur > 1 and
!states[cur].to[parLet]){
        cur = states[cur].sLink;
      }
      if (states[cur].to[parLet]) {
        cur = states[cur].to[parLet];
      }
      states[node].sLink = cur;
    }
    if(node!=1)

suff_tree[states[node].sLink].pb(node);
    for (int i = 0 ; i < ALPHA; i++) {
      if(states[node].to[i])
        qq.push(states[node].to[i]);
    }
  }
}
int next_state(int from, int c) {
  if(states[from].nxt[c])
    return states[from].nxt[c];
  int cur = from;
  while(cur>1&&!states[cur].to[c])
    cur=states[cur].sLink;
  if(states[cur].to[c]) cur =
states[cur].to[c];
  return states[from].nxt[c] = cur;
}
void dfs(int u) {
  for(int v : suff_tree[u]) {
    dfs(v); states[u].cnt += states[v].cnt;
  }
}
}aho;
```

Manacher:

```
//p[0][i] = maxlen of hlf palin arnd half
idx i
//p[1][i] = maxlen of hlf palin arnd idx i,0
based
VI p[2];
void manacher(const string s) {
  int n = s.size(); p[0] = VI(n+1); p[1] =
VI(n);
  for (int z=0; z<2; z++)
    for (int i=0, l=0, r=0; i<n; i++) {
      int t = r - i + !z;
      if (i<r) p[z][i] = min(t, p[z][l+t]);
      int L = i-p[z][i], R = i+p[z][i] - !z;
      while (L>=1 && R+1<n && s[L-1] ==
s[R+1])
        p[z][i]++, L--, R++;
      if (R>r) l=L, r=R;
    }
}
bool ispalin(int l, int r) {
  int mid = (l+r+1)/2, sz = r-l+1;
  return 2*p[sz%2][mid] + sz%2 >=sz;
}
namespace mincost {
    const int V=40100,E=1001000,_inf=0x20;
    int q[V*30],
vis[V],fst[V],pre[V],nxt[E],y[E],f[E],S,T,
flow,tot,tn; const ll inf=1ll<<60; ll
dis[V],c[E],cost;
    void init(int s,int t,int Tn) {
tot=1; tn=Tn; F0R(i,tn) fst[i]=0; S=s;T=t;}
    void add(int u,int v,int ff,int cc) {

tot++;y[tot]=v;nxt[tot]=fst[u];f[tot]=ff;c[t
ot]=cc;fst[u]=tot;
tot++;y[tot]=u;nxt[tot]=fst[v];f[tot]=0;c[to
t]=-cc;fst[v]=tot;
    }
    bool spfa() {
      F0R(i,tn)
dis[i]=inf,vis[i]=0,pre[i]=0;
      dis[S]=0;q[0]=S;vis[S]=1; int t=1;
      F0R(i,t) {
          int u=q[i];
          for (int j=fst[u];j;j=nxt[j]) {
            int v=y[j];
            if (f[j]&&dis[v]>dis[u]+c[j]) {
                dis[v]=dis[u]+c[j];
pre[v]=j;
                if (!vis[v])
vis[v]=1,q[t++]=v;
            }
          }
          vis[u]=0;
      }
      return dis[T]!=inf;
    }
    void augment() {
      int p=T, _f=1<<30; // For pos, set _f
= 0
      while (pre[p])
_f=min(_f,f[pre[p]]),p=y[pre[p]^1];
      flow+=_f;cost+=_f*dis[T]; p=T;
      while (pre[p])
f[pre[p]]-=_f,f[pre[p]^1]+=_f,p=y[pre[p]^1];
    }
    void solve() {
      flow=0,cost=0;vector<ll> ans;
      while (spfa()) augment();
ans.pb(-cost);
      // For pos, ans.pb(cost);
    }
}
// mincost::init(source,sink,total_nodes)
// mincost::add(from,to,cap,cost)
// For neg, set _f=big, and cost = actual
cost
// mincost::solve()
```

Command with necessary flags:

g++ -std=c++17 -Wshadow -Wall -o "%e" -g -fsanitize=address -fsanitize=undefined -D_GLIBCXX_DEBUG

RNG:

```
mt19937
rng((unsigned)chrono::system_clock::now().time_since_epoch().count()); //mt199937_64 for ll
```

Suffix Automaton:

```
struct state {
    int len, link; map<char, int> next;
};
const int MAXLEN = 100000;
state st[MAXLEN * 2];int sz, last;
void sa_init() {
    st[0].len = 0; st[0].link = -1; sz++;
    last = 0;
}
void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)){
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) st[cur].link = 0;
    else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1
              && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link=st[cur].link=clone;
        }
    }
    last = cur;
}
```