r(e(c(u(r(s(i(o(n)))))))))

Contents

1 Combinatorial optimization Dynamic Programming

1.1 Dynamic Programming Optimization - Convex Hull Trick2

```cpp
struct Convex{
  typedef pair<int,int> PII;
  vector<PII> lns;
  bool shit(int a,int b,int c){
   return (lns[c].second-lns[a].second)*(lns[a].first-lns[b].first)<(lns[b].second-lns[a].second)*(lns[a].first-lns[c].first);//if WA comes, it might be because this has overflowen. In that case make, floating point comparison
  }
  //line equation is Ax + B
  void add_line(int A,int B){
   lns.push_back({A,B});
   while(sz(lns)>=3&&shit(sz(lns)-3,sz(lns)-2,sz(lns)-1)){
     lns[sz(lns)-2]=lns[sz(lns)-1];
     lns.pop_back();
   }
  }
  int get(int whr,int x){
   return lns[whr].first*x+lns[whr].second;
  }
  int query(int x){
   int l=0,r=sz(lns)-1;
   while(r>l){
     int m=(l+r)>>1;
     if(get(m,x)>=get(m+1,x)){
      l=m+1;
     }else{
      r=m;
     }
   }
   return get(l,x);
  }
}s;
```

1.2 Dynamic Programming Optimization - Convex Hull Trick3

```cpp
// dp_hull enables you to do the following two operations in amortized O(log n) time:
// 1. Insert a pair (a_i, b_i) into the structure
// 2. For any value of x, query the maximum value of a_i * x + b_i
// All values a_i, b_i, and x can be positive or negative.
struct dp_hull {
  struct segment {
    point p;
    mutable point next_p;
    segment(point _p = {0, 0}, point _next_p = {0, 0}) : p(_p), next_p(_next_p) {}
    bool operator<(const segment &other) const {
      // Sentinel value indicating we should binary search the set for a single x-value.
      if (p.y == LL_INF)
        return p.x * (other.next_p.x - other.p.x) <= (other.p.y - other.next_p.y) * next_p.x;
      return make_pair(p.x, p.y) < make_pair(other.p.x, other.p.y);
    }
  };
  set&lt;segment&gt; segments;
  bool empty() const {
    return segments.empty();
  }
  set&lt;segment&gt;::iterator prev(set&lt;segment&gt;::iterator it) const {
    return it == segments.begin() ? it : --it;
  }
  set&lt;segment&gt;::iterator next(set&lt;segment&gt;::iterator it) const {
    return it == segments.end() ? it : ++it;
  }
  bool bad(set&lt;segment&gt;::iterator it) const {
    return it != segments.begin() && next(it) != segments.end() && left_turn(prev(it)->p, it->p, next(it)->p);
  }
  void insert(const point &p) {
    set&lt;segment&gt;::iterator it = segments.insert(segment(p, p)).first;
    if (bad(it) || (next(it) != segments.end() && it->p.x == next(it)->p.x)) {
      segments.erase(it);
      return;
    }
    if (it != segments.begin() && it->p.x == prev(it)->p.x)
      segments.erase(prev(it));
    while (bad(prev(it)))
      segments.erase(prev(it));
    while (bad(next(it)))
      segments.erase(next(it));
    if (it != segments.begin())
      prev(it)->next_p = it->p;
    if (next(it) != segments.end())
      it->next_p = next(it)->p;
```

```
    }
    void insert(long long a, long long b) {
      insert(point(a, b));
    }
    // Queries the maximum value of ax + by.
    long long query(long long x, long long y = 1) const {
      assert(y > 0);
      set&lt;segment&gt;::iterator it = segments.upper_bound(segment(point(x, LL_INF), point(y, LL_INF)));
      return it->p.x * x + it->p.y * y;
    }
};
```

1.3 Dynamic Programming Optimization - Convex Hull Trick4

```
const int INF = 1e9 + 5;
struct point {
  int x, y;
  point() : x(0), y(0) {}
  point(int _x, int _y) : x(_x), y(_y) {}
};
// reverse_monotonic_dp_hull enables you to do the following two operations in amortized O(1) time:
// 1. Insert a pair (a_i, b_i) into the structure. a_i must be non-decreasing.
// 2. For any value of x, query the maximum value of a_i * x + b_i. x must be non-increasing.
// All values a_i, b_i, and x can be positive or negative.
struct reverse_monotonic_dp_hull {
  vector<point> points;
  int size() const {
    return (int) points.size();
  }
  void clear() {
    points.clear();
    prev_x = INF;
  }
  static int floor_div(int a, int b) {
    return a / b - ((a ^ b) < 0 && a % b != 0);
  }
  static bool bad_middle(const point &a, const point &b, const point &c) {
    // This checks whether the x-value where b becomes better than a comes after the x-value where c becomes better
    // than a. It's fine to round down here if we will only query integer x-values. (C++ division rounds to zero)
    return floor_div(a.y - b.y, b.x - a.x) >= floor_div(b.y - c.y, c.x - b.x);
  }
  void insert(const point &p) {
    assert(size() == 0 || p.x >= points.back().x);
    if (size() > 0 && p.x == points.back().x) {
      if (p.y <= points.back().y)
        return;
      points.pop_back();
    }
    while (size() >= 2 && bad_middle(points[points.size() - 2], points.back(), p))
      points.pop_back();
    points.push_back(p);
  }
  void insert(int a, int b) {
    insert(point(a, b));
  }
  int prev_x = INF;
  // Queries the maximum value of ax + b.
  int query(int x) {
    assert(x <= prev_x);
    prev_x = x;
    while (size() >= 2 && x * (points.back().x - points[size() - 2].x) <= points[size() - 2].y - points.back().y)
      points.pop_back();
    return points.back().x * x + points.back().y;
  }
};
```

1.4 Digit DP

```
/*
f => 1 => tight
f => 0 => restrictless
*/
int si, dp[19][1<<8][2520];
std::vector<int> v;
int dfs(int idx, int f, int mask, int rem){
 // digit dp from idx = sz-1 to 0 is better than 0 to sz-1.
 // reason, you don't need to memset again and agains
 if(idx <= -1){
  int tt = 1;
  bool good = true;
  for (int i = 2; i <= 9; ++i)
  {
   if(mask & (1<<(i-2))){
    good &= (rem % i == 0);
   }
  }
  return good;
 }
 if(f == 0 and dp[idx][mask][rem] != -1){
  return dp[idx][mask][rem];
 }
 int lim = (f) ? v[idx] : 9;
 int ans = 0;
 for (int i = 0; i <= lim; ++i)
 {
  int nf = (v[idx] == i) ? f : 0;
  int nmask = mask;
  if(i > 1){
   nmask |= (1LL<<(i-2));
  }
  ans += dfs(idx-1, nf, nmask, (rem*10+i)%2520);
 }
 if (f == 0)
  dp[idx][mask][rem] = ans;
 return ans;
}
int get(int num){
 v.clear();
 while(num){
  v.pb(num % 10);
  num /= 10;
 }
 si = sz(v);
 int tt = dfs(si-1, 1, 0, 0);
 return tt;
}
int32_t main(){
 int t;
 scanf("%lld", &t);
 memset(dp, -1, sizeof dp);
 while(t--){
  int l, r;
  scanf("%lld %lld", &l, &r);
  int gg = get(r);
  int tt = get(l-1);
  printf("%lld\n", gg-tt);
 }
 return 0;
}
```

1.5 SOS DP

```
// set the max value of bits in mask
const int C = 20;
vector<int32_t> max_subset(1 << C, 0);
// For every mask, computes the max of values[sub] where sub is a submask of mask.
template<typename T_out, typename T_in>
vector<T_out> submask_max(int n, const vector<T_in> &values) {
  vector<T_out> dp(values.begin(), values.end());
  for (int bit = 0; bit < n; bit++)
    for (int mask = 0; mask < 1 << n; mask++)
      if (mask >> bit & 1)
        dp[mask] = max(dp[mask], dp[mask ^ 1 << bit]);
  return dp;
}
/***
make changes to max_subset as question demands, and then use below to get the SOS dp
max_subset = submask_max<int32_t>(C, max_subset);
***/
```

2 Geometry

2.1 Geometry1

```
// https://github.com/jaehyunp/stanfordacm/blob/master/code/Geometry.cc
// C++ routines for computational geometry.
#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>
using namespace std;
double INF = 1e100;
double EPS = 1e-12;
struct PT {
 double x, y;
 PT() {}
 PT(double x, double y) : x(x), y(y) {}
 PT(const PT &p) : x(p.x), y(p.y) {}
 PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
 PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
 PT operator * (double c) const { return PT(x*c, y*c ); }
 PT operator / (double c) const { return PT(x/c, y/c ); }
};
double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
 return os << "(" << p.x << "," << p.y << ")";
}
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p) { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
 return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
// project point c onto line through a and b
// assuming a != b
```

```
PT ProjectPointLine(PT a, PT b, PT c) {
 return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}
// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
 double r = dot(b-a,b-a);
 if (fabs(r) < EPS) return a;
 r = dot(c-a, b-a)/r;
 if (r < 0) return a;
 if (r > 1) return b;
 return a + (b-a)*r;
}
// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
 return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
        double a, double b, double c, double d)
{
 return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
 return fabs(cross(b-a, c-d)) < EPS;
}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
 return LinesParallel(a, b, c, d)
   && fabs(cross(a-b, a-c)) < EPS
   && fabs(cross(c-d, c-a)) < EPS;
}
// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
 if (LinesCollinear(a, b, c, d)) {
  if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
   dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
  if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
   return false;
  return true;
 }
 if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
 if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
 return true;
}
// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
 b=b-a; d=c-d; c=c-a;
 assert(dot(b, b) > EPS && dot(d, d) > EPS);
 return a + b*cross(c, d)/cross(b, d);
}
// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
 b=(a+b)/2;
 c=(a+c)/2;
 return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}
// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
 bool c = 0;
 for (int i = 0; i < p.size(); i++) {
  int j = (i+1)%p.size();
  if ((p[i].y <= q.y && q.y < p[j].y ||
   p[j].y <= q.y && q.y < p[i].y) &&
   q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
   c = !c;
 }
 return c;
}
// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
 for (int i = 0; i < p.size(); i++)
  if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
   return true;
  return false;
}
// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
 vector<PT> ret;
 b = b-a;
 a = a-c;
 double A = dot(b, b);
 double B = dot(a, b);
 double C = dot(a, a) - r*r;
 double D = B*B - A*C;
 if (D < -EPS) return ret;
 ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
 if (D > EPS)
  ret.push_back(c+a+b*(-B-sqrt(D))/A);
 return ret;
}
// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
 vector<PT> ret;
 double d = sqrt(dist2(a, b));
 if (d > r+R || d+min(r, R) < max(r, R)) return ret;
 double x = (d*d-R*R+r*r)/(2*d);
 double y = sqrt(r*r-x*x);
 PT v = (b-a)/d;
 ret.push_back(a+v*x + RotateCCW90(v)*y);
 if (y > 0)
  ret.push_back(a+v*x - RotateCCW90(v)*y);
 return ret;
}
// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion. Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
 double area = 0;
 for(int i = 0; i < p.size(); i++) {
  int j = (i+1) % p.size();
  area += p[i].x*p[j].y - p[j].x*p[i].y;
 }
 return area / 2.0;
}
double ComputeArea(const vector<PT> &p) {
 return fabs(ComputeSignedArea(p));
}
PT ComputeCentroid(const vector<PT> &p) {
 PT c(0,0);
 double scale = 6.0 * ComputeSignedArea(p);
 for (int i = 0; i < p.size(); i++){
  int j = (i+1) % p.size();
  c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
 }
 return c / scale;
}
// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
 for (int i = 0; i < p.size(); i++) {
  for (int k = i+1; k < p.size(); k++) {
   int j = (i+1) % p.size();
   int l = (k+1) % p.size();
   if (i == l || j == k) continue;
   if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
    return false;
  }
 }
 return true;
}
int main() {
 // expected: (-5,2)
 cerr << RotateCCW90(PT(2,5)) << endl;
 // expected: (5,-2)
 cerr << RotateCW90(PT(2,5)) << endl;
 // expected: (-5,2)
 cerr << RotateCCW(PT(2,5),M_PI/2) << endl;
 // expected: (5,2)
 cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;
 // expected: (5,2) (7.5,3) (2.5,1)
 cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
      << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
      << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;
 // expected: 6.78903
 cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;
 // expected: 1 0 1
 cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
      << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
      << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;
 // expected: 0 0 1
 cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
      << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
      << LinesCollinear(PT(1,1), PT
```

```
    (3,5), PT(5,9), PT(7,13)) << endl;
  // expected: 1 1 1 0
  cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;
  // expected: (1,2)
  cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << endl;
  // expected: (1,1)
  cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;
  vector<PT> v;
  v.push_back(PT(0,0));
  v.push_back(PT(5,0));
  v.push_back(PT(5,5));
  v.push_back(PT(0,5));
  // expected: 1 1 1 0 0
  cerr << PointInPolygon(v, PT(2,2)) << " "
       << PointInPolygon(v, PT(2,0)) << " "
       << PointInPolygon(v, PT(0,2)) << " "
       << PointInPolygon(v, PT(5,2)) << " "
       << PointInPolygon(v, PT(2,5)) << endl;
  // expected: 0 1 1 1 1
  cerr << PointOnPolygon(v, PT(2,2)) << " "
       << PointOnPolygon(v, PT(2,0)) << " "
       << PointOnPolygon(v, PT(0,2)) << " "
       << PointOnPolygon(v, PT(5,2)) << " "
       << PointOnPolygon(v, PT(2,5)) << endl;
  // expected: (1,6)
  //           (5,4) (4,5)
  //           blank line
  //           (4,5) (5,4)
  //           blank line
  //           (4,5) (5,4)
  vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  // area should be 5.0
  // centroid should be (1.1666666, 1.166666)
  PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
  vector<PT> p(pa, pa+4);
  PT c = ComputeCentroid(p);
  cerr << "Area: " << ComputeArea(p) << endl;
  cerr << "Centroid: " << c << endl;
  return 0;
}
```

3 Numerical algorithms

3.1 Chinese Reamінder Theorem

```
vector<int> rem, mod;
template<typename T> T extended_euclid(T a, T b, T &x, T &y) {
 T xx = 0, yy = 1; y = 0; x = 1;
 while(b) {
  T q = a / b, t = b;
  b = a % b; a = t;
  t = xx; xx = x - q * xx;
  x = t; t = yy;
  yy = y - q * yy; y = t;
 }
 return a;
}
template<typename T> T mod_inverse(T a, T n) {
 T x, y, z = 0;
 T d = extended_euclid(a, n, x, y);
 return (d > 1 ? -1 : mod_neg(x, z, n));
}
void pre_process() {
 int a = 1, b = 1, m = mod[0];
 crt.push_back({mod[0], {a, b}});
 for(int i = 1; i < mod.size(); ++i) {
  a = mod_inverse(m, mod[i]);
  b = mod_inverse(mod[i], m);
  crt.push_back({m, {a, b}});
  m *= mod[i];
 }
}
int find_crt() {
 int ans = rem[0], m = crt[0].first, a, b;
 for(int i = 1; i < mod.size(); ++i) {
  a = crt[i].second.first;
  b = crt[i].second.second;
  m = crt[i].first;
  ans = (1ll*ans * b * mod[i] + 1ll*rem[i] * a * m) % (m * mod[i]);
 }
 return ans;
}
```

3.2 Linear Diaphontine Equation

```
//linear diaphontine equation
long long mulmod(long long a, long long b, long long c) {
 long long sign = 1;
 if (a < 0) {
  a = -a;
  sign = -sign;
 }
 if (b < 0) {
  b = -b;
  sign = -sign;
 }
 a %= c;
 b %= c;
 long long res = 0;
 while (b > 0) {
  if (b & 1) {
   res = (res + a) % c;
  }
  a = (a + a) % c;
  b >>= 1;
 }
 if (sign == -1) {
  res = (-res) % c;
 }
 return res;
}
template<typename T>
T extgcd(T a, T b, T &x, T &y) {
 if (a == 0) {
  x = 0;
  y = 1;
  return b;
 }
 T p = b / a;
 T g = extgcd(b - p * a, a, y, x);
 x -= p * y;
 return g;
}
template<typename T>
bool diophantine(T a, T b, T c, T &x, T &y, T &g) {
 if (a == 0 && b == 0) {
  if (c == 0) {
   x = y = g = 0;
   return true;
  }
  return false;
 }
 if (a == 0) {
  if (c % b == 0) {
   x = 0;
   y = c / b;
   g = abs(b);
   return true;
  }
  return false;
 }
 if (b == 0) {
  if (c % a == 0) {
   x = c / a;
   y = 0;
   g = abs(a);
   return true;
  }
  return false;
 }
 g = extgcd(a, b, x, y);
 if (c % g != 0) {
  return false;
 }
 T dx = c / a;
 c -= dx * a;
 T dy = c / b;
 c -= dy * b;
 x = dx + mulmod(x, c / g, b);
 y = dy + mulmod(y, c / g, a);
 g = abs(g);
 return true;
}
```

3.3 Mobius Function

```
const int MAX = 1000001;
vector<int> lp, primes, mobius;
void factor_sieve() {
 lp.resize(MAX);
 lp[1] = 1;
 for (int i = 2; i < MAX; ++i) {
  if (lp[i] == 0) {
   lp[i] = i;
   primes.emplace_back(i);
  }
  for (int j = 0; j < primes.size() && primes[j] <= lp[i]; ++j) {
   int x = i * primes[j];
   if (x >= MAX) break;
   lp[x] = primes[j];
  }
 }
}
//Complexity : O(n)
void mobius_sieve() {
 mobius.resize(MAX);
 mobius[1] = 1;
 for(int i = 2; i < MAX; ++i) {
```

```
    int w = lp[i];
    if (lp[i/w] == w) {
     mobius[i] = 0;
    }
    else {
     mobius[i] = -mobius[i/w];
    }
  }
}
```

3.4 Euler Phi using Sieve

```
int phi[N+5];
void phi_calc(){
  for (int i = 0; i < N; i++) {
   phi[i] = i;
  }
  for (int i = 1; i < N; i++)
   for (int j = 2 * i; j < N; j += i)
    phi[j] -= phi[i];
}
```

3.5 NCR Implementation 1

```
#define ll int
ll power(ll a, ll b, ll m = mod){if(b < 0)b += m-1; ll r = 1; while(b){if(b&1)r=(r*a)%m; a = (a*a)%m; b>>=1;}return r;}
int fact[N+10], ifact[N+10];
void pre(int N){
 fact[0] = 1;
 for (int i = 1; i <= N; ++i)
 {
  fact[i] = (i*fact[i-1])%mod;
 }
 ifact[N] = power(fact[N], -1);
 for (int i = N-1; i >= 0; --i)
 {
  ifact[i] = ((i+1)*ifact[i+1])%mod;
 }
}
int nCr(int n, int r){
 if(n < r)return 0;
 return ((fact[n]*ifact[r])%mod * ifact[n-r])%mod;
}
```

3.6 NTT Implementation 1

```
int add(int a, int b, int c) {
 int res = a + b;
 return (res >= c ? res - c : res);
}
int mod_neg(int a, int b, int c) {
 int res; if(abs(a-b) < c) res = a - b;
 else res = (a-b) % c;
 return (res < 0 ? res + c : res);
}
int mul(int a, int b, int c) {
 long long res = (long long)a * b;
 return (res >= c ? res % c : res);
}
int power(long long e, long long n, int m) {
 int x = 1, p = e % m;
 while(n) {
  if(n & 1) x = mul(x, p, m);
  p = mul(p, p, m);
  n >>= 1;
 }
 return x;
}
template<typename T> T extended_euclid(T a, T b, T &x, T &y) {
 T xx = 0, yy = 1; y = 0; x = 1;
 while(b) {
  T q = a / b, t = b;
  b = a % b; a = t;
  t = xx; xx = x - q * xx;
  x = t; t = yy;
  yy = y - q * yy; y = t;
 }
 return a;
}
template<typename T> T mod_inverse(T a, T n) {
 T x, y, z = 0;
 T d = extended_euclid(a, n, x, y);
 return (d > 1 ? -1 : mod_neg(x, z, n));
}
//NTT implementation below
template<typename T, T prime, T root, int logn> struct NTT {
 int n, L, MAX, last, *rev;
 T *omega_powers, root_inv;
 NTT() {
  last = -1;
  MAX = (1 << logn);
  rev = new int[MAX];
  omega_powers = new T[MAX];
  root_inv = mod_inverse(root, prime);
 }
 ~NTT() {
  delete rev;
  delete omega_powers;
 }
 void ReverseBits() {
  if (last != n) {
   for (int i=1, j=0; i<n; ++i) {
    int bit = n >> 1;
    for (; j>=bit; bit>>=1) j -= bit;
    j += bit;
    rev[i] = j;
   }
  }
 }
 void DFT(vector<T> &A, bool inverse = false) {
  for(int i = 0; i < n; ++i)
   if (rev[i] > i) swap(A[i], A[rev[i]]);
  for (int s = 1; s <= L; s++) {
   int m = 1 << s, half = m / 2;
   T wm = inverse ? root_inv : root;
   for(int i = m; i < n; i <<= 1)
     wm = mul(wm , wm, prime);
   omega_powers[0] = 1;
   for(int k = 1; k < half; ++k) {
    omega_powers[k] = mul(omega_powers[k-1], wm, prime);
   }
   for (int k = 0; k < n; k += m) {
    for (int j = 0; j < half; j++) {
     T v = mul(omega_powers[j] , A[k + j + half], prime);
     T u = A[k + j];
     A[k + j] = add(u, v, prime);
     A[k + j + half] = mod_neg(u, v, prime);
    }
   }
  }
  if (inverse) {
   T n_inv = mod_inverse(n, prime);
   for (int i = 0; i < n; i++) A[i] = mul(A[i], n_inv, prime);
  }
 }
 // c[k] = sum_{i=0}^k a[i] b[k-i] mod prime
 vector<T> Multiply(const vector<T> &a, const vector<T> &b) {
  L = 0;
  int sa = a.size(), sb = b.size();
  while ((1 << L) < (sa + sb - 1)) L++;
  n = 1 << L;
  ReverseBits();
  vector<T> aa(n, 0), bb(n, 0), cc;
  for (int i = 0; i < sa; ++i) aa[i] = a[i];
  for (int i = 0; i < sb; ++i) bb[i] = b[i];
  DFT(aa, false); DFT(bb, false);
  for (int i = 0; i < n; ++i) cc.push_back(mul(aa[i], bb[i], prime));
  DFT(cc, true);
  vector<T> ans;
  n = sa + sb - 1;
  for (int i = 0; i < n; ++i) ans.push_back(cc[i]);
  return ans;
 }
};
//prime = 2^k * m + 1
const int prime = 786433;  //equals prime
const int root = 10;   //root^size = 1 mod(prime)
const int size = 1 << 18;  //2^k
NTT<int, prime, root, 18> ntt;
// Usage fftMod<int, 998244353, -1, 23> // g = 3
//    fftMod<int, 1004535809, -1, 19> fft; // g = 3
//    fftMod<int, 469762049, -1, 26> // g = 3
//    fftMod<li, 10000093151233, -1, 26> // g = 5
//    fftMod<li, 1000000523862017, -1, 26> // g = 3
//    fftMod<li, 1000000000949747713, -1, 26> // g = 2
//    fftMod<li, -1, li(1e13), 20>
```

3.7 NTT Implementation 2

```
using int64 = long long;
const int64 INF = 1LL << 58;
template< int mod, int primitiveroot >
struct NumberTheoreticTransform {
 vector< vector< int > > rts, rrts;
 void ensure_base(int N) {
  if(rts.size() >= N) return;
  rts.resize(N), rrts.resize(N);
  for(int i = 1; i < N; i <<= 1) {
   if(rts[i].size()) continue;
   int w = mod_pow(primitiveroot, (mod - 1) / (i * 2));
   int rw = inverse(w);
   rts[i].resize(i), rrts[i].resize(i);
   rts[i][0] = 1, rrts[i][0] = 1;
   for(int k = 1; k < i; k++) {
    rts[i][k] = mul(rts[i][k - 1], w);
    rrts[i][k] = mul(rrts[i][k - 1], rw);
   }
  }
 }
 inline int mod_pow(int x, int n) {
  int ret = 1;
  while(n > 0) {
   if(n & 1) ret = mul(ret, x);
   x = mul(x, x);
   n >>= 1;
  }
  return ret;
 }
 inline int inverse(int x) {
  return mod_pow(x, mod - 2);
 }
 inline int add(int x, int y) {
  x += y;
  if(x >= mod) x -= mod;
  return x;
 }
 inline int mul(int a, int b) {
  return int(1LL * a * b % mod);
 }
 void DiscreteFourierTransform(vector< int > &F, bool rev) {
  const int N = (int) F.size();
  ensure_base(N);
  for(int i = 0, j = 1; j + 1 < N; j++) {
```

```
    for(int k = N >> 1; k > (i ^= k); k >>= 1);
    if(i > j) swap(F[i], F[j]);
   }
   for(int i = 1; i < N; i <<= 1) {
    for(int j = 0; j < N; j += i * 2) {
     for(int k = 0; k < i; k++) {
      int s = F[j + k], t = mul(F[j + k + i], rev ? rrts[i][k] : rts[i][k]);
      F[j + k] = add(s, t), F[j + k + i] = add(s, mod - t);
     }
    }
   }
   if(rev) {
    int temp = inverse(N);
    for(int i = 0; i < N; i++) F[i] = mul(F[i], temp);
   }
  }
  vector< int > Multiply(const vector< int > &A, const vector< int > &B) {
   int sz = 1;
   while(sz < A.size() + B.size() - 1) sz <<= 1;
   vector< int > F(sz), G(sz);
   for(int i = 0; i < A.size(); i++) F[i] = A[i];
   for(int i = 0; i < B.size(); i++) G[i] = B[i];
   DiscreteFourierTransform(F, false);
   DiscreteFourierTransform(G, false);
   for(int i = 0; i < sz; i++) F[i] = mul(F[i], G[i]);
   DiscreteFourierTransform(F, true);
   F.resize(A.size() + B.size() - 1);
   return F;
  }
};
const int mod = 998244353;
int main() {
 int N, K;
 vector< int > tap(1 << 20);
 cin >> N >> K;
 N >>= 1;
 for(int i = 0; i < K; i++) {
  int x;
  cin >> x;
  tap[x]++;
 }
 NumberTheoreticTransform< mod, 3 > ntt;
 ntt.DiscreteFourierTransform(tap, false);
 for(int i = 0; i < tap.size(); i++) {
  tap[i] = ntt.mod_pow(tap[i], N);
 }
 ntt.DiscreteFourierTransform(tap, true);
 int64 ret = 0;
 for(int i = 0; i < tap.size(); i++) {
  ret += 1LL * tap[i] * tap[i] % mod;
  ret %= mod;
 }
 cout << ret << endl;
}
```

3.8 Implementation of FFT

```
//General Modulo FFT implementation
//Quite fast as it uses just 4 FFT calls
//Just call preCompute() in main() before usage
const int MAX = 1e5 + 5;   //Size of Polynomial
const int MOD = 1e9 + 7;
namespace FFTMOD {
const double PI = acos(-1);
const int LIM = 1 << 18;   //2 * 2^ceil(log2(POLY_SIZE))
//Complex class: Quite faster than in-built C++ library as it uses only functions required
template<typename T> class cmplx {
private:
 T x, y;
public:
 cmplx () : x(0.0), y(0.0) {}
 cmplx (T a) : x(a), y(0.0) {}
 cmplx (T a, T b) : x(a), y(b) {}
 T get_real() { return this->x; }
 T get_img() { return this->y; }
 cmplx conj() { return cmplx(this->x, -(this->y)); }
 cmplx operator = (const cmplx& a) { this->x = a.x; this->y = a.y; return *this; }
 cmplx operator + (const cmplx& b) { return cmplx(this->x + b.x, this->y + b.y); }
 cmplx operator - (const cmplx& b) { return cmplx(this->x - b.x, this->y - b.y); }
 cmplx operator * (const T& num) { return cmplx(this->x * num, this->y * num); }
 cmplx operator / (const T& num) { return cmplx(this->x / num, this->y / num); }
 cmplx operator * (const cmplx& b) {
  return cmplx(this->x * b.x - this->y * b.y, this->y * b.x + this->x * b.y);
 }
 cmplx operator / (const cmplx& b) {
  cmplx temp(b.x, -b.y); cmplx n = (*this) * temp;
  return n / (b.x * b.x + b.y * b.y);
 }
};
typedef cmplx<double> COMPLEX;  //change this to long double in case of error
COMPLEX W[LIM], invW[LIM];
void preCompute() {
 for(int i = 0; i < LIM/2; ++i) {
  //change this to long double in case of error
  double ang = 2 * PI * i / LIM;
  double _cos = cos(ang), _sin = sin(ang);
  W[i] = COMPLEX(_cos, _sin);
  invW[i] = COMPLEX(_cos, -_sin);
 }
}
void FFT(COMPLEX *a, int n, bool invert = false) {
 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) {
  for(int i = 0; i < n; i += len) {
   int ind = 0, add = LIM/len;
   for(int j = 0; j < len/2; ++j) {
    COMPLEX u = a[i+j];
    COMPLEX v = a[i+j+len/2] * (invert ? invW[ind] : W[ind]);
    a[i+j] = u + v;
    a[i+j+len/2] = u - v;
    ind += add;
   }
  }
 }
 if (invert) for(int i = 0; i < n; ++i) a[i] = a[i]/n;
}
COMPLEX f[LIM], g[LIM], ff[LIM], gg[LIM];
// c[k] = sum_{i=0}^k a[i] b[k-i] % MOD
vector<int> multiply(vector<int> &A, vector<int> &B) {
 int n1 = A.size(), n2 = B.size();
 int final_size = n1 + n2 - 1, N = 1;
 while(N < final_size) N <<= 1;
 vector<int> C(final_size, 0);
 int SQRTMOD = (int)sqrt(MOD) + 10;
 for(int i = 0; i < N; ++i) f[i] = COMPLEX(), g[i] = COMPLEX();
 for(int i = 0; i < n1; ++i) f[i] = COMPLEX(A[i]%SQRTMOD, A[i]/SQRTMOD);
 for(int i = 0; i < n2; ++i) g[i] = COMPLEX(B[i]%SQRTMOD, B[i]/SQRTMOD);
 FFT(f, N), FFT(g, N);
 COMPLEX X, Y, a1, a2, b1, b2;
 for(int i = 0; i < N; ++i) {
  X = f[i], Y = f[(N-i)%N].conj();
  a1 = (X + Y) * COMPLEX(0.5, 0);
  a2 = (X - Y) * COMPLEX(0, -0.5);
  X = g[i], Y = g[(N-i)%N].conj();
  b1 = (X + Y) * COMPLEX(0.5, 0);
  b2 = (X - Y) * COMPLEX(0, -0.5);
  ff[i] = a1 * b1 + a2 * b2 * COMPLEX(0, 1);
  gg[i] = a1 * b2 + a2 * b1;
 }
 FFT(ff, N, 1), FFT(gg, N, 1);
 for(int i = 0; i < final_size; ++i) {
  long long x = (LL)(ff[i].get_real() + 0.5);
  long long y = (LL)(ff[i].get_img() + 0.5) % MOD;
  long long z = (LL)(gg[i].get_real() + 0.5);
  C[i] = (x + (y * SQRTMOD + z) % MOD * SQRTMOD) % MOD;
 }
 return C;
}
};
using namespace FFTMOD;
//Just call preCompute() in main() before usage
```

3.9 Implementation of FHWT

```
void FWHT(vector<int> &v, bool inverse){
 //note size of v must be a power of 2 mandatorily.
 int deg = v.size();
 for(int len = 1; len * 2 <= deg; len <<= 1){
  for(int i = 0; i < deg; i += len * 2){
   for(int j = 0; j < len; j++){
    int a = v[i+j];
    int b = v[i+j+len];
    v[i+j] = (a+b)%mod;
    v[i+j+len] = (a+mod-b)%mod;
   }
  }
 }
 if (inverse) {
  int inv = power(deg, -1);
  for (int i = 0; i < deg; i++)
   v[i] = (1LL * v[i] * inv)%mod;
 }
}
/*******FHWT****
1) Apply FHWT without taking inverse
2) Depending upon the size of subset required, do v[i] = power(v[i], n), where n is the size of subset required
3) Take inverse FHWT to get the required answer.
****************/
/*********************
Ref :- https://www.hackerearth.com/problem/algorithm/submatrix-queries-7e459f97/editorial/
```

```
If you have a hard time understanding the editorial, this may help you.
Let's talk a little about FFT (Fast Fourier Transform) first. FFT aims to multiply two n-degree polynomials A and B in nlogn using these steps:
1) Calculate FFT of A and B, let them be A' and B'.
2) Calculate array C' where C'[i] = A'[i]*B'[i].
3) Calculate inverse FFT of C' to get the answer (A*B)=C.
Assuming that C[i] = coefficient of x^i in A*B, then the procedure above adds A[j]*B[k] to C[j+k] for every j and k.
Now returning to the question, if we observe carefully, we notice that we want to modify the procedure above to add Cnt[j]*Cnt[k]*Cnt[m] to C[j AND k AND m] for every j, k, m, where Cnt[i] = count of value i in range from l to r. So basically we can imagine Cnt as a polynomial where Cnt[i] is coefficient of x^i which is the count of value i in range from l to r.
Fast Walsh Hadamard transform is a variation of FFT which can be used with two polynomials A and B to add A[j]*B[k] to C[j AND k] for every j and k. So what we want to do with Cnt, is to calculate Cnt^3, which adds Cnt[j]*Cnt[k]*Cnt[m] to C[j AND k AND m] for every j, k, m. That is, C[i] = number of all triplets which have bitwise AND = i. So our answer is obviously C[X]. The steps to do this are very close to FFT:
1) Calculate Fast Walsh Hadamard transform of Cnt, let it be Cnt'.
2) Calculate array C' where C'[i] = Cnt'[i]^3.
3) Calculate inverse Fast Walsh Hadamard transform of C' to get C.
**********************/
```

3.10 Implementation of FHWT for AND operator

```
void FWHT(vector<int> &P, bool inverse){
  //note size of v must be a power of 2 mandatorily.
  int len, i, j, u, v;
  for (len = 1; 2 * len <= M; len <<= 1) {
   for (i = 0; i < M; i += 2 * len) {
    for (j = 0; j < len; j++) {
     u = P[i + j];
     v = P[i + len + j];
     if (!inverse) {
      P[i + j] = v;
      P[i + len + j] = (u + v) % mod;
     } else {
      P[i + j] = (-u + v + mod) % mod;
      P[i + len + j] = u;
     }
    }
   }
  }
}
/**********************
Ref :- https://www.hackerearth.com/problem/algorithm/submatrix-queries-7e459f97/editorial/
If you have a hard time understanding the editorial, this may help you.
Let's talk a little about FFT (Fast Fourier Transform) first. FFT aims to multiply two n-degree polynomials A and B in nlogn using these steps:
1) Calculate FFT of A and B, let them be A' and B'.
2) Calculate array C' where C'[i] = A'[i]*B'[i].
3) Calculate inverse FFT of C' to get the answer (A*B)=C.
Assuming that C[i] = coefficient of x^i in A*B, then the procedure above adds A[j]*B[k] to C[j+k] for every j and k.
Now returning to the question, if we observe carefully, we notice that we want to modify the procedure above to add Cnt[j]*Cnt[k]*Cnt[m] to C[j AND k AND m] for every j, k, m, where Cnt[i] = count of value i in range from l to r. So basically we can imagine Cnt as a polynomial where Cnt[i] is coefficient of x^i which is the count of value i in range from l to r.
Fast Walsh Hadamard transform is a variation of FFT which can be used with two polynomials A and B to add A[j]*B[k] to C[j AND k] for every j and k. So what we want to do with Cnt, is to calculate Cnt^3, which adds Cnt[j]*Cnt[k]*Cnt[m] to C[j AND k AND m] for every j, k, m. That is, C[i] = number of all triplets which have bitwise AND = i. So our answer is obviously C[X]. The steps to do this are very close to FFT:
1) Calculate Fast Walsh Hadamard transform of Cnt, let it be Cnt'.
2) Calculate array C' where C'[i] = Cnt'[i]^3.
3) Calculate inverse Fast Walsh Hadamard transform of C' to get C.
**********************/
```

3.11 Implementation of FHWT - tourist

```
//references taken :- https://www.hackerearth.com/challenges/competitive/june-circuits-19/algorithm/xor-paths-dd39904a/submission/27540843/
namespace fwht {
template<typename T>
void hadamard(vector<T> &a) {
 int n = a.size();
 for (int k = 1; k < n; k <<= 1) {
  for (int i = 0; i < n; i += 2 * k) {
   for (int j = 0; j < k; j++) {
    T x = a[i + j];
    T y = a[i + j + k];
    a[i + j] = x + y;
    a[i + j + k] = x - y;
   }
  }
 }
}
template<typename T>
vector<T> multiply(vector<T> a, vector<T> b) {
 int eq = (a == b);
 int n = 1;
 while (n < (int) max(a.size(), b.size())) {
  n <<= 1;
 }
 a.resize(n);
 b.resize(n);
 hadamard(a);
 if (eq) b = a; else hadamard(b);
 for (int i = 0; i < n; i++) {
  a[i] *= b[i];
 }
 hadamard(a);
 T q = 1 / static_cast<T>(n);
 for (int i = 0; i < n; i++) {
  a[i] *= q;
 }
 return a;
}
} // namespace fwht
/*****USAGE*********
 vector<Mint> cc(1 << 17, 0);
 for (int i = 0; i < n; i++) {
  cc[mark[i]]++;
 }
 cc = fwht::multiply(cc, cc);
*******************/
/**********************
Ref :- https://www.hackerearth.com/problem/algorithm/submatrix-queries-7e459f97/editorial/
If you have a hard time understanding the editorial, this may help you.
Let's talk a little about FFT (Fast Fourier Transform) first. FFT aims to multiply two n-degree polynomials A and B in nlogn using these steps:
1) Calculate FFT of A and B, let them be A' and B'.
2) Calculate array C' where C'[i] = A'[i]*B'[i].
3) Calculate inverse FFT of C' to get the answer (A*B)=C.
Assuming that C[i] = coefficient of x^i in A*B, then the procedure above adds A[j]*B[k] to C[j+k] for every j and k.
Now returning to the question, if we observe carefully, we notice that we want to modify the procedure above to add Cnt[j]*Cnt[k]*Cnt[m] to C[j AND k AND m] for every j, k, m, where Cnt[i] = count of value i in range from l to r. So basically we can imagine Cnt as a polynomial where Cnt[i] is coefficient of x^i which is the count of value i in range from l to r.
Fast Walsh Hadamard transform is a variation of FFT which can be used with two polynomials A and B to add A[j]*B[k] to C[j AND k] for every j and k. So what we want to do with Cnt, is to calculate Cnt^3, which adds Cnt[j]*Cnt[k]*Cnt[m] to C[j AND k AND m] for every j, k, m. That is, C[i] = number of all triplets which have bitwise AND = i. So our answer is obviously C[X]. The steps to do this are very close to FFT:
1) Calculate Fast Walsh Hadamard transform of Cnt, let it be Cnt'.
2) Calculate array C' where C'[i] = Cnt'[i]^3.
3) Calculate inverse Fast Walsh Hadamard transform of C' to get C.
**********************/
```

3.12 Discrete Logarithm

```
// (a ^ x) % m = b, the below finds the value of x
int discreteLog(int a, int b, int m)
{
 a %= m, b %= m;
 if(b == 1)
  return 0;
 int cnt = 0;
```

```
 long long t = 1;
 for(int curg=__gcd(a, m);curg!=1;curg=__gcd(a, m))
 {
  if(b % curg)
   return -1;
  b /= curg, m /= curg, t = (t * a / curg) % m;
  cnt++;
  if(b == t)
   return cnt;
 }
 gp_hash_table<int, int> hash;
 int mid = ((int)sqrt(1.0 * m) + 1);
 long long base = b;
 for(int i=0;i<mid;i++)
 {
  hash[base] = i;
  base = base * a % m;
 }
 base = pow(a, mid, m);
 long long cur = t;
 for(int i=1;i<=mid+1;i++)
 {
  cur = cur * base % m;
  auto it = hash.find(cur);
  if(it != hash.end())
   return i * mid - it->second + cnt;
 }
}
```

3.13 Factorisation and Primality Testing

```
namespace factorisation {
int MAX = 1000001;
vector<int> lp, primes;
void init() {
 lp.resize(MAX);
 lp[1] = 1;
 for (int i = 2; i < MAX; ++i) {
  if (lp[i] == 0) {
   lp[i] = i;
   primes.push_back(i);
  }
  for (int j = 0; j < primes.size() && primes[j] <= lp[i]; ++j) {
   int x = i * primes[j];
   if (x >= MAX) break;
   lp[x] = primes[j];
  }
 }
}
long long Rand() {
 return rand()*(1ll<<48)+rand()*(1ll<<32)+rand()*(1ll<<16)+rand();
}
/*If getting TLE comment below function and use below one but
it can have precision error but chances are almost negligible
*/
long long mulmod(long long a, long long b, long long m) {
 // long long i, res = 0;
 // for(i = 1; i <= b; i*=2) {
 //  if (b&i) {
 //   res += a;
 //   if (res >= m) res -= m;
 //  }
 //  a += a;
 //  if (a >= m) a -= m;
 // }
 // return res;
 long long q = (long long)(((long double)a*(long double)b)/(long double)m);
 long long r = a*b - q*m;
 if (r > m) r %= m;
 if (r < 0) r += m;
 return r;
}
long long power(long long a, long long n, long long m) {
 long long x = 1, y = a;
 while(n) {
  if (n & 1) x = mulmod(x, y, m);
  y = mulmod(y, y, m);
  n >>= 1;
 }
 return x;
}
bool witness(long long a, long long n) {
 long long x, y, u = n - 1, t = 0;
 while(u % 2 == 0) {
  u >>= 1;
  t += 1;
 }
 x = power(a, u, n);
 while(t--) {
  y = x;
  x = power(x, 2, n);
  if(x == 1 && y != 1 && y != n-1)
   return 1;
 }
 return x != 1;
}
bool miller_rabin(long long n) {
 if (n < MAX) return lp[n] == n;
 if (witness(28087,n)) return 0;
 return 1;
}
long long absl(long long x) {
 return (x < 0 ? -x : x);
}
int _c = 1;
long long func(long long x,long long n) {
 long long res = mulmod(x, x, n) + _c;
 return (res >= n ? res % n : res);
}
long long go(long long n) {
 long long x, y, d = 1;
 x = y = rand();
 if (x >= n) x %= n, y %= n;
 while(d == 1) {
  x = func(x, n);
  y = func(func(y, n), n);
  d = __gcd(absl(y - x), n);
 }
 if (d != n) return d;
 return d;
}
void pollard_rho(long long n, int& m, long long s[]){
 long long x;
 if (n == 1) return ;
 if (n < MAX) {
  while (n != 1) {
   int p = lp[n];
   while(n % p == 0) {
    n /= p;
    s[m++] = p;
   }
  }
  return ;
 }
 while (!miller_rabin(n)) {
  for (_c = 1, x = n; x == n; _c = 1 + rand()%(n-1)) {
   x = go(n);
  }
  if(x < 0) break;
  n /= x;
  pollard_rho(x, m, s);
 }
 if(n > 1) s[m++] = n;
}
vector<pair<long long,int>> factorise(long long n) {
 vector<long long> temp;
 while(n % 2 == 0) {
  temp.push_back(2);
  n >>= 1;
 }
 int m = 0;
 long long s[70];
 pollard_rho(n, m, s);
 for(int i = 0; i < m; ++i) {
  temp.push_back(s[i]);
 }
 sort(temp.begin(), temp.end());
 vector<pair<long long,int>> ans;
 for(int i = 0; i < temp.size(); ++i) {
  int j = i, e = 0;
  while(j < temp.size() && temp[j] == temp[i]) {
   e += 1;
   j += 1;
  }
  ans.push_back({temp[i], e});
  i = j - 1;
 }
 return ans;
}
}
using namespace factorisation;
```

3.14 Factorisation and Primality Testing from tourist

```
namespace factorizer {
/*template <typename T>
struct FactorizerVarMod { static T value; };
template <typename T>
T FactorizerVarMod<T>::value;*/
bool IsPrime(uint64_t n) {
 if (n < 2) {
  return false;
 }
 vector<uint32_t> small_primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
 for (uint32_t x : small_primes) {
  if (n == x) {
   return true;
  }
  if (n % x == 0) {
   return false;
  }
 }
 if (n < 31 * 31) {
  return true;
 }
 uint32_t s = __builtin_ctzll(n - 1);
 uint64_t d = (n - 1) >> s;
 function<bool(uint64_t)> witness = [&n, &s, &d](uint64_t a) {
  uint64_t cur = 1, p = d;
  while (p > 0) {
   if (p & 1) {
    cur = (__uint128_t) cur * a % n;
   }
   a = (__uint128_t) a * a % n;
   p >>= 1;
  }
  if (cur == 1) {
   return false;
  }
  for (uint32_t r = 0; r < s; r++) {
   if (cur == n - 1) {
    return false;
   }
   cur = (__uint128_t) cur * cur % n;
  }
  return true;
 };
 vector<uint64_t> bases_64bit = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
 for (uint64_t a : bases_64bit) {
  if (a % n == 0) {
   return true;
  }
  if (witness(a)) {
   return false;
  }
 }
 return true;
}
vector<int> least = {0, 1};
vector<int> primes;
int precalculated = 1;
void RunLinearSieve(int n) {
 n = max(n, 1);
 least.assign(n + 1, 0);
 primes.clear();
 for (int i = 2; i <= n; i++) {
  if (least[i] == 0) {
```

```
   least[i] = i;
   primes.push_back(i);
  }
  for (int x : primes) {
   if (x > least[i] || i * x > n) {
    break;
   }
   least[i * x] = x;
  }
 }
 precalculated = n;
}
void RunSlowSieve(int n) {
 n = max(n, 1);
 least.assign(n + 1, 0);
 for (int i = 2; i * i <= n; i++) {
  if (least[i] == 0) {
   for (int j = i * i; j <= n; j += i) {
    if (least[j] == 0) {
     least[j] = i;
    }
   }
  }
 }
 primes.clear();
 for (int i = 2; i <= n; i++) {
  if (least[i] == 0) {
   least[i] = i;
   primes.push_back(i);
  }
 }
 precalculated = n;
}
void RunSieve(int n) {
 RunLinearSieve(n);
}
template <typename T>
vector<pair<T, int>> MergeFactors(const vector<pair<T, int>>& a, const vector<pair<T, int>>& b) {
 vector<pair<T, int>> c;
 int i = 0;
 int j = 0;
 while (i < (int) a.size() || j < (int) b.size()) {
  if (i < (int) a.size() && j < (int) b.size() && a[i].first == b[j].first) {
   c.emplace_back(a[i].first, a[i].second + b[j].second);
   ++i;
   ++j;
   continue;
  }
  if (j == (int) b.size() || (i < (int) a.size() && a[i].first < b[j].first)) {
   c.push_back(a[i++]);
  } else {
   c.push_back(b[j++]);
  }
 }
 return c;
}
template <typename T>
vector<pair<T, int>> RhoC(const T& n, const T& c) {
 if (n <= 1) {
  return {};
 }
 if ((n & 1) == 0) {
  return MergeFactors({{2, 1}}, RhoC(n / 2, c));
 }
 if (IsPrime(n)) {
  return {{n, 1}};
 }
 int64_t x = 2;
 int64_t saved = 2;
 T power = 1;
 T lam = 1;
 while (true) {
  x = ((__int128) x * x + c) % n;
  T g = __gcd((long long) abs(x - saved), n);
  if (g != 1) {
   return MergeFactors(RhoC(g, c + 1), RhoC(n / g, c + 1));
  }
  if (power == lam) {
   saved = x;
   power <<= 1;
   lam = 0;
  }
  lam++;
 }
 return {};
}
template <typename T>
vector<pair<T, int>> Rho(const T& n) {
 return RhoC(n, static_cast<T>(1));
}
template <typename T>
vector<pair<T, int>> Factorize(T x) {
 if (x <= 1) {
  return {};
 }
 if (x <= precalculated) {
  vector<pair<T, int>> ret;
  while (x > 1) {
   if (!ret.empty() && ret.back().first == least[x]) {
    ret.back().second++;
   } else {
    ret.emplace_back(least[x], 1);
   }
   x /= least[x];
  }
  return ret;
 }
 if (x <= static_cast<int64_t>(precalculated) * precalculated) {
  vector<pair<T, int>> ret;
  if (!IsPrime(x)) {
   for (T i : primes) {
    T t = x / i;
    if (i > t) {
     break;
    }
    if (x == t * i) {
     int cnt = 0;
     while (x % i == 0) {
      x /= i;
      cnt++;
     }
     ret.emplace_back(i, cnt);
     if (IsPrime(x)) {
      break;
     }
    }
   }
  }
  if (x > 1) {
   ret.emplace_back(x, 1);
  }
  return ret;
 }
 return Rho(x);
}
template <typename T>
vector<T> BuildDivisorsFromFactors(const vector<pair<T, int>>& factors) {
 vector<T> divisors = {1};
 for (auto& p : factors) {
  int sz = (int) divisors.size();
  for (int i = 0; i < sz; i++) {
   T cur = divisors[i];
   for (int j = 0; j < p.second; j++) {
    cur *= p.first;
    divisors.push_back(cur);
   }
  }
 }
 sort(divisors.begin(), divisors.end());
 return divisors;
}
} // namespace factorizer
```

4 Graph algorithms

4.1 Bellman Ford

```
//Bellman Ford Algo to find the single source shortest distance
//works even for negative cycle graph
//logic a shortest path can be atmost (n-1) edges long. So run n-1 iteration to find
//shortest distance from source to other vertices.
//also, even after n-1 iteration the distance of any vertex decreases, then we are sure
//that graph consists of negative cycle
vector<pair<pair<int, int>, int> > v;//stores edges and wts in graph in form ((edges), wt)
bool bellman_ford(int source){
  for (int i = 0; i <= n ; ++i){
   dis[i] = inf;//assign max distance to all
  }
  dis[source] = 0;
  for (int i = 0; i < edges; ++i){
   cin>>from>>next>>weight;
   v.push_back({{from, next}, weight});
  }
  for (int i = 0; i < n-1; ++i)
  {
   for(auto it: v){
    int x = it.F.F, y = it.F.S;
    int wt = it.S;
    if(dis[y] > dis[x] + wt){
     dis[y] = dis[x] + wt;
    }
   }
  }
  //we do this process for n-1 times since max. times the change in the distance can occur is n-1;
  for(auto it: v){
   int x = it.F.F, y = it.F.S;
   int wt = it.S;
   if(dis[y] > dis[x] + wt){
    return 0;//negative cycle is present
   }
  }
  return 1;//no negative cycle found. dist[i] stores min distance from the source vertex
}
```

4.2 Biconnected components

```
//Ref: https://github.com/bicsi/code_snippets
namespace CBC {
 vector<int> Stack;
 int Depth[kMaxN], Time[kMaxN], Low[kMaxN];
 vector<int> Nodes[kMaxN];
 int Art[kMaxN];
 int n, cbc;
 void Init(int nn) {
  n = nn;
  for(int i = 1; i <= n; ++i) {
   Time[i] = Low[i] = Art[i] = 0;
   Nodes[i].clear();
  }
  cbc = 0;
  Stack.clear();
 }
 void DFS(int node) {
  static int timer;
  Stack.push_back(node);
  Low[node] = Time[node] = ++timer;
  for(auto vec : G[node]) {
   if(!Time[vec]) {
    Depth[vec] = Depth[node] + 1;
    DFS(vec);
    Low[node] = min(Low[node], Low[vec]);
    if(Low[vec] >= Time[node]) {
     auto &To = Nodes[++cbc];
     do {
      To.push_back(Stack.back());
```

```
      Stack.pop_back();
    } while(To.back() != vec);
    To.push_back(node);
   }
  } else if(Depth[vec] < Depth[node] - 1) {
   Low[node] = min(Low[node], Time[vec]);
  }
 }
}
void MakeArticulationPoints() {
 for(int i = 1; i <= cbc; ++i) {
  for(auto node : Nodes[i])
   Art[node] += 1;
 }
 for(int i = 1; i <= n; ++i)
  Art[i] = (Art[i] >= 2);
}
};
```

4.3 Bipartite Graph

```
std::vector<std::vector<int>> adj;
std::vector<int> color;
bool bipartite(int v = 0, int p = -1, int c = 0)
{
  if (p == -1)
    color.assign(adj.size(), -1);
  if (color[v] >= 0)
    return color[v] == c;
  color[v] = c;
  par[v] = p;
  for (auto u : adj[v]) {
    if (u == p)
      continue;
    if (!bipartite(u, v, c ^ 1))
      return false;
  }
  return true;
}
int32_t main(){_
  cin>>n>>m;
  adj.resize(n);
  for (int i = 0; i < m; ++i)
  {
   int x, y;
   cin>>x>>y;
   --x, --y;
   adj[x].pb(y);
   adj[y].pb(x);
  }
  // bipartite();
  return 0;
}
```

4.4 Bipartite Matching Implementation

```
//Ref: https://github.com/bicsi/code_snippets
struct BipartiteMatcher {
 vector<vector<int>> G;
 vector<int> L, R, Viz;
 BipartiteMatcher(int n, int m) :
  G(n), L(n, -1), R(m, -1), Viz(n) {}
 void AddEdge(int a, int b) {
  G[a].push_back(b);
 }
 bool Match(int node) {
  if(Viz[node])
   return false;
  Viz[node] = true;
  for(auto vec : G[node]) {
   if(R[vec] == -1 || Match(R[vec])) {
    L[node] = vec;
    R[vec] = node;
    return true;
   }
  }
  return false;
 }
 int Solve() {
  bool ok = true;
  while(ok) {
   ok = false;
   fill(Viz.begin(), Viz.end(), 0);
   for(int i = 0; i < L.size(); ++i)
    if(L[i] == -1)
     ok |= Match(i);
  }
  int ret = 0;
  for(int i = 0; i < L.size(); ++i)
   ret += (L[i] != -1);
  return ret;
 }
};
```

4.5 Bipartite Matching with Union Find Implementation

```
struct bipartite_union_find {
  vector<int> parent;
  vector<int> size;
  vector<bool> bipartite;
  vector<bool> edge_parity;
  int components = 0;
  bipartite_union_find(int n = 0) {
    if (n > 0)
      init(n);
  }
  void init(int n) {
    parent.resize(n + 1);
    size.assign(n + 1, 1);
    bipartite.assign(n + 1, true);
    edge_parity.assign(n + 1, false);
    components = n;
    for (int i = 0; i <= n; i++)
      parent[i] = i;
  }
  int find(int x) {
    if (x == parent[x])
      return x;
    int root = find(parent[x]);
    edge_parity[x] = edge_parity[x] ^ edge_parity[parent[x]];
    return parent[x] = root;
  }
  // Returns true if x and y are in the same component.
  bool query_component(int x, int y) {
    return find(x) == find(y);
  }
  // Returns the parity status between x and y. Requires that they are in the same component.
  bool query_parity(int x, int y) {
    int x_root = find(x);
    int y_root = find(y);
    assert(x_root == y_root);
    return edge_parity[x] ^ edge_parity[y];
  }
  // Returns {union succeeded, edge consistent with bipartite conditions}.
  pair<bool, bool> unite(int x, int y, bool different = true) {
    int x_root = find(x);
    int y_root = find(y);
    if (x_root == y_root) {
      bool consistent = !(edge_parity[x] ^ edge_parity[y] ^ different);
      if (!consistent)
        bipartite[x_root] = false;
      return {false, consistent};
    }
    bool needed_parity = edge_parity[x] ^ edge_parity[y] ^ different;
    x = x_root;
    y = y_root;
    if (size[x] < size[y])
      swap(x, y);
    parent[y] = x;
    size[x] += size[y];
    bipartite[x] = bipartite[x] && bipartite[y];
    edge_parity[y] = needed_parity;
    components--;
    return {true, true};
  }
  pair<bool, bool> add_different_edge(int x, int y) {
    return unite(x, y, true);
  }
  pair<bool, bool> add_same_edge(int x, int y) {
    return unite(x, y, false);
  }
};
/**
bipartite_union_find UF;
**/
```

4.6 Cycle Checking in Directed Graph

```
bool vis[N], OnStack[N];
bool iscyclic(int ver){
  vis[ver] = true;
  OnStack[ver] = true;
  for(auto it: v[ver]){
   //adjust as per if graph has weight or not.
   if(!vis[it.fi]){
    if(iscyclic(it.fi)){
     return true;
    }
   }
   else{
    if(OnStack[it.fi]){
     return true;
    }
   }
  }
  OnStack[ver] = false;
  return false;
}
bool cycle(){
  memset(vis, 0, sizeof vis);
  memset(OnStack, 0, sizeof OnStack);
  for (int i = 1; i <= n; ++i)if(iscyclic(i)){
    return true;
  }
  return false;
}
```

4.7 Implementation of lca using Binary lifting

```
struct log_lca {
  int n = 0;
  vector<int> parent, depth;
  vector<vector<int>> adj;
  vector<vector<int>> ancestor;
  log_lca(int _n = 0) {
    init(_n);
  }
  void init(int _n) {
    n = _n;
    parent.resize(n);
    depth.resize(n);
    adj.assign(n, {});
  }
  static int largest_bit(int x) {
    return 31 - __builtin_clz(x);
  }
  void add_edge(int a, int b) {
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
```

```
  void dfs(int node, int par) {
    depth[node] = par < 0 ? 0 : depth[par] + 1;
    parent[node] = par;
    for (int neighbor : adj[node])
      if (neighbor != par)
        dfs(neighbor, node);
  }
  void build() {
    dfs(0, -1);
    ancestor.assign(largest_bit(n) + 1, vector<int>(n));
    ancestor[0] = parent;
    for (int k = 0; k < largest_bit(n); k++)
      for (int i = 0; i < n; i++)
        ancestor[k + 1][i] = ancestor[k][i] < 0 ? -1 : ancestor[k][ancestor[k][i]];
  }
  int get_lca(int a, int b) const {
    if (depth[a] > depth[b])
      swap(a, b);
    int difference = depth[b] - depth[a];
    for (int k = 0; 1 << k <= difference; k++)
      if (difference & 1 << k)
        b = ancestor[k][b];
    if (a == b)
      return a;
    assert(a != b && depth[a] == depth[b]);
    for (int k = largest_bit(depth[a]); k >= 0; k--)
      if (ancestor[k][a] != ancestor[k][b]) {
        a = ancestor[k][a];
        b = ancestor[k][b];
      }
    assert(parent[a] == parent[b]);
    return parent[a];
  }
  int get_dist(int a, int b) const {
    return depth[a] + depth[b] - 2 * depth[get_lca(a, b)];
  }
  int get_kth_ancestor(int v, int k) const {
    for (int i = 0; 1 << i <= k; i++)
      if (k & 1 << i) {
        v = ancestor[i][v];
        if (v < 0)
          break;
      }
    return v;
  }
  int get_kth_node_on_path(int a, int b, int k) const {
    int anc = get_lca(a, b);
    int first_half = depth[a] - depth[anc];
    int second_half = depth[b] - depth[anc];
    assert(0 <= k && k <= first_half + second_half);
    if (k < first_half)
      return get_kth_ancestor(a, k)
      ;
    else
      return get_kth_ancestor(b, first_half + second_half - k);
  }
};
int N;
log_lca lca;
vector<int> center = {-1, -1};
int diameter = 0;
void add_to_center(int p) {
  if (center[0] < 0) {
    center = {p, p};
    return;
  }
  int dist0 = lca.get_dist(p, center[0]);
  int dist1 = lca.get_dist(p, center[1]);
  if (dist0 + dist1 <= diameter)
    return;
  if (dist0 < dist1) {
    swap(dist0, dist1);
    swap(center[0], center[1]);
  }
  int new_diameter = diameter / 2 + dist0;
  center[0] = lca.get_kth_node_on_path(center[0], p, new_diameter / 2 - diameter / 2);
  center[1] = lca.get_kth_node_on_path(center[0], p, new_diameter % 2);
  diameter = new_diameter;
}
int main() {
  scanf("%d", &N);
  lca.init(N);
  for (int i = 0; i < N - 1; i++) {
    int a, b;
    scanf("%d %d", &a, &b);
    a--; b--;
    lca.add_edge(a, b);
  }
  lca.build();
  for (int i = 0; i < N; i++) {
    int p;
    scanf("%d", &p);
    p--;
    add_to_center(p);
    printf("%d%c", min(center[0], center[1]) + 1, i < N - 1 ? ' ' : '\n');
  }
}
```

4.8 MaxFlow Algorithm

```
template<typename T>
struct MaxFlowMinCost {
  struct Edge {
    T cap, flow, cost;
    int to;
  };
  vector<Edge> Edges;
  vector<vector<int>> G;
  int src, dest;
  vector<int> Parent, ParentEdge, InQ;
  vector<T> Dist;
  MaxFlowMinCost& Initialize(int n, int m = 0) {
    G.clear();
    G.resize(n);
    Edges.clear();
    Edges.reserve(m);
    Parent.resize(n);
    ParentEdge.resize(n);
    Dist.resize(n);
    InQ.resize(n);
    return *this;
  }
  void _addEdge(int from, int to, T cap, T cost) {
    int ei = Edges.size();
    Edges.push_back(Edge {cap, 0, cost, to});
    G[from].push_back(ei);
  }
  MaxFlowMinCost& AddEdge(int from, int to, T cap, T cost) {
    _addEdge(from, to, cap, cost);
    _addEdge(to, from, 0, -cost);
    return *this;
  }
  MaxFlowMinCost& SetSourceSink(int src, int dest) {
    this->src = src; this->dest = dest;
    return *this;
  }
  bool Bellman() {
    static queue<int> Q;
    fill(Dist.begin(), Dist.end(), numeric_limits<T>::max());
    fill(Parent.begin(), Parent.end(), -1);
    fill(InQ.begin(), InQ.end(), 0);
    Dist[src] = 0;
    Q.push(src);
    while(!Q.empty()) {
      int node = Q.front();
      Q.pop();
      InQ[node] = 0;
      if(Parent[node] != -1 && InQ[Parent[node]])
        continue;
      for(auto ei : G[node]) {
        auto &e = Edges[ei];
        if(e.flow < e.cap && Dist[e.to] > Dist[node] + e.cost) {
          Dist[e.to] = Dist[node] + e.cost;
          Parent[e.to] = node;
          ParentEdge[e.to] = ei;
          if(!InQ[e.to]) {
            Q.push(e.to);
            InQ[e.to] = 1;
          }
        }
      }
    }
    return Parent[dest] != -1;
  }
  pair<T, T> Compute() {
    T flow = 0, cost = 0;
    while(Bellman()) {
      T m = numeric_limits<T>::max();
      for(int node = dest; node != src; node = Parent[node]) {
        int ei = ParentEdge[node];
        m = min(m, Edges[ei].cap - Edges[ei].flow);
      }
      for(int node = dest; node != src; node = Parent[node]) {
        int ei = ParentEdge[node];
        Edges[ei].flow += m;
        Edges[ei ^ 1].flow -= m;
        cost += Edges[ei].cost * m;
      }
      flow += m;
    }
    return {flow, cost};
  }
};
```

4.9 Prims Algorithm

```
const int MAX = 1e4 + 5;
typedef pair<long long, int> PII;
bool marked[MAX];
vector <PII> adj[MAX];
long long prim(int x)
{
  std::priority_queue<PII, vector<PII>, greater<PII> > Q;
  int y;
  long long minimumCost = 0;
  PII p;
  Q.push(make_pair(0, x));
  while(!Q.empty())
  {
    // Select the edge with minimum weight
    p = Q.top();
    Q.pop();
    x = p.second;
    // Checking for cycle
    if(marked[x] == true)
      continue;
    minimumCost += p.first;
    marked[x] = true;
    for(int i = 0;i < adj[x].size();++i)
    {
      y = adj[x][i].second;
      if(marked[y] == false){
        Q.push(adj[x][i]);
      }
    }
  }
  return minimumCost;
}
int nodes, edges, x, y;
long long weight, minimumCost;
cin >> nodes >> edges;
for(int i = 0;i < edges;++i)
{
  cin >> x >> y >> weight;
```

```
  adj[x].push_back(make_pair(weight, y));
  adj[y].push_back(make_pair(weight, x));
}
// Selecting 1 as the starting node
minimumCost = prim(1);
cout << minimumCost << endl;
```

4.10 SCC Implementation

```
// Strongly connected components decomposition algorithm
//
// Usage:
// Initialize with constructor as the number of nodes
// and add edges accordingly with the AddEdge(u, v) method.
// Then just run Reduce()
//
// ATTENTION: The reduced DAG may contain duplicate edges
struct SCCWrapper {
 int n;
 vector<vector<int>> G, G_T;
 vector<bool> Viz;
 vector<int> Stack;
 vector<int> SCC;       // SCC[i] gives the scc id of node i
 vector<vector<int>> Nodes; // Nodes[i] is the list of nodes in scc i
 vector<vector<int>> G_R;  // The reduced DAG (MAY CONTAIN DUPLICATE EDGES)
 int scc;            // Strongly connected component count
 SCCWrapper(int n) : n(n), Viz(n), G(n), G_T(n), SCC(n) {
  Stack.reserve(n);
  scc = 0;
 };
 void AddEdge(int a, int b) {
  G[a].push_back(b);
  G_T[b].push_back(a);
 }
 void dfs_forward(int node) {
  Viz[node] = true;
  for(auto vec : G[node]) {
   if(!Viz[vec])
    dfs_forward(vec);
  }
  Stack.push_back(node);
 }
 void dfs_backward(int node) {
  Viz[node] = true;
  SCC[node] = scc;
  Nodes[scc].push_back(node);
  for(auto vec : G_T[node]) {
   if(!Viz[vec])
    dfs_backward(vec);
  }
 }
 void Reduce() {
  fill(Viz.begin(), Viz.end(), 0);
  for(int i = 0; i < n; ++i)
   if(!Viz[i])
    dfs_forward(i);
  assert(sz(Stack) == n);
  fill(Viz.begin(), Viz.end(), 0);
  for(int i = n - 1; i >= 0; --i)
   if(!Viz[Stack[i]]) {
    Nodes.push_back(vector<int>());
    dfs_backward(Stack[i]);
    ++scc;
   }
  G_R.resize(scc);
  for(int i = 0; i < n; ++i) {
   for(auto vec : G[i])
    G_R[SCC[i]].push_back(SCC[vec]);
  }
 }
};
```

4.11 Topological Sorting

```
int indeg[N];
vector<int> topo; //Stores lexicographically smallest toposort
vector<int> g[N];
bool toposort() //Returns 1 if there exists a toposort, 0 if there is a cycle
{
 priority_queue<int, vector<int>, greater<int> > pq;
 for(int i=1;i<=n;i++)
  for(auto &it:g[i])
   indeg[it]++;
 for(int i=1;i<=n;i++)
 {
  if(!indeg[i])
   pq.push(i);
 }
 while(!pq.empty())
 {
  int u=pq.top();
  pq.pop();
  topo.push_back(u);
  for(auto &v:g[u])
  {
   indeg[v]--;
   if(!indeg[v])
    pq.push(v);
  }
 }
 if(topo.size()<n)
  return 0;
 return 1;
}
//Problem 1: https://www.spoj.com/problems/TOPOSORT/
//Solution 1: http://p.ip.fi/RTRG
```

4.12 Union Find

```
struct UnionFind
{
  vector<int> parent, size;
  UnionFind(int n){
   parent.resize(n+1);
   size.resize(n+1);
   for (int i = 1; i <= n; ++i)
   {
     parent[i] = i;
     size[i] = 1;
   }
  }
  int find(int x){
   while(x != parent[x]){
     parent[x] = parent[parent[x]];
     x = parent[x];
   }
   return x;
  }
  bool unite(int x, int y){
   x = find(x), y = find(y);
   if(x == y)
     return false;
   if(size[x] < size[y])swap(x, y);
   size[x] += size[y];
   parent[y] = x;
   return true;
  }
  int getSize(int x){
   x = find(x);
   return size[x];
  }
};
/*** USAGE
UnionFind S(n);
****/
```

5 Data structures

5.1 Binary Indexed Tree

```
template <typename T>
struct fenwick_tree {
private:
 int n;
 vector <T> a;
public:
 void initialize(int k) {
  n=k;
  a.assign(n+1,0);
 }
 void update(int pos, T val) {
  for(;pos<=n;pos+=pos&(-pos)) a[pos]+=val;
 }
 T query(int pos) {
  T ans=0;
  for(;pos>=1;pos-=pos&(-pos)) ans+=a[pos];
  return ans;
 }
 T get(int from, int to) {
  return query(to)-query(from-1);
 }
 int lower_bound(T x){
  int index = 0;
  int sum = 0;
  for (int i = (log2(n) + 1); i >= 0; --i)
  {
   if((index+(1<<i)) <= n and (sum + a[index+(1<<i)]) < x){
    sum += a[index+(1<<i)];
    index += (1<<i);
   }
  }
  return index+1;
 }
};
fenwick_tree <int> bit;
```

5.2 Binary Indexed Tree - 2D

```
const int MAX = 1005;
int bit[MAX][MAX];
//Complexity is O(log^2 n)
void update(int x, int y, int val) {
 while (x < MAX) {
  int v = y;
  while (v < MAX) {
   bit[x][v] += val;
   v += (v & -v);
  }
  x += (x & -x);
 }
}
//Complexity is O(log^ 2)
int query(int x, int y) {
 int sum = 0;
 while (x > 0) {
  int v = y;
  while (v > 0) {
   sum += bit[x][v];
   v -= (v & -v);
  }
  x -= (x & -x);
 }
 return sum;
}
//Sum of values within rectangle (x1, y1) to (x2, y2), both inclusive
//(x2, y2) lies above or right to (x1, y1) else sum is 0
//Complexity is 4*O(logn * logm)
int sum(int x1, int y1, int x2, int y2) {
 return query(x2, y2) + query(x1-1, y1-1) - query(x2, y1-1) - query(x1-1, y2);
}
```

5.3 Query over range using SQRT Decomposition

```
//Implementation References :- Neal Wu's submissions on https://codeforces.com/contest/1093/submission/47229079
#include <algorithm>
#include <cassert>
#include <cmath>
#include <iostream>
#include <vector>
using namespace std;
// search_buckets provides two operations on an array:
// 1) set array[i] = x
// 2) count how many i in [start, end) satisfy array[i] < value
// Both operations take sqrt(N log N) time. Amazingly, because of the cache efficiency this is faster than the
// (log N)^2 algorithm until N = 2-5 million.
template<typename T>
struct search_buckets {
  // values are just the values in order. buckets are sorted in segments of BUCKET_SIZE (last segment may be smaller)
  int N, BUCKET_SIZE;
  vector<T> values, buckets;
  search_buckets(const vector<T> &initial = {}) {
    init(initial);
  }
  int get_bucket_end(int bucket_start) const {
    return min(bucket_start + BUCKET_SIZE, N);
  }
  void init(const vector<T> &initial) {
    values = buckets = initial;
    N = values.size();
    BUCKET_SIZE = 3 * sqrt(N * log(N + 1)) + 1;
    cerr << "Bucket size: " << BUCKET_SIZE << endl;
    for (int start = 0; start < N; start += BUCKET_SIZE)
      sort(buckets.begin() + start, buckets.begin() + get_bucket_end(start));
  }
  int bucket_less_than(int bucket_start, T value) const {
    auto begin = buckets.begin() + bucket_start;
    auto end = buckets.begin() + get_bucket_end(bucket_start);
    return lower_bound(begin, end, value) - begin;
  }
  int less_than(int start, int end, T value) const {
    int count = 0;
    int bucket_start = start - start % BUCKET_SIZE;
    int bucket_end = min(get_bucket_end(bucket_start), end);
    if (start - bucket_start < bucket_end - start) {
      while (start > bucket_start)
        count -= values[--start] < value;
    } else {
      while (start < bucket_end)
        count += values[start++] < value;
    }
    if (start == end)
      return count;
    bucket_start = end - end % BUCKET_SIZE;
    bucket_end = get_bucket_end(bucket_start);
    if (end - bucket_start < bucket_end - end) {
      while (end > bucket_start)
        count += values[--end] < value;
    } else {
      while (end < bucket_end)
        count -= values[end++] < value;
    }
    while (start < end && get_bucket_end(start) <= end) {
      count += bucket_less_than(start, value);
      start = get_bucket_end(start);
    }
    assert(start == end);
    return count;
  }
  int prefix_less_than(int n, T value) const {
    return less_than(0, n, value);
  }
  void modify(int index, T value) {
    int bucket_start = index - index % BUCKET_SIZE;
    int old_pos = bucket_start + bucket_less_than(bucket_start, values[index]);
    int new_pos = bucket_start + bucket_less_than(bucket_start, value);
    if (old_pos < new_pos) {
      copy(buckets.begin() + old_pos + 1, buckets.begin() + new_pos, buckets.begin() + old_pos);
      new_pos--;
      // memmove(&buckets[old_pos], &buckets[old_pos + 1], (new_pos - old_pos) * sizeof(T));
    } else {
      copy_backward(buckets.begin() + new_pos, buckets.begin() + old_pos, buckets.begin() + old_pos + 1);
      // memmove(&buckets[new_pos + 1], &buckets[new_pos], (old_pos - new_pos) * sizeof(T));
    }
    buckets[new_pos] = value;
    values[index] = value;
  }
};
int main() {
  int N, M;
  scanf("%d %d", &N, &M);
  vector<int> A(N), B(N);
  vector<int> location(N + 1);
  for (int i = 0; i < N; i++) {
    scanf("%d", &A[i]);
    location[A[i]] = i;
  }
  for (int &b : B) {
    scanf("%d", &b);
    b = location[b];
  }
  search_buckets<int> buckets(B);
  for (int i = 0; i < M; i++) {
    int type;
    scanf("%d", &type);
    if (type == 1) {
      int LA, RA, LB, RB;
      scanf("%d %d %d %d", &LA, &RA, &LB, &RB);
      LA--; LB--;
      printf("%d\n", buckets.less_than(LB, RB, RA) - buckets.less_than(LB, RB, LA));
    } else if (type == 2) {
      int x, y;
      scanf("%d %d", &x, &y);
      x--; y--;
      buckets.modify(x, B[y]);
      buckets.modify(y, B[x]);
      swap(B[x], B[y]);
    } else {
      assert(false);
    }
  }
}
```

5.4 Segmenttree

```
const int MAX = 1e5 + 5;
const int LIM = 3e5 + 5;    // equals 2 * 2^ceil(log2(n))
int a[MAX];
int seg[LIM];
//Complexity: O(1)
//Stores what info. segment[i..j] should store
int combine(int &a, int &b) {
  return a + b;
}
//Complexity: O(n)
void build(int t, int i, int j) {
  if (i == j) {
    //base case : leaf node information to be stored here
    seg[t] = a[i];
    return ;
  }
  int mid = (i + j) / 2;
  build(t*2, i, mid);
  build(t*2 + 1, mid + 1, j);
  seg[t] = combine(seg[2*t], seg[2*t+1]);
}
//Complexity: O(log n)
void update(int t, int i, int j, int x, int y) {
  if (i > x || j < x) {
    return ;
  }
  if (i == j) {
    //base case : leaf node information to be stored here
    seg[t] = y;
    return ;
  }
  int mid = (i + j) / 2;
  update(t*2, i, mid, x, y);
  update(t*2 + 1, mid + 1, j, x, y);
  seg[t] = combine(seg[2*t], seg[2*t+1]);
}
//Complexity: O(log n)
int query(int t, int i, int j, int l, int r) {
  if (l <=i && j <= r) {
    return seg[t];
  }
  int mid = (i + j) / 2;
  if (l <= mid) {
    if (r <= mid) {
      return query(t*2, i, mid, l, r);
    }
    else {
      int a = query(t*2, i, mid, l, r);
      int b = query(t*2 + 1, mid + 1, j, l, r);
      return combine(a, b);
    }
  }
  else {
    return query(t*2 + 1, mid + 1, j, l, r);
  }
}
```

5.5 Segmenttree Lazy Propagation

```
//Segment tree operations: Range update(Lazy propagation) and Range Query
const int MAX = 1e5 + 5;
const int LIM = 3e5 + 5;    // equals 2 * 2^ceil(log2(n))
int a[MAX];
int seg[LIM];
int lazy[LIM];
bool push[LIM];
//Complexity: O(1)
```

```
//Stores what info. segment[i..j] should store
int combine(int &a, int &b) {
 return max(a, b);
}
//Lazy propagation
void propagate(int t, int i, int j) {
 if (push[t]) {
  seg[t] = seg[t] + lazy[t];
  if (i != j) {
   push[t*2] = true;
   push[t*2 + 1] = true;
   lazy[t*2] = lazy[t*2] + lazy[t];
   lazy[t*2 + 1] = lazy[t*2 + 1] + lazy[t];
  }
  push[t] = false;
  lazy[t] = 0;
 }
}
//Complexity: O(n)
void build(int t, int i, int j) {
 push[t] = false;
 lazy[t] = 0;
 if (i == j) {
  //base case : leaf node information to be stored here
  seg[t] = a[i];
  return ;
 }
 int mid = (i + j) / 2;
 build(t*2, i, mid);
 build(t*2 + 1, mid + 1, j);
 seg[t] = combine(seg[2*t], seg[2*t+1]);
}
//Complexity: O(log n)
void update(int t, int i, int j, int l, int r, int x) {
 propagate(t, i, j);
 if (i > r || j < l) {
  return ;
 }
 if (l <= i && j <= r) {
  //base case : leaf node information to be stored here
  lazy[t] += x;
  push[t] = true;
  propagate(t, i, j);
  return ;
 }
 int mid = (i + j) / 2;
 update(t*2, i, mid, l, r, x);
 update(t*2 + 1, mid + 1, j, l, r, x);
 seg[t] = combine(seg[2*t], seg[2*t+1]);
}
//Complexity: O(log n)
int query(int t, int i, int j, int l, int r) {
 propagate(t, i, j);
 if (i > r || j < l) {
  //base case: result of out-of-bound query
  return 0;
 }
 if (l <= i && j <= r) {
  return seg[t];
 }
 int mid = (i + j) / 2;
 if (l <= mid) {
  if (r <= mid) {
   return query(t*2, i, mid, l, r);
  }
  else {
   int a = query(t*2, i, mid, l, r);
   int b = query(t*2 + 1, mid + 1, j, l, r);
   return combine(a, b);
  }
 }
 else {
  return query(t*2 + 1, mid + 1, j, l, r);
 }
}
```

5.6 Implementation of Centroid Decomposition

```
//Ref: https://github.com/bicsi/code_snippets
struct CentroidDecomposer {
 int n;
 vector<set<int>> G;
 vector<int> Size;
 // Depth and parent in the centroid tree
 vector<int> CD, CP;
 CentroidDecomposer(int n) :
  n(n), G(n), Size(n), CD(n), CP(n) {}
 void AddEdge(int a, int b) {
  G[a].insert(b);
  G[b].insert(a);
 }
 int rec_size, rec_centr;
 void ComputeSizeAndCentroid(int node, int par) {
  Size[node] = 1;
  int max_size = 0;
  for(auto vec : G[node])
   if(vec != par) {
    ComputeSizeAndCentroid(vec, node);
    Size[node] += Size[vec];
    max_size = max(max_size, Size[vec]);
   }
  max_size = max(max_size, rec_size - Size[node]);
  if(2 * max_size <= rec_size)
   rec_centr = node;
 }
 void DoUsefulDFS(int node, int par, int cd) {
  Size[node] = 1;
  for(auto vec : G[node])
   if(vec != par) {
    DoUsefulDFS(vec, node, cd);
    Size[node] += Size[vec];
   }
 }
 void RecDecomp(int node, int size, int cp, int cd) {
  // node -> centroid(node)
  rec_size = size;
  ComputeSizeAndCentroid(node, -1);
  node = rec_centr;
  CP[node] = cp;
  CD[node] = cd;
  // You can do whichever DFS you want here
  DoUsefulDFS(node, -1, cd);
  for(auto vec : G[node]) {
   G[vec].erase(node);
   RecDecomp(vec, Size[vec], node, cd + 1);
  }
 }
 void Decompose(int root = 0) {
  RecDecomp(root, n, -1, 0);
  for(auto x : CD)
   assert(x <= __lg(n) + 1);
 }
};
```

5.7 Implementation of Dynamic Segment tree using Trie

```
//more details about dynamic seg tree => https://codeforces.com/blog/entry/69957
//blog1 => https://codeforces.com/blog/entry/19080
//comment of blog1 => https://codeforces.com/blog/entry/19080?#comment-239938
//blog2 => https://www.quora.com/What-are-the-differences-among-dynamic-segment-tree-%E2%80%98implicit-segment-tree%E2%80%99-and-persistent-segment-tree/answer/Egor-Suvorov
//blog3 => https://codeforces.com/blog/entry/60837
//http://p.ip.fi/FG_F
//pointer implementation => https://github.com/ADJA/algos/blob/master/DataStructures/ImplicitSegmentTree.cpp (nice implementation)
//implementation2 => http://ideone.com/0QK4CX
//implementation3 => http://ideone.com/hdI5aX (nice implementation)
//implementation4 => https://ideone.com/bdSh9H
//implementation5 => https://ideone.com/6wyGFR, https://ideone.com/LpAQFl
//implementation4 => http://ideone.com/Oe6jZ2
#include <bits/stdc++.h>
using namespace std;
#define IOS ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
#define endl "\n"
#define int long long
typedef struct data
{
 data* bit[2];
 int cnt = 0;
 int sum = 0;
}trie;
trie* head;
//insert val into the node
//(0000, value) => increment value to 0, 00, 000, 0000
//(0110, value) => increment value to 0, 01, 011, 0110
//(1101, value) => increment value to 1, 11, 110, 1101
//i.e, this function increments the value to all the prefix of the bit representation of x
//function to update value of x.
void insert(int x, int val)
{
 trie* cur = head;
 for(int i = 60; i >= 0; i--)
 {
  int b = (x >> i) & 1;
  if(!cur->bit[b])
   cur -> bit[b] = new trie();
  cur = cur -> bit[b];
  cur->cnt++;
  cur->sum += val;
 }
}
// given x, find the sum of all elements in array with index < x.
int query(int x)
{
 trie* cur = head;
 int ans = 0;
 for(int i = 60; i >= 0; i--)
 {
  int b = (x >> i) & 1;
  if(b == 0 && !cur->bit[b])
   return ans;
  else if(b == 0)
   cur = cur->bit[b];
  else
  {
   if(cur->bit[0] != NULL)
    ans += (cur->bit[0])->sum;// increment count of 0th bit when u move to 1, since
    //we want sum of all elements < x. So if 010, sum(0[0]) + sum(010)
   if(!cur->bit[b])
    return ans;
   cur = cur->bit[b];
  }
 }
 return ans;
}
// this is codenation problem. given x <= 1e18, update a[x] = y
```

```
// query, sum of all elements <= x.
int32_t main()
{
 IOS;
 head = new trie();
 int prev = 0;
 int q, m1 = 1e9, m2 = 1e9;
 cin >> q >> m1 >> m2;
 map<int, int> val;
 while(q--)
 {
  int type, a, b;
  cin >> type;
  if(type == 1)
  {
   cin >> a >> b;
   int x = (a + prev) % m1 + 1;
   int y = (b + prev) % m2 + 1;
   if(val.find(x) != val.end())
    insert(x, -val[x]);
   val[x] = y;
   insert(x, val[x]);
  }
  else
  {
   cin >> a;
   int x = (a + prev) % m1 + 1;
   prev = query(x);
   if(val.find(x) != val.end())
    prev += val[x];
   cout << prev << endl;
  }
 }
 return 0;
}
```

5.8 HLD1

```
class HeavyLight {
  struct Node {
    int jump, subsize, depth, lin, parent;
    vector<int> leg;
  };
  vector<Node> T;
  bool processed;
public:
  HeavyLight(int n) : T(n) {}
  void AddEdge(int a, int b) {
    T[a].leg.push_back(b);
    T[b].leg.push_back(a);
  }
  void Preprocess() {
    dfs_sub(0, -1);
    dfs_jump(0, 0);
    processed = true;
  }
  // Gets the position in the HL linearization
  int GetPosition(int node) {
    assert(processed);
    return T[node].lin;
  }
  // Gets an array of ranges of form [li...ri)
  // that correspond to the ranges you would need
  // to query in the underlying structure
  vector<pair<int, int>> GetPathRanges(int a, int b) {
    assert(processed);
    vector<pair<int, int>> ret;
    while (T[a].jump != T[b].jump) {
      if (T[T[a].jump].depth < T[T[b].jump].depth)
        swap(a, b);
      ret.emplace_back(T[T[a].jump].lin, T[a].lin + 1);
      a = T[T[a].jump].parent;
    }
    if (T[a].depth < T[b].depth) swap(a, b);
    ret.emplace_back(T[b].lin, T[a].lin + 1);
    return ret;
  }
private:
  int dfs_sub(int x, int par) {
    auto &node = T[x];
    node.subsize = 1;
    node.parent = par;
    if (par != -1) {
      node.leg.erase(find(node.leg.begin(), node.leg.end(), par));
      node.depth = 1 + T[par].depth;
    }
    for (auto vec : node.leg)
      node.subsize += dfs_sub(vec, x);
    return node.subsize;
  }
  int timer = 0;
  void dfs_jump(int x, int jump) {
    auto &node = T[x];
    node.jump = jump;
    node.lin = timer++;
    iter_swap(node.leg.begin(), max_element(node.leg.begin(), node.leg.end(),
      [&](int a, int b) { return T[a].subsize < T[b].subsize; }));
    for (auto vec : node.leg)
      dfs_jump(vec, vec == node.leg.front() ? jump : vec);
  }
};
```

5.9 HLD2

```
struct subtree_heavy_light {
  int n = 0;
  bool vertex_mode;
  vector<vector<int>> adj;
  vector<int> parent;
  vector<int> depth;
  vector<int> subtree_size;
  vector<int> tour_start, tour_end;
  vector<int> chain_root;
  list_fenwick_tree full_tree;
  subtree_heavy_light() {}
  subtree_heavy_light(int _n, bool _vertex_mode) {
    init(_n, _vertex_mode);
  }
  void init(int _n, bool _vertex_mode) {
    n = _n;
    vertex_mode = _vertex_mode;
    adj.assign(n, {});
    parent.resize(n);
    depth.resize(n);
    subtree_size.resize(n);
    tour_start.resize(n);
    tour_end.resize(n);
    chain_root.resize(n);
  }
  void add_edge(int a, int b) {
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  void dfs(int node, int par) {
    parent[node] = par;
    depth[node] = par < 0 ? 0 : depth[par] + 1;
    subtree_size[node] = 1;
    // Erase the edge to the parent.
    for (int &neighbor : adj[node])
      if (neighbor == par) {
        swap(neighbor, adj[node].back());
        adj[node].pop_back();
        break;
      }
    for (int &child : adj[node]) {
      dfs(child, node);
      subtree_size[node] += subtree_size[child];
      if (subtree_size[child] > subtree_size[adj[node].front()])
        swap(child, adj[node].front());
    }
  }
  void chain_dfs(int node, int par, bool heavy) {
    static int tour;
    if (par < 0)
      tour = 0;
    chain_root[node] = heavy ? chain_root[parent[node]] : node;
    tour_start[node] = tour++;
    bool heavy_child = true;
    for (int child : adj[node]) {
      chain_dfs(child, node, heavy_child);
      heavy_child = false;
    }
    tour_end[node] = tour;
  }
  void build(const auto &values) {
    if (n == 0) return;
    dfs(0, -1);
    chain_dfs(0, -1, false);
    full_tree.init(n);
    assert((int) values.size() == n);
    for (int i = 0; i < n; i++)
      full_tree.update(tour_start[i], values[i]);
    full_tree.build();
  }
  bool is_ancestor(int a, int b) const {
    return tour_start[a] <= tour_start[b] && tour_start[b] < tour_end[a];
  }
  // Returns the child of a that is an ancestor of b.
  int child_ancestor(int a, int b) const {
    assert(is_ancestor(a, b));
    assert(!adj[a].empty());
    int low = 0, high = adj[a].size() - 1;
    while (low < high) {
      int mid = (low + high) / 2;
      if (tour_start[b] < tour_end[adj[a][mid]])
        high = mid;
      else
        low = mid + 1;
    }
    int child = adj[a][low];
    assert(is_ancestor(child, b));
    return child;
  }
  // void update_subtree(int v, const segment_change &change) {
  //   full_tree.update(tour_start[v] + (vertex_mode ? 0 : 1), tour_end[v], change);
  // }
  int query_subtree(int v, int L, int R) {
    return full_tree.query(tour_start[v] + (vertex_mode ? 0 : 1), tour_end[v], L, R);
  }
  template<typename T_tree_op>
  void process_path(int u, int v, T_tree_op &&op) {
    while (chain_root[u] != chain_root[v]) {
      // Always pull up the chain with the deeper root.
      if (depth[chain_root[u]] > depth[chain_root[v]])
        swap(u, v);
      int root = chain_root[v];
      op(full_tree, tour_start[root], tour_start[v] + 1);
      v = parent[root];
    }
    if (depth[u] > depth[v])
      swap(u, v);
    // u is now an ancestor of v.
    op(full_tree, tour_start[u] + (vertex_mode ? 0 : 1), tour_start[v] + 1);
  }
  int query_path(int u, int v, int L, int R) {
    int answer = 0;
    process_path(u, v, [&](auto &tree, int a, int b) {
      answer += tree.query(a, b, L, R,
```

```
        R);
      });
      return answer;
    }
    // void update_path(int u, int v, const segment_change &change) {
    //   process_path(u, v, [&](auto &tree, int a, int b) {
    //     tree.update(a, b, change);
    //   });
    // }
};
```

5.10 Implementation of CartesianTree

```
/******************************
  Cartesian tree. Can be used as a balanced binary search tree.
  O(logN) on operation.
  Based on problem 2782 from informatics.mccme.ru:
  http://informatics.mccme.ru/mod/statements/view3.php?chapterid=2782
******************************/
#include <bits/stdc++.h>
using namespace std;
const int mod = 1000 * 1000 * 1000;
struct node {
  int x, y;
  node *l, *r;
  node(int new_x, int new_y) {
    x = new_x; y = new_y;
    l = NULL; r = NULL;
  }
};
typedef node * pnode;
void merge(pnode &t, pnode l, pnode r) {
  if (l == NULL)
    t = r;
  else if (r == NULL)
    t = l;
  else if (l->y > r->y) {
    merge(l->r, l->r, r);
    t = l;
  }
  else {
    merge(r->l, l, r->l);
    t = r;
  }
}
void split(pnode t, int x, pnode &l, pnode &r) {
  if (t == NULL)
    l = r = NULL;
  else if (t->x > x) {
    split(t->l, x, l, t->l);
    r = t;
  }
  else {
    split(t->r, x, t->r, r);
    l = t;
  }
}
void add(pnode &t, pnode a) {
  if (t == NULL)
    t = a;
  else if (a->y > t->y) {
    split(t, a->x, a->l, a->r);
    t = a;
  }
  else {
    if (t->x < a->x)
      add(t->r, a);
    else
      add(t->l, a);
  }
}
int next(pnode t, int x) {
  int ans = -1;
  while (t != NULL) {
    if (t->x < x)
      t = t->r;
    else {
      if (ans == -1 || ans > t->x)
        ans = t->x;
      t = t->l;
    }
  }
  return ans;
}
int n, ans, x;
char qt, prev_qt;
pnode root = NULL, num;
int main() {
  //freopen("input.txt","r",stdin);
  //freopen("output.txt","w",stdout);
  scanf("%d\n", &n);
  for (int i = 1; i <= n; i++) {
    scanf("%c %d\n", &qt, &x);
    if (qt == '+') {
      if (prev_qt == '?')
        x = (x + ans) % mod;
      num = new node(x, rand());
      add(root, num);
    }
    else {
      ans = next(root, x);
      printf("%d\n", ans);
    }
    prev_qt = qt;
  }
  return 0;
}
```

5.11 Treap DS

```
class Treap {
  int key, subsize;
  unsigned priority;
  Treap *left, *right;
  static Treap *nill;
  static bool initialized;
  Treap* pull() {
    subsize = (this != nill) + left->subsize + right->subsize;
    return this;
  }
  static void initialize() {
    if (initialized) return;
    initialized = true;
    nill->left = nill->right = nill;
    nill->priority = 0;
    nill->subsize = 0;
  }
  int get_key();
  Treap() {}
public:
  static Treap* Nill() { initialize(); return nill; }
  static Treap* Singleton(int key) {
    initialize();
    Treap* ret = new Treap();
    ret->key = key;
    ret->priority = rand() * RAND_MAX + rand();
    ret->left = ret->right = nill;
    return ret->pull();
  }
  pair<Treap*, Treap*> Split(int key) {
    if (this == nill) return {nill, nill};
    if (get_key() > key) {
      auto p = left->Split(key);
      left = p.second;
      p.second = this->pull();
      return p;
    } else {
      auto p = right->Split(key);
      right = p.first;
      p.first = this->pull();
      return p;
    }
  }
  Treap* Join(Treap *rhs) {
    if (this == nill) return rhs;
    if (rhs == nill) return this;
    if (priority > rhs->priority) {
      right = right->Join(rhs);
      return this->pull();
    } else {
      rhs->left = Join(rhs->left);
      return rhs->pull();
    }
  }
  bool Find(int key) {
    if (this == nill) return false;
    if (get_key() == key) return true;
    if (get_key() > key) return left->Find(key);
    return right->Find(key);
  }
  void Dump() {
    if (this == nill) return;
    left->Dump();
    cerr << get_key() << " ";
    right->Dump();
  }
};
Treap* Treap::nill = new Treap();
bool Treap::initialized = false;
int Treap::get_key() {
  return key;
  // return left->subsize + 1;
}
Treap* Insert(Treap *root, int key) {
  Treap* single = Treap::Singleton(key);
  auto p = root->Split(key);
  return p.first->Join(single)->Join(p.second);
}
Treap* Erase(Treap *root, int key) {
  auto p = root->Split(key);
  auto q = p.second->Split(key + 1);
  return p.first->Join(q.second);
}
```

5.12 Trie XOR

```
const int MAX = 1<<20;
const int LN = 20;
struct node {
 node *child[2];
};
static node trie_alloc[MAX*LN] = {};
static int trie_sz = 0;
node *trie;
node *get_node() {
 node *temp = trie_alloc + (trie_sz++);
 temp->child[0] = NULL;
 temp->child[1] = NULL;
 return temp;
}
//O(log A_MAX)
void insert(node *root, int n) {
 for(int i = LN-1; i >= 0; --i) {
  int x = (n&(1<<i)) ? 1 : 0;
  if (root->child[x] == NULL) {
   root->child[x] = get_node();
  }
  root = root->child[x];
 }
}
//O(log A_MAX)
int query_min(node *root, int n) {
 int ans = 0;
 for(int i = LN-1; i >= 0; --i) {
  int x = (n&(1<<i)) ? 1 : 0;
  assert(root != NULL);
  if (root->child[x] != NULL) {
   root = root->child[x];
  }
  else {
   ans ^= (1<<i);
   root = root->child[1^x];
  }
 }
 return ans;
}
//O(log A_MAX)
int query_max(node *root, int n) {
 int ans = 0;
 for(int i = LN-1; i >= 0; --i) {
  int x = (n&(1<<i)) ? 1 : 0;
  assert(root != NULL);
  if (root->child[1^x] != NULL) {
```

```
    ans ^= (1<<i);
    root = root->child[1^x];
   }
   else {
    root = root->child[x];
   }
  }
  return ans;
 }
```

5.13 Maximum over range RMQ

```
//https://codeforces.com/blog/entry/7677#comment-133682
#define SZ 262144//power of 2, in this case 2^17
int IT[SZ + SZ];
int n, H[N], A[N];
void Put(int a, long long b) {
  a += SZ;
  while (a) {
    IT[a] = max(IT[a], b);
    a >>= 1;
  }
}
long long Max(int b, int e) {
  long long r = 0;
  b += SZ, e += SZ;
  while (b <= e) {
    r = max(r, max(IT[b], IT[e]));
    b = (b + 1) >> 1, e = (e - 1) >> 1;
  }
  return r;
}
```

5.14 Querying over Pairs maximum

```
// https://www.hackerearth.com/challenges/competitive/october-easy-19/algorithm/absolute-tree-6aed7d30/submission/31998549/
// Enables online insertion of (key, value) pairs and querying of maximum value over keys less than a given limit.
// To query minimums instead, set maximum_mode = false.
template<typename T_key, typename T_value, T_value V_INF, bool maximum_mode = true>
struct online_prefix_max {
  map<T_key, T_value> optimal;
  bool better(T_value a, T_value b) {
    return (a < b) ^ maximum_mode;
  }
  // Queries the maximum value in the map over all entries with key < `key_limit`.
  T_value query(T_key key_limit) const {
    auto it = optimal.lower_bound(key_limit);
    if (it == optimal.begin())
      return maximum_mode ? (is_signed<T_value>::value ? -V_INF : 0) : V_INF;
    it--;
    return it->second;
  }
  // Adds an entry to the map and discards entries that are now obsolete.
  void insert(T_key key, T_value value) {
    auto it = optimal.lower_bound(key);
    // Quit if value is suboptimal.
    if (it != optimal.end() && it->first == key) {
      if (!better(value, it->second))
        return;
    } else if (it != optimal.begin()) {
      auto prev_it = it;
      prev_it--;
      if (!better(value, prev_it->second))
        return;
    }
    while (it != optimal.end() && !better(it->second, value))
      it = optimal.erase(it);
    optimal.insert(it, {key, value});
  }
  //joins b inside a => (curr, b) => curr U B
  void join_into(online_prefix_max b) {
    if (optimal.size() < b.optimal.size())
      swap(*this, b);
    for (auto &p : b.optimal)
      insert(p.first, p.second);
  }
};
// Usage
// online_prefix_max<int, int, inf, false>
//vector<online_prefix_max<int, int, inf, false>> pos;
```

5.15 Merge Sort tree

```
const int MAX = 1e5 + 5;
const int LIM = 3e5 + 5;     // equals 2 * 2^ceil(log2(n))
int a[MAX];
vector<int> seg[LIM];
//Complexity : O(n logn)
void build(int t, int i, int j) {
 seg[t].clear();
 if (i == j) {
  seg[t].push_back(a[i]);
  return ;
 }
 int left = t << 1, right = left | 1;
 int mid = (i + j) / 2;
 build(left, i, mid);
 build(right, mid+1, j);
 merge(seg[left].begin(), seg[left].end(), seg[right].begin(), seg[right].end(), back_inserter(seg[t]));
}
//Returns count of i, A[i] <= val, l <= i <= r
//Complexity : O(log^2 n)
int query1(int t, int i, int j, int l, int r, int val) {
 if (i > r || j < l) return 0;
 if (l <= i && j <= r) {
  int pos = upper_bound(seg[t].begin(), seg[t].end(), val) - seg[t].begin();
  return pos;
 }
 int left = t << 1, right = left | 1;
 int mid = (i + j) / 2;
 return query1(left, i, mid, l, r, val) + query1(right, mid+1, j, l, r, val);
}
//Returns count of i, A[i] >= val, l <= i <= r
//Complexity : O(log^2 n)
int query2(int t, int i, int j, int l, int r, int val) {
 if (i > r || j < l) return 0;
 if (l <= i && j <= r) {
  int pos = lower_bound(seg[t].begin(), seg[t].end(), val) - seg[t].begin();
  return (int)seg[t].size() - pos;
 }
 int left = t << 1, right = left | 1;
 int mid = (i + j) / 2;
 return query2(left, i, mid, l, r, val) + query2(right, mid+1, j, l, r, val);
}
```

5.16 Monotonic RMQ

```
template<typename T, bool maximum_mode>
struct monotonic_rmq {
  vector<pair<T, int>> values;
  int front = 0, current_index = 0;
  int size() const {
    return (int) values.size() - front;
  }
  bool better(const T &a, const T &b) const {
    return (a < b) ^ maximum_mode;
  }
  // Adds a value and returns its index.
  int add(const T &x) {
    while (size() > 0 && !better(values.back().first, x))
      values.pop_back();
    values.emplace_back(x, current_index);
    return current_index++;
  }
  // Queries for the maximum (minimum) with index at least the given `index`.
  T query(int index) {
    while (size() > 0 && values[front].second < index)
      front++;
    assert(size() > 0);
    return values[front].first;
  }
};
```

5.17 Mos Algorithm SQRT Decomposition

```
cin>>q;//no of queries
#define ppp pair<pair<int, int> , int>
vector<ppp> queries;
for (int i = 0; i < q; ++i)
{
 int l, r;
 cin>>l>>r;
 queries.pb({{l, r}, i});
}
int root = sqrt(q);
sort(all(queries), [&](ppp x, ppp y){
 int block_x = x.first.first / root;
 int block_y = y.first.first / root;
 if(block_x != block_y)
  return block_x < block_y;
 return x.first.second < y.first.second;
});
#define ll int
ll curl = 0, curr = -1;
vector<int> answer(q);
for(auto i: v){
 ll l = i.fi.fi, r = i.fi.se;
 while(curr<r){//add
   ++curr;
   add(a[curr]);
 }
 while(curl>l){//add
   --curl;
   add(a[curl]);
 }
 while(curr>r){//remove
   remove(a[curr]);
   --curr;
 }
```

```
    while(curl<l){//remove
      remove(a[curl]);
      ++curl;
    }
    answer[i.se] = ans;
    //compute answer accordingly with the add and remove function
  }
  for(auto it: answer)
   cout<<it<<endl;
```

5.18 Persistent Segment-tree

```
struct node {
  int cnt;
  node *lc, *rc;
  node (int val, node *left, node * right) {
   this->cnt = val;
   this->lc = left;
   this->rc = right;
  }
  node* insert(int l, int r, int x, int val);
};
node *null, *root[MAX];
void init() {
  null = new node(0, NULL, NULL);
  null->lc = null;
  null->rc = null;
  root[0] = null;
}
//Complexity: O(n logn)
node* node::insert(int l, int r, int x, int val) {
  if (l > x || r < x) return this;
  else if (l == r) {
   return new node(this->cnt + val, null, null);
  }
  int mid = (l+r)/2;
  return new node(this->cnt + val, this->lc->insert(l, mid, x, val), this->rc->insert(mid+1, r, x, val));
}
//Complexity : O(log n)
int query(node *a, node *b, int l, int r, int k) {
  if (l == r) return l;
  int total = b->lc->cnt - a->lc->cnt, mid = (l+r)/2;
  if (total >= k) return query(a->lc, b->lc, l, mid, k);
  return query(a->rc, b->rc, mid+1, r, k-total);
}
//Sample "print function" for debugging
void traverse(node *a) {
  if (a == null) return;
  printf("%d ", a->cnt);
  traverse(a->lc);
  traverse(a->rc);
}
```

5.19 Persistent Segment-tree with no Pointers

```
#include <bits/stdc++.h>
#define _ ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
#define ll long long
#define pb push_back
#define sz(i) (int)(i.size())
#define fi first
#define se second
#define ld long double
#define P pair<int, int>
using namespace std;
const int N = 5e5 + 10;
// make sure N size is correctttt !!!!!!!!!
int n, root[N], a[N], seg[4*N], rev[N], NEXT_FREE_INDEX = 1, ir = 0, L[4*N], R[4*N];
void build(int id = ir, int l = 1, int r = n){
  if(l == r){
   seg[id] = 0;
   return;
  }
  L[id] = NEXT_FREE_INDEX++;
  R[id] = NEXT_FREE_INDEX++;
  int mid = (l+r)>>1;
  build(L[id], l, mid);
  build(R[id], mid+1, r);
  seg[id] = seg[L[id]] + seg[R[id]];
}
int update(int pos, int id, int l, int r){
  int newID = NEXT_FREE_INDEX++;
  if(l == r){
   seg[newID] = seg[id] + 1;
   return newID;
  }
  int mid = (l + r) >> 1;
  L[newID] = L[id], R[newID] = R[id];
  if(pos <= mid){
   L[newID] = update(pos, L[id], l, mid);
  }
  else{
   R[newID] = update(pos, R[id], mid+1, r);
  }
  seg[newID] = seg[L[newID]] + seg[R[newID]];
  return newID;
}
int query(int id, int newID, int l, int r, int k){
  if(l == r){
   return l;
  }
  int mid = (l + r)>>1;
  if(seg[L[newID]] - seg[L[id]] >= k){
   return query(L[id], L[newID], l, mid, k);
  }
  else{
   return query(R[id], R[newID], mid+1, r, k-seg[L[newID]]+seg[L[id]]);
  }
}
int32_t main(){_
  int q;
  cin>>n;
  cin>>q;
  std::map<int, int> M;
  for (int i = 1; i <= n; ++i)
  {
   cin>>a[i];
   M[a[i]];
  }
  int cnt = 0;
  for(auto it: M)
   M[it.fi] = ++cnt;
  for (int i = 1; i <= n; ++i)
  {
   int x = a[i];
   rev[M[a[i]]] = x;
   a[i] = M[a[i]];
  }
  build(0,1,n);
  root[0] = ir;
  for (int i = 1; i <= n; ++i)
  {
   root[i] = update(a[i], root[i-1], 1, n);
  }
  while(q--){
   int l, r, k;
   cin>>l>>r>>k;
   int tt = query(root[l-1], root[r], 1, n, k);
   cout<<rev[tt]<<'\n';
  }
  //make sure N size is correct !!!!!!!!!
  return 0;
}
```

5.20 Persistent Trie

```
// /home/vivek/.config/sublime-text-3/Packages/User/CF/dist/Codechef/COON2019/sample.cpp
struct node
{
  node *left, *right;
  node(node *left=NULL, node *right=NULL):
   left(left), right(right){}
};
node *null = new node();
node *tree[N];
node *insert(node *p, int key, int radix = 30){
  if(!p)p = null;
  if(radix < 0)return null;
  if(key & (1<<radix)){
   return new node(p->left, insert(p->right, key, radix-1));
  }
  else{
   return new node(insert(p->left, key, radix-1), p->right);
  }
}
int getMin(node *p, int key, int radix = 30){
  if(!p)p = null;
  if(radix < 0)return 0;
  if(key & (1<<radix)){
   if(p->right){
     return getMin(p->right, key, radix-1);
   }
   else{
     return (1<<radix)|getMin(p->left, key, radix-1);
   }
  }
  else{
   if(p->left){
     return getMin(p->left, key, radix-1);
   }
   else{
     return (1<<radix) | getMin(p->right, key, radix-1);
   }
  }
}
int getMax(node *p, int key, int radix = 30){
  if(!p)p = null;
  if(radix < 0)return 0;
  if(key & (1<<radix)){
   if(p->left){
     return (1<<radix)|getMax(p->left, key, radix-1);
   }
   else{
     return getMax(p->right, key, radix-1);
   }
  }
  else{
   if(p->right){
     return (1<<radix) | getMax(p->right, key, radix-1);
   }
   else{
     return getMax(p->left, key, radix-1);
   }
  }
}
void deallocate(node *root){
  if(!root)
   return ;
  deallocate(root->L);
  deallocate(root->R);
  delete root;
}
/******
USAGE
```

```
tree[node] = insert(tree[parent], key_into_consideration);
Ex :- tree[v] = insert(tree[u], key[v]);
Some common mistakes :-
insert(node *p, int key, int radix = ****30*****, check this default value twice...)
deallocate tree after usage
******/
/*
Example Problem :- https://www.codechef.com/MAY17/problems/GPD
*/
```

5.21 Sparse DS - 1D

```
const int MAX = 1e5 + 5;
const int LIM = 17;  //equals ceil(log2(MAX))
vector<int> inp;
int lg[MAX];   //contains log of numbers from 1 to n
int p2[LIM];   //contains powers of 2
int rmq[LIM][MAX];  //sparse table implementation
//Complexity: O(nlog n)
void build_rmq() {
 int n = inp.size();
 for(int i = 2; i <= n; ++i) lg[i] = lg[i/2] + 1;
 p2[0] = 1;
 for(int i = 0; i < n; ++i) rmq[0][i] = inp[i];
 for(int i = 1; i <= lg[n]; ++i) {
  p2[i] = 1<<i;
  int x = n - p2[i], y = p2[i-1];
  for(int j = 0; j <= x; ++j) {
   rmq[i][j] = max(rmq[i-1][j], rmq[i-1][j+y]);
  }
 }
}
//Complexity: O(1)
int query(int i, int j) {
 int x = lg[j-i+1];
 return max(rmq[x][i], rmq[x][j-p2[x]+1]);
}
```

5.22 Sparse DS - 2D

```
const int MAX = 1e3 + 3;
const int LIM = 10;    //equals ceil(log2(MAX))
int inp[MAX][MAX];
int lg[MAX];     //contains log of numbers from 1 to n
int p2[LIM];     //contains powers of 2
int rmq[LIM][LIM][MAX][MAX]; //sparse table implementation
//Complexity: O(nm logn logm)
void build_rmq(int n, int m) {
 for(int i = 2; i <= n; ++i) lg[i] = lg[i/2] + 1;
 for(int i=0; i<LIM; ++i) p2[i] = 1<<i;
 p2[0] = 1;
 for(int i = 0; i < n; ++i) {
  for(int j = 0; j < m; ++j) {
   rmq[0][0][i][j] = inp[i][j];
  }
 }
 for(int k = 0; k < n; ++k) {
  for(int j = 1; j <= lg[m]; ++j) {
   int x2 = m - p2[j], y2 = p2[j-1];
   for(int l = 0; l <= x2; ++l) {
    rmq[0][j][k][l] = max(rmq[0][j-1][k][l], rmq[0][j-1][k][l+y2]);
   }
  }
 }
 for(int i = 1; i <= lg[n]; ++i) {
  int x1 = n - p2[i], y1 = p2[i-1];
  for(int k = 0; k <= x1; ++k) {
   for(int l = 0; l < m; ++l) {
    rmq[i][0][k][l] = max(rmq[i-1][0][k][l], rmq[i-1][0][k+y1][l]);
   }
  }
 }
 for(int i = 1; i <= lg[n]; ++i) {
  int x1 = n - p2[i], y1 = p2[i-1];
  for(int k = 0; k <= x1; ++k) {
   for(int j = 1; j <= lg[m]; ++j) {
    int x2 = m - p2[j], y2 = p2[j-1];
    for(int l = 0; l <= x2; ++l) {
     rmq[i][j][k][l] = max(max(rmq[i-1][j-1][k][l], rmq[i-1][j-1][k][l+y2]), max(rmq[i-1][j-1][k+y1][l], rmq[i-1][j-1][k+y1][l+y2]));
    }
   }
  }
 }
}
//Complexity : O(1)
int query(int L1, int R1, int L2, int R2) {
 int a = L2 - L1 + 1, b = R2 - R1 + 1;
 int A = lg[a], B = lg[b], P2A = p2[lg[a]]-1, P2B = p2[lg[b]]-1;
 int u = max(rmq[A][B][L1][R1], rmq[A][B][L2-P2A][R1]);
 int v = max(rmq[A][B][L1][R2-P2B], rmq[A][B][L2-P2A][R2-P2B]);
 return max(u, v);
}
```

6 String Manipulation

6.1 String Hashing

```
//uses natural mod 2^64 , assume 0 based indexing everywhere
struct PolynomialHashing {
  int N;
  string s;
  char offset = 0;
  long long prime;
  long long *fHash, *rHash, *pk;
  //declare two instances with different primes as base to be more certain of not falling for anti hash cases
  void init(string str, long long pri = 257){
   s = str;
   prime = pri;
   N = s.size();
   fHash = new long long[N], rHash = new long long[N], pk = new long long[N];
   fHash[0] = s[0] - offset + 1;
   pk[0] = 1;
   rHash[N - 1] = s[N - 1] - offset + 1;
   //Complexity : O(n)
   for(int i = 1; i < N; i++) {
     fHash[i] = ((fHash[i - 1] * prime) % mod + s[i] - offset + 1) % mod;
     pk[i] = (pk[i - 1] * prime) % mod;
     rHash[N - 1 - i] = ((rHash[N - i] * prime)%mod + s[N - i - 1] - offset + 1) % mod;
   }
  }
  //front hash of subtring from (l, r)
  long long getFrontHash (long long l, long long r) {
   if(l == 0) return fHash[r];
   if(l > r) return 0;
   return (fHash[r] - (fHash[l - 1] * pk[r - l + 1]) % mod + mod) % mod;
  }
  //reverse hash of subtring from (l,r)
  long long getReverseHash(long long l, long long r) {
   if(r == N - 1) return rHash[l];
   if(l > r) return 0;
   return (rHash[l] - (rHash[r + 1] * pk[r - l + 1]) % mod + mod) % mod;
  }
};
/**
always use pair of hashes, example of primes if 97861 and 257
chances are high that hash collision occurs.
read more at https://codeforces.com/blog/entry/4898
USAGE :-
PolynomialHashing S, T;
S.init(s);
T.init(s, 97861);
**/
```

6.2 KMP Search

```
/*
  where   [i] is the length of the longest proper prefix of the substring s[0   i ] which is also a suffix of this substring . A proper prefix of a string is a prefix that is not equal to the string itself. By definition,   [0]=0.
  in short,   [i] => max length s.t prefix of s[0..i] is same as suffix of s[0..i]
*/
vector<int> prefix_function(string s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i-1];
    while (j > 0 && s[i] != s[j])
      j = pi[j-1];
    if (s[i] == s[j])
      j++;
    pi[i] = j;
  }
  return pi;
}
```

6.3 Z Function

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

6.4 Manchaser algorithm

```
// Manacher's Algorithm
//
// Given a string s, computes the length of the longest
// palindromes centered in each position (for parity == 1)
// or between each pair of adjacent positions (for parity == 0)
//
// Example:
// Manacher("abacaba", 1) => {0, 1, 0, 3, 0, 1, 0}
// Manacher("aabbaa", 0) => {1, 0, 3, 0, 1}
vector<int> Manacher(string s, bool parity) {
  int n = s.size(), z = parity, l = 0, r = 0;
  vector<int> ret(n - !z, 0);
  for (int i = 0; i < n - !z; ++i) {
    if (i + !z < r) ret[i] = min(r - i, ret[l + r - i - !z]);
    int L = i - ret[i] + !z, R = i + ret[i];
    while (L - 1 >= 0 && R + 1 < n && s[L - 1] == s[R + 1])
      ++ret[i], --L, ++R;
    if (R > r) l = L, r = R;
  }
  return ret;
}
```

6.5 Palindromic Tree

```
struct PalTree {
  struct Node {
    map<char, int> leg;
    int link, len, cnt;
  };
  vector<Node> T;
  int nodes = 2;
  PalTree(string str) : T(str.size() + 2) {
    T[1].link = T[1].len = 0;
    T[0].link = T[0].len = -1;
    int last = 0;
    for (int i = 0; i < (int)str.size(); ++i) {
      char now = str[i];
      int node = last;
      while (now != str[i - T[node].len - 1])
        node = T[node].link;
      if (T[node].leg.count(now)) {
        node = T[node].leg[now];
        T[node].cnt += 1;
        last = node;
        continue;
      }
      int cur = nodes++;
      T[cur].len = T[node].len + 2;
      T[node].leg[now] = cur;
      int link = T[node].link;
      while (link != -1) {
        if (now == str[i - T[link].len - 1] && T[link].leg.count(now)) {
          link = T[link].leg[now];
          break;
        }
        link = T[link].link;
      }
      if (link <= 0) link = 1;
      T[cur].link = link;
      T[cur].cnt = 1;
      last = cur;
    }
    for (int node = nodes - 1; node > 0; --node) {
      T[T[node].link].cnt += T[node].cnt;
    }
  }
};
```

7 Miscellaneous

7.1 Implementation of Gauss Jordan

```
struct Gauss
{
 static const int bits = 20;
 int table[bits];
 Gauss()
 {
  for(int i = 0; i < bits; i++)
   table[i] = 0;
 }
 int size()
 {
  int ans = 0;
  for(int i = 0; i < bits; i++)
  {
   if(table[i])
    ans++;
  }
  return ans;
 }
 // Returns true if there exists subset with xor x
 bool can(int x)
 {
  for(int i = bits-1; i >= 0; i--)
   x = min(x, x ^ table[i]);
  return x == 0;
 }
 void add(int x)
 {
  for(int i = bits-1; i >= 0 && x; i--)
  {
   if(table[i] == 0)
   {
    table[i] = x;
    x = 0;
   }
   else
    x = min(x, x ^ table[i]);
  }
 }
 // Returns maximum xor of any subset
 int getBest()
 {
  int x = 0;
  for(int i = bits-1; i >= 0; i--)
   x = max(x, x ^ table[i]);
  return x;
 }
 // Returns minimum xor of any subset with num
 // num = 0 -> minimum subset xor
 int getMin(int num = 0) {
  int res = num;
  for(int i = bits-1; i >= 0; --i) {
   if ((res ^ table[i]) < res) {
    res ^= table[i];
   }
  }
  return res;
 }
 void merge(Gauss &other)
 {
  for(int i = bits-1; i >= 0; i--)
   add(other.table[i]);
 }
};
```

7.2 Implementation of Matrix Exponentation

```
//ref : matrix expo implementation from Ashishgup Github Coding Library Repo
int add(int a, int b)
{
 int res = a + b;
 if(res >= MOD)
  return res - MOD;
 return res;
}
int mult(int a, int b)
{
 long long res = a;
 res *= b;
 if(res >= MOD)
  return res % MOD;
 return res;
}
struct matrix
{
 int arr[SZ][SZ];
 void reset()
 {
  memset(arr, 0, sizeof(arr));
 }
 void makeiden()
 {
  reset();
  for(int i=0;i<SZ;i++)
  {
   arr[i][i] = 1;
  }
 }
 matrix operator + (const matrix &o) const
 {
  matrix res;
  for(int i=0;i<SZ;i++)
  {
   for(int j=0;j<SZ;j++)
   {
    res.arr[i][j] = add(arr[i][j], o.arr[i][j]);
   }
  }
  return res;
 }
 matrix operator * (const matrix &o) const
 {
  matrix res;
  for(int i=0;i<SZ;i++)
  {
   for(int j=0;j<SZ;j++)
   {
    res.arr[i][j] = 0;
    for(int k=0;k<SZ;k++)
    {
     res.arr[i][j] = add(res.arr[i][j] , mult(arr[i][k] , o.arr[k][j]));
    }
   }
  }
  return res;
 }
};
matrix power(matrix a, int b)
{
 matrix res;
 res.makeiden();
 while(b)
 {
  if(b & 1)
  {
   res = res * a;
  }
  a = a * a;
  b >>= 1;
 }
 return res;
}
```

7.3 LIS Implementation in nlogn using Binary Search

```
int lis(vector<int> a){
  // strictly-increasing LIS in nlogn
```

```
  /*
   in case of longest nondecreasing sequences
   change lower_bound to upper_bound
  */
  vector<int> v;
  for (int i = 0; i < sz(a); i++) {
    auto it = lower_bound(v.begin(), v.end(), a[i]);
    if (it != v.end()) *it = a[i];
    else v.push_back(a[i]);
  }
  return sz(v);
}
```

7.4 Ordered Set in C++

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
//find_by_order() and order_of_key().
// The first returns an iterator to the k-th largest element (counting from zero),
// the second    the number of items in a set that are strictly smaller than our item.
/**** USAGE
  ordered_set<int> X;
  X.insert(1);
  X.insert(2);
  X.insert(4);
  X.insert(8);
  X.insert(16);
  cout<<*X.find_by_order(1)<<endl; // 2
  cout<<*X.find_by_order(2)<<endl; // 4
  cout<<*X.find_by_order(4)<<endl; // 16
  cout<<(end(X)==X.find_by_order(6))<<endl; // true
  cout<<X.order_of_key(-5)<<endl; // 0
  cout<<X.order_of_key(1)<<endl; // 0
  cout<<X.order_of_key(3)<<endl; // 2
  cout<<X.order_of_key(4)<<endl; // 2
  cout<<X.order_of_key(400)<<endl; // 5
****/
```

7.5 C++ Random Number Generator

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int rand(int l, int r){
  uniform_int_distribution<int> uid(l, r);
  return uid(rng);
}
//shuffle(v.begin(), v.end(), rng);
```

7.6 Compile build

```
{
 "cmd": ["g++", "-std=c++17", "$file", "-o", "${file_path}/${file_base_name}",
     /*"-Wall",*/ "-Wextra", /*"-pedantic",*/ /*"-O2",*/ /*"-Wshadow",*/ /*"-Wformat=2",*/
     "-Wfloat-equal", // "-Wconversion", "-Wlogical-op", "-Wcast-qual",
      "-Wcast-align", "-D_GLIBCXX_DEBUG", "-D_GLIBCXX_DEBUG_PEDANTIC",
      "-D_FORTIFY_SOURCE=2", /*"-fsanitize=address",*/"-fsanitize=undefined", "-Wall",
      "-fno-sanitize-recover","-fstack-protector", "-DLOCAL"
     ],
 "selector": "source.cpp",
 "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
 "working_dir": "${file_path}",
 "variants":
 [
   {
     "name": "Run",
     "shell": true,
     // "cmd": ["gnome-terminal -e 'bash -c \"${file_path}/${file_base_name}; echo; echo Press any key to continue...; read -n 1 -s\"' > /dev/null"],
     "cmd": ["gnome-terminal -e 'bash -c \"${file_path}/${file_base_name};echo;echo; exec bash\"'"],
   }
 ]
}
// "cmd": ["gnome-terminal -e 'bash -c \"${file_path}/${file_base_name};echo;echo; echo Press ENTER to continue; read line;exit; exec bash\"'"],
// -Wall -Wextra -pedantic -std=c++11 -O2 -Wshadow -Wformat=2 -Wfloat-equal -Wconversion
// -Wlogical-op -Wshift-overflow=2 -Wduplicated-cond -Wcast-qual -Wcast-align
// -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2 -fsanitize=address -fsanitize=undefined
// -fno-sanitize-recover -fstack-protector
// {
// "cmd":["bash", "-c", "g++ -std=c++14 -Wall '${file}' -o '${file_path}/${file_base_name}' && '${file_path}/${file_base_name}'"],
// "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
// "working_dir": "${file_path}",
// "selector": "source.c, source.c++",
// "variants":
// [
//  {
//   "name": "Run",
//   "cmd":["bash", "-c", "g++ -std=c++14 '${file}' -o '${file_path}/${file_base_name}' && '${file_path}/${file_base_name}'"]
//  }
// ]
// }
// {
// "cmd": ["g++", "-std=c++14", "${file}", "-o", "${file_path}/${file_base_name}"],
// "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
// "working_dir": "${file_path}",
// "selector": "source.c, source.c++",
// "variants":
// [
//  {
//   "name": "Run",
//   "cmd":["bash", "-c", "g++ -std=c++1y '${file}' -o '${file_path}/${file_base_name}' && '${file_path}/${file_base_name}'"]
//  }
// ]
// }
/*****Stress Testing Starts*****/
for((i = 1; ; ++i)); do
  echo $i
  ./gen $i > int
  # ./a < int > out1
  # ./brute < int > out2
  # diff -w out1 out2 || break
  diff -w <(./a < int) <(./brute < int) || break
done
```

8 Theory

Combinatorics

Sums

$\sum_{k=0}^{n} k = n(n+1)/2$ $\qquad \binom{n}{k} = \frac{n!}{(n-k)!k!}$

$\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$ $\qquad \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$

$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$ $\qquad \binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$

$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$ $\qquad \binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$

$\sum_{k=0}^{n} k^4 = (6n^5 + 15n^4 + 10n^3 - n)/30$ $\qquad \binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$

$\sum_{k=0}^{n} k^5 = (2n^6 + 6n^5 + 5n^4 - n^2)/12$ $\qquad \binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$

$\sum_{k=0}^{n} x^k = (x^{n+1} - 1)/(x-1)$ $\qquad 12! \approx 2^{28.8}$

$\sum_{k=0}^{n} kx^k = (x - (n+1)x^{n+1} + nx^{n+2})/(x-1)^2$ $\qquad 20! \approx 2^{61.1}$

$1 + x + x^2 + \cdots = 1/(1-x)$

Binomial coefficients

Number of ways to pick a multiset of size k from n elements: $\binom{n+k-1}{k}$
Number of n-tuples of non-negative integers with sum s: $\binom{s+n-1}{n-1}$, at most s: $\binom{s+n}{n}$
Number of n-tuples of positive integers with sum s: $\binom{s-1}{n-1}$
Number of lattice paths from $(0,0)$ to (a,b), restricted to east and north steps: $\binom{a+b}{a}$

Multinomial theorem. $(a_1 + \cdots + a_k)^n = \sum \binom{n}{n_1,\ldots,n_k} a_1^{n_1} \ldots a_k^{n_k}$, where $n_i \geq 0$ and $\sum n_i = n$.

$$\binom{n}{n_1,\ldots,n_k} = M(n_1,\ldots,n_k) = \frac{n!}{n_1!\ldots n_k!}$$

$$M(a,\ldots,b,c,\ldots) = M(a+\cdots+b,c,\ldots)M(a,\ldots,b)$$

Catalan numbers. $C_n = \frac{1}{n+1}\binom{2n}{n}$. $\quad C_0 = 1$, $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$. $C_{n+1} = C_n \frac{4n+2}{n+2}$.
$C_0, C_1, \ldots = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, \ldots$
C_n is the number of: properly nested sequences of n pairs of parentheses; rooted ordered binary trees with $n+1$ leaves; triangulations of a convex $(n+2)$-gon.

Derangements. Number of permutations of $n = 0, 1, 2, \ldots$ elements without fixed points is $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$ Recurrence: $D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$. Corollary: number of permutations with exactly k fixed points is $\binom{n}{k} D_{n-k}$.

Stirling numbers of 1^{st} kind. $s_{n,k}$ is $(-1)^{n-k}$ times the number of permutations of n elements with exactly k permutation cycles. $|s_{n,k}| = |s_{n-1,k-1}| + (n-1)|s_{n-1,k}|$. $\quad \sum_{k=0}^{n} s_{n,k}\, x^k = x^{\underline{n}}$

Stirling numbers of 2^{nd} kind. $S_{n,k}$ is the number of ways to partition a set of n elements into exactly k non-empty subsets. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = 1$. $x^n = \sum_{k=0}^{n} S_{n,k}\, x^{\underline{k}}$

Bell numbers. B_n is the number of partitions of n elements. $B_0, \ldots = 1, 1, 2, 5, 15, 52, 203, \ldots$
$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k = \sum_{k=1}^{n} S_{n,k}$. Bell triangle: $B_r = a_{r,1} = a_{r-1,r-1}$, $a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

Bernoulli numbers. $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1} \sum_{k=0}^{n} \binom{n+1}{k} B_k m^{n+1-k}$.
$\sum_{j=0}^{m} \binom{m+1}{j} B_j = 0$. $\quad B_0 = 1$, $B_1 = -\frac{1}{2}$. $B_n = 0$, for all odd $n \neq 1$.

Eulerian numbers. $E(n,k)$ is the number of permutations with exactly k descents $(i : \pi_i < \pi_{i+1})$ / ascents $(\pi_i > \pi_{i+1})$ / excedances $(\pi_i > i)$ / $k+1$ weak excedances $(\pi_i \geq i)$.
Formula: $E(n,k) = (k+1)E(n-1,k) + (n-k)E(n-1,k-1)$. $\quad x^n = \sum_{k=0}^{n-1} E(n,k)\binom{x+k}{n}$.

Burnside's lemma. The number of orbits under group G's action on set X:
$|X/G| = \frac{1}{|G|}\sum_{g \in G} |X_g|$, where $X_g = \{x \in X : g(x) = x\}$. ("Average number of fixed points.")
Let $w(x)$ be weight of x's orbit. Sum of all orbits' weights: $\sum_{o \in X/G} w(o) = \frac{1}{|G|}\sum_{g \in G}\sum_{x \in X_g} w(x)$.

Number Theory

Linear diophantine equation. $ax + by = c$. Let $d = \gcd(a,b)$. A solution exists iff $d|c$. If (x_0, y_0) is any solution, then all solutions are given by $(x,y) = (x_0 + \frac{b}{d}t, y_0 - \frac{a}{d}t)$, $t \in \mathbb{Z}$. To find some solution (x_0, y_0), use extended GCD to solve $ax_0 + by_0 = d = \gcd(a,b)$, and multiply its solutions by $\frac{c}{d}$.

Linear diophantine equation in n variables: $a_1x_1 + \cdots + a_nx_n = c$ has solutions iff $\gcd(a_1, \ldots, a_n)|c$. To find some solution, let $b = \gcd(a_2, \ldots, a_n)$, solve $a_1x_1 + by = c$, and iterate with $a_2x_2 + \cdots = y$.

Extended GCD

```
// Finds g = gcd(a,b) and x, y such that ax+by=g.
// Bounds: |x|<=b+1, |y|<=a+1.
void gcdext(int &g, int &x, int &y, int a, int b)
```

```
{ if (b == 0) { g = a; x = 1; y = 0; }
  else        { gcdext(g, y, x, b, a % b); y = y - (a / b) * x; }
```

Multiplicative inverse of a modulo m: x in $ax + my = 1$, or $a^{\phi(m)-1} \pmod{m}$.

Chinese Remainder Theorem. System $x \equiv a_i \pmod{m_i}$ for $i = 1, \dots, n$, with pairwise relatively-prime m_i has a unique solution modulo $M = m_1 m_2 \dots m_n$: $x = a_1 b_1 \frac{M}{m_1} + \dots + a_n b_n \frac{M}{m_n} \pmod{M}$, where b_i is modular inverse of $\frac{M}{m_i}$ modulo m_i.

System $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$ has solutions iff $a \equiv b \pmod{g}$, where $g = \gcd(m, n)$. The solution is unique modulo $L = \frac{mn}{g}$, and equals: $x \equiv a + T(b-a)m/g \equiv b + S(a-b)n/g \pmod{L}$, where S and T are integer solutions of $mT + nS = \gcd(m, n)$.

Prime-counting function. $\pi(n) = |\{p \le n : p \text{ is prime}\}|$. $n/\ln(n) < \pi(n) < 1.3n/\ln(n)$. $\pi(1000) = 168$, $\pi(10^6) = 78498$, $\pi(10^9) = 50\ 847\ 534$. n-th prime $\approx n \ln n$.

Miller-Rabin's primality test. Given $n = 2^r s + 1$ with odd s, and a random integer $1 < a < n$.
If $a^s \equiv 1 \pmod{n}$ or $a^{2^j s} \equiv -1 \pmod{n}$ for some $0 \le j \le r-1$, then n is a probable prime. With bases 2, 7 and 61, the test indentifies all composites below 2^{32}. Probability of failure for a random a is at most 1/4.

Pollard-ρ. Choose random x_1, and let $x_{i+1} = x_i^2 - 1 \pmod{n}$. Test $\gcd(n, x_{2^k+i} - x_{2^k})$ as possible n's factors for $k = 0, 1, \dots$ Expected time to find a factor: $O(\sqrt{m})$, where m is smallest prime power in n's factorization. That's $O(n^{1/4})$ if you check $n = p^k$ as a special case before factorization.

Fermat primes. A Fermat prime is a prime of form $2^{2^n} + 1$. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form $2^n + 1$ is prime only if it is a Fermat prime.

Perfect numbers. $n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

Carmichael numbers. A positive composite n is a Carmichael number ($a^{n-1} \equiv 1 \pmod{n}$ for all a: $\gcd(a, n) = 1$), iff n is square-free, and for all prime divisors p of n, $p - 1$ divides $n - 1$.

Number/sum of divisors. $\tau(p_1^{a_1} \dots p_k^{a_k}) = \prod_{j=1}^k (a_j + 1)$. $\sigma(p_1^{a_1} \dots p_k^{a_k}) = \prod_{j=1}^k \frac{p_j^{a_j+1} - 1}{p_j - 1}$.

Euler's phi function. $\phi(n) = |\{m \in \mathbb{N}, m \le n, \gcd(m, n) = 1\}|$.
$\phi(mn) = \frac{\phi(m)\phi(n)\gcd(m,n)}{\phi(\gcd(m,n))}$. $\phi(p^a) = p^{a-1}(p-1)$. $\sum_{d|n} \phi(d) = \sum_{d|n} \phi(\frac{n}{d}) = n$.

Euler's theorem. $a^{\phi(n)} \equiv 1 \pmod{n}$, if $\gcd(a, n) = 1$.

Wilson's theorem. p is prime iff $(p-1)! \equiv -1 \pmod{p}$.

Mobius function. $\mu(1) = 1$. $\mu(n) = 0$, if n is not squarefree. $\mu(n) = (-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. If for all $n \in N$, $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$, and vice versa. $\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$. $\sum_{d|n} \mu(d) = 1$. If f is multiplicative, then $\sum_{d|n} \mu(d) f(d) = \prod_{p|n} (1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n} (1 + f(p))$.

Legendre symbol. If p is an odd prime, $a \in \mathbb{Z}$, then $\left(\frac{a}{p}\right)$ equals 0, if $p|a$; 1 if a is a quadratic residue modulo p; and -1 otherwise. Euler's criterion: $\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \pmod{p}$.

Jacobi symbol. If $n = p_1^{a_1} \cdots p_k^{a_k}$ is odd, then $\left(\frac{a}{n}\right) = \prod_{i=1}^k \left(\frac{a}{p_i}\right)^{k_i}$.

Primitive roots. If the order of g modulo m (min $n > 0$: $g^n \equiv 1 \pmod{m}$) is $\phi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i = \mathrm{ind}_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod{m}$. $\mathrm{ind}_g(a)$ has logarithm-like properties: $\mathrm{ind}(1) = 0$, $\mathrm{ind}(ab) = \mathrm{ind}(a) + \mathrm{ind}(b)$.

If p is prime and a is not divisible by p, then congruence $x^n \equiv a \pmod{p}$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod{p}$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i \equiv a \pmod{p}$, $g^u \equiv x \pmod{p}$. $x^n \equiv a \pmod{p}$ iff $g^{nu} \equiv g^i \pmod{p}$ iff $nu \equiv i \pmod{p}$.)

Discrete logarithm problem. Find x from $a^x \equiv b \pmod{m}$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n = \lceil \sqrt{m} \rceil$, and $x = ny - z$. Equation becomes $a^{ny} \equiv ba^z \pmod{m}$. Precompute all values that the RHS can take for $z = 0, 1, \dots, n-1$, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

Pythagorean triples. Integer solutions of $x^2 + y^2 = z^2$ All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n, \gcd(m, n) = 1$ and $m \not\equiv n \pmod{2}$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

Postage stamps/McNuggets problem. Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax + by$ ($x, y \ge 0$), and the largest is $(a-1)(b-1) - 1 = ab - a - b$.

Fermat's two-squares theorem. Odd prime p can be represented as a sum of two squares iff $p \equiv 1 \pmod 4$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k + 3$ occurs an even number of times in n's factorization.

RSA. Let p and q be random distinct large primes, $n = pq$. Choose a small odd integer e, relatively prime to $\phi(n) = (p-1)(q-1)$, and let $d = e^{-1} \pmod{\phi(n)}$. Pairs (e, n) and (d, n) are the public and secret keys, respectively. Encryption is done by raising a message $M \in Z_n$ to the power e or d, modulo n.

String Algorithms

Burrows-Wheeler inverse transform. Let $B[1..n]$ be the input (last column of sorted matrix of string's rotations.) Get the first column, $A[1..n]$, by sorting B. For each k-th occurence of a character c at index i in A, let $next[i]$ be the index of corresponding k-th occurence of c in B. The r-th fow of the matrix is $A[r]$, $A[next[r]]$, $A[next[next[r]]]$, ...

Huffman's algorithm. Start with a forest, consisting of isolated vertices. Repeatedly merge two trees with the lowest weights.

Graph Theory

Euler's theorem. For any planar graph, $V - E + F = 1 + C$, where V is the number of graph's vertices, E is the number of edges, F is the number of faces in graph's planar drawing, and C is the number of connected components. Corollary: $V - E + F = 2$ for a 3D polyhedron.

Vertex covers and independent sets. Let M, C, I be a max matching, a min vertex cover, and a max independent set. Then $|M| \le |C| = N - |I|$, with equality for bipartite graphs. Complement of an MVC is always a MIS, and vice versa. Given a bipartite graph with partitions (A, B), build a network: connect source to A, and B to sink with edges of capacities, equal to the corresponding nodes' weights, or 1 in the unweighted case. Set capacities of the original graph's edges to the infinity. Let (S, T) be a minimum s-t cut. Then a maximum(-weighted) independent set is $I = (A \cap S) \cup (B \cap T)$, and a minimum(-weighted) vertex cover is $C = (A \cap T) \cup (B \cap S)$.

Matrix-tree theorem. Let matrix $T = [t_{ij}]$, where t_{ij} is the number of multiedges between i and j, for $i \ne j$, and $t_{ii} = -\deg_i$. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T.

Euler tours. Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected. Construction:

```
doit(u):
  for each edge e = (u, v) in E, do: erase e, doit(v)
  prepend u to the list of vertices in the tour
```

Stable marriages problem. While there is a free man m: let w be the most-preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

Stoer-Wagner's min-cut algorithm. Start from a set A containing an arbitrary vertex. While $A \ne V$, add to A the most tightly connected vertex ($z \notin A$ such that $\sum_{x \in A} w(x, z)$ is maximized.) Store cut-of-the-phase (the cut between the last added vertex and rest of the graph), and merge the two vertices added last. Repeat until the graph is contracted to a single vertex. Minimum cut is one of the cuts-of-the-phase.

Tarjan's offline LCA algorithm. (Based on DFS and union-find structure.)

```
DFS(x):
  ancestor[Find(x)] = x
  for all children y of x:
     DFS(y); Union(x, y); ancestor[Find(x)] = x
  seen[x] = true
  for all queries {x, y}:
    if seen[y] then output "LCA(x, y) is ancestor[Find(y)
```

Strongly-connected components. Kosaraju's algorithm.

1. Let G^T be a transpose G (graph with reversed edges.)
1. Call DFS(G^T) to compute finishing times $f[u]$ for each vertex u.
3. For each vertex u, in the order of decreasing $f[u]$, perform DFS(G, u).
4. Each tree in the 3rd step's DFS forest is a separate SCC.

2-SAT. Build an implication graph with 2 vertices for each variable – for the variable and its inverse; for each clause $x \vee y$ add edges $(\overline{x}, y)$ and $(\overline{y}, x)$. The formula is satisfiable iff x and $\overline{x}$ are in distinct SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (i.e. Kosaraju's last step), assigning 'true' to all variables of the current SCC (if it hasn't been previously assigned 'false'), and 'false' to all inverses.

Randomized algorithm for non-bipartite matching. Let G be a simple undirected graph with even $|V(G)|$. Build a matrix A, which for each edge $(u, v) \in E(G)$ has $A_{i,j} = x_{i,j}$, $A_{j,i} = -x_{i,j}$, and is zero elsewhere. Tutte's theorem: G has a perfect matching iff $\det G$ (a multivariate polynomial) is identically zero. Testing the latter can be done by computing the determinant for a few random values of $x_{i,j}$'s over some field. (e.g. Z_p for a sufficiently large prime p)

Prufer code of a tree. Label vertices with integers 1 to n. Repeatedly remove the leaf with the smallest label, and output its only neighbor's label, until only one edge remains. The sequence has length $n-2$. Two isomorphic trees have the same sequence, and every sequence of integers from 1 and n corresponds to a tree. Corollary: the number of labelled trees with n vertices is n^{n-2}.

Erdos-Gallai theorem. A sequence of integers $\{d_1, d_2, \ldots, d_n\}$, with $n-1 \geq d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$ is a degree sequence of some undirected simple graph iff $\sum d_i$ is even and $d_1+\cdots+d_k \leq k(k-1)+\sum_{i=k+1}^{n} \min(k, d_i)$ for all $k = 1, 2, \ldots, n-1$.

Games

Grundy numbers. For a two-player, normal-play (last to move wins) game on a graph (V, E): $G(x) = \operatorname{mex}(\{G(y) : (x, y) \in E\})$, where $\operatorname{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

Sums of games.

- *Player chooses a game and makes a move in it.* Grundy number of a position is xor of grundy numbers of positions in summed games.
- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them.* A position is losing iff each game is in a losing position.
- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.
- *Player must move in all games, and loses if can't move in some game.* A position is losing if any of the games is in a losing position.

Misère Nim. A position with pile sizes $a_1, a_2, \ldots, a_n \geq 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*.

Bit tricks

Clearing the lowest 1 bit: `x & (x - 1)`, all trailing 1's: `x & (x + 1)`
Setting the lowest 0 bit: `x | (x + 1)`
Enumerating subsets of a bitmask m:
`x=0; do { ...; x=(x+1+~m)&m; } while (x!=0);`
`__builtin_ctz`/`__builtin_clz` returns the number of trailing/leading zero bits.
`__builtin_popcount(unsigned x)` counts 1-bits (slower than table lookups).
For 64-bit unsigned integer type, use the suffix ‘`ll`’, i.e. `__builtin_popcountll`.

Math

Stirling's approximation $z! = \Gamma(z+1) = \sqrt{2\pi}\ z^{z+1/2}\ e^{-z}(1+\frac{1}{12z}+\frac{1}{288z^2}-\frac{139}{51840z^3}+\ldots)$

Taylor series. $f(x) = f(a) + \frac{x-a}{1!}f'(a) + \frac{(x-a)^2}{2!}f^{(2)}(a) + \cdots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \ldots$.

$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$

$\ln x = 2(a + \frac{a^3}{3} + \frac{a^5}{5} + \ldots)$, where $a = \frac{x-1}{x+1}$. $\ln x^2 = 2\ln x$.

$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$, $\arctan x = \arctan c + \arctan \frac{x-c}{1+xc}$ (e.g c=.2)

$\pi = 4\arctan 1$, $\pi = 6\arcsin\frac{1}{2}$

List of Primes

1e5	3	19	43	49	57	69	103	109	129	151	153
1e6	33	37	39	81	99	117	121	133	171	183	
1e7	19	79	103	121	139	141	169	189	223	229	
1e8	7	39	49	73	81	123	127	183	213		