Data Structure

BIT

```cpp
// start from 1
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll MAXN = 100005;
ll tree[MAXN];
ll lowbit(int x) { return (x) & (-x); };
void Update(int i, ll x) {
  // increase
  for (int pos = i; pos <= MAXN; pos += lowbit(pos)) {
    tree[pos] += x;
  }
}
ll PrefixQuery(int n) {
  ll ret = 0;
  for (int pos = n; pos; pos -= lowbit(pos)) {
    ret += tree[pos];
  }
  return ret;
}
ll RangeQuery(int ql, int qr) { return PrefixQuery(qr) - PrefixQuery(ql - 1); }
int main() {
  int a[10] = {-1, 4, 2, 1, 5, 6, 7, 2, 1, 4};
  for (int i = 1; i <= 9; i++) {
    Update(i, a[i]);
  }
  for (int i = 1; i <= 9; i++) {
    cout << PrefixQuery(i) << endl;
  }
  return 0;
}

```

Mono Queue

```cpp
#include <bits/stdc++.h>
// monotonic descending queue, segMax at front
using namespace std;

void getSegMax(vector<int>& v, int k, vector<int>& ans) {
  deque<int> que;
  int n = v.size();
  for (int i = 0; i + 1 < k; ++i) {
    while (!que.empty() && v[que.back()] <= v[i]) que.pop_back();
    que.push_back(i);
  }
  for (int i = k - 1; i < n; ++i) {
    while (!que.empty() && v[que.back()] <= v[i]) que.pop_back();
    que.push_back(i);
    while (que.front() <= i - k) que.pop_front();
    ans.push_back(v[que.front()]);
  }
}
void getSegMin(vector<int>& v, int k, vector<int>& ans) {
  deque<int> que;
  int n = v.size();
  for (int i = 0; i + 1 < k; ++i) {
    while (!que.empty() && v[que.back()] >= v[i]) que.pop_back();
    que.push_back(i);
  }
  for (int i = k - 1; i < n; ++i) {
    while (!que.empty() && v[que.back()] >= v[i]) que.pop_back();
    que.push_back(i);
    while (que.front() <= i - k) que.pop_front();
    ans.push_back(v[que.front()]);
  }
}
int main() {
  vector<int> v = {2, 3, 1, 4, 5, 6, 7, 3};
  vector<int> ans;
  getSegMin(v, 3, ans);
  for (auto itm: ans) {
    cout << itm << " ";
  }
  return 0;
}
```

Segment Tree Range

```
#include <iostream>
using namespace std;
using ll = long long;
const int MAXN = 200005;

struct Node {
  // TODO modify to fit the need
  ll l, r;
  ll ans, mulv, addv;
  Node() {}
};
Node tree[MAXN << 2];
ll n, m, q, rawValues[MAXN];

void MergeNode(Node &f, const Node &lc, const Node &rc) {
  // TODO VARY based on different problems
  f.ans = (lc.ans + rc.ans) % m;
  f.addv = 0;
  f.mulv = 1;
}
void NodeAdd(int k, ll addv) {

}
void NodeMul(int k, ll mulv) {

}
void SpreadTag(Node &f, Node &sn) {
  // TODO VARY based on different problems
  ll addv = f.addv, mulv = f.mulv;
  sn.ans = (sn.ans * mulv % m + (sn.r - sn.l + 1) % m * addv % m) % m;
  sn.mulv = sn.mulv * mulv % m;
  sn.addv = (sn.addv * mulv % m + addv) % m;
}
void PushUp(int k) {  // up a level
  MergeNode(tree[k], tree[k << 1], tree[k << 1 | 1]);
}
void PushDown(int k) {  // push the lazy tag down a level
  if (!(tree[k].addv == 0 && tree[k].mulv == 1)) {
    SpreadTag(tree[k], tree[k << 1]);
    SpreadTag(tree[k], tree[k << 1 | 1]);
    // TODO reset father's lazy tag
    tree[k].addv = 0;
    tree[k].mulv = 1;
  }
}
void BuildTree(int k, int l, int r) {
  // prepare the nodes
  tree[k].l = l;
  tree[k].r = r;
  if (l == r) {
    // TODO VARY based on different problems
    tree[k].ans = rawValues[l];
    tree[k].addv = 0;
    tree[k].mulv = 1;
  } else {
    int mid = l + (r - l) / 2;
    BuildTree(k << 1, l, mid);
    BuildTree(k << 1 | 1, mid + 1, r);
    PushUp(k);
  }
}
void UpdateSegMul(int k, int l, int r, ll mulv) {
  if (l <= tree[k].l && tree[k].r <= r) {
    // TODO VARY based on problems
    // record the operation for query with smaller range
    tree[k].ans = tree[k].ans * mulv % m;
    tree[k].mulv = tree[k].mulv * mulv % m;
    tree[k].addv = tree[k].addv * mulv % m;
  } else {
    PushDown(k);
    int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
    if (mid >= l)  // separated update
      UpdateSegMul(k << 1, l, r, mulv);
    if (mid < r) UpdateSegMul(k << 1 | 1, l, r, mulv);
    PushUp(k);
  }
}
void UpdateSegAdd(int k, int l, int r, ll addv) {
  if (l <= tree[k].l && tree[k].r <= r) {
    // TODO VARY based on problems
    tree[k].ans = (tree[k].ans + addv * (tree[k].r - tree[k].l + 1) % m) % m;
    tree[k].addv = (tree[k].addv + addv) % m;
  } else {
    PushDown(k);
    int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
    if (mid >= l)  // separated update
      UpdateSegAdd(k << 1, l, r, addv);
    if (mid < r) UpdateSegAdd(k << 1 | 1, l, r, addv);
    PushUp(k);
  }
}
void UpdateDot(int k, int pos, ll val) {
  if (tree[k].l == tree[k].r) {
    // TODO VARY based on problems
    // tree[k].sum = val;
  } else {
    PushDown(k);
    int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
    if (pos <= mid)  // separated update
      UpdateDot(k << 1, pos, val);
    else
      UpdateDot(k << 1 | 1, pos, val);
    PushUp(k);
  }
}
Node Query(int k, int ql, int qr) {
  if (tree[k].l >= ql && tree[k].r <= qr) return tree[k];
  // when not single, push down firstly, then do the query
  PushDown(k);
  int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
  Node resL, resR, retVal;
  bool hasL = false, hasR = false;
  if (ql <= mid) {
    hasL = true;
    resL = Query(k << 1, ql, qr);
  }
  if (mid < qr) {
    hasR = true;
    resR = Query(k << 1 | 1, ql, qr);
  }
  if (hasL && hasR)
    MergeNode(retVal, resL, resR);
  else if (hasL)
    retVal = resL;
  else if (hasR)
    retVal = resR;
  return retVal;
}
int main() {
  ios::sync_with_stdio(false);
  cin >> n >> q >> m;
  for (int i = 1; i <= n; i++) cin >> rawValues[i];
  ///////////////////////////////
  BuildTree(1, 1, n);
  ///////////////////////////////
  int t, l, r, v;
  while (q--) {
    cin >> t >> l >> r;
    if (t == 3) {
      cout << Query(1, l, r).ans << "\n";
    } else if (t == 1) {
      cin >> v;
      UpdateSegMul(1, l, r, v);
    } else if (t == 2) {
      cin >> v;
      UpdateSegAdd(1, l, r, v);
    }
  }
  return 0;
}
```

Union Set

```
#include <iostream>
using namespace std;
const int MAXN = 100005;
int father[MAXN];
int trank[MAXN];

void Init(int n) {
  for (int i = 0; i < n; ++i) {
    father[i] = i;
    trank[i] = 0;
  }
}
int Find(int x) {
  if (father[x] == x) {
    return x;
  }
  return father[x] = Find(father[x]);
}
void Unite(int x, int y) {
  x = Find(x);
  y = Find(y);
  if (x == y) {
    return;
  }
  if (trank[x] < trank[y]) {
    father[x] = y;
  } else {
    father[y] = x;
    if (trank[x] == trank[y]) {
      trank[x]++;
    }
  }
}
bool inSame(int x, int y) { return Find(x) == Find(y); }
```

Geometry

```
int sgn (double x) { // sign of a double
    if (fabs(x) < eps) return 0;
    else if (x < 0) return -1;
    else return 1;
}
```

3D Sphere

```
#include <bits/stdc++.h>
using namespace std;
const double PI = acos(-1.0);
struct Sphere {
  double x, y, z, r;
  Sphere() {}
  Sphere(double x, double y, double z, double r) : x(x), y(y), z(z), r(r) {}
};
double IntersectionVolume(Sphere o, Sphere t) {
  // basic formula: V = (3 * r - h) * h * h * PI / 3
  // calculated from spinning surface calculus
  if (o.r < t.r) swap(o, t);
  double dis = sqrt((o.x - t.x) * (o.x - t.x) + (o.y - t.y) * (o.y - t.y) +
                    (o.z - t.z) * (o.z - t.z));
  if (dis <= o.r - t.r) {  // completely in
    return 4.0 / 3 * PI * t.r * t.r * t.r;
  } else if (dis <= o.r) {  // center of the smaller sphere in bigger sphere
    // cosA = (b2 + c2 - a2) / 2bc
    double angleb = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
    double anglea = PI - angleb;
    double l = t.r * cos(anglea);
    double H = o.r - l - dis;
    double h = t.r - l;
    return 4.0 / 3 * PI * t.r * t.r * t.r - PI / 3 * (3 * t.r - h) * h * h +
           PI / 3 * (3 * o.r - H) * H * H;
  } else if (dis < o.r + t.r) {  // normal intersection
    double angler = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
    double angleR = acos((o.r * o.r + dis * dis - t.r * t.r) / (2 * o.r * dis));
    double H = o.r - o.r * cos(angleR);
    double h = t.r - t.r * cos(angler);
    return PI / 3 * (3 * t.r - h) * h * h + PI / 3 * (3 * o.r - H) * H * H;
  } else {
    return 0;
  }
}
double IntersectionSurface(Sphere &o, Sphere &t) {
  // basic formula: S = 2 * PI * r * h
  if (o.r < t.r) swap(o, t);
  double dis = sqrt((o.x - t.x) * (o.x - t.x) + (o.y - t.y) * (o.y - t.y) +
          (o.z - t.z) * (o.z - t.z));
  if (dis <= o.r - t.r) {  // completely in
    return 4 * PI * t.r * t.r;
  } else if (dis <= o.r) {  // center of the smaller sphere in bigger sphere
    double angleb = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
    double anglea = PI - angleb;
    double l = t.r * cos(anglea);
    double H = o.r - l - dis;
    double h = t.r - l;
    return 4 * PI * t.r * t.r - 2 * PI * t.r * h + 2 * PI * o.r * H;
  } else if (dis < o.r + t.r) {  // normal intersection
    double angler = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
    double angleR = acos((o.r * o.r + dis * dis - t.r * t.r) / (2 * o.r * dis));
    double H = o.r - o.r * cos(angleR);
    double h = t.r - t.r * cos(angler);
    return 2 * PI * t.r * h + 2 * PI * o.r * H;
  } else {
    return 0;
  }
}
int main() {
  Sphere A, B;
  cin >> A.x >> A.y >> A.z >> A.r;
  cin >> B.x >> B.y >> B.z >> B.r;
  cout << fixed << setprecision(10) << 4*PI*(A.r*A.r+B.r*B.r) - IntersectionSurface(A, B) << endl;
  return 0;
}
```

2D Vector

```
/**
 * structs of
 * point, vector, segment
 * and some operator overloads
 */
// whether a seg AB intersects with a circle O?
// see the endpoints' tangent point (P, Q) angle
// angles: AOP + BOQ < AOB <==> intersect
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
ll MOD = 1e9 + 7;
ll QpowMod(ll bse, ll pwr) {
  ll ret = 1;
  while (pwr) {
    if (pwr & 1) ret = ret * bse % MOD;
    bse = bse * bse % MOD;
    pwr >>= 1;
  }
  return ret;
}
struct Point2 {
  ll x, y;
  Point2() : x(0), y(0) {}
  Point2(ll _x, ll _y) : x(_x), y(_y) {}
  ll Norm2() { return 1ll * x * x + 1ll * y * y; }
  double Norm() { return sqrt(Norm2()); }
  Point2 operator+(const Point2 &po) {
    return Point2(x + po.x, y + po.y);
  }
  Point2 operator-(const Point2 &po) {
    // note the direction
    return Point2(x - po.x, y - po.y);
  }
  bool operator==(const Point2 &po) {
    return x == po.x && y == po.y;
  }
};
typedef Point2 Vector2;
struct Segment2 {
  Point2 s, e;
  Segment2() {}
  Segment2(Point2 _s, Point2 _e) : s(_s), e(_e) {}
};
ll MulCross(const Point2 &p1, const Point2 &p2) {
  return p1.x * p2.y - p1.y * p2.x;
}
ll MulDot(const Point2 &p1, const Point2 &p2) {
  return p1.x * p2.x + p1.y * p2.y;
}
double DisPointToSeg(Point2 p, Point2 s1, Point2 s2) {
  Point2 v1 = p - s1, v2 = s2 - s1;
  if (MulDot(v2, v1) < 0 || MulDot(v2, v1) > v2.Norm2())
    return min(1.0 * (p - s1).Norm(), 1.0 * (p - s2).Norm());
  return abs(1.0 * MulCross(v2, v1) / v2.Norm());
}
int Dis2PointToSeg_INT(Point2 p, Point2 s1, Point2 s2) {
  // square of distance between two points
  Point2 v = p - s1, u = s2 - s1;
  if (MulDot(u, v) < 0 || MulDot(u, v) > u.Norm2())
    return min((p - s1).Norm2(), (p - s2).Norm2()) % MOD;
  return ((MulCross(v, u) % MOD) * (MulCross(v, u) % MOD)) % MOD *
         QpowMod(u.Norm2() % MOD, MOD - 2) % MOD;
}
int main() { return 0; }
```

3D Vector

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
ll MOD = 1e9 + 7;
struct Point3 {
  ll x, y, z;
  Point3() : x(0), y(0), z(0) {}
  Point3(ll _x, ll _y, ll _z) : x(_x), y(_y), z(_z) {}
  ll Norm2() { return x * x + y * y + z * z; }
  double Norm() { return sqrt(Norm2()); }
  Point3 operator+(const Point3 &po) {
    return Point3(x + po.x, y + po.y, z + po.z);
  }
  Point3 operator-(const Point3 &po) {
    return Point3(x - po.x, y - po.y, z - po.z);
  }
  bool operator==(const Point3 &po) {
    return x == po.x && y == po.y && z == po.z;
  }
};
typedef Point3 Vector3;
struct Segment3 {
  Point3 s, e;
  Segment3() {}
  Segment3(Point3 _s, Point3 _e): s(_s), e(_e) {}
};
ll MulDot(const Point3 &p1, const Point3 &p2) {
  return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}
Point3 MulCross(const Point3 &p1, const Point3 &p2) {
  return Point3(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x - p1.x * p2.z, p1.x * p2.y - p1.y * p2.x);
}
int main() {
  Point3 a{0, 0, 1}, b{1, 1, 1};
  Point3 c = MulCross(a, b);
  cout << c.Norm() << endl;
  return 0;
}

```

Math

C_n^m

```cpp
#include <stdio.h>
using ll = long long;
const ll MN = 2000000;
const ll MOD = 1000000007;
int fac[MN + 5], inv[MN + 5];

ll qpowMod(ll bse, ll pwr) {
  ll ret = 1;
  while (pwr) {
    if (pwr & 1) ret = ret * bse % MOD;
    bse = bse * bse % MOD;
    pwr >>= 1;
  }
  return ret;
}
void init() {
  fac[0] = 1;
  for (int i = 1; i <= MN; i++) fac[i] = 1ll * fac[i - 1] * i % MOD;
  inv[MN] = qpowMod(fac[MN], MOD - 2);
  for (int i = MN - 1; i >= 0; i--) inv[i] = 1ll * inv[i + 1] * (i + 1) % MOD;
}
int C(int n, int m) {
  if (m > n) return 0;
  return 1ll * fac[n] * inv[m] % MOD * inv[n - m] % MOD;
}
int main() {
  init();
  printf("%d\n", C(5, 3));
  return 0;
}
```

Euler Primers

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int MAXN = 1e6 + 5;
const int MOD = 1e9 + 7;
// priority_queue<ll, vector<ll>, greater<ll>> minor_que;

int prime[MAXN];
bool vis[MAXN];
int cnt = 0;
ll maxv = -1;
void EulerPrime(int n) {
  for (int i = 2; i <= n; ++i) {
    if (vis[i] == 0) {
      prime[cnt++] = i;
      vis[i] = 1;
    }
    for (int j = 0; i * prime[j] <= n; ++j) {
      vis[i * prime[j]] = 1;
      if (i % prime[j] == 0) break;  // key of O(n)
    }
  }
}
int main() {
  EulerPrime(100);
  for (int i = 0; i < cnt; ++i) printf("%d ", prime[i]);
  printf("\n");
  return 0;
}
```

Josephus Ring

```
// n - 1 规模时留下的最后一人，与 n 规模的相差了一个偏移量 k。J_{n, k} = (J_{n - 1, k} + k) mod n。（从 0 编号，下同，答案加一个偏移即可）
#include <cstdio>
long long josephus(int n, int k) {
  if (n == 1)
    return 0;
  else
    return (josephus(n - 1, k) + k) % n;
}
int main(void) {
  long long n, k;
  scanf("%lld %lld", &n, &k);

  printf("%lld\n", 1 + josephus(n, k));
  return 0;
}
// total n, k-th out, find the m-th out, start from 1
void solve(int casei) {
  cout << "Case #" << casei << ": ";
  long long ans = (K - 1) % (N - M + 1);
  if (K == 1) {
    cout << M << endl;
    return;
  }
  for (ll i = N - M + 2; i <= N; i++) {
    ans = (ans + K) % i; // normal iteration
    // jump forward
    ll rem = (i - ans - 1) / K;
    rem = min(rem, N - i); // limit the times of jump
    i += rem; // jump
    ans += rem * K;
  }
  cout << ans + 1 << endl;
}
```

Matrix Inverse Element

Inverse element of 2x2 matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is $\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}/(ad - bc)$.

Matrix Power

```cpp
#include <bits/stdc++.h>
#define inf 0x3f3f3f3f
using namespace std;
typedef long long ll;
const int N = 205, mod = 998244353, MS = 205;
struct Mat {
  ll a[MS][MS];
  ll n, m;
  Mat(int n = 0, int m = 0) : n(n), m(m) { memset(a, 0, sizeof(a)); }
  Mat operator*(const Mat& B) const {
    Mat C(n, B.m);
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= B.m; j++)
        for (int k = 1; k <= m; k++)
          C.a[i][j] = (C.a[i][j] + a[i][k] * B.a[k][j]) % mod;
    return C;
  }
};
Mat qpow(Mat a, int n) {
  Mat ans(a.n, a.n);
  for (int i = 1; i <= a.n; i++) ans.a[i][i] = 1;
  for (; n; n >>= 1, a = a * a)
    if (n & 1) ans = ans * a;
  return ans;
}
int main() {
  ll n;
  cin >> n;
  string s;
  cin >> s;
  ll now = stol(s);
  Mat A(100, 100);
  A = qpow(A, n);

  Mat B(100, 100);
  B.a[1][1] = 1;
  B = B * A;
  cout << B.a[1][now];
}
```

Quick Power

```cpp
#include <cstdio>
// a^(-1) mod p => a^(p - 2) mod p
// n * n * (n + 1) * (n + 1) / 4 = \sum_{1}^{n} i^3
// n * (n + 1) * (2n + 1) / 6 = \sum_{1}^{n} i^2
using ll = long long;
```

```
ll MOD = 1e9+7;
ll QpowMod(ll bse, ll pwr) {
  ll ret = 1;
  while (pwr) {
    if (pwr & 1) ret = ret * bse % MOD;
    bse = bse * bse % MOD;
    pwr >>= 1;
  }
  return ret;
}
int main() {
  printf("%lld", QpowMod(2, 199) * 6 % MOD);
  return 0;
}
```

Graph

SCC kosaraju

```
#include <cstdio>
#include <stack>
using namespace std;
stack<int> stk;
// adjacent matrix
int mp[10][10];
// reversed graph
int mpt[10][10];
int vst[10];
int clr[10];
int vn, en;
void dfs1(int s) {
  if (vst[s] == 1) return;
  vst[s] = 1;
  // dfs routine
  for (int i = 1; i <= vn; ++i) {
    if (mp[s][i] < 0x3f3f3f3f) {
      dfs1(i);
    }
  }
  // push
  stk.push(s);
}
void dfs2(int s, int cnt) {
  if (vst[s] == 0) return;
  clr[s] = cnt;
  vst[s] = 0;
  for (int i = 1; i <= vn; ++i) {
    if (mpt[s][i] < 0x3f3f3f3f) {
      dfs2(i, cnt);
    }
  }
}
void init() {
  for (int i = 1; i <= vn; ++i) {
    for (int j = 1; j <= vn; ++j) {
      mp[i][j] = mp[j][i] = 0x3f3f3f3f;
      mpt[i][j] = mpt[j][i] = 0x3f3f3f3f;
    }
    mpt[i][i] = mp[i][i] = 0;
  }
}
void SCC_kor() {
  for (int i = 1; i <= vn; ++i) {
    if (vst[i] == 0) dfs1(i);
  }
  int cnt = 1;
  while (!stk.empty()) {
    int s = stk.top();
    stk.pop();
    if (vst[s] == 0) continue;
    dfs2(s, cnt++);
  }
  // vertexes with same value in clr[] is in one SCC
  for (int i = 1; i <= vn; ++i) {
    printf("%d ", clr[i]);
  }
  printf("\n");
}
int main() {
  scanf("%d %d", &vn, &en);
  init();
  for (int i = 1; i <= en; ++i) {
    int fr, to;
    scanf("%d %d", &fr, &to);
    mp[fr][to] = 1;
    mpt[to][fr] = 1;
  }
  SCC_kor();
  return 0;
}
```

SCC tarjan

```
#include <bits/stdc++.h>
using namespace std;
int n, m;
struct node {
  vector<int> nxt;
} g[100000];
int dfn[100000], low[100000], d[100000], col[100000], cnt[100000], stk[100000];
int vis[100000];
int top, deep, colour;
void tarjan(int u) {
  dfn[u] = low[u] = ++deep;
  stk[top++] = u;
  vis[u] = 1;
  for (int i = 0; i < g[u].nxt.size(); i++) {
    int v = g[u].nxt[i];
    if (!vis[v]) {
      tarjan(v);
      low[u] = min(low[v], low[u]);
    } else {
      low[u] = min(low[v], low[u]);
    }
  }
  if (dfn[u] == low[u]) {
    int node;
    colour++;
    while (node != u) {
      node = stk[top - 1];
      top--;
      col[node] = colour;
    }
  }
}
```

String

KMP

```
int nxt[100005];
char t[100005];
void getNxt() {
  nxt[0] = -1;
  int k = -1, j = 0;
  while (t[j] != '\0') {
    if (k == -1 || t[k] == t[j]) {
      nxt[++j] = ++k;
    } else {
      k = nxt[k];
    }
  }
}
```

Manarcher

```
// find the palindrome in O(n)
#include <bits/stdc++.h>
using namespace std;
char s[100005];
int ps = 0;
int p[100005], ctr, maxr, mirr;
void solve() {
  ctr = maxr = 0;
  for (int i = 0; i < ps; ++i) {
    mirr = 2 * ctr - i;
    if (i < maxr) {
      p[i] = min(maxr - i, p[mirr]);
    } else {
      p[i] = 0;
    }
    while (s[i - 1 - p[i]] == s[i + 1 + p[i]]) {
      p[i]++;
    }
    if (p[i] + i > maxr) {
      ctr = i;
      maxr = p[i] + i;
    }
  }
  int maxi = 0;
  for (int i = 0; i < ps; ++i) {
    maxi = p[maxi] < p[i] ? i : maxi;
  }
  printf("%d\n", p[maxi]);
  for (int i = maxi - p[maxi]; i <= maxi + p[maxi]; ++i) {
    if (s[i] != '#') {
      printf("%c", s[i]);
    }
  }
  printf("\n");
}
int main() {
  int Case = 1;
  while (Case--) {
    char c = getchar();
    s[ps++] = '#';
    while (c != '\n') {
      s[ps++] = c;
      s[ps++] = '#';
      c = getchar();
    }
    solve();
  }
  return 0;
}
```

Misc

fastIO

```
namespace GTI
{
    char gc(void)
    {
        const int S=1<<17;
        static char buf[S],*s=buf,*t=buf;
        if (s==t) t=buf+fread(s=buf,1,S,stdin);
        if (s==t) return EOF;
        return *s++;
    }
    int gti(void)
    {
        int a=0,b=1,c=gc();
        for (;!isdigit(c);c=gc()) b^=(c=='-');
        for (;isdigit(c);c=gc()) a=a*10+c-'0';
        return b?a:-a;
    }
};

```

Discretization

```
namespace GTI
{
    char gc(void)
    {
        const int S=1<<17;
        static char buf[S],*s=buf,*t=buf;
        if (s==t) t=buf+fread(s=buf,1,S,stdin);
        if (s==t) return EOF;
        return *s++;
    }
    int gti(void)
    {
        int a=0,b=1,c=gc();
        for (;!isdigit(c);c=gc()) b^=(c=='-');
        for (;isdigit(c);c=gc()) a=a*10+c-'0';
        return b?a:-a;
    }
};
```

Inverse Pair Merge Sort

```cpp
using ll = long long;
ll MAXN = 2e5 + 5;
ll n, q[MAXN], tmp[MAXN];
// [l, r]
ll merge_sort(int l, int r) {
  if (l >= r) return 0;
  ll mid = (l + r) >> 1;
  ll res = merge_sort(l, mid) + merge_sort(mid + 1, r);

  ll k = 0, i = l, j = mid + 1;
  while (i <= mid && j <= r) {
    if (q[i] <= q[j])
      tmp[k++] = q[i++];
    else {
      tmp[k++] = q[j++];
      res += mid - i + 1;
    }
  }
  while (i <= mid) tmp[k++] = q[i++];
  while (j <= r) tmp[k++] = q[j++];
  for (ll i = l, j = 0; i <= r; i++, j++) q[i] = tmp[j];
  return res;
}
```

Modui

```cpp
/**
 * Modui range number of distinct values
 */
#include <bits/stdc++.h>
using namespace std;
#define endl "\n";
#define IOS_ONLY                \
  ios::sync_with_stdio(false); \
  cin.tie(0);                  \
  cout.tie(0);
const int MAXN = 30005, MAXQ = 200005, MAXM = 1000005;
int sq;
struct Query {
  int ql, qr, id;
  bool operator<(const Query &o) const {
    // sqrt(n) partitions, assign sq with sqrt(n) first
    if (ql / sq != o.ql / sq) return ql < o.ql;
    if (ql / sq & 1) return qr < o.qr;  // order by parity
    return qr > o.qr;
  }
} Q[MAXQ];
int A[MAXN], ans[MAXQ], Cnt[MAXM], cur, pl = 1, pr = 0, n;
inline void add(int pos) {
  if (Cnt[A[pos]] == 0) cur++;
  Cnt[A[pos]]++;
}
```

```
inline void del(int pos) {
  Cnt[A[pos]]--;
  if (Cnt[A[pos]] == 0) cur--;
}
int main() {
  IOS_ONLY
  cin >> n;
  sq = sqrt(n);
  for (int i = 1; i <= n; ++i) cin >> A[i];
  int q;
  cin >> q;
  for (int i = 0; i < q; ++i) {  // offline query
    cin >> Q[i].ql >> Q[i].qr;
    Q[i].id = i;
  }
  sort(Q, Q + q);  // sort, KEY of modui
  for (int i = 0; i < q; ++i) {
    while (pl > Q[i].ql) add(--pl);
    while (pr < Q[i].qr) add(++pr);
    while (pl < Q[i].ql) del(pl++);
    while (pr > Q[i].qr) del(pr--);
    ans[Q[i].id] = cur;  // store the rasult
  }
  for (int i = 0; i < q; ++i) cout << ans[i] << endl;
  return 0;
}
```