Contents

1 2ECC

```
struct graph {
  int n, t, sz;
  vector<vector<int>> adj;
  vector<int> tin, low, cmp;
  graph(int n): n(n),adj(n),tin(n),low(n),cmp(n){}
  void add_edge(int u, int v){
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  void dfs(int u, int p){
    tin[u]=low[u]=t++;
    int cnt=0;
    for(int v: adj[u]){
      if(v==p and ++cnt <= 1) continue;
      if(tin[v]!=-1)  low[u] = min(low[u], tin[v]);
      else {
        dfs(v,u);
        low[u] = min(low[u], low[v]);
      }
    }
  }
  void dfs2(int u, int p){
    if(p!=-1 and tin[p]>=low[u]) cmp[u] = cmp[p];
    else  cmp[u] = sz++;
    for(int v: adj[u]){
      if(cmp[v]==-1)  dfs2(v,u);
    }
  }
  void process_2ecc(){
    t = 0, sz = 0;
    for (int i = 0; i < n; ++i){
      tin[i] = low[i] = cmp[i] = -1;
    }
    for (int i = 0; i < n; ++i){
      if(tin[i]==-1)  dfs(i,-1);
    }
    for (int i = 0; i < n; ++i){
      if(cmp[i]==-1)  dfs2(i,-1);
    }
  }
};
```

2 2SAT

```
//CNF: (a | b) ^ (c | d) means (!a -> b) ^ (!b -> a)
// (!a or b) = (-a, b), 1-based indexing
string two_sat(int n, vector<array<int, 2>>
 ↪  clauses) {
  vector<int> adj[2 * n];
  for (auto [a, b]: clauses) {
    if (a > 0)  a = 2 * a - 2;
    else  a = 2 * -a - 1;
    if (b > 0)  b = 2 * b - 2;
    else  b = 2 * -b - 1;
    adj[a ^ 1].push_back(b), adj[b ^
 ↪  1].push_back(a);
  }
  vector<vector<int>> sccs = get_sccs(2 * n, adj);
  int tot_scc = sccs.size();
  vector<int> scc_no(2 * n);
  for (int i = 0; i < tot_scc; ++i) {
    for (int u: sccs[i]) {
      scc_no[u] = i;
    }
  }
  string assignment;
  for (int u = 0; u < n; u++) {
    if (scc_no[2 * u] == scc_no[2 * u + 1]) {
      return "";
    }
    if (scc_no[2 * u] < scc_no[2 * u + 1]) {
      assignment += '-';
    }
    else {
      assignment += '+';
    }
  }
  return assignment;
}
```

3 AHO_CORASICK

```
struct AC{
  const int A = 26;
  vector<vector<int>> nxt, idx;
  vector<int> lnk, out_lnk, ans;
  AC(){newNode();}
  int newNode(){
    nxt.eb(A, 0), idx.eb(0);
    lnk.eb(0),  out_lnk.eb(0),  ans.eb(0);
    return nxt.size()-1;
  }
  void clear(){
    nxt.clear(), idx.clear();
    lnk.clear(), out_lnk.clear(), ans.clear();
    newNode();
  }
  // O(|p|)
  void add(string p, int i){
    int v=0;
    for(char c: p){
      if(!nxt[v][c-'a'])  nxt[v][c-'a'] = newNode();
      v = nxt[v][c-'a'];
    }
    idx[v].eb(i);
  }
  // O(|p1+p2+p3+..|)
  void build(){
    queue<int> q; q.push(0);
    while (!q.empty()){
      int u=q.front();  q.pop();
      for (int i = 0; i < A; ++i){
        int v = nxt[u][i];
        if(!v)  nxt[u][i] = nxt[lnk[u]][i];
        else {
          lnk[v] = u? nxt[lnk[u]][i]: 0;
          out_lnk[v] = idx[lnk[v]].empty()?
 ↪  out_lnk[lnk[v]]: lnk[v];
          q.push(v);
        }
      }
    }
  }
  // O(|T|+match)
  void trav(string T){
    int v=0;
    for(char c: T){
      if(!nxt[v][c-'a'])  v = lnk[v];
      if(nxt[v][c-'a']) v=nxt[v][c-'a'];
      for(auto& i: idx[v]){
        ans[i]++;
      }
      int x = out_lnk[v];
      while(x){
        for(auto& i: idx[x]){
          ans[i]++;
        }
        x = out_lnk[x];
      }
    }
  }
};
//AC ac;  ac.add(pi, i);  ac.build();  ac.trav(T);
```

4 ARTICULATION_BRIDGE

```
vector<int> adj[N];
int t = 0;
vector<int> tin(N, -1), lo(N);
vector<array<int, 2>> ab;
void dfs (int u, int p) {
  tin[u] = lo[u] = t++;
  for (int v: adj[u]) {
    if (v != p) {
      if (tin[v] != -1) {
        lo[u] = min(lo[u], tin[v]);
      }
      else {
        dfs(v, u);
        if (tin[u] < lo[v]) {
          ab.push_back({u, v});
        }
        lo[u] = min(lo[u], lo[v]);
      }
    }
  }
}
dfs(0, -1);
```

5 ARTICULATION_POINT

```
vector<int> adj[N];
int t = 0;
vector<int> tin(N, -1), low(N), ap;
void dfs (int u, int p) {
  tin[u] = low[u] = t++;
  int is_ap = 0, child = 0;
  for (int v: adj[u]) {
    if (v != p) {
      if (tin[v] != -1) {
        low[u] = min(low[u], tin[v]);
      }
      else {
        child++;
        dfs(v, u);
        if (tin[u] <= low[v]) {
          is_ap = 1;
        }
        low[u] = min(low[u], low[v]);
      }
    }
  }
  if ((p != -1 or child > 1) and is_ap)
↪   ap.push_back(u);
}
dfs(0, -1);
```

6 BCC

```
struct graph {
  int n,t=0,cno=0;
  vector<vector<int>> g;
  vector<int> tin, lo, bcomp;
  stack <int> st;
  graph(int n):n(n),g(n),lo(n),bcomp(n){}
  void add_edge(int u, int v){
    g[u].push_back(v);
    g[v].push_back(u);
  }
  void dfs(int v, int p=-1){
    lo[v]=tin[v]=++t;
    st.push(v);
    for(int u:g[v]){
      if(u==p)    continue;
      if(!tin[u]){
        dfs(u, v);
        lo[v]=min(lo[v],lo[u]);
      } else{
        lo[v]=min(lo[v],tin[u]);
      }
    }
    if(tin[v]==lo[v]){
      while (!st.empty()){
        int tp=st.top(); st.pop();
        bcomp[tp]=cno;
        if(tp==v)    break;
      }
      cno++;
    }
  }
  vector<int> bcc(){
    tin.assign(n, 0);
    for (int i = 0; i < n; ++i){
      if(!tin[i])
        dfs(i);
    }
    return bcomp;
  }
};
```

7 BCC_EDGE

```
vector<array<int, 2>> edges, adj[N];
vector <int> tin(N), lo(N), is_ap(N), bcc[N],
↪   bcc_ed[N];
int t = 0, tot = 0;
stack<int> stk;

void pop_bcc(int e) {
  do {
    bcc_ed[tot].push_back(stk.top());  stk.pop();
  } while (bcc_ed[tot].back() != e);
  tot++;
}

void dfs(int u, int p = -1) {
  int ch = 0;
  tin[u] = lo[u] = t++;
  for(auto [v, e] : adj[u]) {
    if (v == p)  continue;
    if (tin[v] != -1) {
      if (tin[u] > tin[v]) {
        lo[u] = min(lo[u], tin[v]);
        stk.push(e);
      }
    }
    else {
      ch++;
      stk.push(e);
      dfs(v, u);
      if ((p != -1 or ch > 1) and tin[u] <= lo[v]) {
        is_ap[u] = 1;
        pop_bcc(e);
      }
      lo[u] = min(lo[u], lo[v]);
    }
  }
}

void procces_bcc(int n) {
  for (int i = 0; i < n; ++i) {
    tin[i] = -1, is_ap[i] = 0;
    bcc_ed[i].clear();
    bcc[i].clear();
  }
  t = tot = 0;

  for (int u = 0; u < n; ++u) {
    if (tin[u] == -1) {
      dfs(u, -1);
      if (!stk.empty()) {
        while (!stk.empty()) {
          bcc_ed[tot].push_back(stk.top());
↪   stk.pop();
        }
        tot++;
      }
    }
  }

  for (int i = 0; i < tot; ++i) {
    for (auto e: bcc_ed[i]) {
      auto [u, v] = edges[e];
      bcc[i].push_back(u);
      bcc[i].push_back(v);
    }
  }

  for (int i = 0; i < tot; ++i) {
    sort(bcc[i].begin(), bcc[i].end());
    bcc[i].erase(unique(bcc[i].begin(),
↪   bcc[i].end()), bcc[i].end());
  }
}
```

8 BINARY_LIFTING

```
void dfs(int u, int p){
  d[u]=d[p]+1;
  tin[u] = t++;
  par[u][0]=p;
  for (int i = 1; i < LGN; ++i){
    par[u][i] = par[par[u][i-1]][i-1];
  }
  for(int& v: tr[u])
    if(v!=p)  dfs(v,u);
  tout[u] = t++;
}
bool is_anc(int u, int v){
  return tin[u]<=tin[v] and tout[u]>=tout[v];
}
int lca(int u, int v){
  if(is_anc(u, v))  return u;
  if(is_anc(v, u))  return v;
  for(int i=LGN-1; i>=0; i--){
    if(!is_anc(par[u][i], v)) u=par[u][i];
  }
  return par[u][0];
}
int dist(int u, int v){
  int w = lca(u, v);
  return d[u]-d[w]+d[v]-d[w];
}
```

9 BIT_TRICKS

```
## Next Combination Mask
int next_combs_mask(int mask) {
  int lsb = -mask & mask;
  return (((mask + lsb) ^ mask) / (lsb << 2)) |
↪   (mask + lsb);
}
## Iterate over submask in decreasing order
for (int submask=mask; submask > 0; submask =
↪   (submask-1)&mask) {
  // do stuff
}
```

10 BLOCK_CUT_TREE

```
vector<int> adj[N];
vector<int> tin(N, -1), lo(N), is_ap(N), bcc[N];
stack<int> stk;
int t = 0, tot = 0;

void pop_bcc(int u, int v) {
  bcc[tot].push_back(u);
  while (bcc[tot].back() != v) {
    bcc[tot].push_back(stk.top());
    stk.pop();
  }
  tot++;
}
void dfs (int u, int p) {
  tin[u] = lo[u] = t++;
  stk.push(u);
  int ch = 0;
  for (auto v: adj[u]) {
    if (v != p) {
      if (tin[v] != -1) {
        lo[u] = min(lo[u], tin[v]);
      }
      else {
        ch++;
        dfs(v, u);
        if ((p != -1 or ch > 1) and tin[u] <=
 ↪  lo[v]) {
          // is_ap[u] = 1;
          pop_bcc(u, v);
        }
        lo[u] = min(lo[u], lo[v]);
      }
    }
  }
}
void process_bcc (int n) {
  for (int u = 0; u < n; ++u) {
    tin[u] = -1;
    is_ap[u] = 0;
    bcc[u].clear();
  }
  t = tot = 0;
  for (int u = 0; u < n; ++u) {
    if (tin[u] == -1) {
      dfs(u, -1);
      if (!stk.empty()) {
        while (!stk.empty()) {
          bcc[tot].push_back(stk.top());
          stk.pop();
        }
        tot++;
      }
    }
  }
}
int nn;
vector<int> comp_num(N), bct_adj[N];
void build_bct(int n) {
  process_bcc(n);
  int nn = tot;
  for (int u = 0; u < n; ++u) {
    if (is_ap[u]) {
      comp_num[u] = nn++;
    }
  }
  for (int i = 0; i < tot; ++i) {
    for (auto u: bcc[i]) {
      if (is_ap[u]) {
        u = comp_num[u];
        bct_adj[i].push_back(u);
        bct_adj[u].push_back(i);
      }
      else {
        comp_num[u] = i;
      }
    }
  }
}
```

11 CDQ

```
## Problems related to pair
 ↪  (https://www.luogu.com.cn/problem/P3157)
  - cdq(l, m)
  - cdq(m + 1, r)
  - handle influence of (l, m) to (m + 1, r)
## Optimization of 1D DP
  - cdq(l, m)
  - handle influence of (l, m) to (m + 1, r)
  - cdq(m + 1, r)
## Convert dynamic array problems to static array
 ↪  problem
```

12 CENTROID_DECOMPOSITION

```
void calc_sz(int u, int p) {
  sz[u] = 1;
  for (auto v: adj[u]) {
    if (v != p and !is_cen[v]) {
      calc_sz(v, u);
      sz[u] += sz[v];
    }
  }
}
int get_cen(int u, int p, int n) {
  for (auto v: adj[u]) {
    if (v != p and !is_cen[v] and 2 * sz[v] > n) {
      return get_cen(v, u, n);
    }
  }
  return u;
}
void decompose(int u=0, int p=-1, int d=0){
  calc_sz(u, p);
  int c = get_cen(u, p, sz[u]);
  is_cen[c] = 1, cpar[c] = p, cdep[c] = d;
  for(int v: adj[c]){
    if(!is_cen[v]) {
      decompose(v,c,d+1);
    }
  }
}
decompose();
```

13 CLOSEST_PAIR_OF_POINTS

```
ll min_dis(vector<array<int, 2>> &pts, int l, int
 ↪  r) {
  if (l + 1 >= r)  return LLONG_MAX;
  int m = (l + r) / 2;
  ll my = pts[m-1][1];
  ll d = min(min_dis(pts, l, m), min_dis(pts, m,
 ↪  r));
  inplace_merge(pts.begin()+l, pts.begin()+m,
 ↪  pts.begin()+r);
  for (int i = l; i < r; ++i) {
    if ((pts[i][1] - my) * (pts[i][1] - my) < d) {
      for (int j = i + 1; j < r and (pts[i][0] -
 ↪  pts[j][0]) * (pts[i][0] - pts[j][0]) < d; ++j) {
        ll dx = pts[i][0] - pts[j][0], dy =
 ↪  pts[i][1] - pts[j][1];
        d = min(d, dx * dx + dy * dy);
      }
    }
  }
  return d;
}
vector<array<int, 2>> pts(n);
sort(pts.begin(), pts.end(), [&] (array<int, 2> a,
 ↪  array<int, 2> b){
  return make_pair(a[1], a[0]) < make_pair(b[1],
 ↪  b[0]);
});
```

14 CONVEX_HULL

```
struct pt {
  int x, y;
};
ll cross(pt a, pt b, pt c) {  //ab*ac
  return 1ll*(b.x-a.x)*(c.y-a.y) -
 ↪  1ll*(c.x-a.x)*(b.y-a.y);
}
vector<pt> convexHull(vector<pt>& p) {
  sort(p.begin(), p.end(), [&] (pt a, pt b) {
    return (a.x==b.x? a.y<b.y: a.x<b.x);
  });
  int n = p.size(), m = 0;
  vector<pt> hull(2*n);
  for (int i = 0; i < n; ++i){
    while (m>=2 and cross(hull[m-2], hull[m-1],
 ↪  p[i]) < 0)  --m;
    hull[m++] = p[i];
  }
  for (int i = n-2, l = m; i >= 0; --i) {
    while(m>=l+1 and cross(hull[m-2], hull[m-1],
 ↪  p[i]) < 0)  --m;
    hull[m++] = p[i];
  }
  hull.resize(m-1);
  return hull;
}
```

15 CONVOLUTION

```
## FFT
struct cplx {
  ld a, b;
  cplx(ld a=0, ld b=0):a(a), b(b) {}
  const cplx operator + (const cplx &z) const {
 ↪  return cplx(a+z.a, b+z.b); }
  const cplx operator - (const cplx &z) const {
 ↪  return cplx(a-z.a, b-z.b); }
  const cplx operator * (const cplx &z) const {
 ↪  return cplx(a*z.a-b*z.b, a*z.b+b*z.a); }
  const cplx operator / (const ld &k) const {
 ↪  return cplx(a/k, b/k); }
};

const ld PI=acos(-1);
vector<int> rev;

void pre(int sz){
  if(rev.size()==sz)  return ;
  rev.resize(sz);
  rev[0]=0;
  int lg_n = __builtin_ctz(sz);
  for (int i = 1; i < sz; ++i)  rev[i] = (rev[i>>1]
 ↪  >> 1) | ((i&1)<<(lg_n-1));
}
void fft(vector<cplx> &a, bool inv){
  int n = a.size();
  for (int i = 1; i < n-1; ++i) if(i<rev[i])
 ↪  swap(a[i], a[rev[i]]);
```

```
  for (int len = 2; len <= n; len <<= 1){
    ld t = 2*PI/len*(inv? -1: 1);
    cplx wlen = {cosl(t), sinl(t)};
    int st = 0;
    for (int st = 0; st < n; st += len){
      cplx w(1);
      for (int i = 0; i < len/2; ++i){
        cplx ev = a[st+i];
        cplx od = a[st+i+len/2]*w;
        a[st+i] = ev+od;
        a[st+i+len/2] = ev-od;
        w = w*wlen;
      }
    }
  }
  if(inv){
    for(cplx &z: a){
      z = z/n;
    }
  }
}

vector<ll> mul(vector<ll> &a, vector<ll> &b){
  int n = a.size(), m = b.size(), sz = 1;
  while (sz < n+m-1)  sz <<= 1;
  vector<cplx> x(sz), y(sz), z(sz);
  for (int i = 0; i < sz; ++i){
    x[i] = cplx(i<n? a[i]: 0, 0);
    y[i] = cplx(i<m? b[i]: 0, 0);
  }
  pre(sz);
  fft(x, 0);
  fft(y, 0);
  for (int i = 0; i < sz; ++i){
    z[i] = x[i] * y[i];
  }
  fft(z, 1);
  vector<ll> c(n+m-1);
  for (int i = 0; i < n+m-1; ++i){
    c[i] = round(z[i].a);
  }
  return c;
}
```

## NTT

```
const int mod = 998244353;
const int root = 15311432;
const int k = 1 << 23;

int root_1;
vector<int> rev;

ll bigmod(ll a, ll b, ll mod){
  a %= mod;
  ll ret = 1;
  while(b){
    if(b&1) ret = ret*a%mod;
    a = a*a%mod;
    b >>= 1;
  }
  return ret;
}

void pre(int sz){
  root_1 = bigmod(root, mod-2, mod);
  if(rev.size()==sz)  return ;
  rev.resize(sz);
  rev[0]=0;
  int lg_n = __builtin_ctz(sz);
  for (int i = 1; i < sz; ++i)  rev[i] = (rev[i>>1] >> 1) | ((i&1)<<(lg_n-1));
}

void fft(vector<int> &a, bool inv){
  int n = a.size();

  for (int i = 1; i < n-1; ++i) if(i<rev[i]) swap(a[i], a[rev[i]]);

  for (int len = 2; len <= n; len <<= 1) {
    int wlen = inv ? root_1 : root;
    for (int i = len; i < k; i <<= 1){
      wlen = 1ll*wlen*wlen%mod;
    }
    for (int st = 0; st < n; st += len) {
      int w = 1;
      for (int j = 0; j < len / 2; j++) {
        int ev = a[st+j];
        int od = 1ll*a[st+j+len/2]*w%mod;
        a[st+j] = ev + od < mod ? ev + od : ev + od - mod;
        a[st+j+len/2] = ev - od >= 0 ? ev - od : ev - od + mod;
        w = 1ll * w * wlen % mod;
      }
    }
  }

  if (inv) {
    int n_1 = bigmod(n, mod-2, mod);
    for (int & x : a)
      x = 1ll*x*n_1%mod;
  }
}

vector<int> mul(vector<int> &a, vector<int> &b){
  int n = a.size(), m = b.size(), sz = 1;
  while (sz < n+m-1)  sz <<= 1;
  vector<int> x(sz), y(sz), z(sz);
  for (int i = 0; i < sz; ++i){
    x[i] = i<n? a[i]: 0;
    y[i] = i<m? b[i]: 0;
  }
  pre(sz);
  fft(x, 0);
  fft(y, 0);
  for (int i = 0; i < sz; ++i){
    z[i] = 1ll* x[i] * y[i] % mod;
  }
  fft(z, 1);
  z.resize(n+m-1);
  return z;
}
```

## Any mod

```
const int N = 3e5 + 9, mod = 998244353;

struct base {
  double x, y;
  base() { x = y = 0; }
  base(double x, double y): x(x), y(y) { }
};
inline base operator + (base a, base b) { return base(a.x + b.x, a.y + b.y); }
inline base operator - (base a, base b) { return base(a.x - b.x, a.y - b.y); }
inline base operator * (base a, base b) { return base(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
inline base conj(base a) { return base(a.x, -a.y); }
int lim = 1;
vector<base> roots = {{0, 0}, {1, 0}};
vector<int> rev = {0, 1};
const double PI = acosl(- 1.0);
void ensure_base(int p) {
  if(p <= lim) return;
  rev.resize(1 << p);
  for(int i = 0; i < (1 << p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1)  <<  (p - 1));
  roots.resize(1 << p);
  while(lim < p) {
    double angle = 2 * PI / (1 << (lim + 1));
    for(int i = 1 << (lim - 1); i < (1 << lim); i++) {
      roots[i << 1] = roots[i];
      double angle_i = angle * (2 * i + 1 - (1 << lim));
      roots[(i << 1) + 1] = base(cos(angle_i), sin(angle_i));
    }
    lim++;
  }
}
void fft(vector<base> &a, int n = -1) {
  if(n == -1) n = a.size();
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = lim - zeros;
  for(int i = 0; i < n; i++) if(i < (rev[i] >> shift)) swap(a[i], a[rev[i] >> shift]);
  for(int k = 1; k < n; k <<= 1) {
    for(int i = 0; i < n; i += 2 * k) {
      for(int j = 0; j < k; j++) {
        base z = a[i + j + k] * roots[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] = a[i + j] + z;
      }
    }
  }
}
//eq = 0: 4 FFTs in total
//eq = 1: 3 FFTs in total
vector<int> multiply(vector<int> &a, vector<int> &b, int eq = 0) {
  int need = a.size() + b.size() - 1;
  int p = 0;
  while((1 << p) < need) p++;
  ensure_base(p);
  int sz = 1 << p;
  vector<base> A, B;
  if(sz > (int)A.size()) A.resize(sz);
  for(int i = 0; i < (int)a.size(); i++) {
    int x = (a[i] % mod + mod) % mod;
    A[i] = base(x & ((1 << 15) - 1), x >> 15);
  }
  fill(A.begin() + a.size(), A.begin() + sz, base{0, 0});
  fft(A, sz);
  if(sz > (int)B.size()) B.resize(sz);
  if(eq) copy(A.begin(), A.begin() + sz, B.begin());
  else {
    for(int i = 0; i < (int)b.size(); i++) {
      int x = (b[i] % mod + mod) % mod;
      B[i] = base(x & ((1 << 15) - 1), x >> 15);
    }
    fill(B.begin() + b.size(), B.begin() + sz,
↪   base{0, 0});
    fft(B, sz);
  }
  double ratio = 0.25 / sz;
  base r2(0,  - 1), r3(ratio, 0), r4(0,  - ratio),
↪   r5(0, 1);
  for(int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    base a1 = (A[i] + conj(A[j])), a2 = (A[i] -
↪   conj(A[j])) * r2;
    base b1 = (B[i] + conj(B[j])) * r3, b2 = (B[i]
↪   - conj(B[j])) * r4;
    if(i != j) {
      base c1 = (A[j] + conj(A[i])), c2 = (A[j] -
↪   conj(A[i])) * r2;
      base d1 = (B[j] + conj(B[i])) * r3, d2 =
↪   (B[j] - conj(B[i])) * r4;
      A[i] = c1 * d1 + c2 * d2 * r5;
      B[i] = c1 * d2 + c2 * d1;
    }
    A[j] = a1 * b1 + a2 * b2 * r5;
    B[j] = a1 * b2 + a2 * b1;
  }
  fft(A, sz); fft(B, sz);
  vector<int> res(need);
  for(int i = 0; i < need; i++) {
    long long aa = A[i].x + 0.5;
    long long bb = B[i].x + 0.5;
    long long cc = A[i].y + 0.5;
    res[i] = (aa + ((bb % mod) << 15) + ((cc % mod)
↪   << 30))%mod;
  }
  return res;
}

vector<int> pow(vector<int>& a, int p) {
  vector<int> res;
  res.emplace_back(1);
  while(p) {
    if(p & 1) res = multiply(res, a);
    a = multiply(a, a, 1);
    p >>= 1;
  }
  return res;
}
int main() {
  int n, k; cin >> n >> k;
  vector<int> a(10, 0);
  while(k--) {
    int m; cin >> m;
    a[m] = 1;
  }
  vector<int> ans = pow(a, n / 2);
  int res = 0;
  for(auto x: ans) res = (res + 1LL * x * x % mod)
↪   % mod;
  cout << res << '\n';
  return 0;
}

## Online NTT
void solve() {
  f[0]=1; // base case
  for(int i=0; i<=MAX; i++) {
    // Doing the part 1
    f[i+1]=(f[i+1]+f[i]*A[0])%mod;
    f[i+2]=(f[i+2]+f[i]*A[1])%mod;
    if(!i) continue;
    // part 2
    int limit=(i&-i);
    for(int p=2; p<=limit; p*=2) {
      convolve(i-p,i-1,p,min(2*p-1,MAX));
    }
  }
}
void convolve(int l1, int r1, int l2, int r2) {
  int n=max(r1-l1+1,r2-l2+1);
  int t=1;
  while(t<n) t<<=1;
  n=t;
  vector<ll> a(n), b(n);
  for(int i=l1; i<=r1; i++) a[i-l1]=f[i];
  for(int i=l2; i<=r2; i++) b[i-l2]=A[i];
  vector<ll> ret=fft::multiply(a,b);
    for(int i=0; i<ret.size(); i++) {
    int idx=i+l1+l2+1;
    if(idx>MAX) break;
    // adding to the appropriate entry
    f[idx]+=ret[i];
    f[idx]%=mod;
  }
}

## FWHT (AND, OR, XOR)
- Time complexity: O(nlogn)
- AND, OR works for any modulo, XOR works for only
 ↪  prime
- size must be power of two

const ll mod = 998244353;

int add (int a, int b) {
  return a + b < mod? a + b: a + b - mod;
}

int sub (int a, int b) {
  return a - b >= 0? a - b: a - b + mod;
}

ll poww (ll a, ll p, ll mod){
  a %= mod;
  ll ret = 1;
  while (p){
    if (p & 1) {
      ret = ret * a % mod;
    }
    a = a * a % mod;
    p >>= 1;
  }
  return ret;
}

void fwht(vector<int> &a, int inv, int f) {
  int sz = a.size();
  for (int len = 1; 2 * len <= sz; len <<= 1) {
    for (int i = 0; i < sz; i += 2 * len) {
      for (int j = 0; j < len; j++) {
        int x = a[i + j];
        int y = a[i + j + len];

        if (f == 0) {
          if (!inv)  a[i + j] = y, a[i + j + len] =
↪   add(x,  y);
          else  a[i + j] = sub(y, x), a[i + j +
↪   len] = x;
        }
        else if (f == 1) {
          if (!inv)  a[i + j + len] = add(x, y);
          else  a[i + j + len] = sub(y, x);
        }
        else {
          a[i + j] = add(x, y);
          a[i + j + len] = sub(x, y);
        }
      }
    }
  }
}

vector<int> mul(vector<int> a, vector<int> b, int
↪   f) { // 0:AND, 1:OR, 2:XOR
  int sz = a.size();
  fwht(a, 0, f);  fwht(b, 0, f);
  vector<int> c(sz);
  for (int i = 0; i < sz; ++i) {
    c[i] = 1ll * a[i] * b[i] % mod;
  }
  fwht(c, 1, f);
  if (f) {
    int sz_inv = poww(sz, mod - 2, mod);
    for (int i = 0; i < sz; ++i) {
      c[i] = 1ll * c[i] * sz_inv % mod;
    }
  }
  return c;
}

## subset convolution
vector<int> subset_conv (vector<int> a, vector<int>
↪   b) {
  int n = a.size();
  int lg = log2(n);
  vector<int> cnt(n);
  vector<vector<int>> fa(lg + 1, vector<int> (n)),
↪   fb(lg + 1, vector<int> (n)), g(lg + 1,
↪   vector<int> (n));
  for (int i = 0; i < n; ++i) {
    cnt[i] = cnt[i >> 1] + (i & 1);
    fa[cnt[i]][i] = a[i] % mod;
    fb[cnt[i]][i] = b[i] % mod;
  }
  for (int k = 0; k <= lg; ++k) {
    fwht(fa[k], 0, 1);  fwht(fb[k], 0, 1);
  }
  for (int k = 0; k <= lg; ++k) {
    for (int j = 0; j <= k; ++j) {
      for (int i = 0; i < n; ++i) {
        g[k][i] = add(g[k][i], 1ll * fa[j][i] *
↪   fb[k - j][i] % mod);
```

```
      }
    }
  }
  for (int k = 0; k <= lg; ++k) {
    fwht(g[k], 1, 1);
  }
  vector<int> c(n);
  for (int i = 0; i < n; ++i) {
    c[i] = g[cnt[i]][i];
  }
  return c;
}
```

16 CPP

```
## Ordered Set
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
tree_order_statistics_node_update> oset;

oset ost;
ost.order_of_key(495): // return 0-based index of
↪   lower_bound
ost.find_by_order(5): // return iterator of 0-based
↪   index value
ost.erase(); ost.size(), ost.insert(2),
↪   st.lower_bound(x);

## unordered_map
struct chash{
  size_t operator()(const pair<int,int>&x)const{
    return hash<long long>()(((long
↪   long)x.first)^(((long long)x.second)<<32));
  }
};
unordered_map<pair<int, int>, int, chash> maf;
maf.reserve(max_len);
maf.max_load_factor(0.25);

## gp_hash_table:
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

struct chash{
  int operator()(ii p) const {
    return p.first*31 + p.second;
  }
};
gp_hash_table<ii, int, chash> cnt;
```

17 DETERMINANT

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
  int k = i;
  for (int j=i+1; j<n; ++j)
    if (abs (a[j][i]) > abs (a[k][i]))
      k = j;
  if (abs (a[k][i]) < EPS) {
    det = 0;
    break;
  }
  swap (a[i], a[k]);
  if (i != k)
    det = -det;
  det *= a[i][i];
  for (int j=i+1; j<n; ++j)
    a[i][j] /= a[i][i];
  for (int j=0; j<n; ++j)
    if (j != i && abs (a[j][i]) > EPS)
      for (int k=i+1; k<n; ++k)
        a[j][k] -= a[i][k] * a[j][i];
}
```

18 DINIC

```
// V^2E, sqrt(E)E, sqrt(V)E(bpm)
// Effective flows are adj[u][3] where adj[u][3] > 0
ll get_max_flow(vector<array<int, 3>> edges, int n,
↪   int s, int t) {
  vector<array<ll, 4>> adj[n];
  for (auto [u, v, c]: edges) {
    adj[u].push_back({v, (int)adj[v].size(), c, 0});
    adj[v].push_back({u, (int)adj[u].size() - 1, 0,
↪   0});
  }

  ll max_flow = 0;
  while (true) {
    queue<int> q;  q.push(s);
    vector<int> dis(n, -1);  dis[s] = 0;
    while (!q.empty()) {
      int u = q.front();  q.pop();
      for (auto [v, idx, c, f]: adj[u]) {
        if (dis[v] == -1 and c > f) {
          q.push(v);
          dis[v] = dis[u] + 1;
        }
      }
    }
    if (dis[t] == -1)  break;
    vector<int> next(n);
    function<ll(int, ll)> dfs = [&] (int u, ll
↪   flow) {
      if (u == t)  return flow;
      while (next[u] < adj[u].size()) {
        auto &[v, idx, c, f] = adj[u][next[u]++];
        if (c > f and dis[v] == dis[u] + 1) {
          ll bn = dfs(v, min(flow, c - f));
          if (bn > 0) {
            f += bn;
            adj[v][idx][3] -= bn;
            return bn;
          }
        }
      }
      return 0ll;
    };

    while (ll flow = dfs(s, LLONG_MAX)) {
      max_flow += flow;
    }
  }
  return max_flow;
}
```

19 DOMINATOR_TREE

```
const int N = 2e5+5;

vector <int> g[N], rg[N], dtree[N], bucket[N];
int sdom[N], par[N], dom[N], dsu[N], lab[N],
↪   arr[N], rev[N], dpar[N], n, ts, src;

void init(int _n, int s) {
  ts = 0, n = _n, src = s;
  for (int i = 1; i <= n; ++i) {
    g[i].clear(), rg[i].clear(), dtree[i].clear(),
↪   bucket[i].clear();
    sdom[i]=par[i]=dom[i]=dsu[i]=lab[i]=arr[i]=rev[⌋
↪   i]=dpar[i]=0;
  }
}
void dfs(int u) {
   ts++; arr[u] = ts; rev[ts] = u;
   lab[ts] = sdom[ts] = dsu[ts] = ts;
   for(int &v : g[u]) {
      if(!arr[v]) { dfs(v); par[arr[v]] = arr[u]; }
      rg[arr[v]].push_back(arr[u]);
   }
}
inline int root(int u, int x = 0) {
   if(u == dsu[u]) return x ? -1 : u;
   int v = root(dsu[u], x + 1);
   if(v < 0) return u;
   if(sdom[lab[dsu[u]]] < sdom[lab[u]]) lab[u] =
↪   lab[dsu[u]];
   dsu[u] = v; return x ? v : lab[u];
}
void build() {
   dfs(src);
   for(int i=n; i; i--) {
      for(int j : rg[i]) sdom[i] =
↪   min(sdom[i],sdom[root(j)]);
      if(i > 1) bucket[sdom[i]].push_back(i);
      for(int w : bucket[i]) {
         int v = root(w);
         if(sdom[v] == sdom[w]) dom[w] = sdom[w];
         else dom[w] = v;
      } if(i > 1) dsu[i] = par[i];
   }
   for(int i=2; i<=n; i++) {
      int &dm = dom[i];
      if(dm ^ sdom[i]) dm = dom[dm];
      dtree[rev[i]].push_back(rev[dm]);
      dtree[rev[dm]].push_back(rev[i]);
      dpar[rev[i]] = rev[dm];
   }
}
```

20 DP_ON_TREE

```
// Rerooting Technique
vector<array<ll, 2>> down(N), up(N);
void dfs() {
  // calculate down dp
}
void dfs2() {
  ll pref = ?;
  for (auto v: adj[u]) {
    // update up[v] and pref
  }
  reverse(adj[u].begin(), adj[u].end());
  ll suf = ?;
  for (auto v: adj[u]) {
    // update up[v] and suf
  }
  for (auto v: adj[u]) {
    dfs2(v)
  }
}
```

21 DP_OPTIMIZATION

```
## CHT

## Online CHT
const ll IS_QUERY = -(1LL << 62);

struct line {
  ll m, b;
  mutable function <const line*()> succ;

  bool operator < (const line &rhs) const {
    if (rhs.b != IS_QUERY) return m < rhs.m;
    const line *s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s -> b < (s -> m - m) * x;
  }
};

struct CHT : public multiset <line> {
  bool bad (iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y -> m == z -> m && y -> b <= z -> b;
    }
    auto x = prev(y);
    if (z == end()) return y -> m == x -> m && y ->
↪   b <= x -> b;
    return 1.0 * (x -> b - y -> b) * (z -> m - y ->
↪   m) >= 1.0 * (y -> b - z -> b) * (y -> m - x ->
↪   m);
  }

  void add (ll m, ll b) {
    auto y = insert({m, b});
    y -> succ = [=] {return next(y) == end() ? 0 :
↪   &*next(y);};
    if (bad(y)) {erase(y); return;}
    while (next(y) != end() && bad(next(y)))
↪   erase(next(y));
    while (y != begin() && bad(prev(y)))
↪   erase(prev(y));
  }

  ll eval (ll x) {
    auto l = *lower_bound((line) {x, IS_QUERY});
    return l.m * x + l.b;
  }
};

// To find maximum
CHT cht;
cht.add(m, c);
y_max = cht.eval(x);

// To find minimum
CHT cht;
cht.add(-m, -c);
y_min = -cht.eval(x);

## DnC
// Divide an array into k parts
// Minimize the sum of squre of each subarray
ll pref[N], dp[N][N];
void compute(int l, int r, int j, int kl, int kr) {
  if (l > r)  return ;
  int m = (l + r) / 2;
  array<ll, 2> best = {LLONG_MAX, -1};
  for (int k = kl; k <= min(m - 1, kr); ++k) {
    best = min(best, {dp[k][j - 1] + (pref[m] -
↪   pref[k]) * (pref[m] - pref[k]), k});
  }
  dp[m][j] = best[0];
  compute(l, m - 1, j, kl, best[1]);
  compute(m + 1, r, j, best[1], kr);
}

## Knuth
// Divide an array into n parts.
// Cost of each division is subarray sum
// Minimize the cost
ll dp[n][n], opt[n][n];
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    dp[i][j] = LLONG_MAX;
  }
  opt[i][i] = i;
  dp[i][i] = 0;
}
for (int i = n - 2; i >= 0; --i) {
  for (int j = i + 1; j < n; ++j) {
    for (int k = opt[i][j - 1]; k <= min(j - 1ll,
↪   opt[i + 1][j]); ++k) {
      if (dp[i][j] >= dp[i][k] + dp[k + 1][j] +
↪   (pref[j + 1] - pref[i])) {
        dp[i][j] = dp[i][k] + dp[k + 1][j] +
↪   (pref[j + 1] - pref[i]);
        opt[i][j] = k;
      }
    }
  }
}
cout << dp[0][n - 1] << "\n";

## Lichao Tree
const int N = int(5e4 + 2);
const ll INF = ll(1e17);
vector<vector<ll> > tree(4*N, {0, INF});
ll f(vector<ll> line, int x){
return line[0] * x + line[1];
}
void insert(vector<ll> line, int lo = 1, int hi =
↪ N, int i = 1){
  int m = (lo + hi) / 2;
  bool left = f(line, lo) < f(tree[i], lo);
  bool mid = f(line, m) < f(tree[i], m);
  if(mid) swap(tree[i], line);
  if(hi - lo == 1) return;
  else if(left != mid) insert(line, lo, m, 2*i);
  else insert(line, m, hi, 2*i+1);
}
ll query(int x, int lo = 1, int hi = N, int i = 1){
  int m = (lo+hi)/2;
  ll curr = f(tree[i], x);
  if(hi-lo==1) return curr;
  if(x<m) return min(curr, query(x, lo, m, 2*i));
  else return min(curr, query(x, m, hi, 2*i+1));
}
```

22 DSU

```
struct DSU {
  int comps;
  vector<int> par, sz;
  DSU(int n): comps(n), par(n), sz(n,1) {
    iota(par.begin(), par.end(), 0);
  }
  int find(int v) {
    return (par[v] == v)? v: (par[v] =
↪   find(par[v]));
  }
  void unite(int u, int v) {
    u = find(u), v = find(v);
    if(u != v){
      if(sz[u] < sz[v]) {
        swap(u, v);
      }
      par[v] = u;
      sz[u] += sz[v];
      comps--;
    }
  }
  int size(int v) {
    return sz[find(v)];
  }
  bool same_set(int u, int v) {
    return find(u) == find(v);
  }
};
```

23 DSU_ON_TREE

```
void dfs(int u, int p) {
  node[tt] = u;
  tin[u] = tt++, sz[u] = 1, hc[u] = -1;
  for (auto v: adj[u]) {
    if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if (hc[u] == -1 or sz[hc[u]] < sz[v]) {
        hc[u] = v;
      }
    }
  }
  tout[u] = tt - 1;
}
void dsu(int u, int p, int keep) {
  for (int v: adj[u]) {
    if (v != p and v != hc[u]) {
      dsu(v, u, 0);
    }
  }
  if (hc[u] != -1) {
    dsu(hc[u], u, 1);
  }
  for (auto v: adj[u]) {
    if (v != p and v != hc[u]) {
      for (int i = tin[v]; i <= tout[v]; ++i) {
        int w = node[i];
        // get ans in case of ans is related to simple path or pair
      }
      for (int i = tin[v]; i <= tout[v]; ++i) {
        int w = node[i];
        // Add contribution of node w
      }

    }
  }
  // Add contribution of node u
  // get ans in case ans is related to subtree
  if (!keep) {
    for (int i = tin[u]; i <= tout[u]; ++i) {
      int w = node[i];
      // remove contribution of node w
    }
    // Data structure in initial state (empty contribution)
  }
}
dfs(0, 0);  dsu(0, 0, 0);
```

24 DSU_WITH_ROLLBACK

```
struct DSU {
  int comps;
  vector<int> par, rnk;
  stack<array<int, 4>> ops;

  DSU(){}
  DSU(int n): comps(n), par(n), rnk(n) {
    iota(par.begin(), par.end(), 0);
  }

  int find(int u) {
    return (par[u] == u)? u: find(par[u]);
  }

  bool unite(int u, int v) {
    u = find(u), v = find(v);
    if (u == v)  return false;

    if (rnk[u] > rnk[v])  swap(u, v);
    ops.push({u, rnk[u], v, rnk[v]});
    par[u] = v;
    if (rnk[u] == rnk[v])  rnk[v]++;
    return true;
  }

  void rollback() {
    if (ops.empty())  return ;
    auto [u, rnku, v, rnkv] = ops.top();  ops.pop();
    par[u] = u, rnk[u] = rnku;
    par[v] = v, rnk[v] = rnkv;
    comps++;
  }
};
```

25 DS_TRICKS

```
## Max prefix query with insertion only
a1 < a2 < a3 < ... < and b1 < b2 < b3 < ... < bn
// query
auto it = dp.lower_bound(a);
if (it != dp.begin()) {
  mx = max(now, prev(it)->second);
}
// insert
it = dp.upper_bound(a);
if (it != dp.begin() and prev(it)->second >= b) {
  continue;
}
it = dp.insert(it, {a, b});
it->second = b;
while (next(it) != dp.end() and next(it)->second <= b) {
  dp.erase(next(it));
}
```

26 DYNAMIC_CONNECTIVITY

```
const int Q = 1e5+5;
vector<array<int, 2>> t[4 * Q];
vector<int> ans(Q);
int q;
struct DSU {
  int n, comps;
  vector<int> par, rnk;
  stack<array<int, 4>> ops;
  DSU(){}
  DSU(int n): n(n), comps(n), par(n), rnk(n) {
    iota(par.begin(), par.end(), 0);
  }
  int find(int u) {
    return (par[u] == u)? u: find(par[u]);
  }
  bool unite(int u, int v) {
    u = find(u), v = find(v);
    if (u == v)  return false;
    comps--;
    if (rnk[u] > rnk[v])  swap(u, v);
    ops.push({u, rnk[u], v, rnk[v]});
    par[u] = v;
    if (rnk[u] == rnk[v])  rnk[v]++;
    return true;
  }
  void rollback() {
    if (ops.empty())  return ;
    auto [u, rnku, v, rnkv] = ops.top();  ops.pop();
    par[u] = u, rnk[u] = rnku;
    par[v] = v, rnk[v] = rnkv;
    comps++;
  }
} dsu;

void add(int l, int r, array<int, 2> ed, int u = 1, int s = 0, int e = q) {
  if (r < s or e < l)  return ;
  if (l <= s and e <= r) {
    t[u].push_back(ed);
    return ;
  }
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  add(l, r, ed, v, s, m);
  add(l, r, ed, w, m + 1, e);
}
void go(int u = 1, int s = 0, int e = q) {
  int rmv = 0;
  for (auto &ed: t[u])  rmv += dsu.unite(ed[0], ed[1]);
  if (s == e)  ans[s] = dsu.comps;
  else {
    int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
    go(v, s, m);
    go(w, m + 1, e);
  }
  while (rmv--)  dsu.rollback();
}
```

27 DnC

```
- Divide an array into k parts
- Minimize the sum of squre of each subarray

ll pref[N], dp[N][N];
void compute(int l, int r, int j, int kl, int kr) {
  if (l > r)  return ;
  int m = (l + r) / 2;
  array<ll, 2> best = {LLONG_MAX, -1};
  for (int k = kl; k <= min(m - 1, kr); ++k) {
    best = min(best, {dp[k][j - 1] + (pref[m] - pref[k]) * (pref[m] - pref[k]), k});
  }
  dp[m][j] = best[0];
  compute(l, m - 1, j, kl, best[1]);
  compute(m + 1, r, j, best[1], kr);
}
for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
    dp[i][j] = 1e17;
  }
}
dp[0][0] = 0;
for (int j = 1; j <= k; ++j) {
  compute(1, n, j, 0, n - 1);
}
cout << dp[n][k] << "\n";
```

28 DnC_SRQ

```
void dnc(int l, int r, vector<int> idx) {
  if (idx.empty())  return;
  if (l==r) {
    for (auto i: idx)  ans[i] = a[l];
      return ;
  }
  int m = (l + r) / 2;
  vectro<int> left(m), right(m + 1);
  for (int i = m - 1; i >= l; --i) {
    left[i] = min(a[i], left[i + 1]);
  }
  for (int i = m; i <= r; ++i) {
    right[i] = min(right[i - 1], a[i]);
  }
  vector<int> nxt[2];
  for (int i: idx) {
    if (l[i] <= m and m <= r[i]) {
      ans[i] = min(left[l[i]], right[r[i]]);
    }
    else {
      nxt[i >= m].push_back(i);
    }
  }
  dnc(l, m, nxt[0]);
  dnc(m + 1, r nxt[1]);
}
```

29 EULER_WALK

```
## Directed graph
vector<int> euler_cycle(vector<int> *adj, int s =
 ↪  0) {
  vector<int> cycle;
  function<void(int)> dfs = [&] (int u) {
    while (!adj[u].empty()) {
      int v = adj[u].back();
      adj[u].pop_back();
      dfs(v);
    }
    cycle.push_back(u);
  };

  dfs(s);
  reverse(cycle.begin(), cycle.end());
  return cycle;
}

## Undirected graph
vector<int> euler_cycle(vector<int> *adj,
 ↪   vector<int> *des_idx, vector<int> *done, int s
 ↪   = 0) {
  vector<int> cycle;

  function<void(int)> dfs = [&] (int u) {
    while (!adj[u].empty()) {
      int i = adj[u].size() - 1;
      if (done[u][i]) {
        adj[u].pop_back();
        continue;
      }
      int v = adj[u][i];
      adj[u].pop_back();
      done[u][i] = 1;
      done[v][des_idx[u][i]] = 1;
      dfs(v);

    }
    cycle.push_back(u);
  };

  dfs(s);
  return cycle;
}

int n, m;  cin >> n >> m;
vector<int> adj[n], des_idx[n], done[n];
vector<int> deg(n);
for (int e = 0; e < m; ++e) {
  int u, v;  cin >> u >> v;  u--, v--;
  des_idx[u].push_back(adj[v].size());
  des_idx[v].push_back(adj[u].size());
  adj[u].push_back(v);
  adj[v].push_back(u);
  done[u].push_back(0);
  done[v].push_back(0);
  deg[u]++, deg[v]++;
}

for (int u = 0; u < n; ++u) {
  if (deg[u] & 1) {
    cout << "IMPOSSIBLE\n";
    return ;
  }
}

vector<int> cycle = euler_cycle(adj, des_idx, done,
 ↪  0);

if (cycle.size() != m + 1) {
  cout << "IMPOSSIBLE\n";
  return ;
}
```

30 FENWICK_TREE

```
## 2D BIT
int n;
int ft[N][N];

void add(int x, int y, int val){
 x++, y++;
 int idx = x;
 while (idx<=n){
   int idy = y;
   while (idy<=n){
     ft[idx][idy] += val;
     idy += idy & -idy;
   }
   idx += idx & -idx;
 }
}

int csum(int x, int y){
 x++, y++;
 int ret = 0;
 int idx = x;
 while (idx > 0){
   int idy = y;
   while (idy > 0){
     ret += ft[idx][idy];
     idy -= idy & -idy;
   }
   idx -= idx & -idx;
 }
 return ret;
}

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

31 GAUSSIAN_ELIMINATION

```
const double eps=1e-9;
const int INF=2;

typedef vector<double> vd;
int gauss(vector<vd>& a, vd& ans) {
  int n = a.size();
  int m = a[0].size()-1;

  vector<int> where(m, -1);
  for(int col=0, row=0; row<n and col<m; ++col,
 ↪   ++row) {
    int sel = row;
    for(int i=row; i<n; ++i){
      if(abs(a[i][col]) > abs(a[sel][col])) sel = i;
    }
    if(abs(a[sel][col]) < eps)  continue;
    for(int j=col; j<=m; ++j) swap(a[sel][j],
 ↪   a[row][j]);
    where[col] = row;
    for(int i=0; i<n; i++){
      if(i!=row){
        double c = a[i][col] / a[row][col];
        for(int j=col; j<=m; ++j){
          a[i][j] -= a[row][j]*c;
        }
      }
    }
  }

  ans.assign(m, 0);
  for (int j = 0; j < m; ++j){
    if(where[j] != -1){
      ans[j] = a[where[j]][m] / a[where[j]][j];
    }
  }

  for (int i = 0; i < n; ++i){
    double sum = 0;
    for (int j = 0; j < m; ++j){
      sum += ans[j] * a[i][j];
    }
    if(abs(sum-a[i][m]) > eps)  return 0;
  }

  for (int j = 0; j < m; ++j){
    if(where[j] == -1)  return INF;
}
  return 1;
}
```

32 GRAPH_CYCLE

```
## Floyd cycle finding algorithm for successor
 ↪   graph:
int hare = x0, tort = x0;
do {
    hare = succ(hare);
    hare = succ(hare);
    tort = succ(tort);
} while (hare != tort);

tort = x0;
while (tort != hare) {
    hare = succ(hare);
    tort = succ(tort);
}
// now hare and tort are the starting node of the
 ↪   cycle
```

```
int len = 0;
do {
  tort = succ(tort);
  len++;
} while (tort != hare);

## Number of basis simple cycle in an u-graph
E - V + number of connected components (using DSU)
Number of back-edge in dfs tree (using DFS)

## Number of eulerian subgraph in an u-graph
2^basis_cycle
```

33 GRAY_CODE

```
int gc(int n){
  return n^(n>>1);
}

int gc_to_dec(int g){
  int d=0;
  while (g){
    d ^= g;
    g >>= 1;
  }
  return d;
}
```

34 HALF_PLANE_INTERSECTION

```
// Redefine epsilon and infinity as necessary. Be mindful of precision errors.
const long double eps = 1e-9, inf = 1e9;

// Basic point/vector struct.
struct Point {
    long double x, y;
    explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {}
    // Addition, substraction, multiply by constant, dot product, cross product.
    friend Point operator + (const Point& p, const Point& q) {
        return Point(p.x + q.x, p.y + q.y);
    }
    friend Point operator - (const Point& p, const Point& q) {
        return Point(p.x - q.x, p.y - q.y);
    }
    friend Point operator * (const Point& p, const long double& k) {
        return Point(p.x * k, p.y * k);
    }
    friend long double dot(const Point& p, const Point& q) {
        return p.x * q.x + p.y * q.y;
    }
    friend long double cross(const Point& p, const Point& q) {
        return p.x * q.y - p.y * q.x;
    }
};

// Basic half-plane struct.
struct Halfplane {
    // 'p' is a passing point of the line and 'pq' is the direction vector of the line.
    Point p, pq;
    long double angle;

    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }
    // Check if point 'r' is outside this half-plane.
    // Every half-plane allows the region to the LEFT of its line.
    bool out(const Point& r) {
        return cross(pq, r - p) < -eps;
    }
    // Comparator for sorting.
    bool operator < (const Halfplane& e) const {
        return angle < e.angle;
    }
    // Intersection point of the lines of two half-planes. It is assumed they're never parallel.
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};

// Actual algorithm
vector<Point> hp_intersect(vector<Halfplane>& H) {
    Point box[4] = {  // Bounding box in CCW order
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };
    for(int i = 0; i<4; i++) { // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    }
    // Sort by angle and start algorithm
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {
        // Remove from the back of the deque while last half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
            dq.pop_back();
            --len;
        }
        // Remove from the front of the deque while first half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }
        // Special case check: Parallel half-planes
        if (len > 0 && fabsl(cross(H[i].pq, dq[len-1].pq)) < eps) {
            // Opposite parallel half-planes that ended up checked against each other.
            if (dot(H[i].pq, dq[len-1].pq) < 0.0)
                return vector<Point>();
            // Same direction half-plane: keep only the leftmost half-plane.
            if (H[i].out(dq[len-1].p)) {
                dq.pop_back();
                --len;
            }
            else continue;
        }
        // Add new half-plane
        dq.push_back(H[i]);
        ++len;
    }
    // Final cleanup: Check half-planes at the front against the back and vice-versa
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back();
        --len;
    }
    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }
    // Report empty intersection if necessary
    if (len < 3) return vector<Point>();
    // Reconstruct the convex polygon from the remaining half-planes.
    vector<Point> ret(len);
    for(int i = 0; i+1 < len; i++) {
        ret[i] = inter(dq[i], dq[i+1]);
    }
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

35 HASHING

```
// Hashing
// Hashvalue(l...r) = hsh[l] - hsh[r + 1] * base ^ (r - l + 1);
// Must call preprocess
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAX = 100009;
ll mods[2] = {1000000007, 1000000009};
//Some back-up primes: 1072857881, 1066517951, 1040160883
ll bases[2] = {137, 281};
ll pwbase[3][MAX];

void Preprocess(){
  pwbase[0][0] = pwbase[1][0] = 1;
  for(ll i = 0; i < 2; i++){
    for(ll j = 1; j < MAX; j++){
      pwbase[i][j] = (pwbase[i][j - 1] * bases[i]) % mods[i];
    }
  }
}
struct Hashing{
  ll hsh[2][MAX];
  string str;
  Hashing(){}
  Hashing(string _str) {str = _str; memset(hsh, 0,
↪  sizeof(hsh)); build();}
  void Build(){
    for(ll i = str.size() - 1; i >= 0; i--){
      for(int j = 0; j < 2; j++){
        hsh[j][i] = (hsh[j][i + 1] * bases[j] +
↪   str[i]) % mods[j];
        hsh[j][i] = (hsh[j][i] + mods[j]) % mods[j];
      }
    }
  }

  pair<ll,ll> GetHash(ll i, ll j){
    assert(i <= j);
    ll tmp1 = (hsh[0][i] - (hsh[0][j + 1] *
↪   pwbase[0][j - i + 1]) % mods[0]) % mods[0];
    ll tmp2 = (hsh[1][i] - (hsh[1][j + 1] *
↪   pwbase[1][j - i + 1]) % mods[1]) % mods[1];
    if(tmp1 < 0) tmp1 += mods[0];
    if(tmp2 < 0) tmp2 += mods[1];
    return make_pair(tmp1, tmp2);
  }
};

/***
    * Everything is 0 based
    * Call precal() once in the program
    * Call update(1,0,n-1,i,j,val) to update the
↪   value of position
      i to j to val, here n is the length of the
↪   string
    * Call query(1,0,n-1,L,R) to get a node
↪   containing hash
      of the position [L:R]
    * Before any update/query
        - Call init(str) where str is the string to
↪   be hashed
        - Call build(1,0,n-1)
***/

namespace strhash {
  int n;
  const int MAX = 100010;
  int ara[MAX];
  const int MOD[] = {2078526727, 2117566807};
  const int BASE[] = {1572872831, 1971536491};

  int BP[2][MAX], CUM[2][MAX];

  void init(char *str) {
    n = strlen(str);
    for(int i=0;i<n;i++) ara[i] = str[i]-'0'+1; ///
↪   scale str[i] if needed
  }

  void precal() {
    BP[0][0] = BP[1][0] = 1;
    for(int i=1;i<MAX;i++) {
      BP[0][i] = ( BP[0][i-1] * (long long) BASE[0]
↪   ) % MOD[0];
      BP[1][i] = ( BP[1][i-1] * (long long) BASE[1]
↪   ) % MOD[1];
    }
  }

  struct node {
    int sz;
    int h[2];
    node() {}
  } tree[4*MAX];

  int lazy[4*MAX];
  inline node Merge(node a,node b) {
    node ret;
    ret.h[0] = ( ( a.h[0] * (long long) BP[0][b.sz]
↪   ) + b.h[0] ) % MOD[0];
    ret.h[1] = ( ( a.h[1] * (long long) BP[1][b.sz]
↪   ) + b.h[1] ) % MOD[1];
    ret.sz = a.sz + b.sz;
    return ret;
  }

  inline void build(int n,int st,int ed) {
    if(st==ed) {
      tree[n].h[0] = tree[n].h[1] = ara[st];
      tree[n].sz = 1;
      return;
    }
    int mid = (st+ed)>>1;
    build(n+n,st,mid);
    build(n+n+1,mid+1,ed);

    tree[n] = Merge(tree[n+n],tree[n+n+1]);
  }

  inline void update(int n,int st,int ed,int id,int
↪   v) {
    if(st>id or ed<id) return;
    if(st==ed and ed==id) {
      tree[n].h[0] = tree[n].h[1] = v;
      return;
    }
    int mid = (st+ed)>>1;
    update(n+n,st,mid,id,v);
    update(n+n+1,mid+1,ed,id,v);
    tree[n] = Merge(tree[n+n],tree[n+n+1]);
  }
  inline node query(int n,int st,int ed,int i,int
↪   j){
    if(st>=i and ed<=j) return tree[n];
    int mid = (st+ed)/2;
    if(mid<i) return query(n+n+1,mid+1,ed,i,j);
    else if(mid>=j) return query(n+n,st,mid,i,j);
    else return Merge(query(n+n,st,mid,i,j),query(n⌋
↪   +n+1,mid+1,ed,i,j));
  }
}
```

36 HLD

```
int tt, tin[N], tout[N], sz[N], par[N][LG], hvc[N];
void dfs(int u, int p) {
  tin[u] = tt++, sz[u] = 1, par[u][0] = p;
  for (int j = 1; j < LG; ++j) {
    par[u][j] = par[par[u][j-1]][j-1];
  }
  int mx = 0;
  for (int &v: adj[u]) {
    if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if (sz[v] > mx) {
        mx = sz[v];
        hvc[u] = v;
      }
    }
  }
  tout[u] = tt-1;
}
int ch_cnt, idx_cnt, chno[N], chd[N], idx[N];
void hld(int u, int p) {
  if(chd[ch_cnt] == -1) {
    chd[ch_cnt] = u;
  }
  chno[u] = ch_cnt, idx[u] = idx_cnt++;
  if(hvc[u] != -1) {
    hld(hvc[u], u);
  }
  for (int &v: adj[u]) {
    if (v != p and v != hvc[u]) {
      ch_cnt++;
      hld(v, u);
    }
  }
}
void ?node_update(int u, int x) {
  ?update(idx[u], x);
}
void ?pupdate_up(int u, int anc) {
  if (chno[u] == chno[anc]) {
    return ?rupdate(idx[anc], idx[u]);
  }
  ?rupdate(idx[chd[chno[u]]], idx[u]);
  ?pupdate_up(par[chd[chno[u]]][0], anc);
}
void ?pupdate(int u, int v) {
  int l = lca(u, v);
  ?pupdate_up(u, l);
  ?pupdate_up(v, l);
}
ll ?node_query(int u) {
  return ?query(idx[u]);
}
int ?pquery_up(int u, int anc) {
  if (chno[u] == chno[anc]) {
    return ?rquery(idx[anc], idx[u]);
  }
  return f(?rquery(idx[chd[chno[u]]], idx[u]),
↪   ?pquery_up(par[chd[chno[u]]][0], anc));
}
int ?rquery(int u, int v) {
  int l = lca(u, v);
  return f(?pquery_up(u, l), ?pquery_up(v, l));
}
adj[u].clear(); hvc[u] = -1;
tt = 0; dfs(0, 0);

chd[ch] = -1;
ch_cnt = 0, idx_cnt = 0; hld(0, 0);
```

37 HOPCROFT_KARP

```
// 1-based
const int N = 1e5+5, INF = 1e8 + 5;

vector <int> g[N];
int n, e, match[N], dist[N];

bool bfs() {
  queue <int> q;
  for (int i = 1; i <= n; ++i) {
    if (!match[i]) dist[i] = 0, q.emplace(i);
    else dist[i] = INF;
  }
  dist[0] = INF;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    if (!u) continue;
    for (int v : g[u]) {
      if (dist[match[v]] == INF) {
        dist[match[v]] = dist[u] + 1,
        q.emplace(match[v]);
      }
    }
  }
  return dist[0] != INF;
}

bool dfs (int u) {
  if (!u) return 1;
  for (int v : g[u]) {
    if (dist[match[v]] == dist[u] + 1 and
↪   dfs(match[v])) {
      match[u] = v, match[v] = u;
      return 1;
    }
  }
  dist[u] = INF;
  return 0;
}

int hopcroftKarp() {
  int ret = 0;
  while (bfs()) {
    for (int i = 1; i <= n; ++i) {
      ret += !match[i] and dfs(i);
    }
  }
  return ret;
}
```

38 HUNGARIAN_ALGORITHM

```
template<typename T>
pair<T, vector<int>> MinAssignment(const
↪   vector<vector<T>> &c) {
  int n = c.size(), m = c[0].size();       //
↪   assert(n <= m);
  vector<T> v(m), dist(m);                 // v:
↪   potential
  vector<int> L(n, -1), R(m, -1);          //
↪   matching pairs
  vector<int> idx(m), prev(m);
  iota(idx.begin(), idx.end(), 0);
  auto residue = [&](int i, int j) { return c[i][j]
↪   - v[j]; };
  for (int f = 0; f < n; ++f) {
    for (int j = 0; j < m; ++j) {
      dist[j] = residue(f, j); prev[j] = f;
    }
    T w; int j, l;
    for (int s = 0, t = 0;;) {
      if (s == t) {
        l = s; w = dist[idx[t++]];
        for (int k = t; k < m; ++k) {
          j = idx[k]; T h = dist[j];
          if (h <= w) {
            if (h < w) { t = s; w = h; }
            idx[k] = idx[t]; idx[t++] = j;
          }
        }
        for (int k = s; k < t; ++k) {
          j = idx[k];
          if (R[j] < 0) goto aug;
        }
      }
      int q = idx[s++], i = R[q];
      for (int k = t; k < m; ++k) {
        j = idx[k];
        T h = residue(i,j) - residue(i,q) + w;
        if (h < dist[j]) {
          dist[j] = h; prev[j] = i;
          if (h == w) {
            if (R[j] < 0) goto aug;
            idx[k] = idx[t]; idx[t++] = j;
          }
        }
      }
    }
  aug:
    for(int k = 0; k < l; ++k)
      v[idx[k]] += dist[idx[k]] - w;
    int i;
    do {
      R[j] = i = prev[j];
      swap(j, L[i]);
    } while (i != f);
  }
  T ret = 0;
  for (int i = 0; i < n; ++i) {
    ret += c[i][L[i]]; // (i, L[i]) is a solution
  }
  return {ret, L};
}
```

39 JOSEPHUS

```
## k = 2
if n = 2^p + e; (p large as possible, e >= 0)
then survivor = 2 * e + 1;
for (int i = 1; i <= n; i++) {
  int x = k * i;  // k = 2
  while (x > n) x = (k * (x - n) - 1) / (k - 1);
  cout << x << " ";
}
int kth(int n, int k, int f = 0) {
  if (n == 1) {
    assert(k == 1);
    return 1;
  }
  if (f == 0) {
    if (2 * k <= n) {
      return 2 * k;
    }
    else {
      f = n & 1;
      k -= n / 2;
      n -= n / 2;
      return 2 * kth(n, k, f) - 1;
    }
  }
  else {
    if (2 * k - 1 <= n) {
      return 2 * k - 1;
    }
    else {
      f = n & 1 ^ 1;
      k -= (n + 1) / 2;
      n -= (n + 1) / 2;
      return 2 * kth(n, k, f);
    }
  }
}

## k != 2
oset s;
for (int i = 1; i <= n; i++) s.insert(i);
for (int sz = n, ord = 0; sz > 0; --sz) {
  ord = (ord + k) % sz;
  auto it = s.find_by_order(ord);
  cout << *it << ' ';
  s.erase(it);
}
```

40 KMP

```
vector<int> get_pi(string& s){
 int n = s.size();
 vector<int> pi(n);
 for (int k = 0, i = 1; i < n; ++i){
   if(s[i] == s[k])  pi[i] = ++k;
   else if(k == 0) pi[i] = 0;
   else  k = pi[k-1], --i;
 }
 return pi;
}
// Period =  n % (n - pi.back() == 0)? n -
↪   pi.back(): n
// Borders = pi.back(), pi[pi.back() - 1], ...
// Prefix palindrome: s + "#" + rev(s)
//  Number of occurrences of each prefix:
vector<int> pref_occur(vector<int> &pi) {
  int n = pi.size();
  vector<int> pref_occur(n + 1);
  for (int i = 0; i < n; ++i) {
    pref_occur[pi[i]]++;
  }
  for (int len = n; len > 0; --len) {
    pref_occur[pi[len - 1]] += pref_occur[len];
    pref_occur[len]++;
  }
  return pref_occur;
}
// Find the length of the longest proper suffix of
↪   a suffix which also its prefix
```

```
// Reverse -> Find prefix function -> Reverse
// Find minimum length string such that given strings occur as substring
```

41 KNUTH_OPTIMIZATION

```
// Divide an array into n parts.
// Cost of each division is subarray sum
// Minimize the  cost
ll dp[n][n], opt[n][n];

for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    dp[i][j] = LLONG_MAX;
  }
  opt[i][i] = i;
  dp[i][i] = 0;
}

for (int i = n - 2; i >= 0; --i) {
  for (int j = i + 1; j < n; ++j) {
    for (int k = opt[i][j - 1]; k <= min(j - 1ll, opt[i + 1][j]); ++k) {
      if (dp[i][j] >= dp[i][k] + dp[k + 1][j] + (pref[j + 1] - pref[i])) {
        dp[i][j] = dp[i][k] + dp[k + 1][j] + (pref[j + 1] - pref[i]);
        opt[i][j] = k;
      }
    }
  }
}

cout << dp[0][n - 1] << "\n";
```

42 KRUSKAL

```
vector<Edge> mst(){
 vector<Edge> res;
 int cnt=0;
 sort(ed.begin(), ed.end());
 DSU dsu(n);
 for(Edge e: ed){
   if(!dsu.same_set(e.u, e.v)){
     res.push_back(e);
     dsu.unite(e.u,e.v);
     if(++cnt >= n-1)  break;
   }
 }
 return res;
}
```

43 LICHAO_TREE

```
const int N = int(5e4 + 2);
const ll INF = ll(1e17);

vector<vector<ll> > tree(4*N, {0, INF});

ll f(vector<ll> line, int x){
  return line[0] * x + line[1];
}

void insert(vector<ll> line, int lo = 1, int hi = N, int i = 1){
  int m = (lo + hi) / 2;
  bool left = f(line, lo) < f(tree[i], lo);
  bool mid = f(line, m) < f(tree[i], m);

  if(mid) swap(tree[i], line);

  if(hi - lo == 1) return;
  else if(left != mid) insert(line, lo, m, 2*i);
  else insert(line, m, hi, 2*i+1);
}

ll query(int x, int lo = 1, int hi = N, int i = 1){
  int m = (lo+hi)/2;
  ll curr = f(tree[i], x);
  if(hi-lo==1) return curr;
  if(x<m) return min(curr, query(x, lo, m, 2*i));
  else return min(curr, query(x, m, hi, 2*i+1));
}
```

44 MANACHER

```
// p[0][i] = half length of longest even palindrome around pos i-1, i and starts at i-p[0][i] and ends at i+p[0][i]-1
// p[1][i] = longest odd (half rounded down) palindrome around pos i and starts at i-p[1][i] and ends at i+p[1][i]
vector<vector<int>> manacher(string &s) {
  int n = s.size();
  vector<vector<int>> p(2, vector<int> (n));
  for (int z = 0; z < 2; ++z) {
    for (int i=0, l=0, r=0; i<n; ++i) {
      int t = r-i+!z;
      if (i<r) {
        p[z][i] = min(t, p[z][l+t]);
      }
      int L = i-p[z][i], R = i+p[z][i]-!z;
      while (L>=1 and R+1<n and s[L-1]==s[R+1]) {
        p[z][i]++, L--, R++;
      }
      if (R>r) {
        l=L, r=R;
      }
    }
  }
  return p;
}
```

45 MATRIX_EXPO

```
using row = vector<int>;
using matrix = vector<row>;
matrix unit_mat(int n) {
  matrix I(n, row(n));
  for (int i = 0; i < n; ++i){
    I[i][i] = 1;
  }
  return I;
}
matrix mat_mul(matrix a, matrix b) {
  int m = a.size(), n = a[0].size();
  int p = b.size(), q = b[0].size();
  // assert(n==p);
  matrix res(m, row(q));
  for (int i = 0; i < m; ++i){
    for (int j = 0; j < q; ++j){
      for (int k = 0; k < n; ++k){
        res[i][j] = (res[i][j] + a[i][k]*b[k][j]) % mod;
      }
    }
  }
  return res;
}
matrix mat_exp(matrix a, int p) {
  int m = a.size(), n = a[0].size();
  // assert(m==n);
  matrix res = unit_mat(m);
  while (p) {
    if (p&1)  res = mat_mul(a, res);
    a = mat_mul(a, a);
    p >>= 1;
  }
  return res;
}
```

46 MCF

```
struct MCF {
  int n;
  vector<vector<array<ll, 5>>> adj;    // v, pos of u in v, cap, cost, flow
  vector<ll> dis, par, pos;

  MCF(int n): n(n), adj(n), dis(n), par(n), pos(n) {}

  void add_edge(int u, int v, int cap, int cost) {
    adj[u].push_back({v, adj[v].size(), cap, cost, 0});
    adj[v].push_back({u, adj[u].size() - 1, 0, -cost, 0});
  }

  ll spfa(int s, int t) {
    dis.assign(n, INF);
    vector<ll> mn_cap(n, INF), inq(n);
    queue<int> q;
    q.push(s), inq[s] = 1, dis[s] = 0;

    while (!q.empty()) {
      int u = q.front();  q.pop();
      inq[u] = 0;
      for (int i = 0; i < adj[u].size(); ++i) {
        auto [v, idx, cap, cost, flow] = adj[u][i];
        if (cap > flow and dis[v] > dis[u] + cost) {
          dis[v] = dis[u] + cost;
          par[v] = u;
          pos[v] = i;
          mn_cap[v] = min(mn_cap[u], cap - flow);
          q.push(v);
          inq[v] = 1;
        }
      }
    }
    return (mn_cap[t] == INF? 0: mn_cap[t]);
  }
  array<ll, 2> get(int s, int t, ll max_flow = INF) {
    ll flow = 0, mc = 0;
    while (ll f = min(spfa(s, t), max_flow - flow)) {
      flow += f;
      mc += f * dis[t];
      int u = t;
      while (u != s) {
        int p = par[u];
        adj[p][pos[u]][4] += f;
        adj[u][adj[p][pos[u]][1]][4] -= f;
        u = p;
      }
    }
    return {flow, mc};
  }
};
MCF mcf(n);
for (int e = 0; e < m; ++e) {
  int u, v, r, c;  cin >> u >> v >> r >> c;  u--, v--;
  mcf.add_edge(u, v, r, c);
}
auto [f, mc] = mcf.get(0, n - 1, k);
```

47 MONOTONIC_STACK

```
vector<int> nsel(vector<int> &a) {
  int n = a.size();
  vector<int> nsel(n);
  stack<int> st;
  st.push(-1);
  for (int i = 0; i < n; ++i) {
    while (st.top() != -1 and a[st.top()] >= a[i]) {
      st.pop();
    }
    nsel[i] = st.top();
    st.push(i);
  }
  return nsel;
}
```

48 MO_ALOGO

```
vector<array<int, 4>> cu(m);
for (int i = 0; i < m; ++i) {
  auto &[b, l, r, idx] = cu[i];
  cin >> l >> r;  l--;
  b = r / B;
  idx = i;
}
sort(cu.begin(), cu.end());

int s = 0, e = -1;
for (auto [b, l, r, i]: cu) {
  while (l < s)  add(--s);
  while (e < r)  add(++e);
  while (s < l)  remove(s++);
  while (r < e)  remove(e--);
  ans[i] = cur_ans;
}
```

49 NOTE

Fibonacci Number: gcd(F$_m$,F$_n$) $= F_gcd(m,n)$

Lucas Number: L(n) = L(n-1) + L(n-2); L(0)=2, L(1)=1 - Number of edge cover of a cycle graph C$_n isL_n$

Catalan Number: C(n+1) = C(0)C(n) + C(1)C(n-1) + ... + C(n)C(0) C(n) = nCr(2n, n) - nCr(2n, n + 1) C(n) = nCr(2n, n) / (n + 1)

- The number of valid bracket sequences of length 2n when a prefix of the sequence is given (among f are first bracket and s are second bracket) C(2n, f, s) = ncr(2n-(f+s), n-f) - ncr(2n-(f+s), (n-f)+(f-s)+1)

Stirling Number of second Kind: "Number of ways to partition a set of n labelled objects into k nonempty unlabelled subsets" S(n, k) = S(n-1, k-1) + k * S(n-1, k) S(n, 1) = 1, S(n, n) = 1

- Time Complexity: $O(k\log n)$ ll get$_s n2(intn, intk)llsn2 = 0; for(inti = 0; i <= k; ++i)llnow = nCr(k,i) * powmod(k-i,n,mod)if Numberofwaystocolora1*ngridwithkcolorssuchthateachcolorisusedatleastonce = k! * sn2(n,k)$

Derangement: "A derangement is a permutation of the elements of a set, such that no element appear in its original position." 1, 0, 1, 2, 9, 44, 265, ... D(n) = (n-1) (D(n-1) + D(n-2)) = n * D(n-1) + (-1)$^n D(0) = 1, D(1) = 0$

Partition number: for (int i = 1; i <= n; ++i) pent[2 * i - 1] = i * (3 * i - 1) / 2; pent[2 * i] = i * (3 * i + 1) / 2; p[0] = 1; for (int i = 1; i <= n; ++i) p[i] = 0; for (int j = 1, k = 0; pent[j] <= i; ++j) if (k < 2) p[i] = add(p[i], p[i - pent[j]]); else p[i] = sub(p[i], p[i - pent[j]]); ++k, k = 3;

Ballot Theorem: Suppose that in an election, candidate A receives a votes and candidate B receives b votes, where a kb for some positive integer k. Compute the number of ways the ballots can be ordered so that A maintains more than k times as many votes as B throughout the counting of the ballots. The solution to the ballot problem is ((a kb)/(a+b)) * C(a+b, a)

Classical Problem F(n, k) = number of ways to color n objects using exactly k colors

Let G(n, k) be the number of ways to color n objects using no more than k colors.

Then, F(n, k) = G(n, k) - C(k, 1) * G(n, k-1) + C(k, 2) * G(n, k-2) - C(k, 3) * G(n, k-3) ...

Determining G(n, k) :

Suppose, we are given a 1 * n grid. Any two adjacent cells can not have same color. Then, G(n, k) = k * ((k-1)$^(n-1)$)

If no such condition on adjacent cells. Then, G(n, k) = k ^{n}

/* 1/(1 - x) = 1 + x + x$^2 + x^3 + ...1/(1-ax) = 1 + ax + (ax)^2 + (ax)^3 + ...1/(1-x)^2 = 1 + 2x + 3x^2 + 4x^3 + ...1/(1-x)^3 = C(2,2) + C(3,2)x + C(4,2)x^2 + C(5,2)x^3 + ...1/(1-ax)^(k+1) = 1 + C(1+k,k)(ax) + C(2+k,k)(ax)^2 + C(3+k,k)(ax)^3 + ...x(x+1)(1-x)^-3 = 1 + x + 4x^2 + 9x^3 + 16x^4 + 25x^5 + ...e^x = 1 + x + (x^2)/2! + (x^3)/3! + (x^4)/4! + ... */$

/* Extended Binomial Theorem C(-n, r) = C(n + r - 1, r) * (-1)$^r //n >= 0andr >= 0C(n,r) = (n(n-1)(n-2)...(n-(r-1)))/r!//worksforanyn(x+a)^-n = a^(-n) + C(-n,1)x^1a^(-n-1) + C(-n,2)x^2a^(-n-2) + C(-n,3)x^3a^(-n-3) + ... */$

$$\sum_{i=1}^{n} \gcd(i,n) = \sum_{d|n} d\phi(\frac{n}{d})$$

$1^4 + 2^4 + 3^4 + ... + n^4 = \frac{n(n+1)(2n+1)(3n2+3n+-1)}{30}$

$S_{(a,n)} = 1^a + 2^a + 3^a + 4^a + ... + n^a$

$S_{(a,n)} = \frac{1}{a+1}[n^{a+1} - (-1)^a + \sum_{i=2}^{a}(-1)^i\binom{a+1}{i}S_{(a-i+1,n)}]$

$1.2 + 2.3 + 3.4 + ... = \frac{1}{3}n(n+1)(n+2)$

$\sum_{i=1}^{n} f_k(i) = \frac{1}{k+1}n(n+1)(n+2)...(n+k) = \frac{1}{k+1}\frac{(n+k)!}{(n-1)!}$

$\sum_{i=0} nix^i = 1 + 2x^2 + 3x^3 + 4x^4 + 5x^5 + ... + nx^n = \frac{(x-(n+1)x^{n+1} + nx^{n+2})}{(x-1)^2}$

Condiational Probability:

$$P(A|B) = \frac{P(A\cap B)}{P(B)}$$

Bayes' Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

50 NUMBER_THEORY

```
ll mulmod(ll a, ll b, ll m){
  a%=m;
  ll res = 0;
  while(b > 0){
    if(b&1)    res=(res+a)%m;
    a=(a<<1)%m;
    b>>=1;
  }
  return res;
}
ll powmod(ll a, ll b, ll m){
  a%=m;
  ll res = 1;
  while(b > 0){
    if(b&1)    res = mulmod(res, a, m);
    a = mulmod(a, a, m);
    b >>= 1;
  }
  return res;
}
ll ext_gcd(ll a, ll b, ll &x, ll &y){
  if(b == 0){
    x=1, y=0;
    return a;
  }
  ll g = ext_gcd(b, a%b, y, x);
  y -= a/b*x;
  return g;
}
ll mod_inv(ll a, ll m){
  ll x, y;
  ll g = ext_gcd(a, m, x, y);
  if(g != 1)  return -1;  //no solution exists
  return (x%m+m)%m;
}

## Linear-sieve
int lpf[N], pm[N], pcnt = 0;
for (int i = 2; i < N; ++i) {
  if (!lpf[i])  lpf[i] = i, pm[pcnt++] = i;
  for (int j = 0; j < sz; ++j) {
    int p = pm[j];
    if (lpf[i] < p or i * p >= N)  break;
    lpf[i * p] = p;
  }
}

## Miller-Rabin
bool isp(ll n){
  if(n==2 || n == 3)  return 1;
  if(n<=1 || n%2==0)  return 0;
  for (int k = 0; k < 10; ++k){
    ll a = 2+rand()%(n-2);
    ll s = n-1;
    while(!(s&1)) s>>=1;
    if(powmod(a, s, n) == 1)  continue;
    int iscomp = 1;
    while(s!=n-1){
      if(powmod(a, s, n)==n-1){
        iscomp = 0;
        break;
      }
      s=s<<1;
    }
    if(iscomp) return 0;
  }
  return 1;
}

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

bool isp(u64 n) {
  if (n < 2)
    return false;
  int r = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    r++;
  }

  for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
 ↪  31, 37}) {
    if (n == a)
      return true;
    if (check_composite(n, a, d, r))
      return false;
  }
  return true;
}

## Prime Factorize of large number(Pollard Rho):
ll f(ll x, ll c, ll n){
  return (mulmod(x,x,n)+c)%n;
}
ll pollard_rho(ll n){
  if(n == 1)  return 1;
  if(n%2 == 0)  return 2;
```

```
  ll x = rand()%(n-2)+2;
  ll y = x;
  ll c = rand()%(n-1)+1;
  ll g = 1;
  while (g == 1){
    x = f(x, c, n);
    y = f(y, c, n);
    y = f(y, c, n);
    g = __gcd(abs(x-y), n);
  }
  return g;
}
vector<ll> prime_factorize(ll n){
  if(n<=1)  return vector<ll>();
  if(isp(n))  return vector<ll> ({n});
  ll d = pollard_rho(n);
  vector<ll> v = factorize(d);
  vector<ll> w = factorize(n/d);
  v.insert(v.end(), w.begin(), w.end());
  sort(v.begin(), v.end());
  return v;
}
// auto pf = prime_factorize(n);

## Number of divisors of n O(n^1/3):
int nod(ll n){
  sieve();
  int ret = 1;
  for (int i = 2; 1LL*i*i*i <= n; ++i){
    if(isp[i]){
      int e = 0;
      while(n%i == 0){
        e++;
        n /= i;
      }
      ret *= e+1;
    }
  }
  ll sq = sqrt(1.0L*n);
  if(isprime(n))  ret *= 2;
  else if(n == sq*sq and isprime(sq))  ret *= 3;
  else if(n!=1) ret *= 4;
  return ret;
}

## Totient function of n:
ll get_phi(ll n) {
  ll phi = n;
  for (int i = 2; i * i <= n; ++i) {
    if (n % i == 0) {
      phi -= phi / i;
      while (n % i == 0) {
        n /= i;
      }
    }
  }
  if (n > 1)  phi -= phi / n;
  return phi;
}

## Totient function from 1 to N:
int phi[N];
iota(phi, phi + N, 0);
for(int i = 2; i < N; ++i){
  if(phi[i] == i){
    phi[i]--;
    for(int j = 2 * i; j < N; j += i){
```

```
      phi[j] -= phi[j] / i;
    }
  }
}

## Smallest inverse phi
ll inv_phi(ll phi, ll n, int pc) {
  if (phi == 1)  return n;
  if (pc == -1)  return INF;
  ll ret = inv_phi(phi, n, pc - 1);
  if (phi % (p[pc] - 1) == 0) {
    phi /= (p[pc] - 1);
    n = n / (p[pc] - 1) * p[pc];
    while (phi % p[pc] == 0) {
      phi /= p[pc];
    }
    ret = min(ret, inv_phi(phi, n, pc - 1));
  }
  return ret;
}

ll phi;  cin >> phi;
if (phi & 1) {
  cout << (phi == 1) << "\n";
}
else {
  for (int i = 1; i * i <= phi; ++i) {
    if (phi % i == 0) {
      if (isp(i + 1)) {
        p.push_back(i + 1);
      }
      if (i * i != phi and isp(phi / i + 1)){
        p.push_back(phi / i + 1);
      }
    }
  }
  sort(p.begin(), p.end());
  ll ans = inv_phi(phi, phi, p.size() - 1);
  cout << (ans == INF? 0: ans) << "\n";
}

## GCD sum function from 1 to N:
ll phi[N], g[N];
void pcgsm(){  //pre calculate gcd sum fucntion
  pcphi();
  for (int i = 1; i < N; ++i){
    for (int j = i; j < N; j+=i){
      g[j] += i*phi[j/i];
    }
  }
}

## All Pair gcd sum:
for (int i = 1; i < N; ++i) {
  for (int j = i; j < N; j += i) {
    gcd_sum[j] += 1ll * phi[i] * (j / i);
  }
  gcd_sum[i] -= i;
  pref_gcd_sum[i] = pref_gcd_sum[i - 1] +
↪  gcd_sum[i];
}

## LCM sum function of n:
ll lsm(ll n){
  ll ret=0;
  for(ll d=1; d*d<=n; d++){
    if(n%d==0){
      ret += d*phi(d);
      if(n/d!=d)  ret += n/d*phi(n/d);
    }
  }
  return (ret+1)*n/2;
}

## LCM sum function from 1 to N
ll phi[N], l[N];
void pclsm(){  //pre calculate lcm sum function
  pcphi();
  for (int i = 1; i < N; ++i){
    for (int j = i; j < N; j+=i){
      l[j] += i*phi[i];
    }
  }
  for (int i = 1; i < N; ++i){
    l[i] = (l[i]+1)*i/2;
  }
}

## All pair lcm sum:
for (int i = 1; i < N; ++i) {
  for (int j = i; j < N; j += i) {
    lcm_sum[j] += i * phi[i];
  }
  lcm_sum[i]++;
  lcm_sum[i] /= 2;
  lcm_sum[i] *= i;
  lcm_sum[i] -= i;
  pref_lcm_sum[i] = lcm_sum[i];
  pref_lcm_sum[i] += pref_lcm_sum[i - 1];
}

## Number of co-prime pairs of an array:
vector<ll> cnt(A);
for (int xi: x) {
  for (int d = 1; d * d <= xi; ++d) {
    if (xi % d == 0) {
      cnt[d]++;
      if (xi / d != d) {
        cnt[xi / d]++;
      }
    }
  }
}

ll ans = 0;
for (int i = 1; i < A; ++i) {
  if (!sq_free[i])  continue;

  ll ways = cnt[i] * (cnt[i] - 1) / 2;
  if (pf[i].size() & 1 ^ 1) {
    ans += ways;
  }
  else {
    ans -= ways;
  }
}

## All pair gcd sum of an array:
vector<ll> cnt(A);
for (auto ai: a) {
  for (int d = 1; d * d <= ai; ++d) {
    if (ai % d == 0) {
      cnt[d]++;
      if (ai / d != d) {
        cnt[ai / d]++;
      }
    }
  }
}

ll sum = 0;
vector<ll> left(A);
iota(left.begin(), left.end(), 0);
for (int i = 1; i < A; ++i) {
  ll add = left[i] * cnt[i] * (cnt[i] - 1) / 2;
  sum += add;
  for (int j = 2 * i; j < A; j += i) {
    left[j] -= left[i];
  }
}


## CRT
ll crt(ll r1, ll m1, ll r2, ll m2){
  if(m1<m2) swap(r1, r2), swap(m1, m2);
  ll p, q, g = bezout(m1, m2, p, q);
  if((r2-r1)%g !=0 )  return -1;  //no solution
  ll x = (r2-r1)%m2*p%m2*m1/g + r1;
  return x<0? x+m1*m2/g: x;
}

ll crt(vector<ll>& r, vector<ll>& m){
  ll x = r[0], M=m[0];
  for (int i = 1; i < r.size(); ++i){
    x = crt(x, M, r[i], m[i]);
    ll g = __gcd(M, m[i]);
    M = (M/g)*(m[i]/g);
  }
  return x;
}

## Discrete Logarithm
ll discrete_log(ll a, ll b, ll m) {
  a %= m, b %= m;
  if(a == 0){
    return (b == 0? 1: -1);
  }

  ll k = 1, add = 0, g;
  while ((g = __gcd(a, m)) > 1) {
    if (b == k) {
      return add;
    }
    if (b % g) {
      return -1;
    }
    b /= g, m /= g, k = (k * a / g) % m, ++add;
  }

  int n = sqrt(m) + 1;

  unordered_map<int, int> vals;
  for (ll q = 0, cur = b; q <= n; ++q) {
    vals[cur] = q;
    cur = (cur * a) % m;
  }

  ll an = 1;
  for (int i = 0; i < n; ++i) {
    an = (an * a) % m;
  }

  for (ll p = 1, cur = k; p <= n; ++p) {
    cur = (cur * an) % m;
    if (vals.count(cur)) {
      return n * p - vals[cur] + add;
    }
  }

  return -1;
}

## Mobius
- mu[1] = 1,
- mu[n] = 0 if n has a squared prime factor,
- mu[n] = 1 if n is square-free with even number of
↪   prime factors
- mu[n] = -1 if n is square-free with odd number of
↪   prime factors

"sum of mu[d] where d | n is 0 ( For n=1, sum is 1
↪  )"

int mu[MAX] = {0};

void Mobius(int N){
  int i, j;
  mu[1] = 1;
  for (i = 1; i <= N; i++){
    if (mu[i]){
      for (j = i + i; j <= N; j += i){
        mu[j] -= mu[i];
      }
    }
  }
}
```

51 ONLINE_CHT

```
const ll IS_QUERY = -(1LL << 62);

struct line {
  ll m, b;
  mutable function <const line*()> succ;

  bool operator < (const line &rhs) const {
    if (rhs.b != IS_QUERY) return m < rhs.m;
    const line *s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s -> b < (s -> m - m) * x;
  }
};

struct CHT : public multiset <line> {
  bool bad (iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y -> m == z -> m && y -> b <= z -> b;
    }
    auto x = prev(y);
    if (z == end()) return y -> m == x -> m && y ->
↪   b <= x -> b;
    return 1.0 * (x -> b - y -> b) * (z -> m - y ->
↪   m) >= 1.0 * (y -> b - z -> b) * (y -> m - x ->
↪   m);
  }
  void add (ll m, ll b) {
    auto y = insert({m, b});
    y -> succ = [=] {return next(y) == end() ? 0 : &*next(y);};
    if (bad(y)) {erase(y); return;}
    while (next(y) != end() && bad(next(y))) erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }

  ll eval (ll x) {
    auto l = *lower_bound((line) {x, IS_QUERY});
    return l.m * x + l.b;
  }
};

// To find maximum
CHT cht;
cht.add(m, c);
y_max = cht.eval(x);

// To find minimum
CHT cht;
cht.add(-m, -c);
y_min = -cht.eval(x);
```

52 PALINDROMIC_TREE

```
const int N = 1e5+10;
struct vertex
{
  int len, link, no_of_suf_pal;
  map<char, int> next;
}pt[N];
int sz, at, cnt[N];
char s[N];

void pt_init(){
  for (int i = 0; i < N; ++i){
    pt[i].next.clear();
  }
  memset(cnt, 0, sizeof(cnt));
  pt[0].len = -1, pt[0].link = 0, pt[0].no_of_suf_pal = 0;
  pt[1].len = 0, pt[1].link = 0, pt[1].no_of_suf_pal = 0;
  sz = at = 1;
}

void pt_extend(int si){   //string index
  while (s[si - pt[at].len - 1] != s[si]) at = pt[at].link;
  int x = pt[at].link, c = s[si]-'a';
  while (s[si - pt[x].len - 1] != s[si])  x = pt[x].link;
  if(!pt[at].next.count(c)){
    pt[at].next[c] = ++sz;
    pt[sz].len = pt[at].len + 2;
    // cnt[pt[at].len+2]++;   //for  finding number of distinct palindrome of lenght k
    pt[sz].link = (pt[sz].len == 1)? 1 : pt[x].next[c];
    // pt[sz].no_of_suf_pal = 1 + pt[pt[sz].link].no_of_suf_pal;  //for finding number of palindrome which last position is si
  }
  // cnt[pt[at].len + 2]++; //for  finding number of palindrome of lenght k
  at = pt[at].next[c];
}

int num_of_pal(int ai){   //distinct palindrome, array index
  int ret = pt[at].ans;
  for(auto x : pt[ai].next)
    ret += num_of_pal(x.second);
  return ret;
}

int main(){
  scanf("%s", s);
  pt_init();
  for (int i = 0; s[i]; ++i){
    pt_extend(i);
  }
  int ans = num_of_pal(0) + num_of_pal(1) - 2;
  printf("%d\n", ans);
  return 0;
}
```

53 PERSISTENT_SEGMENT_TREE

```
## Point Addition & Range Sum:
struct node {
  ll sum;
  node *l, *r;
  node(ll s = 0, node *l = NULL, node *r = NULL): sum(s), l(l), r(r) {}
};
node* add(node *u, int i, int x, int s, int e) {
  if (s == e) return new node(u->sum + x);
  if (!u->l)  u->l = new node(), u->r = new node();
  node *nu = new node(u->sum, u->l, u->r);
  int m = (s + e) / 2;
  if (i <= m)  nu->l = add(nu->l, i, x, s, m);
  else nu->r = add(nu->r, i, x, m + 1, e);
  nu->sum = nu->l->sum + nu->r->sum;
  return nu;
}
ll rsum(node *u, int l, int r, int s, int e) {
  if (!u)  return 0;
  if (s > r or e < l)  return 0;
  if (l <= s and e <= r)  return u->sum;
  int m = (s + e) / 2;
  return rsum(u->l, l, r, s, m) + rsum(u->r, l, r, m + 1, e);
}
vector<node*> root(VER);
root[0] = new node();  // initialization
int ver = 1;
root[k] = add(root[k], i, x, 0, sz - 1);;  // Assign a[i] = x in version k
root[ver++] = root[k];  // Create a new version from kth version
cout << rsum(root[k], l, r, 0, sz - 1) << "\n";  // Range sum of version k
```

```
## count numbers > k in a range
root[0] = new node();
for (int i = 0; i < n; ++i) {
  root[i + 1] = add(root[i], a[i], 1);
}
while (q--) {
  int l, r, k;  cin >> l >> r >> k;  l--, r--;
  int ans = rsum(root[r + 1], k, E - 1) - rsum(root[l], k, E - 1);
  cout << ans << "\n";
}

## kth number in a range: O(logn)
int kth(node *ul, node *ur, int k, int s = 0, int e = E - 1) {
  if (s == e)  return s;
  int m = (s + e) / 2;
  int cnt_left = ur->left->sum - ul->left->sum;
  if (cnt_left >= k)  return kth(ul->left, ur->left, k, s, m);
  else  return kth(ul->right, ur->right, k - cnt_left, m + 1, e);
}

root[0] = new node();
for (int i = 0; i < n; ++i) {
  root[i + 1] = add(root[i], a[i + ], 1);
}
while (q--) {
  int l, r, k;  cin >> l >> r >> k;  l--, r--;
  int x = kth(root[l], root[r + 1], k);
}
```

54 PERSISTENT_TRIE

```
struct node {
  node *nxt[2];
  node() { fill(nxt, nxt + 2, nullptr); }
};
node* add(node *prev, int x) {
  node *new_root = new node();
  node * cur = new_root;;
  for (int idx = IDX - 1; idx >= 0; --idx) {
    int f = (x >> idx) & 1;
    if (prev and prev->nxt[!f])  cur->nxt[!f] = prev->nxt[!f];
    cur->nxt[f] = new node();
    cur = cur->nxt[f];
    if (prev)  prev = prev->nxt[f];
  }
  return new_root;
}
int get_max(node *root, int x) {
  if (!root)  return 0;
  node *u = root;
  int ret = 0;
  for (int idx = IDX - 1; idx >= 0; --idx) {
    int f = (x >> idx) & 1;
    if (u->nxt[!f])  ret += (1 << idx), u = u->nxt[!f];
    else  u = u->nxt[f];
  }
  return ret;
}
```

55 POINT

```
# point in convex poly: O(logn)
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 ); }
};
inline double dot(pt u, pt v) { return u.x*v.x + u.y*v.y; }
inline double cross(pt u, pt v) {return u.x*v.y - u.y*v.x;}
inline double triArea2(pt a,pt b,pt c) { return cross(b-a,c-a); }
inline bool inDisk(pt a, pt b, pt p) { return dot(a-p, b-p) <= 0; }
inline bool onSegment(pt a, pt b, pt p) { return triArea2(a,b,p) == 0 && inDisk(a,b,p); }
// points of the polygon has to be in ccw order
// if strict, returns false when a is on the boundary
inline bool insideConvexPoly(pt* C, int nc, pt  p, bool strict) {
    int st = 1, en = nc - 1, mid;
    while(en - st > 1) {
        mid = (st + en)>>1;
        if(triArea2(C[0], C[mid], p) < 0) en = mid;
        else st = mid;
    }
    if(strict) {
        if(st==1) if(onSegment(C[0],C[st],p)) return false;
        if(en==nc-1) if(onSegment(C[0],C[en],p)) return false;
        if(onSegment(C[st],C[en],p)) return false;
    }
    if(triArea2(C[0], C[st], p) < 0) return false;
    if(triArea2(C[st], C[en], p) < 0) return false;
    if(triArea2(C[en], C[0], p) < 0) return false;
    return true;
}

# check point in polygon: O(n)
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 ); }
};
inline double dot(pt u, pt v) { return u.x*v.x + u.y*v.y; }
inline double cross(pt u, pt v) {return u.x*v.y - u.y*v.x;}
inline double triArea2(pt a,pt b,pt c) { return cross(b-a,c-a); }
inline bool inDisk(pt a, pt b, pt p) { return dot(a-p, b-p) <= 0; }
inline bool onSegment(pt a, pt b, pt p) { return triArea2(a,b,p) == 0 && inDisk(a,b,p); }
// check if segment pq crosses ray from point a
inline bool crossesRay(pt a, pt p, pt q) {
    int mul = (q.y>=a.y) - (p.y>=a.y);
    return (mul * triArea2(a,p,q)) > 0;
}
// if strict, returns false when a is on the boundary
inline bool insidePoly(pt *P, int np, pt a, bool strict = true) {
    int numCrossings = 0;
    for (int i = 0; i < np; i++) {
        if (onSegment(P[i], P[(i+1)%np], a)) return !strict;
        numCrossings += crossesRay(a, P[i], P[(i+1)%np]);
    }
    return (numCrossings & 1); // inside if odd number of crossings
}

# Polar sort:
inline bool up (point p) {
  return p.y > 0 or (p.y == 0 and p.x >= 0);
}
sort(v.begin(), v.end(), [] (point a, point b) {
  return up(a) == up(b) ? a.x * b.y > a.y * b.x : up(a) < up(b);
});

# Convex Hull
struct pt {
  int x, y;
};
ll cross(pt a, pt b, pt c) {  //ab*ac
  return 1ll*(b.x-a.x)*(c.y-a.y) - 1ll*(c.x-a.x)*(b.y-a.y);
}
vector<pt> convexHull(vector<pt>& p) {
  sort(p.begin(), p.end(), [&] (pt a, pt b) {
    return (a.x==b.x? a.y<b.y: a.x<b.x);
  });
  int n = p.size(), m = 0;
  vector<pt> hull(2*n);
  for (int i = 0; i < n; ++i){
    while (m>=2 and cross(hull[m-2], hull[m-1], p[i]) < 0)  --m;
    hull[m++] = p[i];
  }
  for (int i = n-2, l = m; i >= 0; --i) {
    while(m>=l+1 and cross(hull[m-2], hull[m-1], p[i]) < 0)  --m;
    hull[m++] = p[i];
  }
  hull.resize(m-1);
  return hull;
}
```

56 POLAR_SORT

```
inline bool up (point p) {
  return p.y > 0 or (p.y == 0 and p.x >= 0);
}
sort(v.begin(), v.end(), [] (point a, point b) {
  return up(a) == up(b) ? a.x * b.y > a.y * b.x : up(a) < up(b);
});
```

57 POLYNOMIAL_INTERPOLATION

```
// P(x) = a0 + a1x + a2x^2 + ... + anx^n
// y[i] = P(i)
ll eval (vector<ll> y, ll k) {
  int n = y.size() - 1;
  if (k <= n) {
    return y[k];
  }
  vector<ll> L(n + 1, 1);
  for (int x = 1; x <= n; ++x) {
    L[0] = L[0] * (k - x) % mod;
    L[0] = L[0] * inv(-x) % mod;
  }
  for (int x = 1; x <= n; ++x) {
    L[x] = L[x - 1] * inv(k - x) % mod * (k - (x - 1)) % mod;
    L[x] = L[x] * ((x - 1) - n + mod) % mod * inv(x) % mod;
  }
  ll yk = 0;
  for (int x = 0; x <= n; ++x) {
    yk = add(yk, L[x] * y[x] % mod);
  }
  return yk;
}
```

58 SCC

```
void dfs1(int u, vector<int> *adj, vector<int> &vis, vector<int> &order) {
  vis[u] = 1;
  for (int &v: adj[u]) {
    if (!vis[v]) {
      dfs1(v, adj, vis, order);
    }
  }
  order.emplace_back(u);
}
void dfs2(int u, vector<int> *rev_adj, vector<int> &vis, vector<int> &scc) {
  scc.emplace_back(u);
  vis[u] = 1;
  for (int &v: rev_adj[u]) {
    if (!vis[v]) {
      dfs2(v, rev_adj, vis, scc);
    }
  }
}
vector<vector<int>> get_sccs(int n, vector<int> *adj) {
  vector<int> vis(n), order;
  for (int u = 0; u < n; ++u) {
    if (!vis[u]) {
      dfs1(u, adj, vis, order);
    }
  }
  vector<int> rev_adj[n];
  for (int u = 0; u < n; ++u) {
    for (int v: adj[u]) {
      rev_adj[v].emplace_back(u);
    }
  }
  vector<vector<int>> sccs;
  reverse(order.begin(), order.end());
  vis.assign(n, 0);
  for (int u: order) {
    if (!vis[u]) {
      sccs.emplace_back(0);
      dfs2(u, rev_adj, vis, sccs.back());
    }
  }
  return sccs;
}
vector<vector<int>> sccs = get_sccs(n, adj);
int tot_scc = sccs.size();
vector<int> scc_no(n);
for (int i = 0; i < tot_scc; ++i) {
  for (int u: sccs[i]) {
    scc_no[u] = i;
  }
}
```

59 SEGMENT_TREE

```
## Point Assign & Range Query
## Point Assign & Range Min/Max with frequency
## Point Assign & Range Maximum Sum
struct Node {
  ll sum, mx, pref, suf;
} t[4 * N];
Node combine(Node l, Node r) {
  Node ret;
  ret.pref = max(l.pref, l.sum + r.pref);
  ret.suf = max(l.suf + r.sum, r.suf);
  ret.sum = l.sum + r.sum;
  ret.mx = max({l.mx, l.suf + r.pref, r.mx});
  return ret;
}
void update(int idx, int x, int v = 1, int st = 0,
↪  int ed = n - 1) {
  if (st > idx or ed < idx)  return ;
  if (st == ed) {
    t[v].sum = t[v].mx = t[v].pref = t[v].suf = x;
    return ;
  }
  int lc = 2 * v, rc = 2 * v + 1;
  int mid = (st + ed) / 2;
  update(idx, x, lc, st, mid);
  update(idx, x, rc, mid + 1, ed);
  t[v] = combine(t[lc], t[rc]);
}
Node mx_sum(int l, int r, int v = 1, int st = 0,
↪  int ed = n - 1) {
  if (st > r or ed < l)  return {0, ll(-1e15),
↪  ll(-1e15), ll(-1e15)};
  if (l <= st and ed <= r) return t[v];
  int lc = 2 * v, rc = 2 * v + 1;
  int mid = (st + ed) >> 1;
  Node lret = mx_sum(l, r, lc, st, mid);
  Node rret = mx_sum(l, r, rc, mid + 1, ed);
  Node ret = combine(lret, rret);
  return ret;
}
## Point Assign & Range Composite:
- 0 p c d: f_p : = cx + d
                   - 1 l r x: f_r(f_(r -
↪  1)(...f_l(x))) % mod
struct Node {
  int a, b;
} t[3 * N];
void assign(int i, int ai, int bi, int u = 1, int s
↪  = 0, int e = n - 1) {
  if (s > i or e < i)  return ;
  if (s == e) {
    t[u].a = ai;
    t[u].b = bi;
    return ;
  }
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  assign(i, ai, bi, v, s, m);
  assign(i, ai, bi, w, m + 1, e);
  t[u].a = 1ll * t[w].a * t[v].a % mod;
  t[u].b = (1ll * t[w].a * t[v].b + t[w].b) % mod;
}
Node query(int l, int r, int u = 1, int s = 0, int
↪  e = n - 1) {
  if (l > e or r < s)  return {1, 0};
  if (l <= s and e <= r) return t[u];
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  Node lret = query(l, r, v, s, m);
  Node rret = query(l, r, w, m + 1, e);
  Node ret;
  ret.a = 1ll * rret.a * lret.a % mod;
  ret.b = (1ll * rret.a * lret.b + rret.b) % mod;
  return ret;
}
## Range Affine & Range Sum
- 0 l r b c: a[i] : = b * a[i] + c
                      - 1 l r: range sum
struct Node {
  int a = 1, b = 0, sum = 0;
} t[3 * N];
void push(int u, int s, int e) {
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  t[v].sum = (1ll * t[u].a * t[v].sum + 1ll * (m -
↪  s + 1) * t[u].b) % mod;
  t[v].a = 1ll * t[u].a * t[v].a % mod;
  t[v].b = (1ll * t[u].a * t[v].b + t[u].b) % mod;
  t[w].sum = (1ll * t[u].a * t[w].sum + 1ll * (e -
↪  m) * t[u].b) % mod;
  t[w].a = 1ll * t[u].a * t[w].a % mod;
  t[w].b = (1ll * t[u].a * t[w].b + t[u].b) % mod;
  t[u].a = 1;
  t[u].b = 0;
}
void update(int l, int r, int a, int b, int u = 1,
↪  int s = 0, int e = n - 1) {
  if (l > e or r < s)  return ;
  if (l <= s and e <= r) {
    t[u].sum = (1ll * a * t[u].sum + 1ll * (e - s +
↪  1) * b) % mod;
    t[u].a = 1ll * a * t[u].a % mod;
    t[u].b = (1ll * a * t[u].b + b) % mod;
    return ;
  }
  if (t[u].b != 0)  push(u, s, e);
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  update(l, r, a, b, v, s, m);
  update(l, r, a, b, w, m + 1, e);
  t[u].sum = (t[v].sum + t[w].sum) % mod;
}
int rsum(int l, int r, int u = 1, int s = 0, int e
↪  = n - 1) {
  if (l > e or r < s)  return 0;
  if (l <= s and e <= r) return t[u].sum;
  if (t[u].b != -1)  push(u, s, e);
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  int lret = rsum(l, r, v, s, m);
  int rret = rsum(l, r, w, m + 1, e);
  return (lret + rret) % mod;
}
## Range Most Frequent for sorted array:
struct node {
  int pref, pref_freq, suf, suf_freq, max_freq;
} t[3 * MXN];
node combine(node lc, node rc) {
  node res;
  res.pref = lc.pref;
  res.pref_freq = lc.pref_freq;
  if (lc.pref == rc.pref) {
    res.pref_freq += rc.pref_freq;
  }
  res.suf = rc.suf;
  res.suf_freq = rc.suf_freq;
  if (lc.suf == rc.suf) {
    res.suf_freq += lc.suf_freq;
  }
  res.max_freq = max({lc.max_freq, lc.suf ==
↪   rc.pref ? lc.suf_freq + rc.pref_freq : 0, +
↪   rc.max_freq});
  return res;
}
void build(int v = 1, int st = 0, int ed = n - 1) {
  if (st > ed) return ;
  if (st == ed) {
    t[v] = {a[st], 1, a[st] , 1, 1};
    return ;
  }
  int lc = v << 1;
  int rc = lc ^ 1;
  int m = (st + ed) >> 1;
  build(lc, st, m);
  build(rc, m + 1, ed);
  t[v] = combine(t[lc], t[rc]);
}
node query(int l, int r, int v = 1, int st = 0, int
↪  ed = n - 1) {
  if (l > ed or r < st)  return {INT_MIN, 0,
↪  INT_MAX, 0, 0};
  if (l <= st and ed <= r) return t[v];
  int lc = v << 1;
  int rc = lc ^ 1;
  int m = (st + ed) >> 1;
  node lans = query(l, r, lc, st, m);
  node rans = query(l, r, rc, m + 1, ed);
  node res = combine(lans, rans);
  return res;
}
## Point Update & kth non zero
## Point Update & first_above
## Point Update & Range Max Prefix Sum
## Remove Point & Point Query
## Range Inversion Count
struct Node {
  int cnt[40];
  ll inv;
  Node(): inv(0) {memupdate(cnt, 0, sizeof(cnt));}
} t[4 * N];
void update(int i, int x, int y, int v = 1, int s =
↪  0, int e = n - 1) {
  if (s > i or e < i)  return ;
  if (s == e) {
    if (x != -1) t[v].cnt[x]--;
    t[v].cnt[y]++;
    t[v].inv = 0;
    return ;
  }
  int lc = 2 * v, rc = 2 * v + 1;
  int m = (s + e) / 2;
  update(i, x, y, lc, s, m);
  update(i, x, y, rc, m + 1, e);
  t[v].inv = t[lc].inv + t[rc].inv;
  int now = 0;
  for (int j = 0; j < 40; ++j) {
    t[v].inv += 1LL * t[lc].cnt[j] * now;
    now += t[rc].cnt[j];
    t[v].cnt[j] = t[lc].cnt[j] + t[rc].cnt[j];
  }
}
Node count_inv(int l, int r, int v = 1, int s = 0,
↪  int e = n - 1) {
  if (s > r or e < l)  return Node();
  if (l <= s and e <= r) {
    return t[v];
  }
  int lc = 2 * v, rc = 2 * v + 1;
  int m = (s + e) / 2;
  Node lret = count_inv(l, r, lc, s, m);
  Node rret = count_inv(l, r, rc, m + 1, e);
  Node ret;
  ret.inv = lret.inv + rret.inv;
  int now = 0;
  for (int j = 0; j < 40; ++j) {
    ret.inv += 1LL * lret.cnt[j] * now;
    now += rret.cnt[j];
    ret.cnt[j] = lret.cnt[j] + rret.cnt[j];
  }
  return ret;
}
## Point Update & Range Distinct Elements:
void add(int idx, int i, int x, int v = 1, int s =
↪  0, int e = n - 1) {
  if (s > i or e < i)  return ;
  if (s == e) {
    t[idx][v] += x;
    return ;
  }
  int lc = 2 * v, rc = 2 * v + 1;
  int m = (s + e) / 2;
  add(idx, i, x, lc, s, m);
  add(idx, i, x, rc, m + 1, e);
  t[idx][v] = t[idx][lc] + t[idx][rc];
}
int rsum(int idx, int l, int r, int v = 1, int s =
↪  0, int e = n - 1) {
  if (s > r or e < l)  return 0;
  if (l <= s and e <= r) return t[idx][v];
  int lc = 2 * v, rc = 2 * v + 1;
  int m = (s + e) / 2;
  int lret = rsum(idx, l, r, lc, s, m);
  int rret = rsum(idx, l, r, rc, m + 1, e);
  return lret + rret;
}
## Range Addition & Range Weighted Sum:
struct Node {
  ll sum = 0, wsum = 0, p = 0;
} t[3 * N];
void f(int x, int u, int s, int e) {
  t[u].sum += 1LL * (e - s + 1) * x;
  t[u].wsum += 1LL * (e - s + 1) * (e - s + 2) / 2
↪  * x;
  t[u].p += x;
}
void push(int u, int s, int e) {
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  f(t[u].p, v, s, m);
  f(t[u].p, w, m + 1, e);
  t[u].p = 0;
}
void add(int l, int r, int x, int u = 1, int s = 0,
↪  int e = n - 1) {
  if (l > e or r < s)  return;
  if (l <= s and e <= r) {
    f(x, u, s, e);
    return;
  }
  push(u, s, e);
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  add(l, r, x, v, s, m);
  add(l, r, x, w, m + 1, e);
  t[u].sum = t[v].sum + t[w].sum;
  t[u].wsum = t[v].wsum + (t[w].wsum + (m - s + 1)
↪  * t[w].sum);
}
// 1*a[l] + 2*a[l + 1] + 3*a[l+3] + ...
Node rwsum(int l, int r, int u = 1, int s = 0, int
↪  e = n - 1) {
  if (l > e or r < s)  return {0, 0, 0};
  if (l <= s and e <= r) return t[u];
  push(u, s, e);
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  Node ln = rwsum(l, r, v, s, m);
  Node rn = rwsum(l, r, w, m + 1, e);
  return {ln.sum + rn.sum, ln.wsum + (rn.wsum +
↪  max(0, (m - max(l, s) + 1))*rn.sum), 0};
}
## Add Arithmetic Progression on Range & Range Sum
↪  Query:
struct Node {
  ll a = 0, d = 0, sum = 0;
} t[3 * N];
ll sum(ll a, ll d, ll n) {
  return n * (2 * a + (n - 1) * d) / 2;
}
void push(int v, int st, int ed) {
  int lc = 2 * v, rc = 2 * v + 1, md = (st + ed) /
↪  2;
  t[lc].a += t[v].a;
  t[lc].d += t[v].d;
  t[lc].sum += sum(t[v].a, t[v].d, md - st + 1);
  t[rc].a += t[v].a + 1LL * (md - st + 1) * t[v].d;
  t[rc].d += t[v].d;
  t[rc].sum += sum(t[v].a + 1LL * (md - st + 1) *
↪  t[v].d, t[v].d, ed - md);
  t[v].a = t[v].d = 0;
}
void add(int l, int r, int a, int d, int v = 1, int
↪  st = 0, int ed = n - 1) {
  if (l > ed or r < st)  return;
  if (l <= st and ed <= r) {
    t[v].a += a + 1LL * (st - l) * d;
    t[v].d += d;
    t[v].sum += sum(a + 1LL * (st - l) * d, d, (ed
↪  - st + 1));
    return;
  }
  int lc = 2 * v, rc = 2 * v + 1, md = (st + ed) /
↪  2;
  if (t[v].d)  push(v, st, ed);
  add(l, r, a, d, lc, st, md);
  add(l, r, a, d, rc, md + 1, ed);
  t[v].sum = t[lc].sum + t[rc].sum;
}
ll rsum(int l, int r, int v = 1, int st = 0, int ed
↪  = n - 1) {
  if (l > ed or r < st)  return 0;
  if (l <= st and ed <= r) return t[v].sum;
  int lc = 2 * v, rc = 2 * v + 1, md = (st + ed) /
↪  2;
  if (t[v].d)  push(v, st, ed);
  ll lret = rsum(l, r, lc, st, md);
  ll rret = rsum(l, r, rc, md + 1, ed);
  return lret + rret;
}
## Range Update & Number of Segment with only set
↪  value & the total number of set value:
int t[3 * N], t2[3 * N], p[3 * N];
bool lb[3 * N], rb[3 * N]; //lb for left most bit
↪  of interval corresponding to this node
void f(int u, int x, int l, int r) {
  t[u] = (r - l + 1) * x;
  t2[u] = x;
  p[u] = x;
  lb[u] = rb[u] = x;
}
void push(int u, int s, int e) {
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  f(v, p[u], s, m);
  f(w, p[u], m + 1, e);
  p[u] = -1;
}
void assign(int l, int r, bool x, int u = 1, int s
↪  = 0, int e = N - 1) {
  if (l > e or r < s)  return ;
  if (l <= s and e <= r) {
    f(u, x, s, e);
    return ;
  }
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  if (p[u] != -1)  push(u, s, e);
  assign(l, r, x, v, s, m);
  assign(l, r, x, w, m + 1, e);
  t[u] = t[v] + t[w];
  t2[u] = t2[v] + t2[w];
  lb[u] = lb[v];
  rb[u] = rb[w];
  if (rb[v] == 1 and lb[w] == 1)  t2[u]--;
}
Update: assign(l, r, 0 / 1);
Query: t[1]: = total number of set bit, t2[1]: = total  number of segment with only set value

## Make All Elements <= k and Make all elements >= k on range & Point Query:
               const int I = 1e9 + 9;
int t[3 * N], pa[3 * N], pr[3 * N], ar[3 * N]; //pa for propagate adding, pr for propagate remove, ar for check last on is adding(1) or remove(0)
void fg(int x, int u) {   //function for make_greater
  t[u] = max(t[u], x);
  pa[u] = max(pa[u], x);
  pr[u] = max(pr[u], x);
  ar[u] = 1;
}
void fl(int x, int u) {   //function for make_less
  t[u] = min(t[u], x);
  pr[u] = min(pr[u], x);
  pa[u] = min(pa[u], x);
  ar[u] = 0;
}
void push(int u) {
  int v = 2 * u, w = 2 * u + 1;
  if (ar[u] == 0) {
    if (pa[u] != -1) {
      fg(pa[u], v); fg(pa[u], w);
    }
    if (pr[u] != I) {
      fl(pr[u], v); fl(pr[u], w);
    }
  } else {
    if (pr[u] != I) {
      fl(pr[u], v); fl(pr[u], w);
    }
    if (pa[u] != -1) {
      fg(pa[u], v); fg(pa[u], w);
    }
  }
  pa[u] = -1; pr[u] = I;
}
void make_greater(int l, int r, int x, int u = 1, int s = 0, int e = N - 1) {
  if (l > e or r < s)  return;
  if (l <= s and e <= r) {
    fg(x, u);
    return ;
  }
  push(u);
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  make_greater(l, r, x, v, s, m);
  make_greater(l, r, x, w, m + 1, e);
}
void make_less(int l, int r, int x, int u = 1, int s = 0, int e = N - 1) {
  if (l > e or r < s)  return;
  if (l <= s and e <= r) {
    fl(x, u);
    return;
  }
  push(u);
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  make_less(l, r, x, v, s, m);
  make_less(l, r, x, w, m + 1, e);
}
int at(int i, int u = 1, int s = 0, int e = N - 1) {
  if (s == e)  return t[u];
  push(u);
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  if (i <= m)  return at(i, v, s, m);
  else  return at(i, w, m + 1, e);
}
## Range Addition and Range Assign and Range sum
int n;
ll t[3 * N], p[3 * N], p2[3 * N]; //t for sum, p for assign & p2. for add
void pull(int v) {
  t[v] = t[2 * v] + t[2 * v + 1];
}
void push(int v, int st, int ed) {
  int lc = 2 * v, rc = 2 * v + 1, md = (st + ed) / 2;
  if (p[v] != -1) {
    t[lc] = p[v] * (md - st + 1);
    t[rc] = p[v] * (ed - md);
    p[lc] = p[rc] = p[v];
    p2[lc] = p2[rc] = 0;
    p[v] = -1;
  }
  if (p2[v]) {
    t[lc] += p2[v] * (md - st + 1);
    t[rc] += p2[v] * (ed - md);
    p2[lc] += p2[v];
    p2[rc] += p2[v];
    p2[v] = 0;
  }
}
void assign(int l, int r, int x, int v = 1, int st = 0, int ed = n - 1) {
  if (l > ed or r < st)  return;
  if (l <= st and ed <= r) {
    t[v] = 1LL * (ed - st + 1) * x;
    p[v] = x;
    p2[v] = 0;
    return;
  }
  int lc = 2 * v, rc = 2 * v + 1, md = (st + ed) / 2;
  push(v, st, ed);
  assign(l, r, x, lc, st, md);
  assign(l, r, x, rc, md + 1, ed);
  pull(v);
}
void add(int l, int r, int x, int v = 1, int st = 0, int ed = n - 1) {
  if (l > ed or r < st)  return;
  if (l <= st and ed <= r) {
    t[v] += 1LL * (ed - st + 1) * x;
    p2[v] += x;
    return ;
  }
  push(v, st, ed);
  int lc = 2 * v, rc = 2 * v + 1, md = (st + ed) / 2;
  add(l, r, x, lc, st, md);
  add(l, r, x, rc, md + 1, ed);
  pull(v);
}
ll rsum(int l, int r, int v = 1, int st = 0, int ed = n - 1) {
  if (l > ed or r < st)  return 0;
  if (l <= st and ed <= r) return t[v];
  push(v, st, ed);
  int lc = 2 * v, rc = 2 * v + 1, md = (st + ed) / 2;
  ll lret = rsum(l, r, lc, st, md);
  ll rret = rsum(l, r, rc, md + 1, ed);
  return lret + rret;
}
```

60 SHORTEST_PATH

```
## Dijkstra
priority_queue<array<ll, 2>> pq;
vector<ll> dis(n, INF), vis(n);
while (!pq.empty()) {
  auto [d, u] = pq.top();  pq.pop();
  if (vis[u])  continue;
  vis[u] = 1;
  for (auto [v, c]: next[u]) {
    if (dis[v] > d + c) {
      dis[v] = d + c;
      pq.push({dis[v], v});
    }
  }
}
## Bellman-ford
vector<int> bellman_ford(int s){
  vector<int> dis(n, I);
  dis[s]=0;
  while(1){
    int any=0;
    for (auto& e: ed){
      if(dis[e.u]<I){
        if(dis[e.u]+e.cost < dis[e.v]){
          dis[e.v] = dis[e.u]+e.cost;
          any=1;
        }
      }
    }
    if(!any)  break;
  }
  return dis;
}
## Floy-Warshall
for (int k = 0; k < n; ++k) {
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
    }
  }
}
```

61 SOS_DP

```
## Count over subset
for (int i = 0; i < n; ++i)  f[a[i]] = ?;
for (int i = 0; i < i; ++i) {
  for (int mask = 0; mask < (1 << n); ++mask) {
    if (mask&(1<<i)) {
      f[mask] += f[mask^(1<<i)];
    }
  }
}
## Count over superset
for (int i = 0; i < n; ++i)  f[a[i]] = ?;
for (int i = 0; i < n; ++i) {
  for (int mask = (1 << n) - 1; mask >= 0 ; --mask) {
    if (!(mask&(1<<i))) {
      f[mask] += f[mask^(1<<i)];
    }
  }
}
## How many pairs in ara[] such that (ara[i] & ara[j]) = 0
/// N --> Max number of bits of any array element
const int N = 20;
int inv = (1<<N) - 1;
int F[(1<<N) + 10];
int ara[MAX];

/// ara is 0 based
long long howManyZeroPairs(int n,int ara[]) {
    CLR(F);
    for(int i=0;i<n;i++) F[ara[i]]++;
    for(int i = 0;i < N; ++i)
        for(int mask = 0; mask < (1<<N); ++mask){
            if(mask & (1<<i))
                F[mask] += F[mask^(1<<i)];
        }

    long long ans = 0;
    for(int i=0;i<n;i++) ans += F[ara[i] ^ inv];
    return ans;
}

/// To get
    for(int mask = 0;mask < (1<<N); ++mask)
        for(int i = 0;i < (1<<N); ++i)
            if( (mask & i) == mask ) { /// i is a supermask of mask
                F[mask] += A[i];
            }
/// The code is the following
    for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
    for(int i = 0;i < N; ++i)
        for(int mask = (1<<N)-1; mask >= 0; --mask){
            if (!(mask & (1<<i)))
                F[mask] += F[mask | (1<<i)];
        }
## Number of subsequences of ara[0:n-1] such that
## sub[0] & sub[2] & ... & sub[k-1] = 0
const int N = 20;
int inv = (1<<N) - 1;
int F[(1<<N) + 10];
int ara[MAX];
int p2[MAX]; /// p2[i] = 2^i
///0 based array
int howManyZeroSubSequences(int n,int ara[]) {
    CLR(F);
    for(int i=0;i<n;i++) F[ara[i]]++;
    for(int i = 0;i < N; ++i)
        for(int mask = (1<<N)-1; mask >= 0; --mask){
            if (!(mask & (1<<i)))
                F[mask] += F[mask | (1<<i)];
        }
    int ans = 0;
    for(int mask=0;mask<(1<<N);mask++) {
        if(__builtin_popcount(mask) & 1) ans = sub(ans, p2[F[mask]]);
        else ans = add(ans, p2[F[mask]]);
    }
    return ans;
}
## Number of subsequences of ara[0:n-1] such that
## sub[0] | sub[2] | ... | sub[k-1] = Q
int F[(1<<20) + 10], ara[MAX];
int p2[MAX]; /// p2[i] = 2^i
/// ara is 0 based
int howManySubsequences(int n, int ara[], int m, int Q) {
    CLR(F);
    for(int i=0;i<n;i++) F[ara[i]]++;
    if(Q == 0) return sub(p2[F[0]], 1);
    for(int i = 0;i < m; ++i)
        for(int mask = 0; mask < (1<<m); ++mask){
            if (mask & (1<<i))
                F[mask] += F[mask ^ (1<<i)];
        }
    int ans = 0;
    for(int mask=0;mask<(1<<m);mask++) {
        if(mask & Q != mask) continue;
        if(__builtin_popcount(mask ^ Q) & 1) ans = sub(ans, p2[F[mask]]);
        else ans = add(ans, p2[F[mask]]);
    }
    return ans;
}
```

62 SPARSE_TABLE

```
int n, a[N], lg[N], st[N][K];
for (int i = 2; i < N; ++i) {
  lg[i] = lg[i / 2] + 1;
}
void build() {
  for (int k = 0; k < K ; ++k) {
    for (int i = 0; i + (1 << k) <= n; ++i) {
      if (k == 0)  st[i][k] = a[i];
      else  st[i][k] = min(st[i][k - 1], st[i + (1 << (k - 1))][k - 1]);
    }
  }
}
int rmq (int l, int r) {
  int k = lg[r - l + 1];
  return min(st[l][k], st[r - (1 << k) + 1][k]);
}
```

63 SQRT_DECOMPOSITION

```
const int SZ2 = 2e5, SZ = sqrt(SZ2+.0)+1, N = SZ*SZ;
int n, a[N], b[SZ];
void build() {
  for (int i = 0; i < SZ; ++i){
    b[i] = INT_MAX;
  }
  for (int i = 0; i < n; ++i){
    b[i/SZ] = min(b[i/SZ], a[i]);
  }
}
int rmq(int l, int r) {
  int lb = l/SZ, rb = r/SZ;
  int ret = INT_MAX;
  if(lb==rb) {
    for (int i = l; i <= r; ++i){
      ret = min(ret, a[i]);
    }
  } else {
    for (int i = l; i < (lb+1)*SZ; ++i){
      ret = min(ret, a[i]);
    }
    for (int i = lb+1; i < rb; ++i){
      ret = min(ret, b[i]);
    }
    for (int i = rb*SZ; i <= r; ++i){
      ret = min(ret, a[i]);
    }
  }
  return ret;
}
```

64 STRESS_TESTING

```
set -e
g++ -O2 -static -std=gnu++17 gen.cpp -o gen
g++ -O2 -static -std=gnu++17 main.cpp -o main
g++ -O2 -static -std=gnu++17 brute.cpp -o brute
for((i = 1; ; ++i)); do
    echo $i
    ./gen $i > in
    # ./main < in > out
    # ./brute < in > out2
    # diff -w out out2 || break
    diff -w <(./main < in) <(./brute < in) || break
done
```

65 SUFFIX_ARRAY

```
array<vector<int>, 2> get_sa(string& s, int lim=128) {  // for integer, just change string to vector<int> and minimum value of vector must be >= 1
  int n = s.size() + 1, k = 0, a, b;
  vector<int> x(begin(s), end(s)+1), y(n), sa(n), lcp(n), ws(max(n, lim)), rank(n);
  x.back() = 0;
  iota(begin(sa), end(sa), 0);
  for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
    p = j, iota(begin(y), end(y), n - j);
    for (int i = 0; i < n; ++i) if (sa[i] >= j) y[p++] = sa[i] - j;
    fill(begin(ws), end(ws), 0);
    for (int i = 0; i < n; ++i) ws[x[i]]++;
    for (int i = 1; i < lim; ++i) ws[i] += ws[i -
↪   1];
    for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
    swap(x, y), p = 1, x[sa[0]] = 0;
    for (int i = 1; i < n; ++i) a = sa[i - 1], b =
↪   sa[i], x[b] =
      (y[a] == y[b] && y[a + j] == y[b + j]) ? p -
↪   1 : p++;
  }
  for (int i = 1; i < n; ++i) rank[sa[i]] = i;
  for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
    for (k && k--, j = sa[rank[i] - 1]; s[i + k] ==
↪   s[j + k]; k++);
  sa.erase(sa.begin()), lcp.erase(lcp.begin());
  return {sa, lcp};
}
```

66 SUFFIX_AUTOMATON

```
struct state {
  int len, link, cnt_tmp = 0, cnt = 0;
  map<char, int> next;
};

const int MAXLEN = 100000;
state st[2*MAXLEN];
int dp[2*MAXLEN];
int sz, last;

void sa_init() {
  st[0].len = 0;
  st[0].link = - 1;
  sz++;
  last = 0;
  memset(dp, -1, sizeof(dp));
}

void sa_extend(char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  int p = last;
  while (p != -1 && !st[p].next.count(c)) {
    st[p].next[c] = cur;
    p = st[p].link;
  }
  if (p == -1) {
    st[cur].link = 0;
  } else {
    int q = st[p].next[c];
    if (st[p].len + 1 == st[q].len) {
      st[cur].link = q;
    } else {
      int clone = sz++;
      st[clone].len = st[p].len + 1;
      st[clone].next = st[q].next;
      st[clone].link = st[q].link;
      while (p != -1 && st[p].next[c] == q) {
        st[p].next[c] = clone;
        p = st[p].link;
      }
      st[q].link = st[cur].link = clone;
    }
  }
  last = cur;
}

## Count Occurence
int occurence(string p){
  int at = 0;
  for (int i = 0; p[i]; ++i){
    if(st[at].next.count(p[i]) == 0){
      return 0;
    }
    else{
        at = st[at].next[p[i]];
    }
  }
  return st[at].cnt_tmp;
}

vector<int> used;

void dfs(int x){
  used[x]=1;
  for(auto it:st[x].next) {
    if(!used[it.second]) dfs(it.second);
    if(it.first=='#') st[x].cnt_tmp++;
    else st[x].cnt_tmp+=st[it.second].cnt_tmp;
    st[x].cnt+=st[it.second].cnt;
  }
  st[x].cnt+=st[x].cnt_tmp;
}

## number of distinct substring O(n)
long long disub(int at){
  if(dp[at] != -1)
    return 0;
  dp[at] = 0;
  long long ret = 0;
  for(auto x : st[at].next)
    ret += disub(x.second);
  if(at != 0)
    ret += (st[at].len - st[st[at].link].len);
  return ret;
}

## longest common substring: O(|T|)
int lcs (string S, string T) {
  sa_init();
  for (int i = 0; i < S.size(); i++)
    sa_extend(S[i]);

  int v = 0, l = 0, best = 0, bestpos = 0;
  for (int i = 0; i < T.size(); i++) {
    while (v && !st[v].next.count(T[i])) {
      v = st[v].link ;
      l = st[v].len ;
    }
    if (st[v].next.count(T[i])) {
      v = st [v].next[T[i]];
      l++;
    }
    if (l > best) {
      best = l;
      bestpos = i;
    }
  }
  return best;
}

## Distinct Substring
long long disub(int at){
  long long ret = 1;
  for(auto x : st[at].next){
      ret += disub(x.second);
  }
  return ret-1;
}

int main(){
  int T, caseno = 0;
  scanf("%d", &T);
  while(T--){
    int q;    cin >> q;
    sa_init();
    string s;    cin >> s;
    cout << s << endl;
    s += "#";
    for (int i = 0; s[i]; ++i){
      sa_extend(s[i]);
    }
    used.assign(sz,0);
    dfs(0);
    printf("Case %d:\n", ++caseno);
    while (q--){
      string p;    cin >> p;
      int ans = occurence(p);
      cout << ans << endl;
    }
  }
  return 0;
}
1. Finding Pattern
2. Frequency of each stat
3. First Occurrence
4. Last Occurrence
5. All Occurrenc
6. Longest Repeated substring:
7. Count number of different substring
8. Total length of different substring
9. k-th smallest distinct substring
10. K-th smallest substring
11. Smallest Cyclic Shift
12. Find borders
13. Find Periods:
14. Longest Common Substring
```

67 TOP_SORT

```
vector<int> top_sort(int n, vector<int> *adj, bool
↪   &has_cycle) {
  vector<int> indeg(n);
  for (int u = 0; u < n; ++u) {
    for (int v: adj[u]) {
      indeg[v]++;
    }
  }

  queue<int> q;
  for (int u = 0; u < n; ++u) {
    if (!indeg[u]) {
      q.push(u);
    }
  }

  vector<int> order;
  while (!q.empty()) {
```

```
  int u = q.front();  q.pop();
  order.push_back(u);
  for (int v: adj[u]) {
   indeg[v]--;
   if (!indeg[v]) {
    q.push(v);
   }
  }
 }
 has_cycle = order.size() < n;
 return order;
}
```

68 TREAP

```
## Typical TEAP
struct node {
 ll val, prior, sz, sum;
 node *l, *r;
 node(int val, int prior, int sz) : val(val),
↪  prior(prior), sz(sz), sum(0), l(nullptr),
↪  r(nullptr){}
};
using pnode = node*;
pnode root;
pnode new_node(ll val){
 return new node(val, rand(), 1);
}
int get_sz(pnode u){
 return u? u->sz: 0;
}
void update(pnode u){
 if (!u)  return ;
 u->sz = get_sz(u->l) + 1 + get_sz(u->r);
 u->sum = u->val + (u->l? u->l->sum: 0) + (u->r?
↪  u->r->sum: 0);
}
void split(pnode u, pnode &l, pnode &r, ll val){
 if(!u)  l = r = NULL;
 else if(val > u->val) split(u->r, u->r, r, val),
↪  l = u;
 else split(u->l, l, u->l, val), r = u;
 update(u);
}
void merge(pnode &u, pnode l, pnode r){
 if(!l or !r) u = l? l: r;
 if(l->prior > r->prior) merge(l->r, l->r, r),  u
↪  = l;
 else merge(r->l, l, r->l),  u = r;
 update(u);
}
void insert(pnode &u, pnode it){
 if(!u)  u = it;
 else if(it->prior > u->prior) split(u, it->l,
↪  it->r, it->val), u = it;
 else insert(it->val <= u->val ? u->l: u->r, it);
 update(u);
}
void erase(pnode &u, ll val){
 if(!u)  return ;
 if(val == u->val)  merge(u, u->l, u->r);
 else  erase(val < u->val ? u->l: u->r, val);
 update(u);
}
bool present(pnode u, int x){
 if(!u)  return false;
 if(u->val == x)  return true;
 if(u->val < x) return present(u->r, x);
 return present(u->l, x);
}
ll kth(pnode u, int k){
 if(get_sz(u) < k) return INT_MIN;
 if(get_sz(u->l) == k-1)  return u->val;
 if(get_sz(u->l) < k-1) return kth(u->r, k -
↪  get_sz(u->l) - 1);
 return kth(u->l, k);
}
int cnt_less(pnode u, ll x){
 if(!u)  return 0;
 if(x <= u->val)  return cnt_less(u->l, x);
 return get_sz(u->l) + 1 + cnt_less(u->r, x);
}
ll sum_less(pnode u, ll x) {
 if (!u)  return 0;
 if (x <= u->val)  return sum_less(u->l, x);
 return u->val + (u->l? u->l->sum: 0) +
↪  sum_less(u->r, x);
}
insert(root, new_node(4));
insert(root, new_node(2));
insert(root, new_node(7));
cout << cnt_less(root, 5) << " " << sum_less(root,
↪  5) << "\n";
root = NULL;
insert(root, new_node(8));
insert(root, new_node(1));
insert(root, new_node(5));
cout << cnt_less(root, 7) << " " << sum_less(root,
↪  7) << "\n";

## Implicit TREAP
struct node {
 ll val, sum;
 int prior, sz, rev;
 node *l, *r;
 node(){}
 node(ll val): val(val), sum(val), prior(rand()),
↪  sz(1), rev(0), l(nullptr), r(nullptr) {}
};
using pnode = node*;
pnode root;
int get_sz(pnode t) {
 return t? t->sz: 0;
}
ll get_sum(pnode t) {
 return t? t->sum: 0;
}
void update(pnode &t) {
 if (!t)  return ;
 t->sz = get_sz(t->l) + 1 + get_sz(t->r);
 t->sum = get_sum(t->l) + t->val + get_sum(t->r);
}
void push(pnode t) {
 if (t and t->rev) {
  swap(t->l, t->r);
  t->rev = 0;
  if (t->l) {
   t->l->rev ^= 1;
  }
  if (t->r) {
   t->r->rev ^= 1;
  }
 }
}
void merge(pnode &t, pnode l, pnode r){
 push(l);
 push(r);
 if(!l or !r)  t=l?l:r;
 else if(l->prior > r->prior)  merge(l->r, l->r,
↪  r), t=l;
 else  merge(r->l,l,r->l) , t=r;
 update(t);
}
void split(pnode t, pnode &l, pnode &r, int pos,
↪  int add=0) {
 push(t);
 if(!t)  return void(r=l=NULL);
 int cur_pos = get_sz(t->l)+add;
 if(pos > cur_pos) split(t->r, t->r, r, pos,
↪  cur_pos+1), l = t;
 else   split(t->l, l, t->l, pos, add), r=t;
 update(t);
}
void insert(pnode &t, pnode it, int i) {
 pnode t1, t2;
 split(t, t1, t2, i);
 merge(t1, t1, it);
 merge(t, t1, t2);
}
void reverse(pnode &t, int l, int r) {
 pnode lt, mt, rt;
 split(t, t, rt, r + 1);
 split(t, lt, mt, l);
 mt->rev = 1;
 merge(mt, mt, rt);
 merge(t, lt, mt);
}
ll rsum(pnode& t, int l, int r) {
 pnode lt, mt, rt;
 split(t, t, rt, r + 1);
 split(t, lt, mt, l);
 ll ret = mt->sum;
 merge(mt, mt, rt);
 merge(t, lt, mt);
 return ret;
}
int n, q;  cin >> n >> q;
vector<ll> a(n);
for (auto &ai: a) {
 cin >> ai;
}
for (int i = 0; i < n; ++i) {
 insert(root, new node(a[i]), i);
}
while (q--) {
 int tp, l, r;  cin >> tp >> l >> r;  l--, r--;
 if (tp == 1) {
  reverse(root, l, r);
 }
 else {
  cout << rsum(root, l, r) << "\n";
 }
}
```

69 WAVELET_TREE

```
typedef vector<int>::iterator itr;
const int N=1e6+7;
struct WT {
  int sigma=0;
  vector<int> a;
  vector<vector<int>> c;
  vector<vector<ll>> pref;
  set<int> st;
  map<int, int> maf;
  int rmaf[N];
  WT(vector<int> a):a(a){
    for(int& x: a){
      st.insert(x);
    }
    for(int x: st){
      maf[x]=sigma, rmaf[sigma++]=x;
    }
    sigma=st.size();
    c.resize(2*sigma),  pref.resize(2*sigma);
    build(a.begin(), a.end(), 0, sigma-1, 1);
  }

  void build(itr st, itr ed, int lo, int hi, int v){
    if(lo==hi) return;
    int mid = (lo+hi)/2;
    c[v].reserve(ed-st+1), pref[v].reserve(ed-st+1);
    c[v].push_back(0),  pref[v].push_back(0);
    for(itr it=st; it!=ed; ++it){
      c[v].push_back(c[v].back()+(maf[*it]<=mid));
      pref[v].push_back(pref[v].back()+(maf[*it]<=mid)*(*it));
    }
    itr m = stable_partition(st, ed, [=](int x){return maf[x]<=mid;});
    build(st, m, lo, mid, 2*v);
    build(m, ed, mid+1, hi, 2*v+1);
  }

  int count(int k, int idx){
    idx++;
    if(idx <1 or !maf.count(k)) return 0;
    k=maf[k];
    int v=1, lo=0, hi=sigma-1;
    while(lo<hi){
      int mid=(lo+hi)/2;
      int inlft=c[v][idx];
      if(k<=mid){
        idx=inlft, hi=mid, v=2*v;
      } else {
        idx-=inlft, lo=mid+1, v=2*v+1;
      }
    }
    return idx;
  }

  int count(int k, int l, int r){
    return count(k,r)-count(k,l-1);
  }

  int kth(int k, int l, int r){
    r++;
    int v=1, lo=0, hi=sigma-1, mid, inlftl, inlftr;
    while(lo<hi){
      mid = (lo+hi)/2;
      inlftl = c[v][l];
      inlftr = c[v][r];
      if(k<=inlftr-inlftl){
        l=inlftl, r=inlftr;
        hi=mid, v=2*v;
      } else {
        k -= inlftr-inlftl, l-=inlftl, r-=inlftr;
        lo=mid+1, v=2*v+1;
      }
    }
    return rmaf[lo];
  }

  int lte(int k, int l, int r){
    if(!maf.count(k)) return 0;
    k=maf[k];
    r++;
    int v=1, lo=0, hi=sigma-1, ret=0;
    while(lo<hi){
      int mid=(lo+hi)/2;
      int inlftl=c[v][l];
      int inlftr=c[v][r];
      if(k<=mid){
        l=inlftl, r=inlftr;
        hi=mid, v=2*v;
      } else {
        ret += inlftr-inlftl, l-=inlftl, r-=inlftr;
        lo=mid+1, v=2*v+1;
      }
    }
    ret += r-l;
    return ret;
  }

  ll sumlte(int k, int l, int r){
    auto it = st.upper_bound(k);
    if(it==st.begin())  return 0;
    it--;
    k = maf[*it];
    r++;
    int v=1, lo=0, hi=sigma-1;
    ll ret=0;
    while(lo<hi){
      int mid=(lo+hi)/2;
      int inlftl=c[v][l];
      int inlftr=c[v][r];
      if(k<=mid){
        l=inlftl, r=inlftr;
        hi=mid, v=2*v;
      } else {
        ret += pref[v][r]-pref[v][l], l-=inlftl, r-=inlftr;
        lo=mid+1, v=2*v+1;
      }
    }
    ret += (r-l)*rmaf[lo];
    return ret;
  }

  void clear(){
    a.clear();
    for (int i = 0; i < c.size(); ++i){
      c[i].clear();
    }
    for (int i = 0; i < pref.size(); ++i){
      pref.clear();
    }
    st.clear();  maf.clear();
  }
};
```

70 XOR_BASIS

```
// find rank of SLAE modulo 2 field
ll rnk, basis[D];
void insert_vector(ll mask){
  for (int i = D-1; i >= 0; --i){
    if((mask & (1ll << i)) == 0)  continue;
    if(!basis[i]){
      basis[i] = mask;
      rnk++;
      return;
    } else {
      mask ^= basis[i];
    }
  }
}
```

71 Z_ALGORITHM

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