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 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) {
}
```

9 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++;
      }
    }
  }
}
```

```cpp
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;
      }
    }
  }
}
```

10 CDQ

```
## Problems related to pair
  - 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
```

11 CENTROID_DECOMPOSITION

```cpp
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();
```

12 CONVOLUTION

```cpp
## 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;
}
```

13 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;
## 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;
mt19937 rng(chrono::steady_clock::now().time_since_
  epoch().count());
int x = rng() % 495;
```

14 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];
}
```

15 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;
}
```

16 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];
   }
}
```

17 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)
  }
}
```

18 DP_OPTIMIZATION

```
## CHT

## Dynamic 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));
}
```

19 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);
```

20 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));
}
```

21 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();
}
```

22 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;
```

```cpp
  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 ;
}
```

23 GEOMETRY

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

const int N = 3e5 + 9;

const double inf = 1e100;
const double eps = 1e-9;
const double PI = acos((double)-1.0);
int sign(double x) { return (x > eps) - (x < -eps); }
struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &a) const { return PT(x + a.x, y + a.y); }
    PT operator - (const PT &a) const { return PT(x - a.x, y - a.y); }
    PT operator * (const double a) const { return PT(x * a, y * a); }
    friend PT operator * (const double &a, const PT &b) { return PT(a * b.x, a * b.y); }
    PT operator / (const double a) const { return PT(x / a, y / a); }
    bool operator == (PT a) const { return sign(a.x - x) == 0 && sign(a.y - y) == 0; }
    bool operator != (PT a) const { return !(*this == a); }
    bool operator < (PT a) const { return sign(a.x - x) == 0 ? y < a.y : x < a.x; }
    bool operator > (PT a) const { return sign(a.x - x) == 0 ? y > a.y : x > a.x; }
    double norm() { return sqrt(x * x + y * y); }
    double norm2() { return x * x + y * y; }
    PT perp() { return PT(-y, x); }
    double arg() { return atan2(y, x); }
    PT truncate(double r) { // returns a vector with norm r and having same direction
        double k = norm();
        if (!sign(k)) return *this;
        r /= k;
        return PT(x * r, y * r);
    }
};
inline double dot(PT a, PT b) { return a.x * b.x + a.y * b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a - b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a - b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y * b.x; }
inline double cross2(PT a, PT b, PT c) { return cross(b - a, c - a); }
inline int orientation(PT a, PT b, PT c) { return sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, double t) { return PT(a.x * cos(t) - a.y * sin(t), a.x * sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, double t) { return PT(a.x * cos(t) + a.y * sin(t), -a.x * sin(t) + a.y * cos(t)); }
double SQ(double x) { return x * x; }
double rad_to_deg(double r) { return (r * 180.0 / PI); }
double deg_to_rad(double d) { return (d * PI / 180.0); }
double get_angle(PT a, PT b) {
    double costheta = dot(a, b) / a.norm() / b.norm();
    return acos(max((double)-1.0, min((double)1.0, costheta)));
}
bool is_point_in_angle(PT b, PT a, PT c, PT p) { // does point p lie in angle <bac
    assert(orientation(a, b, c) != 0);
    if (orientation(a, c, b) < 0) swap(b, c);
    return orientation(a, c, p) >= 0 && orientation(a, b, p) <= 0;
}
bool half(PT p) {
    return p.y > 0.0 || (p.y == 0.0 && p.x < 0.0);
}
void polar_sort(vector<PT> &v) { // sort points in counterclockwise
    sort(v.begin(), v.end(), [](PT a,PT b) {
        return make_tuple(half(a), 0.0, a.norm2()) < make_tuple(half(b), cross(a, b), b.norm2());
    });
}
struct line {
    PT a, b; // goes through points a and b
    PT v; double c;  //line form: direction vec [cross] (x, y) = c
    line() {}
    //direction vector v and offset c
 line(PT v, double c) : v(v), c(c) {
        auto p = get_points();
        a = p.first; b = p.second;
 }
 // equation ax + by + c = 0
 line(double _a, double _b, double _c) : v({_b, -_a}), c(-_c) {
  auto p = get_points();
        a = p.first; b = p.second;
 }
 // goes through points p and q
 line(PT p, PT q) : v(q - p), c(cross(v, p)), a(p), b(q) {}
 pair<PT, PT> get_points() { //extract any two points from this line
  PT p, q; double a = -v.y, b = v.x; // ax + by = -c
  if (sign(a) == 0) {
            p = PT(0, -c / b);
            q = PT(1, -c / b);
        }
        else if (sign(b) == 0) {
            p = PT(-c / a, 0);
            q = PT(-c / a, 1);
        }
        else {
            p = PT(0, -c / b);
            q = PT(1, (-c - a) / b);
        }
        return {p, q};
 }
    //ax + by + c = 0
    array<double, 3> get_abc() {
        double a = -v.y, b = v.x;
        return {a, b, c};
    }
    // 1 if on the left, -1 if on the right, 0 if on the line
    int side(PT p) { return sign(cross(v, p) - c); }
    // line that is perpendicular to this and goes through point p
    line perpendicular_through(PT p) { return {p, p + perp(v)}; }
```

```cpp
    // translate the line by vector t i.e. shifting it by vector t
    line translate(PT t) { return {v, c + cross(v, t)}; }
    // compare two points by their orthogonal projection on this line
    // a projection point comes before another if it comes first according to vector v
    bool cmp_by_projection(PT p, PT q) { return dot(v, p) < dot(v, q); }
  line shift_left(double d) {
   PT z = v.perp().truncate(d);
   return line(a + z, b + z);
  }
};
// find a point from a through b with distance d
PT point_along_line(PT a, PT b, double d) {
    return a + (((b - a) / (b - a).norm()) * d);
}
// projection point c onto line through a and b assuming a != b
PT project_from_point_to_line(PT a, PT b, PT c) {
    return a + (b - a) * dot(c - a, b - a) / (b - a).norm2();
}
// reflection point c onto line through a and b assuming a != b
PT reflection_from_point_to_line(PT a, PT b, PT c) {
    PT p = project_from_point_to_line(a,b,c);
    return point_along_line(c, p, 2.0 * dist(c, p));
}
// minimum distance from point c to line through a and b
double dist_from_point_to_line(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) / (b - a).norm());
}
// returns true if  point p is on line segment ab
bool is_point_on_seg(PT a, PT b, PT p) {
    if (fabs(cross(p - b, a - b)) < eps) {
        if (p.x < min(a.x, b.x) || p.x > max(a.x, b.x)) return false;
        if (p.y < min(a.y, b.y) || p.y > max(a.y, b.y)) return false;
        return true;
    }
    return false;
}
// minimum distance point from point c to segment ab that lies on segment ab
PT project_from_point_to_seg(PT a, PT b, PT c) {
    double r = dist2(a, b);
    if (fabs(r) < eps) return a;
    r = dot(c - a, b - a) / r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b - a) * r;
}
// minimum distance from point c to segment ab
double dist_from_point_to_seg(PT a, PT b, PT c) {
    return dist(c, project_from_point_to_seg(a, b, c));
}
// 0 if not parallel, 1 if parallel, 2 if collinear
bool is_parallel(PT a, PT b, PT c, PT d) {
    double k = fabs(cross(b - a, d - c));
    if (k < eps){
        if (fabs(cross(a - b, a - c)) < eps && fabs(cross(c - d, c - a)) < eps) return 2;
        else return 1;
    }
    else return 0;
}
// check if two lines are same
bool are_lines_same(PT a, PT b, PT c, PT d) {
    if (fabs(cross(a - c, c - d)) < eps && fabs(cross(b - c, c - d)) < eps) return true;
    return false;
}
// bisector vector of <abc
PT angle_bisector(PT &a, PT &b, PT &c){
    PT p = a - b, q = c - b;
    return p + q * sqrt(dot(p, p) / dot(q, q));
}
// 1 if point is ccw to the line, 2 if point is cw to the line, 3 if point is on the line
int point_line_relation(PT a, PT b, PT p) {
    int c = sign(cross(p - a, b - a));
    if (c < 0) return 1;
    if (c > 0) return 2;
    return 3;
}
// intersection point between ab and cd assuming unique intersection exists
bool line_line_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    double a1 = a.y - b.y, b1 = b.x - a.x, c1 = cross(a, b);
    double a2 = c.y - d.y, b2 = d.x - c.x, c2 = cross(c, d);
    double det = a1 * b2 - a2 * b1;
    if (det == 0) return 0;
    ans = PT((b1 * c2 - b2 * c1) / det, (c1 * a2 - a1 * c2) / det);
    return 1;
}
// intersection point between segment ab and segment cd assuming unique intersection exists
bool seg_seg_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    double oa = cross2(c, d, a), ob = cross2(c, d, b);
    double oc = cross2(a, b, c), od = cross2(a, b, d);
    if (oa * ob < 0 && oc * od < 0){
        ans = (a * ob - b * oa) / (ob - oa);
        return 1;
    }
    else return 0;
}
// intersection point between segment ab and segment cd assuming unique intersection may not exists
// se.size()==0 means no intersection
// se.size()==1 means one intersection
// se.size()==2 means range intersection
set<PT> seg_seg_intersection_inside(PT a,  PT b, PT c,  PT d) {
    PT ans;
    if (seg_seg_intersection(a, b, c, d, ans)) return {ans};
    set<PT> se;
    if (is_point_on_seg(c, d, a)) se.insert(a);
    if (is_point_on_seg(c, d, b)) se.insert(b);
    if (is_point_on_seg(a, b, c)) se.insert(c);
    if (is_point_on_seg(a, b, d)) se.insert(d);
    return se;
}
// intersection  between segment ab and line cd
// 0 if do not intersect, 1 if proper intersect, 2 if segment intersect
int seg_line_relation(PT a, PT b, PT c, PT d) {
    double p = cross2(c, d, a);
    double q = cross2(c, d, b);
    if (sign(p) == 0 && sign(q) == 0) return 2;
    else if (p * q < 0) return 1;
    else return 0;
}
// intersection between segament ab and line cd assuming unique intersection exists
bool seg_line_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    bool k = seg_line_relation(a, b, c, d);
    assert(k != 2);
    if (k) line_line_intersection(a, b, c, d, ans);
    return k;
}
// minimum distance from segment ab to segment cd
double dist_from_seg_to_seg(PT a, PT b, PT c, PT d) {
    PT dummy;
    if (seg_seg_intersection(a, b, c, d, dummy)) return 0.0;
    else return min({dist_from_point_to_seg(a, b, c), dist_from_point_to_seg(a, b, d),
        dist_from_point_to_seg(c, d, a), dist_from_point_to_seg(c, d, b)});
}
// minimum distance from point c to ray (starting point a and direction vector b)
double dist_from_point_to_ray(PT a, PT b, PT c) {
    b = a + b;
    double r = dot(c - a, b - a);
    if (r < 0.0) return dist(c, a);
    return dist_from_point_to_line(a, b, c);
}
// starting point as and direction vector ad
bool ray_ray_intersection(PT as, PT ad, PT bs, PT bd) {
    double dx = bs.x - as.x, dy = bs.y - as.y;
    double det = bd.x * ad.y - bd.y * ad.x;
    if (fabs(det) < eps) return 0;
    double u = (dy * bd.x - dx * bd.y) / det;
    double v = (dy * ad.x - dx * ad.y) / det;
    if (sign(u) >= 0 && sign(v) >= 0) return 1;
    else return 0;
}
double ray_ray_distance(PT as, PT ad, PT bs, PT bd) {
    if (ray_ray_intersection(as, ad, bs, bd)) return 0.0;
    double ans = dist_from_point_to_ray(as, ad, bs);
```

```cpp
    ans = min(ans, dist_from_point_to_ray(bs, bd, as));
    return ans;
}
struct circle {
    PT p; double r;
    circle() {}
    circle(PT _p, double _r): p(_p), r(_r) {};
    // center (x, y) and radius r
    circle(double x, double y, double _r): p(PT(x, y)), r(_r) {};
    // circumcircle of a triangle
    // the three points must be unique
    circle(PT a, PT b, PT c) {
        b = (a + b) * 0.5;
        c = (a + c) * 0.5;
        line_line_intersection(b, b + rotatecw90(a - b), c, c + rotatecw90(a - c), p);
        r = dist(a, p);
    }
    // inscribed circle of a triangle
    circle(PT a, PT b, PT c, bool t) {
        line u, v;
        double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y, c.x - a.x);
        u.a = a;
        u.b = u.a + (PT(cos((n + m)/2.0), sin((n + m)/2.0)));
        v.a = b;
        m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y - b.y, c.x - b.x);
        v.b = v.a + (PT(cos((n + m)/2.0), sin((n + m)/2.0)));
        line_line_intersection(u.a, u.b, v.a, v.b, p);
        r = dist_from_point_to_seg(a, b, p);
    }
    bool operator == (circle v) { return p == v.p && sign(r - v.r) == 0; }
    double area() { return PI * r * r; }
    double circumference() { return 2.0 * PI * r; }
};
//0 if outside, 1 if on circumference, 2 if inside circle
int circle_point_relation(PT p, double r, PT b) {
    double d = dist(p, b);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
// 0 if outside, 1 if on circumference, 2 if inside circle
int circle_line_relation(PT p, double r, PT a, PT b) {
    double d = dist_from_point_to_line(a, b, p);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
//compute intersection of line through points a and b with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, double r, PT a, PT b) {
    vector<PT> ret;
    b = b - a; a = a - c;
    double A = dot(b, b), B = dot(a, b);
    double C = dot(a, a) - r * r, D = B * B - A * C;
    if (D < -eps) return ret;
    ret.push_back(c + a + b * (-B + sqrt(D + eps)) / A);
    if (D > eps) ret.push_back(c + a + b * (-B - sqrt(D)) / A);
    return ret;
}
//5 - outside and do not intersect
//4 - intersect outside in one point
//3 - intersect in 2 points
//2 - intersect inside in one point
//1 - inside and do not intersect
int circle_circle_relation(PT a, double r, PT b, double R) {
    double d = dist(a, b);
    if (sign(d - r - R) > 0)  return 5;
    if (sign(d - r - R) == 0) return 4;
    double l = fabs(r - R);
    if (sign(d - r - R) < 0 && sign(d - l) > 0) return 3;
    if (sign(d - l) == 0) return 2;
    if (sign(d - l) < 0) return 1;
    assert(0); return -1;
}
vector<PT> circle_circle_intersection(PT a, double r, PT b, double R) {
    if (a == b && sign(r - R) == 0) return {PT(1e18, 1e18)};
    vector<PT> ret;
    double d = sqrt(dist2(a,  b));
    if (d > r + R || d + min(r,  R) < max(r,  R)) return ret;
    double x = (d * d - R * R + r * r) / (2 * d);
    double y = sqrt(r * r - x * x);
    PT v = (b - a) / d;
    ret.push_back(a + v * x  +  rotateccw90(v) * y);
    if (y > 0) ret.push_back(a + v * x - rotateccw90(v) * y);
    return ret;
}
// returns two circle c1, c2 through points a, b and of radius r
// 0 if there is no such circle, 1 if one circle, 2 if two circle
int get_circle(PT a, PT b, double r, circle &c1, circle &c2) {
    vector<PT> v = circle_circle_intersection(a, r, b, r);
    int t = v.size();
    if (!t) return 0;
    c1.p = v[0], c1.r = r;
    if (t == 2) c2.p = v[1], c2.r = r;
    return t;
}
// returns two circle c1, c2 which is tangent to line u,  goes through
// point q and has radius r1; 0 for no circle, 1 if c1 = c2 , 2 if c1 != c2
int get_circle(line u, PT q, double r1, circle &c1, circle &c2) {
    double d = dist_from_point_to_line(u.a, u.b, q);
    if (sign(d - r1 * 2.0) > 0) return 0;
    if (sign(d) == 0) {
     cout << u.v.x << ' ' << u.v.y << '\n';
        c1.p = q + rotateccw90(u.v).truncate(r1);
        c2.p = q + rotatecw90(u.v).truncate(r1);
        c1.r = c2.r = r1;
        return 2;
    }
    line u1 = line(u.a + rotateccw90(u.v).truncate(r1), u.b + rotateccw90(u.v).truncate(r1));
    line u2 = line(u.a + rotatecw90(u.v).truncate(r1), u.b + rotatecw90(u.v).truncate(r1));
    circle cc = circle(q, r1);
    PT p1, p2; vector<PT> v;
    v = circle_line_intersection(q, r1, u1.a, u1.b);
    if (!v.size()) v = circle_line_intersection(q, r1, u2.a, u2.b);
    v.push_back(v[0]);
    p1 = v[0], p2 = v[1];
    c1 = circle(p1, r1);
    if (p1 == p2) {
        c2 = c1;
        return 1;
    }
    c2 = circle(p2, r1);
    return 2;
}
// returns area of intersection between two circles
double circle_circle_area(PT a, double r1, PT b, double r2) {
    double d = (a - b).norm();
    if(r1 + r2 < d + eps) return 0;
    if(r1 + d < r2 + eps) return PI * r1 * r1;
    if(r2 + d < r1 + eps) return PI * r2 * r2;
    double theta_1 = acos((r1 * r1 + d * d - r2 * r2) / (2 * r1 * d)),
     theta_2 = acos((r2 * r2 + d * d - r1 * r1)/(2 * r2 * d));
    return r1 * r1 * (theta_1 - sin(2 * theta_1)/2.) + r2 * r2 * (theta_2 - sin(2 * theta_2)/2.);
}
// tangent lines from point q to the circle
int tangent_lines_from_point(PT p, double r, PT q, line &u, line &v) {
    int x = sign(dist2(p, q) - r * r);
    if (x < 0) return 0; // point in cricle
    if (x == 0) { // point on circle
        u = line(q, q + rotateccw90(q - p));
        v = u;
        return 1;
    }
    double d = dist(p, q);
    double l = r * r / d;
    double h = sqrt(r * r - l * l);
    u = line(q, p + ((q - p).truncate(l) + (rotateccw90(q - p).truncate(h))));
    v = line(q, p + ((q - p).truncate(l) + (rotatecw90(q - p).truncate(h))));
    return 2;
}
// returns outer tangents line of two circles
```

```
// if inner == 1 it returns inner tangent lines
int tangents_lines_from_circle(PT c1, double r1, PT c2, double r2, bool inner, line &u, line &v) {
    if (inner) r2 = -r2;
    PT d = c2 - c1;
    double dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) {
        assert(h2 != 0);
        return 0;
    }
    vector<pair<PT, PT>>out;
    for (int tmp: {- 1, 1}) {
        PT v = (d * dr + rotateccw90(d) * sqrt(h2) * tmp) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    u = line(out[0].first, out[0].second);
    if (out.size() == 2) v = line(out[1].first, out[1].second);
    return 1 + (h2 > 0);
}
//O(n^2 log n)
struct CircleUnion {
    int n;
    double x[2020], y[2020], r[2020];
    int covered[2020];
    vector<pair<double, double> > seg, cover;
    double arc, pol;
    inline int sign(double x) {return x < -eps ? -1 : x > eps;}
    inline int sign(double x, double y) {return sign(x - y);}
    inline double SQ(const double x) {return x * x;}
    inline double dist(double x1, double y1, double x2, double y2) {return sqrt(SQ(x1 - x2) + SQ(y1 - y2));}
    inline double angle(double A, double B, double C) {
        double val = (SQ(A) + SQ(B) - SQ(C)) / (2 * A * B);
        if (val < -1) val = -1;
        if (val > +1) val = +1;
        return acos(val);
    }
    CircleUnion() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void init() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void add(double xx, double yy, double rr) {
        x[n] = xx, y[n] = yy, r[n] = rr, covered[n] = 0, n++;
    }
    void getarea(int i, double lef, double rig) {
        arc += 0.5 * r[i] * r[i] * (rig - lef - sin(rig - lef));
        double x1 = x[i] + r[i] * cos(lef), y1 = y[i] + r[i] * sin(lef);
        double x2 = x[i] + r[i] * cos(rig), y2 = y[i] + r[i] * sin(rig);
        pol += x1 * y2 - x2 * y1;
    }
    double solve() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (!sign(x[i] - x[j]) && !sign(y[i] - y[j]) && !sign(r[i] - r[j])) {
                    r[i] = 0.0;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i != j && sign(r[j] - r[i]) >= 0 && sign(dist(x[i], y[i], x[j], y[j]) - (r[j] - r[i])) <= 0) {
                    covered[i] = 1;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            if (sign(r[i]) && !covered[i]) {
                seg.clear();
                for (int j = 0; j < n; j++) {
                    if (i != j) {
                        double d = dist(x[i], y[i], x[j], y[j]);
                        if (sign(d - (r[j] + r[i])) >= 0 || sign(d - abs(r[j] - r[i])) <= 0) {
                            continue;
                        }
                        double alpha = atan2(y[j] - y[i], x[j] - x[i]);
                        double beta = angle(r[i], d, r[j]);
                        pair<double, double> tmp(alpha - beta, alpha + beta);
                        if (sign(tmp.first) <= 0 && sign(tmp.second) <= 0) {
                            seg.push_back(pair<double, double>(2 * PI + tmp.first, 2 * PI + tmp.second));
                        }
                        else if (sign(tmp.first) < 0) {
                            seg.push_back(pair<double, double>(2 * PI + tmp.first, 2 * PI));
                            seg.push_back(pair<double, double>(0, tmp.second));
                        }
                        else {
                            seg.push_back(tmp);
                        }
                    }
                }
                sort(seg.begin(), seg.end());
                double rig = 0;
                for (vector<pair<double, double> >::iterator iter = seg.begin(); iter != seg.end(); iter++) {
                    if (sign(rig - iter->first) >= 0) {
                        rig = max(rig, iter->second);
                    }
                    else {
                        getarea(i, rig, iter->first);
                        rig = iter->second;
                    }
                }
                if (!sign(rig)) {
                    arc += r[i] * r[i] * PI;
                }
                else {
                    getarea(i, rig, 2 * PI);
                }
            }
        }
        return pol / 2.0 + arc;
    }
} CU;
double area_of_triangle(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) * 0.5);
}
// -1 if strictly inside, 0 if on the polygon, 1 if strictly outside
int is_point_in_triangle(PT a, PT b, PT c, PT p) {
    if (sign(cross(b - a,c - a)) < 0) swap(b, c);
    int c1 = sign(cross(b - a,p - a));
    int c2 = sign(cross(c - b,p - b));
    int c3 = sign(cross(a - c,p - c));
    if (c1<0 || c2<0 || c3 < 0) return 1;
    if (c1 + c2 + c3 != 3) return 0;
    return -1;
}
double perimeter(vector<PT> &p) {
    double ans=0; int n = p.size();
    for (int i = 0; i < n; i++) ans += dist(p[i], p[(i + 1) % n]);
    return ans;
}
double area(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i + 1) % n]);
    return fabs(ans) * 0.5;
}
// centroid of a (possibly non-convex) polygon,
// assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
PT centroid(vector<PT> &p) {
    int n = p.size(); PT c(0, 0);
    double sum = 0;
    for (int i = 0; i < n; i++) sum += cross(p[i], p[(i + 1) % n]);
    double scale = 3.0 * sum;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
```

```
        c = c + (p[i] + p[j]) * cross(p[i], p[j]);
    }
    return c / scale;
}
// 0 if cw, 1 if ccw
bool get_direction(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i + 1) % n]);
    if (sign(ans) > 0) return 1;
    return 0;
}
// it returns a point such that the sum of distances
// from that point to all points in p  is minimum
// O(n log^2 MX)
PT geometric_median(vector<PT> p) {
 auto tot_dist = [&](PT z) {
     double res = 0;
     for (int i = 0; i < p.size(); i++) res += dist(p[i], z);
     return res;
 };
 auto findY = [&](double x) {
     double yl = -1e5, yr = 1e5;
     for (int i = 0; i < 60; i++) {
         double ym1 = yl + (yr - yl) / 3;
         double ym2 = yr - (yr - yl) / 3;
         double d1 = tot_dist(PT(x, ym1));
         double d2 = tot_dist(PT(x, ym2));
         if (d1 < d2) yr = ym2;
         else yl = ym1;
     }
     return pair<double, double> (yl, tot_dist(PT(x, yl)));
 };
    double xl = -1e5, xr = 1e5;
    for (int i = 0; i < 60; i++) {
        double xm1 = xl + (xr - xl) / 3;
        double xm2 = xr - (xr - xl) / 3;
        double y1, d1, y2, d2;
        auto z = findY(xm1); y1 = z.first; d1 = z.second;
        z = findY(xm2); y2 = z.first; d2 = z.second;
        if (d1 < d2) xr = xm2;
        else xl = xm1;
    }
    return {xl, findY(xl).first };
}
vector<PT> convex_hull(vector<PT> &p) {
 if (p.size() <= 1) return p;
 vector<PT> v = p;
    sort(v.begin(), v.end());
    vector<PT> up, dn;
    for (auto& p : v) {
        while (up.size() > 1 && orientation(up[up.size() - 2], up.back(), p) >= 0) {
            up.pop_back();
        }
        while (dn.size() > 1 && orientation(dn[dn.size() - 2], dn.back(), p) <= 0) {
            dn.pop_back();
        }
        up.push_back(p);
        dn.push_back(p);
    }
    v = dn;
    if (v.size() > 1) v.pop_back();
    reverse(up.begin(), up.end());
    up.pop_back();
    for (auto& p : up) {
        v.push_back(p);
    }
    if (v.size() == 2 && v[0] == v[1]) v.pop_back();
    return v;
}
 //checks if convex or not
bool is_convex(vector<PT> &p) {
    bool s[3]; s[0] = s[1] = s[2] = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] = 1;
        if (s[0] && s[2]) return 0;
    }
    return 1;
}
// -1 if strictly inside, 0 if on the polygon, 1 if strictly outside
// it must be strictly convex, otherwise make it strictly convex first
int is_point_in_convex(vector<PT> &p, const PT& x) { // O(log n)
    int n = p.size(); assert(n >= 3);
    int a = orientation(p[0], p[1], x), b = orientation(p[0], p[n - 1], x);
    if (a < 0 || b > 0) return 1;
    int l = 1, r = n - 1;
    while (l + 1 < r) {
        int mid = l + r >> 1;
        if (orientation(p[0], p[mid], x) >= 0) l = mid;
        else r = mid;
    }
    int k = orientation(p[l], p[r], x);
    if (k <= 0) return -k;
    if (l == 1 && a == 0) return 0;
    if (r == n - 1 && b == 0) return 0;
    return -1;
}
bool is_point_on_polygon(vector<PT> &p, const PT& z) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
     if (is_point_on_seg(p[i], p[(i + 1) % n], z)) return 1;
    }
    return 0;
}
// returns 1e9 if the point is on the polygon
int winding_number(vector<PT> &p, const PT& z) { // O(n)
    if (is_point_on_polygon(p, z)) return 1e9;
    int n = p.size(), ans = 0;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        bool below = p[i].y < z.y;
        if (below != (p[j].y < z.y)) {
            auto orient = orientation(z, p[j], p[i]);
            if (orient == 0) return 0;
            if (below == (orient > 0)) ans += below ? 1 : -1;
        }
    }
    return ans;
}
// -1 if strictly inside, 0 if on the polygon, 1 if strictly outside
int is_point_in_polygon(vector<PT> &p, const PT& z) { // O(n)
    int k = winding_number(p, z);
    return k == 1e9 ? 0 : k == 0 ? 1 : -1;
}
// id of the vertex having maximum dot product with z
// polygon must need to be convex
// top - upper right vertex
// for minimum dot prouct negate z and return -dot(z, p[id])
int extreme_vertex(vector<PT> &p, const PT &z, const int top) { // O(log n)
    int n = p.size();
    if (n == 1) return 0;
 double ans = dot(p[0], z); int id = 0;
    if (dot(p[top], z) > ans) ans = dot(p[top], z), id = top;
    int l = 1, r = top - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (dot(p[mid + 1], z) >= dot(p[mid], z)) l = mid + 1;
        else r = mid;
    }
    if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
    l = top + 1, r = n - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (dot(p[(mid + 1) % n], z) >= dot(p[mid], z)) l = mid + 1;
        else r = mid;
    }
    l %= n;
    if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
    return id;
}
double diameter(vector<PT> &p) {
    int n = (int)p.size();
    if (n == 1) return 0;
    if (n == 2) return dist(p[0], p[1]);
    double ans = 0;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n] - p[j]) >= 0) {
            ans = max(ans, dist2(p[i], p[j]));
            j = (j + 1) % n;
```

```
        }
        ans = max(ans, dist2(p[i], p[j]));
        i++;
    }
    return sqrt(ans);
}
double width(vector<PT> &p) {
    int n = (int)p.size();
    if (n <= 2) return 0;
    double ans = inf;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j +
↪   1) % n] - p[j]) >= 0) j = (j + 1) % n;
        ans = min(ans,
↪   dist_from_point_to_line(p[i], p[(i + 1) % n],
↪   p[j]));
        i++;
    }
    return ans;
}
// minimum perimeter
double minimum_enclosing_rectangle(vector<PT> &p) {
 int n = p.size();
 if (n <= 2) return perimeter(p);
 int mndot = 0; double tmp = dot(p[1] - p[0], p[0]);
 for (int i = 1; i < n; i++) {
  if (dot(p[1] - p[0], p[i]) <= tmp) {
   tmp = dot(p[1] - p[0], p[i]);
   mndot = i;
  }
 }
 double ans = inf;
 int i = 0, j = 1, mxdot = 1;
 while (i < n) {
  PT cur = p[(i + 1) % n] - p[i];
        while (cross(cur, p[(j + 1) % n] - p[j]) >=
↪   0) j = (j + 1) % n;
        while (dot(p[(mxdot + 1) % n], cur) >=
↪   dot(p[mxdot], cur)) mxdot = (mxdot + 1) % n;
        while (dot(p[(mndot + 1) % n], cur) <=
↪   dot(p[mndot], cur)) mndot = (mndot + 1) % n;
        ans = min(ans, 2.0 * ((dot(p[mxdot], cur) /
    cur.norm() - dot(p[mndot], cur) / cur.norm()) +
↪   dist_from_point_to_line(p[i], p[(i + 1) % n],
↪   p[j])));
        i++;
    }
    return ans;
}
// given n points, find the minimum enclosing
↪   circle of the points
// call convex_hull() before this for faster
↪   solution
// expected O(n)
circle minimum_enclosing_circle(vector<PT> &p) {
    random_shuffle(p.begin(), p.end());
    int n = p.size();
    circle c(p[0], 0);
    for (int i = 1; i < n; i++) {
        if (sign(dist(c.p, p[i]) - c.r) > 0) {
            c = circle(p[i], 0);
            for (int j = 0; j < i; j++) {
                if (sign(dist(c.p, p[j]) - c.r) >
↪   0) {
                    c = circle((p[i] + p[j]) / 2,
↪   dist(p[i], p[j]) / 2);
                    for (int k = 0; k < j; k++) {
                        if (sign(dist(c.p, p[k]) -
↪   c.r) > 0) {
                            c = circle(p[i], p[j],
↪   p[k]);
                        }
                    }
                }
            }
        }
    }
    return c;
}
## 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]);
});
## Angular 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;
}
```

24 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;
}
```

25 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];
      }
    }
```

```cpp
  }

  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));
  }
}
```

26 HLD

```cpp
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);
```

27 HOPCROFT_KARP

```cpp
// 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;
}
```

28 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};
}
```

29 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
```

30 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;
}
```

31 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;
}
```

32 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);
```

33 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;
}
```

34 NUMBER_THEORY

```
## Floor
ll floor (ll n, ll k) {
  if (n >= 0)  return n / k;
  return  (n - (k - 1)) / k;
}
## Ceil
ll ceil (ll n, ll k) {
  if (n >= 0)  return (n + k - 1) / k;
  return n / k;
}
## Highly Composite Number
1e6(240),1e9(1344),1e12(6720),1e14(17280)
## Harmonic Lemma (ceill)
ll i = 1;
while (i < n) {
  ll cval = (n + i - 1) / i;
  ll j = (n + cval - 2) / (cval - 1);
  // ceil(n/i)...ceil(n/(j - 1)) = cval
  cout << i << " " << j - 1  << ": " << cval << "\n";
  i = j;
}
ll bezout(ll a, ll b, ll &x, ll &y){
  if(b == 0){
    x=1, y=0;
    return a;
  }
  ll g = bezout(b, a%b, y, x);
  y -= a/b*x;
  return g;
}
ll mod_inv(ll a, ll m){
  ll x, y;
  ll g = bezout(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;
}
```

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;
}
```

35 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;
}
```

36 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
root[k] = add(root[k], i, x, 0, sz - 1);
root[ver++] = root[k];
cout << rsum(root[k], l, r, 0, sz - 1) << "\n";
## 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);
}
```

37 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;
}
```

38 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;
}
```

39 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;
  }
}
```

40 SEGMENT_TREE

```
## 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;
}
## 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);
}
```

41 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]);
    }
  }
}
```

42 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;
}
```

43 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]);
}
```

44 SPARSE_TABLE_2D

```
int st[N][N][LG][LG];
int a[N][N], lg2[N];
int yo(int x1, int y1, int x2, int y2) {
  x2++;
  y2++;
  int a = lg2[x2 - x1], b = lg2[y2 - y1];
  return max(
         max(st[x1][y1][a][b], st[x2 - (1 <<
↪  a)][y1][a][b]),
         max(st[x1][y2 - (1 << b)][a][b], st[x2 -
↪  (1 << a)][y2 - (1 << b)][a][b])
        );
}
void build(int n, int m) { // 0 indexed
  for (int i = 2; i < N; i++) lg2[i] = lg2[i >> 1]
↪  + 1;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      st[i][j][0][0] = a[i][j];
    }
  }
  for (int a = 0; a < LG; a++) {
    for (int b = 0; b < LG; b++) {
      if (a + b == 0) continue;
      for (int i = 0; i + (1 << a) <= n; i++) {
        for (int j = 0; j + (1 << b) <= m; j++) {
          if (!a) {
            st[i][j][a][b] = max(st[i][j][a][b -
↪  1], st[i][j + (1 << (b - 1))][a][b - 1]);
          } else {
            st[i][j][a][b] = max(st[i][j][a -
↪  1][b], st[i + (1 << (a - 1))][j][a - 1][b]);
          }
        }
      }
    }
  }
}
```

45 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;
}
```

46 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
```

47 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};
}
```

48 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
```

49 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);
}
## 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";
  }
}
```

50 XOR_BASIS

```
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];
  }
}
```

51 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;
}
```

52 note

Binomial Coefficent

- Factoring in: $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
- Sum over k: $\sum_{k=0}^{n}\binom{n}{k} = 2^n$
- Alternating sum: $\sum_{k=0}^{n}(-1)^k\binom{n}{k} = 0$
- Even and odd sum: $\sum_{k=0}^{n}\binom{n}{2k} = \sum_{k=0}^{n}\binom{n}{2k+1}2^{n-1}$
- The Hockey Stick Identity
- (Left to right) Sum over n and k: $\sum_{k=0}^{m}\binom{n+k}{k} = \binom{n+m-1}{m}$
- (Right to left) Sum over n: $\sum_{m=0}^{n}\binom{m}{k} = \binom{n+1}{k+1}$
- Sum of the squares: $\sum_{k=0}^{n}(\binom{n}{k})^2 = \binom{2n}{n}$
- Weighted sum: $\sum_{k=1}^{n}k\binom{n}{k} = n2^{n-1}$
- Connection with the fibonacci numbers: $\sum_{k=0}\binom{n-k}{k} = F_{n+1}$

Fibonacci Number

$$k = A - B$$

$$F_AF_B = F_{k+1}F_A^2 + F_kF_AF_{A-1} \tag{1}$$

$$\sum_{i=0}^{n}F_i^2 = F_{n+1}F_n \tag{2}$$

$$\sum_{i=0}^{n}F_iF_{i+1} = F_{n+1}^2 - (-1)^n \tag{3}$$

$$\sum_{i=0}^{n}F_iF_{i-1} = \sum_{i=0}^{n-1}F_iF_{i+1} \tag{4}$$

$$GCD(F_m, F_n) = F_{GCD(m,n)} \tag{5}$$

$$\sum_{0\le k\le n}\left(\begin{array}{c} n-k \\ k \end{array}\right) = Fib_{n+1} \tag{6}$$

$$\gcd(F_n, F_{n+1}) = \gcd(F_n, F_{n+2}) = \gcd(F_{n+1}, F_{n+2}) = 1 \tag{7}$$

Lucas Theorem

$$\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{n_i}(\mod p)$$

- $\binom{m}{n}$ is divisible by p if and only if at least one of the base-p digits of n is greater than the corresponding base-p digit of m.
- The number of entries in the nth row of Pascal's triangle that are not divisible by $p = \prod_{i=0}^{k}(n_i + 1)$
- All entries in the $(p^k - 1)th$ row are not divisble by p. - $\binom{n}{m} \equiv \lfloor\frac{n}{p}\rfloor(\mod p)$

2nd Kaplansky's Lemma

The number of ways of selecting k objects, no two consecutive,from n labelled objects arrayed in a circle is $\frac{n}{k}\binom{n-k-1}{k-1} = \frac{n}{n-k}\binom{n-k}{k}$

Distinct Objects into Distinct Bins

- n distinct objects into r distinct bins $= r^n$
- Among n distinct objects, exactly k of them into r distincts bins $= \binom{n}{k}r^k$
- n distinct objects into r distinct bins such that each bin contains at least one object $= \sum_{i=0}^{r}(-1)^i\binom{r}{i}(r-i)^n$

Stirling Number 2nd Kind

- Count the number of ways to partition a set of n labelled objects into k nonempty unlabelled subsets.

$$S(n,k) = S(n-1,k-1) + k * S(n-1,k)$$

$$S(0,0) = 1, S(>0,0) = 0, S(0,>0) = 0$$

- Time Complexity: $O(k\log n)$

```
ll get_sn2(int n, int k) {
  ll sn2 = 0;
  for (int i = 0; i <= k; ++i) {
    ll now = nCr(k, i) * powmod(k - i, n, mod) % mod;
    if (i&1) {
      now = now * (mod - 1) % mod;
    }
    sn2 = (sn2 + now) % mod;
  }
  sn2 = sn2 * ifact[k] % mod;
  return sn2;
}
```

- Number of ways to color a 1n grid using k colors such that each color is used at least once = $k!.sn2(n,k)$

Bell Numbers
Counts the number of partitions of a set.

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} \cdot B_k \tag{8}$$

$B_n = \sum_{k=0}^{n} S(n,k)$, where $S(n,k)$ is stirling number of second kind.

Partition Number
- Time Complexity: $O(n\sqrt{n})$

```
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;
  }
}
```

- The number of partitions of a positive integer n into exactly k parts equals the number of partitions of n whose largest part equals k

$$p_k(n) = p_k(n-k) + p_{k-1}(n-1)$$

Coloring
- The number of labeled undirected graphs with n vertices, $G_n = 2^{\binom{n}{2}}$
- The number of labeled directed graphs with n vertices, $G_n = 2^{n(n-1)}$
- The number of connected labeled undirected graphs with n vertices, $C_n = 2^{\binom{n}{2}} - \frac{1}{n}\sum_{k=1}^{n-1} k\binom{n}{k} 2^{\binom{n-k}{2}} C_k = 2^{\binom{n}{2}} - \sum_{k=1}^{n-1}\binom{n-1}{k-1} 2^{\binom{n-k}{2}} C_k$
- The number of k-connected labeled undirected graphs with n vertices, $D[n][k] = \sum_{s=1}^{n}\binom{n-1}{s-1} C_s D[n-s][k-1]$
- Cayley's formula: the number of trees on n labeled vertices = the number of spanning trees of a complete graph with n labeled vertices = n^{n-2}
- Number of ways to color a graph using k color such that no two adjacent nodes have same color
Complete graph = $k(k-1)(k-2)...(k-n+1)$
Tree = $k(k-1)^{n-1}$
Cycle = $(k-1)^n + (-1)^n(k-1)$
- Number of trees with n labeled nodes: n^{n-2}

Lucas Number
Number of edge cover of a cycle graph C_n is L_n
$L(n) = L(n-1) + L(n-2); L(0) = 2, L(1) = 1$

Catalan Number

$$C_{n+1} = C_0C_n + C_1C_{n-1} + C_2C_{n-2} + ... + C_nC_0$$

$$C_n = \binom{2n}{n} - \binom{2n}{n+1}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Derangement

$$D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$$

$$D_0 = 1, D_1 = 0$$

$1, 0, 1, 2, 9, 44, 265, ...$

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}

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

SUM
$1^4 + 2^4 + 3^4 + ... + n^4 = \frac{n(n+1)(2n+1)(3n2+3n+-1)}{30}$
$S_{(n,p)} = 1^p + 2^p + 3^p + 4^p + ... + n^a p$
$S(n,p) = \frac{1}{p+1}[(n+1)^{p+1} - 1 - \sum_{i=0}^{p-1}\binom{p+1}{i} S(n,i)]$
$1.2 + 2.3 + 3.4 + ... = \frac{1}{3}n(n+1)(n+2)$
$\sum_{i=1}^{n} f_k(i) = \frac{1}{k+1} n(n+1)(n+2)...(n+k) = \frac{1}{k+1}\frac{(n+k)!}{(n-1)!}$
$\sum_{i=0} nix^i = 1 + 2x^2 + 3x^3 + 4x^4 + 5x^5 + ... + nx^n = \frac{(x-(n+1)x^{n+1} + nx^{n+2})}{(x-1)^2}$

Probability

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

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

Matching Formula
Normal Graph
MM + MEC = n (without isolated vertex)
IS + VC = G
MaxIS + MVC = G

Bipartite Graph
MaxIS = n - MBM
MVC = MBM
MEC = n - MBM

Solution of $x^2 \equiv a(\mod p)$:
- $ca \equiv cb \pmod m \iff a \equiv b \pmod{\frac{n}{\gcd(n,c)}}$
- $ax \equiv b \pmod m$ has a solution $\iff \gcd(a,m)|b$
- If $ax \equiv b \pmod m$ has a solution, then it has $gcd(a,m)$ solutions and they are separated by $\frac{n}{\gcd(a,m)}$
- $ax \equiv 1 \pmod m$ has a solution or a is invertible $\pmod m \iff \gcd(a,m) = 1$
- $x^2 \equiv 1 \pmod p$ then $x \equiv \pm 1 \pmod p$
- There are $\frac{p-1}{2}$ has no solution.
- There are $\frac{p-1}{2}$ has exaclty two solutions.
- When $p\%4 = 3$, $x \equiv \pm a^{\frac{p+1}{4}}$
- When $p\%8 = 5$, $x \equiv a^{\frac{p+3}{8}}$ *or* $x \equiv 2^{\frac{p-1}{4}} a^{\frac{p+3}{8}}$

Totient
- If p is a prime $(p^k) = p^k - p^{k-1}$
- If a b are relatively prime, $\phi(ab) = \phi(a)\phi(b)$
- $\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})(1 - \frac{1}{p_3})...(1 - \frac{1}{p_k})$
- Sum of coprime to $n = n * \frac{\phi(n)}{2}$
- If $n = 2^k, \phi(n) = 2^{k-1} = \frac{n}{2}$
- For a b, $\phi(ab) = \phi(a)\phi(b)\frac{d}{\phi(d)}$
- $\phi(ip) = p\phi(i)$ whenever p is a prime and it divides i
- The number of $a(1 <= a <= N)$ such that $gcd(a,N) = d$ is $\phi(\frac{n}{d})$
- If $n > 2$, $\phi(n)$ is always even
- Sum of gcd, $\sum_{i=1}^{n} gcd(i,n) = \sum_{d|n} d\phi(\frac{n}{d})$
- Sum of lcm, $\sum_{i=1} nlcm(i,n) = \frac{n}{2}(\sum_{d|n}(d\phi(d)) + 1)$
- $\phi(1) = 1$ and $\phi(2) = 1$ which two are only odd ϕ
- $\phi(3) = 2$ and $\phi(4) = 2$ and $\phi(6) = 2$ which three are only prime ϕ
- Find minimum n such that $\frac{\phi(n)}{n}$ is maximum- Multiple of small primes- $2 * 3 * 5 * 7 * 11 * 13 * ...$

Mobius

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j) = 1] = \sum_{k=1}^{n}\mu(k)\lfloor\frac{n}{k}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} gcd(i,j) = \sum_{k=1}^{n} k \sum_{l=1}^{\lfloor\frac{n}{k}\rfloor}\mu(l)\lfloor\frac{n}{kl}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} gcd(i,j) = \sum_{k=1}^{n}\left(\frac{\lfloor\frac{n}{k}\rfloor(1+\lfloor\frac{n}{k}\rfloor)}{2}\right)^2\sum_{d|k}\mu(d)kd$$